Elektronski fakultet u Nišu
smer: telekomunikacije
predmet: mikroprocesorski sistemi

Pseudonistrukcije MIPS arhitekture

profesor:
Mile Stojčev

student:
Marija Petrović 9746

Sadržaj:

1 RISC koncept

RISC [*Reduced Instruction Set Computer*] koncept javlja se ranih osamdesetih godina prošlog veka i dovodi do radikalnih promena kod ISA (*Instruction Set Architecture*). ISA specificira primitivne komande koje se izvršavaju od strane hardvera, tj. izvršive mašinske instrukcije za dati tip hardvera. Kompleksnost ISA zavisi od formata instrukcije, formata podataka, adresnih načina rada, registara opšte namene, specifikacije opkoda i mehanizama za upravljanje tokom izvršenja programa.
U zavisnosti od izbora ovih osobina razlikujemo dva koncepta:
RISC (*Reduced Instruction Set Computer*) i CISC (*Complex Instruction Set Computer*).

CISC procesori bili su masovno korišćeni od pojave prvog mikroprocesora pa sve do pojave RISC čipova ranih osamdesetih godina prošlog veka. CISC procesore karakteriše veliki broj instrukcija (oko 300) pri čemu se koriste promenljivi formati instrukcija/podataka i relativno mali skup registara opšte namene (od 8 do 24). Kod CISC-ova postoji veliki broj operacija obraćanja memoriji koje koriste oko 20 adresnih načina rada.

Nakon, takoreći, tri decenije razvoja CISC arhitektura, korisnici računara su počeli da procenjuju (evaluiraju) odnos izmedju ISA i dostupnih hardversko/softverskih tehnologija.

Na osnovu analiza programa (uglavnom sprovedenih trasiranjem) ustanovljeno je da samo 25% od svih raspoloživih kompleksnih instrukcija troši 95% ukupnog vremena potrebnog za izvršenje programa. To, drugim rečima, znači da se ostalih 75% hardverski-podržanih instrukcija retko koristi u toku izvršenja programa (manje od 5%). Tako se i rodila ideja o RISC-ovima koja se bazira na sledećoj činjenici: *Učini ono što se najviše koristi najbžim*. Kao posledica ovakvog pristupa ostvarilo se dramatično povećanje performansi u odnosu na CISC dizajn. Projektanti su se vodili sledećim pravilom: kompleksne instrukcije, nakon procene njihovog pojavljivanja u programima, kada je njihov procenat mali treba implementirati softverski (u vidu potprograma), a ne hardverski kakva je bila dotadažnja praksa kod CISC procesora.

Implementacija retko korišćenih instrukcija u softveru omogućava da se skoro svi gradivni blokovi procesora smeste na jedinstveni VLSI čip. Šta više, u okviru jednog RISC procesorskog čipa, na današnjem nivou tehnologije, moguće je ugraditi sada *on-chip* keš kao i veći broj FP jedinica.

Obično, skup instrukcija RISC procesora je manji od 100 instrukcija (kod CISC procesora taj broj je oko 300), pri čemu su instrukcije obima 32- ili 64- bita, ali ne oba formata istovremeno. U principu se koriste od tri do pet adresnih načina rada. Arhitektura RISC-a je tipa *Load/Store*. To znači da sve instrukcije pribavljaju operande iz registara i smeštaju rezultate i registre (*register-based*), a da su *Load* i *Store* jedine dve naredbe pomoću kojih se procesor obraća memoriji.

RISC procesor često koristi dva posebna registarska polja. Jedno polje čine 32 registra za manipulisanje sa celobrojnim vrednostima i adresama (*integer register file*), a drugo poljese sastoji od 32 registra koji se koriste kod manipulisanja brojeva u pokretnom zarezu (*floating point register file*). Neki od RISC procesora koriste i više od 100 registara.

Pored korišćenja registarskih polja velikog obima, podeljeni keševi za instrukcije i podatke dodatno i značajno skraćuju vreme pristupa memoriji.
Osnovna ideja RISC-a je da se najveći broj instrukcija izvrši za jedan ciklus što se postiže pribavljanjem operanda direktnim putem iz registara, bafera preuredjenja ili keša podataka, a ne kao kod CISC-ova iz memorije. U tabeli 1 prikazane su ključne karakteristike pravih RISC procesora.

format instrukcije	fiksna 32-bitna instrukcija
taktna frekvenca	do 1 GHz
polje registara	32-192 registra opšte namene, izdvojeni *integer* i *float-point* registri
broj instrukcija i tipovi	oko 100, najveći broj je registarski zasnovan sa izuzetkom *Load/Store*
adresni način rada	ograničen na 3-5, samo *Load/Store* adresiraju memoriju
dizajn keša	najveći broj koristi podeljeni keš za instrukcije i podatke
CPI, prosečni CPI	1 ciklus za jednostavne operacije, 1.5 ciklus u proseku
upravljačka jedinica CPU-a	najveći broj koristi direktno upravljanje bez upravljačke memorije
tipični reprezentativni procesori	Sun Ultra Spars, MIPS R10000, Power PC 604, HP PA-8000, Digital 21264

Tabela 1. Karakteristike pravih RISC arhitektura

Korišćenjem velikog registarskog polja, bafera podataka, i izdvojenih keševa za instrukcije i podatke, ima povoljan efekat na interno prosledjivanje podataka kroz CPU, kao i eliminisanje nepotrebnih memorisanja (pamćenja) medjurezultata operacija. Sa značajno smanjenom kompleksnošću skupa instrukcija, RISC procesor se može lakše projektovati i realizovati kao jedinstveni VLSI čip. Prednost ovog pristupa su rad na višim frekvencijama, niži prosečni CPI koji se može postići, niži procenat keš promašaja koji se može ostvariti, i bolje mogućnosti koje stoje na raspolaganju za optimizaciju kompilatora.

Sa druge strane, da bi se obavio isti posao, u poređenju sa CISC, potreban je za oko 40% veći broj instrukcija. Ipak, negativni efekat usled povećanja broja instrukcija manji je u odnosu na pozitivne efekte postignute povećanjem taktne frekvencije i smanjenjem prosečnog broja CPI-a.

Drugi nedostatak RISC procesora je taj što je neophodno ugraditi veliki broj internih registara. Međutim razvojem tehnologije izrade VLSI čipova ovaj problem gubi na važnosti.

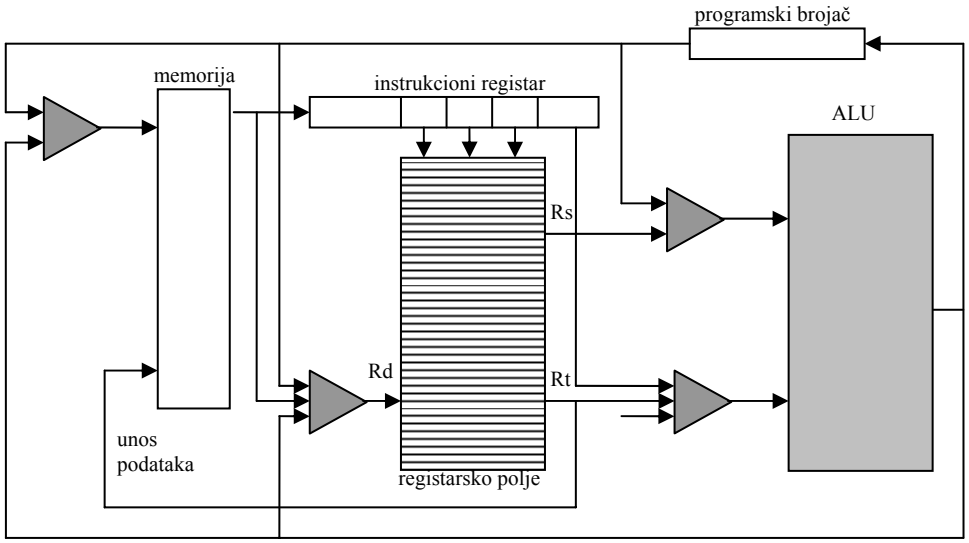
2 Arhitektura MIPS procesora

MIPS procesori su tipični predstavnici RISC koncepta sa Load/Store arhitekturom. To zapravo znači da se sve operacije izvršavaju nad operandima koji se nalaze u registrima na čipu, a memoriji se pristupa isključivo preko Load (memorija-registar) i Store (registar-memorija) instrukcija. Na taj način značajno se poboljšavaju performanse procesora jer je vreme pristupa memoriji znatno duže od vremena pristupa registrima. Prednosti ovog koncepta su:

- smanjenje broja pristupa memoriji čime se smanjuje i zahtevana veličina memeorije
- pojednostavljenje skupa instrukcija
- lakša optimizacija dodele registara od strane kompajlera

Takođe, spoljnu magistralu MIPS procesora čine dve posebne magistrale, jedna za pristup instrukcijama, a druga za pristup podacima. Time se pristup podacima obavlja paralelno sa pribavljanjem instrukcija čime se eliminišu konflikti između instrukcija i podataka koji realno postoje kod sistema zasnovanih na jedinstvenoj magistrali.

Sledeći tekst odnosi se na najjednostavniji model MIPS arhitekture koja je danas široko prihvaćena i označićemo ga sa MIPSxx. Na slici 1 prikazana je pojednostavljena struktura MIPS procesora.



Slika 1. Pojednostavljena struktura MIPS procesora

2.1 Memorija

Memorija je bajt adresabilna sa 32-bitnim adresama (od 0x00000000 do 0xffffffff). Koristi se tzv. *little- endian* način rada, što zapravo znači da se u memoriju na LS bajt poziciju upisuje bajt najmanje težine.

2.2 Registri

MIPS arhitektura definiše sledeće CPU-ove registre:

- 32 registra opšte namene, svi obima 32 bita, ovi registri čine RF polje
- par specijalnih registara HI i LO koji se koriste za čuvanje međurezultata kod izvršavanja operacija množenja i deljenja.
- programski brojač (PC) čuva adresu naredne instrukcije. Kako sve su sve instrukcije MIPS arhitekture dužine 32 bita PC se nakon izvršenja svake instrikcije uveća za 4.

Registri RF polja i njihova namena prikazani su na slici 2.

registar	**broj**	**korišćenje**
zero	0	konstanta nula
at	1	rezervisan za potrebe asemblera
vo v1	2 3	za prenos povratnih vrednosti iz poziva funkcija/poziva potprograma
a0 a1 a2 a3	4 5 6 7	za prenos argumenata funkcijama/potprogramima
t0 t1 t2 t3 t4 t5 t6 t7	8 9 10 11 12 13 14 15	za privremeno promenljive. Ove registre koristi (menja) pozivni program a po pravilu ne bi trebalo da ih koristie (menjaju) pozvani programi (funkcije i procedure)
s0 s1 s2 s3 s4 s5 s6 s7	16 17 18 19 20 21 22 23	za pamćenje privremeno promenljivih. Ove registre pozivni program koristi za memorisanje (pozvana funkcija mora da ih zapamti i obnovi)
t8 t9	24 25	za privremeno promenljive. Ove registre koristi (menja) pozivni program a po pravilu ne bi trebalo da ih koristie (menjaju) pozvani programi (funkcije i procedure)
k0 k1	26 27	rezervisani za kernel OSA

gp	28	pokazivač na globalnu memorijsku oblast
sp	29	pokazivač magacina (*stack pointer*)
tp	30	pokazivač okvira
ra	31	povratna adresa za funkcijske pozive

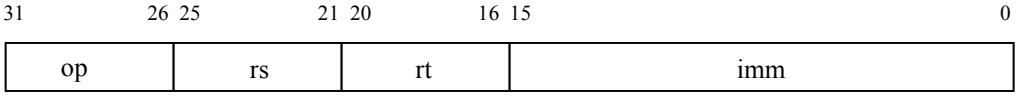
Slika 2. Imenovanje i način korišćenja RF polja

2.3 Formati instrukcija

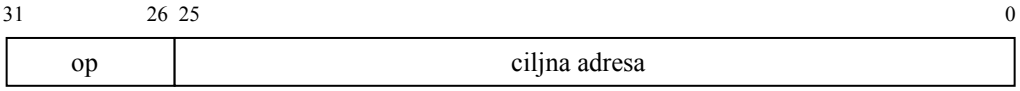
Formati instrukcija koje koristi MIPSxx prikazani su na slici 3.
Korišćeli su sledeći simboli:

- *op*- 6-bitni opkod instrukcije
- *rs*- 5-bitni specifikator izvorišnog registra
- *rd*- 5-bitni specifikator odredišnog registra
- *rt* - 5-bitni specifikator cilnog (izvorišnog/ odredišnog) registra ili uslova grananja
- *imm*- 16-bitni neposredni razmeštaj koji se odnosi na grananje ili adresni razmeštaj
- *ciljna adresa*- 26-bitna ciljna adresa grananja/skoka
- *shamt*- 5-bitna informacija koja ukazuje na iznos pomeranja
- *funct*- 6-bitno funkcijsko polje

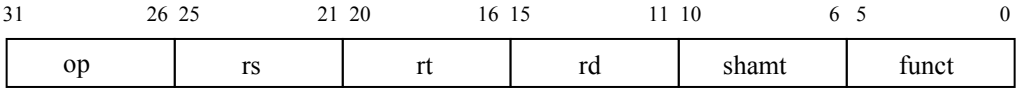
I-tip (neposredni-*immediate*)



J-tip (grananje-*jump*)



R-tip (registarski-*register*)



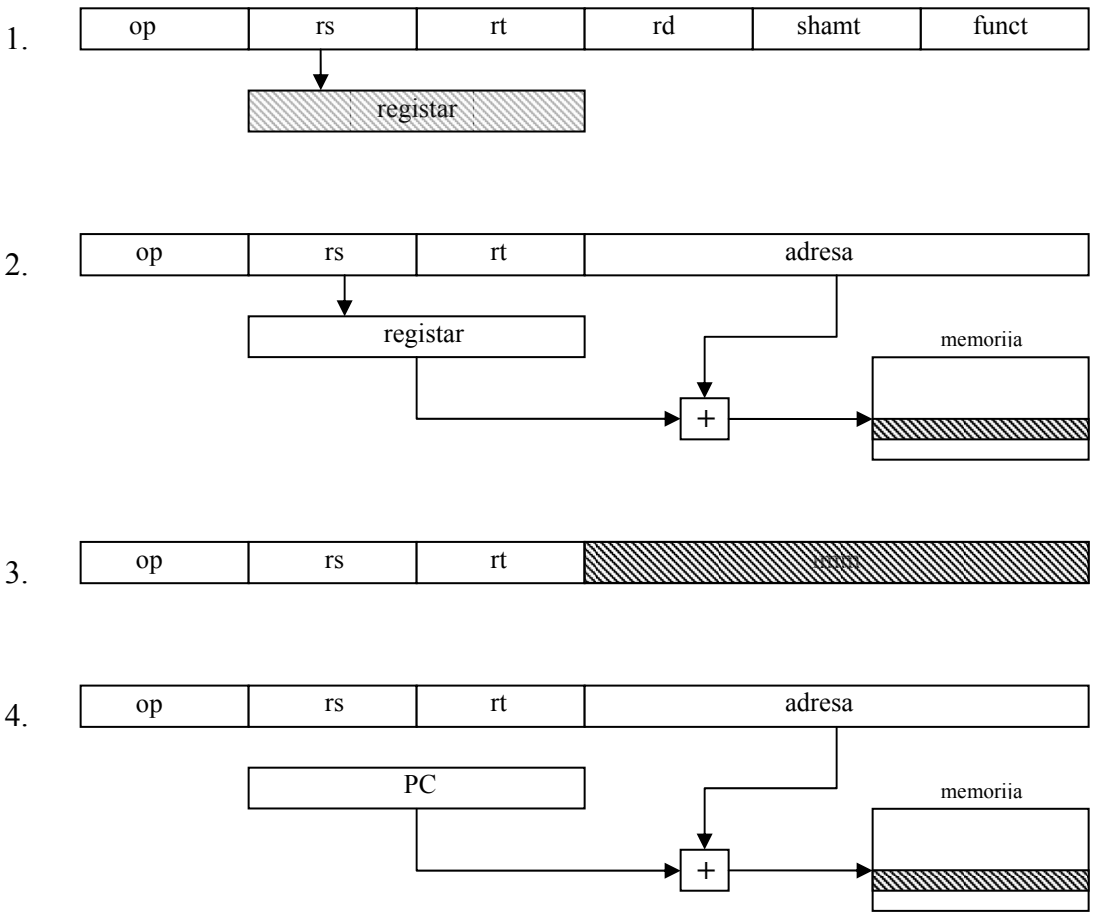
Slika 3. Formati instrukcija kod MIPSxx

2.4 Načini adresiranja operanada

Procesor MIPSxx koristi sledeća četiri adresna načina rada:

1. *registarsko adresiranje*:operand je u registru
2. *bazno adresiranje sa razmeštajem*:operand je u memorijskoj lokaciji čija je adresa zbir sadržaja registra i polja "adresa" instrukcije
3. *neposredno adresiranje*: operand je konstanta i sastavni je deo instrukcije (polje "imm")
4. *PC relativno adresiranje*: adresa predstavlja zbir PC-a i sadržaja polja "adresa" instrukcije

Na slici 4. prikazan je način specifikacije operanada za svaki od četiri adresna načina rada.



Slika 4. Četiri adresna načina rada kod MIPSxx

2.5 Skup insrukcija

Sve instrukcije MIPS arhitekture imaju fiksni obim od 32 bita i mogu se podeliti na sledeće tri grupe:

- regularne instrukcije za manipulisanje *integer* vrednostima (slika 5)
- makro-instrukcije ili tzv. pseudoinstrukcije (slika 4)
- regularne instrukcije za manipulisanje *float-point* vrednostima

ime	**sintaksa**	**obim/ vreme**
Add	add Rd, Rs, Rt	1/1
Add Immediate	addi Rd, Rs, Imm	1/1
Add Immediate Unsigned	addiu Rd, Rs, Imm	1/1
Add Unsigned	addu Rd, Rs, Rt	1/1
And	and Rd, Rs, Rt	1/1
And Immediate	andi Rd, Rs, Imm	1/1
Branch if Equal	beq Rs, Rt, label	1/1
Brenach if Greater than or Equal to Zero	bgez Rs, label	1/1
Brenach if Greater than or Equal to Zero And Link	bgezal Rs, label	1/1
Brenach if Greater than Zero	bgtz Rs, label	1/1
Brenach if Less than or Equal to Zero	blez Rs, label	1/1
Brenach if Less than Zero And Link	bltzal Rs, label	1/1
Brenach if Less than Zero	bltz Rs, label	1/1
Brench if Not Equal	bne Rs, label	1/1
Divide	div Rs, Rt	1/38
Divide Unsigned	divu Rs, Rt	1/38
Jump	j label	1/1
Jump and Link	jal label	1/1
Jump and Link Register	jalr Rd, Rs	1/1
Jump Register	jr Rs	1/1
Load Byte	lb Rt, offset(Rs)	1/1
Load Byte Unsigned	lbu Rt, offset(Rs)	1/1
Load Halfword	lh Rt, offset(Rs)	1/1
Load Halfword Unsigned	lhu Rt, offset(Rs)	1/1
Load Upper Immediate	lui Rt, Imm	1/1
Load Word	lw Rt, offset(Rs)	1/1
Load Word Left	lwl Rt, offset(Rs)	1/1
Load Word Right	lwr Rt, offset(Rs)	1/1
Move From Coprocessor 0	mfc0 Rd, Cs	1/1
Move From High	mfhi Rd	1/1
Move From Low	mflo Rd	1/1
Move to Coprocessor 0	mtc0 Rt, Cd	1/1
Move to High	mthi Rd	1/1
Move to Low	mtlo Rd	1/1
Multiply	mult Rs, Rt	1/32
Multuply Unsigned	multu Rs, Rt	1/32
Nor	nor Rd, Rs, Rt	1/1
Or	or Rd, Rs, Rt	1/1

Or Immediate	ori Rd, Rs, Imm	1/1
Return from Exeption	rfe	1/1
Store Byte	sb Rt, offset (Rs)	1/1
Store Halfword	sh Rt, offset (Rs	1/1
Shift Left Logical	sll Rd, Rt, sa	1/1
Shift Left Logical Variable	sllv Rd, Rt, Rs	1/1
Set on Less then	slt Rd, Rt, Rs	1/1
Set on Less then Immediate	slti Rd, Rs, Imm	1/1
Set on Less then Immediate Unsigned	sltiu Rd, Rs, Imm	1/1
Set on Less then Usigned	sltu Rd, Rt, Rs	1/1
Shift Right Aritmetic	sra Rd, Rs, sa	1/1
Shift Right Aritmetic Variable	srav Rd, Rt, Rs	1/1
Shift Right Logical	srl Rd, Rt, sa	1/1
Shift Right Logical Variable	srlv Rd, Rt, Rs	1/1
Subtract	sub Rd, Rs, Rt	1/1
Subtract Unsigned	subu Rd, Rs, Rt	1/1
Store Word	sw Rt, offset(Rs)	1/1
Store Word Left	swl Rt, offset(Rs)	1/1
Store Word Right	swr Rt, offset(Rs)	1/1
System Call	syscall	1/1
Exclusive Or	xor Rd, Rs, Rt	1/1
Exclusive Or Immediate	xori Rd, Rs, Imm	1/1

Slika 5. Osnovne instrukcije za manipulisanje integer vrednostima

Na slici 6. prikazane su makro ili pseudo instrukcije. Svaki put kada programer specificira neku makro instrukciju asembler je, da bi je izvršio, prevodi u skup koji čine jedna ili više osnovnih (regularnih) instrukcija. O ovome će više biti reči u sekciji 3.

ime	**sintaksa**	**prostor/ vreme**
Absolute Value	abs Rd, Rs	3/3
Branch if Equal to Zero	beqz Rs, label	1/1
Branch if Greater than or Equal	bge Rs,Rt, label	2/2
Branch if Greater than or Equal Unsigned	bgeu Rs, Rt, label	2/2
Branch if Greater than	bgt Rs, Rt, label	2/2
Branch if Greater than Unsigned	bgtu Rs, Rt, label	2/2
Branch if Less than or Equal	ble Rs, Rt, label	2/2
Branch if Less than or Equal Unsigned	bleu Rs, Rt, label	2/2
Branch if Less than	blt Rs, Rt, label	2/2
Branch if Less than Unsigned	bltu Rs, Rt, label	2/2
Branch if Not Equal to Zero	bnez Rs, label	1/1
Branch uncoditional	b label	1/1
Divide	div Rd, Rs, Rt	4/41
Divide Unsigned	divu Rd, Rs, Rt	4/41
Load Adress	la Rd, label	2/2
Load Immediate	li Rd, Imm	2/2
Move	mov Rd, Rs	1/1
Multiply	mul Rd, Rs, Rt	2/32
Multiply (with overflow exception)	mulo Rd, Rs, Rt	2/37

Multiply Unsigned (with overflow exception)	muluu Rd, Rs, Rt	2/35
Negate	neg Rd, Rs	1/1
Negate Unsigned	negu Rd, Rs	1/1
Nop	nop	1/1
Not	not Rd, Rs	1/1
Remainder	rem Rd, Rs, Rt	4/40
Remainder Unsigned	remu Rd, Rs, Rt	4/40
Rotate Left Variable	rol Rd, Rs, Rt	4/4
Rotate Right Variable	ror Rd, Rs, Rt	4/4
Rotate Left Constant	rol Rd, Rs, sa	3/3
Rotate Right Constant	ror Rd, Rs, sa	3/3
Set if Equal	seq Rd, Rs, Rt	4/4
Set if Greater than or Equal	sge Rd, Rs, Rt	4/4
Set if Greater than or Equal Unsigned	sgeu Rd, Rs, Rt	4/4
Set if Greater than	sgt Rd, Rs, Rt	1/1
Set if Greater than Unsigned	sgtu Rd, Rs, Rt	1/1
Set if Less than or Equal	sle Rd, Rs, Rt	4/4
Set if Less than or Equal Unsigned	sleu Rd, Rs, Rt	4/4
Set if Not Equal	sne Rd, Rs, Rt	4/4
Unaligned Load Halfword Unasigned	ulhu Rd, n(Rs)	4/4
Unaligned Load Halfword	ulh Rd, n(Rs)	4/4
Unaligned Load Word	ulw Rd, n(Rs)	2/2
Unaligned Store Halfword	ush Rd, n(Rs)	3/3
Unaligned Store Word	usw Rd, n(Rs)	2/2

Slika 6. Skup makro instrukcija

Instrukcije za manipulisanje *float-point* (realnim) vrednostima koriste posebne *float-point* registre (ima ih 32), ali njima se nećemo baviti u ovom tekstu.

3 Pseudo instrukcije MIPSxx arhitekture

Kao što je već naglašeno u MIPS arhitekturi postoji određen broj makro ili pseudo instrukcija za manipulisanje *integer* vrednostima. Svaku od ovih instrukcija asembler radi izvršenja prevodi u skup od jedne ili više osnovnih (regularnih) instrukcija kao što je navedeno u tekstu koji sledi.

Za svaku pseudoinstrukciju naveden je naziv instrukcije (na engleskom), sintaksa, broj reči koje instrukcija zauzima, vreme potrebno za izvršenje instrukcije (broj ciklusa), zatim kratak opis i skup regularnih instrukcija kojima se data pseudoinstrukcija zamenjuje.

Absolute Value	**abs Rd, Rs**	3/3

U registar `Rd` smešta se apsolutna vrednost sadržaja registra `Rs`.

```
          addu Rd, $0, Rs
          bgez Rs, skip
          sub Rd, $0, Rs
skip:
```

Primer:

```
lui $1, 0xffff
or $t1, $1, -5      # $t1 = -5


addu $t2, $0, $t1     # abs $t2, $t1
bgez $t1, skip
sll $0, $0, 0
sub $t2, $0, $t1
skip:
```

Branch if Equal to Zero	**beqz Rs, Label**	1/1

Vrši se grananje na labelu `Label` ukoliko je sadržaj registra `Rs` jednak nuli.

```
          beq Rs, $0, Label
```

Primer:

```
ori $t0, 0          # $t0=0

beq $t0,$0, label   # beqz $t0, label
sll $0, $0, 0

ori $t0, 2          # ukoliko nije došlo do grananja
                    #    $t0=2
label:
```

Branch if Greater than or Equal	**bge Rs, Rt, Label**	2/2

Vrši se grananje na labelu `Label` ukoliko je sadržaj registra `Rs` veći ili jednak sadržaju registra `Rt`.

```
          slt $1, Rs, Rt
```

```
beq $1, $0, Label
```

Primer:

```
ori $t0, 4          # $t0=4
lui $1, 0xffff
or $t1, $1, -3      # $t1=-3


slt $1, $t1, $t0    # bge $t1, $t0, lab
beq $1, $0,lab
sll $0, $0, 0

ori $t2, 1          # ukoliko nije došlo do grananja
                    #   $t2=1
lab:
```

Branch if Greater than or Equal Unsigned	**bgeu Rs, Rt, Label**	2/2

Porede se sadržaji registara `Rs` i `Rt` kao neoznačeni brojevi. Ako je `Rs` veće ili jednako `Rt` vrši se grananje na labelu `Label`.

```
sltu $1, Rs, Rt
beq $1, $0, Label
```

Primer:

```
ori $t0, 4          # $t0=4
ori $t1, 3          # $t1=3


sltu $1, $t1, $t0   # bgeu $t1, $t0, lab
beq  $1, $0,lab
sll $0, $0, 0

ori $t2, 1          # ukoliko nije došlo do grananja
                    #  $t2=1
lab:
```

Branch if Greater than	**bgt Rs, Rt, Label**	2/2

Porede se sadržaji registara `Rs` i `Rt` kao označeni brojevi. Ako je `Rs` veće od `Rt` vrši se grananje na labelu `Label`.

```
slt $1, Rt, Rs
bne $1, $0, Label
```

Primer:

```
ori $t0, 4          # $t0=4
lui $1,0xffff
or $t1,$1, -3       # $t1=-3


slt $1, $t0, $t1   # bgt $t1, $t0, lab
bne $1, $0,lab
sll $0, $0, 0

ori $t2, 1          # ukoliko nije došlo do grananja
                    #  $t2=1
lab:
```

Branch if Greater than Unsigned	**bgtu Rs, Rt, Label**	2/2

Porede se sadržaji registara `Rs` i `Rt` kao neoznačeni brojevi. Ako je `Rs` veće od `Rt` vrši se grananje na labelu `Label`.

```
sltu $1, Rt, Rs
bne $1, $0, Label
```

Primer:

```
ori $t0, 4          # $t0=4
ori $t1, 3          # $t1=4

sltu $1, $t0, $t1   # bgtu $t1, $t0, lab
bne $1, $0,lab
sll $0, $0, 0

ori $t2, 1          # ukoliko nije došlo do grananja
                    #  $t2=1
lab:
```

Branch if Less than or Equal	**ble Rs, Rt, Label**	2/2

Porede se sadržaji registara `Rs` i `Rt` kao označeni brojevi. Ako je `Rs` manje ili jednako `Rt` vrši se grananje na labelu `Label`.

```
slt $1, Rt, Rs
beq $1, $0, Label
```

Primer:

```
ori $t0, 4          # $t0=4
ori $t1, 3          # $t1=3


slt $1, $t0, $t1    # ble $t1, $t0, lab
beq $1, $0, lab
sll $0, $0, 0

ori $t2, 1          # ukoliko nije došlo do grananja
                    #  $t2=1
lab:
```

Branch if Less than or Equal Unsigned	**bleu Rs, Rt, Label**	2/2

Porede se sadržaji registara `Rs` i `Rt` kao neoznačeni brojevi. Ako je `Rs` manje ili jednako `Rt` vrši se grananje na labelu `Label`.

```
sltu $1, Rt, Rs
beq $1, $0, Label
```

Primer:

```
ori $t0, 4          # $t0=4
ori $t1, 3          # $t1=3


sltu $1, $t0, $t1   # bleu $t1, $t0, lab
beq $1, $0, lab
sll $0, $0, 0

ori $t2, 1          # ukoliko nije došlo do grananja
                    #  $t2=1
lab:
```

Branch if Less than	**blt Rs, Rt, Label**	2/2

Porede se sadržaji registara `Rs` i `Rt` kao označeni brojevi. Ako je `Rs` manje od `Rt` vrši se grananje na labelu `Label`.

```
slt $1, Rs, Rt
bne $1, $0, Label
```

Primer:

```
ori $t0, 4          # $t0=4
ori $t1, 3          # $t1=3

slt $1, $t1, $t0    # blt $t1, $t0, lab
bne $1, $0,lab
sll $0, $0, 0

ori $t2, 1          # ukoliko nije došlo do grananja
                    #  $t2=1
lab:
```

Branch if Less than Unsigned	**bltu Rs, Rt, Label**	2/2

Porede se sadržaji registara `Rs` i `Rt` kao neoznačeni brojevi. Ako je `Rs` manje od `Rt` vrši se grananje na labelu `Label`.

```
sltu $1, Rs, Rt
bne $1, $0, Label
```

Primer:

```
ori $t0, 4          # $t0=4
ori $t1, 3          # 4t1=3

sltu $1, $t1, $t0   # bltu $t1, $t0, lab
bne $1, $0,lab
sll $0, $0, 0

ori $t2, 1          # ukoliko nije došlo do grananja
                    #  $t2=1
lab:
```

Branch if Not Equal to Zero	**bnez Rs, Label**	1/1

Ako sadržaj registra `Rs` nije jednak nuli vrši se grananje na labelu `Label`.

```
bne Rs, $0, Label
```

Primer:

```
ori $t0, 0          # $t0=0

bne $t0,$0, label   # bnez $t0, label
```

```
sll $0, $0, 0

ori $t0, 2          # ukoliko nije došlo do grananja
                    # $t2=1
label:
```

Branch unconditional	**b Label**	1/1

Vrši se bezuslovno grananje na labelu `Label`.

```
beq $0, $0, Label
```

Primer:

```
ori $t1, 1          # $t1=1

beq $0,$0, label    # b label
sll $0, $0, 0

ori $t0, 2          # ukoliko nije doslo do grananja
                    # $t0=2
label:
```

Divide	**div Rd, Rs, Rt**	4/41

32-bitni broj u registru `Rs` deli se 32-bitnim brojem u `Rt`, pri čemu se operandi tretiraju kao označeni celi brojevi. Količnik se smešta u registar `Rd`. Ukoliko je delilac jednak nuli generiše se izuzetak.

```
          bne Rt, $0, divide
          break 0
divide:   div Rs, Rt
          mflo Rd
```

Primer:

```
lui $1, 0xffff
or $t1, $1, -8      # t1=-8

ori $t0,2           # $t0=2

beq $t0, $0, lab    # div $t2, $t1, $t0
sll $0, $0, 0
div $t1, $t0
mflo $t2
lab:
```

Divide Unsigned	**divu Rd, Rs, Rt**	4/41

32-bitni broj u registru `Rs` deli se 32-bitnim brojem u `Rt`, pri čemu se operandi tretiraju kao neoznačeni celi brojevi. Količnik se smešta u registar `Rd`. Ukoliko je delilac jednak nuli generiše se izuzetak.

```
          bne Rt, $0, divide
          break $0
divide:   divu Rs, Rt
          mflo Rd
```

Primer:

```
ori $t1,8  # $t1=8
ori $t0,2  # $t0=2

beq $t0, $0, lab  # divu $t2, $t1, $t0
sll $0, $0, 0
divu $t1, $t0
mflo $t2
lab:
```

Load Address	**la Rd, Label**	2/2

Adresa koja odgovara labeli `Label` se učitava u registar `Rd`.

```
          lui $1, HI(Label)
          ori Rd, $1, LO(Label)
```

Primer:

```
la $t0, target
jr $t0
...
target:
...
```

Load Immediate	**li Rd, value**	2/2

32-bitna vrednost `value` se učitava u registar `Rd`.

```
          lui $1, HI(value)
          ori Rd, $1, LO(value)
```

Primer:

```
lui $1, 0xffff       # negativne vrednosti
```

```
or $t1, $1, -354


ori $t2, 45000        # pozitivne vrednosti
                      #  do 16 bita (0...65 536)

lui $1, 0x1234        # pozitivne vrednosti od
or $t3, $1, 0x5678    # 16 od 32 bita
```

Move	**move Rd, Rs**	1/1

Sadržaj registra `Rs` se kopira u registar `Rd`.

```
addu Rd, $0, Rs
```

Primer:

```
ori $t1, 3          # $t1=3

addu $t2,$t1, $0    # move $t2, $t1
```

Multiply	**mul Rd, Rs, Rt**	2/33

Množe se 32-bitni brojevi u registrima `Rs` i `Rt`, pri čemu se operandi tretiraju kao označeni celi brojevi. Nižih 32 bita rezultata smešta se u registar `Rd`.

```
mult Rs, Rt
mflo Rd
```

Primer:

```
ori $t0, 62000      # $t0 = 62000
lui $1, 0xffff
or $t1, $1, -5      # $t1 = -5

mult $t1, $t0       # mul $t2, $t1, $t0
mflo $t2
```

Multiply (with overflow exception)	**mulo Rd, Rs, Rt**	7/37

Množe se 32-bitni brojevi u registrima `Rs` i `Rt`, pri čemu se operandi tretiraju kao označeni celi brojevi. Nižih 32 bita rezultata smešta se u registar `Rd`. Ukoliko je došlo do prekoračenja generiše se izuzetak.

```
          mult Rs, Rt
          mfhi $1
          mflo Rd
          sra Rd, Rd, 31
          beq $1, Rd, store
          break $0
store:    mflo Rd
```

Primer:

```
ori $t0, 62000        # $t0=62000
ori $t1, 65000        # $t1=65000


mult $t1, $t0         # mulo $t3, $t1, $t0
mfhi $1
mflo $t2
sra $t2, $t2, 31
beq $1, $t2, store
sll $0, $0, 0
break 0               # generiše se izuzetak
store:
mflo $t2
```

Multiply Unsigned (with overflow excep.) **`mulou Rd, Rs, Rt`** 5/35

Množe se 32-bitni brojevi u registrima `Rs` i `Rt`, pri čemu se operandi tretiraju kao neoznačeni celi brojevi. Nižih 32 bita rezultata smešta se u registar `Rd`. Ukoliko je došlo do prekoračenja generiše se izuzetak.

```
          multu Rs, Rt
          mfhi $1
          beq $1, $0, store
          break $0
store:    mflo Rd
```

Primer:

```
lui $t0, 2          # $t0 = 0x20000
lui $t1, 3          # $t1 = 0x30000

multu $t1, $t0      # HI=0x00000006, LO=0x00000000
mfhi $1             # $1=0x00000006
beq $1, $0, store
break 0              # generise se izuzetak
store:
mflo $t2            # $t2=0x00000000
```

Negate	**neg Rd, Rs**	1/1

U registar `Rd` smešta se vrednost suprotna po znaku u odnosu na sadržaj registra `Rs`. Ako dođe do aritmetičkog prekoračenja generiše se izuzetak.

```
sub Rd, $0, Rs
```

Primer:

```
lui $1, 0xffff
or $t1,$1, -3      # $t1=-3

sub $t2, $0, $t1   # neg $t2, $t1
```

Negate Unsigned	**negu Rd, Rs**	1/1

U registar `Rd` smešta se vrednost suprotna po znaku u odnosu na sadržaj registra `Rs`.

```
subu Rd, $0, Rs
```

Primer:

```
ori $t1,$1, 3       # $t1=3

subu $t2, $0, $t1   # negu $t2, $t1
```

Nop	**nop**	1/1

Operacija bez efekta. Koristi se, na primer, u okviru slota kod *branch* instrukcija.

```
sll $0, $0, 0
```

Primer:

```
sll $0, $0, 0    # alternativno: add $0, $0, 0
```

Not	**not Rd, Rs**	1/1

U registar `Rd` smešta se negacija sadržaja registra `Rs`.

```
nor Rd, Rs, $0
```

Primer:

```
ori $t1, 7          # $t1=7

nor $t2, $t1, $0    # not $t2, $t1
```

Remainder	**rem Rd, Rs, Rt**	4/41

32-bitni broj u registru `Rs` deli se 32-bitnim brojem u `Rt`, pri čemu se operandi tretiraju kao označeni celi brojevi. Ostatak se smešta u registar `Rd`. Ukoliko je delilac jednak nuli generiše se izuzetak.

```
          bne Rt, $0, divide
          break $0
divide:   div Rs, Rt
          mfhi Rd
```

Primer:

```
lui $1, 0xffff
or $t1,$1, -10       # $t1=-10
ori $t0, 4           # 4t0= 4

bne $t0, $0, divide  # rem $t2, $t1, $t0
break 0
divide:
div $t1, $t0
mfhi $t2
```

Remainder Unsigned	**remu Rd, Rs, Rt**	4/41

32-bitni broj u registru `Rs` deli se 32-bitnim brojem u `Rt`, pri čemu se operandi tretiraju kao neoznačeni celi brojevi. Ostatak se smešta u registar `Rd`. Ukoliko je delilac jednak nuli generiše se izuzetak

```
          bne Rt, $0, divide
          break $0
divide:   divu Rs, Rt
          mfhi Rd
```

Primer:

```
ori $t1, 10          # $t1=10
ori $t0, 4           # $t0= 4

bne $t0, $0, divide  # remu $t2, $t1, $t0
break 0
divide:
```

```
divu $t1, $t0
mfhi $t2
```

Rotate Left Variable	**rol Rd, Rs, Rt**	4/4

Sadržaj registra `Rs` se rotira ulevo za broj mesta naveden u registru `Rt`, a rezultat se smešta u `Rd`.

```
subu $1, $0, Rt
srlv $1, Rs, $1
sllv Rd, Rs, Rt
or Rd, Rd, $1
```

Primer:

```
ori $t0, 1              # $t0=1
ori $t1, 4              # $t1=4

subu $1, $0, $t0        # rol $t2, $t1, $t0
srlv $1, $t1, $1
sllv $t2, $t1, $t0
or $t2, $t2, $1
```

Rotate Right Variable	**ror Rd, Rs, Rt**	4/4

Sadržaj registra `Rs` se rotira udesno za broj mesta naveden u registru `Rt`, a rezultat se smešta u `Rd`.

```
subu $1, $0, Rt
sllv $1, Rs, $1
srlv Rd, Rs, Rt
or Rd, Rd, $1
```

Primer:

```
ori $t0, 1              # $t0=1
ori $t1, 4              # $t1=4

subu $1, $0, $t0        # ror $t2, $t1, $t0
sllv $1, $t1, $1
srlv $t2, $t1, $t0
or $t2, $t2, $1
```

Rotate Left Constant	**rol Rd, Rs, sa**	3/3

Sadržaj registra `Rs` se rotira za `sa` mesta ulevo i rezultat se smešta u `Rd`.

```
srl $1, Rs, 32-sa
sll Rd, Rs, sa
or Rd, Rd, $1
```

Primer:

```
ori $t1, 1              # $t1=1

srl $1, $t1, 29         # rol $t2, $t1, 3
sll $t2, $t1, 3
or $t2, $t2, $1
```

Rotate Right Constant	**ror Rd, Rs, sa**	3/3

Sadržaj registra `Rs` se rotira za `sa` mesta udesno i rezultat se smešta u `Rd`.

```
sll $1, Rs, 32-sa
srl Rd, Rs, sa
or Rd, Rd, $1
```

Primer:

```
ori $t1, 1              # $t1=1

sll $1, $t1, 29         # ror $t2, $t1, 3
srl $t2, $t1, 3
or $t2, $t2, $1
```

Set if Equal	**seq Rd, Rs, Rt**	4/4

Ako su sadržaji registara `Rs` i `Rt` jednaki registar `Rd` se postavlja na 1, a u suprotnom na 0.

```
        beq Rt, Rs, set
        ori Rd, $0, 0
        beq $0, $0, skip
set:    ori Rd, $0, 1
skip:
```

Primer:

```
ori $t0, 4          # $t0=4
ori $t1, 4          # $t1=4

beq $t0, $t1, set   # seq $t2, $t1, $t0
ori $t2, $0, 0
beq $0, $0, skip
sll $0, $0, 0
set:
ori $t2, 1
skip:
```

Set if Greater than or Equal	**sge Rd, Rs, Rt**	4/4

Porede se sadržaji registara `Rs` i `Rt` kao označeni brojevi. Ako je `Rs` veće ili jednako `Rt` u registar `Rd` se upisuje 1, a u suprotnom 0.

```
          bne Rt, Rs, set
          ori Rd, $0, 1
          beq $0, $0, skip
set:      slt Rd, Rt, Rs
skip:
```

Primer:

```
ori $t0, 4          # $t0=4
ori $t1, 4          # $t1=4


bne $t0, $t1, set   # sge $t2, $t1, $t0
ori $t2, $0, 1
beq $0, $0, skip
sll $0, $0, 0
set:
slt $t2, $t0, $t1
skip:
```

Set if Greater than or Equal Unsigned	**sgeu Rd, Rs, Rt**	4/4

Porede se sadržaji registara `Rs` i `Rt` kao neoznačeni brojevi. Ako je `Rs` veće ili jednako `Rt` u registar `Rd` se upisuje 1, a u suprotnom 0.

```
          bne Rt, Rs, set
          ori Rd, $0, 1
          beq $0, $0, skip
set:      sltu Rd, Rt, Rs
```

```
skip:
```

Primer:

```
li $t0, 4              # $t0=4
li $t1, 4              # $t1=4

bne $t0, $t1, set      # sgeu $t2, $t1, $t0
sll $0, $0, 0
ori $t2, $0, 1
beq $0, $0, skip
set:
sltu $t2, $t0, $t1
skip:
```

Set if Greater than	**sgt Rd, Rs, Rt**	1/1

Porede se sadržaji registara `Rs` i `Rt` kao označeni brojevi. Ako je `Rs` veće od `Rt` u registar `Rd` se upisuje 1, a u suprotnom 0.

```
slt Rd, Rt, Rs
```

Primer:

```
lui $1, 0xffff
or $t0,$1, -3          # $t0=-3
ori $t1, 4             # $t1=4

slt $t2, $t0, $t1      # sgt $t2, $t1, $t0
```

Set if Greater than Unsigned	**sgtu Rd, Rs, Rt**	1/1

Porede se sadržaji registara `Rs` i `Rt` kao neoznačeni brojevi. Ako je `Rs` veće od `Rt` u registar `Rd` se upisuje 1, a u suprotnom 0.

```
sltu Rd, Rt, Rs
```

Primer:

```
ori $t0, 3           # $t0=3
ori $t1, 4           # $t1=4

sltu $t2, $t0, $t1   # sgtu $t2, $t1, $t0
```

Set if Less than or Equal	**sle Rd, Rs, Rt**	4/4

Porede se sadržaji registara `Rs` i `Rt` kao označeni brojevi. Ako je `Rs` manje ili jednako `Rt` u registar `Rd` se upisuje 1, a u suprotnom 0.

```
          bne Rt, Rs, set
          ori Rd, $0, 1
          beq $0, $0, skip
set:      slt Rd, Rs, Rt
skip:
```

Primer:

```
ori $t0, 4          # $t0=4
ori $t1, 4          # $t1=4

bne $t0, $t1, set   # sle $t2, $t1, $t0
ori $t2, $0, 1
beq $0, $0, skip
sll $0, $0, 0
set:
slt $t2, $t1, $t0
skip:
```

Set if Less than or Equal Unsigned	**`sleu Rd, Rs, Rt`**	4/4

Porede se sadržaji registara `Rs` i `Rt` kao neoznačeni brojevi. Ako je `Rs` manje ili jednako `Rt` u registar `Rd` se upisuje 1, a u suprotnom 0.

```
          bne Rt, Rs, set
          ori Rd, $0, 1
          beq $0, $0, skip
set:      sltu Rd, Rs, Rt
skip:
```

Primer:

```
ori $t0, 4          # $t0=4
ori $t1, 4          # $t1=4

bne $t0, $t1, set   # sleu $t2, $t1, $t0
ori $t2, $0, 1
beq $0, $0, skip
sll $0, $0, 0
set:
sltu $t2, $t1, $t0
skip:
```

Set if Not Equal	**sne Rd, Rs, Rt**	4/4

Ako su sadržaji registara `Rs` i `Rt` različiti registar `Rd` se postavlja na 1, a u suprotnom na 0.

```
          beq Rt, Rs, reset
          ori Rd, $0, 1
          beq $0, $0, skip
reset:    ori Rd, $0, 0
skip:
```

Primer:

```
ori $t0, 4              # $t0=4
ori $t1, 4              # $t1=4

beq $t0, $t1, reset     # sne $t2, $t1, $t0
ori $t2, $0, 1
beq $0, $0, skip
sll $0, $0, 0
reset:
ori $t2, $0, 0
skip:
```

Unaligned Load Halfword	**ulh Rd, n(Rs)**	4/4

Označena polureč sa adrese `n(Rs)` se učitava u registar `Rd`. Adresa ne mora biti poravnata. Viša polureč registra `Rd` se ispunjava nulama ili jedinicama u zavisnosti od toga da li je učitana polureč pozitivna ili negativna.

```
          lb Rd, (n+1)(Rs)
          lbu $1, n(Rs)
          sll Rd, Rd, 8
          or Rd, Rd, $1
```

Primer:

```
lui $t0, 0x1000     # $t0=0x10000000

lui $1, 0x9ABC
ori $1, 0xDEF0      # $1 = 0x9ABCDEF0
sw $1, 0($t0)       # 0   1   2   3
                    #  F0 DE BC 9A

lb $t1, 2($t0)      # ulh $t1, 1($t0)
lbu $1, 1($t0)
sll $t1, $t1, 8
```

```
or $t1, $t1, $1
                        # $t1 = 0xFFFFBCDE
```

Unaligned Load Halfword Unsigned	**ulhu Rd, n(Rs)**	4/4

Označena polureč sa adrese `n(Rs)` se učitava u registar `Rd`. Adresa ne mora biti poravnata. Viša polureč registra `Rd` se uvek ispunjava nulama.

```
lbu Rd, (n+1)(Rs)
lbu $1, n(Rs)
sll Rd, Rd, 8
or Rd, Rd, $1
```

Primer:

```
lui $t0, 0x1000     # $t0=0x10000000

lui $1, 0x1234
ori $1, 0x5678       # $1 = 0x12345678
sw $1, 0($t0)        #   0 1   2  3
                      #  78 56 34 12

lbu $t1, 2($t0)      # ulhu $t1, 1($t0)
lbu $1, 1($t0)
sll $t1, $t1, 8
or $t1, $t1, $1
                        # $t1 = 0x00003456
```

Unaligned Load Word	**ulw Rd, n(Rs)**	2/2

Reč sa adrese `n(Rs)` se učitava u registar `Rd`. Adresa ne mora biti poravnata.

```
lwl Rd, (n+3)(Rs)
lwr Rd, n(Rs)
```

Primer:

```
lui $t0, 0x1000       # $t0=0x10000000

lui $1, 0x0123
ori $1, 0x4567        # $1=01234567
sw $1, 0($t0)
lui $1, 0x89AB
ori $1, 0xCDEF
sw $1, 4($t0)        # 0  1  2  3  4  5  6  7
```

```
                        # 67 45 23 01 EF CD AB 89

lwl $t1, 6($t0)         # ulw $t1, 3($t0)
lwr $t1, 3($t0)
                        # $t1 = 0xABCDEF01
```

Unaligned Store Halfword	**ush Rd, n(Rs)**	3/3

Niža polureč iz registra `Rd` se smešta na adresu `n(Rs)`. Adresa ne mora biti poravnata.

```
sb Rd, n(Rs)
srl $1, Rd, 8
sb $1, (n+1)(Rs)
```

Primer:

```
lui $t0, 0x1000          # $t0=0x10000000

ori $t1, 0x1234          # $t1=0x00001234

sb $t1, 3($t0)           # ush $t1, 3($t0)
srl $1, $t1, 8
sb $1, 4($t0)
                          #  0 1  2  3  4  5  6
                          #  00 00 00 34 12 00 00
```

Unaligned Store Word	**usw Rd, n(Rs)**	2/2

Sadržaj registra `Rd` se smešta na adresu `n(Rs)`. Adresa ne mora biti poravnata.

```
swl Rd, (n+3)(Rs)
swr Rd, n(Rs)
```

Primer:

```
lui $t0, 0x1000          # $t0=0x10000000

lui $1, 0x1234
ori $t1, $1, 0x5678      # $t1=12345678

swl $t1, 6($t0)          # usw $t1, 3($t0)
swr $t1, 3($t0)
                         #  0  1  2  3  4  5  6  7
                         #  00 00 00 78 56 34 12 00
```

4 Prošireni skup pseudo instrukcija

Instrukcije koje slede zadaju se kao makro naredbe. Makro naredba predstavlja sekvencu regularnih instrukcija koja se definiše jedanput a može se koristiti više puta u toku izvornog programa.

Svaka makro naredba sastoji se od 3 celine:

- *zaglavlje* predstavlja zapravo ime makro naredbe sa zadatim formalnim parametrima (npr. addx2 $t2, $t1, $t0)
- *telo* čini sekvenca reglarnih instrukcija u koju asembler prevodi makro pri pozivu
- *direktiva koja se odnosi na kraj definicije*

Pri pozivu makroa zadaje se ime i aktuelni parametri.

Kod MIPS mikroprocesora makro naredbe imaju sledeću sintaksu:

```
.macro ime naredbe ($parametri)
 sekvenca instrukcija koje čine makro
 .end_macro
```

Primer:

definisanje makroa:

```
.macro addx2 ($rd, $rs, $rt)
 sll    $1, $rs, 1
addu $rd, $1, $rt
.end_macro
```

poziv makroa:

```
 addx2 ($t2, $t1, $t0)
```

Add with Shift by 1	**addx2 Rd, Rs, Rt**	2/2

Sadržaj registra Rs se množi sa 2, pa se sabira sa sadržajem registra Rt. Dobijeni rezultat se upisuje u registar Rd.

```
sll $1, Rs, 1
addu Rd, $1, Rt
```

Primer:

```
ori $t0, $0, 8                # $t0 = 8
ori $t1, $0, 1                # $t1 = 1

sll $1, $t0, 1                # addx2 $t2, $t1, $t0
addu $t2, $1, $t1             # $t2 = 17
```

Add with Shift by 2	**addx4 Rd, Rs, Rt**	2/2

Sadržaj registra Rs se množi sa 4, pa se sabira sa sadržajem registra Rt. Dobijeni rezultat se upisuje u registar Rd.

```
sll $1, Rs, 2
addu Rd, $1, Rt
```

Primer:

```
ori $t0, $0, 8          # $t0=8
ori $t1, $0, 1          # $t1=1

sll $1, $t0, 2          # addx4 $t2, $t1, $t0
addu $t2, $1, $t1       # $t2 = 33
```

Add with Shift by 3	**addx8 Rd, Rs, Rt**	2/2

Sadržaj registra Rs se množi sa 8, pa se sabira sa sadržajem registra Rt. Dobijeni rezultat se upisuje u registar Rd.

```
sll $1, Rs, 3
addu Rd, $1, Rt
```

Primer:

```
ori $t0, $0, 8          # $t0=8
ori $t1, $0, 1          # $t1=1

sll $1, $t0, 3          # addx8 $t2, $t1, $t0
addu $t2, $1, $t1       # t2 = 65
```

Branch if All Bits Set	**ball Rs, Rt, Label**	3/3

Ukoliko su postavljeni svi bitovi u registru Rs označeni maskom u registru Rt vrši se grananje na adresu Label.

```
nor $1, Rs, $0
and $1, $1, Rt
beq $1, $0, Label
```

Primer:

```
ori $t1, $0, 12         # $t1 = 1100
ori $t0, $0, 11         # $t0 = 1011

nor $1, $t1, $0         # ball $t1, $t0, lab
and $1, $1, $t0
beq $1, $0, lab
sll $0, $0, 0

ori $t2, $0, 1          #  ako nije doslo do
                        #  grananja $t2=1
```

```
lab:
```

Branch if Any Bit Set	**bany Rs, Rt, Label**	2/2

Ukoliko je postavljen bar jedan bit registra Rs označen maskom u registru Rt vrši se grananje na adresu Label.

```
and $1, Rs, Rt
bne $1, $0, Label
```

Primer:

```
ori $t1, $0, 12          # $t1 = 1100
ori $t0, $0, 11          # $t0 = 1011

and $1, $t1, $t0         # bany $t1, $t0, lab
bne $1, $0, lab
sll $0, $0, 0

ori $t2, $0, 1           #  ako nije doslo do
                         #  grananja $t2=1
lab:
```

Branch if Bit Clear	**bbc Rs, Rt, Label**	4/4

Ukoliko je bit određen sadržajem registra Rt obrisan u registru Rs vrši se grananje na adresu Label.

```
ori $1, $0, 1
sllv $1, $1, Rt
and $1, Rs, $1
beq $1, $0, Label
```

Primer:

```
ori $t1, $0, 12          # $t1 = 0000000000001100
ori $t0, $0, 3           # $t0 = 3

ori $1, $0, 1            # bbc $t1, $t0, lab
sllv $1, $1, $t0
and $1, $t1, $1
beq $1, $0, lab          # nema grananja
sll $0, $0, 0

ori $t2, $0, 1           # $t2=1
lab:
```

Branch if Bit Clear Immediate	**bbci Rs, pos, Label**	4/4

Ukoliko je bit određen neposrednim operandom pos obrisan u registru Rs vrši se grananje na adresu Label.

```
ori $1, $0, 1
sll $1, $1, pos
and $1, Rs, $1
beq $1, $0, Label
```

Primer:

```
ori $t1, $0, 12          # $t1 = 0000000000001100

ori $1, $0, 1            # bbci $t1, 3, lab
sll $1, $1, 3
and $1, $t1, $1
beq $1, $0, lab          # nema grananja

sll $0, $0, 0
ori $t2, $0, 1           # $t2=1
lab:
```

Branch if Bit Set	**bbs Rs, Rt, Label**	4/4

Ukoliko je bit određen sadržajem registra Rt postavljen u registru Rs vrši se grananje na adresu Label.

```
ori $1, $0, 1
sllv $1, $1, Rt
and $1, Rs, $1
bne $1, $0, Label
```

Primer:

```
ori $t1, $0, 12          # $t1 = 0000000000001100
ori $t0, $0, 3           # $t0 = 3

ori $1, $0, 1            # bbs $t1, $t0, lab
sllv $1, $1, $t0
and $1, $t1, $1
bne $1, $0, lab          # vrši se grananje

sll $0, $0, 0
ori $t2, $0, 1           # ne izvršava se jer je
                         #  došlo do grananja
lab:
```

Branch if Bit Set Immediate	**bbsi Rs, pos, Label**	4/4

Ukoliko je bit određen neposrednim operandom pos postavljen u registru Rs vrši se grananje na adresu Label.

```
ori $1, $0, 1
sll $1, $1, pos
```

```
and $1, Rs, $1
bne $1, $0, Label
```

Primer:

```
ori $t1, $0, 12          # $t1 = 0000000000001100

ori $t3, $0, $0, 1       # bbsi $t3, 3, lab
sll $t3, $t3, 3
and $t3, $t1, $t3
bne $t3, $0, lab         # vrši se grananje

sll $0,$0, 0
ori $t2, $0, 1           # ne izvršava se jer je
                         # doslo do grananja
lab:
```

Branch if Not All Bits Set	**bnall Rs, Rt, Label**	3/3

Ako u registru Rs nisu postavljeni svi bitovi određeni maskom u registru Rt vrši se grananje na adresu Label.

```
nor $1, Rs, $0
and $1, $1, Rt
bne $1, $0, Label
```

Primer:

```
ori $t1, $0, 12          # $t1 = 1100
ori $t0, $0, 11          # $t0 = 1011

nor $1, $t1, $0          # bnall $t1, $t0, lab
and $1, $1, $t0
bne $1, $0, lab          # vrši se grananje

sll $0, $0, 0
ori $t2, $0, 1           # ne izvršava se jer je
                         # doslo do grananja
lab:
```

Branch if No Bit Set	**bnone Rs, Rt, Label**	2/2

Ako u registru Rs nijedan od bitova određenih maskom u registru Rt nije postavljen vrši se grananje na adresu Label.

```
and $1, Rs, Rt
beq $1, $0, Label
```

Primer:

```
ori $t1, $0, 4           # t1 = 0100
ori $t0, $0, 9           # t2 = 1001
```

```
and $1, $t1, $t0          # bnone $t1, $t0, lab
beq $1, 40, lab

sll $0, $0,0
ori $t2, $0, 2
lab:
```

Branch if Equal Immediate	**beqi Rs, value, Label**	3/3

Ukoliko je sadržaj registra Rs jednak neposrednom operandu value vrši se grananje na adresu Label.

```
lui $1, HI(value)
ori $1, $1, LO(value)
beq Rs, $1, Label
```

Primer:

```
lui $1, 0x12           # beqi $t1, 0x123456, lab
ori $t1, $1, 0x3456
beq $t1,$1, lab

sll $0, $0, 0
ori $t2, $0, 2          # provera grananja
lab:
```

Branch if Not Equal Immediate	**bnei Rs, value, Label**	3/3

Ukoliko je sadržaj registra Rs različit od neposrednog operanda value vrši se grananje na adresu Label.

```
lui $1, HI(value)
ori $1, $1, LO(value)
bne Rs, $1, Label
```

Primer:

```
lui $1, 0x12            # bnei $t1, 0x123456, lab
ori $t1, $1, 0x3456
bne $t1, $1, lab

sll $0, $0, 0
ori $t2, $0, 2          # provera grananja
lab:
```

Maximum Value	**max Rd, Rs, Rt**	4/4

U registar Rd se upisuje veća od vrednosti u registrima Rs i Rt. Sadržaji registara Rs i Rt se porede kao označeni brojevi.

```
slt $1, Rs, Rt
beq $1, $0, skip
```

```
          or Rd, Rs, $0
          or Rd, Rt, $0
skip:
```

Primer:

```
lui $1, 0xffff
ori $t0, $1, 0xfffb      # t0 = -5
ori $t1, $0, 7           # t1 = 7

slt $1, $t1, $t0         # max $t2, $t1, $t0
beq $1, $0, labela
or $t2, $t1, $0          # u slotu grananja
or $t2, $t0, $0
labela:
```

Maximum Value Unsigned	**maxu Rd, Rs, Rt**	4/4

U registar Rd se upisuje veća od vrednosti u registrima Rs i Rt. Sadržaji registara Rs i Rt se porede kao neoznačeni brojevi.

```
          sltu $1, Rs, Rt
          beq $1, $0, skip
          or Rd, Rs, $0
          or Rd, Rt, $0
skip:
```

Primer:

```
ori $t0, $0, 7           # t0 = 7
ori $t1, $0, 5           # t1 = 5

sltu $1, $t1, $t0        # maxu $t2, $t1, $t0
beq $1, $0, labela
or $t2, $t1, $0          # u slotu grananja
or $t2, $t0, $0
labela:
```

Minimum Value	**min Rd, Rs, Rt**	4/4

U registar Rd se upisuje manja od vrednosti u registrima Rs i Rt. Sadržaji registara Rs i Rt se porede kao označeni brojevi.

```
          slt $1, Rs, Rt
          bne $1, $0, skip
          or Rd, Rs, $0
          or Rd, Rt, $0
skip:
```

Primer:

```
lui $1, 0xffff
```

```
ori $t0, $1, 0xfffb      # t0 = -5
ori $t1, $0, 7           # t1 = 7

slt $1, $t1, $t0         # min $t2, $t1, $t0
bne $1, $0, labela
or $t2, $t1, $0          # u slotu grananja
or $t2, $t0, $0
labela:
```

Minimum Value Unsigned	**minu Rd, Rs, Rt**	4/4

U registar Rd se upisuje manja od vrednosti u registrima Rs i Rt. Sadržaji registara Rs i Rt se porede kao neoznačeni brojevi.

```
          sltu $1, Rs, Rt
          bne $1, $0, skip
          or Rd, Rs, $0
          or Rd, Rt, $0
          skip:
```

Primer:

```
ori $t0, $0, 7           # t0 = 7
ori $t1, $0, 5           # t1 = 5

sltu $1, $t1, $t0        # minu $t2, $t1, $t0
bne $1, $0, labela
or $t2, $t1, $0          # u slotu grananja
or $t2, $t0, $0
labela:
```

Move if Equal to Zero	**moveqz Rd, Rs, Rt**	3/3

Ako je sadržaj registra Rt jednak nuli, registar Rs se kopira u Rd.

```
          bne Rt, $0, skip
          sll $0, $0, 0
          addu Rd, $0, Rs
skip:
```

Primer:

```
ori $t1, $0, 5           # $t1=5
ori $t0, $0, 0           # $t0=0

bne $t0, $0, skip        # moveqz $t2, $t1, $t0
sll $0, $0, 0
addu $t2, $0, $t1
skip:
```

Move if Greater Than or Equal to Zero	**movgez Rd, Rs, Rt**	3/3

Ako je sadržaj registra Rt veći ili jednak nuli, registar Rs se kopira u Rd.

```
          bltz Rt, skip
          sll $0, $0, 0
          addu Rd, $0, Rs
skip:
```

Primer:

```
ori $t0, $0, 0          # $t0=0
ori $t1, $0, 5          # $t1=5

bltz $t1, skip          # movgez $t2, $t1, $t0
sll $0, $0, 0
add $t2, $t1, 0
skip:
```

Move if Less Than Zero	**movltz Rd, Rs, Rt**	3/3

Ako je sadržaj registra Rt manji od nule, registar Rs se kopira u Rd.

```
          bgez Rt, skip
          sll $0, $0, 0
          addu $t2, $t1, 0
          skip:
```

Primer:

```
ori $t0, $0, 0          # $t0=0
ori $t1, $0, 5          # $t1=5

bgez $t0, skip          # movltz $t2, $t1, $t0
sll $0, $0,0
add $t2, $t1, 0
skip:
```

Move if Not Equal to Zero	**movnez Rd, Rs, Rt**	3/3

Ako je sadržaj registra Rt različit od nule, registar Rs se kopira u Rd.

```
          beq Rt, skip
          sll $0, $0, 0
          addu $t2, $t1, 0
skip:
```

Primer:

```
ori $t1, $0, 6          # $t1=6
ori $t0, $0, 3          # $t0=3

beq $t0, $0, skip       # movltz $t2, $t1, $t0
```

```
sll $0, $0, 0
add $t2, $t1, 0
skip:
```

Sign Extend	**sext Rd, Rs, pos**	2/2

Uzima se sadržaj registra Rs, pa se bit određen neposrednim operandom pos kopira u sve više pozicije (od 31 do pos+1). Dobijeni rezultat se upisuje u registar Rd.

```
sll Rd, Rs, 31-pos
sra Rd, Rd, 31-pos
```

Primer:

```
ori $t1, $0, 0xFFFC      # t1 = 0x0000FFFC

sll $t2, $t1, 16         # sext $t2, $t1, 15
sra $t2, $t2, 16
                         # t1 = 0xFFFFFFFC (-4)
```

Subtract with Shift by 1	**subx2 Rd, Rs, Rt**	2/2

Sadržaj registra Rs se množi sa 2, pa se od njega oduzima sadržaj registra Rs. Dobijeni rezultat se upisuje u registar Rd.

```
sll $1, Rs, 1
subu Rd, $1, Rt
```

Primer:

```
ori $t1, $0, 10          # $t1=10
ori $t0, $0, 5           # $t0=5

sll $1, $t1, 1           # subx2 $t2, $t1, $t0
subu $t2, $1, $t0        # t2 = 25
```

Subtract with Shift by 2	**subx4 Rd, Rs, Rt**	2/2

Sadržaj registra Rs se množi sa 4, pa se od njega oduzima sadržaj registra Rs. Dobijeni rezultat se upisuje u registar Rd.

```
sll $1, Rs, 2
subu Rd, $1, Rt
```

Primer:

```
ori $t1, $0, 1           # $t1=1
ori $t0, $0, 5           # $t0=5

sll $1, $t1, 2           # subx4 $t2, $t1, $t0
subu $t2, $1, $t0        # t2 = -1
```

Subtract with Shift by 3	**subx8 Rd, Rs, Rt**	2/2

Sadržaj registra Rs se množi sa 8, pa se od njega oduzima sadržaj registra Rs. Dobijeni rezultat se upisuje u registar Rd.

```
sll $1, Rs, 3
subu Rd, $1, Rt
```

Primer:

```
ori $t1, $0, 1          # $t1=1
ori $t0, $0, 5          # $t0=5

sll $1, $t1, 3          # subx8 $t2, $t1, $t0
subu $t2, $1, $t0       # t2 = 3
```

5 Zadatak

5.1 Predmet rada

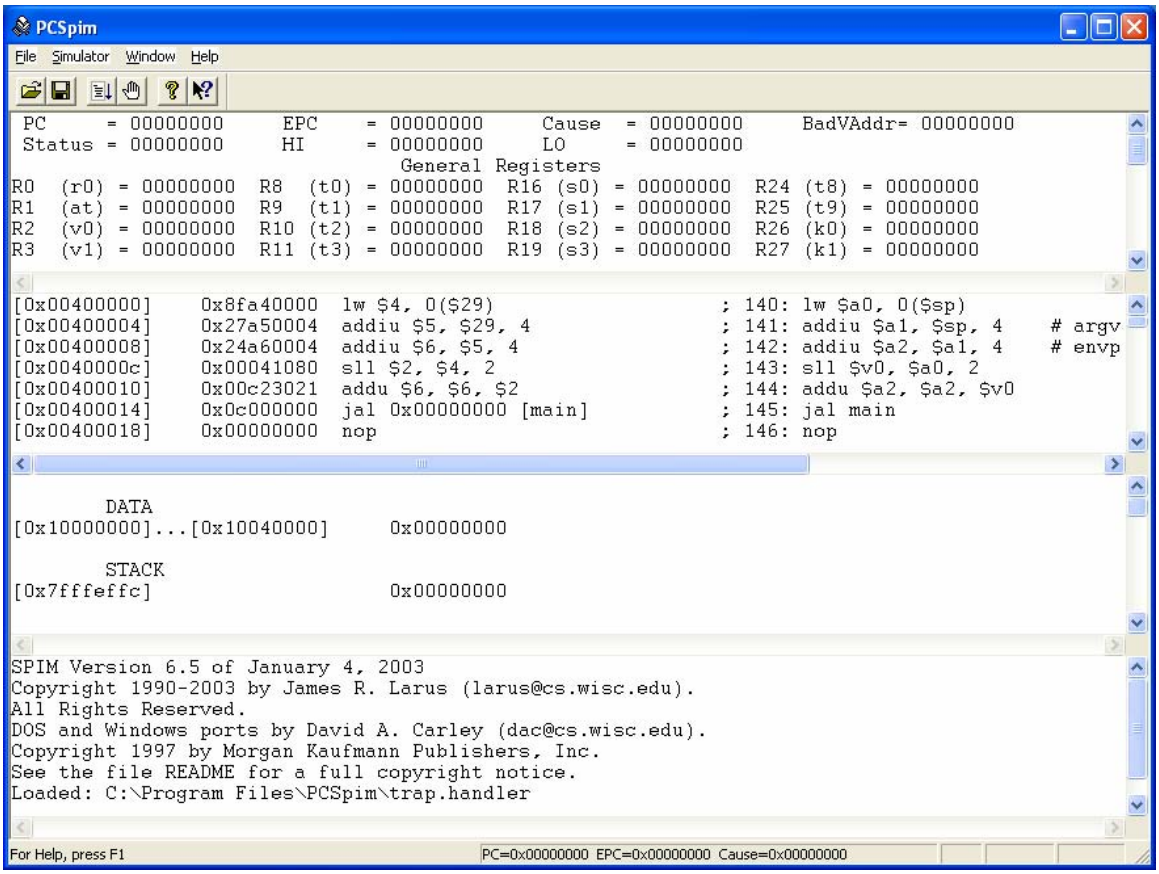
Na praktičnim primerima, upoznati se sa načinom korišćenja pseudo instrukcija i načinom definisanja i korišćenja makro naredbi pri pisanju programa na asemblerskom jeziku MIPS procesora. Kao emulator (program koji simulira izvršenje programa na MIPS procesoru) koristi se PCSpim.

5.2 Postupak rada

korak 1: Pokrenuti *MIPSter* (*Start→All Programs→MIPSter*). Uneti jedan od primera iz sekcije 5.3. i sačuvati ga kao pod nazivom "primer.s".

korak 2: Pokrenuti program PCSpim (*Start→All Programs→PCSpim*). Otvoriće se aplikaconi prozor koji je prikazan na slici 7. Aplikacioni prozor sastoji se iz četiri dela:

- sekcija na vrhu je *meny-bar*. Omogućava selekciju *File* operacije, postavljanje konfiguracije-opcijom *Simulator*, selekciju načina i vrste prikaza-opcijom *Windows* i dobijanje pomoćnih informacija-opcijom *Help.*
- odmah ispod *meny-bar*-a je *toolbar.* Omogućava brzi bristup preko miša do svih alata koje koristi PCSpim.
- najveća sekcija u sredini ekrana je sekcija prikaza koja ima četiri celine:*Register* (vrednosti svih registara CPU i MPU jedinica MIPS procesora), *Text Segment* (instrukcije iz korisničkog programa i sistemski kod), *Data Segment* (podaci koji se nalaze u memoriji i kojima korisnički program manipuliše) i *Messages* (ispisuju se pouke PCSpim-a).
- na dnu velikog prozora nalazi se sekcija *Status bar.* Ova sekcija sadrži informacije o tekućim aktivnostima istatusu simulatora.

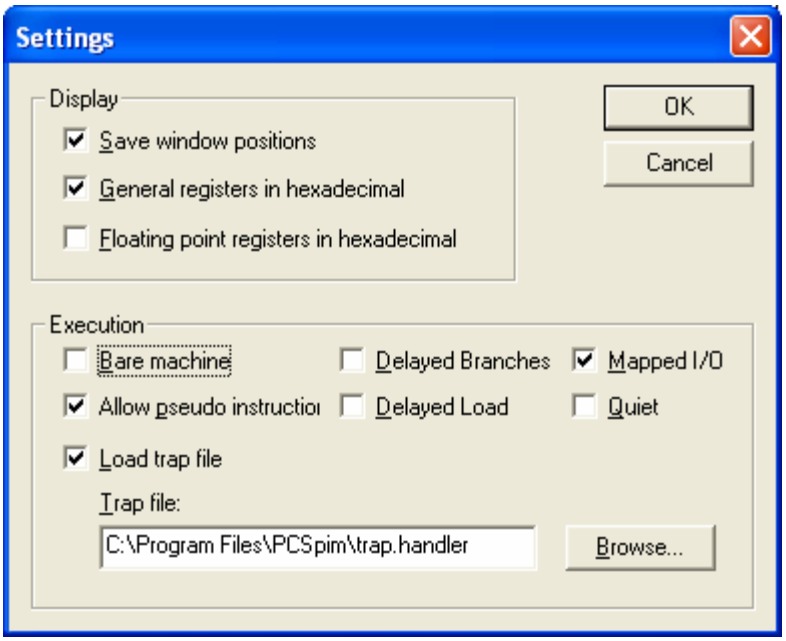


Slika 7. Aplikacioni prozor programa PCSpim

korak 3: Postaviti konfiguraciju simulatora selektovanjem *Simulator→Settings* iz *menu bar*-a tako da *dialog box* koji se otvori izgleda kao na slici 8.

korak 4: Loadovati fajl primer.s selekcijom *Open* tastera iz *toolbar*-a (alternativno može se selektovati *File→Open* iz *menu bar*-a). Ukoliko program ne može da se loaduje PCSpim će pružiti mogućnost promene konfiguracje okoline i automatski će izvršiti *reload* fajla.

Po učitavanju *dialog box* za otvaranje fajlova se zatvara i pojavljuje se glavni aplikacioni prozor sa prikazima instrukcija i podataka. Ako to nije slučaj treba promeniti izgled ekrana selekcijom Windows→*Tile* iz *menu bar*-a.



Slika 8. *Dialog box* programa PCSpim za konfigurisanje okruženja

Svaka instrukcija u tekstualnom segmentu prikazana je u liniji sličnoj sledećoj:

```
[0x0040000c]    0x00041080    sll $2, $4, 2        ; 143: sll $v0, $a0, 2
```

Prvi broj u liniji, smešten između uglastih zagrada je heksadecimalna memorijska adresa instrukcije. Drugi broj je heksadecimalni kod instrukcije. Iza tačke i zareza sledi linija iz fajla koja generiše instrukciju, a broj 143 je broj linije u fajlu. Ako iza tačke i zareza nema nikakve linije znači da je instrukciju generisao PCSpim u procesu prevođenja pseudonstrukcije ili makro naredbe.

korak 5: Pratiti izvršavanje programa uz pomoć tastera F10 ili selektovati *Simulator→Single _Step* iz *menu bar*-a (program se izvršava instrukcija po instrukcija). Izvršenje programa postiže se i pritiskom *Go* tastera iz *toolbar*-a (alternativno može se selektovati *Simulator→Go iz menu bar-a).*

5.3 Primeri

1. Dat je niz *a* od 6 elemenata tipa reč (32 bita). Generisati nizove *b* i *c* čiji su elementi tipa polureč (16 bitova) tako što se u niz *b* smešta niža, a u niz *c* viša polureč odgovarajućeg elementa niza *a*. Pretpostaviti da su nizovi *a*, *b* i *c* smešteni počev od adresa NizA, NizB i NizC, respektivno, pri čemu ove adrese ne moraju biti poravnate.

Rešenje:

```
        la $t0, NizA
        addi $t0, $t0, 24   # kraj niza a
        la $t1, NizB
        addi $t1, $t1, 12   # kraj niza b
        la $t2, NizC
        addi $t2, $t2, 12   # kraj niza c
        la $t7, NizA

petlja: addi $t0, $t0, -4    # pomeramo se za
        addi $t1, $t1, -2    # jedan element unazad
        addi $t2, $t2, -2    # u svim nizovima
        ulw $t5, 0($t0)   # ucitaj rec
        ush $t5, 0($t1)   # sacuvaj nizu polurec u b
        srl $t5, $t5, 16
        ush $t5, 0($t2)   # sacuvaj visu polurec u c
        bne $t0, $t7, petlja   # dok ne stignemo do
                               # pocetka niza
```

2. Napisati program koji ispituje da li je prirodan broj *n* u registru $t0 prost. Ukoliko jeste, u registar $t5 upisati 1, a u suprotnom upisati 0.

Rešenje:

```
        li $t0, 35973     #$t0-broj koji ispitujemo
        li $t1, 2         # $t1 – broj kojim delimo
        srl $t2, $t0, 1   # $t2 = n/2

petlja: remu $t3, $t0, $t1      # $t3 = n mod $t1
        beqz $t3, slozen
        addi $t1, $t1, 1        # u slotu grananja
        bleu $t1, $t2, petlja   # if $t1 <= n/2

        li $t5, 1               # prost
        b kraj
        nop

slozen: move $t5, $0
kraj:
```

3. Proizvod dva označena broja u registrima $t1 i $t2 smestiti u registar $t3. Pri tome ne koristiti instrukcije za množenje označenih brojeva. Ako dođe do prekoračenja generisati izuzetak.

Rešenje:

```
        slt $t5, $t1, $0     # 1 ako je $t1 negativno
        slt $t6, $t2, $0     # 1 ako je $t2 negativno
        abs $t7, $t1
        abs $t8, $t2
        mulou $t3, $t7, $t8 # $t3=abs($t1)*abs($t2)
        xor $t5, $t5, $t6    # odredi znak proizvoda
```

```
            beqz $t5, kraj
            nop

            neg $t3, $t3        # $t3 = -$t3
kraj:
```

4. Dati su nizovi *a* i *b* čiji su elementi 16-bitni označeni brojevi. Formirati niz c po sledećoj formuli:

$$c_i = 4a_i + b_i, \qquad (i = 0, 1, ..., 7)$$

Elementi niza *c* su 32-bitni označeni brojevi. Pretpostaviti da su nizovi *a*, *b* i *c* smešteni počev od adresa NizA, NizB i NizC, respektivno.

Rešenje:

```
            move $t1, $0    # indeks za nizove a i b
            move $t2, $0    # indeks za niz c
petlja:     lh $t5, NizA($t1)
            lh $t6, NizB($t1)
            addx4 $t7, $t5, $t6
            sw $t7, NizC($t2)
            addi $t1, $t1, 2
            addi $t2, $t2, 4
            bnei $t1, 16, petlja
            nop
```

5. U registru $t1 smešten je prirodan broj *n*. Izračunati $m = 10n$ ne koristeći instrukcije za množenje. Ukoliko dobijeni broj *m* nije veći od 120 dodati mu 80 i konačnu vrednost za *m* smestiti u registar $t2.

Rešenje:

```
            sll $t0, $t1, 1         # $t0 = 2n
            addx8 $t2, $t1, $t0     # $t2 = 8n + $t0
            li $t0, 120
            bgtu $t2, $t0, kraj     # ako je $t2 >= 120
            nop
            addi $t2, $t2, 80
kraj:
```

6. Počev od adrese Niz smešten je niz od osam 16-bitnih označenih brojeva. Izračunati sumu svih elemenata ovog niza i 32-bitni rezultat upisati na lokaciju Rez. Pri tome ne koristiti instrukciju `lh` (load halfword).

Rešenje:

```
            move $t1, $0        # suma
            li $t0, 16          # 8 * 2
petlja:     addi $t0, $t0, -2   # predji na prethodni elem.
            lhu $t2, Niz($t0)
            sext $t2, $t2, 15   # prosiri sa 16 na 32 bita
            bnez $t0, petlja
```

```
        add $t1, $t1, $t2   # dodaj u sumu
                            # add je u slotu grananja

        sw $t1, Rez($0)     # sacuvaj rezultat
```

7. Napisati program koji određuje koliko ima brojeva deljivih sa 8 u nizu od 10 elemenata. Niz je smešten počev od adrese Br, a njegovi elementi su tipa reč. Rezultat upisati u registar $t5.

   Rešenje:

```
        move $t5, $0        # brojac = 0
        li $t2, 7           # maska = 00...0111
        li $t0, 40          # 10 * 4
petlja: addi $t0, -4
        lw $t3, Br($t0)
        bany $t3, $t2, dalje # ako je neki od poslednja
        nop                  # tri bita postavljen
        addi $t5, $t5, 1     # broj nije deljiv sa 8, pa se
dalje:  bnez $t0, petlja     # preskace inkrementiranje
```

8. Dati su nizovi *a* i *b* čiji su elementi neoznačeni 32-bitni celi brojevi. Formirati niz *c* čiji su elementi

$$c_i = \max(a_i, b_i)$$

   Nizovi *a*, *b* i *c* su smešteni počev od adresa NizA, NizB i NizC, respektivno, koje ne moraju da budu poravnate. Broj elemenata smešten je u registru $t7.

   Rešenje:

```
        sll $t0, $t7, 2     # $t0 = $t7 * 4
petlja: addi $t0, -4
        ulw $t1, NizA($t0)
        ulw $t2, NizB($t0)
        maxu $t3, $t1, $t2  # $t3 = max($t1, $t2)
        usw $t3, NizC($t0)
        bnez $t0, petlja
```

9. Data u dva niza *a* i *b* čiji su elementi neoznačeni 16-bitni brojevi. Formirati niz *c* čiji su elementi veličine jednog bajta, a dobijaju se po sledećoj formuli:

$$c_i = \begin{cases} 8, & a_i \geq b_i \\ 0, & a_i < b_i \end{cases}$$

   Pretpostaviti da su nizovi smešteni na adresama NizA, NizB i NizC i da imaju po 6 elemenata.

   Rešenje:

```
        li $t0, 12          # za nizove a i b
        li $t9, 6           # za niz c
petlja: addi $t0, $t0, -2       # predji na prethodni
        addi $t9, $t9, -1       #    element
```

```
        lhu $t1, NizA($t0)
        lhu $t2, NizB($t0)
        sgeu $t3, $t1, $t2   # $t3 = 0 ili 1
        sll $t3, $t3, 3      # $t3 = $t3 * 8
        sb $t3, NizC(t9)
        bnez $t0, petlja
```

10. Dati su nizovi *a* i *b* čiji su elementi 32-bitni označeni brojevi. Formirati niz c po sledećoj formuli:

$$c_i = a_i - 8b_i, \quad (i = 0, 1, ..., 9)$$

Elementi niza *c* su 32-bitni označeni brojevi. Pretpostaviti da su nizovi *a*, *b* i *c* smešteni počev od adresa NizA, NizB i NizC, respektivno.

Rešenje:

```
        move $t0, $0           # indeks za nizove a, b i c
petlja: lw $t5, NizA($t0)
        lw $t6, NizB($t0)
        subx8 $t7, $t6, $t5    # $t7 = 8b – a
        neg $t7, $t7           # $t7 = a – 8b
        sw $t7, NizC($t0)
        addi $t0, $t0, 4
        bnei $t0, 40, petlja
        nop
```

11. Modifikovati elemente niza *a* tako da se element a_i rotira za *i* mesta ulevo. Niz *a* sadrži 10 elemenata ($i = 0, 1, ..., 9$) tipa reč, a smešten je počev od adrese NizA.

Rešenje:

```
        move $t0, $0         # za adresiranje elemenata
        move $t1, $0         # i
petlja: lw $t2, NizA($t0)
        rol $t2, $t2, $t1
        sw $t2, NizA($t0)
        addi $t0, $t0, 4
        addi $t1, $t1, 1
        bnei $t1, 10, petlja
        nop
```

12. Dat je niz *a* čiji su elementi 32-bitni neoznačeni celi brojevi. Formirati niz *b* čiji su elementi takođe 32-bitni neoznačeni brojevi na sledeći način: ukoliko je viša polureč elementa a_i jednaka nuli onda je $b_i = a_i$, a u suprotnom se b_i dobija rotacijom a_i za 16 mesta udesno. Broj elemenata nizova *a* i *b* nalazi se u registru $t7, a smešteni su počev od adresa NizA i NizB, respektivno.

Rešenje:

```
        li $t4, 0xFFFF0000       # maska za visa dva bajta
        move $t0, $t7
```

```
            sll $t0, $t0, 2           # $t0 = $t7 * 4
petlja:     addi $t0, $t0, -4
            lw $t1, NizA($t0)
            bnone $t1, $t4, sacuvaj   # ako su visa dva bajta cista,
            nop                       # preskoci rotaciju
            ror $t1, $t1, 16
sacuvaj:    sw $t1, NizB($t0)
            bnez $t0, petlja
            nop
```

13. Na osnovu nizova *a* i *b* generisati niz *c* po sledećoj formuli:

$$c_i = \begin{cases} a_i / b_i, & b_i \neq 0 \\ a_i / 25, & b_i = 0 \end{cases}$$

Nizovi *a*, *b* i *c* smešteni su na adresama NizA, NizB i NizC, i imaju po 10 elementa tipa reč. Svi brojevi su označeni.

Rešenje:

```
            move $t0, 40
            li $t5, 25
petlja:     addi $t0, $t0, -4
            lw $t1, NizA($t0)
            lw $t2, NizB($t0)
            moveqz $t2, $t5, $t2    # ako je $t2 = 0
                                    # zameni ga sa 25
            div $t3, $t1, $t2       # $t3 = $t1 / $t2
            sw $t3, NizC($t0)
            bnez $t0, petlja
            nop
```

14. Dati su nizovi *a* i *b* smešteni počev od adresa NizA i NizB čiji su elementi označeni brojevi tipa reč. Zameniti vredonsti a_i i b_i tamo gde je potrebno tako da važi $a_i \leq b_i$ za $i = 0, 1, ..., 7$. Ne koristiti instrukcije grananja.

Rešenje:

```
            li $t0, 32          # 8 * 4
petlja:     addi $t0, $t0, -4
            lw $t1, NizA($t0)
            lw $t2, NizB($t0)
            min $t3, $t1, $t2
            max $t4, $t1, $t2
            sw $t3, NizA($t0)
            sw $t4, NizB($t0)
            bnez $t0, petlja
            nop
```

15. Odrediti koliko elemenata niza *a* daje ostatak 3 pri deljenju sa 4. Niz ima 10 elemenata tipa polureč i smešten je počev od adrese Niz. Rezultat smestiti u registar $t7.

Rešenje:

```
          li $t0, 20            # 10 * 2
          move $t7, $0          # brojac / rezultat
          li $t2, 3             # maska (0...011)
petlja:   addi $t0, $t0, -2
          lh $t1, Niz($t0)
          bnall $t1, $t2, dalje   # ako najniza dva bita nisu
          nop                     #  postavljena broj ne ispunjava
          addi $t7, $t7, 1        #  dati uslov
dalje:    bnez $t0, petlja
          nop
```

16. U nizu *a* sve negativne elemente zameniti nulom, a pozitivnim elementima promeniti znak. Elementi niza su označeni 16-bitni brojevi. Broj elemenata niza nalazi se na adresi BrElem (podatak tipa bajt), a sam niz smešten je počev od adrese Niz koja ne mora da bude poravnata.

Rešenje:

```
          lbu $t0, BrElem($0)
          sll $t0, $t0, 1       # $t0 = BrElem * 2
petlja:   addi $t0, $t0, -2
          ulh $t1, Niz($t0)
          movltz $t1, $0, $t1
          neg $t1, $t1
          ush $t1, Niz($t0)
          bnez $t0, petlja
          nop
```

Literatura

1. "RISC, CISC i DSP procesori" , Mile Stojčev
2. "Izabrani zadaci iz mikroprocesorskih sistema" , Mile Stijčev i Tatjana Stanković
3. "MIPS32® Architecture For Programmers Volume I: Introduction to the MIPS32® Architecture" , MIPS Technologies, INC
4. "MIPS32® Architecture For Programmers Volume II: The MIPS32™ Instruction Set" , MIPS Technologies, INC
5. "Assemblers, Linkers, and the SPIM Simulator" , James R. Larus Microsoft Research
6. "Laboratorijski praktikum iz predmeta mikroprocesorski sistemi" , T.Stanković, S.Ristić, M.Krstić, I.Andrejić, M.Stojčev