# 1. UVOD

## 1.1. POJAM ARHITEKTURE RAČUNARA

**Arhitektura računara** (*computer architecture*) se bavi problemima upotrebe i pravljenja računara.

Posmatrano sa stanovišta arhitekture računara, upotreba računara se svodi na njegovo programiranje, jer je namena računara da izvršava programe. Način programiranja zavisi od osobina **skupa naredbi** računara. Ovim osobinama se bavi **arhitektura naredbi** (*instruction set architecture*, skraćeno *ISA*).

Cilj pravljenja računara je ostvarenje ili **implementacija** (*implementation*) njegove arhitekture naredbi. Implementacija arhitekture naredbi obuhvata **organizaciju** (*organization*) i **izvedbu** (*hardware*) računara. Organizacija računara se bavi organizacionim komponentama koje obrazuju računar (njihovom namenom i funkcijom), kao i međusobnim odnosima ovih komponenti. Izvedba računara se bavi problemima proizvodnje pomenutih komponenti.

Pojam arhitekture računara obuhvata arhitekturu naredbi i njenu implementaciju, odnosno organizaciju i izvedbu računara.

Između arhitekture naredbi i njene implementacije postoji međuzavisnost, jer implementacija odražava i ograničava arhitekturu naredbi.

## 1.2. MODEL RAČUNARA

Na arhitekturu naredbi utiču programski jezici koji se koriste za programiranje računara. Za tržišno prihvaćene računare je karakteristično da su prilagođeni procedurnim (imperativnim) programskim jezicima. Njihov tipičan predstavnik je programski jezik C.

### PRIMER UPOTREBE PROGRAMSKOG JEZIKA C

Upotreba programskog jezika C može da se ilustruje na primeru nalaženja najvećeg zajedničkog delioca (NZD) dva različita neoznačena broja a i b, za koje važi:

$$a == n \times NZD$$
$$b == m \times NZD$$

Za određivanje najvećeg zajedničkog delioca neoznačenih brojeva a i b potrebno je koeficijente n i m svesti na 1. To se može postići ponavljanjem oduzimanja:

$$|a-b| == |n-m| \times NZD$$

tako da se, nakon svakog oduzimanja, promenljivoj sa većom vrednošću dodeli

razlika. Precizan postupak nalaženja najvećeg zajedničkog delioca opisuje segment programa, izražen programskim jezikom C:

```
unsigned int a = 12;
unsigned int b = 10;
while (a!=b)
 if (a>b)
  a = a-b;
 else
  b = b-a;
```

Po izvršavanju prethodnog segmenta programa važi: a == b == NZD == 2 (radi jednostavnosti, opštost prethodnog segmenta programa je ograničena na određivanje najvećeg zajedničkog delioca neoznačenih brojeva 12 i 10).

## OSOBINE PROCEDURNIH PROGRAMSKIH JEZIKA

Procedurni programski jezici omogućuju opisivanje obrada podataka koji pripadaju **celom**, **realnom**, **znakovnom** ili **logičkom** skupu. Ovi skupovi se nazivaju i **prosti tipovi**, jer se njihove vrednosti ne mogu raščlanjivati na prostije sastojke. Prosti tipovi se nazivaju i **osnovni tipovi** (*fundamental types*), jer predstavljaju osnovu za opisivanje svih obrada podataka. Za opisivanje obrada podataka koriste se operacije procedurnih programskih jezika koje omogućuju rukovanje vrednostima prostih tipova. U ovakve operacije spadaju **aritmetičke**, **relacione** i **logičke operacije**.

Opštost opisima obrada podataka daju **promenljive**. Svaku promenljivu karakterišu njeno ime, tip i vrednost. Promenljivoj se dodeljuje vrednost njenog tipa posredstvom **operacije dodele**. Za opštost opisa obrada podataka su važne i **upravljačke operacije**. Zahvaljujući njima redosled obavljanja operacija nije samo sekvencijalni, nego i alternativni i repetitivni.

Promenljive, vrednosti prostih tipova i operacije (upravljačke i one za rukovanje promenljivim i vrednostima prostih tipova) predstavljaju osnovne elemente procedurnih programskih jezika i ujedno daju uopšteni opis računara. Ovome opisu odgovara **model računara** koji obuhvata **memoriju** i **procesor**. Ovaj model podržava izvršavanje programa koji su izraženi procedurnim programskim jezicima.

## MEMORIJA

Memorija (*memory*) je sastavljena od niza **lokacija**. Lokacije omogućuju predstavljanje promenljivih prostih tipova, ako mogu da sadrže vrednosti prostih tipova (koje se dodeljuju ovim promenljivim). Svaka lokacija poseduje jednoznačnu (numeričku) oznaku ili **adresu** (brojevi 0, 1, 2, ...), po kojoj se razlikuje od drugih lokacija. Pored adrese, lokacija može da poseduje i posebnu (simboličku) oznaku ili **labelu** (*label*). Kada postoji, labela se koristi umesto adrese. Labela odgovara imenu promenljive koju predstavlja labelirana lokacija.

Za lokacije je važna osobina univerzalnosti, koja podrazumeva da svaka lokacija može da sadrži vrednost bilo kog prostog tipa i adresu. Univerzalnost lokacija se postiže **kodiranjem** (predstavljanjem) svih vrednosti realnog, znakovnog i logičkog tipa vrednostima celog tipa (podskupom celih brojeva, koji pripadaju celom tipu). Zbog univerzalnosti lokacija, javlja se problem višeznačne interpretacije sadržaja lokacija.

Preuzimanje vrednosti promenljive, kao i dodela vrednosti promenljivoj, podrazumeva pristupanje lokacijama, radi **čitanja** (preuzimanja), ili **pisanja** (izmene) sadržaja lokacija.

## PROCESOR

Procesor (*processor*) je sastavljen od **sklopova** i **registara**. Sklopovi omogućuju izvršavanje pojedinih operacija (namenjenih za rukovanje vrednostima prostih tipova), a registri sadrže vrednosti prostih tipova koje se obrađuju u toku izvršavanja pojedinih operacija. Registri se razlikuju od memorijskih lokacija po tome što se nalaze u procesoru i po tome što nisu označeni adresom ili labelom, nego posebnom oznakom.

Pre izvršavanja operacije, procesor preuzima **oznaku operacije** i njene **operande**. Operandi su ili vrednosti prostih tipova na koje se operacija odnosi, ili adrese lokacija čiji sadržaji se čitaju ili pišu u toku izvršavanja operacije. Oznaka operacije određuje sklop u kome se izvršava dotična operacija. Oznaka operacije i operandi obrazuju **naredbu** (*instruction*) procesora. Procesor izvršava operacije u toku izvršavanja svojih naredbi. Redosled izvršavanja naredbi je određen programom.

## ADRESIRANJE

Operandi se dele na **ulazne** i **izlazne**. Ulazni operandi određuju (ulazne) vrednosti, koje se obrađuju u toku izvršavanja naredbi. Izlazni operandi određuju (izlazne) lokacije, u koje se smeštaju rezultati izvršavanja naredbi.

Ulazni operand se naziva **neposredni** (*immediate*) operand, kada mu odgovara ulazna vrednost. Neposredni operandi su potrebni, radi konstanti. Ulazni operand se naziva **direktni** (*direct*) operand, kada mu odgovara adresa memorijske lokacije sa ulaznom vrednošću, odnosno, naziva se **registarski** (*register*) operand, kada mu odgovara oznaka registra sa ulaznom vrednošću. Izlazni operand se naziva direktni operand, kada mu odgovara adresa izlazne memorijske lokacije, odnosno, naziva se registarski operand, kada mu odgovara oznaka izlaznog registra. Direktni i registarski operandi su potrebni, radi promenljivih prostih tipova. Ulazni operand se naziva **posredni** (*register indirect*) operand, kada mu odgovara oznaka registra sa adresom memorijske lokacije sa ulaznom vrednošću. Izlazni operand se naziva posredni operand, kada mu odgovara oznaka registra sa adresom izlazne memorijske lokacije. Posredni operandi su potrebni, radi pokazivačkih promenljivih. Ulazni operand se naziva **indeksni** (*indexed*) operand, kada mu

odgovaraju posebna vrednost - indeks i oznaka registra, čiji sadržaj u sumi sa indeksom daje adresu memorijske lokacije sa ulaznom vrednošću. Izlazni operand se naziva indeksni operand, kada mu odgovaraju posebna vrednost - indeks i oznaka registra, čiji sadržaj u sumi sa indeksom daje adresu izlazne memorijske lokacije. Indeksni operandi su potrebni, na primer, radi promenljivih složenih tipova (niz ili slog). Određivanje ulaznih vrednosti, odnosno izlaznih lokacija, se naziva i **adresiranje**, pa tako postoje **neposredno**, **direktno**, **registarsko**, **posredno** i **indeksno adresiranje**.

## SKUP NAREDBI

Skup naredbi procesora se može svesti na podskup aritmetičkih naredbi, kao i na relacione, logičke, upravljačke i naredbe prebacivanja. Od aritmetičkih naredbi dovoljno je da procesor podrži celobrojno sabiranje i oduzimanje, jer se ostale aritmetičke naredbe mogu izraziti pomoću celobrojnog sabiranja i oduzimanja. Zahvaljujući kodiranju vrednosti realnog, znakovnog i logičkog tipa vrednostima celog tipa, relacione naredbe se svode na celobrojno oduzimanje prvog operanda (PO) relacije od drugog (DO) i zaključivanje o relaciji na osnovu vrednosti i predznaka razlike, jer:

1. relacija DO == PO važi, ako je razlika 0,
2. relacija DO != PO važi, ako je razlika različita od 0,
3. relacija DO < PO važi, ako je razlika negativna,
4. relacija DO >= PO važi, ako je razlika pozitivna ili 0,
5. relacija DO > PO važi, ako je razlika pozitivna, a
6. relacija DO <= PO važi, ako je razlika negativna ili 0.

Logičke naredbe obuhvataju logičko i, logičko ili i logičko ne, a upravljačke naredbe bezuslovnu ili uslovnu izmenu redosleda izvršavanja naredbi programa. Naredbe prebacivanja podržavaju dodelu vrednosti promenljivoj tako što omogućuju prebacivanje sadržaja između registara procesora i memorijskih lokacija.

## MAŠINSKE I ASEMBLERSKE NAREDBE

Lokacije mogu da sadrže i naredbe, ako se oznake operacija naredbi (poput njihovih operanada) kodiraju celim brojevima. Ovi brojevi predstavljaju **kodove naredbi**. Kod naredbe sa operandima naredbe obrazuje **mašinsku naredbu**. **Mašinski format naredbe** propisuje kako se interpretiraju cifre celog broja koji predstavlja mašinsku naredbu. Simbolička predstava mašinske naredbe (u kojoj se koriste labele, kao i simboličke oznake za operaciju i operande) se naziva **asemblerska naredba**. Mašinske naredbe pripadaju **mašinskom jeziku**, a asemblerske naredbe pripadaju **asemblerskom jeziku**.

Izvršavanju asemblerskih naredbi obavezno prethodi njihovo prevođenje u mašinski oblik. Prevođenje **asemblerskog programa**, sastavljenog od asemblerskih

naredbi, u **mašinski program**, sastavljen od mašinskih naredbi, obavlja poseban program prevodilac, koji se naziva **asembler**.

Slično asembleru, prevođenje **izvornog programa**, sastavljenog od iskaza procedurnog programskog jezika, u asemblerski (mašinski) program obavlja poseban program prevodilac, koji se naziva **kompajler**.

## 1.3. FIZIČKA OSNOVA MODELA RAČUNARA

Za pravljenje prethodno opisanog modela računara dovoljna je fizička osnova koja omogućuje predstavljanje celih brojeva, jer su vrednosti prostih tipova i delovi naredbi predstavljeni (kodirani) celim brojevima. Cifre celih brojeva se mogu predstaviti raznim nivoima fizičkih veličina, kao što su električni napon ili jačina magnetnog polja. Praktični razlozi sugerišu korišćenje samo 2 nivoa fizičkih veličina, odnosno 2 nivoa signala. To je dovoljno: za prikaz cifara binarnog brojnog sistema (0 : nula, 1 : jedan), ali i za prikaz logičkih vrednosti (0 : netačno, 1 : tačno). Podudarnost cifara binarnog brojnog sistema i logičkih vrednosti omogućuje da se aritmetičke operacije za binarni brojni sistem opišu logičkim funkcijama, ako se binarne cifre interpretiraju kao logičke vrednosti. To potvrđuje tablica sabiranja za binarni brojni sistem (Slika 1.3.1).

| a | b | a+b | Prenos |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Slika 1.3.1 Tablica sabiranja za binarni brojni sistem

Ako se cifre iz tablice sabiranja (Slika 1.3.1) interpretiraju kao logičke vrednosti, a prve dve kolone kao argumenti logičke funkcije, tada treća kolona ove tablice opisuje logičku funkciju isključivo ili (*exclusive or*):

a^b

(oznaka ^ predstavlja operator programskog jezika C za logičku operaciju isključivo ili). Četvrta kolona ove tablice opisuje logičku funkciju i (*and*):

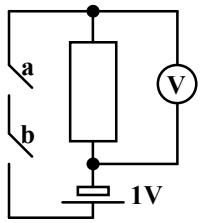
a&b

(oznaka & predstavlja operator programskog jezika C za logičku operaciju i).

Prethodno dozvoljava da se računar napravi kao fizički uređaj koji memoriše logičke vrednosti i obavlja logičke funkcije. Tako, voltmetar iz električnog kola (Slika 1.3.2) pokazuje vrednost logičke funkcije i:

a&b

ako stanja prekidača (0 : otvoren, 1 : zatvoren) zavise od vrednosti logičkih promenljivih a i b.

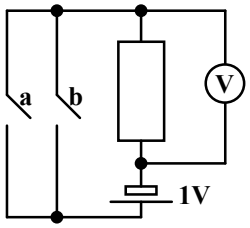



Slika 1.3.2 Električno kolo - ekvivalent logičke funkcije i

Voltmetar iz električnog kola (Slika 1.3.3) pokazuje vrednost logičke funkcije ili (*or*):

a|b

ako stanja prekidača (0 : otvoren, 1 : zatvoren) zavise od vrednosti logičkih promenljivih a i b (oznaka | predstavlja operator programskog jezika C za logičku operaciju ili).

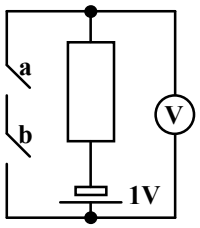



Slika 1.3.3 Električno kolo - ekvivalent logičke funkcije ili

Električna kola omogućuju i ostvarenje logičke negacije. Tako, voltmetar iz električnog kola (Slika 1.3.4) pokazuje vrednost logičke funkcije negirano i (*nand*):

~(a&b)

ako stanja prekidača (0 : otvoren, 1 : zatvoren) zavise od vrednosti logičkih promenljivih a i b (oznaka ~ predstavlja operator programskog jezika C za logičku operaciju negacije).




Slika 1.3.4 Električno kolo - ekvivalent logičke funkcije negirano i

Pomoću logičke funkcije negirano i se mogu izraziti sve ostale logičke funkcije. Ako se kao operator logičke funkcije negirano i odabere oznaka ○:

$$a \circ b \equiv \sim(a \& b)$$

tada sledeći primeri pokazuju kako se pojedine logičke funkcije mogu izraziti pomoću logičke funkcije negirano i:

$$a \,\&\, b \;==\; (a \circ b) \circ (a \circ b)$$
$$a \mid b \;==\; (a \circ a) \circ (b \circ b)$$
$$a \,\hat{}\, b \;==\; ((a \circ a) \circ b) \circ (a \circ (b \circ b))$$
$$a \equiv b \;==\; (a \circ b) \circ ((a \circ a) \circ (b \circ b))$$

Iz prethodnog sledi da se za ostvarenje svih logičkih funkcija može koristiti jedan isti tip električnog kola.

U električnim kolima ulogu prekidača ima tranzistor. Pored ulaznog i izlaznog izvoda, koji su prikazani u sastavu svakog od prekidača na crtežima prethodnih električnih kola, tranzistor ima i treći, upravljački izvod. On određuje provodnost tranzistora, odnosno, stanje prekidača. Na upravljački izvod tranzistora dovodi se, na primer, nivo signala koji odgovara vrednosti neke logičke promenljive, odnosno nivo signala koji odgovara vrednosti neke logičke funkcije, kao što je naponski nivo, prikazan na voltmetrima iz prethodnih električnih kola. Ako se u sastavu prekidača ne prikazuju upravljački izvodi (kao što je to urađeno, radi jednostavnosti, na crtežima prethodnih električnih kola), tada je za svaki prekidač neophodno navesti njegovu upravljačku logičku funkciju, koja se naziva i **prekidačka funkcija**. Podrazumeva se da njena vrednost određuje stanje prekidača, odnosno, da ta vrednost određuje nivo signala na upravljačkom izvodu tranzistora. Argumenti prekidačke funkcije su logičke vrednosti. Oni određuju njenu vrednost, pa, na taj način, omogućuju upravljanje prekidačima. Zato se ovakvi argumenti mogu nazvati **prekidački argumenti**. Na primer, ako logička funkcija:

$$f(a) == a$$

opisuje upravljanje nekim prekidačem, tada od njenog prekidačkog argumenta a zavisi da li je dotični prekidač otvoren ili zatvoren.

Lokacije (i memorijske lokacije i registri procesora) se prave kao nizovi ćelija, nazvanih **biti**. Svakom bitu odgovara poseban fizički uređaj, kao što je flipflop (*flipflop*), sposoban za memorisanje jedne logičke vrednosti, odnosno binarne cifre. Sve lokacije se sastoje od istog i ograničenog broja bita, organizovanih u **bajte** (*byte*) i **reči** (*word*). Ograničena veličina lokacija određuje raspon operanada, odnosno najveći i najmanji operand.

Lokacije se prave pomoću **sekvencijalnih kola** koja omogućuju pamćenje logičkih vrednosti, odnosno binarnih cifara, a sklopovi procesora se prave pomoću **kombinacionih** kola koja ostvaruju pojedine logičke funkcije. Za razliku od kombinacionog kola, čiji izlaz zavisi samo od njegovog ulaza, izlaz sekvencijalnog kola zavisi ne samo od njegovog ulaza, nego i od njegovog stanja.

Iako su principi, na kojima se zasniva rad računara, jednostavni, ipak nije jednostavno od principa stići do pouzdanih i jeftinih komponenti, podesnih za pravljenje praktično upotrebljivog računara. Problemi pravljenja praktično upotrebljivog računara izlaze van okvira ove knjige, jer je ona posvećena samo izgradnji funkcionalno zaokruženog modela računara. Ipak, namena ove knjige je da čitaoca pripremi i za suočavanje sa problemima koji se sreću pri pravljenju praktično upotrebljivog računara.

## 1.4. PITANJA

1. Šta obuhvata pojam arhitekture računara?
2. Koji prosti (osnovni) tipovi postoje?
3. Koji su osnovni elementi procedurnih programskih jezika?
4. Šta obuhvata model računara?
5. Šta karakteriše memoriju?
6. Kako se može interpretirati sadržaj lokacije memorije?
7. Šta karakteriše procesor?
8. Koja adresiranja postoje?
9. Šta ulazi u sastav mašinske naredbe?
10. Šta ulazi u sastav asemblerske naredbe?
11. Kakav zadatak ima asembler?
12. Kakav zadatak ima kompajler?
13. Šta omogućuje podudarnost cifara binarnog brojnog sistema i logičkih vrednosti?
14. Koja logička funkcija opisuje zbir dve binarne cifre?
15. Šta određuje vrednost prekidačke funkcije?
16. Na šta utiče vrednost prekidačkog argumenta?
17. Šta omogućuju sekvencijalna kola?
18. Šta omogućuju kombinaciona kola?

# 2. BROJNI SISTEMI I PREDSTAVE BROJEVA

## 2.1. ARITMETIKA OGRANIČENOG BROJA CIFARA

Celi brojevi se čuvaju u lokacijama u pozicionoj predstavi (najmanje značajna cifra u krajnje desnoj poziciji, a najznačajnija cifra u krajnje levoj poziciji). Poziciona predstava celih brojeva olakšava obavljanje aritmetičkih operacija. Zahvaljujući njoj, postupak sabiranja i oduzimanja, za neoznačene cele brojeve, se svodi na direktnu primenu tablice sabiranja na parove korespondentnih cifara (s desna u levo).

### KOMPLEMENT 10 PREDSTAVA OZNAČENIH CELIH BROJEVA

Za označene cele brojeve postupak sabiranja i oduzimanja komplikuje prisustvo predznaka. To ilustruje primer sabiranja označenih celih brojeva raznih predznaka. U ovom slučaju predznak rezultata je jednak predznaku sabirka sa većom apsolutnom vrednošću. Nakon određivanja predznaka, sledi potpisivanje sabiraka na ispravan način, radi oduzimanja apsolutne vrednosti manjeg sabirka od apsolutne vrednosti većeg sabirka. Dopisivanjem ovako dobijene razlike iza prethodno određenog predznaka nastaje traženi zbir.

Važno svojstvo pozicione predstave celih brojeva je da, za ograničen broj cifara, ona dozvoljava predstavljanje označenih celih brojeva kao neoznačenih. Na taj način se eliminišu komplikacije prilikom njihovog sabiranja i oduzimanja, jer se ne mora voditi računa o predznaku. To pokazuje sledeći primer:

$$68+(-57) == 68+(-57)+1000-1000 == 68+943-1000 == 1011-1000 == 11$$

Ako se u prethodnom primeru posmatraju samo 3 najmanje značajne cifre i ako se podrazumeva dodavanje i oduzimanje $10^3$, tada se sabiranje označenih celih brojeva 68 i -57 može prikazati kao:

```
 068  (68)
+943  (-57+1000)
 011  (1011-1000)
```

Prenos sa najznačajnije pozicije se zanemaruje, jer se podrazumeva dodavanje i oduzimanje $10^3$. Broj 943 je (trocifrena) **komplement** 10 **predstava** broja -57.

Iz prethodnog primera sledi da se određivanje n cifarske komplement 10 predstave negativnog celog broja svodi na njegovo sabiranje sa $10^n$. Međutim, pomenuta komplement 10 predstava se može odrediti i na drugi način. On podrazumeva da se za svaku cifru brojnog sistema odredi njen **komplement**, odnosno cifra koja dopunjava posmatranu cifru do najveće cifre u brojnom sistemu.

Slika 2.1.1 sadrži komplemente cifara dekadnog brojnog sistema, čija je najveća cifra 9.

| cifra | komplement | komentar |
|---|---|---|
| 0 | 9 | 0+9==9 |
| 1 | 8 | 1+8==9 |
| 2 | 7 | 2+7==9 |
| 3 | 6 | 3+6==9 |
| 4 | 5 | 4+5==9 |
| 5 | 4 | 5+4==9 |
| 6 | 3 | 6+3==9 |
| 7 | 2 | 7+2==9 |
| 8 | 1 | 8+1==9 |
| 9 | 0 | 9+0==9 |

Slika 2.1.1 Cifre dekadnog brojnog sistema i njihovi komplementi

Formiranje komplement 10 predstave negativnog celog broja započinje dodavanjem ispred cifara apsolutne vrednosti ovog broja vodećih nula do ukupnog broja cifara. Tako nastane **potpuna** apsolutna vrednost negativnog celog broja. Zatim se zamenjuje svaka od cifara potpune apsolutne vrednosti negativnog celog broja komplementom dotične cifre. Postupak zamene cifara njihovim komplementima se zove **komplementiranje**. Komplementiranjem nastaje komplement 9 predstava negativnog celog broja. Uvećavanjem komplement 9 predstave negativnog celog broja za 1 nastaje komplement 10 predstava negativnog celog broja.

Komplement 10 predstava pozitivnog celog broja nastaje njegovim dopunjavanjem s leva vodećim nulama do ukupnog broja cifara.

U slučaju prethodnog primera, trocifrena komplement 10 predstava broja 68 je 068.

Trocifrena komplement 9 predstava broja -57 je 942. Ona nastaje komplementiranjem cifara broja 057 (odnosno komplementiranjem cifara potpune apsolutne vrednosti broja -57). Trocifrena komplement 10 predstava broja -57 je 943. Ona je za 1 veća od trocifrene komplement 9 predstave istog broja.

Važno je naglasiti da komplementiranjem cifara komplement 10 predstave negativnog celog broja i uvećanjem tako dobijenog broja za 1 nastaje potpuna apsolutna vrednost negativnog celog broja.

U komplement 10 predstavi označenih celih brojeva, sve cifre nose vrednost broja, ali najznačajnija cifra ima i dodatnu ulogu, jer ukazuje na predznak broja. Tako, za komplement 10 predstavu pozitivnih celih brojeva, najznačajnija cifra je uvek 0, a za komplement 10 predstavu negativnih celih brojeva, najznačajnija cifra je uvek 9.

Komplement predstava označenih celih brojeva pojednostavljuje procesor, jer omogućuje da se aritmetika označenih celih brojeva obavlja kao i aritmetika neoznačenih celih brojeva (znači, bez posmatranja predznaka brojeva). Tako sabiranje brojeva -7 i 2 može da bude obavljeno kao sabiranje njihovih trocifrenih komplement 10 predstava 993 i 002, uz direktnu primenu tablice sabiranja na parove korespondentnih cifara i bez razmatranja predznaka. Na ovaj način kao rezultat sabiranja dobije se 995, a to je trocifrena komplement 10 predstava broja -5.

U nastavku su prikazane neke vrednosti trocifrene komplement 10 predstave označenih celih brojeva (njima korespondentne vrednosti, sa predznakom, su navedene u zagradama):

099 (+99)
098 (+98)
097 (+97)
096 (+96)
...
002 (+2)
001 (+1)
000 (0)
999 (-1)
998 (-2)
...
903 (-97)
902 (-98)
901 (-99)
900 (-100)

U komplement 10 predstavi, broj negativnih celih brojeva je za jedan veći od broja pozitivnih celih brojeva.

Aritmetičke operacije označenih celih brojeva u komplement 10 predstavi daju označeni celi broj u komplement 10 predstavi. To ilustruje primer oduzimanja:

010 (+10)
-011 (+11)
999 (-1)

Prilikom obavljanja aritmetičkih operacija, prenos sa najznačajnije pozicije u komplement 10 predstavi celih brojeva se uvek zanemaruje.

### IZLAZAK VAN OPSEGA

Za aritmetiku ograničenog broja cifara vezan je problem pojave cifara koje izlaze **van opsega**, uvedenog ograničavanjem broja posmatranih cifara. Kod

aritmetike neoznačenih celih brojeva, čije sve cifre nose samo vrednost, do izlaska van opsega dolazi nakon pojave prenosa sa najznačajnije pozicije rezultata:

```
 999
+999
 998
```

U prethodnom primeru se pojavio netačan rezultat, jer se prenos ne može prikazati u posmatrane 3 cifre.

Kod aritmetike označenih celih brojeva u komplement 10 predstavi, čija najznačajnija cifra pokazuje da li je reč o pozitivnom (0) ili negativnom (9) broju, pojava cifre različite od 0 ili 9 u najznačajnijoj poziciji rezultata ukazuje na izlazak van opsega:

```
 050          900
+050   ili   -001
 100          899
```

U prethodnim primerima na pojavu netačnog rezultata ukazuju neodgovarajuće cifre u njegovoj najznačajnijoj poziciji.

Važno je zapaziti da interpretacija rezultata zavisi od interpretacije brojeva. Ako se u pretposlednjem primeru celi brojevi interpretiraju kao označeni u trocifrenoj komplement 10 predstavi, tada je rezultat tačan (u ovom slučaju prenos se zanemaruje). Slično, ako se celi brojevi iz poslednjeg primera interpretiraju kao neoznačeni, tada je rezultat tačan, jer nema prenosa sa najznačajnije pozicije.

## POREĐENJE CELIH BROJEVA

Kod određivanja relacija, izlazak van opsega pri celobrojnom oduzimanju prvog operanda (PO) relacije od drugog (DO) ne izaziva probleme, nego pomaže da se izvede ispravan zaključak o važenju relacije.

Za neoznačene cele brojeve:

1. relacija DO == PO važi, ako je rezultat oduzimanja 0,
2. relacija DO != PO važi, ako rezultat oduzimanja nije 0,
3. relacija DO < PO važi, ako se pri oduzimanju javi izlazak van opsega (u obliku prenosa),
4. relacija DO >= PO važi, ako se pri oduzimanju ne javi izlazak van opsega ili je rezultat 0,
5. relacija DO > PO važi, ako se pri oduzimanju ne javi izlazak van opsega i rezultat nije 0, a
6. relacija DO <= PO važi, ako se pri oduzimanju javi izlazak van opsega (u obliku prenosa) ili je rezultat 0.

Za označene cele brojeve:

1. relacija DO == PO važi, ako je rezultat oduzimanja 0,
2. relacija DO != PO važi, ako rezultat nije 0,
3. relacija DO < PO važi, ako je rezultat negativan (najznačajnija cifra 9) ili se pri oduzimanju izađe van opsega (najznačajnija cifra 8),
4. relacija DO >= PO važi, ako je rezultat pozitivan (najznačajnija cifra 0) ili se pri oduzimanju izađe van opsega (najznačajnija cifra 1) ili je rezultat 0,
5. relacija DO > PO važi, ako je rezultat pozitivan (najznačajnija cifra 0) ili se pri oduzimanju izađe van opsega (najznačajnija cifra 1), a
6. relacija DO <= PO važi, ako je rezultat negativan (najznačajnija cifra 9) ili se pri oduzimanju izađe van opsega (najznačajnija cifra 8) ili je rezultat 0.

Važenje prethodnog se može pokazati na primerima svih tipičnih odnosa koji se mogu pojaviti u nekoj relaciji. Za relaciju < sve tipične odnose ilustruju primeri dati tabelarno (Slika 2.1.2).

| drugi operand | prvi operand | < | razlika (komplement 10 predstava) |
|---|---|---|---|
| 88 | 99 | da | 088-099 == 989 |
| -99 | 99 | da | 901-099 == 802 |
| -19 | 19 | da | 981-019 == 962 |
| -99 | -88 | da | 901-912 == 989 |
| 99 | 88 | ne | 099-088 == 011 |
| 99 | -99 | ne | 099-901 == 198 |
| 19 | -19 | ne | 019-981 == 038 |
| -88 | -99 | ne | 912-901 == 011 |

Slika 2.1.2 Primeri važenja relacije <

## VIŠESTRUKA PRECIZNOST

Aritmetika ograničenog broja cifara zahteva **višestruku preciznost**, kada se operacije odnose na vrednosti čiji broj cifara je veći od broja cifara lokacije. U ovom slučaju, delovi vrednosti se čuvaju u raznim lokacijama, a operacije se obavljaju za deo po deo ovih vrednosti (idući od manje značajnih ka značajnijim delovima vrednosti).

## ZAKONI ARITMETIKE

Za aritmetiku ograničenog broja cifara ne važe zakoni aritmetike, jer, zbog mogućeg izlaska van opsega, rezultat zavisi od redosleda obavljanja operacija. To potkrepljuje sledeći primer za trocifrenu preciznost:

(200+800)-500 != 200+(800-500)

u kome su navedeni neoznačeni celi brojevi. U toku formiranja levog zbira dolazi do izlaska van opsega, pa je on u opštem slučaju različit od desnog zbira, u kome se ne javlja izlazak van opsega.

## 2.2. PREDSTAVLJANJE REALNIH BROJEVA

Potpuna poziciona predstava je nepraktična za realne brojeve, jer ona, na primer, podrazumeva da se u broju:

0.000000127

koristi 6 nula za određivanje položaja značajnih cifara iza tačke. Manje prostora zauzima **normalizovana forma**, čiji eksponent određuje položaj značajnih cifara:

$1.27\times10^{-7}$

Još manje prostora je potrebno za **mašinsku normalizovanu formu** (*floating-point*), koja sadrži samo predznak realnog broja, njegovu **frakciju** (*fraction, significand*) i **podešeni eksponent** (*biased exponent*, *excess*). Frakcija obuhvata značajne cifre broja. Podrazumeva se da iza prve od značajnih cifara dolazi decimalna tačka. Eksponent je podešen dodavanjem **konstante podešavanja**, da se ne bi moralo voditi računa o njegovom predznaku i da bi se olakšalo poređenje mašinskih normalizovanih formi. Prethodni broj u mašinskoj normalizovanoj formi sa 10 mesta (MNF10) izgleda:

$0031270000_{MNF10}$

Posmatrano s leva u desno, prva cifra prethodnog broja sadrži predznak (0 : +, 1 : -). Sledeće dve cifre sadrže podešeni eksponent. Dva mesta omogućuju da eksponent bude u rasponu od -9 do +9. On se podešava dodavanjem konstante podešavanja $10==10^{2-1}$, pa je raspon podešenog eksponenta od 1 do 19. Ovako podešeni eksponent omogućuje poređenje mašinskih normalizovanih formi bez njihovog raspakivanja, pri čemu se mora uračunati uticaj predznaka na rezultat poređenja. Preostalih sedam cifara sadrži frakciju. Podrazumeva se da iza prve cifre frakcije dolazi decimalna tačka.

Predstava realnih brojeva u ograničenom broju cifara ograničava i dijapazon realnih brojeva koji se mogu predstaviti, ali i preciznost, jer iz datog dijapazona može da se precizno predstavi samo ograničen broj realnih brojeva. Greška predstave realnih brojeva, koji ne mogu precizno da se predstave, raste sa veličinom eksponenta i kreće se od $10^{-15}$ do $10^{3}$ za MNF10. Tolika je razlika apsolutnih vrednosti 2 najmanja susedna realna broja: $0011000000_{MNF10}$ ($1.0\times10^{-9}$) i

$0011000001_{MNF10}$ $(1.000001\times10^{-9})$, odnosno 2 najveća susedna realna broja: $0199999998_{MNF10}$ $(9.999998\times10^{9})$ i $0199999999_{MNF10}$ $(9.999999\times10^{9})$.

## ARITMETIKA MAŠINSKE NORMALIZOVANE FORME

U aritmetici normalizovane forme, predznak, podešeni eksponenti i frakcije se posmatraju odvojeno. Prvo se odredi predznak rezultata. Sabiranje i oduzimanje započinje izjednačavanjem eksponenata (radi ispravnog potpisivanja frakcija). Uvek se manji eksponent izjednačava sa većim, uz pomeranje decimalne tačke u frakciji i, eventualno, odbacivanje prekobrojnih cifara. Zatim sledi sabiranje ili oduzimanje frakcija i, po potrebi, normalizacija rezultata. Tako, sabiranju brojeva:

$$\begin{aligned} &0031270000_{MNF10} && (1.27\times10^{-7})\\ +&0091000001_{MNF10} && (1.000001\times10^{-1}) \end{aligned}$$

obavezno prethodi izjednačavanje eksponenta prvog broja sa eksponentom drugog broja, pri čemu se odbacuju prekobrojne cifre frakcije 2 i 7. Tako nastaju brojevi:

$$\begin{aligned} &0090000001_{MNF10} && (0.000001\times10^{-1}) && \text{(odbačene su prekobrojne cifre 2 i 7)}\\ +&\underline{0091000001}_{MNF10} && (1.000001\times10^{-1})\\ &0091000002_{MNF10} && (1.000002\times10^{-1}) \end{aligned}$$

Kod množenja, eksponenti se sabiraju, uz umanjivanje njihovog zbira za konstantu podešavanja 10, a frakcije se množe. Kod deljenja, eksponenti se oduzimaju, uz dodavanje njihovoj razlici konstante podešavanja 10, a frakcije se dele. U oba slučaja, na kraju, eventualno, sledi normalizacija rezultata. Prethodno ilustruje primer množenja:

$$\begin{aligned} &0112000000_{MNF10} && (2.0\times10^{1})\\ \times&\underline{0124000000}_{MNF10} && (4.0\times10^{2})\\ &0138000000_{MNF10} && (8.0\times10^{3}) \end{aligned}$$

U aritmetici normalizovane forme do izlaska van opsega (s leva, ali i s desna) dolazi samo u eksponentu, jer se prekobrojne cifre frakcije odbacuju. Odbacivanje prekobrojnih cifara frakcije može da uzrokuje ozbiljne greške, naročito, ako u operaciji učestvuju realni brojevi koji se veoma razlikuju po apsolutnoj vrednosti. Primer za takvu vrstu grešaka pruža sabiranje vrlo mnogo malih realnih brojeva. Kada, u toku sabiranja, apsolutna vrednost zbira postane toliko velika, da su sve cifre frakcija preostalih sabiraka prekobrojne, tada u nastavku sabiranja ne dolazi do uvećanja ovog zbira, iako on može biti manji od sume preostalih (zanemarenih) sabiraka.

## 2.3. BINARNI BROJNI SISTEM

U binarnom brojnom sistemu koriste se samo cifre 0 i 1, njegova **baza** (*radix*) je $2_{10}$, a brojevi se predstavljaju u pozicionoj predstavi. Pri prelasku iz decimalnog u binarni brojni sistem i obratno, neophodna je **konverzija brojeva**.

### POSTUPAK KONVERZIJE BROJEVA

Konverzije brojeva iz brojnog sistema s jednom bazom u brojni sistem s drugom bazom se zasnivaju na izjednačavanju pozicionih predstava pomenutih brojeva, radi izdvajanja nepoznatih cifara. Pri tome u ovim predstavama se koriste cifre brojnog sistema u kome se računa. Postupak konverzije zahteva da se posebno obavlja konverzija celog, a posebno konverzija razlomljenog dela broja.

Do izdvajanja nepoznatih binarnih cifara, pri konverziji celog broja iz decimalnog u binarni brojni sistem, dolazi kada se konvertovani broj deli bazom binarnog brojnog sistema. U primeru, koji sledi, koriste se cifre decimalnog brojnog sistema, jer se u njemu računa:

| | | | |
|---|---|---|---|
| $...+d_1 10^1+d_0 10^0$ | == | $...+b_1 2^1+b_0 2^0$ | |
| | | | |
| $6_{10}$ | == | $...+b_1 2^1+b_0 2^0$ \|:2 | |
| 0 | == | $b_0$ | (ostatak deljenja) |
| $3_{10}$ | == | $...+b_2 2^1+b_1 2^0$ \|:2 | (količnik) |
| 1 | == | $b_1$ | (ostatak deljenja) |
| 1 | == | $...+b_3 2^1+b_2 2^0$ \|:2 | (količnik) |
| 1 | == | $b_2$ | (ostatak deljenja) |
| 0 | | | (količnik) |
| $6_{10}$ | == | $110_2$ | |

Do izdvajanja nepoznatih binarnih cifara, pri konverziji razlomljenog broja iz decimalnog u binarni brojni sistem, dolazi kada se konvertovani broj množi bazom binarnog brojnog sistema. U sledećem primeru koriste se cifre decimalnog brojnog sistema, jer se u njemu računa:

| | | | |
|---|---|---|---|
| $d_{-1}10^{-1}+d_{-2}10^{-2}+...$ | == | $b_{-1}2^{-1}+b_{-2}2^{-2}+...$ | |
| | | | |
| $0.375_{10}$ | == | $b_{-1}2^{-1}+b_{-2}2^{-2}+...$ \|×2 | |
| 0 | == | $b_{-1}$ | (celi deo proizvoda) |
| $0.75_{10}$ | == | $b_{-2}2^{-1}+b_{-3}2^{-2}+...$ \|×2 | (razlomljeni deo proizvoda) |
| 1 | == | $b_{-2}$ | (celi deo proizvoda) |
| $0.5_{10}$ | == | $b_{-3}2^{-1}+b_{-4}2^{-2}+...$ \|×2 | (razlomljeni deo proizvoda) |
| 1 | == | $b_{-3}$ | (celi deo proizvoda) |
| 0 | | | (razlomljeni deo proizvoda) |
| $0.375_{10}$ | == | $0.011_2$ | |

Algoritam konverzije razlomljenog broja iz decimalnog u binarni brojni sistem ne mora da ima kraj. To pokazuje primer konvertovanja razlomljenog broja $0.2_{10}$, prikazan tabelarno (Slika 2.3.1).

| razlomljeni deo proizvoda | proizvod (×2) | celi deo proizvoda |
|---|---|---|
| 0.2 | 0.4 | $0 == b_{-1}$ |
| 0.4 | 0.8 | $0 == b_{-2}$ |
| 0.8 | 1.6 | $1 == b_{-3}$ |
| 0.6 | 1.2 | $1 == b_{-4}$ |
| 0.2 | 0.4 | $0 == b_{-5}$ |

Slika 2.3.1 Konvertovanje razlomljenog broja $0.2_{10}$

U tabelarno datom primeru (Slika 2.3.1) javlja se beskonačna periodičnost, pa je rezultat konverzije približan:

$0.2_{10} \approx 0.0011_2$

Konverzija celog broja iz binarnog u decimalni brojni sistem se svodi na množenje cifara konvertovanog broja bazom binarnog brojnog sistema. U sledećem primeru koriste se cifre decimalnog brojnog sistema, jer se u njemu računa:

$...+b_2 2^2+b_1 2^1+b_0 2^0 \quad == \quad ((...+b_2)\times 2+b_1)\times 2+b_0$

$X_{10} == 110_2$
$X_{10} == ((b_2)\times 2+b_1)\times 2+b_0$

$X_{10} = b_2 == 1$
$X_{10} = X_{10}\times 2+b_1 == 1\times 2+1 == 3$
$X_{10} = X_{10}\times 2+b_0 == 3\times 2+0 == 6$
$X_{10} == 6$
$6_{10} == 110_2$

Konverzija razlomljenog broja iz binarnog u decimalni brojni sistem se svodi na deljenje cifara konvertovanog broja bazom binarnog brojnog sistema. U sledećem primeru koriste se cifre decimalnog brojnog sistema, jer se u njemu računa:

$b_{-1}2^{-1}+b_{-2}2^{-2}+b_{-3}2^{-3}+... == (b_{-1}+(b_{-2}+(b_{-3}+...)/2)/2)/2$

$X_{10} == 0.0011_2$
$X_{10} == (b_{-1}+(b_{-2}+(b_{-3}+(b_{-4})/2)/2)/2)/2$

| | | | | | | |
|---|---|---|---|---|---|---|
| $X_{10}$ | = | $b_{-4}$ | == | 1 | | |
| $X_{10}$ | = | $X_{10}/2+b_{-3}$ | == | $1/2+1$ | == | 1.5 |
| $X_{10}$ | = | $X_{10}/2+b_{-2}$ | == | $1.5/2+0$ | == | 0.75 |
| $X_{10}$ | = | $X_{10}/2+b_{-1}$ | == | $0.75/2+0$ | == | 0.375 |
| $X_{10}$ | = | $X_{10}/2$ | == | $0.375/2$ | == | 0.1875 |
| $0.1875_{10}$ | == | $0.0011_2$ | | | | |

Pošto konverzija $0.2_{10}$ daje približno $0.0011_2$, a konverzija $0.0011_2$ daje $0.1875_{10}$, sledi da je:

$$0.2_{10} \approx 0.1875_{10}$$

odnosno da je greška aproksimacije $0.0125_{10}$.

Za skraćeno predstavljanje brojeva iz binarnog brojnog sistema koristi se heksadecimalni brojni sistem, jer jedna heksadecimalna cifra zamenjuje četiri binarne cifre. U heksadecimalnom brojnom sistemu koriste se cifre 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A ($10_{10}$), B ($11_{10}$), C ($12_{10}$), D ($13_{10}$), E ($14_{10}$) i F ($15_{10}$). Njegova baza je $16_{10}$, a brojevi se predstavljaju u pozicionoj predstavi.

Konverzija broja iz binarnog u heksadecimalni brojni sistem se sastoji od zamena po četiri binarne cifre jednom heksadecimalnom cifrom. Pre toga se binarni broj dopuni s leva vodećim nulama do broja cifara koji je deljiv sa 4:

$$1011111_2 == 5F_{16}$$

$01011111_2 ==$
$0\times2^7+1\times2^6+0\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0 ==$
$(0\times2^3+1\times2^2+0\times2^1+1\times2^0)\times2^4+(1\times2^3+1\times2^2+1\times2^1+1\times2^0)\times2^0 ==$
$0101\times2^4+1111\times2^0 ==$
$5\times16^1+F\times16^0$

(eksponenti i baza 16 su prikazani u decimalnom brojnom sistemu).

Konverzija broja iz heksadecimalnog u binarni brojni sistem se sastoji od zamene svake heksadecimalne cifre sa 4 odgovarajuće binarne cifre.

## 2.4. ARITMETIKA OGRANIČENOG BROJA CIFARA U BINARNOM BROJNOM SISTEMU

I u binarnom brojnom sistemu sabiranje i oduzimanje neoznačenih celih brojeva se sastoji od direktne primene tablice sabiranja na parove korespondentnih cifara. U narednim primerima se smatra da je broj binarnih cifra ograničen na 8:

| | | | |
|---|---|---|---|
| 00000101 | (5) | 00000101 | (5) |
| +00000011 | (3) | -00000011 | (3) |
| 00001000 | (8) | 00000010 | (2) |

### KOMPLEMENT 2 PREDSTAVA OZNAČENIH CELIH BROJEVA

Za označene cele brojeve koristi se komplement 2 predstava, radi izbegavanja komplikacija kod njihovog sabiranja i oduzimanja. Komplement 2 predstava pozitivnih celih brojeva nastaje njihovim dopunjavanjem s leva vodećim nulama (do ukupnog broja cifara):

| | |
|---|---|
| 00000010 | komplement 2 predstava broja +2 |

Komplement 2 predstava negativnih celih brojeva nastaje dodavanjem jedinice komplement 1 predstavi negativnih celih brojeva. Komplement 1 predstava negativnih celih brojeva nastaje tako, što se u potpunoj apsolutnoj vrednosti negativnih celih brojeva svaka nula zameni jedinicom i obrnuto:

| | |
|---|---|
| 00000010 | potpuna apsolutna vrednost broja -2 |
| 11111101 | komplement 1 predstava broja -2 |
| 11111110 | komplement 2 predstava broja -2 |

U nastavku slede neke vrednosti osmocifrene komplement 2 predstave označenih celih brojeva (njima korespondentne decimalne vrednosti, sa predznakom, su navedene u zagradama):

| | |
|---|---|
| 01111111 | (+127) |
| 01111110 | (+126) |
| ... | |
| 00000001 | (+1) |
| 00000000 | (0) |
| 11111111 | (-1) |
| ... | |
| 10000001 | (-127) |
| 10000000 | (-128) |

I u komplement 2 predstavi, broj negativnih brojeva je za jedan veći od broja pozitivnih brojeva.

### IZLAZAK VAN OPSEGA

Kod neoznačenih celih brojeva, pojava prenosa sa najznačajnijeg mesta ukazuje na izlazak van opsega. Kod označenih celih brojeva u komplement 2 predstavi, pojava neodgovarajuće cifre na najznačajnijem mestu rezultata ukazuje na izlazak van opsega (na primer, kod sabiranja dva pozitivna cela broja, pojava cifre 1 na najznačajnijem mestu zbira nedvosmisleno ukazuje na izlazak van opsega). Svi slučajevi izlaska van opsega, kod sabiranja označenih celih brojeva u komplement 2 predstavi, mogu se pregledno prikazati tabelarno (Slika 2.4.1).

| najznačajniji bit prvog sabirka ($S_1$) | najznačajniji bit drugog sabirka ($S_2$) | najznačajniji bit zbira (Z) | izlazak van opsega |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Slika 2.4.1 Izlasci van opsega kod sabiranja označenih celih brojeva u komplement 2 predstavi

U tabeli (Slika 2.4.1), cifre iz prve tri kolone se interpretiraju kao celi brojevi, a cifre iz četvrte kolone se interpretiraju kao logičke vrednosti. Ako se cifre u sve četiri kolone interpretiraju kao logičke vrednosti, tada prve tri kolone sadrže argumente, a poslednja vrednosti logičke funkcije izlaska van opsega kod sabiranja:

$((\sim S_1)\&(\sim S_2)\&Z)|(S_1\&S_2\&(\sim Z))$

Svi slučajevi izlaska van opsega, kod oduzimanja označenih celih brojeva u komplement 2 predstavi, mogu se pregledno prikazati tabelarno (Slika 2.4.2).

| najznačajniji bit umanjenika ($U_1$) | najznačajniji bit umanjioca ($U_2$) | najznačajniji bit razlike (R) | izlazak van opsega |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

Slika 2.4.2 Izlasci van opsega kod oduzimanja označenih celih brojeva u komplement 2 predstavi

U tabeli (Slika 2.4.2), cifre iz prve tri kolone se interpretiraju kao celi brojevi, a cifre iz četvrte kolone se interpretiraju kao logičke vrednosti. Ako se cifre u sve četiri kolone interpretiraju kao logičke vrednosti, tada prve tri kolone sadrže argumente, a poslednja vrednosti logičke funkcije izlaska van opsega kod oduzimanja:

$((\sim U_1)\&U_2)\&R)|(U_1\&(\sim U_2)\&(\sim R))$

Interpretacija rezultata aritmetičkih operacija zavisi od interpretacije brojeva. Na primer, kod sabiranja neoznačenih celih brojeva:

$$\begin{array}{rl} 11111111_2 & (255_{10}) \\ \underline{+11111111_2} & (255_{10}) \\ 11111110_2 & (254_{10}) \end{array}$$

dolazi do izlaska van opsega, jer se javlja prenos sa najznačajnijeg mesta binarnog rezultata. Kod sabiranja neoznačenih celih brojeva:

$$\begin{array}{rl} 01000000_2 & (64_{10}) \\ \underline{+01000000_2} & (64_{10}) \\ 10000000_2 & (128_{10}) \end{array}$$

ne dolazi do izlaska van opsega, jer se ne javlja prenos sa najznačajnijeg mesta binarnog rezultata. Ali, izmena interpretacije brojeva u prethodna dva primera povlači za sobom i izmenu interpretacije rezultata. Tako, kod sabiranja označenih celih brojeva u komplement 2 predstavi:

$$\begin{array}{rl} 11111111_2 & (-1_{10}) \\ \underline{+11111111_2} & (-1_{10}) \\ 11111110_2 & (-2_{10}) \end{array}$$

ne dolazi do izlaska van opsega, jer je predznak binarnog rezultata ispravan. Prenos sa najznačajnijeg mesta binarnog rezultata se zanemaruje, jer je reč o komplement 2 predstavi brojeva. Slično, kod sabiranja označenih celih brojeva u komplement 2 predstavi:

$$\begin{array}{rl} 01000000_2 & (+64_{10}) \\ \underline{+01000000_2} & (+64_{10}) \\ 10000000_2 & (-128_{10}) \end{array}$$

dolazi do izlaska van opsega, jer se javlja neodgovarajuća cifra na najznačajnijem mestu rezultata.

## POREĐENJE CELIH BROJEVA

I u binarnom brojnom sistemu, određivanje relacija se svodi na oduzimanje prvog operanda (PO) relacije od drugog (DO) i na zaključivanje o važenju relacije na osnovu toga:

1. da li je razlika jednaka nuli ili je različita od nule,
2. da li je razlika negativna ili pozitivna,
3. da li se, pri oduzimanju neoznačenih celih brojeva, javlja izlazak van opsega u obliku prenosa ili ne, odnosno,
4. da li, pri oduzimanju označenih celih brojeva, razlika izlazi van opsega ili ne.

Za opisivanje karakteristika razlike zgodno je uvesti posebne **logičke promenljive**: **N** (nula), **M** (minus), **P** (prenos) i **V** (van opsega). Tako logička promenljiva N sadrži 1, ako je razlika jednaka nuli. U suprotnom, ona sadrži 0. Ova logička promenljiva je jednaka negaciji logičkog ili svih cifara razlike. Logička promenljiva M sadrži 1, ako je razlika negativna. U suprotnom, ona sadrži 0. Ova logička promenljiva je jednaka najznačajnijoj cifri razlike. Logička promenljiva P sadrži 1, ako se pri oduzimanju neoznačenih celih brojeva javi prenos. U suprotnom, ona sadrži 0. Ova logička promenljiva je jednaka prenosu sa najznačajnijeg mesta pri oduzimanju. Logička promenljiva V sadrži 1, ako, pri oduzimanju označenih celih brojeva, razlika izađe van opsega. U suprotnom, ona sadrži 0. Ova logička promenljiva je jednaka vrednosti logičke funkcije izlaska van opsega kod oduzimanja.

Pri određivanju relacija, svakoj od prethodne četiri logičke promenljive dodeljuje se, po oduzimanju, odgovarajuća vrednost. To se radi zbog višeznačne interpretacije operanada relacija. Međutim, zaključak o važenju relacije se oslanja samo na logičke promenljive koje su značajne za važeću interpretaciju poređenih

vrednosti. To se može pokazati tabelama (Slika 2.4.3) i (Slika 2.4.4). Tabela (Slika 2.4.3) sadrži pregled logičkih izraza od čije vrednosti zavisi važenje pojedinih relacija za neoznačene brojeve.

| relacija važi | ako je tačan logički izraz |
|---|---|
| DO == PO | N |
| DO != PO | ~N |
| DO < PO | P |
| DO >= PO | ~P |
| DO > PO | (~P)&(~N) |
| DO <= PO | P\|N |

Slika 2.4.3 Prikaz uslova važenja relacija za neoznačene brojeve

Tabela (Slika 2.4.4) sadrži pregled logičkih izraza od čije vrednosti zavisi važenje pojedinih relacija za označene brojeve.

| relacija važi | ako je tačan logički izraz |
|---|---|
| DO == PO | N |
| DO != PO | ~N |
| DO < PO | M^V |
| DO >= PO | ~(M^V) |
| DO > PO | ~(M^V)&(~N) |
| DO <= PO | (M^V)\|N |

Slika 2.4.4 Prikaz uslova važenja relacija za označene brojeve

Razumevanje logičkih izraza, navedenih u ovoj tabeli (Slika 2.4.4) za relacije <, >=, >, i <=, zahteva posmatranje slučajeva u kojima su oba operanda pozitivna, odnosno negativna, odnosno u kojima su operandi različitih predznaka. Tabelarni pregled karakterističnih slučajeva važenja relacije < sadrži Slika 2.4.5.

| drugi operand | prvi operand | < | M | V |
|---|---|---|---|---|
| + | + | da | 1 | 0 |
| - | + | da | 1<br>0 | 0<br>1 |
| - | - | da | 1 | 0 |
| + | + | ne | 0 | 0 |
| + | - | ne | 0<br>1 | 0<br>1 |
| - | - | ne | 0 | 0 |

Slika 2.4.5 Pregled karakterističnih slučajeva važenja relacije <

Relacija < važi, ako su oba njena operanda pozitivna, a drugi je manji od prvog. Tada je razlika uvek negativna i nije van opsega (M == 1 i V == 0). Relacija < važi i kada je njen drugi operand negativan, a prvi pozitivan. Tada je razlika uvek ili negativna i nije van opsega (M == 1 i V == 0), ili pozitivna i van opsega (M == 0 i V == 1). Relacija < važi i ako su njena oba operanda negativna, a drugi je manji od prvog. Tada je razlika uvek negativna i nije van opsega (M == 1 i V == 0). Relacija < ne važi, kada su oba njena operanda pozitivna, a drugi nije manji od prvog. Tada je razlika uvek pozitivna i nije van opsega (M == 0 i V == 0). Relacija < ne važi ni ako je njen drugi operand pozitivan, a prvi negativan. Tada je razlika uvek ili pozitivna i nije van opsega (M == 0 i V == 0), ili negativna i van opsega (M == 1 i V == 1). Relacija < ne važi ni kada su njena oba operanda negativna, a drugi nije manji od prvog. Tada je razlika uvek pozitivna i nije van opsega (M == 0 i V == 0).

### VIŠESTRUKA PRECIZNOST

I u binarnom brojnom sistemu, aritmetika ograničenog broja cifara zahteva višestruku preciznost, kada se operacije odnose na vrednosti čiji broj cifara je veći od broja cifara lokacije. I u ovom slučaju, delovi vrednosti se čuvaju u raznim lokacijama, a operacije se obavljaju za deo po deo ovih vrednosti (idući od manje značajnih ka značajnijim delovima vrednosti).

### ZAKONI ARITMETIKE

Takođe, ni u binarnom brojnom sistemu za aritmetiku ograničenog broja cifara ne važe zakoni aritmetike, jer je, zbog mogućnosti izlaska van opsega, rezultat zavisan od redosleda obavljanja operacija. To potkrepljuje sledeći primer za osmobitnu preciznost:

$$(10000100_2+10100000_2)-10000000_2 \; != \; 10000100_2+(10100000_2-10000000_2)$$

u kome su navedeni neoznačeni celi brojevi (u toku formiranja levog zbira dolazi do izlaska van opsega, pa je on u opštem slučaju različit od desnog zbira, u kome se ne javlja izlazak van opsega).

## 2.5. PREDSTAVLJANJE VREDNOSTI REALNOG TIPA U BINARNOM BROJNOM SISTEMU

Mašinska normalizovana forma se koristi i u binarnom brojnom sistemu za predstavljanje vrednosti realnog tipa. U nastavku se koristi 8 cifarskih pozicija za mašinsku normalizovanu formu (MNF8). U njoj, gledajući s leva u desno, prva pozicija sadrži predznak (0 : +, 1 : -). Naredne 4 pozicije sadrže podešeni eksponent. Kao konstanta podešavanja koristi se vrednost $8==1000_2==2^{4-1}$. Ona se sabira sa eksponentima koji mogu biti u rasponu od -8 do 7, pa se podešeni eksponenti nalaze u dijapazonu od $0000_2==0$ do $1111_2==15$. Poslednje 3 pozicije

sadrže tri manje značajne cifre frakcije. Preostala, najznačajnija cifra frakcije je uvek 1, jer, kod normalizovane forme, tačka dolazi iza prve značajne cifre frakcije, različite od 0. Zato se ova cifra ne prikazuje u frakciji. Znači, frakcija je uvek različita od nule, jer se podrazumeva da je njena najznačajnija cifra 1. Zato se za predstavljanje nule rezerviše nulta vrednost podešenog eksponenta. Slede primeri brojeva, predstavljenih u MNF8:

$10101000_{MNF8}$==$-1.000_2\times2^{-3}$==$-0.125_{10}$
$00000000_{MNF8}$==$0.0$
$01000000_{MNF8}$==$+1.000_2\times2^{0}$==$+1.0_{10}$
$01001011_{MNF8}$==$+1.011_2\times2^{1}$==$+2.75_{10}$
$01110001_{MNF8}$==$+1.001_2\times2^{6}$==$+72.0_{10}$

I u binarnom brojnom sistemu predstava realnih brojeva u ograničenom broju cifara ograničava i dijapazon realnih brojeva, koji se mogu predstaviti, i preciznost, jer iz datog dijapazona može da se precizno predstavi samo ograničen broj realnih brojeva. Pri tome, greška predstave realnih brojeva, koji ne mogu precizno da se predstave, raste sa veličinom eksponenta i kreće se od $2^{-10}$ do $2^4$ za MNF8.

Prethodno uvedena pravila za aritmetiku normalizovane forme važe i u binarnom brojnom sistemu. Znači, do izlaska van opsega (s leva, ali i s desna) dolazi samo u eksponentu, jer se prekobrojne cifre frakcije odbacuju. Odbacivanje prekobrojnih cifara frakcije može da uzrokuje ozbiljne greške.

## VIŠEZNAČNA INTERPRETACIJA CELIH BROJEVA

Kodiranje vrednosti raznih tipova celim brojevima stvara mogućnost višeznačne interpretacije celih brojeva. Na primer, celi binarni broj 01001011 se može interpretirati kao celi decimalni broj 75, ali i kao realan decimalni broj 2.75. Zbog različite interpretacije, neophodna je konverzija, kada se vrednost jednog tipa dodeljuje promenljivoj drugog tipa. Na primer, ako se, bez konverzije, realnoj promenljivoj dodeli binarni broj 01001011, koji odgovara celom decimalnom broju 75, tada će ova promenljiva dobiti vrednost koja odgovara realnom decimalnom broju 2.75, jer binarni broj 01001011 predstavlja i MNF8 za dotični realni broj ($+1.011_2\times2^1$). Zato je potrebno, pre dodele, izvršiti konverziju binarne vrednosti 01001011:

$75_{10}$==$1001011_2$==$1.001011_2\times2^6\approx1.001_2\times2^6$==$72.0_{10}$==$01110001_{MNF8}$

Znači, nakon konverzije, realna promenljiva će dobiti vrednost koja odgovara realnom decimalnom broju 72.0, što je približna vrednost realnog decimalnog broja 75.0. Navedeni primer ukazuje da nije uvek moguće precizno konvertovati vrednost celog tipa u MNF, čak i kada se koristi isti broj bita za njihovo reprezentovanje.

Konverzija je neophodna i u obrnutom smeru. Na primer prilikom dodele celobrojnoj promenljivoj vrednosti $01110001_{MNF8}$, koji odgovara realnom decimalnom broju 72.0, ova promenljiva će dobiti vrednost koja odgovara celom decimalnom broju 113. Zato je potrebno, pre dodele, izvršiti konverziju vrednosti $01110001_{MNF8}$:

$$72.0_{10}==01110001_{MNF8}==1.001_2\times2^6==1001000.0_2==1001000_2==72_{10}$$

Važno je naglasiti da nije uvek moguća konverzija iz MNF u celi tip, čak i kada se koristi isti broj bita za njihovo reprezentovanje, jer MNF pokriva veći dijapazon vrednosti od celog tipa.

## 2.6. PITANJA

1. Šta omogućuje komplement predstava označenih brojeva?
2. Kako komplement predstava pojednostavljuje procesor?
3. Kada se (ne) zanemaruje prenos sa najznačajnije pozicije prilikom aritmetike celih brojeva?
4. Šta ukazuje na izlazak van opsega?
5. Koje delove sadrži mašinska normalizovana forma?
6. Zašto je uvedena konstanta podešavanja (kako se ona koristi)?
7. Kako se obavlja aritmetika brojeva predstavljenih mašinskom normalizovanom formom?
8. Gde dolazi do izlaska van opsega u aritmetici mašinske normalizovane forme (u eksponentu ili frakciji)?
9. Gde se odbacuju prekobrojne cifre u aritmetici mašinske normalizovane forme (u eksponentu ili frakciji)?
10. Kako se obavljaju konverzije brojeva iz decimalnog u binarni brojni sistem (i obrnuto)?
11. Koji binarni broj se dobije nakon konverzije decimalnog broja 4.25?
12. Koji decimalni broj se dobije nakon konverzije binarnog broja 100.01?
13. Koji heksadecimalni broj se dobije nakon konverzije binarnog broja 11010?
14. Koji binarni broj se dobije nakon konverzije heksadecimalnog broja 1A?
15. Koja je komplement 2 predstava binarnog broja -100?
16. Da li se prilikom sabiranja binarnih brojeva 01000100 i 01110000 javlja izlazak van opsega (za obe interpretacije ovih brojeva)?
17. Na osnovu čega se određuje važenje relacija između (ne)označenih brojeva?
18. Koja je mašinska normalizovana forma (MNF8) binarnog broja -101?
19. Koji binarni broj odgovara mašinskoj normalizovanoj formi $11010010_{MNF8}$?

# 3. ASEMBLERSKO PROGRAMIRANJE

## 3.1. NIVOI PROGRAMIRANJA

Asemblersko programiranje se zasniva na korišćenju asemblerskih naredbi. Ono se nalazi na višem nivou od mašinskog programiranja, koje koristi mašinske naredbe. U ovom slučaju viši nivo podrazumeva apstrakciju (zanemarivanje). Tako, u primeru asemblerskog programiranja, apstrakcija znači zamenarivanje mašinskih formata naredbi (čije poznavanje je neophodno za mašinsko programiranje, a nebitno za asemblersko programiranje). Na još višem nivou se nalazi programiranje procedurnim programskim jezicima, koje potpuno zanemaruje arhitekturu naredbi, jer se zasniva na vrlo uopštenom modelu računara. Zato se procedurni programski jezici svrstavaju u grupu **programskih jezika visokog nivoa**, dok se mašinski i asemblerski programski jezici svrstavaju u grupu **programskih jezika niskog nivoa**.

Programiranje programskim jezicima niskog nivoa je teže od programiranja programskim jezicima visokog nivoa, jer zahteva detaljno poznavanje arhitekture naredbi. Razmišljanje na nivou arhitekture naredbi odvlači pažnju od suštine rešavanog problema i tako uzrokuje razne greške. Iz istih razloga su programi, pisani programskim jezicima niskog nivoa, manje pregledni i teži za održavanje nego programi pisani programskim jezicima visokog nivoa. Ali, programiranje na nivou arhitekture naredbi ima i svoje prednosti, jer dozvoljava korišćenje svih mogućnosti računara. Zato je programiranje programskim jezicima niskog nivoa neizbežno uvek kada je potrebno na poseban način iskoristiti mogućnosti računara.

Mašinske jezike je opravdano koristiti samo ako ne postoje prevodioci za asemblerske ili procedurne programske jezike (asembleri i kompajleri). Poznavanje mašinskih jezika je neophodno i za pravljenje pomenutih prevodilaca.

Asemblerske jezike je opravdano koristiti, ne samo kada ne postoje prevodioci za procedurne programske jezike (kompajleri) ili kada se ovakvi prevodioci prave, nego i u slučajevima koji zahtevaju da se mogućnosti računara iskoriste do krajnjih granica (za šta je neophodno vladanje arhitekturom naredbi koju sakrivaju programski jezici visokog nivoa). Radi olakšavanja asemblerskog programiranja, razvoj asemblerskih programa je uputno započeti preciziranjem algoritma. Za to je zgodno koristiti, kad god je moguće, sredstva tipična za programske jezike visokog nivoa. Nakon preciziranja algoritma, pažnja se može potpuno koncentrisati na izražavanje algoritma asemblerskim naredbama. Ovakav pristup asemblerskom programiranju smanjuje mogućnost greške, jer u svakom momentu ograničava broj detalja o kojima se mora voditi računa.

Arhitektura naredbi, koju odražava svaki asemblerski jezik, se uvek odnosi na određen procesor, pa se zato svaki asemblerski jezik vezuje za neki procesor. U ovoj knjizi je to hipotetski procesor, nazvan KONCEPT, čije ime ukazuje da je

njegova prevashodna namena da omogući shvatanje principa na kojima se zasniva rad računara.

## 3.2. ARHITEKTURA NAREDBI PROCESORA KONCEPT

Memorija procesora KONCEPT se sastoji od lokacija koje sadrže po 16 bita. Krajnje desni bit lokacije (označen indeksom 0, na primer $B_0$) odgovara najmanje značajnoj binarnoj cifri, a krajnje levi bit lokacije (označen indeksom 15, na primer $B_{15}$) odgovara najznačajnijoj binarnoj cifri. Pristupanje memorijskoj lokaciji podrazumeva pristupanje svim njenim bitima.

16 bita omogućuje prikaz $2^{16}$ različitih binarnih brojeva. Ako svaki od njih predstavlja adresu samo jedne memorijske lokacije, tada ukupno ima $2^{16}$ adresa i $2^{16}$ njima odgovarajućih memorijskih lokacija procesora KONCEPT.

Adrese svih mogućih memorijskih lokacija obrazuju **adresni prostor** procesora KONCEPT. Znači adresni prostor procesora KONCEPT obuhvata $2^{16}$ različitih adresa i njima pripadnih lokacija (u opštem slučaju adresni prostor procesora obuhvata $2^n$ različitih adresa/lokacija, ako se adrese sastoje od po n bita).

Procesor KONCEPT ima ukupno 16 **registara opšte namene**, označenih kao `%0`, `%1`, `%2`, `%3`, `%4`, `%5`, `%6`, `%7`, `%8`, `%9`, `%10`, `%11`, `%12`, `%13`, `%14` i `%15`. Svaki od registara opšte namene ovog procesora je organizovan kao memorijska lokacija. Procesor KONCEPT ima i **status registar**, koji je takođe organizovan kao memorijska lokacija. Njegov (najmanje značajan) bit $SR_0$ sadrži logičku promenljivu N (nula), bit $SR_1$ logičku promenljivu M (minus), bit $SR_2$ logičku promenljivu P (prenos), a bit $SR_3$ logičku promenljivu V (van opsega).

Procesor KONCEPT podržava neposredno, direktno, registarsko, posredno i indeksno adresiranje.

### NAREDBE ZA CELOBROJNU ARITMETIKU

U repertoaru naredbi procesora KONCEPT nalaze se naredbe za celobrojnu aritmetiku: SABERI, SABERI_P, DODAJ_1, ODUZMI, ODUZMI_P, ODBIJ_1 i UPOREDI.

SABERI je aritmetička naredba, koja omogućuje sabiranje cele vrednosti, određene prvim (ulaznim) operandom (PO), sa celom vrednošću, određenom drugim (ulaznim) operandom (DO). Zbir se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Tabela (Slika 3.2.1) opisuje sabiranje korespondentnih bita prvog i drugog operanda, označenih kao $PO_i$ i $DO_i$, pri čemu se zbir i prenos označavaju kao $Z_i$ i $P_{i+1}$, a prenos iz sabiranja prethodnog para bita kao $P_i$ (i = 0, ... , 15):

| $PO_i$ | $DO_i$ | $P_i$ | $Z_i$ | $P_{i+1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Slika 3.2.1 Tabela sabiranja parova bita

Za $P_0$ se smatra da je uvek 0. Ako se cifre iz svih pet kolona prethodne tabele interpretiraju kao logičke vrednosti, tada iz prethodne tabele sledi važenje sledećih logičkih funkcija:

$Z_i = PO_i \hat{} DO_i \hat{} P_i$
$P_{i+1} = ((PO_i | DO_i) \& P_i) | (PO_i \& DO_i)$

Nakon sabiranja, menjaju se vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(Z_{15}|Z_{14}|\ Z_{13}|Z_{12}|\ Z_{11}|Z_{10}|\ Z_9|Z_8|\ Z_7|Z_6|\ Z_5|Z_4|\ Z_3|Z_2|\ Z_1|Z_0)$
$M = Z_{15}$
$P = P_{16}$
$V = ((\sim PO_{15}) \& (\sim DO_{15}) \& Z_{15}) | (PO_{15} \& DO_{15} \& (\sim Z_{15}))$

(vrednost logičke promenljive V određuje logička funkcija izlaska van opsega kod sabiranja).

SABERI_P je aritmetička naredba, koja omogućuje sabiranje cele vrednosti, određene prvim (ulaznim) operandom (PO) i zatečenog sadržaja logičke promenljive P (prenosa) sa celom vrednošću, određenom drugim (ulaznim) operandom (DO). Zbir se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Ova naredba omogućuje sabiranje u višestrukoj preciznosti, jer u zbir uključuje i zatečeni prenos, preostao iza prethodnog sabiranja. Naredba SABERI_P se razlikuje od naredbe SABERI samo po tome što kao vrednost zatečenog prenosa $P_0$ uzima vrednost logičke promenljive P.

DODAJ_1 je aritmetička naredba, koja se razlikuje od naredbe SABERI samo po tome što se kao vrednost jednog (ulaznog) operanda koristi konstanta 1, pa se ovaj operand ne navodi.

ODUZMI je aritmetička naredba, koja omogućuje oduzimanje cele vrednosti - umanjioca, određene prvim (ulaznim) operandom (PO), od cele vrednosti -

umanjenika, određene drugim (ulaznim) operandom (DO). Razlika se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Tabela (Slika 3.2.2) opisuje oduzimanje korespondentnih bita prvog i drugog operanda, označenih kao $PO_i$ i $DO_i$, pri čemu se razlika i prenos označavaju kao $R_i$ i $P_{i+1}$, a prenos iz oduzimanja prethodnog para bita kao $P_i$ ($i = 0, \ldots, 15$):

| $DO_i$ | $PO_i$ | $P_i$ | $R_i$ | $P_{i+1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Slika 3.2.2 Tabela oduzimanja parova bita

Za $P_0$ se smatra da je uvek 0. Ako se cifre iz svih pet kolona prethodne tabele interpretiraju kao logičke vrednosti, tada iz prethodne tabele sledi važenje sledećih logičkih funkcija:

$R_i = PO_i \hat{} DO_i \hat{} P_i$
$P_{i+1} = ((DO_i | P_i) \& (\sim PO_i)) | (DO_i \& P_i)$

Nakon oduzimanja, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$
$M = R_{15}$
$P = P_{16}$
$V = ((\sim DO_{15}) \& PO_{15} \& R_{15}) | (DO_{15} \& (\sim PO_{15}) \& (\sim R_{15}))$

(vrednost logičke promenljive V određuje logička funkcija izlaska van opsega kod oduzimanja).

ODUZMI_P je aritmetička naredba, koja omogućuje oduzimanje cele vrednosti, određene prvim (ulaznim) operandom (PO) i zatečenog sadržaja logičke promenljive P (prenosa) od cele vrednosti, određene drugim (ulaznim) operandom (DO). Razlika se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Ova naredba omogućuje oduzimanje u višestrukoj preciznosti, jer u razliku uključuje i zatečeni prenos, preostao iza prethodnog oduzimanja. Naredba

ODUZMI_P se razlikuje od naredbe ODUZMI samo po tome što kao vrednost zatečenog prenosa $P_0$ uzima vrednost logičke promenljive P.

ODBIJ_1 je aritmetička naredba, koja se razlikuje od naredbe ODUZMI samo po tome što se kao vrednost jednog (ulaznog) operanda koristi konstanta 1, pa se ovaj operand ne navodi.

UPOREDI je aritmetička naredba koja omogućuje oduzimanje cele vrednosti, određene prvim (ulaznim) operandom (PO) od cele vrednosti, određene drugim (ulaznim) operandom (DO) i postavljanje vrednosti logičkih promenljivih N, M, P i V u skladu sa razlikom. Izvršavanje ove naredbe omogućuje utvrđivanje koja relacija važi između vrednosti njenih ulaznih operanada, tako što u skladu sa razlikom postavlja vrednosti logičkih promenljivih N, M, P i V (za određivanje vrednosti ovih logičkih promenljivih koriste se iste logičke funkcije kao kod aritmetičke naredbe ODUZMI).

## NAREDBE ZA RUKOVANJE BITIMA

Procesor KONCEPT podržava logičke naredbe: I, ILI i NE, koje omogućuju pristupanje pojedinim bitima lokacija. Pristupanje pojedinim bitima lokacija je osnovna namena i naredbi LEVO i DESNO, pa se one svrstavaju u istu grupu sa logičkim naredbama.

I je logička naredba, koja omogućuje određivanje "logičkog i" korespondentnih bita binarnih vrednosti, određenih prvim (ulaznim) operandom (PO) i drugim (ulaznim) operandom (DO). Rezultat izvršavanja ove naredbe se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Obrazovanja bita rezultata ($R_i$) od bita ulaznih operanda ($PO_i$ i $DO_i$, za i = 0, ... , 15), opisuje logička funkcija:

$$R_i = PO_i \& DO_i$$

Nakon izvršavanja naredbe I, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$$
$$M = R_{15}$$
$$P = 0$$
$$V = 0$$

ILI je logička naredba, koja omogućuje određivanje "logičkog ili" korespondentnih bita binarnih vrednosti, određenih prvim (ulaznim) operandom (PO) i drugim (ulaznim) operandom (DO). Rezultat izvršavanja ove naredbe se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Obrazovanja bita rezultata ($R_i$) od bita ulaznih operanda ($PO_i$ i $DO_i$, za i = 0, ... , 15) opisuje logička funkcija:

$R_i = PO_i|DO_i$

Nakon izvršavanja naredbe ILI, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$
$M = R_{15}$
$P = 0$
$V = 0$

NE je logička naredba, koja omogućuje negiranje bita binarne vrednosti, određene prvim (ulaznim) operandom (PO). Rezultat izvršavanja ove naredbe se odlaže u lokaciju, određenu prvim (sada izlaznim) operandom. Obrazovanje bita rezultata ($R_i$) od bita ulaznog operanda ($PO_i$, za i = 0, ... , 15) opisuje logička funkcija:

$R_i = \sim PO_i$

Nakon izvršavanja naredbe NE, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$
$M = R_{15}$
$P = 0$
$V = 0$

LEVO je naredba, koja omogućuje pomeranje svih bita binarne vrednosti, određene prvim (ulaznim) operandom (PO), za jedno mesto u levo. Rezultat izvršavanja ove naredbe se odlaže u lokaciju određenu prvim (sada izlaznim) operandom. Rezultat pomeranja u levo bita binarne vrednosti ulaznog operanda je jednak njenom množenju brojem 2, pa naredba LEVO omogućuje množenje tim brojem. Obrazovanje bita rezultata ($R_i$) od bita ulaznog operanda ($PO_i$, za i = 0, ... , 14) opisuju logičke funkcije:

$R_{i+1} = PO_i$
$R_0 = 0$

Nakon izvršavanja naredbe LEVO, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$
$M = R_{15}$
$P = PO_{15}$
$V = PO_{15} \wedge PO_{14}$

DESNO je naredba, koja omogućuje pomeranje svih bita binarne vrednosti, određene prvim (ulaznim) operandom (PO), za jedno mesto u desno. Rezultat izvršavanja ove naredbe se odlaže u lokaciju određenu prvim (sada izlaznim) operandom. Rezultat pomeranja u desno bita binarne vrednosti ulaznog operanda je jednak njenom deljenju brojem 2, pa naredba DESNO omogućuje deljenje označenih brojeva brojem 2 (ako je deljenik negativan, tada se ispravan količnik dobije tek kada se rezultatu pomeranja doda sadržaj koga, nakon pomeranja, ima logička promenljiva P). Obrazovanje bita rezultata ($R_i$) od bita ulaznog operanda ($PO_i$, za i = 0, ... , 14) opisuju logičke funkcije:

$R_i = PO_{i+1}$
$R_{15} = PO_{15}$

Nakon izvršavanja naredbe DESNO, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$
$M = R_{15}$
$P = PO_0$
$V = 0$

## NAREDBE PREBACIVANJA

Procesor KONCEPT podržava više oblika naredbe prebacivanja PREBACI, koji omogućuju prebacivanje sadržaja između registara procesora i memorijskih lokacija.

PREBACI je naredba, koja omogućuje da se vrednost, određena prvim (ulaznim) operandom, prebaci u lokaciju, određenu drugim (izlaznim) operandom. Izvršavanje ove naredbe ne menja vrednosti logičkih promenljivih N, M, P i V.

## UPRAVLJAČKE NAREDBE

Mašinski oblici prethodnih naredbi se smeštaju, u redosledu izvršavanja, u memorijske lokacije sa uzastopnim rastućim adresama. Zahvaljujući tome, adrese ovakvih memorijskih lokacija, koje su ujedno i **adrese** (mašinskih) **naredbi**, smeštenih u dotične memorijske lokacije, ukazuju na redosled izvršavanja ovih naredbi. Znači pomenute naredbe se izvršavaju u rastućem redosledu svojih adresa. Ovakav redosled izvršavanja naredbi je nepromenljiv i nije podesan za slučajeve u kojima obrada podataka zavisi od vrednosti obrađivanih podataka. Kada obrada

podataka zavisi od vrednosti obrađivanih podataka, potrebno je da se redosled izvršavanja naredbi određuje dinamički (u toku izvršavanja). Upravo to omogućuju **uslovne** i **bezuslovne upravljačke naredbe**.

Procesor KONCEPT podržava više uslovnih upravljačkih naredbi. Jedini operand svake uslovne upravljačke naredbe određuje adresu **ciljne naredbe** koja se izvršava nakon izvršavanja pomenute uslovne upravljačke naredbe i to samo ako je ispunjen uslov ove uslovne upravljačke naredbe. U suprotnom, izvršava se mašinska naredba koja sledi iza pomenute uslovne upravljačke naredbe. Slika 3.2.3 sadrži tabelu sa oznakama svih uslovnih upravljačkih naredbi i logičke izraze koji određuju uslov svake od ovih naredbi:

| Oznaka uslovne upravljačke naredbe | uslov uslovne upravljačke naredbe |
|---|---|
| SKOČI_ZA_== ili SKOČI_ZA_N | N |
| SKOČI_ZA_!= ili SKOČI_ZA_NE_N | ~N |
| SKOČI_ZA_< ili SKOČI_ZA_P | P |
| SKOČI_ZA_>= ili SKOČI_ZA_NE_P | ~P |
| SKOČI_ZA_> | (~P)&(~N) |
| SKOČI_ZA_<= | P\|N |
| SKOČI_ZA_±_< | M^V |
| SKOČI_ZA_±_>= | ~(M^V) |
| SKOČI_ZA_±_> | (~(M^V))&(~N) |
| SKOČI_ZA_±_<= | (M^V) \|N |
| SKOČI_ZA_M | M |
| SKOČI_ZA_NE_M | ~M |
| SKOČI_ZA_V | V |
| SKOČI_ZA_NE_V | ~V |

Slika 3.2.3 Pregled uslovnih upravljačkih naredbi

Oznake uslovnih upravljačkih naredbi, koje se nalaze u istom redu tabele (Slika 3.2.3), predstavljaju sinonime. Jedini rezultat izvršavanja svake od uslovnih upravljačkih naredbi je, eventualno, izmena redosleda izvršavanja naredbi, dok vrednosti logičkih promenljivih N, M, P i V ostaju neizmenjene. Zbog uloge koju imaju za uslovne naredbe, logičke promenljive N, M, P i V se zajedno nazivaju i **uslovni biti** (*condition codes*).

Za uslovne upravljačke naredbe nije važno kako i kada su postavljeni uslovni biti. Ako je vrednosti ovih bita postavilo izvršavanje naredbe UPOREDI, tada

uslovne upravljačke naredbe SKOČI_ZA_== SKOČI_ZA_!=, SKOČI_ZA_<, SKOČI_ZA_>=, SKOČI_ZA_> i SKOČI_ZA_<= omogućuju dinamičke izmene redosleda izvršavanja naredbi, zavisno od važenja relacija za neoznačene cele brojeve, a uslovne upravljačke naredbe SKOČI_ZA_==, SKOČI_ZA_!=, SKOČI_ZA_±_<, SKOČI_ZA_±_>=, SKOČI_ZA_±_> i SKOČI_ZA_±_<= omogućuju isto za označene cele brojeve.

Procesor KONCEPT podržava bezuslovne upravljačke naredbe: SKOČI, POZOVI i NATRAG.

Bezuslovna upravljačka naredba SKOČI ima jedan operand. On određuje adresu ciljne naredbe koja se izvršava nakon izvršavanja naredbe SKOČI. Jedini rezultat izvršavanja upravljačke naredbe SKOČI je izmena redosleda izvršavanja naredbi (vrednosti uslovnih bita ostaju neizmenjene).

Bezuslovna upravljačka naredba POZOVI ima jedan operand. On određuje adresu ciljne naredbe koja se izvršava nakon izvršavanja naredbe POZOVI. Pored izmene redosleda izvršavanja naredbi, izvršavanje upravljačke naredbe POZOVI menja i sadržaj registra `%15`, tako što u njega smešta adresu naredbe koja sledi iza naredbe POZOVI. Adresa, smeštena u registar `%15`, se naziva **povratna adresa** (*return address*), jer omogućuje da, nakon odlaska na izvršavanje ciljne naredbe (koju određuje operand naredbe POZOVI), usledi povratak na izvršavanje naredbe koja sledi iza naredbe POZOVI. Jedini rezultat izvršavanja upravljačke naredbe POZOVI je izmena redosleda izvršavanja naredbi i smeštanje povratne adrese u registar `%15` (vrednosti uslovnih bita ostaju neizmenjene).

Bezuslovna upravljačka naredba NATRAG nema operanada. Jedini rezultat izvršavanja ove naredbe je izmena redosleda izvršavanja naredbi, tako da se izvršavanje nastavlja od naredbe čiju adresu sadrži registar `%15` (vrednosti uslovnih bita ostaju neizmenjene). Podrazumeva se da registar `%15` sadrži povratnu adresu, koju je prethodno pripremilo izvršavanje naredbe POZOVI.

## 3.3. ASEMBLERSKI JEZIK KONCEPT

Opis asemblerskog jezika propisuje način sastavljanja (sintaksu) asemblerskih naredbi i određuje značenje (semantiku) asemblerskih naredbi. Sintaksa se zadaje precizno i koncizno u obliku pravila izraženih pomoću *EBNF* (*Extended Backus-Naur Form*) notacije. U ovoj notaciji, ime pravila se navodi sa leve strane strelice (`->`), a sa desne strane strelice se navode prethodno uvedena imena pravila i simboli koji odgovaraju elementima jezika. *EBNF* notacija koristi razmak za razdvajanje delova pravila, vertikalnu crtu (`|`) za označavanje alternativnih delova pravila, okrugle zagrade za grupisanje delova pravila, uglaste zagrade za označavanje delova pravila koji se mogu pojaviti nijednom ili jednom i vitičaste zagrade za označavanje delova pravila koji se mogu pojaviti nijednom, jednom ili više puta. Znakovi, koje koristi *EBNF* notacija (strelica, razmak, vertikalna crta, okrugle, uglaste i vitičaste zagrade), mogu predstavljati simbole jezika, ako se navedu između navodnika. Primer pravila u *EBNF* notaciji je:

```
malo_slovo -> a|b|c|č|ć|d|đ|e|f|g|h|i|j|k|l|m|n|o|p|r|s|š|t|u|v|z|ž
```

Prethodno pravilo označava da se jedno od slova, navedenih sa desne strane strelice, koristi umesto imena `malo_slovo`. Analogno značenje ima i pravilo `cifra`:

```
cifra -> 0|1|2|3|4|5|6|7|8|9
```

Obrazovanje decimalnog broja opisuje pravilo:

```
decimalni_broj -> cifra{cifra}
```

Podrazumeva se da decimalni broj sadrži najviše do 4 cifre.

Obrazovanje heksadecimalnog broja opisuju pravila:

```
heksa_cifra -> cifra|A|B|C|D|E|F
heksadecimalni_broj -> 0x(heksa_cifra){heksa_cifra}
```

(heksadecimalni broj započinje znakovima 0x, iza kojih slede jedna ili više cifara ili velikih slova `A`, `B`, `C`, `D`, `E`, `F`). Podrazumeva se da heksadecimalni broj sadrži najviše do 6 znakova.

Obrazovanje broja opisuje pravilo:

```
broj -> decimalni_broj|heksadecimalni_broj
```

Obrazovanje labele opisuje pravilo:

```
labela -> malo_slovo{malo_slovo|cifra|_}
```

Podrazumeva se da labela može da sadrži najviše do 30 znakova i da je obavezno jedinstvena u asemblerskom programu.

Obrazovanje neposrednog operanda opisuje pravilo:

```
neposredni_operand -> ($broj|$labela)
```

Kao neposredni operand služi vrednost navedenog broja ili adresa memorijske lokacije koju označava navedena labela.

Obrazovanje direktnog operanda opisuje pravilo:

```
direktni_operand -> labela
```

Kao direktni operand služi memorijska lokacija koju označava navedena labela. Ako je direktni operand ulazni, sadržaj ove lokacije predstavlja ulaznu vrednost, a ako je direktni operand izlazni, ova lokacija se koristi kao izlazna lokacija.

Obrazovanje registarskog operanda opisuju pravila:

```
registar -> %(0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15)
registarski_operand -> registar
```

Kao registarski operand služi registar sa navedenom oznakom. Ako je registarski operand ulazni, sadržaj ovog registra predstavlja ulaznu vrednost, a ako je registarski operand izlazni, ovaj registar se koristi kao izlazna lokacija.

Obrazovanje posrednog operanda opisuje pravilo:

```
posredni_operand -> "("registar")"
```

Kao posredni operand služi memorijska lokacija čiju adresu sadrži navedeni registar. Ako je posredni operand ulazni, sadržaj adresirane lokacije predstavlja ulaznu vrednost, a ako je posredni operand izlazni, adresirana lokacija se koristi kao izlazna lokacija.

Obrazovanje indeksnog operanda opisuje pravilo:

```
indeksni_operand -> (broj|labela)"("registar")"
```

Kao indeksni operand služi memorijska lokacija čiju adresu određuje zbir dva sastojka. Jedan sastojak je navedeni broj ili adresa memorijske lokacije koju označava navedena labela, a drugi sastojak je sadržaj navedenog registra. Ako je indeksni operand ulazni, sadržaj adresirane lokacije predstavlja ulaznu vrednost, a ako je indeksni operand izlazni, adresirana lokacija se koristi kao izlazna lokacija.

Obrazovanje razmaka opisuje pravilo:

```
razmak -> " "{" "}
```

(između apostrofa se nalazi razmak).

Obrazovanje nove linije opisuje pravilo:

```
nova_linija -> "početak linije"{" "}
```

(između prvog para apostrofa se nalazi upravljački znak koji omogućuje pozicioniranje na početak linije, a između drugog para apostrofa se nalazi razmak).

Obrazovanje osnovnih asemblerskih naredbi opisuje pravilo:

```
osnovna_naredba -> nova_linija [labela:] razmak
           ((SABERI|SABERI_P) razmak registarski_operand,registarski_operand
           |(ODUZMI|ODUZMI_P) razmak registarski_operand,registarski_operand
           |UPOREDI           razmak registarski_operand,registarski_operand
           |(I|ILI)           razmak registarski_operand,registarski_operand
           |(DODAJ_1|ODBIJ_1) razmak registarski_operand
           |(NE|LEVO|DESNO)   razmak registarski_operand)
```

Obrazovanje asemblerskih naredbi prebacivanja opisuje pravilo:

```
naredba_prebacivanja -> nova_linija [labela:] razmak
           (PREBACI_RR razmak registarski_operand,registarski_operand
           |PREBACI_NR razmak neposredni_operand,registarski_operand
           |PREBACI_DR razmak direktni_operand,registarski_operand
           |PREBACI_PR razmak posredni_operand,registarski_operand
           |PREBACI_IR razmak indeksni_operand,registarski_operand
           |PREBACI_RD razmak registarski_operand,direktni_operand
           |PREBACI_RP razmak registarski_operand,posredni_operand
           |PREBACI_RI razmak registarski_operand,indeksni_operand)
```

Obrazovanje upravljačkih asemblerskih naredbi opisuje pravilo:

```
upravljačka_naredba -> nova_linija [labela:] razmak
                       ((SKOČI
                       |SKOČI_ZA_==
                       |SKOČI_ZA_!=
                       |SKOČI_ZA_<
                       |SKOČI_ZA_>=
                       |SKOČI_ZA_>
                       |SKOČI_ZA_<=
                       |SKOČI_ZA_±_<
                       |SKOČI_ZA_±_>=
                       |SKOČI_ZA_±_>
                       |SKOČI_ZA_±_<=
                       |SKOČI_ZA_N
                       |SKOČI_ZA_NE_N
                       |SKOČI_ZA_P
                       |SKOČI_ZA_NE_P
                       |SKOČI_ZA_M
                       |SKOČI_ZA_NE_M
                       |SKOČI_ZA_V
                       |SKOČI_ZA_NE_V
                       |POZOVI) razmak labela)
                       |NATRAG)
```

Upravljačke asemblerske naredbe koriste adresu labele (koja je navedena na mestu operanda) kao adresu ciljne naredbe.

Asemblerske naredbe u mašinski oblik prevodi asembler. Podrazumeva se da on smešta mašinske oblike naredbi u memorijske lokacije. Međutim, asembleru se mora izričito navesti da zauzme i eventualno inicijalizuje memorijske lokacije za promenljive. To omogućuju **asemblerske direktive**. Asemblerska direktiva `ZAUZMI` omogućuje zauzimanje više memorijskih lokacija. Njen jedini operand određuje broj memorijskih lokacija koje se zauzimaju. Asemblerska direktiva `NAPUNI` omogućuje zauzimanje jedne memorijske lokacije, čiji sadržaj određuje jedini operand ove direktive. Obrazovanje asemblerskih direktiva opisuje pravilo:

```
direktiva -> nova_linija [labela:] razmak
             (ZAUZMI|NAPUNI) razmak broj
```

Važno je uočiti da su direktive upućene asembleru (prevodiocu), a ne procesoru.

Znači, direktive nisu izvršne (one u toku asembliranja izazivaju zauzimanje memorijskih lokacija, u koje se smeštaju podaci). Za razliku od njih, naredbe su upućene procesoru, pa su, prema tome, izvršne (njima odgovaraju memorijske lokacije koje sadrže mašinske oblike naredbi).

Obrazovanje tela asemblerskog programa opisuje pravilo:

```
telo -> {direktiva
        |osnovna_naredba
        |naredba_prebacivanja
        |upravljačka_naredba}
```

Uputno je da, u telu asemblerskog programa, sve direktive prethode naredbama, radi sprečavanja da se sadržaj memorijske lokacije, predviđene za smeštanje podataka, upotrebi kao mašinski oblik naredbe.

Obrazovanje celog asemblerskog programa, ili njegovog samostalnog dela, opisuje pravilo:

```
program -> POČETAK razmak labela telo nova_linija KRAJ
```

Iza reči `POČETAK` i razamaka navodi se **ulazna labela**. Podrazumeva se da u telu asemblerskog programa ulazna labela neposredno prethodi **ulaznoj asemblerskoj naredbi** od koje započinje izvršavanje asemblerskog programa. Izvršavanje asemblerskih naredbi zatim teče sekvencijalno ka poslednjoj asemblerskoj naredbi iz njegovog tela, ako upravljačke naredbe ne izmene redosled izvršavanja naredbi.

## 3.4. PRIMERI ASEMBLERSKIH PROGRAMA

### RAČUNANJE NAJVEĆEG ZAJEDNIČKOG DELIOCA

Računanje najvećeg zajedničkog delioca opisuje asemblerski program:

```
                POČETAK         ulaz
ulaz:           PREBACI_NR      $12,%0
                PREBACI_NR      $10,%1
ponovo:         UPOREDI         %1,%0
                SKOČI_ZA_==     kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %1,%0
                SKOČI           ponovo
manje:          ODUZMI          %0,%1
                SKOČI           ponovo
kraj:           SKOČI           kraj
                KRAJ
```

Telo asemblerskog programa započinje opisom punjenja registara `%0` i `%1` početnim vrednostima (`PREBACI_NR`). Zatim sledi opis poređenja (`UPOREDI`) vrednosti registara `%0` i `%1`. Ako su pomenute vrednosti jednake, registri `%0` i `%1` sadrže najveći zajednički delilac, pa sledi odlazak (`SKOČI_ZA_==`) u beskonačnu petlju, koja

označava kraj programa. Ako je vrednost registra `%0` manja od vrednosti registra `%1` (`SKOČI_ZA_<`), vrednost registra `%1` se umanjuje za vrednost registra `%0`. U suprotnom, vrednost registra `%0` se umanjuje za vrednost registra `%1`. U oba slučaja, nakon izmene vrednosti veće promenljive, sledi opet poređenje vrednosti ovih promenljivih.

## IZLAZAK VAN OPSEGA KOD NEOZNAČENIH CELIH BROJEVA

Otkrivanje izlaska van opsega, kod sabiranja neoznačenih celih brojeva u dvostrukoj preciznosti, opisuje asemblerski program:

```
                POČETAK         ulaz
a_donji:        NAPUNI          0xFFFF
a_gornji:       NAPUNI          0xFFFF
b_donji:        NAPUNI          0xFFFF
b_gornji:       NAPUNI          0xFFFF
greška:         NAPUNI          0
ulaz:           PREBACI_DR      a_donji,%0
                PREBACI_DR      a_gornji,%1
                PREBACI_DR      b_donji,%2
                PREBACI_DR      b_gornji,%3
                SABERI          %2,%0
                SABERI_P        %3,%1
                SKOČI_ZA_P      van_opsega
                SKOČI           kraj
van_opsega:     PREBACI_NR      $1,%4
                PREBACI_RD      %4,greška
kraj:           SKOČI           kraj
                KRAJ
```

Za promenljive `a` i `b`, čije vrednosti se sabiraju, zauzete su i inicijalizovane po dve lokacije. Za promenljivu `greška`, u koju se smešta indikacija izlaska van opsega (`0` – nije bilo izlaska van opsega, `1` – bilo je izlaska van opsega), zauzeta je jedna lokacija i inicijalizovana na vrednost `0`. Nakon prebacivanja delova sabiraka u registre (`PREBACI_DR`), sabiraju se (`SABERI`) manje značajni (donji) delovi vrednosti promenljivih `a` i `b`. Zatim se sabiraju (`SABERI_P`) značajniji (gornji) delovi vrednosti ovih promenljivih i prenos iz prethodnog sabiranja. Pojava prenosa u drugom sabiranju ukazuje na izlazak van opsega (`SKOČI_ZA_P`), što predstavlja grešku i označava se smeštanjem vrednosti `1` u promenljivu `greška`.

## IZLAZAK VAN OPSEGA KOD OZNAČENIH CELIH BROJEVA

Otkrivanje izlaska van opsega, kod sabiranja označenih celih brojeva u dvostrukoj preciznosti, opisuje asemblerski program:

```
                POČETAK         ulaz
a_donji:        NAPUNI          0xFFFF
a_gornji:       NAPUNI          0xFFFF
b_donji:        NAPUNI          0xFFFF
b_gornji:       NAPUNI          0xFFFF
greška:         NAPUNI          0
ulaz:           PREBACI_DR      a_donji,%0
                PREBACI_DR      a_gornji,%1
                PREBACI_DR      b_donji,%2
                PREBACI_DR      b_gornji,%3
                SABERI          %2,%0
                SABERI_P        %3,%1
                SKOČI_ZA_V      van_opsega
                SKOČI           kraj
van_opsega:     PREBACI_NR      $1,%4
                PREBACI_RD      %4,greška
kraj:           SKOČI           kraj
                KRAJ
```

Jedina razlika ovog i prethodnog programa je u načinu otkrivanja izlaska van opsega, nakon sabiranja značajnijih delova vrednosti promenljivih `a` i `b`. U slučaju sabiranja označenih celih brojeva, na izlazak van opsega ukazuje postavljenost logičke promenljive V (`SKOČI_ZA_V`), jer se podrazumeva komplement 2 predstava označenih brojeva.

## RUKOVANJE MAŠINSKOM NORMALIZOVANOM FORMOM

Rukovanje mašinskom normalizovanom formom binarne predstave realnih brojeva zahteva pristupanje pojedinim bitima lokacija. Ako se za mašinsku normalizovanu formu koristi cela lokacija (16 bita), tada prvi bit s leva može da sadrži predznak broja, narednih 8 bita mogu da sadrže podešeni eksponent (za podešavanje se koristi konstanta $10000000_2 == 2^{8-1}$), a preostalih 7 bita mogu da sadrže 7 manje značajnih bita frakcije (najznačajniji bit frakcije je uvek 1 i ne prikazuje se). Sledi primer broja predstavljenog u upravo opisanoj mašinskoj normalizovanoj formi:

$-1.5_{10} ==$
$-1.1000000_2 \times 2^0 ==$
$1100000001000000_2$

Zauzimanje memorijske lokacije za realnu promenljivu i smeštanje u nju prethodne vrednosti u mašinskoj normalizovanoj formi, opisuje naredna asemblerska direktiva:

```
realan:         NAPUNI          0xC040
```

Izmenu samo predznaka vrednosti prethodne realne promenljive, opisuju asemblerske naredbe:

```
                PREBACI_DR      realan,%0
                PREBACI_NR      $0x7FFF,%1
                I               %1,%0
```

a vraćanje predznaka na prethodnu vrednost, opisuje asemblerska naredba:

```
                PREBACI_NR      $0x8000,%1
                ILI             %1,%0
```

Izdvajanje eksponenta i njegovo pomeranje za 7 mesta u desno, radi njegove pripreme za aritmetiku eksponenata, opisuju asemblerske naredbe:

```
                PREBACI_DR      realan,%0
                PREBACI_NR      $0x7F80,%1
                I               %1,%0
                PREBACI_NR      $7,%2
ponovo:         DESNO           %0
                ODBIJ_1         %2
                SKOČI_ZA_NE_N   ponovo
```

## RUKOVANJE LOGIČKIM VREDNOSTIMA

Dodelu (logičke) vrednosti relacionog izraza dvema promenljivim, opisuje sekvenca programa, izražena programskim jezikom C:

```
int     a,b;
char    c,d;
...
c = (a!=b);
d = c;
```

Prethodnoj sekvenci programa odgovara sekvenca asemblerskog programa:

```
a:              ZAUZMI          1
b:              ZAUZMI          1
c:              ZAUZMI          1
d:              ZAUZMI          1
                ...
                PREBACI_DR      a,%0
                PREBACI_DR      b,%1
                PREBACI_NR      $1,%2
                UPOREDI         %1,%0
                SKOČI_ZA_!=     dalje
                PREBACI_NR      $0,%2
dalje:          PREBACI_RD      %2,c
                PREBACI_RD      %2,d
```

(konstante `1` i `0` predstavljaju vrednosti logičkih promenljivih `c` i `d`).

## RAČUNANJE VREDNOSTI CELOBROJNOG IZRAZA

Dodeljivanje vrednosti celobrojnog izraza promenljivoj opisuje sekvenca programa, izražena programskim jezikom C (podrazumeva se da izraz sadrži samo neoznačene cele vrednosti, za koje izlazak van opsega nije moguć):

```
unsigned a,b,c;
...
c = (a-b)*2+(a+b)/2;
```

Prethodnoj sekvenci programa odgovara sekvenca asemblerskog programa:

```
a:              ZAUZMI          1
b:              ZAUZMI          1
c:              ZAUZMI          1
                ...
                PREBACI_DR      a,%0
                PREBACI_DR      b,%1
                PREBACI_DR      a,%2
                ODUZMI          %1,%2
                LEVO            %2
                SABERI          %1,%0
                DESNO           %0
                SABERI          %0,%2
                PREBACI_RD      %2,c
```

## RUKOVANJE NIZOVIMA

Brojanje dana sa srednjom dnevnom temperaturom iz posmatranog intervala temperatura, i to u određenom periodu godine, opisuje sekvenca programa, izražena programskim jezikom C:

```
int     t[365];
int     prvi,poslednji,donja,gornja,i;
int     broj = 0;
...
for (i=prvi;i<=poslednji;i++)
 if ((t[i]>=donja)&&(t[i]<=gornja))
  broj++;
```

(tabela `t` sadrži srednje dnevne temperature za 365 dana u godini, promenljive `prvi` i `poslednji` sadrže redne brojeve dana na granicama zadanog perioda u godini, promenljive `donja` i `gornja` sadrže temperaturne granice posmatranog intervala temperatura, promenljiva `i` služi kao indeks elemenata tabele `t`, a promenljiva `broj` sadrži broj dana u određenom periodu godine, u kojima je srednja dnevna temperatura iz posmatranog intervala). Prethodnoj sekvenci programa odgovara sekvenca asemblerskog programa:

```
t:              ZAUZMI          365
prvi:           ZAUZMI          1
poslednji:      ZAUZMI          1
donja:          ZAUZMI          1
gornja:         ZAUZMI          1
broj:           ZAUZMI          1
                ...
                PREBACI_NR      $0,%0
                PREBACI_DR      prvi,%1
                PREBACI_DR      poslednji,%2
                PREBACI_DR      donja,%3
                PREBACI_DR      gornja,%4
provera:        UPOREDI         %2,%1
                SKOČI_ZA_>      izlaz
ponovo:         UPOREDI         %3,t(%1)
                SKOČI_ZA_±_<    naredni
                UPOREDI         %4,t(%1)
                SKOČI_ZA_±_>    naredni
                DODAJ_1         %0
naredni:        DODAJ_1         %1
                SKOČI           provera
izlaz:          PREBACI_RD      %0,broj
```

(naredba sa labelom `provera` odgovara proveri kraja `for` iskaza, naredba sa labelom `ponovo` i četiri naredbe iza nje odgovaraju `if` iskazu, a naredba sa labelom `naredni` odgovara povećanju indeksa `for` iskaza).

## RUKOVANJE SLOGOVIMA

Dodelu vrednosti slogu, koji sadrži predstavu vremena, izraženu brojem sati, minuta i sekundi, opisuje sekvenca programa, izražena programskim jezikom C:

```
struct vreme
 {unsigned sat, minut, sekund;} a,b;
       ...
b = a;
```

Prethodnoj sekvenci programa odgovara sekvenca asemblerskog programa:

```
a:              ZAUZMI          3
b:              ZAUZMI          3
                ...
                PREBACI_NR      $0,%0
                PREBACI_IR      a(%0),%1
                PREBACI_RI      %1,b(%0)
                DODAJ_1         %0
                PREBACI_IR      a(%0),%1
                PREBACI_RI      %1,b(%0)
                DODAJ_1         %0
                PREBACI_IR      a(%0),%1
                PREBACI_RI      %1,b(%0)
```

Za slog promenljive `a` i `b` su zauzete po tri memorijske lokacije (za svako polje po jedna lokacija). Registar `%o` sadrži relativnu udaljenost polja ovih promenljivih od početnih memorijskih lokacija ovih promenljivih.

## 3.5. POTPROGRAM

**Potprogram** omogućuje opisivanje izračunavanja vrednosti, odnosno opisivanje rukovanja strukturama podataka (vrednostima složenih tipova). **Opis** potprograma prate njegovi **pozivi**. Razdvajanje opisa i poziva potprograma omogućuje korišćenje (poziv) potprograma samo na osnovu poznavanja njegove namene, a bez poznavanja njegovog opisa. To je naročito važno za postepen razvoj programa, jer znači da se svaki potprogram može koristiti (pozivati) i pre dovršenja njegovog opisa, pod uslovom da je njegova namena precizirana.

Opštost opisu potprograma daje upotreba **parametara**, koja omogućuje da se potprogram odnosi na sve vrednosti iz skupa vrednosti, određenog tipom parametara. Svako izvršavanje potprograma se odnosi na konkretne vrednosti ovih parametara, koje se nazivaju **argumenti**. Njih određuje poziv potprograma. Do izvršavanja potprograma dolazi nakon njegovog poziva. U pozivu potprograma se odredi povratna adresa, od koje se nastavlja izvršavanje programa, nakon izvršavanja potprograma. Izvršavanje potprograma započinje u **ulaznoj tački** opisa potprograma, a završava u **izlaznoj tački** ovog opisa. U ulaznoj tački potprograma se zauzimaju lokacije za **lokalne promenljive** potprograma, a u izlaznoj tački potprograma se ove lokacije oslobađaju. Lokalne promenljive potprograma se nazivaju i **dinamičke**, jer postoje samo u toku izvršavanja potprograma, za razliku od **statičkih** ili **globalnih** promenljivih koje postoje za sve vreme izvršavanja programa i koje su definisane van potprograma. **Područje važenja** (*scope*) lokalnih promenljivih je od mesta njihove definicije do kraja potprograma, a područje važenja globalnih promenljivih je od mesta njihove definicije do kraja programa.

Potprogrami se dele na **funkcije** i **procedure**. Funkcije omogućuju opisivanje izračunavanja vrednosti, odnosno preuzimanje vrednosti promenljive. One imaju jedan ili više **ulaznih parametara** (sa statusom lokalnih promenljivih) i jednu **povratnu vrednost**. Povratna vrednost funkcije se koristi u izrazu, koji sadrži poziv funkcije.

Procedure omogućuju opisivanje izmena vrednosti promenljivih. One mogu da imaju više ulaznih parametara, ali i više **ulazno-izlaznih parametara**.

Računanje najvećeg zajedničkog delioca može biti opisano u obliku funkcije. U nastavku je naveden ovakav opis funkcije, izražen programskim jezikom C:

```
unsigned nzd(unsigned a, unsigned b)
{while (a!=b)
  if (a>b)
   a = a-b;
  else
   b = b-a;
 return(a);};
```

Ulazni parametri `a` i `b`, opisa ove funkcije, određuju vrednosti za koje se računa najveći zajednički delilac. Najveći zajednički delilac se vraća kao povratna vrednost funkcije `nzd`. Ako su `x`, `y` i `z` celobrojne promenljive, tada poziv funkcije `nzd`:

```
z = nzd(12,10);
```

dodeljuje promenljivoj `z` najveći zajednički delilac celih brojeva `12` i `10`, a poziv iste funkcije:

```
z = nzd(x,y);
```

dodeljuje promenljivoj `z` najveći zajednički delilac vrednosti promenljivih `x` i `y`.

Računanje najvećeg zajedničkog delioca može biti opisano i u obliku procedure. U nastavku je naveden ovakav opis procedure, izražen programskim jezikom C:

```
void nzd(unsigned a, unsigned b, unsigned *c)
{while (a!=b)
  if (a>b)
   a = a-b;
  else
   b = b-a;
 *c = a;};
```

Ulazni parametri `a` i `b`, opisa ove procedure, određuju vrednosti za koje se računa najveći zajednički delilac. Najveći zajednički delilac se dodeljuje ulazno-izlaznom parametru `c`. Ako su `x`, `y` i `z` celobrojne promenljive, tada poziv procedure `nzd`:

```
nzd(12,10,&z);
```

dodeljuje promenljivoj `z` najveći zajednički delilac celih brojeva `12` i `10`, a poziv iste procedure:

```
nzd(x,y,&z);
```

dodeljuje promenljivoj `z` najveći zajednički delilac vrednosti promenljivih `x` i `y`.

## ASEMBLERSKI OBLIK POTPROGRAMA

Potprogrami nisu isključivo vezani za programske jezike visokog nivoa. Opisi i pozivi potprograma se mogu javiti i u asemblerskom programu, ali su detalji opisivanja i poziva potprograma prepušteni programeru. On se mora pobrinuti za prenos argumenata, za odlazak na izvršavanje potprograma i za povratak iz potprograma, nakon njegovog izvršavanja. Asemblerski jezik KONCEPT, olakšava

zadatak programera, jer sadrži upravljačku naredbu **POZOVI**, koja omogućuje odlazak na izvršavanje potprograma, ali i smeštanje povratne adrese u registar **%15**. Zahvaljujući tome, povratak iz potprograma omogućuje upravljačka naredba **NATRAG**.

Registri procesora KONCEPT (uz izuzetak registra **%15**) mogu biti iskorišćeni za prenos argumenata, za šta je neophodna usaglašenost opisa i poziva potprograma.

Uz pretpostavku da se registar **%1** koristi za smeštanje vrednosti prvog ulaznog parametra, registar **%2** za smeštanje vrednosti drugog ulaznog parametra, a registar **%0** za smeštanje povratne vrednosti funkcije, asemblerski ekvivalent opisa funkcije **nzd** izgleda:

```
nzd:        UPOREDI       %2,%1
            SKOČI_ZA_=    kraj
            SKOČI_ZA_<    manje
veće:       ODUZMI        %2,%1
            SKOČI         nzd
manje:      ODUZMI        %1,%2
            SKOČI         nzd
kraj:       PREBACI_RR    %1,%0
            NATRAG
```

Ako su za celobrojne promenljive **x**, **y** i **z** zauzete i inicijalizovane memorijske lokacije:

```
x:          NAPUNI        9
y:          NAPUNI        3
z:          ZAUZMI        1
```

tada asemblerske naredbe:

```
            PREBACI_NR    $12,%1
            PREBACI_NR    $10,%2
            POZOVI        nzd
            PREBACI_RD    %0,z
```

opisuju poziv asemblerskog ekvivalenta funkcije **nzd**, u kome se izračuna najveći zajednički delilac celih brojeva **10** i **12**, a povratna vrednost ove funkcije dodeli promenljivoj **z**. U prethodnom primeru, u registre **%1** i **%2** se smeste argumenti pre odlaska na izvršavanje potprograma, a nakon povratka iz potprograma sadržaj registra **%0** (povratna vrednost) se smesti u promenljivu **z**. Slično, asemblerske naredbe:

```
            PREBACI_DR    x,%1
            PREBACI_DR    y,%2
            POZOVI        nzd
            PREBACI_RD    %0,z
```

opisuju poziv asemblerskog ekvivalenta funkcije `nzd`, u kome se računa najveći zajednički delilac vrednosti promenljivih `x` i `y`, a povratna vrednost ove funkcije dodeli promenljivoj `z`.

Uz pretpostavku da se registar `%1` koristi za smeštanje vrednosti prvog ulaznog parametra, registar `%2` za smeštanje vrednosti drugog ulaznog parametra, a registar `%3` za smeštanje adrese trećeg ulazno-izlaznog parametra, asemblerski ekvivalent opisa procedure `nzd` izgleda:

```
nzd:            UPOREDI         %2,%1
                SKOČI_ZA_=      kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %2,%1
                SKOČI           nzd
manje:          ODUZMI          %1,%2
                SKOČI           nzd
kraj:           PREBACI_RP      %1,(%3)
                NATRAG
```

Ako su za celobrojne promenljive `x`, `y` i `z` zauzete i inicijalizovane memorijske lokacije:

```
x:              NAPUNI          9
y:              NAPUNI          3
z:              ZAUZMI          1
```

tada asemblerske naredbe:

```
                PREBACI_NR      $12,%1
                PREBACI_NR      $10,%2
                PREBACI_NR      $z,%3
                POZOVI          nzd
```

opisuju poziv asemblerskog ekvivalenta procedure `nzd`, u kome se izračuna najveći zajednički delilac celih brojeva `10` i `12` i dodeli ulazno-izlaznoj promenljivoj `z`. U prethodnom primeru, pre odlaska na izvršavanje potprograma, u registre `%1` i `%2` se smeste vrednosti čiji najveći zajednički delilac se računa, a u registar `%3` se smesti adresa memorijske lokacije, namenjene za prihvatanje izračunatog najvećeg zajedničkog delioca. Slično prethodnom, opis poziva asemblerskog ekvivalenta procedure `nzd`, u kome se izračuna najveći zajednički delilac vrednosti promenljivih `x` i `y` i dodeli ulazno-izlaznoj promenljivoj `z`, sadrže asemblerske naredbe:

```
                PREBACI_DR      x,%1
                PREBACI_DR      y,%2
                PREBACI_NR      $z,%3
                POZOVI          nzd
```

## 3.6. MAKRO

Korišćenje potprograma uvek povećava broj izvršavanih naredbi, čime se produžava vreme izvršavanja programa. Pored toga, korišćenje potprograma nekad izaziva i povećanje dužine programa, što znači da treba više memorijskih lokacija za smeštanje mašinskog oblika programa. Prethodno se može pokazati na primeru opisa i poziva asemblerskog ekvivalenta procedure `nzd`:

```
x:              NAPUNI          9
y:              NAPUNI          3
z:              ZAUZMI          1
                ...
nzd:            UPOREDI         %2,%1
                SKOČI_ZA_==     kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %2,%1
                SKOČI           nzd
manje:          ODUZMI          %1,%2
                SKOČI           nzd
kraj:           PREBACI_RP      %1,(%3)
                NATRAG
                ...
                PREBACI_DR      x,%1
                PREBACI_DR      y,%2
                PREBACI_NR      $z,%3
                POZOVI          nzd
```

Ako bi poziv potprograma bio zamenjen opisom potprograma, tada bi prethodnoj sekvenci asemblerskog programa odgovarala sledeća sekvenca:

```
x:              NAPUNI          9
y:              NAPUNI          3
z:              ZAUZMI          1
                ...
nzd:            PREBACI_DR      x,%1
                PREBACI_DR      y,%2
                PREBACI_NR      $z,%3
nzd1:           UPOREDI         %2,%1
                SKOČI_ZA_==     kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %2,%1
                SKOČI           nzd1
manje:          ODUZMI          %1,%2
                SKOČI           nzd1
kraj:           PREBACI_RP      %1,(%3)
```

Potprogram, za koga se podrazumeva da se na mestu njegovog poziva pojave naredbe iz njegovog opisa, se naziva **makro** (veliki) potprogram.

Iz prethodnih primera sledi da se, po svakom pozivu, izvrše 2 naredbe manje u slučaju makro potprograma, nego u slučaju običnog potprograma. Zavisnost broja naredbi (odnosno dužine) programa od broja poziva potprograma za ova dva slučaja je prikazana, za prethodni primer, tabelarno (Slika 3.6.1).

| broj poziva | broj naredbi za običan potprogram | broj naredbi za makro potprogram |
|---|---|---|
| 1 | 13 | 11 |
| 2 | 17 | 22 |
| 3 | 21 | 33 |

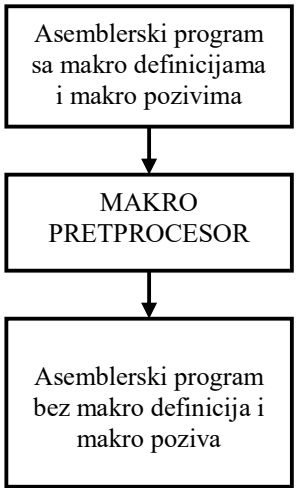
Slika 3.6.1 Zavisnost broja naredbi od broja poziva asemblerskog ekvivalenta potprograma `nzd`

Znači, sa stanovišta dužine programa, u posmatranim slučajevima (Slika 3.6.1) bolji je makro, ako se potprogram koristi samo jedan put.

Upotreba makroa podrazumeva da postoji **makro definicija** sa opisom potprograma i **makro poziv**, koji označava mesto korišćenja makroa. Ako makro definicija, radi opštosti, zahteva parametre, tada odgovarajući makro poziv sadrži adekvatne argumente. Oni zamenjuju parametre u makro definiciji, a onda tako modifikovani opis zamenjuje makro poziv u tekstu programa.

Korišćenje makroa uvek dovodi do skraćenja vremena izvršavanja programa, a dovodi i do smanjenja dužine programa, ako je u programu prisutan samo jedan poziv makroa.

**Zamenu** ili **supstituciju** makro poziva, eventualno modifikovanim, naredbama iz makro definicije obavlja poseban program koji se naziva **makro pretprocesor** (*macro preprocessor*). On procesira (obrađuje) programski tekst pre asemblera i tako, praveći tekstualne izmene, priprema tekst asemblerskog programa za asembliranje (Slika 3.6.2).




Slika 3.6.2 Uloga makro pretprocesora

U toku izmena programskog teksta, makro pozive zamenjuju naredbe iz makro definicija. Tome eventualno prethodi modifikacija ovih naredbi. U toku nje parametre iz makro definicija zamenjuju argumenti iz makro poziva. Zadatak makro pretprocesora se znatno pojednostavljuje, ako u tekstu asemblerskog programa svaka makro definicija prethodi svojim makro pozivima. Makro definicije i makro pozivi su namenjeni makro pretprocesoru (kao što su direktive namenjene asembleru), pa zato makro pozivi nisu izvršni (kao ni direktive). Zbog toga, makro definicije mogu sadržati i direktive, koje zamenjuju odgovarajuće makro pozive.

## OPIS MAKRO DEFINICIJA I MAKRO POZIVA

Za opis obrazovanja makro definicija i makro poziva zgodna je *EBNF* notacija. Tako, pravilo:

```
veliko_slovo -> A|B|C|Č|Ć|D|Đ|E|F|G|H|I|J|K|L|M|N|O|P|R|S|Š|T|U|V|Z|Ž
```

zadaje repertoar znakova, potreban za obrazovanje makro imena. Obrazovanje makro imena opisuje pravilo:

```
ime -> veliko_slovo{veliko_slovo|cifra|_}
```

Podrazumeva se da makro ime sadrži najviše do 30 znakova i da je obavezno jedinstveno u asemblerskom programu.

Obrazovanje makro parametara opisuje pravilo:

```
parametar -> labela|ime
```

Parametar je obavezno jedinstven u makro definiciji.

Obrazovanje makro argumenata opisuje pravilo:

```
argument -> (broj|parametar)[(+|-|*|/)(broj|parametar)]
            |registar
```

Neophodno je da argument bude usaglašen sa ulogom parametra koga zamenjuje. Kada se kao argument pojavi celobrojni izraz, čija oba operanda su brojevi, tada makro pretprocesor izračuna izraz i dobijenu vrednost koristi kao argument.

Obrazovanje makro definicije opisuje pravilo:

```
makro_definicija -> nova_linija ime razmak MAKRO razmak {parametar[,]}
                    telo nova_linija KRAJ
```

(telo makro definicije se ne razlikuje od tela asemblerskog programa).

Obrazovanje makro poziva opisuje pravilo:

```
makro_poziv -> nova_linija [labela:] razmak ime razmak {argument[,]}
```

Obrazovanje proširenog tela asemblerskog programa, odnosno proširenog tela makro definicije, opisuje pravilo:

```
telo -> {direktiva
        |osnovna_naredba
        |naredba_prebacivanja
        |upravljačka_naredba
        |makro_definicija
        |makro_poziv}
```

U makro pozivu se koristi ime prethodno definisanog makroa, sa čijim parametrima se, po broju i mestu, slažu argumenti poziva. Makro pozive zamenjuju tela makroa, u kojima su parametri zamenjeni argumentima. Parametri se mogu pojaviti svugde u telu makro definicije (na primer, na mestu: labele, imena naredbe, imena direktive, operanda, parametra ili argumenta).

## PRIMERI MAKRO DEFINICIJA I MAKRO POZIVA

Makro definicija:

```
NZD             MAKRO           a,b,c
                PREBACI_DR      a,%1
                PREBACI_DR      b,%2
                PREBACI_NR      $c,%3
nzd:            UPOREDI         %2,%1
                SKOČI_ZA_==     kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %2,%1
                SKOČI           nzd
manje:          ODUZMI          %1,%2
                SKOČI           nzd
kraj:           PREBACI_RP      %1,(%3)
                KRAJ
```

opisuje računanje najvećeg zajedničkog delioca. Makro poziv:

```
                NZD             x,y,z
```

zamenjuju naredbe:

```
                PREBACI_DR      x,%1
                PREBACI_DR      y,%2
                PREBACI_NR      $z,%3
nzd:            UPOREDI         %2,%1
                SKOČI_ZA_==     kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %2,%1
                SKOČI           nzd
manje:          ODUZMI          %1,%2
                SKOČI           nzd
kraj:           PREBACI_RP      %1,(%3)
```

Višestruki pozivi makroa `NZD` u jednom asemblerskom programu izazivaju višestruku pojavu istih labela (navedenih u telu ovog makroa). Ovaj problem se rešava ili korišćenjem parametara na mestu spornih labela, ili prepuštanjem makro pretprocesoru da, u ovakvim slučajevima, sam generiše jedinstvene labele.

## MAKRO DEFINICIJA SA MAKRO DEFINICIJOM

Telo makro definicije može sadržati drugu makro definiciju. Pri tome poziv unutrašnjeg makroa obavezno sledi iza poziva vanjskog makroa, jer je tek tada unutrašnja makro definicija potpuno oblikovana (njen konačan izgled, u opštem slučaju, zavisi od argumenata poziva vanjskog makroa). To znači, na primer, da je moguće napisati opštu makro definiciju koja opisuje aritmetiku u dvostrukoj preciznosti. Na osnovu ovakve definicije mogu se stvoriti makro definicije koje opisuju sabiranje ili oduzimanje u dvostrukoj preciznosti. Prethodno ilustruje makro definicija:

```
DVOSTRUKO       MAKRO           NAZIV,OPERACIJA1,OPERACIJA2
NAZIV           MAKRO
                OPERACIJA1      %2,%0
                OPERACIJA2      %3,%1
                KRAJ
                KRAJ
```

(prethodna makro definicija podrazumeva da se u registrima `%0` i `%1` nalaze manje značajni i značajniji deo prvog broja u dvostrukoj preciznosti, a da se u registrima `%2` i `%3` nalaze manje značajni i značajniji deo drugog broja u dvostrukoj preciznosti, kao i da se zanemaruje izlazak van opsega). Makro poziv:

```
                DVOSTRUKO       SABERI_2,SABERI,SABERI_P
```

zamenjuje makro definicija:

```
SABERI_2        MAKRO
                SABERI          %2,%0
                SABERI_P        %3,%1
                KRAJ
```

koja opisuje sabiranje u dvostrukoj preciznosti, a makro poziv:

```
                DVOSTRUKO       ODUZMI_2,ODUZMI,ODUZMI_P
```

zamenjuje makro definicija:

```
ODUZMI_2        MAKRO
                ODUZMI          %2,%0
                ODUZMI_P        %3,%1
                KRAJ
```

koja opisuje oduzimanje u dvostrukoj preciznosti.

## USLOVNO ASEMBLIRANJE

Makro pretprocesor omogućuje oblikovanje, ne samo makro definicija, nego i drugih delova asemblerskog programa, ako podržava uslovno asembliranje. Za uslovno asembliranje je potrebna uslovna direktiva koja određuje pod kojim uslovom makro pretprocesor propušta na asembliranje deo asemblerskog programa, sadržan u telu uslovne direktive. Ako navedeni uslov nije ispunjen, tada se deo asemblerskog programa iz tela uslovne direktive ne uključuje u asemblerski program.

Obrazovanje uslova uslovne direktive opisuje pravilo:

```
uslov -> (broj|parametar)(==|!=|>|<|>=|<=)(broj|parametar)
```

(uslov uslovne direktive ima oblik relacije, od čijeg važenja zavisi tačnost uslova).

Obrazovanje uslovne direktive opisuje pravilo:

```
uslovna_direktiva -> nova_linija USLOVNO razmak uslov telo nova_linija KRAJ
```

Telo uslovne direktive se obrazuje na isti način kao i telo asemblerskog programa ili makro definicije. Uslovna direktiva je upućena makro pretprocesoru (znači, nije izvršna).

Obrazovanje proširenog tela asemblerskog programa, makro definicije i uslovne direktive opisuje pravilo:

```
telo -> {direktiva
        |osnovna_naredba
        |naredba_prebacivanja
        |upravljačka_naredba
        |makro_definicija
        |makro_poziv
        |uslovna_direktiva }
```

Pomoću uslovne direktive je moguće, na primer, opisati obrazovanje tabele, čiji elementi sadrže vrednosti prvih n stepena broja 2. Obrazovanje ovakve tabele opisuje naredna makro definicija koja sadrži rekurzivne makro pozive. Kraj rekurzije nastupa kada prestane važiti uslov uslovne direktive.

```
TABELA          MAKRO           stepen
                USLOVNO         stepen > 0
                TABELA          stepen/2
                NAPUNI          stepen
                KRAJ
                KRAJ
```

Makro poziv:

```
stepeni:        TABELA          8
```

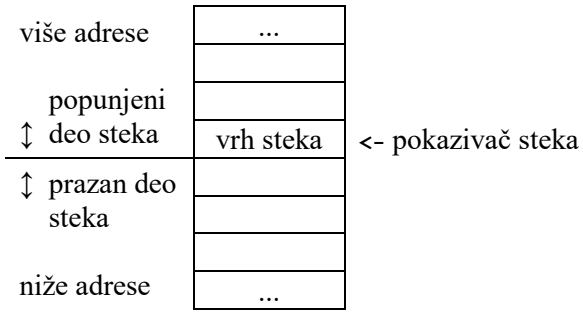
zamenjuju direktive:

```
stepeni:        NAPUNI          1
                NAPUNI          2
                NAPUNI          4
                NAPUNI          8
```

## 3.7. STEK

U naredbi `POZOVI` je predviđeno čuvanje samo jedne povratne adrese (jer samo toliko prostora ima u registru `%15`). Zato ova naredba direktno ne podržava poziv potprograma iz potprograma, ili rekurzivne pozive potprograma, jer se tada mora sačuvati više (unapred nepoznat broj) povratnih adresa. Povratne adrese se koriste u obrnutom redosledu od redosleda svog nastanka, jer nastaju u pozivu potprograma, a koriste se pri povratku iz potprograma. Znači, kao prva se koristi povratna adresa koja je poslednja nastala (pošto se prvi završava poslednje pozvani potprogram), a kao poslednja povratna adresa koja je prva nastala. Opisani redosled nastajanja i korišćenja povratnih adresa dozvoljava da se za njihovo privremeno čuvanje koristi niz memorijskih lokacija, u koji se povratne adrese smeštaju u jednom smeru, a iz koga se povratne adrese preuzimaju u drugom smeru (podrazumeva se da su adrese memorijskih lokacija iz ovog niza uzastopne i rastuće). Ako se povratne adrese smeštaju u pomenuti niz od njegovog kraja ka njegovom početku, a preuzimaju iz pomenutog niza u suprotnom smeru, počev od lokacije, do koje je niz bio popunjen, tada je za rukovanje nizom jedino važno znati adresu lokacije, do koje je niz popunjen. Od ove lokacije se nastavlja punjenje niza ka njegovom početku, odnosno, od nje se niz prazni ka njegovom kraju. Drugim rečima, sve napunjene lokacije niza su koncentrisane na njegovom kraju, a sve slobodne lokacije na njegovom početku. Ovako organizovan niz memorijskih lokacija se naziva **stek** (*stack*). Numerička adresa lokacije, do koje je stek popunjen, se obično čuva u registru, koji se naziva **pokazivač steka** (*stack pointer*).

Slika 3.7.1 sadrži prikaz izgleda steka.




Slika 3.7.1 Izgled steka

Rukovanje stekom obuhvata: zauzimanje memorijskih lokacija za stek, pripremu steka za korišćenje (odnosno, inicijalizaciju pokazivača steka), smeštanje sadržaja u stek i preuzimanje sadržaja iz steka. Opšti opis rukovanja stekom sadrži naredna makro definicija (za nju se podrazumeva da je registar `%1` radni, odnosno da se njegov sadržaj može slobodno menjati.):

```
STEK            MAKRO           veličina,pokazivač_steka
stek:           ZAUZMI          veličina
PRIPREMI_STEK   MAKRO
                PREBACI_NR      $stek,pokazivač_steka
                PREBACI_NR      $veličina,%1
                SABERI          %1,pokazivač_steka
                KRAJ
NA_STEK         MAKRO           registar
                ODBIJ_1         pokazivač_steka
                PREBACI_RP      registar,(pokazivač_steka)
                KRAJ
SA_STEKA        MAKRO           registar
                PREBACI_PR      (pokazivač_steka),registar
                DODAJ_1         pokazivač_steka
                KRAJ
                KRAJ
```

Jedina dva parametra prethodne makro definicije služe za određivanje: veličine steka i oznake registra pokazivača steka. Oni omogućuju potpuno oblikovanje makro definicija, sadržanih u makro definiciji `STEK`. Makro definicija `STEK` je napravljena uz pretpostavku da je broj lokacija steka uvek veći od broja sadržaja koji se smeštaju na stek. Takođe, podrazumeva se da se stek ispravno koristi, odnosno da se sa steka preuzimaju samo u njega smešteni sadržaji.

Makro poziv:

```
                STEK            0x100,%12
```

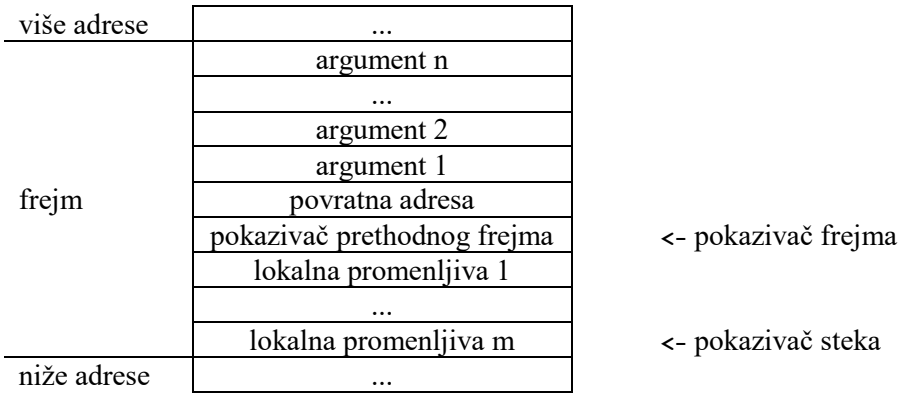
zamenjuju sledeća direktiva i makro definicije:

```
stek:           ZAUZMI          0x100
PRIPREMI_STEK   MAKRO
                PREBACI_NR      $stek,%12
                PREBACI_NR      $0x100,%1
                SABERI          %1,%12
                KRAJ
NA_STEK         MAKRO           registar
                ODBIJ_1         %12
                PREBACI_RP      registar,(%12)
                KRAJ
SA_STEKA        MAKRO           registar
                PREBACI_PR      (%12),registar
                DODAJ_1         %12
                KRAJ
```

koje dovode do zauzimanja 256 memorijskih lokacija ($100_{16}$) za stek, do određivanja registra `%12` kao pokazivača steka i do oblikovanja makro definicija `PRIPREMI_STEK`, `NA_STEK` i `SA_STEKA` (korišćenje prve od njih obavezno prethodi korišćenju poslednje dve).

### FREJM

Stek se koristi, ne samo za čuvanje povratnih adresa, nego i za prenos argumenata u potprogram. Skup lokacija na steku, koje sadrže povratnu adresu i argumente se naziva **frejm** (*stack frame*). Frejm nastaje prilikom poziva potprograma, a nestaje po povratku iz potprograma. Lokacijama frejma se može pristupati posredstvom pokazivača steka, ali tada probleme stvaraju izmene pokazivača steka (koje stalno menjaju udaljenost frejma od vrha steka), kao i istovremeno postojanje više frejmova na steku. Zato je bolje koristiti poseban registar za pristup lokacijama frejma. Takav registar se naziva **pokazivač frejma** (*base pointer*). On pokazuje na baznu lokaciju ili **bazu frejma**. Da bi se za svaki od istovremeno postojećih frejmova mogao odrediti njegov pokazivač, zgodno je uvezati frejmove u listu. To se postiže ako baza najmlađeg frejma sadrži pokazivač prethodnog frejma i tako dalje. Podrazumeva se da aktivni pokazivač frejma pokazuje na bazu najmlađeg frejma. Pored povratne adrese, argumenata i pokazivača prethodnog frejma, frejm može da sadrži i lokalne (dinamičke) promenljive. Slika 3.7.2 sadrži prikaz mogućeg izgleda ovakvog frejma.




Slika 3.7.2 Izgled frejma

Za pristup lokacijama frejma (Slika 3.7.2) zgodno je indeksno adresiranje, koje koristi pokazivač frejma i relativnu udaljnost lokacije frejma od njegove baze. Ako registar `%11` sadrži pokazivač frejma, tada poređenje argumenta 1 i argumenta 2 (Slika 3.7.2) opisuju naredbe:

```
PREBACI_IR     2(%11),%0
PREBACI_IR     3(%11),%1
UPOREDI        %1,%0
```

U prethodnom, a i u sledećem primeru se podrazumeva da su registri `%0` i `%1` radni, odnosno da se njihovi sadržaji mogu slobodno menjati.

Rukovanje frejmom je vezano za poziv potprograma. Ono se može objasniti na primeru za poziv procedure `nzd`, (izražene programskim jezikom C):

```
nzd(x,y,&z);
```

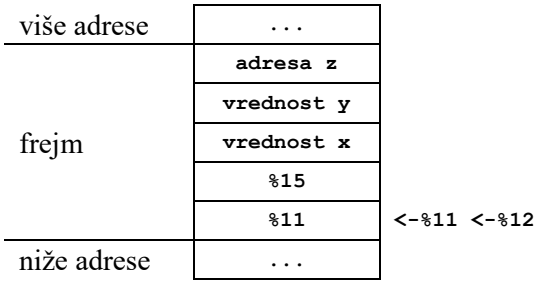
Rukovanje frejmom za prethodni poziv opisuje asemblerska sekvenca:

```
x:             ZAUZMI         1
y:             ZAUZMI         1
z:             ZAUZMI         1
               ...
               PREBACI_NR     $z,%0
               NA_STEK        %0
               PREBACI_DR     y,%0
               NA_STEK        %0
               PREBACI_DR     x,%0
               NA_STEK        %0
               NA_STEK        %15
               POZOVI         nzd
               SA_STEKA       %15
               PREBACI_NR     $3,%0
               SABERI         %0,%12
```

Podrazumeva se da registar `%12` sadrži pokazivač steka, da registar `%11` sadrži pokazivač frejma i da pozvani asemblerski ekvivalent procedure `nzd` izgleda:

```
nzd:        NA_STEK       %11
            PREBACI_RR    %12,%11
            PREBACI_IR    2(%11),%0
            PREBACI_IR    3(%11),%1
ponovo:     UPOREDI       %1,%0
            SKOČI_ZA_==   kraj
            SKOČI_ZA_<    manje
veće:       ODUZMI        %1,%0
            SKOČI         ponovo
manje:      ODUZMI        %0,%1
            SKOČI         ponovo
kraj:       PREBACI_IR    4(%11),%1
            PREBACI_RP    %0,(%1)
            SA_STEKA      %11
            NATRAG
```

Slika 3.7.3 sadrži prikaz stanja na steku, neposredno pre izvršavanja naredbe `UPOREDI`.




Slika 3.7.3 Primer frejma

Gornje (prve) 4 lokacije frejma (Slika 3.7.3) nastanu izvršavanjem naredbi koje prethode naredbi `POZOVI` iz poziva potprograma `nzd`, a preostala lokacija nastane izvršavanjem prve naredbe iz ovoga potprograma. Izvršavanje pretposlednje naredbe ovoga potprograma poništi zadnju lokaciju frejma, a izvršavanja poslednje tri naredbe iz poziva ovoga potprograma ponište prve 4 lokacije frejma.

Korišćenje steka može da uzrokuje **dinamičke greške** u izvršavanju programa (na primer, ako se ispostavi da, za neko izvršavanje programa, broj lokacija steka nije dovoljan za smeštanje svih sadržaja).

U arhitekturi naredbi procesora KONCEPT nisu predviđene posebne naredbe, a ni posebna adresiranja za rukovanje stekom, da bi ovaj procesor bio što jednostavniji. Kada bi procesor KONCEPT podržavao stek, tada bi u njegovom repertoaru naredbi trebalo da postoje naredbe koje su ekvivalentne makroima

`NA_STEK` i `SA_STEKA`. U tom slučaju bi naredbe `POZOVI` i `NATRAG` trebalo da koriste stek umesto registra `%15`.

## 3.8. PITANJA

1. Koji programski jezici niskog i visokog nivoa postoje?
2. U čemu se razlikuju asemblerski i mašinski jezici?
3. Kada je opravdano korišćenje asemblerskog i mašinskog jezika?
4. Koliki adresni prostor omogućuju 2 bita?
5. Koje naredbe omogućuju aritmetiku u višestrukoj preciznosti?
6. Koje naredbe u toku svog izvršavanja ne menjaju uslovne bite?
7. Koje aritmetičke naredbe postoje?
8. Koje naredbe za rukovanje bitima postoje?
9. Koje upravljačke naredbe postoje?
10. Koje kombinacije operanada koriste naredbe prebacivanja?
11. Koje naredbe omogućuju otkrivanje izlaska van opsega?
12. Koje naredbe imaju dva operanda?
13. Koje naredbe imaju jedan operand?
14. Koje naredbe su bez operanada?
15. Koje asemblerske direktive postoje?
16. Šta karakteriše potprogram?
17. Šta karakteriše makro?
18. Šta karakteriše funkciju?
19. Šta karakteriše proceduru?
20. Koje asemblerske naredbe su uvedne radi asemblerskih potprograma?
21. Šta je vezano za poziv asemblerskog potprograma?
22. Šta je vezano za makro poziv?
23. Kako upotreba potprograma utiče na dužinu programa i vreme njegovog izvršavanja?
24. Kako upotreba makroa utiče na dužinu programa i vreme njegovog izvršavanja?
25. Ko obavlja zamenu makro poziva modifikovanim naredbama iz makro definicija?
26. Gde su definisane i kada postoje globalne promenljive?
27. Gde su definisane i kada postoje lokalne promenljive?
28. Šta se smešta na stek?
29. Kako se rukuje stekom?
30. Šta sadrži frejm?