6 Mašinski kod

U Glavi 1 uveden je osnovni koncept prema kojem glavna memorija sadrži i programske instrukcije i podatke. U ovoj glavi biće razmatrana struktura instrukcija smještenih u memoriji.

Većina instrukcija odnosi se na operacije nad podacima koji se nalaze ili u glavnoj memoriji ili u registrima CPU-a. Podaci na koje se instrukcije odnose nazivaju se i operandima.

6.1 Format instrukcije

Svaki program se sastoji od niza funkcionalno nezavisnih koraka - instrukcija. Instrukcije obavljaju različite funkcije koje mogu biti grupisane na sljedeći način:

- Transfer podataka između glavne memorije i CPU registara
- Aritmetičke i logičke operacije sa podacima
- Upravljanje tokom izvršavanja programa
- Ulazno/izlazni (I/O) transfer podataka

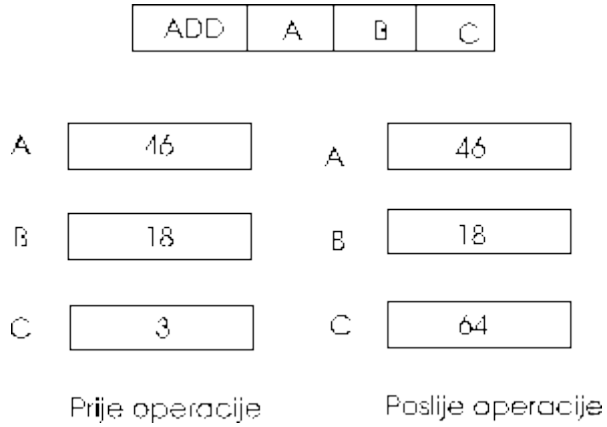
U kojem obliku (formatu) se instrukcije smještaju u memoriju pokazaćemo na primjeru instrukcije za sabiranje iz druge od gore navedenih klasa.

Sa programerskog aspekta najjednostavniji oblik instrukcije za sabiranje bio bi C:=A+B, gdje su A,B,C imena varijabli. Pretpostavimo da su vrijednosti ovih varijabli smještene u memorijske lokacije čije adrese su označene sa A,B,C.

Izraz C:=A+B ima sljedece značenje: Vrijednosti podataka iz lokacija A i B treba da bude doveden u CPU gdje će uz pomoc aritmetičko logičke jedinice (ALU) biti sabrane, a rezultat sabiranja biće smješten u lokaciju C.

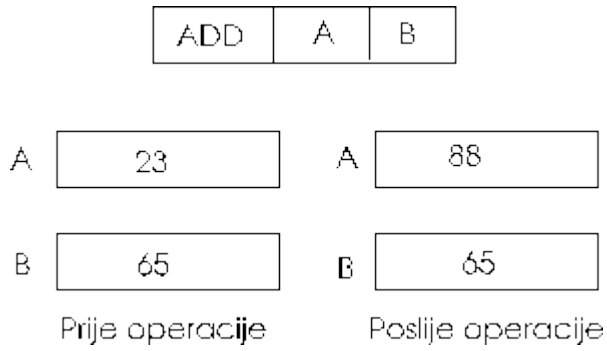
Ako cio gornji izraz treba da bude predstavljen jednom mašinskom instrukcijom, mašinska instrukcija mora imati tri adresna dijela za operande A,B,C.

Tro-adresna instrukcija ADD A,B,C će zahtijevati veliki broj bitova za smještaj sve tri adrese. Slika 6.1 prikazuje sadržaj adresa A,B,C prije i poslije operacije ADD.



Slika 6.1 Troadresna ADD instrukcija

U cilju uštede na dužini instrukcije može se koristiti dvo-adresna instrukcija sa dva adresna dijela (polja): ADD A,B sa sljedećim značenjem: Vrijednosti podataka iz A i B treba dovesti u CPU, sabrati ih i rezultat smjestiti u lokaciju A. Slika 6.2 prikazuje navedeni proces sabiranja u dvo-adresnoj instrukciji.



Slika 6.2 Dvoadresna ADD instrukcija

Naravno, u dvo-adresnoj instrukciji originalna vrijednost iz lokacije A je uništena operacijom sabiranja, što može predstavljati problem. Ako želimo da vrijednost u lokaciji A bude sačuvana moramo za sabiranje koristiti dvije instrukcije kako slijedi:

```
MOVE C,B
ADD C,A
```

gdje instrukcija MOVE C,B ima sljedeće značenje: Kopirati sadržaj lokacije B u lokaciju C.

Predhodno sabiranje se može izvršiti i sa jedno-adresnom instrukcijom. U tom slučaju ćemo koristiti tri instrukcije:

```
LOAD A
ADD B
STORE C
```

Instrukcija LOAD A ima značenje: Kopiraj podatak iz lokacije A u CPU registar, koji se ponekad zove akumulator. Instrukcija STORE C ima značenje: Kopiraj sadržaj akumulatora u memorijsku lokaciju C.

6.2 Skup instrukcija

Sve mašinske instrukcije koje neki računar može da izvršava nazivaju se skupom instrukcija tog računara. Sve instrukcije se mogu svrstati u jednu od sljedeće tri kategorije:

Memorijske - sa operacijama nad podacima u glavnoj memoriji.

Ne-memorijske - sa operacijama bez podataka ili sa podacima u registrima CPU-a.

Ulazno/Izlazne - za operacije sa perifernim jedinicama.

6.2.1 Memorijski orjentisane instrukcije

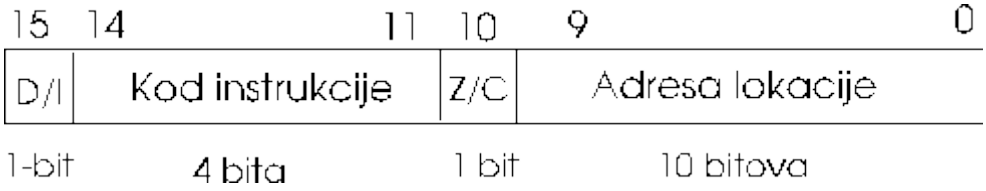
Ova klasa instrukcija obuhvata sve instrukcije koje se odnose bar na jednu memorijsku lokaciju, te takva instrukcija mora sadržati adresno polje.

Nije moguće dati sve instrukcije koje su prisutne u raznim račcunarima, pošto svaki od proizvodjača definiše svoj skup instrukcija u skladu sa arhitekturom računara i onim što proizvodjač misli da je korisno da stavi na raspolaganje programerima. Ovdje će biti nabrojane podklase memorijskih instrukacija koje se nalaze kod većine danas raspoloživih računara:

- **Napuni neki od registara sa sadržajem neke memorijske lokacije.**
- Smjesti sadržaj registra u neku memorijsku lokaciju.
- Saberi sadržaj memorijske lokacije sa sadržajem registra.
- Oduzmi sadržaj memorijske lokacije od sadržaja registra.

- Uporedi sadržaj registra sa sadržajem memorijske lokacije i preskoči sljedeću instrukciju ako nijesu jednaki.
- Povećaj sadržaj memorijske lokacije za 1 i preskoči sljedeću instrukciju ako je rezultat 0.
- Kombinuj sadržaj memorijske lokacije sa sadržajem registra korišćenjem logičkih operacija (AND, OR i sl.).
- Operacije grananja - bezuslovni skok, ulosvni skok, i skok u potprogram.
- Instrukcija pomjeranja bitova (šift) - instrukcije kojima se pomjeraju svi bitovi u memorijskoj lokaciji ulijevo ili udesno.

Da bi detaljnije razmotrili format instrukcije posmatrajmo jednoadresni racunar koji ima dva registra opšte namjene (zvaćemo ih A i B registar), i neka računar ima memorijske lokacije dužine 16 bita. Tipična instrukcija je recimo "Napuni registar A" (napuni registar A sa sadržajem iz neke memorijske lokacije), ili "Dodaj na A" (dodaj sadržaj neke memorijske lokacije na sadržaj registra A). Format takvih instrukcija prikazan je na Slici 6.3.



Slika 6.3 Format instrukcije

Kod jednoadresnih računara osnovni sadržaj instrukcije je kod operacije koju instrukcija vrši i adresa memorijske lokacije koja sadrži podatak potreban operaciji. Na Slici 6.3 za kod operacije je rezervisano 4 bita a za memorijsku adresu 10 bitova. Uloga bitova broj 10 i 15 biće razmotrena nešto kasnije.

Ako je adresa memorijske lokacije data sa 10 bitova to znači da je broj memorijskih lokacija koje se mogu adresirati 2^{10} = 1024. Obično računari imaju znatno veću memoriju od 1024 16-bitne riječi. Ako računar ima, recimo, 1 MB memorije za adresiranje je potrebno 20 bitova, a to s druge strane zahtijeva i da instrukcija bude znatno duža. Da bi se prevazišao ovaj problem (dužine instrukcije) koriste se razne metode adresiranja, koje će biti prikazane nešto kasnije.

6.2.2 Grananje (Branching)

Neke memorijski orjentisane instrukcije ne koriste podatke u memoriji, već služe kao "putokaz" za redosljed izvršavanja instrukcija. Takve instrukcije nazivaju se instrukcijama grananja

programa (branching).

Instrukcije se u glavnoj memoriji nalaze jedna iza druge u nizu kako to zahtijeva algoritam. One se za vrijeme izvršavanja programa, po pravilu, i izvršavaju jedna za drugom u istom redosljedu. Međutim ponekad se pojavljuje potreba da se u zavisnosti od rezultata pojedinih instrukcija mijenja "normalan" tok izvršavanja instrukcija.

Posmatrajmo sljedeći jednostavan primjer:

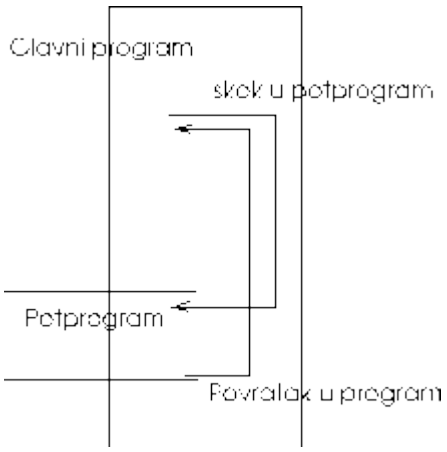
```
        Napuni registar A sa vrijednošću n
CIKLUS:
            instrukcija 1
          instrukcija 2
            .
            .

            Umanji registar A za 1.
            Skoči na CIKLUS ako je A > 0.
```

Posljednjom instrukcijom smo zapravo postigli da se instrukcije iz ciklusa ponavljaju sve dok vrijednost u registru A ne bude jednaka 0, a zatim se program nastavlja sljedećom naredbom koja slijedi iza naredbe "Skoči".

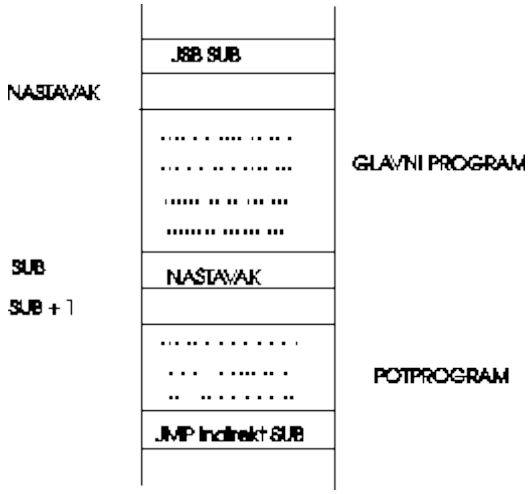
6.2.3 Potprogrami

Specijalan oblik instrukcije grananja je tzv. skok u potprogram. Pod potprogramom podrazumijevamo niz instrukcija koje obavljaju neku opštu funkciju koja se tokom izvršavanja programa moze zahtijevati više puta i iz različitih djelova programa. Posebnost skoka u potprogram se ogleda u tome što se pri grananju mora zapamtiti adresa mjesta u programu odakle je "poziv" (skok) na potrpogram učinjen, kako bi se nakon završetka rada potprograma "vratili" na pravo mjesto u programu. Slika 6.4 prikazuje raspored programa i potprograma u memoriji.



Slika 6.4 [ema izvršavanja potprograma

Kao primjer instrukacija koje omogućavaju pozivanje i vraćanje iz potprograma uzećemo instrukcije JSB (jump to subroutine) i RETURN (povratak) čiji je rad ilustrovan na slici 6.5.



Slika 6.5 Primjer izvršabanja potprograma

Instrukcija JSB smješta u memorijsku lokaciju koja se nalazi na prvoj lokaciji potprograma povratnu adresu (adresu prve instrukcije iza instrukcije JSB). Posljednja instrukcija potrograma je instrukcija indirektog skoka (JMP indirect) kojom se vraćamo na sljedeću instrukciju iz programa.

6.2.4 Ne-memorijske instrukcije

Ne-memorijske instrukcije su one koje ne zahtijevaju podatak iz memorijske lokacije. Tipičan slučaj su instrukcije koje koriste registre kao operande. Kako računari obično sadrže mali broj

registara, za njihovo adresiranje je potrebno svega nekoliko bitova, pa su nememorijske instrukcije znatno kraće od memorijskih. Karakterističan skup ne-memorijskih instrukcija je:

(a) Registar - registar aritemetičke i logičke funkcije kao što su ADD, SUBTRACT, MOVE, AND, OR XOR (ekskluzivno OR).

(b) Šift instrukcije kojima se sadržaj registara pomera ulijevo ili udesno za jednu ili više pozicija.

6.3 Metode adresiranja memorije

Kako je već rečeno zbog uštede u dužini instrukcija potrebno je imati više načina adresiranja memorije. Zato se uvodi pojam efektivne adrese koja je zapravo stvarna adresa u memoriji koja se dobija posle različitih transformacija pri raznim metodama adresiranja.

Ovdje će biti prikazane najčesce korišćene metode adresiranja.

6.3.1 Direktne (apsolutne) adrese

Efektivna adresa lokacije data je u samoj instrukciji. Broj bitova u adresnom polju ograničava memorijski prostor koji može biti adresiran na ovaj način. Ako je broj bitova u adresnom polju instrukcije N tada je maksimalna memorija 2^N lokacija.

6.3.2 Indirektne adrese

Efektivna adresa se dobija tako što se u instrukciji nalazi adresa memorijske lokacije (ili registra) koja sadrži adresu operanda instrukcije. Svrha bita 15 u formatu instrukcije prikazanom u poglavlju 6.2.2 je da označi način adresiranja (Direktno ili Indirektno). Ako je bit 15 postavljen na nulu koristiće se direktno adresiranje, a ako je postavljen na 1 indirektno.

Prednost indirektnog adresiranja je u tome što se čitava memorijska lokacija koristi za adresu (a ne njen dio kao u instrukciji), pa se samim tim može adresirati znatno veća memorija.

6.3.3 Neposredno adresiranje

Kod neposrednog adresiranja umjesto adrese operanda u instrukciji se nalazi sama njegova vrijednost. To je pogodno u slučaju fiksnih podataka (konstanti) koje se ne mijenjaju za vrijeme izvršavanja programa. Vrijednost podatka je, naravno, limitirana brojem bitova unutar instrukcije.

6.3.4 Indeksno adresiranje

Efektivna adresa operanda se izračunava sabiranjem neke vrijednosti (indeksa) sa adresom datom u instrukciji. Vrijednost indeksa je obično smještena u posebnom registru CPU tzv. indeksnom registru. Ovakva vrsta adresiranja je posebno pogodna u slučajevima kada se instrukcija ponavlja za više operanada koji se nalaze jedan za drugim u memoriji (recimo sabiranje niza brojeva).

6.4 Metode adresiranja kod Intel 286/386/486 računara

Osnovni način adresiranja kod Intelovih procesora 286/386/486 je korišćenje koncepta virtuelne memorije. Virtuelni mehanizam se zasniva na korišćenju tzv. baznog registra kojim se adresira segment virtuelne memorije (vidi poglavlje 5.4) i relativne adrese unutar segmenta (displacement). Procesor 286 ima 24-bitni adresni bus pa se njime može adresirati 1MB realne memorije, ali se mehanizmom virtuelne memorije dobija memorijski prostor od 1GB.

Procesori 386/486 imaju 32-tni adresni bus, a takodje i 32-bitne registre (za razliku od 16-bitnih registara kod 286). Na taj način realna memorija koja se moze adresirati je 4GB (2^{32} bajtova) , a virtuelni mehanizam omogućava adresiranje izuzetno velike memorije do 64 tera bajta (2^{46} bajtova).

Instrukcije 286/386/486 računara obuhvataju sljedeće metode adresiranja:

Direktno adresiranje	32-bitna relativna adresa (displacment)
Indirektno adresiranje	efektivna adresa je zadata u jednom od registara
Neposredno adresiranje	vrijednost operanda je u samoj instrukciji
Registarsko adresiranje	vrijednost operanda je u registru
Indeksno adresiranje	vrijednost registra se dodaje na adresu iz instrukcije
Skalirano adresiranje	indeksni registar se množi sa konstantom
Bazno adresiranje	sadržaj registra se kombinuje sa pomakom u instrukciji

Indeksno-bazno sabira se indeksni i bazni registar.

Pored gore navedenih moguće su i kombinacije: skalirano indeksno-bazno adresiranje, indeksno-bazno uz korišćenje pomaka, i skalirano indeksno-bazno sa pomakom.

6.5 RISC (Reduced Instruction Set Computers) računari

Iako je jasno da računar koji bi imao samo nekoliko instrukcija ne može biti od posebnog praktičnog značaja, nije jasno koji je to skup instrukcija koji bi na najbolji način pokrio sve potrebe za programiranje računara opšte namjene. Zato ni među proizvođačima računara nema konsenzusa o skupu instrukcija, pa se u različitim računarima pojavljuju manje ili veće rzlike. Kako se razvijala tehnologija izrade računara tako se sve više i više širio skup instrukcija, a posebno načini adresiranja. To je dovelo do povećanja kompleknosti CPU-a i njihove cijene, a i do problema programiranja tako složenih računara (posebno pri programiranju kompajlera). Zato su neki projektanti racunara dosli na ideju da projektuju računare sa relativno malim brojem instrukcija - RISC arhitekture.

Glavne karakteristike RISC računara su:

- relativno mali broj tipova instrukcija i metoda adresiranja
- instrukcije su optimizovane za zadati programski jezik i njegov kompajler
- fiksan i jednostavan za dekodiranje format instrukcije
- hardverska a ne mikroprogramska kontrolna jedinica
- pristup memoriji ograničen samo na instrukcije LOAD i STORE

U RISC arhitekturama efikasnost se postiže ne samo zbog jednostavnosti instrukcija već i zbog toga što je smanjena komunikacija CPU memorija korišćenjem većeg broja registara nego kod standardnih arhitektura.

Tipičan primjer RISC računara su SPARC računari firme SUN. SPARC procesor se satoji od posebnih jedinica za cjelobrojnu (IU) i decimalnu (floating point FPU) aritmetiku. Registri su duzine 32 bita sa 32-bitni virtuelnim adresnim prostorom. IU sadrzi od 40 do 520 registara, a FPU 32 registra. SPARC ima 55 instrukcija za rad sa cijelim brojevima i 14 instrukcija za decimalne brojeve. Samo se dvije instrukcije koriste za komunikaciju sa memorijom (LOAD i STORE).