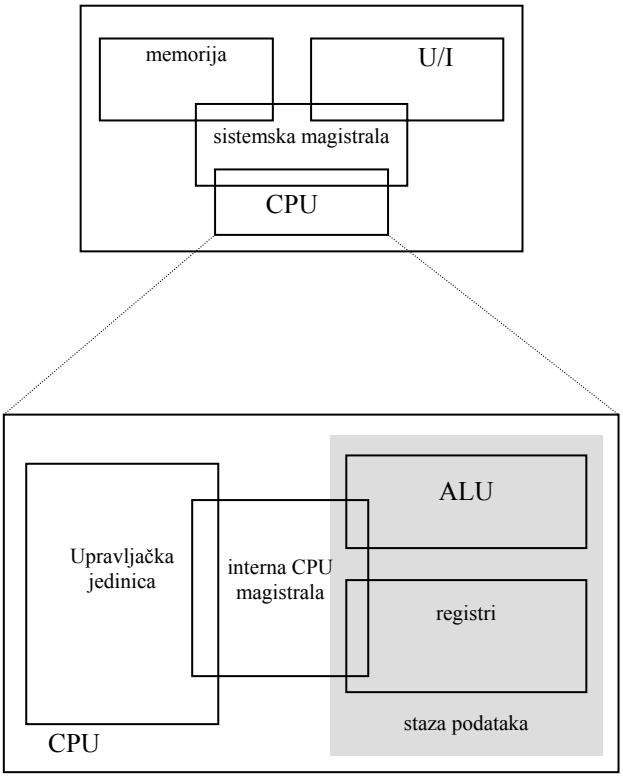
2. STRUKTURA CPU-a

Najinteresantnija i svakako najkompleksnija komponenta računara je CPU. Struktura CPU-a prikazana je na slici 2.1.



Sl. 2.1. Struktura CPU-a.

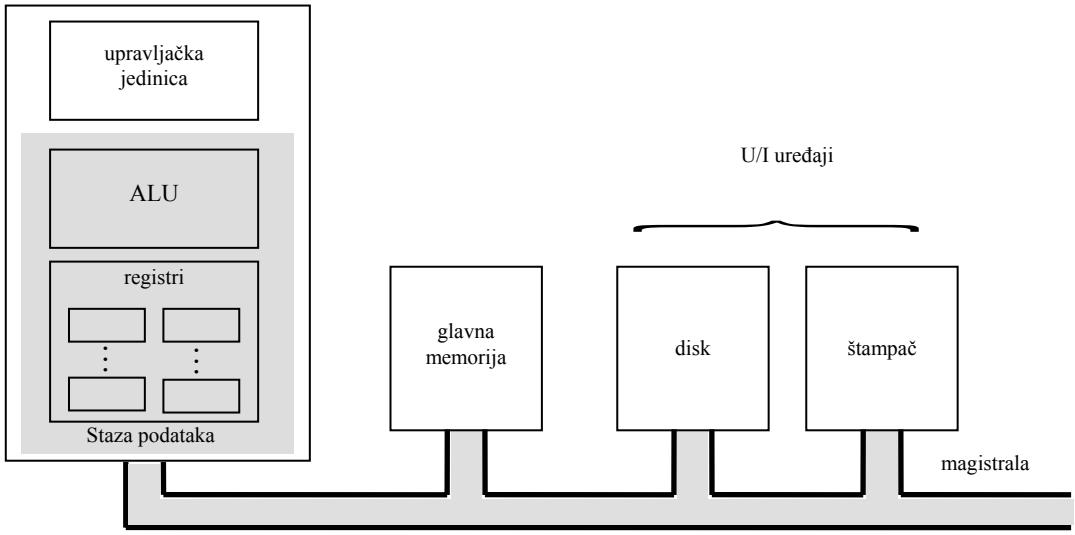
Glavne strukturne komponente CPU-a su:

- **Upravljačka jedinica** - upravlja radom CPU-a a shodno tome i radom računara.
- **Staza podatka** koja obuhvata
  - **Aritmetičko logičku jedinicu (ALU)** - obavlja obradu podataka, tj. izvršava različite aritmetičke i logičke funkcije.
  - **Registre** - obezbeđuju memorijski medijum unutar CPU-a.
- **Interna CPU magistrala** - obezbeđuje mehanizam preko koga se ostvaruje komunikacija između upravljačke jedinice, ALU i registara.

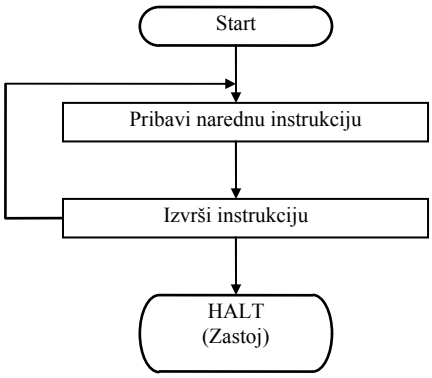
2.1. Princip rada CPU-a

U svom najprostijem obliku računarski sistem prikazan na slici 2.2, ima jednu jedinicu koja izvršava programske instrukcije. Ova jedinica komunicira i upravlja radom ostalih podsistema u sklopu računara. Zbog svoje centralne uloge ovu jedinicu zovemo CPU.

Kao što je prikazano na slici 2.3, CPU obavlja dve glavne funkcije: pribavljanje i izvršenje instrukcija.



Sl. 2.2. Organizacija jednostavnog računara sa jednim CPU-om i dva U/I uređaja.



Sl. 2.3. Osnovni ciklusi instrukcija.

Program čini skup instrukcija koje su smeštene u memoriji. CPU tipično čita (pribavlja) instrukcije iz memorije po jednu u datom trenutku, izvršava svaku instrukciju, a zatim pribavlja narednu. Proces se ponavlja beskonačno dugo. Ova sekvenca je poznata kao ciklus *pribavljanje-dekodiranje-izvršenje*. Obrada koja se zahteva od strane jedne instrukcije zove se ciklus instrukcije (slika 2.3). Kao što se vidi na slici 2.3, svaki ciklus instrukcije čine dva dela koje zovemo *ciklus pribavljanja* i *ciklus izvršenja*. Proces se zaustavlja samo kada se mašina isključi, javi neki tip greške, ili se izvrši programska instrukcija tipa HALT.

2.1.1. Aktivnosti CPU-a

Pribavljena instrukcija je u obliku binarnog koda pomoću koga se specificira koja se akcija preuzima od strane CPU-a. CPU interpretira instrukciju i obavlja željenu akciju. U opštem slučaju akcije CPU-a svrstavamo u jednu od sledećih kategorija:

- **CPU-Memorija** - prenos podataka je tipa CPU→memorija ili memorija←CPU (tj. CPU↔memorija).
- **CPU-U/I** - podaci se prenose ka/iz spoljnjeg sveta prenosom tipa CPU↔U/I moduli.
- **Obrada podataka** - CPU može obavljati neku aritmetičku ili logičku operaciju nad podatkom.
- **Upravljanje (promena toka programskog izvršenja)** - Instrukcijom se može specificirati promena redosleda izvršenja instrukcija (npr. GO TO, JMP itd.). Recimo, CPU može da pribavi instrukciju sa lokacije 214 kojom se specificira da će se kao naredna izvršiti instrukcija sa lokacije 288, a ne sa 215.

2.1.2. Uloga PC-a, MAR-a, MDR-a i IR-a.

Kao što smo već naglasili, instrukcije se pribavljaju iz susednih memorijskih lokacija sve dok se ne naiđe na izvršenje instrukcije grananja (*branch*) ili skoka (*jump*). CPU čuva trag o adresi memorijske lokacije gde je

locirana naredna instrukcija preko namenskog registra CPU-a poznatog kao *programski brojač* (PC). Nakon pribavljanja instrukcije, sadržaj PC-a se ažurira sa ciljem da ukaže na instrukciju koja će se u sekvenci kao naredna izvršiti. Takođe važan registar je *registar naredbi* (IR), u kome se čuva instrukcija koja se trenutno izvršava.

Primer 2.1:

Radi pojednostavljenja usvojimo da svaka instrukcija zauzima jednu memorijsku reč. Koraci koje CPU obavlja u toku izvršenja jedne instrukcije su:

1. Pribavlja sadržaj memorijske lokacije na koju ukazuje PC. Sadržaj ove instrukcije se interpretira kao instrukcija koja treba da se izvrši. Instrukcije se smeštaju u IR, a to se simbolički piše kao

$$IR \leftarrow [[PC]].$$

2. Sadržaj PC-a se povećava za jedan

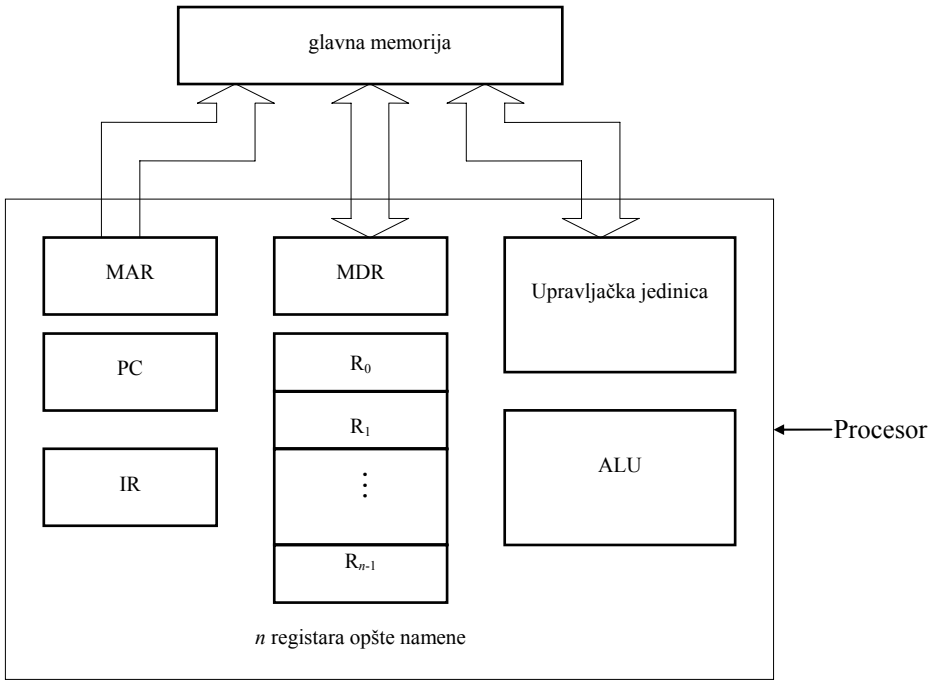
$$PC \leftarrow [PC]+1.$$

3. Izvršava se akcija specificirana od strane instrukcije koja je smeštena u IR.

Koraci 1. i 2. čine fazu pribavljanja, a korak 3. fazu izvršenja.

Pre nego što se temeljitije upustimo u analizu rada CPU-a, neophodno je malo detaljnije izučiti njegovu strukturu. U tom cilju sagledajmo najpre kako se CPU povezuje sa glavnom memorijom (slika 2.4). Pored IR-a i PC-a, u CPU se ugrađuje i nekoliko *registara opšte namene* koji se koriste za čuvanje podataka i adresa. Dva registra olakšavaju komunikaciju sa glavnom memorijom, a to su *adresni registar memorije* (MAR) i *registar za čuvanje podataka iz memorije* (MDR).

Kao što i samo ime ukazuje, MAR se koristi za čuvanje adrese lokacije ka ili iz koje se prenosi podatak, a u MDR-u se čuva podatak koga treba upisati u, ili pročitati iz adresirane lokacije.

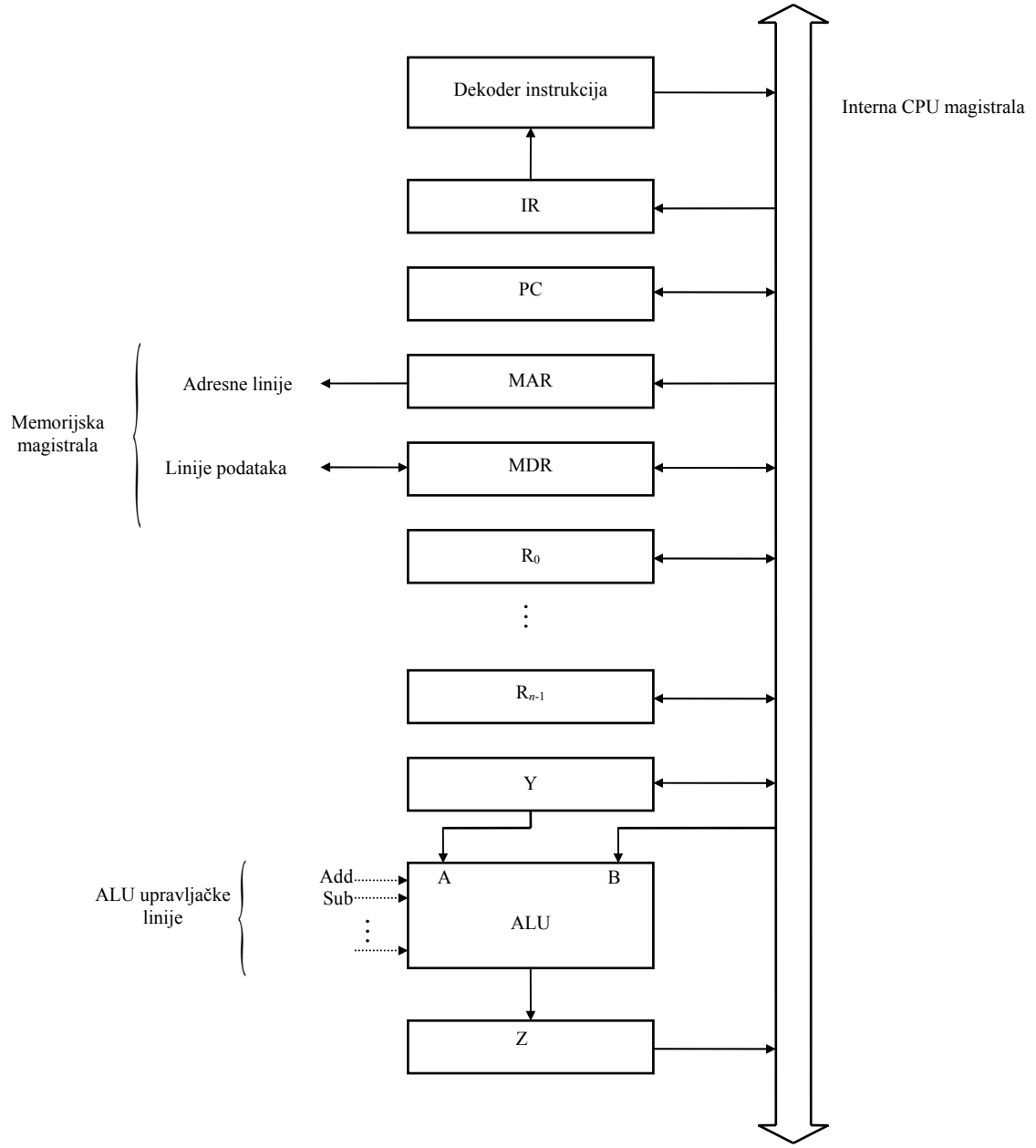


Sl. 2.4. Veza između CPU-a i glavne memorije.

2.2. Interno povezivanje blokova CPU-a

Blokovi CPU-a prikazani na slici 2.4 mogu se organizovati i međusobno povezati na različite načine. Jedna takva organizacija je prikazana na slici 2.5. U ovom slučaju ALU i svi interni registri su povezani preko jedinstvene zajedničke magistrale. Magistrala je naravno interna i ne sme se poistovetiti sa spoljnom magistralom (slika 2.2) preko koje se CPU povezuje sa memorijom i U/I uređajima. Broj i funkcija internih registra R_0 do R_{n-1} menja se od

mašine do mašine. U najvećem broju slučajeva koriste se od strane programera kao registri opšte namene, a neki od njih mogu biti određeni za specijalnu namenu kao što su indeksni registri ili pokazivači magacina.



Sl. 2.5. Interni putevi podataka CPU-a organizovanog oko jedinstvene magistrale.

Registri Y i Z su obično transparentni programeru. Drugim rečima programer ne treba da vodi računa o njihovom postojanju, pošto se njima nikad direktno ne obraća bilo kojom instrukcijom. Oni se unutar CPU-a koriste za interno smeštanje (čuvanje) podataka u toku izvršenja neke instrukcije. Registri Y i Z se nikad direktno ne koriste tako da jedna instrukcija smešta podatke a druga instrukcija ih kasnije koristi.

2.2.1. Elementarne funkcije CPU-a.

Najveći broj operacija koje izvršava CPU može se obaviti jednom od sledećih elementarnih funkcija po unapred specificiranoj sekvenci:

- **Pribavljanje reči iz memorije**

Neka se adresa memorijske lokacije kojoj se pristupa nalazi u R1, a podatak iz memorije smešta u R2. Da bi se ovo realizovalo potrebno je izvršiti sledeću sekvencu operacija:

1. MAR ← [R1].
2. Čitanje.
3. Čekanje na MFC (*Memory Function Completed*) signal.
4. R2 ← [MDR].

Mehanizam prenosa kod koga jedan uređaj inicira prenos (*Read request*) i čeka sve dok se drugi uređaj ne odazove (MFC signal) poznat je kao *asinhroni prenos*. Ovaj mehanizam omogućava prenos podataka između dva nezavisna uređaja koji imaju različite brzine rada. Alternativna šema koja se sreće kod velikog broja računara koristi *sinhroni prenos*. U ovom slučaju, preko jedne od upravljačkih linija magistrale se prenose taktni impulsi fiksne frekvencije. Ovim impulsima se obezbeđuje zajednički sinhronizacioni signal za CPU i memoriju. Memorijska operacija se mora završiti u toku jednog taktnog intervala. Sinhrona šema se, u principu, može svuda implementirati, ali se njom ne može prilagoditi rad uređaja čija je brzina rada različita, sa izuzetkom ako se brzina rada sistema prilagodi najsporijem uređaju.

- **Smeštanje reči u memoriju**

Ako je u R1 adresa memorijske lokacije a u R2 podatak koji treba da se smeti u tu lokaciju, onda procedura za upis reči u memoriju ima sledeći oblik

1. MAR ← [R1]
2. MDR ← [R2], Upis
3. Čekaj na MFC

- **Registarski prenos**

Da bi se obezbedio prenos podataka između registara povezanih na zajedničku magistralu (slika 2.5), ulazi i izlazi registara se moraju kontrolisati (gejtovati), kao što je prikazano na slici 2.6. Ulazni i izlazni stepeni (gejtovi) registra R*i* se upravljaju signalima Ri_{in} i Ri_{out} respektivno. Kada je Ri_{in}=1, podatak koji je dostupan na zajedničkoj magistrali se upisuje (puni) u R*i*. Na sličan način, kada je Ri_{out}=1, sadržaj registra R*i* se postavlja na magistralu. Kada je Ri_{out}=0, magistrala se može koristiti za prenos podataka iz drugih registara.

Analizirajmo slučaj kada se vrši prenos sadržaja registra R1 u registar R4. Da bi se ova aktivnost obavila potrebno je da se obave sledeće akcije:

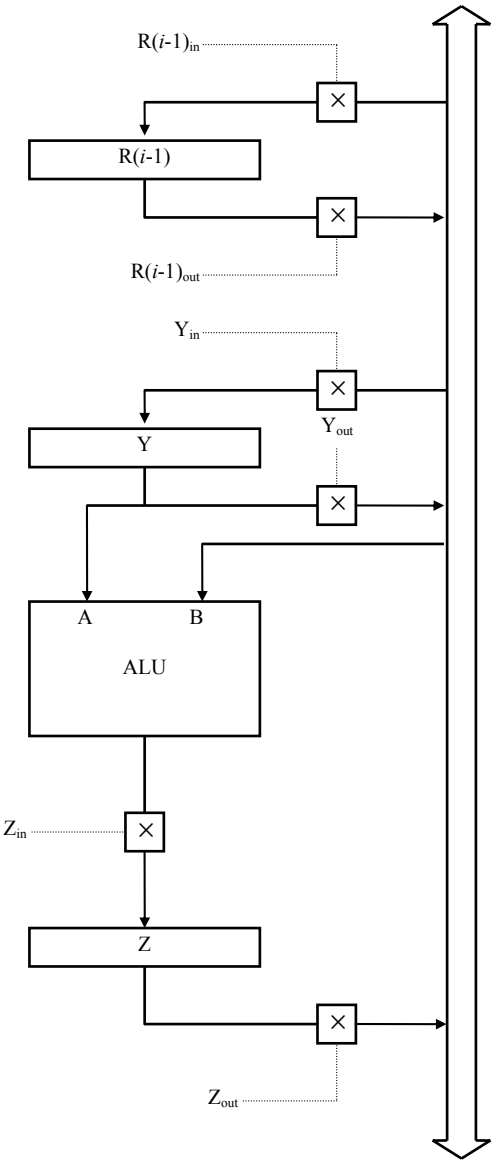
1. Da se dozvoli rad izlaznog stepena (gejta) registra R1 postavljanjem $R1_{out}$=1. Ovim se postavlja sadržaj registra R1 na CPU-ovu magistralu.
2. Da se dozvoli rad ulaznog stepena registra R4 postavljanjem $R4_{in}$=1. Ovim se podatak prisutan na CPU-ovoj magistrali puni (upisuje) u R4.

- **Aritmetičke i logičke operacije**

ALU obavlja aritmetičke i logičke operacije, a realizuje se kao kombinaciono kolo i nema internu memoriju. Zbog toga, da bi se obavilo sabiranje dva broja, oba broja moraju u toku operacije sabiranja da istovremeno budu prisutna na ulazima ALU-a.

Registar Y na slici 2.5 i slici 2.6 se koristi za tu namenu. On se koristi za čuvanje jednog od brojeva dok se drugi dovodi preko magistrale. Rezultat se smešta u registar Z. Sekvenca operacija za sabiranje sadržaja registara R1 i R2 i smeštanje rezultata u R3, ima sledeći oblik:

1. $R1_{out}$, Y_{in}
2. $R2_{out}$, Add, Z_{in}
3. Z_{out}, $R3_{in}$



Sl. 2.6. Kontrolisanje ulaza-izlaza CPU-ovih registara.

2.2.2. Tipovi prenosa CPU-a

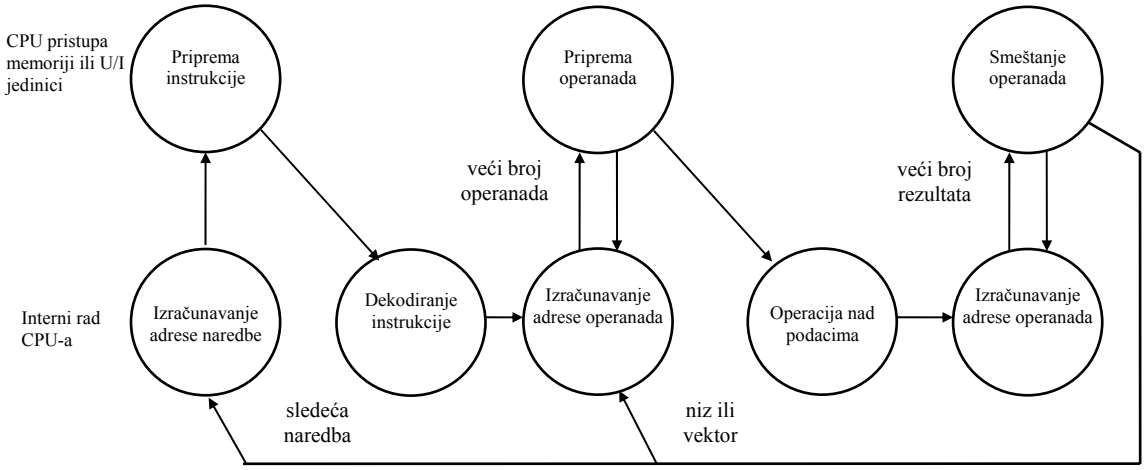
Sa tačke gledišta prenosa podataka, tj. mesta odakle se pribavljaju izvorni operandi i mesta gde se smeštaju odredišni operandi, instrukcije se mogu podeliti u sledeće tri kategorije:

a) **registar-memorija** - ostvaruje se prenos registar↔memorija.
b) **registar-registar** - ostvaruje se prenos registar↔registar.
c) **memorija-memorija** – najpre se ostvaruje prenos memorija→registar za privremeno čuvanje podataka; opciono se obavlja ALU operacija i rezultat smešta u registar za privremeno čuvanje podataka, a u trećem koraku se rezultat smešta u memoriju.

2.2.3. Dijagram stanja ciklusa instrukcija

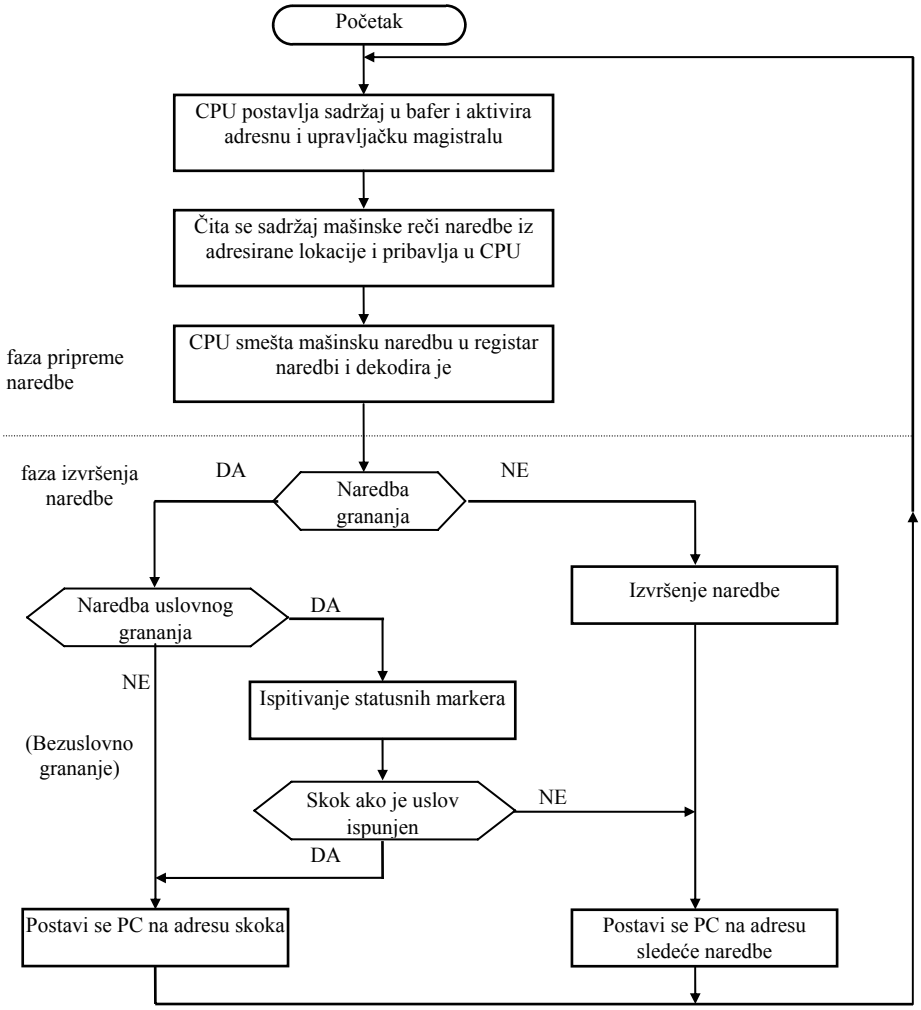
U toku ciklusa izvršenja pojedinih instrukcija može da se javi potreba za većim brojem obraćanja memoriji. Takođe, umesto obraćanja memoriji, instrukcija može da specificira U/I aktivnost. Imajući u vidu ova dodatna razmatranja, na slici 2.7 je prikazan detaljniji pogled na ciklus instrukcije. Slika 2.7 je data u obliku dijagrama stanja. Za bilo koji ciklus instrukcije neka od stanja ne moraju postojati, a druga se mogu izvršavati veći broj puta. Stanja imaju sledeće značenje:

- **Izračunavanje adrese instrukcije (iac)** - određuje se adresa instrukcije koja treba da se izvrši kao naredna. Obično se svodi na dodavanje jedinice adresi prethodne instrukcije.
- **Priprema (pribavljanje) instrukcije (if)** - instrukcija iz specificirane memorijske lokacije se čita i smešta u CPU.
- **Dekodiranje instrukcije (id)** - vrši se analiza instrukcije sa ciljem da se odredi tip operacije koja će se izvesti nad operandima koji se koriste od strane te instrukcije.
- **Izračunavanje adrese operanada (aoc)** - ako se operacijom specificira obraćanje operandu koji je smešten u memoriji ili je dostupan preko U/I, tada se određuje adresa operanda.
- **Priprema (pribavljanje) operanada (of)** - pribavlja se operand iz memorije ili se čita iz U/I uređaja.
- **Operacija nad podatkom (do)** - obavlja se specificirana operacija.
- **Smeštanje operanada (os)** - upisuje se operand u memoriju ili u U/I uređaj.



Sl. 2.7. Dijagram stanja ciklusa instrukcije.

Na slici 2.8 dat je tipični dijagram toka aktivnosti CPU-a prilikom izvršenja jedne naredbe. Kao što se vidi sa slike 2.8, izvršenje instrukcije se grubo može podeliti na dve faze: fazu pripreme naredbe i fazu izvršenja naredbe. Prva faza je identična za sve instrukcije, dok faza izvršenja zavisi od tipa instrukcije.



Sl. 2.8. Tipični dijagram toka aktivnosti CPU-a u toku izvršenja jedne naredbe.