

Univerzitet u Novom Sadu
Fakultet tehničkih nauka
Odsek za energetiku, elektroniku i telekomunikacije
Katedra za elektroniku

Beleške sa predavanja iz predmeta:

MIKROPROCESORSKA ELEKTRONIKA

Pripremio: prof. dr Veljko Malbaša
Novi Sad, novembar 2001. godine

Predgovor

U ovom materijalu nalaze se beleške sa predavanja iz predmeta Mikroprocesorska elektronika koji se, po nastavnom planu i programu Odseka za energetiku, elektroniku i telekomunikacije Fakulteta tehničkih nauka iz 1999. godine, drži u petom i šestom semestru Smera za mikroračunarsku elektroniku i Smera za telekomunikacije sa fondom časova 2+2, 2+2. S obzirom da za ovaj predmet nema odgovarajućeg udžbenika, predmetni nastavnik je preuzeo obavezu da svoje beleške sa predavanja pripremi u štampanom obliku i tako studentima obezbedi priručnik za savlađivanje gradiva. Trebalo bi da vremenom ovaj materijal bude doraden i pripremljen u obliku udžbenika za predmet Mikroprocesorska elektronika.

Treba imati u vidu da je prvo izdanje beležaka rađeno uporedo sa predavanjima, te nastavnik nije imao dovoljno vremena da tekst pripremi po sadržaju i obliku onako kako to zahtevaju univerzitetski standardi. Na primer, izostavljeni su manje važni delovi teksta, kao što su uvodi za pojedina poglavlja, pregled oblasti i praktični primeri koji ilustruju izložene koncepte. Ovi delovi gradiva biće urađeni na predavanjima.

Isti razlog sigurno je uticao na nejednak stil u izlaganju i greške koje se u ovakvim prilikama ne mogu izbeći. Sledeće izdanje beleški trebalo bi da obuhvati celo gradivo sa boljim kvalitetom teksta.

Pored svih nedostataka, nadam se da će ovaj materijal studentima dobro doći u savladavanju gradiva i pripremi ispita iz predmeta Mikroprocesorska elektronika.

Dr Veljko Malbaša
Novi Sad, novembar 2001. godine.

O predmetu

Cilj i sadržaj

U predmetu Mikroprocesorska elektronika izučavaju se osnovni principi rada mikroprocesora i organizacija i projektovanje mikroračunarskih sistema. Savladavanjem gradiva iz ovog predmeta studenti treba da budu u stanju da projektuju jednostavan mikroračunarski sistem i napišu programe koji upravljaju i koriste hardverske resurse mikroračunarskog sistema.

Izloženo gradivo ilustrovano je kroz primere nekih komercijalnih mikroprocesora koji se široko primenjuju u inženjerstvu. Posebna pažnja posvećena je programerskom modelu mikroprocesora i programiranju u mašinskom jeziku kako bi se studentima približila hardverska struktura mikroprocesora.

U okviru predmeta studenti će se upoznati sa tri mikroprocesora. Predavanja počinju upoznavanjem sa jednim jednostavnim mikroprocesorom sa radnim nazivom 'Edulent' (čita se 'edjulent'). Edulent je razvijen na Katedri za elektroniku, isključivo za edukativne svrhe. Laboratorijske vežbe počinju upoznavanjem sa korisničkim interfejsom i pisanjem jednostavnih programa za Edulent.

Organizacija savremenih mikroprocesora ilustrovana je na primeru poznatih mikroprocesora Intel x86, koji se koriste u klasi računara IBM PC. Konačno, projektovanje mikroračunarskih sistema pokazano je kroz primere primene mikrokontrolera iz porodice 8051.

Laboratorijske vežbe

Laboratorijske vežbe su obavezne. Bez urađenih laboratorijskih vežbi student ne može dobiti potpis u indeks niti pristupiti polaganju ispita. Laboratorijske vežbe se izvode u računarskoj učionici u nastavnom bloku Fakulteta tehničkih nauka.

Predviđeno je da u jesenjem semestru budu 4 laboratorijske vežbe, svaka vežba u trajanju od po tri nedelje. U prvoj vežbi studenti se upoznaju i rade jednostavne programe za Edulent. U preostale tri vežbe, od kojih svaka traje takođe po tri nedelje, rade se progami u simboličkom mašinskom jeziku mikroprocesora Intel x86. U prolećnom semestru su vežbe sa mikrokontrolerom 8051.

Redovno pohađanje vežbi podrazumeva da je student prisutan u laboratoriji u bar dve od tri nedelje predviđene za jednu vežbu. Međutim, da bi se priznale vežbe u jednom semestru student može da ima najviše tri nedeljna izostanka sa laboratorijskih vežbi.

Studenti treba da imaju u vidu da je veoma teško dobiti termine za vežbe u računarskoj učionici i da nisu predviđene nadoknade propuštenih vežbi. Samo u posebnim slučajevima (na primer bolest) sa studentom će biti napravljen poseban dogovor o nadoknadi vežbi. U svim ostalim slučajevima, studenti će morati da sačekaju sledeću školsku godinu da bi uradili vežbe i time stekli uslov za polaganje ispita.

Laboratorijske vežbe se ocenjuju i nose 40% od ukupnog broja poena koje student može da prikupi u toku školske godine. Ocena iz svake laboratorijske vežbe formira se na osnovu:

- redovnog pohađanja vežbi,
- praktične demonstracije onoga što je student uradio u toku vežbe i
- testa koji se organizuje na kraju semestra.

Kod demonstracije student treba da praktično pokaže da je uspešno uradio sve elemente koji se zahtevaju u datoj vežbi. Na kraju semestra organizuje se test sa ciljem da studenti pokažu da su samostalno uradili laboratorijske vežbe.

Kolokvijumi

U svakom semestru organizuje se kolokvijumi na kojima studenti odgovaraju na pitanja iz gradiva koje se izlaže na predavanjima. Kolokvijumi nisu obavezni, ali dovoljan broj poena na kolokvijumima oslobađaju studenta završnog ispita.

Predviđena su ukupno četiri kolokvijuma, dva u zimskom i dva u letnjem semestru. Svaki kolokvijum nosi po 15% od maksimalnog broja poena koje student može da prikupi u toku školske godine. Nastavnik će na predavanjima najaviti ukoliko dođe do promene broja kolokvijuma.

Polaganje Ispita

Ispit iz predmeta Mikroprocesorska elektronika sastoji se iz tri dela:

- laboratorijske vežbe,
- kolokvijumi,
- završni ispit.

Ukoliko student uspešno uradi laboratorijske vežbe i prikupi dovoljan broj poena na kolokvijumima, oslobođen je polaganja završnog ispita. Uspešno urađene laboratorijske vežbe nose 40% a kolokvijumi 60% od završne ocene. Završna ocena dobija se na osnovu zbira poena koje student prikupi u toku školske godine u laboratorijskim vežbama i kolokvijumima.

Za studente koji nisu prikupili dovoljan broj poena na kolokvijumima, organizuje se u redovnim ispitnim rokovima završni ispit iz celog gradiva koji se polaže pismeno.

Sledeća tabela pregledno pokazuje procentulno učešće pojedinih delova ispita u završnoj oceni.

deo ispita	Semestar	obavezan	maksimalan broj poena
laboratorijske vežbe	V	Da	20%
	VI	Da	20%
kolokvijumi	V	Ne	30%
	VI	Ne	30%
završni ispit		zavisi od kolokvijuma	60%

1. Model jednostavnog mikroprocesora

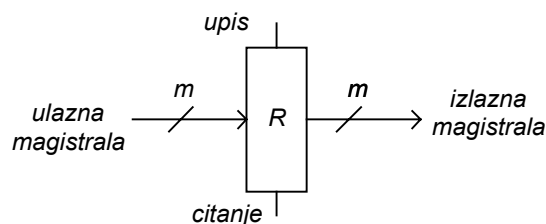
Cilj ovog poglavlja je da studente upozna sa osnovnim elementima i principima organizacije mikroprocesora i mikroračunara i pripremi studente za laboratorijske vežbe sa mikroprocesorom Edulent.

1.1 Osnovni elementi

Registar je osnovni elemenat složenog digitalnog sistema i može se zamisliti kao niz D flip-flopova sa zajedničkim signalom takta i odvojenim signalima za ulaze i izlaze pojedinih flip-flopova. Na prednjoj ivici signala takta binarni podaci koji su na ulazima prevode flip-flopove u odgovarajuća stanja, tako da se podaci sa ulaza prenose na izlaze flip-flopova. Po završetku prednje ivice takta, flip-flopovi više ne menjaju svoje stanje do sledeće prednje ivice signala takta. *Stanje registra* definisano je kao stanje niza flip-flopova koji sačinjavaju dati registar. Kažemo da registar sa m flip-flopova ima m bita.

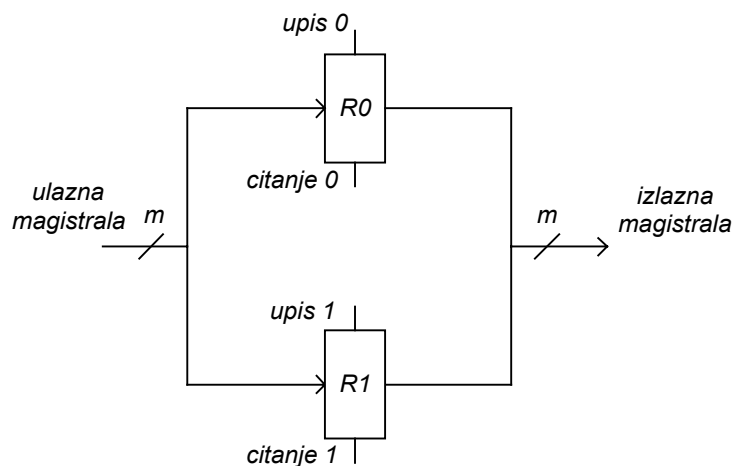
Drugi osnovni elemenat složenog digitalnog sistema su linije za prenos signala. Skup linija za prenos signala koje imaju istu ili sličnu namenu naziva se *magistrala*, engleski *bus*. Na primer, magistrala se koristi za prenos podataka sa izlaza jednog na ulaz drugog registra. Magistrale crtamo kao linije na kojima se stavlja mala kosa crta pored koje se napiše broj signala koji se prenose preko magistrale.

Slika 1.1 prikazuje registar R čiji ulazi su vezani na ulaznu, a izlazi na izlaznu magistralu, od kojih svaka ima po m linija. Indirektno možemo zaključiti da registar R ima m flip-flopova. Pored signala za binarne podatke, koji se prenose preko magistrala, registar ima i dva upravljačka signala. Upravljački signal 'upis', koji je u stvari signal takta, binarne podatke sa ulazne magistrale upisuje u registar. Treba primetiti da se ovom mikrooperacijom prethodno stanje registra nepovratno gubi.



Slika 1.1: Primer povezivanja registra sa magistralama

Signal 'čitanje' trenutno stanje registra R prenosi na izlaznu magistralu. Signal 'čitanje' i mikrooperacija čitanja sadržaja registra neophodni su kod složenih digitalnih sistema kod kojih su izlazi dva ili više registara spojeni na zajedničku izlaznu magistralu. Smisao signala čitanja ilustruje Slika 1.2, na kojoj su prikazana dva registra, R0 i R1, čiji ulazi i izlazi su vezani na zajedničku ulaznu i izlaznu magistralu, respektivno.



Slika 1.2: Primer sistema sa dva registra i zajedničkim magistralama

Mikrooperacije upisa u registre obavljaju se dovođenjem binarnog podatka na ulaznu magistralu i aktiviranjem signala za upis. Aktiviranjem signala 'upis 0' podatak sa ulazne magistrle upisuje se u registar R0, a aktiviranjem signala 'upis 1' podatak sa magistrle upisuje se u registar R1. U ovom primeru dozvoljeno je istovremeno aktiviranje oba signala, što dovodi do upisa ulaznog podatka u oba registra istovremeno.

Operacijom čitanja trebalo bi da se stanje odabranog registra prenese na izlaznu magistralu. Međutim, u ovom slučaju nastaje problem konflikta na izlaznoj magistrali. Treba se prisjetiti da izlazi dva logička kola ili izlazi dva flip-flopa ne smeju da se kratko spoje zato što ako je jedan izlaz na logičkoj 0, a drugi na logičkoj 1, onda dolazi do kratkog spoja izlaznih tranzistora. Ne samo da time dolazi do nedefinisanog stanja na izlazu nego može doći do oštećenja tranzistora.

Konflikt na izlaznoj magistrali može da se, na primer, reši posebnom konstrukcijom izlaznih stepeni flip-flopova, tako da izlazi flip-flopova mogu biti u jednom od tri stanja: 1, 0 ili stanje izlazne impedanse. Aktivni signal čitanja omogućava flip-flopovima da izlazi budu u stanju 0 ili 1, a ako signal čitanja nije aktivan onda su izlazi u stanju visoke impedanse.

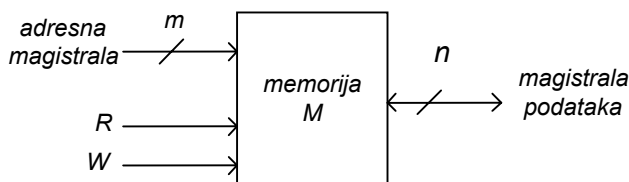
Složeni digitalni sistemi obično se opisuju na visokom nivou apstrakcije, na primer navođenjem mikrooperacija prenosa podataka između registara i magistrala. Razvijeni su i posebni formalni opisi, koji se zasnivaju na primeni formalnih jezika u opisu digitalnih sistema. Jedan takav jezik je RTL, jezik za opis prenosa podataka između registara (skraćenica od engleskog Register Transfer Language). Opis sistema koji prikazuje Slika 1.1 izgleda ovako:

```
upis : R ← ulazna magistrala
čitanje : izlazna magistrala ← R
```

Iz navedenih iskaza u RTL notaciji možemo zaključiti da u sistemu postoji jedan registar, R. Broj bita u registru R nije naveden i u datom primeru nije od interesa. Registar ima dva upravljačka signala. Signal 'upis' aktivira mikrooperaciju kojom se podatak sa ulazne magistrle upisuje u registar R. Signal 'čitanje' prenosi sadržaj registra R na izlaznu magistralu.

Memoriju možemo zamisliti kao niz registara sa zajedničkom ulaznom i zajedničkom izlaznom magistralom. Umesto naziva *memorijski registar* često se koristi naziv *memorijska lokacija*. Memorija se obično organizuje tako što se jedan skup upravljačkih signala, koji se nazivaju *adresa*, koristi za selekciju registra ili memorijske lokacije. Posebna dva upravljačka signala određuju da li se nad selektovanim registrom obavlja mikrooperacija čitanja ili upisa.

Operacije čitanja i upisa mogu se obaviti nad samo jednom memorijskom lokacijom u datom trenutku. Ovo ograničenje koristi se za smanjenje broja spoljnih priključaka za adresne signale tako što se adresni signali se interno dekoduju i tako dobijeni signali koriste za selekciju memorijske lokacije. Na ovaj način m adresnih signala koristi se za adresiranje sa 2^m memorijskih lokacija. Dalje smanjenje broja priključaka postiže se spajanjem ulazne i izlazne magistrale, te se tako umesto $2n$ priključaka za podatke koristi svega n priključaka. Spajanjem ulazne i izlazne magistrale dobija se *magistrala podataka*.



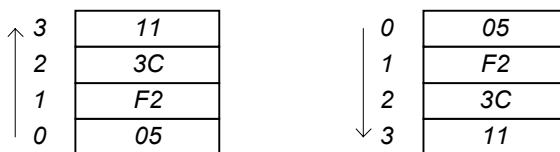
Slika 1.3: Memorija sa m adresnih i n signala za podatke

Slika 1.3 prikazuje memoriju sa m adresnih signala, n signala na magistrali podataka i signalima R za čitanje (engleski *Read*) i W za upis (engleski *Write*). Ova memorija ima 2^m memorijskih lokacija, svaka lokacija sa n bita. U RTL notaciji operacije memorije opisanje su sledećim iskazima:

```
R : magistrala ← M[adresa]
W: M[adresa] ← magistrala
```

Sa $M[adresa]$ označena je memorijska lokacija čija adresa je na adresnoj magistrali. Ukoliko je aktivan signal R memorija iz adresirane memorijske lokacije uzima sadržaj i prenosi ga na magistralu podataka. Aktivan signal W upisuje podatak sa magistrale podataka u adresiranu lokaciju.

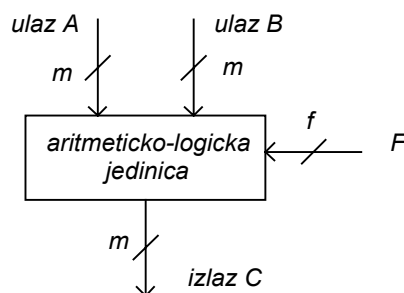
Memorija se često vizuelno prikazuje na kao niz pravougaonika koji predstavljaju memorijske lokacije. Pored pravougaonika stavlja se adresa lokacije, a sadržaj lokacije upisuje u pravougaonik. Slika 1.4 prikazuje memoriju sa 4 lokacije. Sadržaji lokacija običo se pišu u heksadecimalnom brojnem sistemu. Tako u primeru na slici, lokacija sa adresom 0 ima sadržaj 05, sledeća lokacija F2, pa 3C i konačno lokacija sa adresom 3 ima sadržaj 11, sve u heksadecimalnom brojnem sistemu.



Slika 1.4: Grafički prikaz memorijskih lokacija

U grafičkom prikazu, memorijske lokacije mogu da se poredaju tako da adrese rastu nagore (Slika 1.4, levo) ili nadole (Slika 1.4, desno). Naravno, oba prikaza imaju potpuno isto značenje.

Poslednji elemenat koji ćemo pomenuti u uvodu je aritmetičko-logička (AL) jedinica, kombinaciona mreža koja obavlja odabrane aritmetičke i logičke mikrooperacije. Slika 1.5 prikazuje AL jedinicu sa dva ulaza, A i B , i jedan izlaz C , svaki sa po m bita. Treba primetiti da postoji f upravljačkih signala označenih sa F . Ova AL jedinica može da obavi ukupno 2^f različitih mikrooperacija nad ulaznim podacima.



Slika 1.5: Aritmetičko-logička jedinica

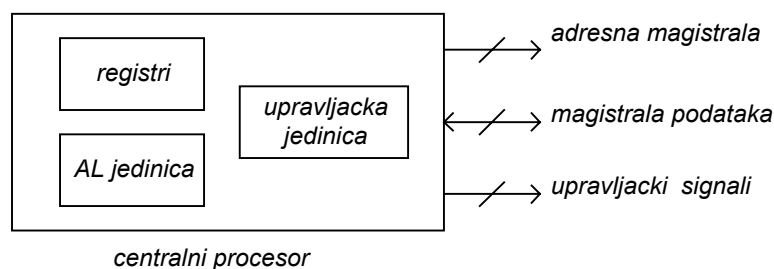
U RTL notaciji, primeri nekih mikrooperacija AL jedinice bi izgledali ovako:

$F=0 : C \leftarrow A$
 $F=1 : C \leftarrow A+B$
 $F=2 : C \leftarrow A-B$

Prema tome, ako su upravljački signali F svi jednaki 0, onda AL jedinica na izlaz C prosleđuje ulaz A . Ako upravljački signali imaju vrednosti 00...01, odnosno decimalno 1, onda AL jedinica na izlazu C daje sumu ulaza A i B , i tako dalje.

1.2 Procesor

Pogodnim vezivanjem registara i aritmetičko logičke jedinice dobija se *procesor*, jedinica koja je može da obavlja operacije nad podacima kao što su kopiranje podataka iz jednog registra u drugi ili sabiranje sadržaja dva registra. *Centralna procesorska jedinica* ili *centralni procesor*, pored registara i AL jedinice ima i upravljačku jedinicu koja upravlja ostalim delovima procesora i računarskog sistema. Slika 1.6 prikazuje blok šemu jednostavnog centralnog procesora.



Slika 1.6: Blok šema jednostavnog centralnog procesora

Spoljni priključci centralnog procesora grupišu se u tri magistrale: adresnu magistralu, magistralu podataka i magistralu preko koje se prenose upravljački signali. Preko adresne magistrale prenose se adresni signali koji mogu da se koriste, na primer, za adresiranje memorijske jedinice. Magistrala podataka služi za prenos podataka, na primer za prenos sadržaja neke memorijske lokacije u centralni procesor.

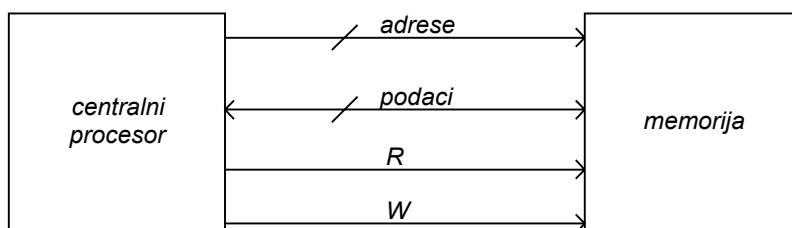
Treba primetiti da je adresna magistrala jednosmerna, što znači da se adresni signali uvek prenose od centralnog procesora prema nekoj drugoj jedinici računarskog sistema. Sa druge strane, magistrala podataka je dvosmerna, tako da se podaci mogu prenositi iz centralnog procesora prema nekoj drugoj jedinici ili u suprotnom smeru. Takođe, treba primetiti da na slici nisu navedeni brojevi linija u magistralama, jer je to podatak koji za uvodno izlaganje nije od interesa.

Preko upravljačkih signala centralni procesor upravlja svim ostalim jedinicama računarskog sistema. Primeri upravljačkih signala su R (čitanje) i W (upis) koji određuju mikrooperaciju koju memorija treba da izvrši. Sve upravljačke signale, uključujući signale za upravljanje registrima i AL jedinicom kao i signale koji upravljaju spoljnim jedinicama, generiše upravljačka jedinica. Upravljačka jedinica je sekvencijalna mreža koja generisanjem upravljačkih signala organizuje rad celog računarskog sistema.

Naravno, pored navedenih spoljnih priključaka za magistrale, centralni procesor ima priključke za napajanje i masu, priključak za sinhronizacioni signal (*clock*) i još neke signale ćemo kasnije pomenuti.

1.3 Osnovni princip rada

U najjednostavnijem slučaju računarski sistem sastoji se od centralne procesorske jedinice koja je preko magistrala vezana sa memorijskom jedinicom. Signali ove dve jedinice su obično međusobno kompatibilni u pogledu naponskih nivoa, značenja, broja, ponašanja u toku vremena i tako dalje, i mogu se neposredno povezati linijama za prenos signala. Slika 1.7 prikazuje način povezivanja centralnog procesora i memorije: adresna magistrala procesora veže se na adresnu magistralu memorije, magistrala podataka procesora veže se sa magistralom podataka memorije, a upravljački signali čitanja (R) i upisa (W) sa procesora vežu se na odgovarajuće signale memorije.



Slika 1.7: Veza centralnog procesora i memorije

Osnovna operacija koja se obavlja u ovakvom sistemu je prenos podataka između centralnog procesora i memorije. Prenos podataka obično zahteva nekoliko koraka koji se moraju izvršiti jedan za drugim u toku vremena. Na primer, prenos nekog podatka iz memorije u registar centralnog procesora mogao bi da se izvrši na sledeći način:

- Centralni procesor odredi adresu memorijske lokacije iz koje treba pročitati sadržaj.
- Centralni procesor stavi adresu na adresnu magistralu i aktivira signal čitanja R .
- Memorija obavi operaciju čitanja tako što uzme sadržaj iz adresirane memorijske lokacije i stavi ga na magistralu podataka.
- Podatak sa magistrale podataka centralni procesor stavi u jedan od svojih registara.

U RTL notaciji prenos podatka iz memorije u registar R centralnog procesora opisuje se iskazom:

$$R \leftarrow M[adresa]$$

Na sličan način centralni procesor može da organizuje prenos sadržaja registra R u neku memorijsku lokaciju, u RTL notaciji označeno sa:

$$M[adresa] \leftarrow R$$

Umesto RTL opisa, operacije koje procesor obavlja predstavljaju se instrukcijama koje se sastoje od nizova pisanih simbola kao što su slova i brojevi. U instrukcijama se koriste ključne reči za opis

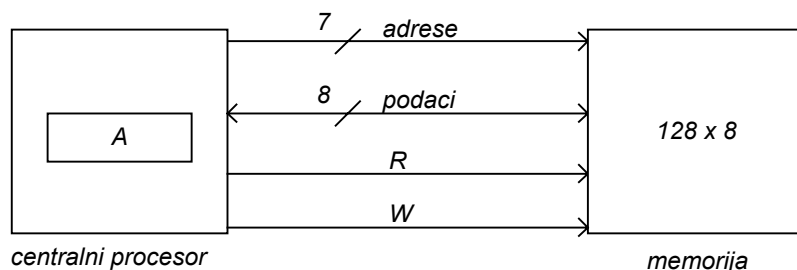
operacija. Na primer, za prenos podatka sa jednog mesta na drugo, recimo za prenos iz memorije u registar procesora ili obratno, koristi se engleska reč MOVE (pomeri ili prenesi). Instrukcija za prenos sadržaja memorijske lokacije u registar *R* procesora mogla bi da ima sledeći oblik:

MOVE R,adresa

Ovde se podrazumeva da prvi niz simbola (*MOVE*) označava operaciju, drugi niz (*R*) označava odredište u koje se upisuje podatak, a treći niz (*adresa*) označava adresu memorijske lokacije iz koje se uzima podatak. Na sličan način bi instrukcija kojom se sadržaj registra *R* prenosi u adresiranu memorijsku lokaciju imala sledeći oblik:

MOVE adresa,R

Slika 1.8 prikazuje osnovnu konfiguraciju računara Edulent, koji ćemo koristiti za ilustraciju principa rada računara. Memorijska jedinica Edulenta sastoji se iz jednog memorijskog modula koji ima 128 memorijskih lokacija. Svaka memorijska lokacija ima 8 bita. Za adresiranje memorijskih lokacija koristi se adresna magistrala sa 7 adresnih signala. Upravljačka jedinica centralnog procesora generiše signale *R* za čitanje iz memorije i *W* za upis u memoriju. Centralni procesor ima jedan registar za smeštanje podataka koji se naziva akumulator i koji je označen sa *A*.



Slika 1.8: Osnovna konfiguracija računara Edulent

Za ilustraciju instrukcija za prenos podataka posmatrajmo početni sadržaj memorije, Slika 1.9. U ovom primeru adrese rastu nadole i označene su decimalnim brojevima od 0 do 127. U pravougaonike koji predstavljaju memorijske lokacije upisani su sadržaji memorijskih lokacija, koji su odabrani proizvoljno i predstavljeni u heksadecimalnom brojnom sistemu.

0	00
1	11
2	22
3	33
4	44
5	55
6	66
7	77
...	
127	...

Slika 1.9: Početni sadržaj memorije

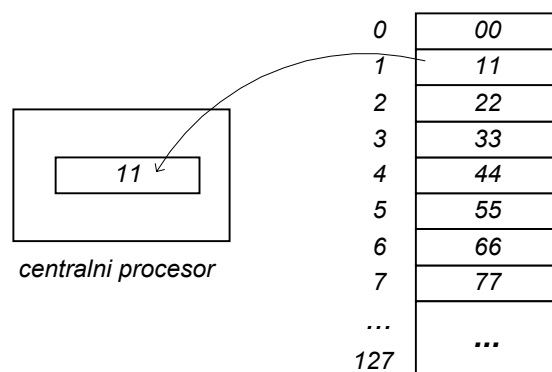
Pretpostavimo da sadržaj lokacija sa adresama 1, 2 i 3 treba preneti (prekopirati) u lokacije sa adresama 5, 6 i 7. Jedini način da Edulent obavi ovaj zadatak je da sadržaj memorijske lokacije prvo prenese u registar *A* i da onda iz registra *A* sadržaj prenese u traženu memorijsku lokaciju. Kada završi sa prvim prenosom, procesor treba da uradi drugi, a zatim treći prenos. Niz instrukcija, ili program, koji obavlja ovaj prenos izgleda ovako:

```

MOV A,adresa1    ; prenesi sadržaj lokacije 1 u akumulator
MOV adresa5,A    ; prenesi sadržaj akumulatora u lokaciju 5
MOV A,adresa2    ; ponovi za lokacije 2 i 6
MOV adresa6,A
MOV A,adresa3    ; ponovi za lokacije 3 i 7
MOV adresa7,A

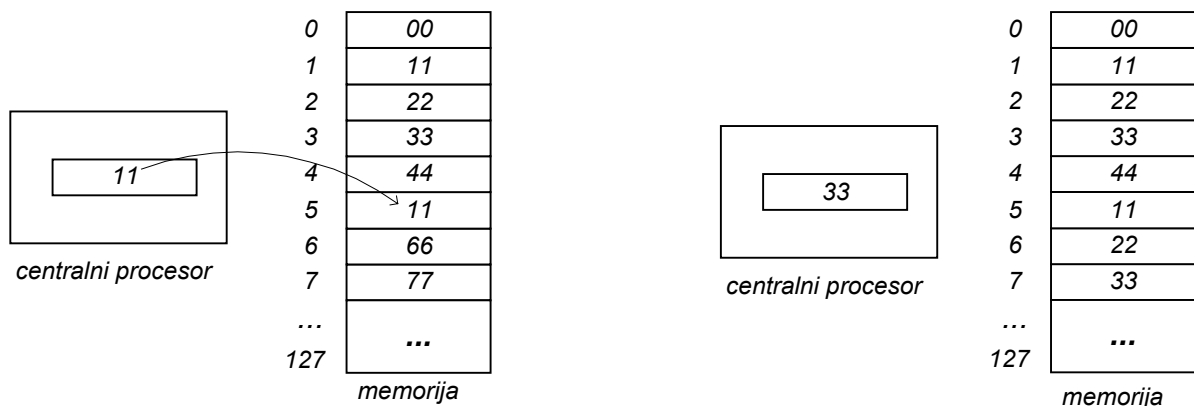
```

Naravno, ovde smo pretpostavili da Edulent niz instrukcija izvršava po redosledu kojim su napisane. Dakle, Edulent izvrši prvu instrukciju, pa kad nju završi, onda izvrši sledeću i tako dalje. Komentar instrukcija napisan je iza znaka ;. Izvršenje ovog programa pratićemo kroz stanja Edulenta, odnosno sadržaja akumulatora i sadržaja memorijskih lokacija. Slika 1.10 prikazuje stanje Edulenta posle izvršenja instrukcije *MOV A,adresa1*. Naravno, strelica opisuje prenos podataka na visokom nivou apstrakcije; jasno je da se adresa memorijske lokacije (1) prenosi preko adresne magistrale, a sadržaj memorijske lokacije (11) preko magistrale podataka.



Slika 1.10: Stanje Edulenta posle instrukcije *MOV A, 1*

Sledeća instrukcija, *MOV adresa5,A* prenosi sadržaj akumulatora (11) u lokaciju sa adresom 5, Slika 1.11 levo. U ovom trenutku, na isti način može da počne prenošenje sadržaja lokacije sa adresom 2 u lokaciju sa adresom 6. Ovaj prenos obavljaju sledeće dve instrukcije.



Slika 1.11: Stanje Edulenta posle druge instrukcije (levo) i posle poslednje instrukcije (desno)

Na kraju, poslednje dve instrukcije završavaju prenos sadržaja iz lokacije 3 u lokaciju 7. Stanje Edulenta posle izvršenja poslednje instrukcije pokazuje Slika 1.11 desno.

Važno je primetiti da računar izvršava instrukcije sekvencijalno, jednu po jednu po redosledu kojim su napisane. Drugo, treba primetiti da su instrukcije veoma jednostavne i za obavljanje zadataka

koji nisu trivijalni neophodno je da računar izvrši veliki broj instrukcija. *Program* je niz instrukcija koje se koriste za rešavanje nekog problema.

Edulent ima instrukciju za sabiranje kojom se sadržaj akumulatora *A* sabira sa adresiranoj memorijskoj lokaciji, a rezultat se smešta u akumulator. Instrukcija i RTL opis izgledaju ovako:

```
ADD A,adresa      ;  $A \leftarrow A + M[adresa]$ 
```

U sledećem primeru treba sabirati tri broja koji se nalaze u lokacijama sa adresama 8, 9 i 10, a rezultat staviti u lokaciju sa adresom 11. Program izgleda ovako:

```
MOV A,adresa8      ; stavi prvi broj u akumulator
ADD A,adresa9      ; saberi drugi broj
ADD A,adresa10     ; saberi treći broj
MOV adresa11,A     ; stavi rezultat u lokaciju sa adresom 11
```

U programu prepoznavamo tri celine. Prvo se formira početni sadržaj akumulatora *A* tako što se prvi broj iz niza brojeva koji se sabiraju stavi u akumulator. Naravno, početni sadržaj može se formirati i tako što se u akumulatora stavi 0.

Druga celina sastoji se iz ponavljanja instrukcije kojom se sledeći broj u nizu sabira sa sadržajem akumulatora i rezultat se stavlja natrag u akumulator. Na ovaj način se sabiraju jedan po jedan broj i suma akumulira registru *A* odakle potiče naziv 'akumulator'. Na kraju se tražena suma stavlja u lokaciju sa adresom 11.

Na ovaj način mogu se sabirati sadržaji proizvoljnog broja memorijskih lokacija u nizu. Međutim, očigledno je da ovakav način pisanja programa nije efikasan zato što se za operacije koje se ponavljaju koriste uvek nove instrukcije. Očigledno da nije dobro na ovaj način napisati program koji sabira 100 ili na primer milion brojeva. Druga primedba odnosi se na tačnost programa. Pošto se sabiraju 8-bitni brojevi a za rezultat koristi 8-bitni akumulator, može se dogoditi da rezultat ima više od 8 bita i da ne može da stane u akumulator. U tom slučaju nastaje greška koju ovaj program nije u stanju da otkrije.

1.4 Mašinski jezik i format instrukcije

Savremeni digitalni računari projektovani su po principima koje je formulisao matematičar von Neumann sredinom prošlog veka. Ključna osobina von Neumann-ovih računara je da se memorija koristi za smeštanje programa koje računar izvršava i podataka koje računar obrađuje u toku izvršenja programa. Preme tome, računari su programabilni i mogu da izvršavaju različite programe.

Treba imati u vidu da memorijske lokacije mogu da sadrže samo binarne brojeve. Zbog toga je neophodno sve instrukcije koje računar može da izvrši predstaviti u obliku binarnih brojeva. Na primer, instrukcija Edulenta:

```
MOV A,adresa8
```

predstavlja se binarnim brojem 0001 0001 0000 1000, ili heksadecimalno 1108. Dakle, operaciju sabiranja sadržaja akumulatora *A* sa sadržajem memorijske lokacije sa adresom 8 i smeštanje dobijene sume u akumulator, koju u RTL-u opisujemo sa:

```
 $A \leftarrow A + M[8]$ 
```

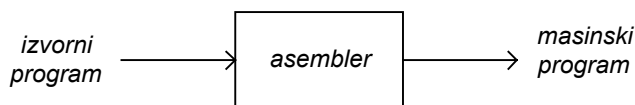
možemo predstaviti na dva načina. Ako koristimo nizove simbola koji uključuju slova i brojeve, onda operaciju predstavljamo u obliku:

ADD A,adresa8

Ovaj oblik naziva se *simbolička mašinska instrukcija*. Simboličke mašinske instrukcije pišu se po pravilima koja su sastavni deo *simboličkog mašinskog jezika*. Na primer, jedno pravilo je da instrukcija počinje nizom slova koji predstavlja operaciju kao što je u ovom slučaju sabiranje.

Kao što smo već videli, ista instrukcija može da se predstavi binarnim brojem 0001 0001 0000 1000. Treba obratiti pažnju da se instrukcija može zapisati u memoriju samo u obliku binarnog broja i to je jedini oblik koji centralni procesor razume i može da interpretira. Ovaj oblik predstavljanja instrukcije naziva se *mašinska instrukcija*, a pravila pisanja *mašinski jezik*. Sa druge strane, simbolički mašinski jezik koristi se zato što je razumljiviji programeru. Programer koji piše programe u simboličkim mašinskom jeziku mnogo je efikasniji i pravi manje grešaka nego programer koji piše programe u mašinskom jeziku. *Izvorni program* je program napisan na simboličkom mašinskom jeziku, a *mašinski program* je program koji se sastoji iz mašinskih instrukcija.

Izvorni pogrami moraju se prevesti na mašinski jezik pre izvršenja programa na računaru. Iako prevođenje može da se uradi ručno, u praksi se koriste *asembleri*, programi koji automatski prevode izvorni program u mašinski program. Ulaz assemblera je izvorni program, a izlaz program predstavljen u mašinskom jeziku, spreman za izvršenje na računaru, Slika 1.12.



Slika 1.12: Assembler prevodi izvorni program u mašinski program

Ne treba da zbunjuje činjenica da se jedan program, kao što je assembler, koristi za prevođenje drugih programa iz jednog oblika u drugi. Računar koji izvršava assembler koristi se za automatsko prevođenje izvornog programa u mašinski program. U nekim slučajevima neophodno je obaviti dodatne operacije nad mašinskim programom da bi se on pripremio za izvršenje. Mi ćemo za sada smatrati da je mašinski program u obliku koji može da se neposredno izvrši i nazivaćemo ga izvršni oblik mašinskog programa ili prostije *izvršni program*. Računar koji je obavio prevođenje izvornog programa na mašinski jezik ili neki drugi računar mogu da se koriste za izvršavanje dobijenog izvršnog programa.

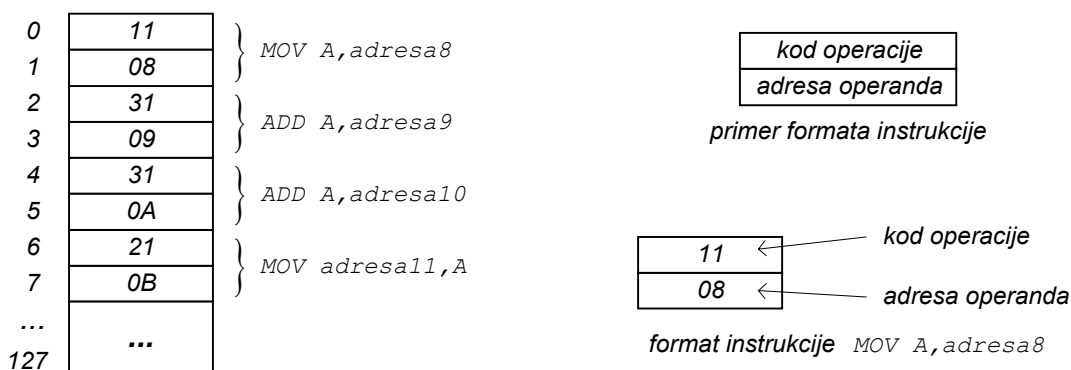
Posmatrajmo ponovo primer izvornog programa za sabiranje tri broja i pogledajmo kako izgleda njegov oblik u mašinskom jeziku.

<i>MOV A,adresa8</i>	<i>;</i>	<i>0001 0001 0000 1000</i>	<i>(hex: 11 08)</i>
<i>ADD A,adresa9</i>	<i>;</i>	<i>0011 0001 0000 1001</i>	<i>(hex: 31 09)</i>
<i>ADD A,adresa10</i>	<i>;</i>	<i>0011 0001 0000 1010</i>	<i>(hex: 31 0A)</i>
<i>MOV adresa11,A</i>	<i>;</i>	<i>0010 0001 0000 1011</i>	<i>(hex: 21 0B)</i>

U prvoj koloni su instrukcije simboličkog mašinskog jezika, a u drugoj koloni su instrukcije u mašinskom jeziku predstavljene u binarnom brojnom sistemu. Umesto binarnih obično koristimo heksadecimalne brojeve koji su u primeru napisani u zagradama.

Zanimljivo je pogledati kako program izgleda u memoriji, Slika 1.13, levo. Vidi se da je svaka instrukcija predstavljena sa dva bajta i smeštena u dve susedne memorijske lokacije. U opštem slučaju, instrukcije mogu da budu predstavljene različitim brojem bajtova, pre prema tome mogu da

zauzimaju različit broj memorijskih lokacije. *Format instrukcije* je način predstavljanja instrukcije u binarnom brojnem sistemu i pokazuje značenje pojedinih grupa bitova.



Slika 1.13: Izgled programa u memoriji (levo) i format jedne instrukcije (desno)

Slika 1.13 desno gore prikazuje primer formata, u kome prvi bajt predstavlja operaciju koju treba izvršiti, a drugi bajt predstavlja adresu podatka koji se obrađuje u toku izvršenja instrukcije. Binarni broj koji predstavlja operaciju koja se izvršava naziva se *kod operacije*. *Operand* je podatak koji se obrađuje ili prenosi sa jednog mesta na drugo. Instrukcija *MOV* za prenos podataka, Slika 1.13 desno dole, saglasna je navedenom formatu, u kome prvi bajt 11 (heksadecimalno) predstavlja kod operacije, a drugi bajt 09 je adresa operanda koji treba preneti iz lokacije sa adresom 08 u akumulator.

Već na prvi pogled može se videti da sve instrukcije programa za sabiranje, Slika 1.13 levo, imaju isti format. Na primer, kod operacije instrukcije *ADD* za sabiranje je 31 (heksadecimalno) a drugi bajt je adresa operanda koji treba sabrati sa sadržajem akumulatora. Tako instrukcije

```
ADD A,adresa9      ; 0011 0001 0000 1001    (hex: 31 09)
ADD A,adresa10     ; 0011 0001 0000 1010    (hex: 31 0A)
```

imaju isti kod operacije, dok je adresa operanda u prvoj instrukciji 09 a u drugoj 0A (heksadecimalno).

1.5 Izvršavanje instrukcija

Kao što je već rečeno, računar izvršava instrukcije jednu za drugom, *sekvencijalno*, po redosledu kojim su upisane u memorijske lokacije. Sekvencijalno izvršavanje instrukcija je pogodno zato što se inkrementiranjem adrese instrukcije koja se izvršava dobija adresa instrukcije koju treba sledeću izvršiti. Mehanizam određivanja adrese instrukcije koju treba izvršiti oslanja se na *programski brojač*, specijalizovani registar sa paralelnim ulazom i paralelnim izlazom, koji ima upravljački signal za povećanje sadržaja za jedan. Programski brojač obično se označava sa *PC* od engleskog *program counter*.

Ideja korišćenja programskog brojača je jednostavna i zasniva se na sledećem algoritmu:

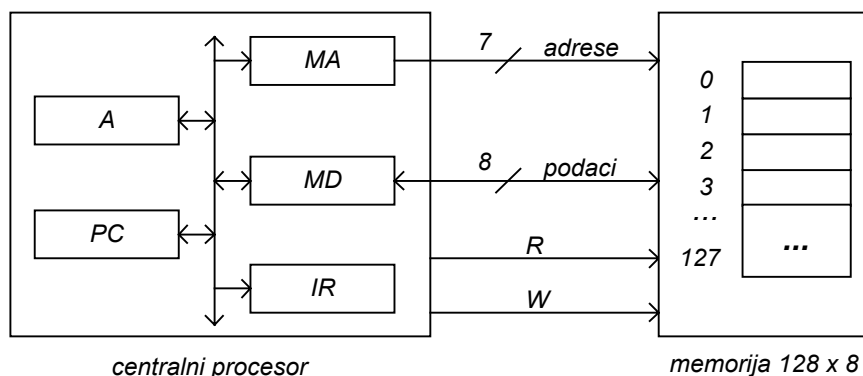
1. Stavi u programski brojač adresu prve instrukcije programa.
2. Izvrši instrukciju čija adresa je u programskom brojaču.
3. Povećaj sadržaj programskog brojača za 1.
4. Pređi na korak 2.

Prema tome, programski brojač uvek sadrži adresu instrukcije koju treba izvršiti, a inkrementiranjem programskog brojača centralni procesor računa adrese i izvršava jednu po jednu instrukciju programa. Naravno, ovo je samo osnovna ideja koja ne daje odgovore na pitanja kao što su kako se početna adresa stavlja u programski brojač i kako se menja redosled izvršavanja instrukcija u programu.

Centralni procesor sadrži još tri specijalna registra koji se koriste kod izvršenja instrukcije:

- Instrukcijski registar (IR), koji prihvata kod operacije instrukcije koju treba izvršiti.
- Memorijski adresni registar (MA), koji sadrži adresu koja se pojavljuje na adresnoj magistrali.
- Registar podataka (MD, od engleskog *Memory Data*), dvosmerni registar koji prihvata podatak sa magistrale podataka kod mikrooperacije čitanja, odnosno koji sadrži podataka koji se stavlja na magistralu podataka kod mikrooperacije upisa.

Slika 1.14 prikazuje računar Edulent sa specijalizovanim registrima centralnog procesora: PC, MA, MD i IR. Vertikalna dvosmerna strelica unutar centralnog procesora predstavlja internu magistralu koja se koristi za prenos podataka između registara centralnog procesora. Neke funkcionalne jedinice centralnog procesora, kao što su aritmetičko-logička i upravljačka jedinica, radi jednostavnosti nisu prikazane na slici.



Slika 1.14: Edulent sa registrima MA, MD i IR centralnog procesora

Registri MA i MD koriste se za komunikaciju centralnog procesora sa spoljnim magistralama. Osnovne mikrooperacije koje se koriste u komunikaciji sa spoljnim magistralama su:

$adresna\ magistrala \leftarrow MA$
 $magistrala\ podataka \leftarrow MD$
 $MD \leftarrow magistrala\ podataka$

Prema tome, sadržaj registra MA može se preneti na adresnu magistralu, ali ne i obratno, sadržaj magistrale podataka ne može da se prenese u registar MA. Prema tome, adresna magistrala je jednosmera, u smeru od centralnog procesora. Magistrala podataka je dvosmerna i sadržaj registra MD i magistrale podataka može da se razmenjuje u oba smera.

Interna magistrala koristi se za izvršavanje sledećih mikrooperacija prenosa sadržaja između registara centralnog procesora:

$A \leftarrow MD, PC \leftarrow MD, MA \leftarrow MD, IR \leftarrow MD$
 $MD \leftarrow A$

Kombinacijom navedenih mikrooperacija može se organizovati izvršenje instrukcija. Na primer, instrukcija *MOV A,adresa* izvršava se sekvencijalnim izvršenjem sledećih mikrooperacija:

```

MA ← PC      ; prenos adrese instrukcije u MA
MD ← M[MA]    ; prenos koda operacije iz memorije u MD
PC ← PC+1     ; povećanje sadržaja PC za 1
IR ← MD       ; prenos koda operacije u IR

MA ← PC      ; prenos adrese narednog bajta u MA
MD ← M[MA]    ; prenos adrese operanda iz memorije u MD
PC ← PC+1     ; povećanje sadržaja PC za 1
MA ← MD       ; prenos adrese operanda iz MD u MA
MD ← M[MA]    ; prenos operanda iz memorije u MD
A ← MD        ; prenos operanda iz MD u A

```

Dakle, izvršenje ove instrukcije sastoji se iz sekvencijalnog izvršenja 10 mikrooperacija. Neke od ovih mikrooperacija mogu da se izvršavaju paralelno, odnosno u istoj period sekvencijalnog signala, ali to za sada nije od interesa. Izvršenje mikrooperacija može se pratiti preko sadržaja registara centralnog procesora. Neka je sadržaj memorije kao što prikazuje Slika 1.13 i neka je početni sadržaj registara (heksadecimalno):

$PC = 00, A = ??, MA = ??, MD = ??, IR = ??$

Sa ?? označen je sadržaj koji može biti slučajan i nije bitan. Pošto je sadržaj registra PC jednak 00, centralni procesor će izvršiti instrukciju koja je u memorijskog lokaciji sa adresom 0. Sadržaj memorijske lokacije sa adresom 0 je 11 (sve u heksadecimalnom brojnem sistemu), a to je kod operacije za instrukciju *MOV A, adresa*, pri čemu je adresa operanda u sledećoj lokaciji, sa adresom 1, i jednaka je 08. Izvršenje instrukcije počinje mikrooperacijom $MA \leftarrow PC$, koja sadržaj registra PC prebacuje na internu magistralu, a sa interne magistrale u registar *MA*. Sadržaji registara posle ove mikrooperacije su:

$PC = 00, A = ??, MA = 00, MD = ??, IR = ??$

U ovom trenutku sadržaj registra *MA* prosleđuje se na adresnu magistralu i upravljačka jedinica generiše upravljački signal *R* (čitanje). Memorija, sa svoje strane, nadgleda upravljačke signale *R* i *W* i kada otkrije da je jedan od njih aktivan uradi odgovarajuću mikrooperaciju. U ovom slučaju, to je mikrooperacija čitanja, pa će memorija da sadržaj memorijske lokacije čija adresa je na adresnoj magistrali (00), a to je 11, da prebaci na magistralu podataka. Centralni procesor sadržaj magistrale podataka upisuje u registar MD, što je simbolički predstajeno sledećom mikrooperacijom $MD \leftarrow M[MA]$. Sadržaji registara su sada:

$PC = 00, A = ??, MA = 00, MD = 11, IR = ??$

Sledeća mikrooperacija, $PC \leftarrow PC+1$, povećava sadržaj programskog brojača za 1, tako da PC sada pokazuje na sledeću memorijsku lokaciju sa adresom 1. Mikrooperacija $IR \leftarrow MD$ sadržaj registra MD, a to je kod operacije, prenosi u instrukcijski registar *IR*:

$PC = 01, A = ??, MA = 00, MD = 11, IR = 11$

Upravljačka jedinica dekodira sadržaj instrukcijskog registra *IR*, zaključuje da se radi o instrukciji *MOV A, adresa* i generiše sve neophodne signale koji dovode do izvršenja preostalih mikrooperacija ove instrukcije. U ovom trenutku potrebno je prvo uzeti adresu operanda (08) koja se nalazi u memorijskoj lokaciji na koju pokazuje registar PC (01), što obavljaju mikroinstrukcije $MA \leftarrow PC$ i $MD \leftarrow M[MA]$. Naravno, kada uzme adresu operanda koja je sada u registru MD, centralni procesor povećava sadržaj programskog brojača ($PC \leftarrow PC+1$) i prenosi adresu operanda u registar *MA* (mikrooperacija $MA \leftarrow MD$). U ovom trenutku sadržaji registara su:

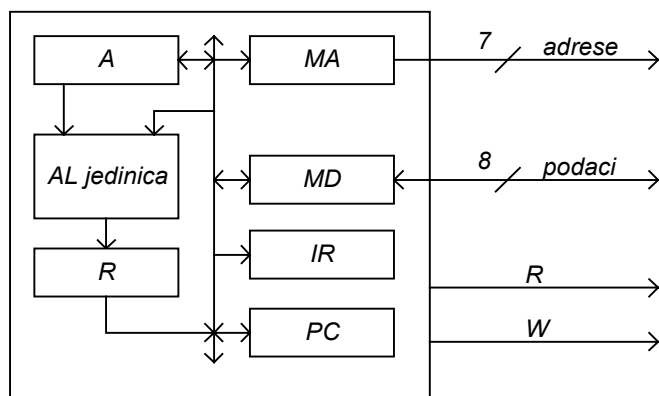
$PC = 02, A = ??, MA = 08, MD = 08, IR = 11$

Sada je sve spremno za uzimanje operanda koji je u lokaciji sa adresom 08 i njegovo prebacivanje u akumulator. Poslednje dve mikrooperacije iz memorijske lokacije čija adresa je u MA (08) uzimaju sadržaj (na primer, neka je to 88) i prenose ga prvo u registar MD , a zatim iz registra MD u akumulator, što dovodi do kompletiranja ove instrukcije i sadržaja registara:

$PC = 02, A = 88, MA = 08, MD = 88, IR = 11$

Na ovom mestu, u akumulatoru je traženi operand koji je prenesen iz memorijske lokacije čija adresa je u adresnom polju instrukcije MOV , a u programskom brojaču je adresa sledeće instrukcije koju treba izvršiti. Na sličan način centralni procesor izvršava sledeću instrukciju i tako redom ostale instrukcije programa.

Za izvršenje instrukcije $ADD A, adresa$ neophodna je aritmetičko-logička jedinica, koja je povezana na način koji prikazuje Slika 1.15. Jedan ulaz AL jedinice dolazi iz akumulatora, a drugi sa interne magistrale. Kao posledica ovakvog vezivanja ulaza AL jedinice, jedan od operandu u operacijama koje zahtevaju dva operandu, uvek mora da bude u akumulatoru. Drugi operand može da se dovede iz nekog drugog registra, na primer MD , preko interne magistrale.



Slika 1.15: Centralni procesor Edulenta sa AL jedinicom

Izlaz iz AL jedinice ne može da se dovede neposredno na internu magistralu, zato što bi došlo do konflikta jer je operand koji se dovodi na drugi ulaz AL jedinice već na internoj magistrali. Zato postoji prihvatni registar R koji služi za privremeno prihvatanje rezultata AL jedinice.

Korišćenje AL jedinice ilustrovaćemo na primeru izvršenja instrukcije $ADD A, adresa$ koja sabira sadržaja akumulatora sa operandom u memorijskoj lokaciji sa adresom $adresa$ i rezultat stavlja u akumulator:

```

MA ← PC      ; prenos adrese instrukcije u MA
MD ← M[MA]   ; prenos koda operacije iz memorije u MD
PC ← PC+1    ; povećanje sadržaja PC za 1
IR ← MD      ; prenos koda operacije u IR

MA ← PC      ; prenos adrese narednog bajta u MA
MD ← M[MA]   ; prenos adrese operanda iz memorije u MD
PC ← PC+1    ; povećanje sadržaja PC za 1
MA ← MD      ; prenos adrese operanda iz MD u MA
MD ← M[MA]   ; prenos operanda iz memorije u MD
R ← A+MD     ; sabiranje operanda sa A i smeštanje sume u R
A ← R        ; prenos sume iz R u A

```

Zanimljivo je da je niz mikrooperacija u obe instrukcije veoma sličan. Zapravo prve 4 mikrooperacije identične su za sve instrukcije i nazivaju se *pripremna faza*. Ostale mikrooperacije razlikuju se kod svih instrukcija i nazivaju se *izvršna faza*. Dakle, izvršenje instrukcije sastoji se iz dve faze: u toku pripremne faze centralni procesor uzima kod operacije iz memorije i smešta ga u registar IR, a u izvršnoj fazi izvršava se instrukcija čiji je kod operacije u IR. Po završetku jedne, adresa naredne instrukcije koju treba izvršiti automatski se nalazi u registru IR. Na taj način pripremna faza naredne instrukcije može da krene odmah po završetku izvršne faze prethodne instrukcije.

Lista mikrooperacija koje dovode do izvršenja instrukcije sadrži detalje koji se podrazumevaju. Na primer, prenos koda operacije iz memorije u registar IR sastoji se iz tri mikrooperacije:

```
MA ← PC      ; prenos adrese instrukcije u MA
MD ← M[MA]   ; prenos koda operacije iz memorije u MD
IR ← MD      ; prenos koda operacije u IR
```

Da bi smo pojednostavili opis mikrooperacija, često izostavljamo mikrooperacije koje se podrazumevaju. Na primer, navedene tri mikrooperacije za prenos koda operacije iz memorije u registar IR, pojednostavljeno mogu da se predstavljaju jednom mikrooperacijom:

```
IR ← M[PC]   ; prenos koda operacije u iz memorije u IR
```

Naravno, treba uvek imati u vidu da je ovaj način predstavljanje pojednostavljen i da je na hardverskom nivou neophodno izvršiti sve tri mikrooperacije. Izostavljanjem mikrooperacije koje se podrazumevaju dobijamo opis instrukcija na višem nivou apstrakcije koji je lakši za razumevanje. Na primer, izvršenje instrukcije *MOV A,adresa* predstavlja se sledećim apstraktnim opisom:

```
PF:  IR ← M[PC]   ; prenos koda operacije u iz memorije u IR
      PC ← PC+1   ; povećanje sadržaja PC za 1

IF:  MA ← M[PC]   ; prenos adrese operanda iz memorije u registar MA
      PC ← PC+1   ; povećanje sadržaja PC za 1
      A ← M[MA]   ; prenos operanda iz memorije u akumulator
```

Sa *PF* obeležena je pripremna, a sa *IF* izvršna faza instrukcije. Pošto su pripremne faze svih instrukcija iste, možemo ih izostaviti, pa se tako apstraktni opis instrukcije svodi na apstraktni opis njene izvršne faze. Na najvišem nivou apstrakcije možemo još više pojednostaviti opis instrukcija tako što izostavljamo sve nepotrebne mikrooperacije a ostavljamo samo mikrooperacije koje dovode do izmene sadržaja registara koji su vidljivi programeru. Tako bi izvršna faza instrukcije *MOV A,adresa* izgledala ovako:

```
A ← M[adresa] ; prenos operanda iz memorije u akumulator
```

Na ovom nivou apstrakcije ne vide se detalji kao što su inkrementiranje programskog brojača PC i korišćenje memorijskog adresnog registra MA. Sa druge strane, apstraktni opis ima prednost jer nije vezan za konkretnu realizaciju centralnog procesora. Mi ćemo u daljem tekstu koristiti sva tri nivoa apstrakcije, već prema potrebi, a nastojaćemo da uvek bude jasno koji nivo apstrakcije je u pitanju.

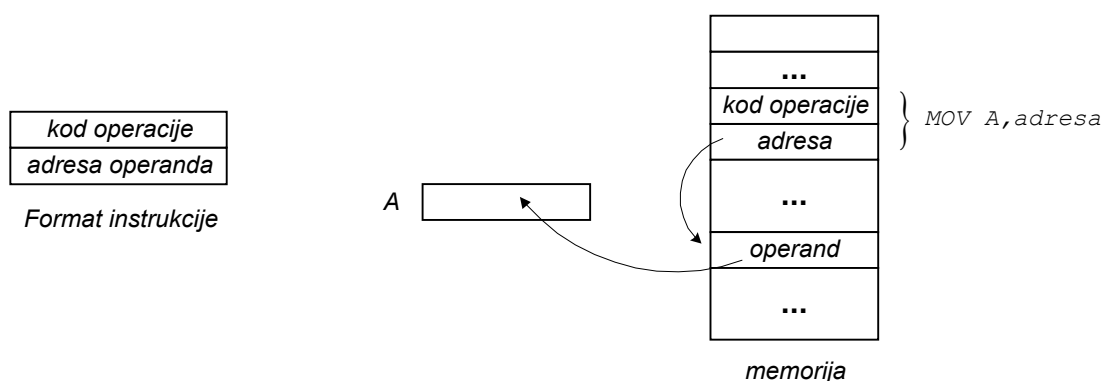
Apstraktni opis instrukcije *ADD A,adresa* izgleda ovako:

```
A ← A + M[adresa] ; sabiranje sadržaja A sa operandom koji je u
                  ; memoriji i stavljanje rezultata u A
```

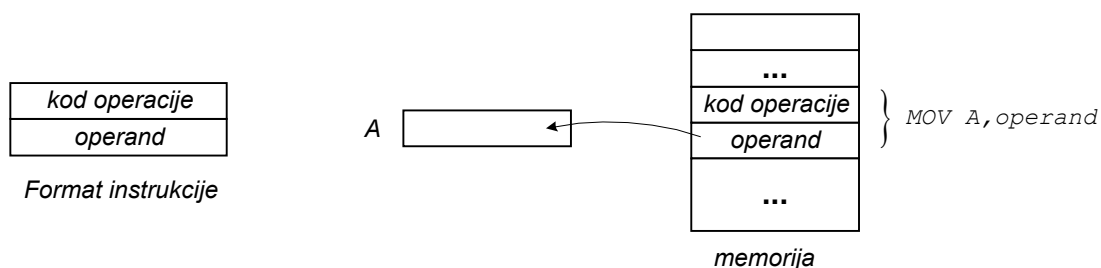
1.6 Načini adresiranja

Jedan broj instrukcija manipuliše podatke, koje nazivamo operandi, a koji se nalaze u memoriji ili registrima centralnog procesora. Na primer, instrukcija može da prenese operand sa jednog mesta na drugo, da sabira dva operanda ili da pomeri bite operanda za jedno mesto ulevo. Postoje različiti načini izračunavanja adrese operanda, a u ovom odeljku ćemo videti četiri načina adresiranja koje podržava Edulent.

Do sada smo videli jedan način adresiranja, *direktno adresiranje*, kod koga je adresa operanda sastavni deo instrukcije. Slika 1.16 ilustruje direktno adresiranje na primeru instrukcije *MOV A,adresa*. Format instrukcije prikazan je na slici levo, a grafička ilustracija na slici desno. Centralni procesor uzima adresu operanda, koja je sastavni deo instrukcije, pristupa memorijskoj lokaciji sa tom adresom i operand iz memorijske lokacije prenosi u akumulator.



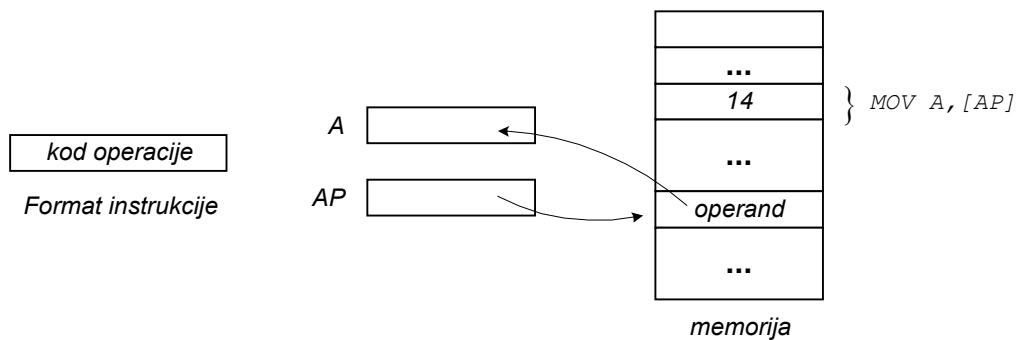
Slika 1.16: Ilustracija direktnog adresiranja



Slika 1.17: Ilustracija neposrednog adresiranja

Kod *neposrednog adresiranja* operand je sastavni deo instrukcije. Kao što pokazuje Slika 1.17 na primeru instrukcije *MOV A,operand*, operand koji se nalazi posle koda operacije, prenosi se iz memorije u akumulator.

Za *registarsko indirektno* adresiranje neophodan je registar u kome je smeštena adresa operanda. Na primer, Edulent ima registar AP (Address Pointer), u kome se pre izvršenja instrukcije mora smestiti adresa operanda. Slika 1.18 ilustruje kako se, u toku izvršenja instrukcije *MOV A, [AP]* sadržaj registra AP koristi kao adresa memorijske lokacije iz koje se operand prenosi u akumulator A.

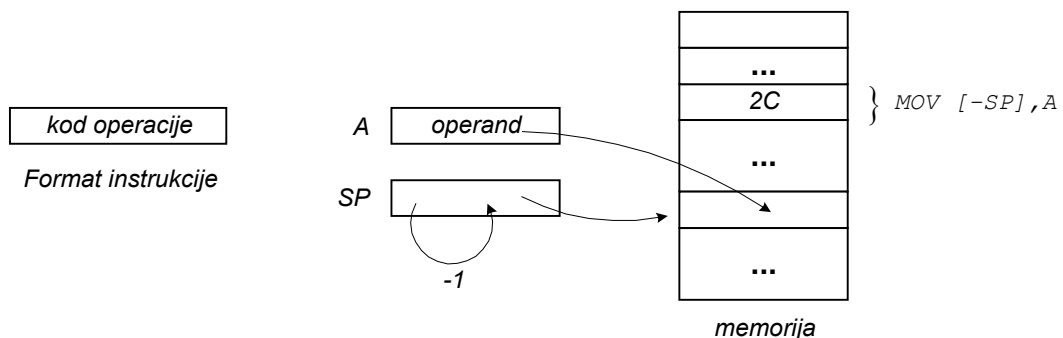


Slika 1.18 : Ilustracija registarskog indirektnog adresiranja

Pošto je neophodno adresni registar inicijalizovati i podešavati njegov sadržaj, skup instrukcija mora da podržava rad sa adresnim registrom. Na prime, Edulent ima sledeće instrukcije za rad sa registrom AP:

```
MOV AP,const      ; punjenje AP, neposredn adresiranje
MOV AP,adresa     ; punjenje AP, direktno adresiranje
MOV adresa,AP     ; prenos AP u memoriju, direktno adresiranje
```

Registarsko indirektno adresiranje modifikuje se tako što adresni registar može da se automatski inkrementira ili dekrementira u toku izvršenja instrukcije. Edulent za ovaj način adresiranja koristi specijalizovani registar SP (Stack Pointer). *Registarsko indirektno adresiranje sa predekrementom* ilustruje Slika 1.19 na primeru instrukcije `MOV [-SP], A`. Sadržaj registra SP prvo se smanjuje za 1, a zatim se u memorijsku lokaciju čija adresa je u SP stavlja sadržaj akumulatora.

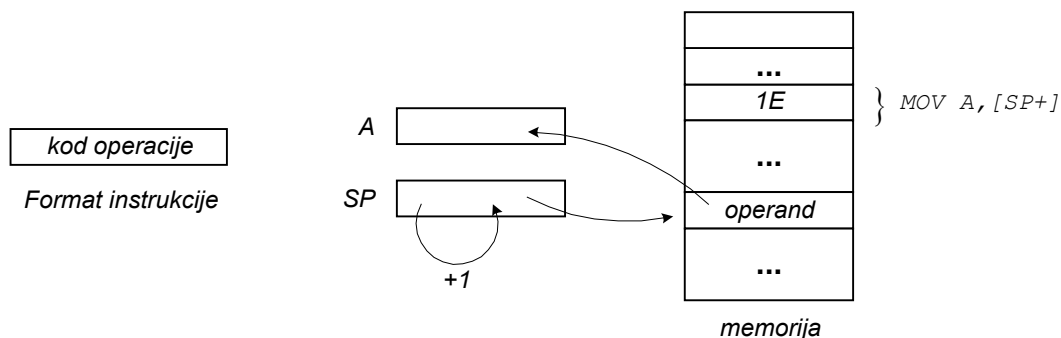


Slika 1.19: Ilustracija registarskog indirektnog adresiranja sa predekrementom

Apstraktni opis izvršne faze instrukcije `MOV [-SP], A` izgleda ovako:

```
SP ← SP - 1      ; smanjenje sadržaja SP za 1
M[SP] ← A        ; prenos sadržaja A u lokaciju sa adresom SP
```

Registarsko indirektno adresiranje sa postinkrementom ilustruje Slika 1.20. Kod ovog načina adresiranja prvo se sadržaj memorijske lokacije, čija adresa je u registru SP, prenese u akumulator, a zatim sadržaj SP poveća za 1.



Slika 1.20: Ilustracija registarskog indirektnog adresiranja sa postinkrementom

Apstraktni opis izvršne faze instrukcije $MOV\ A, [SP+]$ obuhvata sledeće dve mikrooperacije:

$A \leftarrow M[SP]$; prenos sadržaja lokaciju sa adresom SP u akumulator A
 $SP \leftarrow SP+1$; povećanje sadržaja SP za 1

Instrukcija $MOV\ [-SP], A$ koja sadržaj akumulatora prenosi u memoriju adresiranu registrom SP na engleskom se naziva 'push' i kod nekih assemblera (na primer kod simboličkog mašinskog jezika za mikroprocesor Intel 80x86) označava sa $PUSH\ A$. Slično se instrukcija $MOV\ A, [SP+]$ koja prenosi sadržaja memorijske lokacije iz memorije adresirane registrom SP u akumulator naziva 'pop' i označava sa $POP\ A$. Ove dve instrukcije obično se koriste u paru, tako da za svaku 'push' instrukciju postoji odgovarajuća 'pop' instrukcija.

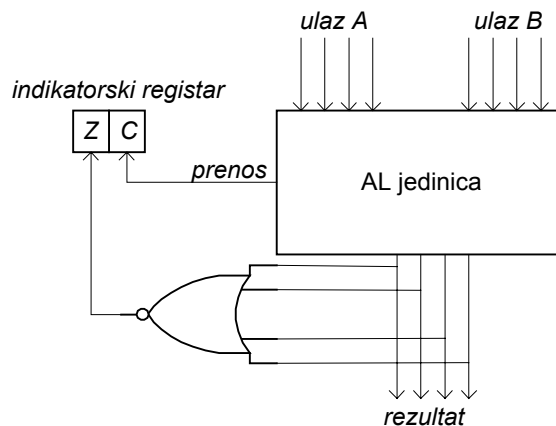
Treba obratiti pažnju da su izvršne faze ovih dveju instrukcija slične i da se odnose jedna naprama drugoj kao u ogledalu: instrukcija 'push' prvo smanjuje sadržaj SP i zatim prenosi sadržaj akumulatora u memorijsku lokaciju čija adresa je u SP, dok 'pop' prvo prenese sadržaj lokacije čija je adresa u SP iz memorije u akumulatora i onda poveća sadržaj SP za 1. Deo operativne memorije koji adresira SP naziva se *stek memorija*. Prema tome, 'push' instrukcija stavlja podatak u stek memoriju, a 'pop' uzima podatak iz stek memorije.

1.7 Indikatorski registar

U toku izvršenja nekih instrukcija, kao što su na primer aritmetičke instrukcije, mogu da se dobiju dve vrste informacija od interesa. Prvo, dobija se rezultat izvršenja instrukcije, na primer suma dva operanda. Drugo, može se dobiti informacija o statusu izvršenja instrukcije, na primer da li je rezultat jednak nuli, da li se dogodio prenos u toku izvršenja instrukcije, ili da li je rezultat pozitivan ili negativan.

Indikatorski ili statusni registar koristi se za prihvatanje statusa izvršenja instrukcije. Edulent ima indikatorski registar od svega dva bita, za nulu (Z) prenos (C). Slika 1.21 prikazuje princip generisanja ovih bita u slučaju 4-bitne AL jedinice. Svi izlazni biti rezultata dovode se na ulaze NILI kola koje na svom izlazu daje 1 ako su svi biti rezultata jednaki nuli, a 0 ako rezultat nije jednak nuli. Izlaz NILI kola dovodi se na ulaz Z bita indikatorskog registra.

AL jedinica realizuje se na osnovu paralelnog sabirača koji ima ulazni i izlazni prenos. Izlazni prenos paralelnog sabirača dovodi se na ulaz C bita indikatorskog registra.



Slika 1.21: Generisanje bita za prenos (C) i nulu (Z) indikatorskog registra

Indikatorski registar može imati više bita, na primer za indikaciju da li je rezultat pozitivan, da li se dogodilo prekoračenje i tako dalje. Sadržaj indikatorskog registra koristi se pre svega u instrukcijama uslovnog programskog skoka. Primeri instrukcija uslovnog programskog skoka dati su u narednom odeljku.

1.8 Skup instrukcija Edulenta

Instrukcije prenosa podataka obuhvataju prenos podataka između memorije i registara A i AP, pri čemu se koriste sva četiri načina adresiranja. Slede instrukcija prenosa zajedno sa apstraktnim opisom njihovih izvršnih faza.

```

MOV A,adresa    ; A ← M[adresa]
MOV AP,adresa   ; AP ← M[adresa]
MOV A,const     ; A ← const
MOV AP,const    ; AP ← const
MOV adresa,A    ; M[adresa] ← A
MOV adresa,AP   ; M[adresa] ← AP
MOV A,[AP]      ; A ← M[AP]
MOV A,[SP+]     ; A ← M[SP], SP ← SP+1
MOV AP,[SP+]    ; AP ← M[SP], SP ← SP+1
MOV [-SP],A     ; SP ← SP-1, M[SP] ← A
MOV [-SP],AP    ; SP ← SP-1, M[SP] ← AP

```

Edulent ima samo instrukcije sabiranja i oduzimanja:

```

ADD A,adresa    ; A ← A + M[adresa]
ADD A,const     ; A ← A + const
ADD AP,const    ; AP ← AP + const
ADD A,[AP]      ; A ← A + M[AP]
SUB A,adresa    ; A ← A - M[adresa]
SUB A,const     ; A ← A - const
SUB AP,const    ; AP ← AP - const
SUB A,[AP]      ; A ← A - M[AP]

```

Logičke i pomeračke instrukcije:

```
NOT           ;  $A \leftarrow \neg A$  (zamena 0 sa 1 i 1 sa 0)
OR adresa    ;  $A \leftarrow A \vee M[\textit{adresa}]$ 
OR const     ;  $A \leftarrow A \vee \textit{const}$ 
OR [AP]      ;  $A \leftarrow A \vee M[AP]$ 
AND adresa   ;  $A \leftarrow A \wedge M[\textit{adresa}]$ 
AND [AP]     ;  $A \leftarrow A \wedge M[AP]$ 
AND const    ;  $A \leftarrow A \wedge \textit{const}$ 
XOR adresa   ;  $A \leftarrow A \oplus M[\textit{adresa}]$ 
XOR const    ;  $A \leftarrow A \oplus \textit{const}$ 
XOR [AP]     ;  $A \leftarrow A \oplus M[AP]$ 
SHR           ;  $A \leftarrow$  prethodni sadržaj  $A$  pomeren za 1 mesto udesno
```

Edulent ima tri instrukcije grananja ili programskog skoka. Instrukcija bezuslovnog programskog skoka je:

```
JMP adresa    ;  $PC \leftarrow \textit{adresa}$ 
```

Lista mikroinstrukcija ove instrukcije izgleda ovako:

```
MA  $\leftarrow$  PC      ; prenos adrese instrukcije u MA
MD  $\leftarrow$  M[MA]   ; prenos koda operacije iz memorije u MD
PC  $\leftarrow$  PC+1    ; povećanje sadržaja PC za 1
IR  $\leftarrow$  MD      ; prenos koda operacije u IR

MA  $\leftarrow$  PC      ; prenos adrese narednog bajta u MA
MD  $\leftarrow$  M[MA]   ; prenos adrese programskog skoka iz memorije u MD
PC  $\leftarrow$  PC+1    ; povećanje sadržaja PC za 1
PC  $\leftarrow$  MD      ; prenos adrese programskog skoka iz MD u PC
```

U toku izvršne faze adresa programskog skoka prenosi se iz registra MD u programski brojač PC. Mikroprocesor će po završetku ove instrukcije izvršiti instrukciju čija adresa je u PC, a to je instrukcija sa adresom koja je navedena u JMP instrukciji. Naravno, mikroinstrukcija $PC \leftarrow PC+1$ zapravo nije neophodna, ona je stavljena samo da bi ilustrovala ideju da se adresa naredne instrukcije koja je u PC zamenjuje adresom instrukcije programskog skoka.

Apstraktni opis izvršne faze instrukcije programskog skoka je jednostavan:

```
PC  $\leftarrow$  adresa ; stavi u PC adresu programskog skoka
```

Instrukcije uslovnog grananja ispituju odgovarajuće uslove u indikatorskom registru i u zavisnosti od ispunjenja tih uslova u programski brojač stavljaju adresu programskog skoka:

```
JZ adresa     ; ako je Z=1 u PC stavi adresu programskog skoka
JC adresa     ; ako je C=1 u PC stavi adresu programskog skoka
```

U sledećem programu, koji sabira prvih n celih brojeva, ilustrovana je primena instrukcija bezuslovnog i uslovnog programskog skoka.

```

PROGRAM "Sabiranje prvih n celih brojeva"
DATA
    n    DB    0x3    ; poslednji broj koji treba sabrati
    rez  DB    0x0    ; mesto za rezultat
ENDDATA

CODE
    MOV A,0x0    ; pocetni sadraj akumulatora
    MOV AP,n     ; AP kontrolise petlju
Lab1: ADD A,n     ; sabiranje
    SUB AP,0x1   ; sledeci broj koji treba sabrati
    JZ  Lkraj    ; kraj ako je AP=0, uslovni skok na Lkraj
    MOV n,AP     ; smanjenje n za 1
    JMP Lab1     ; bezuslovan programski skok
Lkraj: MOV rez,A ; smestanje rezultata u lokaciju rez
    END
ENDPROGRAM

```

Edulent ima dve instrukcije za rad sa procedurama (potprogramima). Osnovni problem kod instrukcija *CALL* za poziv potprograma je adresa povratka, zato što negde treba zapamtiti adresu na koju programska kontrola treba da se vrati kada se završi potprogram. Jednostavno rešenje koje se koristi u gotovo svim digitalnim računarima sastoji se u korišćenju steka za čuvanje adrese povratka.

Po završetku potprograma programska kontrola treba da se vrati na prvu instrukciju koja sledi posle instrukcije *CALL* kojom se poziva potprogram. Pošto se adresa naredne instrukcije nalazi u programskom brojaču PC, sadržaj PC stavlja se na stek, a u PC se stavlja adresa potprograma. Prema tome, apstraktni opis (izvršne faze) instrukcije *CALL adresaproc* izgleda ovako:

```

temp ← M[PC]      ; prenos adrese potprograma u prihvatni registar
PC ← PC + 1       ; PC sada sadrzi adresu povratka
SP ← SP - 1       ; smanjenje sadržaja SP za 1
M[SP] ← PC        ; stavljanje adrese povratka iz PC na stek
PC ← temp         ; prenos adrese potprograma u PC

```

Naravno, realizacija ove instrukcije zavisi od konkretnog mikroprocesora. U suštini, kada se završi pripremna faza, programski brojač pokazuje na memorijsku lokaciju u kojoj se nalazi adresa potprograma. Mikroprocesor uzima ovu adresu, privremeno je smešta u neki prihvatni registar (na primer u Edulentu to može biti registar R ili MD, pa čak i AP) i povećava sadržaj programskog brojača PC za 1. U ovom trenutku PC pokazuje na instrukciju koja sledi neposredno posle instrukcije *CALL*, što znači da je u programskom brojaču PC adresa povratka. Ova adresa se stavlja na stek (sa predekrementom) i u PC se prenosi adresa potprograma iz prihvatnog registra. Jasno je da će sledeća instrukcija koju mikroprocesor izvrši biti prva instrukcija potprograma zato što je njena adresa u programskom brojaču PC.

Instrukcija *RET* povratka u glavni program je slična instrukciji 'pop' koja adresu povratka sa steka vraća u programski brojač PC. Apstraktni opis izvršne faze instrukcije *RET* izgleda ovako:

```

PC ← M[SP]      ; prenos adrese povratka sa steka u PC
SP ← SP + 1     ; povećanje SP za 1

```


Sledeći program ilustruje korišćenje instrukcija poziva potprograma i povratka u glavni program na primeru potprograma za sabiranje prvih n celih brojeva.

```
PROGRAM "Korisćenje potprograma za sabiranje prvih n celih brojeva"
```

```
DATA
```

```
    n    DB    0x3    ; poslednji broj koji treba sabrati  
    rez  DB    0x0    ; mesto za rezultat  
    tmp  DB    0x0    ; promenljiva koju koristi potprogram
```

```
ENDDATA
```

```
CODE
```

```
    MOV A,n    ; ulazni parametar potprograma je u A  
    CALL Saberi  
    MOV rez, A ; stavljanje rezultata u rez  
    END
```

```
PROCEDURE Saberi ; potprogram
```

```
    MOV tmp,A ; stavljanje ulaznog parametra u tmp  
    MOV A,0x0 ; pocetni sadraj akumulatora  
    MOV AP,tmp ; AP kontrolise petlju
```

```
Lab1:  ADD A,tmp ; sabiranje  
        SUB AP,0x1 ; sledeci broj koji treba sabrati  
        JZ  Lkraj ; kraj ako je AP=0, uslovni skok na Lkraj  
        MOV tmp,AP ; smanjenje tmp za 1  
        JMP Lab1 ; bezuslovan programski skok  
Lkraj:  RET      ; povratak u glavni program, rezultat je u A
```

```
ENDPROCEDURE
```

```
ENDPROGRAM
```

2. Programerski model

Sasvim je jasno da programer, koji piše programe u mašinskom ili simboličkom mašinskom jeziku, mora da poznaje skup instrukcija mikroprocesora i načine adresiranja. Sa druge strane, programer u nisu od interesa neki drugi detalji, kao što je vrednost napona napajanja ili vrsta kućišta u koji je mikroprocesor smešten. Programerski model mikroprocesora obuhvata one karakteristike mikroprocesora koje su neophodne za pisanje programa u mašinskom ili simboličkom mašinskom jeziku. U ovom poglavlju, na primeru komercijalnog mikroprocesora Intel x86, izložen je programerski model jednog tipičnog mikroprocesora.

2.1 Elementi programerskog modela

Programerski model mikroprocesora obuhvata sledeća elemente:

- registri,
- načini adresiranja,
- skup instrukcija i
- formati podataka.

Skup registara obuhvata sve registre mikroprocesora koji su vidljivi programeru. Pojam 'vidljivi' odnosi se na registre kojih programer mora da bude svestan u toku pisanja programa. Tipičan primer su registri opšte namene kojima mikroprocesor pristupa u toku izvršavanja instrukcija u kojima programer eksplicitno navodi imena registara. Naravno, mikroprocesor može da pristupa registrima koji nisu eksplicitno navedeni u instrukciji, kao što je to slučaj, na primer, kod rada sa stekom. Instrukcije za rad sa stekom ne navode ime pokazivača steka, ali ipak programer mora da zna da te instrukcije menjaju sadržaj tog registra.

Načini adresiranja su važni kod pisanja programa koji rade sa složenim strukturama podataka, na primer sa jedno- ili više-dimenzijskim nizovima. Savremeni mikroprocesori dozvoljavaju da jedna instrukcija koristi različite načine adresiranja i time efektivno uvećavaju raznolikost skupa instrukcija.

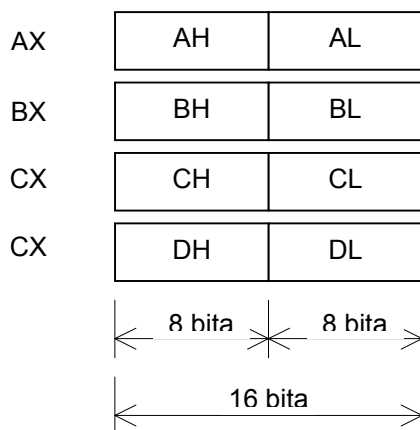
Imajući u vidu da se jedna operacija, kao što je na primer sabiranje, može modifikovati tako što se koriste različiti načini adresiranja, onda se postavlja pitanje da li se tom modifikacijom dobija nova instrukcija ili se radi o istoj instrukciji. Obično se podrazumeva da postoji osnovni skup instrukcija, a da način adresiranja daje samo modifikaciju iste instrukcije.

Savremeni mikroprocesori imaju mogućnosti rada sa različitim formatima podataka. Prvi mikroprocesori izvršavali su operacije samo nad celim brojevima. Kako su se mogućnosti mikroprocesora povećavale, tako su dodavani novi formati podataka. Brojevi u pokretnom zarezu uvedeni su sa uvođenjem hardverske jedinice za aritmetičke operacije sa brojevima u pokretnom zarezu. Neki mikroprocesori imaju mogućnost rada sa pojedinačnim bitovima. Operacije sa negativnim brojevima zahtevaju korišćenje drugog komplementa i tako dalje. Budući da jednostavni mikroprocesori rade samo sa celim brojevima, u ovom poglavlju neće se posebno obrađivati formati podataka.

2.2 Registri

Skup registara mikroprocesora obično se deli na registre opšte namene i specijalizovane registre. Ideja uvođenja registara opšte namene zasnovana je na činjenici da je pristup registrima mikroprocesora za red veličine brži nego pristup memorijskim lokacijama. Prema tome, ako se u registre opšte namene smeste često korišćeni podaci, onda će program da se brže izvršava u odnosu na program koji te podatke čuva u memorijskim lokacijama.

Mikroprocesor x86 ima četiri 16-bitna registra opšte namene, AX, BX, CX i DX, Slika 2.1. Svaki od registara podeljen je na viši i niži bajt, koji mogu da se nezavisno adresiraju. Prema tome, oznaka AX koristi se za 16-bitni registar, AL označava donji, a AH gornji bajt registra AX.



Slika 2.1: Registri opšte namene

Prenos sadržaja iz jednog u drugi registar opšte namene često se koristi u programiranju. Na primer, instrukcija:

```
MOV AX, BX
```

sadržaj registra BX preko interne magistrale prenosi u registar AX. Posle izvršenja ove instrukcije, sadržaj registra BX ostaje nepromenjen, novi sadržaj registra AX jednak je sadržaju registra BX, a prethodni sadržaj registra AX je nepovratno izbrisan. Kao i ranije, ovu operaciju označavaćemo sa:

$AX \leftarrow BX$

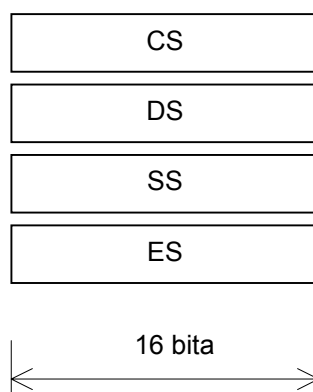
Kod prenosa 8-bitnih reči, programer mora navesti 8-bitne delove registara, odnosno više ili niže bajtove registara opšte namene:

```
MOV AL, BH      ; AL ← BH  
MOV CH, CL      ; CH ← CL
```

Nije dozvoljeno mešati 8- i 16-bitne operande. Na primer, sledeće kombinacije registara dovode do greške:

```
MOV AX, BH      ; AX je 16-bitni, a BH 8-bitni registar  
MOV CH, CX      ; CH je 8-bitni, a CX 16-bitni registar
```

Mikroprocesor x86 ima četiri 16-bitna segmentna registra CS, DS, SS i ES, Slika 2.2, koji se isključivo koriste za izračunavanje memorijskih adresa. Registar CS uvek se koristi kod izračunavanja adresa memorijske lokacije u kojoj je smeštena instrukcija koja treba da se izvrši. Registar DS koristi se za izračunavanje adresa memorijskih lokacija u kojima su smešteni podaci, a registar CS kod izračunavanja adresa memorijskih lokacija koje se koriste za stek. Registar ES nema unapred definisanu namenu.



Slika 2.2: Segmentni registri

Mikroprocesor x86 ima memorijski adresni prostor od 1 Mb i koristi 20 bita za adresiranje memorije. Ovaj mikroprocesor koristi poseban postupak u kojem se na osnovu dva cela pozitivna 16-bitna broja, koja se nazivaju *segment* i *odstojanje*, izračunava memorijskih adresa koja ima 20 bita. Postupak se sastoji u tome da se prvo 16 bita segmenta pomere za 4 mesta ulevo pri čemu se najniža 4 bita popune nulama i tako se dobije jedan 20-bitni broj. Odstojanje se proširi sa četiri nule na najznačajnijim mestima i na taj način dobije drugi 20-bitni broj. Dobijeni 20-bitni brojevi se saberu, prenos se zanemari i dobijeni 20-bitni broj koristi se kao 20-bitna memorijska adresa.

Neka je segment 04A2 (heksadecimalno) i neka je odstojanje 0481 (heksadecimalno). U binarnom brojnem sistemu, segment je 0000 0100 1010 0010, a odstojanje 0000 0100 1000 0001. Posle pomeranja segmenta za 4 mesta ulevo, popunjavanja odstojanja sa 4 nule na najznačajnijim mestima i sabiranja dobija se sledeći rezultat:

```
Segment:  0000 0100 1010 0010 0000
Odstojanje: 0000 0000 0100 1000 0001
Suma:      0000 0100 1110 1010 0001
```

Prema tome, dobijena adresa od 20 bita je 04EA1 (heksadecimalno). Pošto je pomeranje za 4 mesta ulevo jednako množenju sa brojem 10 (heksadecimalno), u heksadecimalnom brojnem sistemu operacija izračunavanja memorijske adrese označava se sa:

$$\text{adresa} = (\text{segment} \times 10) + \text{odstojanje}$$

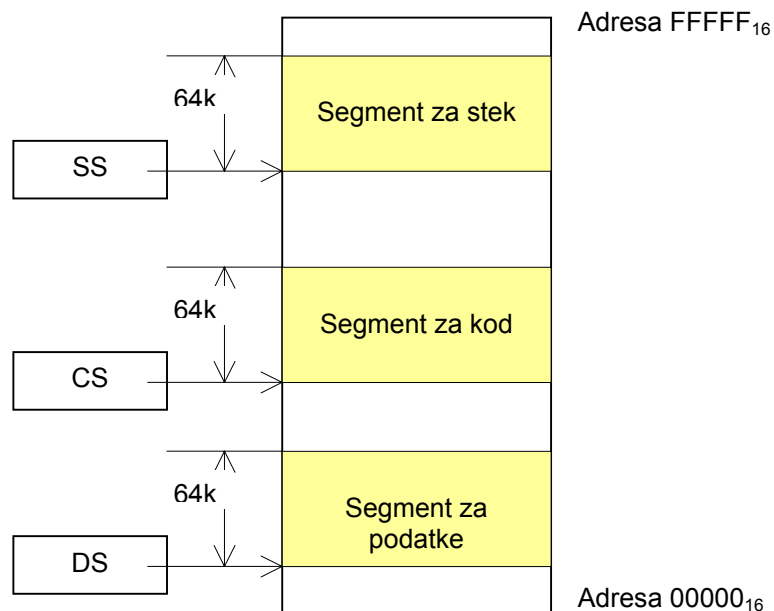
U prethodnom primeru je $04A2 \times 10 + 0481 = 04EA1$, sve u heksadecimalnom brojnem sistemu.

Segmenti registri sadrže segmente koji se koriste kod izračunavanja memorijske adrese. Ako je, na primer, odstojanje u registru BX, onda se adresa računa na sledeći način:

$$\text{adresa} = (\text{CS} \times 10) + \text{BX}$$

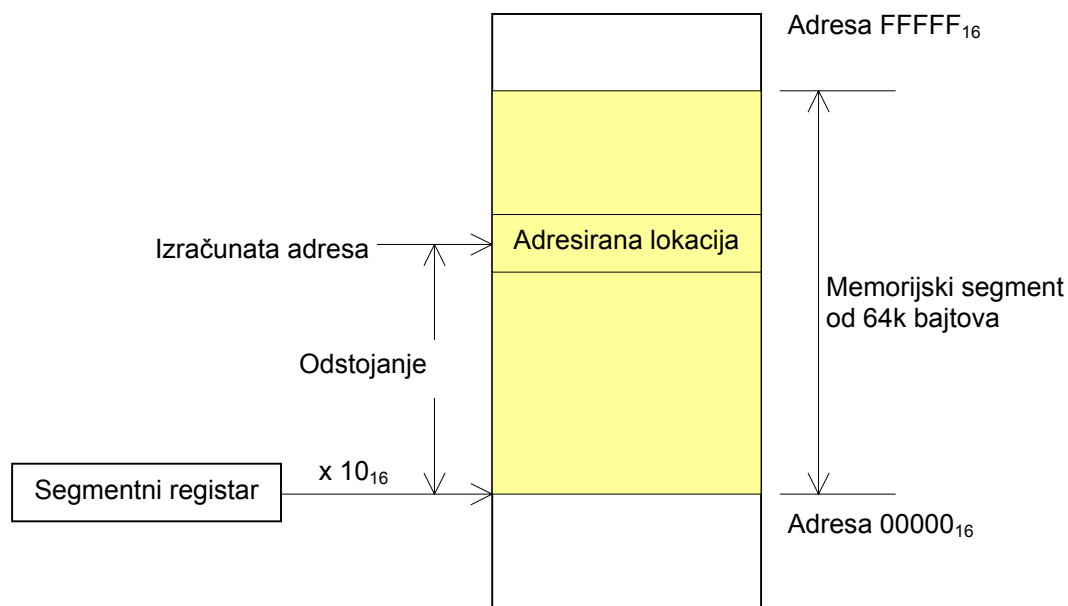
Pošto je odstojanje celi pozitivan 16-bitni broj između 0 i $(64k-1)$ odnosno $FFFF_{16}$, sabiranjem odstojanja i segmenta mogu se izračunati adrese memorijskih lokacija između brojeva $(\text{segment} \times 10_{16})$ i $(\text{segment} \times 10_{16} + FFFF_{16})$. Prema tome, korišćenjem sadržaja jednog segmentnog registra mogu se izračunati adrese memorijskih lokacija u rasponu od 64k. Skup memorijskih lokacija koje se mogu izračunati na osnovu jednog segmeneta naziva se *memorijski segment* ili često samo *segment*. Ne treba da zbunjuje ova terminologija, u kojoj se broj koji se koristi u izračunavanju adresa naziva segment, a skup od 64k memorijskih lokacija takođe naziva segment. Iz konteksta je uvek jasno šta se podrazumeva pod segmentom.

Segmentni registri svojim sadržajem definišu četiri memorijska segmenta: kodni segment (na njega pokazuje registar CS), segment za podatke (DS), segment za stek (SS) i ekstra segment (ES). Segmenti mogu biti razdvojeni, kao što pokazuje Slika 2.3 za tri segmenta, ali pošto nema ograničenja na sadržaje segmentnih registara, memorijski segmenti mogu da se delimično ili potpuno preklapaju.



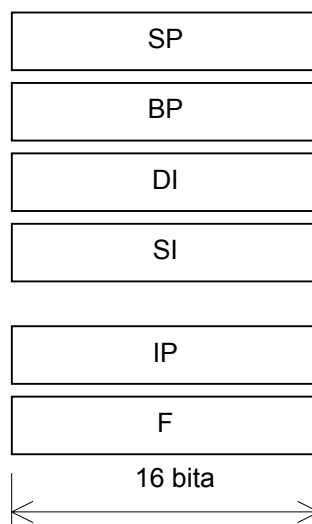
Slika 2.3: Memorijski adresni prostor sa tri segmenta

Slika 2.4 vizuelno prikazuje način izračunavanja memorijske adrese. Sadržaj segmentnog registra množi se sa 10_{16} i dobijena vrednost predstavlja početnu adresu memorijskog segmenta. Na ovu adresu dodaje se vrednost odstojanja i tako dobija adresa memorijske lokacije.



Slika 2.4: Vizuelni prikaz izračunavanju memorijske adrese

Slika 2.5 prikazuje 16-bitne registre posebne namene: pokazivač steka (SP), bazni registar (BP), dva indeksna registra (DI i SI), programski brojač (IP) i indikatorski registar (F).



Slika 2.5: Registri posebne namene

Pokazivač steka koristi se za izračunavanje adrese memorijskih lokacija kod izvršavanja instrukcija koje rade sa stekom. Segment steka uvek je u registru SS, pa je adresa memorijske lokacije na vrhu steka:

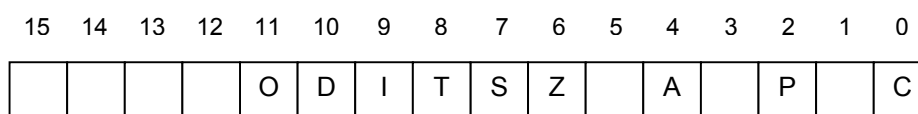
$$\text{Adresa} = \text{SS} \times 10_{16} + \text{SS}$$

Bazni registar BP i indeksni registri DI i SI koriste se kod nekih načina adresiranja koji će biti pokazani u sledećem odeljku.

Programski brojač IP (Instruction Pointer) služi kao odstojanje kod izračunavanja adrese memorijske lokacije u kojoj se nalazi naredna instrukcija koju mikroprocesor treba da izvrši. Segmentni registar CS uvek pokazuje na segment sa kodom (instrukcijama) pa je adresa memorijske lokacije sa narednom instrukcijom:

$$\text{Adresa} = \text{CS} \times 10_{16} + \text{IP}$$

Slika 2.6 prikazuje format indikatorskog registra F (Flags). Od ukupno 16 bita indikatorskog registra koristi se ukupno 9 bita. Za nas su od interesa u ovom trenutku četiri indikatorska bita: prenos (C, Carry, bit broj 0), nula (Z, Zero, 6), znak (S, Sign, 7) i prekoračenje (O, Overflow, 11).

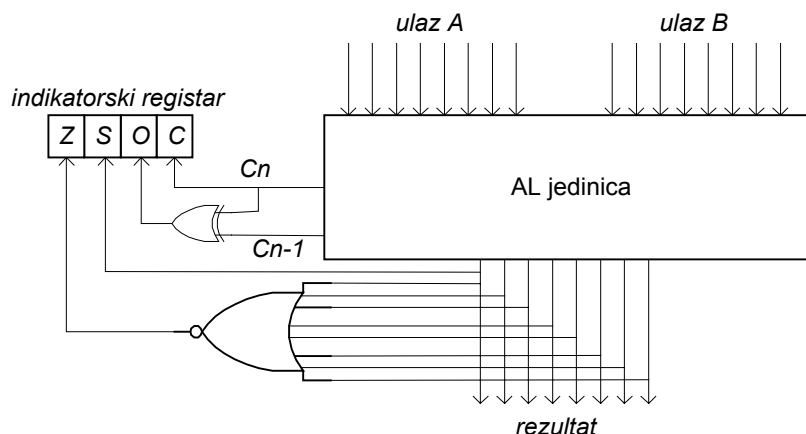


Slika 2.6: Format indikatorskog registra F

Indikatorski bitovi nula (Z) i prenos (C) već su objašnjeni u prethodnom poglavlju. Bit znak (S) koristi se kod aritmetičkih operacija kod kojih su negativni brojevi predstavljeni drugim ili potpunim komplementom. Treba se podsetiti da najznačajniji bit rezultata pokazuje da li je broj pozitivan ili negativan: nula ukazuje na pozitivan, a jedinica na negativan rezultat.

Indikatorski bit prekoračenje (O) pokazuje da li je kod izvođenja aritmetičkih operacija nastupilo prekoračenje, odnosno da li je kod sabiranja 8- (ili 16-) bitnih brojeva dobijen rezultat koji ima više od 8 (ili 16) bita. Očigledno je da prekoračenje može nastupiti samo u dva slučaja: sabiranjem dva pozitivna ili sabiranjem dva negativna broja. U slučaju sabiranja jednog pozitivnog i jednog negativnog broja ne može se dogoditi prekoračenje.

Prekoračenje je najlakše ustanoviti upoređenjem ulaznog i izlaznog bita za prenos na najznačajnijem bitu AL jedinice. Ako su ova dva bita različita, onda je nastupilo prekoračenje, u suprotnom nema prekoračenja. Bit prekoračenje O postavlja se na 1 ukoliko je nastupilo prekoračenje. Slika 2.7 prikazuje način generisanja bita Z, S, O i C u slučaju 8-bitne AL jedinice.



Slika 2.7: Način generisanja indikatorskih bita Z, S, O i C kod 8-bitne AL jedinice

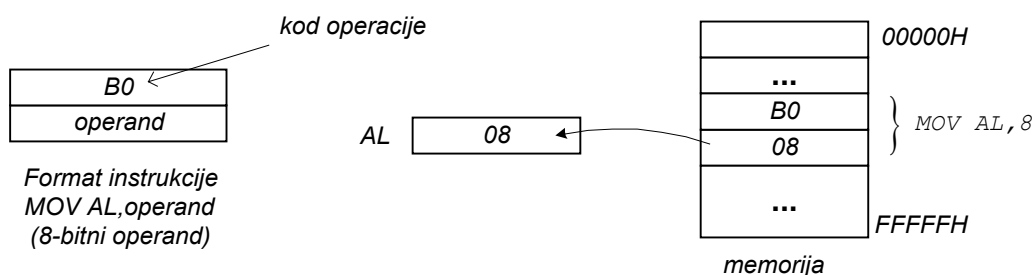
2.3 Načini adresiranja

U toku izvršavanja nekih instrukcija, mikroprocesor izvršava operacije nad podacima, koji se nazivaju *operandi*. Jednostavan primer je instrukcija *MOV* koja operand iz memorijske lokacije ili registra mikroprocesora kopira u drugu memorijsku lokaciju ili drugi registar. Mikroprocesori podržavaju različite načine izračunavanja adrese operandada. Smatra se da je bolji mikroprocesor koji ima više načina izračunavanja adrese operandada zato što je pruža mogućnost efikasnog pristupa različitim strukturama podataka u odnosu na mikroprocesor koji ima skromnije načine izračunavanja adrese operandada. Izračunata adresa operandada naziva se *efektivna adresa*.

U ovom odeljku opisani su osnovni načini adresiranja na primeru instrukcije *MOV*. Opšti oblik ove instrukcije je *MOV X,Y* gde je *Y* memorijska lokacija ili registar u kome se nalazi operand, a *X* je memorijska lokacija ili registar u koji se stavlja kopija operandada.

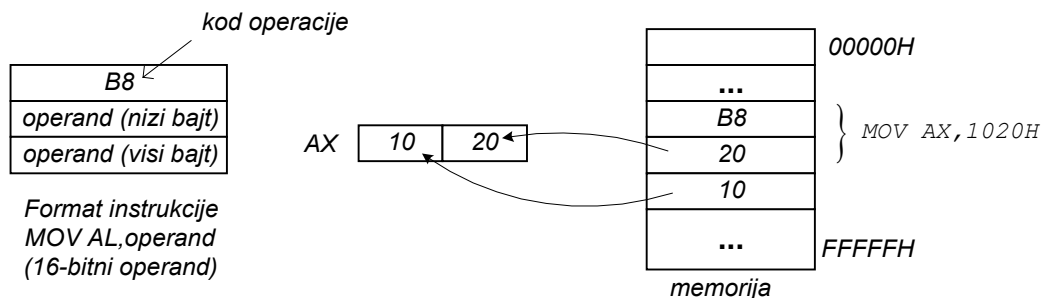
Neposredno adresiranje

Kao što je već rečeno u prethodnom poglavlju, kod neposrednog adresiranja operand je sastavni deo instrukcije. Na primer instrukcija *MOV AL,8* ima kod operacije B0₁₆, pa je prvi bajt instrukcije B0₁₆. Drugi bajt instrukcije je operand 08, tako da ova instrukcija zauzima dva bajta, Slika 2.8. Obratiti pažnju da je na ovoj slici memorija prikazana tako da memorijske adrese rasu nadole.



Slika 2.8: Ilustracija neposrednog adresiranja na primeru instrukcije *MOV AL,8*

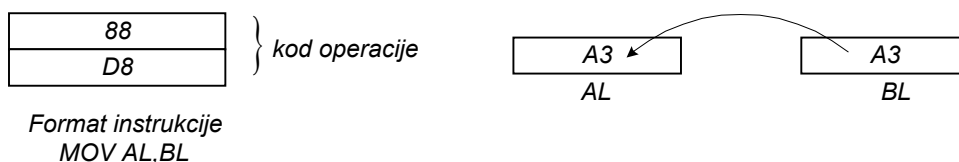
Ista instrukcija može da koristi 16-bitni operand, na primer *MOV AX,1020H*. U ovom slučaju instrukcija zauzima tri bajta: prvi bajt ($B8_{16}$) je kod operacije, drugi je niži bajt operanda (20_{16}), a treći je viši bajt operanda (10_{16}). Slika 2.9. Treba primetiti da iako je mnemonik *MOV* za ovu instrukciju isti za obe dužine operanda, kod operacije u prvom slučaju je $B0_{16}$ a u drugom slučaju $B8_{16}$. Dalje, na primeru instrukcije sa 16-bitnim operandom vidi se da ovaj mikroprocesor niži bajt operanda smešta odmah iza koda operacije, a zatim u sledeću lokaciju smešta viši bajt operanda. Ovaj redosled je kod nekih drugih mikroprocesora obratan.



Slika 2.9: Neposredno adresiranje sa 16-bitnim operandom

Registarsko adresiranje

Kod registarskog adresiranja operand se nalazi u registru centralnog procesora i kopira u drugi (ili isti!) registar centralnog procesora. Na primer, instrukcija *MOV AL,BL* kopiju sadržaja registra BL smešta u registar AL, Slika 2.10, naravno pri tome se prethodni sadržaj registra AL nepovratno gubi. Treba primetiti da kod operacije ove instrukcije, za razliku od prethodnih instrukcija, ima dva bajta.



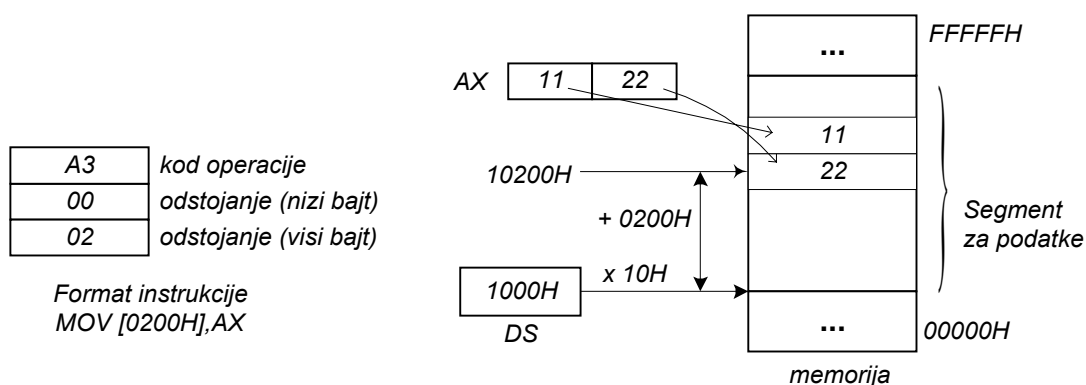
Slika 2.10: Registarsko adresiranje

Mikroprocesor x86 ne dozvoljava neposredno adresiranje sa segmentnim registrima. Dakle, ako hoćemo da postavimo sadržaj segmentnog registra DS na 2030_{16} ne može se koristiti neposredno adresiranje, pa prema tome instrukcija *MOV DS,2030H* nije dozvoljena. U ovom primeru treba koristiti dve instrukcije, prvu koja u jedan od registar opšte namene smešta operand korišćenjem neposrednog adresiranja i drugu koja primenog registarskog adresiranja smešta traženi segment iz registra opšte namene u segmentni registar:

```
MOV AX,2030H      ; AX ← 2030H
MOV DS,AX          ; DS ← AX
```

Direktno adresiranje

Efektivna adresa kod direktnog adresiranja nalazi se u segmentu za podatke, na odstojanju koje je sastavni deo instrukcije. U simboličkom mašinskom jeziku odstojanje se stavlja u uglastu zagradu. Smer prenosa operanda može biti iz memorije u registar (*MOV registar,[odstojanje]*) ili u suprotnom smeru (*MOV [odstojanje], registar*). Na primer, instrukcija *MOV [0200], AX* prenosi 16-bitni sadržaj registra AX u memorijsku lokaciju (od dva bajta) koja se nalazi na efektivnoj adresi.



Slika 2.11: Direktno adresiranje

Efektivna adresa dobija se tako što se sadržaj segmentnog registra DS pomeri za 4 mesta ulevo (isto što i pomnoži sa 10H) i sabere sa odstojanjem, Slika 2.11. U dobijenu 8-bitnu memorijsku lokaciju stavlja se donji bajt registra AX, a u sledeću lokaciju stavlja se sadržaj gornjeg bajta registra AX. U našem primeru, ako je sadržaj registra DS jednak 1000H, pomeranje za 4 binarna mesta ulevo daje 10000H, i kada se sabere odstojanje 0200H dobije se efektivna adresa 10200H. U memorijsku lokaciju sa adresom 10200H smešta se 22H (donji bajt registra AX), a u memorijsku lokaciju sa adresom 10201H smešta se 11H (gornji bajt registra AX). Operaciju instrukcije *MOV [0200], AX* opisujemo na sledeći način:

$$M[DS \times 10H + 0200H] \leftarrow AL \quad (\text{donji bajt registra AX})$$

$$M[DS \times 10H + 0200H + 1] \leftarrow AH \quad (\text{gornji bajt registra AX})$$

Radi jednostavnosti često ovaj opis skraćeno pišemo u obliku:

$$M[DS \times 10H + 0200H] \leftarrow AX$$

U prethodnom izrazu podrazumeva se prenos 16-bitnog operanda jer je naveden 16-bitni registar AX i podrazumeva se redosled smeštanja donjeg i gornjeg bajta registra AX u 8-bitne memorijske lokacije. Naravno, sve je jednostavnije u slučaju 8-bitnog operanda. Na primer, instrukcija *MOV CH, [0200]* operand od 8 bita, koji se nalazi na efektivnoj adresi u memoriji, prenosi u registar CH:

$$CH \leftarrow M[DS \times 10H + 0200H]$$

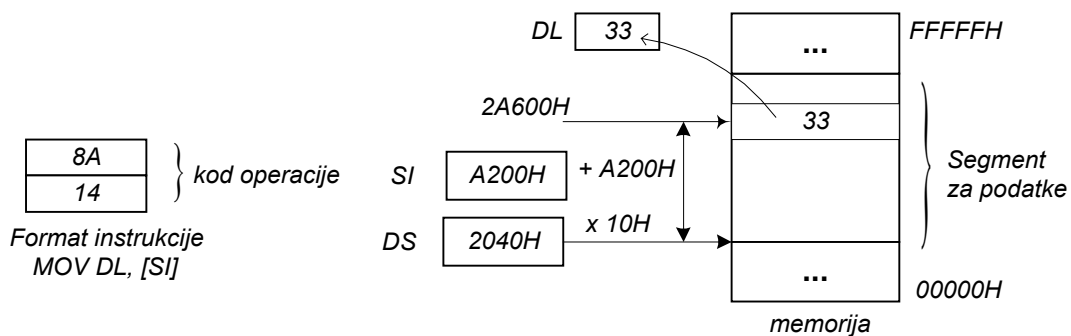
U praksi je dosta teško određivati odstojanja operandada koji se nalaze u memoriji. Ovaj problem rešava se tako što se u programima napisanim u simboličkom mašinskom jeziku koriste labela, a onda se prepusti assembleru da na osnovu labela izračuna odstojanja. Na primer, ako je 8-bitni operand u memorijskog lokaciji koja je označena labelom *PODI*, onda je instrukcija za prenos ovog operanda u registar CH:

$$MOV \ CH, \ PODI$$

Svakako je ovaj način direktnog adresiranja mnogo pogodniji i zato se preporučuje korišćenje labela kod direktnog adresiranja.

Registarsko indirektno adresiranje

Jedina razlika između direktnog i registarskog indirektnog adresiranja je u tome što je kod prvog načina adresiranja odstojanje sastavni deo instrukcije, a u drugom je odstojanje u jednom od četiri registra: BX, BP, SI ili DI. Tako instrukcija *MOV DL, [SI]* prenosi u registar DL sadržaj memorijske lokacije koja je na odstojanju SI u segmentu za podatke, Slika 2.12.



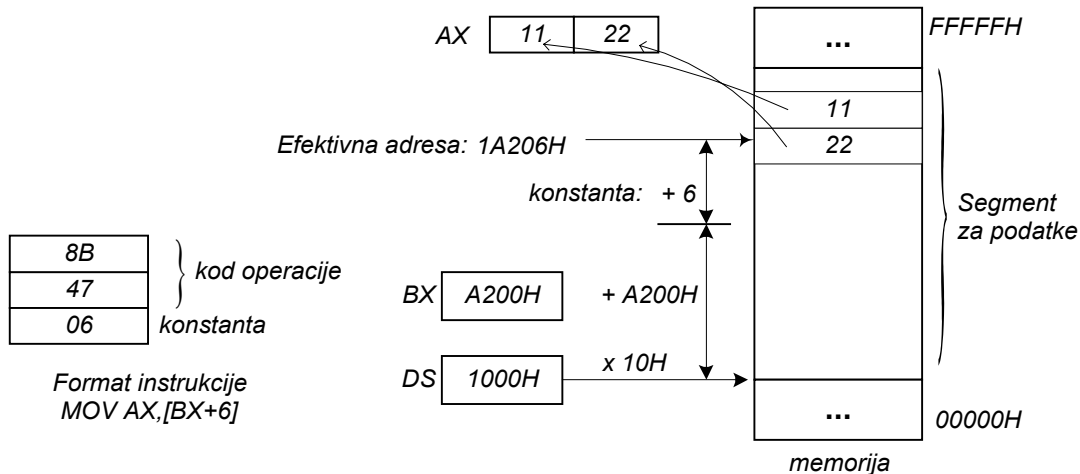
Slika 2.12: Registarsko indirektno adresiranje

U primeru sa slike, sadržaj registra DS je 2040H, a registra SI je A200H. Kada se sadržaj DS pomeri za 4 binarna mesta ulevo dobije se 20400H, ovaj broj se sabere sa odstojanjem iz SI (A200H) i dobije efektivna adresa 2A600H. Ako je sadržaj memorijske lokacije sa ovom adresom 33H, onda se u registar DL upisuje 33H.

Registarsko indirektno adresiranje pogodno je kod pristupa podacima koji su sekvencijalno smešteni u memoriji. Povećavanjem sadržaja registra koji sadrži odstojanje dobijaju se odstojanja uzastopnih podataka u segmentu za podatke.

Bazno adresiranje

Bazno adresiranje dobija se modifikacijom registarskog indirektnog adresiranja tako što se odstojanje iz registra sabere sa 8- ili 16-bitnom konstantom, Slika 2.13.



Slika 2.13: Bazno adresiranje sa 8-bitnom konstantom

Bazno adresiranje ima oblik *MOV registar, [registar + konstanta]* ili *MOV, [registar + konstanta]*, registar pri čemu postoji ograničenje da samo registri BX i BP mogu da se koriste za odstojanje. Na primer, kod izvršavanja instrukcije *MOV AX, [BX + 6]* efektivna adresa dobija se tako što se sadržaj registra DS pomeri 4 binarna mesta ulevo, sabere sa sadržajem registra BX i dobijeni rezultat sabere sa konstantom 6. Iz memorijske lokacije sa ovom adresom uzima se 8-bitni podatak i smešta u registar AL, a iz sledeće memorijske lokacije sadržaj se smešta u registar AH.

Ako registar DS sadrži 1000H, a registar BX sadrži A200H, onda se efektivna adresa dobija na sledeći način:

$$\text{Efektivna adresa} = 1000\text{H} \times 10\text{H} + \text{A200H} + 6 = 1\text{A206H}$$

Izvršavanje instrukcije *MOV AX,[BX + 6]* formalno može da se opiše na sledeći način:

$$\begin{aligned}AL &\leftarrow M[DS \times 10H + A200H + 6] \\AH &\leftarrow M[DS \times 10H + A200H + 6 + 1]\end{aligned}$$

Jednostavnije:

$$AL \leftarrow M[DS \times 10H + A200H + 6]$$

Umesto 8-bitne konstante, bazno adresiranje može da koristi 16-bitnu konstantu. Konstante se interpretiraju kao celi brojevi koji mogu biti pozitivni ili negativni, pri čemu su negativni brojevi predstavljeni svojim drugim komplementom.

Treba imati u vidu da ako se kod baznog adresiranja registar BP koristi za odstojanje, onda se za računanje efektivne adrese koristi segment iz registra SS (umesto registra DS).

Indeksno adresiranje

Indeksno adresiranje potpuno je identično baznom adresiranju, samo što se umesto baznih registara BX i BP koriste indeksni registri SI i DI. Efektivna adresa računa se uvek sa segmentom iz registra DS.

Bazno indeksno adresiranje

Kombinovanjem baznog i indeksnog adresiranja (bez konstante) dobija se bazno indeksno adresiranje. Ovaj način adresiranja ima jedan od dva sledeća oblika:

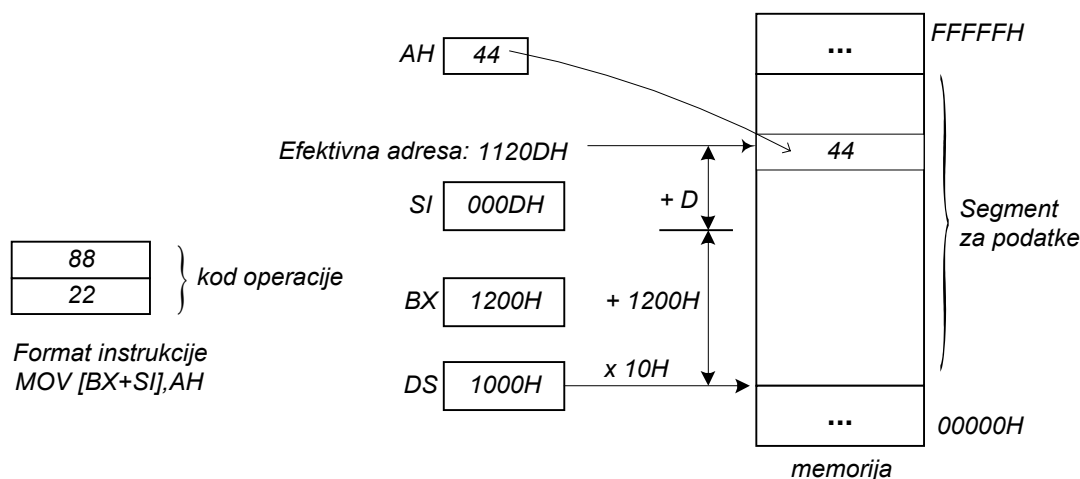
$$\begin{aligned}MOV\ reg1, [reg2 + reg3] \\MOV\ [reg2 + reg3], reg1\end{aligned}$$

Registar *reg2* može biti samo BX ili BP, a registar *reg3* samo registar SI ili DI. Primer instrukcije koja koristi bazno indeksno adresiranje je *MOV [BX+SI],AH*. Efektivna adresa dobija se kada se na početnu adresu segmenta za podatke doda sadržaj registara BX i SI:

$$Efektivna\ adresa = DS \times 10H + BX + SI$$

Slika 2.14 ilustruje primer ove instrukcije kada je DS=1000H, BP=1200H i SI=000DH. Efektivna adresa je:

$$Efektivna\ adresa = 10000H + 1200H + 000DH = 1120DH$$



Slika 2.14: Bazno indeksno adresiranje

Formalno, instrukcija *MOV [BX+SI],AH* obavlja sledeću operaciju:

$$M[DS \times 10H + BX + SI] \leftarrow AH$$

U našem primeru sadržaj registra AH, koji je 44H, smešta se u memorijsku lokaciju sa adresom 1120DH.

Ukoliko bazno indeksno adresiranje koristi registar BP, efektivna adresa računa se tako što se umesto segmentnog registra DS koristi registar SS.

Bazno indeksno adresiranje pogodno je za pristup dvodimenzionalnim nizovima podataka koji su sekvencijalno smešteni u memoriju. Na primer, dvodimenzionalna matrica može se smestiti u memoriju tako da se prvo smeste svi elementi prve vrste, zatim svi elementi druge vrste i tako dalje. Ako u registar BX stavimo brojač vrsta, a u registar SI brojač kolona, onda podešavanjem sadržaja registra BX određujemo vrstu kojoj se pristupa, a inkrementiranjem registra SI pristupa se elementima izabrane vrste.

Bazno indeksno adresiranje sa dodavanjem konstante na efektivnu adresu

Ovaj način adresiranja je u stvari opšti oblik baznog indeksnog adresiranja:

```
MOV reg1, [reg2 + reg3 + konstanta]
MOV [reg2 + reg3 + konstanta], reg1
```

Kao i ranije, *reg2* može da bude BX ili BP, a *reg3* SI ili DI. Ako se koristi BX, efektivna adresa je:

$$\text{Efektivna adresa} = DS \times 10H + BX + SI + \text{konstanta}$$

U slučaju da se koristi BP, efektivna adresa je:

$$\text{Efektivna adresa} = SS \times 10H + BX + SI + \text{konstanta}$$

Direktno adresiranje kod programskog skoka

Programski skok je instrukcija koja menja sekvencijalno izvršavanje instrukcija. Pošto mikroprocesor x86 koristi registre CS i IP za određivanje efektivne adrese instrukcije koju treba izvršiti, programski skok menja sadržaje jednog ili oba ova registra.

Ukoliko instrukcija programskog skoka menja samo sadržaj registra IP, onda se to naziva *kratki programski skok* (*near jump*). Kako sadržaj registra CS ostaje nepromenjen, kratki programski

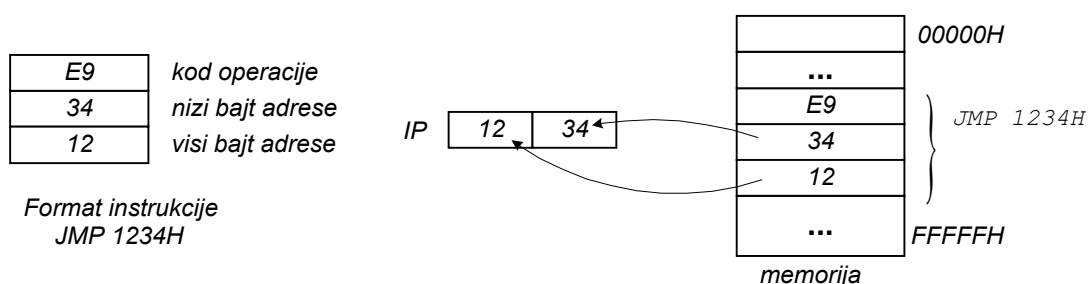
skok može da promeni redosled izvršavanja instrukcija tako da je sledeća instrukcija unutar istog memorijskog segmenta.

Dugački (udaljeni) programski skok (far jump) menja sadržaje oba registra, CS i IP, tako da sledeća instrukcija koja će se izvršiti može biti bilo gde u memoriji.

Instrukcija kratkog programskog skoka *JMP adresa* obavlja operaciju:

$$IP \leftarrow adresa$$

Ova instrukcija ima tri bajta, gde je prvi bajt kod operacije (E9H), a drugi i treći bajt sadrže odstojanje koje treba smestiti u registar IP. Slika 2.15 ilustruje primer instrukcije *JMP 1234H*. Operand ove instrukcije je heksadecimalni broj 1234H koji se smešta u registar IP. Bez obzira koji je bio prethodni sadržaj IP, sledeća instrukcija koju će mikroprocesor izvršiti nalazi se na odstojanju 1234H u kodnom segmentu na koji pokazuje registar CS.



Slika 2.15: Kratki programski skok

U pisanju programa umesto adrese programskog skoka navode se labele, na primer *JMP petlja* ili *JMP kraj*. Asembler zamenjuje imena labela odstojanjem pridružene memorijske lokacije i na taj način olakšava pisanje programa.

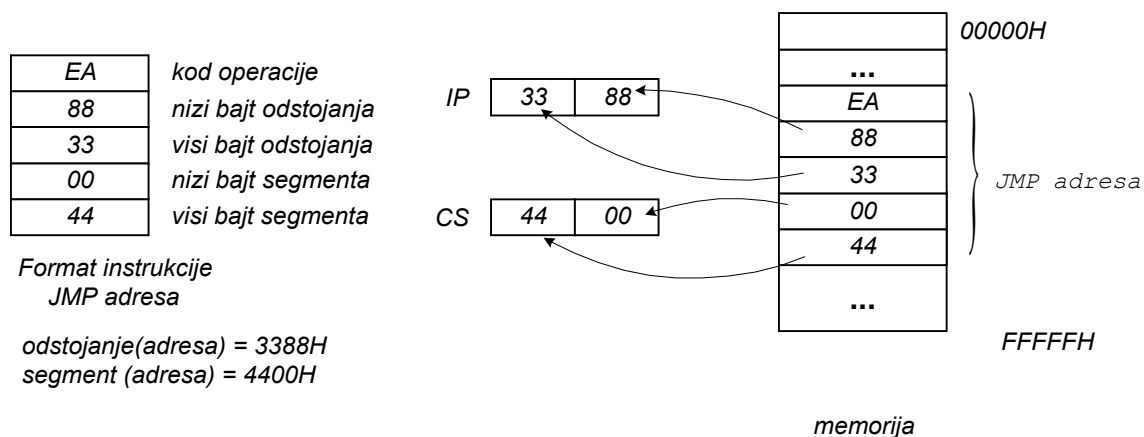
Instrukcija koja koristi dugački skok na sličan način koristi labele. Asembler, naravno, mora da razlikuje labele koje se nalaze unutar segmenta od labela koje se nalaze u nekom drugom memorijskom segmentu. Ukoliko to nije drugačije urađeno, programer može eksplicitno da labele za dugačke programske skokove označi direktivom *FAR*.

Instrukcija dugačkog programskog skoka *JMP adresa* obavlja operaciju:

$$IP \leftarrow odstojanje(adresa)$$

$$CS \leftarrow segment(adresa)$$

Ova instrukcija ima pet bajtova: prvi bajt je kod operacije (EAH), drugi i treći bajt sadrže novo odstojanje a četvrti i peti bajt sadrže novu vrednost segmenta. Slika 2.16 ilustruje instrukciju *JMP adresa*, gde je *adresa* labela koja je prodružena segmentu 4400H i odstojanju 3388H. Nakon izvršenja ove instrukcije, novi sadržaj registra IP biće 3388H, a novi sadržaj registra CS 4400H. Prema tome, kad se izvrši ova instrukcija, sledeća instrukcija koja će se izvršiti nalazi se na efektivnog adresi 47388H.



Slika 2.16: Dugački programski skok

Relativno adresiranje kod programskog skoka

Očigledni nedostatak direktnog adresiranja kod programskog skoka je u tome što ako se program premesti na neko drugo mesto u memoriji, moraju se menjati operandi svih instrukcija programskog skoka koje koriste direktno adresiranje. Na primer, uzmimo da neka instrukcija programskog skoka treba da adresira memorijsku lokaciju koja se nalazi osam bajtova od početne instrukcije programa. Ako se program premesti na neko drugo mesto unutar segmenta, onda vrednost odstojanja za ovu instrukciju programskog skoka mora da se promeni. Dalje, ako se program premesti u neki drugi segment, onda mora da se menja i odstojanje i segment instrukcije direktnog programskog skoka.

Jedno rešenje navedenog problema je da se program prevodi svaki put kada se premešta sa jednog mesta u memoriji na drugo. Drugi pristup je korišćenje relativnih adresa, tako da mašinski kod programa ostaje isti bez obzira u kom delu segmenta se program nalazi. Očigledno je da premeštanje programa iz jednog segmenta u drugi nije kritično zato što se pre izvršenja programa podesi vrednost registra CS a oblik programa ostaje nepromenjen.

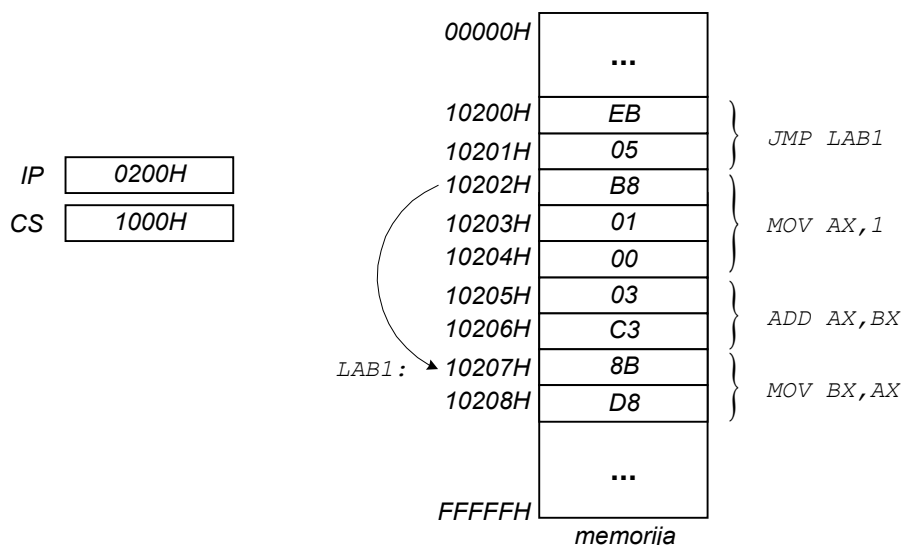
Ideja relativnog adresiranja kod programskog skoka sastoji se u određivanju nove adrese u odnosu na trenutnu vrednost programskog brojača IP. Sastavni deo instrukcije je relativna adresa koja se sabira sa sadržajem programskog brojača i dobijena vrednost smešta u programski brojač. Ako se program premesti na neko drugo mesto u memoriji, sabiranjem relativne adrese sa sadržajem programskog brojača dobija se tačna vrednost adrese programskog skoka bez obzira što je sadržaj programskog brojača promenjen.

Relativni programski skok može da koristi relativnu adresu od jednog ili dva bajta koji se intepretiraju kao celi brojevi koji mogu biti pozitivni ili negativni. Negativni brojevi predstavljeni su svojim drugim komplementom. Relativna adresa od 1 bajta omogućava programski skok od – 128 do 127 memorijskih lokacija u odnosu na trenutno odstojanje u programskom brojaču IP. Relativna adresa od 2 bajta dozvoljava programski skok u opsegu između –32k i +32k u odnosu na odstojanje u programskom brojaču IP.

Programski skok sa relativnom adresom od 1 bajta ilustrovan je na delu programa:

```
JMP LAB1
MOV AX,1
ADD AX,BX
LAB1:  MOV BX,AX
```

Pretpostavimo da je ovaj deo programa preveden na mašinski jezik i smešten u memoriju, u segment 1000H i na odstojanju 0200H, Slika 2.17.



Slika 2.17: Izgleda programa sa relativnim programskim skokom unapred

U trenutku kada mikroprocesor počne izvršavanje prve instrukcije *JMP LAB1* ovog dela programa, registar CS ima sadržaj 1000H a registar IP sadržaj 0200H. Mikroprocesor uzme kod operacije *EB*₁₆ i poveća sadržaj programskog brojača IP za 1, na 0201H. Mikroprocesor zatim uzme operand *05*₁₆ koji u ovom slučaju predstavlja relativnu adresu, i ponovo poveća sadržaj programskog brojača za 1, na 0202H. U ovom trenutku mikroprocesor izvršava instrukciju čiji kod operacije je EB, a to je relativni programski skok, i na trenutni sadržaj programskog brojača (0202H) dodaje relativnu adresu 05H. Rezultat je 0207H, što znači da će umesto instrukcije na odstojanju 0202H, mikroprocesor izvršiti instrukciju na odstojanju 0207H.

Ovde treba primetiti nekoliko važnih detalja. Prvo, odstojanje se računa od memorijske lokacije koja sledi iza instrukcije relativnog programskog skoka, u našem slučaju, to je instrukcija sa efektivnom adresom 10202. U toj lokaciji nalazi se instrukcija koja bi se izvršila da nema programskog skoka.

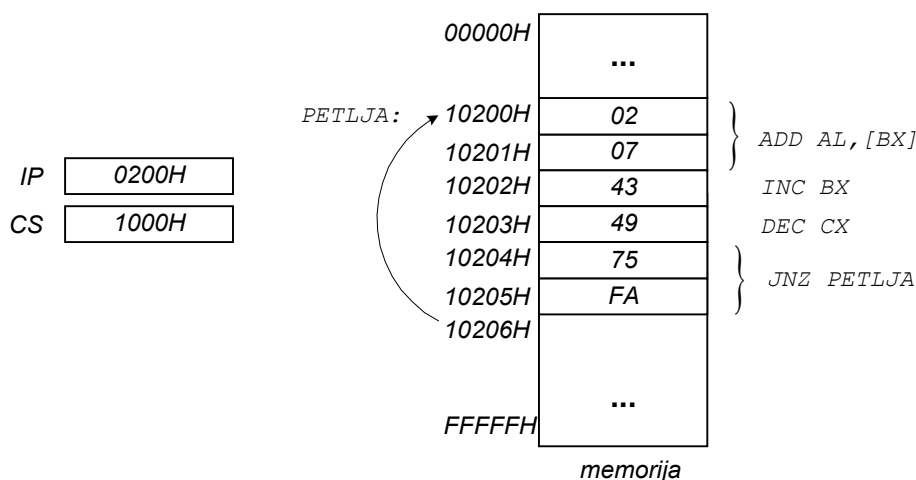
Drugo, prevodilac (asembler) vodi računa o labelama i njihovim pridruženim odstojanjima. Na osnovu programa prevedenog na mašinski jezik, asembler izračunava relativne adrese i stavlja ih u odgovarajuće delova instrukcija. Na primer, asembler izračuna da je relativno odstojanje od instrukcije koja sledi iza *JMP LAB1* do memorijske lokacije obeležene sa *LAB1* ukupno 5 bajtova i tu vrednost stavlja u memorijsku lokaciju sa odstojanjem 0201H, na mesto operanda instrukcije *JMP LAB1*.

Konačno, program ima očigledan nedostatak, jre instrukcija *MOV AX, 1* nema labelu, a pošto je pre nje instrukcija bezuslovnog programskog skoka, ova instrukcija se nikada neće izvršiti.

Ukoliko je labela pre instrukcije relativnog programskog skoka, onda se radi o relativnom programskom skoku unazad i tada je relativna adresa negativna, odnosno predstavljena u drugom komplementu. Uzmimo, na primer, sledeći deo programa:

```
PETLJA:  ADD AL, [BX]
          INC BX
          DEC CX
          JNZ PETLJA
```

Slika 2.18 prikazuje izgleda ovog dela programa u memorijskom segmentu 1000H, na odstojanju 0200H. U trenutku kada mikroprocesor krene da izvršava instrukciju uslovnog relativnog programskog skoka *JNZ PETLJA*, vrednost registra IP je 0204H. Mikroprocesor uzme kod operacije 75H, poveća IP za 1, uzme operand FAH, ponovo poveća IP za, tako da IP ima vrednost 0206 i pokazuje na sledeću instrukciju. Sada počinje izvršavanje instrukcije *JNZ* tako što mikroprocesor proverava vrednost indikatora Z. Ako je Z=1 izvršava se relativni programski skok, a ako je Z=0, onda se izvršava naredna instrukcija na adresi 10206H.



Slika 2.18: Relativni programski skok unazad

Pretpostavimo da je Z=1 i da se izvršava relativni programski skok. Relativna adresa je FAH, ili binarno 1111 1010, najznačajniji biti je 1, što znači da je relativna adresa negativna i da je predstavljena drugim komplementom. Pošto je registar IP 16-bitni, relativna adrese se proširuje sa 8 na 16 bita, ali tako da se zadrži znak, prema tome proširuje se sa jedinicama na najznačajnijim mestima i dobija 16-bitna relativna adresa FFFAH. Sabiranjem odstojanja iz IP, sa 16-bitnom relativnom adresom FFFAH dobija se sledeći rezultat:

$$\begin{array}{r}
 \text{IP} = 0206\text{H} = 0000\ 0010\ 0000\ 0110 \\
 + \quad \text{FFFAH} = 1111\ 1111\ 1111\ 1010 \\
 \hline
 0000\ 0010\ 0000\ 0000_2 = 0200\text{H}
 \end{array}$$

Prema tome, dobija se odstojanje 0200H, a to je odstojanje na kome je labela *PETLJA*. U suštini radi se o tome da je broj FA_{16} negativan i da je to njegov komplement. Ako pretvorimo broj u binarni brojni sistem dobijamo $1111\ 1010_2$, ako ga ponovo komplementiramo dobijamo $-(0000\ 0101 + 1) = -0000\ 0110_2 = -6_{10}$. Kad na odstojanje 0206H dodamo -6 dobija se odstojanje 0200_{16} a to je odstojanje na kome je labela *PETLJA*.

Na sličan način izvršava se relativni programski skok sa relativnom adresom od 2 bajta. U ovom slučaju nije neophodno proširivati relativni adresu sa 8 na 16 bitova, nego se jednostavno relativna adresa odmah sabire sa sadržajem programskog brojača IP.

2.4 Skup instrukcija

U ovom odeljku ukratko su izložene osnovne osobine nekih tipičnih instrukcija mikroprocesora x86. Detaljni opis može se naći u publikacijama proizvođača.

Instrukcije prenosa podataka

Već smo videli da je osnovna instrukcija prenosa podataka *MOV*. Ova instrukcija ima veliki broj različitih modifikacija, može da radi sa registrima i memorijskim lokacijama i sa različitim načinima adresiranja. Veoma moćne su instrukcije koje rade sa nizovima podataka. Sa jednom ovakvom instrukcijom može se prekopirati niz do 64k bajtova sa jednog mesta u memoriji na drugo.

Instrukcije za rad sa stekom

Stek mikroprocesora x86 nalazi se u memorijskom segmentu na koji pokazuje registar SS. Efektivna adresa kod pristupa steku računa se na osnovu segmenta koji je u SS i odstojanja koje je u pokazivaču steka SP:

$$\text{Efektivna adresa} = SS \times 10H + SP$$

Dve osnovne instrukcije koriste se za stavljanje podataka na stek i uzimanje podataka sa steka. Instrukcija *PUSH reg* stavlja sadržaj registra *reg* na stek, a instrukcija *POP reg* uzima reč sa vrha steka i prenosi je u registar *reg*.

Izvršna faza instrukcije *PUSH AX* ima sledeći oblik:

```
SP ← SP - 1
M[SSx10H + SP] ← AH
SP ← SP - 1
M[SSx10H + SP] ← AL
```

Izvršna faza instrukcije *POP AX* ima sledeći oblik:

```
AL ← M[SSx10H + SP]
SP ← SP + 1
AH ← M[SSx10H + SP]
SP ← SP + 1
```

Aritmetičke i logičke instrukcije

Ovaj skup sadrži uobičajene instrukcije za sabiranje, oduzimanje, povećanje i smanjenje sadržaja registra za 1 i tako dalje. Vredi pomenuti instrukciju *ADC*, koja omogućava sabiranje dva operanda sa sadržajem prenosa C, čime se omogućava sabiranje brojeva koji zauzimaju više reči.

Po pravilu ove instrukcije daju rezultat i dovode do promene indikatorskih bita u skladu sa efektima izvršenja instrukcije. Zanimljiva je instrukcija *CMP operand1, operand2* koja oduzima drugi operand od prvoga i na osnovu efekta ove operacije postavlja indikatorske bite. Rezultat

oduzimanje se gubi. Smisao ove instrukcije je u upoređenju dva broja, jer se na osnovu bita Z, C, S i O može odrediti relativni odnos dva operanda.

Logičke instrukcije obuhvataju operacije I, ILI, NE i ekskluzivno ILI. Svaka od ovih operacija obavlja navedenu logičku operaciju nad pojedinim bitima.

Instrukcija AND obavlja logičku I operaciju nad operandima i često se koristi za brisanje određenih bita u registru, pri čemu ostali biti ostaju nepromenjeni. Na primer, ako želimo izbrisati najznačajniji bit u registru AL, a da pri tome sve ostali biti ostanu nepromenjeni, treba izvršiti instrukciju:

AND AL, 7FH

Operand 7FH naziva se *maska* i formira se tako što se stavlja 0 na mesto koje treba izbrisati (upisati 0), a 1 na mesta koja treba da ostanu nepromenjena. U našem slučaju 7FH = 0111 1111, pa se prema tome operacijom I briše najznačajniji bit.

Instrukcija OR, koja obavlja logičku ILI operaciju, koristi se za upisivanje 1 u odabrane bite, pri čemu ostali biti ostaju nepromenjeni. Na primer, sledeća instrukcija upisuje 1 u bit broj 2 registra CL:

OR CL, 02H

Maska je u ovom slučaju 02H, odnosno maska sadrži jedinica na mestima bita u kojima treba upisati jedinice, a nule na svim ostalim mestima.

Logička operacija XOR, koja izvršava ekskluzivnu ILI operaciju, koristi se za invertovanje odabranih bita. Na primer, sledeća instrukcija invertuje 4 najznačajnija bita registra BH, pri čemu ostali biti ostaju nepromenjeni:

XOR BH, F0H

Naravno, u ovom slučaju maska sadrži jedinice na mestima na kojima treba invertovati bite.

Instrukcije za rad sa procedurama

Dve instrukcije podržavaju rad sa procedurama: CALL i RET. Ključni zadatak instrukcije CALL je da adresu povratka smesti na stek, i da u registre CS i IP stavi adresu potprograma. Instrukcija RET uzima adresu povratka sa steka i stavlja u registre CS i IP.

Obe instrukcije imaju dve verzije, jednu koja se koristi za procedure koje su u istom memorijskom segmentu i drugu koja se koristi za procedure koje su u različitom memorijskom segmentu. U ovom odelju kratko je opisana prva verzija, koja radi sa procedurama iz istog memorijskog segmenta.

Instrukcija *CALL* kojom se poziva potprogram u istom segmentu ima tri bajta: prvi bajt je kod operacija, a drugi i treći bajt sadrže relativnu adresu potprograma. Izvršna faza instrukcije *CALL potprogram* ima sledeći izgled (*temp* je neki privremeni registar):

```
temp(nizi) ← M[CS×10H + IP]    ; niži bajt adrese potprograma
IP ← IP + 1
temp(visi) ← M[CS×10H + IP]    ; viši bajt adrese potprograma
IP ← IP + 1
SP ← SP - 1
```

$$\begin{aligned}
M[SSx10H + SP] &\leftarrow IP(visi \text{ bajt}) \\
SP &\leftarrow SP - 1 \\
M[SSx10H + SP] &\leftarrow IP(nizi \text{ bajt}) \\
IP &\leftarrow IP + temp
\end{aligned}$$

Izvršna faza instrukcije *RET* ima sledeći izgled:

$$\begin{aligned}
IP(nizi \text{ bajt}) &\leftarrow M[SSx10H + SP] \\
SP &\leftarrow SP + 1 \\
IP(visi \text{ bajt}) &\leftarrow M[SSx10H + SP] \\
SP &\leftarrow SP + 1
\end{aligned}$$

3. Prekidi

Osnovni ciljevi uvođenja mehanizma prekida su povećanje produktivnosti mikroprocesora i sinhronizacija rada mikroprocesora sa događajima u njegovoj okolini. Podsystem prekida omogućava mikroprocesoru da, pod dejstvom nekog spoljnog događaja, prekine izvršavanje tekućeg programa i pređe na izvršavanje programa koji je pripremljen za obradu tog događaja. Na primer, u mikroračunarskom sistemu mogu da postoje dva programa: jedan, koji obavlja statističku obradu podataka smeštenih na disku, i drugi, koji prihvata podatke preko komunikacione linije. Ukoliko komunikaciona linija nije aktivna, mikroprocesor obrađuje podatke sa diska. Međutim, u slučaju da treba da su pristigli podaci, mikroprocesor mora da prekine statističku obradu i izvrši program za prijem podataka sa komunikacione linije. Jasno, ukoliko mikroprocesor na vreme ne obavi prijem podataka, oni mogu biti nepovratno izgubljeni.

U ovom poglavlju izložene su osnovne ideje i način realizacije podsistema prekida.

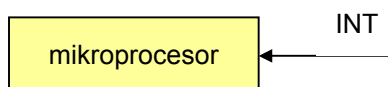
3.1 Osnovna ideja i problemi vezani sa prekidom

U opštem slučaju prekid se koristi u primenama u kojima mikroprocesor obavlja dva zadatka, jedan nižeg prioriteta, koji se obično izvršava kontinualno u vremenu i drugi, višeg prioriteta, koji se izvršava povremeno, na pobudu nekim spoljnim događajem. Ovakav scenario može se uporediti sa osobom koja čita knjigu i treba da odgovara na telefonske pozive. Čitanje knjige je zadatak nižeg prioriteta, koji dugo traje i može da se obavlja ako osoba nema nešto drugo da radi. Podrazumeva se da obavljanje ovog zadatka neće biti ugroženo ako osoba odloži čitanje za neki kraći vremenski interval. Pretpostavlja se da je odgovor na telefonski poziv zadatak višeg prioriteta i zahteva da taj zadatak mora da se obavi kada se pojavi potreba za njim. Drugim rečima, zahtev je da osoba mora odgovoriti na poziv pre nego što telefon prestane da zvoni.

U slučaju da telefon zazvoni, osoba koja čita knjigu može da primeni različite taktike. Na primer, može pokušati da pročita celu knjigu, pa tek onda da odgovori na poziv, taktika koja skoro sigurno ne daje dobre rezultate. Ili, na primer, osoba može da prekine čitanje knjige, zanemari do tada pročitani deo knjige, odgovori na telefonski poziv i onda krene da čita knjigu od početka. U krajnjoj linije, može se zamisliti i taktika u kojoj osoba prvo čeka telefonski poziv, odgovori na njega, i onda krene sa čitanjem knjige.

Intuitivno, najbolje rezultate daje taktika po kojoj osoba čita knjigu, a kada telefon zazvoni, zapamti mesto do koga je došla u čitanju, odgovori na telefonski poziv, a zatim nastavi sa čitanjem od zapamćenog mesta u knjizi. Ova taktika daje dobru produktivnost i sinhronizaciju sa pozivima koji se događaju u trenucima koje osoba ne može da predvidi.

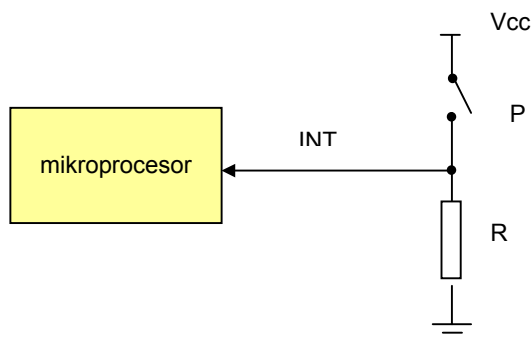
Da bi dobio informaciju o spoljnjem događaju, mikroprocesor ima poseban ulazni signal koji se obično naziva INT, od engleskog 'interrupt' (prekid), Slika 3.1. Radi jednostavnosti, ostali signali mikroprocesora, kao što su magistrale i upravljački signali, nisu predstavljeni na slici.



Slika 3.1: Ulazni signal INT za prekid

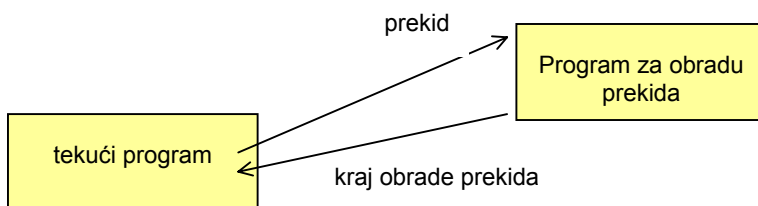
Naravno, u okviru mikroprocesorskog sistema mora da postoji elektronsko kolo koje, kada detektuje spoljni događaj, generiše signal prekida. Kao primer uzmimo da mikroprocesor treba da

obradi signal prekida koji je nastao kao posledica aktiviranja nekog mehaničkog prekidača. Mehnički prekidač može biti vezan na tipku tastature, tako da pritisak prsta na tipku aktivira prekidač i time signal prekida. Prekidač može aktivirati neki mehaničkih događaj, na primer zatvaranje vrata na liftu ili nailazak dela mašine na prepreku.



Slika 3.2: Jednostavan način generisanja signala prekida mehaničkim prekidačem

Slika 3.2 prikazuje jednostavno električno kolo koje generiše signal prekida INT u slučaju da je signal prekida aktivan na logičkoj jedinici. Kada je prekidač otvoren, signal INT je preko otpornika R vezan na masu, pa je na niskom naponskom nivou, odnosno na nivou logičke 0. Kada se prekidač zatvori, signal INT je vezan na napon napajanja Vcc i ima vrednost logičke 1. Prema tome, zatvaranjem prekidača aktivira se logička 1 na ulazu INT i time signalizira mikroprocesoru signal prekida.



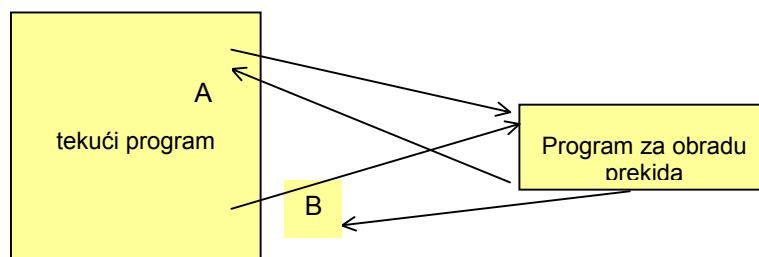
Slika 3.3: Tekući program i program za obradu prekida

Pre nego što se koristi mehanizam prekida, u operativnoj memoriji mikroračunara mora da postoji program za obradu prekida, Slika 3.3. Program koji mikroprocesor obrađuje kada nema prekida obično se naziva 'tekući program'. Kada se aktivira signal prekida, mikroprocesor sačeka izvršavanje instrukcije koja je u toku i pređe na izvršavanje programa za obradu prekida. Instrukcija u kojoj je tekući program prekinut naziva se tačka prekida.

Po završetku programa za obradu prekida mikroprocesor mora da se vrati u tekući (prekinuti) program i nastavi izvršavanje od instrukcije u kojoj je prekinuto izvršavanje tekućeg programa. Preciznije, tekući program nastavlja izvršenje od instrukcije koja sledi posle tačke prekida (zato što je instrukcija na tački prekida već izvršena).

Naravno, mehanizam prekida mora da reši pitanje povratne adrese, odnosno mora da na neki način omogući mikroprocesoru da nastavi izvršavanje tekućeg programa na mestu u kojem je bio prekinut.

Slika 3.4 ilustruje problem povratne adrese kod prekida.



Slika 3.4: Ilustracija problema povratne adrese

Ako je tekući program prekinut u tački A, onda po završetku programa za obradu prekida, tekući program mora da se nastavi u tački A. Međutim, ako je tekući program prekinut u tački B, onda mora da se nastavi u istoj tački. Drugim rečima, mehanizam prekida mora da omogući tačnu povratnu adresu, koja će posle završetka programa za obradu prekida, obezbediti nastavak tekućeg programa u tački prekida. Slično kao kod poziva potprograma, problem povratne adrese efikasno se rešava korišćenjem steka, odnosno stavljanjem povratne adrese na stek. Kod povratka u tekući program mikroprocesor uzme sa steka povratnu adresu na kojoj se nalazi instrukcija od koje se nastavlja izvršavanje tekućeg programa.

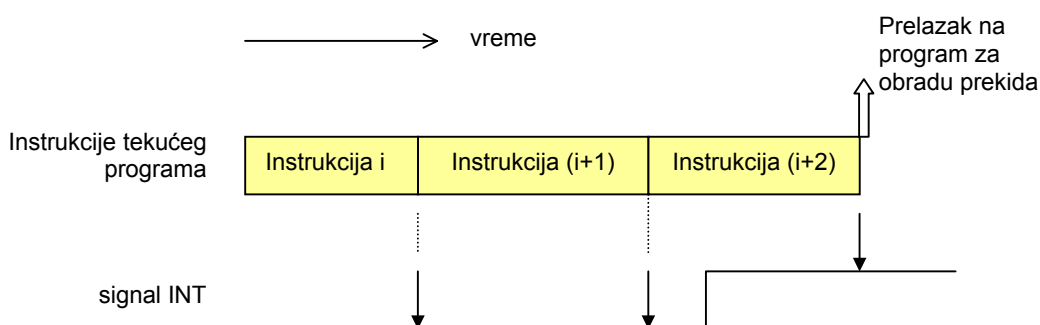
U praksi obično postoje višestruki izvori prekida. Na primer, u mikroračunarskom sistemu mogu da postoje prekidi koje generiše tastatura, signal realnog vremena, masovna memorija, komunikaciona linija i tako dalje. U slučaju višestrukih izvora prekida, podsistem za prekid mora da obezbedi:

- programe za obradu različitih signala prekida,
- prepoznavanja izvora prekida,
- rešavanje prioriteta prekida kada se istovremeno aktivira više od jednog prekidnog signala,
- prelazak na program za obradu prekida koji odgovara izvoru prekida i
- postupak u slučaju da se, u toku obrade jednog prekida, aktivira neki drugi signal prekida.

U daljem tekstu se detaljnije obrađuje odziv mikroprocesora na signal prekida, rešavanje problema vezanih za višestruke prekide i druga pitanja vezana za podsistem prekida.

3.2 Odziv mikroprocesora na signal prekida

Upravljačka jedinica mikroprocesora nadgleda ulazni signal prekida, neka je taj signal označen sa INT, i proverava stanje ovog signala na kraju izvršenja svake instrukcije. Ako signal INT nije aktivan, mikroprocesor izvršava sledeću instrukciju, a ako je signal INT aktivan mikroprocesor prelazi na obradu prekida.



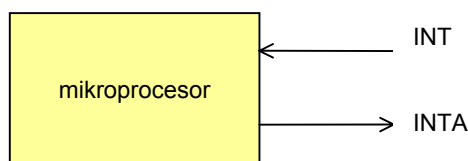
Slika 3.5: Upravljačka jedinica proverava stanje signala INT na kraju svake instrukcije

Slika 3.5 ilustruje postupak provere stanja signala INT, koji je predstavljen linijom sa dva nivoa. Niži nivo predstavlja logičku 0, a viši nivo logičku 1. Izvršavanje instrukcije u vremenu predstavljeno je pravougaonikom u koji je upisan broj instrukcije koja se izvršava. U primeru na slici počinjemo da posmatramo izvršavanje tekućeg programa od instrukcije sa rednim brojem i , posle koje sledi instrukcija $(i+1)$, zatim $(i+2)$ i tako dalje.

Na završetku svake instrukcije, upravljačka jedinica proverava da li je signal INT aktivan, a trenuci provere (koji se podudaraju sa završetkom instrukcije) na slici su predstavljeni vertikalnim strelicama. Na primer, posle instrukcije sa rednim brojem i signal INT nije aktivan i nastavlja se izvršenje instrukcije $(i+1)$. Po završetku instrukcije sa rednim brojem $(i+1)$ signal INT takođe nije aktivan i nastavlja se izvršenje instrukcije $(i+2)$. Međutim, po završetku instrukcije $(i+2)$ signal prekida je aktivan i mikroprocesor prelazi na program za obradu prekida.

Treba primetiti da trenutak aktiviranja signala INT nije od značaja, odnosno signal INT može da se aktivira u bilo kojem trenutku u toku izvršenja instrukcije $(i+2)$. Upravljačka jedinica ignoriše stanje signala INT u toku izvršenja instrukcije, prema tome važno je samo da je u trenutku provere taj signal stabilan i aktivan.

Kasnije ćemo videti da postoje razlozi da mikroprocesor obavesti perifernu jedinicu, koja je zahtevala prekid, da je prekid prihvaćen i da mikroprocesor prelazi na obradu signala prekida. Obično mikroprocesor ima poseban spoljni izlazni signal koji se aktivira kada mikroprocesor prihvati prekid. Slika 3.6 prikazuje mikroprocesor sa signalom prekida INT i signalom da je prekid prihvaćen, koji se često označava sa INTA (engleski: *Interrupt Acknowledge*).



Slika 3.6: Mikroprocesor sa signalom za prekid (INT) i signalom da je prekid prihvaćen (INTA)

Različiti mikroprocesori na različit način reaguju na signal prekida. U principu, odziv mikroprocesora na aktivan signal prekida obuhvata sledeće korake:

- aktivirati signala INTA,
- odrediti početnu adresu programa za obradu prekida,
- staviti na stek povratnu adresu koja se nalazi u programskom brojaču,
- staviti u programski brojač početnu adresu programa za obradu prekida,
- signal INTA prevesti u neaktivno stanje i
- nastaviti izvršenje instrukcija.

Uz pretpostavku da povratna adresa staje u jednu memorijsku reč, mikroprogram koji određuje odziv mikroprocesora na signal prekida izgleda ovako:

```

SP ← SP-1           ; mesto na steku za povratnu adresu
M[SP] ← PC          ; povratna adresa iz PC na stek
INTA ← 1            ; prekid prihvaćen, aktivirati signal INTA
PC ← adresa programa za obradu prekida
INTA ← 0            ; signal INTA prevesti u neaktivno stanje

```

Naravno, ključni problem u ovom mikroprogramu je određivanje adrese programa za obradu prekida. Ovaj problem obrađen je u narednom odeljku.

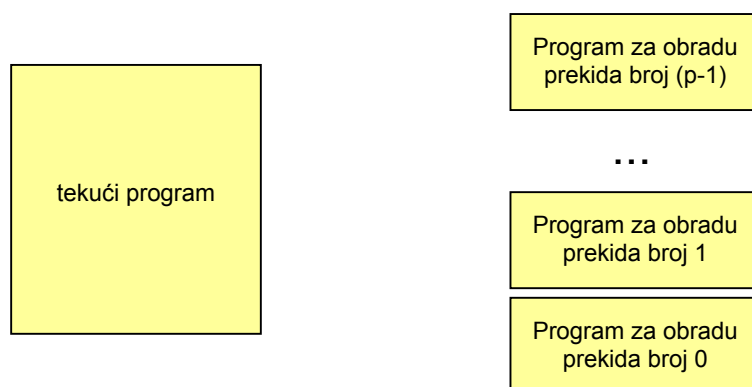
Mikroprogram za odziv mikroprocesora na signal prekida sličan je mikroprogramu instrukcije CALL poziva potprograma. Ipak, postoje značajne razlike, na primer zato što mikroprogram instrukcije CALL ne utiče na signal INTA.

Povratak iz programa za obradu prekida u prekinuti program obavlja instrukcija RETI (*Return from Interrupt*). Izvšna faza mikroprograma instrukcije RETI ima sledeći izgled:

$PC \leftarrow M[SP]$; povratna adresa sa steka u PC
 $SP \leftarrow SP+1$; podešavanje pokazivača steka

3.3 Vektorski prekid

Kao što je već napomenuto, u mikroračunarskom sistemu obično postoji više izvora prekida. U tom slučaju, projektant mikroračunarskog sistema mora da pripremi više programa za obradu prekida, u opštem slučaju po jedan program za svaki signal prekida, Slika 3.7.



Slika 3.7: Svaki signal prekida ima svoj program za obradu prekida

Mehanizam prekida koristi posebnu strukturu podataka koja se naziva *tabela vektora prekida*. Elementi ove tabele su *vektori prekida*, jedna ili više memorijskih lokacija u koje se smeštaju adrese programa za obradu prekida. Slika 3.8 prikazuje tabelu vektora prekida, na levoj strani nepopunjenu, a na desnoj strani popunjenu početnim adresama programa za obradu prekida.

Broj vektora prekida

m-1	
	...
1	
0	

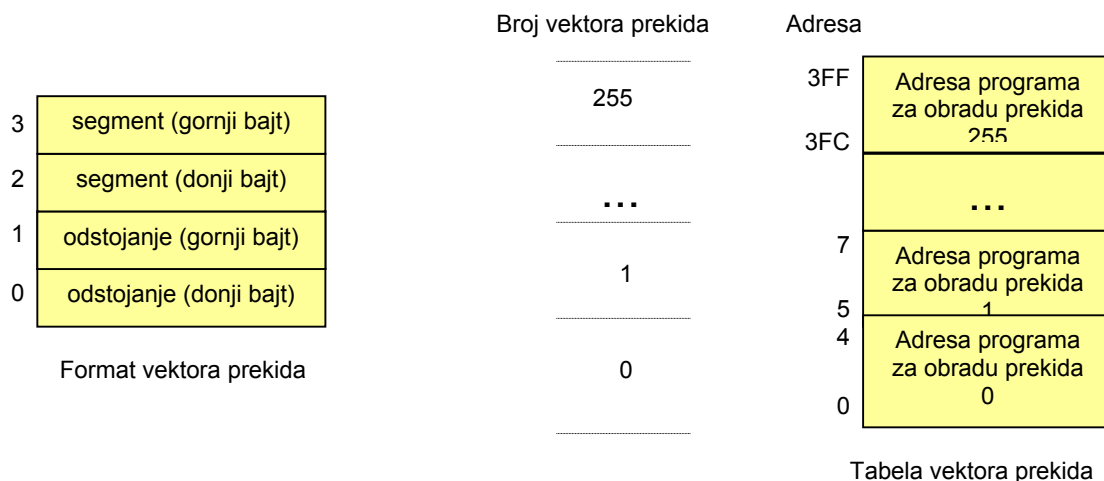
Broj vektora prekida

m-1	adresa programa za obradu prekida m-1
	...
1	adresa programa za obradu prekida 1
0	adresa programa za obradu prekida 0

Slika 3.8: Tabela vektora prekida: prazna (levo) i popunjena (desno)

Sa leve strane tabele navedeni su brojevi vektora prekida, celi brojevi koji počinju od 0. U vektore prekida stavljaju se početne adrese programa za obradu prekida, Slika 3.8. Naravno, nije neophodno da brojevi vektora prekida odgovaraju rednom broju prekida, kao što je to urađeno na prethodnoj slici. Kao što je već rečeno, u opštem slučaju vektor prekida ima više memorijskih lokacija, tako da broj vektora prekida ne mora da istovremeno predstavlja i adresu vektora prekida.

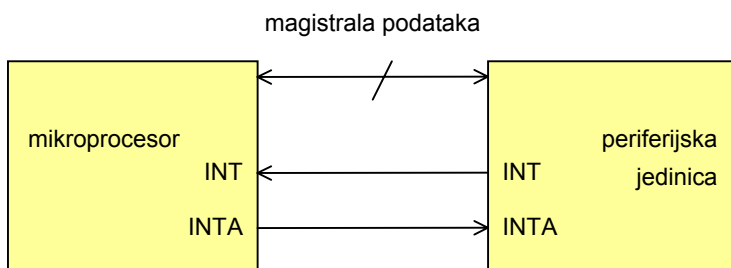
Slika 3.9 prikazuje primer tabele vektora prekida mikroprocesora x86. Tabela ima ukupno 256 vektora prekida, a svaki vektor prekida zauzima četiri bajta. Format vektora prekida, prikazan na slici levo, sastoji se od dva bajta za odstojanje i dva bajta za segment. Prema tome, vektor prekida sadrži segment i odstojanje početne adrese programa za obradu prekida, što znači da program za obradu prekida može biti smešten u bilo kom segmentu operativne memorije.



Slika 3.9: Tabela vektora prekida mikroprocesora x86

Kod mikroprocesora x86 tabela vektora prekida uvek je smeštena na dnu memorijskog prostora i zauzima $256 \times 4 = 1$ kB. Pošto vektor prekida ima četiri bajta, adresa vektora prekida dobija se množenjem broja vektora prekida sa 4. Tako je adresa vektora prekida broj 0 jednaka 0, vektora prekida broj 1 je 4, vektora prekida broj 2 je 8 i tako dalje. Adresa poslednjeg vektora prekida broj 255 je 3FC heksadecimalno.

Na osnovu broja vektora prekida mikroprocesor određuje adresu vektora prekida i iz njega uzima početnu adresu programa za obradu prekida. Prema tome, da bi odredio početnu adresu prekidnog programa mikroprocesoru treba saopštiti broj vektora prekida. Ovaj zadatak obično obavlja jedinica koja generiše prekid: u toku prihvatanja prekida jedinica koja je generisala prekid stavlja na magistralu podataka broja vektora prekida koji mikroprocesor pročita i dalje obavlja opisane operacije.



Slika 3.10: Veza mikroprocesora i jedinice koja generiše signal prekida

Slika 3.10 prikazuje vezu između mikroprocesora i periferijske jedinice koja generiše signal prekida. U okviru odziva na signal prekida, mikroprocesor aktivira signal da je prekid prihvaćen INTA. Sa svoje strane, na signal INTA periferijska jedinica se identifikuje tako što na magistralu podataka stavi svoj broj vektora prekida. Mikroprocesor uzima broj vektora prekida sa magistrale podataka i na osnovu ovog broja određuje lokaciju vektora prekida i konačno iz vektora prekida uzima početnu adresu programa za obradu prekida. Početna adresa se prenosi u programski brojač i mikroprocesor kreće da obrađuje instrukcije programa za obradu prekida. U sledećem mikroprogramu, koji opisuje odziv mikroprocesora na signal prekida, pretpostavljeno je da broj

vektora prekida istovremeno predstavlja adresu odgovarajućeg vektora prekida. Naravno, u slučaju mikroprocesora x86, broj vektora prekida treba pomnožiti sa 4.

```

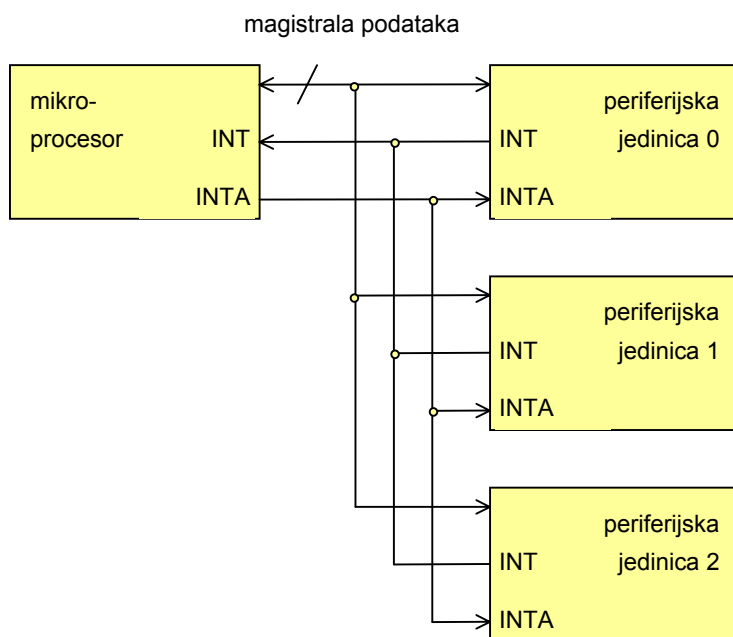
SP ← SP-1          ; mesto na steku za povratnu adresu
M[SP] ← PC         ; povratna adresa iz PC na stek
INTA ← 1           ; na signal INTA per. jedinica stavlja na
                  ; magistralu podataka broj vektora prekida
temp ← magistrala podataka ; temp = broj vektora prekida
PC ← M[temp]       ; PC = početna adresa prekidnog programa
INTA ← 0           ; signal INTA prevesti u neaktivno stanje

```

Naravno, ostaje pitanje kako periferna jedinica zna 'svoj' broj vektora prekida? Periferna jedinica obično ima programabilni registar u koji mikroprocesor, u toku inicijalizacije mikroracunarskog sistema, upisuje broj vektora prekida. Time je zaokružen problem odreživanja početne adrese programa za obradu prekida. Naravno, u slučaju da postoje višestruke jedinice koje mogu da generišu prekid, svaka od njih mora da na prihvaćen 'svoj' signal prekida odgovori stavljanjem 'svog' broja vektora prekida na magistralu podataka.

3.4 Kontroler prekida

Kao što je rečeno, u složenom mikroracunarskom sistemu postoji više jedinica koje generišu signal prekida. Svaka od ovih jedinica mora da ima programabilni registar u koji mikroprocesor upisuje broj vektora prekida. Slika 3.11 prikazuje primer mikroracunarskog sistema sa tri jedinice (označene sa 0, 1 i 2), koje mogu da generišu signal prekida.



Slika 3.11: Primer mikroracunarskog sistema sa tri jedinice koje generišu signal prekida

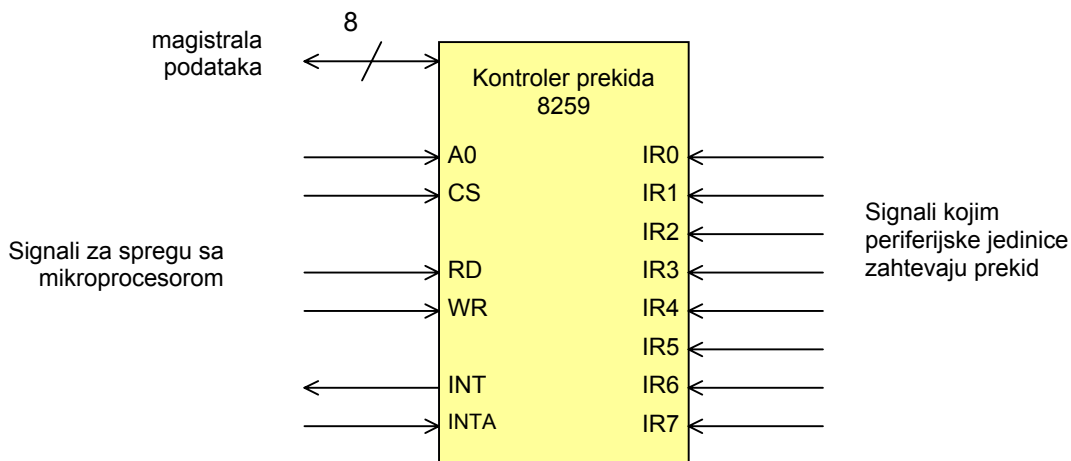
Svaka periferna jedinica, koja može da zahteva prekid, mora da ima programabilni registar u koji mikroprocesor upisuje broj vektora prekida. Međutim, ovakva konfiguracija podsistema prekida u nekim slučajevima nije pogodna. Neke jedinice koje zahtevaju prekid, mogu da budu veoma jednostavne, kao na primer generator signala realnog vremena. Iako se u suštini sastoji samo od oscilatora stabilne frekvencije, generatoru signala realnog vremena mora se dodati logika sa programabilnim registrom i upravljačkom jedinicom koja omogućava upis u registra i prenos sadržaja registra na magistralu podataka.

Postoje i drugi problemi vezani za konfiguraciju koju prikazuje

Slika 3.11. Na primer, pošto postoji samo jedan signal prekida INT na koji su vezane sve tri jedinice, nije jasno kako se rešava problem identifikacije jedinice koja je zahtevala prekid. Nije jasno ni kako se rešava prioritet prekida u slučaju da dve ili tri jedinice istovremeno zahtevaju prekid. Na kraju, nije jasno ni koja od jedinica na signal INTA stavlja broj vektora prekida na magistralu podataka.

Neka od ovih pitanja mogu se rešiti uvođenjem dodatnog hardvera. Na primer, mikroprocesor može da ima više od jednog ulaznog signala za prekid. U tom slučaju, unutar mikroprocesora može da se ugradi mehanizam vezan za prioritet prekida i identifikaciju jedinice koja je generisala prekid.

Jedno efikasno rešenje problema vezanih za višestruke prekid sastoji se u korišćenju posebnog integrisanog kola koje se naziva *kontroler prekida*. Slika 3.12 prikazuje jedan takav kontroler, označen brojem 8259, koji se koristi sa mikroprocesorima 8259. Radi jednostavnosti, izostavljeni su neki signali kontrolera, na primer napajanje, masa i neki manje važni upravljački signali.

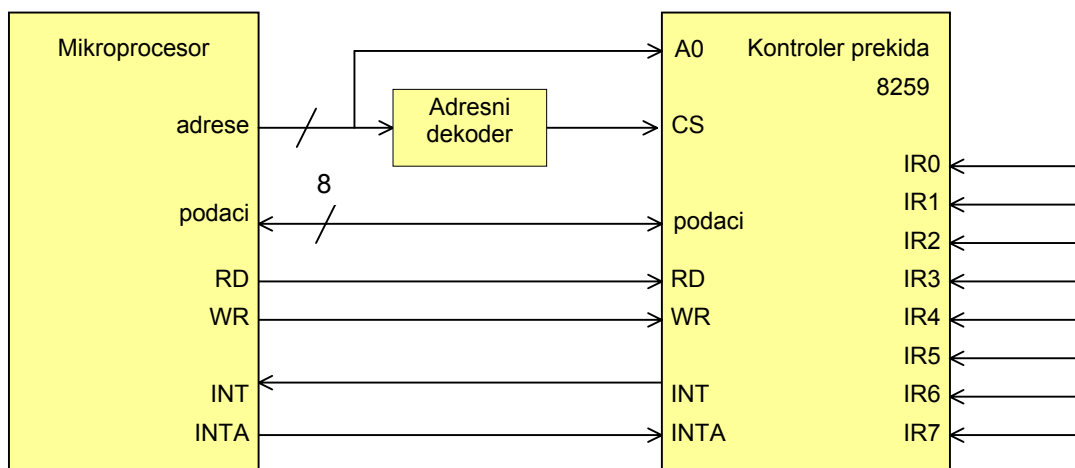


Slika 3.12: Kontroler prekida 8259

Kontroler prekida ima dva podskupa signala: jedan koji se koristi za spregu sa jedinicama koje generišu prekid i drugi, koji služi za spregu sa mikroprocesorom. Kontroler prekida obavlja nekoliko zadataka:

- nadgleda signale kojim periferijske jedinice zahtevaju prekid i određuje prioritet signala prekida,
- generiše signal prekida koji prosleđuje mikroprocesoru i
- obezbeđuje mikroprocesoru broj vektora prekida.

Pošto kontroler prekida prihvata više (na primer u slučaju 8259 ukupno 8) ulaznih signala kojima periferijske jedinice zahtevaju prekid, a sa druge strane generiše jedinstven signal prekida mikroprocesoru, efektivno kontroler prekida multiplicira broj signala prekida. Tako 8259 osam puta povećava broj signala prekida koji se mogu dovesti na mikroprocesor. Kontroleri prekida mogu se kaskadno vezivati pa tako, sa dva nivoa kaskadne veze, devet kontrolera prekida 8259 opslužuju ukupno 64 signala zahteva za prekid.



Slika 3.13: Primer sprege mikroprocesora i kontrolera prekida 8259

Slika 3.13 prikazuje primer sprege između mikroprocesora i kontrolera prekida, uz pretpostavku da mikroprocesor ima 8-bitnu magistralu podataka i da su signali RD, WR, INT i INTA međusobno kompatibilni. Kao što se vidi, linije magistrale podataka su neposredno međusobno povezane. Linije za signale čitanja i upisa (RD i WR) kao i signale zahteva za prekid (INT) i prekid prihvaćen (INTA) takođe su neposredno povezane. Međutim, linije za adresne signale, osim signala najmanje težine (A0) vode se na ulaz adresnog dekodera, koji za određenu kombinaciju adresnih signala generiše signal selekcije (CS, engleski Chip Select) kontrolera prekida.

Kada je signal CS aktivan, mikroprocesor može da pristupi internim programabilnim registrima kontrolera prekida. Pristup programabilnim registrima neophodan je da bi mikroprocesor programirao kontroler, odnosno upisao binarne informacije koje određuju način rada kontrolera. Mikroprocesor na isti način može da pročita sadržaje registara u kojima se nalazi informacije o stanju kontrolera prekida.

U toku inicijalizacije mikroračunarskog sistema, mikroprocesor programira kontroler prekida tako što u njegove programabilne registre upiše sledeće podatke koji određuju:

- konfiguraciju, na primer da li kontroler radi u kaskadnoj vezi i kakav je prioritet ulaznih signala zahteva za prekid (IR),
- način rada, na primer da li su signali zahteva za prekid od perifernih jedinica aktivni na prednjoj ivici ili na logičkoj 1, da li je dozvoljen prekid sa nekog od ulaznih zahteva i slično,
- broj vektora prekida za svaki signal zahteva za prekid.

Jednom kada je programiran, kontroler prekida je spreman za rad. Postupak rada kontrolera prekida obuhvata sledeće korake:

- Kontroler nadzire ulazne signale (IR) zahteva za prekid i signal selekcije (CS). Ukoliko ni jedan od ovih signala nije aktivan, kontroler je u stanju čekanja i ne preduzima nikakve akcije.
- Mikroprocesor može, aktiviranjem signala CS da pristupi internim programabilnim registrima kontrolera bilo da promeni način rada bilo da pročita stanje kontrolera.
- Ako se aktivira jedan od ulazanih signala IR i ako je dozvoljen prekid sa tog ulaza, kontroler aktivira signal prekida INT. Ukoliko mikroprocesor prihvati prekid i aktivira signal INTA, kontroler na magistralu podataka stavlja broj vektora prekida koji odgovara aktivnom ulaznom signalu zahteva za prekid.
- Ukoliko su dva ili više ulaznih signala zahteva za prekid aktivni, kontroler odabira signal sa najvišim prioritetom i obavlja prethodni korak.

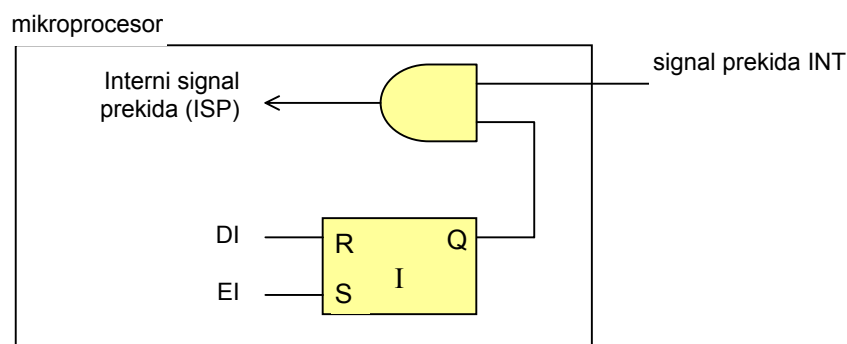
Tabela 3.1: Signali zahteva za prekid u klasi računara IBM PC

IRQ	Broj vektora prekida	Periferijska jedinica
IRQ0	08	Signal realnog vremena
IRQ1	09	Tastatura
IRQ2	0A	Prekid sa pratećeg kontrolera
IRQ3	0B	Linija COM2 za serijsku komunikaciju
IRQ4	0C	Linija COM1 za serijsku komunikaciju
IRQ5	0D	Priključak LPT2 za paralelni prenos podataka
IRQ6	0E	Flopi disk
IRQ7	0F	Priključak LPT1 za paralelni prenos podataka
IRQ8	70	Signal realnog vremena (CMOS)
IRQ9	71	(ne koristi se)
IRQ10	72	(ne koristi se)
IRQ11	73	(ne koristi se)
IRQ12	74	Miš (kod klase računara PS2)
IRQ13	75	Matematički koprocessor
IRQ14	76	Hard disk
IRQ15	77	(ne koristi se)

Tabela 3.1 prikazuje signale IRQ zahteva za prekid od periferijskih jedinica u klasi računara IBM PC, njihove vektore prekida i periferijske jedinice koje generišu zahteve za prekid. Prvih 8 zahteva dolazi od vodećeg, a drugih 8 od pratećeg kontrolera prekida. Kao što se vidi, brojevi vektora prekida su sekvencijalni i kod vodećeg kontrolera idu od broja 08 do 0F (heksadecimalno), a kod pratećeg od 70 do 77 (heksadecimalno).

3.5 Zabrana prekida

U nekim slučajevima neophodno je da mikroprocesor ignoriše signal prekida, odnosno sprečiti prihvatanje prekida. Na primer, u toku inicijalizacije mikroprocesor u programabilne registre periferijskih jedinica, odnosno u kontroler prekida, upisuje brojeve vektora prekida. Očigledno je da u toku inicijalizacije podsistem prekida još nije pripremljen i prihvatanje prekida bi dovelo do nepredvidivog odziva mikroprocesora. Jednostavan način da se to spreči je da se zabrani prihvatanje prekida. Zabrana prekida još se naziva *maskiranje* prekida.



Slika 3.15: Logička šema zabrane prekida

Zabrana prekida obično se zasniva na korišćenju jednog flip-flopa i logičkog I kola, Slika 3.15. Spoljni signal prekida INT dovodi se na ulaz logičkog I kola, koje se nalazi unutar

mikroprocesora. Drugi ulaz logičkog I kola dovodi se sa Q izlaza RS flip-flopa I. Izlaz logičkog I kola predstavlja interni signal prekida (ISP) koji se vodi na upravljačku jedinicu mikroprocesora. Prema tome, upravljačka jedinica nadzire interni signal prekida ISP umesto spoljnog signala prekida INT.

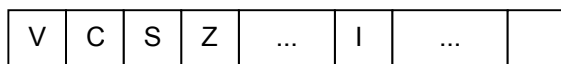
Kada je flip-flop I u stanju logičke 0, na izlazu logičkog I kola je logička 0 bez obzira na stanje signala INT. Stanje logičke 0 flip-flopa I odgovara stanju u kome je prekid zabranjen (maskiran). Ako je flip-flop I u stanju logičke 1, onda izlaz logičkog kola zavisi od stanja signala prekida INT: ako je INT na logičkoj 0 izlaz logičkog kola je isto na 0, a ako je INT na logičkoj 1, izlaz logičkog kola takođe je na jedinici. Dakle stanje logičke 1 flip-flopa I odgovara stanju u kome je prekid dozvoljen.

Stanje flip-flopa I kontroliše se softverski sa dve instrukcije. Instrukcija DI (Disable Interrupt) prevodi flip-flop I u stanje logičke 0 tako što na ulazu R generiše pozitivni implus koji resetuje flip-flop. Instrukcija EI (Enable Interrupt) generiše pozitivni impuls na S ulazu i tako setuje flip-flop I. Izvršne faze ovih instrukcija su jednostavne:

```
DI:   R ← 1      ; ulaz R flip-flopa na logičku 1
      R ← 0      ; ulaz R na logičku 0

EI:   S ← 1      ; ulaz S flip-flopa na logičku 1
      S ← 0      ; ulaz S na logičku 0
```

Flip-flop I za zabranu prekida obično je u sastavu indikatorskog registra mikroprocesora, Slika 3.16. Kada je bit I indikatorskog registra na logičkoj 1 prekid je dozvoljen, a kada je na logičkoj 0 prekid je zabranjen.



Slika 3.16: Primer indikatorskog registra sa bitom I za zabranu (dozvolu) prekida

Uzastopni prekid mogu da ugroze korektno izvršavanje programa za obradu prekida. Zato se u toku obrade prekida ili bar u kritičnom delu programa za obradu prekida često se zabranjuje prekid da bi se prekidni program (ili njegov kritični deo) zaštitio od daljih uzasponih prekida. Oblik programa za obradu prekida sa zabranom daljih prekida izgleda ovako:

```
DI          ; zabrana prekida
PUSH R1     ; čuvanje sadržaja registara na steku
...
PUSH Rn
...
(telo programa za obradu prekida)
...
POP Rn      ; restauracija sadržaja registara
...
POP R1
EI          ; dozvola prekida
RETI       ; povratak u prekinuti program
```

U programu za obradu prekida prvo se instrukcijom DI sprečava prihvatanje novih prekida i zatim sadržaji registara mikroprocesora čuvaju na steku. Ovaj korak je neophodan da bi sadržaji registara prekinutog programa bili isti u trenutku prekida i u trenutku povratka iz programa za obradu prekida. Zatim sledi telo programa za obradu prekida u kome se obavlja zadatak koji se odnosi na događaj koji je vezan za prekidni signal.

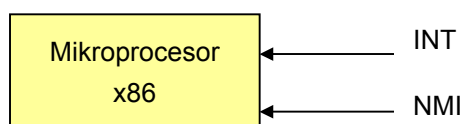
Po završetku tela programa za obradu prekida, u registre opšte namene vraćaju se prvobitni sadržaji i, pre povratka u prekinuti program, dozvoljava prekid. Naravno, instrukcije DI i EI mogu da se koriste za zaštitu samo dela prekidnog programa. U slučaju da podsistem prekida dozvoljava višestruke prekide, odnosno prekidanje prekidnog programa, onda se instrukcije DI i EI izostavljaju iz prekidnog programa.

3.6 Spoljni (hardverski) prekidi

Mehanizam prekida u savremenim mikroprocesorima uopšten je tako što je proširen skup događaja koji izazivaju prekide. Na osnovu događaja koji ih izazivaju, prekidi mogu da se grupišu u:

- spoljne,
- unutrašnje i
- softverske.

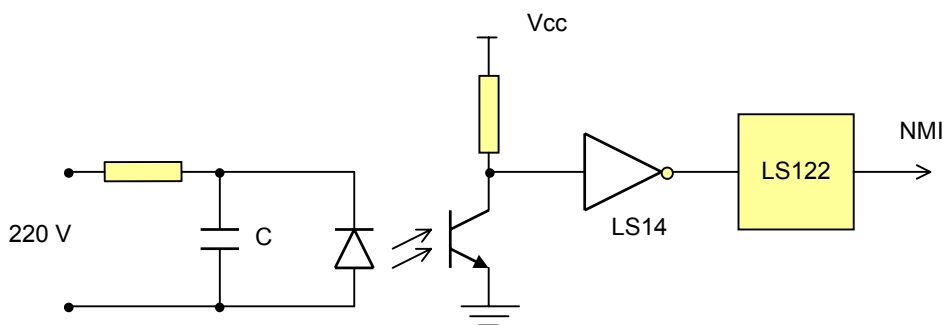
Spoljne prekide, objašnjene u prethodnim odeljcima, izazivaju događaji u okolini mikroprocesora koji aktiviraju ulazni signal prekida. Na primer, mikroprocesor x86 ima dva ulazna signala prekida, INT i NMI, Slika 3.17. Signal INT je standardni signal prekida, koji periferijske jedinice aktiviraju kada nastupi događaj koji se obrađuje programom za obradu prekida.



Slika 3.17: Mikroprocesor x86 ima dva ulazna signala prekida

Osnovna razlika između signala prekida INT i signala NMI (Non Maskable Interrupt) je u tome što mikroprocesor ne može da zabrani prekid NMI. Pored toga, pošto je višeg prioriteta, signal NMI koristi se za događaje koji su od vitalnog interesa za pravilan rad mikroračunarskog sistema. Tipična primena signala NMI je za javljanje događaja kao što je nestanak napona napajanja ili za grešku u radu sa memorijom ili periferijskim jedinicama.

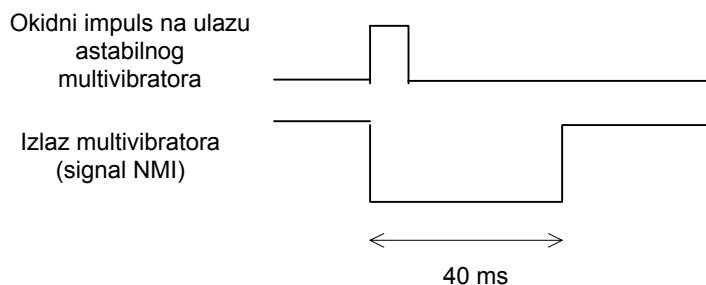
Kao ilustraciju korišćenja hardverskog signala prekida NMI posmatraćemo jednostavno kolo za detekciju prestanka napona napajanja mikroračunara, Slika 3.18.



Slika 3.18: Kolo za detekciju prestanka napona napajanja

Ulazni deo kola sastoji se iz opto sprega, odnosno od svetleće diode i foto tranzistora koji se koriste za galvansko odvajanje naizmeničnog mrežnog napona od ostatka mikroračunarskog kola. Svaka poluperioda napona napajanja aktivira svetleću diodu koja emituje svetlost i pobuđuje foto tranzistor. Impuls sa kolektora uobličava se Šmitovim invertorom 74LS14 i koristi za okidanje monostabilnog multivibratora 74LS122.

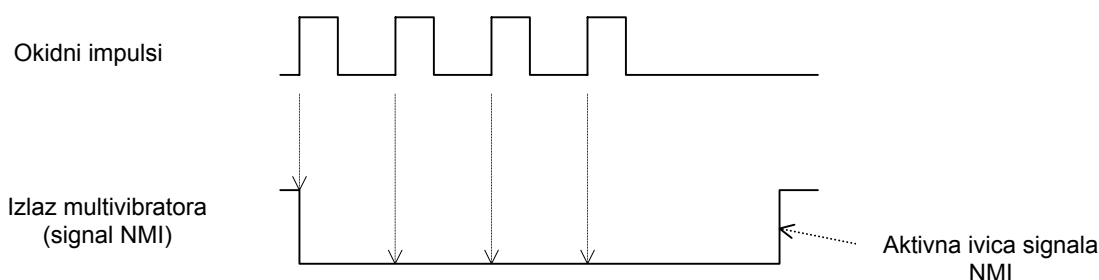
Vremenska konstanta monostabilnog multivibratora podesi se tako da negativni impuls traje 40 ms, što predstavlja dve periode mrežnog napona, Slika 3.19. Izlaz multivibratora vodi se neposredno na ulaz za nemaskirajući prekid NMI mikroprocesora x86. Treba imati u vidu da je signal NMI aktivan na prednjoj ivici (prelaz sa logičke 0 na logičku 1). Prema tome, u primeru koji prikazuje Slika 3.19, izlaz multivibratora bi posle 40 ms doveo do pojave prednje ivice signala NMI i mikroprocesor bi detektovao aktivan signal prekida NMI.



Slika 3.19: Negativni impuls multivibratora traje 40 ms

Pošto svaka poluperiode mrežnog napona generiše okidni impuls, astabilni multivibrator se okida u svakoj periodi mrežnog napona. Jednom kada multivibrator počne da generiše na izlazu negativni impuls trajanja 40 ms, posle 20 ms (jedna perioda mrežnog napona) stiže novi okidni impuls koji ponovo okida multivibrator. Prema tome, sve dok postoji mrežni napon okidni impulsi se generišu, okidaju monostabilni multivibrator i njegov izlaz je stalno na logičkoj 0.

Ukoliko se dogodi da nestane mrežni napon napajanja, okidni impulsi prestaju da se generišu i izlaz multivibrator će posle jedne periode mrežnog napona preći sa logičke 0 na logičku 1, Slika 3.19. Pošto je izlaz multivibratora doveden na ulaz NMI, mikroprocesor će ovu prednju ivicu protumačiti kao aktivan signal prekida i preći na program za obradu prekida NMI. Naravno, program za obradu prekida treba da preduzme korake predviđene za nestanak napona napajanja, na primer da što je moguće brže prekopira podatke iz operativne memorije na disk ili u memoriju koja ima baterijsko napajanje.



Slika 3.20: Kod prestanka napona napajanja multivibrator generiše prednju ivicu signala NMI

Kod detekcije spoljnjeg prekida NMI mikroprocesor x86 automatski koristi broj vektora prekida koji je jednak 2. Prema tome, početna adresa programa za obradu prekida NMI mora da bude stavljena u vektor prekida koji odgovara broju 2.

3.7 Unutrašnji prekidi

Kod savremenih mikroprocesora neki unutrašnji događaji mogu da dovedu do unutrašnjeg signala prekida koji se obrađuje na identičan način kao i spoljni signal prekida. Na primer, sledeći događaji često se koriste kao izvori unutrašnjih prekida:

- pokušaj deljenja nulom,
- pokušaj izvršavanja nedefinisanog koda operacije i
- pojava prekoračenja kod izvršavanja aritmetičkih operacija.

Smisao unutrašnjih prekida je u poboljšanju stabilnosti ili robusnosti rada mikroprocesora. Ukoliko mikroprocesor iz nekog razloga pokuša da izvrši nepostojeći kod operacije, posledice su nepredvidive, a najčešće će mikro mikroračunarski sistem da pređe u nedefinisano stanje i prestaje da radi na predviđeni način. Jedini izlaz iz takvog stanje je resetovanje, odnosno prevođenje sistema u početno stanje, pri čemu se svi podaci u operativnoj memoriji nepovratno gube.

U ovakvom primeru, mnogo celishodniji način reagovanja mikroprocesora je da se pokušaj izvršenja nepostojećeg koda operacije interpretira kao događaj koji izaziva prekid i da se zatim prekid obradi tako da mikroračunarski sistem nastavi sa radom. Mikroračunarski sistem može da programsku kontrolu vrati operativnom sistemu i na taj način omogući ostalim programima da nastave normalno izvršavanje.

Posebna vrsta unutrašnjeg prekida je izvršenje instrukcija ‘korak-po-korak’, engleski termin ‘trace’ ili ‘single-step’. U režimu rada korak-po-korak mikroprocesor generiše unutrašnji prekid po završetku svake instrukcije tekućeg programa. Ideja unutrašnjeg prekida korak-po-korak je da se omogući korisniku da po završetku svake instrukcije aktivira poseban prekidni program u kome može da se proverava način izvršavanja tekućeg programa.

Unutrašnji prekid korak-po-korak zasniva se na posebnom bitu T u sastavu indikatorskog registra: ukoliko je bit T na logičkoj 0 unutrašnji prekid korak-po-korak nije dozvoljen, a ukoliko je T na logičkoj 1 unutrašnji prekid korak-po-korak aktivira se po završetku svake instrukcije tekućeg programa, Slika 3.21.

V	C	S	Z	...	I	T	...	
---	---	---	---	-----	---	---	-----	--

Slika 3.21: Ako je bit T indikatorskog registra na logičkoj 1, prekid korak-po-korak je dozvoljen

Program za obradu prekida korak-po-korak (jedan primer takvog programa je DEBUG) obično je napravljen tako da korisniku omogućava sledeće operacije:

- prikazivanje i izmenu sadržaja registara mikroprocesora,
- prikazivanje i izmenu sadržaja zahtevanog dela memorije,
- nastavak izvršavanja tekućeg programa.

Naravno, prekid korak-po-korak mora da se zabrani kod prelaska na program za obradu prekida. U suprotnom, već posle prve instrukcije programa za obradu prekida aktivirao bi se prekid korak-po-korak što bi dovelo do beskrajnog aktiviranja i prekidanje ovog programa. Po pravilu, ovaj problem jednostavno se rešava tako što mikroprocesor kod prekida korak-po-korak u bit T upisuje 0. Pošto se na ovaj način menja sadržaj indikatorskog registra, odziv mikroprocesora na signal prekida obično obuhvata i čuvanje sadržaja indikatorskog registra na steku.

Uz pretpostavku da broj vektora prekida istovremeno predstavlja adresu odgovarajućeg vektora prekida, mikroprogram prelaska na program za obradu prekida sa ovom modifikacijom izgleda ovako:

```

SP ← SP-1          ; mesto na steku za indikatorski registar
M[SP] ← ind.reg.   ; indikatorski registar na stek

I ← 0              ; zabrana svih prekida
T ← 0              ; zabrana prekida korak-po-korak

SP ← SP-1          ; mesto na steku za povratnu adresu
M[SP] ← PC         ; povratna adresa iz PC na stek

INTA ← 1           ; aktiviranje signal INTA
temp ← mag.pod.    ; temp = broj vektora prekida (sa magistrale pod)
PC ← M[temp]       ; PC = pocetna adresa prekidnog programa
INTA ← 0           ; signal INTA prevesti u neaktivno stanje

```

Naravno, instrukcija RETI povratka iz prekidnog programa mora da vrati sadržaj indikatorskog registra sa steka:

```

PC ← M[SP]         ; povratna adresa u PC
SP ← SP+1          ; SP pokazuje na sacuvani sadraj ind. registra

ind.reg. ← M[SP]   ; restauracija sadraja indikatorskog registra
SP ← SP+1          ; SP pokazuje na vrh steka

```

Kod mikroprocesora x86 prekid korak-po-korak uvek je vezan sa brojem vektora prekida 1.

3.8 Softverski prekidi

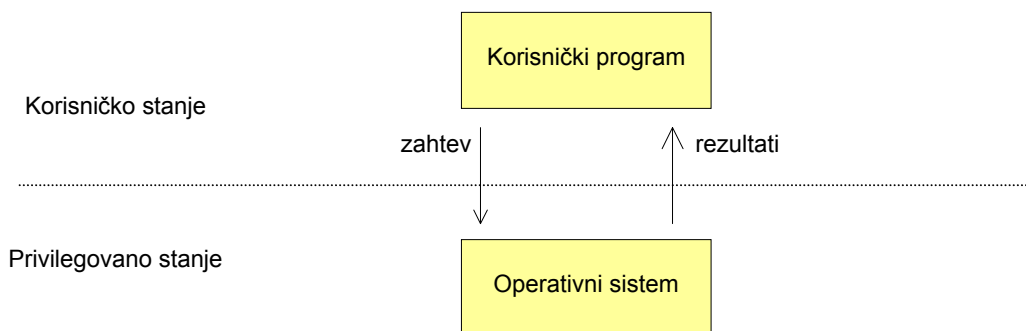
Softverski prekid uvodi se radi pvišenja stepena zaštite u mikroračunarskom sistemu. Mikroračunarski sistem treba da nastavi da ispravno radi čak i u slučaju da korisnik napravi namernu ili nenamernu grešku. Iz iskustva se zna da su programi koji obavljaju ulazno-izlazne operacije i rade sa perifernim jedinicama složeni u odnosu na standardne korisničke programe. Pored toga, programi koji neposredno pristupaju perifernim jedinicama mogu da dovedu do fatalnih grešaka: da izbrišu fajlove sa diska ili izbrišu podatke koji su važni za normalan rad sistema.

Jedan način da se ovi problemi reše je uvođenje dva generalizovana stanja mikroprocesora: *korisničko* i *privilegovano*. U datom trenutku mikroprocesor može da se nalazi u samo jednom od ova dva stanja.

U korisničkom stanju izvršavaju se svi korisnički (aplikativni) programi. U ovom stanju mikroprocesor može da izvrši podskup instrukcija i može da pristupi samo nekim resursima mikroračunarskog sistema. Između ostalog, pristup nije dozvoljen perifernim jedinicama, sistemskim programima, sistemskim podacima i sično. U korisničkom stanju, na primer, mikroprocesor ne može da izvršava ulazno-izlazne operacije. Na ovaj način, mikroračunarski sistem je zaštićen od grešaka koje mogu da se nalaze u korisničkim programima. Takođe, mikroračunarski sistem zaštićen je od korisnika koji namerno pokušaju da naruše ispravnost sistemskih informacija, programa i drugih resursa sistema.

Uvođenjem korisničkog stanja i ograničenjem operacija koje korisnički programi mogu da izvrše, mikroračunarski sistem je zaštićen od korisnika. Ostale aktivnosti neophodne za rad mikroračunara obavlja operativni sistem – skup programskih modula koji upravljaju resursima sistema. Podrazumeva se da je operativni sistem kvalitetno projektovan i realizovan, da je temeljno testiran i obezbeđuje pouzdan i stabilan rad mikroračunarskog sistema. Operativni sistem izvršava se u privilegovanom stanju, u kome je mikroprocesoru dozvoljeno da izvršava sve instrukcije i pristupa svim resursima mikroračunarskog sistema.

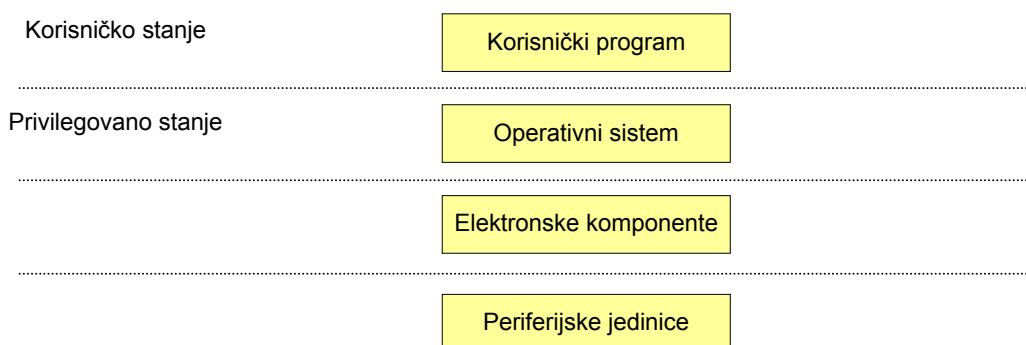
Jedan problem u ovakvom pristupu je što korisnički programi moraju da koriste kritične resurse sistema, kao što su ulazno-izlazne jedinice. Na primer, korisnički program mora da koristi memorijski prostor na disku za čuvanje podataka i čitanje prethodno zapisanih podataka ili mora da koristi štampač za štampanje izveštaja. Rešenje problema je jednostavno i sastoji se u tome da operativni sistem obavlja operacije koje su zabranjene korisničkim programima. U jednostavnom slučaju, korisnički program pošalje zahtev operativnom sistemu sa informacijama o zadatku koji treba da se obavi, zatim operativni sistem obavi traženi zadatak i vrati korisničkom programu informacije o rezultatima dobijenim obavljanjem zadatka, Slika 3.22.



Slika 3.22: Operativni sistem obavlja zadatke koje zehteva korisnički program

Na primer, korisnički program može da zahteva od operativnog sistema da odštampa neki tekst. U tom slučaju, korisnički program pošalje operativnom sistemu zahtev za štampanje zajedno sa informacijama o fajlu u kome se nalazi tekst. Operativni sistem uzme tekst iz fajla i zatim, preko ulazno-izlaznog podsistema generiše sve signale koji su neophodni da se zahtevani tekst odštampa. Po završetku štampanja korisničkom programu prosleđuje se informacija o uspešnosti štampanja.

Iz navedenog primera jasno je da operativni sistem mora da ima neposrednu interakciju sa elektronskim komponentama mikroračunarskog sistema. Ova interakcija obavlja se na fizičkom nivou, tako što operativni sistem, preko elektronskih komponenata, generiše upravljačke signale za perifernijske jedinice i preko istog podsistema prima signale od perifernih jedinica.

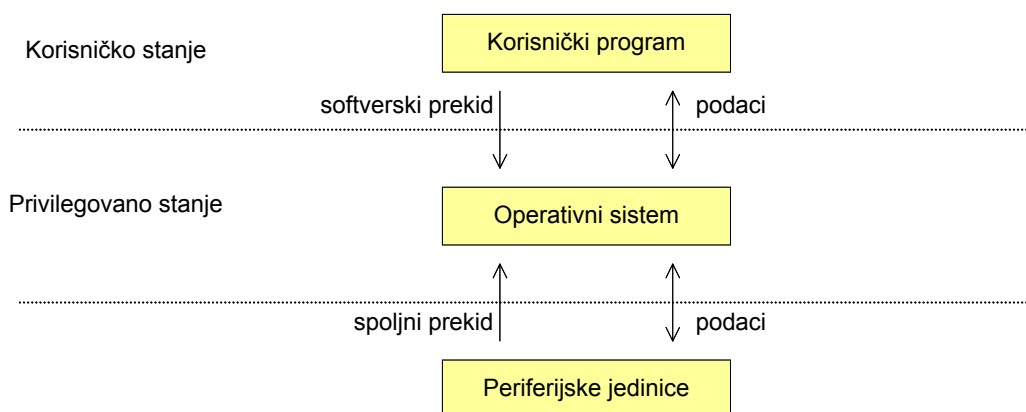


Slika 3.23: Slojevi u mikroračunarskom sistemu

Na ovaj način dolazi se do slojeva softverskih i elektromehaničkih komponenata mikroračunarskog sistema, Slika 3.23. Osnovna odlika ovakve strukture je da samo susedni slojevi mogu da međusobno komuniciraju. Tako, na primer, korisnički program može da komunicira samo sa operativnim sistemom i ne može da neposredno pristupi elektronskim komponentama. Radi jednostavnosti, u ovom delu teksta smatraćemo da perifernijske jedinice, kao što je disk ili štampač, obuhvataju i pridružene elektronske komponente (kontrolere).

Operativni sistem, koji upravlja perifernim jedinicama, mora da obrađuje spoljne prekide od perifernih jedinica. U pojednostavljenom obliku, operativni sistem sastoji se od skupa programa za obradu spoljnih prekida. Sasvim je jasno da aktiviranje spoljnog prekida dovodi do prenosa programske kontrole na operativni sistem.

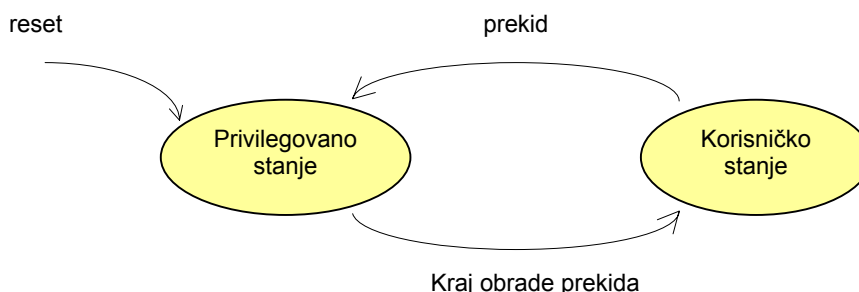
Da bi se standardizovao pristup operativnom sistemu, uvodi se *softverski prekid*, specijalna instrukcija koja ima isti efekat kao i aktiviranje spoljnog ili unutrašnjeg prekida. Izvršavanjem instrukcije za softverski prekid, korisnički program aktivira program za obradu prekida koji se nalazi u okviru operativnog sistema. Program za obradu prekida dobija od korisničkog programa sve parametre neophodne za izvršenje zahtevane operacije i zatim izvršava tu operaciju.



Slika 3.24: Operativni sistem aktivira se prekidom

Prema tome, operativni sistem može se aktivirati samo preko mehanizma prekida, Slika 3.24. Korisnički program prenosi programsku kontrolu operativnom sistemu izvršavanjem instrukcije softverskog prekida, a perifernije jedinice aktiviraju operativni sistem generisanjem spoljnog signala prekida. Naravno, između operativnog sistema sa jedne strane i korisničkog programa i perifernih jedinica, sa druge strane, prenose se i podaci, ali nećemo ulaziti u te detalje.

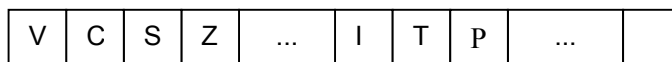
Ako korisničko i privilegovano stanje mikroprocesora predstavimo ovalnim linijama, a prelaze iz jednog u drugo stanje predstavimo strlicama, dobijamo graf generalizovanih stanja mikroprocesora, Slika 3.25-. Kod uključenja ili reset, mikroprocesor se prvo nalazi u privilegovanom stanju i izvrši deo operativnog sistema koji obavlja inicijalizaciju mikroračunarskog sistema. Po završetku inicijalizacije ili programa za obradu prekida, mikroprocesor prelazi u korisničko stanje. Ako se u bilo kom trenutku dogodi prekid, mikroprocesor prelazi u privilegovano stanje i vrši obradu prekida.



Slika 3.25: Graf stanja mikroprocesora

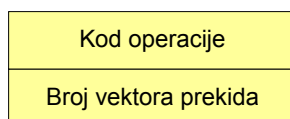
Pošto omogućava poziv uslužnih rutina operativnog sistema, softverski prekid još se naziva 'sistemski poziv'.

Stanje mikroprocesora obično je određeno jednim bitom (na primer sa oznakom P) indikatorskog registra, Slika 3.26. Kada je bit P na logičkoj 1, mikroprocesor je u privilegovanom stanju, a kada je bit P na logičkoj 0, bit P je u korisničkom stanju. Da bi se sprečila izmena stanja bita P, pristup indikatorskom registru obično je instrukcija koja može da se izvrši samo u privilegovanom stanju.



Slika 3.26: Indikatorski registar sa bitom P koji označava stanje mikroprocesora

Slika 3.27 prikazuje format instrukcije za softverski prekid. Pored koda operacije, u sastavu instrukcije nalazi se i broj vektora prekida. Mikroprogram izvršne faze instrukcije za softverski prekid veoma je sličan mikroprogramu koji određuje odziv mikroprocesora na spoljni prekid. Jedina razlika je što mikroprocesor, umesto da uzima broj vektora prekida od periferijske jedinice, koristi broj vektora prekida koji predstavlja operand instrukcije za softverski prekid. Može se reći da izvršna faza instrukcije za softverski prekid simulira odziv mikroprocesora na signal prekida.



Slika 3.27: Format instrukcije za softverski prekid

Uz istu pretpostavku da broj vektora prekida istovremeno predstavlja adresu odgovarajućeg vektora prekida, mikroprogram za izvršnu fazu instrukcije za softverski prekid ima sledeći izgled.

```

SP ← SP-1          ; mesto na steku za indikatorski registar
M[SP] ← ind.reg.   ; indikatorski registar na stek

I ← 0              ; zabrana svih prekida
T ← 0              ; zabrana prekida korak-po-korak
P ← 1              ; prelazak u privilegovano stanje

temp ← M[PC]        ; temp = broj vektora prekida
PC ← PC+1           ; u PC je povratna adresa

SP ← SP-1          ; mesto na steku za povratnu adresu
M[SP] ← PC          ; povratna adresa iz PC na stek

PC ← M[temp]        ; PC = pocetna adresa prekidnog programa

```

Mikroprogram izvršne faze instrukcije povratka iz prekidnog programa može da prevede mikroprocesor u korisničko stanje:

```

PC ← M[SP]          ; povratna adresa u PC
SP ← SP+1           ; SP pokazuje na sacuvani sadraj ind. registra
ind.reg. ← M[SP]    ; restauracija sadraja indikatorskog registra
P ← 0               ; prelazak u korisničko stanje
SP ← SP+1           ; SP pokazuje na vrh steka

```

Naravno, ukoliko instrukcija povratka iz prekidnog programa ne prevodi automatski mikroprocesor u korisničko stanje, onda operativni sistem mora softverskim putem da upravlja sadržajem bita P.

Mikroprocesor x86 ima instrukciju INT n, koja generiše softverski prekid sa brojem vektora prekida koji je jednak n. Kod IBM PC računara instrukcija INT sa operandom 21H (broj vektora prekida 21 heksadecimalno) predstavlja poziv operativnom sistemu DOS. Po usvojenoj konvenciji,

registar AH sadrži kodirani broj operacije (funkcije) koju DOS treba da izvrši. Pre izvršenja instrukcije INT 21H treba staviti traženi broj operacije (funkciju) u registar AH.

Na primer, broj 08H predstavlja operaciju ulaza sa tastature. Kada se pozove operativni sistem DOS sa registrom AH u koji je smešten broj 08H, DOS će u registru AL vratiti ASCII znak koji je unesen preko tastature. Sledeće dve instrukcije smeštaju broj operacije u AL i pozivaju DOS:

```
MOV AH,08H      ; operacija ulaza sa tastature
INT 21H          ; sistemski poziv DOS-u
```

Operacija kodirana brojem 02H prikazuje na ekranu znak čiji ASCII kod je u registru DL. Kombinacijom operacija za čitanje sa tastature i prikazivanja na ekranu, dobija se sledeći niz instrukcija koje učitani znak sa tastature prikazuju na ekranu.

```
MOV AH,08H      ; operacija ulaza sa tastature
INT 21H          ; sistemski poziv, znak sa tastature je u AL
MOV DL,AL        ; ASCII znak sa tastature u DL
MOV AH,02H       ; broj operacije prikaza znaka na ekranu
INT 21H          ; znak se prikazuje na ekranu
```

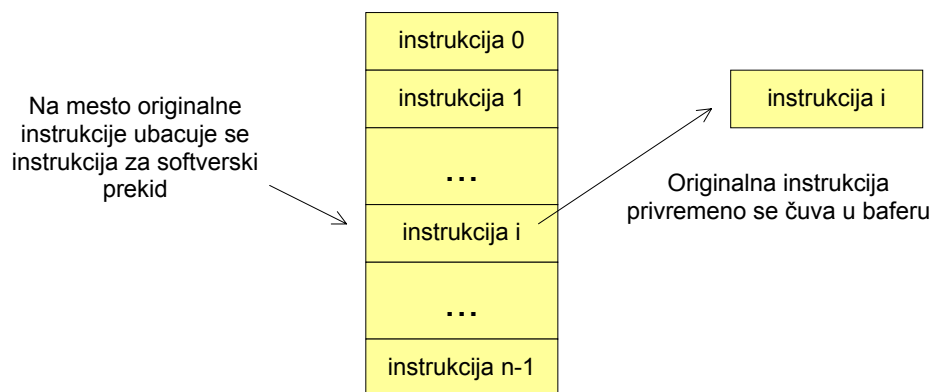
Softverski prekid može veoma korisno da posluži kod testiranja i ispravljanja grešaka u programu. Već smo rekli da izvršavanje programa 'korak-po-korak' omogućava praćenje izvršenja programa posle svake instrukcije. Problem nastaje kada program uđe u petlju koju izvršava veliki broj puta. Već posle nekoliko prolaza korisniku je jasno da li petlja radi ispravno. Međutim, da bi se proverio ostali deo programa, korisnik mora da sačeka da se petlja kompletno završi korak-po-korak, da bi mogao da nastavi sa testiranjem programa.

U ovakvim slučajevima, instrukcija za softverski prekid umetne se na mesto u programu na kome korisnik hoće da proveriti stanje izvršenja programa. Mesto na kome se nalazi instrukcija za softverski prekid naziva se *tačka prekida* (breakpoint). Kada mikroprocesor dođe do tačke prekida i izvrši instrukciju za softverski prekid, aktivira se program za obradu prekida (na primer DEBUG) koji korisniku omogućava sledeće akcije:

- uvid u sadržaje registara i memorije,
- izmenu sadržaja registara i memorije,
- postavljanje novih tačaka prekida i
- brisanje tačaka prekida.

Naravno, u programu mogu da budu ni jedna, jedna ili više tačaka prekida. Tačka prekida realizuje se tako što se originalna instrukcija programa na kome se postavlja tačka prekida zameni sa posebnom instrukcijom za softverski prekid koja ima samo kod operacije, Slika 3.28. Pošto ova instrukcija zauzima samo jedan bajt, onda može da zameni bilo koju drugu instrukciju u programu.

Aktiviranjem programa za testiranje i otklanjanje grešaka (DEBUG) korisnik prvo postavlja prekidne tačke na mestima na kojima želi da proverava stanje programa. Program za testiranje i otklanjanje grešaka originalne instrukcije čuva u privremenim baferima, na njihovo mesto stavlja instrukcije za softverski prekid i aktivira korisnički program. Kada mikroprocesor dođe do prekidne tačke, ponovo se aktivira program za testiranje i otklanjanje grešaka i korisnik sada može da proverava stanje programa.



Slika 3.28: Ubacivanje prekidne tačke na mesto instrukcije sa rednim brojem i

Kada korisnik završi proveru stanja programa, originalna instrukcija se vraća na svoje mesto i korisnički program nastavlja sa izvršenjem dok ne dođe do sledeće prekidne tačke u kojoj se celi ciklus ponavlja.

Kod mikroprocesora x86 instrukcija INT 3 za softverski prekid sa brojem vektora prekida 3 koristi se za realizaciju prekidne tačke. Zanimljivo je da ova instrukcija ima samo jedan bajt i kod operacije koji se razlikuje od instrukcija sa softverskim prekidom koji je različit od 3.

4. Operativna memorija

Memorija u mikroračunarskom sistemu može da se podeli na operativnu i masovnu memoriju. Mikroprocesor može da neposredno pristupi i izvrši program i obrađuje podatke koji su smešteni u operativnoj memoriji. Za razliku od operativne memorije, mikroprocesor ne može da neposredno pristupa masovnoj memoriji. Da bi izvršio program koji je smešten na masovnoj memoriji, kao što je disk, mikroračunarski sistem prvo mora da taj program prebaci sa diska u operativnu memoriju i tek onda da ga izvrši.

U ovom poglavlju izložene su osnovne karakteristike i projektovanje podsistema operativne memorije.

4.1 Organizacija operativne memorije

Sve informacije u mikroračunarskom sistemu predstavljaju se nizovima binarnih cifara, 0 i 1. Najmanja količina informacije predstavljena je jednom binarnom cifrom i naziva se bit (akronim od engleskog naziva *Binary Digit*). Zadatak memorije je da pamti binarne informacije i da ih, na zahtev, stavi na raspolaganje nekoj drugoj jedinici računara.

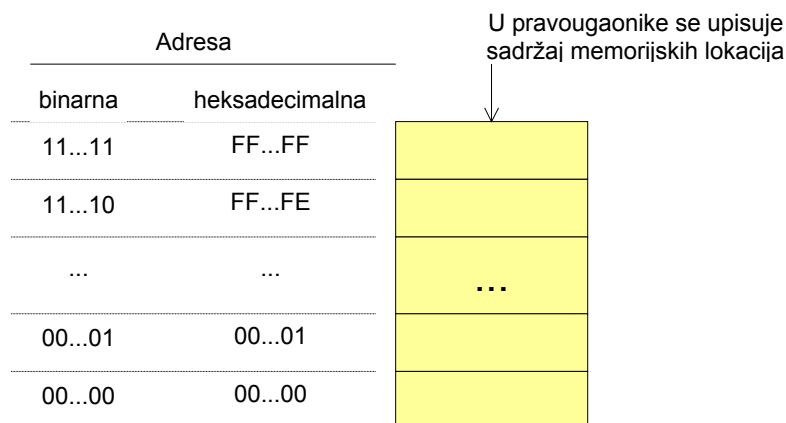
Osnovni sastavni deo memorije je memorijska ćelija, koja može biti u jednom od dva različita stanja kodirana nulom (0) i jedinicom (1). Dakle, memorijska ćelija može da pamti informaciju od jednog bita. Realizacija memorijskih ćelija zavisi od tehnologije izrade, a osnovne osobine ćelija su mogućnost prevođenja ćelija u stanje 0 ili 1, održavanju i kasnijem čitanju stanja memorijskih ćelija.

Memorija ima definisane dve operacije: upis i čitanje. Operacijom upisa binarna informacija smešta se u memoriju tako što se memorijske ćelije prevode u stanja koja predstavljaju binarne cifre ulazne informacije. Podrazumeva se da memorijske ćelije posle operacije upisa ostaju u stanjima u koja su prevedena operacijom upisa, što znači da 'pamte' upisanu informaciju.

Operacijom čitanja detektuju se stanja memorijskih ćelija i time 'čita' informacija koja je prethodno upisana u memoriju. Termin 'pristup' koristi se za operaciju čitanja ili upisa.

Radi veće brzine prenosa podataka memorijske ćelije su grupisane tako da mikroprocesor pristupa paralelno svim ćelijama u jednoj grupi. Grupa memorijskih ćelija kojima se paralelno pristupa naziva se *memorijska lokacija* ili *memorijski registar*. U memorijske lokacije smeštaju se informacije od više bita, koje se nazivaju memorijske reči. Dužina memorijske lokacije je broj memorijskih ćelija u lokaciji i jednak je broju bita memorijske reči koja može da se smesti u lokaciju. Reč dužine 8 bita naziva se bajt.

Memorija je organizovana u obliku linearnog niza memorijskih lokacija. Svakoj memorijskoj lokaciji pridružen je jedinstven binarni broj koji se naziva adresa. Organizacija memorije određena je brojem memorijskih lokacija, pridruženim adresama i brojem memorijskih ćelija u svakoj lokaciji. Slika 4.1 prikazuje grafički simbol za memoriju. Svaka memorijska lokacija predstavljena je pravougaonikom, pored koga se piše adresa lokacije, a u pravougaonik upisuje sadržaj memorijske lokacije.



Slika 4.1: Grafičko predstavljanje memorije

Organizacija memorije iskazuje se uređenim parom (l,n), gde je l broj memorijskih lokacija, a n broj memorijskih ćelija u svakoj lokaciji. Broj l je iz skupa brojeva 2^m gde je m broj adresnih signala memorije, pa je uobičajeno da se organizacija memorije predstavlja u obliku ($2^m \times n$). Radi jednostavnijeg pisanja uvedene su sledeće jedinice:

$$\begin{aligned}
 1 \text{ k (kilo)} &= 2^{10} \\
 1 \text{ M (mega)} &= 2^{20} = 2^{10} \text{ k} \\
 1 \text{ G (giga)} &= 2^{30} = 2^{10} \text{ M} \\
 1 \text{ T (tera)} &= 2^{40} = 2^{10} \text{ G}
 \end{aligned}$$

Primeri organizacije memorije su (1k x 4), (4k x 8), (16k x 8), (256k x 1) i (1M x 1).

Kapacitet memorije je količina informacije izražene u bitima koja može da se upiše u memoriju. Kapacitet se jednostavno izračunava tako što se broj memorijskih lokacija pomnoži brojem ćelija u jednoj lokaciji. U sledećoj tabeli dati su neki primeri organizacije memorije i kapaciteti izraženi u bitima i bajtovima.

Organizacija	Kapacitet (u bitima)	Kapacitet (u bajtovima)
1k x 4	4k	512
4k x 8	32k	4k
16k x 8	128k	16k
256k x 1	256k	32k
1M x 1	1M	128k

Prilikom pristupa neophodno je memoriji saopštiti adresu lokacije kojoj se pristupa i vrstu operacije koju memorija treba da izvrši. Zato memorija poseduje linije za prenos adresnih signala, za prenos memorijskih reči i za upravljačke signale. Dva jednostavna modele memorije prikazuje Slika 4.2, levo.



Slika 4.2: Model memorije sa signalima R i W (levo) i signalima CS i R/W (desno)

Adresne i upravljačke linije imaju smer prenosa signala prema memoriji. Linije za prenos signala podataka u opštem slučaju su dvosmerne tako da kod operacije upisa signali se prenose prema memoriji, a kod operacije čitanja imaju suprotan smer.

Treba primetiti da upravljački signali mogu biti aktivni kada su na logičkoj 0 ili na logičkoj 1. Aktivna logička 0 obeležava se tako što se iznad naziva upravljačkog signala stavi crtica. Isključivo zbog ograničenja editora teksta, u ovim skriptama će se podvući naziv upravljačkog signala koji je aktivan na logičkoj 0.

U praksi se često sreću memorije koje umesto upravljačkih signala R i W imaju upravljačke signale CS i $\overline{R/W}$. Upravljački signal CS aktivira memoriju, a signal $\overline{R/W}$ određuje operaciju čitanja (ako je na logičkoj 1) ili upisa (ako je na logičkoj 0). U sledećoj tabeli date su kombinacije upravljačkih signala i njihovo značenje. Iz tabele se vidi da signali R i W imaju zabranjenu kombinaciju ($R=W=1$) dok signali CS i $\overline{R/W}$ nemaju zabranjene kombinacije.

Sa signalima R i W		Sa signalima CS i $\overline{R/W}$		Operacija
R	W	CS	$\overline{R/W}$	
0	0	0	X	nema operacije
0	1	1	0	upis
1	0	1	1	čitanje
1	1			nije dozvoljeno

Ako sa A označimo adresu na adresnim linijama, sa $M[A]$ memorijsku reč koja je u lokaciji sa adresom A, a MR memorijsku reč koja je na linijama za podatke, onda se memorija opisuje sledećim mikrooperacijama:

$R: MR \leftarrow M[A]$; operacija čitanja
 $W: M[A] \leftarrow MR$; operacija upisa

Alternativno, ako se koriste upravljački signali CS i $\overline{R/W}$, mikrooperacije memorije izgledaju ovako:

$CS \wedge (\overline{R/W}=1): MR \leftarrow M[A]$; operacija čitanja
 $CS \wedge (\overline{R/W}=0): M[A] \leftarrow MR$; operacija upisa

Najznačajniji tehnološki parametri memorije su tehnologija izrade, brzina, način pristupa, trajnost upisanih informacija i mogućnost izvršavanja operacije upisa.

Tehnologija izrade. Memorija se može realizovati na različite načine u zavisnosti od tehnologije izrade memorijskih ćelija. Realizacija memorijskih ćelija mora biti takva da omogući pouzdano raspoznavanje dva različita stanja. Na primer memorijske ćelije mogu da se realizuju na sledeće načine:

- Poluprovodnički flip-flovi, koji mogu biti u jednom od dva stabilna stanja.
- Delići feromagnetnog materijala, koji mogu biti namagnetisani u jednom od dva smera.
- Tačke na optičkom disku mogu biti svetle ili tamne, odnosno odbijati ili upijati laserski snop svetlosti.
- Kondenzatori malih dimenzija mogu biti napunjeni ili prazni.
- Veze između vrsta i kolona u matrici mogu biti uspostavljene ili prekinute.

Od tehnologije izrade zavise svi ostali parametri memorije. Savremena mikroprocesori mogu da se neposredno spregnu samo sa poluprovodničkim memorijama. Memorije napravljenije primenom drugih tehnologija zahtevaju dodatna elektronska kola ili elektromehaničke podsisteme koji prilagođavaju memoriju uslovima koje zahteva mikroprocesor.

Brzina memorije određuje se na osnovu vremena koje protekne od trenutka kada su aktivirani upravljački i adresni signali do trenutka kada se, kod operacije upisa, ulazna memorijska reč smesti u adresiranu memorijsku lokaciju, odnosno kod operacije čitanja, kada se memorijska reč iz adresirane memorijske lokacije prenese na spoljne linije za podatke. Ova definicija je neformalna i odnosi se na najnepovoljniji slučaj – drugim rečima navedena vremena su takva da garantuju da će memorija obaviti predviđene operacije.

Način pristupa memoriji može biti sekvencijalan ili slučajan. Memorije sa sekvencijalnim pristupom obično imaju upisno/učitnu glavu koja obavlja operaciju upisa i čitanja nad elementom koji je u kontaktu sa glavom. Kod sekvencijalnog pristupa, upisno/učitna glava mora preći preko svih elemenata (lokacija) koji se nalaze između elementa na kome se trenutno nalazi glava do zahtevanog elementa. Tipičan primer memorije sa sekvencijalnim pristupom je magnetna traka kod koje upisno/učitna glava mora da pređe preko dela trake između trenutnog položaja glave do adresiranog elementa na traci. Očigledno je da vreme pristupa kod sekvencijalnih memorija zavisi od trenutne pozicije upisno/učitne glave u odnosu na element na kome se pristupa.

Kod memorija sa slučajnim pristupom vreme pristupa za sve lokacije je isto. Primeri ovakvih memorija su poluprovodničke memorije tipa RAM i ROM.

Neke vrste masovnih memorija, kao što su optički i feromagnetni diskovi, imaju način pristupa koji je poboljšan u odnosu na sekvencijalni pristup. Pristup adresiranom elementu obavlja se u dva koraka: u prvom koraku se upisno/učitna glava direktno pozicionira iznad kružne staze na kojoj se nalazi traženi element, a zatim se sekvencijalnim pristupom dolazi do tog elementa.

Trajnost upisanih informacija Razlikujemo memorije kod kojih nestanak napona napajanja dovodi do gubitka svih zapisanih informacija i memorije koje zadržavaju sadržaj posle prestanka napona napajanja. Praktično sve optičke memorije i memorije sa feromagnetnim ćelijama, na primer magnetni diskovi i magnetne trake, zadržavaju sadržaj posle prestanka napona napajanja. Poluprovodničke memorije tipa ROM takođe zadržavaju sadržaj posle prestanka napona napajanja.

Poluprovodničke memorije tipa RAM gube sadržaj kada se isključi napon napajanja. Međutim, klasi memorija, koje se nazivaju dinamički RAM, nije dovoljan neprekidan napon napajanja da bi se upisani sadržaj održao duže vreme. Memorijske ćelije dinamičkog RAM-a napravljene su u obliku kondenzatora minijaturnih dimenzija. Stanja memorijskih ćelija razlikuje se po količini naelektrisanja u kondenzatoru: ako je kondenzator napunjen, ćelija je u stanju logičke 1, a ako je prazan, ćelija je u stanju logičke 0. Napunjen kondenzator se vremenom prazni i time se gubi sadržaj dinamičkog RAM-a. Stoga je neophodno periodično osvežavanje, koje se ostvaruje tako što se redom čitaju sve memorijske lokacije. Kod operacije čitanja, elektronska kola za čitanje detektuju stanje kondenzatora, proslede signale na linije za podatke, a zatim pročitani sadržaj ponovo zapišu u istu memorijsku lokaciju i na taj način regenerišu napunjenost kondenzatora. Osvežavanje dinamičkih RAM-ova standardno se radi u intervalima reda veličine 2 ms, što znači da se u tom vremenskom intervalu moraju osvežiti sve memorijske ćelije.

Mogućnost upisa U nekim praktičnim primenama operacija upisa u memoriju nije neophodna. Na primer, programi za obavljanje operacija u kalkulatoru trajno su zapisani u memoriji kalkulatora i nema potrebe da se menjaju. U ovakvim primenama koristi se memorija tipa ROM (Read Only Memory) kod kojih se sadržaj jednom upiše i kasnije se ne menja. Kod poluprovodničkih memorija tipa ROM sadržaj se formira u toku izrade integrisanog kola. Postoji varijanta programabilnog poluprovodničkog ROM-a (PROM), kod koga su sve ćelije u jednom stanju, a korisnik može proizvoljne ćelije posebnim postupkom prevesti u drugo stanje. Postupak prevođenja ćelija u drugo stanje nije inverzan, odnosno memorijske ćelije ne mogu se vratiti u početno stanje. Postupak upisa

sadržaja u PROM naziva se programiranje i obavlja se primenom posebnog uređaja koji se naziva PROM programator.

Poluprovodničke memorije tipa EPROM (Erasable ROM) mogu se programirati (upisati novi sadržaj primenom EPROM programatora) i poseduju mogućnost brisanja sadržaja, odnosno prevođenja memorijskih ćelija u početno stanje.

Programiranje ROM-a obavlja se tako što se integrisano kolo stavi u podnožje programatora i zatim aktivira postupak kojim se izabrane memorijske ćelije prevode u traženo stanje. Prevođenje ćelija najčešće se svodi na primenu električnih struja koje spaljuju veze između vrsta i kolona kod PROM-a ili na primenu električnog polja koje prevodi nosioce naelektrisanja na provodna ostrvca u izolatoru kod EPROM-a. U oba slučaja postupak je relativno spor. Važno je zapaziti da programiranje ROM-a ne može obaviti mikroprocesor koji je neposredno spregnut sa PROM-om ili EPROM-om u mikroračunarskom sistemu.

Razlikuju se dva načina brisanja sadržaja, jedan koji koristi ultravioletnu svetlost (UV EPROM) i drugi koji koristi efekat električnog polja (EEPROM). Postupak brisanja UV EPROM-a sastoji se u izlaganju integrisanog kola ultraljubičastoj svetlosti koja ima dovoljnu energiju da neutrališe naelektrisanje provodnih ostrvaca u izolatoru. Kod EEPROM-a koristi se električno polje za neutralisanje naelektrisanja provodnih ostrvaca.

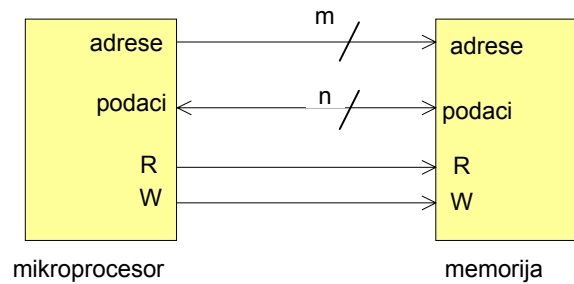
Kod realizacije memorijskog podsistema mikroračunara često se postavljaju konflikti zahtevi: veliki kapacitet, visoka brzina i niska cena. Zato se memorijski podsistem realizuje iz dva dela: operativne i masovne memorije. Operativna memorija realizuje se od poluprovodničkih memorijskih komponenata, koje mogu da se direktno spregnu sa mikroprocesorom. Poluprovodnička memorija mora da bude dovoljno brza da ne usporava rad mikroprocesora koji često pristupa operativnoj memoriji. Naravno, nepovoljne karakteristike poluprovodničke memorije su mali kapacitet, visoka cena i, u slučaju RAM-a, gubitak sadržaja kod prestanka napona napajanja.

Masovna memorija realizuje se od komponenata koje imaju veliki kapacitet i nisku cenu, kao što su na primer feromagnetni diskovi. Naravno, masovne memorije su po pravilu spore i ne mogu da se neposredno spregnu sa mikroprocesorom.

4.2 Sprega mikroprocesora i memorije

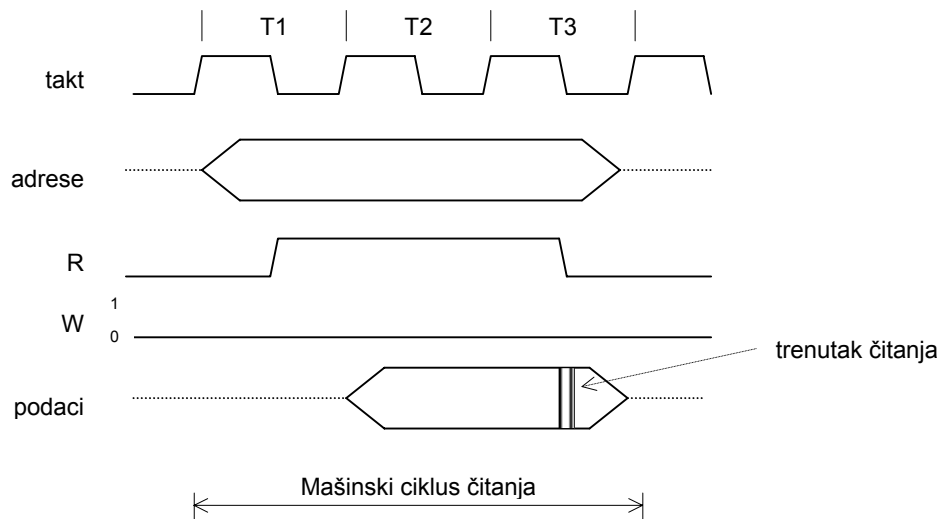
Kao što je već ranije rečeno, mikroprocesor i poluprovodnička memorija su međusobno kompatibilni u smislu da se mogu neposredno međusobno povezati. Neposredno veza znači da se adresni signali mikroprocesora vežu sa adresnim signalima memorije, signali podataka mikroprocesora sa signalima podataka memorije i upravljački signali mikroprocesora sa odgovarajućim signalima memorije.

Slika 4.3 prikazuje neposrednu vezu mikroprocesora i memorije u slučaju da imaju isti broj adresnih signala (m) i signala za podatke (n) i da su upravljački signali mikroprocesora i memorije R i W .



Slika 4.3: Neposredna veza mikroprocesora i memorije koji imaju isti broj adresnih signala (m) i signala za podatke (n)

Sprega mikroprocesora i memorije može biti *sinhrona* i *asinhrona*. Kod sinhronne sprege podrazumeva se da je memorija dovoljno brza da operacije čitanja i upisa obavi u predviđenom vremenskom intervalu. Slika 4.4 prikazuje vremenski dijagram ciklusa čitanja kod sinhronne sprege. Ciklus traje ukupno tri periode sinhronizacionog signala (signala takta). Na početku periode T1 mikroprocesor stavi na adresne linije signale adrese memorijske lokacije iz koje treba pročitati podatak, a zatim, kad su adresni signali dovoljno stabilni aktivira signal čitanja R. Za sve vreme ciklusa čitanja signal W je na logičkoj 0, odnosno u neaktivnom stanju.

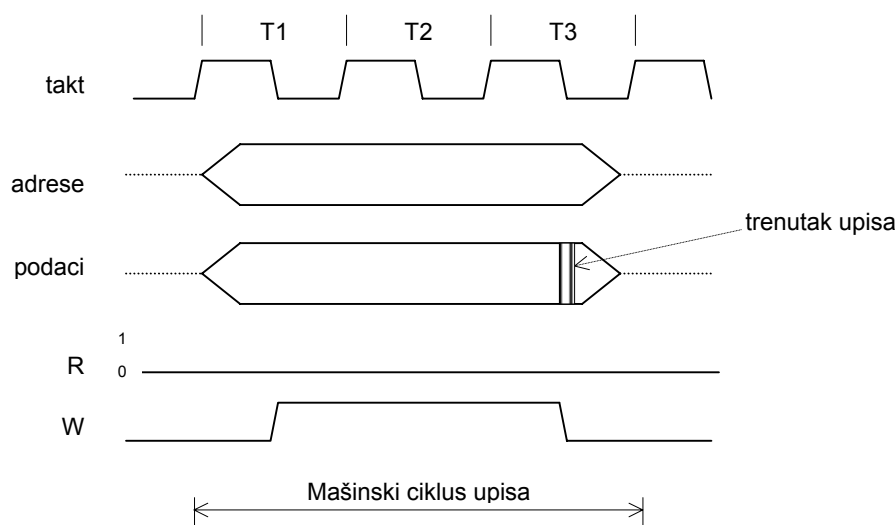


Slika 4.4: Vremenski dijagram čitanja kod sinhronne sprege

U trenutku kada primi aktivan signal čitanja R, memorija dekoduje adresne signale, pristupi adresiranoj memorijskog lokaciji i sadržaj te lokacije prosledi na magistralu podataka. Svakako da memorijskim kolima treba izvesno vreme da obavi sve ove operacije, tako da se sadržaj adresirane memorijske lokacije pojavljuje na linijama za podatke sa kašnjenjem. Mikroprocesor ostavlja memoriji tačno određeno vreme, u navedenom primeru to je do opadajuće ivice impulsa T3, kada mikroprocesor signale sa magistrale podataka učitava u interni registar. Posle toga adresni i upravljački signali (u ovom slučaju signal R) prevode se u neaktivno stanje, čime se završava ciklus čitanja. Ciklus čitanja naziva se još mašinski ciklus.

Ciklus upisa, Slika 4.5, počinje tako što mikroprocesor stavlja na adresne linije adresne signale, a na linije za podatke signale podatka koji treba upisati u adresiranu memorijsku lokaciju. Posle toga aktivira se signal W za upis, dok signal R za čitanje ostaje na logičkoj 0. Po isteku dovoljno dugog vremenskog intervala, u ovom primeru do opadajuće ivice impulsa T3 memorija treba da podatak

sa magistrale upiše u adresiranu memorijsku lokaciju. Posle tog trenutka, svi signali se vraćaju u početno stanje i sve je spremno za sledeći ciklus.



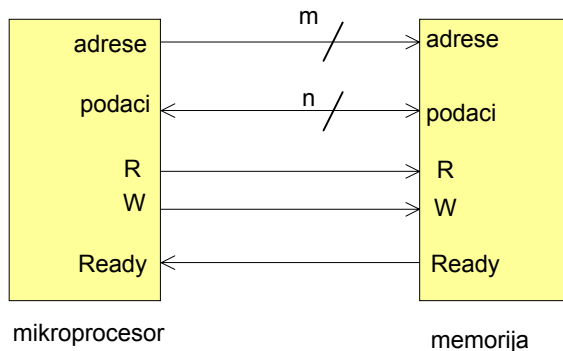
Slika 4.5: Mašinski ciklus upisa kod sinhronne sprege

Ciklusi upisa i čitanja nazivaju se mašinski ciklusi. Izvršenje instrukcije sastoji se iz nekoliko mašinskih ciklusa: na primer jedan ciklus za čitanje koda operacije, drugi ciklus za čitanje operanda, treći ciklus za interne operacije i četvrti ciklus za upis rezultata u memoriju.

Sinhrona sprega mikroprocesora i memorije, na primer sprega koju prikazuje Slika 4.3, ima sledeće karakteristike:

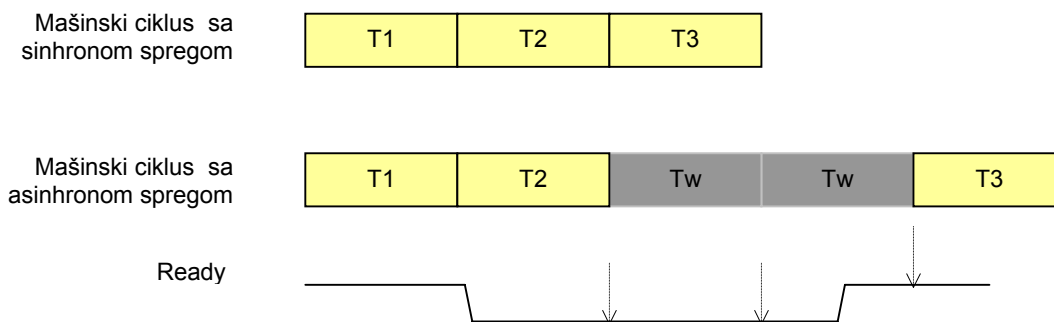
- Mikroprocesor nema povratnu informaciju da je memorija završila zahtevanu operaciju. Prema tome, mikroprocesor mora da memoriji ostavi dovoljno vremena da obavi operaciju čitanja ili upisa i to u najnepovoljnijem slučaju.
- Mikroprocesor može da radi onoliko brzo koliko dozvoljava najsporija memorijska jedinica u sistemu.
- Sinhrona sprega je jednostavna.
- Ako memorijski modul nije dovoljno brz, onda treba smanjiti frekvenciju takta mikroprocesora ili preći na korišćenje brže memorije.

Kod asinhronne sprege memorija posebnim signalom (koristićemo engleski termin *Ready*) saopštava mikroprocesoru da je završila operaciju čitanja odnosno upisa, Slika 4.6. Mikroprocesor nadgleda signal Ready i produžava mašinski ciklus sve do trenutka kada memorija završi zahtevanu operaciju.



Slika 4.6: Asinhrona sprega mikroprocesora i memorije

Slika 4.7 prikazuje mašinski ciklus sa asinhronom spregom u poređenju sa istim ciklusom kod sinhronne sprege. Radi jednostavnosti, ovde su pravougaonicima sa upisanim oznakom perioda signala takta predstavljeni svi signali koji su navedeni u prethodnim vremenskim dijagramima. Tako na primer mašinski ciklus kod sinhronne sprege, Slika 4.7 (gornji dijagram), traje tri perioda takta označena sa T1, T2 i T3.



Slika 4.7: Primer asinhronog mašinskog ciklusa sa umetanjem perioda čekanja Tw

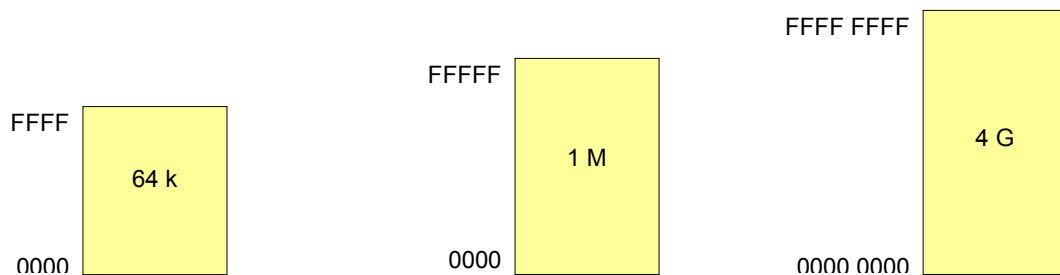
Kod asinhronne sprege, na početku mašinskog ciklusa, na primer na kraju periode takta T1, memorija prevede signal Ready na logičku jedinicu i time obaveštava mikroprocesor da je započela zahtevanu operaciju, ali da je još nije završila. Na kraju periode T2 mikroprocesor proverava signal Ready i ako je na logičkoj nuli, produžava mašinski ciklus umetanjem periode čekanja Tw. Na kraju periode Tw, mikroprocesor ponovo proverava Ready, ako je i dalje na logičkoj 0 onda umetne još jednu period čekanja Tw. Ovaj postupak se ponavlja dok memorija ne završi zahtevanu operaciju i obavesti mikroprocesor tako što vrati signal Ready na logičku 1. Kada detektuje visoki nivo signala Ready, mikroprocesor završi mašinski ciklus tako što obavi mikrooperacije predviđene periodom T3.

Karakteristike asinhronne sprege:

- Mikroprocesor dobija obaveštenje da je memorija završila predviđene operacije, pa je sprega mikroprocesora i memorije pouzdanija.
- Mikroprocesor može da radi sa memorijama različite brzine i prilagođava trajanje mašinskog ciklusa brzini svake memorijske jedinice.
- Asinhrona sprega je složenija u odnosu na sinhronu spregu.

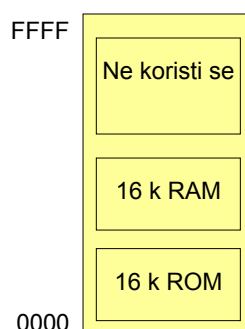
4.3 Memorijski adresni prostor

Skup svih memorijskih adresa koje mikroprocesor može da generiše naziva se memorijski adresni prostor. Ovaj prostor grafički predstavljamo pravougaonikom, slično kao i memoriju, a sa strane pravougaonika pišemo adrese (obično u heksadecimalnom sistemu). Slika 4.8 prikazuje memorijske adresne prostore mikroprocesora sa 16, 20 i 32 adresna signala. Običaj je da se memorijski adresni prostor većeg kapaciteta predstavlja pravougaonikom većih dimenzija (ali naravno dimenzije pravougaonika i kapacitet memorije nisu međusobno proporcionalni!). U grafičkom predstavljanju često se izostavlja broj signala magistrale podataka.



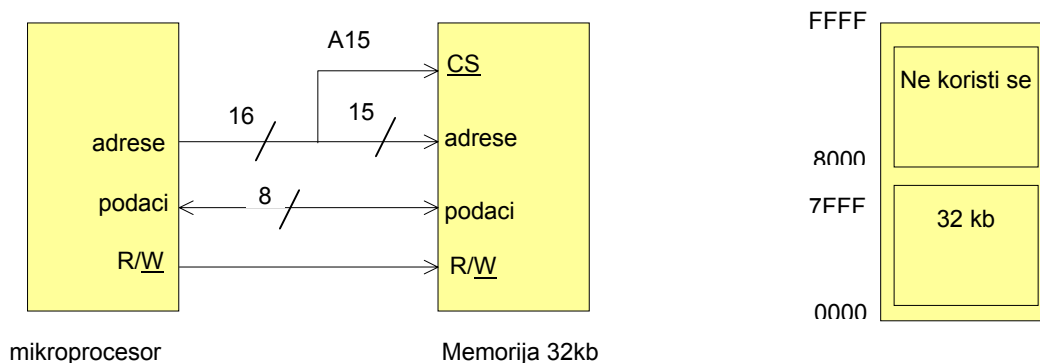
Slika 4.8: Primeri memorijskog adresnog prostora mikroprocesora sa 16, 20 i 32 adresna signala

Kod grafičkog predstavljanje memorijskog adresnog prostora korisno je označiti delove adresnog prostora koji zauzimaju fizičke memorijske jedinice i delove memorijskog prostora koji se ne koristi. Slika 4.9 prikazuje memorijski adresni prostor mikroprocesora sa 16 adresnih signala (64 k) u mikroračunarskom sistemu sa 16 k memorije tipa ROM, koji se nalazi na dnu memorijskog adresnog prostora (adrese od 0000 do 3FFF) i 16 k memorije tipa RAM, koja se nalazi u delu memorijskog prostora koji je 'iznad' ROM-a (adrese od 4000 do 7FFF). Preostali deo memorije, od adrese 8000 (heksadecimalno) do FFFF (heksadecimalno) ne koristi se.



Slika 4.9: Primer memorijskog adresnog prostora sa fizičkom memorijom i delom koji se ne koristi

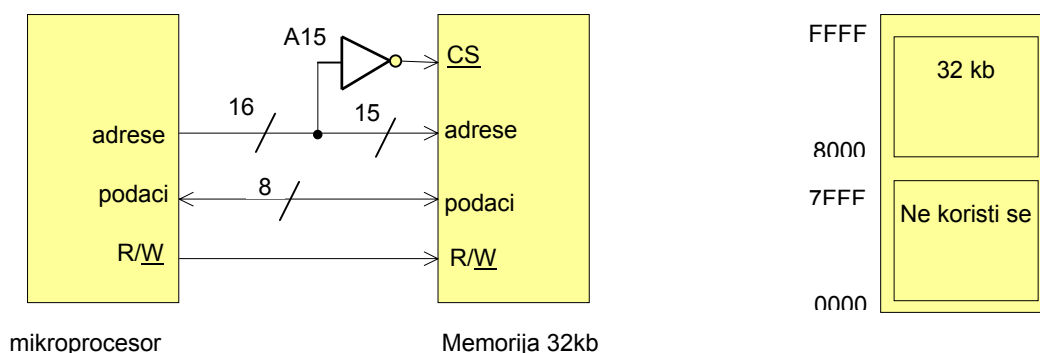
Uzmimo, na primer, mikroprocesor sa 16 adresnih signala i 8 signala za podatke koji je spregnut sa memorijom 32k x8, Slika 4.10, levo. Memorija organizacije 32k x 8, ima 15 adresnih linija i 8 linija za podatke i upravljačke signale CS i R/W. Memorija se aktivira kad je signal CS na logičkoj 0. Linije za podatke mikroprocesora direktno su vezane na linije za podatke memorije, a signal R/W (pretpostavimo da mikroprocesor ima takav upravljački signal) vezan je na signal iste namene memorije.



Slika 4.10: Primer sprege mikroprocesora i memorije (levo) i odgovarajući memorijski adresni prostor (desno)

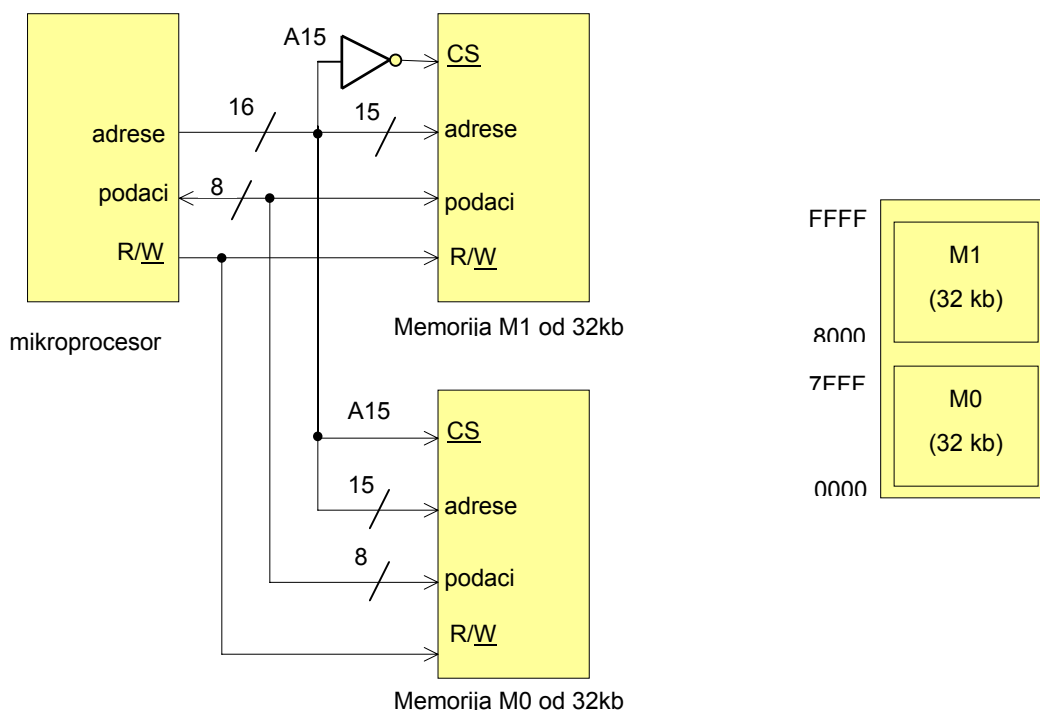
Vezivanje linija adresne magistrale nije jednoznačno, zato što mikroprocesor ima 16, a memorija 15 adresnih linija. U ovakvim slučajevima manje značajne linije adresne magistrale mikroprocesora vezuje se na odgovarajuće linije memorije, pri čemu više značajne adresne linije mikroprocesora nemaju linije na memoriji na koje mogu da se vežu. U datom primeru, 15 linija mikroprocesora, od A0 do A14 vežu se na linije memorije od A0 do A14. Linija A15 mikroprocesora ostaje slobodna i može se koristiti na tri načina. Slika 4.10 prikazuje slučaj kad je linija A15 mikroprocesora dovedena na signal CS memorije. U ovom slučaju, kad je A15=0 memorija aktivirana, a kada je A15=1, memorija nije aktivirana. Očigledno je u ovom slučaju memorijski adresni prostor kao na Slici 4.10 desno: donjih 32 kb adresnog prostora zauzima fizička memorija, a ne koristi se gornjih 32kb adresnog prostora.

Ako se umesto signala A15 na ulaz CS memorije dovede invertovani signal A15, onda će 32kb fizičke memorije biti u gornjoj polovini adresnog prostora, a donja polovina se neće koristiti, Slika 4.11.



Slika 4.11: Modifikovana sprega mikroprocesora i memorije (levo) i odgovarajući memorijski adresni prostor (desno)

Kombinacijom prethodnih slučajeva mogu se vezati dva memorijska modula od po 32 kb tako da jedan bude u donjoj, a drugi u gornjoj polovini adresnog prostora, Slika 4.12.



Slika 4.12: Sprega mikroprocesora sa dva memorijska modula, M0 i M1

Naravno, postoji varijanta da se mikroprocesor spregne sa memorijskim modulom od 32kb, tako da se signal A15 uopšte ne koristi. U tom slučaju, memorijski modul bi se javljao kod svakog pristupa memoriji, odnosno jedan isti modul bi se pojavljiva u donjoj i gornjoj polovini adresnog prostora.

4.4 Adresni dekoderi

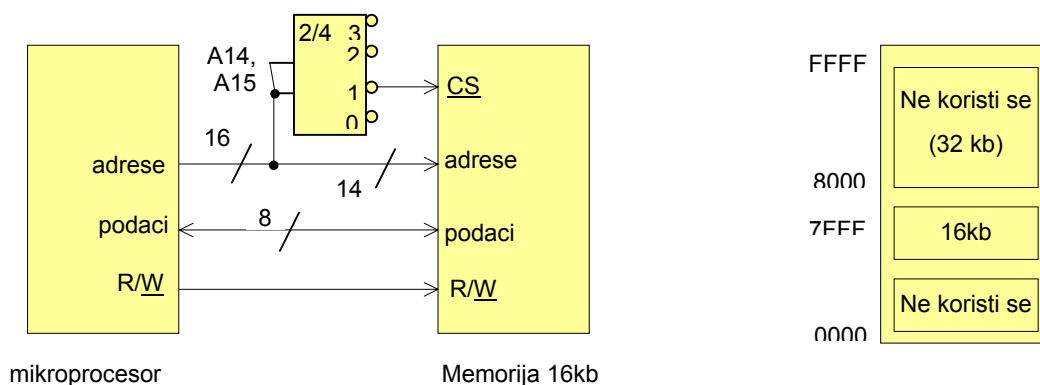
Neka mikroprocesor ima m adresnih linija i neka se spreže sa memorijskim modulom (ili modulima) koji imaju k adresnih linija. Iz prethodnih primera vidi se da se adresne linije mikroprocesora dele na dve grupe:

- Prva grupa od k adresnih linija neposredno se spaja na k ulaznih adresnih linijija memorijskih modula tako što se signali A_0, A_1, \dots, A_{k-1} mikroprocesora spajaju na signale A_0, A_1, \dots, A_{k-1} memorije.
- Preostalih $(m-k)$ adresnih linija koriste se za adresne dekodere.

Adresni dekoderi su kombinacione logičke mreže koje na osnovu ulaznih adresnih signala generišu CS signale za memorijske module. Ako je p broj ulaznih signala adresnog dekodera, onda razlikujemo sledeća tri slučaja:

- Ako je $p = m-k$, onda se radi potpuno adresno dekodiranje.
- Ako je $0 < p < m-k$, onda je dekodiranje adresa delimično.
- Ako je $p = 0$, nema dekodiranja adresa.

Uzmimo na primer mikroprocesor sa 16 adresnih signala i 8 signala za podatke i memorijski modul od 16 kb. Pošto memorijski modul ima 14 adresnih signala, onda preostaju dva signala, A14 i A15, koji se dovode na ulaz adresnog dekodera, Slika 4.13.



Slika 4.13: Primena adresnog dekodera sa 2 ulaza i 4 izlaza

Neka su izlazi dekodera $\overline{CS_3}$, $\overline{CS_2}$, $\overline{CS_1}$ i $\overline{CS_0}$ i neka je funkcija dekodera definisana sledećom tablicom.

A15	A14	$\overline{CS_3}$	$\overline{CS_2}$	$\overline{CS_1}$	$\overline{CS_0}$
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

Mesto memorijskog modula u adresnom prostoru zavisi od izlaza dekodera koji se veže na ulaz CS za selekciju memorijskog modula. U primeru sa slike 4.13 izlaz CS1 koristi se za aktiviranje memorije, pa zato memorijski modul smešten u adresnom prostoru koji prikazuje Slika 4.13 desno. U sledećoj tabeli date su memorijske adrese memorijskog modula sa Slike 4.13 u zavisnosti od izlaza adresnog dekodera koji se koristi kao signal selekcije CS. Naravno, u sistem mogu da se stave dva, tri ili četiri memorijska modula sa proizvoljnim korišćenjem izlaza adresnog dekodera za selekciju memorijskih modula.

Signal adresnog dekodera	Opseg adresa memorijsko modula
<u>CS3</u>	C000 do FFFF
<u>CS2</u>	8000 do BFFF
<u>CS1</u>	4000 do 7FFF
<u>CS0</u>	0000 do 3FFF

4.5 Projektovanje memorijske jedinice

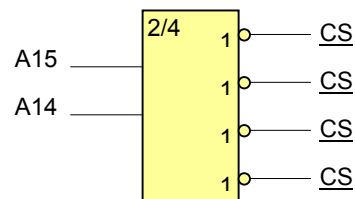
Klasičan problem kod projektovanja jedinice operativne memorije nastaje kada projektant nema poluprovodničke memorijske module koji odgovaraju specifikaciji memorijske jedinice. Razlikujemo sledeća tri slučaja:

- Memorijski moduli imaju dovoljan broj bita u memorijskim lokacijama, ali nemaju dovoljan broj memorijskih lokacija. Na primer, zahteva se memorijska jedinica od 128k x 8, a na raspolaganju su memorijski moduli sa organizacijom 32k x 8.
- Memorijski moduli imaju dovoljan broj lokacija, ali nemaju dovoljan broj bita u memorijskim lokacijama. Na primer, zahteva se memorijska jedinica od 32k x 16, a na raspolaganju su memorijski moduli 32k x 8.
- Kombinacija prethodna dva slučaja, odnosno raspoloživi memorijski moduli nemaju ni dovoljan broj lokacija ni dovoljan broj bita u lokacijama.

Projektovanje memorijske jedinice svodi se na projektovanje adresnog dekodera i povezivanje zadatih memorijskih modula u memorijsku jedinicu sa zadatom specifikacijom. Kao primer uzmimo da treba projektovati memorijsku jedinicu organizacije 128k x 8 na dnu adresnog prostora mikroprocesora sa 20 adresnih signala, ako su na raspolaganju memorijski moduli organizacije 32k x 8.

Očigledno je da nam za rešenje ovog problema trebaju 4 memorijska modula, zato što će 4 x (32k x 8) dati memorijsku jedinicu 128k x 8. Iz organizacije memorijskih modula koji su na raspolaganju, vide da oni imaju 15 adresnih linija i 8 linija za podatke. Prema tome, od ukupno 20 adresnih signala mikroprocesora, donjih 15 signala neposredno se vezuju na odgovarajućih 15 adresnih signala memorijskih modula.

Preostalih 5 adresnih signala koristi se za adresni dekodera. Treba primetiti da pošto se koriste 4 memorijska modula, dovoljna su 2 adresna signala, A16 i A15, za generisanje signala za selekciju (CS3, CS2, CS1 i CS0). Preostala tri signala A19, A18 i A17 koriste se za generisanje signala dozvole rada adresnog dekodera. Prema tome, može se koristiti adresni dekodera sa Slike 4.13, ali sa signalom dozvole koji se aktivira kada je A19 = A18 = A17 = 0.



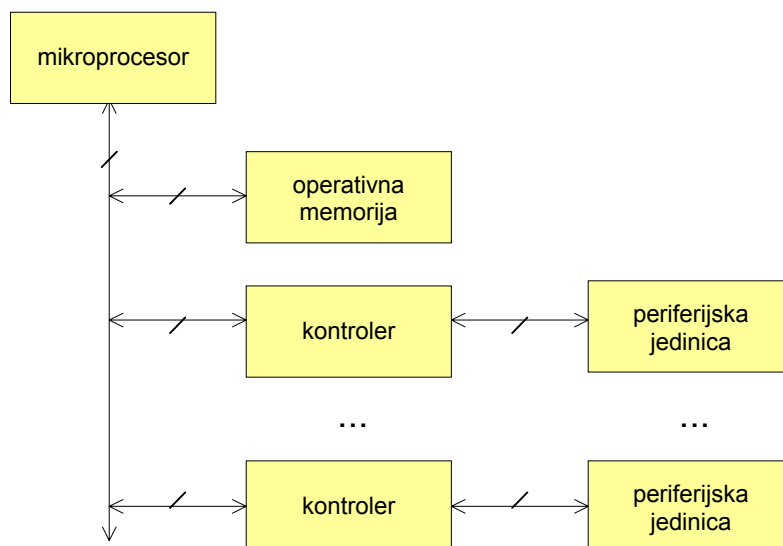
Slika 4.14: Primena adresnog dekodera sa 2 ulaza i 4 izlaza

5. Ulazno-izlazni podsistem

Osnovne funkcionalne jedinice mikroračunarskog sistema su centralni procesor, memorija i ulazno-izlazne jedinice. U ovom poglavlju ukratko su izloženi osnovni principi sprežanja ulazno-izlaznih jedinica sa centralnim procesorom i organizacije ulazno/izlaznog podsistema

5.1 Organizacija ulazno-izlaznog podsistema

Slika 5.1 predstavlja jednostavnu organizaciju mikroračunarskog sistema sa centralnim procesorom, memorijom i ulazno-izlaznim podsistemom. Ulazno-izlazni podsistem sastoji se od perifernih jedinica i kontrolera. Perifernije jedinice su obično elektromehanički uređaji koji omogućavaju ulaz i izlaz podataka. Primeri izlaznih perifernih jedinica su monitor sa ekranom za prikazivanje teksturalnih i grafičkih informacija i štampač. Ulazne jedinice, na primer tastatura i miš, služe za unošenje podataka u računarski sistem.

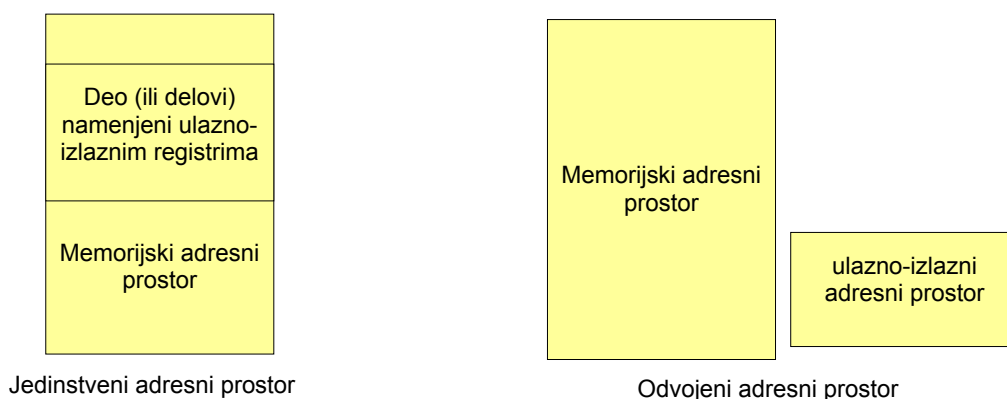


Slika 5.1: Jednostavna organizacija mikroračunarskog sistema

Kontroleri (ili adapteri) su elektronske komponente koje služe za prilagođenje ili interfejs između perifernih jedinica i mikroprocesora. Treba imati u vidu da su perifernije jedinice specifične i nisu prilagođene načinu i brzini rada mikroprocesora. Kontroleri obično sadrže sve neophodne elemente i logiku koja je neophodna za efikasno upravljanje perifernim jedinicama i prilagođenje između perifernih jedinica i mikroprocesora u pogledu brzine rada, naponskih nivoa i sinhronizacije. Prema tome, mikroprocesor nema neposredan pristup perifernim jedinicama, već ‘vidi’ samo kontrolere i komunicira sa kontrolerima.

Kontroleri su programabilne jedinice koje poseduju svoje upravljačke jedinice sposobne da obavljaju zadatke vezane za upravljanje i komunikaciju sa perifernim jedinicama. Kontroleri poseduju programabilne registre preko kojih se razmenjuju upravljačke informacije i podaci sa mikroprocesorom. Naravno, programabilni registri imaju svoje adrese i mikroprocesor mora da generiše tačnu adresu programabilnog registra kome pristupa. U jednostavnim primerima ulazno-izlaznih podsistema, programabilni registri mogu da se neposredno koriste za ulaz i izlaz podataka. Stoga ćemo sve registre koji se nalaze u ulazno-izlaznom podsistemu nazivati ulazno-izlazni registri.

Ulazno-izlazni registri mogu da se nalaze u jedinstvenom adresnom prostoru zajedno sa memorijom ili da budu odvojenom, ulazno-izlaznom adresnom prostoru, Slika 5.2. Prednost jedinstvenog adresnog prostora je u tome što sve instrukcije koje se koriste za pristup memoriji mogu da se koriste i za pristup ulazno-izlaznim registrima i što se memorijski i ulazno-izlazni registri tretiraju na isti način. Prednost odvojenog adresnog prostora je što omogućava jednostavniju organizaciju sistema u kome se jedan adresni prostor koristi isključivo za memorijske, a drugi za ulazno-izlazne komponente.



Slika 5.2: Zajednički (levo) i odvojeni (desno) adresni prostor

Mikroprocesor koristi iste linije za prenos adresa i podataka kod memorijskog i ulazno-izlaznog adresnog prostora i zato mora da ima posebne signale kojima označava vrstu pristupa. U slučaju da signal M označava pristup memorijskom adresnom prostoru a signal IO pristup ulazno-izlaznom adresnom prostoru, sledeća tabela opisuje operacije mikroprocesora u zavisnosti od logičkih nivoa ovih signala. Signal R označava ciklus čitanja, a W ciklus upisa.

M	IO	R	W	Operacija
1	0	1	0	Čitanje iz memorije
1	0	0	1	Upis u memoriju
0	1	1	0	Čitanje iz ulazno-izlaznog registra
0	1	0	1	Upis u ulazno-izlazni registar
1	1	x	x	Nije dozvoljeno
x	x	1	1	Nije dozvoljeno

5.2 Primer jednostavnog izlaza

Primer sa izlaznim registrom na čije izlaze su vezane svetleće diode (LED).

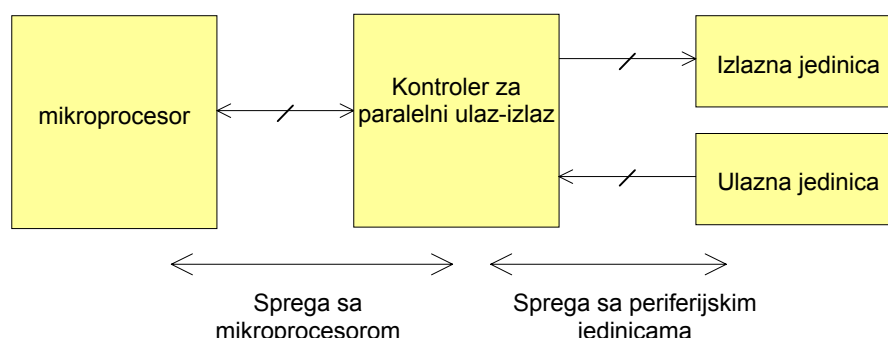
5.3 Primer jednostavnog ulaza

Primer sa ulaznim registrom preko koga se čitaju stanja osam prekidača.

5.4 Kontroler za paralelni ulaz-izlaz

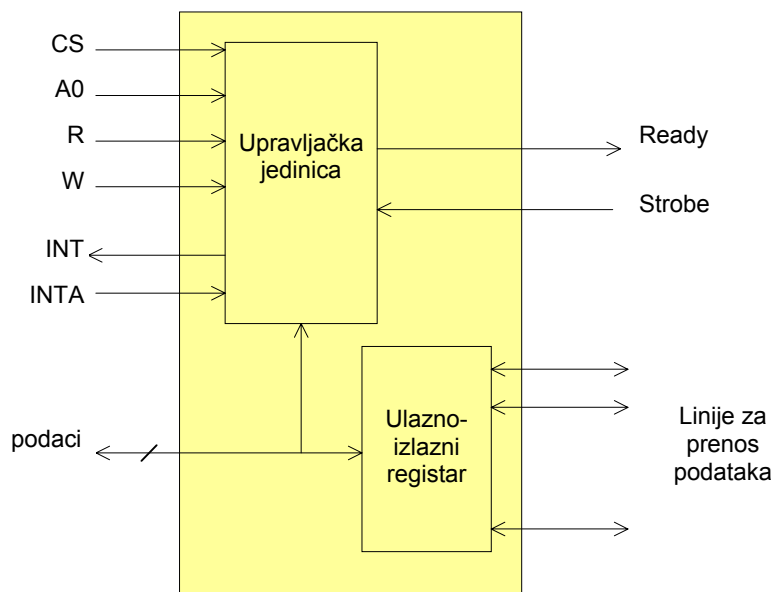
Jednostavni primer ulazno-izlaznog kontrolera je kontroler za paralelni ulaz i izlaz podataka. Ovaj kontroler sadrži registre za podatke koji se mogu konfigurisati tako da služe za paralelni ulaz ili izlaz podataka. Pored toga, ovi kontroleri poseduju programabilne registre koji se koriste za konfigurisanje i izbor načina rada kontrolera.

Kontroler poseduje sa jedne strane signale za spregu sa mikroprocesorom, a sa druge strane signale za spregu sa ulazno-izlaznim jedinicama, Slika 5.3.



Slika 5.3: Sprega kontrolera za paralelni ulaz-izlaz sa mikroprocesorom i ulazno-izlaznim jedinicama

Slika 5.4 prikazuje strukturu jednostavnog kontrolera za paralelni ulaz-izlaz, gde su sa leve strane nacrtani signali za spregu sa mikroprocesorom, a sa desne signali za spregu sa ulazno-izlaznim jedinicama. U mikroprocesorskom sistemu mora da postoji dekodler ulazno-izlaznih adresa koji generiše signal CS za aktiviranje kontrolera. U slučaju da poseduje više ulazno-izlaznih registara kontroler mora da ima dodatne adresne signale. U primeru sa slike postoji samo jedan adresni signal A0. Signali R i W određuju operacije čitanja i upisa koje se obavljaju nad registrima kontrolera.



Slika 5.4: Struktura jednostavnog kontrolera za paralelni ulaz-izlaz

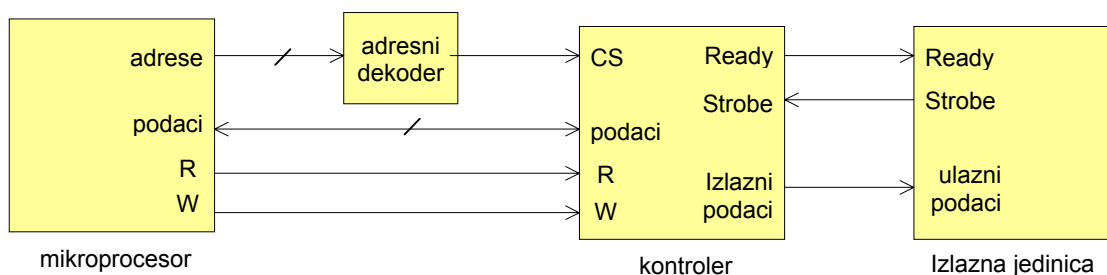
Kontroler ima linije za pristup magistrali podataka mikroprocesorskog sistema preko kojih mikroprocesor komunicira sa kontrolerom. Konačno, kontroler obično ima signal INT kojim zahteva prekid od mikroprocesora i signal INTA kojim mikroprocesor obaveštava kontroler da je prekid prihvaćen.

Na strani perifernjske jedinice kontroler ima linije za prenos podataka i dve linije za prenos signala za sinhronizaciju, koji se obično nazivaju *Ready* i *Strobe*. Uloge ova dva signala različite su kod ulaznih i izlaznih operacija. Na primer, kod izlaza paralelnih podataka prema perifernjskoj jedinici značenja mogu biti:

Ready = 0, na izlaznim linijama prema perifernjskoj jedinici je 'stari' podatak,
 = 1, na izlaznim linijama je novi podatak.

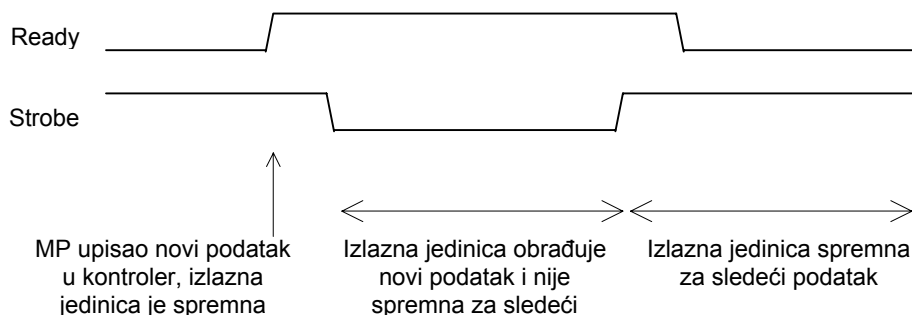
Strobe = 0, perifernjska jedinica nije spremna da prihvati novi podatak,
 = 1, perifernjska jedinica je spremna da prihvati novi podatak.

Slika 5.5 prikazuje konfiguracija sistema sa mikroprocesorom, kontrolerom za paralelni ulaz-izlaz (konfigurisan za izlaz podataka) i perifernjskom izlaznom jedinicom. Radi jednostavnosti nisu prikazane linije za prenos signala A0 i signala za prekide INT i INTA.



Slika 5.5: Sistem sa mikroprocesorom, kontrolerom za paralelni ulaz-izlaz i perifernjskom jedinicom.

Postupak izlaza podataka prema perifernjskoj jedinici započinje tako što mikroprocesor upiše u kontroler podatak koji treba preneti perifernjskoj jedinici. Posle toga mikroprocesor može da radi neke druge operacije i prepušta kontroleru da prenese podatak do perifernjske jedinice. Osnovni problem u prenosu podataka je što perifernjska jedinica može da bude spora, odnosno da zahteva neko vreme da bi obradila primljeni podatak. Na primer, ako je izlazna jedinica DA konvertor, onda je potrebno neko vreme za konverziju iz digitalnog u analogni oblik. Ako je izlazna jedinica štampač, onda je potrebno neko vreme za štampanje primljenog znaka. Prema tome, očigledno je da kontroler mora da prilagodi brzinu prenosa izlaznih podataka mogućnostima izlazne jedinice.



Slika 5.6: Postupak sinhronizacije (rukovanje) između kontrolera i izlazne jedinice

Postupak sinhronizacije između kontrolera i izlazne jedinice ilustruje vremenski dijagram, Slika 5.6. U početku je izlazna jedinica spremna da prihvati novi podatak i stavlja signal Strobe na

logičku 1. Kada mikroprocesor upiše novi izlazni podatak, kontroler prenosi taj podatak na linije za prenos podataka do izlazne jedinice i stavlja signal Ready na logičku 1, čime obaveštava izlaznu jedinicu da je novi podatak spreman. Izlazna jedinica prevodi signal Strobe na logičku 0 i ostavlja ga na toj vrednost sve dok obrađuje novi podatak.

U trenutku kad je završila obradu novog signala (na primer kad je štampač odštampao primljeni znak) izlazna jedinica prevodi signal Strobe na logičku 1 i tako obaveštava kontroler da je spremna za naredni podatak. Kontroler prevodi signal Ready na logičku 0 i tako stavlja do znanja izlaznoj jedinici da je na izlaznim linijama podatak koji je već obrađen.

Dalji postupak zavisi od načina prenosa izlaznih podataka. Na primer, tipično bi bilo da kontroler generiše signal prekida i tako obavesti mikroprocesor da je izlaz prethodnog podatka završen. U programu za obradu prekida mikroprocesor može da odredi sledeći podatak i da ga upiše u kontroler čime se postupak izlaza podataka ponavlja. Ovakav način sinhronizacije između kontroler i izlazne jedinice naziva se 'rukovanje' (engleski: *handshaking*).

5.5 Primer: Kontroler 8255

Primer kontrolera 8255 za paralelni ulaz-izlaz. Tipična primena kontrolera 8255 i programska podrška.

5.6 Tajmeri

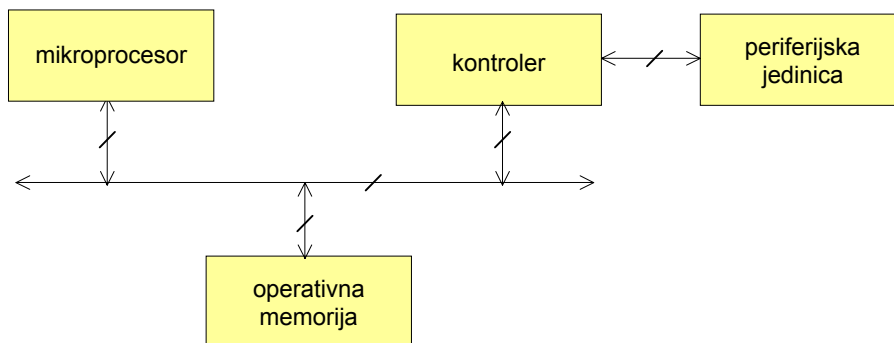
Primer tajmera 8254 i primena u računaru klase PC.

5.7 Primer: evidencija realnog vremena

Primena mikrokontrolera 8051 u evidenciji realnog vremena.

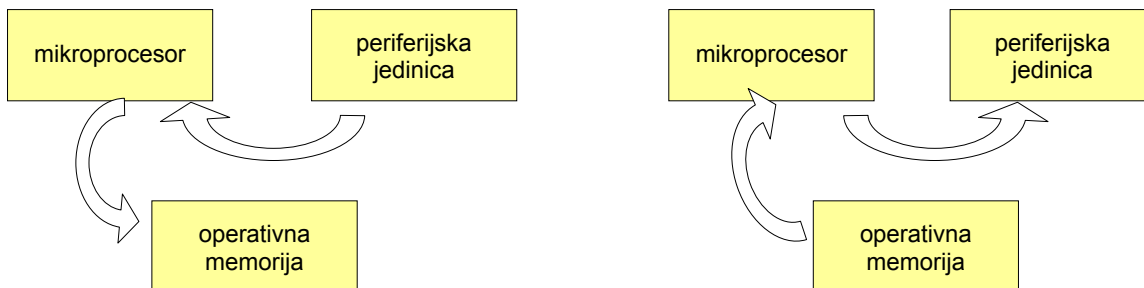
5.8 Direktan pristup memoriji

Posmatrajmo jednostavan mikroračunarski sistem sa mikroprocesorom, operativnom memorijom i jednom perifernom jedinicom, Slika 5.7. Radi jednostavnijeg objašnjenja direktnog pristupa memoriji, posmatraćemo kontroler i perifernu jedinicu zajedno, kao jedan blok.



Slika 5.7: Blok dijagram jednostavnog mikroračunarskog sistema

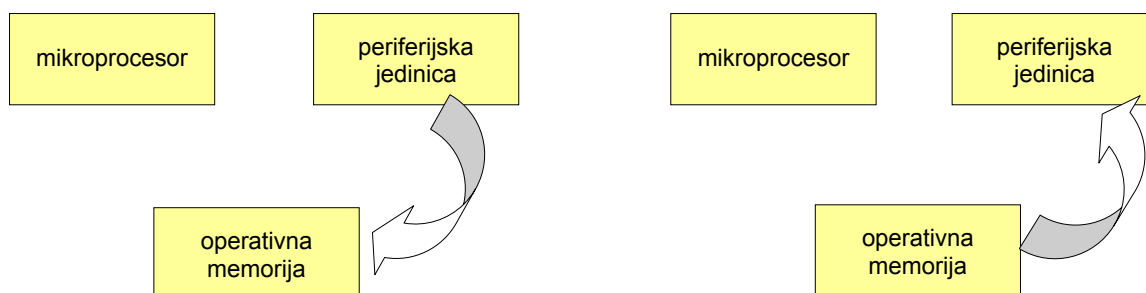
Kod prenosa podataka sa perifernog jedinice u operativnu memoriju, svaki pojedinačni podatak mora prvo da se prenese u registar mikroprocesora, a zatim iz registra mikroprocesora u operativnu memoriju, Slika 5.8 (levo). Naravno, kod prenosa u suprotnom smeru, podatak mora prvo iz memorije da se prenese u registar mikroprocesora, a zatim iz registra mikroprocesora u perifernu jedinicu, Slika 5.8 (desno).



Slika 5.8: Prenos podataka sa perifernog jedinice u operativnu memoriju (levo) i u suprotnom smeru (desno)

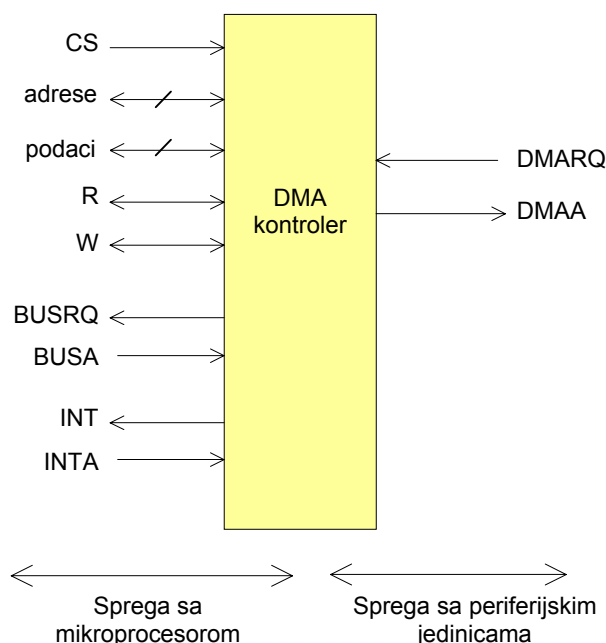
Očigledno je ovakav način prenosa spor i neefikasan. Spor je zato što mikroprocesor mora da izvrši niz instrukcija koje priličuju perifernoj jedinici, izračunavaju adresu memorije, pristupaju memoriji i tako dalje. Prenos nije efikasan zato što mikroprocesor u toku prenosa ne može da obavlja bilo kakve druge operacije.

Ideja direktnog pristupa memoriji (*Direct Memory Access, DMA*) sastoji se u tome da se podaci neposredno prenose između perifernog jedinice i operativne memorije, bez ikakvog učešća mikroprocesora, Slika 5.9. Naravno, mikroprocesorski sistem mora da ima poseban DMA kontroler koji organizuje i upravlja ovakvim načinom prenosa podataka.



Slika 5.9: Ideja direktnog pristupa memoriji kod prenosa podataka iz periferijske jedinice u operativnu memoriju (levo) i u suprotnom smeru (desno)

Slika 5.10 prikazuje spoljne signale DMA kontrolera. Signal CS koristi se za aktiviranje DMA kontrolera kod pristupa mikroprocesora programabilnim registrima kontrolera. Signali adresne magistrale, magistrale podataka i upravljački signali R i W su dvosmerni i mogu se prevesti u stanje visoke impendanse. Signali INT i INTA koriste se za mehanizam prekida.



Slika 5.10: Osnovni spoljni signali DMA kontrolera

Da bi organizovao direktan pristup memoriji DMA kontroler mora prvo da preuzme upravljanje magistralama i generiše adresne signale i signale R i W. Pošto u svakom trenutku samo jedna jedinica može upravljati magistralama, jedinica koja hoće da preuzme upravljanje magistralama mora od mikroprocesora da traži dozvolu. Signali *BUSRQ* (*Bus Request*) i *BUSA* (*Bus Acknowledge*) služe za sinhronizaciju pristupa magistralama između DMA kontrolera i mikroprocesora. Logičkom 1 na liniji *BUSRQ* DMA kontroler traži od mikroprocesora upravljanje magistralama, a logičkom 1 na liniji *BUSA* mikroprocesor obaveštava DMA kontroler da dozvoljava upravljanje magistralama.

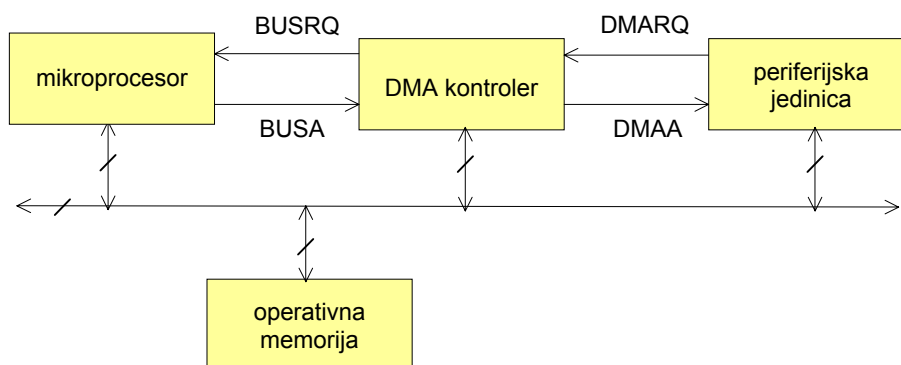
Dok mikroprocesor upravlja magistralama, DMA kontroler drži svoje linije za adresnu magistralu i linije R i W u stanju visoke impendanse. Sa druge strane, kada prepusti DMA kontroleru upravljanje magistralama, mikroprocesor adresnu magistralu i signale R i W prevodi u stanje visoke impendanse.

Signalom DMARQ (*DMA Request*) periferna jedinica obaveštava DMA kontroler da je spremna za DMA prenos, a signalom DMAA (*DMA Acknowledge*) DMA kontroler obaveštava perifernu jedinicu da je započeo ciklus DMA prenosa.

U postupku primene DMA razlikuju se tri perioda vremena: (i) pre DMA prenosa, (ii) u toku DMA prenosa i (iii) posle DMA prenosa. Pre DMA prenosa mikroprocesor programira perifernu jedinicu i DMA kontroler. Na primer, u slučaju diska, mikroprocesor u kontroler diska upisuje sektore kojima treba pristupiti, smer prenosa podataka i količinu podataka koje treba preneti. U DMA kontroler mikroprocesor upisuje smer prenosa podataka, količinu podataka koje treba preneti i početnu memorijsku adresu zone u koju treba upisati podatke sa perifernu jedinicu ili iz koje treba čitati podatke koji se prenose perifernu jedinici.

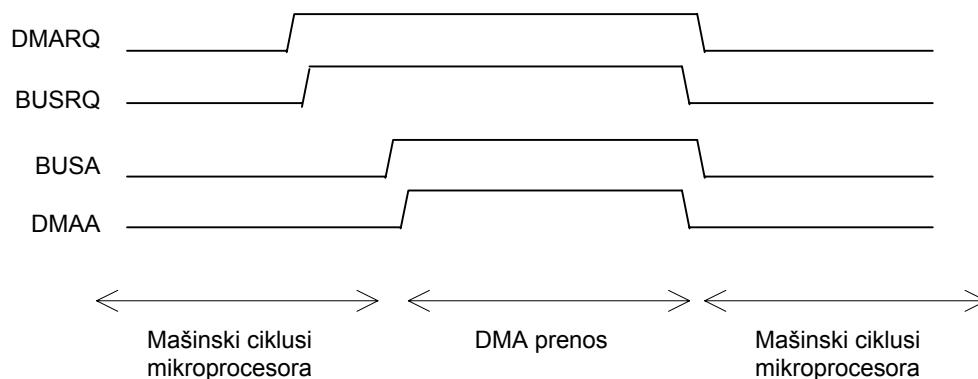
Jednom kada završi programiranje perifernu jedinicu i DMA kontrolera, mikroprocesor ne učestvuje u DMA prenosu i može da obavlja druge operacije. DMA kontroler organizuje DMA prenos tako što za svaki podatak koji treba preneti između perifernu jedinicu i memorije, traži od mikroprocesora pristup magistralama i kada dobije dozvolu pristupa, generiše sve signale koji su neophodni za pristup memoriji.

Postupak prenosa jednog podatka, na primer iz perifernu jedinicu u memoriju, počinje tako što periferna jedinica signalom DMARQ javlja DMA kontroleru da je spremna za prenos, Slika 5.11. DMA kontroler zatim aktivira signal BUSRQ kojim od mikroprocesora traži upravljanje magistralama. Kada mikroprocesor završi tekući mašinski ciklus, prevodi svoje magistrale u stanje visoke impedanse i signalom BUSA obaveštava DMA kontroler da ima dozvolu pristupa magistralama. Sada DMA kontroler signalom DMAA obaveštava perifernu jedinicu da započinje ciklus DMA prenosa i periferna jedinica prenosi podatak na magistralu podataka. Istovremeno, DMA kontroler generiše adresne signale i signal W. Podatak sa magistrale podataka memorija smešta u adresiranu memorijsku lokaciju i time se završava DMA prenos jednog podatka.



Slika 5.11: Konfiguracija mikropcesorskog sistema za DMA prenos

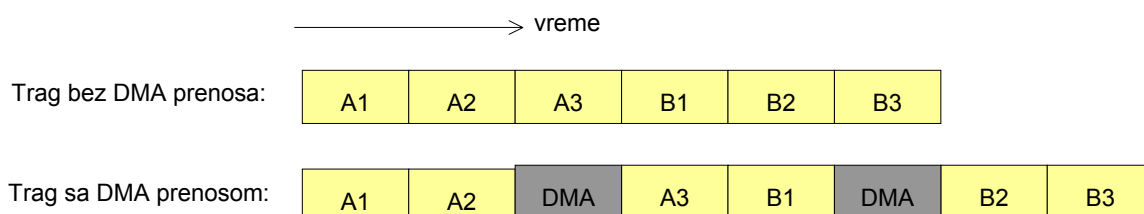
Slika 5.12 prikazuje vremenske dijagrame signala koji učestvuju u sinronizaciji kod DMA prenosa. Po završetku ciklusa DMA prenosa, DMA kontroler svoje spoljne signale prevodi u stanje visoke impendanse a signale BUSRQ i DMAA u neaktivno stanje (logička 9). Periferna jedinica svoj signal DMARQ prevodi u logičku 0. Mikroprocesor signal BUSA prevodi u neaktivno stanje (logička 9) i nastavlja da izvršava svoje naredne mašinske cikluse. U ovom trenutku sve jedinice su spremne da ponove ciklus DMA prenosa.



Slika 5.12: Vremenski dijagrami sinhronizacije kod DMA prenosa

Kod svakog DMA prenosa, DMA kontroler smanjuje za jedan sadržaj registra u kome se nalazi broj podataka koje treba preneti i inkrementira (ili dekrementira) adresu memorijske lokacije kojoj se pristupa u toku DMA prenosa. Kada se prenesu svi podaci DMA kontroler obično generiše prekid kojim obaveštava mikroprocesor da je DMA prenos je završen. Mikroprocesor može da preduzme dalje akcije, na primer da obradi prenesene podatke ili da pripremi novi DMA prenos.

Efekat DMA prenosa može se posmatrati preko traga mikroprocesorskog sistema, Slika 5.13. Pravougaonici predstavljaju mašinske cikluse, a radi jednostavnosti uzeto je da svi ciklusi imaju isto trajanja. Mašinski ciklusi mikroprocesora su na primer, čitanje iz memorije, upis u memoriju, interne operacije i pristup ulazno-izlaznim registrima. Gornji trag predstavlja izvršenje dve instrukcije, A i B, sa po tri mašinska ciklusa, označena sa A1, A2 i A3, odnosno B1, B2 i B3.



Slika 5.13: Trag mikroprocesorskog sistema bez DMA prenosa (gore) i sa DMA prenosom (dole)

Donji trag predstavlja izvršenje instrukcija A i B u slučaju da su se dogodila dva mašinska ciklusa sa DMA prenosom, jedan između ciklusa A2 i A3 i drugi između B1 i B2. U toku mašinskog ciklusa A2 mikroprocesor je primio zahteva za DMA prenos, sačekao da završi tekući ciklus i predao DMA kontroleru upravljanje magistralama. DMA kontroler je obavio DMA prenos u toku jednog mašinskog ciklusa i vratio mikroprocesoru upravljanje magistralama. Mikroprocesor nastavlja izvršenje instrukcije A tako što izvrši naredni mašinski ciklus, A3, i time završava instrukciju A. Na sličan način obavlja se i drugi DMA prenos.

Iz traga se vidi da je kod DMA prenosa trag isti kao i bez DMA prenosa, samo što s vremena na vreme DMA kontroler obavlja DMA prenos. Ovakav način korišćenja mašinskih ciklusa naziva se 'krađa ciklusa' (*cycle stealing*).

5.9 Organizacija ulaza-izlaza

U odnosu na sinhronizaciju sa perifernom jedinicom, u praksi se obično koriste tri načina organizovanja ulaza-izlaza:

- Programski ulaz-izlaz,
- Primena prekida i
- DMA prenos.

Kod programskog ulaza-izlaza mikroprocesor u petlji proverava da li je periferna jedinica spremna za prenos podataka. Tek kada je periferna jedinica spremna, petlja se završava i mikroprocesor obavlja ulazno-izlazne operacije. Očigledno je da ovakav način sinhronizacije nije pogodan jer nepotrebno zauzima vreme mikroprocesora. Sve dok je u petlji, mikroprocesor ne može da obavlja nikakve druge operacije.

Primena prekida otklanja ove nedostatke tako što mikroprocesor nemora da čeka na perifernu jedinicu već može da izvršava neke druge programe. U trenutku kada je spremna za prenos, periferna jedinica generiše prekid. Mikroprocesor prekida izvršavanje tekućeg programa i izvrši program za obradu prekida u kome se obavljaju ulazno-izlazne operacije. Mikroprocesor zatim nastavlja sa izvršenjem prekinutog programa, a periferna jedinica će ponovo generisati prekid kada bude spremna za naredni prenos.

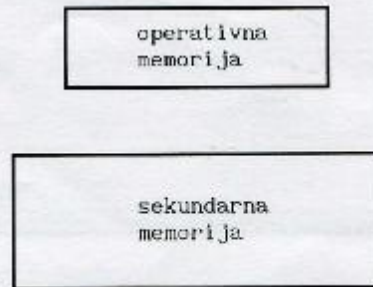
Prekidna organizacija ulazno-izlaznih operacija nije pogodna za prenos velikog broja podataka, na primer kod rada sa diskom ili komunikacionom karticom. U ovim primerima, prelaz na program za obradu prekida i izvršavanje programa za obradu prekida traju suviše dugo pa se ne može postići visoka brzina prenosa.

Najbrži prenos podataka postiže se DMA prenosom koji je pogodan za ulaz ili izlaz velike količine podataka. Naravno, ovaj način prenosa zahteva dodatnu hardversku jedinicu (DMA kontroler).

MEMORIJSKA HIJERARHIJA

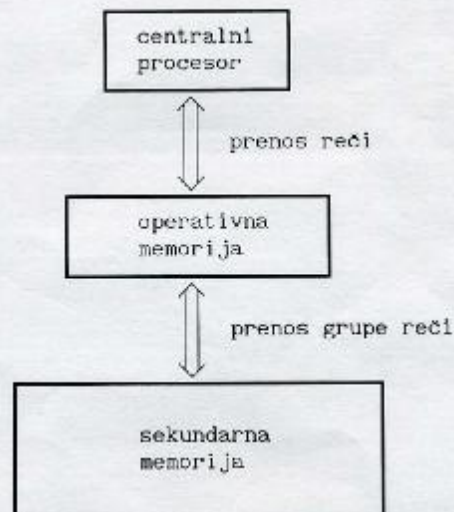
OPERATIVNA I SEKUNDARNA MEMORIJA

Memorijski podsistem deli se na operativnu (ili primarnu) i masovnu (ili sekundarnu) memoriju, sl. 1.



Sl. 1: Operativna i sekundarna memorija (relativni odnos kapaciteta kvalitativno odgovara veličini pravougaonika)

Centralni procesor može da neposredno pristupa samo lokacijama **operativne memorije**, sl. 2. Zato se u operativnu memoriju smeštaju programi i podaci koje centralni procesor izvršava, odnosno obrađuje. Ova memorija se obično realizuje u poluprovodničkoj tehnologiji.



Sl. 2: Principijelna šema računara sa operativnom i sekundarnom memorijom

Imajući u vidu da poluprovodnička memorija ima visoku cenu po bitu u upoređenju sa drugim komponentama, programi i podaci koje centralni

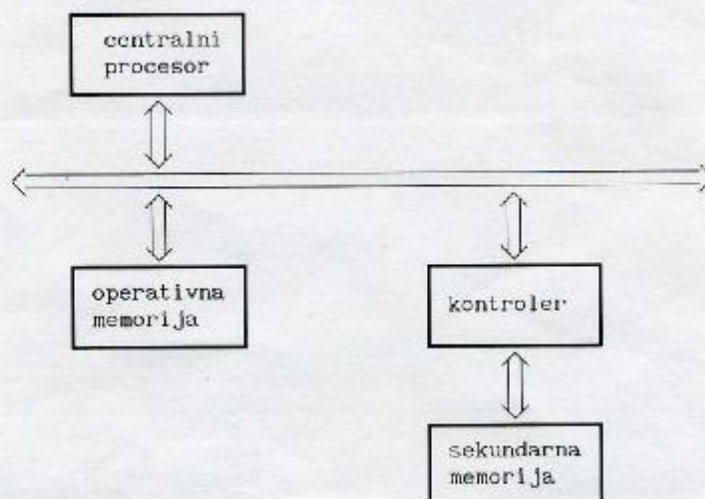
procesor u datom trenutku ne koristi smeštaju se u sekundarnu memoriju koja ima nižu cenu po bitu. Na taj način memorijskom podsistemu operativni deo obezbeđuje brzinu koja odgovara centralnom procesoru, a sekundarni deo obezbeđuje veliki kapacitet. Sekundarna memorija realizuje se primenom magnetnih diskova i traka i sličnim komponentama kojima centralni procesor ne može neposredno da pristupi.

Svi programi i podaci obično su smešteni u sekundarnoj memoriji. Program koji treba da se izvrši prenosi se iz sekundarne memorije u primarnu memoriju. Centralni procesor pristupa operativnoj memoriji i izvršava program. Sledeće program koji treba da se izvrši prenosi se takode iz sekundarne u operativnu memoriju. Ukoliko pre toga u operativnoj memoriji postoji neki program ili podaci koji će se kasnije koristiti, oni se prenose u operativnu memoriju i tamo čuvaju do trenutka kada zatrebaaju. U tabeli T1 date su osnovne osobine operativne i sekundarne memorije.

Tabela T 1: Osnovne osobine operativne i sekundarne memorije

operativna memorija	procesor neposredno pristupa, brza, malog kapaciteta, poluprovodnička, gubi sadržaj kod prestanka napajanja <i>bucna gata do duzgy</i>
sekundarna memorija	procesor ne može neposredno da pristupi, spora, većeg kapaciteta, magnetni medijum, panti sadržaj kod prestanka napajanja <i>tucna gata do duzgy</i>

U radu sa sekundarnom memorijom korisnik je u velikoj meri angažovan: eksplicitno navodi, preko komandi koje izvršava operativni sistem, koje informacije treba preneti između operativne i sekundarne memorije. Operativni sistem u potpunosti upravlja prenosom podataka pri čemu koristi specijalizovane hardverske komponente. Principijelna šema računarskog sistema sa operativnom i sekundarnom memorijom prikazana je na sl. 3.



Sl. 3: Principijelna šema računara sa operativnom i sekundarnom memorijom

Ako, na primer, program koji korisnik želi da izvrši nije u operativnoj memoriji, korisnik preko odgovarajuće komande saopštava operativnom sistemu ime program koji treba preneti iz sekundarne u operativnu memoriju. Operativni sistem na osnovu tabela u kojima su smešteni podaci o sadržaju sekundarne memorije određuje koje delove sekundarne memorije treba preneti u operativnu memoriju. Na osnovu ovih podataka operativni sistem zadaje komande kontroleru koji je specijalizovana hardverska jedinica prilagođena karakteristikama komponentata sekundarne memorije. Kontroler obavlja sve operacije neophodne za prenos traženog programa iz sekundarne u operativnu memoriju. U ovom trenutku program je na raspolaganju centralnom procesoru i korisnik može uneti komandu za izvršenje programa.

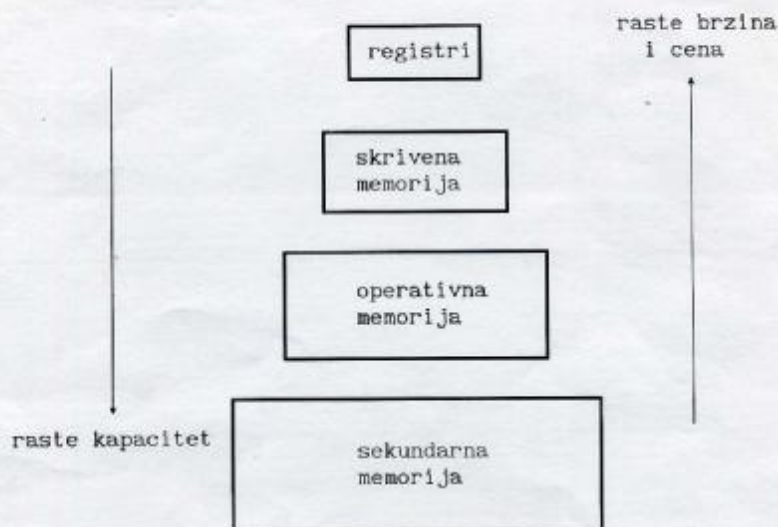
→ Zasniva se na **LOKALNOSTI** generisanih adresa:

MEMORIJSKA HIJERARHIJA

Ako je MP u jednom trenutku generisao adresu a , onda je velika verovatnoća da će u sledećem trenutku generisati adresu koja je bliska a .

Ubrzanje rada memorijskog podsistema postiže se daljim unapređenjem operativne memorije. Označimo skup adresa i sadržaja pridruženih lokacija operativne memorije kao operativni memorijski prostor. Delovi tog prostora mogu se držati u malim, ali vrlo brzim memorijskim komponentama. Ako centralni procesor često pristupa tim delovima operativnog memorijskog prostora onda se postiže ubrzanje rada računara.

Radi opštosti pristupa, registri opšte namene centralnog procesora mogu se smatrati delom memorije, koji ima najkraće vreme pristupa. Na ovaj način dolazi se do memorijske hijerarhije prikazane na sl. 4.



Sl. 3: Memorijska hijerarhija (relativni odnos kapaciteta kvalitativno odgovara veličini pravougaonika)

Na vrhu memorijske hijerarhije nalaze se registri centralnog procesora koji se mogu smatrati najbržim delom memorije, ali sa najmanjim kapacitetom. Skrivena memorija (engl. cache memory) u logičkom smislu nalazi se između

centralnog procesora i operativne memorije, i u nju se prenose delovi operativnog memorijskog prostora sa kojima centralni procesor u datom trenutku radi. Skrivena memorija realizuje se komponentama malog kapaciteta, ali velike brzine, što obezbeđuje efikasan rad memorijskog podsistema. Ako upravljanje memorijom obezbedi da u najvećem broju slučajeva centralni procesor pristupa skrivenoj memoriji onda će se značajno ubrzati rad računarskog sistema u odnosu na brzinu rada sistema u kome centralni procesor pristupa samo operativnoj memoriji.

Principijelna šema računarskog sistema sa skrivenom i operativnom memorijom prikazana je na sl. 4.



Sl. 4: Principijelna šema računara sa skrivenom i operativnom memorijom

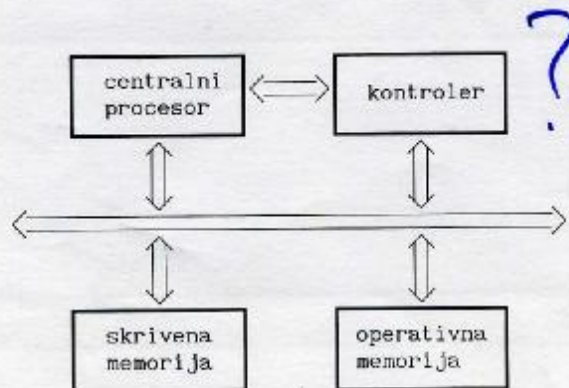
Centralni procesor može neposredno da pristupi lokacijama u skrivenoj i operativnoj memoriji, s obzirom da je skrivena memorija brža, centralni procesor prvo pokušava da pristupi lokacijama u skrivenoj memoriji. Ako je tražena informacija u skrivenoj memoriji, pristup se uspešno završava. Ako tražena informacija nije u skrivenoj memoriji, onda se deo operativnog memorijskog prostora sa traženom adresom prenosi iz operativne u skrivenu memoriju, a zatim se pristup završava. S obzirom da je prenesena ne samo tražena adresa i sadržaj već deo operativnog memorijskog prostora sa traženom adresom, očekuje se da će u sledećem pristupu tražena adresa i sadržaj već biti u skrivenoj memoriji. Osnove rada skrivene memorije opisani su u sledećem odeljku.

SKRIVENA MEMORIJA

Imajući u vidu da se provera sadržaja skrivene memorije mora vršiti u toku mašinskog ciklusa čitanja ili upisa, upravljanje skrivenom memorijom treba biti dovoljno brzo da ne usporava rad centralnog procesora. Tražena brzina postiže se primenom brzih kontrolera i jednostavnim algoritmom upravljanja. Kontroler skrivene memorije obavlja sledeće zadatke:

- upravlja prenosom delova operativnog memorijskog prostora,
- vodi evidenciju o delovima operativnog memorijskog prostora koji se nalaze u skrivenoj memoriji i
- određuje da li se adresirana reč nalazi u skrivenoj memoriji.

Blok šema računarskog sistema sa centralnim procesorom, skrivenom memorijom, kontrolerom i operativnom memorijom prikazana je na sl. 5.



Sl. 5: Principijelna šema računara sa skrivenom i operativnom memorijom

U opštem slučaju operacija čitanja (upisa) u računarskom sistemu sa skrivenom memorijom izvršava se na sledeći način:

- centralni procesor generiše adresu i upravljačke signale,
- kontroler proverí da li se adresirana reč nalazi u skrivenoj memoriji,
- ako je adresirana reč u skrivenoj memoriji reč se preko magistrala prenosi u procesor (kod upisa je obratno) i uspešno se završava mašinski ciklus čitanja (upisa),
- ako adresirana reč nije u skrivenoj memoriji kontroler prekida mašinski ciklus centralnog procesora, odredi deo skrivene memorije i u taj deo prenese kopiju dela operativnog memorijskog prostora u kome je adresirana reč,
- kontroler dozvoljava centralnom procesoru da nastavi prekinuti mašinski ciklus,
- centralni procesor čita (upisuje) adresiranu reč i završava mašinski ciklus čitanja (upisa).

Ako se, kod mašinskog ciklusa čitanja ili upisa, adresirana reč nalazi u skrivenoj memoriji kaže se da se dogodio pogodak, a u suprotnom je promašaj. Očigledno je da će računar sa skrivenom memorijom biti efikasan ako je dovoljno visok procenat pogodaka. U praksi je pokazano da je granica od oko 75 % pogodaka kritična: iznad te granice skrivena memorija ubrzava rad računara.

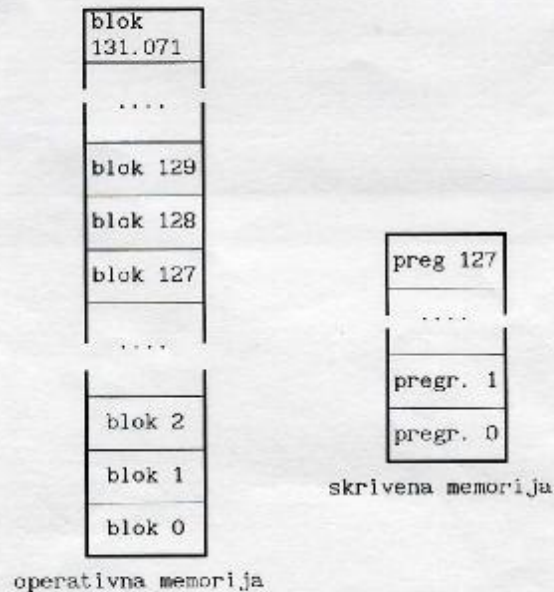
Organizacija skrivene memorije

Operativni memorijski prostor deli se na **blokove** od b reči, a skrivena memorija na **pregratke** sa b memorijskih lokacija, sl. 6.

Ako je kapacitet operativne memorije k_o , a kapacitet skrivene memorije k_s , onda su broj blokova n_b i broj pregradaka n_p :

$$n_b = k_o / b \quad n_p = k_s / b$$

pri čemu, da bi se postigla povoljna cena, mora da važi odnos $n_b \gg n_p$. U praks su kapaciteti operativne i skrivene memorije reda veličine Mb i kb, respektivno.



Sl. 6: Operativna memorija od 1Mb, skrivena memorija od 1kb i blokovi (pregradci) od 8b

S obzirom da postoji mnogo više blokova nego pregradaka, osnovni problem upravljanja skrivenom memorijom svodi se na određivanje pregradka u koji treba smestiti dati blok. Pravilo za određivanje pregradka u koji se prenosi blok naziva se funkcija preslikavanja. Iz praktičnih razloga, a pre svega radi postizanja maksimalne brzine upravljanja skrivenom memorijom, funkcija preslikavanja mora biti jednostavna. Razmotrićemo tri funkcije preslikavanja:

- direktno,
- asocijativno i
- kombinovano preslikavanje.

Direktno preslikavanje

Kod direktnog preslikavanja svaki blok može da se prenese samo u jedan, unapred određeni, pregradak. Na sl. 6. dat je primer operativne memorije kapaciteta 1Mb, skrivene memorije kapaciteta 1kb i veličine bloka 8b.

Kod direktnog preslikavanja unapred se određuje pregradak za svaki blok. Pošto nije dobro da se susedni blokovi preslikavaju u isti pregradak, najjednostavnije je obezbediti maksimalnu udaljenost blokova koji se preslikavaju u isti pregradak. Ako se blokovi podele u podskupove tako da se blokovi iz istog podskupa preslikavaju u isti pregradak, onda su za primer sa sl. 6 redni brojevi blokova iz istog podskupa:

$$\text{pregradak } 0 = \{\text{blokovi: } 0, 128, 256, \dots, \}$$

$$\text{pregradak } 1 = \{\text{blokovi: } 1, 129, 257, \dots, \}$$

$$\text{pregradak } 127 = \{\text{blokovi: } 127, 255, \overset{383}{\cancel{511}}, \dots, \}$$

383
511
7

Način određivanja broja pregradka u kome se nalazi adresirana reč može se utvrditi na osnovu posmatranja adresa reči koje pripadaju istom podskupu. Za posmatrani primer adrese su date na sl. 7.

memorijska adresa																				blok	preg-radak		
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	131071	127		
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0				
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1	128	0		
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0				
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	127	127		
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0			0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				

Sl. 7: Odnos adrese, broja bloka i broja pregradka (podskupa)

Iz ovog primera vidi se da najmanje značajna 3 bita adrese određuju adresu reči u jednom bloku (pošto je broj reči u bloku 8). Dalje se vidi da pošto je broj podskupova 128 (jednak broju pregradaka), sledećih 7 bita adrese su isti za svaki blok iz istog podskupa i oni određuju broj pregradka u koji se preslikava blok u kome se nalazi adresirana reč.

Preostalih 10 najznačajnijih bita adrese različiti su za svaki blok unutar jednog podskupa i oni su ključ (engl. tag) za identifikaciju blokova unutar podskupa koji se preslikavaju u isti pregradak. Prema tome, pojedini delovi adrese koriste se na način prikazan na sl. 8.



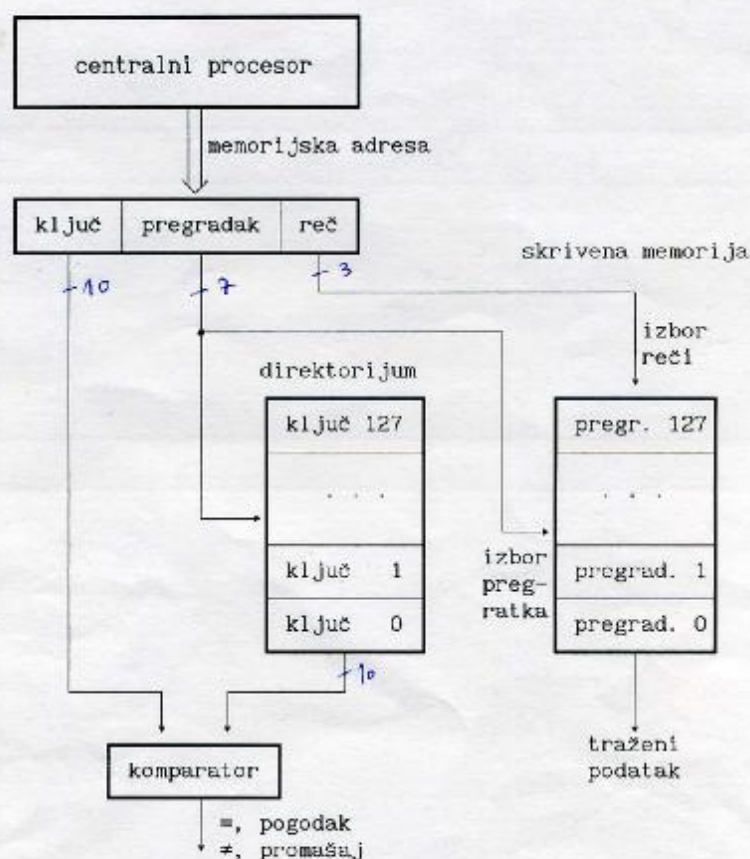
Sl. 8: Format memorijske adrese kod direktnog preslikavanja

Kontroler skrivene memorije mora voditi evidenciju o blokovima koji se nalaze u skrivenoj memoriji. Iz prethodnog razmatranja vidi se da je dovoljno za svaki pregradak pamtiti ključ bloka koji je preslikan u taj pregradak. Na sl. 9 prikazan je način provere da li je blok koji sadrži adresiranu reč u skrivenoj memoriji.

Ključevi blokova smeštenih u skrivenoj memoriji upisuju se u **direktorijum**, brzu memoriju koja za svaki pregradak ima jednu memorijsku lokaciju dužine koja odgovara broju pregradaka. Organizacija direktorijuma za primer sa sl. 7 je 128×10 bita.

Kada centralni procesor generiše adresu, kontroler skrivene memorije uzima deo adrese **pregradak** (za posmatrani primer biti od 3 do 9), koje koristi za adresiranje direktorijuma. U lokaciji direktorijuma, koja je na zadatoj adresi, nalazi se ključ bloka koji je smešten u odgovarajućem pregradku. Ako je ključ jednak ključu iz generisane adrese (za posmatrani primer najznačajnijih 10 bita) onda se dogodio pogodak i kontroler dozvoljava procesoru da nesmetano završi započeti pristup memoriji. Najmanje značajni biti adrese (reč) koriste se za **adresiranje reči u okviru pregradka** skrivene memorije i iz te lokacije uzima se podatak (kod čitanja) ili se u tu lokaciju upisuje podatak (kod upisa).

Ako ključevi nisu jednaki, kontroler zaustavlja pristup procesora i organizuje prenos traženog bloka iz operativne memorije u adresirani pregradak. Naravno, ukoliko se u pregradku nalazi blok čiji je sadržaj izmenjen operacijom upisa, kontroler mora da taj blok vrati u operativnu memoriju pre nego što njegovo mesto zauzme traženi blok.



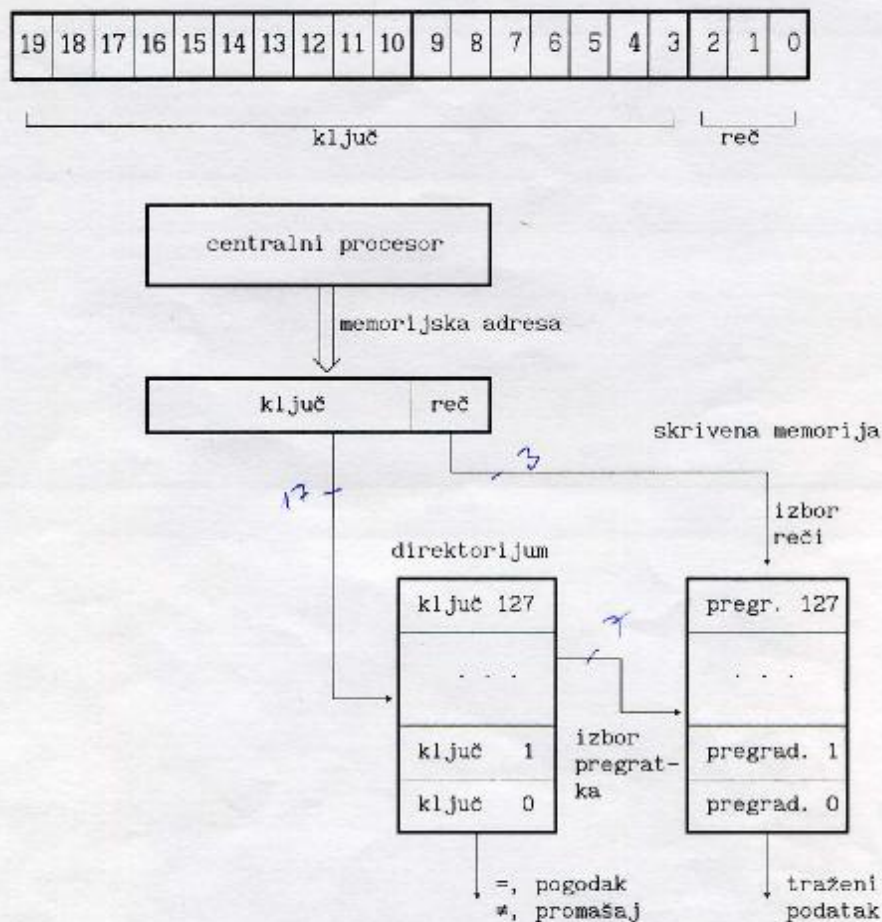
Sl. 9: Principijelna šema provere da li je adresirani blok u skrivenoj memoriji kod direktnog preslikavanja

Kada se traženi blok (uprimeru su blokovi od 8 bajta) prenese iz operativne memorije u pregradak, ključ tog bloka upiše se u direktorijum i kontroler dozvoljava procesoru da završi započeti pristup.

Asocijativno preslikavanje

Kod asocijativnog preslikavanja svaki blok može da se preslika u bilo koji pregradak. U ovom slučaju moraju se pretraživati sve lokacije direktorijuma i upoređivati traženi ključ sa svim ključevima koji se nalaze u direktorijumu. Radi što bržeg pretraživanja direktorijum se realizuje u obliku **asocijativne memorije**.

S obzirom da blok može biti u bilo kome pregratku, ključ bloka sastoji se iz svih bita adrese osim bita za **adresiranje reči unutar bloka**. Za primer sa sl. 7 format adrese u slučaju asocijativnog preslikavanja i postupak provere da li je adresirani blok u skrivenoj memoriji prikazani su na sl. 10.



Sl.10: Principijelna šema provere da li je adresirani blok u skrivenoj memoriji kod asocijativnog preslikavanja

Kada centralni procesor generiše adresu, **ključ** (u primeru 17 bita) upoređuje se sa svim ključevima smeštenim u direktorijumu. U slučaju pogotka, iz direktorijuma se dobija adresa pregradka u kome se nalazi blok sa adresiranom reči. Polje **reč** memorijske adrese (u primeru najmanje značajnih 3 bita) adresira lokaciju unutar pregradka u koju se upisuje podataka (kod upisa) ili iz koje se čita podatak (kod čitanja).

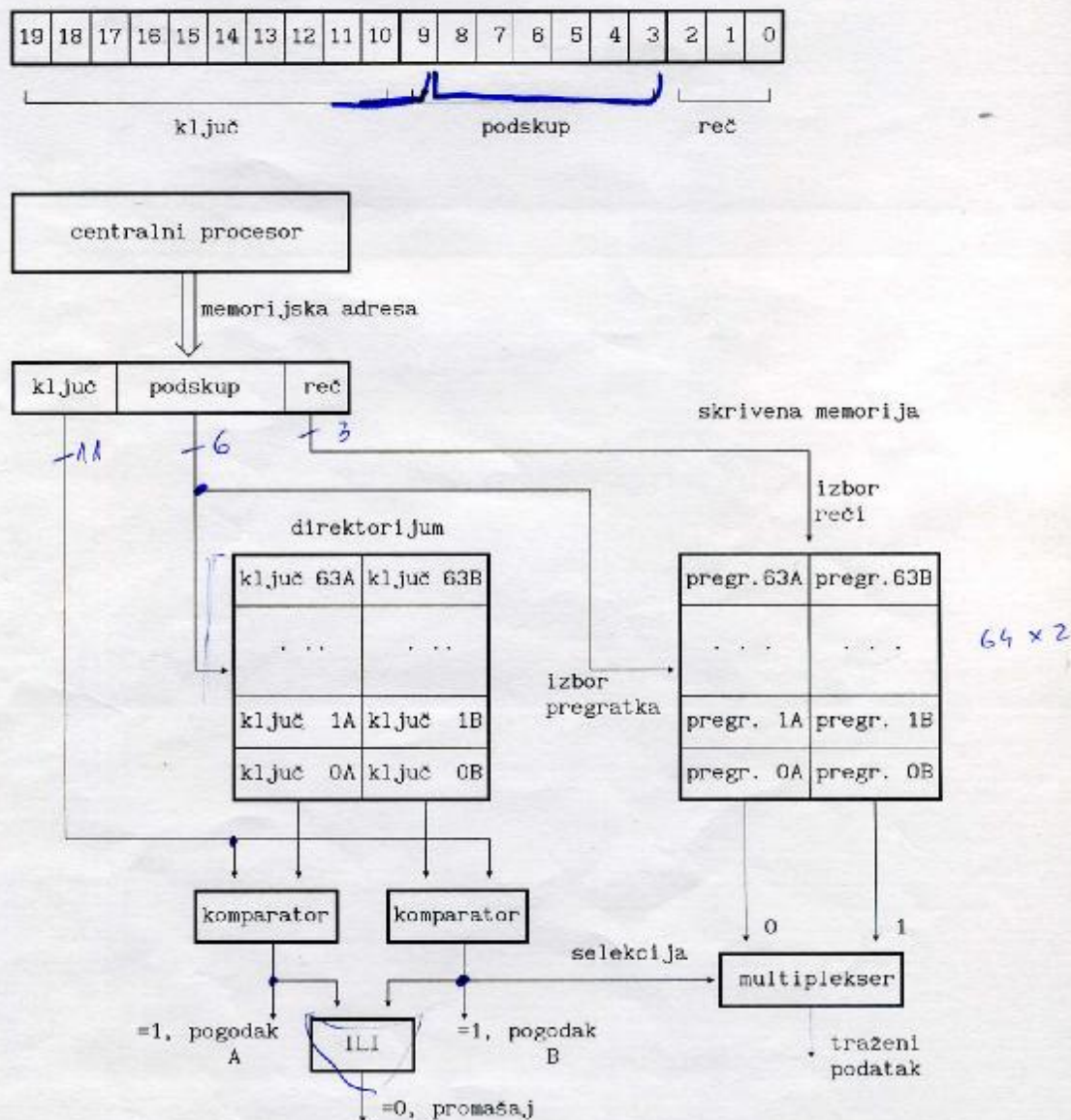
Ukoliko se dogodio **promašaj**, kontroler zaustavlja centralni procesor i preduzima akcije prenosa traženog bloka iz operativne memorije u skrivenu memoriju. Ovaj postupak sličan je postupku opisanom za direktno preslikavanje.

Kombinovano preslikavanje

Direktno preslikavanje daje veoma slab procenat pogodaka ako centralni procesor naizmenično pristupa različitim blokovima koji se preslikavaju u isti pregradak. Asocijativno preslikavanje efikasno rešava ovaj problem ali

je realizacija asocijativnog preslikavanja složena: zahteva primenu asocijativne memorije i dugačkog ključa u poređenju sa ključem za direktno preslikavanje. U praksi se primenjuje preslikavanje koje koristi dobre osobine ove dve vrste preslikavanja.

Kombinovano preslikavanje dozvoljava da se blokovi iz istog podskupa mogu preslikati u dva ili više pregradaka. Na sl. 11 prikazan je primer provere ključa kod kombinovanog preslikavanja sa dva pregradka po podskupu.



Sl. 11: Format adrese i postupak provere ključa kod kombinovanog preslikavanja sa dva pregradka po podskupu

Pretpostavimo da operativna i skrivena memorija imaju kapacitet kao što je prikazano na sl. 6. U tom slučaju skrivena memorija ima isto 128 pregradaka, ali su oni organizovani u parovima: 64 para pregradaka, dva pregratka A i B, po jednom podskupu blokova. Na taj način moguće je da istovremeno budu dva bloka iz istog podskupa u skrivenoj memoriji, jedan u pregratku A i drugi u pregratku B.

Direktorijum se takođe sastoji iz parova lokacija, tako da je u lokaciji A (B) ključ bloka smeštenog u pridruženom pregratku A (B).

Format adrese ^{gledam} ~~identičan~~ je formatu kod direktnog preslikavanja, ali je hardverska podrška složenija. Kada centralni procesor generiše adresu, polje podskup (odgovara polju pregradak kod direktnog preslikavanja) koristi se za adresiranje direktorijuma, iz koga se uzimaju ključevi A i B blokova smeštenih u pregratke A i B traženog podskupa. Upoređenje sa traženim ključem može dati sledeće rezultate:

- pogodak, blok je u pregratku A,
- pogodak, blok je u pregratku B i
- promašaj.

U slučaju pogotka, primenom multipleksera selektuje se reč iz pregratka u kome je traženi blok.

Kod promašaja kontroler zaustavlja rad centralnog procesora i preduzima akcije prenosa traženog bloka u skrivenu memoriju. S obzirom da se blok može preneti u dva pregratka, neophodno je prvo odrediti pregradak, zatim preneti traženi blok iz operativne memorije u pregradak i smestiti ključ prenetog bloka u odgovarajuću lokaciju (A ili B) direktorijuma. Posle toga kontroler dozvoljava centralnom procesoru da završi započeti mašinski ciklus.

Problem upisa u skrivenu memoriju

Kod operacije upisa menja se sadržaj skrivene memorije što ima za posledicu da sadržaj pregratka nije identičan odgovarajućem bloku operativne memorije. Ovaj problem rešava se na dva načina.

Upis u skrivenu memoriju. Kod operacije upisa podatak se upisuje u skrivenu memoriju tako da je sadržaj pregratka nije identičan odgovarajućem bloku operativne memorije. U ovom slučaju za svaki pregradak vodi se evidencija da li je sadržaj pregratka izmenjen. Ako jeste, kod zamene blokova, prethodni blok se prenosi u operativnu memoriju pre nego što se novi blok upiše u pregradak. Ovaj pristup nije dovoljno efikasan zato što se gubi vreme kod prenosa bloka iz skrivene u operativnu memoriju.

Upis u skrivenu i operativnu memoriju. Kod operacije upisa podatak se istovremeno upisuje u skrivenu i operativnu memoriju. Na taj način obezbeđeno je da su sadržaji pregradaka i odgovarajućih blokova operativne memorije uvek isti i nije neophodno prenositi blokove iz skrivene memorije u operativnu memoriju. Iako operacija upisa u operativnu memoriju usporava rad računara sa skrivenom memorijom to usporenje nije značajno s obzirom da je eksperimentalno pokazano da od ukupnog broja pristupa memoriji tek oko 15% odnosi se na upis, a 85% pristupa na čitanje.