Univerzitet Singidunum

Violeta Tomašević

OSNOVI ARHITEKTURE I ORGANIZACIJE RAČUNARA

Beograd, 2019.

UNIVERZITET SINGIDUNUM

Violeta Tomašević

OSNOVI ARHITEKTURE I ORGANIZACIJE RAČUNARA

Prvo izdanje

Beograd, 2019.

OSNOVI ARHITEKTURE I ORGANIZACIJE RAČUNARA

Autor:
dr Violeta Tomašević

Recenzenti:
dr Jovan Đorđević
dr Milan Milosavljević

Izdavač:
UNIVERZITET SINGIDUNUM
Beograd, Danijelova 32
www.singidunum.ac.rs

Za izdavača:
dr Milovan Stanišić

Tehnička obrada:
Violeta Tomašević

Dizajn korica:
Aleksandar Mihajlović

Godina izdanja:
2019.

Tiraž:
1200 primeraka

Štampa:
Caligraph, Beograd

ISBN: 978-86-7912-696-2

P r e d g o v o r

Ova knjiga je nastala kao rezultat potrebe za odgovarajućim udžbenikom iz predmeta Arhitektura računara koji autorka drži na Fakultetu za informatiku i računarstvo i Tehničkom fakultetu Univerziteta Singidunum. Pri pisanju je učinjen napor da knjiga bude prihvatljiva za čitaoce bez nekog većeg predznanja iz oblasti računarstva. Namenjena je i prilagođena prosečnom studentu, jer je osnovni cilj autorke bio da svi studenti koji slušaju predmet mogu na razumljiv i lak način da savladaju predviđeno gradivo. Upravo iz tog razloga, knjiga ne prikazuje punu širinu i složenost razmatranih problema, već je prvenstveno orijentisana ka praktičnim aspektima koji su ilustrovani brojnim primerima.

Knjiga je podeljena u četrnaest poglavlja: *Matematičke osnove*, *Logička kola*, *Logičke funkcije*, *Standardni moduli*, *Komponente računara*, *Mehanizmi*, *Programske instrukcije*, *Procesorski registri*, *Adresni modovi*, *Instrukcijski set*, *Programiranje*, *Memorijski sistem*, *Organizacija ulaza/izlaza* i *Magistrala*.

U prvom poglavlju izložen je matematički aparat na kome se zasniva rad računarskog sistema. S obzirom da se podaci u računaru predstavljaju i obrađuju u binarnom obliku, najveća pažnja posvećena je binarnom brojnom sistemu. Osim njega, razmatran je i heksadecimalni brojni sistem. U poglavlju su dati postupci konverzije brojeva između binarnog, decimalnog i heksadecimalnog sistema, kao i osnovne aritmetičke operacije nad binarnim brojevima. Takođe su opisani načini predstavljanja različitih tipova podataka u računaru, uključujući cele brojeve, realne brojeve i podatake znakovnog tipa.

Osnovne logičke operacije su tema drugog poglavlja. One se realizuju odgovarajućim logičkim kolima koja se, kao gradivni elementi, koriste u prekidačkim mrežama.

Treće poglavlje uvodi pojam logičke funkcije kojom se mogu predstaviti razne funkcionalnosti u računarskom sistemu. Opisana su tri načina za predstavljanje logičke funkcije: pomoću kombinacione tablice, u algebarskom obliku i pomoću Karnoove karte. Dat je postupak realizacije logičkih funkcija pomoću prekidačkih mreža. Na kraju je izložen metod minimizacije logičkih funkcija pomoću Karnoove karte u cilju smanjenja složenosti realizacije.

U četvrtom poglavlju opisano je više prekidačkih mreža koje predstavljaju standardne module kombinacionog (koderi, dekoderi, multiplekseri, demultiplekseri, sabirači i aritmetičko-logička jedinica) i sekvencijalnog (registri, brojači i memorije) tipa. Za svaki modul dati su njegova funkcionalnost, osnovne osobine, mogućnosti primene i način realizacije.

Peto poglavlje opisuje osnovne komponente računara na opštem nivou: procesor, memoriju, ulazno/izlazne jedinice i magistralu. Uvode se i pojmovi arhitekture i organizacije računara, uz jasno isticanje razlike između njih.

U šestom poglavlju su izložena tri važna koncepta koja doprinose većoj efikasnosti rada računara: *pipeline* mehanizam, *DMA* koncept prenosa i mehanizam prekida.

Sedmo poglavlje je posvećeno različitim formatima instrukcija, u zavisnosti od broja polja u adresnom delu instrukcije. Dat je i kratak pregled tipova operanada nad kojima se instrukcije izvršavaju, kao i mogućih lokacija za smeštaj operanada.

U osmom poglavlju su opisane dve vrste procesorskih registara: interni registri i programski dostupni registri. Takođe je dat i detaljan opis faza u ciklusu izvršavanja instrukcije.

Deveto poglavlje se bavi načinima adresiranja operanada u instrukcijama, tj. adresnim modovima. Razmatrane su četiri grupe adresnih modova: neposredno adresiranje, direktna adresiranja, indirektna adresiranja i adresiranja sa pomerajem. Mogućnosti primene pojedinih adresnih modova su ilustrovane opisom situacija u kojima ih je pogodno koristiti.

Pojam instrukcijskog seta uveden je u desetom poglavlju. Detaljno je predstavljen instrukcijski set koga čine standardne instrukcije svrstane u pet grupa: aritmetičke instrukcije, logičke instrukcije, pomeračke instrukcije, instrukcije prenosa i instrukcije skoka.

Tema jedanaestog poglavlja je programiranje na različitim nivoima apstrakcije, počevši od najnižeg, mašinskog nivoa, preko programiranja na simboličkom, asemblerskom jeziku, do najvišeg nivoa apstrakcije koji odgovara programiranju na višim programskim jezicima. Posebna pažnja je posvećena ulozi arhitekture računara na svakom nivou programiranja.

Dvanaesto poglavlje se bavi memorijskim sistemom računara. Nakon uvođenja hijerarhijske organizacije memorijskog sistema, detaljno su izloženi principi rada keš memorije, operativne memorije i virtuelne memorije. Keš memorija je opisana kroz tehnike preslikavanja blokova, tehnike zamene blokova i tehnike ažuriranja operativne memorije. Koncept vituelne memorije dat je na primeru virtuelne memorije sa straničnom organizacijom.

U trinaestom poglavlju je opisana organizacija ulaza/izlaza. Data je opšta struktura *U/I* uređaja i predstavljeni su mogući načini prenosa podataka između *U/I* uređaja i procesora/memorije.

Poslednje poglavlje je posvećeno prenosu podataka unutar računara. Opisana je struktura magistrale putem tri vrste linija za prenos: adresnih linija, linija podataka i kontrolnih (upravljačkih) linija. Posebna pažnja je posvećena postupcima arbitracije na magistrali, tj. odlučivanja o tome kome će magistrala biti

dodeljena u datom trenutku. Predstvaljene su dve vrste magistrala: asinhrona i sinhrona.

Na kraju svakog poglavlja, u okviru *Vežbanja*, data su pitanja i zadaci za samostalno rešavanje koji studentima treba da posluže kao provera znanja na temu koja je razmatrana u poglavlju. Pitanja i zadaci su odabrani tako da u potpunosti pokrivaju predviđeno gradivo, pa se mogu iskoristiti za pripremanje ispita.

Zahvaljujem se prof. dr Jovanu Đorđeviću na pomoći pri odabiru gradiva, materijalima i korisnim sugestijama tokom izrade ove knjige.

Biću zahvalna svima onima koji mi ukažu na greške ili daju korisne savete za buduće ispravke i dopune ovog materijala.

Beograd, septembar 2018.god. Autorka

UNIVERZITET
PRIVATNI
AKREDITOVANI
UNIVERZITET
SINGIDUNUM

S A D R Ž A J

Literatura

UNIVERZITET
PRIVATNI
AKREDITOVANI
UNIVERZITET
SINGIDUNUM

1 Matematičke osnove

Računar je elektronski uređaj koji služi za obradu podataka. Da bi se razumeo način rada računara, neophodno je upoznati se sa osnovnim matematičkim aparatom na kome se taj rad zasniva. Tu se, pre svega, misli na binarni brojni sistem. Ovaj sistem pripada klasi pozicionih brojnih sistema i pogodan je za opisivanje situacija sa dva stanja, što je primereno mogućnostima današnje elektronske tehnologije. Međutim, binarna predstava nije bliska korisnicima računara jer su oni navikli na decimalni brojni sistem. Kao neka vrsta posrednika između ova dva sistema, razmatra se heksadecimalni sistem. Primena navedenih brojnih sistema zahteva poznavanje postupaka konverzije podataka iz jednog sistema u drugi.

Podaci se u računaru predstavljaju u binarnom obliku. Njihova obrada podrazumeva izvršavanje određenih operacija nad binarnim brojevima (na primer, osnovnih aritmetičkih operacija: sabiranja, oduzimanja, množenja i deljenja).

U zavisnosti od problema koji se rešava, koriste se podaci različitog tipa, kao što su prirodni brojevi, celi brojevi, realni brojevi, podaci znakovnog tipa, itd. U zavisnosti od tipa, podaci se u računaru predstavljaju na različite načine.

1.1. Pozicioni brojni sistemi

Pozicioni brojni sistemi su sistemi zapisivanja brojeva u kojima vrednost broja zavisi od:

- *cifara* upotrebljenih za zapisivanje broja

- *pozicije* svake cifre u broju

Broj različitih cifara koje se mogu koristiti u zapisu bilo kog broja naziva se *osnovom brojnog sistema*.

Najčešće korišćeni pozicioni brojni sistem je decimalni (dekadni) brojni sistem. Osnova ovog sistema je 10 (cifre od 0 do 9). Mesne vrednosti cifara na susednim pozicijama u decimalnom broju razlikuju se 10 puta. Tako, cifra na mestu jedinica (pozicija 0) doprinosi vrednosti broja sa 1, cifra na mestu desetica (pozicija 1) sa 10, cifra na mestu stotina (pozicija 2) sa 100, itd.

Decimalni brojevi $68_{(10)}$ i $291_{(10)}$ imaju različite vrednosti zato što su zapisani različitim ciframa (videti prvi uslov u definiciji pozicionog brojnog sistema). Osim upotrebljenih cifara, na vrednost decimalnog broja utiču i pozicije na kojima se cifre nalaze (drugi uslov u definiciji pozicionog brojnog sistema). Tako, iako su zapisani istim ciframa, decimalni brojevi $276_{(10)}$ i $762_{(10)}$ imaju različite vrednosti zato što su cifre u njima zapisane u drugačijem redosledu. Oznaka $_{(10)}$ u indeksu navedenih brojeva označava osnovu brojnog sistema u kome su brojevi zapisani.

U opštem slučaju, bilo koji pozitivan ceo broj X u pozicionom brojnom sistemu može se zapisati u sledećem obliku:

$$X = a_n q^n + a_{n-1} q^{n-1} + \ldots + a_2 q^2 + a_1 q^1 + a_0 q^0 \qquad (1)$$

pri čemu je:

q – prirodan broj koji predstavlja osnovu brojnog sistema

a_i, $0 \le i \le n$ – cifre u zapisu broja X (svaka cifra mora da pripada dozvoljenom skupu cifara S_q za dati brojni sistem)

n – broj cifara u zapisu broja X umanjen za 1 (jer je najniža pozicija u broju pozicija 0)

U skladu sa (1), decimalni broj $3827_{(10)}$ može se zapisati u obliku:

$$3827 = 3 \cdot 10^3 + 8 \cdot 10^2 + 2 \cdot 10^1 + 7 \cdot 10^0$$

U ovom primeru broj cifara u zapisu je 4, pa je $n = 3$, osnova brojnog sistema je $q = 10$, a cifre u zapisu broja, $a_3 = 3$, $a_2 = 8$, $a_1 = 2$ i $a_0 = 7$, pripadaju skupu cifara koji odgovara decimalnom brojnom sistemu $S_{10} = \{0,1,2,3,4,5,6,7,8,9\}$.

Pozicioni brojni sistemi mogu imati proizvoljnu osnovu, ali u praktičnim primenama najzastupljeniji su sistemi prikazani u tabeli 1.1.

Brojni sistem	Osnova brojnog sistema (q)	Skup dozvoljenih cifara (S_q)
binarni	2	$S_2 = \{0,1\}$
oktalni	8	$S_8 = \{0,1,2,3,4,5,6,7\}$
decimalni	10	$S_{10} = \{0,1,2,3,4,5,6,7,8,9\}$
heksadecimalni	16	$S_{16} = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

Tabela 1.1 Najčešće korišćeni pozicioni brojni sistemi

1.1.1 Binarni brojni sistem

Binarni brojni sistem je sistem u kome se za predstavljanje brojeva koriste samo dve cifre: 0 i 1. Primer binarnog broja je $1011_{(2)}$ (broj 2 u indeksu je osnova brojnog sistema i važno ga je navesti, jer isti zapis postoji i u drugim sistemima, na primer, u decimalnom sistemu bi ovo bio broj hiljadu jedanaest). Ovakav način predstavljanja podataka je pogodan za primenu u računarskim i drugim digitalnim sistemima. Naime, u ovim sistemima postoji česta potreba za opisivanjem stanja kada „ima signala“ ili „nema signala“, neki uređaj je „uključen“ ili „isključen“, podatak je „raspoloživ“ ili „nije raspoloživ“ i slično, što se može predstaviti binarnim vrednostima 0 i 1.

Pošto je osnova binarnog brojnog sistema $q = 2$, formula kojom je predstavljen pozitivan ceo broj X u ovom sistemu dobija oblik:

$$X = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_2 2^2 + a_1 2^1 + a_0 2^0 \qquad (2)$$

Iz ove jednačine sledi da se mesne vrednosti cifara na susednim pozicijama u binarnom broju razlikuju 2 puta. To znači da cifra na poziciji 0 (prva cifra sa desne strane u broju) doprinosi vrednosti binarnog broja sa $2^0 = 1$, cifra na poziciji 1 sa $2^1 = 2$, cifra na poziciji 2 sa $2^2 = 4$, cifra na poziciji 3 sa $2^3 = 8$, itd.

Binarno-decimalne konverzije brojeva

Iako binarni brojni sistem najviše odgovara mogućnostima savremene elektronske tehnologije, za ljude je mnogo prihvatljiviji decimalni brojni sistem sa kojim dolaze u dodir u ranom periodu svog života i dalje ga aktivno primenjuju i usvajaju. Stoga, da bi čovek mogao da razume način funkcionisanja računara, neophodno je da poznaje načine konvertovanja podataka iz binarnog u decimalni oblik i obrnuto.

Konverzija binarnog broja u decimalni vrši se primenom jednačine (2). Decimalna vrednost koja odgovara zadatom binarnom broju dobija se kao suma mesnih vrednosti na kojima binarni broj ima vrednost 1.

Neka je dat binarni broj $10010110_{(2)}$. Primenom jednačine (2), decimalna vrednost ovog broja računa se na sledeći način:

$$X = 1\cdot2^7 + 0\cdot2^6 + 0\cdot2^5 + 1\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 0\cdot2^0$$

$$X = 1\cdot2^7 + 1\cdot2^4 + 1\cdot2^2 + 1\cdot2^1 = 1\cdot128 + 1\cdot16 + 1\cdot4 + 1\cdot2 = 150_{(10)}$$

Zadati binarni broj ima vrednost 1 na pozicijama 1, 2, 4 i 7. Mesne vrednosti na tim pozicijama su 2^1, 2^2, 2^4 i 2^7, pa se decimalna vrednost broja dobija njihovim sabiranjem.

Konverzija decimalnog broja u binarni odvija se u dva koraka:

1. zadati decimalni broj podeliti sa 2 sa ostatkom; ostatak zapisati, a rezultat deljenja ponovo podeliti sa 2 sa ostatkom; dobijeni ostatak opet zapisati, a rezultat deljenja ponovo podeliti sa 2 sa ostatkom; ovaj postupak ponavljati sve dok se kao rezultat deljenja ne dobije vrednost 0
2. binarni broj formirati od zapisanih ostataka u obrnutom redosledu od onoga u kome su nastajali

Neka je dat decimalni broj $169_{(10)}$. Primenom prvog koraka dobija se:

	rezultat deljenja	ostatak	
169 : 2 =	84	(1)	(*LSB*)
84 : 2 =	42	(0)	
42 : 2 =	21	(0)	
21 : 2 =	10	(1)	
10 : 2 =	5	(0)	
5 : 2 =	2	(1)	
2 : 2 =	1	(0)	
1 : 2 =	0	(1)	(*MSB*)

U skladu sa drugim korakom, od dobijenih ostataka može se formirati binarni broj koji odgovara zadatom decimalnom broju $169_{(10)}$. To se radi tako što poslednji ostatak predstavlja cifru najveće težine u binarnom broju (*MSB – most significant bit*), a prvi ostatak cifru najmanje težine (*LSB – least significant bit*). Dakle, decimalni broj $169_{(10)}$ se u binarnom obliku predstavlja zapisom $10101001_{(2)}$.

Kao što se vidi, binarni zapis nekog broja uglavnom sadrži više cifara od njegovog decimalnog zapisa. Na primer, broj $169_{(10)}$ se u dekadnom sistemu

zapisuje pomoću tri cifre, dok je za njegovu binarnu predstavu potrebno osam cifara. Predugačak zapis nije pogodan za čoveka, pa predstavlja osnovni nedostatak binarnog brojnog sistema.

Aritmetičke operacije nad binarnim brojevima

Obrada binarnih podataka u računaru zahteva intenzivnu primenu osnovnih aritmetičkih operacija: sabiranja, oduzimanja, množenja i deljenja. Pri obavljanju ovih operacija, binarni brojevi koji u njima učestvuju tretiraju se kao celina, što znači da se uzima u obzir međusobni uticaj susednih pozicija u broju. Aritmetičke operacije nad binarnim brojevima izvode se po istim pravilima kao i aritmetičke operacije nad decimalnim brojevima (oba sistema su poziciona), s tim što se mora imati u vidu da se mesne vrednosti susednih pozicija razlikuju 2 puta, a ne 10 kao u decimalnom brojnom sistemu.

Sabiranje je binarna operacija jer se obavlja nad dva binarna broja. Slično sabiranju decimalnih brojeva, binarno sabiranje se vrši tako što se sabiraju vrednosti na istim pozicijama u binarnim brojevima. Ukoliko zbir na nekoj poziciji premaši vrednost 1 (bude 2 ili više), javlja se prenos za narednu poziciju. Prenos koji se pojavi na nekoj poziciji mora se uzeti u obzir pri sabiranju na narednoj poziciji.

Pošto se u binarnim brojevima na jednoj poziciji mogu naći samo 0 ili 1, moguće su sledeće četiri situacije (prvi sabirak je cifra prvog broja, a drugi cifra drugog broja na istoj poziciji):

$0_{(2)} + 0_{(2)} = 0_{(2)}$ $\quad$ $0_{(2)} + 1_{(2)} = 1_{(2)}$

$1_{(2)} + 0_{(2)} = 1_{(2)}$ $\quad$ $1_{(2)} + 1_{(2)} = 10_{(2)}$

Ukoliko na nekoj poziciji postoji prenos sa niže pozicije, može nastati još jedna situacija (prvi i drugi sabirak imaju značenje kao i ranije, a treći sabirak predstavlja prenos):

$1_{(2)} + 1_{(2)} + 1_{(2)} = 11_{(2)}$

Može se zapaziti da se u dve situacije (kada su rezultati sabiranja $10_{(2)} = 2_{(10)}$ i $11_{(2)} = 3_{(10)}$) javlja prenos za narednu poziciju (premašena je vrednost 1). U oba slučaja, stvarni rezultat sabiranja na tekućoj poziciji predstavlja cifru najmanje težine u broju (u prvom slučaju to je cifra 0, a u drugom 1), dok se 1 prenosi na narednu poziciju.

Neka su a i b cifre na istoj poziciji u binarnim brojevima koje treba sabrati. Označimo sa c_{ul} prenos sa prethodne pozicije, a sa c_{iz} prenos za narednu poziciju.

Rezultat sabiranja na posmatranoj poziciji je *s*. Uz uvedene oznake, postupak sabiranja binarnih vrednosti *a* i *b* može se predstaviti tabelom 1.2.

Tabela 1.2 ilustruje prethodno opisane situacije. Na primer, ukoliko treba sabrati vrednosti $a = 1$ i $b = 1$, a prenos sa niže pozicije ne postoji $c_{ul} = 0$ (vrsta 4), rezultat sabiranja je $10_{(2)}$, što znači da je rezultat na posmatranoj poziciji $s = 0$, a prenos za narednu poziciju $c_{iz} = 1$. Slično, ako su $a = 1$ i $b = 0$, i postoji prenos sa niže pozicije $c_{ul} = 1$ (vrsta 7), rezultat sabiranja je $10_{(2)}$, što znači da je rezultat na tekućoj poziciji $s = 0$, a prenos za narednu poziciju $c_{iz} = 1$. Na sličan način se mogu analizirati i sve ostale vrste u tabeli.

c_{ul}	a	b	c_{iz}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabela 1.2 Sabiranje binarnih brojeva

U nastavku će operacija sabiranja biti prikazana na jednom primeru. Neka su dati binarni brojevi $A = 10110111_{(2)}$ i $B = 10011010_{(2)}$ koje treba sabrati. Postupak sabiranja se može predstaviti na sledeći način:

c_{ul}		1	0	1	1	1	1	1	0
A			1	0	1	0	1	1	1	1
B	+		1	0	0	1	1	0	1	0
$A+B$		1	0	1	0	0	1	0	0	1

Brojevi *A* i *B* sabiraju se tako što se najpre saberu cifre na poziciji 0. Kao rezultat dobija se 1+0 = 1 i nema prenosa za narednu poziciju. Zatim se sabiraju cifre na poziciji 1, tj. 1+1 = 10. Rezultat na poziciji 1 je 0, a prenos za narednu poziciju je 1. Na poziciji 2 sabiraju se cifre 1 i 0, ali se mora uključiti i prenos 1 koji dolazi sa niže pozicije. Tako se dobija 1+0+1 = 10, pa je na poziciji 2 rezultat 0 i postoji prenos za narednu poziciju. Cifre na poziciji 3 sabiraju se prenosom sa niže pozicije, tj. 1+1+1 = 11, pa je rezultat na ovoj poziciji 1 i postoji prenos za narednu poziciju. Ovaj postupak se nastavlja dok se ne dođe do najviše pozicije u brojevima koji se sabiraju.

Oduzimanje binarnih brojeva predstavlja binarnu operaciju koja se vrši tako što se oduzimaju vrednosti na istim pozicijama u brojevima. Kao i kod oduzimanja decimalnih brojeva, i u ovom slučaju, ukoliko je vrednost od koje se oduzima manja od vrednosti koja se oduzima, neophodno je uzeti pozajmicu sa više pozicije. Ta pozajmica, kada pređe na nižu poziciju, ima vrednost 2 (u decimalnom sistemu je vredela 10), jer je u binarnom sistemu odnos mesnih vrednosti između susednih pozicija 2.

Pošto se binarni brojevi zapisuju samo nulama i jedinicama, pri oduzimanju su moguće sledeće situacije (umanjenik je cifra prvog broja, a umanjilac cifra drugog broja na datoj poziciji):

$0_{(2)} - 0_{(2)} = 0_{(2)}$ $\quad$ $1_{(2)} - 0_{(2)} = 1_{(2)}$ $\quad$ $1_{(2)} - 1_{(2)} = 0_{(2)}$

Ostalo je još da se razmotri slučaj $0_{(2)} - 1_{(2)}$. Pošto je umanjenik manji od umanjioca, mora se uzeti pozajmica sa naredne više pozicije. To znači da se na narednoj poziciji vrednost umanjuje za 1, a na tekućoj poziciji se dodaje vrednost $10_{(2)} = 2_{(10)}$. Tako rezultat oduzimanja na tekućoj poziciji postaje 1 (2–1 = 1). Zatim se prelazi na oduzimanje na narednoj poziciji, pri čemu se mora voditi računa da je sa nje uzeta pozajmica.

Treba zapaziti da se na istoj poziciji mogu pojaviti dve pozajmice: pozajmica koja je od tekuće pozicije tražena sa niže pozicije i pozajmica koja je sa tekuće pozicije tražena od naredne, više pozicije.

Neka su *a* i *b* cifre na istoj poziciji u binarnim brojevima koje treba oduzeti. Označimo sa p_{ul} pozajmicu koja je data prethodnoj poziciji, a sa p_{iz} pozajmicu od naredne pozicije. Rezultat oduzimanja na posmatranoj poziciji je *r*. Sada se postupak oduzimanja binarnih vrednosti *a* i *b* može predstaviti tabelom 1.3.

p_{ul}	*a*	*b*	p_{iz}	*r*
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

Tabela 1.3 Oduzimanje binarnih brojeva

Razmotrimo drugu vrstu u tabeli. Dakle, treba naći razliku *a*–*b* = 0–1, pri čemu nema pozajmice sa niže pozicije ($p_{ul} = 0$). Da bi se to uradilo, mora se sa više pozicije uzeti pozajmica, pa je zato $p_{iz} = 1$. Ova pozajmica vredi 2 na nižoj poziciji,

pa kada se od nje oduzme 1 dobija se rezultat $r = 1$. Nešto komlikovanija situacija data je u osmoj vrsti. U ovom slučaju treba oduzeti vrednosti $a–b = 1–1$, ali imajući u vidu da je sa ove pozicije već data pozajmica nižoj poziciji jer je $p_{ul} = 1$. Da bi se sprovelo oduzimanje, najpre treba uzeti pozajmicu sa više pozicije. Ova pozajmica na tekućoj poziciji ima vrednost 2, od čega treba jednu jedinicu odvojiti za pozajmicu p_{ul}, dok druga ostaje na razmatranoj poziciji. Tako se na tekućoj poziciji dobija 1+1 = 2, pa kad se od toga oduzme $b = 1$, dobija se rezultat $r = 1$. Na sličan način mogu se analizirati i sve ostale vrste u tabeli.

Operacija oduzimanja biće ilustrovana na sledećem primeru. Neka su dati binarni brojevi $A = 10110111_{(2)}$ i $B = 10011010_{(2)}$ koje treba oduzeti. Postupak oduzimanja može se predstaviti na sledeći način:

					$2_{(10)}$				
				0	0	$2_{(10)}$			
A		1	0	1	1	0	1	1	1
B	-	1	0	0	1	1	0	1	0
A–B		0	0	0	1	1	1	0	1

Brojevi *A* i *B* oduzimaju se tako što se najpre oduzmu cifre na poziciji 0. Kao rezultat, dobija se 1–0 = 1. Na sličan način se oduzimaju i vrednosti na pozicijama 1 i 2. Na poziciji 3 potrebno je naći razliku 0–1. Da bi se to uradilo, sa više pozicije se uzima pozajmica, tako da na njoj vrednost postaje 0. Pozajmica dolazi na poziciju 3 kao decimalna vrednost 2 i nakon oduzimanja, dobija se rezultat 1. Zatim se prelazi na oduzimanje na poziciji 4. Opet se pojavljuje isti slučaj da treba oduzeti 0–1. Stoga se sa naredne pozicije uzima pozajmica, pa na narednoj poziciji ostaje 0, a pozajmica na poziciji 4 postaje decimalna vrednost 2. Nakon oduzimanja, rezultat na poziciji 4 je 1. Postupak oduzimanja se nastavlja na pozicijama 5, 6 i 7 i dobija se konačni rezultat.

Množenje binarnih brojeva obavlja se po istim pravilima kao i množenje višecifrenih decimalnih brojeva, s tim što se prilikom sabiranja vodi računa da se radi u binarnom brojnom sistemu. Postupak množenja se može opisati sledećim koracima:

- svakom cifrom drugog činioca pomnožiti prvi činilac
- dobijene parcijalne proizvode napisati jedan ispod drugog, pomerene za jedno mesto ulevo
- sabrati sve parcijalne proizvode kao binarne brojeve

Dati postupak se može ispratiti na sledećem primeru:

```
1 1 0 0 · 1 1 0 1 =                 1 1 0 0
                        0 0 0 0
                    1 1 0 0
+       1 1 0 0
-----------------------------------------
    1 0 0 1 1 1 0 0
```

Deljenje binarnih brojeva vrši na sličan način kao deljenje decimalnih brojeva, s tim što se prilikom oduzimanja uzima u obzir da se radi u binarnom brojnom sistemu. Deljenje dva binarna broja obavlja se na sledeći način:

- grupu cifara deljenika (počevši sa leve strane) podeliti deliocem
- dobijeni rezultat pomnožiti deliocem, potpisati ispod grupe cifara u deljeniku i primeniti binarno oduzimanje
- spustiti sledeću cifru deljenika, a zatim ponavljati opisani postupak sve dok se ne dobije potpisani binarni broj koji je manji od delioca

Sledi primer deljenja dva binarna broja po opisanom postupku:

```
1 0 0 0 1 0 0 0 1 : 1 1 0 1 = 1 0 1 0 1
- 1 1 0 1 ↓ ↓ ↓ ↓
-----------
  0 1 0 0 0 0
    - 1 1 0 1
    -----------
      0 0 1 1 0 1
        - 1 1 0 1
        -----------
          0 0 0 0
```

1.1.2 Heksadecimalni brojni sistem

Da bi se ublažio problem sa dužinom zapisa binarnog broja, može se koristiti heksadecimalni sistem. Zapis broja u ovom sistemu zahteva manje cifara nego u decimalnom sistemu i znatno manje cifara nego u binarnoj predstavi, što je mnogo prihvatljivije za čoveka. Iako računar operiše nad binarnim brojevima, rezultati u binarnom obliku mogu se vrlo jednostavno prevesti u heksadecimalni oblik zahvaljajući pogodnom odnosu osnova ova dva sistema ($2^4 = 16$).

Heksadecimalni brojni sistem je sistem u kome se za zapisivanje brojeva koristi 16 heksadecimalnih cifara: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E i F. Pošto se u svakom brojnom sistemu za označavanje jedne cifre mora koristiti samo jedan simbol, to su u heksadecimalnom sistemu za predstavljanje brojeva od 10 do 15 usvojena početna slova abecede. U tabeli 1.4 date su decimalne, heksadecimalne i binarne vrednosti brojeva od 0 do 15.

Decimalna vrednost	Heksadecimalna vrednost	Binarna vrednost
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Tabela 1.4 Decimalna i binarna predstava heksadecimalnih cifara

S obzirom da je osnova heksadecimalnog brojnog sistema 16, jednačina kojom se predstavlja pozitivan ceo broj X u ovom sistemu ima oblik:

$$X = a_n 16^n + a_{n-1} 16^{n-1} + ... + a_2 16^2 + a_1 16^1 + a_0 16^0 \qquad (3)$$

Iz jednačine (3) sledi da se mesne vrednosti na susednim pozicijama u heksadecimalnom broju razlikuju 16 puta. To znači da cifra na poziciji 0 doprinosi vrednosti heksadecimalnog broja sa $16^0 = 1$, cifra na poziciji 1 sa $16^1 = 16$, cifra na poziciji 2 sa $16^2 = 256$, itd.

Heksadecimalno-decimalne konverzije brojeva

Postupci konvertovanja heksadecimalnog broja u decimalni i obrnuto vrlo su slični ranije opisanim postupcima konvertovanja binarnog broja u decimalni i obrnuto. Jedina razlika je u osnovi brojnog sistema koja je sada 16.

Konverzija heksadecimalnog broja u decimalni obavlja se primenom jednačine (3).

Neka je dat heksadecimalni broj $5E3_{(16)}$. Primenom jednačine (3), decimalna vrednost ovog broja računa se na sledeći način:

$X = 5 \cdot 16^2 + 14 \cdot 16^1 + 3 \cdot 16^0 = 5 \cdot 256 + 14 \cdot 16 + 3 \cdot 1 = 1507_{(10)}$

U poslednjem izrazu, pošto se radi sa decimalnim vrednostima, heksadecimalna cifra E predstavljena je svojom decimalnom vrednošću 14.

Konverzija decimalnog broja u heksadecimalni obavlja se u sledećim koracima:

1. zadati decimalni broj podeliti sa 16 sa ostatkom; ostatak zapisati, a rezultat deljenja ponovo podeliti sa 16 sa ostatkom; ovaj postupak ponavljati sve dok se kao rezultat deljenja ne dobije vrednost 0
2. heksadecimalni broj formirati od zapisanih ostataka u obrnutom redosledu od onoga u kome su nastajali; pre formiranja broja sve ostatke treba konvertovati u heksadecimalni oblik

Neka je dat decimalni broj $4328_{(10)}$. U skladu sa korakom 1 dobija se sledeće:

	rezultat deljenja	ostatak
4328 : 16 =	270	(8)
270 : 16 =	16	(14 = E)
16 : 16 =	1	(0)
1 : 16 =	0	(1)

Od dobijenih ostataka formira se heksadecimalni broj tako što se poslednji ostatak uzima kao cifra na najvišoj poziciji, a zatim se redom ispisuju heksadecimalne cifre sve dok se ne dođe do prvog ostatka koji predstavlja cifru na najnižoj poziciji u heksadecimalnom broju. Dakle, decimalni broj $4328_{(10)}$ se u heksadecimalnom obliku predstavlja sa $10E8_{(16)}$.

Konverzije brojeva između binarnog i heksadecimalnog sistema su znatno jednostavnije. Razlog za to leži u činjenici da osnova heksadecimalnog sistema odgovara četvrtom stepenu osnove binarnog sistema.

Heksadecimalno-binarne konverzije brojeva

Zahvaljujući pogodnom odnosu osnova binarnog i heksadecimalnog brojnog sistema, svaka heksadecimalna cifra može se predstaviti pomoću četiri binarne cifre. To znatno olakšava prevođenje brojeva iz jednog sistema u drugi, pa su postupci konverzije vrlo jednostavni.

Konverzija binarnog broja u heksadecimalni vrši se po sledećem postupku:

1. napraviti grupe od po četiri cifre u binarnom broju počevši sa desne strane, tj. od najniže pozicije

2. svaku grupu predstaviti jednom heksadecimalnom cifrom prema tabeli 1.4
3. heksadecimalni broj formirati od dobijenih heksadecimalnih cifara u redosledu u kome su odgovarajuće grupe raspoređene u binarnom broju

Neka je dat binarni broj $110111110_{(2)}$. On se deli u tri grupe od po četiri cifre na sledeći način:

$$1 \mid 1011 \mid 1110$$

Grupa 1110 odgovara heksadecimalnoj cifri E, grupa 1011 cifri B, a 1 cifri 1. Stoga se zadati binarni broj u heksadecimalnom obliku predstavlja kao $1BE_{(16)}$.

Konverzija heksadecimalnog broja u binarni obavlja se u sledećim koracima:

1. svaku cifru heksadecimalnog broja predstaviti grupom od četiri binarne cifre prema tabeli 1.4
2. binarni broj formirati spajanjem grupa u redosledu u kome se nalaze

Neka je dat heksadecimalni broj $3A9_{(16)}$. Cifre u ovom broju se u binarnom obliku zapisuju sa:

$3_{(16)} = 3_{(10)} = 0011_{(2)}$

$A_{(16)} = 10_{(10)} = 1010_{(2)}$

$9_{(16)} = 9_{(10)} = 1001_{(2)}$

Binarni broj se formira jednostavnim spajanjem grupa, pa je:

$3A9_{(16)} = 0011\ 1010\ 1001_{(2)} = 1110101001_{(2)}$

1.2. Predstavljanje podataka u računaru

Širok spektar problema koji se mogu rešiti pomoću računara zahteva korišćenje raznovrsnih tipova podataka za opisivanje raznih objekata, njihovih aktivnosti i međusobnih veza. Preglednosti radi, podaci su po tipu klasifikovani u dve osnovne grupe: statički tipovi i dinamički tipovi. Pod statičkim tipovima podrazumevaju se tipovi podataka kod kojih je unapred definisana unutrašnja struktura svakog podatka. Za razliku od njih, kod dinamičkih tipova struktura podataka se slobodno menja tokom rada.

Statički tipovi podataka obuhvataju skalarne i strukturirane podatke. Pod skalarnim tipovima podrazumevaju se najprostiji tipovi podataka čije su vrednosti skalari, tj. veličine koje predstavljaju elementarne celine i nema potrebe da se dalje razlažu na komponente. Podatak je strukturiran ako se sastoji od više komponenata koje se nalaze u precizno definisanom odnosu (na primer, matrice).

U ovom poglavlju detaljno će biti razmotrena tri skalarna tipa podataka koja se najčešće koriste: celobrojni (*integer*), realni (*real*) i znakovni tip podataka (*character*).

1.2.1 Predstavljanje celih brojeva

U dosadašnjem izlaganju razmatrano je samo predstavljanje pozitivnih celih brojeva u binarnom obliku. Međutim, za rešavanje mnogih problema zahteva se korišćenje označenih, tj. pozitivnih i negativnih, celih brojeva.

Za označavanje brojeva u dekadnom brojnom sistemu koriste se oznake „+“ (za predstavljanje pozitivnih brojeva, ili se ovaj znak izostavlja) i „–“ (za predstavljanje negativnih brojeva). Ove oznake se pišu ispred zapisa koji određuje apsolutnu vrednost decimalnog broja. U binarnom brojnom sistemu, ovakav način predstavljanja označenih brojeva nije moguć zato što je u njemu dozvoljeno korišćenje samo dva simbola, 0 i 1 (računar, takođe, može da prepozna samo ova dva znaka). Problem označavanja u binarnom brojnom sistemu najjednostavnije se može rešiti tako što se ispred apsolutne vrednosti broja doda jedna cifra koja predstavlja znak. Može se usvojiti, na primer, da 0 označava da je broj pozitivan, a 1 da je negativan. Ovakav način zapisivanja binarnih brojeva naziva se predstavljanje binarnih brojeva pomoću znaka i apsolutne vrednosti.

Neoznačeni broj $7_{(10)} = 111_{(2)}$ može se označiti pomoću znaka i apsolutne vrednosti na sledeći način:

$+ 7_{(10)} = 0111_{(2)}$ označen pozitivan binarni broj

$- 7_{(10)} = 1111_{(2)}$ označen negativan binarni broj

Iako je navedeni način označavanja vrlo jednostavan u pogledu formiranja zapisa broja, on nema značajniju primenu. Glavni razlog za to je što se nad binarnim brojevima zapisanim pomoću znaka i apsolutne vrednosti teško obavljaju aritmetičke operacije. Naime, ovako označeni brojevi se ne mogu posmatrati kao celina, već se uvek mora posebno voditi računa o cifri na mestu najveće težine koja predstavlja znak, a posebno o vrednosti broja. Mnogo efikasniji način označavanja binarnih brojeva je pomoću komplementa dvojke.

Komplement dvojke

Komplement dvojke ili puni komplement je postupak binarnog predstavljanja označenih celih brojeva u računaru. Po ovom postupku, označeni brojevi se predstavljaju pomoću *n*-cifarske binarne reči čiji najstariji razred *S* označava predznak broja. Kao i u slučaju zapisa pomoću znaka i apsolutne vrednosti, zapis u komplementu dvojke, takođe, počinje cifrom $S = 0$ ukoliko je broj nenegativan (pozitivan ili 0), a cifrom $S = 1$ ako je broj negativan. Broj 0 se zapisuje nulama na svim pozicijama.

Neka je *Z* zadati ceo broj, a *K* binarni zapis broja *Z*. Pozitivan ceo broj $+|Z|$ i negativan ceo broj $-|Z|$ mogu se predstaviti u binarnom obliku na sledeći način:

$$+|Z| : \quad K^+ = (|Z|)_{(2)}$$

$$-|Z| : \quad K^- = (2^n - |Z|)_{(2)} = (2^n - K^+)_{(2)}$$

gde je $|Z|$ apsolutna vrednost broja *Z*. Veličina K^- se naziva *punim komplementom* veličine K^+, zato što važi $K^+ + K^- = 2^n$. Takođe, može se reći i obrnuto, tj. da je veličina K^+ puni komplement veličine K^-.

Na primer, brojevi +15 i –15 se pomoću 16-cifarske binarne reči ($n = 16$) predstavljaju kao:

$+15$: $K^+ = (|15|)_{(2)} =$ 0000 0000 0000 1111

-15: $K^- = (2^{16} - |15|)_{(2)} =$ 1 0000 0000 0000 0000

– 0000 0000 0000 1111

1111 1111 1111 0001

Iz primera se vidi da se efekat oduzimanja broja od 2^n može postići i invertovanjem tog broja (jedinice se zamene nulama, a nule jedinicama) uz dodavanje vrednosti 1. Tako važi:

polazni broj	0000 0000 0000 1111
invertovani broj	1111 1111 1111 0000
sabiranje sa 1	1111 1111 1111 0000 + 1 = 1111 1111 1111 0001

Na osnovu navedenog, može se zaključiti sledeće:

- *pozitivan broj* u komplementu dvojke dobija se *dodavanjem cifre 0* ispred binarnog zapisa koji odgovara apsolutnoj vrednosti tog broja
- *negativan broj* u komplementu dvojke dobija se tako što se

- ispred binarnog zapisa apsolutne vrednosti broja *dopiše 0*
- zatim se sve binarne cifre *invertuju*
- dobijeni broj se *sabere sa* 1

Slede primeri nalaženja komplementa dvojke jednog pozitivnog ($+7_{(10)}$) i jednog negativnog ($-10_{(10)}$) celog broja:

+7:	0000 0000 0000 0111	(dodavanje nule, puni komplement)
–10:	0000 0000 0000 1010	(dodavanje nule)
	1111 1111 1111 0101	(invertovanje)
	+1	(sabiranje sa 1)
	1111 1111 1111 0110	(puni komplement)

Komplement dvojke ima osobinu da ukoliko se dva puta uzastopno primeni nad nekim brojem, kao rezultat se dobija polazni broj. Ovo će biti pokazano na primeru broja $13_{(10)}$:

polazni broj (+13):	0000 0000 0000 1101
invertovanje:	1111 1111 1111 0010
sabiranje sa 1:	+1
puni komplement (–13):	1111 1111 1111 0011
polazni broj (–13):	1111 1111 1111 0011
invertovanje:	0000 0000 0000 1100
sabiranje sa 1:	+1
puni komplement (+13):	0000 0000 0000 1101

Vrednost pozitivnog broja u komplementu dvojke se ne menja ako se ispred cifre najveće težine doda proizvoljan broj nula (vodeće nule). Na primer, važi $0111_{(2)} = 00000000111_{(2)}$. Slično, vrednost negativnog broja u komplementu dvojke ostaje ista ako se ispred broja doda proizvoljan broj jedinica (vodeće jedinice). Na primer, $1001_{(2)} = 1111111001_{(2)}$.

Iako je postupak nalaženja komplementa dvojke jednostavan, može se dalje pojednostaviti. To se postiže tako što se:

- polazni binarni broj *podeli na dva dela*, levi i desni; desni deo čine *prva jedinica sa desne strane* u broju i sve nule koje slede iza nje (broj nula može biti i 0); preostale cifre čine levi deo broja
- komplement dvojke se formira tako što se *invertuje levi deo* broja, dok *desni deo* broja ostaje *nepromenjen*

Kada se ovaj postupak primeni za nalaženje komplementa dvojke binarnog broja $01010010010000_{(2)}$ dobija se:

polazni broj	=	010100100	10000
		levi deo	desni deo
komplement dvojke	=	10101101110000	

Ovim pojednostavljenim postupkom izbegnuto je bilo kakvo računanje komplementa dvojke, već se za svaki binarni broj može direktno napisati njegov puni komplement.

Pretpostavimo da je poznat n-cifarski binarni broj $a_{n-1}.....a_1a_0$ koji predstavlja zapis nekog označenog celog broja u komplementu dvojke. Decimalna vrednost ovog broja X računa se pomoću formule:

$$X = -a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + ... + a_1 2^1 + a_0 2^0 \qquad (4)$$

Primenom formule (4) na brojeve zapisane u komplementu dvojke iz ranijih primera, $0111_{(2)}$ i $10110_{(2)}$, dobijaju se njihove decimalne vrednosti $X1$ i $X2$, respektivno:

$$X1 = -0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 4 + 2 + 1 = +7_{(10)}$$

$$X2 = -1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = -16 + 4 + 2 = -10_{(10)}$$

Opseg neoznačenih brojeva

Pre određivanja opsega brojeva koji se mogu predstaviti u komplementu dvojke, biće analiziran jednostavniji slučaj nalaženja opsega neoznačenih brojeva.

Pretpostavimo da se zapis neoznačenog binarnog broja sastoji od n cifara. Pošto se na mestu svake cifre u broju mogu pojaviti dve vrednosti (0 ili 1), broj različitih kombinacija (brojeva) koje se mogu generisati sa n cifara je 2^n. Pošto je uobičajeno da skup brojeva sadrži nulu, jedna od kombinacija se mora upotrebiti za predstavljanje broja 0. Preostalih 2^n-1 kombinacija služe za predstavljanje brojeva

od 1 do 2^n–1. Odavde sledi da se opseg neoznačenih binarnih brojeva zapisanih sa *n* cifara može dobiti po sledećoj formuli:

$0 \leq x \leq 2^n–1$ odnosno $x \in \{0, 1,\ldots, 2^n\text{-}1\}$

gde je *x* neoznačeni binarni broj.

Primenom ove formule dobijaju se opsezi neoznačenih brojeva koji se predstavljaju pomoću četiri, odnosno osam binarnih cifara na sledeći način:

$n = 4$: $0 \leq x \leq 2^4\text{-}1$ $0 \leq x \leq 15$

$n = 8$: $0 \leq x \leq 2^8\text{-}1$ $0 \leq x \leq 255$

Pomoću četiri binarne cifre mogu se zapisati neoznačeni brojevi iz opsega $x \in \{0, 1,\ldots, 15\}$, a pomoću osam cifara brojevi iz opsega $x \in \{0, 1,\ldots, 255\}$.

Opseg brojeva predstavljenih u komplementu dvojke

Za razliku od neoznačenih, označeni brojevi obuhvataju pozitivne, negativne brojeve i nulu. Pretpostavimo da se zapis označenog binarnog broja predstavljenog u komplementu dvojke sastoji od *n* binarnih cifara. Kao što je ranije rečeno, broj različitih kombinacija koje se mogu generisati sa ovim brojem cifara je 2^n. Pošto se jedna od kombinacija mora upotrebiti za predstavljanje broja 0, preostalih 2^n–1 kombinacija služe za predstavljanje pozitivnih i negativnih brojeva. S obzirom da se pozitivni i negativni brojevi ravnopravno koriste, može se uzeti da polovina kombinacija služi za predstavljanje pozitivnih, a druga polovina za predstavljanje negativnih brojeva. Međutim, broj 2^n–1 je neparan, pa se broj pozitivnih i broj negativnih brojeva koji se mogu predstaviti u komplementu dvojke moraju razlikovati za 1. Obično se uzima da je

broj pozitivnih brojeva = $(2^n–1–1)/2 = 2^n/2–2/2 = 2^{n-1}–1$

broj negativnih brojeva = $1+(2^n–1–1)/2 = 2^{n-1}$

Na osnovu navedenog, opseg označenih binarnih brojeva zapisanih u komplementu dvojke sa *n* cifara računa se po sledećim formulama:

$x \leq 2^{n-1}–1$ za $x \geq 0$

$|x| \leq 2^{n-1}$ za $x < 0$ odnosno $x \in \{–2^{n-1}, ..., –1, 0, 1,\ldots, 2^{n-1}–1\}$

gde je x označeni binarni broj u komplementu dvojke.

Ako se ove formule primene za $n = 4$ i $n = 8$, mogu se naći opsezi četvorocifrenih i osmocifrenih binarnih brojeva predstavljenih u komplementu dvojke:

$n = 4$: $\quad -2^3 \leq x \leq 2^3\text{-}1 \quad -8 \leq x \leq 7$

$n = 8$: $\quad -2^7 \leq x \leq 2^7\text{-}1 \quad -128 \leq x \leq 127$

Dakle, pomoću četiri binarne cifre mogu se zapisati označeni brojevi u komplementu dvojke koji pripadaju skupu $x \in \{-8, ..., -1, 0, 1, \ldots, 7\}$, a pomoću osam cifara brojevi iz skupa $x \in \{-128, ..., -1, 0, 1, \ldots, 127\}$.

Aritmetičke operacije nad brojevima u komplementu dvojke

Nad celim binarnim brojevima zapisanim u komplementu dvojke mogu se efikasno obavljati osnovne aritmetičke operacije sabiranja i oduzimanja. Prilikom izvršavanja ovih operacija, brojevi se posmatraju kao jedinstvene celine, tj. znak se smatra sastavnim delom broja i nad njim se obavljaju operacije na isti način kao i nad svim ostalim ciframa broja.

Oduzimanje označenih binarnih brojeva A i B predstavljenih u komplementu dvojke, tj. $A - B$, može se realizovati pomoću operacije sabiranja. To se postiže tako što se umanjenik A uzima kao prvi sabirak, a umanjilac B sa negativnim predznakom kao drugi sabirak u zbiru:

$$A - B = A + (-B)$$

U nastavku će biti razmotrena samo operacija aritmetičkog sabiranja brojeva u punom komplementu, zato što se operacija oduzimanja, primenom prethodne formule, lako može svesti na operaciju sabiranja.

U računaru se operacija sabiranja (oduzimanja) kontroliše pomoću *dva indikatora prenosa*, C i P (*carry bits*) i *indikatora prekoračenja* V (*overflow bit*).

Indikator prenosa C predstavlja prenos iz *najstarijeg razreda* prilikom sabiranja. Ukoliko je $C = 1$, to ukazuje da je rezultat operacije duži od raspoloživog prostora, što ne mora da znači da je dobijeni rezultat pogrešan. Indikator P je prenos iz *razreda do najstarijeg* (prvi susedni razred). Može se pokazati da je rezultat operacije ispravan ukoliko su prenosi iz dva najstarija razreda, C i P, isti (bilo obe nule, ili obe jedinice). Glavnu ulogu u utvrđivanju ispravnosti rezultata ima indikator prekoračenja V koji se računa na sledeći način:

ako je $C = P$, onda je $V = 0$

ako je $C \neq P$, onda je $V = 1$

Rezultat operacije je ispravan ako je $V = 0$, a neispravan ako je $V = 1$.

Sabiranje dva cela broja u računaru se obavlja na sledeći način:

a) oba sabirka se predstave u komplementu dvojke
b) dobijeni puni komplementi se saberu po pravilima binarnog sabiranja ne vodeći računa o znaku; prilikom sabiranja pamte se prenosi između razreda
c) na osnovu zapamćenih prenosa, odrede se vrednosti indikatora *C* i *P*
d) na osnovu *C* i *P* izračuna se indikator *V* koji pokazuje da li je dobijeni rezultat ispravan ili ne

Ovaj postupak biće ilustrovan kroz nekoliko primera za slučaj četvorocifrenih brojeva ($n = 4$).

Primer 1: Sabrati brojeve $A = +3$ i $B = +4$.

puni komplement *A*: 0011

puni komplement *B*: 0100

zbir: $0111 = -0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = +7$ (tačan rezultat)

prenosi: 0000

$C = 0$, $P = 0$, $V = 0$, ispravan rezultat

Primer 2: Sabrati brojeve $A = +3$ i $B = -4$.

puni komplement *A*: 0011

puni komplement *B*: 1100

zbir: $1111 = -1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = -1$ (tačan rezultat)

prenosi: 0000

$C = 0$, $P = 0$, $V = 0$, ispravan rezultat

Primer 3: Sabrati brojeve $A = +7$ i $B = +1$.

puni komplement *A*: 0111

puni komplement *B*: 0001

zbir: $1000 = -1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = -8$ (netačan rezultat)

prenosi: 0111

$C = 0$, $P = 1$, $V = 1$, neispravan rezultat (rezultat van opsega)

Primer 4: Sabrati brojeve $A = +7$ i $B = -7$.

puni komplement A: 0111

puni komplement B: 1001

zbir: $0000 = -0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 0$ (tačan rezultat)

prenosi: 1111

$C = 1$, $P = 1$, $V = 0$, ispravan rezultat

Primer 5: Sabrati brojeve $A = -7$ i $B = -4$.

puni komplement A: 1001

puni komplement B: 1100

zbir: $0101 = -0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = +5$ (netačan rezultat)

prenosi: 1000

$C = 1$, $P = 0$, $V = 1$, neispravan rezultat (rezultat van opsega)

Da bi rezultat sabiranja binarnih brojeva zapisanih u komplementu dvojke bio ispravan, treba da pripada dozvoljenom opsegu brojeva koji se mogu predstaviti datim brojem cifara. Ukoliko to nije slučaj, dolazi do prekoračenja i greške prilikom izračunavanja. Prekoračenje se javlja, na primer, pri sabiranju brojeva $+6_{(10)}$ i $+7_{(10)}$:

puni komplement $+6_{(10)}$: 0110

puni komplement $+7_{(10)}$: 0111

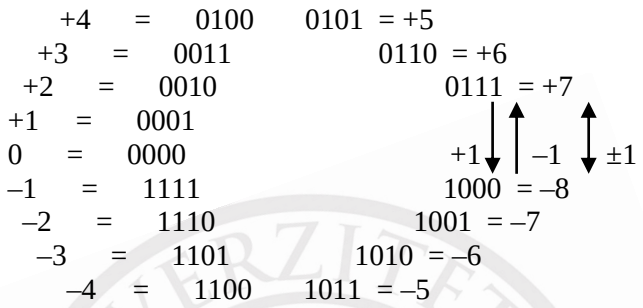
zbir: $1101 = -1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = -3$

prenosi: 0110

$C = 0$, $P = 1$, $V = 1$, neispravan rezultat

Kao što se vidi iz primera, nakon sabiranja dobijen je netačan rezultat $-3_{(10)}$, umesto ispravnog $+13_{(10)}$. Do greške je došlo zato što se u komplementu dvojke pomoću četiri binarne cifre mogu predstaviti samo brojevi iz skupa {-8,...,0,…,7}, a $+13_{(10)}$ ne pripada ovom skupu.

Može se pokazati da između celih brojeva predstavljenih u komplementu dvojke postoji kružna povezanost koja nastaje zbog toga što se najveća i najmanja vrednost iz dozvoljenog opsega razlikuju za 1. Tako, za $n = 4$, kružna povezanost se može prikazati kao na slici 1.1.



Slika 1.1 Kružna povezanost brojeva predstavljenih u komplementu dvojke

Kao što se vidi, kada se +7 (0111) sabere sa 1 dobija se –8 (1000), odnosno kada se od –8 oduzme 1 dobija se +7. Zbog kružne povezanosti, ukoliko se desi prekoračenje opsega, ono se propagira po krugu. Tako, u primeru kojim je ilustrovano prekoračenje, dobijeni rezultat $+13_{(10)}$ može se propagirati po krugu za 6 pozicija od broja +7 (+7 je najveći broj iz opsega, a ispravan rezultat je od njega veći za 6), čime se stiže upravo do broja –3, što je bio rezultat sabiranja.

1.2.2 Predstavljanje realnih brojeva

Rešavanje mnogih problema zahteva korišćenje realnih brojeva. Realni brojevi se u računaru obično predstavljaju u tzv. pokretnom zarezu (*floating point*).

Pokretni zarez

Pokretni zarez predstavlja način zapisivanja realnih brojeva pomoću tri komponente: *znaka Z, stvarnog eksponenta Es* i *mantise M*. Znak određuje da li je broj pozitivan ili negativan, mantisa sadrži značajne cifre broja, a stvarni eksponent služi za skaliranje. Decimalna vrednost *V* ovako zapisanog realnog broja računa se po formuli:

$$V = (Z)\ M \cdot q^{Es}$$

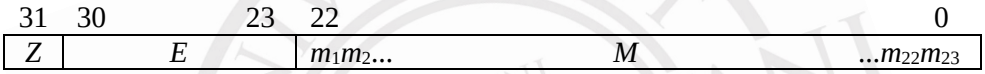
gde je q osnova brojnog sistema.

Postoje različiti standardi za predstavljanje brojeva u pokretnom zarezu. Broj cifara koje se koriste za predstavljanje navedenih komponenata, kao i formati u

kojima su one zapisane razlikuju se u zavisnosti od primenjenog standarda. U računaru se najčešće koristi binarna reprezentacija realnih brojeva zapisanih u formatu pokretnog zareza standardizovana od strane organizacije *IEEE* (*Institute of Electrical and Electronics Engineers*). Standard čiji je zvaničan naziv *IEEE* 754 definisan je u dva formata: sa jednostrukom ili običnom preciznošću i sa dvostrukom preciznošću. U nastavku će biti predstavljen jednostruki format *IEEE* 754 standarda.

Po standardu *IEEE* 754, realni broj u pokretnom zarezu zapisuje se u obliku 32-bitne binarne reči prikazane na slici 1.2 u kojoj je:

- *bit na mestu najveće težine* rezervisan za *znak*
- sledećih 8 *bitova* (1 bajt) namenjeno predstavljanju *eksponenta*
- *preostala* 23 *bita* namenjena predstavljanju *mantise*



Slika 1.2 Zapis u pokretnom zarezu po standardu IEEE 754 (jednostruki format)

Zapisani broj je pozitivan ako binarna cifra koja predstavlja znak (Z) ima vrednost 0, a negativan ako ova cifra ima vrednost 1.

Pomoću osam bitova predviđenih za predstavljanje eksponenta E mogu se zapisati decimalni brojevi iz opsega od 0 do 255. Međutim, da bi u pokretnom zarezu mogli da se predstavljaju i mali i veliki realni brojevi, potrebno je obezbediti da stvarni eksponent Es može da ima i pozitivne i negativne vrednosti. To je postignuto transliranjem (pomeranjem) navedenog opsega za 127. Oduzimanjem broja 127 od granica opsega, postignuto je da vrednost stvarnog eksponenta bude u opsegu od –127 (0–127 = –127) do 128 (255–127 = 128). Imajući ovo u vidu, prilikom generisanja zapisa broja, stvarni eksponent treba najpre uvećati za 127 (da bi se dobila neka vrednost iz opsega od 0 do 255), a zatim dobijenu vrednost u binarnom obliku uključiti u zapis. Ovako definisan eksponent naziva se *uvećanim eksponentom* (E). Iz ovoga sledi da se prilikom određivanja decimalne vrednosti stvarnog eksponenta mora od decimalne vrednosti 8-cifarskog eksponenta E u zapisu oduzeti broj 127.

Brojevi se često normalizuju tako što se decimalna tačka postavlja desno od prve nenulte cifre u broju. Na primer, brojevi $5.138 \cdot 10^{38}$ i $3.98 \cdot 10^{-24}$ jesu normalizovani, dok brojevi $513.8 \cdot 10^{36}$ i $39.8 \cdot 10^{-25}$ nisu normalizovani.

Neka su m_1, m_2, ..., m_{23} cifre mantise u redosledu prikazanom na slici. Po razmatranom standardu, mantisa je normalizovana, što znači da je njen bit najveće težine uvek jednak 1, pa stvarna mantisa ima 24-bitni oblik $1.m_1m_{2...}m_{23}$. Kao što se

vidi, bit najveće težine je izostavljen iz zapisa, jer je unapred poznat. Decimalna vrednost mantise određuje se po formuli:

$$M = 2^0 + m_1 \cdot 2^{-1} + m_2 \cdot 2^{-2} + ... + m_{22} \cdot 2^{-22} + m_{23} \cdot 2^{-23} \quad (5)$$

Pošto je $2^0 = 1$, a zbir ostalih članova u formuli manji od 1 (što se može matematički dokazati), sledi da vrednost mantise mora biti između 1 i 2.

Po standardu *IEEE* 754, decimalni broj 0 može se zapisati na dva načina:

- pomoću 32 nule
- pomoću jedinice i 31 nule

U nastavku je opisano kako se na osnovu zadatog zapisa može pronaći realni decimalni broj kome taj zapis odgovara, i obrnuto, kako se zadati decimalni broj može predstaviti u pokretnom zarezu.

Postupak nalaženja decimalne vrednosti binarnog broja zapisanog u pokretnom zarezu po standardu *IEEE* 754 odvija se na sledeći način:

1) *određivanje znaka Z*: ako je *bit najveće težine* u zapisu 0, broj je pozitivan (+), a ako je 1, broj je negativan (–)
2) *nalaženje eksponenta Es*: 8-bitnu binarnu vrednost eksponenta (koja sledi posle bita najveće težine) *konvertovati u decimalnu vrednost*, a zatim od dobijenog decimalnog broja *oduzeti broj* 127
3) *određivanje mantise M*: primeniti *formulu* (5) za računanje mantise na poslednje 23 cifre u zapisu
4) *računanje decimalne vrednosti broja V*: primeniti *formulu* $V = (Z)\ M \cdot 2^{Es}$

Opisani postupak biće ilustrovan na jednom primeru. Neka je dat zapis broja u pokretnom zarezu $01000000111100000000000000000000_{(2)}$. Da bi se odredila decimalna vrednost ovog broja, najpre u zapisu treba uočiti sve njegove komponente.

0	10000001	11100000000000000000000
Z	*E*	*M*

Zatim, za uočene komponente, treba primeniti prethodno navedene korake:

1) određivanje znaka *Z*: bit najveće težine je 0, pa je broj pozitivan (+)
2) nalaženje eksponenta *Es*: binarna vrednost eksponenta u zapisu je 10000001, što odgovara decimalnoj vrednosti $E = 1 \cdot 2^7 + 1 \cdot 2^0 = 129$; pošto je eksponent u zapisu uvećan, njegova stvarna vrednost predstavlja razliku $Es = 129 - 127 = 2$

3) određivanje mantise *M*: prema formuli za računanje mantise dobija se da je

$$M = 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} + 1\cdot 2^{-3} = 1 + 1/2 + 1/4 + 1/8 = 15/8$$

4) računanje decimalne vrednosti broja *V*:

$$V = (Z)\, M\cdot 2^{Es} = + (15/8)\cdot 2^2 = +7.5_{(10)}$$

Po standardu *IEEE* 754, postupak konvertovanja decimalnih realnih brojeva u zapis u pokretnom zarezu obavlja se na sledeći način:

1) celobrojni deo realnog broja konvertuje se u binarni zapis po ranije opisanom standardnom postupku konverzije celih brojeva iz decimalnog u binarni brojni sistem
2) razlomljeni deo decimalnog broja konvertuje se u binarni zapis tako što se
   - pomnoži osnovom brojnog sistema 2
   - od dobijenog rezultata, ceo deo se proglasi rezultujućom cifrom
   - razlomljeni deo dobijenog rezultata se pomnoži osnovom brojnog sistema 2 i postupak se ponavlja kroz određen broj ciklusa
3) od binarnih zapisa dobijenih u koracima 1) i 2) formira se binarna vrednost broja tako što se najpre navede rezultat koraka 1), zatim tačka i nakon toga rezultat koraka 2)
4) dobijeni binarni broj se normalizuje
5) na osnovu normalizovanog oblika odrede se komponente *Z*, *Es* i *M* i formira zapis u pokretnom zarezu

Ovaj postupak biće ilustrovan na primeru nalaženja zapisa u pokretnom zarezu koji odgovara realnom broju iz prethodnog primera $R = 7.5$. Ceo deo ovog broja je 7. On se u binarnom obliku predstavlja sa $111_{(2)}$. Razlomljeni deo broja *R* je 0.5. Binarni zapis ovog dela broja se dobija na sledeći način:

	rezultat množenja	ceo deo
$0.5 \cdot 2 =$	1.0	(1)
$0.0 \cdot 2 =$	0.0	(0)

binarni zapis razlomljenog dela = $10000.....{}_{(2)}$

Prema koraku 3) formira se binarna vrednost polaznog broja:

$$7.5_{(10)} = 111.100000000...{}_{(2)}$$

Normalizacijom desne strane prethodnog izraza dobija se da se polazni broj može napisati i u obliku:

$$7.5_{(10)} = 1.11100000000...{}_{(2)} \cdot 2^2$$

Iz normalizovanog oblika može se zaključiti da je:

- znak broja pozitivan (+)
- vrednost uvećanog eksponenta $Es = 2 + 127 = 129_{(10)} = 10000001_{(2)}$
- vrednost mantise $M = 11100000000000000000000_{(2)}$

Tako je dobijeno da se broj 7.5 se u pokretnom zarezu po standardu *IEEE* 754 (jednostruki format) predstavlja u vidu binarnog niza:

$$0\ 10000001\ 11100000000000000000000_{(2)}$$

što je polazni broj u prethodnom primeru.

Sledi još jedan primer nalaženja zapisa u pokretnom zarezu koji odgovara realnom broju $R = 123.45$. Ceo deo ovog broja je 123. On se konvertuje u binarni oblik na sledeći način:

	rezultat deljenja	ostatak	
123 : 2 =	61	(1)	(LSB)
61 : 2 =	30	(1)	
30 : 2 =	15	(0)	
15 : 2 =	7	(1)	
7 : 2 =	3	(1)	
3 : 2 =	1	(1)	
1 : 2 =	0	(1)	(MSB)

$$123_{(10)} = 1111011_{(2)}$$

Razlomljeni deo polaznog broja je 0.45. Binarni zapis ovog dela dobija se na način opisan u koraku 2:

	rezultat množenja	ceo deo
$0.45 \cdot 2 =$	0.9	(0)
$0.9 \cdot 2 =$	1.8	(1)
$0.8 \cdot 2 =$	1.6	(1)
$0.6 \cdot 2 =$	1.2	(1)
$0.2 \cdot 2 =$	0.4	(0)
$0.4 \cdot 2 =$	0.8	(0)

binarni zapis razlomljenog dela = $011100.....{}_{(2)}$

Pošto je poslednji rezultat množenja 0.8, u narednim ciklusima će se vrednosti $1100_{(2)}$ ponavljati (razlomljeni deo 0.8 se već pojavio u trećoj vrsti). Tako se dobija da je tačna vrednost polaznog broja:

$123.45_{(10)} = 1111011.011100110011001100110011001100110011001100110011001100...{}_{(2)}$

Ovaj primer ilustruje tipičnu situaciju u kojoj se konačan racionalni decimalni broj konvertuje u beskonačno periodičan racionalni binarni broj. Dobijeni binarni broj se može napisati u normalizovanom obliku kao:

$$123.45_{(10)} = 1.11101101110011001100110011001100110011001100110011001100...{}_{(2)} \cdot 2^6$$

Na osnovu poslednjeg izraza, formira se zapis u pokretnom zarezu na sledeći način:

- znak broja je pozitivan
- uvećani eksponent se računa kao $Es = 6 + 127 = 133_{(10)} = 10000101_{(2)}$
- mantisa je $M = 11101101110011001100110_{(2)}$

Realan broj 123.45 se u pokretnom zarezu po standardu IEEE 754 (jednostruki format) predstavlja u obliku:

$$0\ 10000101\ 11101101110011001100110_{(2)}.$$

1.2.3 Predstavljanje podataka znakovnog tipa

Prilikom komunikacije korisnika sa računarom neophodno je obezbediti da se podaci pojavljuju u obliku koji je čitljiv za čoveka. Stoga podaci treba da budu predstavljeni u vidu znakova iz određenog skupa koji obično obuhvata velika i mala slova abecede, decimalne cifre, specijalne znakove i kontrolne znakove. Pod *specijalnim znacima* podrazumevaju se svi oni znaci koji postoje na tastaturi, a nisu ni cifre ni slova. Oni se mogu odštampati. Specijalni znaci su, na primer, !, #, $, %, *, (,), =, +, -, ? i drugi. *Kontrolni znaci* su znaci koji se ne mogu odštampati niti prikazati na ekranu računara, već služe za upravljanje ulazno/izlaznim uređajima. Primer kontrolnih znakova su zvučni signal, znaci za pomeranje papira štampača i slično.

U praksi se koristi više metoda za binarno predstavljanje podataka znakovnog tipa. Najpoznatiji od njih je standard *ASCII* (*American Standard Code for Information Interchange*). Po ovom standardu, znakovi se predstavljaju u obliku 8-cifarskog binarnog broja. Osnovni skup *ASCII* znakova, prikazan u tabeli 1.5, daje jednoznačnu vezu između znakova (kolona *ASCII*) i njihovih decimalnih (kolona *Dec*), odnosno heksadecimalnih (kolona *Hex*) kodova.

Dec	*Hex*	*ASCII*	*Dec*	*Hex*	*ASCII*	*Dec*	*Hex*	*ASCII*	*Dec*	*Hex*	*ASCII*
0	00	NUL	32	20		64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	“	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	’	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	TAB	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Tabela 1.5 Osnovni skup *ASCII* znakova

Vežbanja

P1. Šta je pozicioni brojni sistem? Šta je osnova brojnog sistema? Na koji način se bilo koji pozitivan ceo broj može predstaviti u pozicionom brojnom sistemu sa osnovom *q*? Objasniti formulu. Ako je u formuli q=10, a_0=2, a_1=5, a_2=9 i a_3=1, koji broj je zapisan? Kolika je mesna vrednost cifre na poziciji 4 u nekom binarnom broju?

P2. Opisati postupak određivanja komplementa dvojke pozitivnog i negativnog broja. Dati primer skraćenog postupka određivanja komplementa dvojke binarnog broja. Navesti formulu za određivanje decimalne vrednosti broja zapisanog u komplementu dvojke i objasniti je. Koja cifra se može dopisivati ispred negativnog broja predstavljenog u komplementu dvojke, a da se vrednost tog broja ne promeni? Kako se određuje indikator prekoračenja prilikom sabiranja brojeva u komplementu dvojke?

P3. Detaljno objasniti jednostruki standard *IEEE* 754 za predstavljanje realnih brojeva u pokretnom zarezu. Zašto je i za koliko eksponent u zapisu uvećan?

Z1. Naći decimalne i heksadecimalne vrednosti sledećih binarnih brojeva:

a) $100111_{(2)}$
b) $11010_{(2)}$
c) $1000101_{(2)}$
d) $1010001_{(2)}$
e) $101001110_{(2)}$

Z2. Predstaviti u binarnom i heksadecimalnom obliku zadate decimalne brojeve:

a) $56_{(10)}$
b) $88_{(10)}$
c) $124_{(10)}$
d) $231_{(10)}$
e) $426_{(10)}$

Z3. Konvertovati date heksadecimalne brojeve u binarne i decimalne brojeve:

a) $58_{(16)}$
b) $40C_{(16)}$
c) $21E_{(16)}$
d) $1F1_{(16)}$
e) $426_{(16)}$

Z4. Data su dva neoznačena binarna broja A i B. Izračunati: $A + B$ i $A - B$.

a) $A = 1011010$ $B = 11011$
b) $A = 1011100$ $B = 110101$
c) $A = 1001011$ $B = 101111$
d) $A = 10011111$ $B = 101011$
e) $A = 101001110$ $B = 11100010$

Z5. Data su dva neoznačena binarna broja A i B. Izračunati: $A \cdot B$ i $A:B$.

a) $A = 100111$ $B = 1101$
b) $A = 11001$ $B = 101$
c) $A = 101010$ $B = 110$

Z6. Predstaviti u komplementu dvojke sledeće brojeve:

a) $75_{(10)}$
b) $355_{(10)}$
c) $-33_{(10)}$
d) $-82_{(10)}$
e) $-100_{(10)}$

Z7. Izračunati decimalnu vrednost sledećih binarnih brojeva zapisanih u komplementu dvojke:

a) $100101010_{(2)}$
b) $0110110_{(2)}$
c) $11010010_{(2)}$
d) $1111101_{(2)}$
e) $00011111_{(2)}$

Z8. Navesti formulu za određivanje mogućeg opsega neoznačenih binarnih brojeva zapisanih sa n cifara. Primenom ove formule odrediti sa koliko se najmanje cifara mogu zapisati sledeći neoznačeni brojevi:

a) $47_{(10)}$
b) $82_{(10)}$
c) $100_{(10)}$

Z9. Na osnovu formule za određivanje dozvoljenog opsega označenih binarnih brojeva zapisanih u komplementu dvojke sa n cifara odrediti sa koliko se najmanje cifara mogu zapisati sledeći označeni brojevi:

a) $71_{(10)}$
b) $-121_{(10)}$
c) $-99_{(10)}$

Z10. Navedene decimalne brojeve predstaviti u komplementu dvojke, a zatim naći njihov zbir.

a) $133_{(10)}$ $-91_{(10)}$
b) $35_{(10)}$ $-17_{(10)}$
c) $-22_{(10)}$ $40_{(10)}$
d) $-14_{(10)}$ $-7_{(10)}$

Z11. Brojeve $A = -53_{(10)}$ i $B = 9_{(10)}$ predstaviti u komplementu dvojke, a zatim naći njihovu razliku $A - B$.

Z12. Odrediti decimalnu vrednost zadatih brojeva zapisanih u pokretnom zarezu po standardu *IEEE* 754:

a) $01000000111000000000000000000000_{(2)}$
b) $40C00000_{(16)}$
c) $42400000_{(16)}$
d) $11000010001110000000000000000000_{(2)}$
e) $C2700000_{(16)}$

Z13. Zadate decimalne brojeve zapisati u pokretnom zarezu u jednostrukom formatu po standardu *IEEE* 754.

a) $17.74_{(10)}$
b) $-8.25_{(10)}$
c) $240.1_{(10)}$
d) $-15_{(10)}$
e) $22.56_{(10)}$

2 Logička kola

U osnovi funkcionisanja svakog računarskog sistema je prosleđivanje informacije o tome da li u datom trenutku negde u sistemu postoji signal ili ne. Ova vrsta informacije se može predstaviti binarnim ciframa 0 i 1. Na primer, može se usvojiti da vrednost 0 označava odsustvo signala, a vrednost 1 da signal postoji. Pošto binarne vrednosti 0 i 1 tako dobijaju logičko značenje, one se u računarskim sistemima često nazivaju i *logičkim vrednostima*.

U prethodnom poglavlju uveden je pojam binarnog broja koji predstavlja niz binarnih cifara 0 i 1 određene dužine. Takođe je pokazano kako se nad binarnim brojevima mogu obavljati različite aritmetičke operacije (sabiranje, oduzimanje, množenje i deljenje). Zajednička osobina svih aritmetičkih operacija jeste da se binarni brojevi nad kojima se one obavljaju posmatraju kao jedinstvena celina. To je neophodno zato što kod ovih operacija postoji međusobni uticaj između različitih pozicija u binarnim brojevima. Naime, na rezultat aritmetičke operacije na određenoj poziciji utiču ne samo binarne vrednosti na toj poziciji, već i binarne vrednosti koje se nalaze na susednim pozicijama.

Često se javlja potreba za poređenjem binarnih brojeva na nivou svake binarne cifre pojedinačno (na primer, utvrđivanje da li su dva skupa signala koja se pojavljuju na različitim lokacijama u sistemu istovetna, pri čemu je svaki signal predstavljen jednom logičkom vrednošću). S obzirom da se u tu svrhu ne mogu koristiti aritmetičke operacije, u računarskim sistemima svoju široku primenu nalaze i tzv. *logičke operacije*.

Logičke operacije se izvode nad binarnim brojevima na nivou pojedinačnih cifara. To znači da se binarni brojevi ne posmatraju kao jedinstvene celine (kao u slučaju aritmetičkih operacija), već kao niz međusobno nezavisnih binarnih

vrednosti. Rezultat logičke operacije nad binarnim brojevima predstavlja niz parcijalnih rezultata dobijenih primenom te logičke operacije nad binarnim ciframa na svim pozicijama u zadatim binarnim brojevima.

U nastavku će biti opisane logičke operacije sabiranja, množenja, negacije i ekskluzivnog sabiranja (prve tri predstavljaju osnovu Bulove algebre).

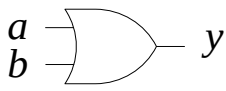
Pošto se logička operacija, u stvari, primenjuje nad pojedinačnim binarnim ciframa, broj mogućih kombinacija ovih cifara je konačan. Stoga se značenje logičke operacije može opisati pomoću kombinacione tablice. U tablici se navode sve moguće kombinacije ulaznih vrednosti (binarne vrednosti nad kojima se izvršava logička operacija), kao i rezultat operacije koji odgovara svakoj ulaznoj kombinaciji. Time je logička operacija u potpunosti definisana.

Logičke operacije se realizuju pomoću komponenata koje se nazivaju *logičkim kolima*. Logička kola imaju ulaze, na koje se dovode signali koji odgovaraju binarnim vrednostima nad kojima treba obaviti operaciju, i izlaz na kome se generiše signal koji predstavlja rezultat operacije. Različitim logičkim operacijama odgovaraju različita logička kola predstavljena odgovarajućim grafičkim simbolima.

2.1 Logičko sabiranje

Logičko sabiranje ili *ILI* operacija (*OR operation*) je binarna operacija jer omogućava sabiranje dve binarne vrednosti, *A* i *B*. Simbol za ovu operaciju je '+', pa se rezultat operacije *y* može predstaviti kao $y = a+b$. S obzirom da binarne vrednosti mogu biti samo 0 ili 1, broj mogućih kombinacija za sabiranje je četiri. U kombinacionoj tablici na slici 2.1 navedene su sve moguće kombinacije za *a* i *b*, kao i rezultat sabiranja *y* za svaku od njih. Na istoj slici prikazan je i grafički simbol logičkog *ILI* kola koje realizuje logičko sabiranje.

a	*b*	*y*
0	0	0
0	1	1
1	0	1
1	1	1



Slika 2.1 Kombinaciona tablica i grafički simbol za logičku *ILI* operaciju

Na osnovu kombinacione tablice može se zaključiti da je rezultat logičkog sabiranja 0 samo ako su obe binarne vrednosti koje se sabiraju jednake 0. To je ekvivalentno tvrđenju da je rezultat logičkog sabiranja 1 ako je bar jedna od vrednosti koje se sabiraju jednaka 1.

Postupak logičkog sabiranja dva višecifrena binarna broja odvija se tako što se posebno sabiraju binarne vrednosti na svakoj poziciji, a zatim se tako dobijeni rezultati posmatraju kao jedan binarni broj. To će biti ilustrovano na jednom primeru.

Data su dva binarna broja koja treba logički sabrati: $A = 10101_{(2)}$ i $B = 11001_{(2)}$. Rezultat Y je, takođe, binarni broj od pet cifara koje se nalaze na pozicijama 4, 3, 2, 1 i 0, posmatrano s leva na desno.

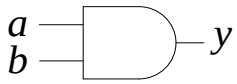
pozicija	4 3 2 1 0
A	1 0 1 0 1
B	1 1 0 0 1
$Y = A+B$	1 1 1 0 1

Vrednost na poziciji 0 u rezultatu se dobija kada se logički saberu vrednosti na poziciji 0 u brojevima A i B, tj. 1+1 = 1. Vrednost na poziciji 1 se dobija kada se logički saberu vrednosti na poziciji 1 u brojevima A i B, tj. 0+0 = 0. Na sličan način dobijaju se i vrednosti na pozicijama 2, 3 i 4, tako da se kao konačan rezultat logičkog sabiranja dobija $Y = A+B = 11101_{(2)}$. Treba uočiti da se dobijeni rezultat razlikuje od rezultata aritmetičkog sabiranja sprovedenog nad zadatim brojevima ($101110_{(2)}$).

2.2 Logičko množenje

Logičko množenje ili *I* operacija (*AND operation*) je binarna operacija koja omogućava množenje dve binarne vrednosti, a i b. Simbol za ovu operaciju je '·', pa se rezultat operacije y može predstaviti kao $y = a \cdot b$, ili jednostavnije $y = ab$. Slično kao kod logičkog sabiranja, broj mogućih kombinacija za množenje je četiri. Sve moguće kombinacije za a i b, kao i rezultat množenja y za svaku od njih prikazane se u kombinacionoj tablici na slici 2.2. Na slici je, takođe, prikazan i grafički simbol logičkog kola koje realizuje logičko množenje.

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1



Slika 2.2 Kombinaciona tablica i grafički simbol za logičku *I* operaciju

Iz prikazane kombinacione tablice može se videti da je rezultat logičkog množenja 1 samo ako su obe binarne vrednosti jednake 1, tj. rezultat je 0 ako je bar jedna od binarnih vrednosti koje se množe jednaka 0.

Dva višecifrena binarna broja se logički množe tako što se posebno množe binarne vrednosti na svakoj poziciji, a zatim se tako dobijeni rezultati posmatraju kao jedan binarni broj. Ovaj postupak će biti prikazan na primeru.

Neka su data dva binarna broja koja treba logički pomnožiti: $A = 11010_{(2)}$ i $B = 11001_{(2)}$. Rezultat Y ima, takođe, pet cifara koje se nalaze na pozicijama 4, 3, 2, 1 i 0, posmatrano s leva na desno.

pozicija	4 3 2 1 0
A	1 1 0 1 0
B	1 1 0 0 1
$Y = AB$	1 1 0 0 0

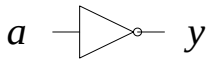
Vrednost na poziciji 0 u rezultatu se dobija kada se logički pomnože vrednosti na poziciji 0 u brojevima A i B, tj. $0 \cdot 1 = 0$. Vrednost na poziciji 1 se dobija kada se logički pomnože vrednosti na poziciji 1, tj. $1 \cdot 0 = 0$. Na sličan način dobijaju se i vrednosti na pozicijama 2, 3 i 4, tako da se kao konačan rezultat množenja dobija $Y = AB = 11000_{(2)}$.

2.3 Komplementiranje

Komplementiranje, negacija, invertovanje ili *NE* operacija (*NOT operation*) predstavlja unarnu operaciju jer se izvodi nad samo jednom binarnom vrednošću, a. Simbol za ovu operaciju je '–', pa se rezultat operacije y može predstaviti kao $y = \bar{a}$. S obzirom da binarna vrednost može biti samo 0 ili 1, moguće su samo dve ulazne kombinacije. U kombinacionoj tablici na slici 2.3 navedene su obe kombinacije za a, kao i rezultat komplementiranja y za svaku od njih. Takođe, prikazan je i grafički simbol logičkog kola koje realizuje komplementiranje. U računarskoj tehnici ovo logičko kolo se obično naziva *invertor*.

Na osnovu kombinacione tablice može se izvesti zaključak da je rezultat komplementiranja uvek suprotna logička vrednost od one koju treba komplementirati. Takođe treba zapaziti da je rezultat dvostrukog komplementiranja neke binarne vrednosti sama ta vrednost (komplement komplementa neke vrednosti je polazna vrednost).

a	*y*
0	1
1	0



Slika 2.3 Kombinaciona tablica i grafički simbol za logičku *NE* operaciju

Višecifreni binarni broj se komplementira tako što se posebno komplementira binarna vrednost na svakoj poziciji, a zatim se tako dobijeni rezultat posmatra kao jedan binarni broj. Ovaj postupak će biti prikazan na primeru.

Neka je dat binarni broj koga treba komplementirati: $A = 1001011_{(2)}$. Rezultat *Y* ima sedam cifara koje se nalaze na pozicijama 6, 5, ..., 1 i 0, posmatrano s leva na desno.

pozicija	6 5 4 3 2 1 0
A	1 0 0 1 0 1 1
$Y = \bar{A}$	0 1 1 0 1 0 0

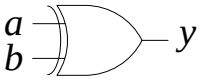
Vrednost na poziciji 0 u rezultatu se dobija kada se komplementira vrednost na poziciji 0 u broju *A*, tj. $\bar{1} = 0$. Vrednost na poziciji 1 se dobija kada se komplementira vrednost na poziciji 1 u broju *A*, tj. $\bar{1} = 0$. Na sličan način se dobijaju i vrednosti na pozicijama od 2 do 6, tako da se kao konačan rezultat komplementiranja dobija $Y = \bar{A} = 0110100_{(2)}$.

2.4 Logičko ekskluzivno sabiranje

Ekskluzivno sabiranje ili *EKSILI* operacija (*XOR operation*) je binarna operacija jer se obavlja nad dve binarne vrednosti, *a* i *b*. Simbol za ovu operaciju je '$\oplus$', pa se rezultat operacije *y* može predstaviti kao $y = a \oplus b$. Slično kao kod logičkog sabiranja i množenja, broj mogućih kombinacija za ekskluzivno sabiranje je četiri. Sve moguće kombinacije za *a* i *b*, kao i rezultat ekskluzivnog sabiranja *y* za svaku od njih prikazane se u kombinacionoj tablici na slici 2.4. Na slici je, takođe, prikazan i grafički simbol logičkog kola koje realizuje logičko ekskluzivno sabiranje.

Rezultat ekskluzivnog sabiranja je 0 samo ako su obe binarne vrednosti koje se sabiraju međusobno jednake. Takođe, može se reći i da je rezultat ekskluzivnog sabiranja 1 samo ako su vrednosti koje se sabiraju međusobno različite. Iz ovog poslednjeg tvrđenja je potekao i naziv ove operacije o isključivosti, tj. ekskluzivnosti.

a	*b*	*y*
0	0	0
0	1	1
1	0	1
1	1	0



Slika 2.4 Kombinaciona tablica i grafički simbol za *EKSILI* operaciju

Dva višecifrena binarna broja se ekskluzivno sabiraju tako što se posebno ekskluzivno sabiraju binarne vrednosti na svakoj poziciji, a zatim se tako dobijeni rezultati posmatraju kao jedan binarni broj. Ovaj postupak će biti prikazan na primeru.

Neka su data dva binarna broja koja treba sabrati ekskluzivno *ILI* operacijom: $A = 10010_{(2)}$ i $B = 11111_{(2)}$. Rezultat Y ima pet cifara koje se nalaze na pozicijama 4, 3, 2, 1 i 0, posmatrano s leva na desno.

pozicija	4 3 2 1 0
A	1 0 0 1 0
B	1 1 1 1 1
$Y = A \oplus B$	0 1 1 0 1

Vrednost na poziciji 0 u rezultatu se dobija kada se ekskluzivno saberu vrednosti na poziciji 0 u brojevima A i B, tj. $0 \oplus 1 = 1$. Vrednost na poziciji 1 se dobija kada se ekskluzivno saberu vrednosti na poziciji 1, tj. $1 \oplus 1 = 0$. Na sličan način dobijaju se i vrednosti na pozicijama 2, 3 i 4, tako da se kao konačan rezultat ekskluzivnog sabiranja dobija $Y = A \oplus B = 01101_{(2)}$.

Vežbanja

P1. Prikazati kombinacione tablice za sledeće logičke operacije:

a) logičko množenje
b) logičko sabiranje
c) komplementiranje
d) ekskluzivno sabiranje

P2. Nacrtati grafičke simbole za logička kola koja realizuju logičke operacije *I, ILI, NE* i *EKSILI*. Ako se na ulaze logičkog kola dovedu vrednosti 1 i 0, i na izlazu dobije vrednost 0, o kom kolu se radi?

P3. U čemu je razlika između aritmetičkih i logičkih operacija? Koliko bitova ima rezultat aritmetičkog, a koliko logičkog sabiranja dva *n*-bitska binarna broja?

Z1. Izvršiti logičke operacije AB, $A+B$, $A \oplus B$ i $\bar{A}$ nad sledećim brojevima:

a) $A = 110101010_{(2)}$ i $B = 101011111_{(2)}$
b) $A = 101011010_{(2)}$ i $B = 110110101_{(2)}$
c) $A = 11010101_{(2)}$ i $B = 01001100_{(2)}$
d) $A = 1101001010_{(2)}$ i $B = 0110111000_{(2)}$
e) $A = 11110111_{(2)}$ i $B = 11011111_{(2)}$

Z2. Naći aritmetički, a zatim i logički zbir sledećih binarnih brojeva:

a) $11010110_{(2)}$ i $10011111_{(2)}$
b) $100101100_{(2)}$ i $110011011_{(2)}$
c) $1001011_{(2)}$ i $1000110_{(2)}$
d) $1010101_{(2)}$ i $1100101_{(2)}$
e) $1111111_{(2)}$ i $1111111_{(2)}$

Z3. Naći aritmetički, a zatim i logički proizvod sledećih binarnih brojeva:

a) $1100_{(2)}$ i $1000_{(2)}$
b) $1110_{(2)}$ i $1001_{(2)}$
c) $1011_{(2)}$ i $1110_{(2)}$

d) $101_{(2)}$ i $110_{(2)}$
e) $111_{(2)}$ i $101_{(2)}$

Z4. Izvršiti logičke operacije $\overline{A \cdot B}$, $\overline{A + B}$ i $\overline{A \oplus B}$ nad sledećim brojevima:

a) $A = 101010_{(2)}$ i $B = 101111_{(2)}$
b) $A = 1011010_{(2)}$ i $B = 1110101_{(2)}$

3 Logičke funkcije

Kombinovanjem logičkih kola mogu se dobiti vrlo složene logičke strukture koje obezbeđuju potrebnu funkcionalnost u računarskom sistemu. Te strukture se opisuju logičkim funkcijama. S obzirom da se binarna logika (0 ili 1, „ima“ ili „nema“ signala) jednostavno može predstaviti prekidačkim elementom, logičke funkcije se često nazivaju i *prekidačkim funkcijama*.

Između aritmetičkih i logičkih funkcija postoje sličnosti, ali i značajne razlike. Logičke funkcije su jednostavnije od aritmetičkih jer operišu nad binarnim skupom vrednosti. Međutim, način razmišljanja karakterističan za logičke funkcije se često teško usvaja zbog navike stečene u radu sa aritmetičkim funkcijama u ranijem periodu.

Logička funkcija Y se, slično aritmetičkoj, može definisati nad proizvoljnim brojem promenljivih A, B, C, ... na sledeći način:

$$Y = f(A, B, C, ...)$$

Osnovna razlika u odnosu na aritmetičku funkciju je u tome što vrednost logičke funkcije Y mora pripadati skupu $\{0,1\}$. Takođe, i vrednosti promenljivih u logičkim funkcijama moraju pripadati istom skupu, tj. mogu biti samo 0 ili 1. Pošto se 0 i 1 nazivaju logičkim vrednostima, to se i promenljive u logičkim funkcijama nazivaju logičkim promenljivama.

Logičke promenljive u logičkoj funkciji su međusobno povezane logičkim operacijama (*I, ILI, NE,...*). Vrednost logičke funkcije za zadate vrednosti logičkih promenljivih dobija se kao rezultat izvršavanja logičkih operacija korišćenih u logičkoj funkciji.

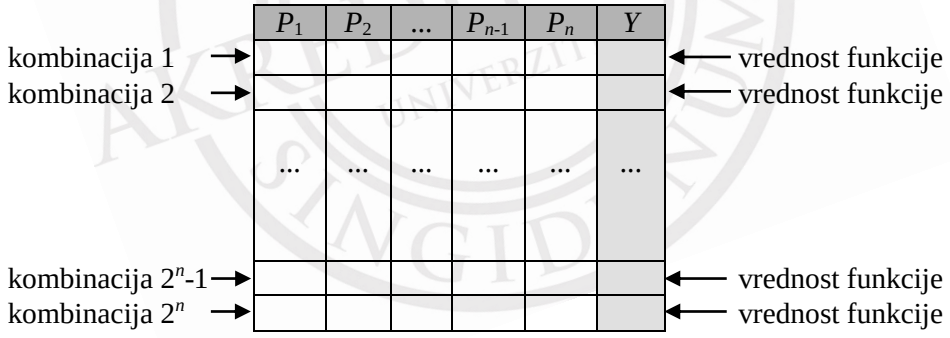
3.1 Predstavljanje logičkih funkcija

Logičke funkcije se mogu predstaviti na više načina. U nastavku će biti opisana tri često primenjivana načina: pomoću kombinacione tablice, u vidu algebarskog izraza i pomoću Karnoovih karti. Zatim će biti pokazano kako se može preći sa jednog na drugi način predstavljanja logičke funkcije.

3.1.1 Kombinacione tablice

Pošto logičke funkcije zavise od konačnog broja logičkih promenljivih (n), pri čemu svaka promenljiva može imati samo dve vrednosti, 0 ili 1, to je broj mogućih različitih kombinacija vrednosti promenljivih 2^n. Iz ovoga sledi da logičke funkcije imaju konačnu oblast definisanosti. Zahvaljujući tome, one se mogu predstaviti pomoću tablica koje se nazivaju kombinacionim tablicama ili tablicama istinitosti.

Kombinaciona tablica predstavlja tabelu u kojoj su date vrednosti logičke funkcije za sve moguće kombinacije vrednosti logičkih promenljivih od kojih ona zavisi. Na slici 3.1 prikazana je kombinaciona tablica kojom je predstavljena logička funkcija Y koja zavisi od n logičkih promenljivih $P_1, P_2, ..., P_n$.



Slika 3.1 Izgled kombinacione tablice za logičku funkciju n promenljivih

Leva strana tabele sadrži n kolona, za svaku promenljivu po jednu, i 2^n vrsta. Jedna vrsta odgovara jednoj kombinaciji vrednosti iz skupa {0,1} svih logičkih promenljivih koje se javljaju u funkciji. To znači da jedna kolona sadrži logičke vrednosti koje njoj odgovarajuća logička promenljiva ima u svim mogućim kombinacijama. Desna strana kombinacione tablice ima jednu kolonu koja odgovara samoj funkciji Y, i 2^n vrsta. Svaka vrsta sadrži vrednost koju logička funkcija ima za kombinaciju vrednosti promenljivih navedenu u istoj vrsti u levom delu tabele.

Pri popunjavanju svih mogućih kombinacija u levom delu tabele, lako može da dođe do grešaka (da se neke kombinacije ponove, ili da neke nedostaju) ukoliko se one popunjavaju nasumice. Stoga je dobro koristiti neki sistematičan pristup. Na primer, može se najpre popuniti poslednja kolona u levom delu tabele naizmeničnim upisivanjem vrednosti 0 i 1. Zatim se popunjava susedna kolona (levo od već popunjene) naizmeničnim upisom po dve vrednosti 0 i 1, pa sledeća susedna kolona sa po 4 vrednosti 0 i 1, pa sledeća po 8 vrednosti 0 i 1, i tako dalje (uvek po 2^i vrednosti 0 i 1) sve dok se ne popuni prva kolona u levom delu tabele. Ovakvo popunjavanje garantuje da su sve moguće kombinacije za zadati broj promenljivih unete.

Primer dobro popunjene kombinacione tablice koja predstavlja logičku funkciju četiri promenljive *A*, *B*, *C* i *D* dat je na slici 3.2.

A	*B*	*C*	*D*	*Y*
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Slika 3.2 Izgled kombinacione tablice za logičku funkciju četiri promenljive

Može se zapaziti da su kombinacije unete u skladu sa prethodno izloženim postupkom. Značenje bilo koje vrste u tabeli slično je sledećem tumačenju koje se odnosi, na primer, na drugu vrstu: ako promenljive *A*, *B* i *C* imaju vrednost 0, a promenljiva *D* vrednost 1, vrednost logičke funkcije je 1.

Kombinacione tablice su vrlo pregledan i jednostavan način za predstavljanje logičkih funkcija. Međutim, s obzirom da imaju 2^n vrsta, ovaj način predstavljanja logičkih funkcija postaje nepogodan i pri manjem broju promenljivih, tj. malim vrednostima *n*. Na primer, ako je $n = 5$, broj vrsta u tabeli je $2^5 = 32$, a ako je $n = 6$

broj vrsta raste na $2^6 = 64$. Ovakve dimenzije čine tabelu nepreglednom i nepraktičnom za rad.

Predstavljanje logičke funkcije pomoću tablice istinitosti biće ilustrovano na dva primera.

Prvi primer se odnosi na predstavljanje većinske logike pomoću kombinacione tablice. Pretpostavimo da tri glasača *A*, *B* i *C* glasaju za neki predlog tako što zaokružuju *za* (ako podržavaju predlog) ili *protiv* (ako ne podržavaju predlog). Označimo glas *za* predlog logičkom vrednošću 1, a glas *protiv* predloga logičkom vrednošću 0. Predlog je *usvojen* ako su dva ili više glasača glasala *za*, u suprotnom je *odbijen*. Označimo *usvojen* predlog logičkom vrednošću 1, a *odbijen* logičkom vrednošću 0. Zadatak je da se funkcija *Y*, koja pokazuje da li je predlog usvojen ili ne, predstavi tablicom istinitosti.

U rešavanju navedenog problema, najpre treba uočiti da funkcija *Y* zavisi od tri promenljive koje predstavljaju glasače *A*, *B* i *C*, pošto njihovi glasovi direktno utiču na vrednost funkcije, tj. da li je predlog usvojen ili ne. Dakle, leva strana kombinacione tablice ima tri kolone i $2^3 = 8$ vrsta. Desna strana tablice ima jednu kolonu i osam vrsta. Sada sledi popunjavanje svih mogućih kombinacija po vrstama leve strane tabele. Kombinacije se formiraju tako što, ukoliko je glasač glasao *za*, u polje njemu odgovarajuće kolone upisuje se vrednost 1, a ako je glasao *protiv*, upisuje se vrednost 0. Nakon što se popuni leva strana tabele, pristupa se određivanju vrednosti logičke funkcije *Y* za svaku od kombinacija i njenom upisivanju u odgovarajuće polje u desnom delu tablice. Vrednost funkcije je 1 ako je predlog *usvojen*, tj. ako je u odgovarajućoj kombinaciji u levom delu tabele više jedinica nego nula (to znači da su bar dva glasača glasala za predlog). U suprotnom, vrednost funkcije je 0. Nakon unosa vrednosti funkcije, dobija se tražena kombinaciona tablica koja je prikazana na slici 3.3.

A	*B*	*C*	*Y*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Slika 3.3 Kombinaciona tablica koja odgovara većinskoj logici

Drugi primer ima za cilj predstavljanje logičke funkcije *Y* koja generiše signal za pokretanje lifta (označen logičkom vrednošću 1) u vidu kombinacione tablice.

Pretpostavimo da se način funkcionisanja lifta može opisati pomoću tri logičke promenljive:

A - ima vrednost 1 ako su spoljna vrata lifta zatvorena, a 0 ako su otvorena

B - ima vrednost 1 ako su unutrašnja vrata lifta zatvorena, a 0 ako su otvorena

C - ima vrednost 1 ako se u liftu neko nalazi, a 0 ako je lift prazan

Signal za pokretanje lifta treba generisati samo u dve situacije: ako su spoljna i unutrašnja vrata zatvorena i u liftu ima nekoga, ili ako su spoljna vrata zatvorena i lift je prazan.

Slično kao i u prethodnom primeru, najpre treba uočiti da funkcija *Y* zavisi od tri promenljive *A*, *B* i *C*, pošto njihove vrednosti direktno utiču na to da li će se lift pokrenuti ili ne. Dakle, leva strana kombinacione tablice ima tri kolone i $2^3 = 8$ vrsta. Desna strana tablice ima jednu kolonu i osam vrsta. Sledi popunjavanje svih mogućih kombinacija po vrstama leve strane tabele. Kombinacije se formiraju tako što se za svaku promenljivu, u polje njoj odgovarajuće kolone upisuje vrednost 1 ili 0 zavisno od stanja koje ta promenljiva ima u toj kombinaciji. Na primer, ako posmatrana kombinacija podrazumeva da su spoljna vrata lifta zatvorena, a unutrašnja otvorena i da u liftu ima nekog, promenljiva *A* će imati vrednost 1, promenljiva *B* vrednost 0 i promenljiva *C* vrednost 1. Nakon što se popuni leva strana tabele, pristupa se određivanju vrednosti logičke funkcije *Y* za svaku od kombinacija i njenom upisivanju u odgovarajuće polje u desnom delu tablice. Vrednost funkcije je 1 ako kombinacija opisuje jednu od dve navedene situacije za pokretanje lifta. U suprotnom, vrednost funkcije je 0. Tražena kombinaciona tablica prikazana je na slici 3.4.

A	*B*	*C*	*Y*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Slika 3.4 Kombinaciona tablica koja opisuje funkcionisanje lifta

3.1.2 Algebarski oblik

Iako se svaka prekidačka funkcija može predstaviti kombinacionom tablicom, za funkcije sa većim brojem promenjivih to nije pogodna forma predstavljanja. U tom slučaju je bolje koristiti druge načine predstavljanja, kao na primer algebarski prikaz logičke funkcije.

U algebarskom prikazu, logička funkcija se predstavlja algebarskim izrazom koga čine logičke promenljive međusobno povezane logičkim operacijama (*I, ILI, NE*, ...). Postoje različiti načini algebarskog predstavljanja logičke funkcije. Jedan od često korišćenih načina je primena tzv. *savršenih normalnih formi*. Savršene normalne forme se pojavljaju u dva oblika, kao:

- savršena disjunktivna normalna forma (*SDNF*)
- savršena konjunktivna normalna forma (*SKNF*)

Neka je data logička funkcija Y koja zavisi od n logičkih promenljivih označenih sa $A_1, A_2, ..., A_n$. Bilo koja logička promenljiva, osim svoje originalne vrednosti (na primer A) ima i svoju negiranu vrednost ($\bar{A}$). Označimo sa $\tilde{A}$ promenljivu A ili njenu negiranu vrednost $\bar{A}$, tj. $\tilde{A} = A$ ili $\tilde{A} = \bar{A}$. Uvedimo sada dva nova pojma: potpuni proizvod i potpuna suma.

Potpuni proizvod predstavlja logički proizvod $\tilde{A}_1\tilde{A}_2 \ ... \ \tilde{A}_n$. Dakle, to je proizvod u kome se pojavljuju sve promenljive od kojih logička funkcija zavisi (zbog toga se proizvod i zove potpunim), s tim što su neke od promenjivih predstavljene svojom originalnom, a neke negiranom vrednošću. Potpuni proizvod ima vrednost 1 samo za jednu kombinaciju vrednosti promenljivih, dok za sve ostale kombinacije ima vrednost 0.

Na sličan način, *potpuna suma* se definiše kao logički zbir $\tilde{A}_1+\tilde{A}_2+ \ ... \ +\tilde{A}_n$. U potpunoj sumi se, takođe, pojavljuju sve promenljive funkcije (što opravdava naziv potpuna) bilo u svom originalnom ili negiranom obliku. Potpuna suma ima vrednost 0 samo za jednu kombinaciju vrednosti promenljivih, dok za sve ostale kombinacije ima vrednost 1.

Primenom uvedenih pojmova mogu se definisati *SDNF* i *SKNF*. *SDNF* predstavlja logički *zbir potpunih proizvoda*, a *SKNF* logički *proizvod potpunih suma*.

Postupak predstavljanja logičke funkcije u algebarskom obliku korišćenjem *SDNF* opisan je teoremom čije će tvrđenje biti dato bez dokaza:

Teorema 1: Svaka logička funkcija $Y = f(A_1, A_2, ..., A_n)$, izuzev konstante 0, može se na jedinstven način napisati u obliku

$$Y = P_1 + P_2 + ... + P_m \qquad (m \leq 2^n)$$

gde su P_1, P_2, ..., P_m potpuni proizvodi koji odgovaraju kombinacijama vrednosti promenljivih za koje funkcija Y ima vrednost 1, tj. kao *SDNF*.

Na sličan način, logička funkcija se u algebarskom obliku može predstaviti i korišćenjem *SKNF* u skladu sa sledećom teoremom:

Teorema 2: Svaka logička funkcija $Y = f(A_1, A_2, ..., A_n)$, izuzev konstante 1, može se na jedinstven način napisati u obliku

$$Y = S_1 S_2 ... S_m \qquad (m \leq 2^n)$$

gde su S_1, S_2, ..., S_m potpune sume koje odgovaraju kombinacijama vrednosti promenljivih za koje funkcija Y ima vrednost 0, tj. kao *SKNF*.

Tvrđenja ovih teorema biće ilustrovana na jednom primeru. Neka je data logička funkcija Y koja zavisi od tri logičke promenljive, $Y = f(A_1, A_2, A_3)$. Neka funkcija ima vrednost 1 za sledeće kombinacije vrednosti promenljivih A_1, A_2 i A_3: 010, 100, 101 i 111. Za sve ostale kombinacije vrednosti promenljivih, funkcija ima vrednost 0.

Prema prvoj teoremi, za kombinacije za koje funkcija ima vrednost 1 mogu se formirati sledeći potpuni proizvodi:

$$P_1 = \bar{A}_1 A_2 \bar{A}_3, \quad P_2 = A_1 \bar{A}_2 \bar{A}_3, \quad P_3 = A_1 \bar{A}_2 A_3 \quad \text{i} \quad P_4 = A_1 A_2 A_3.$$

Potpuni proizvodi su formirani tako što, ukoliko je vrednost promenljive u kombinaciji 1, promenljiva ulazi u proizvod sa svojom originalnom vrednošću, a ako je 0, u proizvod se uključuje negirana vrednost promenljive. Sada se, po tvrđenju teoreme, logička funkcija Y može predstaviti u algebarskom obliku u vidu *SDNF*, ili tzv. sume proizvoda kao:

$$Y = \bar{A}_1 A_2 \bar{A}_3 + A_1 \bar{A}_2 \bar{A}_3 + A_1 \bar{A}_2 A_3 + A_1 A_2 A_3$$

Dakle, logička funkcija se predstavlja u vidu sume proizvoda tako što se operacijom logičkog sabiranja *ILI* povežu svi potpuni proizvodi koji odgovaraju kombinacijama vrednosti promenljivih za koje funkcija ima vrednost 1. Iz datog algebarskog izraza sledi da ukoliko logičke promenljive dobiju vrednosti koje odgovaraju jednoj od gore navedenih kombinacija, taj proizvod u zbiru postaje 1 (ostali proizvodi su 0), pa vrednost funkcije postaje 1 (rezultat logičkog sabiranja je 1 ako je bar jedan sabirak jednak 1).

Ista logička funkcija može se predstaviti i u vidu *SKNF*, ili tzv. proizvoda suma. Prema drugoj teoremi, za kombinacije na kojima funkcija ima vrednost 0, a to su kombinacije 000, 001, 011 i 110, mogu se formirati sledeće potpune sume:

$S_1 = A_1+A_2+A_3$, $S_2 = A_1+A_2+\bar{A}_3$, $S_3 = A_1+\bar{A}_2+\bar{A}_3$ i $S_4 = \bar{A}_1+\bar{A}_2+A_3$.

Potpune sume su formirane tako što, ukoliko je vrednost promenljive u kombinaciji 0, promenljiva ulazi u sumu sa svojom originalnom vrednošću, a ako je 1, u sumu se uključuje negirana vrednost promenljive. Sada se, po tvrđenju druge teoreme, logička funkcija *Y* može predstaviti u algebarskom obliku u vidu *SKNF* ili proizvoda suma kao:

$$Y = (A_1+A_2+A_3)(A_1+A_2+\bar{A}_3)(A_1+\bar{A}_2+\bar{A}_3)(\bar{A}_1+\bar{A}_2+A_3)$$

Dakle, logička funkcija se predstavlja u vidu proizvoda suma tako što se operacijom logičkog množenja *I* povežu sve potpune sume koje odgovaraju kombinacijama vrednosti promenljivih za koje funkcija ima vrednost 0. Iz navedenog algebarskog izraza sledi da ukoliko logičke promenljive dobiju vrednosti koje odgovaraju jednoj od kombinacija 000, 001, 011 ili 110, ta suma u proizvodu postaje 0, pa vrednost funkcije postaje 0 (rezultat logičkog množenja je 0 ako je bar jedan činilac jednak 0).

3.1.3 Karnoove karte

Osim kombinacionih tablica, za predstavljanje logičkih funkcija često se koristi još jedan tabelarni prikaz, a to je Karnoova (*Karnaugh*) karta. Ova karta, kao i kombinaciona tablica, predstavlja tabelu u kojoj su date vrednosti logičke funkcije za sve moguće kombinacije vrednosti logičkih promenljivih od kojih funkcija zavisi. Razlika između ovih tabela je u njihovoj organizaciji.

Karnoova karta koja prikazuje logičku funkciju *Y* sa *n* logičkih promenljivih ima ukupno 2^n polja (toliko ima i mogućih kombinacija vrednosti promenljivih). Ukoliko je *n* paran broj, tabela ima $2^{n/2}$ vrsta i $2^{n/2}$ kolona, a ako je *n* neparan broj, $2^{(n-1)/2}$ vrsta i $2^{(n+1)/2}$ kolona (ili obrnuto, $2^{(n-1)/2}$ kolona i $2^{(n+1)/2}$ vrsta). Ovakvim rasporedom vrsta i kolona postiže se da Karnoova karta ima *izgled što sličniji kvadratu*, što doprinosi njenoj preglednosti.

Svakom polju u Karnoovoj karti odgovara jedna kombinacija vrednosti logičkih promenljivih funkcije. O kojoj kombinaciji je reč, određeno je *binarnim oznakama vrsta i kolona*. Naime, svakoj vrsti i koloni pridružena je binarna oznaka koja ukazuje na vrednosti koje odgovarajuća logička promenljiva ima u toj vrsti ili koloni. Da bi se pokazao način formiranja oznaka vrsta i kolona, najpre će biti uvedene neke pretpostavke.

Pretpostavimo da je skup svih *n* promenljivih funkcije podeljen u dva podskupa. Broj elemenata u podskupovima određuje broj vrsta, odnosno kolona u Karnoovoj karti. Tako, ukoliko je *n* paran broj, podskupovi imaju po *n*/2 elemenata, a ako je *n* neparan broj, jedan podskup ima (*n*-1)/2 elemenata, a drugi

(n+1)/2 elemenata. Pretpostavimo sada da je jedan podskup pridružen vrstama, a drugi kolonama (u slučaju parnog n, potpuno je svejedno koji podskup se pridružuje vrstama, a koji kolonama, dok u slučaju neparnog n, broj vrsta i kolona mora biti usklađen sa brojem elemenata u podskupu). Izbor konkretnih logičkih promenljivih koje će se naći u podskupovima može biti napravljen na različite načine i za svaki učinjeni izbor, Karnoova karta je ispravno definisana. Na osnovu uvedenih pretpostavki, oznake vrsta i kolona se formiraju kao sve moguće kombinacije vrednosti promenljivih koje se pojavljuju u podskupovima pridruženim vrstama, odnosno kolonama.

Pri raspoređivanju kombinacija po poljima Karnoove karte mora se poštovati pravilo da se kombinacije koje odgovaraju susednim poljima u Karnoovoj karti razlikuju samo u jednoj cifri. To se postiže tako što se poštuje pravilo da se i *susedne oznake* vrsta, odnosno kolona, takođe mogu *razlikovati samo u jednoj cifri* (Grejov kod). Karnoova karta se popunjava tako što se u svako polje unosi vrednost koju funkcija ima za kombinaciju promenljivih koja odgovara tom polju.

U nastavku će biti opisan postupak generisanja Karnoove karte za funkciju n promenljivih po koracima:

1. Od skupa logičkih promenljivih formirati dva podskupa promenljivih V_1 i V_2 sa približno istim brojem članova (detaljna analiza broja elemenata u podskupovima data je ranije). Koje će promenljive biti u jednom, a koje u drugom podskupu, nije od značaja. Neka podskup V_1 ima n1 elemenata, a podskup V_2 n2 elemenata (mora da važi $n1 + n2 = n$).
2. Nacrtati tablicu sa 2^{n1} vrsta i 2^{n2} kolona.
3. U gornji levi ugao karte, iznad vrsta upisati nazive svih logičkih promenljivih koje pripadaju podskupu V_1, a levo od kolona nazive svih promenljivih koje pripadaju podskupu V_2.
4. Sve moguće kombinacije vrednosti promenljivih iz podskupa V_1 upisati kao oznake vrsta. Voditi računa da susedne oznake moraju da se razlikuju samo u jednoj binarnoj cifri.
5. Sve moguće kombinacije vrednosti promenljivih iz podskupa V_2 upisati kao oznake kolona. Takođe, treba voditi računa da susedne oznake moraju da se razlikuju samo u jednoj binarnoj cifri.
6. U svako polje karte upisati binarnu vrednost koju funkcija ima za kombinaciju vrednosti promenljivih koja odgovara oznaci vrste i oznaci kolone tog polja.

Opisani postupak će biti primenjen za predstavljanje logičke funkcije Y koja zavisi od četiri promenljive A, B, C i D pomoću Karnoove karte. Neka funkcija Y ima vrednost 1 samo ako su vrednosti svih promenljivih međusobno jednake.

1. Od skupa logičkih promenljivih {*A*, *B*, *C*, *D*} koji ima $n = 4$ elementa formiraju se dva podskupa V_1 i V_2 od po $n/2=2$ elementa ($n1=2$ i $n2=2$). Neka je $V_1 = \{A, B\}$, a $V_2 = \{C, D\}$ (raspored promenljivh po podskupovima je mogao da bude i drugačiji).
2. Sledi crtanje tablice sa $2^{n1} = 2^2 = 4$ vrste i $2^{n2} = 2^2 = 4$ kolone.
3. U gornji levi ugao karte, iznad vrsta upisuju se promenljive iz podskupa V_1, tj. *AB*, a levo od kolona promenljive iz podskupa V_2, tj. *CD*.
4. Kao oznake vrsta unose se sve kombinacije vrednosti promenljivih *A* i *B* (ima ih $2^2 = 4$), pri čemu se vodi računa da se susedne oznake razlikuju samo u jednoj binarnoj cifri. Moguć redosled oznaka vrsta je: 00, 01, 11, 10.
5. Kao oznake kolona unose se sve kombinacije vrednosti promenljivih *C* i *D* (ima ih $2^2 = 4$), pri čemu se, takođe, vodi računa da se susedne oznake razlikuju samo u jednoj binarnoj cifri. Moguć redosled oznaka kolona je: 00, 01, 11, 10.
6. Sledi popunjavanje karte vrednostima funkcije. Pošto funkcija *Y* ima vrednost 1 samo ako promenljive *A*, *B*, *C* i *D* imaju istu logičku vrednost, to postoje samo dve kombinacije za koje je taj uslov ispunjen: 0000 i 1111. Dakle, samo dva polja polja u tabeli imaju vrednost 1. Prvo polje ima oznaku vrste 00 i oznaku kolone 00, a drugo polje oznaku vrste 11 i oznaku kolone 11. Konačni izgled Karnoove karte prikazan je na slici.

AB \ CD	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	0	0	1	0
10	0	0	0	0

Sledi još jedan primer u kome će Karnoovom kartom biti predstavljena funkcija koja zavisi od tri promenjive *A*, *B* i *C*. Neka je vrednost funkcije 1 ako bar dve od promenljivih imaju vrednost 1.

1. Od skupa logičkih promenljivih {*A*,*B*,*C*} koji ima $n = 3$ elementa, formiraju se podskupovi V_1 od $(n-1)/2 = 1$ elementa i V_2 od $(n+1)/2 = 2$ elementa ($n1 = 1$ i $n2 = 2$). Neka je $V_1 = \{A\}$, a $V_2 = \{B, C\}$.
2. Sledi crtanje tablice sa $2^{n1} = 2^1 = 2$ vrste i $2^{n2} = 2^2 = 4$ kolone.
3. U gornji levi ugao karte, iznad vrsta upisuju se promenljive iz podskupa V_1, tj. *A*, a levo od kolona promenljive iz podskupa V_2, tj. *BC*.

4. Kao oznake vrsta unose se sve moguće vrednosti promenljive A ($2^1 = 2$).
5. Kao oznake kolona unose se sve kombinacije vrednosti promenljivih B i C (ima ih $2^2 = 4$), pri čemu se vodi računa da se susedne oznake razlikuju samo u jednoj binarnoj cifri. Moguć redosled oznaka kolona je: 00, 01, 11, 10.
6. Sledi popunjavanje karte vrednostima funkcije. Pošto funkcija Y ima vrednost 1 samo ako bar dve promenljive imaju vrednost 1, to su moguće kombinacije za koje je taj uslov ispunjen: 011, 101, 110 i 111. Dakle, četiri polja polja u tabeli imaju vrednost 1. Izgled popunjene Karnoove karte prikazan je na slici.

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Logička funkcija se može jednostavno definisati i zadavanjem polja u Karnoovoj karti u kojima se nalaze jedinice (ili nule). Da bi se to omogućilo, poljima u Karnoovoj karti pridružuju se indeksi. *Indeks* nekog polja predstavlja binarnu kombinaciju vrednosti promenljivih koja odgovara tom polju predstavljenu u decimalnom obliku. Ilustracije radi, na slici 3.5 date su dve Karnoove karte za funkciju četiri promenljive. U karti levo, u svakom polju je naznačena binarna kombinacija vrednosti promenljivih koja odgovara tom polju, dok su u karti desno naznačeni indeksi koji su dobijeni konverzijom binarnih kombinacija iz karte levo u decimalne brojeve.

AB \ CD	00	01	11	10
00	0000	0010	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Slika 3.5 Indeksiranje Karnoove karte za funkciju sa četiri promenljive

Sada se, uz ovako definisane indekse, logička funkcija koja ima vrednost 1 za kombinacije vrednosti promenljivih *ABCD*: 0100, 1101, 1010, 1000 i 1111, može jednostavno zadati izrazom

$Y(1) = \{4, 8, 10, 13, 15\}$ ili izrazom $Y(0) = \{0, 1, 2, 3, 5, 6, 7, 9, 11, 12\}$

i predstaviti Karnoovom kartom na slici.

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	0	1	1	0
10	1	0	0	1

Ovde treba imati na umu da raspored indeksa u Karnoovoj karti zavisi od izbora promenljivih koje se nalaze u podskupovima V_1 i V_2 pridruženim vrstama, odnosno kolonama. U primeru je dat uobičajen izbor promenljivih (koji će i nadalje biti korišćen) da skupu V_1 pripada „prvih“ $n1$ promenljivih, a skupu V_2 ostale promenljive funkcije.

Karnoove karte su vrlo pregledne i efikasne u slučaju predstavljanja logičkih funkcija sa malim brojem promenljivih. Već za funkcije pet promenljivih, javljaju se problemi vezani za susednost, koji utiču na otežanu dalju primenu karti. Ovi problemi se rešavaju na različite načine (crtanje više tablica manjih dimenzija, uvođenje tablica redukovanih dimenzija i sl.) koji ovom prilikom neće biti razmatrani.

3.1.4 Promena načina predstavljanja logičke funkcije

Pošto se ista logička funkcija može predstaviti na bilo koji od opisanih načina, u ovom poglavlju će biti pokazano kako se funkcija prikazana u jednom obliku jednostavno može prevesti u drugi oblik.

Kombinaciona tablica u sumu proizvoda i obrnuto

U poglavlju o predstavljanju logičke funkcije u algebarskom obliku, razmatrana su dva algebarska metoda: suma proizvoda i proizvod suma. S obzirom da će u nastavku fokus biti isključivo na sumi proizvoda, ovde će biti dat samo postupak prevođenja kombinacione tablice u ovaj način algebarskog predstavljanja logičke funkcije.

Suština postupka prevođenja kombinacione tablice u sumu proizvoda i obrnuto je u tome da je broj vrsta u kojima je vrednost funkcije 1 u kombinacionoj tablici jednak broju članova u sumi proizvoda (za svaku ovakvu vrstu formira se po

jedan član). Dakle, kombinacije vrednosti promenljivih za koje funkcija ima vrednost 0 ne utiču na sumu proizvoda.

Na osnovu kombinacione tablice, suma proizvoda se može generisati na sledeći način:

1. U kombinacionoj tablici uočiti skup logičkih promenljivih od kojih funkcija zavisi, i pronaći sve vrste, tj. kombinacije vrednosti logičkih promenjivih, za koje logička funkcija ima vrednost 1.
2. Za svaku nađenu vrstu formirati potpuni proizvod koji joj odgovara (ako je vrednost neke promenljive u vrsti 0, ona u proizvod ulazi kao negirana vrednost, a ako je 1, kao originalna vrednost).
3. Napisati algebarski izraz koji logički sabira sve formirane potpune proizvode, čime je logička funkcija predstavljena sumom proizvoda.

Sledi primer prevođenja kombinacione tablice date na slici 3.6 u sumu proizvoda. Kao što se iz kombinacione tablice vidi, radi se o funkciji tri promenjive *A*, *B* i *C*, što znači da svaki potpuni proizvod sadrži te tri promenjive. Suma proizvoda se dobija logičkim sabiranjem svih potpunih proizvoda koji odgovaraju vrstama u kojima logička funkcija ima vrednost 1.

A	*B*	*C*	*Y*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

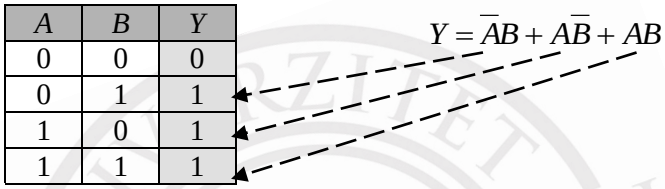
Slika 3.6 Generisanje sume proizvoda na osnovu kombinacione tablice

Postupak generisanja kombinacione tablice na osnovu date sume proizvoda je sledeći:

1. Na osnovu sume proizvoda uočiti skup *n* logičkih promenljivih od kojih funkcija zavisi (*n* odgovara broju promenljivih u bilo kom članu sume).
2. Za taj skup promenjivih nacrtati kombinacionu tablicu koja ima $1+2^n$ vrsta i $1+n$ kolona (uključujući zaglavlje tabele). U tablicu uneti nazive promenljivih i naziv funkcije *Y*.
3. U levi deo tablice uneti sve moguće kombinacije vrednosti promenljivih.

4. Za svaki potpuni proizvod u sumi proizvoda pronaći odgovarajuću kombinaciju u levom delu kombinacione tablice, pa za nju uneti vrednost logičke funkcije 1 u poslednju kolonu tablice. Preostala polja u poslednjoj koloni popuniti nulama.

Primer generisanja kombinacione tablice na osnovu sume proizvoda dat je na slici 3.7. Pošto svaki potpuni proizvod u sumi ima samo dve promenljive, to kombinaciona tablica ima pet vrsta i tri kolone. Kombinacije za koje funkcija ima vrednost 1, direktno se čitaju iz sume proizvoda.



Slika 3.7 Generisanje kombinacione tablice na osnovu sume proizvoda

Kombinaciona tablica u Karnoovu kartu i obrnuto

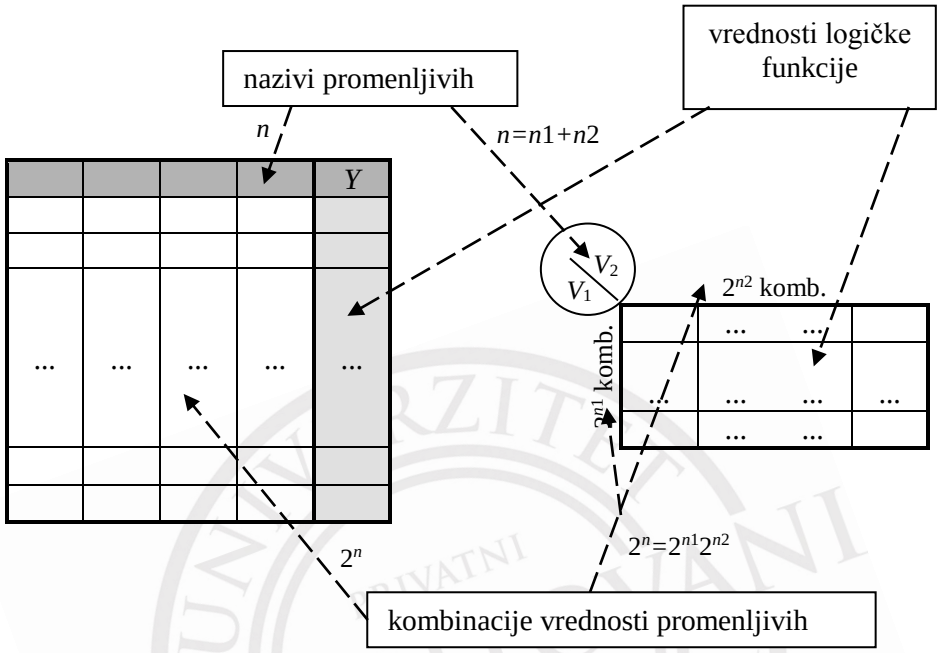
Kombinaciona tablica i Karnoova karta sadrže iste informacije o logičkoj funkciji, samo što su one u njima drugačije organizovane (videti sliku 3.8).

To su sledeće informacije:

a) *nazivi logičkih promenljivih*. U kombinacionoj tablici, imena promenljivih predstavljaju nazive kolona u levom delu tablice, dok se u Karnoovoj karti nalaze u levom gornjem uglu, kao elementi podskupova V_1 i V_2.

b) *kombinacije vrednosti logičkih promenljivih*. U kombinacionoj tablici, sve moguće kombinacije vrednosti promenljivih nalaze se u vrstama levog dela tablice. U Karnoovoj karti, svaka kombinacija je podeljena u dva dela: oznaku vrste i oznaku kolone, pa se tek njihovim spajanjem dobija kombinacija koja odgovara nekom polju u karti.

c) *vrednosti logičke funkcije za kombinacije logičkih promenljivih*. U kombinacionoj tablici, vrednosti funkcije se nalaze u koloni u desnom delu tablice, dok se u Karnoovoj karti nalaze u poljima karte.



Slika 3.8 Poređenje kombinacione tablice i Karnoove karte

Imajući u vidu navedeno, kombinaciona tablica se prevodi u Karnoovu kartu na sledeći način:

1. Skup promenjivih zadat u kombinacionoj tablici podeliti u dva podskupa V_1 i V_2. Zatim, na osnovu dimenzija ovih podskupova, izračunati dimenzije Karnoove karte. Nakon crtanja karte, u gornji levi ugao upisati promenljive iz skupova V_1, odnosno V_2.
2. Formirati oznake vrsta (kao sve moguće kombinacije vrednosti promenjivih iz skupa V_1) i oznake kolona (kao sve moguće kombinacije vrednosti promenljivih iz skupa V_2).
3. U svako polje Karnoove karte uneti vrednost logičke funkcije za kombinaciju koja odgovara tom polju. Vrednost funkcije za neku kombinaciju čita se iz poslednje kolone kombinacione tablice i vrste koja odgovara toj kombinaciji u kombinacionoj tablici.

Primer prevođenja kombinacione tablice u Karnoovu kartu dat je na slici 3.9. Kao što se iz kombinacione tablice vidi, radi se o funkciji Y koja zavisi od četiri promenjive A, B, C i D koje su, radi formiranja Karnoove karte svrstane u dva podskupa $\{A, B\}$ i $\{C, D\}$, pa su dimenzije Karnoove karte 4 x 4. Vrednosti

funkcije su direktno preuzete iz kombinacione tablice kao što je označeno strelicama na slici.

A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

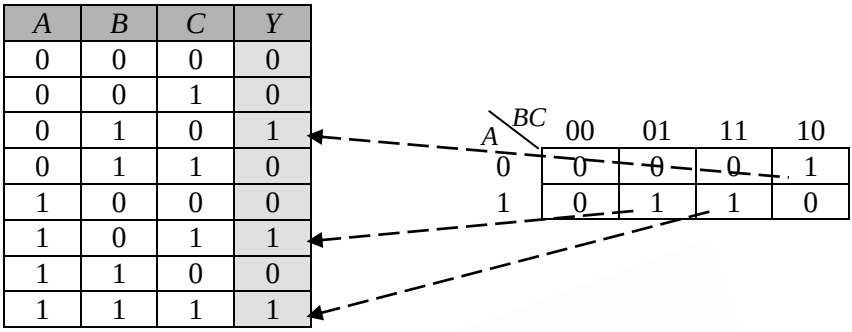
AB \ *CD*	00	01	11	10
00	0	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	0	0	1

Slika 3.9 Generisanje Karnoove karte na osnovu kombinacione tablice

Postupak prevođenja Karnoove karte u kombinacionu tablicu je sledeći:

1. Formirati skup svih promenjivih od kojih funkcija zavisi unijom podskupova V_1 i V_2 u Karnoovoj karti. Za taj skup promenjivih nacrtati kombinacionu tablicu (n promenljivih zahteva kombinacionu tablicu koja ima $1+2^n$ vrsta i $1+n$ kolona). U tablicu uneti nazive promenljivih i naziv funkcije *Y*.
2. U levi deo tablice uneti sve moguće kombinacije vrednosti promenljivih.
3. Za svaku kombinaciju u kombinacionoj tablici, naći polje u Karnoovoj karti koje joj odgovara (po oznakama vrsta i oznakama kolona) i iz njega pročitati vrednost funkcije. Pročitanu vrednost uneti u kombinacionu tablicu kao vrednost funkcije za tu kombinaciju.

Primer prevođenja Karnoove karte u kombinacionu tablicu dat je na slici 3.10. Kao što se iz Karnoove karte vidi, radi se o funkciji tri promenjive *A*, *B* i *C*, pa su dimenzije kombinacione tablice 9 x 4. Vrednosti funkcije se direktno preuzimaju iz Karnoove karte iz polja koja odgovaraju kombinacijama u kombinacionoj tablici.



Slika 3.10 Generisanje kombinacione tablice na osnovu Karnoove karte

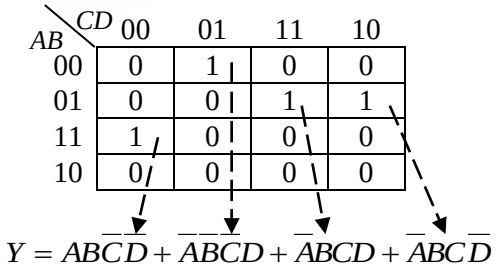
Karnoova karta u sumu proizvoda i obrnuto

Postupak prevođenja Karnoove karte u sumu proizvoda i obrnuto u osnovi je isti kao i u slučaju kombinacione tablice.

Na osnovu Karnoove karte, suma proizvoda se može generisati na sledeći način:

1. U Karnoovoj karti uočiti skup logičkih promenljivih od kojih funkcija zavisi (unija podskupova V_1 i V_2), i pronaći sva polja za koje logička funkcija ima vrednost 1.
2. Za svako nađeno polje formirati potpuni proizvod koji odgovara kombinaciji vrednosti promenljivih za to polje (kombinacija se dobija spajanjem oznake vrste i oznake kolone). Ako je vrednost promenljive u kombinaciji 0, ona u proizvod ulazi kao negirana vrednost, a ako je 1, kao originalna vrednost.
3. Napisati algebarski izraz koji logički sabira sve formirane potpune proizvode, čime je logička funkcija predstavljena sumom proizvoda.

Sledi primer prevođenja Karnoove karte date na slici 3.11 u sumu proizvoda.



Slika 3.11 Generisanje sume proizvoda na osnovu Karnoove karte

Kao što se iz Karnoove karte vidi, radi se o funkciji četiri promenjive *A*, *B*, *C* i *D*, pa svaki potpuni proizvod sadrži te četiri promenljive. Suma proizvoda se dobija logičkim sabiranjem svih potpunih proizvoda koji odgovaraju poljima u kojima logička funkcija ima vrednost 1.

Postupak formiranja Karnoove karte na osnovu sume proizvoda je sledeći:

1. Na osnovu sume proizvoda formirati skup logičkih promenljivih od kojih funkcija zavisi (sve promenljive koje se pojavljuju u bilo kom članu sume).
2. Skup promenljivih podeliti u dva podskupa V_1 i V_2. Na osnovu dimenzija ovih podskupova, izračunati dimenzije Karnoove karte. Nakon crtanja karte, u gornji levi ugao upisati promenljive iz skupova V_1, odnosno V_2.
3. Formirati oznake vrsta (kao sve moguće kombinacije vrednosti promenljivih iz skupa V_1) i oznake kolona (kao sve moguće kombinacije vrednosti promenljivih iz skupa V_2).
4. Svaki potpuni proizvod u sumi proizvoda predstaviti odgovarajućom kombinacijom vrednosti promenljivih. Ukoliko se promenljiva u proizvodu pojavljuje kao originalna vrednost, ona u kombinaciju ulazi kao logička vrednost 1, a ako se pojavljuje kao negirana, u kombinaciju ulazi kao 0.
5. Za dobijene kombinacije, u Karnoovoj karti pronaći polja koja im odgovaraju i u njih uneti vrednost logičke funkcije 1. U ostala polja Karnoove karte uneti vrednost 0.

Primer generisanja Karnoove karte na osnovu sume proizvoda dat je na slici 3.12. Pošto svaki potpuni proizvod u sumi ima četiri promenljive, to Karnoova karta ima dimenzije 4 x 4. Prvi član u sumi proizvoda predstavlja potpuni proizvod koji odgovara kombinaciji vrednosti promenljivih 1000, pošto se u njemu promenljiva *A* javlja kao originalna, a promenljive *B*, *C* i *D* kao negirane vrednosti. Ovoj kombinaciji odgovara polje koje se nalazi u četvrtoj vrsti i prvoj koloni Karnoove karte, pa u njega treba uneti vrednost funkcije 1. Na sličan način se za sve ostale članove sume proizvoda popunjavaju polja u kojima Karnoova karta ima vrednost 1. Preostala polja imaju vrednost 0.

$$Y = A\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}\overline{B}C\overline{D}$$

AB \ CD	00	01	11	10
00	0	0	0	1
01	0	1	0	0
11	0	0	0	0
10	1	0	0	0

Slika 3.12 Generisanje Karnoove karte na osnovu sume proizvoda

3.2 Realizacija logičkih funkcija

Način funkcionisanja računarskog sistema u velikoj meri se može opisati logičkim funkcijama. S obzirom da one, kao i sve druge funkcije, predstavljaju apstraktan pojam, da bi se prešlo sa opisa sistema na sistem koji zaista radi, neophodno je fizički realizovati logičke funkcije koje opisuju sistem. Logičke funkcije se realizuju pomoću prekidačkih mreža koje predstavljaju najvažnije komponente računara i drugih digitalnih sistema

Prekidačke mreže se sastoje od skupa elemenata povezanih tako da realizuju zadatu logičku funkciju. To znači da kada se na ulaze mreže dovedu određeni binarni signali, na njenim izlazima se dobijaju, u skladu sa logičkom funkcijom koju mreža realizuje, očekivane binarne vrednosti.

Prema načinu realizacije, prekidačke mreže mogu biti kombinacione i sekvencijalne. Glavna osobina *kombinacione mreže* je da vrednost na njenom izlazu (vrednost logičke funkcije) zavisi samo od trenutnog stanja na njenim ulazima (vrednosti logičkih promenljivih funkcije). Kombinacione mreže se realizuju kao kompozicija logičkih elemenata. Logički elementi, po pravilu, realizuju samo jednu jednostavnu prekidačku funkciju (imaju samo jedan izlaz). Određeni su funkcijom koju realizuju, grafičkim simbolom i nazivom elementa. Logički elementi su, na primer, logička kola *I*, *ILI*, *NE*, ekskluzivno *ILI*, itd. Za razliku od kombinacione mreže, kod *sekvencijalne mreže* vrednost na izlazu zavisi ne samo od trenutnog stanja na ulazima, već i od stanja u kome se mreža nalazi u datom trenutku (vrednosti unutar mreže). Sekvencijalne mreže se realizuju kao kompozicija logičkih i memorijskih elemenata, ili samo logičkih elemenata. Da li će kompozicija logičkih elemenata predstavljati kombinacionu ili sekvencijalnu mrežu, zavisi od načina povezivanja. Memorijski elementi se obično realizuju kao kompozicija logičkih elemenata. Glavna osobina memorijskih elemenata je da imaju samo dva stanja na koja utiču povratne veze koje postoje u strukturi ovih elemenata.

Rad sa prekidačkim mrežama podrazumeva rešavanje dve vrste problema. U nekim slučajevima potrebno je za postojeću prekidačku mrežu odrediti funkciju koju ona obavlja. Rešavanje ovog problema je predmet analize prekidačke mreže. Međutim, moguća je i obrnuta situacija u kojoj je potrebno zadatu logičku funkciju realizovati pomoću prekidačke mreže. Ovaj postupak se naziva sintezom prekidačke mreže.

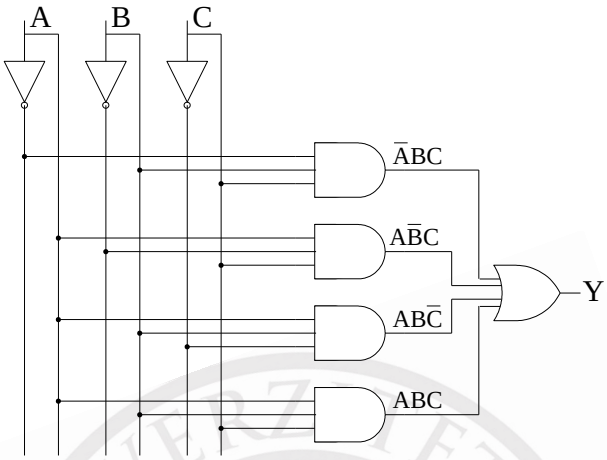
U ovom poglavlju će biti opisan jedan postupak sinteze prekidačke mreže koja realizuje logičku funkciju zadatu u aglebarskom obliku (ukoliko je funkcija predstavljena na neki drugi način, ranije opisanim postupcima može se prevesti u algebarski oblik).

Neka je data logička funkcija *Y* koja zavisi od *n* promenljivih međusobno povezanih logičkim operacijama. Ona se može realizovati prekidačkom mrežom koja:

- ima *n* ulaza koji odgovaraju logičkim promenljivama i jedan izlaz koji predstavlja vrednost funkcije *Y*
- ima onoliko različitih vrsta logičkih kola (*NE, I, ILI,* ...) koliko ima različitih logičkih operacija u funkciji
- ima onoliko logičkih kola jedne vrste koliko ih je potrebno za obavljanje svih logičkih operacija te vrste u funkciji

Ovo će biti ilustrovano na primeru funkcije $Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$. Prekidačka mreža koja realizuje ovu funkciju ima tri ulaza koja odgovaraju promenljivama *A, B* i *C* i jedan izlaz *Y*. U strukturi prekidačke mreže postoje tri vrste logičkih kola: *NE, I* i *ILI*. Pošto se svaka promenljiva u funkciji pojavljuje i kao negirana vrednost, prekidačka mreža mora da sadrži tri invertora (*NE* kola). Osim toga, za realizaciju svakog člana u zbiru potrebno je po jedno *I* kolo, što ukupno zahteva četiri logička *I* kola (ovde se dopušta da logičko *I* kolo ima više ulaza i jedan izlaz). Takođe, potrebno je i jedno *ILI* kolo sa više ulaza za realizaciju logičkog zbira.

U praktičnoj realizaciji, pogodno je poći od izlaza prekidačke mreže. Kao što se iz funkcije vidi, vrednost *Y* se dobija kao logički zbir četiri binarna signala. Stoga izlaz prekidačke mreže treba istovremeno da bude i izlaz logičkog *ILI* kola. Na ulaze *ILI* kola treba dovesti četiri binarna signala koji odgovaraju članovima zbira. To se može postići tako što se na svaki ulaz *ILI* kola dovede izlaz *I* kola koje realizuje odgovarajući član zbira. Da bi *I* kolo realizovalo član zbira, na njegove ulaze treba dovesti odgovarajuću kombinaciju logičkih promenljivih koje se u njemu pojavljuju. Ako se promenljiva u članu javlja kao originalna vrednost, na ulaz *I* kola se dovodi direktno sa ulaza prekidačke mreže, a ako se javlja kao negirana vrednost, mora se dovesti sa izlaza logičkog *NE* kola na čiji je ulaz dovedena originalna vrednost te promenljive sa ulaza prekidačke mreže. Opisani postupak realizacije se može ispratiti na slici 3.13 koja predstavlja prekidačku mrežu koja implementira zadatu logičku funkciju.



Slika 3.13 Realizacija logičke funkcije pomoću prekidačke mreže

3.3 Minimizacija logičkih funkcija

Struktura prekidačke mreže (broj i tip logičkih kola i način njihovog povezivanja) zavisi od algebarskog izraza kojim je predstavljena logička funkcija. Ista logička funkcija se može napisati u vidu različitih algebarskih izraza koji ne moraju biti podjednako pogodni za praktičnu realizaciju. Sa stanovišta ekonomičnosti i efikasnosti, cilj je da se za realizaciju funkcije upotrebi što manji broj logičkih kola uz što manje veza između njih. Postupak određivanja *najprostijeg algebarskog izraza* kojim se logička funkcija može predstaviti naziva se minimizacijom logičke funkcije.

Postoje različite metode minimizacije logičkih funkcija. Najopštija podela metoda minimizacije je na grafičke i algoritamske. Grafičke metode se zasnivaju na vizuelnoj analizi grafički predstavljene logičke funkcije. Ove metode su teorijski vrlo jednostavne, a pogodne su za funkcije sa manje promenljivih (do 6). Algoritamske metode koriste razne algoritme za transformisanje analitički ili tabelarno predstavljene funkcije. Složenije su od grafičkih metoda, ali su zato efikasne i u slučaju funkcija sa znatno većim brojem promenljivih.

U ovom poglavlju će biti predstavljena najčešće korišćenja grafička metoda minimizacije koja se zasniva na primeni Karnoove karte. Metoda će biti samo opisana, tj. tvrdnje i pravila na kojima ona zasniva neće biti dokazivani.

Postupak minimizacije logičke funkcije pomoću Karnoove karte sprovodi se u tri koraka:

1. zadatu logičku *funkciju predstaviti Karnoovom kartom* popunjenom na ranije opisan način
2. od polja Karnoove karte u kojima logička funkcija ima vrednost 1 *formirati pravougaone površine* poštujući unapred definisana pravila
3. na osnovu pravougaonih površina, po određenim pravilima, *ispisati minimalni zapis* logičke funkcije u vidu sume proizvoda

Pri *formiranju pravougaonih površina* treba poštovati sledeća *pravila*:

- pravougaone površine sadrže samo ona polja Karnoove karte u kojima logička funkcija ima vrednost 1 (nijedno polje sa vrednošću 0 ne može pripadati pravougaonoj površini)
- broj polja u pravougaonoj površini može biti samo 2^k, k = 0,1,2,... (površina može imati samo 1,2,4,8, ... polja)
- jednu pravougaonu površinu mogu da čine samo susedna polja u kojima je vrednost logičke funkcije 1
- pravougaone površine treba da budu što je moguće veće (da sadrže što više polja), a njihov broj što manji, jer se tako postiže najveći stepen minimizacije
- prema potrebi, isto polje može se naći u više pravougaonih površina

Pod *susednim poljima* u Karnoovoj karti podrazumevaju se:

- dva polja koja imaju jednu zajedničku stranicu
- dva polja koja se nalaze u istoj vrsti, s tim što je jedno polje u prvoj, a drugo u poslednjoj koloni (ova susednost se može videti ako se Karnoova karta urola kao cilindar po vertikali)
- dva polja koja se nalaze u istoj koloni, s tim što je jedno polje u prvoj, a drugo u poslednjoj vrsti (ova susednost se može videti ako se Karnoova karta urola kao cilindar po horizontali)

Nakon formiranja pravougaonih površina, može se pristupiti ispisivanju minimalnog zapisa logičke funkcije. Minimalni zapis ima oblik sume proizvoda sa onoliko članova (proizvoda) koliko ima pravougaonih površina, jer se za svaku pravougaonu površinu generiše po jedan član (proizvod) sume. Proizvod koji odgovara jednoj pravougaonoj površini dobija se analizom oznaka vrsta i kolona za sva polja koja pripadaju toj površini. On se formira tako što se za svaku logičku promenljivu pojedinačno analizira da li ona ulazi u proizvod i, ako ulazi, na koji način. Ova analiza se obavlja po sledećim *pravilima*:

- ako u oznakama vrsta koje odgovaraju pravougaonoj površini promenljiva ima vrednost i 0 i 1 (u oznaci jedne vrste ima vrednost 1, a u oznaci neke druge vrste vrednost 0), ta promenljiva ne ulazi u proizvod
- ako u svim oznakama vrsta koje odgovaraju pravougaonoj površini promenljiva ima vrednost 1, ta promenljiva ulazi u proizvod sa svojom originalnom vrednošću
- ako u svim oznakama vrsta koje odgovaraju pravougaonoj površini promenljiva ima vrednost 0, ta promenljiva ulazi u proizvod sa svojom negiranom vrednošću
- ako u oznakama kolona koje odgovaraju pravougaonoj površini promenljiva ima vrednost i 0 i 1 (u oznaci jedne kolone ima vrednost 1, a u oznaci neke druge kolone vrednost 0), ta promenljiva ne ulazi u proizvod
- ako u svim oznakama kolona koje odgovaraju pravougaonoj površini promenljiva ima vrednost 1, ta promenljiva ulazi u proizvod sa svojom originalnom vrednošću
- ako u svim oznakama kolona koje odgovaraju pravougaonoj površini promenljiva ima vrednost 0, ta promenljiva ulazi u proizvod sa svojom negiranom vrednošću

Iako opisani postupak minimizacije pomoću Karnoove karte izgleda prilično složeno, već nakon nekoliko urađenih primera, on prelazi u rutinu. U nastavku je dato nekoliko karakterističnih primera minimizacije logičke funkcije *Y* koja zavisi od četiri logičke promenljive *A*, *B*, *C* i *D*.

Primer 1. Minimizirati logičku funkciju zadatu Karnoovom kartom na slici.

AB \ CD	00	01	11	10
00	0	1	0	1
01	0	0	0	1
11	1	1	0	1
10	0	0	0	1

Rešenje: Pri formiranju pravougaonih površina, pogodno je poći od onih polja sa vrednošću 1 koja nemaju susednih polja, ili, ako takvih nema, od polja sa najmanjim brojem susednih polja. Ovaj princip se dalje može koristiti do kraja postupka formiranja pravougaonih površina.

U ovom primeru postoji samo jedno polje koje nema susednih, a to je polje kome odgovara kombinacija 0001. Stoga se od njega formira prva pravougaona

površina koja ima samo jedno polje. Sledeće polje sa najmanje suseda odgovara kombinaciji 1101. Ono ima samo jednog suseda (polje 1100) i stoga mora sa njim formirati pravougaonu površinu. Polje 1100 ima i drugog suseda, polje 1110, ali za njih nema potrebe formirati pravougaonu površinu, jer je polje 1110 bolje uključiti u veću pravougaonu površinu sa ostalim poljima poslednje kolone.

Pošto su formirane pravougaone površine, može se ispisati minimalna forma logičke funkcije. Ona predstavlja sumu proizvoda koja ima tri sabirka. Neka prvi sabirak odgovara pravougaonoj površini sa jednim poljem, drugi površini sa dva polja, a treći površini sa četiri polja.

Prvi sabirak u sumi se generiše sledećom analizom:

- pošto u oznaci prve vrste promenljivama *A* i *B* odgovaraju vrednosti 0, obe promenljive ulaze u proizvod kao negirane vrednosti
- pošto u oznaci druge kolone promenljivoj *C* odgovara vrednost 0, a promenljivoj *D* vrednost 1, onda promenljiva *C* ulazi u proizvod kao negirana, a promenljiva *D* kao originalna vrednost

Drugi sabirak se generiše sledećom analizom:

- pošto u oznaci treće vrste promenljivama *A* i *B* odgovaraju vrednosti 1, obe promenljive ulaze u proizvod kao originalne vrednosti
- pošto u oznakama prve i druge kolone promenljivoj *C* odgovara vrednost 0, ona u proizvod ulazi kao negirana vrednost
- pošto u oznakama prve i druge kolone promenljivoj *D* odgovara vrednost 0 u prvoj koloni, a vrednost 1 u drugoj, to vrednost ove promenljive ne utiče na vrednost funkcije u ovoj površini, pa promenljiva *D* ne ulazi u proizvod

Treći sabirak se generiše sledećom analizom:

- pošto u oznakama vrsta (od prve do četvrte) promenljivama *A* i *B* odgovaraju vrednosti bilo 0 bilo 1, ove promenljive ne ulaze u proizvod
- pošto u oznaci četvrte kolone promenljivoj *C* odgovara vrednost 1, a promenljivoj *D* vrednost 0, onda promenljiva *C* ulazi u proizvod kao originalna, a promenljiva *D* kao negirana vrednost

Na osnovu sprovedene analize, dobija se sledeća minimalna forma funkcije:

$$Y = \overline{A}\,\overline{B}\,\overline{C}D + AB\overline{C} + C\overline{D}$$

Iz dobijenog izraza se vidi da je proizvod u sumi jednostavniji ukoliko odgovara pravougaonoj površini sa više polja (veći je stepen minimizacije).

Primer 2. Naći minimalnu formu logičke funkcije zadate skupom indeksa

$$Y(1) = \{2, 3, 6, 8, 9, 12, 13\}.$$

Rešenje: U ovom primeru postoje dva polja (0011 i 0110) koja imaju samo po jedno susedno polje. Stoga se za svako od ovih polja mora napraviti pravougaona površina koja obuhvata to polje i njemu susedno. Pošto je susedno polje u oba slučaja isto (polje 0010), to će ono biti uključeno u dve pravougaone površine, što je dozvoljeno. Poslednja pravougaona površina obuhvata četiri preostale susedne jedinice.

AB \ CD	00	01	11	10
00	0	0	1	1
01	0	0	0	1
11	1	1	0	0
10	1	1	0	0

Minimalna forma logičke funkcije ima tri člana. Neka prvi član odgovara površini koju čine polja 0011 i 0010, drugi površini koju čine polja 0010 i 0110, a treći površini od četiri polja.

Prvi član se generiše sledećom analizom:

- pošto u oznaci prve vrste promenljivama *A* i *B* odgovaraju vrednosti 0, obe promenljive ulaze u proizvod kao negirane vrednosti
- pošto u oznakama treće i četvrte kolone promenljivoj *C* odgovara vrednost 1, a promenljivoj *D* i 0 i 1, to u proizvod ulazi samo promenljiva *C* i to kao originalna vrednost

Drugi član se generiše sledećom analizom:

- pošto u oznaci poslednje kolone promenljivoj *C* odgovara vrednost 1, a promenljivoj *D* vrednost 0, to promenljiva *C* ulazi u proizvod kao originalna vrednost, a promenljiva *D* kao negirana
- pošto u oznakama prve i druge vrste promenljivoj *A* odgovara vrednost 0, a promenljivoj *B* i 0 i 1, to u proizvod ulazi samo promenljiva *A* i to kao negirana vrednost

Treći član se generiše sledećom analizom:

- pošto u oznakama treće i četvrte vrste promenljivoj *A* odgovara vrednost 1, a promenljivoj *B* i 0 i 1, to u proizvod ulazi samo promenljiva *A* i to kao originalna vrednost
- pošto u oznakama prve i druge kolone promenljivoj *C* odgovara vrednost 0, a promenljivoj *D* i 0 i 1, to u proizvod ulazi samo promenljiva *C* i to kao negirana

Na osnovu sprovedene analize, dobijena je sledeća minimalna forma funkcije:

$$Y = \overline{A}\,\overline{B}C + \overline{A}C\overline{D} + A\overline{C}$$

Iz navedenog izraza može se zaključiti da pravouganim površinama sa istim brojem polja odgovaraju proizvodi sa istim brojem promenljvih (za površine od dva polja, proizvodi sadrže tri promenljive, a za površinu od četiri polja, samo dve). Ova činjenica može da posluži za brzu proveru ispravnosti formiranih proizvoda.

Primer 3. Minimizirati logičku funkciju zadatu skupom indeksa

$$Y(1) = \{1, 3, 4, 6, 9, 11\}.$$

Rešenje: Ovaj primer ilustruje susednost prve i poslednje vrste, odnosno prve i poslednje kolone. U skladu sa rasporedom polja sa vrednošću 1, mogu se napraviti dve pravougaone površine: prvu čine polja 0001, 0011, 1001 i 1011, a drugu polja 0100 i 0110.

AB \ CD	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	0	0	0	0
10	0	1	1	0

Minimalna forma logičke funkcije ima dva člana. Neka prvi član odgovara površini od četiri polja, a drugi površini od dva polja.

Prvi član se generiše sledećom analizom:

- pošto u oznakama prve i četvrte vrste promenljivoj *B* odgovara vrednost 0, a promenljivoj *A* i 0 i 1, to u proizvod ulazi samo promenljiva *B* i to kao negirana vrednost

- pošto u oznakama druge i treće kolone promenljivoj *D* odgovara vrednost 1, a promenljivoj *C* i 0 i 1, to u proizvod ulazi samo promenljiva *D* i to kao originalna vrednost

Drugi član se generiše sledećom analizom:

- pošto u oznaci druge vrste promenljivoj *A* odgovara vrednost 0, a promenljivoj *B* vrednost 1, to promenljiva *B* ulazi u proizvod kao originalna vrednost, a promenljiva *A* kao negirana
- pošto u oznakama prve i četvrte kolone promenljivoj *D* odgovara vrednost 0, a promenljivoj *C* i 0 i 1, to u proizvod ulazi samo promenljiva *D* i to kao negirana vrednost

Na osnovu sprovedene analize, dobijena je sledeća minimalna forma funkcije:

$$Y = \overline{B}D + \overline{A}\overline{B}\overline{D}$$

Primer 4. Minimizirati logičku funkciju zadatu Karnoovom kartom na sledećoj slici.

Rešenje: Ovaj primer ilustruje susednost uglova Karnoove karte. Naime, u skladu sa ranije datim opisom susednosti polja, polja u uglovima Karnoove karte (0000, 0010, 1000 i 1010) su susedna i formiraju prvu pravougaonu površinu. Drugu pravougaonu površinu formira svih osam polja u levom delu karte (cilj je da površine budu što veće).

AB \ CD	00	01	11	10
00	1	1	0	1
01	1	1	0	0
11	1	1	0	0
10	1	1	0	1

Minimalna forma logičke funkcije ima dva člana. Neka prvi član odgovara površini od četiri polja, a drugi površini od osam polja.

Prvi član se generiše sledećom analizom:

- pošto u oznakama prve i četvrte vrste promenljivoj *B* odgovara vrednost 0, a promenljivoj *A* i 0 i 1, to u proizvod ulazi samo promenljiva *B* i to kao negirana vrednost
- pošto u oznakama prve i četvrte kolone promenljivoj *D* odgovara vrednost 0, a promenljivoj *C* i 0 i 1, to u proizvod ulazi samo promenljiva *D* i to kao negirana vrednost

Drugi član se generiše sledećom analizom:

- pošto u oznakama vrsta (od prve do četvrte) promenljivama *A* i *B* odgovaraju bilo 0 bilo 1, to one ne ulaze u proizvod
- pošto u oznakama prve i druge kolone promenljivoj *C* odgovara vrednost 0, a promenljivoj *D* i 0 i 1, to u proizvod ulazi samo promenljiva *C* i to kao negirana vrednost

Na osnovu sprovedene analize, dobijena je sledeća minimalna forma funkcije:

$$Y = \overline{B}\,\overline{D} + \overline{C}$$

Vežbanja

P1. Navesti postupke za predstavljanje logičkih funkcija? Ukratko dati njihovo poređenje i objasniti kada se koji postupak primenjuje.

P2. Koliko vrsta i kolona ima kombinaciona tablica za predstavljanje logičke funkcije koja zavisi od tri promenljive? Šta se nalazi u levom, a šta u desnom delu tablice?

P3. Objasniti pojmove: potpuna suma i potpuni proizvod. Šta predstavlja *SDNF*, a šta *SKNF*?

P4. Šta predstavljaju oznake vrsta i kolona u Karnoovoj karti? Šta se upisuje u polja Karnoove karte?

P5. Navesti sva pravila susednosti pri minimizaciji logičke funkcije pomoću Karnoove karte.

Z1. Logičke funkcije zadate u algebarskom obliku predstaviti:

a) kombinacionim tablicama
b) u vidu Karnoovih karti

$$Y = \overline{A}BCD + \overline{A}BC\overline{D} + AB\overline{C}D + A\overline{B}\overline{C}D$$
$$Y = \overline{A}\overline{B}CD + AB\overline{C}D + ABCD + A\overline{B}\overline{C}D + A\overline{B}CD$$
$$Y = \overline{A}B\overline{C}D + \overline{A}BCD + ABCD$$
$$Y = \overline{A}\overline{B}C\overline{D} + AB\overline{C}\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D}$$
$$Y = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC + AB\overline{C}$$

Z2. Logičke funkcije (Y) prikazane datim kombinacionim tablicama predstaviti:

a) u obliku suma proizvoda
b) u vidu Karnoovih karti

A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

A	*B*	*C*	*Y*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Z3. Logičke funkcije prikazane Karnoovim kartama na slici predstaviti:

a) u algebarskom obliku
b) pomoću kombinacionih tablica

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	0	1	1
11	0	1	0	0
10	0	0	1	0

AB \ CD	00	01	11	10
00	0	1	0	1
01	0	0	1	0
11	0	1	0	0
10	0	1	0	0

AB \ C	0	1
00	1	0
01	0	1
11	0	0
10	0	1

Z4. Logička funkcija $Y = f(A,B,C,D)$ ima vrednost 1 samo ako promenljive B i D imaju različite vrednosti. Predstaviti ovu funkciju pomoću kombinacione tablice, Karnoove karte i u obliku *SDNF*.

Z5. Zadate logičke funkcije realizovati pomoću prekidačkih mreža:

a) $Y = \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C} + AB\overline{C}$
b) $Y = \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\overline{B}CD + \overline{A}BCD + A\overline{B}C\overline{D}$
c) $Y = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}\,\overline{C}$

Z6. Primenom Karnoove karte minimizirati logičke funkcije predstavljene sledećim sumama proizvoda:

a) $Y = \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D}$

b) $Y = \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D}$

c) $Y = \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + A\overline{B}\overline{C}D + A\overline{B}CD$

d) $Y = \overline{A}\overline{B}CD + AB\overline{C}D + ABCD + A\overline{B}\overline{C}D + A\overline{B}CD$

e) $Y = \overline{A}\overline{B}CD + \overline{A}BCD + \overline{A}BC\overline{D} + A\overline{B}CD$

Z7. Izvršiti minimizaciju logičkih funkcija (Y) koje zavise od četiri logičke promenljive (A, B, C i D), a zadate su pomoću indeksa u Karnoovoj karti:

a) $Y(1) = \{0, 1, 2, 5, 8, 10\}$
b) $Y(1) = \{2, 4, 6, 8, 10, 14\}$
c) $Y(0) = \{3, 9, 12, 13, 14\}$
d) $Y(1) = \{0, 6, 7, 8, 14, 15\}$
e) $Y(0) = \{4, 6, 7, 8, 12\}$

UNIVERZITET
PRIVATNI
AKREDITOVANI
UNIVERZITET
SINGIDUNUM

4 Standardni moduli

Pri projektovanju računarskih i drugih digitalnih sistema važnu ulogu imaju logičke, tj. prekidačke mreže. Pomoću njih se mogu ispuniti najrazličitiji funkcionalni zahtevi. Da bi se to postiglo, potrebno je uložiti dosta rada i vremena kako za projektovanje, tako i za samu realizaciju mreža. Olakšavajuća okolnost je ta što se u različitim modelima i verzijama računarskih i drugih digitalnih sistema javljaju mnogi zajednički zahtevi. Radi uštede u vremenu potrebnom za razvoj sistema, prekidačke mreže koje se često koriste izdvojene su kao standardni moduli. Procedure i postupci za razvoj standardnih modula su definisani, ali njihovo poznavanje nije neophodno za upotrebu modula. Naime, za korišćenje i uključivanje modula u složenije logičke strukture dovoljno je znati način njihovog funkcionisanja, tj. vezu koja postoji između njihovih ulaza i izlaza.

U zavisnosti od elemenata od kojih su napravljeni, standardni moduli mogu biti kombinacionog ili sekvencijalnog tipa. U nastavku će biti opisano po nekoliko najčešće korišćenih modula svakog tipa.

4.1. Standardni kombinacioni moduli

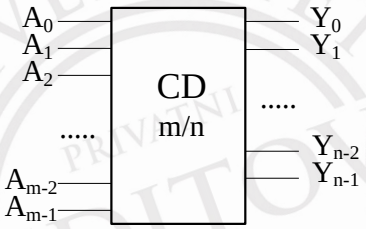
Standardni kombinacioni moduli su moduli koji sadrže samo logičke elemente. Od brojnih modula ovog tipa, za ovu priliku izdvojeni su sledeći: koder, dekoder, multiplekser, demultiplekser, sabirač i aritmetičko-logička jedinica.

Za svaki od navedenih modula biće opisano čemu služi i kako se realizuje, a za neke od njih (dekoder, multiplekser) biće dati i primeri korišćenja u drugim logičkim strukturama.

4.1.1 Koderi

Koderi su kombinacione mreže sa više ulaza i više izlaza koje služe za kodiranje informacija, tj. predstavljanje informacija u binarnom obliku. Informacija koju treba kodirati dovodi se na jedan od ulaza kodera (obično se označava vrednošću 1). Da bi koder ispravno radio, na svim ostalim ulazima mora biti vrednost 0. Dakle, u datom trenutku *može biti aktivan jedan i samo jedan ulaz* kodera. Na izlazima kodera se generiše *binarna vrednost koja odgovara rednom broju aktivnog ulaza*. Ukoliko se istovremeno dovedu signali na dva ili više ulaza kodera (što je fizički moguće uraditi), kodirana vrednost na izlazu neće biti ispravna.

Grafički simbol kodera sa *m* ulaza i *n* izlaza prikazan je na slici 4.1.



Slika 4.1 Koder *m*/*n*

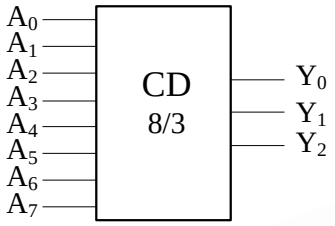
Koder koji ima *n* izlaza može da generiše najviše 2^n različitih kodnih kombinacija na svom izlazu. Pošto svaka kombinacija predstavlja kodiranu vrednost informacije dovedene na jedan od ulaza kodera, to znači da broj ulaza kodera ne može da bude veći od 2^n. U zavisnosti od broja ulaza i izlaza, razlikuju se dve vrste kodera:

- *potpuni koderi*, kod kojih važi $m = 2^n$ (imaju 2^n ulaza i *n* izlaza, što znači da su u potpunosti iskorišćene mogućnosti kodovanja)
- *nepotpuni koderi*, kod kojih važi $m < 2^n$ (imaju manje od 2^n ulaza i *n* izlaza, što znači da se neke kodne kombinacije nikad ne mogu pojaviti na izlazu)

U nastavku će biti detaljno opisan primer potpunog kodera 8/3. Ovaj koder ima osam ulaza i tri izlaza kao što je prikazano na slici 4.2.

Kao što je rečeno, u datom trenutku može biti aktivan (vrednost 1) samo jedan od ulaza kodera. Kod razmatranog kodera, u zavisnosti od rednog broja aktivnog ulaza, na izlazu se generiše kombinacija bitova koja odgovara tom rednom broju. Na primer, ako se signal dovede na ulaz broj 3 (A_3=1, ostali ulazi imaju vrednost 0), na izlazima će se generisati binarna vrednost 011 (Y_2=0, Y_1=1 i Y_0=1). Ovakav

način funkcionisanja kodera može se opisati kombinacionom tablicom prikazanom na slici 4.3.



Slika 4.2 Koder 8/3

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

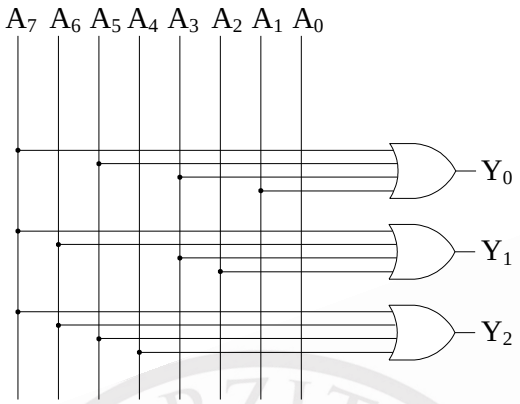
Slika 4.3 Kombinaciona tablica kodera 8/3

Leva strana tablice ne sadrži sve moguće kombinacije ulaznih promenljivih (njihov broj je $2^8 = 256$), već samo one u kojima je samo jedan od ulaza aktivan (8 kombinacija). To je u skladu sa već opisanim načinom funkcionisanja kodera (ostale kombinacije se ne mogu pojaviti na ulazu pri regularnom radu kombinacione mreže, pa nema potrebe unositi ih u tablicu). Na desnoj strani tablice date su funkcije izlaza Y_2, Y_1 i Y_0 koje, za datu ulaznu kombinaciju, formiraju odgovarajuću 3-bitsku binarnu vrednost.

Zavisnost izlaza od ulaza se može predstaviti sledećim *funkcijama izlaza* (videti kombinacionu tablicu):

$$Y_0 = A_1 + A_3 + A_5 + A_7$$
$$Y_1 = A_2 + A_3 + A_6 + A_7$$
$$Y_2 = A_4 + A_5 + A_6 + A_7$$

Na osnovu ovih funkcija, razmatrani koder 8/3 može se realizovati pomoću prekidačke mreže predstavljene na slici 4.4.

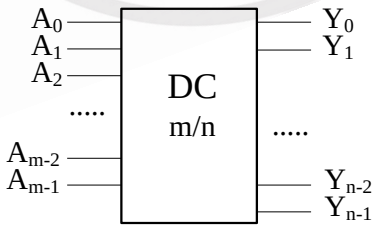


Slika 4.4 Prekidačka mreža koja realizuje koder 8/3

4.1.2 Dekoderi

Dekoderi su kombinacione mreže sa više ulaza i više izlaza čija je funkcija inverzna funkciji kodera. Oni služe za dekodovanje binarno kodiranih informacija. Binarno kodirana informacija se dovodi na ulaze dekodera, a na jednom od njegovih izlaza se generiše aktivna vrednost (obično vrednost 1). *Redni broj* tog *izlaza odgovara decimalnoj vrednosti binarne kombinacije na ulazima*. Na svim ostalim izlazima dekodera je vrednost 0. Dakle, kod dekodera, u datom trenutku *aktivan je jedan i samo jedan izlaz*. Sada ne može da dođe do greške (kao u slučaju kodera) zato što korisnik može da postavlja samo ulazne signale, dok se izlazni signali generišu automatski u skladu sa logikom rada samog dekodera.

Grafički simbol dekodera sa *m* ulaza i *n* izlaza prikazan je na slici 4.5.

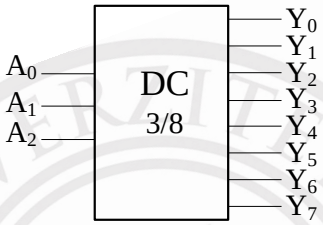


Slika 4.5 Dekoder *m*/*n*

Kod dekodera koji ima *m* ulaza, na njih se može dovesti najviše 2^m različitih binarno kodiranih informacija. Pošto svakoj ulaznoj informaciji odgovara po jedan izlaz, to znači da broj izlaza dekodera ne može da bude veći od 2^m. U zavisnosti od broja ulaza i izlaza, razlikuju se dve vrste dekodera:

- *potpuni dekoderi*, kod kojih važi $n = 2^m$ (imaju m ulaza i 2^m izlaza, što znači da su u potpunosti iskorišćene mogućnosti dekodovanja)
- *nepotpuni dekoderi*, kod kojih važi $n < 2^m$ (imaju m ulaza i manje od 2^m izlaza, što znači da se neke binarne kombinacije nikad ne pojavljuju na ulazu)

Ilustracije radi, sledi detaljan opis potpunog dekodera 3/8 koji ima tri ulaza i osam izlaza kao što je prikazano na slici 4.6.



Slika 4.6 Dekoder 3/8

U datom trenutku na ulaze dekodera dovodi se binarna kombinacija od tri bita. U zavisnosti od dovedene kombinacije, aktivira se samo jedan od izlaza dekodera i to onaj čiji redni broj odgovara decimalnoj vrednosti kombinacije bita na ulazu. Na primer, ako se na ulaze dekodera dovede kombinacija A_0=0, A_1=1, A_2=1, aktiviraće se izlaz sa rednim brojem 6 (Y_6=1, ostali izlazi imaju vrednost 0). Ovakav način funkcionisanja dekodera može se opisati kombinacionom tablicom prikazanom na slici 4.7.

A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Slika 4.7 Kombinaciona tablica dekodera 3/8

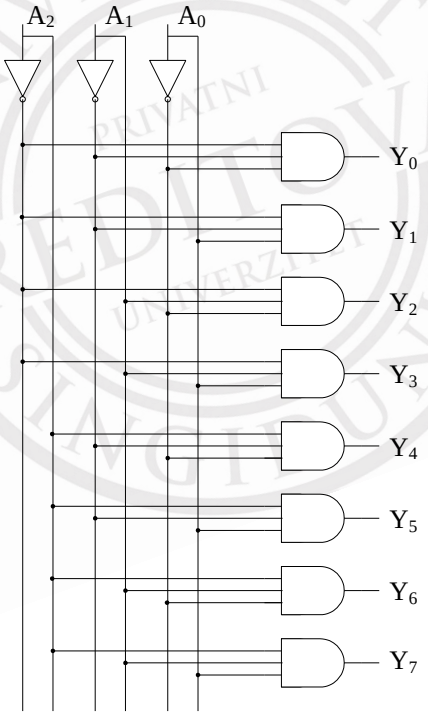
Pošto se radi o potpunom dekoderu, leva strana tablice sadrži sve moguće kombinacije ulaznih promenljivih (2^3 = 8 kombinacija). Na desnoj strani tablice

date su vrednosti izlaza Y_7, Y_6, Y_5, Y_4, Y_3, Y_2, Y_1 i Y_0 . Kao što se vidi, za datu kombinaciju na ulazu, aktivan je samo odgovarajući izlaz.

Zavisnost izlaza od ulaza može se predstaviti sledećim logičkim funkcijama (videti kombinacionu tablicu):

$$Y_0 = \bar{A}_2 \bar{A}_1 \bar{A}_0 \qquad Y_4 = A_2 \bar{A}_1 \bar{A}_0$$
$$Y_1 = \bar{A}_2 \bar{A}_1 A_0 \qquad Y_5 = A_2 \bar{A}_1 A_0$$
$$Y_2 = \bar{A}_2 A_1 \bar{A}_0 \qquad Y_6 = A_2 A_1 \bar{A}_0$$
$$Y_3 = \bar{A}_2 A_1 A_0 \qquad Y_7 = A_2 A_1 A_0$$

Na osnovu navedenih funkcija izlaza, dekoder 3/8 se može realizovati pomoću prekidačke mreže predstavljene na slici 4.8.



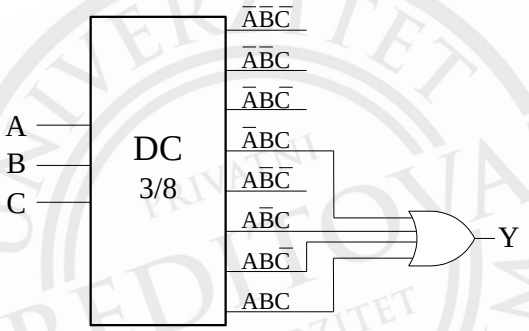
Slika 4.8 Prekidačka mreža koja realizuje dekoder 3/8

Dekoderi, kao i ostale logičke mreže, nalaze svoju primenu i u složenijim logičkim strukturama. U nastavku će biti opisan primer korišćenja dekodera za realizaciju zadate logičke funkcije.

Neka je data funkcija

$$Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC\ .$$

Za realizaciju ove funkcije pogodno je koristiti jedno *ILI* kolo sa četiri ulaza. Na svaki od ulaza treba dovesti po jedan član u sumi, tj. jedan proizvod. S obzirom da svaki proizvod predstavlja kombinaciju tri binarne promenljive, može se zapaziti da on odgovara jednoj od prekidačkih funkcija izlaza prethodno opisanog potpunog dekodera 3/8. Na primer, proizvodu $\bar{A}BC$ odgovara funkcija izlaza Y_3. To znači da se za realizaciju proizvoda u zadatoj funkciji može upotrebiti pomenuti dekoder na način prikazan na slici 4.9.



Slika 4.9 Realizacija logičke funkcije primenom dekodera 3/8

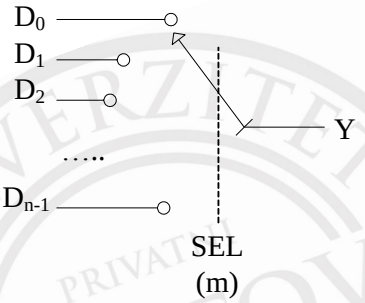
Kao što se vidi, logičke promenljive *A*, *B* i *C* od kojih zavisi funkcija predstavljaju ulaze dekodera, dok se vrednost funkcije *Y* dobija na izlazu *ILI* kola. Kada se na ulaz dekodera dovede kombinacija koja odgovara nekom članu u funkciji, onda se na odgovarajućem izlazu generiše vrednost 1 koja se direktno prosleđuje na ulaz *ILI* kola, pa funkcija *Y* ima vrednost 1. Ako se na ulaz dekodera dovede kombinacija koja ne odgovara nijednom članu funkcije, onda se na odgovarajućem izlazu dekodera generiše vrednost 1, ali taj izlaz nije povezan sa *ILI* kolom, pa ne utiče na njegov izlaz. S obzirom da su svi ostali izlazi dekodera neaktivni, na izlazu *ILI* kola funkcija *Y* ima vrednost 0. Ovime je pokazano da data šema zaista realizuje zadatu logičku funkciju.

4.1.3 Multiplekseri

Multiplekseri su kombinacione mreže sa više ulaza i jednim izlazom koje obavljaju funkciju digitalnog višepoložajnog prekidača. Oni imaju dve vrste ulaza:

- *informacione ulaze*, na koje se dovode ulazni signali multipleksera
- *selekcione ulaze*, koji selektuju jedan od informacionih ulaza

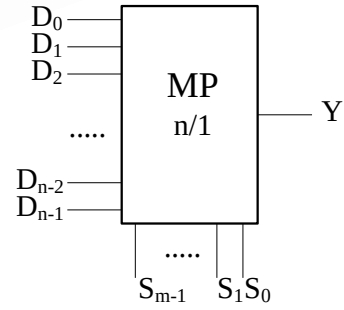
Multiplekser radi tako što samo jedan od binarnih signala dovedenih na informacione ulaze D_0, D_1,..., D_{n-1} prosleđuje na izlaz *Y*. Izbor binarnog signala koji će biti prosleđen na izlaz zavisi od *m* signala dovedenih na selekcione ulaze. Stoga se multiplekseri često nazivaju i *selektorima*. Ovakva funkcionalnost odgovara digitalnom višepoložajnom prekidaču prikazanom na slici 4.10.



Slika 4.10 Višepoložajni prekidač koji odgovara multiplekseru

U skladu sa binarnom kombinacijom dovedenom na selekcione ulaze, prekidač se postavlja u određeni položaj i direktno povezuje sa odgovarajućim informacionim ulazom. Signal koji postoji na tom informacionom ulazu se zatim prosleđuje na izlaz. Promenom kombinacije na selekcionim ulazima, položaj prekidača se menja, pa se sada na izlaz prosleđuje vrednost sa informacionog ulaza čiji redni broj odgovara novoj selekcionoj kombinaciji.

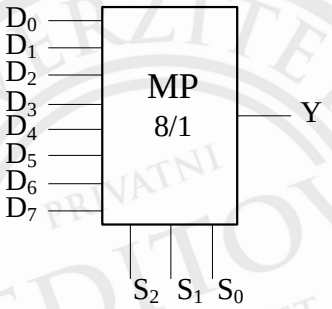
Grafički simbol multipleksera sa *n* informacionih ulaza, *m* selekcionih ulaza i jednim izlazom prikazan je na slici 4.11.



Slika 4.11 Multiplekser *n*/1

Broj informacionih ulaza multipleksera zavisi od broja njegovih selekcionih ulaza. Kod multipleksera koji ima m selekcionih ulaza, broj različitih binarnih kombinacija koje se na njima mogu generisati je 2^m. Pošto svaka kombinacija selektuje samo jedan informacioni ulaz, broj informacionih ulaza je $n = 2^m$. Ovaj odnos se može sagledati i na drugi način. Ako multiplekser ima n informacionih ulaza, za njihovu selekciju potrebno je obezbediti isto toliko različitih binarnih kombinacija na selekcionim ulazima. Stoga je broj potrebnih selekcionih ulaza $m = \log_2 n$.

U nastavku će biti opisan multiplekser 8/1 koji ima osam informacionih ulaza, tri selekciona ulaza i jedan izlaz, kao što je to prikazano na slici 4.12.



Slika 4.12 Multiplekser 8/1

Neka su na informacione ulaze multipleksera dovedeni željeni signali. U datom trenutku, dovođenjem selekcionih signala, na selekcionim ulazima se formira binarna kombinacija koja predstavlja redni broj informacionog ulaza sa koga se binarna vrednost (0 ili 1) direktno prosleđuje na izlaz. Ovakav način funkcionisanja multipleksera može se opisati kombinacionom tablicom prikazanom na slici 4.13.

S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

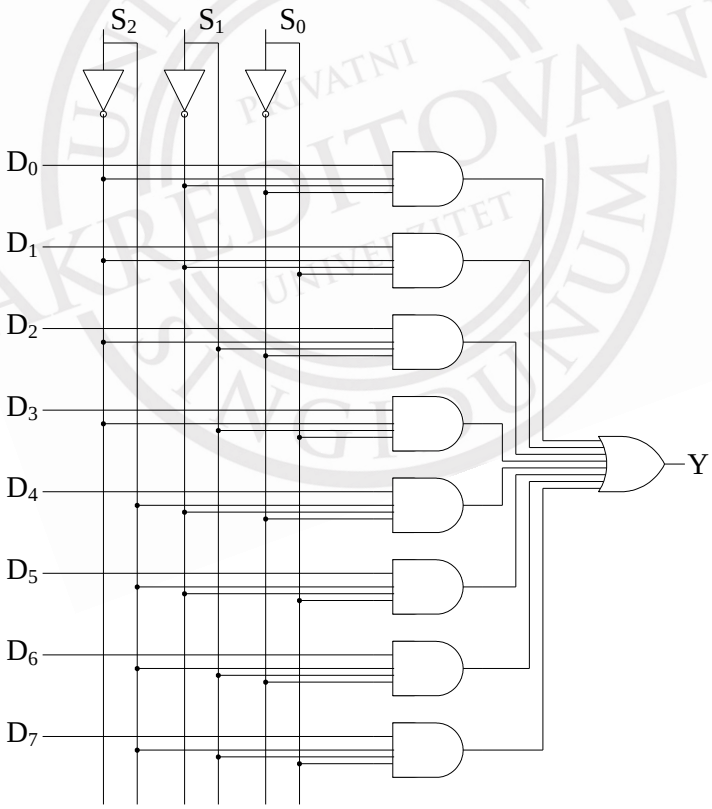
Slika 4.13 Kombinaciona tablica multipleksera 8/1

Leva strana tablice sadrži sve moguće binarne kombinacije koje se mogu pojaviti na selekcionim ulazima ($2^3 = 8$ kombinacija). U desnom delu tablice date su vrednosti izlaza koje zavise, ne samo od selekcionih signala, već i od trenutnih vrednosti signala na informacionim ulazima.

Zavisnost izlaza od ulaza može se, na osnovu kombinacione tablice, predstaviti sledećom logičkom funkcijom:

$$Y = D_0 \overline{S_2}\,\overline{S_1}\,\overline{S_0} + D_1 \overline{S_2}\,\overline{S_1} S_0 + D_2 \overline{S_2} S_1 \overline{S_0} + \ldots + D_7 S_2 S_1 S_0$$

Kao što se vidi, za zadatu kombinaciju na selekcionim ulazima (na primer 001) izlaz dobija vrednost odgovarajućeg informacionog ulaza (za navedeni primer to je vrednost D_1). Na osnovu date funkcije izlaza, multiplekser 8/1 se može realizovati pomoću prekidačke mreže predstavljene na slici 4.14.



Slika 4.14 Prekidačka mreža koja realizuje multiplekser 8/1

Multiplekser se može iskoristiti za realizaciju zadate logičke funkcije.

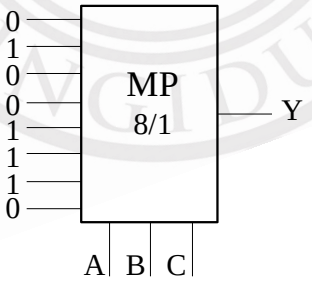
Neka je data funkcija

$$Y = \overline{A}\overline{B}C + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C}.$$

Ova funkcija se može predstaviti sledećom kombinacionom tablicom

A	*B*	*C*	*Y*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Iz tablice se vidi da je vrednost funkcije *Y* jednaka 1 samo za sledeće kombinacije promenljivih *A*, *B* i *C*: 001, 100, 101 i 110. Ukoliko se usvoji da promenljive *A*, *B* i *C* predstavljaju selekcione signale multipleksera 8/1 (tip multipleksera je uslovljen brojem selekcionih signala kojih ima 3), a funkcija *Y* izlaz multipleksera, onda se zadata funkcija može realizovati tako što se na informacione ulaze koji odgovaraju navedenim kombinacijama dovede vrednost 1, a na sve ostale informacione ulaze vrednost 0 (videti sliku 4.15).



Slika 4.15 Realizacija logičke funkcije primenom multipleksera 8/1

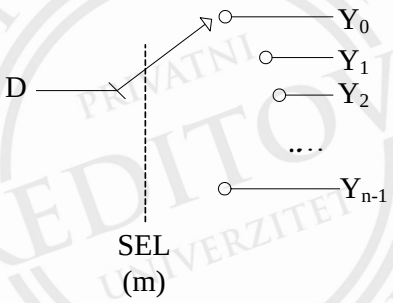
Dakle, svaki put kada se na selekcione ulaze dovede kombinacija koja odgovara nekom od proizvoda iz zadate funkcije (tada proizvod ima vrednost 1), prekidač se povezuje sa informacionim ulazom na kome je vrednost 1, pa se ta vrednost prosleđuje na izlaz. Ako se na selekcione ulaze dovede neka od preostalih kombinacija, prekidač se povezuje na informacioni ulaz na kome je vrednost 0, pa je i vrednost funkcije na izlazu multipleksera 0.

4.1.4 Demultiplekseri

Demultiplekseri su kombinacione mreže sa više ulaza i više izlaza čija je funkcija inverzna funkciji multipleksera. Oni, takođe, obavljaju funkciju digitalnog višepoložajnog prekidača, ali u obrnutom smeru. Imaju dve vrste ulaza:

- *informacioni ulaz*, na koji se dovodi ulazni signal
- *selekcione ulaze*, koji selektuju jedan od izlaza

Demultiplekser radi tako što binarni signal doveden na informacioni ulaz prosleđuje na jedan od izlaza. Izbor izlaza na koji se prosleđuje ulazni signal zavisi od signala dovedenih na selekcione ulaze. Zbog toga se demultiplekser ponekad naziva i *distributorom*. Ovakva funkcionalnost odgovara digitalnom višepoložajnom prekidaču prikazanom na slici 4.16.

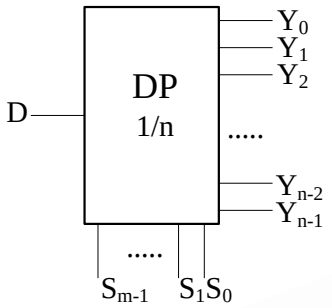


Slika 4.16 Višepoložajni prekidač koji odgovara demultiplekseru

U skladu sa binarnom kombinacijom dovedenom na selekcione ulaze, prekidač se postavlja u odgovarajući položaj i neposredno povezuje sa jednim od izlaza. Signal koji postoji na informacionom ulazu direktno se prosleđuje na taj izlaz.

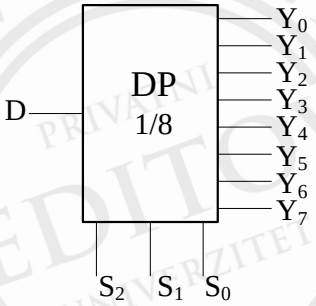
Grafički simbol demultipleksera sa jednim informacionim ulazom, m selekcionih ulaza i n izlaza prikazan je na slici 4.17.

Broj izlaza demultipleksera uslovljen je brojem njegovih selekcionih ulaza. Kod demultipleksera koji ima m selekcionih ulaza, broj različitih binarnih kombinacija koje se na njima mogu generisati je 2^m. Pošto svaka kombinacija selektuje samo jedan izlaz, broj izlaza je, takođe, $n = 2^m$. Može se reći i sledeće: ako demultiplekser ima n izlaza, za njihovu selekciju potrebno je obezbediti isto toliko različitih binarnih kombinacija na selekcionim ulazima. Stoga je broj potrebnih selekcionih ulaza $m = \log_2 n$.



Slika 4.17 Demultiplekser 1/*n*

Realizacija demultipleksera će biti objašnjena na primeru demultipleksera 1/8 koji ima jedan informacioni ulaz, tri selekciona ulaza i osam izlaza (videti sliku 4.18).



Slika 4.18 Demultiplekser 1/8

Neka je na informacioni ulaz demultipleksera doveden željeni signal. U datom trenutku, dovođenjem selekcionih signala, na selekcionim ulazima se formira binarna kombinacija koja predstavlja redni broj izlaza na koji će biti prosleđen signal sa informacionog ulaza. Ovakav način funkcionisanja demultipleksera može se predstaviti kombinacionom tablicom prikazanom na slici 4.19.

S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	*D*
0	0	1	0	0	0	0	0	0	*D*	0
0	1	0	0	0	0	0	0	*D*	0	0
0	1	1	0	0	0	0	*D*	0	0	0
1	0	0	0	0	0	*D*	0	0	0	0
1	0	1	0	0	*D*	0	0	0	0	0
1	1	0	0	*D*	0	0	0	0	0	0
1	1	1	*D*	0	0	0	0	0	0	0

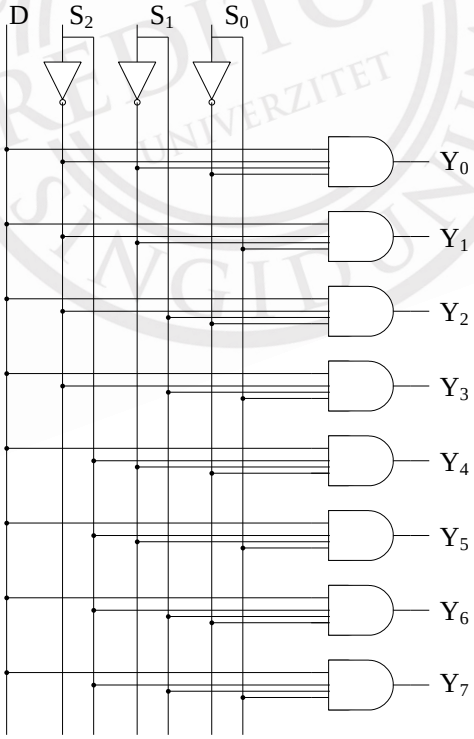
Slika 4.19 Kombinaciona tablica demultipleksera 1/8

Leva strana tablice sadrži sve moguće kombinacije koje se mogu pojaviti na selekcionim ulazima ($2^3 = 8$ kombinacija). U desnom delu tablice date su vrednosti izlaza koje zavise, ne samo od selekcionih signala, već i od trenutne vrednosti signala na informacionom ulazu.

Na osnovu kombinacione tablice, zavisnost izlaza od ulaza može se predstaviti sledećim funkcijama izlaza:

$$Y_0 = D\overline{S_2}\,\overline{S_1}\,\overline{S_0} \qquad Y_4 = DS_2\,\overline{S_1}\,\overline{S_0}$$
$$Y_1 = D\overline{S_2}\,\overline{S_1}S_0 \qquad Y_5 = DS_2\,\overline{S_1}S_0$$
$$Y_2 = D\overline{S_2}S_1\,\overline{S_0} \qquad Y_6 = DS_2S_1\,\overline{S_0}$$
$$Y_3 = D\overline{S_2}S_1S_0 \qquad Y_7 = DS_2S_1S_0$$

Kao što se vidi, za zadatu kombinaciju na selekcionim ulazima (na primer 101) odgovarajući izlaz (izlaz 5) dobija vrednost sa informacionog ulaza (*D*). Na osnovu datih funkcija izlaza, demultiplekser 1/8 može se realizovati pomoću prekidačke mreže predstavljene na slici 4.20.



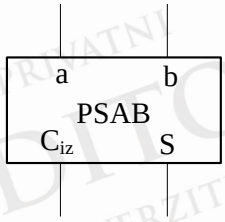
Slika 4.20 Prekidačka mreža koja realizuje demultiplekser 1/8

4.1.5 Sabirači

Sabirači su kombinacione mreže koje realizuju operaciju aritmetičkog sabiranja dva *jednobitna* binarna broja. U zavisnosti od načina sabiranja, razlikuju se dve vrste sabirača:

- polusabirači
- potpuni sabirači

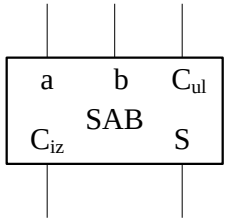
Polusabirač predstavlja mrežu koja ima dva ulaza na koja se dovode ulazni signali (*a* i *b*) koje treba sabrati (slika 4.21). Ulazni signali predstavljaju binarne cifre 0 ili 1. Mreža ima i dva binarna izlaza *S* i C_{iz}. Na izlazu *S* dobija se rezultat sabiranja, dok se na izlazu C_{iz} dobija prenos u stariji (viši) razred. Polusabirač nema mogućnost potpunog sabiranja zato što ne koristi prenos iz prethodnog (nižeg) razreda. Stoga se on ne može primeniti za sabiranje višecifrenih binarnih brojeva.



Slika 4.21 Polusabirač

Za razliku od polusabirača, *potpuni sabirač* uzima u obzir i prenos iz prethodnog razreda. On ima iste ulaze i izlaze kao polusabirač, samo što mu je dodat još jedan, treći ulaz C_{ul} na koji se dovodi binarni signal koji predstavlja prenos iz prethodnog razreda. Zahvaljujući ovom ulazu, potpuni sabirač može da se koristi za sabiranje višecifrenih binarnih brojeva.

Grafički simbol potpunog sabirača prikazan je na slici 4.22.



Slika 4.22 Potpuni sabirač

Postupak sabiranja dva jednobitna binarna broja može se predstaviti kombinacionom tablicom prikazanom na slici 4.23.

Leva strana tablice sadrži sve moguće kombinacije koje se mogu pojaviti na ulazima potpunog sabirača, dok su na desnoj strani date vrednosti na izlazima sabirača za svaku ulaznu kombinaciju.

C_{ul}	a	b	C_{iz}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Slika 4.23 Kombinaciona tablica potpunog sabirača

Zavisnost izlaza od ulaza može se predstaviti sledećim funkcijama izlaza:

$$S = a \oplus b \oplus C_{ul}$$
$$C_{iz} = (a \oplus b)C_{ul} + ab$$

Ove funkcije izvedene su na osnovu kombinacione tablice i izraza kojima se opisuje funkcija izlaza logičkog ekskluzivno *ILI* kola (videti kombinacionu tablicu za ovu operaciju, poglavlje 2):

$$a \oplus b = \bar{a}b + a\bar{b} \qquad \text{ili} \qquad \overline{a \oplus b} = \bar{a}\bar{b} + ab$$

Sledi izvođenje funkcija izlaza potpunog sabirača:

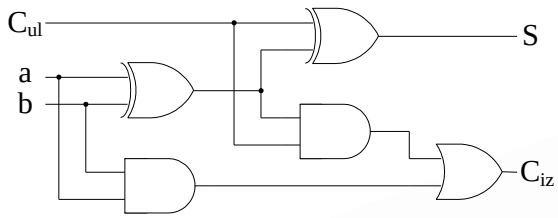
$$S = \overline{C_{ul}}\bar{a}b + \overline{C_{ul}}a\bar{b} + C_{ul}\bar{a}\bar{b} + C_{ul}ab = (\bar{a}b + a\bar{b})\overline{C_{ul}} + (\bar{a}\bar{b} + ab)C_{ul}$$

$$S = (a \oplus b)\overline{C_{ul}} + (\overline{a \oplus b})C_{ul} = a \oplus b \oplus C_{ul}$$

$$C_{iz} = \overline{C_{ul}}ab + C_{ul}\bar{a}b + C_{ul}a\bar{b} + C_{ul}ab = (\bar{a}b + a\bar{b})C_{ul} + ab(\overline{C_{ul}} + C_{ul})$$

$$C_{iz} = (a \oplus b)C_{ul} + ab$$

Na osnovu funkcija izlaza, potpuni sabirač se može realizovati pomoću prekidačke mreže prikazane na slici 4.24.

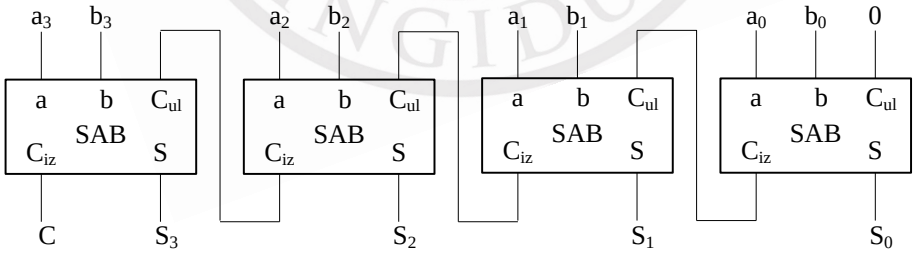


Slika 4.24 Prekidačka mreža koja realizuje potpuni sabirač

Potpuni sabirači se najčešće koriste za sabiranje višecifrenih binarnih brojeva. Pošto svaki potpuni sabirač vrši sabiranje samo na nivou jedne binarne cifre (jednog razreda), sabiranje *višecifrenih* brojeva se ostvaruje kaskadnim povezivanjem onoliko potpunih sabirača koliko ima cifara u binarnim brojevima koje treba sabrati. *Kaskadna veza* potpunih sabirača ostvaruje se tako što se izlaz za prenos u viši razred (C_{iz}) jednog sabirača vezuje na ulaz za prenos iz nižeg razreda (C_{ul}) sabirača koji se nalazi u sledećem razredu.

Pomoću potpunih sabirača mogu se sabirati kako neoznačeni brojevi, tako i označeni brojevi predstavljeni u komplementu dvojke.

U nastavku će biti opisan četvorobitni sabirač koji omogućava sabiranje dva četvorocifrena binarna broja $A = a_3a_2a_1a_0$ i $B = b_3b_2b_1b_0$. Za realizaciju ovog sabirača potrebno je kaskadno povezati četiri potpuna sabirača (za svaki razred po jedan) kao što je to prikazano na slici 4.25.



Slika 4.25 Realizacija četvorobitnog sabirača

Kao što se vidi, na ulaz C_{ul} sabirača u najnižem razredu dovedena je vrednost 0 zato što nema prenosa iz nižeg razreda. U svim ostalim razredima, sabirači su povezani tako što je na njihov ulaz C_{ul} doveden signal sa izlaza C_{iz} sabirača u prethodnom razredu, dok je njihov izlaz C_{iz} vezan za ulaz sabirača C_{ul} u narednom razredu. Na taj način je obezbeđeno ispravno sabiranje koje uključuje prenose

između razreda. Izuzetak predstavlja izlaz C_{iz} sabirača u najvišem razredu koji nije nigde povezan, već se uključuje u rezultat.

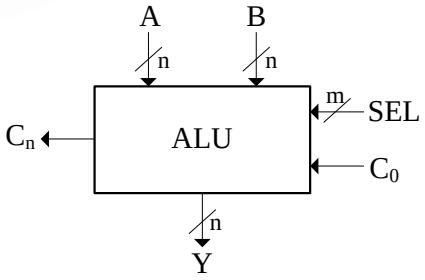
Postupak sabiranja na primeru četvorocifrenih brojeva $A = 1011$ i $B = 1101$, čije su cifre u vidu binarnih signala dovedene na ulaze sabirača, odvija se na sledeći način:

- sabirač za najniži razred (razred 0) izvrši binarno sabiranje vrednosti dovedenih na njegove ulaze, tj. $a_0+b_0+0 = 1+1+0 = 10_{(2)}$; pošto rezultat ima dve cifre, cifra 0 predstavlja rezultat sabiranja u nultom razredu i signal koji odgovara vrednosti 0 pojavljuje se na izlazu S_0, a cifra 1 prelazi u viši razred tako što se odgovarajući signal pojavljuje na izlazu C_{iz} i prosleđuje kao prenos do sabirača u sledećem razredu
- sabirač koji odgovara prvom razredu radi na isti način, tj. nalazi zbir $a_1+b_1+C_{ul} = 1+0+1 = 10_{(2)}$; na njegovim izlazima se pojavljuju vrednosti $S_1 = 0$ i $C_{iz} = 1$
- sabirač koji odgovara drugom razredu sabira $a_2+b_2+C_{ul} = 0+1+1 = 10_{(2)}$; na njegovim izlazima se pojavljuju vrednosti $S_2 = 0$ i $C_{iz} = 1$
- sabirač koji odgovara najvišem razredu sabira $a_3+b_3+C_{ul} = 1+1+1 = 11_{(2)}$; na njegovim izlazima se pojavljuju vrednosti $S_3 = 1$ i $C_{iz} = 1$; konačan rezultat sabiranja je binarni broj 11000 koji se dobija na izlazima C_{iz} i S_3 sabirača u najvišem razredu i izlazima S_2, S_1 i S_0 ostalih sabirača

4.1.6 Aritmetičko-logičke jedinice

Aritmetičko-logička jedinica (*ALU – Arithmetic Logic Unit*) je višefunkcionalna kombinaciona mreža koja realizuje skup aritmetičkih i logičkih operacija nad *n*-bitskim binarnim brojevima. Zbog svoje funkcionalnosti, ona predstavlja centralni deo svakog procesora i njene karakteristike direktno utiču na mogućnosti procesora. *ALU* se obično izrađuje kao integrisana komponenta.

Grafički simbol aritmetičko–logičke jedinice prikazan je na slici 4.26.



Slika 4.26 Aritmetičko-logička jedinica

ALU ima sledeće ulaze:

A i *B* – *n*-bitski binarni ulazi nad čijim vrednostima se izvršava operacija
C_0 – ulaz bitan za pojedine operacije
SEL – *m*-bitski upravljački ulaz

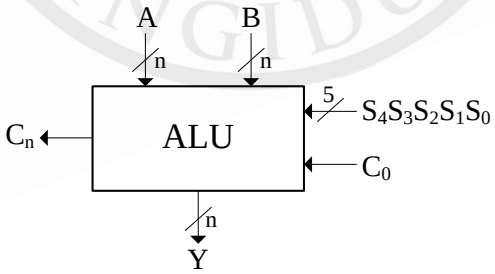
Izlazi *ALU* su:

C_n – izlaz bitan za pojedine operacije
Y – *n*-bitski binarni izlaz na kome se dobija rezultat operacije

U datom trenutku, *ALU* može da obavlja samo jednu operaciju. Izbor te operacije, koja se vrši nad binarnim brojevima dovedenim na ulaze *A* i *B*, uslovljen je signalom koji u tom trenutku postoji na upravljačkom ulazu *SEL*. Pošto se radi o *m*-bitskom ulazu, na njega je moguće dovesti 2^m različitih binarnih kombinacija, što znači da je maksimalan broj operacija koje *ALU* može da izvrši, takođe, 2^m. Ako selektovana operacija zahteva početnu vrednost nekog parametra, ta vrednost se dovodi na ulaz C_0. Rezultat operacije se dobija na izlazu *Y*, dok se izlaz C_n koristi po potrebi, ukoliko selektovana operacija to zahteva.

Opisani način rada *ALU* odnosi se samo na njenu funkcionalnost, a ne i na realizaciju samih operacija. Mreže koje realizuju operacije nalaze se unutar bloka koji predstavlja *ALU* i o njima će biti reči malo kasnije.

Rad *ALU* biće objašnjen na primeru aritmetičko-logičke jedinice sa 5-bitnim upravljačkim ulazom *SEL* ($S_4S_3S_2S_1S_0$), kao što je prikazano na slici 4.27.



Slika 4.27 Aritmetičko-logička jedinica sa pet selekcionih signala

Ova *ALU* može da izvrši 2^5=32 operacije. Uobičajeno je da su u skupu operacija podjednako zastupljene aritmetičke i logičke operacije, pa se može usvojiti da *ALU* iz primera realizuje 16 aritmetičkih i 16 logičkih operacija navedenih u tabeli 4.1.

Pri definisanju operacija treba voditi računa o oznakama operatora, jer se iste oznake (na primer „+“) koriste i za aritmetičke („+“ je oznaka za sabiranje) i za logičke operacije („+“ je oznaka za logičku *ILI* operaciju). Zato je u tabeli usvojeno da oznake *plus* i *minus* predstavljaju aritmetičke operacije sabiranja i oduzimanja.

Upravljački signal S_4 obično se koristi za selekciju tipa operacije, tj. određuje da li se radi o aritmetičkoj ili logičkoj operaciji. U primeru, ako je $S_4 = 1$ obavljaju se logičke operacije, a ako je $S_4 = 0$, aritmetičke (videti tabelu). Nakon izbora tipa operacije, potrebno je selektovati konkretnu operaciju iz odabranog skupa (skupa aritmetičkih ili skupa logičkih operacija). To se obavlja pomoću signala $S_3S_2S_1S_0$. Ova četiri signala mogu da formiraju $2^4 = 16$ različitih kombinacija što je dovoljno za selekciju operacija u odabranom skupu.

Neka su na ulaze *ALU* dovedeni binarni signali $A = 1001$ i $B = 1100$, kao i upravljački signal $SEL = 10001$. Na osnovu signala *SEL* može se zaključiti da *ALU* treba da izvrši logičku operaciju $\overline{A+B}$ (iz $S_4 = 1$ sledi da je operacija logička, a kombinacija 0001 određuje samu operaciju). Kao rezultat, na izlazu *Y* će se pojaviti vrednost 0010.

S_3	S_2	S_1	S_0	$S_4 = 1$	$S_4 = 0$
0	0	0	0	$Y = \overline{A}$	$Y = A$ *plus* C_0
0	0	0	1	$Y = \overline{A+B}$	$Y = (A+B)$ *plus* C_0
0	0	1	0	$Y = \overline{A}B$	$Y = (A+\overline{B})$ *plus* C_0
0	0	1	1	$Y = 0000$	$Y = 0$ - C_0
0	1	0	0	$Y = \overline{AB}$	$Y = A$ *plus* $(A\overline{B})$ *plus* C_0
0	1	0	1	$Y = \overline{B}$	$Y = (A+B)$ *plus* $(A\overline{B})$ *plus* C_0
0	1	1	0	$Y = A \oplus B$	$Y = A$ *minus* B *minus* $\overline{C}_0$
0	1	1	1	$Y = A\overline{B}$	$Y = (A\overline{B})$ *minus* $\overline{C}_0$
1	0	0	0	$Y = \overline{A}+B$	$Y = A$ *plus* (AB) *plus* C_0
1	0	0	1	$Y = \overline{A \oplus B}$	$Y = A$ *plus* B *plus* C_0
1	0	1	0	$Y = B$	$Y = (A+\overline{B})$ *plus* (AB) *plus* C_0
1	0	1	1	$Y = AB$	$Y = (AB)$ *minus* C_0
1	1	0	0	$Y = 1111$	$Y = A$ *plus* A *plus* C_0
1	1	0	1	$Y = A+\overline{B}$	$Y = (A+B)$ *plus* A *plus* C_0
1	1	1	0	$Y = A+B$	$Y = (A+\overline{B})$ *plus* A *plus* C_0
1	1	1	1	$Y = A$	$Y = A$ *minus* C_0

Tabela 4.1 Moguć skup operacija za *ALU* sa pet selekcionih ulaza

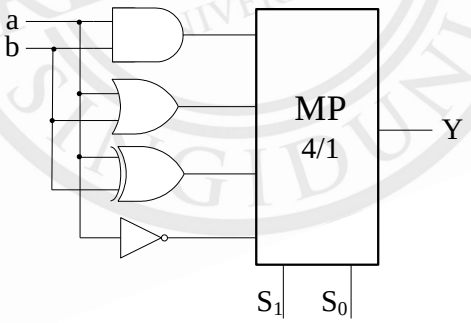
U nastavku će biti reči o realizaciji operacija unutar bloka koji predstavlja *ALU*. S obzirom da realizacija zavisi od broja i vrste operacija koje *ALU* može da izvrši, ona može biti vrlo složena. Međutim, princip realizacije je jednostavan i biće objašnjen na primeru *ALU* koja obavlja osnovne logičke operacije (*I*, *ILI*, *EKSILI* i *NE*) nad jednobitnim binarnim brojevima *a* i *b*.

Za realizaciju *ALU* sa četiri logičke operacije dovoljno je koristiti 2-bitni upravljački signal *SEL* (S_1S_0). Neka su vrednosti ovog signala koje odgovaraju pojedinim operacijama date u tabeli 4.2.

Ukoliko se signali S_1 i S_0 iskoriste kao selekcioni signali multipleksera 4/1, *ALU* se može realizovati tako što se izlaz multiplesera proglasi izlazom *ALU Y*, a na svaki od ulaza multipleksera (prema tabeli 4.2) dovede izlaz prekidačke mreže koja realizuje konkretnu operaciju (videti sliku 4.28).

S_1	S_0	Y
0	0	ab
0	1	$a+b$
1	0	$a \oplus b$
1	1	$\bar{a}$

Tabela 4.2 Skup osnovnih operacija *ALU* u primeru



Slika 4.28 Realizacija *ALU* koja obavlja osnovne logičke operacije

U primeru, prekidačke mreže koje realizuju operacije su vrlo jednostavne jer se sastoje samo od jednog logičkog kola koje odgovara operaciji. U slučaju većeg broja složenijih operacija i prekidačke mreže su složenije. Ovakvim načinom realizacije, kada se na ulaze *ALU* (ulaze *a* i *b* u primeru) dovedu signali koji odgovaraju binarnim brojevima nad kojima treba izvršiti operaciju, na ulazima multipleksera dobijaju se rezultati svih operacija. Međutim, na izlaz multipleksera (tj. izlaz *ALU*) prosleđuje se rezultat samo one operacije koja je selektovana upravljačkim signalima S_1 i S_0.

4.2. Standardni sekvencijalni moduli

Standardni sekvencijalni moduli su moduli koji, osim logičkih, sadrže i memorijske elemente. U ovom poglavlju su predstavljena tri važna modula ovog tipa: registri, brojači i memorije. Za svaki modul dati su njegova funkcionalnost, osnovne karakteristike i situacije u kojima se modul koristi.

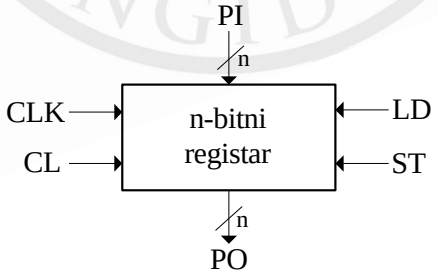
4.2.1 Registri

Registri su sekvencijalne mreže koje se u računarima i drugim digitalnim sistemima koriste za pamćenje binarnih podataka, tj. binarnih reči. Veličina registra zavisi od dužine binarne reči (broja bitova) koja se u njemu čuva (8-bitni, 16-bitni, 32-bitni, itd. registri). Binarne reči se po potrebi mogu upisivati u registar, ili se iz njega čitati. U zavisnosti od načina upisa i čitanja, razlikuju se dve vrste registara:

- registri sa *paralelnim* upisom i čitanjem (paralelni registri)
- registri sa *serijskim* upisom i čitanjem (serijski registri)

Paralelni registri

Kod paralelnih registara, svi biti binarne reči upisuju se u registar *istovremeno*, tj. u jednom taktu. Takođe, prilikom čitanja sadržaja registra, svi biti binarne reči se čitaju istovremeno. Grafički simbol paralelnog registra dat je na slici 4.29.



Slika 4.29 Paralelni registar

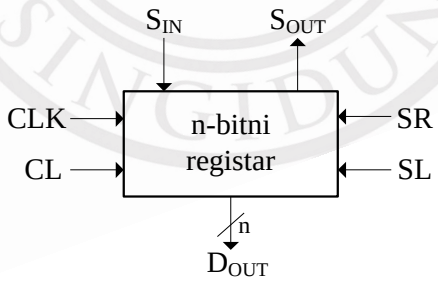
PI (*Parallel Inputs*) predstavlja informacioni paralelni ulaz na koji se dovodi *n*-bitska binarna reč koju treba upisati u registar. *PO* (*Parallel Outputs*) je informacioni paralelni izlaz na kome se pojavljuje *n*-bitska binarna reč koja je pročitana iz registra. Upravljački ulaz *LD* (*Load*) dozvoljava upis sadržaja sa *PI* u

registar. S obzirom da se često javlja potreba za brisanjem sadržaja registra, tj. postavljanjem *n*-bitske binarne reči koja se sastoji samo od nula u registar, izdvojen je poseban ulaz *CL* (*Clear*) koji to omogućava. Osim ovog ulaza, ponekad se koristi i ulaz *ST* (*Set*) koji podržava upis jedinice u sve razrede registra. Za rad registra neophodan je i takt koji obezbeđuje sinhronizaciju prilikom obavljanja operacija upisa i čitanja. Takt se dovodi na ulaz *CLK* (*Clock*).

Serijski registri

Serijski ili *pomeračkі registar* omogućava pomeranje binarne reči koja se u njemu nalazi za jedno mesto od ulaza ka izlazu. U zavisnosti od načina pomeranja, postoje različite vrste pomeračkih registara. Kod registara sa pomeranjem udesno, bit iz najnižeg razreda se izbacuje iz registra, a svi ostali biti se pomeraju za jednu poziciju niže. U najviši razred, koji ostaje prazan, upisuje se novi bit sa serijskog ulaza. Na sličan način rade i serijski registri sa pomeranjem ulevo. Kod njih se bit najveće težine izbacuje iz registra, a ostali biti se pomeraju za jedno mesto ulevo. Na mesto bita najmanje težine upisuje se novi bit sa ulaza. Oba pomenuta registra se mogu realizovati i tako da operacija pomeranja predstavlja rotaciju, tj. da se bit koji je izbačen iz registra upisuje na mesto novog bita. Osim opisane, pomeračkі registri mogu imati i dodatnu funkcionalnost o kojoj ovde neće biti reči, kao na primer, pomeranje u oba smera (bidirekcioni registri), paralelni upis binarne reči i sl.

Grafički simbol pomeračkog registra prikazan je na slici 4.30.



Slika 4.30 Serijski registar

S_{OUT} (*Serial Output*) je jednobitski serijski informacioni izlaz na kome se pojavljuje bit koji je pročitan i izbačen iz registra prilikom pomeranja sadržaja. S_{IN} (*Serial Input*) predstavlja informacioni jednobitskii serijski ulaz na koji se dovodi bit koji treba uneti u registar. Informacioni izlaz D_{OUT} (*Data Output*) omogućava čitanje trenutnog sadržaja registra, tj. *n*-bitske reči koja se u registru nalazi u tom trenutku. I kod pomeračkog, kao i kod paralelnog registra, postoje ulazi *CL* i *CLK* koji imaju ranije opisanu ulogu. Serijski registar, takođe, ima i ulaze *SR* (*Shift*

Right) i *SL* (*Shift Left*) na koje se dovode signali koji određuju smer pomeranja sadržaja registra (za jedno mesto udesno - *SR*, odnosno ulevo - *SL*).

Pomerački registri se koriste u slučajevima kada postoji potreba za serijskim prijemom ili slanjem podataka (bit po bit). Serijski prenos je znatno sporiji od paralelnog, ali je zato jeftiniji (koristi manji broj linija za prenos). Zbog broja linija, verovatnoća greške pri serijskom prenosu je manja nego pri paralelnom. To ga čini pogodnim za primenu u lošijim uslovima, na primer u industrijskim halama gde ima buke, prašine, vibracija i dr.

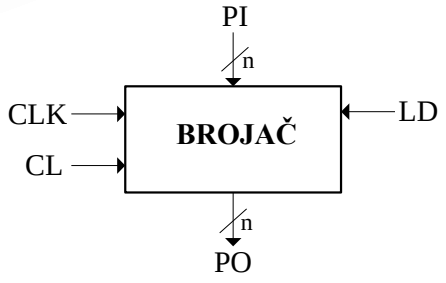
4.2.2 Brojači

Brojači su sekvencijalne mreže koje se u digitalnim sistemima i uređajima koriste za brojanje različitih događaja i merenje vremenskih intervala. Oni rade tako što broje promene nekog signala dovedenog na njihov ulaz i na izlazu generišu binarni broj koji odgovara broju promena učinjenih do posmatranog trenutka. Obično je reč je o taktovanim mrežama, što znači da se stanje na njihovom izlazu menja samo po nailasku takta. U zavisnosti od načina brojanja, razlikuju se:

- *inkrementirajući* brojači koji broje unapred, tj. u rastućem redosledu brojanja
- *dekrementirajući* brojači koji broje unazad, tj. u opadajućem redosledu brojanja

Broj različitih binarnih brojeva koje brojač može da generiše na svom izlazu naziva se *moduo* brojača. Inkrementirajući brojač po modulu *m* radi tako što, u skladu sa taktom, generiše binarne brojeve od 0 do *m*-1 (ukupno *m* brojeva), a zatim se resetuje i ponovo počinje da broji od 0. Dekrementirajući brojač po modulu *m* generiše binarne brojeve od *m*-1 do 0, a zatim se vraća u stanje *m*-1 i nastavlja sa radom.

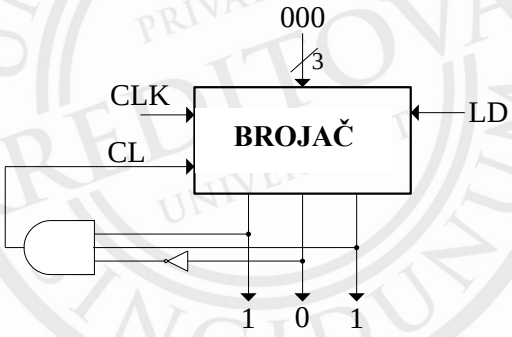
Grafički simbol brojača prikazan na slici 4.31.



Slika 4.31 Brojač

Na ulaz *PI* (*Parallel Inputs*) dovodi se *n*-bitska binarna reč koja predstavlja početno stanje od koga brojač počinje da broji. Upravljački ulaz *LD* (*Load*) omogućava upis početnog stanja sa ulaza *PI* u brojač. Na izlazu *PO* (*Parallel Outputs*) generiše se *n*-bitska binarna reč koja predstavlja trenutno stanje brojača. Za resetovanje, tj. brisanje sadržaja brojača upisom *n*-bitske binarne reči koja se sastoji samo od nula, izdvojen je poseban ulaz *CL* (*Clear*). Za rad brojača neophodan je i takt koji omogućava promenu stanja brojača i dovodi se na ulaz *CLK* (*Clock*). Kod ovog brojača ne postoji mogućnost eksplicitnog zadavanja tipa brojača (inkrementirajući ili dekrementirajući), jer je taj izbor već ugrađen u blok koji predstavlja unutrašnjost brojača.

Brojač predstavljen grafičkim simbolom broji po modulu 2^n, što je uslovljeno dužinom binarne reči koja se može smestiti u brojač (n). Međutim, često se javlja potreba za brojačima čiji moduo nije stepen dvojke (na primer, brojač po modulu 6, 10 i sl.). Da bi se realizovao brojač po željenom modulu, potrebno je dodati *eksternu (spoljašnju) logiku* koja bi to omogućila. Uloga eksterne logike će biti objašnjena na primeru brojača po modulu 6, čija je realizacija data na slici 4.32.



Slika 4.32 Realizacija brojača po modulu 6

Brojač po modulu 6 na svom izlazu generiše brojeve od 0 do 5, tj. u binarnom obliku od $000_{(2)}$ do $101_{(2)}$. Stoga je dovoljno koristiti 3-bitski brojač na čijem se ulazu *PI* i izlazu *PO* pojavljuju 3-bitske binarne vrednosti. Pošto početna vrednost nije eksplicitno zadata, može se usvojiti da brojač počinje da broji od 0, pa je na ulaz *PI* dovedena vrednost 000. Ova vrednost se upisuje u brojač aktiviranjem ulaza *LD*. Sa dovođenjem signala takta na ulaz *CLK*, brojač počinje da broji u rastućem redosledu. Pri svakom narednom impulsu takta, vrednost brojača (na izlazu *PO*) se uvećava za 1. U trenutku kada brojač stigne do broja 5, potrebno ga je vratiti na 0, što se može postići brisanjem njegovog sadržaja, tj. aktiviranjem ulaza *CL*. Signal za aktiviranje ulaza *CL* generiše se od strane eksterne logike. Naime, ova logika je napravljena tako da registruje trenutak kada se na linijama izlaza *PO* pojavi kombinacija koja odgovara najvećem broju do kojeg brojač može

da stigne (moduo umanjen za 1), što je u ovom primeru 101. U tom trenutku, logičko *I* kolo na svom izlazu generiše vrednost 1 koja se prosleđuje na ulaz *CL* i brojač se resetuje. Da bi logičko *I* kolo proizvelo ovaj signal, na njegove ulaze su direktno dovedeni signali sa linija *PO* izlaza na kojima kombinacija ima vrednost 1, a preko invertora sa linija na kojima kombinacija ima vrednost 0.

4.2.3 Memorije

Memorije su komponente koje se koriste za pamćenje velikog broja binarnih reči. Idealna memorija bi trebalo da ima sledeće karakteristike:

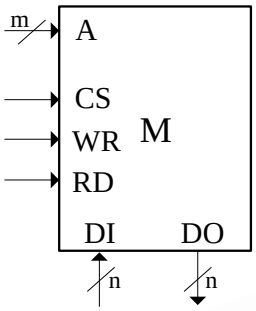
- visoku gustinu pakovanja koja direktno utiče na kapacitet memorije
- trajno čuvanje podataka
- kratko vreme upisa i čitanja podataka, što bitno utiče na brzinu rada
- veliki broj upisa pre otkaza memorije, što utiče na pouzdanost
- nisku potrošnju električne energije
- nisku cenu

S obzirom da je u praksi nemoguće ispuniti sve ove zahteve istovremeno, razvijeno je više vrsta memorija koje zadovoljavaju pojedine karakteristike u većoj ili manjoj meri. Izbor memorije koja će se koristiti zavisi od konkretnih potreba. Po opštoj podeli, memorije mogu biti:

- nepermanentne
- permanentne

Nepermanentne memorije su memorije čiji se *sadržaj gubi* u slučaju prestanka napajanja (isključenja sistema, nestanka struje i sl.). Najčešće korišćena nepermanentna memorija je memorija sa ravnopravnim pristupom *RAM* (*Random Access Memory*). Naziv ove memorije potiče od činjenice da kod nje vreme potrebno za upis i čitanje binarne reči ne zavisi od mesta na kome se binarna reč nalazi u memoriji.

RAM memorija se sastoji od memorijske matrice sa 2^m ćelija (ovaj broj ćelija predstavlja kapacitet memorije), pri čemu u svaku ćeliju može da se smesti jedna *n*-bitska reč. Svakoj ćeliji je pridružen broj, tj. adresa. Pristup ćeliji omogućava adresni dekoder koji takođe ulazi u sastav *RAM* memorije. Kada se neka adresa u binarnom obliku dovede na ulaz dekodera, na odgovarajućoj izlaznoj liniji pojavljuje se aktivna vrednost signala čime se adresirana ćelija selektuje za upis ili čitanje. Na slici 4.33 dat je grafički simbol *RAM* memorije.



Slika 4.33 RAM memorija

Kada se na ulaz *A* (*Address*) dovede adresa, dekoder (koji se nalazi unutar bloka) selektuje odgovarajuću ćeliju. Ukoliko u tom trenutku na ulazima *WR* (*Write*) i *CS* (*Chip Select*) postoje aktivne vrednosti signala (na primer 1), u selektovanu ćeliju se upisuje podatak koji trenutno postoji na ulazu *DI* (*Data Input*). Međutim, ako aktivne vrednosti signala postoje na ulazima *RD* (*Read*) i *CS*, iz selektovane ćelije će biti pročitana binarna reč koja će se pojaviti na izlazu *DO* (*Data Output*). Čitanjem se ne menja binarna reč u selektovanoj ćeliji. Operacije čitanja i upisa se ne mogu obavljati istovremeno. Da bi se to osiguralo, memorije obično imaju jedinstven ulaz za signale čitanja i upisa, pa jedna binarna vrednost (na primer 0) predstavlja aktivni signal za čitanje, a druga binarna vrednost (1) aktivni signal za upis. Na ovaj način postaje nemoguće da oba signala budu aktivna u istom trenutku.

RAM memorije se mogu realizovati na različite načine, što zavisi od primenjenje tehnologije. Za njihovo konstruisanje, osim logičkih i memorijskih elemenata, mogu se koristiti i elektronske komponente, na primer kondenzatori. U skladu sa ovim, uvedena je podela *RAM* memorija na:

- statičke – *SRAM* (*Static RAM*)
- dinamičke – *DRAM* (*Dynamic RAM*)

Statičke memorije su napravljene od bistabilnih memorijskih elementa koji omogućavaju pamćenje sadržaja sve dok se on ne promeni, ili dok se ne isključi napajanje (reč je o nepermanentnim memorijama). Njihove osnovne karakteristike su: mala gustina pakovanja, velika brzina rada (pošto je vreme pristupa vrlo kratko i reda je *ns*), mala verovatnoća greške, mala potrošnja električne energije, visoka cena nabavke.

Dinamičke memorije sadrže memorijske elemente koji imaju kondenzatore, tako da se njihov sadržaj mora povremeno osvežavati. Upravo zbog stalnog osvežavanja, u naziv ovih memorija uveden je termin „dinamička“. Proces osvežavanja je vrlo brz (reda *ns*), u potpunosti se obavlja u hardveru memorije (na

svakih par *ms*) i neprimetan je za korisnika. Karakteristike dinamičkih memorija su: jednostavna proizvodnja, velika gustina pakovanja, mala potrošnja električne energije, niska cena nabavke, mala brzina rada, manja pouzdanost, tj. veća verovatnoća greške.

U tabeli 4.3 dat je uporedni pregled razmatranih vrsta memorija po zadatim osobinama.

Osobina	Statička memorija	Dinamička memorija
brzina	znatno veća	
kapacitet po čipu		znatno veći
potrošnja	troši manje el.energije	
cena		niža
verovatnoća greške	pouzdanija	
broj upisa pre otkaza	praktično beskonačan	praktično beskonačan

Tabela 4.3 Poređenje statičkih i dinamičkih memorija

Permanentne memorije su memorije kod kojih gubitak napajanja ne utiče na njihov trenutni sadržaj. Za razliku od nepermanentnih memorija koje su čitajuće/upisne (*RW – Read/Write*), u permanentne memorije obično se ne može upisivati u toku rada sistema, već je moguće samo iz njih čitati sadržaj. Takve memorije se nazivaju fiksnim memorijama, ili *ROM* (*Read Only Memory*).

Način upisa podataka u *ROM* zavisi od korišćene tehnologije. Najjednostavniji postupak je upis sadržaja prilikom proizvodnje čipa. Ovako upisan sadržaj se ne može menjati.

Veću fleksibilnost imaju programabilne *ROM*, tj. *PROM* (*Programmabile ROM*) u koje sam korisnik može da upiše sadržaj koji se kasnije, takođe, ne može menjati.

Najveću fleksibilnost imaju *EPROM* (*Erasable PROM*) koje, osim upisa, imaju i mogućnost brisanja sadržaja. Brisanje se obavlja tako što se kroz mali stakleni prozor koji se nalazi na kućištu memorije propusti ultraljubičasta svetlost koja osveli unutrašnjost čipa u trajanju od dvadesetak minuta. Nakon toga, čip je prazan i može se ponovo programirati. Ipak, jedan čip se može izbrisati samo nekoliko desetina puta, nakon čega postaje neupotrebljiv.

Noviji oblik permanentne memorije je *fleš* (*flash* – munja), čiji naziv asocira na veliku brzinu pristupa. Ova memorija koristi tehnologiju elektronskog upisa i brisanja podataka, pri čemu se pristupa samo blokovima podataka (ne može se pristupati pojedinačnim bajtovima). Čitanje se obavlja bajt po bajt. Karakteristike ove memorije su: velika brzina čitanja, veliki kapacitet i velika pouzdanost.

Vežbanja

P1. Šta su koderi i čemu služe? Koje vrste kodera postoje i u čemu se one razlikuju? Kada koder ispravno radi? Objasniti.

P2. Šta su dekoderi i čemu služe? Objasniti princip rada dekodera. Koje vrste dekodera postoje i u čemu se one razlikuju?

P3. Objasniti princip rada multipleksera. Kakava veza postoji između broja informacionih i broja selekcionih ulaza multipleksera?

P4. Šta je demultiplekser i koja je njegova uloga? Nacrtati grafički simbol i objasniti princip rada demultipleksera.

P5. Šta je sabirač? Koje vrste sabirača postoje i u čemu se one razlikuju? Dati grafički prikaz potpunog sabirača. Kako se realizuje sabiranje dva binarna broja sa po *n* cifara pomoću potpunih sabirača? Objasniti.

P6. Šta je aritmetičko-logička jedinica i čemu služi? Nacrtati grafički simbol *n*-bitske *ALU* i objasniti princip njenog rada kroz funkcionalnost njenih ulaza i izlaza.

P7. Šta su funkcije izlaza logičke mreže (kodera, dekodera, itd.)?

P8. Šta su registri i čemu služe? Koje vrste registara postoje i u čemu se one razlikuju?

P9. Dati grafički prikaz *n*-bitskog paralelnog registra i objasniti princip rada kroz funkcionalnost njegovih ulaza i izlaza.

P10. Dati grafički prikaz *n*-bitskog serijskog registra i objasniti princip njegovog rada kroz funkcionalnost njegovih ulaza i izlaza. Kako se drugačije naziva serijski registar?

P11. Šta su brojači i čemu služe? Šta je moduo brojača? Koje vrste brojača postoje i u čemu se one razlikuju? Nacrtati grafički simbol *n*-bitskog brojača i objasniti princip njegovog rada kroz funkcionalnost njegovih ulaza i izlaza.

P12. Čemu služe memorije? Navesti karakteristike idealne memorije. Koja je osnovna podela memorija?

P13. Objasniti razliku između permanentne i nepermanentne memorije.

P14. Šta je *RAM* memorija? Nacrtati grafički simbol *RAM* memorije i na osnovu njega objasniti kako se podatak upisuje, a kako čita iz memorije.

P15. Navesti vrste *RAM* memorija u zavisnosti od njihove realizacije. Ukratko opisati svaku od njih.

P16. Dati uporedni prikaz statičkih i dinamičkih *RAM* memorija po sledećim karakteristikama: brzini, kapacitetu, potrošnji, ceni, verovatnoći greške i broju upisa pre otkaza.

P17. Šta su *ROM*, *PROM* i *EPROM*? Navesti njihove mogućnosti.

P18. Šta je *flash* memorija? Navesti njene osnovne karakteristike.

Z1. Realizovati potpuni koder 8/3 (dati kombinacionu tablicu, funkcije izlaza i način realizacije pomoću logičkih kola).

Z2. Ako potpuni koder ima *m* ulaza i *n* izlaza, kakva veza postoji između *m* i *n*? Koliko izlaza ima potpuni koder sa četiri ulaza?

Z3. Realizovati potpuni dekoder 2/4 (dati kombinacionu tablicu, funkcije izlaza i način realizacije pomoću prekidačke mreže).

Z4. Ako potpuni dekoder ima *m* ulaza i *n* izlaza, kakva veza postoji između *m* i *n*? Koliko ulaza ima potpuni dekoder sa osam izlaza?

Z5. Primenom dekodera realizovati zadate logičke funkcije:

a) $Y = A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} + \overline{A}B\overline{C}$

b) $Y = \overline{A}\overline{B}C + A\overline{B}\overline{C} + AB\overline{C}$

c) $Y = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC + AB\overline{C}$

Objasniti postupak realizacije.

Z6. Realizovati multiplekser 8/1 (dati kombinacionu tablicu, funkcije izlaza i način realizacije pomoću logičkih kola).

Z7. Koliko informacionih ulaza i izlaza ima multiplekser sa četiri selekciona ulaza?

Z8. Primenom multipleksera realizovati zadate logičke funkcije:

a) $Y = \overline{A}B\overline{C} + A\overline{B}\overline{C} + \overline{A}\overline{B}C + AB\overline{C}$
b) $Y = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}BC + \overline{A}B\overline{C}$
c) $Y = \overline{A}\overline{B}C + \overline{A}BC + AB\overline{C}$

Objasniti postupak realizacije.

Z9. Realizovati demultiplekser 1/4 (dati kombinacionu tablicu, funkcije izlaza i način realizacije pomoću logičkih kola).

Z10. Koliko informacionih ulaza i izlaza ima demultiplekser sa četiri selekciona ulaza?

Z11. Pomoću potpunih sabirača realizovati 3-bitni sabirač. Objasniti postupak realizacije.

Z12. Realizovati 1-bitnu ALU koja može nad dva jednobitna binarna broja *a* i *b* da obavi zadate logičke operacije:

a) $\bar{a}$, $a \oplus b$ i ab
b) $\bar{a}$, $\bar{a} \oplus b$, $a+b$ i ab
c) $\bar{a} \oplus b$, $\bar{a}+b$ i ab

Objasniti postupke realizacije.

Z13. Realizovati brojač po modulu:

a) 7
b) 15
c) 24

Objasniti postupak realizacije.

UNIVERZITET
PRIVATNI
AKREDITOVANI
UNIVERZITET
SINGIDUNUM

5 Komponente računara

Računar je digitalni elektronski uređaj koji služi za rešavanje raznovrsnih problema putem obrade podataka. Koristi se kada postoji potreba da se ulazni podaci, koji predstavljaju polazno stanje problema, obrade po određenom postupku i generišu izlazni podaci koji predstavljaju rešenje tog problema. Iako podaci mogu biti u različitim formatima (tekst, numeričke vrednosti, odmerci signala, itd.), oni se u računaru prevode u binarni oblik (nizove nula i jedinica), jer samo tako mogu biti obrađivani. U binarnom obliku su, osim podataka, i upravljački signali, instrukcije, itd. Stoga je neophodno da u svakom trenutku računarski sistem zna šta svaka binarna reč tačno predstavlja i kako treba da se interpretira.

5.1 Osnovni pojmovi

Obrada podataka podrazumeva izvršavanje niza *operacija* na koje se može razložiti postupak rešavanja problema. Operacije se izvršavaju nad binarnim rečima. Postoje *unarne* i *binarne* operacije. Unarne su one operacije koje se izvršavaju nad jednom binarnom reči, a binarne nad dvema. Operacije se realizuju pomoću kombinacionih i sekvencijalnih mreža od kojih su neke opisane u prethodnom poglavlju (registri, memorije, sabirači, itd.).

U računaru, operacije se predstavljaju pomoću *instrukcija* (naredbi, komandi), koje su, kao i podaci, u binarnom obliku. Instrukcija se izvršava nad podacima koji se nazivaju *operandi*. Na primer, brojevi koje treba sabrati su operandi za operaciju sabiranja.

Uređeni niz instrukcija se naziva *program*. Program precizno definiše sve aktivnosti u okviru obrade, kako po njihovom sadržaju, tako i po redosledu izvođenja. Svi programi u računaru se jednim imenom nazivaju *softver* (*software*). Programi se mogu svrstati u dve kategorije: *sistemski* i *aplikativni* softver. Sistemskim softverom se obično nazivaju programi koje proizvođači računara isporučuju zajedno sa računarom. To su najčešće programi bez kojih računar ne može da radi (operativni sistem), kao i neki uslužni programi koje korisnici primenjuju nezavisno od problema kojim se bave (prevodioci, sistemi za upravljanje fajlovima, i sl.). Aplikativni softver čine programi koje koriste razne kategorije korisnika za potrebe rešavanja svojih problema (tekst editori, igrice, programi za crtanje, namenski programi, itd.). Strogu granicu između sistemskog i aplikativnog softvera ponekad je teško definisati.

Za izvršavanje programa zadužen je *hardver* koga čine fizičke komponente računara. Izborom programa koji će se izvršiti menja se način obrade podataka, dok hardver računara ostaje neizmenjen. Osnovne hardverske komponente računara su:

- *ulazno/izlazni uređaji*, koji omogućavaju unos ulaznih podataka u računar, kao i prikaz rezultata obrade
- *memorija*, koja služi za čuvanje podataka i programa
- *procesor*, koji obavlja obradu podataka izvršavanjem programskih instrukcija
- *magistrala*, koja služi za povezivanje hardverskih komponenata i razmenu sadržaja između njih

Da bi uneo podatke i programe u računar, korisnik mora da pristupi nekoj od jedinica za ulaz koje računar poseduje. To su obično tastatura ili miš. Rezultate dobija putem izlazne jedinice, na primer monitora, štampača, plotera i sl. Dakle, računar ostvaruje komunikaciju sa okruženjem (na primer, sa korisnikom ili sa drugim računarima) preko ulazno/izlaznih jedinica.

Podaci i programi koje korisnik unese u računar smeštaju se u operativnu memoriju. Ona komunicira sa ostalim delovima računara, pri čemu ima najživlju komunikaciju sa procesorom. S obzirom da je operativna memorija brza i zbog toga skupa, ona je ograničenog, relativno malog kapaciteta. Zato se svi podaci i programi sa kojima računar trenutno ne radi smeštaju u eksternu, tj. spoljašnju memoriju koja je znatno sporija, ali ima mnogo veći kapacitet od operativne memorije. Primeri eksterne memorije su hard disk, fleš memorija, *DVD, CD* i sl.

Procesor izvršava program instrukciju po instrukciju i to onim redosledom kojim su instrukcije navedene u programu, sve dok ne dođe do kraja programa. Ovaj postupak se može predstaviti ciklusima koji sadrže sledeća četiri koraka:

- dohvatanje instrukcije iz memorije

- dohvatanje operanada, ukoliko to zahteva instrukcija
- izvršavanje instrukcije
- upis rezultata u memoriju ili slanje na neku izlaznu jedinicu ako se to u instrukciji zahteva

Kao što se vidi iz navedenih koraka, između procesora i memorije odvija se vrlo intenzivna komunikacija, jer je potreba za čitanjem instrukcija i operanada iz memorije, kao i upisivanjem rezultata u nju vrlo česta. Prenos instrukcija i podataka između procesora i memorije obavlja se preko bidirekcionih (dvosmernih) veza koje čine magistralu. Izbor memorijske lokacije iz koje treba uzeti podatak ili u koju treba upisati podatak određen je adresom koju generiše procesor. Adresa se šalje memoriji preko unidirekcionih (jednosmernih) veza koje su, takođe, deo megistrale. Prenos po magistrali je paralelan, što znači da se svi biti memorijske reči prenose istovremeno. Širina magistrale po kojoj se prenose adrese, u većini slučajeva, određuje kapacitet operativne memorije. Tako, ako adresa ima n bitova, pomoću nje se može adresirati $C = 2^n$ različitih memorijskih lokacija, pa C predstavlja kapacitet memorije.

Upis binarnog podatka u memoriju obavlja se u sledećim koracima:

- procesor adresira željenu memorijsku lokaciju postavljanjem njene adrese na adresnu magistralu
- procesor postavlja podatak koji treba da se upiše na magistralu podataka
- procesor šalje komandu memoriji da obavi upis podatka

Postupak čitanja binarnog podatka iz memorije odvija se na sledeći način:

- procesor adresira memorijsku lokaciju iz koje želi da pročita podatak postavljanjem njene adrese na adresnu magistralu
- procesor šalje komandu memoriji da na magistralu podataka postavi adresirani podatak
- procesor preuzima podatak sa magistrale podataka

Pošto je procesor odgovoran za izvršavanje instrukcija, on mora da ima mogućnost lokalnog čuvanja izvesne količine podataka. Za to se koriste brzi registri koji imaju različite namene. Na primer, postoji registar u kome se čuva adresa naredne instrukcije koju treba dohvatiti iz memorije, zatim registar sa instrukcijom koju treba izvršiti, registri u koje se smeštaju operandi, registri opšte namene, itd.

Da bi dohvatio neku instrukciju, procesor mora da ima informaciju o tome u kojoj lokaciji u memoriji se ona nalazi. Na početku rada, adresa prve instrukcije

programa se hardverski unosi u registar procesora u kome se čuva adresa naredne instrukcije. Pošto se ovaj registar obično realizuje kao brojač, adresa svake sledeće instrukcije programa dobija se automatski inkrementiranjem brojača. Na osnovu poznate adrese, iz memorije se doprema odgovarajuća instrukcija i smešta u registar za instrukcije, nakon čega sledi njeno izvršavanje.

Izvršavanje instrukcije se svodi na obavljanje onih operacija koje su njome specificirane. Da bi instrukcija mogla da se izvrši, neophodno je prethodno obezbediti raspoloživost operanada bitnih za njeno izvršenje. Stoga, instrukcija u svom formatu mora da specificira, osim operacije koju procesor treba da obavi, i podatke nad kojima se operacija obavlja (ili adrese na kojima se ti podaci nalaze). Po izvršenju instrukcije, rezultat se po potrebi prenosi u operativnu memoriju, ili se prosleđuje nekoj izlaznoj jedinici.

Poznavanje računara se može posmatrati sa dva aspekta: sa aspekta arhitekture i sa aspekta organizacije računara.

Arhitektura računara obuhvata sve ono što je potrebno znati o računaru da bi za njega mogli da se pišu i na njemu izvršavaju programi. Može se reći da arhitekturu računara čini skup atributa koji je vidljiv za programera. Da bi programer napisao program koji može da se izvrši na računaru, treba da poznaje, na primer, tipove podataka, formate instrukcija, instrukcijski set, načine adresiranja, skup programski dostupnih registara, itd.

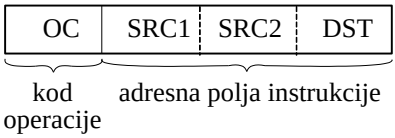
Organizacija računara se bavi načinima realizacije pojedinih delova računara. To je skup atributa transparentan (nevidljiv) za programera. Organizaciji pripadaju kontrolni signali, korišćene memorijske tehnologije, tehnike realizacije upravljačkih jedinica, tehnike realizacije keš memorija, veze računara sa periferijama i dr.

U narednim poglavljima biće detaljnije opisani pojedini atributi kako arhitekture, tako i organizacije računara.

5.2 Memorija

Memorija predstavlja skup lokacija (ćelija) u kojima se čuva binarni sadržaj. Grupisanjem jedne ili više lokacija dobija se *memorijska reč* kojoj se može pristupiti preko *m*-bitske adrese. Posmatranjem memorijske reči ne može se zaključiti da li ona predstavlja instrukciju ili operand, ali računar uvek zna kako treba da je protumači, tj. interpretira.

Binarnu reč koja predstavlja instrukciju, računar interpretira tako što je podeli u dva dela kao na slici 5.1.



Slika 5.1 Jedna interpretacija instrukcije

Prvi deo instrukcije je *kod operacije* (*OC – Operation Code*) koji specificira vrstu operacije, na primer, sabiranje, oduzimanje, itd. Drugi deo instrukcije sadrži *adresna polja instrukcije*. U zavisnosti od vrste operacije, broj polja u adresnom delu može biti različit. To mogu biti polja izvorišnih (*SRC – Source*) i odredišnog (*DST – Destination*) operanda. Polja *izvorišnih operanada* (*SRC1* i *SRC2*) sadrže adrese lokacija u kojima se nalaze operandi nad kojima se operacija izvršava, ili same operande. Polje *odredišnog operanda* (*DST*) uvek sadrži adresu lokacije u koju treba smestiti rezultat operacije.

Binarnu reč koja predstavlja operand, računar može da interpretira na različite načine u zavisnosti od tipa operanda. Na slici 5.2 predstavljen je *n*-bitski operand koji se može interpretirati kao: neoznačeni ceo broj, označeni ceo broj u drugom komplementu, veličina u pokretnom zarezu, niz alfanumeričkih znakova, itd.

Slika 5.2 Operand

5.3 Procesor

Procesor (*CPU – Central Processing Unit*) je osnovna komponenta računara namenjena obradi podataka. Centralni deo procesora je aritmetičko-logička jedinica (*ALU*) koja služi za izvršavanje instrukcija programa. Osim *ALU*, u procesoru se nalazi i određen broj *procesorskih registara* sa različitim namenama u kojima se privremeno čuvaju male količine podataka.

Da bi *ALU* mogla da izvrši instrukciju, potrebno je najpre da se ta instrukcija prenese iz memorije u procesor. Adresa memorijske lokacije u kojoj je smeštena instrukcija nalazi se u programskom brojaču (*PC – Program Counter*). *Programski brojač* je procesorski registar u kome se čuva adresa naredne instrukcije koju treba izvršiti. *PC* se realizuje kao inkrementirajući brojač zato što se instrukcije programa obično čuvaju u memoriji na uzastopnim adresama.

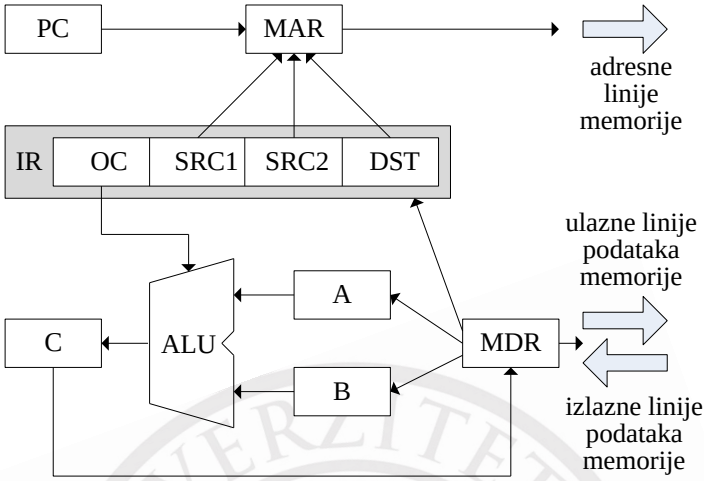
Procesor ostvaruje komunikaciju sa memorijom samo preko dva svoja registra:

- *adresnog registra memorije* (*MAR – Memory Address Register*), koji sadrži adresu memorijske lokacije kojoj treba pristupiti u cilju upisa/čitanja; ovaj registar je magistralom povezan sa adresnim ulazom memorije
- *prihvatnog registra podatka memorije* (*MDR – Memory Data Register*), koji prihvata podatak pročitan sa izlaznih linija podataka memorije ili prihvata podatak koji treba proslediti do ulaznih linija podataka radi upisa u memoriju

Prenos instrukcije iz memorije u procesor se vrši tako što se najpre adresa iz *PC* prosledi u *MAR* (a samim tim i do adresnog ulaza memorije), zatim se instrukcija pročita iz ćelije čija je adresa upisana u *MAR* (instrukcija se pojavi na izlaznim linijama podataka memorije) i prosledi do *MDR*. S obzirom da *MDR* služi za prihvatanje različitih vrsta podataka, on nije pogodan za duže čuvanje instrukcije. Zato je uveden poseban procesorski registar u koji se smešta instrukcija iz *MDR*. To je *instrukcijski registar* (*IR – Instruction Register*). *ALU* izvršava instrukciju koja se nalazi u *IR*.

Dovođenje instrukcije u *IR* nije dovoljan uslov da ona bude izvršena. Iz instrukcije, *ALU* saznaje o kojoj instrukciji se radi na osnovu njenog koda operacije. Međutim, mnoge instrukcije zahtevaju operande za svoje izvršavanje. Na primer, instrukcija sabiranja zahteva dva operanda čiji zbir treba izračunati. Operandi se, takođe, nalaze u memoriji, pa je i njih potrebno preneti u procesor pre izvršenja instrukcije. Prenos operanada se odvija na isti način kao i prenos instrukcije, korišćenjem registara *MAR* i *MDR*. Kada operandi stignu u *MDR*, šalju se u posebne procesorske prihvatne registre za izvorišne operande koji su direktno povezani sa ulazima *ALU*. Sada postoje svi uslovi da *ALU* izvrši instrukciju. Nakon izvršenja, rezultat se pojavljuje u prihvatnom registru rezultata i prosleđuje do *MDR*. Time je okončan proces izvršavanja instrukcije.

Na slici 5.3 data je struktura procesora iz koje se vide unutrašnje veze koje postoje između *ALU* i procesorskih registara, kao i spoljašnje veze procesora sa memorijom. Prihvatni registri za izvorišne operande su označeni sa *A* i *B*, dok je prihvatni registar rezultata označen sa *C*.



Slika 5.3 Struktura procesora

Detaljan redosled aktivnosti koje treba obaviti prilikom izvršavanja instrukcije nad dva operanda dat je u tabeli 5.1. Adresni ulaz memorije označen je sa *M*(*A*), ulazne linije podataka memorije sa *M*(*DI*), a izlazne linije podataka memorije sa *M*(*DO*). Oznake *DI* i *DO* su skraćenice od *Data Input* i *Data Output*, respektivno. Izvršavanje instrukcije je označeno sa *ALU*(*IR*(*OC*),*A*,*B*), što znači da se instrukcija sa kodom operacije *OC* izvršava nad operandima koji se nalaze u registrima *A* i *B*. Oznaka (*M*(*A*),*M*(*DI*))→*M* predstavlja upis podatka sa *M*(*DI*) u memorijsku ćeliju sa adresom na *M*(*A*).

Proces	Aktivnosti
Prenos instrukcije	*PC*→*MAR*→*M*(*A*), *M*(*DO*)→*MDR*→*IR*
Prenos prvog operanda	*SRC*1→*MAR*→*M*(*A*), *M*(*DO*)→*MDR*→*A*
Prenos drugog operanda	*SRC*2→*MAR*→*M*(A), *M*(*DO*)→*MDR*→*B*
Izvršavanje instrukcije	*ALU*(*IR*(*OC*),*A*,*B*)→*C*→*MDR*→*M*(*DI*)
Upis rezultata u memoriju	*DST*→*MAR*→*M*(*A*),(*M*(*A*),*M*(*DI*))→*M*

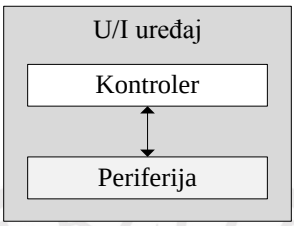
Tabela 5.1 Primer izvršavanja instrukcije

5.4 Ulazno/izlazni uređaji

Osim unosa podataka u računar i prikaza dobijenih rezultata, ulazno/izlazni uređaji imaju još jednu važnu ulogu. Oni omogućavaju korisnicima da specificiraju

ulazne/izlazne podatke u obliku koji im najviše odgovara. S obzirom da računar obrađuje samo binarne podatke, ulazno/izlazni uređaji konvertuju korisničke formate u binarni oblik i obrnuto.

U/I uređaj se sastoji od dve komponente: periferije i kontrolera periferije (slika 5.4).



Slika 5.4 Struktura *U/I* uređaja

Periferija se realizuje kao standardan uređaj, na primer, tastatura, miš, štampač, itd. Između periferije i kontrolera obavlja se kontrolisana razmena podataka. Zato svaka periferija mora da ima:

- linije za prenos podataka (podaci se prenose u obliku specifičnom za datu periferiju)
- upravljačke linije, kojima se šalju komande periferiji
- statusne linije, kojima se prenosi trenutni status periferije

Način prenosa podataka od/do periferije definisan je protokolom koji je specifičan za svaku periferiju.

Periferija može biti *ulazna* (podaci se sa periferije prosleđuju do procesora ili memorije) i *izlazna* (podaci iz memorije ili procesora se prosleđuju u periferiju).

Kontroler upravlja radom periferije. Za svaku periferiju postoji poseban kontroler. Osnovne uloge kontrolera su:

- fizičko povezivanje periferije sa memorijom i procesorom putem magistrale
- programska kontrola prenosa podataka od/do periferije

Programska kontrola omogućava da se:

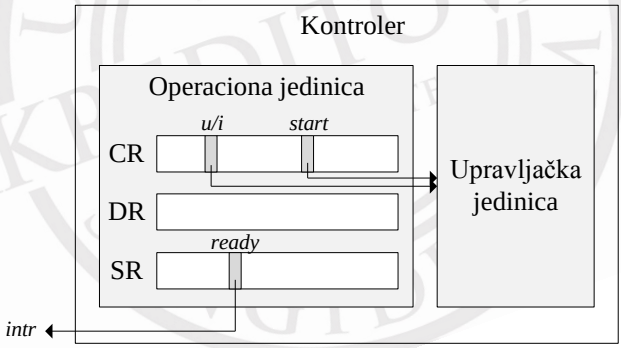
- startuje i zaustavi kontroler programskim putem
- dobije informacija o tome da li je podatak prenet iz ulazne periferije u kontroler

- dobije informacija o tome da li podatak prenet iz kontrolera u izlaznu periferiju
- izvrši prenos podatka iz kontrolera ulazne periferije u memoriju ili procesor
- izvrši prenos podatka iz memorije ili procesora u kontroler izlazne periferije

Kontroler periferije se sastoji od upravljačke jedinice i operacione jedinice (slika 5.5). *Upravljačka jedinica* generiše signale za prenos podataka iz ulazne periferije u kontroler ili iz kontrolera u izlaznu periferiju.

Operacionu jedinicu čine registri kojima se pristupa programski, tj. tokom izvršavanja instrukcija:

- registar podatka (*DR – Data Register*)
- upravljački registar (*CR – Control Register*)
- statusni registar (*SR – Status Register*)



Slika 5.5 Struktura kontrolera periferije

DR služi za prihvatanje podatka iz ulazne periferije, ili za prihvatanje podatka koji treba da se prosledi izlaznoj periferiji.

U sadržajima *CR* i *SR*, svaki bit, tj. fleg (*flag*), ima drugačije značenje. U *CR* se nalazi fleg *start* koji omogućava aktiviranje (kada *start* ima aktivnu vrednost) ili zaustavljanje kontrolera (kada *start* ima neaktivnu vrednost). Nakon aktiviranja kontrolera, na osnovu vrednosti flega *u/i* određuje se smer prenosa podatka, tj. da li je periferija ulazna (*u/i* ima aktivnu vrednost) ili izlazna (*u/i* ima neaktivnu vrednost). *SR* sadrži bit *ready* koji indicira da je podatak prenet iz ulazne periferije u *DR* ili da je podatak iz *DR* prenet u izlaznu periferiju. U oba slučaja treba generisati signal prekida *intr* koji kontroler šalje procesoru. Kada procesor primi

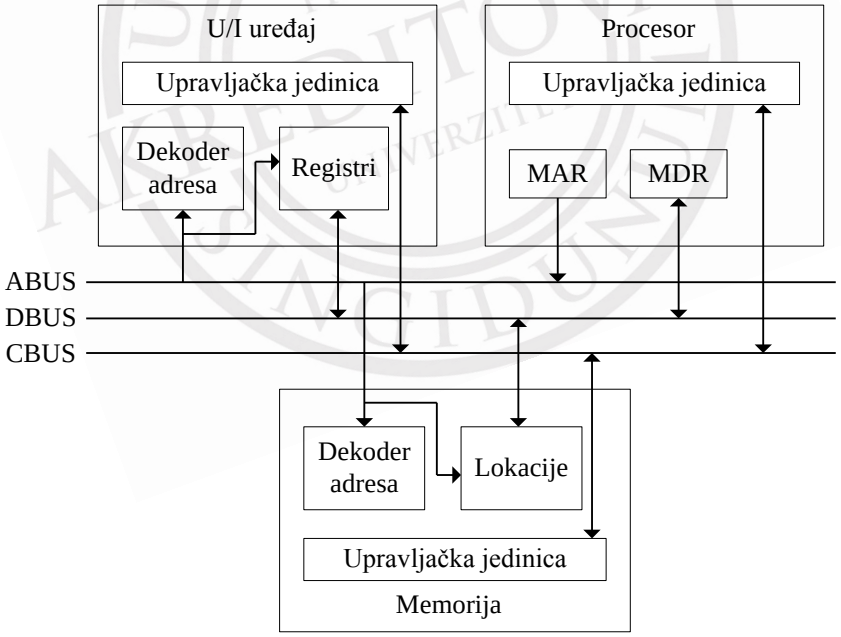
intr signal, prekida izvršavanje tekućeg programa i prelazi na izvršavanje prekidne rutine u okviru koje se sadržaj iz *DR* prenosi u odgovarajuću lokaciju u memoriji ili u procesor (može i obrnuto); nakon toga, procesor nastavlja sa izvršavanjem tekućeg programa.

5.5 Magistrala

Magistrala predstavlja uređeni skup linija koje povezuju osnovne komponente računara: procesor, memorijske jedinice i *U/I* uređaje. Tok prenosa sadržaja od jedne do druge komponente naziva se *ciklus na magistrali*.

Komponenta koja inicira ciklus naziva se *gazda* (*master*), a komponenta sa kojom se ostvaruje ciklus *sluga* (*slave*). Ulogu gazde obično ima procesor, a ulogu sluge memorija ili *U/I* uređaj.

Na slici 5.6 prikazano je kako se povezuju procesor, memorija i *U/I* uređaj na magistralu.



Slika 5.6 Priključivanje komponenata na magistralu

Skup linija po kojima gazda šalje adresu memorijske lokacije ili adresu registra *U/I* uređaja naziva se *adresna magistrala* (*ABUS – Address bus*). Skup

linija po kojima se prenose podaci naziva se *magistrala podataka* (*DBUS – Data bus*). *DBUS* je bidirekciona (dvosmerna) zato što prilikom upisa gazda šalje slugi podatak koji treba upisati, a prilikom čitanja sluga šalje pročitani podatak gazdi. Skup linija po kojima gazda šalje signale za upis ili čitanje sadržaja, ili sluga šalje signale da su upis ili čitanje realizovani, naziva se *upravljačka magistrala* (*CBUS – Control bus*).

Svaka komponenta ima upravljački deo koji se priključuje na *CBUS*. Sa *DBUS* dvosmernu vezu ostvaruju *MDR* procesora, linije za podatke memorije i registri operacione jedinice kontrolera *U*/*I* uređaja. Sa *ABUS* su povezani *MAR* procesora i dekoderi adresa memorije i *U*/*I* uređaja. Smerovi povezivanja sa *ABUS* zavise od uloge komponente, tj. da li je ona gazda ili sluga.

Vežbanja

P1. Objasniti sledeće pojmove: operacija, operand, instrukcija i program.

P2. Šta se podrazumeva pod arhitekturom, a šta pod organizacijom računara? U čemu je suštinska razlika između ova dva pojma?

P3. Navesti osnovne hardverske komponente računara i objasniti njihovu ulogu.

P4. Objasniti strukturu programske instrukcije.

P5. Detaljno opisati proces izvršavanja programa (gde se program nalazi, ko ga izvršava i kako).

P6. Na koji način procesor ostvaruje komunikaciju sa memorijom?

P7. Dati opštu strukturu *U/I* uređaja i objasniti ulogu pojedinih komponenata. Čemu služi kontroler periferije, od kojih komponenata se sastoji i koja je njihova uloga?

P8. Od kojih linija se sastoji magistrala i čemu te linije služe? Šta označavaju sledeći pojmovi: ciklus na magistrali, gazda i sluga?

P9. Objasniti način priključivanja procesora, memorije i *U/I* uređaja na magistralu.

6 Mehanizmi

U organizaciji računara značajnu ulogu imaju razni koncepti koji su vremenom uvođeni u cilju poboljšanja efikasnosti rada računara. U nastavku su ukratko opisana tri značajna koncepta.

6.1 *Pipeline* mehanizam

Pipeline predstavlja koncept koji je uveden sa ciljem da se ubrza proces izvršavanja programa. Koncept se zasniva na korišćenju konkurentnosti prilikom izvršavanja instrukcija. Pretpostavimo da se proces izvršavanja instrukcije odvija u sledeće četiri faze:

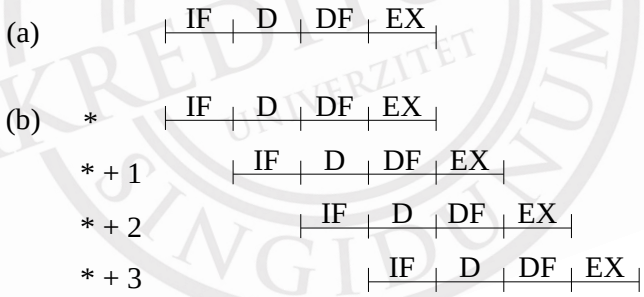
- *IF* (*Instruction Fatch*): dohvatanje instrukcije iz memorije i njeno smeštanje u instrukcijski registar u procesoru
- *D* (*Decode*): dekodiranje operacionog koda instrukcije
- *DF* (*Data Fatch*): dohvatanje operanada nad kojima se izvršava instrukcija iz memorije ili iz *U*/*I* uređaja
- *EX* (*Execute*): izvršavanje instrukcije

Uobičajeni postupak izvršavanja instrukcije prikazan je na slici 6.1(a). Instrukcija se izvršava tako što prolazi kroz faze *IF*, *D*, *DF* i na kraju *EX*.

Ideja u *pipeline* konceptu je da se omogući da se u istom trenutku različite instrukcije nalaze u različitim fazama izvršavanja, čime bi se ostvarila ušteda u vremenu izvršavanja programa. Na primer, naredna instrukcija se može dohvatati

iz memorije dok traje dekodiranje tekuće instrukcije. S obzirom da su ove dve aktivnosti međusobno nezavisne, nema nikakvih prepreka da se obavljaju istovremeno. *Pipeline* mašine upravo pokušavaju da obezbede stalnu zauzetost po svim fazama, tj. da u svakom taktu postoji po jedna instrukcija u svakoj od faza izvršavanja. Na slici 6.1(b) prikazan je *pipeline* koncept na primeru izvršavanja četiri uzastopne instrukcije od kojih se prva nalazi na nekoj adresi *.

Kao što se vidi, u prvom taktu dohvata se prva instrukcija iz memorije, u drugom se ona dekodira i istovremeno dohvata druga instrukcija iz memorije. U trećem taktu dohvataju se operandi za prvu instrukciju, druga instrukcija se dekodira i istovremeno se dohvata treća instrukcija iz memorije. U četvrtom taktu je ostvareno najveće ubrzanje pošto u svim fazama postoji po jedna instrukcija koja se obrađuje: prva instrukcija je u fazi izvršenja, druga u fazi dohvatanja operanada, treća u fazi dekodovanja i četvrta u fazi dopremanja iz memorije. U petom taktu izvršava se druga instrukcija, dohvataju se operandi za treću, a dekodira četvrta. U šestom taktu treća instrukcija se izvršava, a za četvrtu se dopremaju operandi. Na kraju, u sedmom taktu, izvršava se četvrta instrukcija. Dakle, sve četiri instrukcije su izvršene u periodu od 7 taktova, dok bi bez primenjenog *pipeline* koncepta za njihovo izvršenje bilo potrebno 16 taktova, po 4 za svaku instrukciju.



Slika 6.1 Izvršavanje instrukcija (a) bez *pipeline*-a (b) sa *pipeline*-om

Ubrzanje koje se može ostvariti primenom *pipeline* koncepta zavisi od broja faza u izvršavanju instrukcije, tj. broja stepeni (*stages*) ili dubine *pipeline*-a. U datom primeru broj stepeni je 4, pa je maksimalno ubrzanje približno četiri puta pod uslovom da je mašina dobro podešena. Naravno, postizanje ubrzanja ima svoju cenu. Ona se ogleda u većoj složenosti i većoj ceni procesora, jer u procesor treba ugraditi ovakav mehanizam izvršavanja instrukcija. Ipak, svi današnji procesori podržavaju *pipeline* koncept.

6.2 *DMA* mehanizam

DMA (*Direct Memory Access*) je koncept prisutan u svim modernim računarima koji omogućava direktan prenos podataka između periferije i memorije, bez posredovanja procesora. Zahvaljujući njemu, procesor, kao najskuplji i najopterećeniji resurs u računarskom sistemu, oslobođen je mnogih nepotrebnih poslova. Naime, u sistemu bez *DMA*, da bi preneo podatke od izvora do odredišta (na primer sa neke periferije do memorije) procesor mora da kopira deo po deo podataka sa izvora i prosleđuje ih na odredište. Tokom prenosa, procesor nije raspoloživ ni za kakvu drugu aktivnost. Otežavajuća okolnost je još i ta što su periferne jedinice mnogo sporije od operativne memorije i procesora, pa bi procesor gubio dodatno vreme. Uvođenjem *DMA* koncepta, procesor se oslobađa ovog posla i za vreme transfera podataka od izvora do odredišta može da obavlja druge poslove. Ipak, treba imati u vidu da u tim poslovima ne može da koristi magistralu kojom se obavlja *DMA* prenos.

Pri *DMA* prenosu važnu ulogu ima *DMA* kontroler. To je komponenta koja preuzima poslove oko prenosa koje je u sistemu bez *DMA* obavljao procesor. *DMA* ciklus (ceo tok prenosa) inicira procesor slanjem odgovarajuće komande *DMA* kontroleru. Nakon toga, potpunu odgovornost za prenos preuzima *DMA* kontroler. On generiše adrese i upravlja transferom podataka između računara i *U/I* uređaja. Po završetku prenosa, *DMA* kontroler obaveštava procesor da je prenos završen i da je magistrala slobodna.

DMA transfer se primenjuje u sledećim slučajevima:

- kada je potrebno preneti veliku količinu podataka između periferije i memorije
- kada je periferija relativno brza (kao na primer hard disk)

6.3 Mehanizam prekida

Mehanizam prekida je vrlo važan koncept u računarskim sistemima koji je uveden kako bi se izbeglo da procesor troši vreme čekajući na spoljašnje događaje. Značaj ovog mehanizma biće ilustrovan na konkretnom primeru rešavanja problema štampanja koji sledi.

Problem: Poznato je da su periferije znatno sporije od procesora. Pretpostavimo da procesor treba da pošalje veću količinu podataka štampaču na štampanje. Zbog ograničenih mogućnosti štampača, podaci se moraju slati u više delova. Kada procesor pošalje jedan deo na štampanje, mora da sačeka da štampač završi svoj posao, tj. da odštampa prispele podatke, kako bi mu

poslao sledeći deo. Čekanje na periferiju (dok ona završi svoj posao) predstavlja izgubljeno vreme za procesor, jer je on tada besposlen. S obzirom da je procesorsko vreme dragoceno, bilo je neophodno naći neko prihvatljivo rešenje za prevazilaženje ovog problema.

Rešenje problema: Problem je rešen tako što je uveden mehanizam prekida koji dopušta procesoru da izvršava druge aktivnosti dok periferija obavlja svoj posao.

Mehanizam prekida omogućava efikasniji rad računara sa periferijama tako što, za razliku od principa čekanja na periferiju da završi svoj posao, uvodi novi princip opsluživanja periferije na njen zahtev. To znači da kada procesor pošalje podatke periferiji, odmah nastavlja dalje sa radom ne čekajući da periferija završi posao. U trenutku kada periferija obavi posao, ona obaveštava procesor da je slobodna i tada procesor preduzima mere za slanje sledeće količine podataka periferiji.

Postupak opsluživanja prekida odvija se na sledeći način:

- Kada periferija postane spremna za prijem podataka od procesora, ona to signalizira procesoru slanjem *zahteva za prekidom*. Zahtev sadrži *kod prekida* koji identifikuje periferiju koja ga je poslala (procesor opslužuje veći broj periferija).
- Po prijemu zahteva, procesor završava izvršavanje tekuće instrukcije i na kratko prekida izvršavanje tekućeg programa kako bi opslužio prispeli zahtev.
- Svaki procesor ima skup prekida (tj. periferija) koje je u stanju da opsluži. Za svaki prekid unapred je definisana tzv. *prekidna rutina* koja predstavlja potprogram koji treba da se izvrši u slučaju prispeća zahteva za tim prekidom. Prekidne rutine su smeštene u memoriji na određenim adresama. Sve adrese prekidnih rutina smeštene su u *tabelu prekida (Interrupt Pointer Table*) koja se, takođe, nalazi u memoriji. Procesor opslužuje prekid tako što na osnovu koda prekida iz prispelog zahteva pronalazi u tabeli prekida adresu odgovarajuće prekidne rutine i izvršava tu rutinu. U prekidnoj rutini se opslužuje odgovarajuća periferija (šalju se ili primaju podaci sa nje).
- Po završetku prekidne rutine, procesor se vraća tamo gde je stao u programu koji je izvršavao u trenutku nailaska zahteva za prekidom i nastavlja sa radom.

Da bi opisani mehanizam prekida mogao uspešno da funkcioniše, neophodno je obezbediti da se pri povratku iz prekidne rutine procesor nađe i potpuno istim uslovima u kojima je bio kada je počeo opsluživanje prekida. Ti uslovi se nazivaju

kontekstom procesora. To znači da kada procesor prihvati zahtev za prekidom, pre nego što pređe na njegovo opsluživanje, mora da sačuva kontekst u kome je izvršavao tekući program. Nakon završetka prekidne rutine, sačuvani kontekst procesora se restaurira (postavljaju se sačuvane vrednosti u registre) i procesor zna odakle treba da nastavi izvršavanje započetog programa i kakav je bio status nakon poslednje operacije koju je *ALU* izvršila. Pamćenje konteksta procesora je neophodno zato što se prekidna rutina izvršava kao i svaki drugi program, što znači da se tom prilikom koriste isti procesorski registri, pa obično dolazi do izmene njihovog sadržaja.

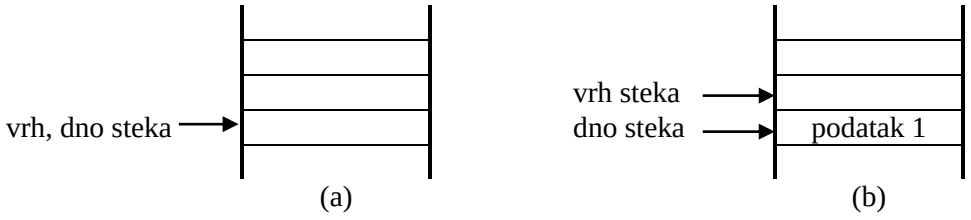
Kontekst procesora obično čine sadržaji sledećih procesorskih registara:

- statusnog registra, jer se u njemu nalaze važne informacije o trenutnom stanju procesora
- programskog brojača, jer se u njemu nalazi adresa naredne instrukcije programa čije se izvršavanje privremeno prekida

Pamćenje konteksta se vrši automatski, neposredno pre prelaska na izvršavanje prekidne rutine. Takođe, po završetku prekidne rutine, pomenuti registri se automatski reinicijalizuju na stare, sačuvane vrednosti.

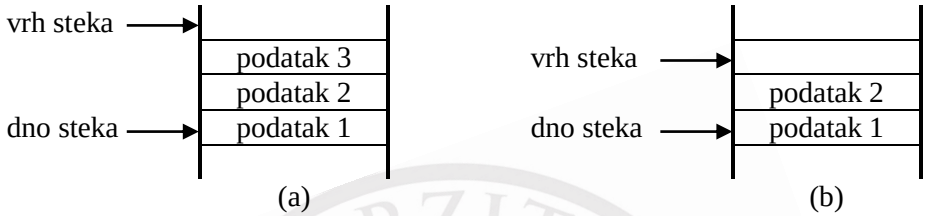
Kontekst procesora se čuva u posebnom delu memorije koji se naziva *stek* (*stack*). Stek podržava *LIFO* (*last in, first out*) disciplinu pristupa, što znači da se sa steka prvo uzimaju podaci koji su poslednji u njega uneti. Deo memorije predviđen za stek ograničen je pokazivačima na memorijske lokacije koje predstavljaju *dno steka* i *vrh steka*. Pristup steku, bilo u cilju upisa, bilo radi uzimanja nekog podatka, uvek se vrši preko vrha steka. Stek može biti implementiran na različite načine. Jedna od implementacija prikazana je na slikama 6.2 i 6.3.

Kada je stek prazan, vrh steka i dno steka ukazuju na istu memorijsku lokaciju (slika 6.2(a)). Pri stavljanju prvog podatka na stek, podatak se upisuje u memorijsku lokaciju na koju ukazuje vrh steka, a zatim se vrh steka pomera na prvu narednu lokaciju (slika 6.2(b)). Pokazivač na dno steka se ne pomera (uvek ukazuje na istu lokaciju). U ovoj implementaciji vrh steka uvek ukazuje na narednu praznu lokaciju na koju treba upisati sledeći podatak.



Slika 6.2 Upis podatka u stek

Čitanje podatka sa steka obavlja se tako što se najpre proveri da li je stek prazan (oba pokazivača ukazuju na istu lokaciju). Ako stek nije prazan (slika 6.3(a)), pokazivač na vrh steka se pomeri na prethodnu lokaciju, a zatim se iz nje preuzima podatak (slika 6.3(b)). Vrh steka ostaje na praznoj lokaciji iz koje je uzet podatak, tako da postoje uslovi za eventualni budući upis novog podatka.



Slika 6.3 Čitanje podatka sa steka

Prekidi se mogu klasifikovati na različite načine. Jedna od podela je na:

- *spoljašnje* ili *eksterne* prekide
- *unutrašnje* ili *interne* prekide

Spoljašnje prekide iniciraju periferije, dok unutrašnji prekidi mogu biti posledica izvršavanja instrukcije prekida, ili posledica neke neregularnosti u izvršavanju tekuće instrukcije. Prekidima se pridružuju *prioriteti* koji ukazuju na njihovu važnost. Prednost u opsluživanju imaju prekidi sa većim prioritetom. Interni prekidi su višeg prioriteta od eksternih, tako da ako u toku opsluživanja nekog eksternog prekida pristigne zahtev za internim prekidom, procesor prekida opsluživanje eksternog prekida i prelazi na opsluživanje internog prekida. Međutim, ako u toku opsluživanja internog prekida stigne zahtev za eksternim prekidom, procesor će ga eventualno prihvatiti tek kada završi započeti posao.

Vežbanja

P1. Objasniti *pipeline* mehanizam izvršavanja instrukcija.

P2. Šta predstavlja direktan pristup memoriji (*DMA*) i kada se primenjuje?

P3. Koji problem rešava mehanizam prekida i koje rešenje nudi?

P4. Opisati postupak opsluživanja prispelog zahteva za prekidom.

P5. Navesti vrste prekida, ukratko ih objasniti i uporediti po prioritetu.

P6. Šta je stek? Objasniti princip njegovog funkcionisanja na primeru upisa, odnosno čitanja nekog podatka sa steka.

UNIVERZITET
PRIVATNI
AKREDITOVANI
UNIVERZITET
SINGIDUNUM

7 Programske instrukcije

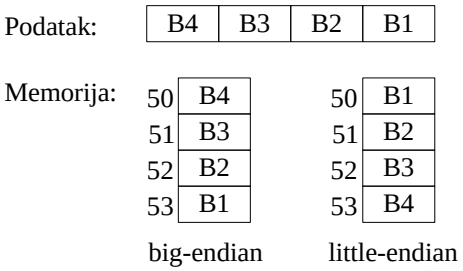
Obrada podataka u računaru sprovodi se izvršavanjem programskih instrukcija u zadatom redosledu. Instrukcije se obično izvršavaju nad nekim operandima. Svaka instrukcija ima precizno definisan format (izgled), u okviru koga su specificirani tipovi i vrednosti operanada. U ovom poglavlju biće predstavljeni formati instrukcija koji se najčešće koriste u savremenim procesorima.

7.1 Tipovi podataka

U računaru, binarni nizovi nula i jedinica mogu da predstavljaju podatke različitih tipova: neoznačene cele brojeve, cele brojeve sa znakom, brojeve u drugom komplementu, brojeve u pokretnom zarezu, alfanumeričke znakove, itd. Na primer, binarni niz 100011 može da se tumači kao 35 (neoznačeni ceo broj), -3 (ceo broj sa znakom) ili -29 (broj u drugom komplementu).

Podaci bilo kog tipa mogu biti različite dužine, na primer 8, 16, 32, 64 bita. Dužina podataka je vrlo bitna zato što su prostori za smeštaj podataka (memorijske lokacije, registri) u računaru ograničeni. Tako se može desiti da je podatak duži od veličine memorijske lokacije u koju bi ga trebalo upisati. U ovom slučaju, podatak se mora smestiti u više sukcesivnih memorijskih lokacija, što komplikuje rad sa tim podatkom.

Pretpostavimo da podatak od 4 bajta treba da se upiše u memoriju čije lokacije imaju dužinu 1 bajt (slika 7.1).



Slika 7.1 Upis podatka u memoriju

Zbog svoje dužine, podatak mora biti upisan u četiri uzastopne memorijske lokacije, na primer sa adresama 50, 51, 52 i 53. Postoje dva načina upisa ovog podatka:

- da se bajt najveće težine (*B*4) upiše u lokaciju sa najmanjom adresom (50), a zatim da se redom upisuju ostali bajtovi; ovaj postupak se naziva *big-endian*
- da se bajt najmanje težine (*B*1) upiše u lokaciju sa najmanjom adresom (50), a zatim da se redom upisuju ostali bajtovi; ovaj postupak se naziva *little-endian*

Ovakvo smeštanje podataka zahteva složeniju realizaciju procesora. Naime, procesor mora da zna da, u slučaju kada se traži podatak sa adrese 50, treba očitati četiri bajta sa adresa 50, 51, 52 i 53 i od pročitanih sadržaja formirati podatak po odgovarajućem postupku.

7.2 Formati instrukcija

Svaka instrukcija u računaru predstavljena je u binarnom obliku. Značenje bitova u ovom nizu nula i jedinica određeno je formatom instrukcije. Format sadrži sledeće informacije:

- *vrstu operacije* koju treba izvršiti
- *tip operanada* nad kojima se izvršava operacija
- *specifikaciju* izvorišnih i odredišnih *operanada*

Dužina instrukcije zavisi od veličine memorije i njene organizacije, širine magistrale, kao i složenosti procesora. Odluka o dužini instrukcije bitno utiče na fleksibilnost računara sa stanovišta programera. Programerima više odgovaraju raznovrsnije i moćnije instrukcije, zato što su tada programi koje pišu kraći.

Međutim, ovakve instrukcije su duže (po broju bitova), pa je potrebno više prostora za njihov smeštaj u memoriji. Pošto su ovi zahtevi oprečni, obično se pravi balans (*trade-off*) između njih.

Kao što je ranije rečeno, format instrukcije sadrži kod operacije i adresni deo instrukcije. Kodom operacije se specificira vrsta operacije i tip operanda. Tip operanda određuje iz koliko susednih memorijskih lokacija treba pročitati operand, ili u koliko susednih lokacija treba upisati operand i kako ga interpretirati.

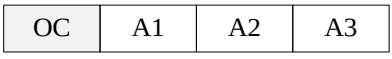
Ako se ista vrsta operacije primenjuje nad operandima različitih tipova ili različitih dužina, neophodno je za svaki operand definisati poseban kod operacije. Na primer, ako procesor podržava rad sa neoznačenim celim brojevima dužine 8 i 16 bitova i brojevima predstavljenim u pokretnom zarezu dužine 32 i 64 bita, za operaciju množenja je neophodno definisati četiri različita koda operacije. Primena jednog od ovih kodova operacije, direktno određuje tip operanda nad kojim će operacija biti izvršena.

Adresni deo instrukcije obično sadrži 3, 2, 1 ili 0 polja za specifikaciju operanada. Prema broju polja u adresnom delu instrukcija koje mogu da izvršavaju, procesori mogu biti troadresni, dvoadresni, jednoadresni ili nulaadresni. Procesor jednog tipa može da izvršava i instrukcije sa manje operanada (na primer, troadresni procesor može da izvršava i dvoadresne instrukcije). Procesor uvek mora da zna kako da interpretira sva polja u formatu instrukcije koju izvršava.

Za operande koji su zadati u adresnom delu instrukcije se kaže da su *eksplicitno definisani*. Međutim, ne moraju svi operandi da budu specificirani u adresnim poljima instrukcije. Na primer, jednoadresni format ne pruža mogućnost zadavanja dva operanda. U ovom slučaju se unapred zna gde se nalazi drugi operand. Za njega se kaže da je *implicitno definisan*.

7.2.1 Troadresni format

Troadresni format instrukcije je prikazan na slici 7.2.



Slika 7.2 Troadresni format instrukcije

Osim koda operacije *OC*, u formatu se nalaze još tri adresna polja za operande koji su eksplicitno definisani. Interpretacija ovih polja može biti različita. Obično se u poljima *A*1 i *A*2 nalaze adrese izvorišnih operanada, a u polju *A*3 adresa odredišnog operanda. Međutim, postoje i drugačije interpretacije. Na primer, može u polju *A*1 da bude adresa odredišnog operanda, a u poljima *A*2 i *A*3 adrese izvorišnih operanada. U poljima namenjenim izvorišnim operandima, umesto

adresa mogu da se nalaze i sami operandi. U polju koje odgovara odredišnom operandu uvek mora da bude adresa, jer rezultat operacije nije unapred poznat.

Dobra strana ovog formata je u tome što se binarna operacija (nad dva operanda) izvršava jednom instrukcijom. Loša strana formata je u velikoj dužini instrukcije.

Procenu dužine troadresne instrukcije možemo ilustrovati na primeru korišćenja memorije kapaciteta 1GB. Ova memorija ima 2^{30} lokacija dužine 1B ($1GB = 10^9B \approx 2^{30}B$). Dakle, za adresiranje jedne lokacije ove memorije potrebno je 30 bitova. Tri adrese u formatu instrukcije imaju dužinu 90 bitova. Ako je kod operacije 8-bitni, dužina cele instrukcije je oko 100 bitova. To znači da je za smeštaj jedne instrukcije potrebno bar 12 memorijskih lokacija, tj. relativno veliki prostor. Problem nije samo u prostoru, već i u brzini čitanja ove instrukcije. Da bi se kompletirala ova instrukcija, potrebno je 12 obraćanja memoriji, što usporava njeno izvršavanje.

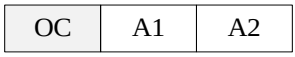
Izvršavanje instrukcije u troadresnom procesoru biće prikazano na primeru izračunavanja izraza $A + B \rightarrow C$. Za računanje ovog izraza je dovoljna samo jedna instrukcija:

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

Data instrukcija se izvršava tako što se izvorišni operandi, koji se nalaze u memorijskim lokacijama čije su adrese *A* i *B*, saberu i dobijeni zbir smesti u memorijsku lokaciju čija je adresa *C*. Operacija sabiranja definisana je kodom operacije `ADD`.

7.2.2 Dvoadresni format

Dvoadresna instrukcija je prikazana na slici 7.3.



Slika 7.3 Dvoadresni format instrukcije

Ovaj format se interpretira tako što se smatra da oba adresna polja predstavljaju izvorišne operande. Lokacija odredišnog operanda se zadaje implicitno kao jedna od izvorišnih lokacija (za svaki procesor se definiše koja je to lokacija). Na primer, može se uzeti da se rezultat operacije smešta u memorijsku ćeliju sa adresom prvog izvorišnog operanda. To znači da se pre izvršenja

instrukcije u datoj ćeliji nalazi izvorišni operand, a nakon izvršenja instrukcije odredišni operand (izvorišni operand se gubi).

U poređenju sa troadresnim formatom, dobra strana dvoadresnog formata je u tome što je instrukcija kraća, a loša strana u tome što je potrebno više instrukcija za izračunavanje izraza.

Povećanje broja instrukcija biće ilustrovano na ranijem primeru izračunavanja izraza $A + B \rightarrow C$. Kod dvoadresnih procesora, za računanje ovog izraza su potrebne dve instrukcije:

```
MOV A,C
ADD C,B
```

Kod operacije `MOV` definiše operaciju prenosa podatka, a kod operacije `ADD`, operaciju sabiranja. Neka u datom procesoru lokacija odredišnog operanda prilikom sabiranja odgovara adresi prvog izvorišnog operanda. Prva instrukcija se izvršava tako što se sadržaj lokacije na adresi *A* pročita i smesti u lokaciju sa adresom *C*. Ovo je bilo potrebno uraditi da bi se za drugu instrukciju obezbedilo da prvi izvorišni operand bude na odgovarajućem mestu. U drugoj instrukciji, izvorišni operandi iz lokacija sa adresama *C* i *B* se sabiraju i rezultat smešta u lokaciju sa adresom *C*.

7.2.3 Jednoadresni format

Format jednoadresne instrukcije je prikazan na slici 7.4.

Slika 7.4 Jednoadresni format instrukcije

U ovom formatu, eksplicitno je definisan samo jedan izvorišni operand, dok su drugi izvorišni operand i odredišni operand implicitno definisani. Podrazumeva se da se oni čuvaju u posebnom procesorskom registru koji se naziva *akumulator*.

Dobra strana ovog formata je u maloj dužini instrukcije, a loša strana je u tome što je potrebno još više instrukcija za izračunavanje izraza nego u slučaju dvoadresnog formata.

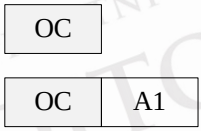
Za raniji primer računanja izraza $A + B \rightarrow C$, kod jednoadresnih procesora, potrebne su tri instrukcije:

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

Kod operacije `LOAD` definiše operaciju prenosa podatka u akumulator, kod operacije `ADD` operaciju sabiranja, a kod operacije `STORE` operaciju prenosa podatka iz akumulatora. Prva instrukcija se izvršava tako što se sadržaj lokacije na adresi *A* pročita i smesti u akumulator. Druga instrukcija čita sadržaj iz lokacije sa adresom *B*, sabira ga sa sadržajem akumulatora i rezultat smešta u akumulator. Treća instrukcija sadržaj akumulatora prenosi u lokaciju sa adresom *C*.

7.2.4 Nulaadresni format

Nulaadresni procesori podržavaju rad sa instrukcijama u dva formata prikazana na slici 7.5.



Slika 7.5 Formati instrukcija kod nulaadresnih procesora

Prvi format je nulaadresni, jer nema adresnih polja. Oba izvorišna i odredišni operand su implicitno definisani. Za njihovo smeštanje se koristi *stek*.

Drugi format je jednoadresni i predstavlja izuzetak kod nulaadresnih procesora. Zbog čuvanja operanada na steku, bilo je neophodno uvesti dve jednoadresne instrukcije za rad sa stekom: `PUSH` i `POP`. Instrukcija `PUSH` upisuje na vrh steka sadržaj lokacije čija je adresa data u adresnom polju, dok instrukcija `POP`, sadržaj sa vrha steka upisuje u lokaciju čija je adresa navedena u adresnom polju.

U odnosu na ostale formate, dužina instrukcije sa nulaadresnim formatom je najmanja, ali je potrebno najviše instrukcija za izračunavanje izraza.

Izraz iz ranijeg primera, $A + B \rightarrow C$ se kod nulaadresnih procesora računa pomoću četiri instrukcije:

```
PUSH A
PUSH B
ADD
POP C
```

Prva instrukcija se izvršava tako što se sadržaj lokacije na adresi *A* pročita i smesti na vrh steka. Druga instrukcija čita sadržaj iz lokacije sa adresom *B* i takođe ga smešta na vrh steka (prethodni sadržaj sa vrha steka se pomera za jedno mesto niže u steku). Treća instrukcija čita dva operanda sa vrha steka, sabira ih i rezultat smešta na vrh steka. Četvrta instrukcija čita sadržaj sa vrha steka i upisuje ga u lokaciju sa adresom *C*.

Analizom opisanih postupaka izračunavanja izraza iz primera primenom procesora različite adresnosti, mogu se izvesti sledeći zaključci:

- sadržaji memorijskih lokacija na adresama *A*, *B* i *C* u svim slučajevima su ostali nepromenjeni
- u svim slučajevima se na drugačiji način formira traženi zbir u lokaciji sa adresom *C*
- svaki od slučajeva ima drugačije memorijske i vremenske zahteve

7.3 Lociranje operanada

Operandi nad kojima se izvršavaju programske instrukcije se mogu nalaziti:

- u memoriji
- u procesorskim registrima
- u adresnim poljima formata instrukcije

U zavisnosti od toga gde su smešteni operandi, razlikuju se tri vrste arhitektura:

- memorija – memorija
- memorija – registar
- registar – registar

Kod arhitekture memorija – memorija, za smeštaj operanada (izvorišnih i odredišnih) koriste se isključivo memorijske lokacije. U ovoj arhitekturi se mogu primenjivati svi navedeni formati instrukcija.

Arhitektura memorija – registar podrazumeva da se jedan izvorišni operand nalazi u registru, drugi u memoriji, a odredišni operand bilo u memoriji ili u registru. U ovoj arhitekturi se obično koristi dvoadresni format.

U arhitekturi registar – registar, svi operandi se nalaze isključivo u registrima. U ovoj arhitekturi se obično koristi troadresni format.

Vežbanja

P1. Objasniti *big-endian* i *little-endian* postupke upisa podatka u memoriju.

P2. Koje informacije se mogu dobiti iz formata instrukcije?

P3. U čemu je razlika između eksplicitno i implicitno definisanih operanada?

P4. Objasniti strukturu, prednosti i nedostatke troadresnog formata instrukcije.

P5. Kakvu strukturu imaju dvoadresne instrukcije? Navesti prednosti i nedostatke dvoadresnog formata. Da li kod dvoadresnih instrukcija izvorišni i odredišni operandi mogu da budu na istim adresama u memoriji?

P6. Dati strukturu jednoadresnog formata instrukcije i uporediti ovaj format sa dvoadresnim formatom. Ukoliko se binarna operacija (na primer, sabiranje) izvršava pomoću jednoadresne instrukcije, gde se nalazi odredišni operand?

P7. Gde su smešteni operandi nulaadresnih instrukcija? Da li nulaadresni procesori podržavaju rad sa jednoadresnim instrukcijama?

P8. Navesti gde se sve mogu nalaziti operandi nad kojima se mogu izvršavati programske instrukcije. U zavisnosti od toga, koje vrste arhitektura postoje?

Z1. Na koji način će 32-bitna reč 0x00ABCDEF biti smeštena u 8-bitnu memorijsku lokaciju sa adresom 100, ako procesor podržava upis po principu:

a) *big-endian*
b) *little-endian*

Koje vrednosti se nalaze u lokaciji sa adresom 102 u ova dva slučaja?

Z2. Ako procesor podržava rad sa neoznačenim celim brojevima dužine 16 i 32 bita i brojevima predstavljenim u pokretnom zarezu dužine 64 bita, koliko je kodova operacije potrebno definisati za operaciju sabiranja?

Z3. Memorija kapaciteta 1MB se sastoji od lokacija dužine 1B. Proceniti dužinu troadresne instrukcije, ako je njen kod operacije dužine 1B. Koliko lokacija u memoriji je potrebno za smeštaj jedne ovakve instrukcije?

Z4. Napisati program za izračunavanje vrednosti izraza $Y = (A + B) \cdot C$ koji se može izvršavati na procesoru koji podržava:

a) 0 – adresni format instrukcije

b) 1 – adresni format instrukcije

c) 2 – adresni format instrukcije

d) 3 – adresni format instrukcije

Na raspolaganju su sledeće instrukcije: `ADD`, `MUL`, `MOV`, `PUSH/POP` i `LOAD/STORE`.

Polazne pretpostavke:

- dvoadresni format instrukcije ima oblik [*OC* | *SRC*1/*DST* | *SRC*2]
- troadresni format instrukcije ima oblik [*OC* | *SRC*1 | *SRC*2 | *DST*]

gde su *OC* – kod operacije, *SRC*1 – prvi izvorišni operand, *SRC*2 – drugi izvorišni operand, a *DST* – odredišni operand

Z5. Odredi izraze (*Y*) čije vrednosti računaju dati programi:

```
ADD B,C,Y     MOV B,Y      LOAD A       PUSH A
DIV A,Y,Y     ADD Y,C      DIV B        PUSH B
              MUL Y,A      STORE Y      ADD
                           LOAD C       PUSH C
                           DIV Y        PUSH D
                           STORE Y      ADD
                                        MUL
                                        POP Y
```

UNIVERZITET
PRIVATNI
AKREDITOVANI
UNIVERZITET
SINGIDUNUM

8 Procesorski registri

U svakom procesoru postoji skup registara koji služe za privremeno čuvanje malih količina podataka neophodnih za pravilan rad procesora. Skup procesorskih registara definiše projektant procesora tokom procesa projektovanja. Prema mogućnostima pristupa, procesorski registri se mogu svrstati u dve grupe: interni registri i programski dostupni registri.

8.1 Interni registri

Interni registri su registri procesora kojima se ne može programski pristupati. Ovim registrima se pristupa samo iz ugrađenih algoritama po kojima se instrukcija izvršava, pa stoga pripadaju organizaciji računara. Služe za čuvanje sadržaja u različitim fazama izvršavanja neke instrukcije, a njihov sadržaj se može koristiti samo u okviru te instrukcije.

Interni registri obuhvataju kontrolne registre i statusni registar.

8.1.1 Kontrolni registri

Tokom izvršavanja instrukcije, za smeštaj sadržaja koji se prenosi između memorije i procesora, koriste se sledeći kontrolni registri:

- programski brojač (*PC*) – sadrži adresu naredne instrukcije u memoriji
- instrukcijski registar (*IR*) – sadrži instrukciju

- adresni registar memorije (*MAR*) – sadrži adresu memorijske lokacije kojoj se pristupa
- registar podatka memorije (*MDR*) – sadrži podatak pročitan iz memorije ili podatak koji treba upisati u memoriju

Treba napomenuti da se *PC* obično inkrementira implicitno, nakon svake instrukcije, ali se može postaviti i eksplicitno instrukcijama skoka. To znači da mu se može pristupiti i programski, tj. da se u slučaju skokova ponaša kao programski dostupan registar.

Da bi bili procesirani, podaci se moraju dovesti na ulaze aritmetičko-logičke jedinice. *MDR* može direktno da bude priključen na *ALU*.

8.1.2 Statusni registar

Statusni registar sadrži određen broj indikatora, tj. flegova kojima se opisuje trenutno stanje procesora. Indikatori odgovaraju pojedinačnim bitovima u registru. Postavljaju se nezavisno jedan od drugog i predstavljaju različite statusne informacije. Često se statusni registar naziva *PSW* (*Program Status Word*) registrom, što je preuzeto iz *IBM* arhitekture.

Indikatori se mogu svrstati u dve grupe:

- statusni indikatori
- upravljački indikatori

Statusni indikatori se postavljaju hardverski na osnovu rezultata dobijenog izvršavanjem instrukcije, a proveravaju se softverski u instrukcijama uslovnog skoka. Uobičajeni statusni indikatori dati su u tabeli 8.1.

Indikator	Opis
N (*negative flag*)	postavlja se na 1 ako je rezultat operacije negativan
Z (*zero flag*)	postavlja se na 1 ako je rezultat operacije 0
C (*carry flag*)	postavlja se na 1 ako ima prenosa ili pozajmice pri aritmetičkim operacijama nad neoznačenim veličinama
V (*overflow flag*)	postavlja se na 1 ako ima prekoračenja pri aritmetičkim operacijama nad celobrojnim veličinama sa znakom

Tabela 8.1 Statusni indikatori

Upravljački indikatori se postavljaju softverski tokom izvršavanja posebnih instrukcija programa, a proveravaju se hardverski. Zbog upravljačkih indikatora, može se reći da je statusni registar delom programski dostupan.

Za mehanizam prekida, značajan upravljački indikator je *I* (*Interrupt flag*) koji se postavlja na 1 ako su dozvoljeni maskirajući prekidi.

8.2 Programski dostupni registri

Programski dostupan registar je registar procesora u koji se programskim putem, tj. prilikom izvršavanja instrukcije, može upisati/pročitati sadržaj. Ovom registru se može pristupati na dva načina:

- eksplicitno, specificiranjem njegove adrese u adresnom polju instrukcije
- implicitno, izborom koda operacije instrukcije koji automatski ukazuje na određeni registar

Za razliku od internih registara koji pripadaju organizaciji računara, programski dostupni registri su deo arhitekture računara. Oni služe za čuvanje rezultata izvršavanja neke instrukcije koji se kasnije mogu koristiti pri izvršavanju drugih instrukcija. Broj i uloga ovih registara razlikuje se od procesora do procesora.

Najčešće korišćeni programski dostupni registri su:

- registri podataka – *DR* (*Data registers*)
- adresni registri – *AR* (*Address registers*)
- registri opšte namene – *GPR* (*General-purpose registers*)

Osim navedenih, ovoj grupi registara mogu da pripadaju programski brojač (zbog instrukcija uslovnog skoka) i statusni registar (zbog upravljačkih indikatora).

Registri podataka služe samo za čuvanje podataka i ne mogu se koristiti pri računanjima adresa operanada. Procesor ovim podacima pristupa znatno brže nego podacima koji se nalaze u memoriji, jer je pristup memoriji skoro za red veličine sporiji. U registre podataka se obično smeštaju međurezultati obrade, kao i podaci koji se više puta koriste prilikom nekih izračunavanja.

Razlozi za uvođenje registara podataka su:

- sekvencijalna priroda obrade, pri kojoj se često rezultat jedne operacije koristi kao ulazni podatak za narednu operaciju (rezultat prve operacije se

smešta u *DR*, a ne u memoriju, pa mu se pri izvršavanju naredne operacije brže pristupa)

- pri obradi je velika verovatnoća da, ako se jednom pristupi nekom podatku, isti podatak bude uskoro opet korišćen

Adresni registri omogućavaju brže generisanje adresa, jer se tokom izvršavanja programa adrese (ili njihovi delovi) uzimaju iz ovih registara, a ne iz memorije, što bi bilo znatno sporije. Adresni registri mogu biti opšteg karaktera, ali mogu biti i specijalizovani u zavisnosti od primenjenog načina (moda) adresiranja. Adresni modovi su detaljno opisani u poglavlju 9. Specijalizovani adresni registri su, na primer, bazni i indeksni registri.

Kao što im ime kaže, *registri opšte namene* mogu da imaju različite uloge. U njima se mogu čuvati podaci (kao u registrima podataka) ili adrese (kao u adresnim registrima).

U zavisnosti od načina projektovanja, postoje procesori koji:

- imaju specijalizovane registre (registre podataka, adresne registre, bazne registre i indeksne registre)
- imaju samo registre opšte nemene, koji preuzimaju uloge specijalizovanih registara

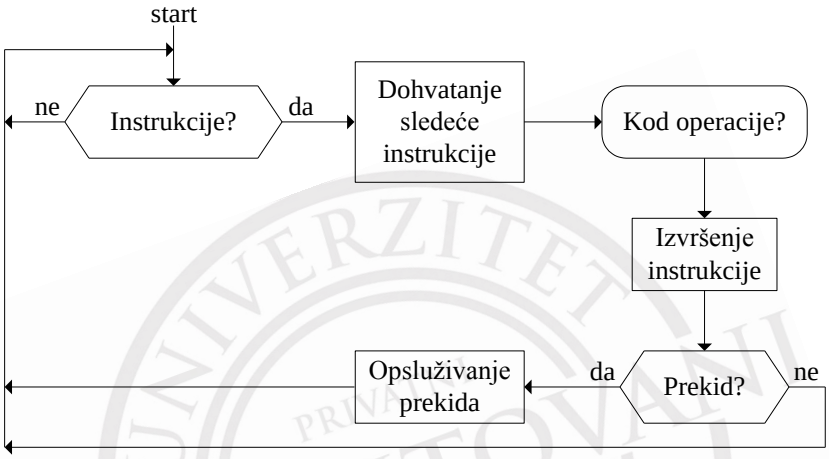
8.3 Instrukcijski ciklus

Procesor izvršava program tako što iz memorije dohvata jednu po jednu instrukciju, dok god ih ima. Na osnovu operacionog koda instrukcije, *ALU* izvršava odgovarajuću operaciju. Po završetku izvršenja svake instrukcije, procesor proverava da li ima zahteva za prekidom. Ukoliko ima, procesor opslužuje prekid izvršavanjem konkretne prekidne rutine. Nakon toga, dohvata sledeću instrukciju iz memorije. Ako nema zahteva za prekidom, procesor odmah dohvata narednu instrukciju. Ovaj proces se naziva instrukcijskim ciklusom i prikazan je na slici 8.1. To je osnovni ciklus rada računara.

Dohvatanje instrukcije iz memorije se obavlja sledećom sekvencom mikro operacija:

- sadržaj programskog brojača se prenosi u adresni registar memorije, $PC \rightarrow MAR$
- inkrementira se sadržaj programskog brojača (ova mikro operacija može da se radi istovremeno sa pristupom memoriji), $PC + 1 \rightarrow PC$

- pročitani sadržaj memorije predstavlja instrukciju koja se smešta u memorijski registar podatka, $M \rightarrow MDR$
- sadržaj memorijskog registra podatka se prenosi u instrukcijski registar, $MDR \rightarrow IR$



Slika 8.1 Instrukcijski ciklus

Sekvenca mikro operacija koje je potrebno obaviti prilikom izvršavanja instrukcije zavisi od same instrukcije i biće prikazna na dva primera instrukcija sabiranja.

Primer 1: `ADD R1,R2,R0` $(R1) + (R2) \rightarrow R0$

Sekvenca mikro operacija:

- adrese registara *R*1, *R*2 i *R*0 se preuzimaju direktno iz formata instrukcije
- sadržaji registara *R*1 i *R*2 se dovode na ulaze *ALU* radi sabiranja
- izlaz iz *ALU* se prenosi u registar *R*0

Primer 2: `ADD X,R0` $M(X) + (R0) \rightarrow R0$

Sekvenca mikro operacija:

- adresa memorijske lokacije *X* se uzima iz formata instrukcije i smešta u *MAR*
- podatak pročitan iz memorije (sadržaj na adresi *X*) se smešta u *MDR*

- sadržaji registara *MDR* i *R*0 se dovode na ulaze *ALU* radi sabiranja
- izlaz iz *ALU* se prenosi u registar *R*0

Opsluživanje prekida podrazumeva izvšavanje sledeće sekvence mikro operacija:

- sadržaj programskog brojača se prenosi u memorijski registar podatka (mora se sačuvati zbog povratka u tekući program), $PC \rightarrow MDR$
- u memorijski adresni registar se upisuje adresa *A* na kojoj će se sačuvati sadržaj programskog brojača, $A \rightarrow MAR$
- u programski brojač se upisuje adresa prve instrukcije u prekidnoj rutini *A*1, $A1 \rightarrow PC$
- stara vrednost programskog brojača iz *MDR* se smešta u memoriju na adresu iz *MAR*, $MDR \rightarrow M(MAR)$

Vežbanja

P1. U zavisnosti od načina pristupa, koje vrste procesorskih registara postoje? Da li oni pripadaju arhitekturi ili organizaciji računara? Zašto?

P2. Koji procesorski registri čine interne registre procesora? Objasniti njihovu ulogu.

P3. Koje vrste indikatora sadrži statusni registar? Kako se postavljaju vrednosti ovih indikatora, a kako se proveravaju? Šta označavaju uobičajeni statusni indikatori u statusnom registru: *N*, *Z*, *C* i *V*?

P4. Na koje načine se može uticati na vrednost programskog brojača?

P5. Koji su najčešće korišćeni programski dostupni registri? Čemu oni služe? Kako se može pristupati ovim registrima?

P6. Da li se registri podataka mogu koristiti za računanje adresa operanada?

P7. Navesti korake u okviru instrukcijskog ciklusa i detaljno objasniti svaki od njih. Nacrtati sliku.

Z1. Data je memorija kapaciteta 256MB. Odrediti mogući sadržaj adresnog registra (*MAR*) u sledećim slučajevima:

a) ako je dužina memorijske reči 4 bajta
b) ako se adresira 56. memorijska reč
c) ako se adresiraju samo parne 16-bitne reči u memoriji (numerisane sa 0, 2, 4, itd.)

Z2. Program od četiri instrukcije, zajedno sa potrebnim operandima, smešten je u memoriju na sledeći način:

Adresa lokacije	Sadržaj lokacije	
...	...	
0x60	0x1034	
0x61	0x2462	Instrukcije
0x62	0x8389	
0x63	0x2247	
...	...	
0x70	0x5772	
0x71	0x9898	Operandi
0x72	0x1388	

Da li tokom izvršavanja ovog programa mogu da važe sledeća tvrđenja:

a) u *PC* se nalazi vrednost 0x62
b) u *IR* se nalazi vrednost 0x60
c) u *MAR* se nalazi vrednost 0x5772
d) u *MDR* se nalazi vrednost 0x1034
e) u *PC* se nalazi vrednost 0x1388
f) u *IR* se nalazi vrednost 0x9898
g) u *MAR* se nalazi vrednost 0x71
h) u *MDR* se nalazi vrednost 0x72

9 Adresni modovi

Adresni modovi predstavljaju *načine adresiranja operanada* u instrukciji. U zavisnosti od toga gde se operandi fizički nalaze, uvedeni su različiti adresni modovi. Ako se operand nalazi u nekoj memorijskoj lokaciji, adresni mod specificira kako se formira adresa te lokacije. Jednostavniji slučaj je kada se operand nalazi u nekom procesorskom registru, jer tada adresa registra istovremeno predstavlja i adresu operanda. Do operanda se najlakše dolazi ako je on direktno upisan u adresno polje u formatu instrukcije.

Adresni modovi programerima pružaju brojne mogućnosti kao što su: korišćenje pokazivača, korišćenje brojača za kontrolu petlji, indeksiranje podataka, itd. Zahvaljujući njima, redukovan je broj bitova u adresnim poljima instrukcija, čime je smanjena veličina instrukcija, a samim tim i prostor potreban za njihovo smeštanje.

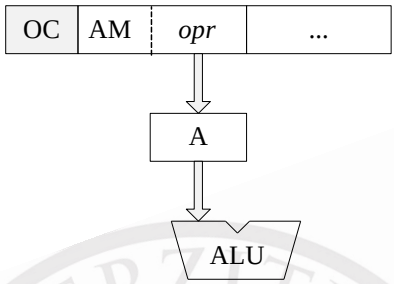
U ovom poglavlju, opisani su najčešće korišćeni adresni modovi: neposredno adresiranje, direktno adresiranje, indirektno adresiranje i adresiranja sa pomerajem, kao i njihove varijante.

9.1 Neposredno adresiranje

Neposredno adresiranje je najjednostavniji adresni mod koji podrazumeva da se *operand* nalazi *unutar same instrukcije*. Pošto svaka promena operanda zahteva izmene u svim instrukcijama u kojima se taj operand koristi, ovaj adresni mod ima malu fleksibilnost.

Neposredno adresiranje se može koristiti samo za adresiranje izvorišnih operanada, jer zadavanje odredišnog operanda na ovaj način nema smisla.

Na slici 9.1 prikazan je postupak neposrednog adresiranja.



Slika 9.1 Neposredno adresiranje

Adresno polje u formatu instrukcije koje specificira izvorišni operand je podeljeno u dva dela. Jedan broj bitova definiše adresni mod (*AM*), dok ostali bitovi predstavljaju sam operand (*opr*). *AM* ukazuje da je izvorišni operand neposredno adresiran, pa binarni niz *opr* treba tumačiti kao vrednost operanda. Da bi se instrukcija izvršila, potrebno je da se vrednost *opr* upiše u prihvatni registar podatka (*A*) na ulazu u *ALU*.

Sledi primer neposredno adresiranog operanda.

```
MOV #500,R1          (# je oznaka za neposredno adresiranje)
```

Pri izvršavanju ove instrukcije, operand 500 se direktno preuzima iz formata instrukcije i upisuje u registar čija je adresa *R*1.

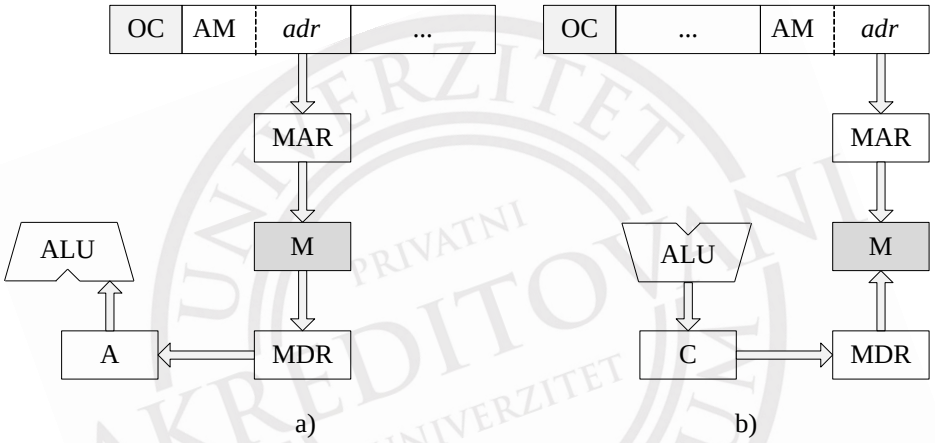
9.2 Direktno adresiranje

Direktno adresiranje je adresni mod koji podrazumeva da se *u adresnom polju instrukcije* navodi *adresa na kojoj se nalazi operand*. U zavisnosti od toga da li ta adresa predstavlja adresu memorijske lokacije ili adresu procesorskog registra, postoje dve varijante direktnog adresiranja: memorijsko direktno adresiranje i registarsko direktno adresiranje.

9.2.1 Memorijsko direktno adresiranje

Memorijsko direktno adresiranje je adresni mod kod koga se adresa memorijske lokacije u kojoj se nalazi operand navodi u formatu instrukcije. Ovaj mod je fleksibilniji od neposrednog adresiranja zato što se vrednost operanda može promeniti samo izmenom sadržaja određene memorijske lokacije, bez intervenisanja na instrukcijama. Na ovaj način se mogu specificirati i izvorišni i odredišni operandi.

Na slici 9.2 prikazan je postupak memorijskog direktnog adresiranja.



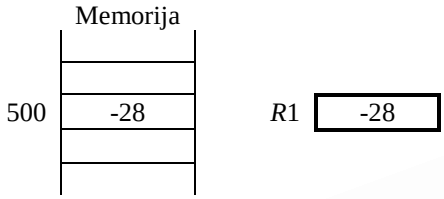
Slika 9.2 Memorijsko direktno adresiranje (a) izvorišnog i (b) odredišnog operanda

Adresno polje operanda (izvorišnog ili odredišnog) sastoji se od adresnog moda (*AM*) i adrese (*adr*). *AM* ukazuje da se koristi memorijsko direktno adresiranje, pa binarni niz *adr* treba tumačiti kao adresu memorijske lokacije u kojoj se čuva operand. Da bi se instrukcija izvršila, najpre je potrebno je da se vrednost *adr* upiše u *MAR* i prosledi do memorije. Ako se adresira izvorišni operand, njegova vrednost se čita iz memorije, prosleđuje u *MDR* i dalje u prihvatni registar A na ulazu *ALU*, koja izvršava instrukciju. U slučaju odredišnog operanda, sadržaj u *MAR* predstavlja adresu u memoriji na koju treba upisati rezultat operacije. Dakle, kada *ALU* generiše rezultat, on se pojavljuje u prihvatnom registru *C* na njenom izlazu, odakle se prosleđuje do *MDR*. Na kraju se sadržaj *MDR* upisuje u memorijsku lokaciju sa adresom iz *MAR*.

Sledi primer memorijskog direktnog adresiranja operanda.

```
MOV 500,R1
```

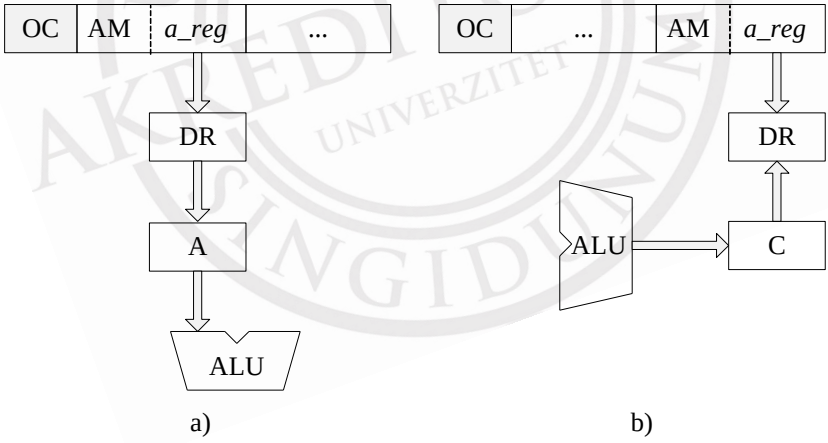
Pri izvršavanju ove instrukcije, operand se čita iz memorijske lokacije sa adresom 500 i upisuje u registar *R*1.



9.2.2 Registarsko direktno adresiranje

Registarsko direktno adresiranje je adresni mod kod koga se u formatu instrukcije navodi adresa registra (to može da bude registar podatka ili neki registar opšte namene) u kome se nalazi operand. Ovaj mod je, takođe, fleksibilniji od neposrednog adresiranja, a omogućava adresiranje i izvorišnih i odredišnih operanada.

Na slici 9.3 prikazan je postupak registarskog direktnog adresiranja.



Slika 9.3 Registarsko direktno adresiranje (a) izvorišnog i (b) odredišnog operanda

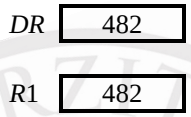
Adresno polje operanda (izvorišnog ili odredišnog) sastoji se od adresnog moda (*AM*) i adrese registra (*a_reg*). *AM* ima istu ulogu kao kod prethodno opisanih adresnih modova. Ako se adresira izvorišni operand, iz registra sa adresom *a_reg* (registar *DR*) se pročita vrednost operanda i prosledi u prihvatni registar *A* na ulazu *ALU*, koja zatim izvršava instrukciju nad tim operandom. U slučaju adresiranja odredišnog operanda, rezultat koji je generisala *ALU* se

pojavljuje u izlaznom prihvatnom registru *C*, a zatim se upisuje u registar čija je adresa *a_reg*.

Sledi primer registarskog direktnog adresiranja operanda.

```
MOV DR,R1
```

Pri izvršavanju instrukcije, operand se čita iz registra podatka čija je adresa *DR* i upisuje u registar *R*1.



9.3 Indirektno adresiranje

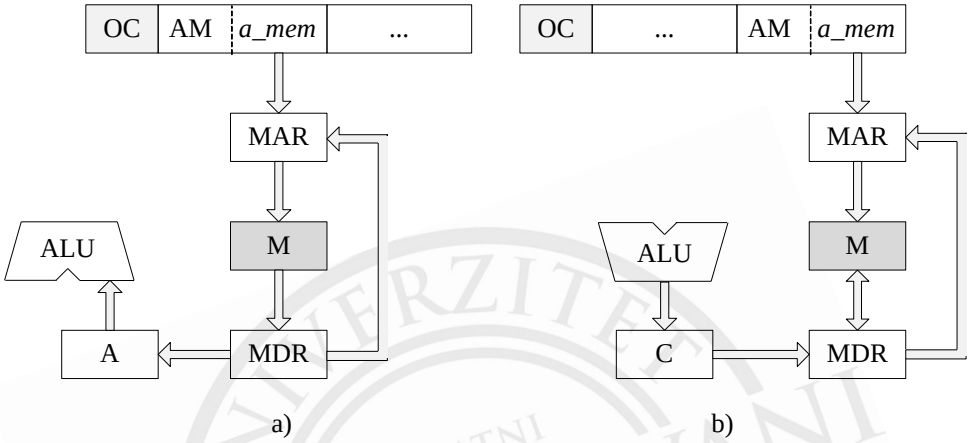
Indirektno adresiranje je adresni mod koji podrazumeva da se *u adresnom polju instrukcije* navodi *adresa na kojoj se nalazi adresa operanda*. Adresa u polju instrukcije može da predstavlja adresu memorijske lokacije ili adresu procesorskog registra, pa postoje dve varijante indirektnog adresiranja: memorijsko indirektno adresiranje i registarsko indirektno adresiranje. U oba slučaja, operand se fizički nalazi u memoriji.

9.3.1 Memorijsko indirektno adresiranje

Memorijsko indirektno adresiranje je adresni mod kod koga se u formatu instrukcije navodi adresa memorijske lokacije koja sadrži adresu memorijske lokacije u kojoj se nalazi operand. Ovaj mod se koristi za specificiranje i izvorišnih i odredišnih operanada.

Na slici 9.4 prikazan je postupak memorijskog indirektnog adresiranja. Adresno polje operanda (izvorišnog ili odredišnog) sastoji se od adresnog moda (*AM*) i adrese memorijske lokacije (*a_mem*). Da bi se instrukcija izvršila, najpre je potrebno je da se vrednost *a_mem* upiše u *MAR* i prosledi do memorije. Sa ove adrese, iz memorije se pročita sadržaj *a_op* i prosledi u *MDR*. Vrednost *a_op* predstavlja adresu memorijske lokacije za smeštaj operanda. Zatim se sadržaj *MDR* upisuje u *MAR* i prosleđuje do adresnih linija memorije. Ako se adresira izvorišni operand, sa adrese u *MAR*, iz memorije se čita operand koji se smešta u *MDR* i dalje prosleđuje u prihvatni registar *A* na ulazu *ALU*, koja zatim izvršava instrukciju. U slučaju odredišnog operanda, sadržaj u *MAR* predstavlja adresu u memoriji na koju treba upisati rezultat operacije. Dakle, kada *ALU* generiše

rezultat, on se pojavljuje u prihvatnom registru *C* na njenom izlazu, odakle se prosleđuje u *MDR*. Sadržaj *MDR* se zatim upisuje u memorijsku lokaciju sa adresom u *MAR*.

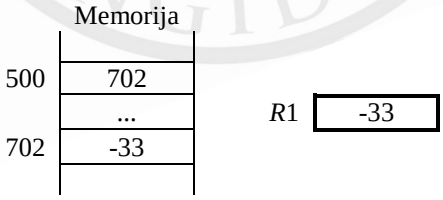


Slika 9.4 Memorijsko indirektno adresiranje (a) izvorišnog i (b) odredišnog operanda

Sledi primer memorijskog indirektnog adresiranja operanda.

`MOV (500),R1` (zagrade označavaju da je operand indirektno adresiran)

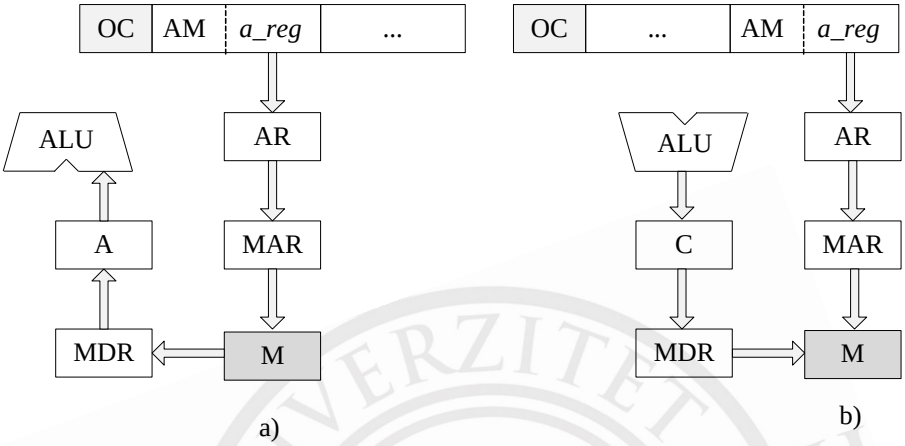
Pri izvršavanju instrukcije, operand se čita iz memorijske lokacije čija se adresa nalazi u memoriji na adresi 500 i upisuje u registar *R*1.



9.3.2 Registarsko indirektno adresiranje

Registarsko indirektno adresiranje je adresni mod kod koga se u formatu instrukcije navodi adresa registra koji sadrži adresu memorijske lokacije u kojoj se nalazi operand. Registar može da bude jedan od adresnih registara ili neki registar opšte namene. Ovaj mod se koristi za specificiranje i izvorišnih i odredišnih operanada.

Na slici 9.5 prikazan je postupak registarskog indirektnog adresiranja.



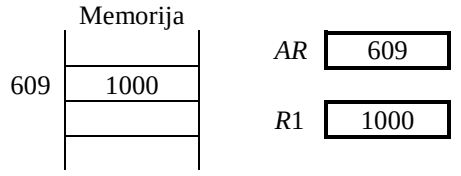
Slika 9.5 Registarsko indirektno adresiranje (a) izvorišnog i (b) odredišnog operanda

Adresno polje operanda (izvorišnog ili odredišnog) sastoji se od adresnog moda (*AM*) i adrese registra (*a_reg*). Da bi se instrukcija izvršila, najpre se pročita sadržaj registra čija je adresa *a_reg*, upiše u *MAR* i prosledi do memorije. Ako se adresira izvorišni operand, sa adrese u *MAR*, iz memorije se pročita operand koji se smešta u *MDR* i dalje prosleđuje u prihvatni registar *A* na ulazu *ALU*, koja zatim izvršava instrukciju. U slučaju odredišnog operanda, sadržaj u *MAR* predstavlja adresu u memoriji na koju treba upisati rezultat operacije. Dakle, kada *ALU* generiše rezultat, on se pojavljuje u prihvatnom registru *C* na njenom izlazu, odakle se prosleđuje u *MDR*. Sadržaj *MDR* se zatim upisuje u memorijsku lokaciju sa adresom u *MAR*.

Sledi primer registarskog indirektnog adresiranja operanda.

`MOV (AR),R1` (zagrade označavaju da je operand indirektno adresiran)

Pri izvršavanju instrukcije, operand se čita iz memorije sa adrese koja se nalazi u registru *AR* i upisuje u registar *R*1.



U nekim primenama, na primer pri realizaciji steka, postoji potreba da se adresiraju operandi u uzastopnim memorijskim lokacijama. Da bi se olakšalo ovakvo adresiranje, uvedene su dve varijante registarskog indirektnog adresiranja: adresiranje sa autoinkrementiranjem i adresiranje sa autodekrementiranjem.

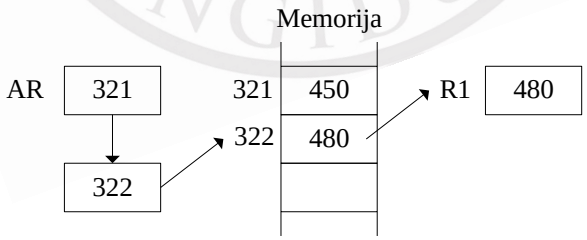
Registarsko indirektno adresiranje sa autoinkrementiranjem

Kod adresiranja sa autoinkrementiranjem, u formatu instrukcije se navodi adresa registra u kome se nalazi adresa memorijske lokacije koja se automatski inkrementira (uvećava za 1). To znači da se, nakon pristupa prvom operandu, vrednost u registru menja tako da postaje adresa drugog operanda koji se nalazi u narednoj memorijskoj lokaciji. U zavisnosti od trenutka u kome se vrši inkrementiranje, postoje dve mogućnosti:

- *preinkrement adresiranje* (oznaka +(*R*)) – sadržaj registra *R* se najpre inkrementira, pa se nova vrednost uzima kao adresa memorijske lokacije operanda
- *postinkrement adresiranje* (oznaka (*R*)+) – sadržaj registra *R* se uzima kao adresa memorijske lokacije operanda, pa se onda inkrementira

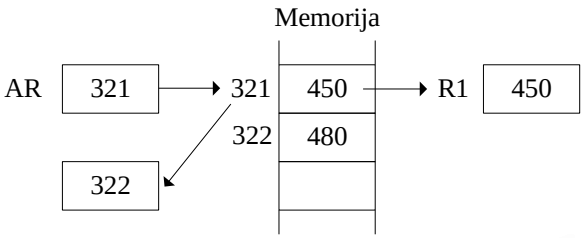
Mogućnosti adresiranja sa autoinkrementiranjem su ilustrovane na primeru instrukcije `MOV (AR),R1` koja prenosi podatak iz memorijske lokacije u registar *R*1. Adresa memorijske lokacije se određuje na osnovu sadržaja registra *AR*.

U slučaju preinkrementiranja (data instrukcija postaje `MOV +(AR),R1`), adresa memorijske lokacije iz koje se uzima operand se računa tako što se sadržaj *AR* uveća za 1.



preinkrementiranje

U slučaju postinkrementiranja (data instrukcija postaje `MOV (AR)+,R1`), sadržaj *AR* predstavlja adresu memorijske lokacije iz koje se uzima podatak.



postinkrementiranje

Kao što se vidi, sadržaji registra *AR* pre (321) i posle (322) izvršenja instukcije su isti u oba slučaja, iako su adresirani različiti operandi (u prvom slučaju 480, a u drugom 450).

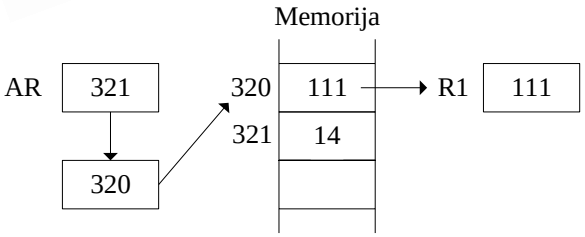
Registarsko indirektno adresiranje sa autodekrementiranjem

Kod adresiranja sa autodekrementiranjem, u formatu instrukcije se navodi adresa registra koji sadrži adresu memorijske lokacije koja se automatski dekrementira (umanjuje za 1). U zavisnosti od trenutka u kome se vrši dekrementiranje, postoje dve mogućnosti:

- *predekrement adresiranje* (oznaka –(*R*)) – sadržaj registra *R* se najpre dekrementira, pa se nova vrednost uzima kao adresa memorijske lokacije operanda
- *postdekrement adresiranje* (oznaka (*R*)–) – sadržaj registra *R* se uzima kao adresa memorijske lokacije operanda, pa se onda dekrementira

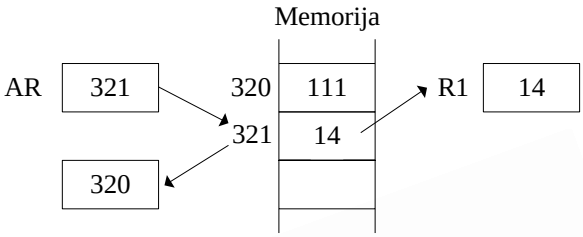
Mogućnosti adresiranja sa autodekrementiranjem su ilustrovane na prethodno opisanom primeru instrukcije `MOV (AR),R1`.

U slučaju predekrementiranja (data instrukcija postaje `MOV -(AR),R1`), adresa memorijske lokacije iz koje se uzima operand se računa tako što se sadržaj *AR* umanji za 1.



predekrementiranje

U slučaju postdekrementiranja (data instrukcija postaje `MOV (AR)-,R1`), sadržaj *AR* predstavlja adresu memorijske lokacije iz koje se uzima podatak.



postdekrementiranje

Slično kao i kod adresiranja sa autoinkrementiranjem, sadržaji registra *AR* pre (321) i posle (320) izvršenja instrukcije su isti u oba slučaja, iako su adresirani različiti operandi (u prvom slučaju 111, a u drugom 14).

9.4 Adresiranja sa pomerajem

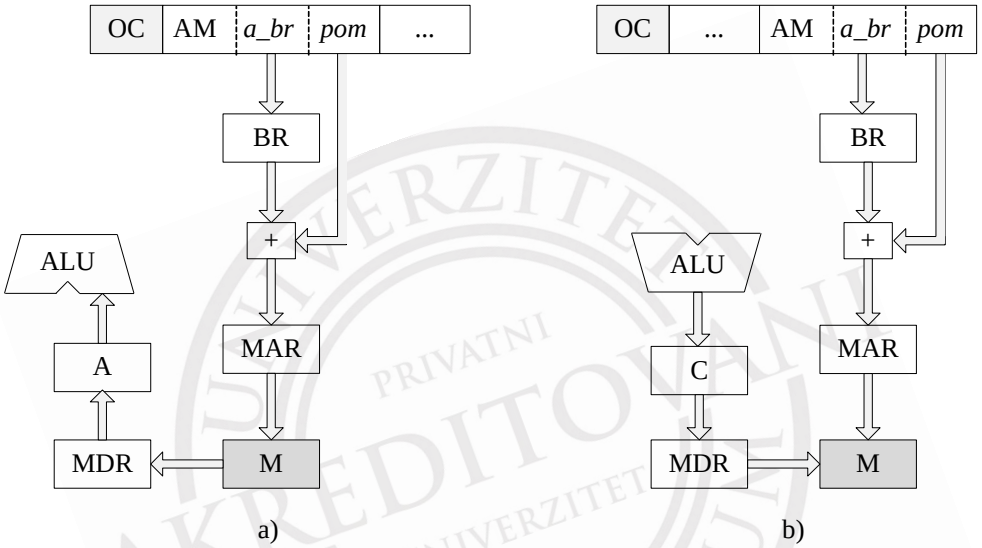
Načini adresiranja operanada kod kojih se u adresnim poljima instrukcije, osim adresa registara, navode i fiksne vrednosti, tzv. pomeraji, nazivaju se adresiranja sa pomerajem. Pri generisanju adrese operanda, pomeraj se sabira sa sadržajem registara. U zavisnosti od korišćenih registara, postoje različite vrste adresiranja sa pomerajem: bazno adresiranje, indeksno adresiranje, bazno-indeksno adresiranje i relativno adresiranje.

9.4.1 Bazno adresiranje

Bazno adresiranje sa pomerajem je adresiranje kod koga se u formatu instrukcije navode adresa baznog registra i pomeraj. Adresa memorijske lokacije u kojoj se nalazi operand se dobija *sabiranjem sadržaja baznog registra i pomeraja*. Ovaj adresni mod se može koristiti za adresiranje i izvorišnih i odredišnih operanada kod procesora koji imaju bazne registre.

Na slici 9.6 prikazan je postupak baznog adresiranja sa pomerajem. Adresno polje operanda (izvorišnog ili odredišnog) se sastoji od tri komponente: adresnog moda (*AM*), adrese baznog registra (*a_br*) i pomeraja (*pom*). Adresa memorijske lokacije za smeštaj operanda se dobija tako što se pročita sadržaj baznog registra čija je adresa *a_br* i sabere sa *pom*. Operandu se pristupa tako što se ova adresa upiše u *MAR* i prosledi do adresnih linija memorije. Ako se adresira izvorišni operand, sa adrese u *MAR*, iz memorije se pročita operand koji se smešta u *MDR* i

dalje prosleđuje u prihvatni registar *A* na ulazu *ALU*, koja zatim izvršava instrukciju. U slučaju odredišnog operanda, sadržaj u *MAR* predstavlja adresu u memoriji na koju treba upisati rezultat operacije. Dakle, kada *ALU* generiše rezultat, on se pojavljuje u prihvatnom registru *C* na njenom izlazu, odakle se prosleđuje u *MDR*. Sadržaj *MDR* se zatim upisuje u memorijsku lokaciju sa adresom u *MAR*.

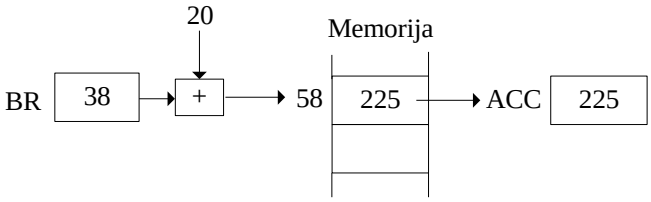


Slika 9.6 Bazno adresiranje sa pomerajem (a) izvorišnog i (b) odredišnog operanda

Sledi primer instrukcije koja u akumulator upisuje operand koji je adresiran primenom baznog adresiranja:

`LOAD 20(BR)` (`20` predstavlja pomeraj, a *BR* adresu baznog registra)

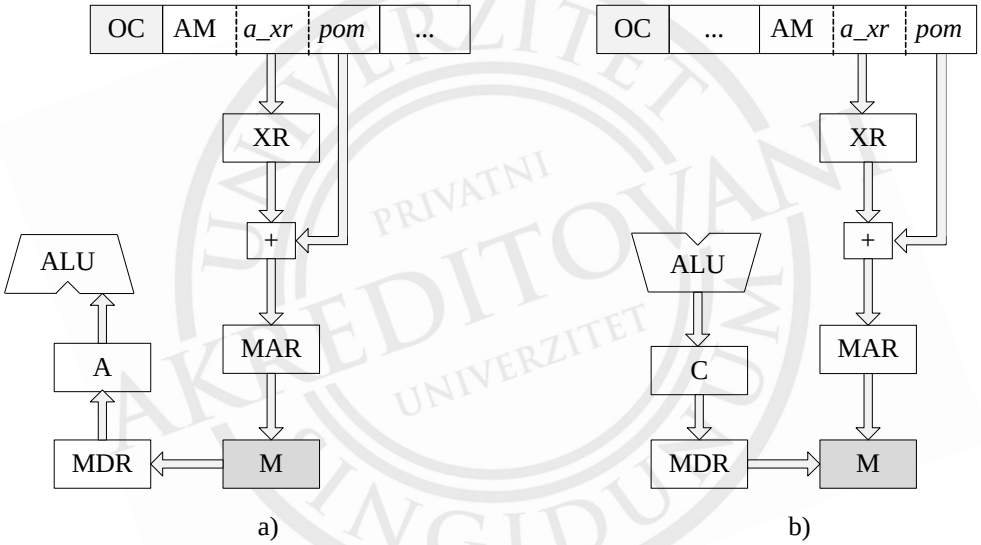
Pri izvršavanju instrukcije, operand se čita iz memorije sa adrese koja se dobija sabiranjem pomeraja 20 sa sadržajem baznog registra *BR*.



9.4.2 Indeksno adresiranje

Indeksno adresiranje sa pomerajem je adresiranje kod koga se u formatu instrukcije navode adresa indeksnog registra i pomeraj. Adresa memorijske lokacije u kojoj se nalazi operand se formira *sabiranjem sadržaja indeksnog registra i pomeraja*. Ovaj adresni mod se može koristiti za adresiranje i izvorišnih i odredišnih operanada kod procesora koji imaju indeksne registre.

Na slici 9.7 prikazan je postupak indeksnog adresiranja sa pomerajem.



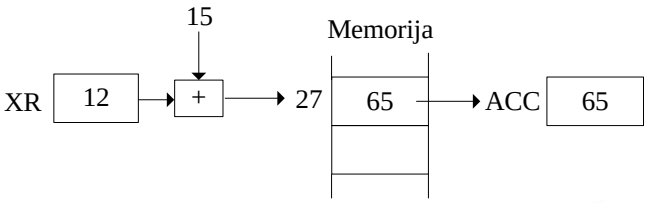
Slika 9.7 Indeksno adresiranje sa pomerajem (a) izvorišnog i (b) odredišnog operanda

Kod ovog načina adresiranja, postupak je isti kao kod baznog adresiranja sa pomerajem, s tim što se umesto baznog, koristi indeksni registar *XR* čija je adresa *a_xr*.

Sledi primer instrukcije koja u akumulator upisuje operand koji je adresiran primenom indeksnog adresiranja:

`LOAD 15(XR)` (`15` označava pomeraj, a *XR* adresu indeksnog registra)

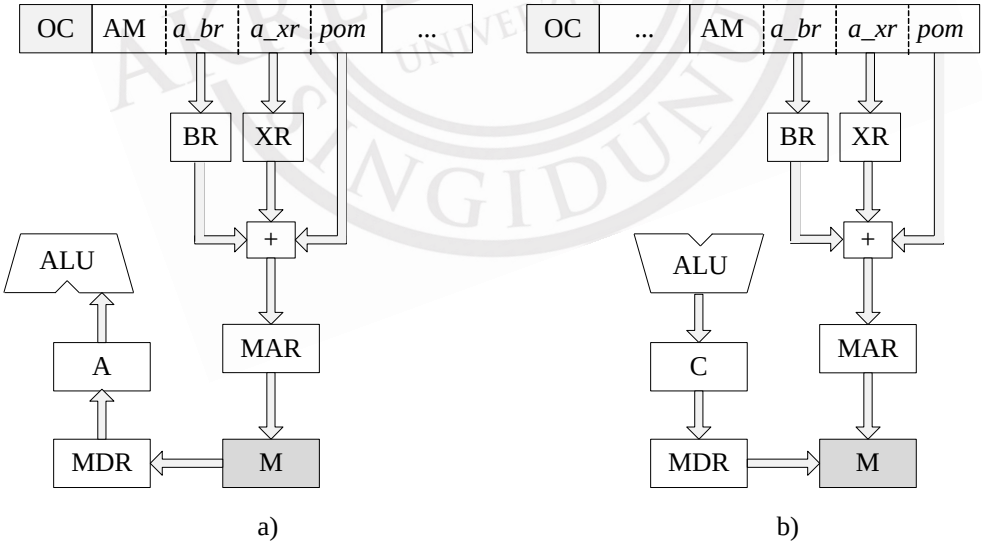
Pri izvršavanju instrukcije, operand se čita iz memorije sa adrese koja se dobija sabiranjem pomeraja 15 sa sadržajem indeksnog registra *XR*.



9.4.3 Bazno-indeksno adresiranje

Bazno-indeksno adresiranje sa pomerajem je adresiranje kod koga se u formatu instrukcije navode adresa baznog registra, adresa indeksnog registra i pomeraj. Adresa memorijske lokacije u kojoj se nalazi operand se formira *sabiranjem sadržaja baznog registra, sadržaja indeksnog registra i pomeraja*. Ovaj adresni mod se može koristiti za adresiranje i izvorišnih i odredišnih operanada, u slučaju procesora koji imaju bazne i indeksne registre. Kod procesora koji imaju samo registre opšte namene, sa pomerajem se sabiraju sadržaji dva registra opšte namene.

Na slici 9.8 prikazan je postupak bazno-indeksnog adresiranja sa pomerajem.



Slika 9.8 Bazno-indeksno adresiranje sa pomerajem (a) izvorišnog i (b) odredišnog operanda

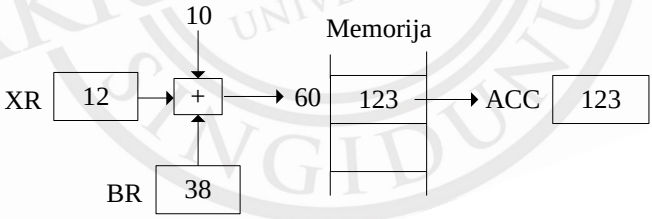
Adresno polje operanda (izvorišnog ili odredišnog) sastoji se od četiri komponente: adresnog moda (*AM*), adrese baznog registra (*a_br*), adrese indeksnog registra (*a_xr*) i pomeraja (*pom*). Adresa memorijske lokacije rezervisane za operand se dobija tako što se pročitaju sadržaji baznog registra čija je adresa *a_br* i indeksnog registra čija je adresa *a_xr* i saberu sa *pom.* Ova adresa se upisuje u *MAR* i prosleđuje do adresnih linija memorije. Ako se adresira izvorišni operand, sa adrese u *MAR*, iz memorije se pročita operand koji se smešta u *MDR* i dalje prosleđuje u prihvatni registar *A* na ulazu *ALU*, koja zatim izvršava instrukciju. U slučaju odredišnog operanda, sadržaj u *MAR* predstavlja adresu u memoriji na koju treba upisati rezultat operacije. Dakle, kada *ALU* generiše rezultat, on se pojavljuje u prihvatnom registru *C* na njenom izlazu, odakle se prosleđuje u *MDR*. Sadržaj *MDR* se zatim upisuje u memorijsku ćeliju sa adresom u *MAR*.

Sledi primer instrukcije koja u akumulator upisuje operand koji je adresiran primenom bazno-indeksnog adresiranja:

```
LOAD 10(BR,XR)
```

(`10` je pomeraj, *BR* adresa baznog, a *XR* adresa indeksnog registra)

Pri izvršavanju instrukcije, operand se čita iz memorije sa adrese koja se dobija sabiranjem pomeraja 10 sa sadržajima baznog i indeksnog registra.

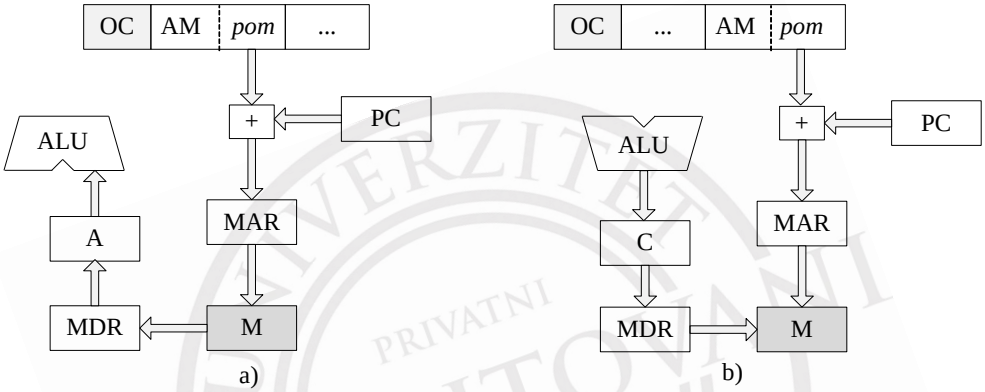


9.4.4 Relativno adresiranje

Relativno adresiranje sa pomerajem je adresiranje kod koga se u formatu instrukcije navodi samo pomeraj, a adresa memorijske lokacije u kojoj se nalazi operand se formira *sabiranjem pomeraja sa sadržajem programskog brojača*. Ovaj adresni mod se može koristiti za adresiranje i izvorišnih i odredišnih operanada.

Na slici 9.9 prikazan je postupak relativnog adresiranja sa pomerajem. Adresno polje operanda (izvorišnog ili odredišnog) sastoji se od dve komponente: adresnog moda (*AM*) i pomeraja (*pom*). Adresa memorijske lokacije za smeštaj operanda se dobija tako što se pročita sadržaj programskog brojača i sabere sa pomerajem *pom*. Ovako dobijena adresa se upisuje u *MAR* i prosleđuje do adresnih

linija memorije. Ako se adresira izvorišni operand, sa adrese u *MAR*, iz memorije se pročita operand koji se smešta u *MDR* i dalje prosleđuje u prihvatni registar *A* na ulazu *ALU*, koja zatim izvršava instrukciju. U slučaju odredišnog operanda, sadržaj u *MAR* predstavlja adresu u memoriji na koju treba upisati rezultat operacije. Dakle, kada *ALU* generiše rezultat, on se pojavljuje u prihvatnom registru *C* na njenom izlazu, odakle se prosleđuje u *MDR*. Sadržaj *MDR* se zatim upisuje u memorijsku ćeliju sa adresom u *MAR*.



Slika 9.9 Relativno adresiranje sa pomerajem (a) izvorišnog i (b) odredišnog operanda

9.5 Primena adresnih modova

Sve češća potreba za zahtevnijom obradom velikih količina raznovrsnih podataka, dovela je do povećanja složenosti softvera. Postalo je veoma važno da se podacima što brže pristupa, tj. da se oni adresiraju na što efikasniji način. To je uslovilo uvođenje velikog broja adresnih modova primenljivih u različitim situacijama.

Najjednostavniji adresni mod, neposredno adresiranje, pogodan je za definisanje konstanti.

Direktno memorijsko adresiranje se koristi za adresiranje jednostavnih promenljivih, tako što njihova imena direktno adresiraju memorijske lokacije.

Relativno adresiranje je pogodno pri programskim skokovima, jer se skokovi obično izvode na obližnje instrukcije. Zadavanjem pomeraja, mogu se ostvariti skokovi i unapred i unazad po instrukcijama programa. Zahvaljujući relativnom adresiranju, programski kod je poziciono nezavisan, tj. može se smestiti bilo gde u memoriji bez potrebe za podešavanjem adresa.

Registarsko indirektno adresiranje se koristi za adresiranje nizova, tabela i sličnih struktura podataka, zato što pruža mogućnost automatskog inkrementiranja ili dekrementiranja adresa. Zbog malog broja registara, adrese registara su kraće od adresa memorijskih lokacija, pa način adresiranja pomoću registara štedi memoriju jer je broj bitova u instrukcijama manji.

Indeksno adresiranje je efikasno pri indeksiranju članova niza, dok se bazno adresiranje koristi za pristup poljima neke strukture ili nekog objekta (adresa strukture se smešta u bazni registar, a pomeraj određuje polje unutar te strukture).

Bazno-indeksno adresiranje je pogodno pri radu sa dvodimenzionalnim nizovima. U ovom slučaju, adresa niza se zadaje pomerajem, dok se adresa vrste smešta u bazni registar, a adresa elementa vrste, tj. indeks kolone u indeksni registar. Ovaj adresni mod se može koristiti i pri radu sa nizovima čiji su elementi strukture ili objekti. Tada se obično adresa niza smešta u bazni registar, relativna adresa strukture u indeksni registar, a pomeraj predstavlja adesu polja u strukturi.

Vežbanja

P1. Šta su adresni modovi?

P2. Kako se naziva adresni mod kod koga se operand nalazi unutar same instrukcije? Koje su osobine tog adresnog moda? Da li se odredišni operand može adresirati na ovaj način?

P3. Objasniti postupak direktnog adresiranja. U čemu je razlika između memorijskog direktnog i registarskog direktnog adresiranja?

P4. U čemu je razlika između direktnog i indirektnog adresiranja? Šta sadrži adresno polje kod registarskog indirektnog adresiranja?

P5. Koje vrste registarskog indirektnog adresiranja postoje. Detaljno ih objasniti na primerima.

P6. Navesti vrste adresiranja sa pomerajem.

P7. Kako se formira adresa operanda kod baznog adresiranja sa pomerajem?

P8. Od koliko komponenata se sastoji adresno polje izvorišnog operanda kod indeksnog adresiranja?

P9. Da li se bazno-indeksno adresiranje može primeniti kod procesora koji nemaju bazne i indeksne registre?

P10. Objasniti postupak relativnog adresiranja operanda. Šta omogućava ovaj način adresiranja?

P11. Koji adresni mod je pogodan za:
   a) specifikaciju konstanti
   b) specifikaciju jednostavnih varijabli
   c) rad sa tabelama
   d) rad sa nizovima

e) rad sa dvodimenzionalnim nizovima

Z1. U adresnom polju neke instrukcije nalazi se decimalna vrednost 24. Gde se nalazi odgovarajući operand, ako je primenjeno:

a) neposredno adresiranje
b) memorijsko direktno adresiranje
c) memorijsko indirektno adresiranje
d) registarsko direktno adresiranje
e) registarsko indirektno adresiranje

Z2. Dat je računarski sistem sa procesorom koji podržava 1 – adresni format instrukcija i memorijom u kojoj se nalazi sledeći sadržaj:

Adresa lokacije	Sadržaj lokacije
...	...
68	12
69	45
70	100
71	68
72	66
73	71
...	...

Odrediti vrednosti koje se upisuju u akumulator (*ACC*) izvršavanjem sledećih instrukcija:

```
LOAD #71          – neposredno adresiranje
LOAD 71           – direktno memorijsko adresiranje
LOAD (71)         – indirektno memorijsko adresiranje
LOAD #68
LOAD 69
LOAD (73)
```

Instrukcija `LOAD` *opr* izvršava operaciju *opr* → *ACC*.

Z3. Registri *R*1 i *R*2 sadrže vrednosti 1100 i 3800, respektivno. Odrediti adrese operanda u memoriji u sledećim instrukcijama:

```
MOV 20(R1),R3
ADD R1,30(R1,R2)
SUB R2,-(R1)
MUL (R2)+,R3
```

Z4. Sadržaj programskog brojača *PC* je 50. U memorijskoj lokaciji sa adresom 50 nalazi se instrukcija koja u adresnom polju ima kao referencu na operand 40. Deo memorije ima sledeći sadržaj:

Adresa lokacije	Sadržaj lokacije
...	
40	70
...	...
50	40
...	...
60	35
...	...
70	21
...	...
90	13
...	...

Odrediti vrednost operanda nad kojim se izvršava instrukcija ako je primenjeno:

a) direktno memorijsko adresiranje
b) indirektno memorijsko adresiranje
c) relativno adresiranje sa pomerajem

Z5. U memoriji na adresi $250_{(10)}$ se nalazi instrukcija čiji je operand relativno adresiran.

a) ako se operand fizički nalazi u memoriji na adresi $300_{(10)}$, odrediti vrednost pomeraja u adresnom polju instrukcije
b) ako se operand nalazi u memoriji na adresi $220_{(10)}$, odrediti 8-bitnu binarnu vrednost pomeraja

Z6. U adresni registar *AR* upisana je adresa $17_{(10)}$. Odrediti sadržaj registra podatka *DR* i sadržaj registra *AR* nakon izvršenja instrukcije `MOV AR,DR`

ako je za adresiranje prvog operanda u instrukciji iskorišćeno registarsko indirektno adresiranje sa:

a) preinkrementiranjem
b) postinkrementiranjem
c) predekrementiranjem
d) postdekrementiranjem

Deo memorije ima sledeći izgled:

Adresa lokacije	Sadržaj lokacije
...	...
15	38
16	20
17	176
18	72
19	60
20	91
...	...

Z7. Za izgled memorije kao u prethodnom zadatku, odrediti vrstu registarskog indirektnog adresiranja prvog operanda i početni sadržaj adresnog registra *AR*, ako su nakon izvršenja instrukcije `MOV AR,DR` dobijeni sledeći sadržaji registara:

a) sadržaj *DR*: 60 sadržaj *AR*: 20
b) sadržaj *DR*: 72 sadržaj *AR*: 18

Z8. Procesor podržava bazno-indeksno adresiranje. Neka instrukcija se izvršava nad operandom koji se nalazi u memorijskoj lokaciji na adresi $2015_{(10)}$. U trenutku izvršavanja ove instrukcije, bazni registar sadrži vrednost $12824_{(10)}$, a indeksni registar vrednost $8_{(10)}$. Kolika je 16-bitna vrednost pomeraja navedena u instrukciji?

10 Instrukcijski set

Instrukcijski set predstavlja *skup svih instrukcija* koje procesor može da izvrši. Izbor instrukcija u setu zavisi od namene procesora. Procesori opšte namene uglavnom podržavaju standardne instrukcije, dok procesori specijalne namene implementiraju i nestandardne instrukcije.

Zbog velikog broja, standardne instrukcije su tematski svrstane u sledeće grupe: aritmetičke instrukcije, logičke instrukcije, pomeračke instrukcije, instrukcije prenosa i instrukcije skoka. Standardne instrukcije su detaljno opisane u ovom poglavlju.

Nestandardne instrukcije se definišu prema konkretnim potrebama. To su, na primer, instrukcije nad celobrojnim veličinama promenljive dužine, instrukcije za rad sa stringovima, instrukcije kontrole petlji, itd.

10.1 Aritmetičke instrukcije

Aritmetičke instrukcije su instrukcije koje realizuju standardne aritmetičke operacije: sabiranje, oduzimanje, množenje i deljenje.

Instrukcije sabiranja

U tabeli 10.1 prikazane su standardne instrukcije sabiranja:

- `ADD`, koja aritmetički sabira dva izvorišna operanda i zbir smešta u odredišni operand

- `INC`, koja inkrementira (uvećava za 1) vrednost izvorišnog operanda i rezultat smešta u odredišni operand

U tabeli 10.1, kao i u ostalim tabelama u ovom poglavlju, korišćene su sledeće oznake:

n.a. – neposredno adresiran operand

ACC – akumulator (procesorski registar)

TS – vrh steka (*Top of Stack*)

Operacija	Instrukcija	Izvorišni operandi	Odredišni operand	Napomena
sabiranje	`ADD a,b,c`	`a` i `b`	`c`	`c` nije n.a.
	`ADD a,b`	`a` i `b`	`a` (ili `b`)	`a` (ili `b`) nije n.a.
	`ADD a`	ACC i `a`	ACC	
	`ADD`	TS i TS	TS	
inkrementiranje	`INC a,b`	`a`	`b`	`b` nije n.a.
	`INC a`	`a`	`a`	`a` nije n.a.
	`INC`	TS	TS	

Tabela 10.1 Instrukcije sabiranja

Instrukcije oduzimanja

U tabeli 10.2 prikazane su standardne instrukcije oduzimanja:

- `SUB`, koja aritmetički oduzima drugi izvorišni operand od prvog i razliku smešta u odredišni operand
- `DEC`, koja dekrementira (umanjuje za 1) vrednost izvorišnog operanda i rezultat smešta u odredišni operand
- `CMP`, koja poredi izvorišne operande tako što od prvog oduzima drugi, pa dobijeni *rezultat nigde ne upisuje*, već ga samo proverava i na osnovu njega postavlja indikatore *N*, *Z*, *C* i *V* statusnog registra; indikator *N* ima vrednost 1 ako je rezultat negativan, indikator *Z* ima vrednost 1 ako je rezultat 0, indikator *C* ima vrednost 1 ako je bilo pozajmice pri oduzimanju, a indikator *V* ima vrednost 1 ako je došlo do prekoračenja opsega; svrha postavljanja indikatora je da se omogući realizacija uslovnih

skokova u programu (instrukcija uslovnog skoka proverava indikatore i utvrđuje da li je uslov za skok ispunjen ili nije; ako je uslov ispunjen, ostvaruje se skok, a ako nije, nastavlja se sa sekvencijalnim izvršavanjem instrukcija programa)

Operacija	Instrukcija	Izvorišni operandi	Odredišni operand	Napomena
oduzimanje	`SUB a,b,c`	`a` i `b`	`c`	`c` nije n.a.
	`SUB a, b`	`a` i `b`	`a` (ili `b`)	`a` (ili `b`) nije n.a.
	`SUB a`	ACC i `a`	ACC	
	`SUB`	TS i TS	TS	
dekrementiranje	`DEC a,b`	`a`	`b`	`b` nije n.a.
	`DEC a`	`a`	`a`	`a` nije n.a.
	`DEC`	TS	TS	
aritmetičko poređenje	`CMP a,b,c`	`a` i `b`		`c` se ne koristi
	`CMP a,b`	`a` i `b`		
	`CMP a`	ACC i `a`		
	`CMP`	TS i TS		

Tabela 10.2 Instrukcije oduzimanja

Instrukcija množenja

U tabeli 10.3 je prikazana standardna instrukcija množenja:

- `MUL`, koja aritmetički množi izvorišne operande i proizvod smešta na odredište

Operacija	Instrukcija	Izvorišni operandi	Odredišni operand	Napomena
množenje	`MUL a,b,c`	`a` i `b`	`c`	`c` nije n.a.
	`MUL a,b`	`a` i `b`	`a` (ili `b`)	`a` (ili `b`) nije n.a.
	`MUL a`	ACC i `a`	ACC	
	`MUL`	TS i TS	TS	

Tabela 10.3 Instrukcija množenja

Instrukcija deljenja

U tabeli 10.4 je data standardna instrukcija deljenja:

- `DIV`, koja deli prvi izvorišni operand sa drugim i količnik smešta na odredište

Operacija	Instrukcija	Izvorišni operandi	Odredišni operand	Napomena
deljenje	`DIV a,b,c`	`a` i `b`	`c`	`c` nije n.a.
	`DIV a,b`	`a` i `b`	`a` (ili `b`)	`a` (ili `b`) nije n.a.
	`DIV a`	ACC i `a`	ACC	
	`DIV`	TS i TS	TS	

Tabela 10.4 Instrukcija deljenja

10.2 Logičke instrukcije

Logičke instrukcije su instrukcije koje realizuju standardne logičke operacije: logičko množenje (*I* operacija), logičko sabiranje (*ILI* operacija), ekskluzivno sabiranje (ekskluzivno *ILI* operacija) i negaciju (*NE* operacija). Osim ovih, biće predstavljena još jedna logička instrukcija, a to je logičko upoređivanje.

U tabeli 10.5 prikazane su sledeće logičke instrukcije:

- `AND`, koja logički množi dva izvorišna operanda i rezultat smešta u odredišni operand
- `OR`, koja logički sabira dva izvorišna operanda i rezultat smešta u odredišni operand
- `XOR`, koja ekskluzivno sabira dva izvorišna operanda i rezultat smešta u odredišni operand
- `NOT`, koja komplementira vrednost izvorišnog operanda i rezultat smešta u odredišni operand
- `TST`, koja izvršava operaciju logičkog množenja nad izvorišnim operandima, dobijeni *rezultat nigde ne upisuje*, već ga samo proverava i na osnovu njega postavlja indikatore *N*, *Z*, *C* i *V* statusnog registra

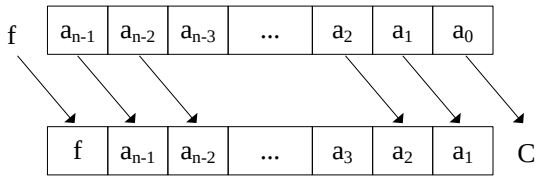
Operacija	Instrukcija	Izvorišni operandi	Odredišni operand	Napomena
logičko množenje	`AND a,b,c`	`a` i `b`	`c`	`c` nije n.a.
	`AND a,b`	`a` i `b`	`a` (ili `b`)	`a` (ili `b`) nije n.a.
	`AND a`	ACC i `a`	ACC	
	`AND`	TS i TS	TS	
logičko sabiranje	`OR a,b,c`	`a` i `b`	`c`	`c` nije n.a.
	`OR a,b`	`a` i `b`	`a` (ili `b`)	`a` (ili `b`) nije n.a.
	`OR a`	ACC i `a`	ACC	
	`OR`	TS i TS	TS	
ekskluzivno logičko sabiranje	`XOR a,b,c`	`a` i `b`	`c`	`c` nije n.a.
	`XOR a,b`	`a` i `b`	`a` (ili `b`)	`a` (ili `b`) nije n.a.
	`XOR a`	ACC i `a`	ACC	
	`XOR`	TS i TS	TS	
negacija	`NOT a,b`	`a`	`b`	`b` nije n.a.
	`NOT a`	`a`	`a`	`a` nije n.a.
	`NOT`	TS	TS	
logičko upoređivanje	`TST a,b,c`	`a` i `b`		`c` se ne koristi
	`TST a,b`	`a` i `b`		
	`TST a`	ACC i `a`		
	`TST`	TS i TS		

Tabela 10.5 Logičke instrukcije

10.3 Pomeračke instrukcije

Pomeračke instrukcije su instrukcije koje realizuju pomeranje binarne reči za jedno mesto ulevo ili udesno.

Na slici 10.1 prikazan je postupak pomeranja binarne reči $a_{n-1}...a_0$ udesno.

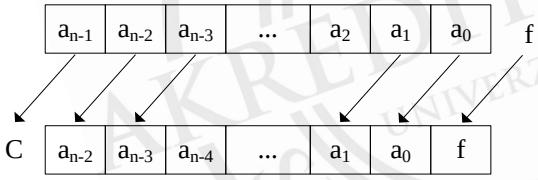


f	Instrukcija
a_{n-1}	aritmetičko pomeranje
0	logičko pomeranje
a_0	rotiranje
C	rotiranje kroz ind. C

Slika 10.1 Pomeranje binarne reči udesno

Kao što se vidi, bitovi $a_{n-1}...a_1$ se pomeraju za poziciju niže. Bit a_0 se upisuje u indikator *C* statusnog registra. Najstariji bit u rezultatu, *f*, može da dobije jednu od četiri vrednosti: a_{n-1}, 0, a_0 ili vrednost indikatora *C* (pre upisa a_0). U zavisnosti od toga koja od četiri vrednosti se upisuje u *f*, postoje četiri instrukcije pomeranja udesno: aritmetičko pomeranje udesno, logičko pomeranje udesno, rotiranje udesno i rotiranje udesno kroz indikator *C* (vrednost *f* za svaku vrstu instrukcije data je u tabeli na slici 10.1).

Na sličan način se realizuje i pomeranje binarne reči ulevo (slika 10.2).



f	Instrukcija
a_0	aritmetičko pomeranje
0	logičko pomeranje
a_{n-1}	rotiranje
C	rotiranje kroz ind. C

Slika 10.2 Pomeranje binarne reči ulevo

Bitovi $a_{n-2}...a_0$ se pomeraju za poziciju više. Bit a_{n-1} se upisuje u indikator *C* statusnog registra. Najniži bit u rezultatu, *f*, može da dobije jednu od četiri vrednosti: a_0, 0, a_{n-1} ili vrednost indikatora *C* (pre upisa a_{n-1}), tako da postoje četiri instrukcije pomeranja ulevo: aritmetičko pomeranje ulevo, logičko pomeranje ulevo, rotiranje ulevo i rotiranje ulevo kroz indikator *C* (vrednost *f* za svaku vrstu instrukcije data je u tabeli na slici).

U tabeli 10.6 prikazane su sledeće pomeračke instrukcije:

- `ASR` (*Arithmetic Shift Right*), `LSR` (*Logic Shift Right*), `ROR` (*Rotate Right*) i `RORC` (*Rotate Through Carry Right*), koje izvorišni operand pomeraju za jedno mesto udesno i rezultat smeštaju u odredišni operand

- `ASL` (*Arithmetic Shift Left*), `LSL` (*Logic Shift Left*), `ROL` (*Rotate Left*) i `ROLC` (*Rotate Through Carry Left*), koje izvorišni operand pomeraju za jedno mesto ulevo i rezultat smeštaju u odredišni operand

Operacija	Instrukcija	Izvorišni operandi	Odredišni operand	Napomena
aritmetičko pomeranje udesno	`ASR a,b,c`	`a`	`b`	`c` se ne koristi
	`ASR a,b`	`a`	`b`	
	`ASR a`	`a`	`a`	
	`ASR`	TS	TS	
logičko pomeranje udesno	`LSR a,b,c`	`a`	`b`	`c` se ne koristi
	`LSR a,b`	`a`	`b`	
	`LSR a`	`a`	`a`	
	`LSR`	TS	TS	
rotacija udesno	`ROR a,b,c`	`a`	`b`	`c` se ne koristi
	`ROR a,b`	`a`	`b`	
	`ROR a`	`a`	`a`	
	`ROR`	TS	TS	
rotacija udesno kroz indikator C	`RORC a,b,c`	`a`	`b`	`c` se ne koristi
	`RORC a,b`	`a`	`b`	
	`RORC a`	`a`	`a`	
	`RORC`	TS	TS	
aritmetičko pomeranje ulevo	`ASL a,b,c`	`a`	`b`	`c` se ne koristi
	`ASL a,b`	`a`	`b`	
	`ASL a`	`a`	`a`	
	`ASL`	TS	TS	
logičko pomeranje ulevo	`LSL a,b,c`	`a`	`b`	`c` se ne koristi
	`LSL a,b`	`a`	`b`	
	`LSL a`	`a`	`a`	
	`LSL`	TS	TS	

rotacija ulevo	`ROL a,b,c`	`a`	`b`	`c` se ne koristi
	`ROL a,b`	`a`	`b`	
	`ROL a`	`a`	`a`	
	`ROL`	TS	TS	
rotacija ulevo kroz indikator C	`ROLC a,b,c`	`a`	`b`	`c` se ne koristi
	`ROLC a,b`	`a`	`b`	
	`ROLC a`	`a`	`a`	
	`ROLC`	TS	TS	

Tabela 10.6 Pomeračke instrukcije

10.4 Instrukcije prenosa

Instrukcije prenosa su instrukcije koje realizuju prenos podatka sa jednog mesta u računaru na drugo. Podaci se mogu nalaziti na različitim mestima:

- u memorijskim lokacijama - *ML*
- u procesorskim registrima - *PR*
- u instrukciji (neposredno adresiranje) - *NAI*
- u registrima kontrolera periferija i to
  - registrima koji se tretiraju isto kao memorijske lokacije (*U*/*I* adresni prostor je memorijski preslikan) - *RKP*
  - registrima koji se tretiraju posebno (*U*/*I* i memorijski adresni prostor su razdvojeni) - *RKR*
- u akumulatoru (kod jednoadresnih instrukcija) - *AK*
- na steku (kod nulaadresnih instrukcija) - *ST*

U tabeli 10.7 prikazane su standardne instrukcije prenosa:

- `MOV`, koja izvorišni operand prenosi na odredište; izvorišni operanad može da bude u *ML*, *PR*, *RKP* ili *NAI*, a odredišni operand u *ML*, *PR* ili *RKP*
- `IN`, koja vrši prenos izvorišnog operanda iz *RKR* na odredište; koristi se kod 2-adresnih i 1-adresnih procesora sa razdvojenim *U*/*I* i memorijskim adresnim prostorima; odredišni operand je u *PR* (ako je instrukcija 2-adresna) ili u *AK* (ako je instrukcija 1-adresna)

- `OUT`, koja vrši prenos izvorišnog operanda na odredište u *RKR*; koristi se kod 2-adresnih i 1-adresnih procesora sa razdvojenim *U*/*I* i memorijskim adresnim prostorima; izvorišni operand je u *PR* (ako je instrukcija 2-adresna) ili u *AK* (ako je instrukcija 1-adresna)
- `LOAD`, koja vrši prenos izvorišnog operanda u akumulator (*AK*); izvorišni operand može da bude u *ML*, *PR*, *RKP* ili *NAI*
- `STORE`, koja obavlja prenos sadržaja akumulatora na odredište; odredišni operand se nalazi u *ML*, *PR* ili *RKP*
- `PUSH`, koja vrši prenos izvorišnog operanda na stek (*ST*); izvorišni operanad može da bude u *ML*, *PR*, *RKP* ili *NAI*
- `POP`, koja sadržaj sa vrha steka prenosi na odredište; odredišni operand može da bude u *ML*, *PR* ili *RKP*

Operacija	Instrukcija	Izvorišni operand	Odredišni operand	Napomena
prenos u/iz memorije/registara	`MOV a,b`	`a`	`b`	`a` → `b`
prenos do/od periferije	`IN a,b`	`a`	`b`	RKR→PR
	`OUT a,b`	`a`	`b`	PR→RKR
	`IN a`	`a`	ACC	RKR→ACC
	`OUT a`	ACC	`a`	ACC→RKR
prenos u/iz akumulatora	`LOAD a`	`a`	ACC	`a` →ACC
	`STORE a`	ACC	`a`	ACC→ `a`
prenos na/sa steka	`PUSH a`	`a`	TS	`a` →TS
	`POP a`	TS	`a`	TS→ `a`

Tabela 10.7 Instrukcije prenosa

10.5 Instrukcije skoka

Instrukcije skoka su instrukcije koje modifikuju sadržaj programskog brojača (*PC*) i na taj način omogućavaju nesekvencijalno izvršavanje programa, što je korisno u mnogim situacijama. Novi sadržaj *PC* predstavlja adresu memorijske lokacije u kojoj se nalazi naredna instrukcija koju treba izvršiti. Nakon izvršenja

ove instrukcije, nastavlja se sa sekvencijalnim izvršavanjem programa od date adrese uz inkrementiranje *PC*.

Instrukcije skoka se mogu klasifikovati u sledeće grupe:

- instrukcije bezuslovnog skoka
- instrukcije uslovnog skoka
- instrukcije za rad sa potprogramima
- instrukcije za rad sa prekidnim rutinama

Instrukcije bezuslovnog skoka

Instrukcije bezuslovnog skoka modifikuju *PC* uvek, bez postavljanja bilo kakvih uslova. To su sledeće instrukcije:

`JMP` *a*

Ova instrukcija se izvršava tako što se u *PC* upisuje vrednost adresnog polja *a*. Vrednost *a* predstavlja adresu memorijske lokacije u kojoj se nalazi instrukcija koju treba sledeću izvršiti. Instrukcija `JMP` se koristi kada je tokom prevođenja (kompajliranja) programa poznata adresa instrukcije na koju treba skočiti.

`JMPIND` *a*

U ovoj instrukciji, polje *a* specificira adresu memorijske lokacije u kojoj se nalazi instrukcija koju treba sledeću izvršiti. Specifikacija zavisi od korišćenog adresnog moda. Instrukcija se izvršava tako što se najpre računa adresa instrukcije, a zatim se rezultat upisuje u *PC*. Instrukcija `JMPIND` se koristi ukoliko u vreme prevođenja programa nije poznata adresa instrukcije na koju treba skočiti, već se ona prethodno računa tokom izvršavanja programa.

Sledi primer koji ilustruje razliku između navedenih instrukcija. Ako je adresa instrukcije na koju treba skočiti poznata, na pr. 145, koristi se instrukcija `JMP` 145. Po izvršenju ove instrukcije skoka, u *PC* će biti upisana vrednost 145 i procesor će preći na izvršavanje instrukcije na adresi 145. Ukoliko adresa instrukcije na koju treba skočiti nije poznata, mora se koristiti instrukcija skoka `JMPIND` *a*. U ovom slučaju, adresa se mora izračunati izvršavanjem instrukcija koje u programu prethode instrukciji `JMPIND` i smestiti na odgovarajuće mesto, na pr. u adresni registar *AR*3 ili u memorijsku lokaciju 1030. Neka je izračunata adresa 889. Ako se

ona upiše u *AR*3, onda se poljem *a* u JMPIND instrukciji mora specificirati registarsko indirektno adresiranje uz primenu *AR*3. Ako se 889 upiše u memorijsku lokaciju 1030, onda se poljem *a* mora specificirati memorijsko indirektno adresiranje i adresa 1030. U oba slučaja, nakon izvršenja instrukcije skoka, u *PC* je upisana vrednost 889.

Instrukcije uslovnog skoka

Instrukcije uslovnog skoka modifikuju *PC* samo ukoliko je ispunjen neki uslov. Opšti format ovih instrukcija je:

```
CO p
```

CO označava kod operacije (ovo je samo opšti oblik, dok su konkretni kodovi operacija dati u tabeli 10.8), a *p* vrednost pomeraja sa kojim treba napraviti relativan skok u odnosu na tekući sadržaj *PC*, samo ukoliko je uslov ispunjen. Na primer, ako je trenutna vrednost u *PC* 625, a *p* ima vrednost 12, to znači da se sledeća instrukcija koju treba izvršiti nalazi u memorijskoj lokaciji sa adresom 637.

Pošto je vrednost *p* u drugom komplementu, skok može biti unapred (ako je *p* pozitivan broj) ili unazad (ako je *p* negativan broj). Uslov za skok je specificiran kodom operacije i predstavlja proveru indikatora *N*, *Z*, *C* i *V* statusnog registra.

Instrukcije uslovnog skoka se izvršavaju u tri koraka:

- sadržaj *PC* se sabira sa *p* i tako dobija adresa memorijske lokacije u kojoj se nalazi instrukcija koju treba sledeću izvršiti (na koju treba skočiti)
- na osnovu vrednosti indikatora *N*, *Z*, *C* i *V* proverava se da li je uslov za skok ispunjen ili nije
- ako je uslov ispunjen, adresa izračunata u prvom koraku se upisuje u *PC*

U tabeli 10.8 prikazane su instrukcije uslovnog skoka.

Osim navedenih, postoje i instrukcije uslovnog skoka kod kojih se adresa instrukcije na koju treba skočiti direktno navodi u formatu instrukcije skoka, umesto pomeraja. Ove instrukcije realizuju tzv. apsolutni skok. One imaju vrlo sličan format kao instrukcije koje realizuju relativan skok (tabela 10.8) uz dve razlike:

- kodovi operacije ne počinju slovom B (*Branch*), već slovom J (*Jump*)
- *p* ne predstavlja pomeraj, već adresu instrukcije na koju treba skočiti

Primeri ovih instrukcija su: JEQL *adr*, JLSS *adr*, JNVF *adr*, itd.

Instrukcija	Opis	Uslov za skok
`BEQL` *p*	skok na jednako	Z = 1
`BNEQ` *p*	skok na nejednako	Z = 0
`BGRTU` *p*	skok na veće nego (bez znaka)	C v Z = 0
`BGREU` *p*	skok na veće nego ili jednako (bez znaka)	C = 0
`BLSSU` *p*	skok na manje nego (bez znaka)	C = 1
`BLEQU` *p*	skok na manje nego ili jednako (bez znaka)	C v Z = 1
`BGRT` *p*	skok na veće nego (sa znakom)	(N ⊕ V) v Z = 0
`BGRE` *p*	skok na veće nego ili jednako (sa znakom)	N ⊕ V = 0
`BLSS` *p*	skok na manje nego (sa znakom)	N ⊕ V = 1
`BLEQ` *p*	skok na manje nego ili jednako (sa znakom)	(N ⊕ V) v Z = 1
`BNEG` *p*	skok na N = 1	N = 1
`BNNG` *p*	skok na N = 0	N = 0
`BOVF` *p*	skok na V = 1	V = 1
`BNVF` *p*	skok na V = 0	V = 0

Tabela 10.8 Instrukcije uslovnog skoka

Instrukcije za rad sa potprogramima

Instrukcije za rad sa potprogramima čine instrukcija skoka na potprogram (`JSR`) i instrukcija povratka iz potprograma (`RTS`), čiji su opisi dati u nastavku:

`JSR` *a*

Vrednost adresnog polja *a* predstavlja adresu memorijske lokacije u kojoj se nalazi prva instrukcija potprograma. Instrukcija `JSR` se izvršava tako što se najpre tekući sadržaj *PC* prenese na stek, a zatim se vrednost *a* upiše u *PC*. Prenos na stek je neophodan zbog ispravnog povratka iz potprograma.

`RTS`

Ova instrukcija mora da bude poslednja instrukcija u potprogramu. Izvršava se tako što se sadržaj sa steka upisuje u *PC*. Očigledno je da ovaj sadržaj mora da odgovara sadržaju koji je `JSR` instrukcijom prenet na stek.

Instrukcije za rad sa prekidnim rutinama

Instrukcije za rad sa prekidnim rutinama su instrukcija skoka na prekidnu rutinu (`INT`) i instrukcija povratka iz prekidne rutine (`RTI`) koje imaju sledeće formate:

`INT` *a*

Vrednost adresnog polja *a* predstavlja adresu memorijske lokacije u kojoj se nalazi prva instrukcija prekidne rutine. Instrukcija `INT` se izvršava tako što se tekući sadržaji *PC* i statusnog registra prenesu na stek, a zatim se vrednost *a* upiše u *PC*. U nekim realizacijama mehanizma prekida, adrese prekidnih rutina se nalaze u tabeli prekida. U ovom slučaju, *a* predstavlja broj ulaza u tabeli prekida iz koga treba pročitati adresu prekidne rutine i smestiti je u *PC*.

`RTI`

Ova instrukcija mora da bude poslednja instrukcija u prekidnoj rutini. Izvršava se tako što se sadržaji sa steka upisuju u statusni registar i *PC*. Ovi sadržaji moraju da odgovaraju sadržajima koji su `INT` instrukcijom preneti na stek.

Vežbanja

P1. Šta je instrukcijski set? Koje vrste instrukcija čine standardni instrukcijski set?

P2. Navesti primere aritmetičkih i logičkih instrukcija.

P3. Koje vrste pomeranja (ulevo/udesno) mogu biti realizovane pomeračkim instrukcijama?

P4. Gde se mogu nalaziti operandi čiji prenos na drugu lokaciju može biti realizovan instrukcijama prenosa? Koje standardne instrukcije prenosa se koriste za pristup *U*/*I* uređajima, koje za pristup akumulatoru, a koje za pristup steku?

P5. Kako su klasifikovane instrukcije skoka? U čemu je razlika između instrukcija bezuslovnog i uslovnog skoka? Kako se proverava ispunjenost uslova za skok? Ako je uslov skoka ispunjen, instrukcija skoka menja sadržaj jednog procesorskog registra. Kojeg?

Z1. Registri *A*, *B*, *C* i *D* sadrže vrednosti 56, 192, 1000 i 380, respektivno. Ako se sadržaji ovih registara najpre stave na stek u redosledu *A*, *D*, *B*, *C*, a zatim se sadržaj steka pročita i smesti u *C*, *A*, *D*, *B*, koje vrednosti tada sadrže ovi registri?

Z2. Na raspolaganju je stek-orijentisan procesor koji može da izvršava instrukcije `PUSH` i `POP`, kao i 0-adresne instrukcije. Prikazati sadržaj steka po izvršenju sledećih instrukcija:

```
PUSH #5
PUSH #2
PUSH #8
SUB
PUSH #7
MUL
ADD
```

Z3. Ako pre izvršenja svake instrukcije akumulator sadrži vrednost 0x15, odrediti njegov sadržaj nakon izvršenja sledećih instrukcija:

```
ADD #0x07
CMP 0x10
AND #0xA3
LOAD #0xB5
OUT 0x33
```

Z4. Odrediti sadržaj akumulatora nakon izvršenja sledećih instrukcija, ako je njegova inicijalna vrednost 0xBA:

```
STORE 50
ASR 50
DEC 50
ROL 50
LOAD 50
```

UNIVERZITET
PRIVATNI
AKREDITOVANI
UNIVERZITET
SINGIDUNUM

11 Programiranje

Računarski program može biti napisan na različitim nivoima apstrakcije. Najvišem nivou apstrakcije odgovaraju programi pisani na višim programskim jezicima, kao što su Java ili *C++*. Ovi programi su mašinski nezavisni i za njihovo pisanje nije neophodno detaljno poznavanje arhitekture računara. Međutim, procesor ne može direktno da ih izvršava. Svaki procesor može da izvršava samo programe pisane na mašinskom jeziku koji je specifičan za njegovu arhitekturu. Programi na mašinskom jeziku predstavljaju nulti nivo apstrakcije u odnosu na arhitekturu sistema. Zbog problema pri pisanju programa na mašinskom jeziku, uveden je asemblerski jezik kao simbolička reprezentacija mašinskog jezika. On je jednostavniji za programiranje, a programi pisani na asemblerskom jeziku odgovaraju nešto višem nivou apstrakcije u odnosu na programe pisane na mašinskom jeziku.

11.1 Mašinski jezik

Mašinski jezik je kolekcija mašinskih instrukcija predstavljenih *u binarnom obliku*, tj. u vidu nizova nula i jedinica. Programi pisani na mašinskom jeziku su prilagođeni konkretnom hardveru (arhitekturi sistema) i izvršavaju se bez prevođenja. Da bi mašinski kod mogao da se izvrši, dovoljno je samo da bude smešten u memoriju i aktiviran.

Pisanje i razumevanje programa na mašinskom jeziku je teško i podložno greškama. S obzirom da se ovi programi sastoje samo od nizova binarnih cifara, svaka modifikacija predstavlja rizik od pojave grešaka jer se izvodi na nivou

bitova. Danas se praktično ne programira na mašinskom jeziku, ali je u nastavku opisan ovaj proces da bi se bolje razumeo značaj mašinskog jezika u kontekstu savremenog programiranja.

Pisanje programa na mašinskom jeziku je moguće samo ukoliko su poznati elementi arhitekture računara, kao što su: veličina i organizacija memorije, procesorski registri, formati instrukcija i instrukcijski set. Ilustracije radi, neka je na raspolaganju jednostavan hipotetički računar koji ima memoriju *M* kapaciteta 4096 8-bitnih lokacija i procesor koji među svojim 16-bitnim registrima ima akumulator (*ACC*) i registar podatka (*DR*). Neka procesor podržava instrukcijski set u kome se, između ostalih, nalaze 16-bitne instrukcije čiji je opis dat u tabeli 11.1.

Kod operacije	Mnemonik	Operand	Opis
0000	`STOP`		zaustavljanje izvršenja
0001	`LD`	`a`	M(`a`) → ACC
0010	`ST`	`a`	ACC → M(`a`)
0011	`MOVAC`		ACC → DR
0101	`ADD`		ACC + DR → ACC

Tabela 11.1 Deo instrukcijskog seta

Pretpostavimo da je za ovaj računar potrebno napisati program na mašinskom jeziku koji sabira sadržaj memorijske lokacije čija je adresa 12 sa sadržajem memorijske lokacije čija je adresa 14 i dobijeni zbir smešta u memorijsku lokaciju sa adresom 16. Traženi program ima izgled kao na slici 11.1.

```
0001 0000 0000 1100
0011 0000 0000 0000
0001 0000 0000 1110
0101 0000 0000 0000
0010 0000 0001 0000
0000 0000 0000 0000
```

Slika 11.1 Primer mašinskog koda

Kao što se vidi, program na mašinskom jeziku je težak za razumevanje i pronalaženje grešaka (debagovanje). Na slici 11.2 je prikazan isti program smešten u memoriju, počeviši od lokacije sa adresom 0. Uzeto je da se, inicijalno, u memorijskim lokacijama sa adresama 12, 14 i 16 (poslednje tri vrste) nalaze vrednosti 350, 96 i 0, respektivno.

Radi lakšeg razumevanja programa, u desnom delu slike 11.2, dati su kratak opis odgovarajućih instrukcija programa i kodovi operacija tih instrukcija. Adrese memorijskih lokacija su 12-bitne zato što je kapacitet memorije $4096=2^{12}$ lokacija. Pošto su memorijske ćelije 8-bitne, svaka 16-bitna instrukcija je smeštena u dve uzastopne ćelije, što se vidi iz redosleda navedenih adresa (0, 2, 4, 6, itd.). Ovakav program procesor može da izvrši. Rezultat izvršavanja bi bio promena sadržaja u lokaciji sa adresom 16 na novu vrednost 446.

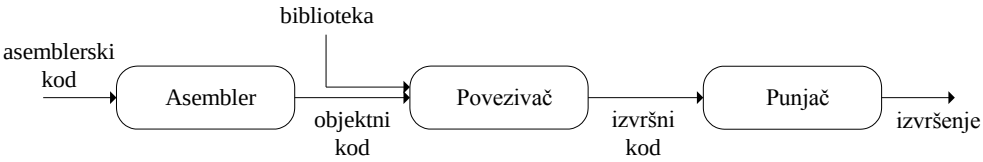
Adresa	Instrukcija	Opis	KO
0000 0000 0000	0001 0000 0000 1100	M(12) → ACC	LD: 0001
0000 0000 0010	0011 0000 0000 0000	ACC → DR	MOVAC: 0011
0000 0000 0100	0001 0000 0000 1110	M(14) → ACC	LD: 0001
0000 0000 0110	0101 0000 0000 0000	ACC + DR → ACC	ADD: 0101
0000 0000 1000	0010 0000 0001 0000	ACC → M(16)	ST: 0010
0000 0000 1010	0000 0000 0000 0000	Stop	STOP: 0000
0000 0000 1100	0000 0001 0101 1110	350	
0000 0000 1110	0000 0000 0110 0000	96	
0000 0001 0000	0000 0000 0000 0000	0	

Slika 11.2 Mašinski kod u memoriji

11.2 Asemblerski jezik

U cilju lakšeg programiranja, nakon mašinskog, uveden je asemblerski jezik. Kao i mašinski jezik, asemblerski jezik je hardverski zavisan, što znači da svaki procesor ima svoj asemblerski jezik. Asemblerske instrukcije su jednostavnije za razumevanje od mašinskih, zato što su predstavljene mnemonicima i simboličkim adresama kojima čovek može lakše da upravlja.

Da bi program napisan na asemblerskom jeziku mogao da se izvrši, neophodno je svaku njegovu instrukciju prevesti u odgovarajuću mašinsku instrukciju. Za prevođenje se koristi poseban program koji se naziva *asembler* (*assembler*). Uloga asemblera je da sve simboličke adrese u programu zameni numeričkim, simboličke kodove operacija mašinskim kodovima, rezerviše mesto za instrukcije i podatke u memoriji, predstavi konstante u mašinskom obliku, itd. Rezultat rada asemblera je odgovarajući mašinski kod. Ukoliko ima potrebe, dobijeni kod treba povezati sa bibliotekama ili drugim programima pomoću posebnog programa koji se naziva *povezivač* (*linker*). Ovako dobijeni izvršni kod se pomoću *punjača* (*loader*) smešta u memoriju i zatim se aktivira njegovo izvršavanje. Opisani postupak je prikazan na slici 11.3.



Slika 11.3 Proces pripreme asemblerskog programa za izvršenje

Asemblerski program se sastoji od niza asemblerskih instrukcija - iskaza. Svaki asemblerski iskaz se piše u posebnoj liniji programa i u opštem slučaju se sastoji od četiri polja:

Labela	Kod operacije	Operand	Komentar

Labela služi za simboličko imenovanje memorijskih adresa ili podataka. To je identifikator koji se može koristiti u drugim linijama koda za skok na liniju sa labelom. Komentar omogućava pisanje objašnjenja. Ova dva polja su opcionog karaktera. Obavezno polje predstavlja kod operacije koji ima već poznato značenje. Polje za operand je obavezno kod nekih instrukcija i označava podatak ili dodatnu informaciju za kod operacije. Sledi primer dve asemblerske instrukcije:

```
START   LD    X
        BRA   START
```

Prva instrukcija (čiji je kod operacije `LD`) kopira sadržaj iz lokacije sa adresom *X* u akumulator. `START` predstavlja labelu. Druga instrukcija podrazumeva skok (kod operacije `BRA`) na iskaz sa labelom `START`.

Asemblerski program može da sadrži i iskaze koji se ne izvršavaju, već samo služe asembleru kao komande koje utiču na njegov način rada pri prevođenju asemblerskog koda u mašinski. Ovi iskazi se nazivaju *direktivama* ili *pseudo-operacijama*. To nisu asemblerske instrukcije i ne generišu nikakav mašinski kod. Direktive omogućavaju da se isti program prevede na različite načine zavisno od parametara koje zadaje programer. Primenjuju se, na primer, za rezervaciju ili inicijalizaciju prostora za smeštaj promenljivih ili za kontrolu smeštaja programskog koda. Primer direktive `W` koja rezerviše 16-bitnu reč u memoriji za labelu `X` i inicijalizuje je na 120 je:

```
X   W    120
```

Sledi primer programa napisanog na asemblerskom jeziku (slika 11.4). To je program koji može da se izvrši (nakon asembliranja) na istom hipotetičkom računaru koji je prethodno opisan u poglavlju 11.1. a rešava i isti zadatak (sabira sadržaje dve memorijske lokacije i rezultat smešta u treću lokaciju).

```
     LD X          \  ACC ← X
     MOVAC         \  DR ← ACC
     LD Y          \  ACC ← Y
     ADD           \  ACC ← ACC + DR
     ST Z          \  Z ← ACC
     STOP
X    W  350  \ rezervacija reci inicijalizovane na 350
Y    W   96  \ rezervacija reci inicijalizovane na 96
Z    W    0  \ rezervacija reci inicijalizovane na 0
```

Slika 11.4 Primer asemblerskog koda

Kao što se vidi, ovaj program je jednostavniji za razumevanje i testiranje od ranije datog programa na mašinskom jeziku. Jedina razlika je u tome što ovde nije eksplicitno definisano u kojim lokacijama se nalaze podaci (ranije su to bile fiksne lokacije sa adresama 12, 14 i 16), već se lokacije rezervišu tokom asembliranja. Posebnu pogodnost kod asemblerskog programiranja predstavlja mogućnost pisanja komentara (\).

Asemblerski jezici imaju primenu pri izradi sistemskih programa kada je potrebna direktna interakcija sa hardverom, pri izradi aplikacija sa ograničenim resursima, kod sistema sa posebnom namenom (*embedded systems*), itd. Programiranje na asemblerskom jeziku može rezultovati mašinskim kodom koji je kraći i brži od koda nastalog primenom viših programskih jezika.

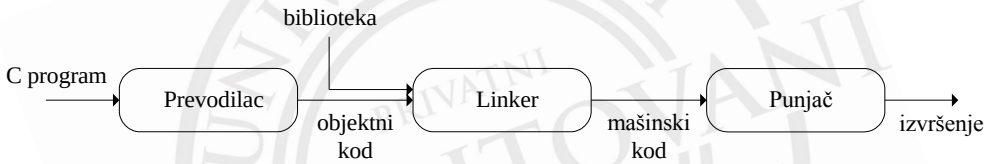
Prednosti programiranja u asemblerskom jeziku su:

- jasniji proces izvršavanja instrukcija
- prikaz načina na koji se podaci čuvaju u memoriji
- vidljiva interakcija između programa i operativnog sistema, procesora, *U/I* jedinica,...
- lakše kasnije programiranje na višim programskim jezicima zbog poznavanja procesa u računaru

11.3 Viši programski jezici

Viši programski jezici, kao što su *C*, *Visual Basic*, *Pearl*, *PHP*, *Python* i drugi, podrazumevaju korišćenje promenljivih, nizova, objekata, složenih aritmetičkih i logičkih izraza, petlji, funkcija, potprograma i sl. Po tome se razlikuju od mašinskih ili asemblerskih jezika koji se direktno obraćaju hardveru računara, na primer, memorijskim lokacijama, registrima, steku, itd.

Da bi program napisan na nekom višem programskom jeziku mogao da se izvrši, najpre se mora prevesti u objektni kod pomoću posebnog programa koji se naziva *prevodilac* (*compiler*). Moduli objektnog koda se zatim povezuju u celinu pomoću linkera. Tako se dobija mašinski (izvršni) kod koji se može uneti u memoriju i izvršiti. Na slici 11.5 prikazan je proces pripreme programa pisanog na višem programskom jeziku za izvršenje.



Slika 11.5 Priprema programa pisanog na *C* programskom jeziku za izvršenje

Viši programski jezici imaju sintaksu koja poseduje elemente prirodnog jezika, pa su zato jednostavni za programiranje. Na primer, ranije razmatrano sabiranje sadržaja dve memorijske lokacije i smeštaj rezultata u treću, u programskom jeziku *C* se realizuje na sledeći način:

```
int a=360, b=96;
int c=a+b;
```

Prednosti programiranja u višim programskim jezicima su:

- veća ekspresivnost i konciznost
- potrebno je manje vremena za razvoj softvera
- lakša verifikacija koda, otkrivanje i ispravljanje grešaka
- lakše održavanje programskog koda
- veća mogućnost prenosivosti koda na druge platforme
- veća pouzdanost i sigurnost

Vežbanja

P1. Kako izgleda program napisan u mašinskom jeziku? Šta je potrebno obezbediti da bi ovaj program mogao da se izvrši? Navesti prednosti i nedostatke programiranja na mašinskom jeziku.

P2. U čemu su sličnosti, a u čemu razlike između mašinskog i asemblerskog jezika?

P3. Objasniti postupak pripreme asemblerskog koda za izvršenje. Dati sliku. Šta predstavlja asembler? Koja je njegova uloga?

P4. Prikazati opšti format jedne asemblerske instrukcije i objasniti ga. Da li se sve asemblerske komande izvršavaju? Ako ne, koje se ne izvršavaju i kakva je njihova uloga?

P5. Navesti prednosti programiranja na asemblerskom jeziku. Dati primere primene asemblerskog jezika.

P6. U čemu je osnovna razlika između viših programskih jezika i asemblerskog jezika?

P7. Objasniti postupak pripreme za izvršenje koda pisanog na nekom višem programskom jeziku. Dati sliku. Navesti prednosti programiranja na višim programskim jezicima.

Z1. Primenom standardnog instrukcijskog seta, napisati program na asemblerskom jeziku koji računa vrednost izraza $5\cdot(a + b) - 2$, gde su a i b simbolička imena za memorijske reči na adresama 800 i 802. Rezultat smestiti u registar sa adresom *AX*.

Z2. Napisati program na asemblerskom jeziku koji izvršava operaciju množenja $Z \leftarrow X\cdot Y$. *X*, *Y* i *Z* predstavljaju adrese memorijskih lokacija u kojima se nalaze vrednosti 5, 15 i 0, respektivno. Koristiti standardan instrukcijski set.

Z3. Napisati program koji učitava niz pozitivnih celih brojeva iz opsega [0,500] i upisuje ih sukcesivno u 8-bitne memorijske lokacije počevši od adrese 1000.

Unos niza se završava kada se učita vrednost 0. Program, takođe, treba da izračuna zbir unetih brojeva i upiše ga u lokaciju sa adresom 950. Na raspolaganju su 16-bitni procesorski registri *AX*, *BX* i *CX*. *U*/*I* uređaju se pristupa preko 16-bitnog registra *R*.

Z4. Primenom standardnog skupa instrukcija napisati program koji pronalazi najmanju i najveću vrednost upisanu u memorijske lokacije od adrese 50 do adrese 100 i smešta ih na stek. Smatrati da se u ćelijama nalaze pozitivni celi brojevi. Na raspolaganju su registri opšte namene *AX*, *BX*, *CX* i *DX*.

Z5. Euklidov algoritam za određivanje najvećeg zajedničkog delioca (*NZD*) realizovan je u višem programskom jeziku *C* na sledeći način:

```
int nzd_Euklid(int x, int y) {
   while (x != y) {
   if (x > y)
        x = x – y;
   else
        y = y – x;
   }
   return x;
}
```

Napisati ovaj program na asemblerskom jeziku, ako se parametri *x* i *y* učitavaju iz periferije, a rezultat se, takođe, šalje u periferiju. Proveriti realizaciju asemblerskog programa na primeru $x = 9$, $y = 6$.

Z6. Data su dva niza, *A* i *B*, sa jednakim brojem članova *N*, $N < 100$. Oba niza su smeštena u memoriju, i to niz *A* počevši od lokacije sa adresom 100, a niz *B* od lokacije od adresom 200. Napisati program koji računa izraz:

$$D = \sum_{i=0}^{N-1} A_i \cdot B_i$$

gde su A_i i B_i *i*-ti članovi nizova *A* i *B*. Zbog veće efikasnosti, koristiti postinkrement registarsko indirektno adresiranje.

Z7. Grupa od *n* studenata je radila tri testa: *T*1, *T*2 i *T*3. Rezultati na testovima su uneti u memoriju kao na slici:

Adresa	Sadržaj	
...	...	
N	n	← broj studenata
REZ	ID studenta	← Student 1
REZ+1	Poeni T1	← rezultat na testu T1
REZ+2	Poeni T2	← rezultat na testu T2
REZ+3	Poeni T3	← rezultat na testu T3
REZ+4	ID studenta	Student 2
REZ+5	Poeni T1	
REZ+6	Poeni T2	
REZ+7	Poeni T3	
...	...	Student 3

Napisati program u asemblerskom jeziku koji za svaki test pojedinačno računa prosečan broj poena koji su studenti ostvarili na tom testu i tako dobijene rezultate upisuje u lokacije *RT*1, *RT*2 i *RT*3.

Z8. Računanje aritmetičke sredine tri broja koja se nalaze u memoriji na adresama *A*1, *A*2 i *A*3, realizovano je pomoću programa na asemblerskom jeziku:

```
MOV A1,R
ADD R,A2
ADD R,A3
DIV R,3
```

Naći grešku u datom asemblerskom kodu.

Z9. Dat je asemblerski kod koji od tri neoznačena cela broja na adresama *A*1, *A*2 i *A*3 pronalazi najveći i upisuje ga na stek:

```
        CMP A1,A2
        BGRTU LAB1
        CMP A2,A3
        BGRTU LAB3
        PUSH A3
LAB3    PUSH A2
LAB1    CMP A1,A3
        BGRTU LAB2
        PUSH A3
LAB2    PUSH A1
```

```
EXIT   STOP
```

Naći greške u datom kodu.

12 Memorijski sistem

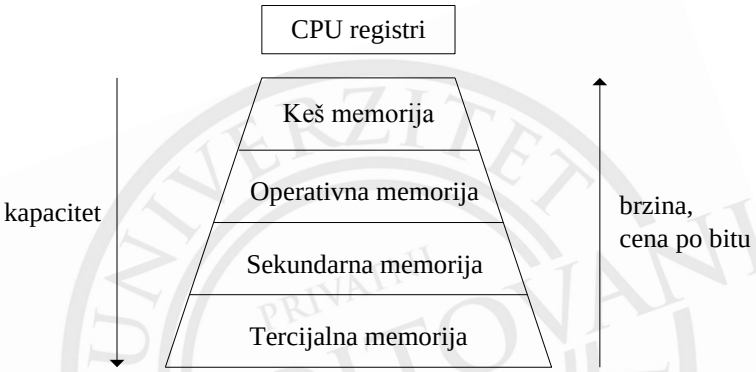
Memorije služe za smeštaj i čuvanje podataka i programa. U računaru postoji više vrsta memorija sa različitim ulogama. One čine memorijski sistem računara. Memorije su organizovane u *hijerarhiju* na čijem vrhu se nalaze brze i skupe memorijske jedinice malog kapaciteta, dok su na dnu sporije i jeftinije jedinice koje imaju veliki kapacitet. Ideja o hijerarhijskoj organizaciji memorija potiče još iz 1946.g. a predložio ju je *Von Neumann*.

Hijerarhijska organizacija memorijskog sistema prikazana je na slici 12.1. Na vrhu hijerarhije se nalaze procesorski registri koji omogućavaju čuvanje podataka unutar samog procesora. U registrima se nalaze podaci kojima se najbrže pristupa. Zatim sledi brza i skupa keš memorija relativno malog kapaciteta, kao prelaz ka operativnoj memoriji znatno većeg kapaciteta, ali manje brzine. Na sledećem nivou su sekundarne memorije, kao što su hard disk, optički diskovi (*CD*, *DVD*) i *USB flash*. Na dnu hijerahije su tercijalne memorije, obično magnetne trake, koje se danas vrlo retko koriste.

Na slici su strelicama označeni parametri po kojima je uspostavljena hijerarhija: brzina, kapacitet i cena po bitu. Smer strelice ukazuje na porast određenog parametra.

Poznavanje memorijske hijerarhije je važno zato što može značajno da utiče na performanse napisanog softvera. Na primer, ukoliko su podaci koje softver koristi smešteni u bržim memorijskim jedinicama, program će se brže izvršavati. Efikasnost hijerarhijske organizacije zasniva se na principu ne tako čestog prenošenja podataka u bržu memoriju uz njihovo višestruko korišćenje pre nego što se zamene novim podacima. Ovaj princip je ostvarljiv zahvaljujući fenomenu lokalnosti referenci. *Lokalnost referenci* potiče od činjenice da, u datom

vremenskom periodu, programi često pristupaju nekom određenom delu memorije. Postoje dva oblika lokalnosti: prostorna i vremenska lokalnost. *Prostorna lokalnost* je pojava da, ukoliko se program referiše na jednu adresu, postoji velika verovatnoća da će se uskoro referisati i na obližnje adrese u memoriji (na primer, na uzastopne instrukcije pri sekvencijalnom izvršavanju programa). *Vremenska lokalnost* je fenomen da ako se program referiše na neku adresu u memoriji, postoji velika verovatnoća da će joj uskoro opet pristupiti (na primer, nekoj instrukciji u programskoj petlji).



Slika 12.1 Memorijska hijerarhija

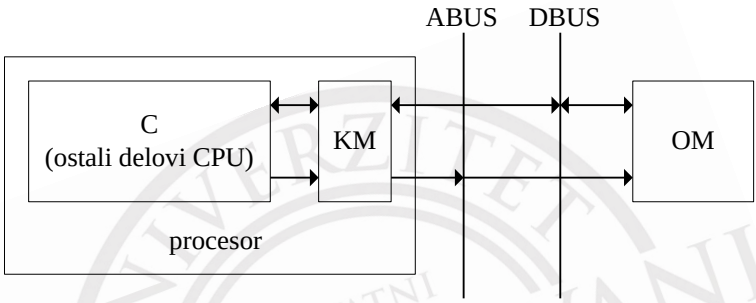
12.1 Keš memorija

Ideju o keš memoriji je prvi predstavio *Wilkes* 1965.g, dok je sam termin nastao kasnije. Uvođenje keš memorije je omogućilo brži pristup sadržaju operativne memorije, što je dovelo do značajnog ubrzanja rada računara. Suština ideje je da se mali deo operativne memorije za koji se očekuje da će uskoro biti korišćen, kopira u keš memoriju, koja je znatno brža od operativne, a nalazi se unutar procesora ili vrlo blizu njemu. Kada procesor zahteva neki podatak, najpre se proverava da li se on nalazi u keš memoriji, pa ako se nalazi, podatak se uzima iz keš memorije (bez obraćanja operativnoj memoriji) čime se postižu značajne vremenske uštede.

Na slici 12.2 prikazan je princip rada keš memorije (*KM*) koja se nalazi unutar procesora. Keš memorija je putem linija podataka i adresnih linija povezana sa ostalim komponentama procesora (označenim sa *C*), kao i sa operativnom memorijom (*OM*).

Kada procesor želi da pristupi nekom podatku, *C* generiše adresu lokacije u *OM* u kojoj se nalazi taj podatak i prosleđuje je do *KM* gde se proverava da li se

traženi podatak nalazi u *KM*. Pošto je u početnom trenutku *KM* prazna, podatak se ne nalazi u *KM*, već u *OM*. Zato se taj podatak, kao i podaci na susednim adresama u *OM*, prenose (u vidu bloka podataka) u *KM*. Nakon toga, podatak se čita iz *KM* i prosleđuje procesoru. Ovaj postupak se ponavlja za sve podatke potrebne procesoru. Postepeno, *KM* se puni podacima, i u jednom trenutku će zahtevani podatak biti u *KM*. Tada će on moći direktno da se pročita iz *KM*, bez obraćanja *OM*, pa će pristup podatku biti brži.



Slika 12.2 Princip rada keš memorije

Kao što se vidi iz opisanog postupka, procesor generiše jednu adresu za pristup podatku. Međutim, podatak se može nalaziti ili u *KM* ili u *OM*, očigledno na različitim adresama zbog razlika u kapacitetima ove dve memorije. Prema tome, da bi se pristupilo podatku, potrebno je obezbediti prevođenje, tj. mapiranje adrese u skladu sa memorijom kojoj se pristupa. Funkciju mapiranja obavlja jedinica za upravljanje memorijom (*MMU – Memory Management Unit*), koja se obično implementira kao deo *CPU*, mada može da bude i posebno integrisano kolo.

Da bi proces keširanja podataka bio moguć, potrebno je rešiti tri problema:

- *problem evidencije blokova OM koji se trenutno nalaze u KM*; da bi se po zadatoj adresi pristupilo podatku, neophodno je znati da li je blok *OM* u kome se podatak nalazi ranije prebačen u *KM* i, ako jeste, gde je smešten; ovaj problem je rešen uvođenjem *tehnika preslikavanja* koje precizno evidentiraju sadržaj *KM* u svakom trenutku
- *problem odlučivanja o tome koji blok treba izbaciti iz pune KM* da bi se oslobodio prostor za upis novog bloka *OM*; ovaj problem rešavaju *tehnike zamene*
- *problem ažuriranja sadržaja OM* u slučaju promene sadržaja *KM*; pošto se isti blok podataka nalazi i u *OM* i u *KM*, svaka izmena sadržaja u *KM* mora se reflektovati i na *OM* radi održanja njihove konzistentnosti; ovaj problem rešavaju *tehnike ažuriranja*

12.1.1 Tehnike preslikavanja

Mehanizam rada *KM* zahteva da ona bude podeljena u blokove koji se sastoje od određenog broja uzastopnih memorijskih lokacija. Broj lokacija u bloku se naziva dužinom bloka. Na sličan način je podeljena i *OM*. Dužine blokova *OM* i *KM* su iste. Blokovi obe memorije su numerisani počevši od 0.

Tehnike preslikavanja definišu postupak unosa bloka iz *OM* u *KM*. Pri upisu bloka *OM* u blok *KM*, redosled lokacija unutar blokova se ne menja. U ovom poglavlju opisane su tri tehnike preslikavanja: direktno preslikavanje, asocijativno preslikavanje i set-asocijativno preslikavanje.

Direktno preslikavanje

Direktno preslikavanje je najjednostavnija tehnika preslikavanja. Zasniva na tome da se određeni *blok OM smešta uvek u isti blok KM*.

Pretpostavimo da je u *KM* potrebno upisati *i*-ti blok *OM*. Neka je *NCB* ukupan broj blokova u *KM*. Broj bloka *j* u *KM* u koji treba upisati dolazeći blok *OM* se računa po formuli:

$$j = i \bmod NCB$$

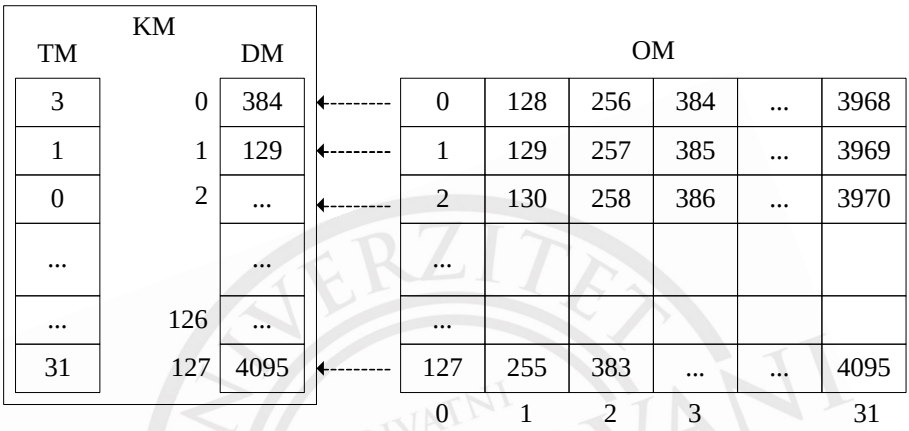
Kao što se vidi, više blokova *OM* se preslikava u isti blok *KM*, pa je ovo *više-na-jedan* tehnika mapiranja. Na primer, za *NCB* = 8, blokovi 14, 22 i 30 iz *OM* se upisuju u blok 6 u *KM* (zato što je 14 mod 8 = 22 mod 8 = 30 mod 8 = 6).

Pošto za funkcionisanje keš mehanizma nije dovoljno samo čuvati blokove podataka u *KM*, već se o njima mora voditi i evidencija o poreklu, *KM* je podeljena na dva dela. U prvom delu, koji se naziva *memorija tagova* (*TM – Tag Memory*), čuvaju se podaci o evidenciji blokova, dok se u drugom delu, koji se naziva *memorija podataka* (*DM – Data Memory*), čuvaju blokovi podataka iz *OM*.

Sledi primer direktnog preslikavanja (slika 12.3). Neka *OM* sadrži *NMB* = 4096 blokova, a *KM* 128 blokova (*NCB*). Veličina bloka je *B* = 16 memorijskih lokacija (reči). Raspored blokova u memorijama prikazan je slici 12.3. Blokovi *OM* su organizovani u 128 vrsta po 32 bloka (4096/128 = 32). *DM* se sastoji od 128 blokova (označenih od 0 do 127) u koje su upisani blokovi *OM* 384, 129, ..., 4095.

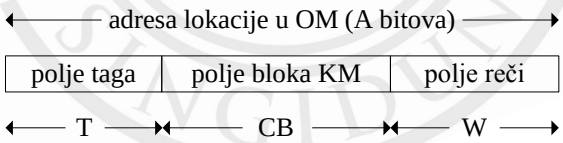
Prema ranije datoj formuli, svi blokovi iz prve vrste u *OM* se preslikavaju u blok 0 u *DM*. Takođe, svi blokovi iz druge vrste *OM* se preslikavaju u blok 1 u *DM*, itd. S obzirom da se više blokova *OM* iz jedne vrste može upisati u isti blok *KM*, neophodno je uvek imati evidenciju o tome koji blok iz vrste je trenutno upisan. To se postiže pomoću *TM*. Za svaki blok u *KM* postoji odgovarajuća

lokacija u *TM* koja sadrži redni broj bloka *OM* u odgovarajućoj vrsti koji se trenutno nalazi u *KM* (broj kolone). Ovaj redni broj se naziva tag. Na primer, sadržaj prve lokacije u *TM* je 3 zato što se u bloku 0 u *KM* nalazi blok 384 iz *OM* koji je u trećoj koloni *OM* (brojanje kolona počinje od 0).



Slika 12.3 Primer direktnog preslikavanja

Kod direktnog preslikavanja, adresu podatka dobijenu od procesora, *MMU* interpretira tako što je podeli u tri polja (slika 12.4): *polje taga* (*T* bitova), *polje bloka KM* (*CB* bitova) i *polje memorijske reči unutar bloka* (*W* bitova).



Slika 12.4 Interpretacija adrese kod direktnog preslikavanja

Dužine polja (u broju bitova) određuju se na sledeći način:

$T = \log_2 (NMB/NCB)$

$CB = \log_2 NCB$

$W = \log_2 B$

Dužina adrese (*A* bitova) memorijske lokacije u *OM* koju je generisao procesor, računa se kao:

$A = \log_2 (B \cdot NMB)$

U razmatranom primeru, primenom navedenih formula dobija se da je:

$T = \log_2 (4096/128) = 5 \qquad CB = \log_2 128 = 7 \qquad W = \log_2 16 = 4$

$A = \log_2 (16 \cdot 4096) = 16$

Kao što se vidi, dobijeno je da važi $T + CB + W = A = 16$, pa je adresa lokacije u *OM* 16-bitna.

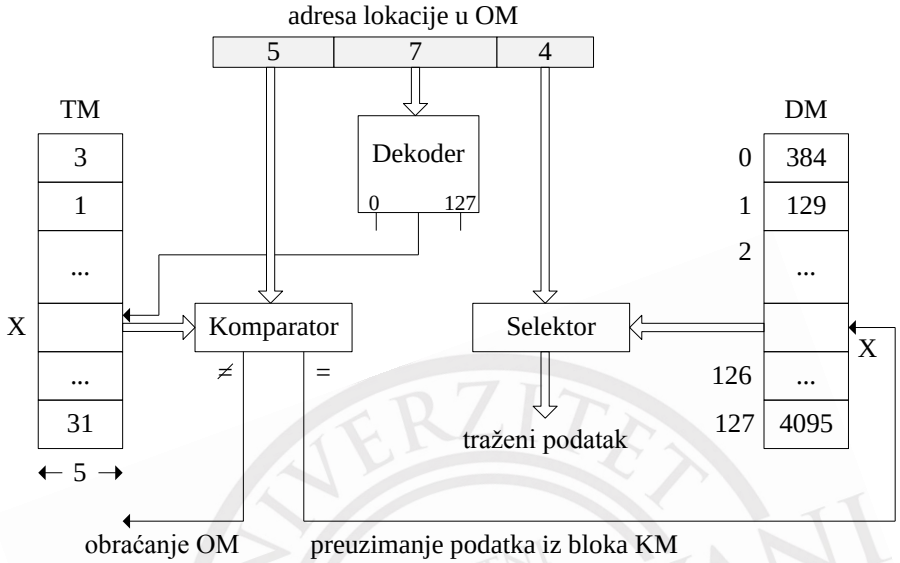
U nastavku će, na razmatranom primeru, biti pokazano da je adresa koju generiše procesor istovremeno i adresa podatka u *OM* i adresa tog podatka u *KM* (ukoliko se on tamo nalazi).

Neka je procesor generisao 16-bitnu adresu $1011010011000101_{(2)} = 46277_{(10)}$. Blokovi u *OM* i *KM*, pozicije reči unutar bloka, kao i tagovi, broje se počevši od 0. U *OM*, podatak se nalazi u bloku 2892 na poziciji 5 unutar bloka od 16 reči, jer je 46278 / 16 = 2892 (6). Blok 2892 u *OM* se direktno preslikava u blok 76 u *KM*, zato što je 2893 mod 128 = 77. Vrednost taga za blok 2892 u *OM* je 22, jer je 2893/128 = 22 (77). S druge strane, ako se zadata adresa podeli u tri polja od 5, 7 i 4 bita, dobija se da polje taga sadrži vrednost $10110_{(2)} = 22_{(10)}$, polje bloka *KM* vrednost $1001100_{(2)} = 76_{(10)}$, a polje memorijske reči vrednost $0101_{(2)} = 5_{(10)}$. Kao što se vidi, ovako dobijene vrednosti odgovaraju prethodno izračunatim, što znači da se preko iste adrese pristupa podatku i u *OM*, i u *KM*.

Kada od procesora dobije adresu podatka, *MMU* je tumači kao strukturu od tri polja (*T*-polje, *CB*-polje, *W*-polje) i izvršava sledeći protokol:

- na osnovu *CB*-polja određuje blok u *KM* u kome bi trebalo da se nalazi podatak
- poredi sadržaj *T*-polja sa odgovarajućom vrednošću u *TM* za taj blok; ako su vrednosti iste, podatak se nalazi u *KM*, a ako nisu, onda je podatak u *OM*
- ako je podatak u *KM*, na osnovu sadržaja *W*-polja određuje se njegova pozicija u bloku, pa mu se može pristupiti
- ako podatak nije u *KM*, potrebno je blok iz *OM* preneti u *KM*, podatak proslediti procesoru i ažurirati *TM* i *DM*

Za razmatrani primer, opisani protokol je dat na slici 12.5.



Slika 12.5 Postupak direktnog preslikavanja - primer

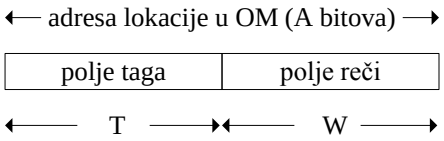
Kao što se vidi na slici, dekoder na osnovu sadržaja *CB*-polja određuje blok u *DM* (*X*) i njemu odgovarajući ulaz u *TM* (takođe *X*). Komparator poredi tag pročitan iz *X* u *TM* sa sadržajem *T*-polja. Ako su vrednosti jednake, podatak je u bloku *KM*. Selektor izdvaja traženi podatak iz bloka *X* u *DM* na osnovu sadržaja *W*-polja. Ukoliko podatak nije u *KM*, realizuje se obraćanje *OM*.

Prednost direktnog preslikavanja je u jednostavnosti postupka i direktnom određivanju mesta u *KM* gde će biti smešten blok *OM*. Nedostatak je u neefikasnom korišćenju *KM*. Naime, česte su situacije da više blokova konkuriše za isto mesto u *KM*, a da su neki drugi blokovi u *KM* prazni.

Asocijativno preslikavanje

Asocijativno preslikavanje je fleksibilnije od direktnog preslikavanja, zato što se kod ove tehnike *blok OM može smestiti u bilo koji slobodan blok KM*.

U slučaju asocijativnog preslikavanja, adresa koju generiše procesor se interpretira pomoću dva polja (slika 12.6): *polja taga* (*T* bitova) i *polja memorijske reči* (*W* bitova).



Slika 12.6 Interpretacija adrese kod asocijativnog preslikavanja

Polje taga jednoznačno identifikuje blok *OM* koji se nalazi u *KM*, dok polje memorijske reči identifikuje traženi podatak unutar bloka.

Dužine polja (u broju bitova) se određuju na sledeći način:

$T = \log_2 NMB$ (*NMB* je broj blokova u *OM*)

$W = \log_2 B$ (*B* je broj memorijskih reči u bloku)

Dužina adrese (*A* bitova) memorijske lokacije u *OM* koju je generisao procesor, računa se kao:

$A = \log_2 (B \cdot NMB)$

Primenom ovih formula u ranije razmatranom primeru dobija se da je:

$T = \log_2 4096 = 12$ $W = \log_2 16 = 4$

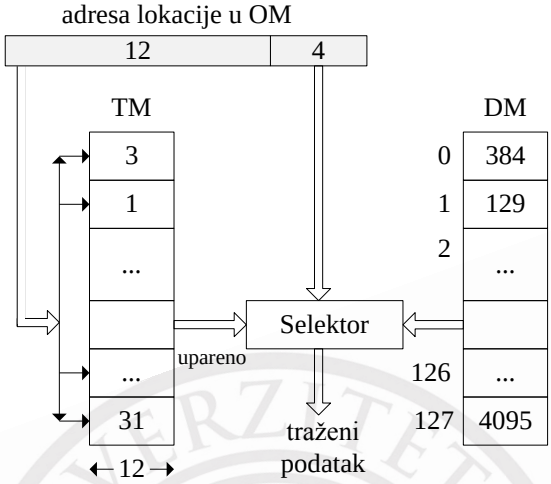
$A = \log_2 (16 \cdot 4096) = 16$

S obzirom da važi $T + W = A = 16$, adresa lokacije u *OM* je 16-bitna.

Kada od procesora dobije adresu podatka, čija je struktura (*T*-polje, *W*-polje), *MMU* izvršava sledeći protokol:

- ispituje da li se vrednost *T*-polja nalazi među postojećim tagovima u *TM*; ako se nalazi, podatak je u *KM*, inače je u *OM*
- ako je u *KM*, podatak se nalazi u bloku *KM* koji odgovara tagu, pa mu se može pristupiti na osnovu sadržaja *W*-polja
- ako podatak nije u *KM*, potrebno je blok iz *OM* preneti u *KM*, podatak proslediti procesoru i ažurirati *TM* i *DM*

Na slici 12.7 prikazan je navedeni protokol za razmatrani primer.



Slika 12.7 Postupak asocijativnog preslikavanja - primer

Suštinu asocijativnog preslikavanja predstavlja pretraživanje tagova u *TM*. Ukoliko bi ovo pretraživanje bilo sekvencijalno (redom), trajalo bi neprihvatljivo dugo. Zato se *TM* realizuje kao asocijativna memorija koja omogućava paralelno (istovremeno) pretraživanje, kao što je prikazano na slici. Paralelno pretraživanje zahteva dodatni hardver, što poskupljuje realizaciju i predstavlja glavni nedostatak tehnike asocijativnog preslikavanja.

Prednost asocijativnog preslikavanja je u efikasnom korišćenju *KM*, jer nema restrikcija po pitanju smeštaja dolazećeg bloka *OM*.

Set-asocijativno preslikavanje

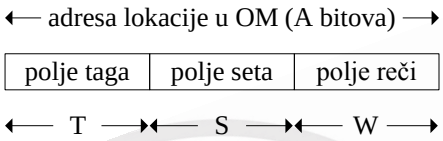
Set-asocijativno preslikavanje predstavlja kompromis između direktnog i asocijativnog preslikavanja. Ideja je da se zadrže jednostavnost direktnog i efikasnost asocijativnog preslikavanja tako što bi se *KM* podelila u setove sa određenim brojem blokova. *Blok OM bi se uvek preslikavao u isti set u KM* (slično direktnom preslikavanju), *ali u bilo koji slobodan blok u tom setu* (slično asocijativnom preslikavanju).

Pretpostavimo da je *KM* podeljena u *NS* setova. Blok *i* u *OM* se mapira u set *s* *KM* prema formuli:

$$s = i \bmod NS$$

Unutar seta *s*, *i*-ti blok *OM* se može smestiti u bilo koji blok *KM* koji pripada tom setu.

Kod ove tehnike, adresa podatka koju generiše procesor se interpretira pomoću tri polja (slika 12.8): *polja taga* (*T* bitova), *polja seta* (*S* bitova) i *polja memorijske reči* (*W* bitova). Polje taga identifikuje blok u setu, polje seta identifikuje set, dok polje memorijske reči identifikuje traženi podatak unutar bloka.



Slika 12.8 Interpretacija adrese kod set-asocijativnog preslikavanja

Ako je broj blokova u setu *BS*, dužine polja (u broju bitova) određuju se na sledeći način:

$T = \log_2 (NMB \cdot BS/NCB)$ (*NMB* je broj blokova u *OM*, a *NCB* u *KM*)
$S = \log_2 NS$
$W = \log_2 B$ (*B* je broj memorijskih reči u bloku)

Dužina adrese (*A* bitova) memorijske lokacije u *OM* koju je generisao procesor, računa se kao:

$A = \log_2 (B \cdot NMB)$

Primenom ovih formula u razmatranom primeru, uz *BS* = 4 i *NS* = 128/4 = 32, dobija se da je:

$T = \log_2 (4096 \cdot 4/128) = 7$ $S = \log_2 32 = 5$ $W = \log_2 16 = 4$
$A = \log_2 (16 \cdot 4096) = 16$

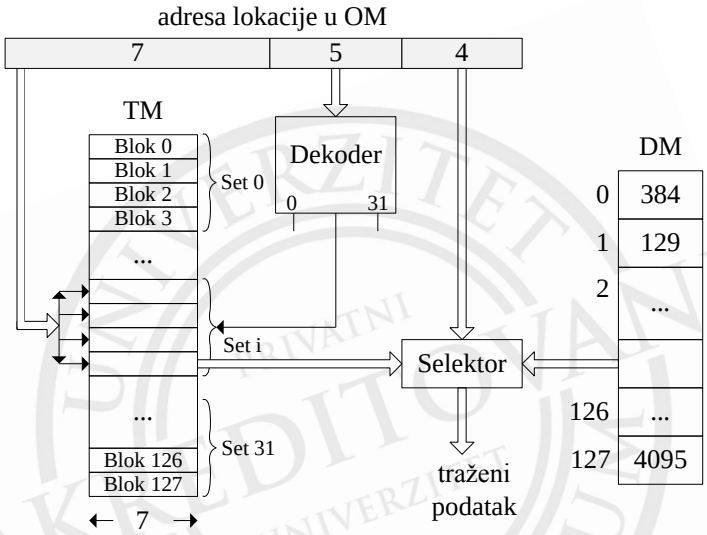
S obzirom da važi $T + S + W = A = 16$, adresa lokacije u *OM* je 16-bitna.

Kada dobije adresu podatka, čija je struktura (*T*-polje, *S*-polje, *W*-polje), *MMU* izvršava sledeći protokol:

- na osnovu sadržaja *S*-polja direktno određuje set u koji se blok *OM* mapira
- ispituje da li se sadržaj *T*-polja nalazi među postojećim tagovima u *TM* za taj set; ako se nalazi, podatak je u *KM*, inače je u *OM*

- ako je podatak u *KM*, pronalazi se unutar odgovarajućeg bloka u *DM* na osnovu vrednosti u *W*-polju
- ako podatak nije u *KM*, potrebno je blok iz *OM* preneti u *KM*, podatak proslediti procesoru i ažurirati *TM* i *DM*

Na slici 12.9 prikazan je opisani protokol za razmatrani primer.



Slika 12.9 Postupak set-asocijativnog preslikavanja - primer

Kao što se vidi, dekoder na osnovu sadržaja S-polja određuje set i njemu odgovarajuće ćelije sa tagovima u *TM*. Zatim se asocijativnim pretraživanjem ispituje da li u ovim ćelijama postoji vrednost iz *T*-polja. Ako postoji, na osnovu taga se određuje blok u setu u kome se nalazi podatak. Selektor izdvaja traženi podatak iz odgovarajućeg bloka u *DM* na osnovu sadržaja *W*-polja. Ukoliko podatak nije u KM, realizuje se obraćanje *OM*.

Opisana tehnika nije toliko efikasna kao asocijativno preslikavanje, ali je preuzela jednostavnost direktnog preslikavanja.

12.1.2 Tehnike zamene

Tehnike zamene rešavaju problem izbora bloka koji će biti izbačen iz pune *KM* kako bi se na njegovo mesto upisao dolazeći blok *OM*. Potreba za tehnikama zamene postoji samo kod asocijativnog i set-asocijativnog preslikavanja, dok se u slučaju direktnog preslikavanja unapred zna lokacija upisa dolazećeg bloka.

Izbor bloka koji će biti zamenjen se može sprovesti:

- po slučajnom izboru
- po vremenu koje su upisani blokovi proveli u *KM* (*FIFO* tehnika)
- po trenucima poslednjeg korišćenja blokova upisanih u *KM* (*LRU* tehnika)

Tehnika slučajanog izbora

U računaru postoji *generator slučajnih brojeva* (*random generator*) koji po uključenju računara počne da generiše brojeve iz zadatog opsega. Tehnika slučajnog izbora koristi izlaz ovog generatora u trenutku zamene na osnovu koga određuje broj bloka *KM* koga treba zameniti blokom *OM*. Ova tehnika je prvi put bila upotrebljena u *Intel*-ovoj *iAPX* seriji mikroprocesora. Tehnika je vrlo jednostavna, a njen glavni nedostatak je u tome što ne uzima u obzir lokalnost referenci, pa se može desiti da se izbaci blok koji bi trebalo uskoro da se koristi.

FIFO tehnika

FIFO (*First-In-First-Out*) tehnika kao kriterijum uvodi vreme koje su blokovi proveli u *KM* od svog unosa to tekućeg trenutka. Izbor se pravi tako što se iz *KM izbacuje onaj blok koji je najviše vremena proveo u njoj*. Dakle, to je blok koji je, od svih trenutno raspoloživih blokova u *KM*, prvi unet. Za sprovođenje ove tehnike, neophodno je vođenje evidencije o vremenima upisa blokova u *KM*, što ovu tehniku čini složenijom od tehnike slučajnog izbora. *FIFO* tehnika je pogodna za programe koji se pravolinijski izvršavaju jer kod njih lokalnost referenci nije bitna.

LRU tehnika

Najefikasniji postupak zamene je *LRU* (*Least Recently Used*) tehnika. Ona se zasniva na praćenju istorije korišćenja blokova koji se nalaze u *KM*, za šta je zadužen *keš kontroler*. Kriterijum za izbacivanje bloka iz *KM* je vreme njegovog poslednjeg korišćenja. Naime, iz *KM* se *izbacuje blok sa najranijim vremenom poslednjeg korišćenja*.

Keš kontroler može da prati istoriju korišćenja blokova na različite načine. Jedna moguća realizacija je da se svakom bloku *KM* pridruži brojač. Vrednost brojača je veća ako je blok ranije korišćen. Radi lakšeg razumevanja realizacije, može se smatrati da se brojači nalaze u listi po redosledu korišćenja njima odgovarajućih blokova. Na čelu liste je brojač bloka koji je poslednji bio korišćen (ovaj brojač ima najmanju vrednost). Kada procesor želi da pristupi nekom

podatku, najpre se proverava da li se blok sa tim podatkom nalazi u *KM*. Ako se nalazi, pročita se trenutna vrednost njegovog brojača *b*. Nakon pristupa podatku, brojači blokova *KM* se ažuriraju na sledeći način:

- brojač bloka u kome se nalazi podatak se resetuje na 0 (time se brojač bloka sa podatkom stavlja na čelo liste jer se podatku upravo pristupilo)
- svi brojači sa vrednošću manjom od *b* se inkrementiraju za 1 (pomeraju se za jedno mesto od čela liste, jer je na čelo liste stavljen brojač bloka sa podatkom)
- svi brojači sa vrednošću većom od *b* se ne menjaju (oni su se u listi nalazili iza brojača bloka sa podatkom, pa promena na čelu liste nema uticaja na ove brojače)

U slučaju da se traženi podatak ne nalazi u *KM*, potrebno je uraditi sledeće:

- blok *KM* sa najvećom vrednošću brojača zameniti blokom *OM* u kome se nalazi podatak
- brojač zamenjenog bloka postaviti na 0 (time se on postavlja na čelo liste)
- vrednosti svih ostalih brojača inkrementirati za 1 (jer je brojač sa kraja liste prebačen na čelo liste)

LRU tehnika, kod koje je keš kontroler realizovan na opisani način, biće ilustrovana na jednom primeru. Neka se *KM* sastoji od 10 blokova koji su popunjeni blokovima OM 10, 3, 15, 1, 6, 9, 12, 5, 32 i 4, kao što je prikazano na slici 12.10. Svakom bloku *KM* pridružen je brojač sa odgovarajućom vrednošću. Procesor zahteva podatke iz blokova *OM* 6 i 11, redom.

	KM	Vrednosti brojača		
		početne	Blok 6	Blok 11
0	Blok 10	3	4	5
1	Blok 3	4	5	6
2	Blok 15	1	2	3
3	Blok 1	0	1	2
4	Blok 6	5	**0**	1
5	Blok 9	10	10	11
6	Blok 12	6	6	7
7	Blok 5	2	3	4
8	Blok 32	7	7	8
9	Blok 4 → Blok 11	12	12	**0**

Slika 12.10 *LRU* tehnika - primer

Prvi podatak (iz bloka 6) se već nalazi u 4. bloku *KM*, čiji brojač ima vrednost 5. Nakon pristupa podatku, ovaj brojač se resetuje na 0, brojači prva četiri bloka *KM*, kao i brojač 7. bloka se inkrementiraju (jer su imali vrednost manju od 5), dok brojači ostalih blokova ostaju nepromenjeni (jer su imali vrednost veću od 5).

Drugi podatak (iz bloka 11) se trenutno ne nalazi u *KM*, pa je potrebno blok 11 iz *OM* preneti u *KM*. Blok *OM* se upisuje u blok *KM* čiji brojač ima najveću vrednost (12), a to je blok 9. Nakon upisa, brojač 9. bloka se resetuje na 0, a svi ostali brojači se inkrementiraju (jer su imali vrednost manju od 12). Kao što se vidi, u ovom slučaju je urađena zamena bloka po kriterijumu koji podržava *LRU* tehnika.

12.1.3 Tehnike ažuriranja

Tehnike ažuriranja se bave problemom koherencije, tj. usklađivanja sadržaja *KM* i *OM*. U *KM* se u svakom trenutku nalazi određen broj kopija blokova *OM*. Procesor može da pristupa podacima u *KM* i da menja njihove vrednosti. Svaka promena, učinjena samo u bloku *KM*, dovodi do neusaglašenosti između sadržaja tog bloka i njegovog originala koji se nalazi u *OM*. Za korektan rad, neophodno je usklađivanje sadržaja *OM* sa promenama u *KM*, što omogućavaju tehnike ažuriranja.

Postoje četiri tehnike ažuriranja:

- tehnika upisa ako blok postoji u *KM*
- tehnika upisa ako blok ne postoji u *KM*
- tehnika čitanja ako blok postoji u *KM*
- tehnika čitanja ako blok ne postoji u *KM*

Tehnika upisa ako blok postoji u *KM* (*Cache Write Policy Upon a Cache Hit*) se bavi problemom ažuriranja u slučaju upisa podatka u neki blok *KM*. Ova tehnika pruža dve mogućnosti ažuriranja. Prva mogućnost je trenutni upis (*write-trough*) kod koga se svaka operacija upisa realizuje tako što se istovremeno podatak upisuje i u *KM* i u odgovarajući blok *OM*. Druga mogućnost je odloženi upis (*write-back*) kod koga se podatak upisuje samo u blok *KM*, dok se upis u *OM* odlaže sve dok se ne pojavi potreba da se konkretan blok izbaci iz *KM*. U trenutku zamene bloka *KM*, ispituje se da li je taj blok menjan. Ukoliko jeste, blok *KM* se upisuje u blok *OM*, a ako nije, onda se samo blok *KM* zameni dolazećim blokom *OM*. Da bi pomenuto ispitivanje bilo moguće, neophodno je svakom bloku *KM* pridružiti jedan bit (*dirty bit*) koji ima vrednost 1 ako je postojao bar jedan upis u taj blok, inače ima vrednost 0.

Tehnika upisa ako blok ne postoji u *KM* (*Cache Write Policy Upon a Cache Miss*) se bavi problemom ažuriranja u slučaju upisa podatka u blok *OM* koji nije ranije prenet u *KM*. Ovde, takođe, postoje dve mogućnosti ažuriranja. Prva je dovlačenje bloka (*write-allocate*), što podrazumeva prenos i upis bloka *OM* u *KM*, nakon čega se primenjuje prethodno opisana tehnika. Druga mogućnost je nedovlačenje bloka (*write-no-allocate*) i tada se podatak upisuje samo u blok *OM*.

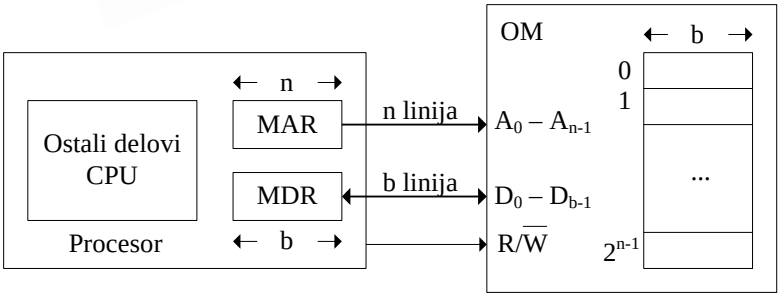
Tehnika čitanja ako blok postoji u *KM* (*Cache Read Policy Upon a Cache Hit*) je jednostavna jer se podatak direktno čita iz *KM*.

Tehnika čitanja ako blok ne postoji u *KM* (*Cache Read Policy Upon a Cache Miss*) se bavi problemom čitanja podatka iz nekog bloka *OM*. Tehnika pruža dve mogućnosti. Prva mogućnost je direktno prosleđivanje, pri čemu se blok *OM* prenosi u *KM*, a podatak se odmah po pristizanju u *KM* prosleđuje procesoru. Druga mogućnost je prosleđivanje sa zadrškom, pri čemu se najpre ceo blok *OM* upiše u *KM*, pa se tek onda pročita podatak iz njega i prosledi procesoru.

12.2 Operativna memorija

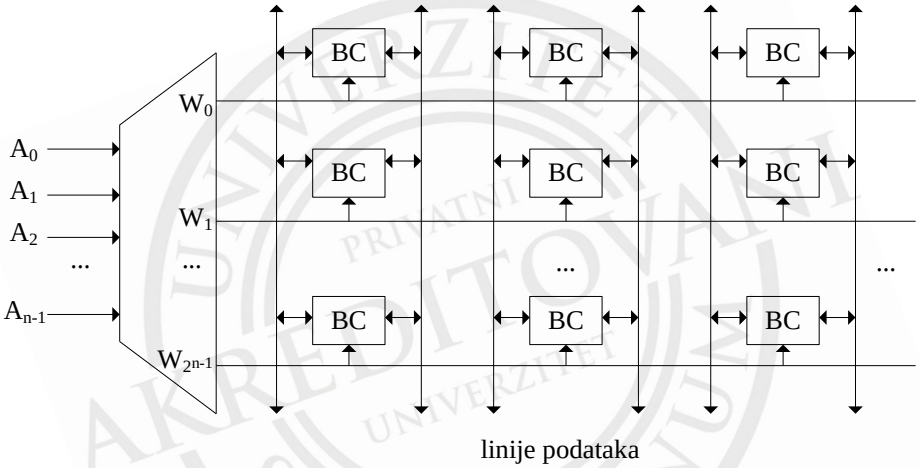
Operativna memorija (*Main Memory*) je radna memorija računara u kojoj se čuvaju *trenutno aktivni* programi i podaci. Po aktiviranju, aplikacija se upisuje u operativnu memoriju, nakon čega može da se koristi. Pri izlasku iz aplikacije, deo memorije u kome je bila smeštena aplikacija se oslobađa i stavlja na raspolaganje drugim programima.

Komunikacija između procesora i operativne memorije (*OM*) je prikazana na slici 12.11. Veza između ove dve komponente se obično ostvaruje preko dva registra, *MAR* (prihvata adrese memorijskih lokacija) i *MDR* (prihvata podatke). Veličina *MAR* (*n* bitova) je usklađena sa kapacitetom *OM* (2^n ćelija), dok veličina *MDR* (*b* bitova) odgovara veličini ćelije u memoriji (*b* bitova). *MAR* je povezan sa *OM* preko adresnih linija memorije, a *MDR* preko linija za podatke.



Slika 12.11 Povezivanje procesora i *OM*

Unutrašnja struktura *OM* se može vizualizovati u vidu mreže osnovnih ćelija (BC - *Basic Cells*) raspoređenih po vrstama i kolonama (slika 12.12). Svaka *osnovna ćelija* sadrži informaciju od 1 bita. Sadržaji svih ćelija u jednoj vrsti čine jednu memorijsku reč, tj. sadržaj jedne memorijske lokacije. Sastavni deo *OM* je i *adresni dekoder* čiji su ulazi direktno povezani na adresne linije memorije. Na osnovu prispele adrese $A_{n-1}...A_0$, adresni dekoder generiše signale na selekcionim linijama vrsta $W_2{}^{n-1}...W_0$ (horizontalne linije) koje su povezane sa osnovnim ćelijama. Takođe, na osnovne ćelije se dovode i linije podataka memorije (vertikalne linije).



Slika 12.12 Struktura *OM*

U datom trenutku, aktivira se samo jedan izlaz adresnog dekodera, tj. jedna selekciona linija, dok su sve ostale selekcione linije neaktivne. Aktivna selekciona linija se koristi za upis podatka koji se trenutno nalazi na linijama podataka u sve osnovne ćelije u vrsti, ili za čitanje podataka iz svih osnovnih ćelija u vrsti i njihovo postavljanje na linije podataka.

Svaka osnovna ćelija se realizuje skupom tranzistora povezanih na odgovarajući način, tako da može da bude samo u dva stabilna stanja (0 ili 1). Zbog načina realizacije, osnovne ćelije su obično priključene na dve linije podataka, pri čemu jedna služi za ulaz sadržaja u ćeliju, a druga za izlaz.

Operativna memorija se realizuje u integrisanoj tehnologiji u vidu čipova. Memorijski čipovi istog kapaciteta mogu da imaju različit broj pinova u zavisnosti od unutrašnje organizacije memorije. Neka je memorija sa *R* vrsta i *C* kolona (u strukturi) označena sa *R* x *C*. U tabeli 12.1 je dat primer određivanja broja pinova

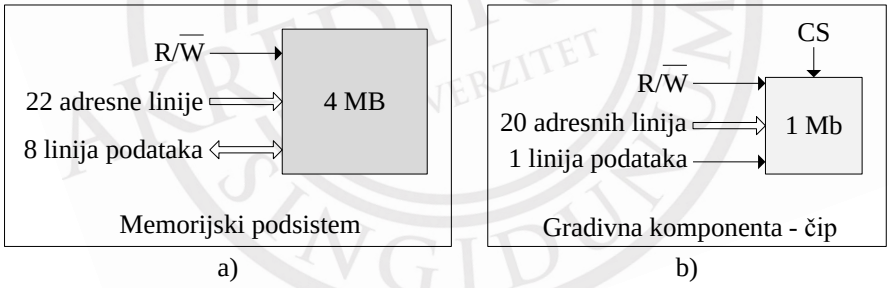
memorije kapaciteta 4Kb za četiri različita načina njene organizacije. Kao što se vidi, najviše pinova (24) zahteva organizacija 256 x 16 (256 16-bitnih ćelija).

Organizacija	Broj linija podataka (C)	Broj selekcionih linija (R)	Broj adresnih linija ($A=\log_2 R$)	Broj pinova ($A + C$)
4096 x 1	1	4096	12	13
1024 x 4	4	1024	10	14
512 x 8	8	512	9	17
256 x 16	16	256	8	24

Tabela 12.1 Određivanje broja pinova po čipu

Pri projektovanju memorijskog sistema računara, *OM* se realizuje u vidu više čipova određenog kapaciteta. Kapacitet čipa može da bude ograničavajući faktor koji utiče na realizaciju, što će biti objašnjeno na jednom primeru.

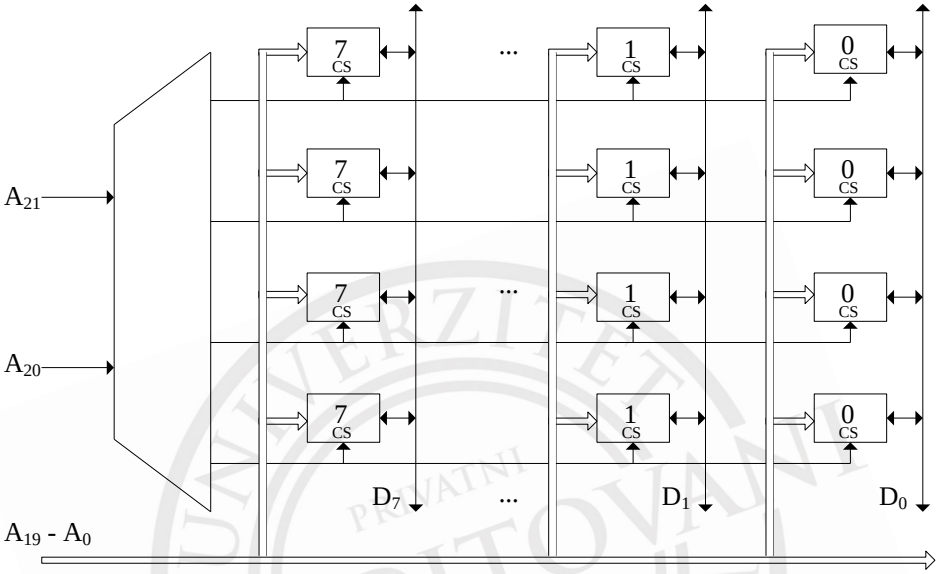
Neka je potrebno projektovati memorijski podsistem kapaciteta 4MB korišćenjem čipova kapaciteta 1Mb. Zahtevi memorijskog podsistema su dati na slici 12.13a, a zahtevi čipa, kao gradivnog elementa, na slici 12.13b.



Slika 12.13 Memorijski zahtevi - primer

Memorijski podsistem od 4MB se može organizovati u vidu $4 \cdot 2^{10} \cdot 2^{10} = 2^{22}$ ćelija kapaciteta 1B, što zahteva 22 adresne linije i 8 linija podataka. Čip kapaciteta 1Mb je organizovan u vidu 2^{20} ćelija kapaciteta 1b, što zahteva 20 adresnih linija i 1 liniju podataka. Dakle, broj adresnih linija i linija podataka nije odgovarajući u ova dva slučaja, pa je potrebno smisliti način povezivanja čipova u okviru podsistema. S obzirom da je kapacitet podsistema 32 puta veći od kapaciteta jednog čipa ($8 \cdot 2^{22}/2^{20}$), podsistem treba da sadrži 32 čipa. Čipovi se mogu organizovati u 4 vrste po 8 čipova. Od 22 adresne linije memorijskog podsistema, 20 linija se može iskoristiti za adresiranje unutar čipova, a preostale dve linije za selekciju čipova u vrsti. Selekcija čipova u vrsti se vrši pomoću dekodera 2/4, tako što se na ulaze dekodera dovedu preostale dve adresne linije, a signali sa izlaza se sprovedu do *CS* (*Chip Select*) ulaza čipova. Kada je *CS* ulaz aktivan, moguć je

pristup čipu, u suprotnom, nije. Opisana realizacija memorijskog podsistema prikazana je na slici 12.14.



Slika 12.14 Realizacija memorijskog podsistema - primer

Kao što se vidi, 8 čipova u jednoj vrsti realizuju 2^{20} memorijskih reči od 1B, a svi čipovi zajedno $4 \cdot 2^{20} \cdot 1B = 4MB$. Podatak, koga čine signali D_7-D_0, adresira se unutar čipa putem linija A_{19}-A_0. Nakon postavljanja adrese i podatka na pomenute linije, linijama A_{20} i A_{21} se selektuje samo jedna vrsta i podatak se (na nivou bita) istovremeno upisuje u sve čipove u toj vrsti na zadatu adresu. Sličan postupak se sprovodi i pri operaciji čitanja.

12.3 Virtuelna memorija

Kada procesor zahteva neku programsku instrukciju ili neki podatak, oni bi trebalo da se nalaze u operativnoj memoriji. Međutim, s obzirom da *OM* ima organičen kapacitet, može se desiti da u njoj nema dovoljno slobodnog prostora za smeštaj željenog programa i podataka. Povećanje kapaciteta *OM* bi značajno povećalo njenu cenu. U praksi, nije neophodno da se ceo program koji se izvršava ili svi potrebni podaci nalaze u *OM*. Delovi programa koji se trenutno ne izvršavaju mogu da budu na hard disku, dok su delovi programa koji se trenutno izvršavaju u *OM*. Ovakav način rada zahteva uvođenje mehanizama za prenos delova programa ili podataka iz *OM* na hard disk i obrnuto. Ovi mehanizmi čine koncept virtuelne

memorije (*VM – Virtual Memory*). Tehnika *VM* služi za *„proširivanje“ fizičke veličine OM* na potrebnu veličinu. Prenos delova programa između *OM* i hard diska je transparentan za korisnika i obavlja ga operativni sistem.

Princip rada *VM* je sličan principu rada keš memorije. Razlike su uglavnom implementacione. Međutim, cena „promašaja“ u virtuelnom pristupu (to je slučaj kada podatak nije u *OM*) je 100−200 puta veća od cene „promašaja“ u keš pristupu (slučaj kada podatak nije u *KM*). To je razumljivo s obzirom da se radi o različitim nivoima u memorijskoj hijerarhiji (*OM*−hard disk i *KM*−*OM*). Ovde se pojam cene pre svega odnosi na vreme potrebno za pristup podatku.

Kod savremenih računara, prilikom pisanja programa, programeri imaju na raspolaganju celokupnu memoriju, tj. *OM* i hard disk. Adrese koje se generišu u programima nazivaju se *virtuelnim adresama*, a njihov opseg, *virtuelni adresni prostor – VAP*. Ove adrese ne odgovaraju fizičkim adresama u *OM* (na primer, 32-bitni procesor može da adresira 4GB memorije, a *OM* u tom računaru može da ima kapacitet od samo 1GB). Opseg adresa u *OM* predstavlja *realni adresni prostor – RAP*, a adresa lokacije u *OM* se naziva *realnom adresom*. Za prevođenje virtuelne adrese u odgovarajuću realnu (fizičku) adresu u *OM*, zadužena je jedinica za upravljanje memorijom – *MMU*.

Da bi virtuleni pristup mogao da funkcioniše, neophodno je voditi evidenciju o tome koji delovi programa se trenutno nalaze u *OM* i u kojem njenom delu. Za vođenje evidencije se koriste tabele preslikavanja koje se nalaze u *OM*.

Prema načinu podele virtuelnog adresnog prostora, postoje tri tipa *VM*:

- *VM* sa straničnom organizacijom
- *VM* sa segmentnom organizacijom (*VAP* se deli na delove promenljive veličine – segmente, koji se smeštaju u *OM* u prostore odgovarajuće veličine; kada se *OM* napuni, segment iz *OM* se prenosi na hard disk, a novi segment sa hard diska se upisuje u *OM*)
- *VM* sa segmentno-straničnom organizacijom (*VAP* se deli na delove promenljive veličine – segmente, a zatim se segmenti dele na delove fiksne veličine – stranice; *RAP* se deli u blokove jednake veličini stranice; stranice segmenata se smeštaju u blokove *OM*; kada se *OM* napuni, neka od stranica se prenosi na hard disk, a nova stranica sa hard diska se unosi u *OM*)

12.3.1 Stranična organizacija

Stranična organizacija *VM* podrazumeva podelu *VAP* na delove fiksne veličine koji se nazivaju *stranicama* i podelu *RAP* na delove fiksne veličine koji se nazivaju *blokovima*. Veličina stranice je jednaka veličini bloka (obično 2 do 16 KB). Pri

izboru veličine stranice (a samim tim i bloka), treba voditi računa da ona ne bude ni previše mala (često bi se pristupalo hard disku, a to dugo traje), ni previše velika (mnogi preneti delovi stranice se ne bi koristili). Stranice se smeštaju u blokove *OM*. Kada se *OM* napuni, neka od stranica se prenosi na hard disk, kako bi se na njeno mesto upisala nova stranica sa hard diska (koja sadrži traženi podatak).

Kod stranične organizacije *VM*, virtuelna adresa sadrži dva polja:

- *page* – broj stranice; dužina ovog polja je *p* bitova
- *word* – adresa reči unutar stranice; dužina ovog polja je *w* bitova

Realna adresa, takođe, ima dva polja:

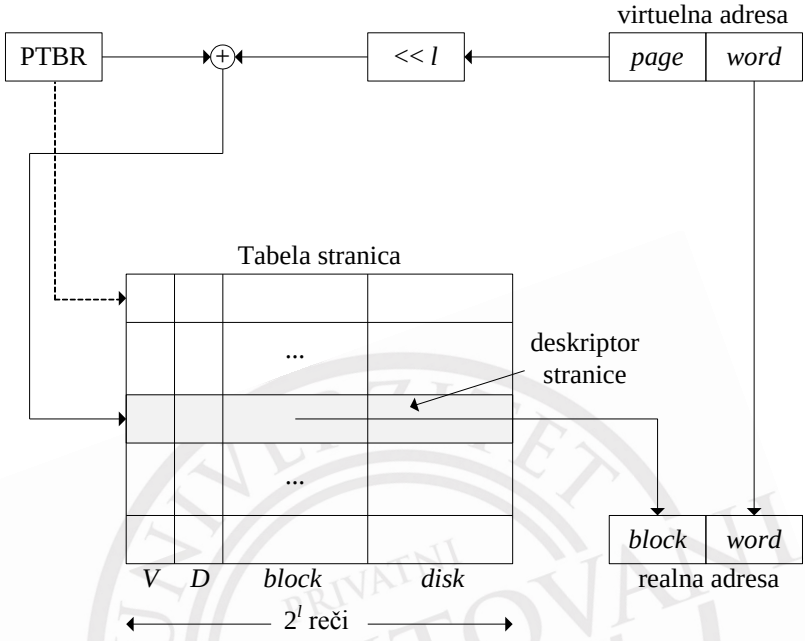
- *block* – broj bloka; dužina ovog polja je *b* bitova
- *word* – adresa reči unutar bloka; dužina ovog polja je *w* bitova

Dakle, veličina *VAP* je 2^{p+w}, a veličina *RAP* 2^{b+w} adresa. Evidencija o trenutnom sadržaju *OM* se vodi pomoću *tabele stranica* (*PT – Page Table*). *PT* se nalazi u *OM*, a njena početna adresa u posebnom procesorskom registru *PTBR* (*Page Table Base Register*). Pošto u *VAP* ima 2^p stranica, *PT* ima 2^p ulaza (vrsta). Svakom ulazu odgovara po jedan *deskriptor stranice*, koji sadrži sve potrebne informacije o toj stranici. Na osnovu sadržaja *PTBR* i broja stranice, *MMU* može da pristupi deskriptoru te stranice.

Deskriptor stranice (*DS*) se sastoji od četiri polja:

- *V* (1 bit) – statusni bit koji pokazuje da li je stranica u *OM* memoriji ili nije; ovaj bit postavlja operativni sistem (*OS*) prilikom dovlačenja stranice sa hard diska u *OM*; bit omogućava *OS* da stranicu proglasi nevalidnom bez njenog uklanjanja iz *OM*
- *D* (1 bit) – statusni bit koji pokazuje da li je stranica modifikovana ili nije; ovaj bit postavlja *MMU* prilikom operacije upisa, a koristi ga *OS* da odluči da li stranicu treba upisati na hard disk pre izbacivanja iz *OM*
- *block* (*b* bitova) – broj bloka *OM* u kome se nalazi data stranica (važi ako je *V* = 1); polje postavlja *OS* prilikom dovlačenja stranice sa hard diska, a koristi ga *MMU* za formiranje realne adrese
- *disk* (*d* bitova) – adresa stranice na hard disku

Preslikavanje virtuelne u realnu adresu se izvodi hardverski kao što je prikazano na slici 12.15.



Slika 12.15 Preslikavanje virtuelne u realnu adresu

Postupak preslikavanja se odvija u sledećim koracima:

1. polje *page* u virtuelnoj adresi predstavlja broj stranice, a istovremeno i broj ulaza u *PT* u kome se nalazi deskriptor te stranice; ako svaki *DS* zauzima po 2^l memorijskih reči, sadržaj polja *page* treba pomeriti za *l* mesta ulevo (što odgovara množenju sa 2^l) kako bi se dobio ofset (pomeraj) početka deskriptora u odnosu na početak *PT*
2. u registru *PTBR* se nalazi adresa početka *PT*; kada se sadržaj ovog registra sabere sa ofsetom dobijenim u koraku 1, dobija se adresa početne lokacije deskriptora stranice
3. počev od adrese dobijene u koraku 2, iz *DS* se čita reč koja odgovara polju *V* i na osnovu nje proverava da li se stranica trenutno nalazi u *OM*
4. ako je stranica u *OM*, iz *DS* se čita sadržaj polja *block* koji predstavlja broj bloka *OM* u kome se stranica nalazi; konkatenacijom (spajanjem) sadržaja polja *block* i polja *word* iz virtuelne adrese, formira se realna adresa
5. ako stranica nije u *OM*, generiše se prekid u cilju dovlačenja stranice sa hard diska; ovaj deo posla odrađuje *OS* softverskim putem

Vežbanja

P1. Šta čini memorijski sistem računara? Nacrtati hijerarhijsku organizaciju memorijskog sistema računara i objasniti je.

P2. Opisati princip rada keš memorije. Dati sliku.

P3. Šta je mapiranje adresa? Ko obavlja funkciju mapiranja?

P4. Koje probleme je trebalo rešiti da bi proces keširanja podataka bio moguć?

P5. Čemu služe tehnike preslikavanja? Koje vrste tehnika preslikavanja postoje?

P6. U čemu je suština tehnike direktnog preslikavanja? Prikazati strukturu keš memorije i objasniti je. Na koji način *MMU* interpretira adresu podatka dobijenu od procesora kod direktnog preslikavanja? Po kom protokolu radi *MMU* kada dobije adresu podatka od precesora u slučaju direktnog preslikavanja? Navesti prednosti i nedostatke direktnog preslikavanja.

P7. U čemu je suština tehnike asocijativnog preslikavanja? Kako *MMU* interpretira adresu dobijenu od procesora u slučaju asocijativnog preslikavanja i po kom protokolu tada radi?

P8. Kako se vrši pretraživanje memorije tagova kod asocijativnog preslikavanja? Koje su posledice takvog pretraživanja? U čemu je prednost asocijativnog u odnosu na direktno preslikavanje?

P9. U čemu je suština tehnike set-asocijativnog preslikavanja? Objasniti interpretaciju adrese podatka koju *MMU* dobija od procesora u slučaju set-asocijativnog preslikavanja. Po kom protokolu radi *MMU* kada dobije adresu podatka od procesora kod set-asocijativnog preslikavanja?

P10. Koji problem rešavaju tehnike zamene? Kod kojih tehnika preslikavanja se koriste tehnike zamene?

P11. Koje tehnike zamene bloka keš memorije postoje?

P12. Objasniti tehniku slučajnog izbora bloka keš memorije za zamenu.

P13. Koji blok keš memorije treba zameniti ako se koristi *FIFO* tehnika zamene?

P14. Objasniti *LRU* tehniku zamene.

P15. Čemu služe tehnike ažuriranja? Koje tehnike ažuriranja postoje?

P16. Objasniti mogućnosti ažuriranja *OM* u slučaju upisa podatka u blok koji se nalazi/ne nalazi u keš memoriji.

P17. Objasniti mogućnosti ažuriranja *OM* u slučaju čitanja podatka iz bloka koji se nalazi/ne nalazi u keš memoriji.

P18. Koja je uloga operativne memorije u računarskom okruženju? Kako se povezuju procesor i operativna memorija? Dati sliku.

P19. Prikazati opšti oblik unutrašnje strukture operativne memorije. Šta predstavljaju osnovne ćelije u strukturi?

P20. Objasniti koncept virtuelne memorije. Šta su virtuelne, a šta realne adrese? U čemu je razlika između virtuelnog i realnog adresnog prostora? Koji tipovi virtuelne memorije postoje?

P21. Navesti glavne osobine virtuelne memorije sa straničnom organizacijom. Dati strukturu virtuelne i realne adrese.

P22. Šta je tabela stranica? Šta predstavlja i kako izgleda deskriptor stranica? Objasniti značenja svih polja u deskriptoru.

P23. Predstaviti postupak preslikavanja virtuelne u realnu adresu kod virtulene memorije sa straničnom organizacijom.

Z1. Kolika je dužina (u bitima) adrese podatka koji se nalazi u memoriji kapaciteta 4G memorijskih reči?

Z2. Ako se blok sastoji od 8 16-bitnih memorijskih reči, koliki je kapacitet bloka i koliko blokova ima u 1GB memorijskog prostora?

Z3. Kapacitet operativne memorije (*OM*) je 128MB, a kapacitet keš memorije (KM) 256KB. Ako je veličina bloka 4 8-bitne memorijske reči, odrediti:

a) broj blokova u *OM* i *KM*

b) koliko blokova *OM* se preslikava u isti blok *KM*, ako se koristi metod direktnog preslikavanja

Z4. Na raspolaganju su operativna memorija koja se sastoji od 2048 blokova i keš memorija sa 128 blokova. Dužina svakog bloka je 8 memorijskih reči. Ukoliko *KM* implementira tehniku direktnog preslikavanja, odrediti:

a) broj bitova u poljima *MMU* interpretacije adrese koju generiše procesor

b) blok i poziciju podatka u *OM* i *KM* ako procesor generiše adresu 01110001001111

c) gde se nalazi podatak ako mu procesor pristupa preko adrese 01110001000111

Z5. Na raspolaganju je keš memorija sa 256 blokova od po 4 memorijske reči (označene sa 0, 1, 2 i 3). Prvih pet blokova *KM* (uključujući i odgovarajuće tagove) je popunjeno sadržajem kao na slici. *KM* koristi direktno preslikavanje blokova.

TM		DM 0. reč	1. reč	2. reč	3. reč
100101	0	A1	B5	C4	41
010111	1	74	11	45	58
100111	2	90	**E4**	2D	22
101100	3	52	35	89	C7
100011	4	4F	18	F8	B2
...	...	...	...	...	...

Odrediti:

a) adresu podatka $E4_{(16)}$

b) da li se podatak sa adrese $B00F_{(16)}$ u *OM* nalazi u *KM*; ako se nalazi, pročitati ga

c) da li se podatak sa adrese $1810_{(16)}$ u *OM* nalazi u *KM*; ako se nalazi, pročitati ga

Z6. Data je keš memorija koja implementira asocijativno preslikavanje. Ona je podeljena na blokove od po 8 memorijskih reči. Njen trenutni sadržaj je prikazan na slici.

TM		DM							
		0	1	2	3	4	5	6	7
1001011000000	0	23	4F	77	36	B5	27	D6	0F
0011100010111	1	22	2B	56	2B	45	28	33	BC
1001110000011	2	7F	00	A6	AB	C0	3D	FD	5E
0011001010101	3	9F	C8	2A	21	4C	78	**CF**	2A
1000111101001	4	48	18	90	2D	10	30	F3	3C
...	...	...					...	...	...

Odrediti:

a) adresu u *OM* na kojoj se nalazi podatak $CF_{(16)}$
b) da li se podatak sa adrese $8F4A_{(16)}$ u *OM* nalazi u *KM*; ako se nalazi, pročitati ga

Z7. Keš memorija sa set-asocijativnim preslikavanjem kapaciteta 64KB podeljena je na setove od po 4 bloka. Svaki blok sadrži 16 8-bitnih reči. Na blokove iste veličine podeljena je i operativna memorija kapaciteta 1GB. U koji set *KM* treba smestiti blok *OM* koji sadrži podatak na adresi $29ABCDE8_{(16)}$? Identifikovati tag i prvu i poslednju adresu u bloku.

Z8. Projektovati memorijski podsistem kapaciteta 64MB korišćenjem *RAM* čipova kapaciteta 16Mb. Svaki čip ima liniju za selekciju čipa (*CS*) i linije za upis/čitanje. Izračunati broj potrebnih čipova i grafički dati kompletnu realizaciju memorijskog podsistema.

Z9. Računar koristi *RAM* čipove kapaciteta 128 x 8.

a) Koliko čipova je potrebno da bi se napravila memorija kapaciteta 2048B?
b) Koliko adresnih linija je potrebno za pristup ovoj memoriji? Koliko adresnih linija su zajedničke za sve čipove?
c) Koliko linija treba dekodirati za *CS* ulaze čipova? Specificirati tip dekodera koji se mora primeniti.

Z10. Na raspolaganju su memorijski sistemi organizovani na sledeće načine:

64K x 8, 64K x 32, 32K x 16, 32K x 32, 128K x 32, 128K x 64.

Za svaki memorijski sistem odrediti broj bitova u *MAR* i *MDR* registrima, ako se memorija adresira na nivou:

a) jednog bajta
b) dva bajta

Z11. Na raspolaganju su *RAM* čipovi 2K x 8. Koliko je ovakvih čipova potrebno da bi se napravila memorija kapaciteta 1MB? Koliko bi čipova 1K x 16 bilo potrebno za realizaciju iste memorije?

13 Organizacija ulaza/izlaza

Zbog raznolikosti zahteva u pogledu unosa podataka u računar i prikaza rezultata obrade, postoji veliki broj *U/I* uređaja koji se međusobno razlikuju po funkcionalnosti, načinu transfera podataka, brzini rada i sl. Međutim, bez obzira na razlike, rad sa svim *U/I* uređajima je organizovan na isti način. To je postignuto zahvaljujući uvođenju univerzalnog modela *U/I* uređaja (sa stanovišta računara).

Svi *U/I* uređaji se sastoje od *periferije* i *kontrolera periferije*. Periferija realizuje osnovnu funkcionalnost uređaja. Kontroler služi za prihvat podataka, upravljanje periferijom i komunikaciju sa ostatkom računara. Dakle, računar pristupa *U/I* uređaju preko kontrolera periferije, dok mu je sama periferija transparentna.

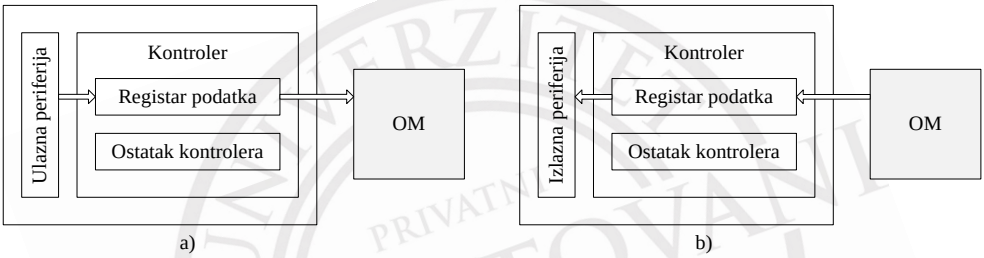
Kontroler periferije sadrži *upravljačku jedinicu* (upravljačka logika) i *operacionu jedinicu* (određen broj registara). Uloga upravljačke logike je da na osnovu sadržaja registara operacione jedinice organizuje preuzimanje podataka iz ulazne periferije ili upis podataka u izlaznu periferiju, kao i njihovo prosleđivanje do/od procesora/memorije. Registri kontrolera omogućavaju da se kompletna organizacija ulaza/izlaza na programskom nivou svede na upis i čitanje sadržaja ovih registara.

Kontroler periferije sadrži sledeće registre:

- *upravljački registar* (*CR – Control Register*), koji omogućava startovanje i zaustavljanje kontrolera periferije programskim putem; pod startovanjem se podrazumeva otpočinjanje prenosa podataka do/iz periferije
- *statusni registar* (*SR – Status Register*), koji sadrži indikatore o spremnosti podatka da se prenese na odredište (u periferiju, ili u memoriju/procesor)

- *registar podatka* (*DR – Data Register*), koji sadrži podatak koji treba preneti u periferiju ili u memoriju/procesor
- *registar broja ulaza* (*ER – Entry Register*), koji sadrži kod prekida koji se šalje procesoru

Važan pojam u funkcionisanju ulaza/izlaza predstavlja *spremnost podatka*. Podatak se smatra spremnim za upis u memoriju kada je prenet iz ulazne periferije u registar podatka kontrolera (slika 13.1a). Slično, podatak je spreman za upis u izlaznu periferiju kada je prenet iz memorije u registar podatka kontrolera (slika 13.1b).



Slika 13.1 Spremnost podatka

Prenos podatka između registra podatka kontrolera i memorije se može realizovati na dva načina: *programski* (čitanjem ili upisom u registar podatka kontrolera) ili *radom samog kontrolera*.

Postoje dve mogućnosti prenosa podatka programskim putem. Prva mogućnost se zasniva na *proveri indikatora statusnog registra* u cilju utvrđivanja da li je podatak spreman za prenos (čitanje *ready* flega). Ako jeste, podatak se prosleđuje do odredišta. Druga mogućnost je da upravljačka logika *generiše signal prekida* svaki put kada podatak postane spreman za prenos. Ukoliko procesor na zahtev za prekidom odgovori slanjem signala potvrde, kontroler periferije šalje procesoru sadržaj registra broja ulaza (*ER*). Na osnovu broja ulaza, procesor pronalazi adresu prekidne rutine i izvršava je. Tokom izvršavanja prekidne rutine, obavlja se prenos podatka na odredište.

Prenos podatka može da obavi i sam kontroler tako što njegova upravljačka logika realizuje ciklus upisa podatka iz registra podatka u memoriju, ili čitanja podatka iz memorije i njegovog prihvata u registar podatka. U ovom slučaju, *U/I* uređaj podržava *DMA* prenos pošto preuzima na sebe organizaciju i realizaciju transfera podataka između *U/I* uređaja i memorije/procesora. Da bi ostvario *DMA* prenos, kontroler mora da poseduje dodatne hardverske komponente (registre za adresu i broj podataka, kao i odgovarajuću kombinacionu mrežu za prepoznavanje ciklusa čitanja i upisa). Ukoliko dođe do pojave neke greške pri prenosu,

upravljačka logika postavlja indikatore statusnog registra. U određenim trenucima, programski se čita sadržaj statusnog registra i utvrđuje da li je rad sa periferijom regularan ili nije. Po završetku prenosa ili u slučaju otkrivanja neregularnosti, u upravljački registar se upisuje sadržaj na osnovu koga upravljačka logika zaustavlja prenos.

Vežbanja

P1. Detaljno objasniti strukturu *U/I* uređaja. Koje programski dostupne registre sadrži kontroler periferije i koja je njihova uloga?

P2. Kada je podatak spreman za upis u memoriju/izlaznu periferiju pri realizaciji ulazno-izlaznih aktivnosti?

P3. Na koje načine se može realizovati prenos podatka između registra podatka kontrolera i memorijske lokacije?

P4. Ko može da realizuje ulazno-izlazne aktivnosti?

P5. Objasniti značenja sledećih bitova u registrima operacione jedinice kontrolera periferije: *start*, *u/i*, *ready* i *enable*.

P6. U čemu je prednost korišćenja kontrolera sa *DMA* u odnosu na kontroler bez *DMA* u *U/I* uređaju?

P7. Šta se menja u strukturi *U/I* uređaja ako mu se doda mogućnost *DMA* prenosa?

14 Magistrala

Osnovna uloga magistrale je da omogući prenos sadržaja između komponenata računara. Sadržaj se obično prenosi između procesorskih registara, memorijskih lokacija i registara *U/I* uređaja. Osim ove, uloga magistrale je i da obezbedi takt sistemu, prenosi upravljačke signale, vrši arbitraciju pristupa raznih jedinica na magistralu, omogući konfigurisanje naprednijih *U/I* mehanizama prenosa, itd.

Uobičajeni scenario koji ilustruje princip rada na magistrali sadrži sledeće korake:

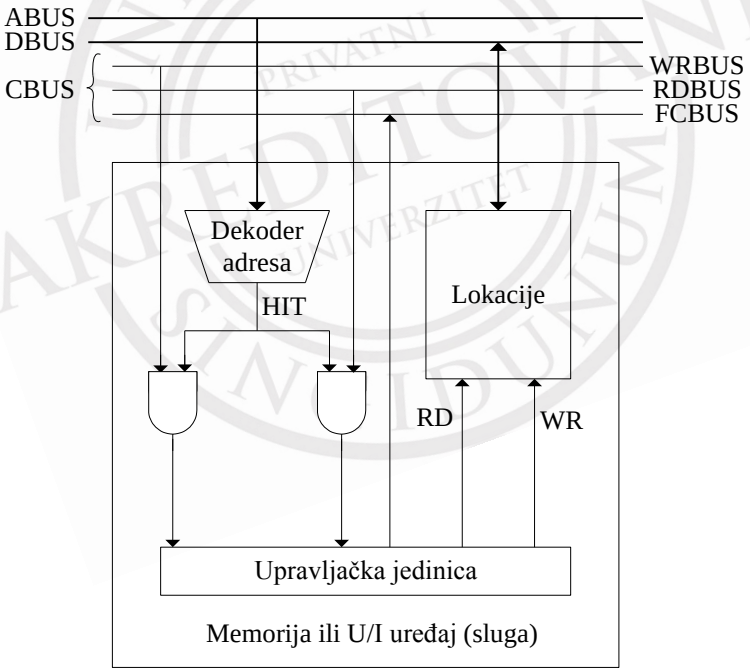
- gazda šalje adresu memorijske lokacije ili registra *U/I* uređaja na *ABUS*
- u slučaju upisa, gazda šalje podatak na *DBUS*
- sve sluge koje su povezane na *ABUS* dobijaju poslatu adresu
- pomoću svojih dekodera adresa, sve sluge proveravaju da li adresa odgovara nekoj od njihovih adresa
- samo jedan sluga prepoznaje da se poslata adresa odnosi na njega
- sa malim zakašnjenjem (dovoljnim da sluge provere adresu), gazda šalje na *CBUS* signal upisa/čitanja svim slugama
- upis/čitanje izvršava samo sluga koji je adresiran poslatom adresom
- u slučaju čitanja, pročitani podatak se šalje na *DBUS* i gazda ga prihvata

14.1 Adresni prostori

Memorijski adresni prostor predstavlja opseg adresa koje se mogu koristiti za adresiranje memorijskih lokacija. *U/I adresni prostor* čini opseg adresa za adresiranje registara unutar *U/I* uređaja.

Ukoliko se registrima *U/I* uređaja i memorijskim lokacijama pristupa pomoću istih programskih instrukcija, kaže se da je *U/I adresni prostor memorijski preslikan*. U slučaju da postoje posebne instrukcije za pristup memorijskim lokacijama, a posebne za pristup registrima *U/I* uređaja, kaže se da su *U/I i memorijski adresni prostori razdvojeni*.

Na slici 14.1 prikazan je *U/I* adresni prostor koji je memorijski preslikan. U ovom slučaju, sve sluge se tretiraju na isti način, bez obzira na to da li su memorijske jedinice ili *U/I* uređaji.

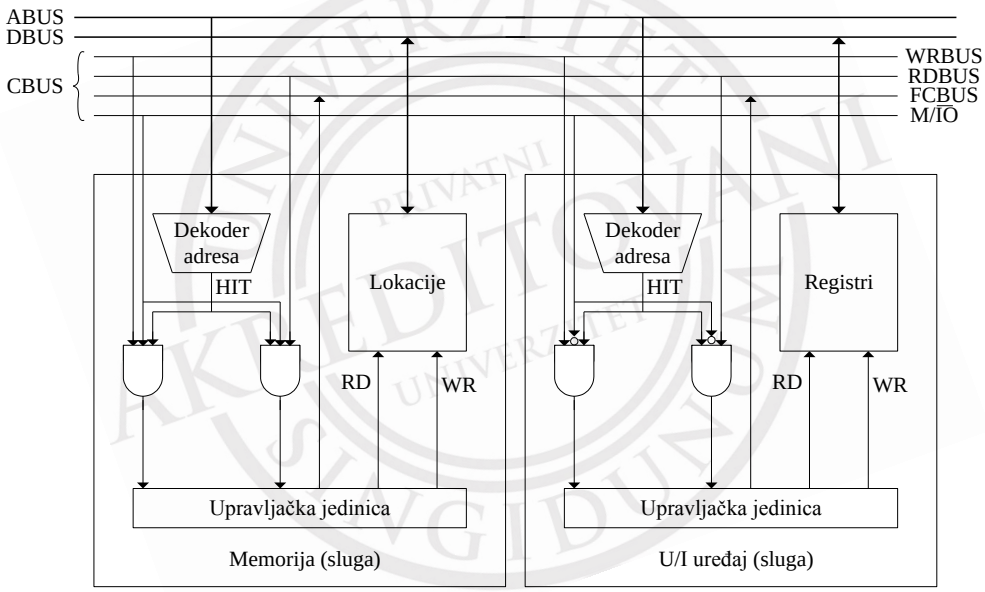


Slika 14.1 Memorijski preslikan *U/I* adresni prostor

Kao što se vidi, kontrolna magistrala (*CBUS*) je razložena na tri linije: *RDBUS*, *WRBUS* i *FCBUS*. *RDBUS*/*WRBUS* su kontrolne linije po kojima gazda šalje signale upisa/čitanja koji se dovode na ulaze logičkih množača u slugi. Po *ABUS* stiže adresa koja se dovodi na ulaze dekodera. Ako dekoder prepozna da je

posmatrani sluga adresiran, na svom izlazu će generisati signal *HIT* = 1 i jedno od logičkih kola će generisati signal upravljačkoj jedinici. Upravljačka jedinica prepoznaje o kom kolu se radi i generiše odgovarajući signal za upis/čitanje memorijske lokacije. U slučaju upisa, podatak se uzima sa *DBUS* i upisuje u ćeliju, dok se u slučaju čitanja, sadržaj adresirane lokacije prosleđuje na *DBUS*. Nakon toga, upravljačka jedinica generiše signal završetka upisa/čitanja i šalje ga na *FCBUS*. Signal sa *FCBUS* primaju svi u sistemu, ali samo gazda reaguje na njega tako što sadržaj sa *DBUS* upisuje u svoj *MDR* i završava čitanje (u slučaju čitanja) ili završava upis (u slučaju upisa).

Razdvojeni *U*/*I* i memorijski adresni prostori su prikazani na slici 14.2.



Slika 14.2 Razdvojeni adresni prostori

Pošto se u ovom slučaju memorija i *U*/*I* uređaj mogu adresirati istim adresama, neophodno je odrediti kome se gazda obraća. Zato je uvedena nova linija, $M/\overline{IO}$, u *CBUS*. Aktivna vrednost signala na $M/\overline{IO}$, znači da se gazda obraća memoriji, a neaktivna, da se obraća *U*/*I* uređaju. U realizaciji, neophodno je logičkim kolima dodati još po jedan ulaz na koji se dovodi $M/\overline{IO}$ signal. Kod *U*/*I* uređaja, ovaj ulaz mora biti invertovan, da bi izlaz kola bio aktivan kada je signal $M/\overline{IO}$ neaktivan. Kada upravljačka jedinica dobije signal sa izlaza nekog logičkog množača, prepoznaje da li se radi o upisu ili čitanju i generiše odgovarajući signal *RD*/*WR*. Ovaj signal se dovodi do memorijskih lokacija (ako je

adresirana memorija) ili registara (ako je adresiran *U/I* uređaj) i obavlja se odgovarajuća operacija. Po završetku operacije, upravljačka jedinica generiše signal završetka upisa/čitanja i šalje ga na *FCBUS*.

14.2 Arbitracija

Arbitracija podrazumeva donošenje odluke o tome koja komponenta u datom trenutku može da realizuje ciklus na magistrali. Mehanizam arbitracije je bitan u računarskim sistemima u kojima postoji više jedinica koje mogu da imaju ulogu gazde. U savremenim računarskim sistemima, obično je više procesora i *U/I* uređaja sa *DMA* prenosom priključeno na magistralu, pa je neophodno u svakom trenutku znati koja jedinica raspolaže magistralom.

Procesom arbitracije upravlja *kontroler magistrale* (*bus controller*) ili *arbitrator*. Značajnu ulogu u ovom procesu imaju tri kontrolna signala:

- *BB – Bus Busy*, koji *indicira zauzeće* magistrale (ako *BB* ima vrednost 1, znači da je magistrala zauzeta, tj. ranije je već dodeljena nekoj jedinici koja ima ulogu gazde)
- *BR – Bus Request*, koji predstavlja *zahtev za magistralom* upućen kontroleru magistrale od strane neke jedinice
- *BG – Bus Grant*, koji predstavlja *signal dozvole* za korišćenje magistrale poslat od strane kontrolera magistrale

Kada neka jedinica treba da realizuje ciklus na magistrali, ona šalje kontroleru zahtev za magistralom. Kontroler joj dodeljuje magistralu, ukoliko ona nije trenutno zauzeta.

Postoji više tehnika arbitracije:

- tehnika ulančavanja (*Daisy chain*)
- tehnika prozivanja (*Polling*)
- tehnika nezavisni zahtev/dozvola (*Independent request/grant*)

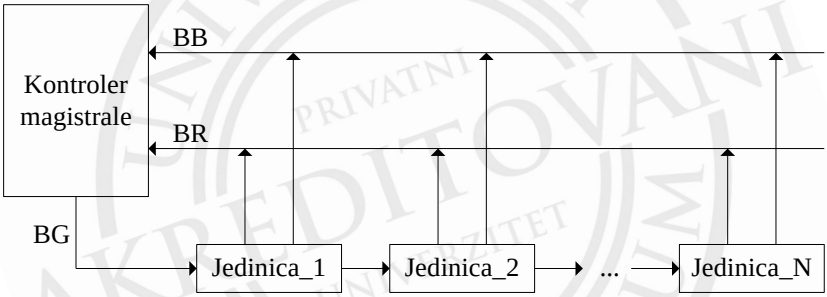
14.2.1 Tehnika ulančavanja

Tehnika ulančavanja podrazumeva da su *jedinice ulančane po prioritetu* tako da se jedinica sa najvišim prioritetom nalazi na čelu lanca. Na slici 14.3 je prikazan način povezivanja jedinica, pri čemu *Jedinica_*1 ima najviši, a *Jedinica_N* najniži prioritet.

Proces arbitracije otpočinje tako što neke od jedinica šalju kontroleru zahteve za magistralom po *BR* liniji. Ako je magistrala trenutno slobodna (*BB* signal je neaktivan), kontroler magistrale šalje signal po *BG* liniji koji najpre dolazi do jedinice na početku lanca. Kada signal *BG* stigne do neke jedinice, ona može da reaguje na dva načina:

- ukoliko nije poslala zahtev za magistralom, da propusti signal do sledeće jedinice u lancu
- ukoliko jeste poslala zahtev za magistralom, da zaustavi dalje prosleđivanje signala *BG* i postavi aktivnu vrednost na *BB* liniju, čime postaje gazda na magistrali

Ovakvim postupkom je postignuto da magistralu uvek zauzme jedinica sa najvišim prioritetom.



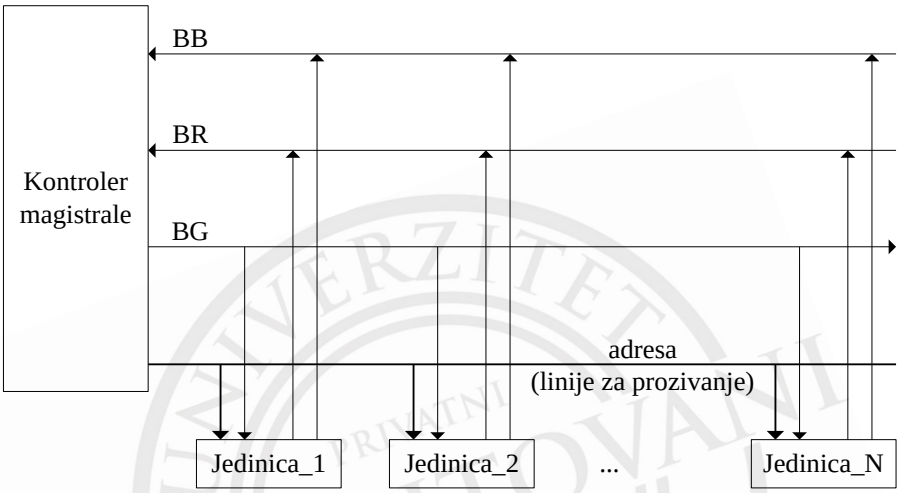
Slika 14.3 Tehnika ulančavnja

14.2.2 Tehnika prozivanja

Kod tehnike prozivanja, jedinice nisu međusobno povezane, već je *svaka jedinica pojedinačno priključena na kontrolne linije BB*, *BR* i *BG*. Osim toga, uvedene su i posebne *linije za prozivanje* po kojima se šalju adrese jedinica (slika 14.4). Broj linija za prozivanje zavisi od broja jedinica. Ako je broj jedinica *N*, broj adresnih linija je $\log_2 N$.

Kada neka od jedinica želi da ostvari ciklus na magistrali, ona po *BR* liniji šalje zahtev kontroleru magistrale. U istom trenutku, osim ove jedinice, zahteve mogu da upute i druge jedinice u sistemu. Kada dobije aktivan signal po *BR* liniji, kontroler magistrale najpre proverava da li je magistrala slobodna ispitivanjm signala na *BB* liniji. Ako jeste, kontroler proziva jedinice tako što šalje njihove adrese (jednu po jednu) po linijama za prozivanje. Redosled adresa je u skladu sa unapred utvrđenim prioritetima jedinica (u kontroler magistrale je ugrađena lista

adresa po prioritetima). Od svih jedinica koje su poslale zahteve, prva se proziva ona koja ima najviši priritet. Ona prepoznaje svoju adresu i postaje selektovana, a prozivanje se prekida. Nakon toga, kontroler šalje signal *BG* koga prihvata samo selektovana jedinica. Ona generiše signal *BB* i postaje gazda na magistrali.

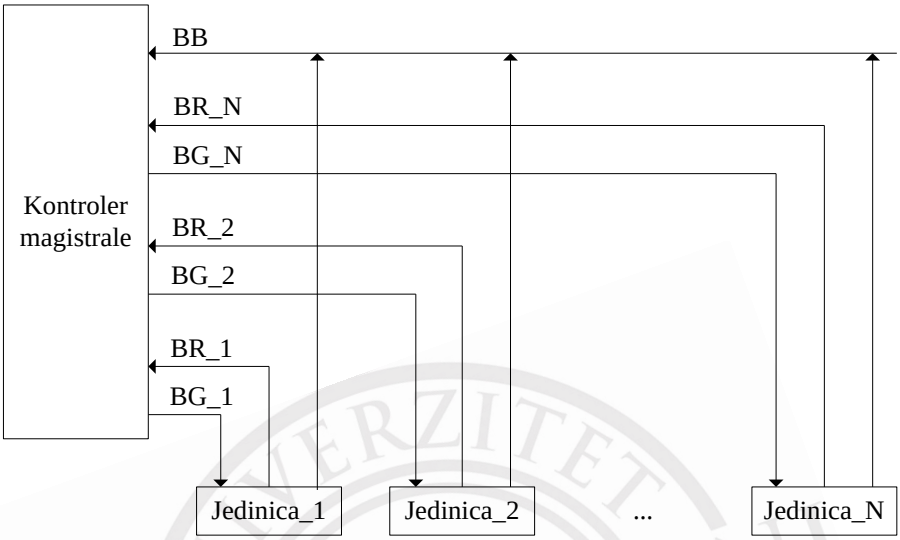


Slika 14.4 Tehnika prozivanja

14.2.3 Tehnika nezavisni zahtev/dozvola

Tehnika nezavisni zahtev/dozvola podrazumeva da *svaka jedinica ima nezavisnu BR i BG liniju* do kontrolera magistrale (slika 14.5).

Arbitracija počinje kada jedna ili više jedinica pošalju zahteve po linijama *BR* kontroleru magistrale. *Kontroler određuje koja* od *jedinica* koje su poslale zahteve *ima najviši prioritet*, pa samo njoj šalje signal dozvole po njenoj *BG* liniji. Jedinica prima *BG* signal, postavlja *BB* signal i postaje gazda na magistrali. Da bi ova tehnika bila izvodljiva, logika koja određuje prioritete jedinica mora biti ugrađena u kontroler magistrale.



Slika 14.5 Tehnika nezavisni zahtev/dozvola

14.2.4 Procesor kao arbitrator

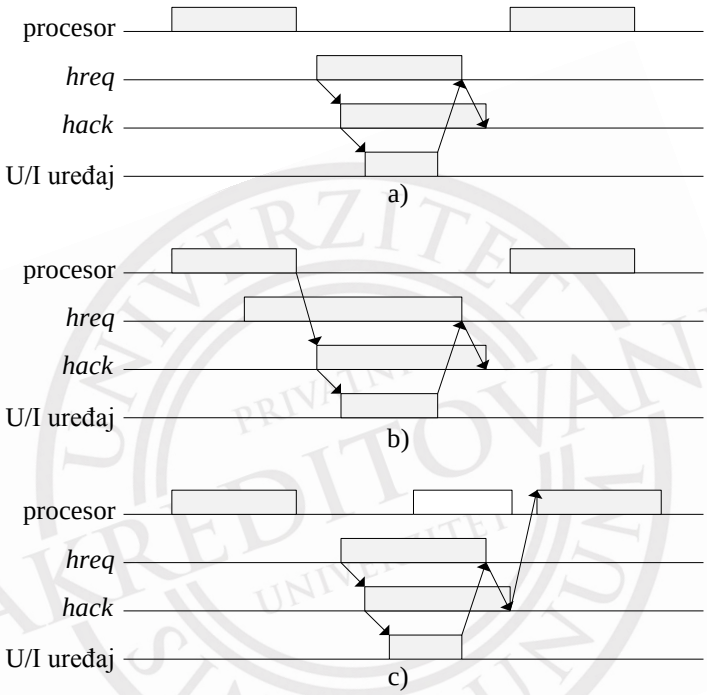
U sistemima u kojima procesor ima ulogu arbitratora, magistrala je uvek u posedu procesora. *U/I* uređaj mora procesoru da pošalje zahtev, pa tek kada od njega dobije dozvolu, može da izvrši ciklus na magistrali. Komunikacija između procesora i *U/I* uređaja se ostvaruje preko dve linije:

- *hreq*, po kojoj *U/I* uređaj šalje zahtev za korišćenje magistrale
- *hack*, po kojoj procesor šalje dozvolu za korišćenje magistrale

Komunikacija počinje tako što *U/I* uređaj postavlja signal na *hreq* liniji na aktivnu vrednost, a zatim čeka dovolu za pristup magistrali u vidu aktivnog signala na *hack*. Nakon dobijanja dozvole, *U/I* uređaj realizuje ciklus na magistrali, sve vreme držeći aktivnu vrednost signala na *hreq*. Po završetku ciklusa, *U/I* uređaj postavlja signal na *hreq* liniji na neaktivnu vrednost. Zatim, procesor postavlja neaktivnu vrednost signala na *hack* liniju i *U/I* gubi dozvolu za korišćenje magistrale.

U komunikaciji su moguće tri situacije koje su prikazane na slici 14.6. Slika 14.6a ilustruje situaciju kada ni procesor ni *U/I* uređaj ne čekaju na realizaciju ciklusa na magistrali. U ovom slučaju, *U/I* uređaj upućuje zahtev u trenutku kada je magistrala slobodna, tako da odmah (sa malim zakašnjenjem) dobija dozvolu i realizuje ciklus. Na slici 14.6b prikazana je situacija u kojoj *U/I* uređaj upućuje

zahtev u trenutku kada procesor već koristi magistralu. Stoga, procesor mora najpre da završi svoj ciklus na magistrali, pa tek onda da pošalje dozvolu *U/I* uređaju. Moguća je i situacija u kojoj procesor čeka da *U/I* uređaj završi ciklus na magistrali (slika 14.6c). Iz ovih situacija se može zaključiti da kada neka jedinica dobije magistralu, ona joj ne može biti oduzeta sve dok ne završi svoj ciklus na magistrali.



Slika 14.6 Komunikacija između procesora i *U/I* uređaja

Na magistrali se mogu realizovati: ciklus upisa, ciklus čitanja i ciklus prihvatanja koda prekida.

14.3 Vrste magistrala

U zavisnosti od dinamike aktivnosti na magistrali, postoje dve vrste magistrala: asinhrone i sinhrone. *Asinhronim* se nazivaju one magistrale na koje su priključuju jedinice koje pri radu koriste sopstveni signal takta. Stoga se na asinhrone magistrale mogu priključivati vrlo raznovrsne jedinice. Za sinhronizaciju ovih jedinica, koriste se *handshake* protokoli. *Sinhronim* se smatraju one magistrale na koje se priključuju jedinice koje koriste isti (zajednički) signal takta. Ove magistrale imaju fiksni protokol komunikacije, relativno u odnosu na takt. To znači

da se unapred zna koliko taktova traje upis/čitanje, pa se nakon tog vremena smatra da je operacija upisa/čitanja završena, tj. pri upisu, da je podatak smešten na odredište, a pri čitanju, da je podatak raspoloživ na *DBUS*. Pošto je protokol unapred definisan, sinhrona magistrala može biti vrlo brza. Glavni nedostatak sinhrone magistrale je taj što ona mora da bude relativno kratka kako bi prenos u taktu mogao regularno da se obavi.

Sledi objašnjenje načina rada asinhrone magistrale pri realizaciji različitih ciklusa na magistrali.

Ciklus čitanja podatka se realizuje tako što gazda šalje adresu podatka na *ABUS* i aktivan signal na *RDBUS*, čime zahteva čitanje podatka u slugi. Po završetku čitanja, sluga šalje podatak na *DBUS* i signal *FCBUS* koji gazdi signalizira da je podatak raspoloživ.

Ciklus upisa podatka se realizuje tako što gazda šalje adresu na *ABUS*, podatak na *DBUS* i aktivan signal na *WRBUS*, čime startuje upis. Nakon upisa podatka, sluga šalje gazdi signal *FCBUS*, čime signalizira da mu podatak i adresa više nisu potrebni.

Ciklus prihvatanja koda prekida se realizuje tako što procesor šalje signal potvrde prekida *inta* i startuje čitanje registra *ER* u *U/I* uređaju. Nakon čitanja, *U/I* uređaj šalje pročitani sadržaj na *DBUS* i signal *FCBUS* da je kod prekida raspoloživ.

Rad sinhrone magistrale pri realizaciji ciklusa čitanja, upisa i prihvatanja koda prekida je opisan u nastavku.

Ciklus čitanja podatka se realizuje tako što gazda šalje adresu podatka na *ABUS* i aktivan signal na *RDBUS*, čime zahteva čitanje u slugi. Pošto je vreme čitanja podatka fiksno, nakon tog vremena, gazda očekuje da je podatak raspoloživ na *DBUS* linijama, pa ga upisuje u prihvatni registar podatka, a adresu sa *ABUS* uklanja.

Pri realizaciji ciklusa upisa, gazda šalje adresu na *ABUS*, podatak na *DBUS* i aktivan signal na *WRBUS*, čime zahteva upis. Pošto je vreme upisa fiksno, posle određenog vremena, gazda očekuje da je podatak upisan i uklanja adresu i podatak sa *ABUS* i *DBUS*, respektivno.

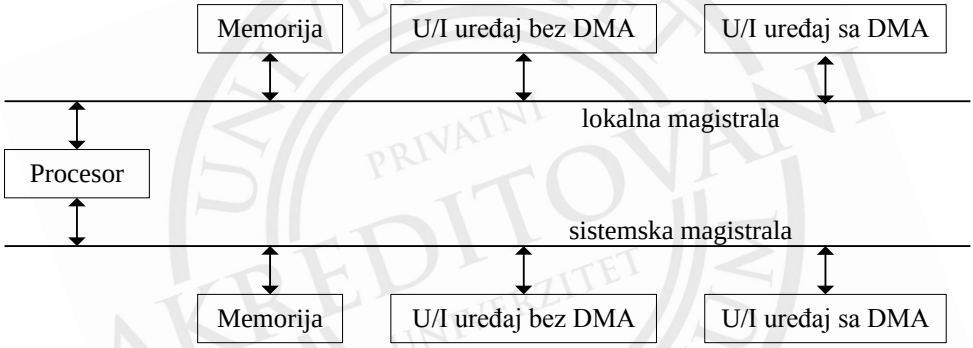
Ciklus prihvatanja koda prekida se realizuje tako što procesor šalje signal potvrde prekida *inta* i zahteva čitanje registra *ER* u *U/I* uređaju. Pošto je vreme čitanja fiksno, nakon tog vremena, gazda očekuje da je kod prekida na *DBUS* i upisuje ga u svoj prihvatni registar.

Mnogi savremeni računarski sistemi se projektuju tako da imaju više magistrala. Na taj način se može smanjiti vreme čekanja na realizaciju ciklusa na

magistrali, a takođe, mogu se efikasno kombinovati osobine asinhrone i sinhrone magistrale.

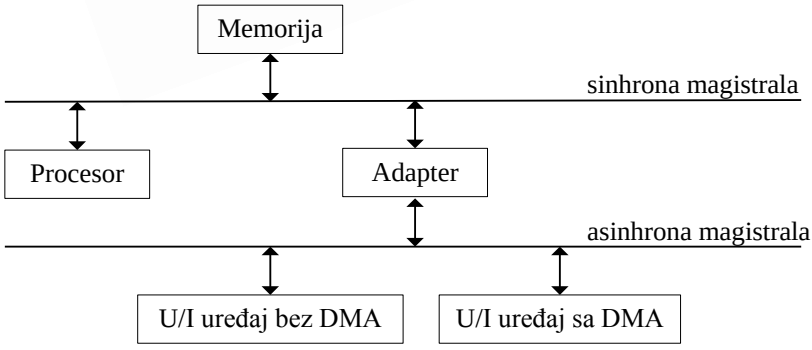
Uobičajena struktura sistema sa dve magistrale je prikazana na slici 14.7. Jedna od magistrala se naziva sistemskom, a druga lokalnom i često se obe magistrale realizuju kao asinhrone.

Kao što se vidi, neki od uređaja su priključeni na sistemsku, a neki na lokalnu magistralu, dok je procesor priključen na obe magistrale. U zavisnosti od toga kako su priključeni uređaji, adresni prostor je podeljen na dva dela. Jedan deo čine adrese za adresiranje lokacija u uređajima priključenim na sistemsku, a drugi deo na lokalnu magistralu. Procesor na osnovu adrese određuje na kojoj magistrali se ciklus realizuje. Zbog postojanja dve magistrale, procesor kao gazda, brže realizuje svoje cikluse na magistrali jer može da koristi obe magistrale.



Slika 14.7 Sistem sa sistemskom i lokalnom magistralom

U nekim konfiguracijama je moguće kombinovati magistrale različitog tipa. Na slici 14.8 je prikazan sistem sa jednom asinhronom i jednom sinhronom magistralom.



Slika 14.8 Sistem sa asinhronom i sinhronom magistralom

Adresni prostor je i u ovom slučaju podeljen na dva dela. Jedan deo čine adrese memorijskih lokacija, dok drugi deo čine adrese registara u *U*/*I* uređajima. Procesor sve cikluse realizuje na sinhronoj magistrali. Na adrese iz *U*/*I* adresnog prostora reaguje adapter, koji realizuje cikluse sa *U*/*I* uređajima na asinhronoj magistrali.

Vežbanja

P1. Šta je magistrala? Navesti uloge koje magistrala ima u računarskom sistemu.

P2. Šta predstavlja i kako se ostvaruje ciklus na magistrali? Ko može da bude gazda, a ko sluga na magistrali? Koji ciklusi se mogu realizovati na magistrali?

P3. Koje vrste *U*/*I* adresnih prostora postoje? U čemu je razlika između njih?

P4. Šta se podrazumeva pod arbitracijom na magistrali? Koje komponente mogu da imaju ulogu arbitratora na magistrali? Šta predstavljaju signali *BB*, *BR* i *BG* u procesu arbitracije?

P5. Koje tehnike arbitracije postoje?

P6. Objasniti tehniku ulančavanja u procesu arbitracije. Dati sliku.

P7. U čemu je razlika između tehnike ulančavanja i tehnike prozivanja?

P8. Kako se odvija proces arbitracije ako se koristi tehnika nezavisni zahtev/dozvola? Dati sliku.

P9. U čemu je razlika između asinhrone i sinhrone magistrale?

P10. Kako se realizuje ciklus prihvatanja koda prekida na asinhronoj magistrali?

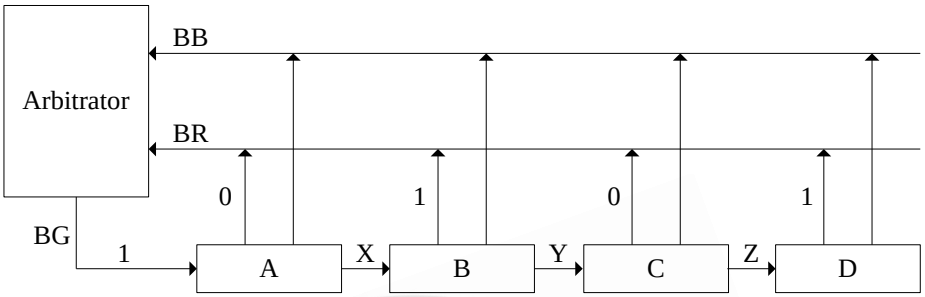
P11. Kako se realizuje ciklus upisa na sinhronoj magistrali?

Z1. Procesor je povezan sinhronom magistralom sa memorijom kapaciteta 512M 32-bitnih reči. Koliku širinu imaju adresna, kontrolna i magistrala podataka?

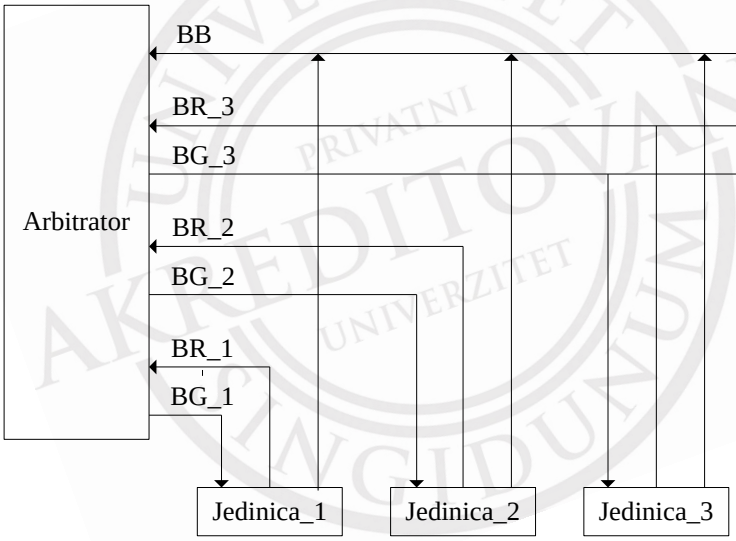
Z2. Na magistralu su priključena četiri uređaja *A*, *B*, *C* i *D*. Arbitracija se izvodi primenom tehnike ulančavanja. Za signale date na slici, odrediti:

a) vrednost signala *X*, *Y* i *Z*

b) koji uređaj šalje signal na *BB* liniju



Z3. Tri jedinice su priključene na magistralu kao na slici.



Logika rada arbitratora data je sledećom tablicom:

BR_3	BR_2	BR_1	BG_3	BG_2	BG_1
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

Odrediti koja jedinica će dobiti magistralu ako istovremeno:

a) prva i druga jedinica pošalju zahteve
b) sve tri jedinice pošalju zahteve

Z4. Na magistralu koja koristi tehniku prozivanja su priključena četiri uređaja *A*, *B*, *C* i *D* čije su adrese 00, 01, 10 i 11, respektivno. Arbitrator implementira sledeću sekvencu prozivanja: 10, 00, 11, 01.

a) Ako uređaji *B* i *D* istovremeno zahtevaju izlazak na magistralu, koji od njih će dobiti magistralu i zašto?
b) Za svaki od uređaja, predložiti logički sklop koji unutar uređaja prepoznaje adresu tog uređaja.

Literatura

1. John Hennessy and David Patterson, *Computer Architecture: A Quantitative Approach*, Elsevier, 2012.

2. Andrew Tanenbaum, *Arhitektura i organizacija računara*, Mikro knjiga (prevod), 2007.

3. Jovan Đorđević, Arhitektura i organizacija računara – materijali, http://rti.etf.bg.ac.rs/, 2015.

4. William Stallings, *Computer Organization and Architecture: Designing for Performance*, 8th Edition, Prentice Hall, 2010.

5. Mostafa Abd-El-Barr and Hesham El-Rewini, *Fundamentals of Computer Organization and Architecture*, Wiley & Sons, Series on Parallel and Distributed Computing, 2004.

6. Sajjan Shiva, *Computer Organization, Design and Architecture*, 4th Edition, Taylor & Frances Group, 2007.

7. Morris Mano, *Computer System Architecture*, ISBN: 978-0131755635. 1992.

8. Carl Hammaher, Zvonko Vranešić and Safwat Zaky, *Computer Organization*, 5th Edition, Mc Graw-Hill, Inc. 2002.

9. Darryl Gove, *Multicore Application Programming*, Addison-Wesley, 2011.

10. Shammem Akhter and Jason Roberts, *Multi-Core Programming: Increasing Performance through Software Multi-threading*, Intel Press, Digital Edition, 2006.

11. Balaji Venu, *Multi-core processors – An overview*, arxsiv.org, 2011.

12. Andras Vajda, *Programming Many-Core Chips*, Springer Science + Business Media, 2011.

13. Violeta Tomašević, *Osnovi računarske tehnike*, Univerzitet Singidunum, 2016.

CIP - Каталогизација у публикацији - Народна библиотека Србије, Београд

004.2(075.8)

ТОМАШЕВИЋ, Виолета, 1965-
Osnovi arhitekture i organizacije računara / Violeta Tomašević. - 1. izd. - Beograd : Univerzitet Singidunum, 2019 (Beograd : Caligraph). - 230 str. : ilustr. ; 24 cm

Tiraž 1.200. - Bibliografija: str. [231].

ISBN 978-86-7912-696-2

a) Архитектура рачунара

COBISS.SR-ID 273604108

www.singidunum.ac.rs

9 788679 126962

Violeta Tomašević

OSNOVI ARHITEKTURE I ORGANIZACIJE RAČUNARA

Ova knjiga je nastala kao rezultat potrebe za odgovarajućim udžbenikom iz predmeta Arhitektura računara koji autorka drži na Fakultetu za informatiku i računarstvo i Tehničkom fakultetu Univerziteta Singidunum. Pri pisanju je učinjen napor da knjiga bude prihvatljiva za čitaoce bez nekog većeg predznanja iz oblasti računarstva. Namenjena je i prilagođena prosečnom studentu, jer je osnovni cilj autorke bio da svi studenti koji slušaju predmet mogu na razumljiv i lak način da savladaju predviđeno gradivo. Upravo iz tog razloga, knjiga ne prikazuje punu širinu i složenost razmatranih problema, već je prvenstveno orijentisana ka praktičnim aspektima koji su ilustrovani brojnim primerima.

Knjiga je podeljena u četrnaest poglavlja: Matematičke osnove, Logička kola, Logičke funkcije, Standardni moduli, Komponente računara, Mehanizmi, Programske instrukcije, Procesorski registri, Adresni modovi, Instrukcijski set, Programiranje, Memorijski sistem, Organizacija ulaza/izlaza i Magistrala.

Na kraju svakog poglavlja, u okviru Vežbanja, data su pitanja i zadaci za samostalno rešavanje koji studentima treba da posluže kao provera znanja na temu koja je razmatrana u poglavlju. Pitanja i zadaci su odabrani tako da u potpunosti pokrivaju predviđeno gradivo, pa se mogu iskoristiti za pripremanje ispita.

Univerzitet Singidunum