

JOVAN ĐORĐEVIĆ

**ARHITEKTURA
I
ORGANIZACIJA
RAČUNARA**

ARHITEKTURA RAČUNARA

BEOGRAD, 2012.

DJM

PREDGOVOR

Ova knjiga je napisana kao osnovni udžbenik iz arhitekture i organizacije računara i pokriva osnovne koncepte iz arhitekture i organizacije procesora, memorije, ulaza/izlaza i magistrale.

Sistemi.

Autor

Beograd

novembra 2011.

SADRŽAJ

PREDGOVOR	1
SADRŽAJ	3
1 ARHITEKTURA RAČUNARA	7
1.1 PROGRAMSKI DOSTUPNI REGISTRI	7
1.2 TIPOVI PODATAKA	20
1.2.1 <i>Opšte napomene</i>	20
1.2.2 <i>Predstavljanje podataka</i>	20
1.2.2.1 CELOBROJNE VELIČINE	20
1.2.2.2 VELIČINE U POKRETNOM ZAREZU	21
1.2.2.3 ALFANUMERIČKI NIZ	22
1.2.2.4 NUMERIČKI NIZ	22
1.3 FORMATI INSTRUKCIJA	24
1.3.1 <i>OPERACIJA I TIP PODATKA</i>	25
1.3.2 <i>IZVORIŠNI I ODREDIŠNI OPERANDI</i>	25
1.3.2.1 Broj eksplicitno specificiranih operanada	25
1.3.2.2 Moguće lokacije operanada.....	29
1.4 NAČINI ADRESIRANJA	30
1.4.1 <i>Registarsko direktno adresiranje</i>	30
1.4.2 <i>Registarsko indirektno adresiranje</i>	31
1.4.3 <i>Memorijsko direktno adresiranje</i>	33
1.4.4 <i>Memorijsko indirektno adresiranje</i>	34
1.4.5 <i>Bazno adresiranje sa pomerajem</i>	35
1.4.6 <i>Indeksno adresiranje sa pomerajem</i>	36
1.4.7 <i>Registarsko indirektno adresiranje sa pomerajem</i>	37
1.4.8 <i>Bazno-indeksno adresiranje sa pomerajem</i>	38
1.4.9 <i>Registarska indirektna adresiranja sa autoinkrement i autodekrementiranjem</i>	40
1.4.10 <i>Relativno adresiranje sa pomerajem</i>	41
1.4.11 <i>Neposredno adresiranje</i>	41
1.5 SKUP INSTRUKCIJA	42
1.5.1 <i>STANDARDNE INSTRUKCIJE</i>	42
1.5.1.1 INSTRUKCIJE PRENOSA	42
1.5.1.2 ARITMETIČKE INSTRUKCIJE.....	44
1.5.1.3 LOGIČKE INSTRUKCIJE	56
1.5.1.4 INSTRUKCIJE POMERANJA	57
1.5.1.5 INSTRUKCIJE SKOKA.....	60
1.5.1.6 MEŠOVITE INSTRUKCIJE	64
1.5.2 <i>NESTANDARDNE INSTRUKCIJE</i>	68

1 ARHITEKTURA RAČUNARA

U ovoj glavi se razmatraju elementi arhitekture računara koju čine programski dostupni registri, tipovi podataka, formati instrukcija, načini adresiranja, skup instrukcija i mehanizam prekida.

1.1 PROGRAMSKI DOSTUPNI REGISTRI

Programski dostupni registri procesora su registri u koje je moguće programskom putem izvršavanjem instrukcija procesora upisivati vrednosti i iz kojih je moguće programskom putem izvršavanjem instrukcija procesora očitavati vrednosti. U instrukcijama kojima se pristupa ovim registrima se registar u koji treba upisati vrednost ili iz koga treba očitati vrednost specificira ili eksplicitno nekim od adresnih polja instrukcije ili implicitno kodom operacije instrukcije. Ovi registri su namenjeni da se u njih nekom od instrukcija upiše neka vrednost, pa da se ta vrednost nekim kasnijim instrukcijam čita. Ovi registri su deo arhitekture računara.

Ove registre treba razlikovati od registara procesora koje projektant procesora ubacuje da bi prema nekom svom pristupu projektovanja procesora obezbedio čuvanje neophodnih sadržaja prilikom prolaska kroz sve korake iz kojih se sastoji izvršavanje jedne instrukcije. Vrednost koja se u neki od ovih registara upisuju u nekom od koraka izvršavanja instrukcije koriste se u nekom ili nekim kasnijim koracima izvršavanja te iste instrukcije, ali ne neke od sledećih instrukcija. Upisivanje vrednosti u ove registre i čitanje vrednosti iz ovih registara je nemoguće specificirati instrukcijama. To je određeno usvojenim algoritmima izvršavanja svake instrukcije posebno. Ovi registri su deo organizacije računara.

Funkcije i broj programski dostupnih registara se razlikuje od procesora do procesora. Ovde se daju oni programski dostupni registri koji se često sreću kod komercijalno raspoloživih procesora i to programski brojač PC, registri podataka DR, adresni registri AR, bazni registri BR, indeksni registri XR, registri opšte namene GPR, programska statusna reč PSW, ukazivač na vrh steka SP i ukazivač na okvir steka FP.

Programski brojač PC je standardni programski brojač procesora čija se vrednost implicitno koristi kao adresa memorijske lokacije sa koje se čita instrukcija. Vrednost programskog brojača PC se menja i to implicitno prilikom svakog čitanja instrukcije kada se vrši inkrementiranje programskog brojača PC (odeljak 1.3) i eksplicitno instrukcijama skoka kada se u programski brojač PC upisuje nova vrednost (odeljak 1.5.1.5).

Registri podataka DR se koriste kod registarskog direktnog adresiranja (odeljak 1.4.1). Registrima podataka se pristupa programskim putem instrukcijama prenosa (odeljak 1.5.1.1) i aritmetičkim (odeljak 1.5.1.2), logičkim (odeljak 1.5.1.3) i pomeračkim instrukcijama (odeljak 1.5.1.4). Registri podataka se uvode da bi se ubrzao pristup podacima, tako što bi se tokom izvršavanja programa pristupalo podacima u registrima podataka procesora umesto u memorijskim lokacijama, a rezultat je činjenice da je pristup registrima skoro za red veličine brži od pristupa memorijskim lokacijama. Korišćenje registara podataka radi ubrzavanja pristupa podacima ima smisla ukoliko se u toku nekog računanja javi potreba da se viša puta koristi neki podatak. U ovom slučaju, moguće je, najpre, dati podatak, programskim putem izvršavanjem instrukcije prenosa, prebaciti iz memorijske lokacije u neki od registara podataka, a zatim, kad god postoji potreba za datim podatkom, podatak čitati iz registra

podatka. Ubrzanje se postiže time što umesto da se svaki put kada postoji potreba za datim podatkom ide u memorijsku lokaciju ide se u registar podatka. Pored toga moguće je međurezultate računanja ostavljati u registrima podataka, da bi kasnije, kada oni budu potrebni, njima moglo da se pristupa čitanjem odgovarajućih registara podataka. Ubrzanje se postiže time što se međurezultat ne smešta u memorijsku lokaciju i što kasnije kada postoji potreba za njim umesto da se ide u memorijsku lokaciju ide se u registar podatka. Konačne rezultate računanja treba na kraju, programskim putem izvršavanjem instrukcija prenosa, prebaciti iz registara podataka u memorijske lokacije. Ovakvo korišćenje registrima podataka ima opravdanja iz dva razloga. Prvi je uočeni vremenski lokalitet prilikom rada sa skalarnim veličinama koji ukazuje da ako se jedanput pristupilo nekom podatku postoji velika verovatnoća da će se posle toga više puta javiti potreba za pristup istom podatku. Drugi je sekvencijalna priroda računanja za koju je karakteristično da se rezultat jedne operacije veoma često koristi kao podatak za drugu operaciju.

Adresni registri AR se koriste kod registarskog indirektnog adresiranja (odeljak 1.4.2) i autoinkrement i autodekrement načina adresiranja (odeljak 1.4.9). Adresnim registrima se pristupa programskim putem instrukcijama prenosa (odeljak 1.5.1.1) i aritmetičkim (odeljak 1.5.1.2), logičkim (odeljak 1.5.1.3) i pomeračkim instrukcijama (odeljak 1.5.1.4) kada se sadržaj adresnog registra koristi kao adresa memorijske lokacije sa koje se čita ili u koju se upisuje podatak, kao i instrukcijom skoka JMPIND (odeljak 1.5.1.5) kada se sadržaj adresnog registra koristi kao adresa skoka koju treba upisati u programski brojač PC. Adresni registri se uvode da bi se ubrao pristup adresama, tako što bi se tokom izvršavanja programa pristupalo adresama u adresnim registrima procesora umesto u memorijskim lokacijama, a rezultat je činjenice da je pristup registrima skoro za red veličine brži od pristupa memorijskim lokacijama. Adresni registar i odgovarajuće adresiranje se koriste u situacijama kada tokom izvršavanja programa treba najpre sa nekoliko instrukcija izračunati adresu elementa neke složene strukture podataka i smestiti je u neki od adresnih registara, pa onda ili instrukcijom prenosa ili aritmetičkom, logičkom ili pomeračkom instrukcijom sadržaj adresnog registra koristiti kao adresu memorijske lokacije sa koje se čita ili u koju se upisuje podatak. Adresni registar i odgovarajuće adresiranje se koriste i u situacijama kada tokom izvršavanja programa treba najpre sa nekoliko instrukcija izračunati adresu skoka, pa onda instrukcijom skoka JMPIND sadržaj adresnog registra koristiti kao adresu skoka koju treba upisati u programski brojač PC. U oba slučaja je moguće sračunatu adresu upisati umesto u adresni registar u memorijsku lokaciju i do te adrese doći memorijskim indirektnim adresiranjem.

Bazni registri BR se koriste kod baznog (odeljak 1.4.5) i bazno-indeksnog adresiranja (odeljak 1.4.8). Zbir sadržaja specificiranog baznog registra i pomeraja kod baznog adresiranja, odnosno baznog registra, indeksnog registra i pomeraja kod bazno-indeksnog adresiranja, predstavlja adresu memorijske lokacije na kojoj se nalazi izvorišni ili odredišni operand.

Indeksni registri XR se koriste kod indeksnog (odeljak 1.4.6) i bazno-indeksnog adresiranja (odeljak 1.4.8). Zbir sadržaja specificiranog indeksnog registra i pomeraja kod indeksnog adresiranja, odnosno baznog registra, indeksnog registra i pomeraja kod bazno-indeksnog adresiranja, predstavlja adresu memorijske lokacije na kojoj se nalazi izvorišni ili odredišni operand.

Registri opšte namene GPR se koriste na isti način kao registri podataka, adresni registri, bazni registri i indeksni registri. Registri opšte namene GPR se javljaju kod onih procesora kod kojih ne postoje posebni registri podataka, adresni registri, bazni registri i indeksni registri. Za razliku od procesora sa posebnim registrima podataka, adresnim registrima,

baznim registrima i indeksnim registrima, gde se registri iz svake grupe registara koriste samo uz odgovarajuća adresiranja, procesori sa registrima opšte namene mogu da koriste bilo koji od registara uz bilo koje adresiranje.

Registar PSW je standardna programska statusna reč procesora sastavljena od određenog broja bitova, koji se obično nazivaju indikatori. Bitovi programske statusne reči PSW se nezavisno postavljaju i koriste po pravilima definisanim posebno za svaki bit. Međutim, u određenim situacijama, kao kada se skače na prekidnu rutinu i vraća iz nje, sa bitovima programske statusne reči PSW se postupa na isti način, pa se zato uzima da oni predstavljaju razrede jednog registra. U programskoj statusnoj reči PSW postoje dve grupe bitova i to bitovi statusnog i bitovi upravljačkog karaktera (slika 1). Ovi bitovi se nazivaju i indikatori.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	-	-	-	-	-	-	-	-	-	-	-	V	C	Z	N

Slika 1 Struktura registra PSW

Bitovi statusnog karaktera su:

- N—bit koji se postavlja na 1 u slučaju da je rezultat operacije negativan,
- Z—bit koji se postavlja na 1 u slučaju da je rezultat operacije jednak 0,
- C—bit koji se postavlja na 1 u slučaju prenosa/pozajmice u aritmetici celobrojnih veličina bez znaka i
- V—bit koji se postavlja na 1 u slučaju prekoračenja u aritmetici celobrojnih veličina sa znakom (odjeljak 1.5.1.2).

Bit upravljačkog karaktera je:

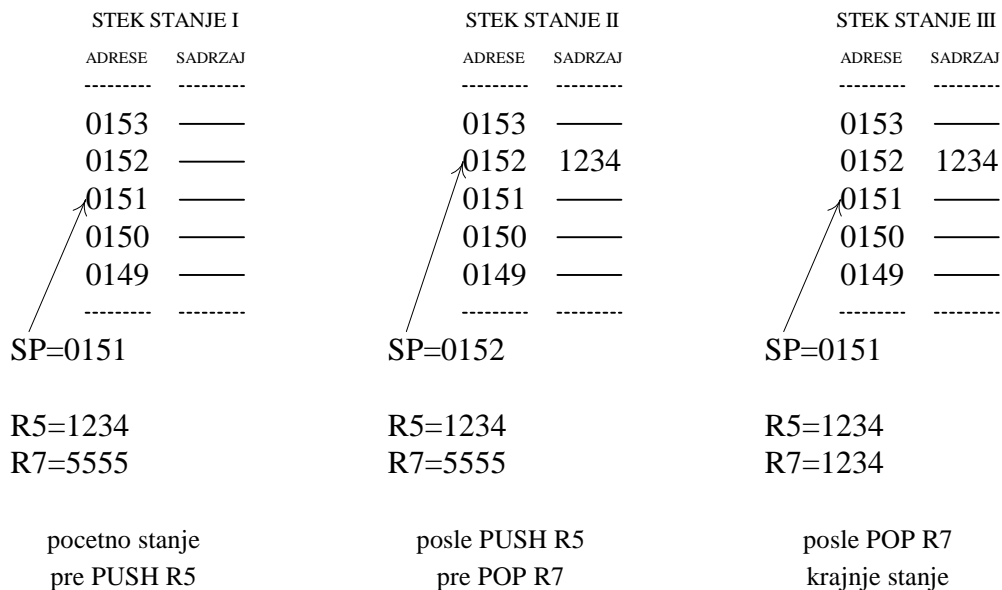
- I—bit koji je jednak 1 ako treba da budu dozvoljeni maskirajući prekidi.

Bitovi statusnog karaktera N, Z, C i V, se postavljaju hardverski na osnovu rezultata izvršavanja instrukcija (odjeljak 1.5.1.2, 1.5.1.3 i 1.5.1.4), a proveravaju softverski instrukcijama uslovnog skoka (odjeljak 1.5.1.5). Bitovi upravljačkog karaktera se postavljaju softverski kao rezultat izvršavanja posebnih instrukcija (odjeljak 1.5.1.6), a proveravaju hardverski u saglasnosti sa algoritmom izvršavanja instrukcija.

Registar SP je ukazivač na vrh steka kada je stek je organizovan u operativnoj memoriji. Registar SP se pojavljuje u procesorima kao podrška za realizaciju LIFO strukture podataka. Prilikom upisa u LIFO strukturu podataka ili čitanja iz LIFO strukture podataka sadržaj registra SP se koristi kao adresa memorijske lokacije u koju treba upisati podatak ili iz koje treba očitati podatak. Tom prilikom se vrši i ažuriranje i to inkrementiranje ili dekrementiranje sadržaja registra SP. LIFO strukturu podataka je moguće realizovati na četiri načina u zavisnosti od toga da li se kod upisa vrši inkrementiranje ili dekrementiranje sadržaja registra SP, pa se kaže da stek raste prema višim ili prema nižim lokacijama, i da li sadržaj registra SP ukazuje na zadnju zauzetu lokaciju ili prvu slobodnu lokaciju. Realizacija upisa na stek i čitanja sa steka za četiri moguće realizacije steka su prikazane u daljem tekstu.

Stek raste prema višim lokacijama a registar SP ukazuje na zadnju zauzetu lokaciju

Kod upisa na stek najpre se vrši inkrementiranje registra SP, pa sa posle toga sadržaj registra SP koristi kao adresa memorijske lokacije u koju se upisuje. Kod čitanja sa steka najpre se sadržaj registra SP koristi kao adresa memorijske lokacije sa koje se čita sadržaj, pa se posle toga vrši dekrementiranje sadržaja registra SP. Upisa na stek i čitanje sa steka za ovakvu realizaciju steka su ilustrovani na slici 2 .



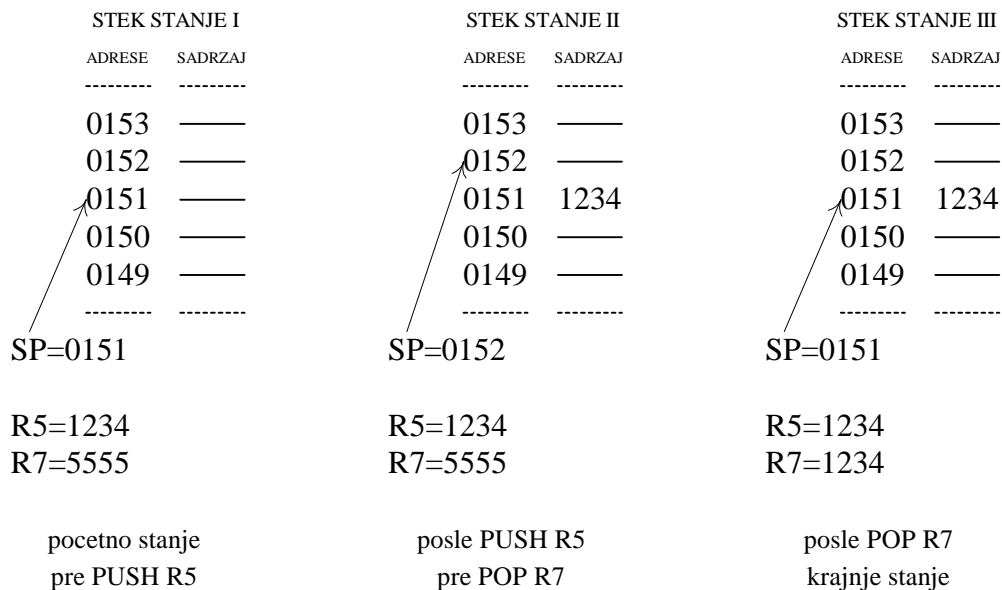
Slika 2 Stek raste prema višim lokacijama i registar SP ukazuje na zadnju zauzetu lokaciju

Kao početno stanje pre izvršenja instrukcije PUSH R5 je uzeto da se u registru R5 nalazi vrednost 1234, u registru R7 vrednost 5555 i da registar SP ukazuje na memorijsku lokaciju na adresi 0151. Uzeto je da se najpre instrukcijom PUSH R5 sadržaj registra R5 stavlja na vrh steka. U okviru izvršavanja instrukcije PUSH R5 sadržaj 0151 registra SP se najpre inkrementira pa se posle njegov sadržaj 0152 koristi kao adresa memorijske lokacije u koju se upisuje sadržaj 1234 registra R5. Potom se instrukcijom POP R7 skida sadržaj sa vrha steka i upisuje u registar R7. U okviru izvršavanja instrukcije POP R7 sadržaj 0152 registra SP se koristi kao adresa memorijske lokacije sa koje se najpre čita sadržaj 1234 i upisuje u registar R7, pa se posle sadržaj registra SP dekrementira na vrednost 0151.

Stek raste prema višim lokacijama a registar SP ukazuje na prvu slobodnu lokaciju

Kod upisa na stek najpre se sadržaj registra SP koristi kao adresa memorijske lokacije u koju se upisuje, pa se posle toga vrši inkrementiranje registra SP. Kod čitanja sa steka najpre se vrši dekrementiranje sadržaja registra SP, pa se posle toga sadržaj registra SP koristi kao adresa memorijske lokacije sa koje se čita sadržaj. Upisa na stek i čitanje sa steka za ovakvu realizaciju steka su ilustrovani na slici 3.

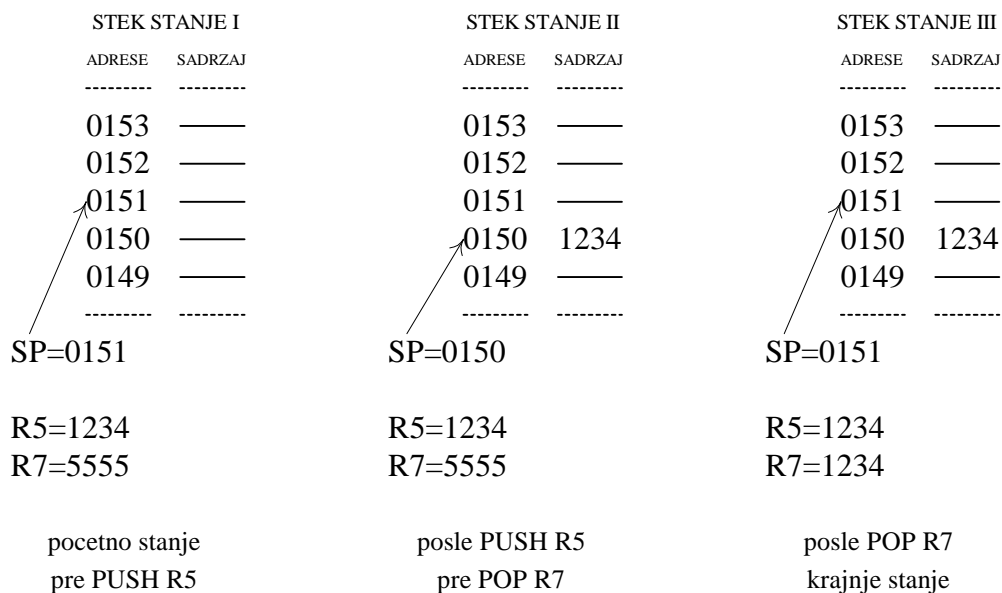
Kao početno stanje pre izvršenja instrukcije PUSH R5 je uzeto da se u registru R5 nalazi vrednost 1234, u registru R7 vrednost 5555 i da registar SP ukazuje na memorijsku lokaciju na adresi 0151. Uzeto je da se najpre instrukcijom PUSH R5 sadržaj registra R5 stavlja na vrh steka. U okviru izvršavanja instrukcije PUSH R5 najpre se sadržaj 0151 registra SP koristi kao adresa memorijske lokacije u koju se upisuje sadržaj 1234 registra R5, pa se posle njegov sadržaj inkrementira na vrednost 0152. Potom se instrukcijom POP R7 skida sadržaj sa vrha steka i upisuje u registar R7. U okviru izvršavanja instrukcije POP R7 sadržaj 0152 registra SP se najpre dekrementira na vrednost 0151, pa se posle koristi kao adresa memorijske lokacije sa koje se čita sadržaj 1234 i upisuje u registar R7.



Slika 3 Stek raste prema višim lokacijama i registar SP ukazuje na prvu slobodnu lokaciju

Stek raste prema nižim lokacijama a registar SP ukazuje na zadnju zauzetu lokaciju

Kod upisa na stek najpre se vrši dekrementiranje registra SP, pa sa posle toga sadržaj registra SP koristi kao adresa memorijske lokacije u koju se upisuje. Kod čitanja sa steka najpre se sadržaj registra SP koristi kao adresa memorijske lokacije sa koje se čita sadržaj, pa se posle toga vrši dekrementiranje sadržaja registra SP. Upisa na stek i čitanje sa steka za ovakvu realizaciju steka su ilustrovani na slici 4 .



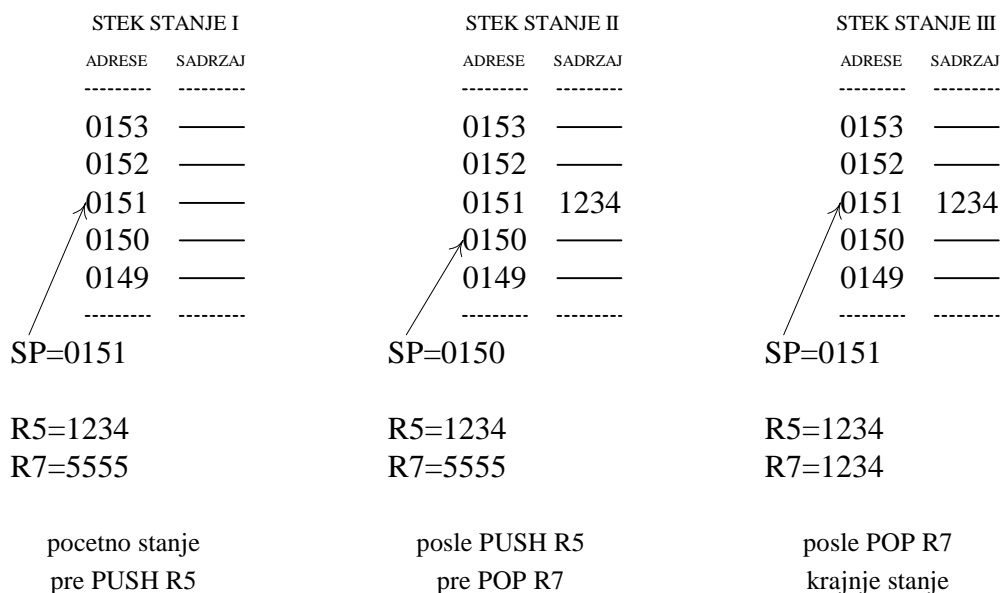
Slika 4 Stek raste prema nižim lokacijama i registar SP ukazuje na zadnju zauzetu lokaciju

Kao početno stanje pre izvršenja instrukcije PUSH R5 je uzeto da se u registru R5 nalazi vrednost 1234, u registru R7 vrednost 5555 i da registar SP ukazuje na memorijsku lokaciju na adresi 0151. Uzeto je da se najpre instrukcijom PUSH R5 sadržaj registra R5 stavlja na vrh steka. U okviru izvršavanja instrukcije PUSH R5 sadržaj 0151 registra SP se najpre dekrementira pa se posle njegov sadržaj 0150 koristi kao adresa memorijske lokacije u koju se upisuje sadržaj 1234 registra R5. Potom se instrukcijom POP R7 skida sadržaj sa vrha steka i upisuje u registar R7. U okviru izvršavanja instrukcije POP R7 sadržaj 0150 registra SP se

koristi kao adresa memorijske lokacije sa koje se najpre čita sadržaj 1234 i upisuje u registar R7, pa se posle sadržaj registra SP inkrementira na vrednost 0151.

Stek raste prema nižim lokacijama a registar SP ukazuje na prvu slobodnu lokaciju

Kod upisa na stek najpre se sadržaj registra SP koristi kao adresa memorijske lokacije u koju se upisuje, pa se posle toga vrši dekrementiranje registra SP. Kod čitanja sa steka najpre se vrši inkrementiranje sadržaja registra SP, pa se posle toga sadržaj registra SP koristi kao adresa memorijske lokacije sa koje se čita sadržaj. Upisa na stek i čitanje sa steka za ovakvu realizaciju steka su ilustrovani na slici 5 .



Slika 5 Stek raste prema nižim lokacijama i registar SP ukazuje na prvu slobodnu lokaciju

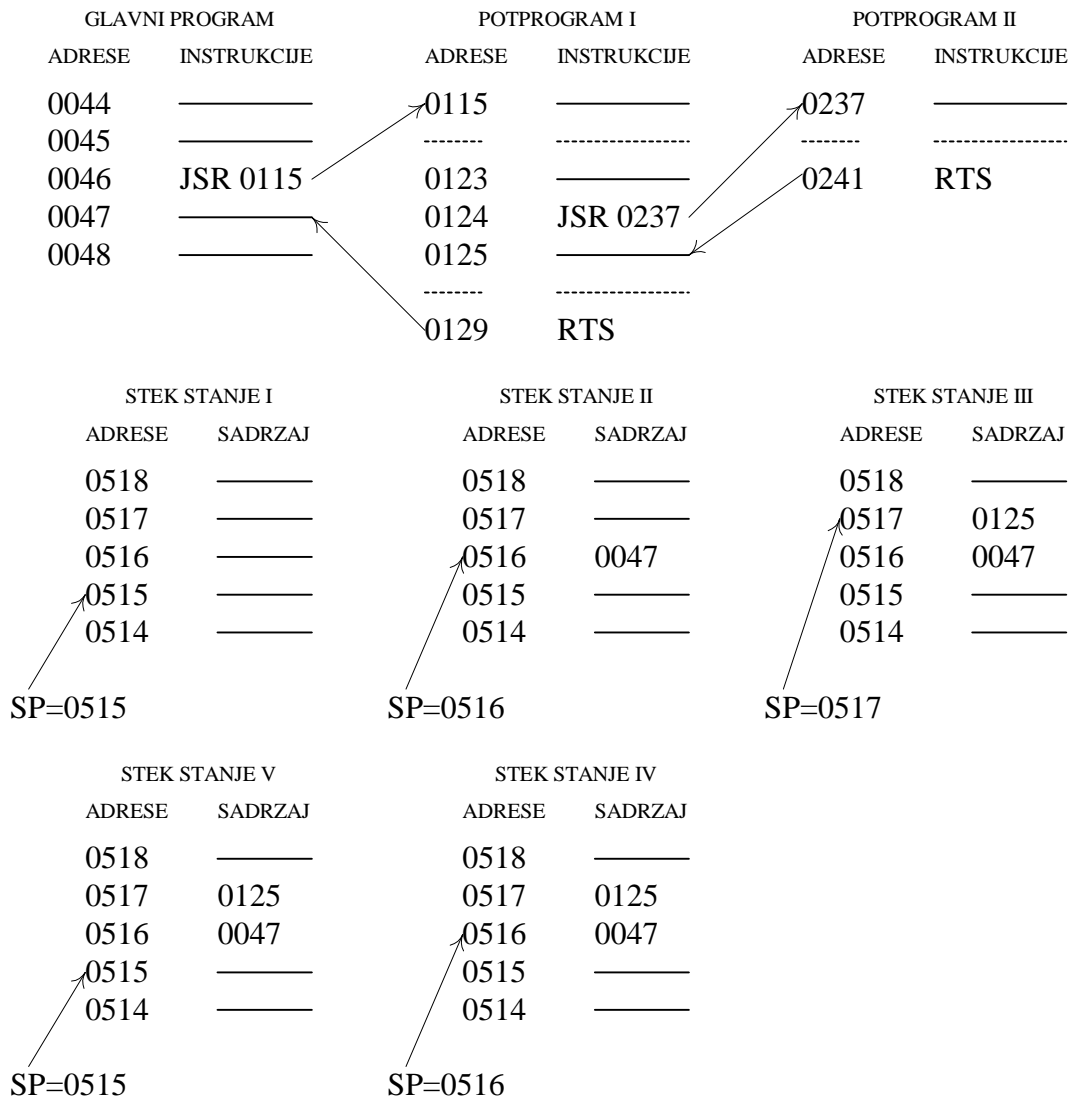
Kao početno stanje pre izvršenja instrukcije PUSH R5 je uzeto da se u registru R5 nalazi vrednost 1234, u registru R7 vrednost 5555 i da registar SP ukazuje na memorijsku lokaciju na adresi 0151. Uzeto je da se najpre instrukcijom PUSH R5 sadržaj registra R5 stavlja na vrh steka. U okviru izvršavanja instrukcije PUSH R5 najpre se sadržaj 0151 registra SP koristi kao adresa memorijske lokacije u koju se upisuje sadržaj 1234 registra R5, pa se posle njegov sadržaj dekrementira na vrednost 0150. Potom se instrukcijom POP R7 skida sadržaj sa vrha steka i upisuje u registar R7. U okviru izvršavanja instrukcije POP R7 sadržaj 0150 registra SP se najpre inkrementira na vrednost 0151, pa se posle koristi kao adresa memorijske lokacije sa koje se čita sadržaj 1234 i upisuje u registar R7.

Stek se koristi kao upravljački stek i kao aritmetički stek. Upravljački stek se koristi kod skoka na potprogram i skoka na prekidnu rutinu za čuvanje informacija neophodnih za povratak iz potprograma i povratak iz prekidne rutine. Aritmetički stek se koristi kod nula-adresnih ili stek procesora kao implicitno izvorište i odredište operanada. Postoje realizacije kod kojih se jedan stek koristi i kao upravljački stek i kao aritmetički stek, a postoje i realizacije kod kojih postoje posebno upravljački stek i posebno aritmetički stek.

Jedna od situacija kada se stek koristi kao upravljački stek prikazana je na slici 6. Uzeto je da stek raste prema višim lokacijama i da registar SP ukazuje na zadnju zauzetu lokaciju.

Tokom izvršavanja glavnog programa dolazi se na instrukciju JSR 0115 sa adrese 0046 kojom treba da se realizuju skok na potprogram I koji počinje od adrese 0115. Stanje na steku je označeno sa stek stanje I. Pri čitanju instrukcije sa adrese 0046 programski brojač PC je inkrementiran i ima vrednost 0047, pa se pri izvršavanju instrukcije JSR 0115 na vrh steka

stavlja 0047 i u programski brojač PC upisuje vrednost 0115. Stanje na steku po skoku na potprogram I je označeno sa stek stanje II.



Slika 6 Upravljački stek

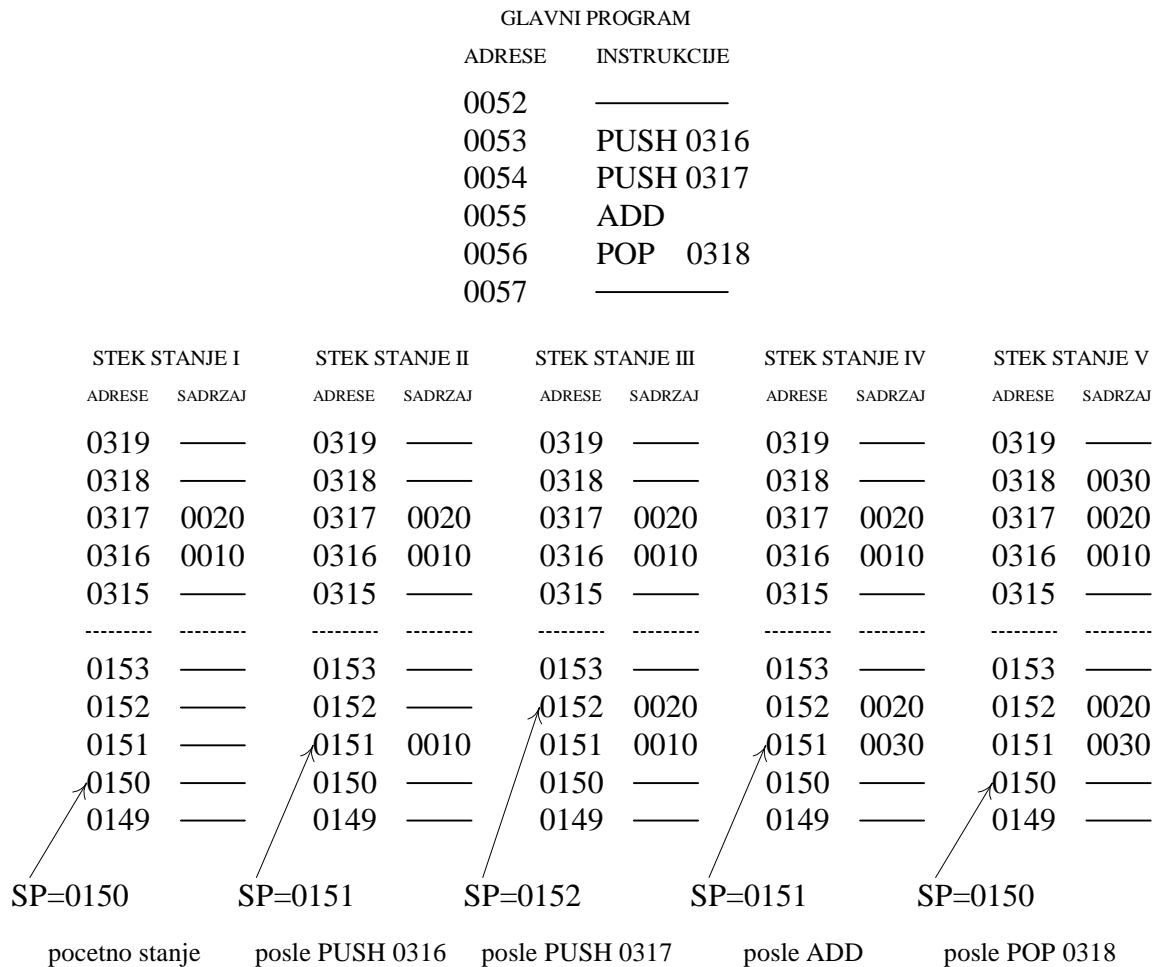
Tokom izvršavanja potprograma I dolazi se na instrukciju JSR 0237 sa adrese 0124 kojom treba da se realizuju skok na potprogram II koji počinje od adrese 0237. Stanje na steku je označeno sa stek stanje II. Pri čitanju instrukcije sa adrese 0124 programski brojač PC je inkrementiran i ima vrednost 0125, pa se pri izvršavanju instrukcije JSR 0237 na vrh steka stavlja 0125 i u programski brojač PC upisuje vrednost 0237. Stanje na steku po skoku na potprogram II je označeno sa stek stanje III.

Tokom izvršavanja potprograma II dolazi se na instrukciju RTS sa adrese 0241 kojom treba da se realizuju povratak na potprogram I i to na instrukciju sa adrese 0125. Stanje na steku je označeno sa stek stanje III. Pri izvršavanju instrukcije RTS sa vrha steka se skida 0125 i upisuje u programski brojač PC. Stanje na steku po povratku na potprogram I je označeno sa stek stanje IV.

Tokom izvršavanja potprograma I dolazi se na instrukciju RTS sa adrese 0129 kojom treba da se realizuju povratak na glavni program i to na instrukciju sa adrese 0047. Stanje na steku je označeno sa stek stanje IV. Pri izvršavanju instrukcije RTS sa vrha steka se skida 0047 i

upisuje u programski brojač PC. Stanje na steku po povratku na glavni program je označeno sa stek stanje V.

Jedna od situacija kada se stek koristi kao aritmetički stek prikazana je na slici 6. Uzeto je da stek raste prema višim lokacijama i da registar SP ukazuje na zadnju zauzetu lokaciju.



Slika 7 Aritmetički stek

Tokom izvršavanja glavnog programa dolazi se na instrukciju PUSH 0316 sa adrese 0053 kojom treba da se na vrh steka stavi sadržaj 0010 memorijske lokacije sa adrese 0316. Stanje na steku je označeno sa stek stanje I. Pri izvršavanju instrukcije PUSH 0316 najpre se sadržaj registra SP inkrementira sa vrednosti 0150 na vrednost 0151, zatim se sadržaj 0010 memorijske lokacije sa adrese 0316 čita i na kraju se u memorijsku lokaciju na adresi 0151 određenoj sadržajem registra SP upisuje sadržaj 0010. Po izvršavanju instrukcije PUSH 0316 stanje na steku je označeno sa stek stanje II.

Sledeća instrukcija glavnog programa je instrukcija PUSH 0317 sa adrese 0054 kojom treba da se na vrh steka stavi sadržaj 0020 memorijske lokacije sa adrese 0317. Stanje na steku je označeno sa stek stanje II. Pri izvršavanju instrukcije PUSH 0317 najpre se sadržaj registra SP inkrementira sa vrednosti 0151 na vrednost 0152, zatim se sadržaj 0020 memorijske lokacije sa adrese 0317 čita i na kraju se u memorijsku lokaciju na adresi 0152 određenoj sadržajem registra SP upisuje sadržaj 0020. Po izvršavanju instrukcije PUSH 0317 stanje na steku je označeno sa stek stanje III.

Potom dolazi instrukcija glavnog programa ADD sa adrese 0055 kojom treba sa vrha steka da se skine najpre sadržaj 0020, pa sadržaj 0010, izvrši njihovi sabiranje i na vrh steka stavi

sadržaj 0030. Stanje na steku je označeno sa stek stanje III. Pri izvršavanju instrukcije ADD najpre se sadržaj 0020 čita sa adrese 0152 određene sadržajem registra SP i sadržaj registra SP dekrementira sa vrednosti 0152 na vrednost 0151, zatim se sadržaj 0010 čita sa adrese 0151 određene sadržajem registra SP i sadržaj registra SP dekrementira sa vrednosti 0151 na vrednost 0150, potom se sabiranjem sadržaja 0020 i 0010 dobija suma 0030 i na kraju se sadržaj registra SP inkrementira sa vrednosti 0150 na vrednost 0151 koja predstavlja adresu memorijske lokaciju u koju se upisuje dobijena suma 0030. Po izvršavanju instrukcije ADD stanje na steku je označeno sa stek stanje IV.

Na kraju glavnog programa je instrukcija POP 0318 sa adrese 0056 kojom treba sa vrha steka da se skine sadržaj 0030 i upiše u memorijsku lokaciju sa adrese 0318. Stanje na steku je označeno sa stek stanje IV. Pri izvršavanju instrukcije POP 0318 najpre se sadržaj 0030 čita sa adrese 0151 memorijske lokacije određene sadržajem registra SP, sadržaj registra SP dekrementira sa vrednosti 0151 na vrednost 0150 i očitani sadržaj upisuje u memorijsku lokaciju na adresi 0318. Po izvršavanju instrukcije POP 0318 stanje na steku je označeno sa stek stanje V.

Ukazivač na okvir steka FP (frame pointer) ukazuje na bazu okvira (frame-a) koji se formira na steku kod poziva potprograma. Koristi se za pristup

- parametrima koji se predaju kod poziva potprograma i
- lokalnim promenljivim veličinama kod jezika sa dinamičkim dodeljivanjem prostora kod ulaska u potprogram i oslobađanjem prostora kod napuštanja potprograma.

Primer 1.

PROGRAM PRIMER;

begin

integer par1, par2;

...

procedure PRIM (x,y);

begin

integer a, b, c;

...

begin

PUSH BP
MOVE SP, BP
PUSH R1
PUSH R2
PUSHPSW
SUB SP, 3

...

end;

...
ADD SP, 3
POPSPW
POP R2
POP R1
POP BP
RTS

end;

...

begin

...

CALL PRIM (par1,par2);

PUSH par1
PUSH par2
JSR PRIM
ADD SP, 2

...

end;

end;

Primer 2.

```
PROGRAM PRIMER;
begin
  integer par1, par2;
  ...
  procedure TEST (z);
  begin
    integer e, f;
    ...
    begin
      PUSH BP
      MOVE SP, BP
      PUSHPSW
      SUB SP, 2
      ...
    end;
    ...
  end;
  ...
  procedure PRIM (x,y);
  begin
    integer a, b, c;
    ...
    begin
      PUSH BP
      MOVE SP, BP
      PUSH R1
      PUSH R2
      PUSHPSW
      SUB SP, 3
      ...
      CALL TEST(c);
      ...
      PUSH c
      JSR TEST
      ADD SP,1
      ...
    end;
    ...
  end;
  ...
  ...
  begin
    ...
    CALL PRIM (par1, par2);
    ...
    PUSH par1
    PUSH par2
    JSR PRIM
    ADD SP, 2
    ...
  end;
end;
```

Primer 3.

PROGRAM PRIMER;

begin

integer par1, par2;

...

procedure PRIM (x,y);

begin

integer a, b, c;

...

begin

PUSH BP
MOVE SP, BP
PUSH R1
PUSH R2
PUSHPSW
SUB SP, 3

...

CALL PRIM (a,b);

...
PUSH a
PUSH b
JSR PRIM
ADD SP, 2

...

end;

...
ADD SP, 3
POPSPW
POP R2
POP R1
POP BP
RTS

end;

...

begin

...

CALL PRIM (par1,par2);

PUSH par1
PUSH par2
JSR PRIM
ADD SP, 2

...

end;

end;

1.2 TIPOVI PODATAKA

U ovom poglavlju se, najpre, daju neke opšte napomene vezane za različite načine predstavljanja podataka u računaru, a zatim i mogući načini predstavljanja podataka.

1.2.1 Opšte napomene

Podatke je moguće predstaviti na više različitih načina, kao, na primer, celobrojna veličina bez znaka, celobrojna veličina sa znakom, koja dalje može da se predstavi kao znak i veličina, prvi komplement i drugi komplement, zatim pokretni zarez, alfanumerički niz, numerički niz decimalnih brojeva itd. Pored toga za svaki način predstavljanja moguće je da podaci budu predstavljeni na različitim dužinama. Kao primer se može uzeti celobrojna veličina bez znaka koja se u savremenim računarima veoma često predstavlja na dužinama 8, 16, 32 i 64 bita, čime se postižu različiti opsezi predstavljanja celobrojnih veličina bez znaka i to od 0 do 2^8-1 , od 0 do $2^{16}-1$, od 0 do $2^{32}-1$ i od 0 do $2^{64}-1$.

Kao posledica predstavljanja podataka na više različitih načina jedna ista kombinacija 0 i 1 predstavljaće različite vrednosti podataka. Kao primer se može uzeti binarna reč 10000011 koju je moguće interpretirati na više načina. To je

+131, ako se interpretira kao celobrojna vrednost bez znaka,

-3, ako se interpretira kao celobrojna vrednost sa znakom u načinu predstavljanja znak i veličina,

-125, ako se interpretira kao celobrojna veličina sa znakom u načinu predstavljanja prvi komplement,

-124, ako se interpretira kao celobrojna veličina sa znakom u načinu predstavljanja drugi komplement i

+83, ako se interpretira kao numerički niz binarno kodiranih decimalnih brojeva bez znaka.

Podatak čija je dužina veća od dužine adresibilne memorijske lokacije smešta se u niz susednih memorijskih lokacija. Pri tome se kao adresa takvog podatka zadaje adresa samo jedne i to najniže memorijske lokacije. Kao primer se može uzeti podatak dužine 4 bajta za koji se kao adresa zadaje 55, pri čemu je širina memorijske reči 1 bajt. Ovaj podatak se smešta u memorijske lokacije na adresama 55, 56, 57 i 58, a kao njegova adresa zadaje se 55. Stvar je realizacije procesora da u slučaju podatka dužine 4 bajta za koji se kao adresa zadaje 55 očita 4 bajta sa adresa 55, 56, 57 i 58.

Podaci dužine veće od dužine adresibilne memorijske lokacije mogu da se u memoriji smeštaju na dva načina. Kod prvog načina, na najnižoj adresi je najstariji bajt podatka, a na višim adresam redom mlađi bajtovi podatka. Ovakav način smeštanja se naziva big-endian. Kod drugog načina, na najnižoj adresi je najmlađi bajt podatka, a na višim adresam redom stariji bajtovi podatka. Ovakav način smeštanja se naziva little-endian.

1.2.2 Predstavljanje podataka

Tipovi podataka predstavljaju različite načine predstavljanja podataka binarnim rečima. Najčešće korišćeni tipovi podataka su celobrojne veličine, veličine u pokretnom zrezu, alfanumerički niz i numerički niz.

1.2.2.1 CELOBROJNE VELIČINE

Celobrojne veličine mogu da budu bez znaka i sa znakom.

Ako se binarna reč dužine n bitova, u kojoj su bitovi označeni sa $a_{n-1}a_{n-2}...a_1a_0$, interpretira kao celobrojna veličina bez znaka, onda ona predstavlja podatak A čija se vrednost izračunava pomoću izraza

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

Uz takav način interpretiranja bitova binarne reči, predstavljaju se celobrojne veličine bez znaka u opsegu 0 do 2^n-1 .

Međutim, ako se ista binarna reč intrpretira kao celobrojna veličina sa znakom u drugom komplementu, onda ona predstavlja podatak A čija se vrednost izračunava pomoću izraza

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Uz ovakav način interpretiranja bitova binarne reči, predstavljaju se celobrojne vrednosti sa znakom u opsegu -2^{n-1} do $2^{n-1}-1$.

Binarne reči koje predstavljaju celobrojne veličine se mogu interpretirati i na druge načine kao, na primer, celobrojne veličine sa znakom predstavljene kao znak i veličina, celobrojne veličine sa znakom u prvom komplementu itd.

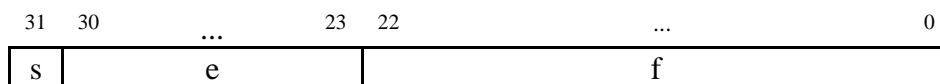
Celobrojne veličine mogu da budu fiksne i promenljive dužine.

Celobrojne veličine fiksne dužine se u računarima predstavljaju na fiksnim dužinama od 8, 16, 32 i 64 bita i njihova dužina i način intrpretacije su određeni poljem koda operacije instrukcije.

Celobrojne veličine promenljive dužine se predstavljaju na različitim dužinama u nekom opsegu dužina koji je definisan za određeni računar i koji odgovara najvećoj dužini celobrojne veličina fiksne dužine datog računara. Ako je ta dužina 64 bita, onda i celobrojne veličine promenljive dužine mogu da budu u opsegu dužina do 64 bit. Celobrojne veličine promenljive dužine se predstavljaju pomoću tri operanda koji definišu početnu adresu memorijske lokacije (A), poziciju najmlađeg bita celobrojne veličine (P) i dužinu (S).

1.2.2.2 VELIČINE U POKRETNOM ZAREZU

Veličine u pokretnom zarezu imaju polje znaka (s), polje eksponenta (e) i mantise (f). Po standardu dužine veličina u pokretnom zarezu su 32 i 64 bita. U slučaju veličine u pokretnom zarezu dužine 32 bita, ta veličina je predstavljena kao na slici 8.



Slika 8 Veličina u pokretnom zarezu

Vrednost v se dobija na sledeći način:

1. za $e=255$ i $f \neq 0$, v je Not a Number bez obzira na s,
2. za $e=255$ i $f=0$, $v=(-1)^s \infty$,
3. za $0 < e < 255$, $v=(-1)^s 2^{e-127}$ (1.f),
4. za $e=0$ i $f \neq 0$, $v=(-1)^s 2^{e-126}$ (0.f) – denormalizovani broj,
5. za $e=0$ i $f=0$, $v=(-1)^s 0$ (nula).

1.2.2.3 ALFANUMERIČKI NIZ

Alfanumerički niz (string) je niz karaktera kodiranih sa 8 bitova, koji se zadaje sa dva operanda i to: A – adresa prvog bajta niza i L – dužina niza u bajtovima.

1.2.2.4 NUMERIČKI NIZ

Numerički niz (decimalni broj) je celobrojna veličina bez znaka ili sa znakom predstavljena kao niz binarno kodiranih decimalnih cifara, koji se zadaje sa dva operanda i to: A – adresa prvog bajta niza i L – dužina niza u bajtovima. U zavisnosti od toga da li se za predstavljanje binarno kodiranih decimalnih cifara i znaka koristi osam bitova ili četiri bita, razlikuju se dva osnovna formata numeričkih nizova i to: nepakovani format i pakovani format, respektivno.

Nepakovani format

Nepakovani format numeričkih nizova koristi 8 bitova za kodiranje decimalnih cifara i znaka. Nepakovani format se pojavljuje u dve varijante u zavisnosti od toga kako se predstavlja znak i to:

- *trailing format* kod koga se znak numeričkog niza utvrđuje na osnovu znaka najmlađe cifre koja je data zadnjim bajtom niza i
- *leading separate format* kod koga se znak numeričkog niza utvrđuje na osnovu posebnog bajta koji je prvi bajt niza.

U *trailing format*-u se posebno kodiraju pozitivne cifre od 0 do 9 i negativne cifre od -0 do -9 (slika). Četiri najstarija bita svake cifre predstavljaju znak, a četiri najmlađa bita veličinu decimalne binarno kodirane cifre. Sve pozitivne cifre imaju četiri najstarija bita sa vrednošću 3, a negativne sa vrednošću 7.

Na slikama je prikazano kako se predstavljaju trocifreni decimalni brojevi 123 i -123. Oba broja zauzimaju po tri bajta, počev od adrese A. Cifre stotica i desetica kao starije cifre predstavljene su pozitivnim ciframa 1 i 2, respektivno, a cifre jedinica, kao najmlađe cifre, predstavljene su pozitivnim i negativnim ciframa 3, respektivno. Na osnovu znaka cifara 3 određeni su znaci brojeva 123 i -123.

<u>cifra</u>	<u>kod</u>
+0	30
+1	31
+2	32
...	...
+8	38
+9	39
-0	70
-1	71
-2	72
...	...
-8	78
-9	79

Slika Kodiranje cifara u *trailing format*-u

<u>+123</u>		<u>adresa</u>
3	1	A
3	2	A+1
3	3	A+2

<u>-123</u>		<u>adresa</u>
3	1	A
3	2	A+1
7	3	A+2

Slika Predstavljanje brojeva 123 i -123 u *trailing format-u*

U *leading separate format-u* se posebno kodiraju cifre od 0 do 9, a posebno znak + i - (slika).

Na slikama je prikazano kako se predstavljaju trocifreni decimalni brojevi 123 i -123. Oba broja zauzimaju po četiri bajta, počev od adrese A. Na adresi A je bajt koji predstavlja znak, a na preostale tri adrese bajtovi koji odgovaraju ciframa stotica, desetica i jedinica. U ovom formatu brojevi 123 i -123 se razlikuju jedino u prvom bajtu.

znak	kod
+	2B
-	2D

cifra	kod
0	30
1	31
2	32
...	...
8	38
9	39

Slika Kodiranje znakova i cifara u *leading separate format-u*

+123		adresa	-123		adresa
2	B	A	2	D	A
3	1	A+1	3	1	A+1
3	2	A+2	3	2	A+2
3	3	A+3	3	3	A+3

Slika Predstavljanje brojeva 123 i -123 u *leading separate format-u*

Pakovani format

Pakovani format numeričkih nizova koristi 4 bita za kodiranje decimalnih cifara od 0 do 9 i znakova + i - (slika). Cifre od 0 do 9 su predstavljene binarnim vrednostima od 0 do 9, znak + binarnim vrednostima A, C, E ili F, i znak - binarnim vrednostima B ili D. Za znak + se najčešće koristi binarna vrednost C, a za znak - binarna vrednost D.

Na slikama je prikazano kako se predstavljaju decimalni brojevi 123 i -12. Oba broja zauzimaju po dva bajta, počev od adrese A. Znak broja 123 određen je sa C, a znak broja -12 sa D.

cifra ili znak	kod
0	0
1	1
2	2
...	...
8	8
9	9
+	A, C, E ili F
-	B ili D

Slika Kodiranje cifara u pakovanom formatu

+123		adresa	-12		adresa
1	2	A	0	1	A
3	C	A+1	2	D	A+1

1.3 FORMATI INSTRUKCIJA

Formatom instrukcije se specificiraju dve vrste informacija neophodne za izvršavanje instrukcija programa i to:

- operacija i tip podatka sa kojim operacija treba da se realizuje i
- izvorišni i odredišni operandi.

Za izvršavanje instrukcija programa treba da se zna i odakle uzeti sledeću instrukcije po završetku izvršavanja tekuće instrukcije. U svim daljim razmatranjima pretpostaviće se danas uobičajeni pristup da je to određeno tekućom vrednošću programskog brojača PC. Kod ovog pristupa kod čitanja instrukcije kao adresa memorijske lokacije uvek se koristi tekuća vrednost programskog brojača PC, koja se tom prilikom inkrementira. Kao rezultat ovakvog pristupa instrukcije se mogu jedino sekvencijalno čitati i izvršavati.

Međutim, u programima kojima se rešavaju određeni problemi veoma često postoji potreba da se na osnovu rezultata računanja realizuju grananja u programu. To ima za posledicu da u nekim situacijam ne treba preći na čitanje i izvršavanje prve sledeće instrukcije u sekvenci, već na neku instrukciju van sekvence, što je moguće postići jedino ako se u programski brojač upiše adresa memorijske lokacije sa koje treba produžiti sa čitanjem i izvršavanjem instrukcija programa. To se postiže posebnim instrukcijama skokova, koje treba da se ubace u delove programa kada treba odstupiti od sekvencijalnog izvršavanja programa. Jedini efekat instrukcija skokova je upisivanje nove vrednosti u programski brojač PC.

Ubacivanje instrukcija skokova na mestima na kojima treba realizovati grananja u programu povećava ukupan broj instrukcija u programima, pri čemu instrukcije skokova ne učestvuju u računanjima problema koji se rešava, već samo obezbeđuju korektan redosled izvršavanja instrukcija programa. Statistika izvršavanja instrukcija pokazuje da u proseku posle 5 do 7 instrukcija izvršenih u sekvenci treba odstupiti od sekvencijalnog izvršavanja programa. S obzirom da u svakoj takvoj situaciji treba ubaciti instrukciju skoka, u proseku se ukupan broj instrukcija u programu povećava za oko 15%.

Bilo je pokušaja i sa pristupom kod koga je u formatu instrukcije, pored informacija o operaciji i tipu podatka sa kojim operacija treba da se realizuje i izvorišnim i odredišnim operandima, bila i informacija o sledećoj instrukciji. Kod tog pristupa adresa sledeće instrukcije koju treba čitati i izvršavati nije određena tekućom vrednošću programskog brojača, već vrednošću dela instrukcije sa informacijom o sledećoj instrukciji. To zahteva da u slučaju kada treba produžiti sa sekvencijalnim izvršavanjem instrukcija informacija o sledećoj instrukciji bude adresa prve sledeće instrukcije u sekvenci, dok u situaciji kada treba odstupiti od sekvencijalnog izvršavanja instrukcija informacija o sledećoj instrukciji bude adresa memorijske lokacije na kojoj se nalazi prva instrukcija programa na koji treba skočiti. U tom slučaju nema potrebe za instrukcijama skoka.

Međutim, zbog izražene sekvencijalnosti u izvršavanju instrukcija programa i činjenice da u proseku tek posle 5 do 7 instrukcija izvršenih u sekvenci postoji potreba za odstupanjem od sekvencijalnog izvršavanja instrukcija, instrukcije koje se izvršavaju sekvencijalno imale bi kao informaciju o sledećoj instrukciji adresu instrukcije koja je sledeća u sekvenci, dok bi samo instrukcija posle koje se odstupi sa sekvencijalnim izvršavanjem instrukcija imala kao informaciju o sledećoj instrukciji adresu instrukcije koja nije sledeća u sekvenci već je prva instrukcija programa na koji treba skočiti. Ukoliko se uzmu sadašnji kapaciteti operativne memorije koji su nekoliko giga reči, tada je potrebno tridesetak bitova za specifikaciju adrese

jedne memorijske lokacije. To bi kod ovog pristupa zahtevalo da svaka instrukcija zbog informacije o sledećoj instrukciji bude duža tridesetak bitova. Zbog toga je danas uobičajen pristup da u formatu instrukcije ne postoji informacija o sledećoj instrukciji već da se tokom sekvencijalnog izvršavanja instrukcija implicitno kao adresa sledeće instrukcije koristi tekuća vrednost programskog brojača koja se inkrementira posle svakog čitanja instrukcije, a da se kada treba odstupiti od sekvencijalnog izvršavanja instrukcija eksplicitno instrukcijom skoka vrši upis adrese sledeće instrukcije u programski brojač PC. Interesantno je da se u mikroprogramima kod mikroprogramskih realizacija upravljačkih jedinica skokovi u mikroprogramu javljaju u proseku posle svake druge ili treće mikroinstrukcije, pa je stoga dosta uobičajeno da u formatu mikroinstrukcije pored dela kojim se specificira koje mikrooperacije treba da se realizuju postoji i deo sa adresom mikroinstrukcije na koju treba preći u slučaju da se odstupa od sekvencijalnog izvršavanja mikroinstrukcija.

Informacije i operaciji i tipu podatka sa kojim operacija treba da se realizuje i izvorišnim i odredišnim operandima se specificiraju odgovarajućim poljima instrukcije. U zavisnosti od toga kako se ove dve vrste informacija specificiraju, zavisi kakve je struktura polja u formatu instrukcije. Na osnovu toga se govori o različitim formatim instrukcija. U daljim razmatranjima se daju funkcije i struktura najpre polja sa specifikacijom operanda i tipa operacije, a zatim i polja sa specifikacijom izvorišnih i odredišnih operanada.

1.3.1 OPERACIJA I TIP PODATKA

Ovim poljem, koje se obično naziva polje koda operacije, se specificira operacija koju treba izvršiti i tip podatka nad kojim datu operaciju treba izvršiti. Pod operacijom se misli na operacije iz skupa instrukcija, kao što su operacije prenosa, aritmetičke operacije, logičke operacije, operacije pomeranja i rotiranja i upravljačke operacije. Pod tipom podatka se misli iz koliko susednih memorijskih lokacija treba pročitati binarne i na koji, od više mogućih načina, ih treba interpretirati.

Stoga je neophodno procesor tako realizovati da se na osnovu vrednosti koda operacije utvrđuje

- iz koliko susednih memorijskih lokacija, počev od one koja je specificirana poljem sa specifikacijom izvorišnih i odredišnih operanada, treba očitati memorijskih reči da bi se dobio operand potrebne dužine,
- kako očitane binarne vrednosti treba interpretirati i
- koju operaciju treba realizovati.

Kao ilustracija ovog pristupa može se uzeti da u procesoru koji od tipova podatak podržava

- celobrojne veličine bez znaka dužine 8, 16, 32 i 64 bita,
- celobrojne veličine sa znakom predstavljene u drugom komplementu dužine 8, 16, 32 i 64 bita i

- veličine u pokretnom zarezu dužine 32 i 64 bita
- mora da postoji 10 kodova operacija za operaciju množenja.

1.3.2 IZVORIŠNI I ODREDIŠNI OPERANDI

Ovim poljem se eksplicitno specificiraju operandi. Postoje više varijanti ovog polja koje nastaju kao posledica sledeća dva elementa:

- broj eksplicitno specificiranih operanada i
- moguće lokacije operanada.

1.3.2.1 Broj eksplicitno specificiranih operanada

U slučaju binarnih operacija, kao što su aritmetičke i logičke operacije, potrebne su dve izvorišne i jedna odredišna lokacija. U zavisnosti od toga koliko se lokacija eksplicitno

definiše u formatu instrukcije, procesori se dele na troadresne, dvoadresne, jednoadresne i nulaadresne procesore. Kod procesora kod koji se u formatu instrukcije eksplicitno ne definišu sve tri lokacije, za one lokacije kojih nema eksplicitno definisanih u formatu instrukcije zna se implicitno gde su.

Kod procesora sa troadresnim formatom instrukcija (slika 9) eksplicitno su poljima A1, A2 i A3 definisane sve tri lokacije. Kod nekih procesora polja A1 i A2 definišu adrese izvorišnih lokacija, a A3 adresu odredišne lokacije. Međutim, ima procesora kod kojih polja A2 i A3 definišu adrese izvorišnih lokacija, a A1 adresu odredišne lokacije.

OC	A1	A2	A3
----	----	----	----

Slika 9 Troadresni format instrukcije

Ukoliko u procesoru sa troadresnim formatom instrukcije treba sračunati izraz

$$C \leq A + B$$

kojim se sabiraju sadržaji memorijskih lokacija čije su adrese simbolički označene sa A i B i rezultat smešta u memorijsku lokaciju čija je adresa simbolički označena sa C, tada se to može realizovati instrukcijom

ADD A, B, C

u kojoj je simbolički sa ADD označena vrednost polja koda operacije sabiranja, a sa A, B i C adrese dve izvorišne i jedne odredišne lokacije. Tokom izvršavanja ove instrukcije iz memorijskih lokacija sa adresa A i B čitaju se operandi, izvršava operacija sabiranja i rezultat smešta u memorijsku lokaciju na adresi C.

Dobra strana troadresnog formata je da se jednom instrukcijom realizuje sračunavanje izraza. Loša strana je velika dužina instrukcije. Ukoliko se uzmu sadašnji kapaciteti operativne memorije koji su nekoliko giga reči, tada je potrebno tridesetak bitova za specifikaciju adrese jedne memorijske lokacije. Ukoliko se uzme da je za polje operacije potrebno oko osam bitova, dobija se da je dužina instrukcije oko 100 bitova. Uz pretpostavku da je širina reči memorijske lokacije jedan bajt, za smeštanje jedne instrukcije u operativnu memorije potrebno je barem 12 bajtova. To stvara dva problema. Prvi je da je za smeštanje programa sa tako dugim instrukcijama potreban dosta veliki memorijski prostor. Drugi je da se tokom izvršavanja programa dosta vremena gubi na veliki broj, u datom primeru to je 12, obraćanja operativnoj memoriji radi čitanja instrukcije, što usporava izvršavanje instrukcija a time i programa.

Kod procesora sa dvoadresnim formatom instrukcija (slika 10) eksplicitno su poljima A1 i A2 definisane dve izvorišne lokacije. Odredišna lokacija se ne definiše eksplicitno u formatu instrukcije, već se jedna od izvorišnih lokacija A1 ili A2 koristi i kao odredišna lokacija. Koja se od izvorišnih lokacija A1 ili A2 koristi i kao odredišna lokacija, definiše se posebno za svaki procesor. Ima procesora kod kojih i izvorišna lokacija A1 i izvorišna lokacija A2 mogu da budu odredišne lokacije. U tom slučaju za istu operaciju mora da postoje dva različita koda operacije i to jedan koji određuje da je A1 odredišna lokacija i drugi koji određuje da je A2 odredišna lokacija.

OC	A1	A2
----	----	----

Slika 10 Dvoadresni format instrukcije

Ukoliko kod procesora sa dvoadresnim formatom instrukcija treba sračunati izraz

$$C \leq A + B$$

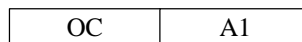
kojim se sabiraju sadržaji memorijskih lokacija čije su adrese simbolički označene sa A i B i rezultat smešta u memorijsku lokaciju čija je adresa simbolički označena sa C, tada se to mora realizovati instrukcijama

```
MOV C, A
ADD C, B
```

u kojima su simbolički sa MOV i ADD označene vrednosti polja koda operacija prenosa i sabiranja, respektivno, a sa A, B i C adrese eksplicitno definisanih izvorišnih i odredišnih lokacija. Tokom izvršavanja instrukcije prenosa MOV iz memorijske lokacija sa adrese A čita se operand i smešta u memorijsku lokaciju na adresi C. Tokom izvršavanja instrukcije sabiranja ADD iz memorijskih lokacija sa adresa C i B čitaju se operandi, izvršava operacija sabiranja i rezultat smešta u memorijsku lokaciju na adresi C.

Dobra strana dvoadresnog formata u odnosu na troadresni format je da je instrukcija kraća. Međutim, sada su potrebne dve instrukcije da se realizuje sračunavanje istog izraza. Zbog toga što je za instrukciju ADD memorijska lokacija C i izvorišna i odredišna lokacija, potrebno je da se najpre instrukcijom MOV prebaci sadržaj memorijske lokacije A u memorijsku lokaciju C i da se zatim instrukcijom ADD iz memorijskih lokacija sa adresa C i B čitaju operandi, izvrši operacija sabiranja i rezultat smesti u memorijsku lokaciju na adresi C. Sadržaji u memorijskim lokacijama A, B i C su identični pre i posle sračunavanja izraza i za procesor sa dvoadresnim i za procesor s troadresnim formatom instrukcija, a razlika je u načinu kako se u memorijskoj lokaciji C formira suma sadržaja memorijskih lokacija A i B. Međutim, i pored skraćivanja instrukcije za dužinu specifikacije jednog operanda, još uvek je velika dužina instrukcije, pa, mada nešto blaži, ostaju problemi veličine memorijskog prostora potrebnog za smeštanje programa i vremena potrebnog za čitanje instrukcija.

Kod procesora sa jednoadresnim formatom instrukcija (slika 11) eksplicitno se poljem A1 definiše jedna izvorišna lokacija. Kod ovih procesora postoji poseban registar procesora koji se obično naziva akumulator koji predstavlja jedno implicitno izvorište i implicitno odredište.



Slika 11 Jednoadresni format instrukcije

Ukoliko kod procesora sa jednoadresnim formatom instrukcija treba sračunati izraz $C \leftarrow A + B$

kojim se sabiraju sadržaji memorijskih lokacija čije su adrese simbolički označene sa A i B i rezultat smešta u memorijsku lokaciju čija je adresa simbolički označena sa C, tada se to mora realizovati instrukcijama

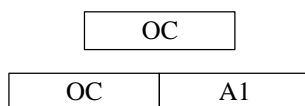
```
LOAD A
ADD B
STORE C
```

u kojima su simbolički sa LOAD, ADD i STORE označene vrednosti polja koda operacija prenosa u akumulator, sabiranja i prenosa iz akumulatora, respektivno, a sa A, B i C adrese eksplicitno definisanih izvorišnih i odredišnih lokacija. Najpre se tokom izvršavanja instrukcije prenosa u akumulator LOAD iz memorijske lokacije sa adrese A čita se operand i smešta u akumulator, zatim se tokom izvršavanja instrukcije sabiranja ADD iz akumulatora i memorijske lokacije sa adrese B čitaju operandi i izvršava operacija sabiranja i na kraju se rezultat tokom izvršavanja instrukcije prenosa iz akumulatora STORE u memorijsku lokaciju sa adrese C čita operand iz akumulatora i smešta u memorijsku lokaciju na adresi C.

Dobra strana jednoadresnog formata u odnosu na dvoadresni format je da je instrukcija kraća. Međutim, sada su potrebne tri instrukcije da se realizuje sračunavanje istog izraza.

Zbog toga što je za instrukciju ADD akumulator i izvorišna i odredišna lokacija, potrebno je da se najpre instrukcijom LOAD prebaci sadržaj memorijske lokacije A u akumulator i da se zatim instrukcijom ADD iz akumulator i memorijske lokacije sa adrese B čitaju operandi, izvrši operacija sabiranja i rezultat smesti u akumulator i da se instrukcijom STORE prebaci rezultat iz akumulatora u memorijsku lokaciju C. Sadržaji u memorijskim lokacijama A, B i C su identični pre i posle sračunavanja izraza za procesore sa jednoadresnim, dvoadresnim i troadresnim formatom instrukcija, a razlika je u načinu kako se u memorijskoj lokaciji C formira suma sadržaja memorijskih lokacija A i B. Međutim, i pored skraćivanja instrukcije za dužinu specifikacije dva operanda, ima onih koji smatraju da je još uvek je velika dužina instrukcije, pa, mada još nešto blaži, ostaju problemi veličine memorijskog prostora potrebnog za smeštanje programa i vremena potrebnog za čitanje instrukcija.

Kod procesora sa nulaadresnim formatom instrukcija (slika 12) u formatu instrukcije postoji samo polje koda operacije i ne postoji ni jedno polje kojim bi se eksplicitno definisale izvorišne i odredišne lokacije. Kod ovih procesora vrh steka je implicitno izvorište za oba operanda i implicitno odredište za rezultat. Izuzetak su dve jednoadresne instrukcije i to instrukcija PUSH kojom se sadržaj iz neke memorijske lokacije smešta na vrh steka i instrukcija POP kojom se sadržaj sa vrha steka smešta u neku memorijsku lokaciju.



Slika 12 Nulaadresni format instrukcije

Ukoliko kod procesora sa nulaadresnim formatom instrukcija treba sračunati izraz

$$C \leftarrow A + B$$

kojim se sabiraju sadržaji memorijskih lokacija čije su adrese simbolički označene sa A i B i rezultat smešta u memorijsku lokaciju čija je adresa simbolički označena sa C, tada se to mora realizovati instrukcijama

```
PUSH A
PUSH B
ADD
POP C
```

u kojima su simbolički sa PUSH, ADD i POP označene vrednosti polja koda operacija prenosa na vrh steka, sabiranja i prenosa sa vrha steka, respektivno, a sa A, B i C adrese eksplicitno definisanih izvorišnih i odredišnih lokacija. Najpre se tokom izvršavanja instrukcije PUSH iz memorijske lokacije sa adrese A čita operand i smešta na vrh steka, potom se tokom izvršavanja instrukcije PUSH iz memorijske lokacije sa adrese B čita operand i smešta na vrh steka, zatim se tokom izvršavanja instrukcije sabiranja ADD sa vrha steka čitaju dva operanda, izvršava operacija sabiranja i dobijeni rezultat smešta na vrh steka i na kraju se tokom izvršavanja instrukcije POP skida rezultat sa vrha steka i smešta u memorijsku lokaciju na adresi C.

Dobra strana nulaadresnog formata u odnosu na jednoadresni format je da je instrukcija kraća. Međutim, sada su potrebne četiri instrukcije da se realizuje sračunavanje istog izraza. Zbog toga što je za instrukciju ADD vrh stek izvorišna lokacija za oba operanda i odredišna lokacija za rezultat, potrebno je da se najpre sa dve instrukcije PUSH prebace sadržaji memorijskih lokacija A i B na vrh steka, da se zatim instrukcijom ADD sa steka čitaju oba operanda, izvrši operacija sabiranja i rezultat smesti na vrh steka i da se instrukcijom POP prebaci rezultat sa vrha steka u memorijsku lokaciju C. Sadržaji u memorijskim lokacijama A, B i C su identični pre i posle sračunavanja izraza za procesore sa nulaadresnim,

jednoadresnim, dvoadresnim i troadresnim formatom instrukcija, a razlika je u načinu kako se u memorijskoj lokaciji C formira suma sadržaja memorijskih lokacija A i B.

Nulaadresni format je pogodan i za generisanje koda prilikom prevođenaja programa napisanih u nekom visokom programskom jeziku. Prevodioci visokih programskih jezika prevode konstrukcije jezika prvo na međukod koji je nezavisan od procesora, a zatim na osnovu međukoda generišu kod za svaki konkretan procesor. Međukod se obično daje u obliku inverzne poljske notacije. Kod koji se od međukoda generiše za procesore sa troadresnim, dvoadresnim i jednoadresnim formatima instrukcija nije efikasan i primenom optimizatora koda moguće ga je poboljšati i do tri puta. Međutim, u slučaju procesora sa nulaadresnim formatom instrukcija generisanje koda je jednostavno i uvek se dobija najefikasniji mogući kod.

U skupu instrukcija pored instrukcija kojima se realizuju binarne operacije, kao što su aritmetičke i logičke operacije, postoje i instrukcije kojima se realizuju operacije pomeranja, kao i instrukcije prenosa i upravljačke instrukcije. Razmatranim troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija mogu da se predstave instrukcije kojima se realizuju operacije pomeranja i instrukcije prenosa. Međutim, upravljačke instrukcije predstavljaju poseban slučaj u odnosu na razmatrane formate instrukcija i realizuju se na isti način za sve razmatrane formate instrukcija.

1.3.2.2 Moguće lokacije operanada

Moguće lokacije operanada su memorijske lokacije, registri procesora i neposredne veličine u instrukciji. Ima više varijanti realizacije ovog polja instrukcije, a motivi kod njegovog definisanja su da:

- da ovaj deo bude kraći da bi se manje memorije zauzimalo za programe i brže očitavale instrukcije (adresa memorije, neposredna veličina ili adresa registra),
- da se brže dolazi do operanada (memorije, neposredna veličina ili registar), i
- da se pruži podrška za moguću primenu nekih tehnika realizacije procesora (pipeline).

Na osnovu mogućih lokacija operanada razlikuju se tri vrste arhitekture:

- memorija – memorija,
- memorija – registar i
- registar – registar.

U slučaju arhitekture memorija – memorija, svi operandi sa kojima se radi, i to i izvorišni i odredišni, su isključivo u memoriji. Postoje i troadresni i dvoadresni i jednoadresni i nulaadresni formati ove arhitekture.

U slučaju arhitekture memorija – registar, jedan izvorišni operand je uvek registar, drugi izvorišni operand je memorija, a odredište ili registar ili memorija. Postoji obično dvoadresni format ove arhitekture.

U slučaju arhitekture registar – registar, svi operandi sa kojima se radi, i to i izvorišni i odredišni, su isključivo u registrima, a memorijskim lokacijama se isključivo pristupa instrukcijama load i store. Postoje obično troadresni format ove arhitekture.

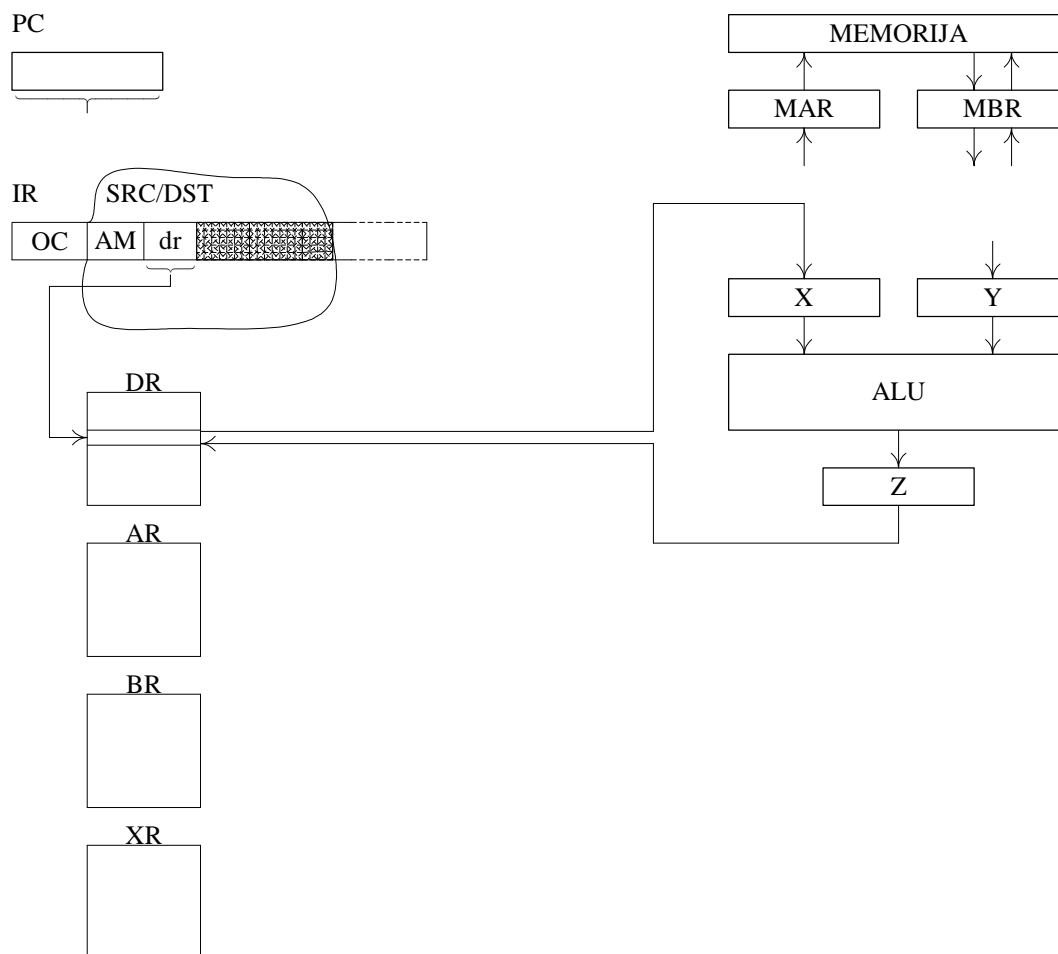
Ovo je neka osnovna podela, pri čemu i u odnosu na nju ima varijanti. Najdrastičniji primer su procesori koji su u osnovi memoria – memorija, ali koji dozvoljavaju da se kroz adresiranje specificira za svaki od operanada ne samo memorija, već i registar i neposredna veličina. Time se pokrivaju sve varijante.

1.4 NAČINI ADRESIRANJA

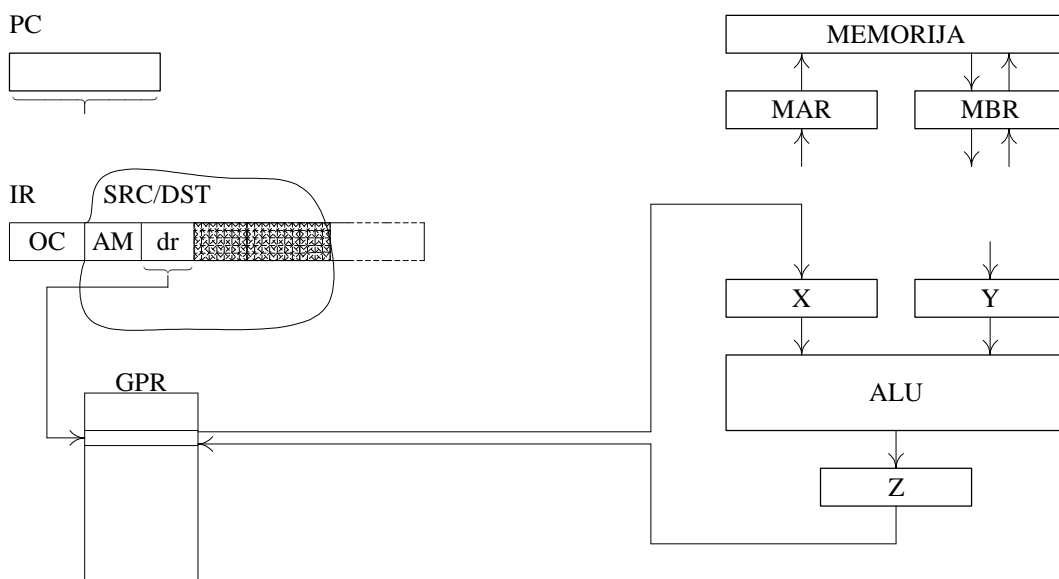
Načini adresiranja određuju da li je operand sadržaj neke memorijske lokacije, nekog od registara podataka ili registara opšte namene procesora ili neposredna veličina u samoj instrukciji. Načini adresiranja specifiiciraju i kako treba formirati adresu memorijske lokacije ukoliko je operand sadržaj neke memorijske lokacije. Najčešći načini adresiranja su registarsko direktno adresiranje, registarsko indirektno adresiranje, memorijsko direktno adresiranje, memorijsko indirektno adresiranje, bazno adresiranje sa pomerajem, indeksno adresiranje sa pomerajem, registarsko indirektno adresiranje sa pomerajem, bazno-indeksno adresiranje sa pomerajem, postdekrement adresiranje, preinkrement adresiranje, relativno adresiranje sa pomerajem i neposredno adresiranje.

1.4.1 Registarsko direktno adresiranje

Registarsko direktno adresiranje je adresiranje kod koga se operand nalazi u jednom od registara podataka (slika 13) ili registara opšte namene procesora (slika 14). Kod onih procesora kod kojih postoje posebno registri podataka (DR), adresni registri (AR), bazni registri (BR) i indeksni registri (XR), operand se nalazi u jednom od registara podataka (DR), dok kod onih procesora kod kojih postoje samo registri opšte namene (GPR), operand se nalazi u jednom od registara opšte namene. U oba slučaja u adresnom delu instrukcije u kome se specificira operand, pored grupe bitova kojima se zadaje način adresiranja (AM), postoji i grupa bitova kojima se zadaje adresa registra podatka ili registra opšte namene (dr).



Slika 13 Registarsko direktno adresiranje



Slika 14 Registarsko direktno adresiranje

Registarsko direktno adresiranje može da se koristi i za izvorišni operand (SRC) i za odredišni operand (DST). Ukoliko se registarsko direktno adresiranje koristi za izvorišni operand, tada se sa zadate adrese (dr) operand čita iz registra podataka ili registra opšte namene i upisuje u neki od prihvatnih registara podataka procesora, a ukoliko se registarsko direktno adresiranje koristi za odredišni operand, tada se na zadatoj adresi u registar podataka (DR) ili registar opšte namene (GPR) upisuje podatak iz nekog od prihvatnih registara podataka procesora. Na slikama je uzeto da se operand koji se čita upisuje u prihvatni registar podataka procesora X na ulazu ALU, a da se podatak koji se upisuje uzima iz prihvatnog registra podataka procesora Z na izlazu ALU.

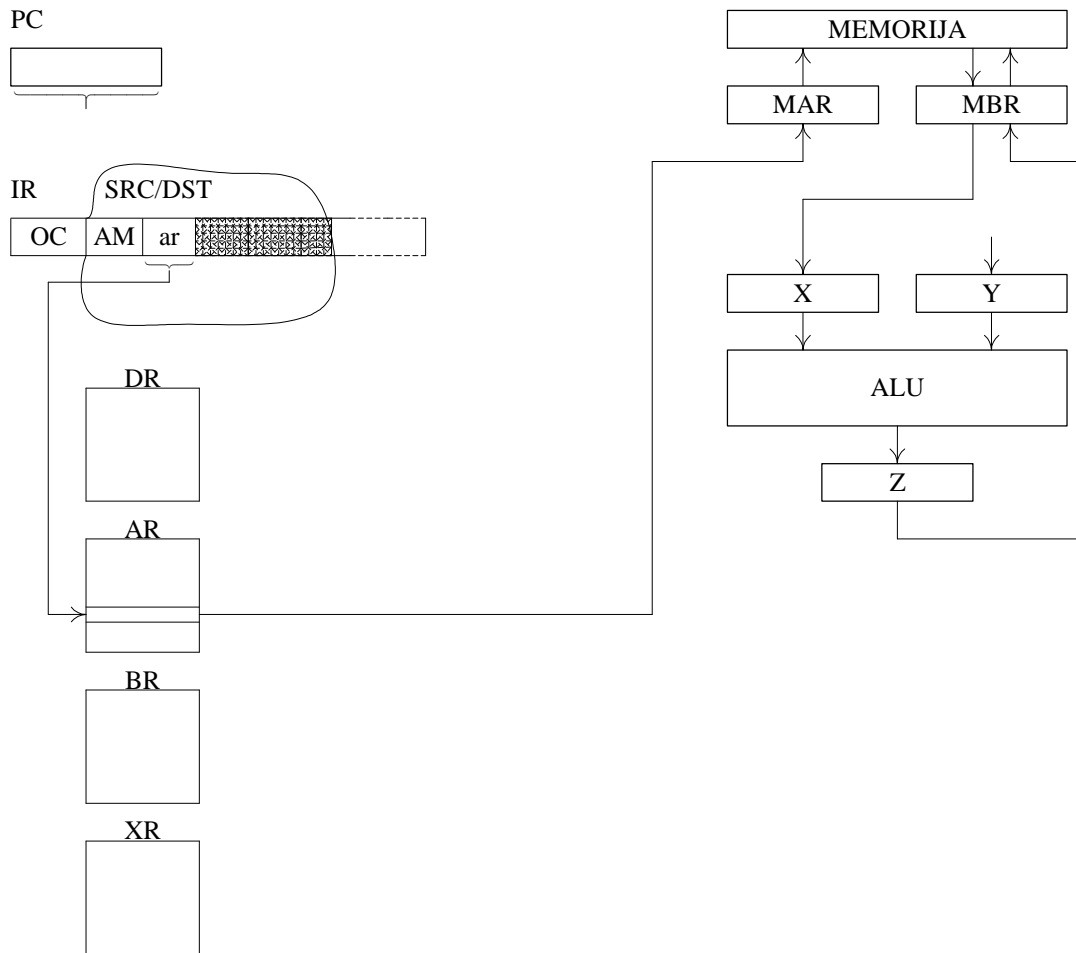
Na slikama je, takođe, uzeto i da zbog nekih drugih načina adresiranja dužina adresnog dela instrukcije veća od one koja je potrebna za registarsko direktno adresiranja, pa se zato neki bitovi adresnog dela instrukcije ne koriste u slučaju registarskog direktnog adresiranja. Ti bitovi su na slikama šrafirani. Kod nekih procesora, zbog toga što je iz određenih razloga važno da dužina instrukcije bude fiksna, adresni delovi instrukcija su fiksni i dimenzionisani prema slučaju kada je broj potrebnih bitova najveći. Tada se, kao što je i dato na slikama neki od bitova ne koriste. U ovom slučaju očitavanje instrukcija je jednostavno, ali instrukcije zauzimaju veći memorijski prostor.

Kod nekih drugih procesora, zbog toga što je iz određenih razloga važno da dužina instrukcija bude što je moguće više kraća, adresni delovi instrukcija su promenljive dužine i sadrže samo onoliko bitova koliko zadati način adresiranja zahteva. Tada obične nema bitova koji se ne koriste. U ovom slučaju instrukcije zauzimaju manji memorijski prostor, ali je očitavanje instrukcija je složeno.

1.4.2 Registarsko indirektno adresiranje

Registarsko indirektno adresiranje je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se nalazi u jednom od adresnih registara (slika 15) ili registara opšte namene procesora (slika 16). Kod onih procesora kod kojih postoje posebno registri podataka (DR), adresni registri (AR), bazni registri (BR) i indeksni registri (XR), adresa memorijske lokacije se nalazi u jednom od adresni registri (AR), dok kod

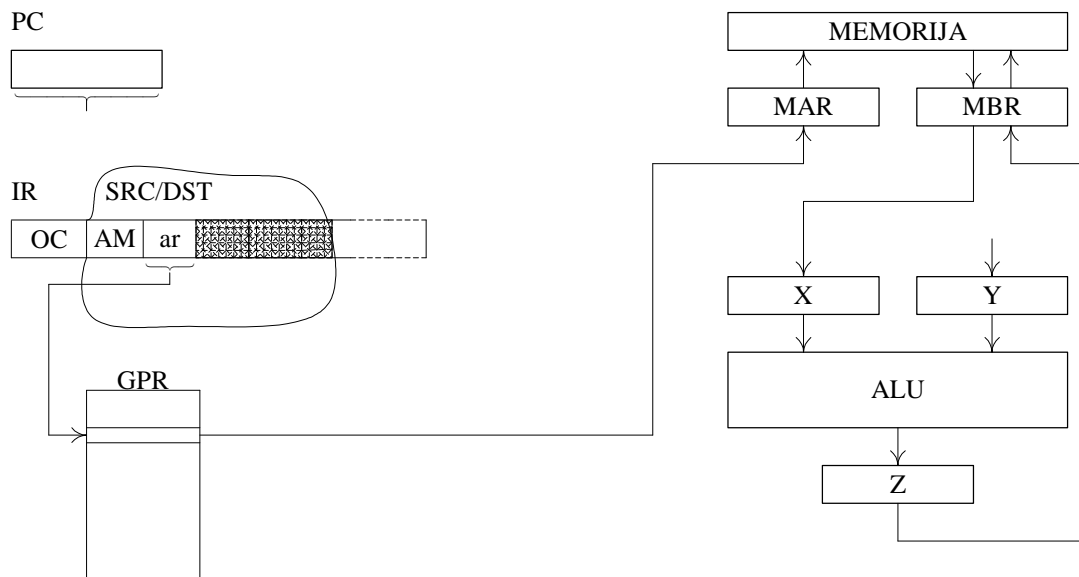
onih procesora kod kojih postoje samo registri opšte namene (GPR), adresa memorijske lokacije se nalazi u jednom od registara opšte namene. U oba slučaja u adresnom delu instrukcije u kome se specificira operand, pored grupe bitova kojima se zadaje način adresiranja (AM), postoji i grupa bitova kojima se zadaje adresa adresnog registra ili registra opšte namene (ar).



Slika 15 Registarsko indirektno adresiranje

Registarsko indirektno adresiranje može da se koristi i za izvorišni operand (SRC) i za odredišni operand (DST). Ukoliko se registarsko indirektno adresiranje koristi za izvorišni operand, tada se sa zadate adrese (ar) čita sadržaj adresnog registra ili registra opšte namene i prebacuje u adresni registar memorije (MAR), iz memorijske lokacije određene sadržajem adresnog registra memorije čita se operand i prihvata u prihvatni registar podataka memorije (MDR) i iz njega upisuje u neki od prihvatnih registara podataka procesora. Ukoliko se registarsko indirektno adresiranje koristi za odredišni operand, tada se sa zadate adrese (ar) čita sadržaj adresnog registra ili registra opšte namene i prebacuje u adresni registar memorije (MAR), iz nekog od prihvatnih registara podataka procesora se podatak upisuje u prihvatni registar podataka memorije (MDR) i iz njega upisuje u memorijsku lokaciju određenu sadržajem adresnog registra memorije. Na slikama je uzeto da se operand koji se čita upisuje u prihvatni registar podataka procesora X na ulazu ALU, a da se podatak koji se upisuje uzima iz prihvatnog registra podataka procesora Z na izlazu ALU.

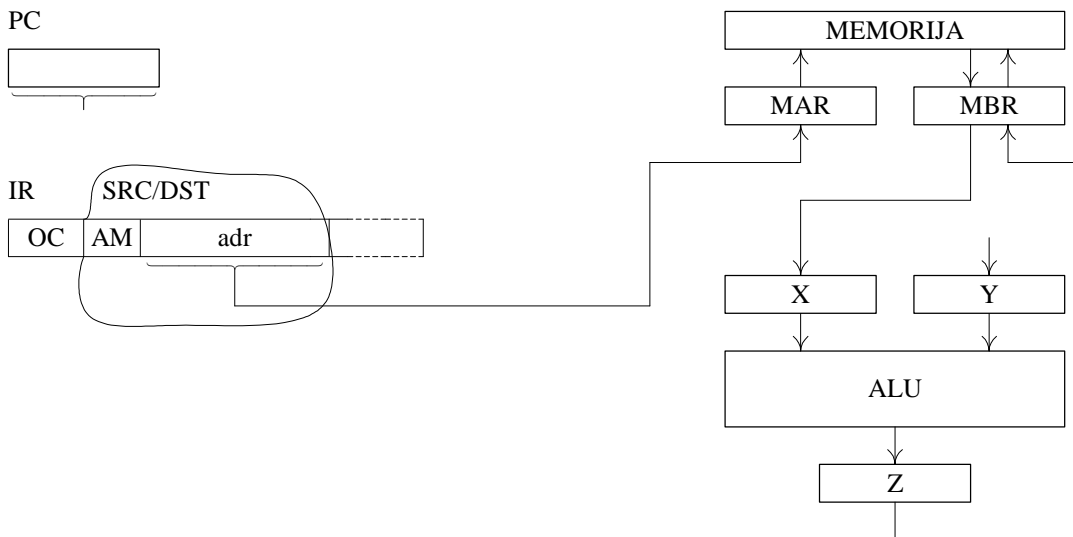
Objašnjenje za šrafiranu grupu bitova u adresnom delu instrukcije je isto kao i za registarsko direktno adresiranje.



Slika 16 Regstarsko indirektno adresiranje

1.4.3 Memorijsko direktno adresiranje

Memorijsko direktno adresiranje je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a u adresnom delu instrukcije u kome se specificira operand, pored grupe bitova kojima se zadaje način adresiranja (AM), postoji i grupa bitova kojima se zadaje adresa memorijske lokacije (adr) (slika 17). Memorijsko direktno adresiranje se javlja i kod onih procesora kod kojih postoje posebno registri podataka (DR), adresni registri (AR), bazni registri (BR) i indeksni registri (XR) i kod onih procesora kod kojih postoje samo registri opšte namene (GPR). U oba slučaja adresa memorijske lokacije se formira na identičan način.



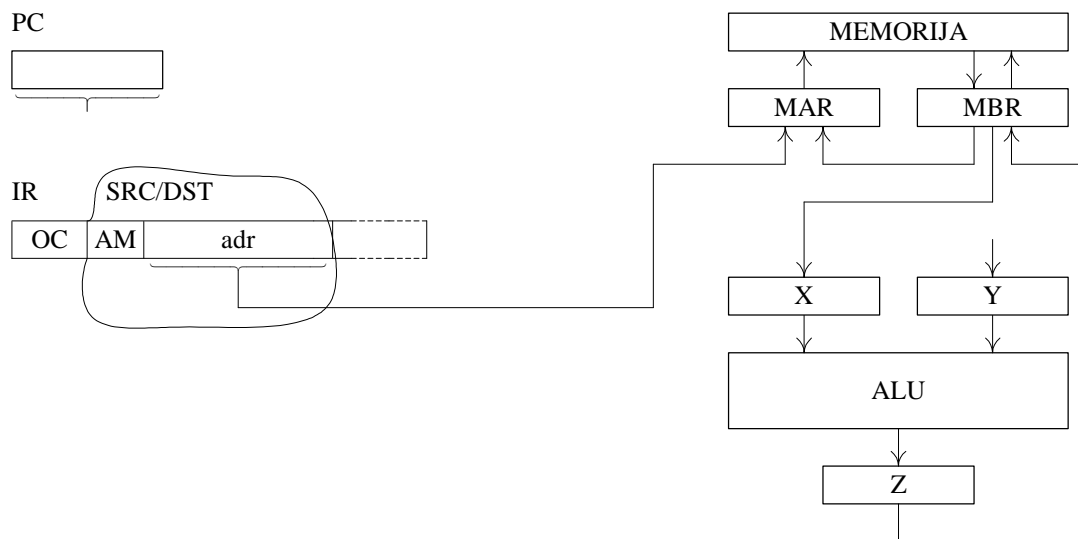
Slika 17 Memorijsko direktno adresiranje

Memorijsko direktno adresiranje može da se koristi i za izvorišni operand (SRC) i za odredišni operand (DST). Ukoliko se memorijsko direktno adresiranje koristi za izvorišni operand, tada se iz adresnog dela instrukcije čita adresa memorijske lokacije (adr) i prebacuje u adresni registar memorije (MAR), iz memorijske lokacije određene sadržajem adresnog registra memorije čita se operand i prihvata u prihvatni registar podataka memorije (MDR) i iz njega upisuje u neki od prihvatnih registara podataka procesora. Ukoliko se memorijsko direktno adresiranje koristi za odredišni operand, tada se iz adresnog dela instrukcije čita

adresa memorijske lokacije (adr) i prebacuje u adresni registar memorije (MAR), iz nekog od prihvatnih registara podataka procesora se podatak upisuje u prihvatni registar podatka memorije (MDR) i iz njega upisuje u memorijsku lokaciju određenu sadržajem adresnog registra memorije. Na slikama je uzeto da se operand koji se čita upisuje u prihvatni registar podatka procesora X na ulazu ALU, a da se podatak koji se upisuje uzima iz prihvatnog registra podatka procesora Z na izlazu ALU.

1.4.4 Memorijsko indirektno adresiranje

Memorijsko indirektno adresiranje je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a u adresnom delu instrukcije u kome se specificira operand, pored grupe bitova kojima se zadaje način adresiranja (AM), postoji i grupa bitova kojima se zadaje adresa memorijske lokacije (adr) na kojoj se nalazi adresa operanda (slika 18). Memorijsko indirektno adresiranje se javlja i kod onih procesora kod kojih postoje posebno registri podataka (DR), adresni registri (AR), bazni registri (BR) i indeksni registri (XR) i kod onih procesora kod kojih postoje samo registri opšte namene (GPR). U oba slučaja adresa memorijske lokacije se formira na identičan način.



Slika 18 Memorijsko indirektno adresiranje

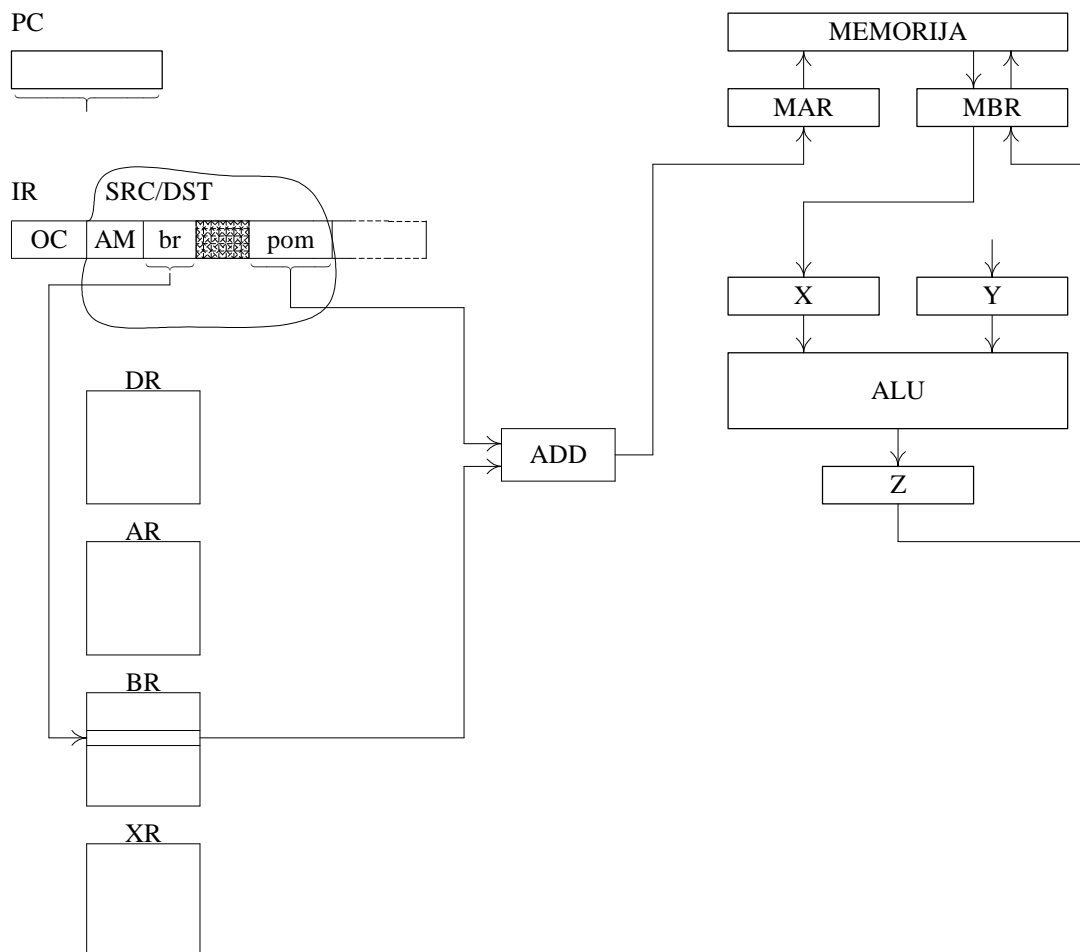
Memorijsko indirektno adresiranje može da se koristi i za izvorišni operand (SRC) i za odredišni operand (DST). Ukoliko se memorijsko indirektno adresiranje koristi za izvorišni operand, tada se iz adresnog dela instrukcije čita adresa memorijske lokacije (adr) i prebacuje u adresni registar memorije (MAR), iz memorijske lokacije određene sadržajem adresnog registra memorije čita se adresa memorijske lokacije operanda i prihvata u prihvatni registar podatka memorije (MDR), iz njega upisuje u adresni registar memorije (MAR), iz memorijske lokacije određene sadržajem adresnog registra memorije čita se operand i prihvata u prihvatni registar podatka memorije (MDR), iz njega upisuje u neki od prihvatnih registara podataka procesora.

Ukoliko se memorijsko direktno adresiranje koristi za odredišni operand, tada se iz adresnog dela instrukcije čita adresa memorijske lokacije (adr) i prebacuje u adresni registar memorije (MAR), iz memorijske lokacije određene sadržajem adresnog registra memorije čita se adresa memorijske lokacije operanda i prihvata u prihvatni registar podatka memorije (MDR) i iz njega upisuje u adresni registar memorije (MAR), iz nekog od prihvatnih registara podataka procesora se podatak upisuje u prihvatni registar podatka memorije (MDR) i iz njega upisuje u memorijsku lokaciju određenu sadržajem adresnog registra memorije. Na slikama je

uzeto da se operand koji se čita upisuje u prihvatni registar podatka procesora X na ulazu ALU, a da se podatak koji se upisuje uzima iz prihvatnog registra podatka procesora Z na izlazu ALU.

1.4.5 Bazno adresiranje sa pomerajem

Bazno adresiranje sa pomerajem je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se dobija sabiranjem sadržaja jednog od baznih registara i pomeraja (slika 19). U adresnom delu instrukcije u kome se specificira operand, pored grupe bitova kojima se zadaje način adresiranja (AM), postoji i grupa bitova kojima se zadaje adresa baznog registra (br) i pomeraj. Bazno adresiranje sa pomerajem se javlja kod onih procesora kod kojih postoje posebno registri podataka (DR), adresni registri (AR), bazni registri (BR) i indeksni registri (XR). Sličan način formiranja adrese memorijske lokacije postoji i kod onih procesora kod kojih postoje samo registri opšte namene (GPR). Kod njih se adresa memorijske lokacije dobija sabiranjem sadržaja jednog od registara opšte namene (GPR) i pomeraja, a adresiranje se obično naziva registarsko indirektno adresiranje sa pomerajem.



Slika 19 Bazno adresiranje sa pomerajem

Bazno adresiranje sa pomerajem može da se koristi i za izvorišni operand (SRC) i za određišni operand (DST). Ukoliko se bazno adresiranje sa pomerajem koristi za izvorišni operand, tada se čita sadržaj baznog registra na osnovu zadate adrese (br) iz adresnog dela instrukcije i pomeraj (pom) iz adresnog dela instrukcije, formira njihova suma i prebacuje u adresni registar memorije (MAR), iz memorijske lokacije određene sadržajem adresnog

registra memorije čita se operand i prihvata u prihvatni registar podatka memorije (MDR) i iz njega upisuje u neki od prihvatnih registara podataka procesora.

Ukoliko se bazno adresiranje sa pomerajem koristi za odredišni operand, tada se čita sadržaj baznog registra na osnovu zadate adrese (br) iz adresnog dela instrukcije i pomeraj (pom) iz adresnog dela instrukcije, formira njihova suma i prebacuje u adresni registar memorije (MAR), iz nekog od prihvatnih registara podataka procesora se podatak upisuje u prihvatni registar podatka memorije (MDR) i iz njega upisuje u memorijsku lokaciju određenu sadržajem adresnog registra memorije. Na slikama je uzeto da se operand koji se čita upisuje u prihvatni registar podatka procesora X na ulazu ALU, a da se podatak koji se upisuje uzima iz prihvatnog registra podatka procesora Z na izlazu ALU.

Objašnjenje za šrafiranu grupu bitova u adresnom delu instrukcije je isto kao i za registarsko direktno adresiranje.

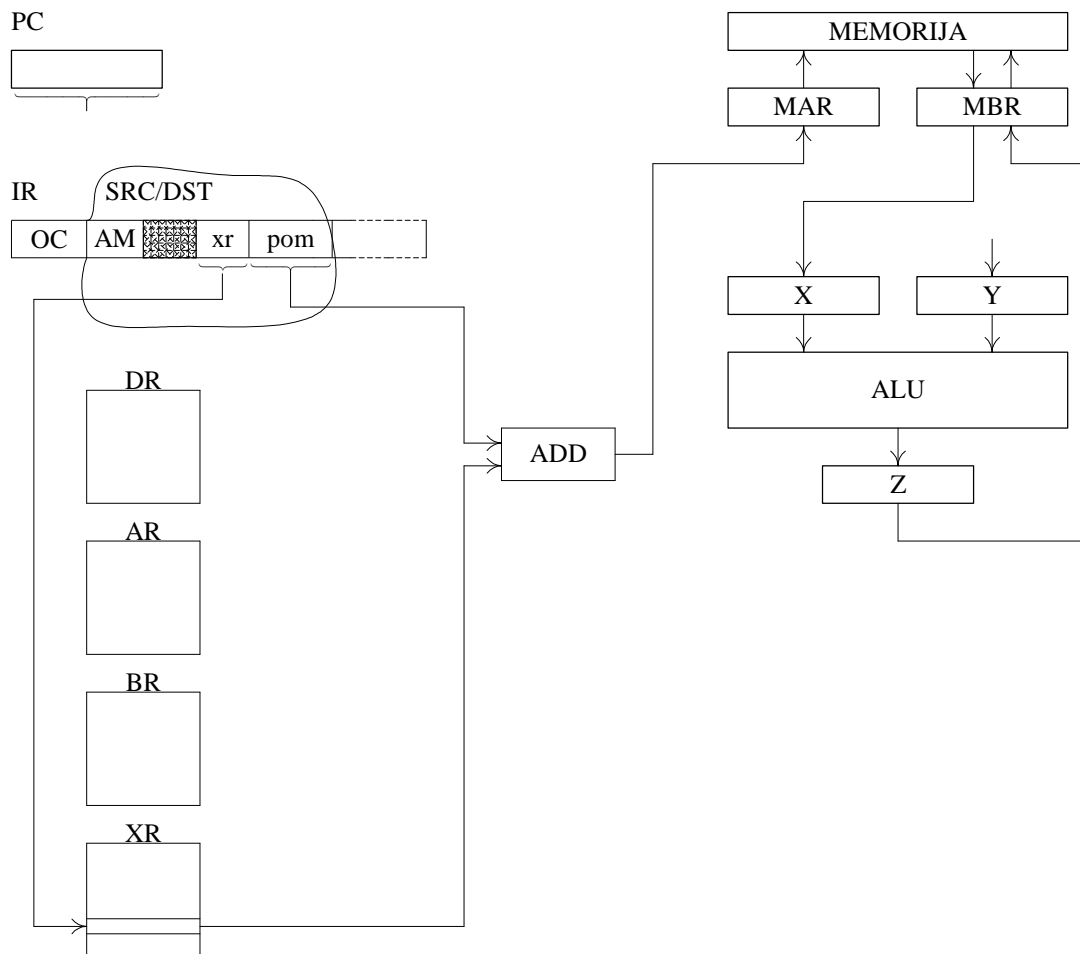
1.4.6 Indeksno adresiranje sa pomerajem

Indeksno adresiranje sa pomerajem je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se dobija sabiranjem sadržaja jednog od indeksnih registara i pomeraja (slika 20). U adresnom delu instrukcije u kome se specificira operand, pored grupe bitova kojima se zadaje način adresiranja (AM), postoji i grupa bitova kojima se zadaje adresa indeksnog registra (xr) i pomeraj. Indeksno adresiranje sa pomerajem se javlja kod onih procesora kod kojih postoje posebno registri podataka (DR), adresni registri (AR), bazni registri (BR) i indeksni registri (XR). Sličan način formiranja adrese memorijske lokacije postoji i kod onih procesora kod kojih postoje samo registri opšte namene (GPR). Kod njih se adresa memorijske lokacije dobija sabiranjem sadržaja jednog od registara opšte namene (GPR) i pomeraja, a adresiranje se obično naziva registarsko indirektno adresiranje sa pomeraje.

Indeksno adresiranje sa pomerajem može da se koristi i za izvorišni operand (SRC) i za odredišni operand (DST). Ukoliko se indeksno adresiranje sa pomerajem koristi za izvorišni operand, tada se čita sadržaj indeksnog registra na osnovu zadate adrese (xr) iz adresnog dela instrukcije i pomeraj (pom) iz adresnog dela instrukcije, formira njihova suma i prebacuje u adresni registar memorije (MAR), iz memorijske lokacije određene sadržajem adresnog registra memorije čita se operand i prihvata u prihvatni registar podatka memorije (MDR) i iz njega upisuje u neki od prihvatnih registara podataka procesora.

Ukoliko se indeksno adresiranje sa pomerajem koristi za odredišni operand, tada se čita sadržaj indeksnog registra na osnovu zadate adrese (br) iz adresnog dela instrukcije i pomeraj (pom) iz adresnog dela instrukcije, formira njihova suma i prebacuje u adresni registar memorije (MAR), iz nekog od prihvatnih registara podataka procesora se podatak upisuje u prihvatni registar podatka memorije (MDR) i iz njega upisuje u memorijsku lokaciju određenu sadržajem adresnog registra memorije. Na slikama je uzeto da se operand koji se čita upisuje u prihvatni registar podatka procesora X na ulazu ALU, a da se podatak koji se upisuje uzima iz prihvatnog registra podatka procesora Z na izlazu ALU.

Objašnjenje za šrafiranu grupu bitova u adresnom delu instrukcije je isto kao i za registarsko direktno adresiranje.

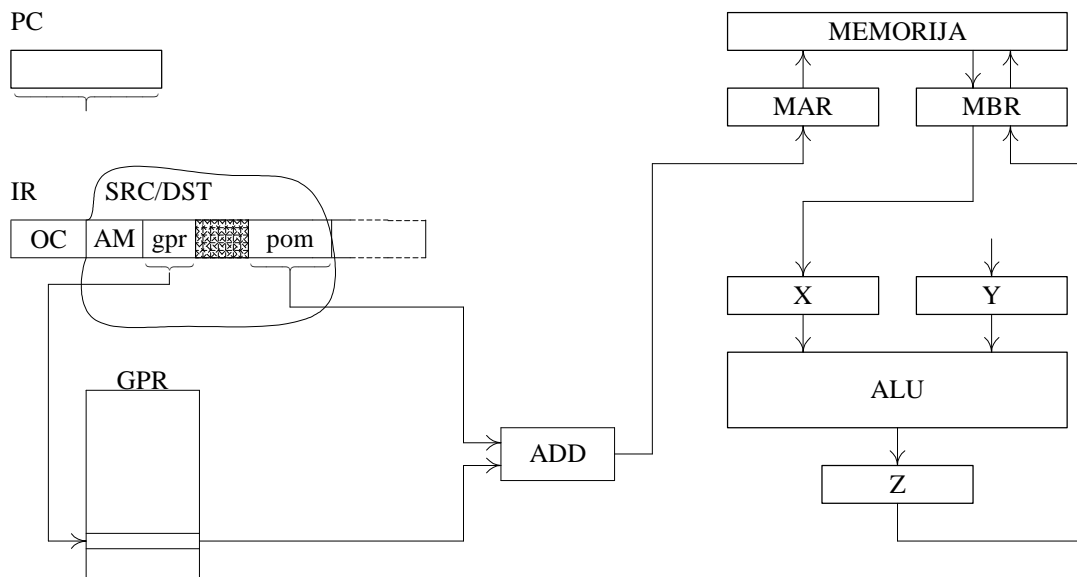


Slika 20 Indeksno adresiranje sa pomerajem

1.4.7 Registarско indirektno adresiranje sa pomerajem

Registarско indirektno adresiranje sa pomerajem je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se dobija sabiranjem sadržaja jednog od registara opšte namene i pomeraja (slika 21). U adresnom delu instrukcije u kome se specificira operand, pored grupe bitova kojima se zadaje način adresiranja (AM), postoji i grupa bitova kojima se zadaje adresa registra opšte namene (gpr) i pomeraj (pom). Registarско indirektno adresiranje sa pomerajem se javlja kod procesora kod kojih postoje registri opšte namene (GPR). Sličan način formiranja adrese memorijske lokacije postoji i kod onih procesora kod kojih postoje posebno registri podataka (DR), adresni registri (AR), bazni registri (BR) i indeksni registri (XR). Kod njih se adresa memorijske lokacije dobija sabiranjem sadržaja jednog od baznih registara (BR) ili indeksnih registara (XR) i pomeraja, a adresiranje se obično naziva bazno adresiranje sa pomerajem ili indeksno adresiranje sa pomerajem, respektivno.

Registarско indirektno adresiranje sa pomerajem može da se koristi i za izvorišni operand (SRC) i za odredišni operand (DST). Ukoliko se registrarско indirektno adresiranje sa pomerajem koristi za izvorišni operand, tada se čita sadržaj registra opšte namene na osnovu zadate adrese (gpr) iz adresnog dela instrukcije i pomeraj (pom) iz adresnog dela instrukcije, formira njihova suma i prebacuje u adresni registar memorije (MAR), iz memorijske lokacije određene sadržajem adresnog registra memorije čita se operand i prihvata u prihvatni registar podataka memorije (MDR) i iz njega upisuje u neki od prihvatnih registara podataka procesora.



Slika 21 Registarsko indirektno adresiranje sa pomerajem

Ukoliko se registarsko indirektno adresiranje sa pomerajem koristi za odredišni operand, tada se čita sadržaj registra opšte namene na osnovu zadate adrese (gpr) iz adresnog dela instrukcije i pomeraj (pom) iz adresnog dela instrukcije, formira njihova suma i prebacuje u adresni registar memorije (MAR), iz nekog od prihvatnih registara podataka procesora se podatak upisuje u prihvatni registar podatka memorije (MDR) i iz njega upisuje u memorijsku lokaciju određenu sadržajem adresnog registra memorije. Na slikama je uzeto da se operand koji se čita upisuje u prihvatni registar podatka procesora X na ulazu ALU, a da se podatak koji se upisuje uzima iz prihvatnog registra podatka procesora Z na izlazu ALU.

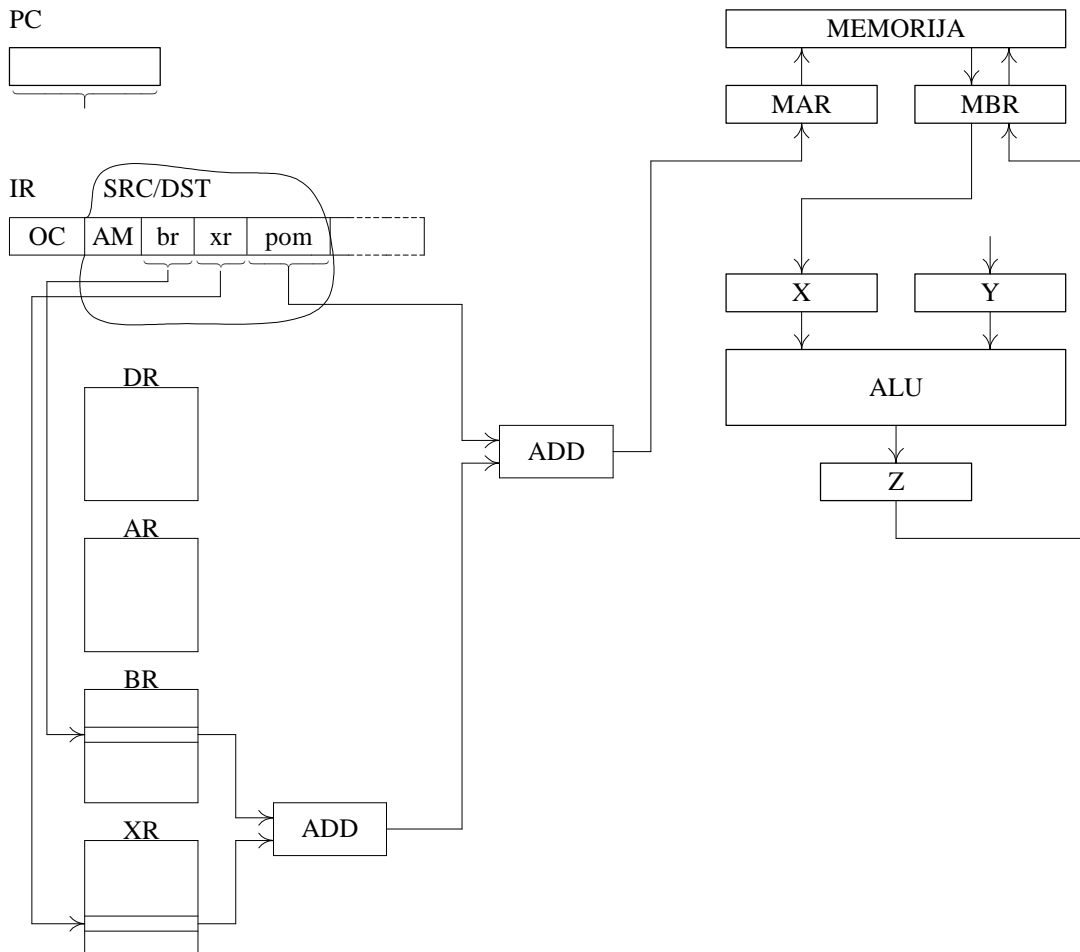
Objašnjenje za šrafiranu grupu bitova u adresnom delu instrukcije je isto kao i za registarsko direktno adresiranje.

1.4.8 Bazno-indeksno adresiranje sa pomerajem

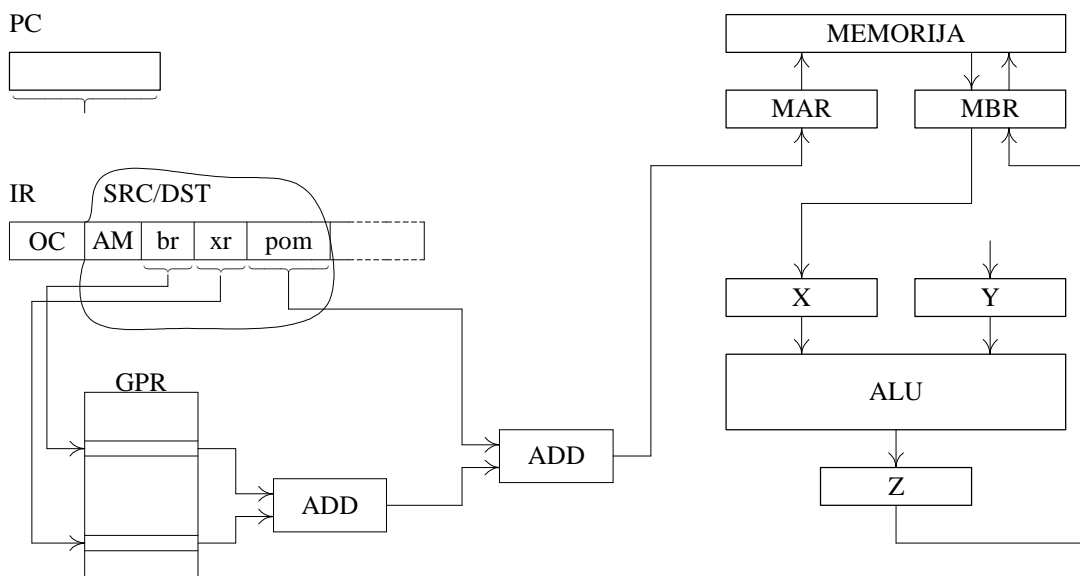
Bazno indeksno adresiranje sa pomerajem je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se dobija sabiranjem sadržaja jednog od baznih registara, jednog od indeksnih registara i pomeraja (slika 22) ili sabiranjem sadržaja dva registra opšte namene procesora i pomeraja (slika 23). Kod onih procesora kod kojih postoje posebno registri podataka (DR), adresni registri (AR), bazni registri (BR) i indeksni registri (XR), u formiranju adrese memorijske lokacije učestvuje jedan od baznih registara (BR) i jedan od indeksnih registara (XR), dok kod onih procesora kod kojih postoje samo registri opšte namene (GPR), u formiranju adrese memorijske lokacije učestvuju dva od registara opšte namene (GPR), pri čemu jedan ima funkciju baznog registra a drugi funkciju indeksnog registra. U oba slučaja u adresnom delu instrukcije u kome se specificira operand, pored grupe bitova kojima se zadaje način adresiranja (AM), postoji i grupa bitova kojima se zadaje adresa baznog registra ili registra opšte namene koji ima funkciju baznog registra (br), adresa indeksnog registra ili registra opšte namene koji ima funkciju indeksnog registra (xr) i pomeraj (pom). Bazno indeksno adresiranje sa pomerajem može da se koristi i za izvorišni operand (SRC) i za odredišni operand (DST).

Ukoliko se bazno indeksno adresiranje sa pomerajem koristi za izvorišni operand, tada se čita sadržaj baznog registra ili registra opšte namene na osnovu zadate adrese (br) iz adresnog dela instrukcije, sadržaj indeksnog registra ili registra opšte namene na osnovu zadate adrese (xr) iz adresnog dela instrukcije i pomeraj (pom) iz adresnog dela instrukcije, formira njihova

suma i prebacuje u adresni registar memorije (MAR), iz memorijske lokacije određene sadržajem adresnog registra memorije čita se operand i prihvata u prihvatni registar podatka memorije (MDR) i iz njega upisuje u neki od prihvatnih registara podataka procesora.



Slika 22 Bazno indeksno adresiranje sa pomerajem



Slika 23 Bazno indeksno adresiranje sa pomerajem

Ukoliko se bazno indeksno adresiranje sa pomerajem koristi za određeni operand, tada se čita sadržaj baznog registra ili registra opšte namene na osnovu zadate adrese (br) iz adresnog dela instrukcije, sadržaj indeksnog registra ili registra opšte namene na osnovu zadate adrese (xr) iz adresnog dela instrukcije i pomeraj (pom) iz adresnog dela instrukcije, formira njihova suma i prebacuje u adresni registar memorije (MAR), iz nekog od prihvatnih registara podataka procesora se podatak upisuje u prihvatni registar podatka memorije (MDR) i iz njega upisuje u memorijsku lokaciju određenu sadržajem adresnog registra memorije. Na slikama je uzeto da se operand koji se čita upisuje u prihvatni registar podatka procesora X na ulazu ALU, a da se podatak koji se upisuje uzima iz prihvatnog registra podatka procesora Z na izlazu ALU.

1.4.9 Registarska indirektna adresiranja sa autoinkrement i autodekrementiranjem

Registarska indirektna adresiranja sa autoinkrementiranjem i autodekrementiranjem su veoma slična registarskom indirektnom adresiranju, jer se i kod ovih adresiranja kao i kod registarskog indirektnog adresiranja operand nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se nalazi u jednom od adresnih registara (slika 15) ili registara opšte namene procesora (slika 16). Sve što važi za registarsko indirektno adresiranje važi i za registarska indirektna adresiranja sa autoinkrementiranjem i autodekrementiranjem. Jedina razlika je u tome da se kod registarskih indirektnih adresiranja sa autoinkrementiranjem i autodekrementiranjem tom prilikom vrši ili inkrementiranje ili dekrementiranje adresnog registra ili registra opšte namene.

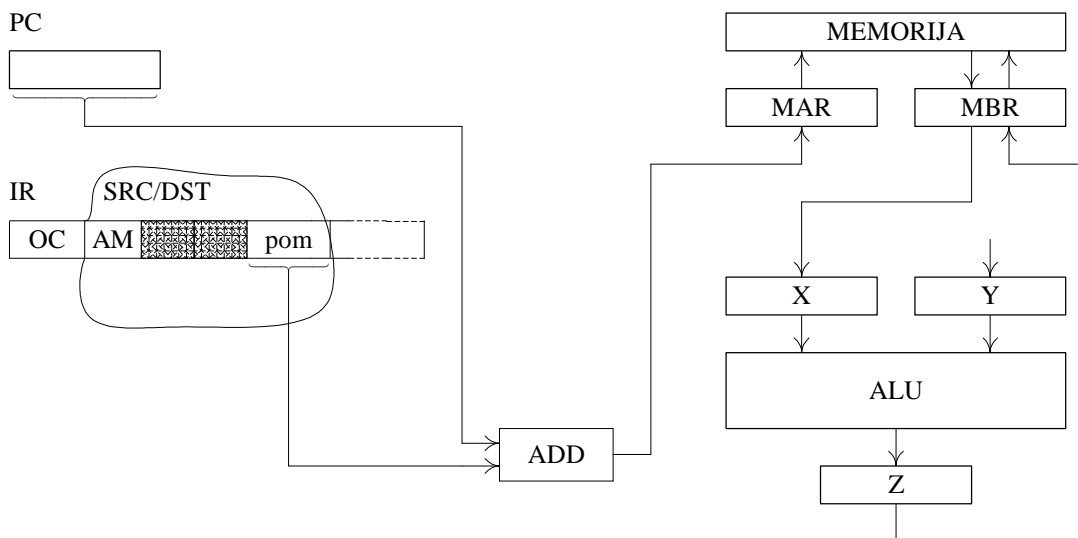
U zavisnosti od toga da li se prvo inkrementira registar pa posle toga njegov sadržaj koristi kao adresa memorijske lokacije ili se prvo sadržaj registra koristi kao adresa memorijske lokacije pa se posle toga inkrementira, registarsko indirektno adresiranje sa autoinkrementiranjem se javlja kao preinkrement adresiranje i postinkrement adresiranje, respektivno. Slučna je situacija i sa registarskim indirektnim adresiranjem sa autodekrementiranjem koje se javlja kao predekrement adresiranje i postdekrement adresiranje u zavisnosti od toga da li se prvo dekrementira registar pa posle toga njegov sadržaj koristi kao adresa memorijske lokacije ili se prvo sadržaj registra koristi kao adresa memorijske lokacije pa se posle toga dekrementira, respektivno. Uobičajene oznake za ova adresiranja su +(R) za preinkrement, (R)+ za postinkrement, -(R) za predekrement i (R)- za postdekrement. Procesori sa ovim adresiranjima se obično tako realizuju da ukoliko postoji -(R) tada postoji i (R)+ adresiranje, a ukoliko postoji +(R) tada postoji i (R)-. Time se omogućava realizacija steka bez korišćenja registra SP.

Kod procesora kod kojih postoji -(R) i (R)+ moguće je realizovati stek koji raste naviše i ukazuje na prvu slobodnu lokaciju, kao i stek koji raste naniže i ukazuje na zadnju zauzetu. Ukoliko stek koji raste naviše i ukazuje na prvu slobodnu lokaciju tada radi upisa na poziciji odredišta treba koristiti (R)+ i radi čitanja na poziciji izvorišta treba koristiti -(R). Ukoliko stek raste naniže i ukazuje na zadnju zauzetu, tada radi upisa na poziciji odredišta treba koristiti -(R) i radi čitanja na poziciji izvorišta treba koristiti (R)+ .

Kod procesora kod kojih postoji +(R) i (R)- moguće je realizovati stek koji raste naviše i ukazuje na zadnju zauzetu lokaciju, kao i stek koji raste naniže i ukazuje na prvu slobodnu lokaciju. Ukoliko stek raste naviše i ukazuje na zadnju zauzetu lokaciju tada radi upisa na poziciji odredišta treba koristiti +(R) i radi čitanja na poziciji izvorišta treba koristiti (R)-. Ukoliko stek raste naniže i ukazuje na prvu slobodnu lokaciju, tada radi upisa na poziciji odredišta treba koristiti (R)- i radi čitanja na poziciji izvorišta treba koristiti +(R).

1.4.10 Relativno adresiranje sa pomerajem

Relativno adresiranje sa pomerajem je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se dobija sabiranjem sadržaja programskog brojača PC i pomeraja (slika 24). U adresnom delu instrukcije u kome se specificira operand, pored grupe bitova kojima se zadaje način adresiranja (AM), postoji i grupa bitova kojima se zadaje pomeraj (pom). Relativno adresiranje sa pomerajem se javlja i kod onih procesora kod kojih postoje posebno registri podataka (DR), adresni registri (AR), bazni registri (BR) i indeksni registri (XR) i kod onih procesora kod kojih postoje samo registri opšte namene (GPR). U oba slučaja adresa memorijske lokacije se formira na identičan način. Relativno adresiranje sa pomerajem može da se koristi i za izvorišni operand (SRC) i za odredišni operand (DST).



Slika 24 Relativno adresiranje sa pomerajem

Ukoliko se relativno adresiranje sa pomerajem koristi za izvorišni operand, tada se sabiraju sadržaj programskog brojača PC i pomeraj (pom) iz adresnog dela instrukcije i njihova suma prebacuje u adresni registar memorije (MAR), iz memorijske lokacije određene sadržajem adresnog registra memorije čita se operand i prihvata u prihvatni registar podatka memorije (MDR) i iz njega upisuje u neki od prihvatnih registara podataka procesora.

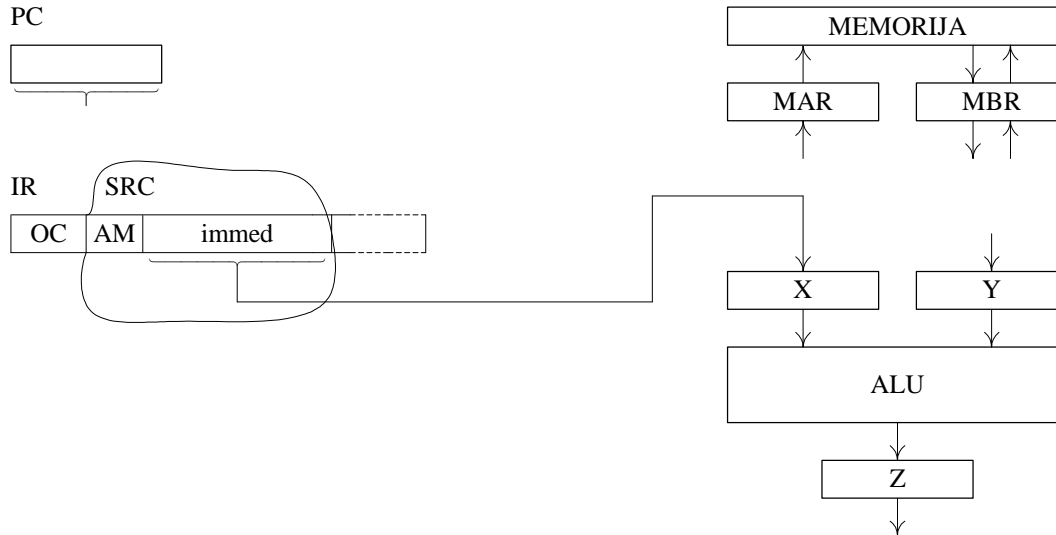
Ukoliko se relativno adresiranje sa pomerajem koristi za odredišni operand, tada se sabiraju sadržaj programskog brojača PC i pomeraj (pom) iz adresnog dela instrukcije i njihova suma prebacuje u adresni registar memorije (MAR), iz nekog od prihvatnih registara podataka procesora se podatak upisuje u prihvatni registar podatka memorije (MDR) i iz njega upisuje u memorijsku lokaciju određenu sadržajem adresnog registra memorije. Na slikama je uzeto da se operand koji se čita upisuje u prihvatni registar podatka procesora X na ulazu ALU, a da se podatak koji se upisuje uzima iz prihvatnog registra podatka procesora Z na izlazu ALU.

Objašnjenje za šrafiranu grupu bitova u adresnom delu instrukcije je isto kao i za registarsko direktno adresiranje.

1.4.11 Neposredno adresiranje

Neposredno adresiranje je adresiranje kod koga se operand nalazi neposredno u samoj instrukciji (slika 25). U adresnom delu instrukcije u kome se specificira operand, pored grupe bitova kojima se zadaje način adresiranja (AM), postoji i grupa bitova kojima se zadaje

operand (immed). Neposredno adresiranje se javlja i kod onih procesora kod kojih postoje posebno registri podataka (DR), adresni registri (AR), bazni registri (BR) i indeksni registri (XR) i kod onih procesora kod kojih postoje samo registri opšte namene (GPR). U oba slučaja do operand se dolazi na identičan način. Neposredno adresiranje može da se koristi samo za izvorišni operand (SRC), dok se za odredišni operand (DST) ne koristi.



Slika 25 Neposredno adresiranje

Ukoliko se neposredno adresiranje koristi za izvorišni operand, tada se operand (immed) iz adresnog dela instrukcije čita i upisuje u neki od prihvatnih registara podataka procesora. Na slici je uzeto da se operand koji se čita upisuje u prihvatni registar podatka procesora X na ulazu ALU,

Neposredno adresiranje se ne koristi za odredišni operand, jer bi trebalo, ukoliko bi se koristilo, da se podatak iz nekog od prihvatnih registara podataka procesora upiše u grupu bitova predviđenu za operand (podatak) iz adresnog dela instrukcije. Praksa je da se radi zaštite od mogućih grešaka dozvoljava da se iz onog dela memorije gde se nalaze instrukcije samo čita a ne i upisuje. Zbog toga se procesori obično tako realizuju da se pokušaj upisa korišćenjem neposrednog adresiranja za odredišni operand detektuje i generiše unutrašnji prekid greške u adresiranju.

1.5 SKUP INSTRUKCIJA

Skup instrukcija specificira operacije koje mogu da se izvršavaju u procesoru. Skup instrukcija čine standardne instrukcije i nestandardne instrukcije.

1.5.1 STANDARDNE INSTRUKCIJE

Standardne instrukcije uključuje operacije čijim kombinovanjem svaki problem koji treba da se reši u računaru može da se predstavi programom. Standardne instrukcije se u nekom vidu nalaze u svakom procesoru. Skup standardnih instrukcija čine instrukcije prenosa, aritmetičke instrukcije, logičke instrukcije, instrukcije pomeranja i rotiranja, instrukcije skoka i mešovite instrukcije.

1.5.1.1 INSTRUKCIJE PRENOSA

Podaci se nalaze u memorijskim lokacijama, registrima procesora i u instrukciji. Zbog toga mora da postoje instrukcije prenosa za prebacivanje podataka između lokacija na kojima mogu da se nađu. Treba napomenuti da podaci mogu da budu i u registrima kontrolera periferija, pri čemu se njima pristupa na dva načina. Kod nekih procesora registri kontrolera

periferija se ne izdvajaju kao posebno mesto gde mogu da budu podaci već se tretiraju na isti način kao i memorijske lokacije, pa ne postoje posebne instrukcije za prenos podatak u i iz registara kontrolera periferija već se registrima kontrolera periferija pristupa istim instrukcijama kojima se pristupa memorijskim lokacijama. Kod ovih procesora se kaže da je ulazno izlazni adresni prostor memorijski preslikan. Kod drugih procesora registri kontrolera periferija se izdvajaju kao posebno mesto gde mogu da budu podaci i postoje posebne instrukcije za prenos podatak u i iz registara kontrolera periferija. Kod ovih procesora se kaže da su ulazno izlazni i memorijski adresni prostori razdvojeni.

Pored toga kod procesora sa jednoadresnim formatom instrukcija akumulator je implicitno izvorište i odredište. Zato postoji potreba za instrukcijom prenosa kojom bi podatak preneo u akumulator i dalje mogao da koristi kao izvorišni operand. Akumulator je i implicitno odredište, pa postoji potreba za instrukcijom prenosa kojom bi se podatak prebacio iz akumulatora u željenu lokaciju. Slična je situacija i kod procesora sa nulaadresnim formatom instrukcija kod kojih je vrh steka implicitno izvorište i odredište. Zato postoji potreba za instrukcijom prenosa kojom bi podatak preneo na vrh steka i dalje mogao da koristi kao izvorišni operand. Vrh steka je i implicitno odredište, pa postoji potreba za instrukcijom prenosa kojom bi se podatak prebacio sa vrha steka u željenu lokaciju.

U daljem tekstu se razmatraju moguće instrukcije prenosa.

MOV a, b

Ova instrukcija se javlja kod procesora sa dvoadresnim formatom instrukcija. Sa MOV je simbolički označeno polje koda operacije, a sa *a* i *b* specifikacije jednog izvorišnog i jednog odredišnog operanda. U daljim razmatranjima će se uzeti da je *a* izvorište, a *b* odredište, mada može da bude i obrnuto. U zavisnosti od specificiranog načina adresiranja *a* može da bude registar procesora, memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i neposredna veličina u instrukciji. Slična je situacija i sa *b*, pre čemu, zbog toga što se radi o odredištu, *b* samo ne može da bude neposredna veličina u instrukciji. Specificiranjem odgovarajućih adresiranja moguće je realizovati prenose iz memorije (registra kontrolera periferije) u memoriju (registar kontrolera periferije), iz memorije (registra kontrolera periferije) u registar procesora, iz registra procesora u memoriju (registar kontrolera periferije), iz registra procesora u registar procesora, neposredno iz instrukcije u memoriju (registar kontrolera periferije) i neposredno iz instrukcije u registar procesora.

Kod procesora sa troadresnim formatom instrukcija sve je isto, samo što se polje za specifikaciju drugog izvorišta ne koristi.

IN regper, regproc

OUT regproc, regper

Ove instrukcije se javljaju kod procesora sa dvoadresnim formatom instrukcija, ukoliko su ulazno izlazni i memorijski adresni prostori razdvojeni. Sa IN je simbolički označeno polje koda operacije za prenos iz registra kontrolera periferije u registar procesora, pri čemu je sa *regper* direktno data adresa registra kontrolera periferije, dok je sa *regproc* direktno data adresa registra procesora. Sa OUT je simbolički označeno polje koda operacije za prenos iz registra procesora u registar kontrolera periferije, pri čemu *regproc* i *regper* imaju identično značenje kao i u slučaju instrukcije IN. Uobičajeno je kod ovih instrukcija da su prenosi samo između registara kontrolera periferija i registara procesora i da se njihove adrese direktno daju.

IN regper

OUT *regper*

Ove instrukcije se javljaju kod procesora sa jednoadresnim formatom instrukcija, ukoliko su ulazno izlazni i memorijski adresni prostori razdvojeni. Oznake IN, OUT i *regper* imaju identično značenje kao i u slučaju procesora sa dvoadresnim formatom instrukcija. Kod instrukcije IN prenos je iz registra kontrolera periferije čija je adresa direktno data sa *regper* i akumulatora koji je implicitno odredište, dok je kod instrukcije OUT prenos iz akumulatora koji je implicitno izvorište i registra kontrolera periferije čija je adresa direktno data sa *regper*.

LOAD *a*

STORE *b*

Ove instrukcija se javlja kod procesora sa jednoadresnim formatom instrukcija. Sa LOAD i STORE je simbolički označeno polje koda operacije, a sa *a* i *b* specifikacije izvorišnog i odredišnog operanda, respektivno. Instrukcijom LOAD se operand sa izvorišta *a* prenosi u akumulator kao implicitno odredište, dok se instrukcijom STORE sadržaj akumulatora kao implicitno izvorište prenosi u odredište *b*. U zavisnosti od specificiranog načina adresiranja *a* može da bude registar procesora, memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i neposredna veličina u instrukciji. Slična je situacije i sa *b*, pre čemu, zbog toga što se radi o odredištu, *b* samo ne može da bude neposredna veličina u instrukciji. U slučaju instrukcije LOAD specificiranjem odgovarajućih adresiranja za izvorište *a* moguće je u akumulator preneti operand iz memorije (registra kontrolera periferije), iz registra procesora i neposredno iz instrukcije. U slučaju instrukcije STORE specificiranjem odgovarajućih adresiranja za odredište *b* moguće je iz akumulator preneti operand u memoriju (registar kontrolera periferije) i registar procesora, dok prenos u instrukciju korišćenjem neposrednog adresiranja nije dozvoljen.

PUSH *a*

POP *b*

Ove instrukcija se javlja kod procesora sa nulaadresnim formatom instrukcija. Sa PUSH i POP je simbolički označeno polje koda operacije, a sa *a* i *b* specifikacije izvorišnog i odredišnog operanda, respektivno. Instrukcijom PUSH se operand sa izvorišta *a* prenosi na stek kao implicitno odredište, dok se instrukcijom POP sadržaj sa steka kao implicitno izvorište prenosi u odredište *b*. U zavisnosti od specificiranog načina adresiranja *a* može da bude registar procesora, memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i neposredna veličina u instrukciji. Slična je situacije i sa *b*, pre čemu, zbog toga što se radi o odredištu, *b* samo ne može da bude neposredna veličina u instrukciji. U slučaju instrukcije PUSH specificiranjem odgovarajućih adresiranja za izvorište *a* moguće je na stek preneti operand iz memorije (registra kontrolera periferije), iz registra procesora i neposredno iz instrukcije. U slučaju instrukcije POP specificiranjem odgovarajućih adresiranja za odredište *b* moguće je sa steka preneti operand u memoriju (registar kontrolera periferije) i registar procesora, dok prenos u instrukciju korišćenjem neposrednog adresiranja nije dozvoljen.

1.5.1.2 ARITMETIČKE INSTRUKCIJE

Aritmetičkim instrukcijama se realizuju standardne aritmetičke operacije sabiranja, oduzimanja, množenja i deljenja.

ADD *a, b, c*

Ovo je format instrukcije sabiranja kod procesora sa troadresnim formatom instrukcija. Sa ADD je simbolički označeno polje koda operacije, a sa *a*, *b* i *c* specifikacije dva izvorišna i jednog odredišnog operanda. U daljim razmatranjima će se uzeti da su *a* i *b* izvorišta, a *c*

odredište, mada se često dešava i da je a odredište, a b i c izvorišta. Tokom izvršavanja ove instrukcije sa lokacija specificiranih sa a i b najpre se čitaju dva izvorišna operanda, zatim se nad njima realizuje operacija sabiranja i na kraju se dobijeni rezultat smešta u lokaciju specificiranu sa c . U zavisnosti od nezavisno specificiranih načina adresiranja za a i b , izvorišni operandi mogu da budu bilo koja kombinacija registara procesora, memorijskih lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registara kontrolera periferije) i neposrednih veličina u instrukciji. Slična je situacija i sa c , pri čemu, zbog toga što se radi o odredištu, c samo ne može da bude neposredna veličina u instrukciji. Neke od mogućih kombinacija su:

- a , b i c su memorijske lokacije (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan a , b i c mogu da budu bilo koja kombinacija stvarnih memorijskih lokacija i registara kontrolera periferije),
- a , b i c su registri procesora,
- a je neposredna veličina, b je registar procesora i c memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije),
- a je memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije), b je neposredna veličina i c registar procesora, itd.

ADD a , b

Ovo je format instrukcije sabiranja kod procesora sa dvoadresnim formatom instrukcija. Sa ADD je simbolički označeno polje koda operacije, a sa a i b specifikacije dva izvorišna operanda, pri čemu je a ili b implicitno odredište. U daljim razmatranjima će se uzeti da je a jedno izvorište i implicitno odredište, a b drugo izvorište, mada se često dešava i da je a prvo izvorište, a b drugo izvorište i implicitno odredište. Tokom izvršavanja ove instrukcije sa lokacija specificiranih sa a i b najpre se čitaju dva izvorišna operanda, zatim se nad njima realizuje operacija sabiranja i na kraju se dobijeni rezultat smešta u lokaciju specificiranu sa a . U zavisnosti od specificiranog načina adresiranja za b , drugi izvorišni operand može da bude registar procesora, memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i neposredna veličina u instrukciji. Slična je situacija i sa a , pri čemu, zbog toga što se radi ne samo i prvom izvorištu nego i o implicitnom odredištu, a samo ne može da bude neposredna veličina u instrukciji. Neke od mogućih kombinacija su:

- a i b su memorijske lokacije (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan a i b mogu da budu bilo koja kombinacija stvarnih memorijskih lokacija i registara kontrolera periferije),
- a i b su registri procesora,
- a je registar procesora i b memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije),
- a je memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i b je neposredna veličina, itd.

ADD a

Ovo je format instrukcije sabiranja kod procesora sa jednoadresnim formatom instrukcija. Sa ADD je simbolički označeno polje koda operacije, a sa a specifikacija drugog izvorišnog operanda, pri čemu je akumulator implicitno izvorište prvog operanda i implicitno odredište rezultat. Tokom izvršavanja ove instrukcije se najpre iz akumulatora implicitno čita prvi izvorišni operand i sa lokacije specificirane sa a se čita drugi izvorišni operand, zatim se nad njima realizuje operacija sabiranja i na kraju se dobijeni rezultat smešta u akumulator kao implicitno odredište. U zavisnosti od specificiranog načina adresiranja za a , drugi izvorišni operand može da bude registar procesora, memorijska lokacija (a kod procesora kod kojih je

ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i neposredna veličina u instrukciji. Neke od mogućih kombinacija su:

- a je memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan a može da bude i registar kontrolera periferije), pa se vrši sabiranje akumulatora i stvarne memorijske lokacije ili registra kontrolera periferije i rezultat smešta u akumulator,

- a je registar procesora, pa se vrši sabiranje akumulatora i registra procesora i rezultat smešta u akumulator i

- a je neposredna veličina, pa se vrši sabiranje akumulatora i neposredne veličine i rezultat smešta u akumulator.

ADD

Ovo je format instrukcije sabiranja kod procesora sa nulaadresnim formatom instrukcija. Sa ADD je simbolički označeno polje koda operacije, pri čemu je vrh steka implicitno izvorište za oba izvorišna operanda i implicitno odredište. Tokom izvršavanja ove instrukcije sa vrha steka se implicitno čita najpre prvi a potom i drugi izvorišni operand, zatim se nad njima realizuje operacija sabiranja i na kraju se dobijeni rezultat smešta na vrh steka kao implicitno odredište.

Slična je situacija i sa formatima instrukcija za operacije oduzimanja, množenja i deljenja. Format instrukcija za ove operacije za procesore sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija su:

SUB a, b, c

SUB a, b

SUB a

SUB

MUL a, b, c

MUL a, b

MUL a

MUL

DIV a, b, c

DIV a, b

DIV a

DIV

Eksplisitna specifikacija izvorišnih i odredišnih operanada poljima a , b i c je ista kao i kod instrukcije sabiranja. Implicitna izvorišta i odredišta u formatima instrukcija gde ih ima, su, takođe, ista kao i kod instrukcije sabiranja.

Podaci nad kojima se izvršavaju operacije sabiranja, oduzimanja, množenja i deljenja mogu da budu predstavljeni na više načina, kao na primer celobrojne veličine bez znaka, celobrojne veličine sa znakom i to obično u drugom komplementu, pokretni zarez, pri čemu to može da se čini na više različitih dužina, kao na primer bajt, dva bajta, četiri bajta itd. Za svaki mogući način predstavljanja podataka i određenu dužinu podatka mora da postoji poseban kod operacije za svaku od operacija sabiranja, oduzimanja, množenja i deljenja. Kao ilustracija može se uzeti da u procesoru, koji od tipova podataka podržava celobrojne veličine bez znaka dužine 8, 16, 32 i 64 bita, celobrojne veličine sa znakom predstavljene u drugom komplementu dužine 8, 16, 32 i 64 bita i veličine u pokretnom zarezu dužine 32 i 64 bita, mora da postoji 10 kodova operacija za operaciju množenja. Isto važi i za operaciju deljenja. Međutim, kod instrukcija sabiranja i oduzimanja nad celobrojnim veličinama bez znaka i celobrojnim vrednostima sa znakom u drugom komplementu to nije neophodno, jer se istim postupkom sabiranja i oduzimanja binarnih reči određene dužine dobija korektan rezultat i

kada se binarne reči interpretiraju kao celobrojne veličine bez znaka i kao celobrojne veličine sa znakom u drugom komplementu.

Kao primer za operaciju sabiranja uzete su binarne reči 1010 i 0010 za koje se uobičajenim postupkom sabiranja binarnih reči dobija binarna reč 1100. Ako se binarne reči interpretiraju kao celobrojne veličine bez znaka, tada je

- binarna reč prvog sabirka 1010 jednaka decimalno +10,
- binarna reč drugog sabirka 0010 jednaka decimalno +2 i
- binarna reč dobijene sume 1100 jednaka decimalno +12,

što jeste korektan rezultat. Međutim, ako se iste binarne reči interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, tada je

- binarna reč prvog sabirka 1010 jednaka decimalno -6,
- binarna reč drugog sabirka 0010 jednako decimalno +2 i
- binarna reč dobijene sume 1100 jednaka decimalno -4,

što, takođe, jeste korektan rezultat.

Kao primer za operaciju oduzimanja uzete su iste binarne reči 1010 i 0010 za koje se uobičajenim postupkom oduzimanja binarnih reči dobija binarna reč 1000. Ako se binarne reči interpretiraju kao celobrojne veličine bez znaka, tada je

- binarna reč prvog sabirka 1010 jednaka decimalno +10,
- binarna reč drugog sabirka 0010 jednaka decimalno +2 i
- binarna reč dobijene sume 1000 jednaka decimalno +8,

što jeste korektan rezultat. Međutim, ako se iste binarne reči interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, tada je

- binarna reč prvog sabirka 1010 jednaka decimalno -6,
- binarna reč drugog sabirka 0010 jednaka decimalno +2 i
- binarna reč dobijene sume 1000 jednaka decimalno -8,

što, takođe, jeste korektan rezultat.

Zbog toga za operaciju sabiranja nad celobrojnim veličinama bez znaka i celobrojnim veličinama sa znakom u drugom komplementu iste dužine ne postoji potreba za dva različita koda operacije. Kao ilustracija se može uzeti ranije pomenuti procesor koji od tipova podataka podržava celobrojne veličine bez znaka dužine 8, 16, 32 i 64 bita, celobrojne veličine sa znakom predstavljene u drugom komplementu dužine 8, 16, 32 i 64 bita i veličine u pokretnom zarezu dužine 32 i 64 bita. U ovom procesoru mora da postoji 6 kodova operacija za operaciju sabiranja i to 4 koda operacije za operaciju sabiranja nad celobrojnim veličinama bez znaka i celobrojnim veličinama sa znakom u drugom komplementu dužine 8, 16, 32 i 64 bita, kao i 2 koda operacije za operaciju sabiranja nad veličinama u pokretnom zarezu dužine 32 i 64 bita. Isto važi i za operaciju oduzimanja.

Među aritmetičkim instrukcijama se pojavljuje i poseban slučaj operacije sabiranja kod koje je implicitno određeno da je vrednost koja se sabira jednaka 1, kao i poseban slučaj operacije oduzimanja kod koje je implicitno određeno da je vrednost koja se oduzima jednaka 1. To su instrukcije inkrementiranja i dekrementiranja.

INC a, b

Ova je format instrukcije inkrementiranja koji se javlja kod procesora sa dvoadresnim formatom instrukcija. Sa INC je simbolički označeno polje koda operacije, a sa a i b specifikacije izvorišnog i odredišnog operanda, respektivno. U daljim razmatranjima će se uzeti da je a izvorište a b odredište, mada se često dešava i da je a odredište a b izvorište. Tokom izvršavanja ove instrukcije sa lokacije specificirane sa a najpre se čita izvorišni operand, zatim se njegova vrednost uveća za 1 i na kraju se dobijeni rezultat smešta u lokaciju

specificiranu sa b . U zavisnosti od specificiranog načina adresiranja za a izvorišni operand može da bude registar procesora, memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registara kontrolera periferije) i neposredna veličina u instrukciji. Slična je situacija i sa b , pri čemu, zbog toga što se radi o odredištu, b samo ne može da bude neposredna veličina u instrukciji. Neke od mogućih kombinacija su:

- a i b su memorijske lokacije (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan a i b mogu da budu bilo koja kombinacija stvarnih memorijskih lokacija i registara kontrolera periferije),
- a i b registri procesora,
- a je neposredna veličina i b je registar procesora,
- a je memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i b je registar procesora, itd.

INC a

Ovo je format instrukcije inkrementiranja koji se javlja kod procesora sa jednoadresnim formatom instrukcija. Sa INC je simbolički označeno polje koda operacije, a sa a specifikacija izvorišnog i implicitnog odredišnog operanda. Tokom izvršavanja ove instrukcije sa lokacije specificirane sa a najpre se čita izvorišni operand, zatim se njegova vrednost uveća za 1 i na kraju se dobijeni rezultat smešta u lokaciju specificiranu sa a . U zavisnosti od specificiranog načina adresiranja za a izvorišni i implicitni odredišni operand može da bude registar procesora i memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije). Zbog toga što je a ne samo izvorište već i implicitno odredište, a ne može da bude neposredna veličina u instrukciji.

INC

Ovo je format instrukcije inkrementiranja kod procesora sa nulaadresnim formatom instrukcija. Sa INC je simbolički označeno polje koda operacije, pri čemu je vrh steka implicitno izvorište operanda i implicitno odredište. Tokom izvršavanja ove instrukcije sa vrha steka se implicitno čita operand, zatim se njegova vrednost uveća za 1 i na kraju se dobijeni rezultat smešta na vrh steka kao implicitno odredište.

Slična je situacija i sa formatima instrukcija za operaciju dekrementiranja. Formatima instrukcija za ovu operaciju za procesore sa dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija su:

DEC a, b

DEC a

DEC

Eksplisitna specifikacija izvorišnih i odredišnih operanada poljima a i b je ista kao i kod instrukcije inkrementiranja. Implicitna izvorišta i odredišta u formatima instrukcija gde ih ima, su, takođe, ista kao i kod instrukcije inkrementiranja.

Podaci nad kojima se izvršavaju operacije inkrementiranja i dekrementiranja mogu da budu predstavljeni kao celobrojne veličine bez znaka i celobrojne veličine sa znakom i to obično u drugom komplementu, pri čemu to može da se čini na više različitih dužina, kao na primer bajt, dva bajta, četiri bajta itd. Međutim, inkrementiranje se za određenu dužinu operanda realizuje na isti način i nad celobrojnim veličinama bez znaka i nad celobrojnim veličinama sa znakom u drugom komplementu, pa nisu potrebna 2 već samo 1 kod operacije. Ovo je objašnjeno za instrukciju sabiranja i važi i za instrukciju inkrementiranja kao poseban slučaj instrukcije sabiranja. Zbog toga različiti kodovi operacije treba da postoje samo za različite dužine operanada. Kao ilustracija ovog pristupa može se uzeti da u procesoru koji od tipova podatak podržava celobrojne veličine bez znaka dužine 8, 16, 32 i 64 bita i celobrojne veličine

sa znakom predstavljene u drugom komplementu dužine 8, 16, 32 i 64 bita mora da postoje 4 operacije za operaciju inkrementiranja. Isto važi i za operaciju dekrementiranja.

Među aritmetičkim instrukcijama se pojavljuje i poseban slučaj operacije oduzimanja kod koje se rezultat oduzimanja nikuda ne upisuje, već se samo vrši provera dobijene vrednosti i na osnovu toga vrši postavljanje indikatora N, Z, C i V u programskoj statusnoj reči PSW procesora. Ova instrukcija se naziva aritmetičko upoređivanje. Format instrukcije upoređivanja za procesore sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija su:

CMP *a, b, c*

CMP *a, b*

CMP *a*

CMP

Sa CMP je simbolički označeno polje koda operacije, a sa *a* i *b* specifikacije dva izvorišna operanda. Sa *c* je označeno polje za specifikacija odredišnog operanda koje postoji kod procesora sa troadresnim formatom instrukcija, mada se u slučaju operacije upoređivanja polje *c* ne koristi. Razlika između instrukcija upoređivanja kod procesora sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija je samo u tome kako se dolazi do dva izvorišna operanda. U slučaju procesora sa troadresnim i dvoadresnim formatima instrukcija izvorišni operandi se dobijaju sa lokacija specificiranih sa *a* i *b*, u slučaju procesora sa jednoadresnim formatom instrukcija jedan izvorišni operand se dobija implicitno iz akumulatora a drugi izvorišni operand iz lokacije specificirane sa *a*, dok se u slučaju procesora sa nulaadresnim formatom instrukcija oba izvorišna operanda dobijaju implicitno sa vrha steka. Dalje izvršavanje instrukcije je identično za sva četiri formata instrukcije, jer se od prvog izvorišnog operanda oduzme drugi izvorišni operand, izvrši se provera dobijene vrednosti i na osnovu toga vrši postavljanje indikatora N, Z, C i V u programskoj statusnoj reči PSW procesora. Podaci nad kojima se izvršava oduzimanje mogu da budu predstavljeni na više načina, kao na primer celobrojne veličine bez znaka, celobrojne veličine sa znakom i to obično u drugom komplementu, pokretni zarez, pri čemu to može da se čini na više različitih dužina, kao na primer bajt, dva bajta, četiri bajta itd. Za svaki mogući način predstavljanja podataka i određenu dužinu podatka mora da postoji poseban kod operacije upoređivanja.

Indikator N se postavlja na vrednost 1 ukoliko je rezultat izvršene operacije oduzimanja negativan, dok se u suprotnom slučaju postavlja na vrednost 0. Ovaj indikator ima smisla da se postavi na vrednost 1 jedino ukoliko se oduzimanje realizuje nad tipovima podataka kod kojih postoje i negativne i pozitivne vrednosti, kao na primer celobrojne vrednosti sa znakom u drugom komplementu. Ovaj indikator bi trebalo da se postavlja na vrednost 0 ukoliko se oduzimanje realizuje nad tipovima podataka kod kojih postoje samo pozitivne vrednosti, kao na primer celobrojne vrednosti bez znaka. Međutim, usvojeno pravilo je da se indikator N postavlja na vrednost najstarijeg bita rezultata operacije oduzimanja.

Indikator Z se postavlja na vrednost 1 ukoliko je rezultat izvršene operacije oduzimanja nula, dok se u suprotnom slučaju postavlja na vrednost 0. Ovaj indikator se postavlja za sve tipove podataka.

Indikator C se jedino postavlja ukoliko se oduzimanje realizuje nad tipovima podataka kod kojih postoje jedino pozitivne vrednosti, dok se tada indikator V postavlja na vrednost 0. Indikator C se postavlja na 1 ukoliko je prvi izvorišni operand manji od drugog izvorišnog operanda i predstavlja pozajmicu, dok se u suprotnom slučaju postavlja na 0.

Indikator V se jedino postavlja ukoliko se oduzimanje realizuje nad tipovima podataka kod kojih postoje i pozitivne i negativne vrednosti, dok se tada indikator C postavlja na vrednost 0. Indikator V se postavlja na vrednost 1 ukoliko je prvi izvorišni operand pozitivna veličina i drugi izvorišni operand je negativna veličina i imaju takve vrednosti da se prilikom njihovog odizimanja javlja prekoračenje. Indikator V se postavlja na vrednost 1 i ukoliko je prvi izvorišni operand negativna veličina i drugi izvorišni operand je pozitivna veličina i imaju takve vrednosti da se prilikom njihovog odizimanja javlja prekoračenje. U suprotnom slučaju indikator V ima vrednost 0.

Ovakvo postavljanje indikatora N, Z, C i V je deo jednog od mehanizama za realizaciju uslovnih skokova u programu. Taj mehanizam predviđa da se instrukcijom CMP na osnovu vrednosti dva izvorišna operanda postave indikatori N, Z, C i V i da se potom instrukcijom uslovnog skoka vrši odgovarajuća provera indikatora N, Z, C i V da bi se utvrdilo da li je uslov za skok ispunjen ili nije i da se ukoliko je uslov ispunjen skače u programu, dok se u suprotnom produžava sa sekvencijalnim izvršavanjem programa (tabela 1). Uslovnih skokova ima za svaku od 6 relacija i to jednako, nije jednako, veće, veće ili jednako, manje i manje ili jednako, pri čemu se relacije veće, veće ili jednako, manje i manje ili jednako definišu posebno za aritmetiku bez znaka i za aritmetiku sa znakom. Zbog toga ima 10 uslovnih skokova. Kao ilustracija ovog mehanizma za realizaciju uslovnih skokova u daljem tekstu je dato nekoliko primera.

Tabela 1 Uslovi za skok

instrukcija	značenje	uslov
BEQL	jednako	$Z = 1$
BNEQ	nejednako	$Z = 0$
BGRTU	veće nego bez znaka	$C \vee Z = 0$
BGREU	veće nego ili jednako bez znaka	$C = 0$
BLSSU	manje nego bez znaka	$C = 1$
BLEQU	manje nego ili jednako bez znaka	$C \vee Z = 1$
BGRT	veće nego sa znakom	$(N \oplus V) \vee Z = 0$
BGRE	veće nego ili jednako sa znakom	$N \oplus V = 0$
BLSS	manje nego sa znakom	$(N \oplus V) = 1$
BLEQ	manje nego ili jednako sa znakom	$(N \oplus V) \vee Z = 1$

Kao prvi primer za operaciju CMP uzete su binarne reči 0101 i 0101. Ukoliko se posmatraju binarne reči 0101 i 0101 vidi se da su jednake. Na osnovu toga se zaključuje da uslov za skok treba da bude ispunjen za instrukcije BEQL (jednako), BGREU (veće nego ili jednako bez znaka), BLEQU (manje nego ili jednako bez znaka), BGRE (veće nego ili jednako sa znakom) i BLEQ (manje nego ili jednako sa znakom), dok uslovi za skok ne treba da budu ispunjeni za instrukcije BNEQ (nejednako), BGRTU (veće nego bez znaka), BLSSU (manje nego bez znaka), BGRT (veće nego sa znakom) i BLSS (manje nego sa znakom).

Ukoliko se sada u okviru izvršavanja instrukcije CMP realizuje oduzimanje binarnih reči 0101 i 0101, kao rezultat se dobija binarna reč 0000. Na osnovu toga se postavljaju $N=0$ i $Z=1$. Ako se prilikom ovog oduzimanja binarne reči 0101 i 0101 interpretiraju kao celobrojne veličine bez znaka, prilikom ovog oduzimanja nema pozajmice. Na osnovu toga se postavlja $C=0$. Ako se prilikom ovog oduzimanja binarne reči 0101 i 0101 interpretiraju kao celobrojne veličine bez znaka, prilikom ovog oduzimanja nema prekoračenja. Na osnovu toga je $V=0$. Dobijene vrednosti za N, Z, C i V sada treba zameniti u izrazima za uslov koji treba da bude ispunjen za svaki od 10 uslovnih skokova da bi se skok realizovao (tabela). Na osnovu toga se dobija da je uslov

$Z = 1$ za instrukciju BEQL (jednako) ispunjen,
 $C = 0$ za instrukciju BGREU (veće nego ili jednako bez znaka) ispunjen,
 $C \vee Z = 1$ za instrukciju BLEQU (manje nego ili jednako bez znaka) ispunjen,
 $N \oplus V = 0$ za instrukciju BGRE (veće nego ili jednako sa znakom) ispunjen,
 $(N \oplus V) \vee Z = 1$ za instrukciju BLEQ (manje nego ili jednako sa znakom) ispunjen

i da uslov

$Z = 0$ za instrukciju BNEQ (nejednako) nije ispunjen,
 $C \vee Z = 0$ za instrukciju BGRTU (veće nego bez znaka) nije ispunjen,
 $C = 1$ za instrukciju BLSSU (manje nego bez znaka) nije ispunjen,
 $(N \oplus V) \vee Z = 0$ za instrukciju BGRT (veće nego sa znakom) nije ispunjen
 $(N \oplus V) = 1$ za instrukciju BLSS (manje nego sa znakom) nije ispunjen.

Kao drugi primer za operaciju CMP uzete su binarne reči 0011 i 0010. Ukoliko se posmatraju binarne reči 0011 i 0010 vidi se da su nejednake. Takođe se vidi da bez obzira na to da li se binarne reči 0011 i 0010 posmatraju kao celobrojne veličine bez znaka ili kao celobrojne veličine sa znakom u drugom komplementu, da je binarna re 0011 veća od binarne reči 0010. Na osnovu toga se zaključuje da uslov za skok treba da bude ispunjen za instrukcije BNEQ (nejednako), BGRTU (veće nego bez znaka), BGREU (veće nego ili jednako bez znaka), BGRT (veće nego sa znakom) i BGRE (veće nego ili jednako sa znakom), dok uslovi za skok ne treba da budu ispunjeni za instrukcije BEQL (jednako), BLSSU (manje nego bez znaka), BLEQU (manje nego ili jednako bez znaka), BLSS (manje nego sa znakom) i BLEQ (manje nego ili jednako sa znakom).

Ukoliko se sada u okviru izvršavanja instrukcije CMP realizuje oduzimanje binarnih reči 0011 i 0010, kao rezultat se dobija binarna reč 0001. Na osnovu toga se postavljaju $N=0$ i $Z=0$. Ako se prilikom ovog oduzimanja binarne reči 0011 i 0010 interpretiraju kao celobrojne veličine bez znaka, prilikom ovog oduzimanja nema pozajmice. Na osnovu toga se postavlja $C=0$. Ako se prilikom ovog oduzimanja binarne reči 0011 i 0010 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, prilikom ovog oduzimanja nema prekoračenja. Na osnovu toga je $V=0$. Dobijene vrednosti za N , Z , C i V sada treba zameniti u izrazima za uslov koji treba da bude ispunjen za svaki od 10 uslovnih skokova da bi se skok realizovao (tabela). Na osnovu toga se dobija da je uslov

$Z = 0$ za instrukciju BNEQ (nejednako) je ispunjen,
 $C \vee Z = 0$ za instrukciju BGRTU (veće nego bez znaka) je ispunjen
 $C = 0$ za instrukciju BGREU (veće nego ili jednako bez znaka) ispunjen,
 $(N \oplus V) \vee Z = 0$ za instrukciju BGRT (veće nego sa znakom) je ispunjen
 $N \oplus V = 0$ za instrukciju BGRE (veće nego ili jednako sa znakom) je ispunjen,

i da uslov

$Z = 1$ za instrukciju BEQL (jednako) nije ispunjen
 $C = 1$ za instrukciju BLSSU (manje nego bez znaka) nije ispunjen,
 $C \vee Z = 1$ za instrukciju BLEQU (manje nego ili jednako bez znaka) nije ispunjen,
 $(N \oplus V) = 1$ za instrukciju BLSS (manje nego sa znakom) nije ispunjen.
 $(N \oplus V) \vee Z = 1$ za instrukciju BLEQ (manje nego ili jednako sa znakom) nije ispunjen.

Kao treći primer za operaciju CMP uzete su binarne reči 0010 i 0011. Ukoliko se posmatraju binarne reči 0010 i 0011 vidi se da su nejednake. Takođe se vidi da bez obzira na to da li se binarne reči 0010 i 0011 posmatraju kao celobrojne veličine bez znaka ili kao celobrojne veličine sa znakom u drugom komplementu, da je binarna reč 0010 manja od binarne reči 0011. Na osnovu toga se zaključuje da uslov za skok treba da bude ispunjen za instrukcije BNEQ (nejednako), BLSSU (manje nego bez znaka), BLEQU (manje nego ili

jednako bez znaka), BLSS (manje nego sa znakom) i BLEQ (manje nego ili jednako sa znakom), dok uslovi za skok ne treba da budu ispunjeni za instrukcije BEQL (jednako), BGRTU (veće nego bez znaka), BGREU (veće nego ili jednako bez znaka), BGRT (veće nego sa znakom) i BGRE (veće nego ili jednako sa znakom),

Ukoliko se sada u okviru izvršavanja instrukcije CMP realizuje oduzimanje binarnih reči 0010 i 0011, kao rezultat se dobija binarna reč 1111. Na osnovu toga se postavljaju $N=1$ i $Z=0$. Ako se prilikom ovog oduzimanja binarne reči 0010 i 0011 interpretiraju kao celobrojne veličine bez znaka, prilikom ovog oduzimanja ima pozajmice. Na osnovu toga se postavlja $C=1$. Ako se prilikom ovog oduzimanja binarne reči 0010 i 0011 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, prilikom ovog oduzimanja nema prekoračenja. Na osnovu toga je $V=0$. Dobijene vrednosti za N , Z , C i V sada treba zameniti u izrazima za uslov koji treba da bude ispunjen za svaki od 10 uslovnih skokova da bi se skok realizovao (tabela). Na osnovu toga se dobija da je uslov

$Z = 0$ za instrukciju BNEQ (nejednako) je ispunjen,

$C = 1$ za instrukciju BLSSU (manje nego bez znaka) je ispunjen,

$C \vee Z = 1$ za instrukciju BLEQU (manje nego ili jednako bez znaka) je ispunjen,

$(N \oplus V) = 1$ za instrukciju BLSS (manje nego sa znakom) je ispunjen.

$(N \oplus V) \vee Z = 1$ za instrukciju BLEQ (manje nego ili jednako sa znakom) je ispunjen.

i da uslov

$Z = 1$ za instrukciju BEQL (jednako) nije ispunjen,

$C \vee Z = 0$ za instrukciju BGRTU (veće nego bez znaka) nije ispunjen,

$C = 0$ za instrukciju BGREU (veće nego ili jednako bez znaka) nije ispunjen,

$(N \oplus V) \vee Z = 0$ za instrukciju BGRT (veće nego sa znakom) nije ispunjen i

$N \oplus V = 0$ za instrukciju BGRE (veće nego ili jednako sa znakom) nije ispunjen.

Kao četvrti primer za operaciju CMP uzete su binarne reči 1110 i 1011. Ukoliko se posmatraju binarne reči 1110 i 1011 vidi se da su nejednake. Takođe se vidi da bez obzira na to da li se binarne reči 1110 i 1011 posmatraju kao celobrojne veličine bez znaka ili kao celobrojne veličine sa znakom u drugom komplementu, da je binarna reč 1110 veća od binarne reči 1011. Na osnovu toga se zaključuje da uslov za skok treba da bude ispunjen za instrukcije BNEQ (nejednako), BGRTU (veće nego bez znaka), BGREU (veće nego ili jednako bez znaka), BGRT (veće nego sa znakom) i BGRE (veće nego ili jednako sa znakom), dok uslovi za skok ne treba da budu ispunjeni za instrukcije BEQL (jednako), BLSSU (manje nego bez znaka), BLEQU (manje nego ili jednako bez znaka), BLSS (manje nego sa znakom) i BLEQ (manje nego ili jednako sa znakom).

Ukoliko se sada u okviru izvršavanja instrukcije CMP realizuje oduzimanje binarnih reči 1110 i 1011, kao rezultat se dobija binarna reč 0011. Na osnovu toga se postavljaju $N=0$ i $Z=0$. Ako se prilikom ovog oduzimanja binarne reči 1110 i 1011 interpretiraju kao celobrojne veličine bez znaka, prilikom ovog oduzimanja nema pozajmice. Na osnovu toga se postavlja $C=0$. Ako se prilikom ovog oduzimanja binarne reči 1110 i 1011 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, prilikom ovog oduzimanja nema prekoračenja. Na osnovu toga je $V=0$. Dobijene vrednosti za N , Z , C i V sada treba zameniti u izrazima za uslov koji treba da bude ispunjen za svaki od 10 uslovnih skokova da bi se skok realizovao (tabela). Na osnovu toga se dobija da je uslov

$Z = 0$ za instrukciju BNEQ (nejednako) je ispunjen,

$C \vee Z = 0$ za instrukciju BGRTU (veće nego bez znaka) je ispunjen

$C = 0$ za instrukciju BGREU (veće nego ili jednako bez znaka) ispunjen,

$(N \oplus V) \vee Z = 0$ za instrukciju BGRT (veće nego sa znakom) je ispunjen

$N \oplus V = 0$ za instrukciju BGRE (veće nego ili jednako sa znakom) je ispunjen,
i da uslov

$Z = 1$ za instrukciju BEQL (jednako) nije ispunjen

$C = 1$ za instrukciju BLSSU (manje nego bez znaka) nije ispunjen,

$C \vee Z = 1$ za instrukciju BLEQU (manje nego ili jednako bez znaka) nije ispunjen,

$(N \oplus V) = 1$ za instrukciju BLSS (manje nego sa znakom) nije ispunjen.

$(N \oplus V) \vee Z = 1$ za instrukciju BLEQ (manje nego ili jednako sa znakom) nije ispunjen.

Kao peti primer za operaciju CMP uzete su binarne reči 1011 i 1110. Ukoliko se posmatraju binarne reči 1011 i 1110 vidi se da su nejednake. Takođe se vidi da bez obzira na to da li se binarne reči 1011 i 1110 posmatraju kao celobrojne veličine bez znaka ili kao celobrojne veličine sa znakom u drugom komplementu, da je binarna reč 1011 manja od binarne reči 1110. Na osnovu toga se zaključuje da uslov za skok treba da bude ispunjen za instrukcije BNEQ (nejednako), BLSSU (manje nego bez znaka), BLEQU (manje nego ili jednako bez znaka), BLSS (manje nego sa znakom) i BLEQ (manje nego ili jednako sa znakom), dok uslovi za skok ne treba da budu ispunjeni za instrukcije BEQL (jednako), BGRTU (veće nego bez znaka), BGREU (veće nego ili jednako bez znaka), BGRT (veće nego sa znakom) i BGRE (veće nego ili jednako sa znakom),

Ukoliko se sada u okviru izvršavanja instrukcije CMP realizuje oduzimanje binarnih reči 1011 i 1110, kao rezultat se dobija binarna reč 1101. Na osnovu toga se postavljaju $N=1$ i $Z=0$. Ako se prilikom ovog oduzimanja binarne reči 1011 i 1110 interpretiraju kao celobrojne veličine bez znaka, prilikom ovog oduzimanja ima pozajmice. Na osnovu toga se postavlja $C=1$. Ako se prilikom ovog oduzimanja binarne reči 1011 i 1110 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, prilikom ovog oduzimanja nema prekoračenja. Na osnovu toga je $V=0$. Dobijene vrednosti za N , Z , C i V sada treba zameniti u izrazima za uslov koji treba da bude ispunjen za svaki od 10 uslovnih skokova da bi se skok realizovao (tabela). Na osnovu toga se dobija da je uslov

$Z = 0$ za instrukciju BNEQ (nejednako) je ispunjen,

$C = 1$ za instrukciju BLSSU (manje nego bez znaka) je ispunjen,

$C \vee Z = 1$ za instrukciju BLEQU (manje nego ili jednako bez znaka) je ispunjen,

$(N \oplus V) = 1$ za instrukciju BLSS (manje nego sa znakom) je ispunjen.

$(N \oplus V) \vee Z = 1$ za instrukciju BLEQ (manje nego ili jednako sa znakom) je ispunjen.

i da uslov

$Z = 1$ za instrukciju BEQL (jednako) nije ispunjen,

$C \vee Z = 0$ za instrukciju BGRTU (veće nego bez znaka) nije ispunjen,

$C = 0$ za instrukciju BGREU (veće nego ili jednako bez znaka) nije ispunjen,

$(N \oplus V) \vee Z = 0$ za instrukciju BGRT (veće nego sa znakom) nije ispunjen i

$N \oplus V = 0$ za instrukciju BGRE (veće nego ili jednako sa znakom) nije ispunjen.

Kao šesti primer za operaciju CMP uzete su binarne reči 1010 i 0010. Ukoliko se posmatraju binarne reči 1010 i 0010 vidi se da su nejednake. Takođe se vidi da ukoliko se binarne reči 1010 i 0010 posmatraju kao celobrojne veličine bez znaka da je binarna reč 1010 veća od binarne reči 0010, dok u slučaju kada se binarne reči 1010 i 0010 posmatraju kao celobrojne veličine sa znakom u drugom komplementu, da je binarna reč 1010 manja od binarne reči 0010. Na osnovu toga se zaključuje da uslov za skok treba da bude ispunjen za instrukcije BNEQ (nejednako), BGRTU (veće nego bez znaka), BGREU (veće nego ili jednako bez znaka), BLSS (manje nego sa znakom) i BLEQ (manje nego ili jednako sa znakom), dok uslovi za skok ne treba da budu ispunjeni za instrukcije BEQL (jednako),

BLSSU (manje nego bez znaka), BLEQU (manje nego ili jednako bez znaka), BGRT (veće nego sa znakom) i BGRE (veće nego ili jednako sa znakom).

Ukoliko se sada u okviru izvršavanja instrukcije CMP realizuje oduzimanje binarnih reči 1010 i 0010, kao rezultat se dobija binarna reč 1000. Na osnovu toga se postavljaju $N=1$ i $Z=0$. Ako se prilikom ovog oduzimanja binarne reči 1010 i 0010 interpretiraju kao celobrojne veličine bez znaka, prilikom ovog oduzimanja nema pozajmice. Na osnovu toga se postavlja $C=0$. Ako se prilikom ovog oduzimanja binarne reči 1010 i 0010 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, prilikom ovog oduzimanja nema prekoračenja. Na osnovu toga je $V=0$. Dobijene vrednosti za N , Z , C i V sada treba zameniti u izrazima za uslov koji treba da bude ispunjen za svaki od 10 uslovnih skokova da bi se skok realizovao (tabela). Na osnovu toga se dobija da je uslov

$Z = 0$ za instrukciju BNEQ (nejednako) je ispunjen,

$C \vee Z = 0$ za instrukciju BGRTU (veće nego bez znaka) je ispunjen

$C = 0$ za instrukciju BGREU (veće nego ili jednako bez znaka) ispunjen,

$(N \oplus V) = 1$ za instrukcijuBLSS (manje nego sa znakom) je ispunjen.

$(N \oplus V) \vee Z = 1$ za instrukciju BLEQ (manje nego ili jednako sa znakom) je ispunjen

i da uslov

$Z = 1$ za instrukciju BEQL (jednako) nije ispunjen

$C = 1$ za instrukciju BLSSU (manje nego bez znaka) nije ispunjen,

$C \vee Z = 1$ za instrukciju BLEQU (manje nego ili jednako bez znaka) nije ispunjen

$(N \oplus V) \vee Z = 0$ za instrukcijuBGRT (veće nego sa znakom) nije ispunjen

$N \oplus V = 0$ za instrukciju BGRE (veće nego ili jednako sa znakom) nije ispunjen.

Kao sedmi primer za operaciju CMP uzete su binarne reči 0010 i 1010. Ukoliko se posmatraju binarne reči 0010 i 1010 vidi se da su nejednake. Takođe se vidi da ukoliko se binarne reči 0010 i 1010 posmatraju kao celobrojne veličine bez znaka da je binarna reč 0010 manja od binarne reči 1010, dok u slučaju kada se binarne reči 0010 i 1010 posmatraju kao celobrojne veličine sa znakom u drugom komplementu, da je binarna reč 0010 veća od binarne reči 1010. Na osnovu toga se zaključuje da uslov za skok treba da bude ispunjen za instrukcije BNEQ (nejednako), BLSSU (manje nego bez znaka), BLEQU (manje nego ili jednako bez znaka), BGRT (veće nego sa znakom) i BGRE (veće nego ili jednako sa znakom), dok uslovi za skok ne treba da budu ispunjeni za instrukcije BEQL (jednako), BGRTU (veće nego bez znaka), BGREU (veće nego ili jednako bez znaka), BLSS (manje nego sa znakom) i BLEQ (manje nego ili jednako sa znakom).

Ukoliko se sada u okviru izvršavanja instrukcije CMP realizuje oduzimanje binarnih reči 0010 i 1010, kao rezultat se dobija binarna reč 1000. Na osnovu toga se postavljaju $N=1$ i $Z=0$. Ako se prilikom ovog oduzimanja binarne reči 0010 i 1010 interpretiraju kao celobrojne veličine bez znaka, prilikom ovog oduzimanja ima pozajmice. Na osnovu toga se postavlja $C=1$. Ako se prilikom ovog oduzimanja binarne reči 0010 i 1010 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, prilikom ovog oduzimanja ima prekoračenja. Na osnovu toga je $V=1$. Dobijene vrednosti za N , Z , C i V sada treba zameniti u izrazima za uslov koji treba da bude ispunjen za svaki od 10 uslovnih skokova da bi se skok realizovao (tabela). Na osnovu toga se dobija da je uslov

$Z = 0$ za instrukciju BNEQ (nejednako) je ispunjen,

$C = 1$ za instrukciju BLSSU (manje nego bez znaka) je ispunjen,

$C \vee Z = 1$ za instrukciju BLEQU (manje nego ili jednako bez znaka) je ispunjen,

$(N \oplus V) \vee Z = 0$ za instrukciju BGRT (veće nego sa znakom) nije ispunjen i

$N \oplus V = 0$ za instrukciju BGRE (veće nego ili jednako sa znakom) nije ispunjen

i da uslov

$Z = 1$ za instrukciju BEQL (jednako) nije ispunjen,

$C \vee Z = 0$ za instrukciju BGRTU (veće nego bez znaka) nije ispunjen,

$C = 0$ za instrukciju BGREU (veće nego ili jednako bez znaka) nije ispunjen,

$(N \oplus V) = 1$ za instrukciju BLSS (manje nego sa znakom) je ispunjen.

$(N \oplus V) \vee Z = 1$ za instrukciju BLEQ (manje nego ili jednako sa znakom) je ispunjen.

U datim primerima je ilustrovan jedan od mehanizama za realizaciju uslovnih skokova u programu koji predviđa da se instrukcijom CMP na osnovu vrednosti dva izvorišna operanda postave indikatori N, Z, C i V i da se potom instrukcijom uslovnog skoka vrši odgovarajuća provera indikatora N, Z, C i V da bi se utvrdilo da li je uslov za skok ispunjen ili nije i da bi se ukoliko je uslov ispunjen napravio skok u programu, dok bi se u suprotnom produžilo sa sekvencijalnim izvršavanjem programa (tabela 1). To praktično znači da u program treba ubaciti instrukciju CMP i jednu instrukciju uslovnog skoka na svim onim mestima na kojima se realizuju uslovni skokovi. Ubacivanje ovih instrukcija u program značajno povećava ukupan broj instrukcija programa, što direktno utiče na efikasnost izvršavanja programa.

Da bi se umanjili negativni efekti ubacivanja instrukcija CMP na mestima uslovnih granjanja, sastavni deo izvršavanja svih aritmetičkih operacija je i postavljanje indikatora N, Z, C i V na osnovu rezultata izvršene operacije. Indikator N se postavlja na vrednost najstarijeg bita rezultata operacije. Indikator Z se postavlja na vrednost 1 ukoliko je rezultat izvršene operacije nula, dok se u suprotnom slučaju postavlja na vrednost 0. Indikator C se jedino postavlja ukoliko se operacija sabiranja ili oduzimanja realizuje nad tipovima podataka kod kojih postoje jedino pozitivne vrednosti, dok se tada indikator V postavlja na vrednost 0. Indikator C se postavlja na vrednost 1 ukoliko postoji prenos ili pozajmica kod operacija sabiranja i oduzimanja, respektivno, dok se za operacije množenja i deljenja posebno definiše. Indikator V se jedino postavlja ukoliko se operacija sabiranja ili oduzimanja realizuje nad tipovima podataka kod kojih postoje i pozitivne i negativne vrednosti, dok se tada indikator C postavlja na vrednost 0. Indikator V se postavlja na vrednost 1 ukoliko postoji prekoračenje kod operacija sabiranja i oduzimanja, dok se za operacije množenja i deljenja posebno definiše. Stoga kad god je moguće u programima realizovati uslovne skokove na osnovu vrednosti indikatora N, Z, C i V koje je postavila zadnja aritmetička instrukcija pre instrukcije instrukcije skoka, nema potrebe stavljati instrukciju CMP kojom bi se upoređivala dobijena vrednost aritmetičke operacije sa 0.

Kao primer za operaciju ADD uzete su binarne reči 1010 i 0010 čijim se sabiranjem dobija binarna reč 1100. Ukoliko se posmatra binarna reč 1100 vidi se da je različita od nule. Takođe se vidi da ukoliko se binarne reči 1010 i 0010 posmatraju kao celobrojne veličine bez znaka da je binarna reč 1010 veća od binarne reči 0010, dok u slučaju kada se binarne reči 1010 i 0010 posmatraju kao celobrojne veličine sa znakom u drugom komplementu, da je binarna reč 1010 manja od binarne reči 0010. Na osnovu toga se zaključuje da uslov za skok treba da bude ispunjen za instrukcije BNEQ (nejednako), BGRTU (veće nego bez znaka), BGREU (veće nego ili jednako bez znaka), BLSS (manje nego sa znakom) i BLEQ (manje nego ili jednako sa znakom), dok uslovi za skok ne treba da budu ispunjeni za instrukcije BEQL (jednako), BLSSU (manje nego bez znaka), BLEQU (manje nego ili jednako bez znaka), BGRT (veće nego sa znakom) i BGRE (veće nego ili jednako sa znakom).

Ukoliko se sada u okviru izvršavanja instrukcije ADD realizuje sabiranje binarnih reči 1010 i 0010, kao rezultat se dobija binarna reč 1100. Na osnovu toga se postavljaju $N=1$ i $Z=0$. Ako se prilikom ovog sabiranja binarne reči 1010 i 0010 interpretiraju kao celobrojne veličine bez znaka, prilikom ovog sabiranja nema prenosa. Na osnovu toga se postavlja $C=0$.

Ako se prilikom ovog sabiranja binarne reči 1010 i 0010 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, prilikom ovog sabiranja nema prekoračenja. Na osnovu toga je $V=0$. Dobile vrednosti za N , Z , C i V sada treba zameniti u izrazima za uslov koji treba da bude ispunjen za svaki od 10 uslovnih skokova da bi se skok realizovao (tabela). Na osnovu toga se dobija da je uslov

$Z = 0$ za instrukciju BNEQ (nejednako) je ispunjen,

$C \vee Z = 0$ za instrukciju BGRTU (veće nego bez znaka) je ispunjen

$C = 0$ za instrukciju BGREU (veće nego ili jednako bez znaka) ispunjen,

$(N \oplus V) = 1$ za instrukciju BLSS (manje nego sa znakom) je ispunjen.

$(N \oplus V) \vee Z = 1$ za instrukciju BLEQ (manje nego ili jednako sa znakom) je ispunjen

i da uslov

$Z = 1$ za instrukciju BEQL (jednako) nije ispunjen

$C = 1$ za instrukciju BLSSU (manje nego bez znaka) nije ispunjen,

$C \vee Z = 1$ za instrukciju BLEQU (manje nego ili jednako bez znaka) nije ispunjen

$(N \oplus V) \vee Z = 0$ za instrukciju BGRT (veće nego sa znakom) nije ispunjen

$N \oplus V = 0$ za instrukciju BGRE (veće nego ili jednako sa znakom) nije ispunjen.

1.5.1.3 LOGIČKE INSTRUKCIJE

Logičkim instrukcijama se realizuju standardne logičke operacije I, ILI, ekskluzivno ILI i komplementiranja. Format instrukcija za logičke operacije I, ILI i ekskluzivno ILI za procesore sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija su:

AND a, b, c

AND a, b

AND a

AND

OR a, b, c

OR a, b

OR a

OR

XOR a, b, c

XOR a, b

XOR a

XOR

pri čemu su sa AND, OR i XOR simbolički označena polja koda operacije za logičke operacije I, ILI i ekskluzivno ILI, respektivno. Eksplicitna specifikacija izvorišnih i odredišnih operanada poljima a , b i c je ista kao i kod instrukcije sabiranja. Implicitna izvorišta i odredišta u formatima instrukcija gde ih ima, su, takođe, ista kao i kod instrukcije sabiranja.

Podaci nad kojima se izvršavaju logičke operacije I, ILI i ekskluzivno ILI su binarne reči, pri čemu to može da se čini na više različitih dužina, kao na primer bajt, dva bajta, četiri bajta itd. Za svaku moguću dužinu binarnih reči mora da postoji poseban kod operacije za svaku od operacija I, ILI i ekskluzivno ILI. Kao ilustracija može se uzeti da ako u procesoru binarne reči mogu da budu dužine 8, 16, 32 i 64 bita, mora da postoje 4 kodova operacija za operaciju I.

Formati instrukcija za logičku operaciju komplementiranja za procesore sa dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija su:

NOT a, b

NOT a

NOT

pri čemu su sa NOT simbolički označeno polje koda operacije za logičku operaciju komplementiranja. Eksplicitna specifikacija izvorišnih i odredišnih operanada poljima a i b je ista kao i kod instrukcije inkrementiranja. Implicitna izvorišta i odredišta u formatima instrukcija gde ih ima, su, takođe, ista kao i kod instrukcije inkrementiranja.

Podaci nad kojima se izvršava logička operacija NOT su binarne reči, pri čemu to može da se čini na više različitih dužina, kao na primer bajt, dva bajta, četiri bajta itd. Za svaku moguću dužinu binarnih reči mora da postoji poseban kod operacije za operacija NOT. Kao ilustracija može se uzeti da ako u procesoru binarne reči mogu da budu dužine 8, 16, 32 i 64 bita, mora da postoje 4 kodova operacija za operaciju NOT.

Sastavni deo izvršavanja svih logičkih instrukcija je i postavljanje indikatora N, Z, C i V na osnovu dobijenog rezultata. Indikator N se postavlja na vrednost najstarijeg bita rezultata operacije. Indikator Z se postavlja na vrednost 1 ukoliko je rezultat izvršene operacije nula, dok se u suprotnom slučaju postavlja na vrednost 0. Indikatori C i V se postavljaju na vrednost 0. To omogućava da se u situacijama kada to ima smisla posle logičke instrukcije stavi instrukcija uslognog skoka. Pri tome skok na jednako i nejednako ima smisla da se koristi, jer ukazuje da li je binarna reč dobijena kao rezultat neke logičke operacije 0 ili različita od 0. Ostali uslovni skokovi nemaju smisla da se koriste, jer oni pretpostavljaju da je binarna reč rezultata logičke operacije prilikom postavljanja indikatora C i V interpretirana kao celobrojna vrednost bez znaka i sa znakom u drugom komplementu, respektivno, što nije slučaj.

Među logičkim instrukcijama se pojavljuje i poseban slučaj operacije logičko I kod koje se rezultat logičke I operacije nikuda ne upisuje, već se samo vrši provera dobijene vrednosti i na osnovu toga vrši postavljanje indikatora N, Z, C i V u programskoj statusnoj reči PSW procesora. Ova instrukcija se naziva logičko upoređivanje. Format instrukcije upoređivanja za procesore sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija su:

TST a, b, c

TST a, b

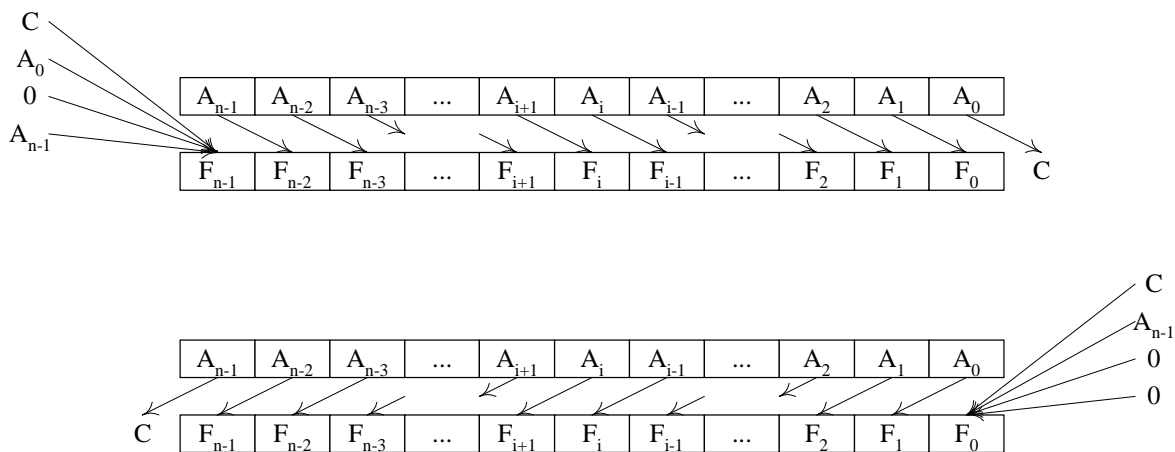
TST a

TST

pri čemu je sa TST simbolički označeno polje koda operacije za operaciju logičkog upoređivanja. Eksplicitna specifikacija izvorišnih i odredišnih operanada poljima a , b i c je ista kao i kod instrukcije CMP. Implicitna izvorišta i odredišta u formatima instrukcija gde ih ima, su, takođe, ista kao i kod instrukcije CMP. Podaci nad kojima se izvršava operacija logičkog upoređivanja su binarne reči, pri čemu to može da se čini na više različitih dužina, kao na primer bajt, dva bajta, četiri bajta itd. Za svaku moguću dužinu binarnih reči mora da postoji poseban kod operacije Kao ilustracija može se uzeti da ako u procesoru binarne reči mogu da budu dužine 8, 16, 32 i 64 bita, mora da postoje 4 koda operacije za operaciju logičkog upoređivanja.

1.5.1.4 INSTRUKCIJE POMERANJA

Instrukcijama pomeranja se realizuje pomeranje binarne reči za jedno mesto udesno ili jedno mesto ulevo (slika 26).



Slika 26 Pomeranje udesno i ulevo

Kod pomeranja udesno binarne reči $A_{n-1}A_{n-2}\dots A_1A_0$ razredi binarne reči $A_{n-1}A_{n-2}\dots A_2A_1$ postaju razredi $F_{n-2}F_{n-3}\dots F_1F_0$ binarne reči rezultata $F_{n-1}F_{n-2}\dots F_1F_0$. Najstariji bit F_{n-1} binarne reči rezultata može da dobije jednu od vrednosti A_{n-1} , 0, A_0 ili C i u zavisnosti koju od te četiri vrednosti dobije razlikuju se četiri instrukcije pomeranja udesno i to aritmetičko pomeranje udesno, logičko pomeranje udesno, rotiranje udesno i rotiranje udesno kroz indikator C, respektivno. U sva četiri slučaja najmlađi bit A_0 binarne reči $A_{n-1}A_{n-2}\dots A_1A_0$ se upisuje u indikator C registra PSW.

Kod pomeranja ulevo binarne reči $A_{n-1}A_{n-2}\dots A_1A_0$ razredi binarne reči $A_{n-2}A_{n-3}\dots A_1A_0$ postaju razredi $F_{n-1}F_{n-2}\dots F_2F_1$ binarne reči rezultata $F_{n-1}F_{n-2}\dots F_1F_0$. Najmlađi bit F_0 binarne reči rezultata može da dobije jednu od vrednosti 0, 0, A_{n-1} ili C i u zavisnosti koju od te četiri vrednosti dobije razlikuju se četiri instrukcije pomeranja ulevo i to aritmetičko pomeranje ulevo, logičko pomeranje ulevo, rotiranje ulevo i rotiranje ulevo kroz indikator C, respektivno. U sva četiri slučaja najstariji bit A_{n-1} binarne reči $A_{n-1}A_{n-2}\dots A_1A_0$ se upisuje u indikator C registra PSW.

Formati instrukcije aritmetičkog pomeranja udesno za procesore sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija su:

- ASR a, b, c
- ASR a, b
- ASR a
- ASR

Sa ASR je simbolički označeno polje koda operacije, a sa a i b specifikacije izvorišnih i odredišnih operanada. Sa c je označeno polje za specifikaciju trećeg operanda koje postoji kod procesora sa troadresnim formatom instrukcija, mada se u slučaju operacije pomeranja polje c ne koristi. Razlika između instrukcija pomeranja kod procesora sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija je samo u tome kako se dolazi do izvorišnog i odredišnog operanda. U slučaju procesora sa troadresnim i dvoadresnim formatima instrukcija izvorišni i odredišni operandi se dobijaju sa lokacija specificiranih sa a i b , u slučaju procesora sa jednoadresnim formatom instrukcija operand specificiran sa a je i izvorišni i odredišni operand, dok se u slučaju procesora sa nulaadresnim formatom instrukcija vrh steka implicitno izvorište i odredište. Dalje izvršavanje instrukcije je identično za sva četiri formata instrukcije, jer se izvorišni operand pomeri i smesti u odredište. Indikator N se postavlja na vrednost najstarijeg bita rezultata, indikator Z na 1 ili 0 u zavisnosti od toga da li je dobijeni rezultat 0 ili nije 0, indikator C se postavlja na vrednost najmlađeg bita izvorišnog operanda i indikator V se postavlja na vrednost 0. Podaci nad kojima se izvršava

pomeranje su binarne reči koje mogu da budu na više različitih dužina, kao na primer bajt, dva bajta, četiri bajta itd. Za svaku moguću dužinu binarnih reči mora da postoji poseban kod operacije za operacija ASR. Kao ilustracija može se uzeti da ako u procesoru binarne reči mogu da budu dužine 8, 16, 32 i 64 bita, mora da postoje 4 kodova operacija za operaciju ASR.

Slična je situacija i sa formatima instrukcija za operacije oduzimanja, množenja i deljenja. Formati instrukcija za ove operacije za procesore sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija su:

LSR a, b, c
LSR a, b
LSR a
LSR
ROR a, b, c
ROR a, b
ROR a
ROR
RORC a, b, c
RORC a, b
RORC a
RORC
ASL a, b, c
ASL a, b
ASL a
ASL
LSL a, b, c
LSL a, b
LSL a
LSL ROR a, b, c
ROL a, b
ROL a
ROL
ROLC a, b, c
ROLC a, b
ROLC a
ROLC

Eksplisitna specifikacija izvorišnih i odredišnih operanada poljima a, b i c je ista kao i kod instrukcije ASR. Implicitna izvorišta i odredišta u formatima instrukcija gde ih ima, su, takođe, ista kao i kod instrukcije sabiranja.

Podaci nad kojima se izvršavaju pomeranja su binarne reči, pri čemu to može da se čini na više različitih dužina, kao na primer bajt, dva bajta, četiri bajta itd. Za svaku moguću dužinu binarnih reči mora da postoji poseban kod operacije za svaku od operacija pomeranja. Kao ilustracija može se uzeti da ako u procesoru binarne reči mogu da budu dužine 8, 16, 32 i 64 bita, mora da postoje 4 kodova operacija za svaku operaciju pomeranja.

Ukoliko se binarna reč 1000 pomeri logički udesno dobija se binarna reč 0100. Ukoliko se sada i binarna reč 0100 pomeri logički udesno dobija se 0010. Ukoliko se binarne reči 1000, 0100 i 0010 interpretiraju kao celobrojne veličine bez znaka, i time predstavljaju vrednosti 8, 4 i 2, uočava se da se logičkim pomeranjem udesno realizuje deljenje sa 2. Međutim, ukoliko se binarne reči 1000, 0100 i 0010 interpretiraju kao celobrojne veličine sa znakom u drugom

komplementu, i time predstavljaju vrednosti -8, 4 i 2, uočava se da se logičkim pomeranjem udesno ne realizuje deljenje sa 2.

Ukoliko se binarna reč 1000 pomeri aritmetički udesno dobija se binarna reč 1100. Ukoliko se sada i binarna reč 1100 pomeri aritmetički udesno dobija se 1110. Ukoliko se binarne reči 1000, 1100 i 1110 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, i time predstavljaju vrednosti -8, -4 i -2, uočava se da se aritmetičkim pomeranjem udesno realizuje deljenje sa 2.

Ukoliko se binarna reč 0010 pomeri aritmetički ili logički ulevo dobija se binarna reč 0100. Ukoliko se sada i binarna reč 0100 pomeri aritmetički ili logički ulevo dobija se 1000. Ukoliko se binarne reči 0010, 0100 i 1000 interpretiraju kao celobrojne veličine bez znaka, i time predstavljaju vrednosti 2, 4 i 8, uočava se da se aritmetičkim ili logičkim pomeranjem ulevo realizuje množenje sa 2.

Ukoliko se binarna reč 1110 pomeri aritmetički ili logički ulevo dobija se binarna reč 1100. Ukoliko se sada i binarna reč 1100 pomeri aritmetički ili logički ulevo dobija se 1000. Ukoliko se binarne reči 1110, 1100 i 1000 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, i time predstavljaju vrednosti -2, -4 i -8, uočava se da se aritmetičkim ili logičkim pomeranjem ulevo realizuje množenje sa 2.

1.5.1.5 INSTRUKCIJE SKOKA

Korišćenje programskog brojača PC kao adrese memorijske lokacije sa koje se čita instrukcija i njegovo inkrementiranje prilikom svakog čitanja instrukcije, kao posledicu ima sekvencijalnost u izvršavanju instrukcija programa. Stoga se na mestima na kojima treba odstupiti od sekvencijalnosti stavljaju instrukcije skoka. Njihov efekat je modifikovanje vrednosti programskog brojača PC vrednošću koja predstavlja adresu memorijske lokacije sa koje treba produžiti sa izvršavanjem instrukcija.

Instrukcije skoka se svrstavaju u sledeće grupe: instrukcije bezuslovnog skoka, instrukcije uslovnog skoka, instrukcije skoka na potprogram i povratka iz potprograma, instrukcija skoka u prekidnu rutinu i instrukcija povratka iz prekidne rutine.

Instrukcija bezuslovnog skoka

Format instrukcije bezuslovnog skoka je

JMP adresa

pri čemu je sa *JMP* simbolički označeno polje koda operacije, a sa *adresa* polje sa vrednošću adrese memorijske lokacije na kojoj se nalazi instrukcija na čije izvršavanje treba preći. Izvršavanja instrukcije *JMP* se sastoji u upisivanju vrednost iz polja *adresa* u programski brojač PC.

Kao ilustracija se može uzeti instrukcija

JMP 3579

u kojoj 3579 predstavlja vrednost polja *adresa*. Izvršavanja instrukcije *JMP* se sastoji u upisivanju vrednost 3579 iz polja *adresa* u programski brojač PC. Ova instrukcija se koristi ukoliko je u vreme prevođenja poznata adresa instrukcije na koju treba skočiti.

U slučaju kada adresa instrukcije na koju treba skočiti nije poznata u vreme prevođenja, već se izračunava tokom izvršavanja programa, koristi se instrukcija bezuslovnog skoka čiji je format

JMPIND *a*

pri čemu je sa JMPIND simbolički označeno polje koda operacije, a sa *a* polje sa specifikacijom adrese memorijske lokacije na kojoj se nalazi instrukcija na čije izvršavanje treba preći. Polje *a* ima identičnu strukturu kao polja u prethodno razmatranim instrukcijama kojima se eksplicitno specificiraju izvorišni i odredišni operandi korišćenjem različitih načina adresiranja (odjeljak 1.4), pri čemu su dozvoljena samo memorijska adresiranja, dok direktno registarsko i neposredno adresiranje nisu dozvoljeni. Izvršavanje instrukcije JMPIND se sastoji u sračunavanju adrese memorijske lokacije na osnovu specificiranog memorijskog adresiranja i upisivanju sračunate adrese memorijske lokacije u programski brojač PC.

Kao ilustracija se može uzeti da je najpre nekim instrukcijama pre instrukcije JMPIND *a* kao adresa memorijske lokacije na koju treba skočiti sračunata vrednost 3579, da je ta vrednost upisana u adresni registar AR5 i da instrukcijom JMPIND *a* vrednost 3579 iz adresnog registra AR5 treba upisati u programski brojač PC. Ukoliko je u instrukciji JMPIND *a* poljem *a* specificirano registarsko indirektno adresiranje i kao adresni registar specificiran registar AR5, tada sadržaj 3579 registra AR5 predstavlja adresu na koju treba skočiti, pa se vrednost 3579 upisuje u programski brojač PC.

Kao ilustracija se može uzeti da je najpre nekim instrukcijama pre instrukcije JMPIND *a* kao adresa memorijske lokacije na koju treba skočiti sračunata vrednost 3579, da je ta vrednost upisana u memorijsku lokaciju na adresi 1234 i da instrukcijom JMPIND *a* vrednost 3579 iz memorijske lokacije na adresi 1234 treba upisati u programski brojač PC. Ukoliko je u instrukciji JMPIND *a* poljem *a* specificirano memorijsko indirektno adresiranje i adresa 1234, tada 1234 predstavlja adresu memorijske lokacije na kojoj se nalazi vrednost 3579 koja predstavlja adresu memorijske lokacije na koju treba skočiti, pa se vrednost 3579 upisuje u programski brojač PC.

Efekat upisivanja adrese skoka 3579 u programski brojač PC koji se ostvaruje instrukcijom JMP 3579

moguće je ostvariti i instrukcijom

JMPIND *a*

ukoliko je adresa skoka 3579 poznata u vreme prevođenja. Ukoliko je u instrukciji JMPIND *a* poljem *a* specificirano memorijsko direktno adresiranje i adresa 3579, tada 3579 predstavlja adresu memorijske lokacije na koju treba skočiti, pa se ta vrednost upisuje u programski brojač PC.

Instrukcije uslovnog skoka

Format instrukcija uslovnog skoka je

BEQL *pom*, BNEQ *pom*,

BGRTU *pom*, BGREU *pom*, BLSSU *pom*, BLEQU *pom*,

BGRT *pom*, BGRE *pom*, BLSS *pom*, BLEQ *pom*,

BNEG *pom*, BNNG *pom*, BOVF *pom* i BNVF *disp*

pri čemu je sa BEQL, BNEQ, BGRTU, BGREU, BLSSU, BLEQU, BGRT, BGRE, BLSS, BLEQ, BNEG, BNNG, BOVF i BNVF simbolički označeno polje koda operacije, a sa *pom* polje sa vrednošću pomeraja sa kojim treba napraviti relativan skok u odnosu na tekuću vrednost programskog brojača ukoliko je uslov za skok ispunjen. Tokom izvršavanja instrukcije uslovnog skoka se sabiranjem tekuće vrednosti programskog brojača PC i pomeraja *pom* sračunava adresa memorijske lokacije skoka, na osnovu vrednosti indikatora N, Z, C i V se proverava da li je uslov za skok specificiran poljem koda operacije ispunjen ili ne i na osnovu toga sračunata adresa memorijske lokacije skoka upisuje ili ne upisuje u programski brojač PC, respektivno. Time se ili prelazi na izvršavanje prve instrukcije u novoj sekvenci

instrukcija ili produžava sa izvršavanjem prve sledeće instrukcije u sekvenci. Ovde se podrazumeva da se prilikom čitanja instrukcije uslovnog skoka vrši inkrementiranje vrednosti programskog brojača PC i da tekuća vrednost programskog brojača sa kojom se sabira pomeraj predstavlja adresu prve sledeće instrukcije posle instrukcije uslovnog skoka. Pomeraj *pm* se zadaje kao celobrojna veličina sa znakom u drugom komplementu, pa se relativni skokovi u odnosu na tekuću vrednost programskog brojača PC realizuju i unapred i unazad.

Ovakva realizacija uslovnih skokova u programu podrazumeva da aritmetičke, logičke i pomeračke instrukcije na osnovu rezultata odgovarajuće operacije postavljaju indikatore N, Z, C i V i da se potom instrukcijom uslovnog skoka vrši odgovarajuća provera indikatora N, Z, C i V da bi se utvrdilo da li je uslov za skok ispunjen ili nije i da bi se na osnovu toga ili skočilo u programu ili produžilo sa sekvencijalnim izvršavanjem programa (tabela 2). Uslovnih skokova ima za svaku od 6 relacija i to jednako, nejednako, veće, veće ili jednako, manje i manje ili jednako, pri čemu se relacije veće, veće ili jednako, manje i manje ili jednako definišu posebno za aritmetiku bez znaka i za aritmetiku sa znakom. Zbog toga ima 10 uslovnih skokova.

Tabela 2 Instrukcije uslovnog skoka

instrukcija	značenje	uslov
BEQL	skok na jednako	$Z = 1$
BNEQ	skok na nejednako	$Z = 0$
BGRTU	skok na veće nego (bez znaka)	$C \vee Z = 0$
BGREU	skok na veće nego ili jednako (bez znaka)	$C = 0$
BLSSU	skok na manje nego (bez znaka)	$C = 1$
BLEQU	skok na manje nego ili jednako (bez znaka)	$C \vee Z = 1$
BGRT	skok na veće nego (sa znakom)	$(N \oplus V) \vee Z = 0$
BGRE	skok na veće nego ili jednako (sa znakom)	$N \oplus V = 0$
BLSS	skok na manje nego (sa znakom)	$(N \oplus V) = 1$
BLEQ	skok na manje nego ili jednako (sa znakom)	$(N \oplus V) \vee Z = 1$
BNEG	skok na $N = 1$	$N = 1$
BNNG	skok na $N = 0$	$N = 0$
BOVF	skok na $V = 1$	$V = 1$
BNVF	skok na $V = 0$	$V = 0$

U nekim situacijama postoji potreba i da se uslovni skokovi realizuju na osnovu pojedinačnih vrednosti indikatora N, Z, C, i V. Instrukcijama BEQL, BNEQ, BGREU i BLSSU je to moguće uraditi sa indikatorima Z i C. Da bi ta mogućnost postojala i za indikatore N i V, postoje i instrukcije uslovnog skoka BNEG, BNNG, BOVF i BNVF.

Ima instrukcija uslovnog skoka kod kojih se adresa skoka dobija na isti način kao i u slučaju instrukcije bezuslovnog skoka, pa se u formatu instrukcije umesto pomeraja specificira adresa. Dosta je uobičajeno da mnemonici instrukcija uslovnog skoka kod kojih se realizuje relativni skok počinju slovom B (Branch), dok mnemonici instrukcija uslovnog skoka kod kojih se realizuje apsolutni skok počinju slovom J (Jump). U tom slučaju bi formati instrukcija uslovnog skoka, koje realizuju iste uslovne skokove kao i instrukcije u tabeli , ali kod kojih bi se umesto relativnog realizovao apsolutni skok, bili

JEQL *adresa*, JNEQ *adresa*,
 JGRTU *adresa*, JGREU *adresa*, JLSSU *adresa*, JLEQU *adresa*,
 JGRT *adresa*, JGRE *adresa*, JLSS *adresa*, JLEQ *adresa*,
 JNEG *adresa*, JNNG *adresa*, JOVF *adresa* i JNVF *disp*

pri čemu je sa JEQL, JNEQ, JGRTU, JGREU, JLSSU, JLEQU, JGRT, JGRE, JLSS, JLEQ, JNEG, JNNG, JOVF i JNVF simbolički označeno polje koda operacije, a sa *adresa* polje sa vrednošću adrese na koju treba napraviti apsolutni skok ukoliko je uslov za skok ispunjen.

Instrukcije skoka na potprogram i povratka iz potprograma

Format instrukcije skoka na potprogram je

JSR *adresa*

pri čemu je sa JSR simbolički označeno polje koda operacije, a sa *adresa* polje sa vrednošću adrese memorijske lokacije na kojoj se nalazi prva instrukcija potprograma na čije izvršavanje treba preći. Izvršavanje instrukcije JSR se sastoji u stavljanju na stek tekuće vrednosti programskog brojača PC i upisivanju vrednosti iz polja *adresa* u programski brojač PC. Ovde se podrazumeva da se prilikom čitanja instrukcije JSR vrši inkrementiranje vrednosti programskog brojača PC i da tekuća vrednost programskog brojača koja se stavlja na stek predstavlja adresu prve sledeće instrukcije posle instrukcije JSR.

Format instrukcije povratka iz potprograma je

RTS

pri čemu je sa RTS simbolički označeno polje koda operacije. Instrukcija RTS mora da bude zadnja instrukcija potprograma. Izvršavanje instrukcije RTS se sastoji u skidanju vrednosti sa steka i upisivanju u programski brojač PC. Ovde se podrazumeva da je tokom izvršavanja potprograma broj stavljanja vrednosti na stek jednaku broju skidanja vrednosti sa steka, čime se obezbeđuje da instrukcija RTS skida sa steka i upisuje u programski brojač PC onu vrednost koju je instrukcija JSR stavila na stek.

Instrukcija skoka na prekidnu rutinu i instrukcija povratka iz prekidne rutine

Format instrukcije skoka na prekidnu rutinu je

INT *adresa*

pri čemu je sa INT simbolički označeno polje koda operacije, a sa *adresa* polje sa vrednošću adrese memorijske lokacije na kojoj se nalazi prva instrukcija prekidne rutine na čije izvršavanje treba preći kao rezultat izvršavanja instrukcije INT. Izvršavanje instrukcije INT se sastoji u stavljanju na stek tekuće vrednosti programskog brojača PC, programske statusne reči PSW, a u nekim realizacijama procesora i svih programski dostupnih registara, i upisivanju vrednosti iz polja *adresa* u programski brojač PC. Ovde se podrazumeva da se prilikom čitanja instrukcije INT vrši inkrementiranje vrednosti programskog brojača PC i da tekuća vrednost programskog brojača PC koja se stavlja na stek predstavlja adresu prve sledeće instrukcije posle instrukcije INT. U nekim realizacijama mehanizma prekida adrese prekidnih rutina se smeštaju u posebnu tabelu koja se naziva interapt vektor tabela. U tom slučaju u polju *adresa* instrukcije INT se nalazi broj ulaza u interapt vektora tabelu iz koga treba očitati adresu prekidne rutine i upisati u programski brojač PC

Format instrukcije povratka iz povratka iz prekidne rutine je

RTI

pri čemu je sa RTI simbolički označeno polje koda operacije. Instrukcija RTI mora da bude zadnja instrukcija prekidne rutine. Izvršavanje instrukcije RTI se sastoji u skidanju vrednosti sa steka i upisivanju u programski dostupne registre, ukoliko se radi o realizacijama procesora kod kojih se prilikom skoka na prekidnu rutinu ovi registri čuvaju na steku, programsku statusnu reč i programski brojač PC. Ovde se podrazumeva da je tokom izvršavanja prekidne rutine broj stavljanja vrednosti na stek jednaku broju skidanja vrednosti sa steka, čime se obezbeđuje da instrukcija RTI skida sa steka vrednosti onih registara koje je instrukcija INT stavila na stek.

1.5.1.6 MEŠOVITE INSTRUKCIJE

U skupu instrukcija mogu da se nađu i instrukcije kojima se omogućava zadavanje različitih režima rada računara, pruža podrška za testiranje programa itd. Ovde će biti date same neke od njih, jer je za njihovo razumevanje potrebno poznavanje mogućih režima rada računara. Kao primer se može uzeti razred I u registru PSW koji vrednostima 1 i 0 određuje režime rada računara sa reagovanjem i bez reagovanja na zahteve za prekid koji dolaze od kontrolera periferija, respektivno. Instrukcijom INTE se zadaje režim rada u kome se reaguje na prekide koji dolaze od kontrolera periferija, dok se instrukcijom INTD zadaje režim rada u kome se ne reaguje na prekide koji dolaze od kontrolera periferija. Instrukcija NOP ne proizvodi nikakvo dejstvo.

Format instrukcije zadavanja režima rada u kome se reaguje na zahteve za prekid je

INTE

pri čemu je sa INTE simbolički označeno polje koda operacije. Izvršavanje instrukcije INTE se sastoji u upisivanju vrednosti 1 u razred I registra PSW.

Format instrukcije zadavanja režima rada u kome se ne reaguje na zahteve za prekid je
INTD

pri čemu je sa INTE simbolički označeno polje koda operacije. Izvršavanje instrukcije INTE se sastoji u upisivanju vrednosti 0 u razred I registra PSW.

Format instrukcije bez dejstva je

NOP

pri čemu je sa NOP simbolički označeno polje koda operacije. Izvršavanje instrukcije NOP se sastoji samo u čitanju instrukcije i inkrementiranju programskog brojača PC, posle čega se prelazi na izvršavanje sledeće instrukcije.

1.5.2 NESTANDARDNE INSTRUKCIJE

Nestandardne instrukcije uključuje operacije kojima se omogućuje jednostavnije preslikavanje konstrukcija iz programskih jezika i operativnih sistema u instrukcije procesora, nego ukoliko se to čini standardnim instrukcijama. Nestandardne instrukcije se javljaju u nekom vidu kod procesora CISC tipa dok se kod procesora RISC tipa ne javljaju. Skup nestandardnih instrukcija čine instrukcije nad celobrojnim veličinama promenljive dužine, string instrukcije, decimalne instrukcije, instrukcije kontrole petlji itd.

Instrukcije nad celobrojnim veličinama promenljive dužine

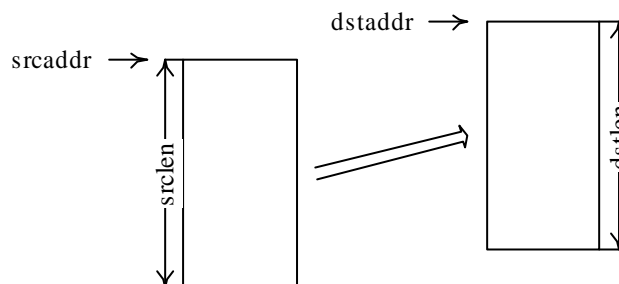
String instrukcije

MOVC *srclen, srcaddr, fill, dstlen, dstaddr*

Instrukcija MOVC realizuje kopiranje karaktera izvorišnog stringa u karaktere odredišnog stringa. početna adresa izvorišnog stringa je specificirana operandom *srcaddr*, dužina izvorišnog stringa je specificirana operandom *srclen*, početna adresa odredišnog stringa je specificirana operandom *dstaddr*, dužina odredišnog stringa je specificirana operandom *dstlen* i karakter FILL je specificiran operandom *fill*. Izvorišni i odredišni string mogu biti različitih dužina. Kopiranje karaktera izvorišnog stringa u karaktere odredišnog stringa se realizuje uvek na dužini odredišnog stringa. Karakter FILL se koristi samo u slučaju da je izvorišni string kraći od odredišnog stringa. Početne adrese i dužine izvorišnog i odredišnog stringa mogu tako da budu zadate da dođe do njihovog delimičnog ili potpunog preklapanja.

Rezultat izvršavanja instrukcije MOVC je, pored kopiranja sadržaja jednog dela memorije u drugi deo memorije, i postavljanje indikatora Z i C. Indikator Z ukazuje da li su dužine izvorišnog i odredišnog stringa jednake ili ne, dok indikator C, u slučaju da dužine izvorišnog i odredišnog stringa nisu jednake, ukazuje da li je odredišni string duži od izvorišnog ili ne.

Ako su izvorišni i odredišni string jednake dužine ($srclen = dstlen$) karakteri izvorišnog stringa će u celosti biti kopirani u karaktere odredišnog stringa (slika 1). Indikator Z će biti postavljen na jedan ($Z=1$), a indikator C na nulu ($C=0$).



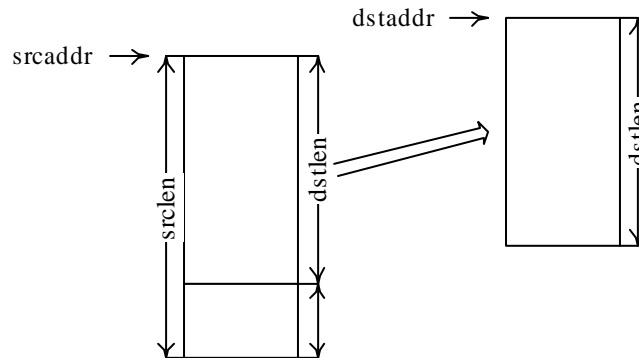
Slika 1 Instrukcija MOVC za $srclen = dstlen$

Ako je izvorišni string duži od odredišnog stringa ($srclen > dstlen$) biće preneto samo prvih *dstlen* karaktera izvorišnog stringa u odredišni string, a preostali karakteri izvorišnog stringa neće biti preneti (slika 2). U ovom slučaju indikatori Z i C će biti postavljeni na nulu ($Z=0$, $C=0$).

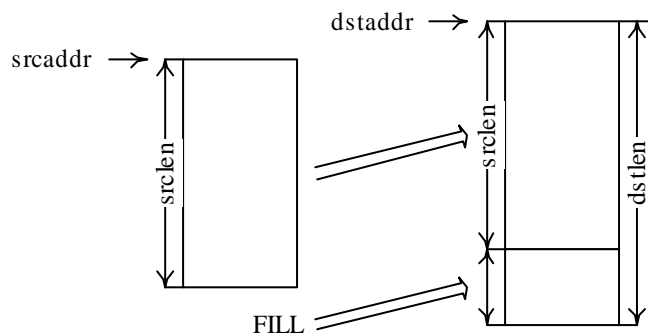
Ako je izvorišni string kraći od odredišnog stringa ($srclen < dstlen$) svi karakteri izvorišnog stringa će biti preneti u prvih *srclen* karaktera odredišnog stringa, a preostali karakteri

odredišnog stringa do dužine odredišnog stringa će biti popunjeni karakterom FILL (slika3). Indikator Z će biti postavljen na nulu ($Z=0$), a indikator C na jedan ($C=1$).

Tokom izvršavanja instrukcije MOVC, i to nakon kopiranja jednog karaktera izvorišnog stringa u element odredišnog stringa, sadržaji lokacija *srcaddr* i *dstaddr* se inkrementiraju, a sadržaji lokacija *srclen* i *dstlen* dekrementiraju. Tada sadržaj lokacije *srcaddr* predstavlja adresu sledećeg karaktera izvorišnog koji treba da bude kopiran, sadržaj lokacije *dstaddr* adresu sledećeg karaktera odredišnog stringa koji treba da bude popunjen, lokacije *srclen* broj karaktera izvorišnog stringa koji još uvek nisu kopirani, a lokacije *dstlen* broj nepopunjenih karaktera odredišnog stringa.



Slika 2 Instrukcija MOVC za $srclen > dstlen$



Slika 3 Instrukcija MOVC za $srclen < dstlen$

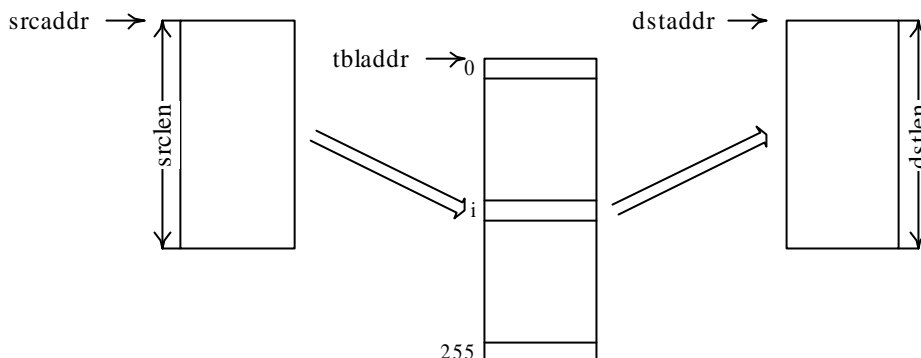
MOVTC *srclen, srcaddr, fill, tbladdr, dstlen, dstaddr*

Instrukcija MOVTC realizuju prevođenje karaktera izvorišnog stringa i njihovo smeštanje u odgovarajuće karaktere odredišnog stringa. Za prevođenje se koristi translaciona tabela koja ima 256 ulaza. Element odredišnog stringa popunjava se karakterom koji se nalazi unutar translacione tabele na adresi koja je određena vrednošću karaktera izvorišnog stringa koji se prevodi. Početna adresa izvorišnog stringa je *srcaddr*, dužina izvorišnog stringa je *srclen*, početna adresa odredišnog stringa je *dstaddr*, dužina odredišnog stringa je *dstlen*, adresa translacione tabele je *tbladdr* i karakter FILL je specificiran operandom *fill*. Izvorišni i odredišni string mogu biti različitih dužina. Prevođenje karaktera izvorišnog stringa u karaktere odredišnog stringa se realizuje uvek na dužini odredišnog stringa. Karakter FILL se koristi samo u slučaju da je izvorišni string kraći od odredišnog stringa. Početne adrese i dužine izvorišnog i odredišnog stringa i početna adresa translacione tabele mogu tako da budu zadate da dođe do njihovog delimičnog ili potpunog preklapanja.

Rezultat izvršavanja instrukcije MOVTC je, pored prevođenja sadržaja jednog dela memorije i popunjavanja prevedenim vrednostima drugog dela memorije, i postavljanje

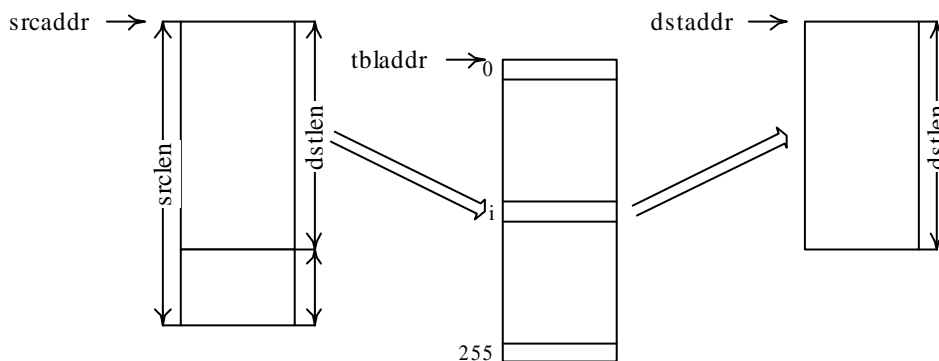
indikatora Z i C. Indikator Z ukazuje da li su dužine izvorišnog i odredišnog stringa jednake ili ne, dok indikator C, u slučaju da dužine izvorišnog i odredišnog stringa nisu jednake, ukazuje da li je odredišni string duži od izvorišnog ili ne.

Ako su izvorišni i odredišni string jednake dužine ($srclen = dstlen$) svaki karakter izvorišnog stringa će biti preveden i smešten na odgovarajuće mesto unutar odredišnog stringa (slika 4). Indikator Z će biti postavljen na jedan ($Z=1$), a indikator C na nulu ($C=0$).



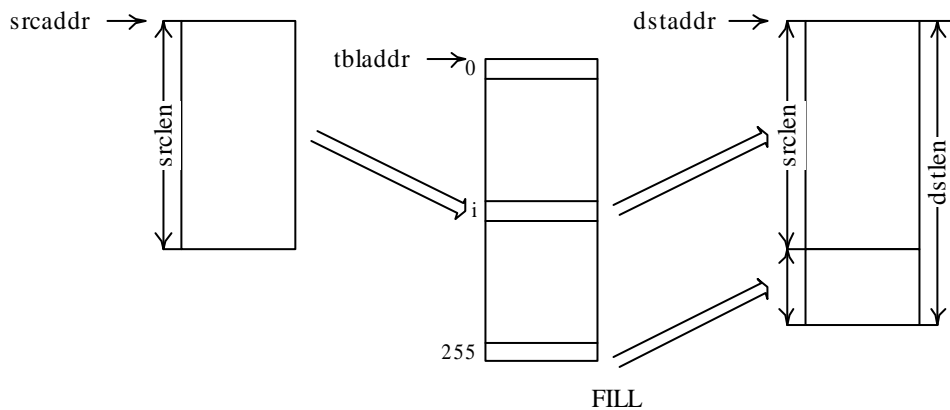
Slika 4 Instrukcija MOVTC za $srclen = dstlen$

Ako je izvorišni string duži od odredišnog stringa ($srclen > dstlen$) biće prevedeno samo prvih $dstlen$ karaktera izvorišnog stringa i smešteno na odgovarajuće mesto unutar odredišnog stringa, a preostali karakteri izvorišnog stringa neće biti prevedeni (slika 5). U ovom slučaju indikatori Z i C će biti postavljeni na nulu ($Z=0, C=0$).



Slika 5 Instrukcija MOVTC za $srclen > dstlen$

Ako je izvorišni string kraći od odredišnog stringa ($srclen < dstlen$) svi karakteri izvorišnog stringa će biti prevedeni i prevedene vrednosti smeštene u prvih $srclen$ karaktera odredišnog stringa, a preostali karakteri odredišnog stringa do dužine odredišnog stringa će biti popunjeni karakterom FILL (slika 6). Indikator Z će biti postavljen na nulu ($Z=0$), a indikator C na jedan ($C=1$).

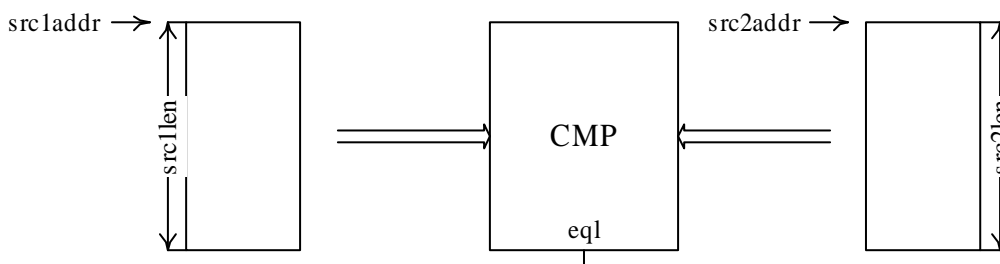


Slika 6 Instrukcija MOVTC za $srcLen < dstLen$

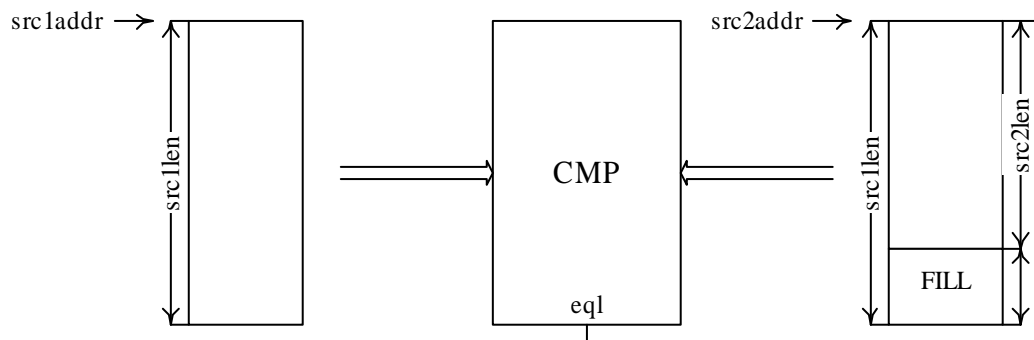
Tokom izvršavanja instrukcije MOVTC, i to nakon prevođenja jednog karaktera izvorišnog stringa i smeštanja prevedene vrednosti u karakter odredišnog stringa, sadržaji lokacija *srcaddr* i *dstaddr* se inkrementiraju, a sadržaji lokacija *srcLen* i *dstLen* dekrementiraju. Tada sadržaj lokacija *srcaddr* predstavlja adresu sledećeg karaktera izvorišnog koji treba da bude preveden, sadržaj lokacija *dstaddr* adresu sledećeg karaktera odredišnog stringa na kojoj prevedeni karakter treba da bude smešten, lokacije *srcLen* broj karaktera izvorišnog stringa koji još uvek nisu prevedeni, a lokacije *dstLen* broj nepopunjenih karaktera odredišnog stringa.

CMPC *src1Len, src1addr, fill, src2Len, src2addr*

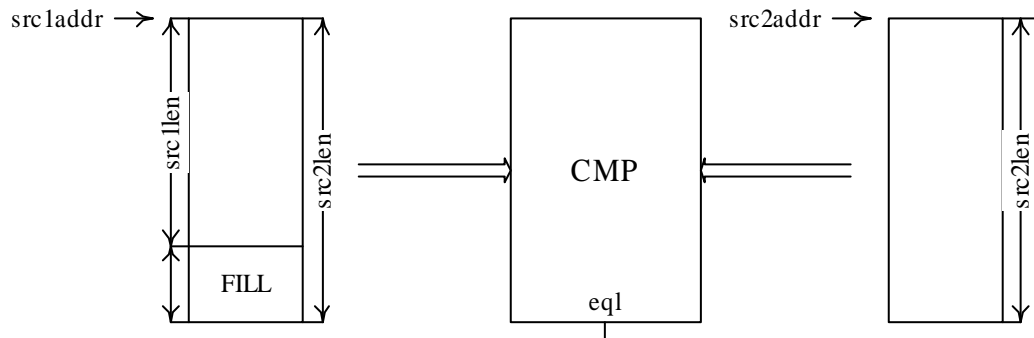
Instrukcija CMPC (COMPARE CHARACTERS) upoređuje karaktere dva izvorišna stringa radi utvrđivanja da li su dva stringa identična ili ne (slike 7, 8 i 9). Upoređivanje karaktera dva izvorišna stringa se realizuje na dužini dužeg izvorišnog stringa ukoliko postoji jednakost karaktera koji se upoređuju ili se završava ranije i to kada se prvi put otkrije nejednakost karaktera koji se upoređuju. Početna adresa prvog izvorišnog stringa je *src1addr*, dužina prvog izvorišnog stringa je *src1Len*, početna adresa drugog izvorišnog stringa je *src2addr*, dužina drugog izvorišnog stringa je *dst2Len* i karakter FILL je specificiran operandom *fill*. Izvorišni stringovi mogu da budu različitih dužina. Karakter FILL se koristi samo u slučaju da izvorišni stringovi nisu istih dužina. U slučaju različitih dužina izvorišnih stringova kada se iscrpe karakteri kraćeg izvorišnog stringa karakter FILL se koristi za poređenje sa preostalim karakterima dužeg izvorišnog stringa. Početne adrese i dužine izvorišnih stringova mogu tako da budu zadate da dođe do njihovog delimičnog ili potpunog preklapanja.



Slika 7 Instrukcija CMPC za $srcLen = dstLen$



Slika 8 Instrukcija CMPC za $src1len > src2len$



Slika 9 Instrukcija CMPC za $src1len < src2len$

Rezultat izvršavanja instrukcije CMPC je utvrđivanje da li su dva izvorišna stringa identična ili ne. Stringovi su identični ukoliko se upoređivanje karaktera dva izvorišna stringa realizuje na dužini dužeg izvorišnog stringa i pri tome postoji jednakost karaktera koji se upoređuju. Stringovi nisu identični ukoliko se upoređivanje karaktera dva izvorišna stringa završava ranije i to kada se prvi put otkrije nejednakost karaktera koji se upoređuju. Rezultat izvršavanja instrukcije CMPC se daje postavljanjem indikatora Z i C. Indikator Z ukazuje da li su dva izvorišna stringa identična ili ne, dok indikator C, u slučaju da se otkrije da neki par karaktera dva izvorišna stringa nije identičan, ukazuje koji od ta dva karaktera ima veću vrednost. Ukoliko su stringovi identični indikator Z bit će biti postavljen na vrednost jedan ($Z=1$), a indikator C na vrednost nula ($C=0$). Ukoliko stringovi nisu identični indikator Z će biti postavljen na vrednost nula ($Z=0$), a vrednost indikatora C zavisice od vrednosti karaktera stringova za koje je utvrđeno da nisu jednaki. Ako su vrednosti takve da je vrednost karaktera prvog izvorišnog stringa manja od vrednosti karaktera drugog izvorišnog stringa indikator C će biti postavljen na vrednost 1 ($C=1$). Ako su vrednosti takve da je vrednost karaktera prvog izvorišnog stringa veća od vrednosti karaktera drugog izvorišnog stringa indikator C će biti postavljen na vrednost 0 ($C=0$).

Tokom izvršavanja instrukcije CMPC, i to nakon poređenja jednog karaktera prvog izvorišnog stringa i jednog karaktera drugog izvorišnog stringa, sadržaji lokacija *src1addr* i *src2addr* se inkrementiraju, a sadržaji lokacija *src1len* i *src2len* dekrementiraju. Tada sadržaj lokacije *src1addr* predstavlja adresu sledećeg karaktera prvog izvorišnog koji treba da se upoređuje, lokacije *src2addr* adresu sledećeg karaktera drugog izvorišnog stringa koji treba da se upoređuje, lokacije *src1len* broj karaktera prvog izvorišnog stringa koji još uvek nisu upoređivani, a lokacije *src2len* broj karaktera drugog izvorišnog stringa koji još uvek nisu upoređivani.

LOCC *len, addr, char*

Instrukcija LOCC (LOCATE CHARACTER) izvršava operaciju poređenja između svakog karaktera izvorišnog stringa i karaktera poređenja (slika 10). Operacija se realizuje ili na

dužini izvorišnog stringa ukoliko je rezultat operacije nejednakost za svaki karakter izvorišnog stringa ili se završava ranije i to kada se prvi put otkrije da je rezultat operacije jednakost. Početna adresa s izvorišnog tringa je *srcaddr*, dužina izvorišnog stringa je *srclen* i karakter poređenja je specificiran operandom *char*.

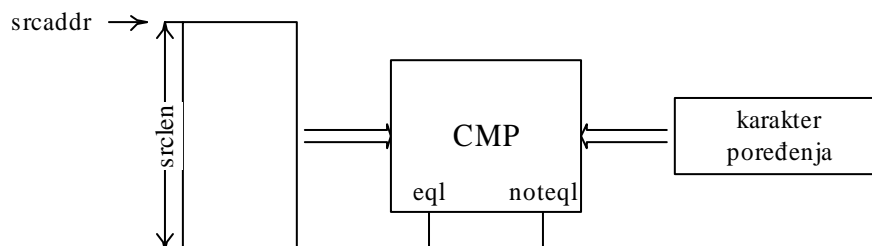
Rezultat izvršavanja instrukcije LOCC je utvrđivanje da li su svi karakteri izvorišnog stringa različiti od karaktera poređenja ili ne. Rezultat izvršavanja instrukcije LOCC se daje postavljanjem indikatora Z na vrednost jedan ili nula, dok se indikator C uvek postavlja na vrednost nula (C=0). Ukoliko su svi karakteri izvorišnog stringa različiti od karaktera poređenja indikator Z bit će biti postavljen na vrednost jedan (Z=1). Ukoliko se pojavi karakter izvorišnog stringa identičan karakteru poređenja indikator Z bit će biti postavljen na vrednost jedan (Z=0).

Tokom izvršavanja instrukcije LOCC, i to nakon poređenja svakog karaktera izvorišnog stringa i karaktera poređenja, sadržaj lokacije *srcaddr* se inkrementira, a lokacije *srclen* se dekrementira. Tada sadržaj lokacije *srcaddr* predstavlja adresu sledećeg karaktera izvorišnog stringa za koji poređenje treba da se izvrši, a lokacije *srclen* broj karaktera izvorišnog stringa za koje poređenje treba da se izvrši.

SKPC *len, addr, char*

Instrukcija SKPC (SKIP CHARACTERS) izvršava operaciju poređenja između svakog karaktera izvorišnog stringa i karaktera poređenja (slika 10). Operacija se realizuje ili na dužini izvorišnog stringa ukoliko je rezultat operacije jednakost za svaki karakter izvorišnog stringa ili se završava ranije i to kada se prvi put otkrije da je rezultat operacije nejednakost. Početna adresa izvorišnog stringa je *addr*, dužina izvorišnog stringa je *len* i karakter poređenja je specificiran operandom *char*.

Rezultat izvršavanja instrukcije SKPC je utvrđivanje da li su svi karakteri izvorišnog stringa identični karakteru poređenja ili ne. Rezultat izvršavanja instrukcije SKPC se daje postavljanjem indikatora Z na vrednost jedan ili nula, dok se indikator C uvek postavlja na vrednost nula (C=0). Ukoliko su svi karakteri izvorišnog stringa identični karakteru poređenja indikator Z bit će biti postavljen na vrednost jedan (Z=1). Ukoliko se pojavi karakter izvorišnog stringa različit od karaktera poređenja indikator Z bit će biti postavljen na vrednost jedan (Z=0).



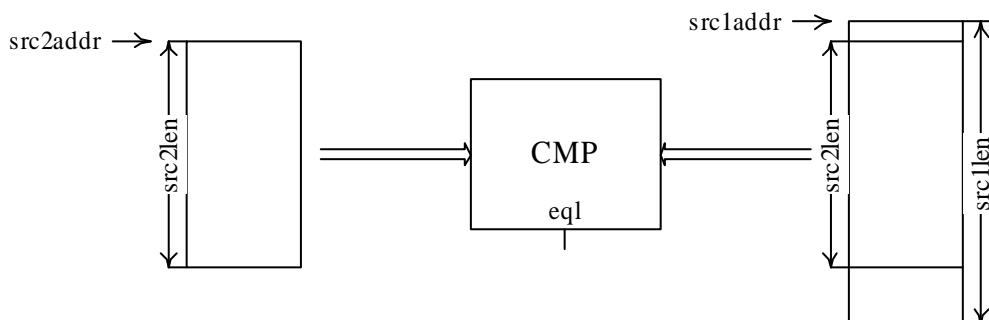
Slika 10 Instrukcije SKPC i LOCC

Tokom izvršavanja instrukcije SKPC, i to nakon poređenja svakog karaktera izvorišnog stringa i karaktera poređenja, sadržaj lokacije *addr* se inkrementira, a lokacije *len* dekrementira. Tada sadržaj lokacije *addr* predstavlja adresu sledećeg karaktera izvorišnog stringa za koji poređenje treba da se izvrši, a lokacije *len* broj karaktera izvorišnog stringa za koje poređenje treba da se izvrši.

MATCHC *src1len, src1addr, src2len, src2addr*

Instrukcija MATCHC (MATCH CHARACTERS) izvršava operaciju poređenja između karaktera izvorišnog stringa i karaktera podstringa (slika 11). Operacija se realizuje ili na dužini izvorišnog stringa ukoliko je rezultat operacije nejednakost ili se završava ranije i to kada se prvi put otkrije da je rezultat operacije jednakost. Početna adresa izvorišnog stringa je *src1addr*, dužina izvorišnog stringa je *src1len*, početna adresa podstringa je *src2addr* i dužina podstringa je *src2len*.

Rezultat izvršavanja instrukcije MATCHC je utvrđivanje da li se u nekom delu izvorišnog stringa nalazi niz karaktera identičan sa karakterima podstringa. Rezultat izvršavanja instrukcije MATCHC se daje postavljanjem indikatora Z na vrednost jedan ili nula, dok se indikator C uvek postavlja na vrednost nula (C=0). Ukoliko podstring nije pronađen u izvorišnom stringu indikator Z bit će biti postavljen na vrednost jedan (Z=1). Ukoliko je podstring pronađen u izvorišnom indikator Z bit će biti postavljen na vrednost nula (Z=0).



Slika 11 Instrukcija MATCHC

Tokom izvršavanja instrukcije MATCHC, i to nakon svakog neuspešnog poređenja karaktera podstringa i karaktera izvorišnog stringa na nekoj od mogućih pozicija podstringa u izvorišnom stringu, sadržaj lokacije *src1addr* se inkrementira, a lokacije *src1len* dekrementira. Tada sadržaj lokacije *src1addr* predstavlja adresu sledećeg karaktera izvorišnog stringa od kojeg počinje sledeća moguća pozicija podstringa u izvorišnom stringu, a lokacije *src1len* broj preostalih karaktera prvog izvorišnog stringa od kojih još uvek nije pokušano pronalaženje podstringa u izvorišnom stringu. U izvorišnom stringu ima smisla pretraživati podstring jedino ukoliko je broj preostalih karaktera izvorišnog stringa veći od ili jednak dužini podstringa.

Decimalne instrukcije

Instrukcije kontrole petlji

ACB *limit, step, index, displ*

Instrukcija ACB (Add Compare and Branch) realizuje relativni skok sa pomerajem *displ* u odnosu na tekuću vrednost programskog brojača PC pod uslovom da je suma vrednosti parametara *index* i *step* manja ili jednaka vrednosti parametra *limit*. Ukoliko je uslov za skok ispunjen ažurira se vrednost parametra *index* sumom vrednosti parametara *index* i *step*. Izvršavanje instrukcije ACB se može predstaviti na sledeći način:

if ((index + step) leq limit) then (PC <= PC + displ),

index <= index + step

AOB limit, index, displ

Instrukcija AOB (Add One and Branch) realizuje relativni skok sa pomerajem *displ* u odnosu na tekuću vrednost programskog brojača PC pod uslovom da je suma vrednosti parametara *index* i 1 manja ili jednaka vrednosti parametra *limit*. Ukoliko je uslov za skok ispunjen ažurira se vrednost parametra *index* sumom vrednosti parametara *index* i 1. Izvršavanje instrukcije AOB se može predstaviti na sledeći način:

```
if ((index + 1) leq limit) then (PC <= PC + displ),  
index <= index + 1
```

SOB index, displ

Instrukcija SOB (Subtract One and Branch) realizuje relativni skok sa pomerajem *displ* u odnosu na tekuću vrednost programskog brojača PC pod uslovom da je vrednosti parametara *index* umanjena za 1 veća ili jednaka 0. Ukoliko je uslov za skok ispunjen ažurira se vrednost parametra *index* vrednošću parametara *index* umanjenom za 1. Izvršavanje instrukcije SOB se može predstaviti na sledeći način:

```
if ((index - 1) geq 0) then (PC <= PC + displ),  
index <= index - 1
```

CASE selector, base, limit, displ[0], displ[1],... displ[limit-base]

Instrukcija CASE realizuje relativni skok sa pomerajem *displ[selector-base]* u odnosu na tekuću vrednost programskog brojača PC pod uslovom da je vrednosti parametara *selector* veća ili jednaka vrednosti parametra *base* i manja ili jednaka vrednosti parametra *limit*. U suprotno slučaju se prelazi na sledeću instrukciju. Izvršavanje instrukcije CASE se može predstaviti na sledeći način:

```
if ((selector geq base) and (selector leq limit))  
    then PC <= PC + displ[selector-base]  
    else PC <= PC + (limit-base+1)
```

BBS base, pos, displ

Instrukcija BBS (Branch on Bit Set) realizuje relativni skok sa pomerajem *displ* u odnosu na tekuću vrednost programskog brojača PC pod uslovom da je vrednost jednobitnog polja na poziciji određenoj vrednošću parametra *pos* u bajtu očitanoj sa memorijske adrese date vrednošću parametra *base* 1. Izvršavanje instrukcije BBS se može predstaviti na sledeći način:

```
if (POLJE(pos, 1, base) eql 1) then (PC <= PC + displ)
```

BBS base, pos, displ

Instrukcija BBC (Branch on Bit Clear) realizuje relativni skok sa pomerajem *displ* u odnosu na tekuću vrednost programskog brojača PC pod uslovom da je vrednost jednobitnog polja na poziciji određenoj vrednošću parametra *pos* u bajtu očitanoj sa memorijske adrese date vrednošću parametra *base* 0. Izvršavanje instrukcije BBC se može predstaviti na sledeći način:

if (POLJE(pos, 1, base) *eq* 0) *then* (PC <= PC + *displ*)