2. RAČUNALO, STROJNI JEZIK I DIGITALNA LOGIKA

U prethodnom poglavlju ustanovljeno je kako postoji više različitih ISA. U narednim izlaganjima analizirati će se različiti procesori s aspekta ISA i s aspekta programera. Različitosti procesora na ovoj razini mogu se podijeliti na:

- veličina i broj spremnika unutar CPUa,
- način na koji je organizirana glavna memorija,
- način na koji naredbe pristupaju operandima u spremnicima i memoriji.

Naredbe koje procesor koristi, a kao što je već spomenuto mogu se podijeliti na tri osnovne skupine:

- naredbe za dohvat i pohranu podataka,
- aritmetičke i logičke naredbe,
- naredbe za upravljanje tokom.

Zato će se u ovom poglavlju prikazati različiti tipovi spremnika, organizacije memorije i načina pristupa operandima ali na apstraktnoj razini kako bi se stekao utisak o idealnom procesoru.

Nakon toga opisati će se jednostavan RISC, (*Simple RISC,* skr. SRC) na razini ISA. Ovakav procesor ima mnoga svojstva današnjih modernih RISC-ova uz nužno zanemarenje nekih svojstava stvarnih RISC procesora koja bi značajno zakomplicirala ova razmatranja. Ova zanemarenja i pojednostavljenja omogućiti će nam da jednostavno pristupimo projektiranju računala na logičkoj razini. Ovo ne bi bilo moguće kada bi se radilo o složenom komercijalnom procesoru.

SRC će prvo biti opisan tekstualno, riječima i slikama, a potom i formalno primjenom RTN (*Register Transfer Notation*). Ovakav opis omogućava da se procesor opiše jednostavno i nedvosmisleno, a značajan je za sklopovsku realizaciju funkcija procesora.

Nakon analize računala s aspekta programera, ova razmatranja okončati će se pristupom s gledišta projektanta logičkih sklopova. Tada će se razmatrati neke od tehnika projektiranja računala s aspekta logičkih sklopova. Projektant logičkih sklopova primarno je usredotočen na prijenos podataka između spremnika, te prijenos podataka ALU na obradu i konačno prihvat i pohranu rezultata obrade. Razmatranja se odnose na realizaciju navedenih funkcija na razini logičkih vrata. Ovdje se po prvi put uvodi vremensko vođenje ovih procesa tzv. *timing*. Prijenos među spremnicima odvija se preko sabirnica, tako da se ukazuje na značaj njihovog projektiranja.

2.1. Podjela računala i njihovih naredbi

Prvo će se razmotriti struktura glavne memorije i spremnika kod različitih procesora. Glavna memorija obično se promatra kao polje jediničnih podataka najmanje veličine koje procesor naredbom može dohvatiti. Danas je to oktet ili osam bita, a npr. 64 bitovni realni broj je pohranjen kao niz od osam okteta. Naredba koja radi s realnim brojem pristupa ovom nizu okteta kao cjelini.

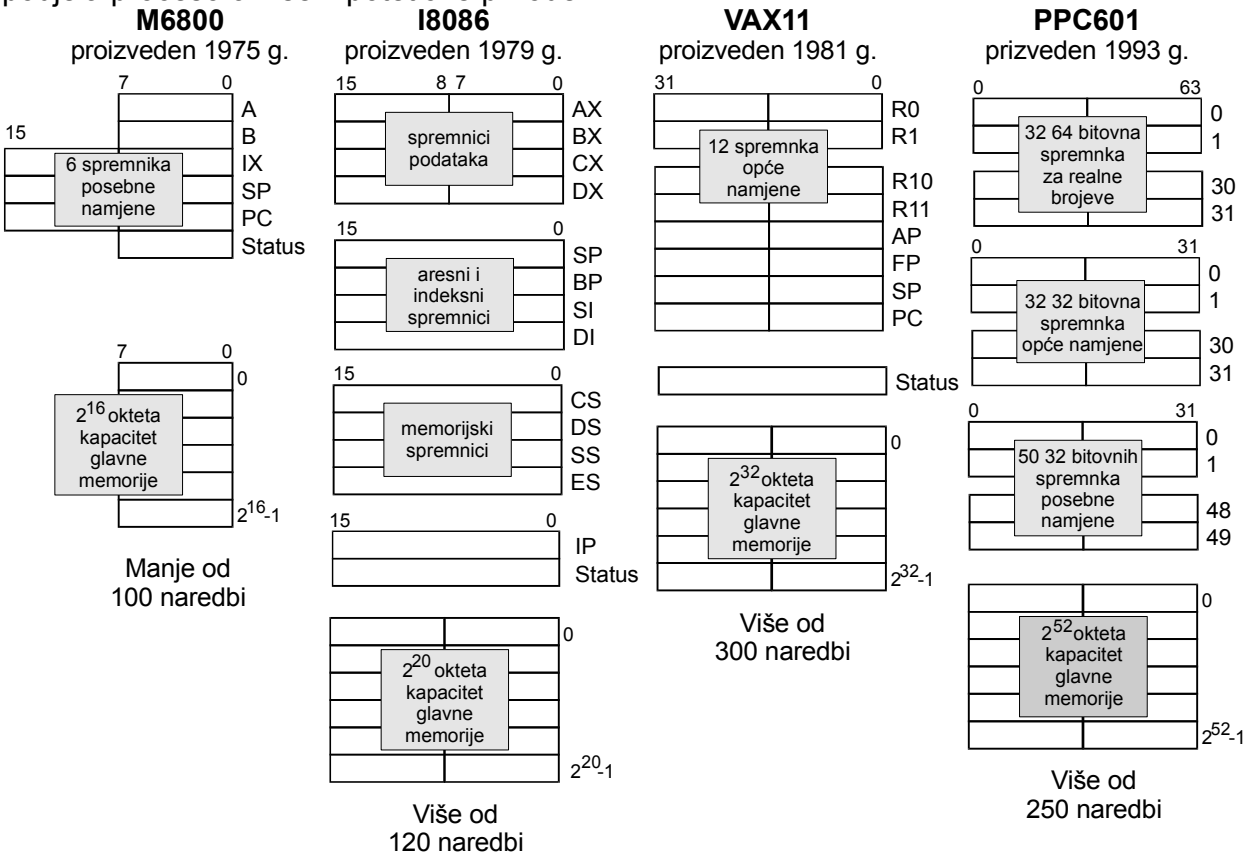
Nasuprot glavnoj memoriji koja obično ima jednostavnu strukturu, spremnici procesora međusobno se razlikuju ne samo po veličini nego po i tipu informacije (podatku) koju mogu pohraniti. Oni se nazivaju spremnici posebne namjene (*special purpose registers*). Tako

svaki procesor ima posebne spremnike za pohranu podataka i pohranu adresa. Npr. Motorola 6800, kao jedan od prvih osam bitovnih procesora, ima tri 16 bitovna adresna spremnika PC (programsko brojilo), SP (*Stack Pointer*, pokazivač stoga) IX (*Index Register*) od kojih svaki ima posebnu namjenu, te dva osam bitovna spremnika za podatke A i B. Ovi spremnici nazivaju se *akumulatori* ili *aritmetički spremnici* i koriste se za privremenu pohranu operanada i rezultata aritmetičkih i logičkih operacija. Poseban osam bitovni spremnik *Status* je niz bita koji daju informaciju o stanju procesora kao i rezultatima aritmetičkih i logičkih operacija (rezultat nula, negativan, dijeljenje s nulom, pretek i sl.).

Opisani primjer zorno prikazuje kako procesor može biti klasificiran temeljem strukture i funkcije svojih spremnika. Tako se razlikuju:

- procesori temeljeni na akumulatoru,
- procesori temeljeni na stogu (*stack*),
- procesori temeljeni na spremnicima opće namjene (*general purpose registers*).

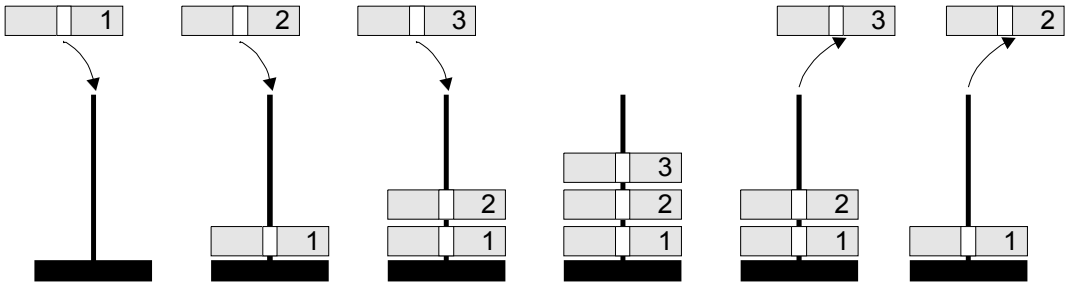
Stvarni procesori ne spadaju striktno u ni jednu od skupina iako se može ustvrditi da neko od rješenja dominira u njegovoj izvedbi. Tako se npr. prema slici 2.1. Motorola M6800 i Intel I8086 mogu klasificirati kao procesori temeljni na akumulatorima, dok su Digital VAX i Motorola PPC601 procesori temeljeni na opće-namjenskim spremnicima iako svi navedeni procesori raspolažu s naredbama i spremnicima koji pripadaju drugim tipovima procesora. Tako svi procesori mogu obavljati naredbe sa stogom PUSH i POP. Zato je navedena podjela procesora više hipotetične prirode.



Slika 2.1. *Programerski model različitih procesora.*

Procesori temeljeni na akumulatoru imaju ograničen broj akumulatora, obično samo jedan, zajedno s spremnicima za pohranu adresa. Naziv akumulator potječe od činjenice da ovaj spremnik služi za pohranu jednog operanda aritmetičke ili logičke operacije, te kao spremnik u koji se po izvođenju operacije pohranjuje njen rezultat. Tako on praktički služi za akumulaciju podataka te mu odatle potječe i naziv. U vrijeme prvih procesora kada je memorija bila skupa ovakva računala su bila naročito popularna zato jer je uvijek jedan operand bio u akumulatoru te je u naredbi bilo potrebni specificirati samo memorijsku lokaciju drugog operanda. Rezultat operacije se uvijek pohranjivao u akumulator. Zato se ovakvi procesori nazivaju procesori s jednom adresom. Naredbe ovakvih procesora zato su bili znatno kraće (dva okteta) i program je zauzimao manje memorije. Ovakav pristup ima bitnih ograničenja kod proračuna aritmetičkih i logičkih operacija te zahtjeva učestalo premještanje podataka iz memorije u akumulator i rezultata iz akumulatora u memoriju. Zbog opterećenja sabirnice koja povezuje glavnu memoriju i procesor ovakvo rješenje napušteno je kod modernijih procesora.

Spremnici procesora temeljenog na stogu organizirani su na prvi unutra zadnji van (*first in last out*) principu. Operandi se stavljaju (*PUSH*) iz memorije na stog te se za potrebe operacije vade (*POP*) sa stoga obrnutim redom. Rezultat operacije pohranjuje se na stog.



Slika 2.2. *Princip stoga.*

Ovakav princip omogućava da se u okviru aritmetičke naredbe ne specificiraju operandi. Njihovo mjesto je apriori poznato na stogu kao i lokacija rezultata. Ovakvi procesori poznati su pod nazivom procesori s 0 adresa.

Procesori temeljeni na spremnicima opće namjene imaju skup spremnika opće namjene. Nasuprot procesorima temeljenim na akumulatoru i stogu svaki spremnik procesora može se koristiti bilo za pohranu operanda, rezultata i adresa. Praktički svaki suvremeni procesor ima skup spremnika opće namjene kao npr. VAX ili PPC601. Kod ovih procesora razlika je u cjelobrojnim spremnicima koji mogu sadržavati operande, rezultate i adrese i spremnika za realne brojeve koji samo mogu sadržati operande i rezultate. Ovakvo rješenje postavlja i dodatne zahtjeve na naredbe jer unutar naredbe potrebno je specificirati gdje se nalaze operandi (dvije adrese) te gdje će se pohraniti rezultat. Naravno ukoliko procesor ima samo 32 spremnika opće namjene tada se oni mogu adresirati 5 bitovnim brojem, što znači da je naredbu potrebno produžiti za 15 bita.

Ponovno je za napomenuti da današnji procesori predstavljaju mješavinu navedenih rješenja iako su temeljeni na opće-namjenskim spremnicima sadrže i neke spremnike posebne namjene. Tako npr. Motorola PPC601 ima 50 spremnika posebne namjene.

2.2. Skup naredbi procesora

Skup naredbi procesora predstavlja repertoar njegovih akcija odnosno skup naredbi koje on može protumačiti te aktivnosti koje su rezultat te naredbe. Mada se mnogi skloni razmišljanju da su ALU naredbe najznačajnije jer one izvode koristan rad, ali i naredbe koje dopremaju operande i pohranjuju u memoriju rezultate operacija te upravljaju tokom programa značajne su jednako za uspješno obavljanje sveukupnog zadatka.

Što sve mora naredba specificirati? Bez obzira na tip procesora ili na njegovu strukturu spremnika te prirodu naredbe koja se izvodi, svaka naredba mora imati sljedeće četiri specifikacije eksplicitno ili implicitno:

1. Koju operaciju je potrebno izvesti. Ova informacija je određena tzv. operacijskim kodom ili skr. *opcode*.
2. Gdje se nalazi operand ili operandi, ako uopće operacija zahtjeva operande. Operand ili operandi mogu biti u glavnoj memoriji, spremnicima procesora ili U/I međusklopu. Lokacija operanda može biti eksplicite zadana kao memorijska lokacija ili spremnik procesora ili implicite zadana kao u slučaju procesora s akumulatorom ili stogom. Tada je apriori poznato da se jedan operand nalazi u akumulatoru ili da su svi operandi na stogu.
3. Gdje upisati rezultat operacije. Kao i kod operanada, lokacija gdje se upisuje rezultat može biti eksplicite zadana (spremnik ili memorija) ili implicite poznata (akumulator ili stog).
4. Gdje se nalazi sljedeća naredba koju je potrebno izvesti. Program je niz naredbi koje se ukoliko ne dolazi do grananja izvode sekvencijalno jedna za drugom. Tako su one o pohranjene u memoriji pa ako se ne izvodi naredba za grananje procesor implicitno zna gdje se nalazi sljedeća naredba na osnovi duljine tekuće naredbe. U slučaju grananja procesor eksplicitno dobiva informaciju gdje se nalazi sljedeća naredba.

Primjer je naredba procesora Motorola MC68000 MOVE.W D4,D5 opisana u tablici:

MC 68000 Simbolički jezik	**Strojni jezik**
MOVE.W D4,D5	0011 101 000 000 100

1. Operaciju koju je potrebno izvesti je prebacivanje riječi.
2. Operand se nalazi u spremniku D4.
3. Rezultat se pohranjuje u spremnik D5
4. Sljedeća naredba se implicitno nalazi na riječi koja slijedi tekuću naredbu.

Za primijetiti je da se štedi na duljini naredbe specificiranjem sljedeće naredbe implicitno. Odavde slijedi temeljna smjernica procesora s akumulatorom i stogom, a to je zadavanje što više podataka o naredbi implicitno (kod akumulatora potrebno zadati naredbu i eventualno jedan operand, a kod stoga samo naredbu koju je potrebno izvesti). Ovakvim pristupom smanjuje se veličina i vrijeme izvođenja naredbe.

Svaki projektant računala pristupa rješavanju navedenih pitanja individualno što rezultira da praktički svaki tip procesora ima svoje strojne naredbe koje, za razliku od naredbi u višim jezicima, nisu prenosive s procesora na procesor. Ovdje se ne misli samo na različitost operacijskog koda koji zadaje aktivnost koju je potrebno izvršiti nego i simbolički opis naredbe. Tako npr. slične naredbe su **mov** i **ld**, a razlikuje se i raspored varijabli. Tako npr. kod nekih naredbi nakon simbola operacije slijedi lokacija rezultata, a kod drugih lokacija operanada. Ovo sve skupa znatno otežava posao programerima pa oni nisu spremni tako lako preći s korištenja jednog procesora na drugi. U ovom smjeru značajan napredak uveo je Intel I80x86 a i Motorola MC68x00 kod kojih je zadržana kompatibilnost koda s prethodnim generacijama.

Slijedi opis triju tipova naredbi procesora uz razmatranja kako pojedine naredbe rješavaju zadatke 2. 3. i 4. Kao što je navedeno zadaci 2. i 3. rješavaju problem smještaja operanada i rezultata, a to je i ujedno najsloženiji dio pri projektiranju naredbe. Zato će se ovom problemu posvetiti posebna pažnja, posebice putovima dohvata i pohrane podataka te modovima adresiranja.

2.2.1. Naredbe za prijenos podataka

Gdje su podaci pohranjeni u računalu? Primarno mjesto za pohranu podataka je glavna memorija. Iz glavne memorije podaci se prebacuju u spremnike procesora na obradu i time postaju sastavni dio stanja procesora (*processor state*). Obrađeni podaci iz spremnika se ponovno vraćaju u glavnu memoriju. Naravno postoje i alternativne lokacije za pohranu podataka a to su različiti ulazno/izlazni međusklopovi (tikovnica, serijski ulaz/izlaz itd.). Osim spremnika procesora koji su vidljivi programeru, kako je već rečeno postoje i spremnici za privremenu pohranu naredbi, operanada i međurezultata. Sljedeća tablica prikazuje naredbi za prebacivanje podataka različitih procesora.

Naredba	**Značenje**	**Računalo**
MOV A, B	Kopira 16 bita s memorijske lokacije A na lokaciju B	VAX 11
LDA A,Adr	Puni akumulator A sadržajem memorijske lokacije Adr	M6800
lwz R3, A	Kopira 32 bita s memorijske lokacije A u spremnik R3	PPC601
li $3, 455	U spremnik $3 upisuje se 32 bitovna cjelobrojna vrijednost 455	MIPS R3000
mov R4, dout	Kopira 16 bitovni podatak iz spremnika R4 na U/I adresu dout	DEC PDP 11
IN, AL, KBD	Kopira oktet s U/I adrse KBD u spremnik AL	Intel Pentium
LEA.L (A0), A2	Kopiraj podatak s adrese upisane u spremnik A0 (sadržaj A0) u spremnik A2	

Za primijetiti je raznolikost kako u izgledu naredbi tako i u vrsti operacije koju obavljaju: memorija →memorija, memorija → spremnik, konstanta → spremnik, spremnik → U/I, U/I → spremnik. Općenito izvor podataka može biti glavna memorija, ulazno/izlazni međusklop (npr. tipkovnica), spremnici ili konstanta koja je sastavni dio naredbe. Odredište podatka može biti samo glavna memorija, ulazno/izlazni međusklop ili spremnici. Veličina podatka

koji se prenosi je proizvoljna i može biti oktet, riječ, dvostruka riječ, realni broj, niz znakova ili pak cjeli memorijski blok. U praksi nije isključeno da je potrebno više naredbi da bi se jedan podatak prebacio iz izvora k odredištu. Npr. Motorola M6800 ograničena je na prebacivanje svega jednog okteta u jedan od akumulatora pa je potrebno dvije naredbe da bi se prebacila riječ iz memorije u dva akumulatora.

Procesori dodatno razlikuju adrese od podataka. Tako npr. LEA naredba definira adresu koja se upisuje u adresni spremnik kako bi se pristupilo podatku.

2.2.2. ALU naredbe

Aritmetička i logička jedinica skr. ALU je srce procesora jer ona izvršava obradu nad podacima. Ona izvodi aritmetičke operacije zbrajanja, oduzimanja, množenja i dijeljenja te logičke operacije I, ILI, NE, rotaciju, posmak i sl.

Vezano uz ALU naredbe postavlja se pitanje kakav pristup operandima dozvoliti. Da li dozvoliti da se operandi nalaze u glavnoj memoriji ili da su barem jedan ili oba operanda pohranjeni u spremnicima ili da je jedan od operanada sastavni dio naredbe. Da li dozvoliti da se rezultat operacije pohranjuje direktno u memoriju ili se mora pohraniti u jedan od spremnika opće namjene ili pak u implicitno zadani spremnik kao npr. kod procesora temeljenih na akumulatoru. U domeni projektanta arhitekture računala je da razriješi ove dileme i da se odluči za neko od rješenja. Tako kod VAX11 je dozvoljeno da oba operanda kao i rezultat operacije budu pohranjeni bilo u memoriji bilo u jednom od spremnika opće namjene. Noviji RISC procesori naprotiv ograničavaju da se operandi moraju nalaziti u spremnicima opće namjene tako da je prije izvođenja aritmetičke ili logičke operacije potrebno dopremiti operande iz memorije u spremnike kao i pohraniti rezultat iz spremnika u memoriju. Sljedeća tablica prikazuje neke aritmetičke i logičke operacije. I ovdje je za primijetiti šarolikost sintakse naredbi.

Naredba	**Značenje**	**Računalo**
MULF A, B,C	Množi 32 bitovne realne brojeve s memorijskih lokacija A i B i sprema rezultat na memorijsku lokaciju C	VAX 11
nabs r3, r1	Pohranjuje apsolutnu vrijednost iz spremnika r1 u spremnik r3	PPC601
ori $2, $1, 255	Izvodi logičku operaciju OR vrijednosti pohranjene u spremnik $1 s brojem 255 i rezultat pohranjuje u spremnik $2	MIPS R3000
DEC R2	Umanjuje za 1 16 bitovnu vrijednost iz spremnika R2	DEC PDP 11
SHL AX,4	Pomak u lijevo za 4 mjesta 16 bitovnu vrijednosti pohranjenu u spremniku AX	Intel 8086

2.2.3. Naredbe za grananje

U prethodnom poglavlju rečeno je da programsko brojilo, PC, pokazuje na sljedeću naredbu koju procesor mora izvesti. Tako PC upravlja tokom programa. Za vrijeme normalnog (sekvencijalnog) izvođenja programa, PC se automatski povećava za dužinu naredbe koja se izvodi kako bi pokazivao na sljedeću naredbu koju je potrebno izvesti. Promjena toka programa ili grananje na naredbu koja nije neposredno iza naredbe koja se izvodi zahtijeva proračun ciljne adrese (*target address*) odnosno adrese s koje će se

nastaviti izvođenje programa. Ciljnu adresu određuje rezultat naredbe za grananje (*branch*) ili skoka (*jump*). Ciljna adresa se zatim upisuje u PC.

Grananje može biti bezuvjetno, kao npr. C **goto** naredba, ili uvjetno kao npr. C **if** naredba. Kod bezuvjetnog grananja izvođenje se uvijek nastavlja na adresi određenoj naredbom za grananje (npr. jmp 0A10H $\Rightarrow$ upiši u PC 0A10H).

Kod uvjetnog grananja grananje će se izvršiti u ovisnosti o rezultatu određene aritmetičke ili logičke naredbe. Da bi se mogle realizirati ova naredbe aritmetičko-logička jedinica u ovisnosti o rezultatu operacije postavlja ili poništava određene zastavice. Te zastavice sastavni su dio posebnog spremnika koji se obično naziva riječ stanja procesora (*Processor Status Word*) skr. PSW ili spremnik uvjeta (*Condition Code Register*) skr. CC. Ove zastavice, bitovi, obično su nula (*zero*) koja se postavlja u 1 ako je rezultat operacije nula i negativan (*negative*) koji se postavlja ako je rezultat operacije negativan. Tako npr. C naredba

if (x<0) x = -x;

ne može se jednoznačno prevesti u niz simboličkih naredbi već to ovisi o posebnosti procesora. Kod VAX11 ova naredba prevela bi se u sljedeći simbolički kod:

```
CMP     X, 0      ; oduzima od X nula
BGE     OVER      ; uspoređuje n-zastavicu i ako je 0 grana se na adresu OVER
MNEG    X, X
OVER:
```

Sljedeća tablica pokazuje neke od naredbi za grananje za različite procesore.

Naredba	Značenje	Računalo
BLSS A; Adr	Izvođenje programa nastavlja se na adresi Adr ukoliko je bit najmanjeg značaja (LSB) s memorijske lokacija A postavljen u 1	VAX 11
bun r2	Izvođenje programa nastavlja se na adresi pohranjenoj u spremniku R2 ukoliko rezultat prethodne operacije nad realnim brojevima nije bio broj	PPC601
beq $2, $1, 32	Izvođenje programa nastavlja se na adresi (PC + 4 + 32) ukoliko je sadržaj spremnika $1 i $2 jednak	MIPS R3000
SOB R4, Petlja	Dekrementiraj sadržaj spremnika R4 i nastavi izvođenje na lokaciji Petlja ukoliko je sadržaj spremnika R4 različit od nule	DEC PDP 11
JCXZ Adr	Izvođenje programa nastavlja se na adresi Adr ukoliko je sadržaj spremnika CX različit od 0	Intel 8086

Interesantno je uočiti naredbu **SOB** koja automatski umanjuje za jedinični iznos i ispituje sadržaj spremnika. Ova naredba pogodna je za implementaciju **for**, **while** i **repeat** petlji.

2.2.4. Naredbe 4-, 3-, 2-, 1- i 0-adresa i procesor s spremnicima opće namjene

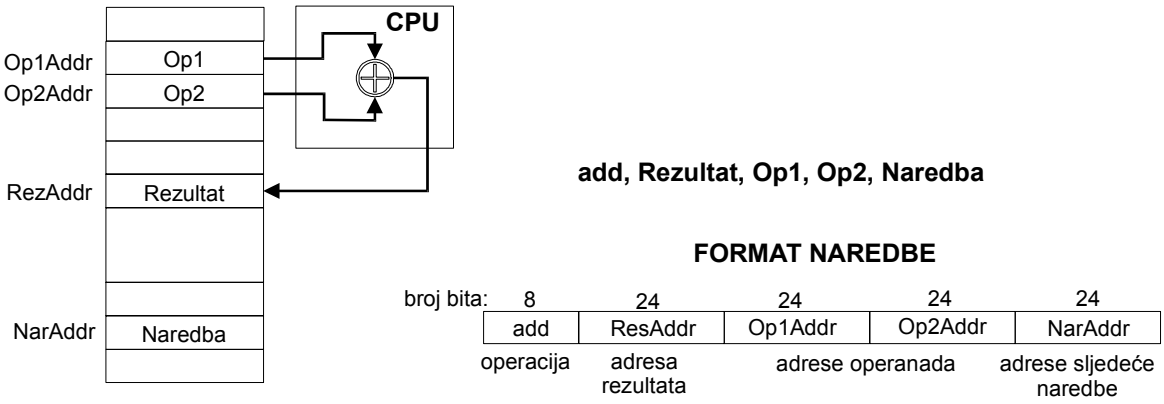
Skup naredbi koje procesor može izvoditi moraju biti jednoznačno kodirane nizom bitova da zadovolje temeljna četiri uvjeta postavljena na naredbu (koja naredba, gdje su operandi, gdje pohraniti rezultat, koja je sljedeća naredba). Projektant skupa naredbi ima za cilj minimizirati broj bita (dužinu naredbe) kojom opisuje naredbu, a da ujedno zadrži maksimalno moguću fleksibilnost u programiranju. Naravno ovo su dva oprečna zahtjeva između kojih je potrebno naći kompromis. Svakako da je najjednostavije ukoliko su sve naredbe iste dužine (npr. 32 bita) čime je krajnje pojednostavljeno određivanje adrese sljedeće naredbe. Ovakav pristup predstavlja temelj današnjih RISC procesora.

Prilikom projektiranja procesora polazi se od pretpostavke da procesor mora obaviti aritmetičku ili logičku operaciju s dva operanda. U tom smislu pet podataka mora biti specificirano:

1. operaciju koju je potrebno obaviti,
2. lokacija prvog operanda,
3. lokacija drugog operanda,
4. lokacija pohrane rezultata,
5. lokacija sljedeće naredbe.

Rješavanje problema kodiranja naredbe objasniti će se na primjeru zamišljenog procesora koji ima 24 bitovnu adresnu sabirnicu (24 bita adresira memoriju $\Rightarrow 2^{24}$ memorijskih riječi) te može izvesti 128 naredbi. Znači da je za jednoznačno kodiranje naredbi potrebno 7 bita, što se može zaokružiti na 1 oktet (8 bita).

Naredba sa četiri adrese i opcode specificira eksplicitno adrese četiri posljednja podatka (adrese dvaju operanada, rezultata i sljedeće naredbe). Slika 2.3. prikazuje kako programer vidi ovakav procesor te kakav je format naredbe.

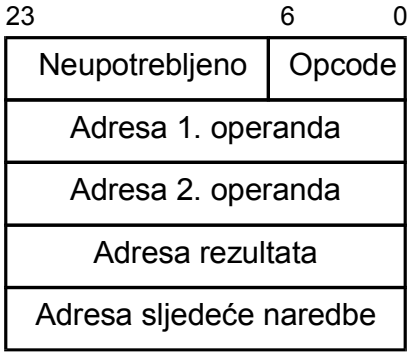


Slika 2.3. *Naredba sa četiri adrese.*

Za primijetiti je da nema spremnika procesora kojima programer može direktno pristupati. Oba operanda dohvaćaju se direktno iz memorija, a rezultat operacije pohranjuje se direktno natrag u memoriju. Adresa sljedeće naredbe eksplicitno je zadana tekućom naredbom.

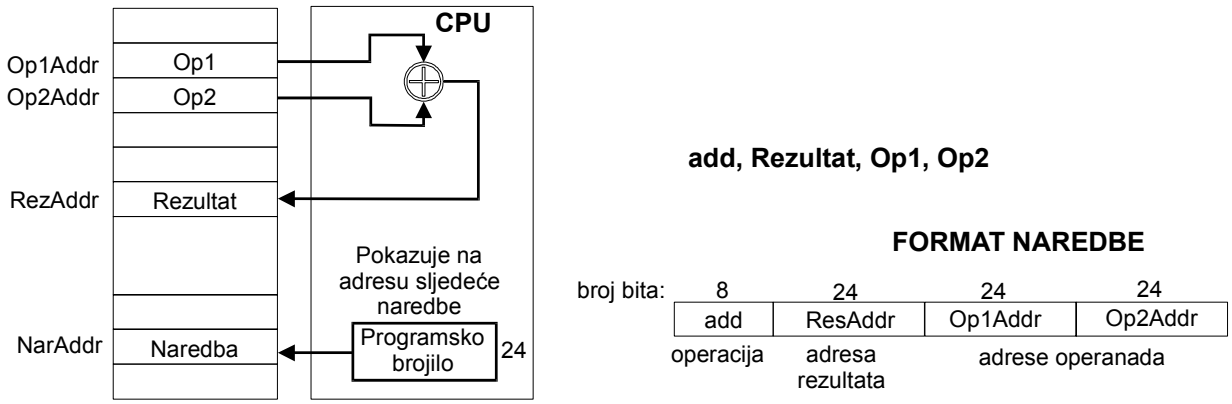
Neka je za svaku adresu potrebno 3 okteta (24 bita) pa je ukupna dužina naredbe 13 okteta. Neka procesor pristupa memoriji preko 24 bitovne sabirnice (24 bita dohvaćaju se

kao najmanja cjelina) tada je za dohvat naredbe potrebno pet puta pristupiti memoriji. Naredba je pohranjena u memoriji na sljedeći način:



Da bi se izvela naredba potreban je sljedeći broj pristupa memoriji: 5 za dohvat naredbe, 2 za dohvat operanada i 1 za pohranu rezultata. To znači da je sveukupno potrebno 8 pristupa memoriji da bi se zbrojila dva broja. Zbog velike dužine naredbe, kao i velikog broja pristupa memoriji za vrijeme izvođenja naredbe ovakva arhitektura se praktički ne koristi.

Naredba s tri adrese i opcode. Uvođenjem programskog brojila PC, kao posebnog spremnika procesora koji pokazuje na adresu sljedeće naredbe koju je potrebno izvesti, eliminira se potreba da se u okviru naredbe specificira adresa sljedeće naredbe kod svih naredbi osim kod naredbi za grananje. Sada je u nadležnosti upravljačke jedinice da izračuna duljinu tekuće naredbe te da za taj iznos uveća iznos PC-a. Naredba i dalje sadrži adrese operanada i adresu rezultata. Na slici 2.4. prikazan je format naredbe koja sadrži tri adrese.

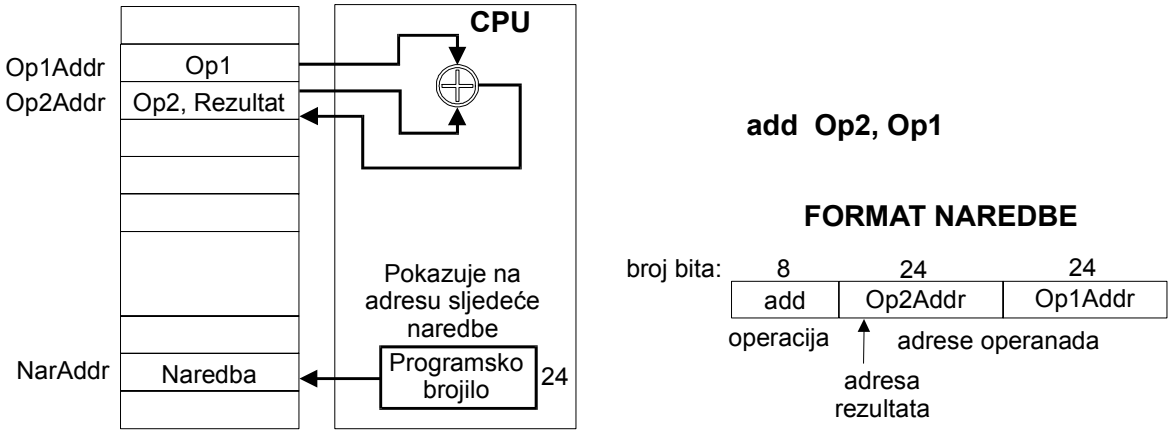


Slika 2.4. *Naredba s tri adrese.*

Slika 2.4. pokazuje da također naredbe i podaci (operandi i rezultata) mogu biti pohranjeni na različitim mjestima u memoriji. Naredba je kraća za 3 okteta i dužine je 10 okteta, a tijekom izvođenje naredbe potrebno je 7 puta pristupiti memoriji.

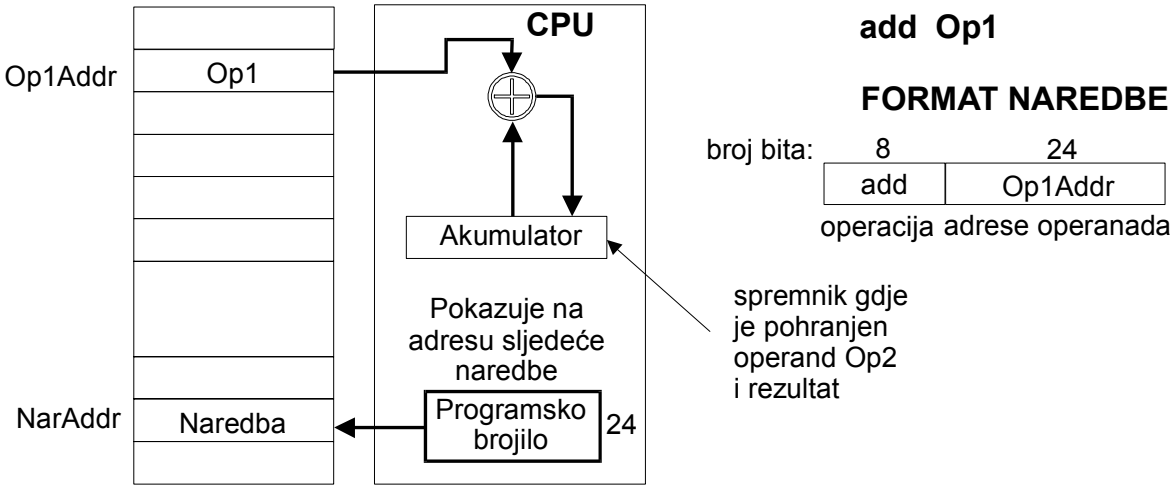
Naredba s dvije adrese i opcode. Smanjenje dužine naredbe može se postići ukoliko se rezultat operacije pohrani na mjesto drugog operanda. Ovim je potrebno unutar naredbe specificirati samo operacijski kod i adrese prvog i drugog operanda. Prelazak s naredbe s tri adrese na naredbu s dvije adrese ne zahtijeva nikakve dodatne spremnike unutar

procesora. U ovom slučaju dužina naredbe se smanjuje na 7 okteta, a izvođenje na 6 pristupa memoriji.



Slika 2.5. *Naredba s dvije adrese.*

Naredba s jednom adresom i opcode, procesor temeljen na akumulatoru. Neka se procesor proširi s jednim spremnikom posebne namjene (akumulatorom) u koji će se pohraniti jedan operand a po obavljanju operacije rezultat. Slika 2.6. prikazuje model procesora s akumulatorom kao i format naredbe za zbrajanje.



Slika 2.6. *Naredba s jednom adresom, procesor s akumulatorom.*

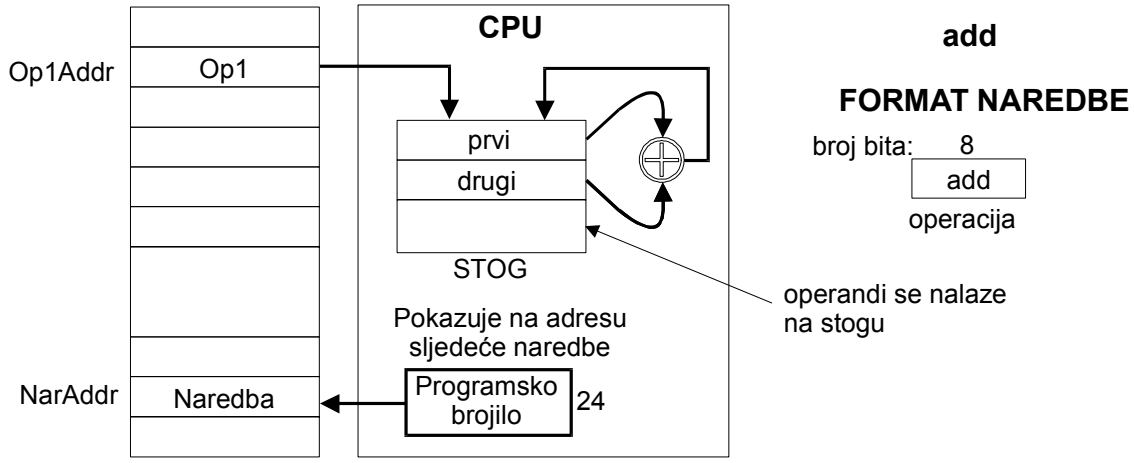
Kako procesor ima samo jedan akumulator tada ga nije potrebno eksplicite navoditi u naredbi.

Ovakav procesor zahtijeva posebne naredbe kojima se operand upisuje u akumulator (**load**), te naredba kojom se rezultat naredbe pohranjuje natrag u memoriju (**store**). Ove naredbe su također naredbe s jednom adresom. Tako bi za zbrajanje dvaju brojeva bilo potrebno izvesti sljedeći kod:

```
lda     Op2     ; u akumulator se upisuje Op2 (sadržaj adrese Op2Addr)
add     Op1     ; zbroji vrijednost u acc s Op1 (sadržajem adrese Op1Addr)
sta     Rez     ; pohrani rezultata na adresu AdrRez
```

Procesori temeljeni na akumulatoru bili su jako popularni kao prvih velikih računala i miniračunala, kao i kod prvih mikroprocesora Intel I8080 i Motorola M6800.

Naredba s nula adresa i samim opcode, procesor temeljen na stogu. Ukoliko se umjesto akumulatora uvede posebna struktura spremnika koji se pune po principu prvi unutra posljednji van, tzv. stog, tada u okviru naredbe nije potrebno specificirati niti adrese operanada niti adresu rezultata. Operandi se stavljaju na stog posebnom naredbom **push**. Aritmetička jedinica očitava operande sa stoga i stavlja rezultat operacije na stog. Kako su unaprijed poznate lokacije gdje se nalaze operandi te gdje se pohranjuje rezultat, naredba ne treba sadržavati ove podatke te je dovoljno unutar naredbe specificirati operacijski kod. Slika prikazuje procesor temeljen na stogu.



Slika 2.7. *Naredba s nula adresa, procesor temeljen na stogu.*

Za izvođenje zbrajanja dvaju brojeva potrebno je izvesti sljedeći kod

```
push    Op2     ; stavlja se Op2 na stog
push    Op1     ; stavlja se Op1 na stog
add             ; izvodi se zbrajanje prva dva podatka sa stoga
pop     Rez     ; rezultat se s vrha stoga upisuje u memoriju
```

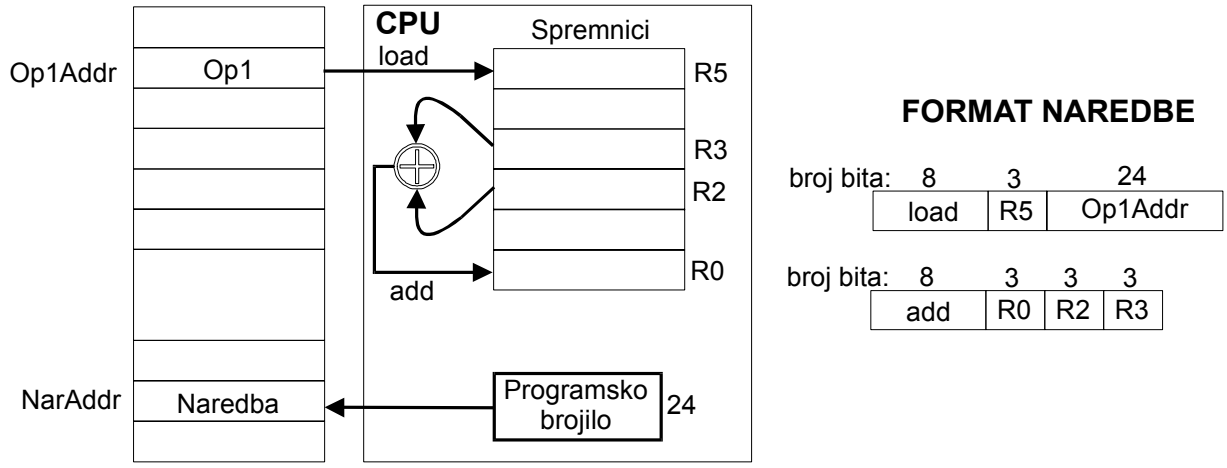
Nedostatak ovakvih procesora je što se operandi uvijek moraju nalaziti na prve dvije lokacije stoga. Sama naredba za zbrajanje ima najmanje moguću dužinu i ne zahtijeva pristup memoriji, ali za izvođenje zbrajanja potrebne su dvije **push** naredbe, **add** naredba i jedna **pop** naredba. **push** i **pop** naredba mora uz operacijski kod sadržavati i adresu operanda, odnosno adresu odredišta pa je dužine 1 oktet za opcode i 3 okteta za adresu, odnosno potrebna su dva pristupa memoriji za dohvat naredbe i jedan za dohvat operanda. Ovo razmatranje vodi zaključku da je kod ovog procesora za zbrajanje potrebno 9 pristupa memoriji i da je veličina programa 2 x 24 + 24 + 24 bita = 96 bita = 12 okteta.

Sljedeća tablica prikazuje usporedbu izvođenja izraza a = (b + c)*d – e na četiri opisane arhitekture:

3 adrese	2 adrese	1 adresa	0 adresa
add a, b, c	load a, b	load b	push b
mpy a, a, d	add a, c	add c	push c
sub a, a, e	mpy a, d	mpy d	add
	sub a, e	sub e	push d
		store a	mpy
			push e
			sub
			pop

Procesori sa spremnicima opće namjene, naredba s 1 | adresom. Procesori s akumulatorom ili stogom imaju prednost da se međurezultati mogu ostaviti u procesoru za daljnju obradu. Ovim se smanjuje broj pristupa glavnoj memoriji. Kod složenih operacije posebno je pogodno da se više međurezultata zadrži unutar procesora. Za realizaciju navedenoga procesor mora ima više spremnika za pohranu podataka. Naredba tada mora razlikovati u kojem se spremniku nalaze operandi, odnosno u koji spremnik je potrebno pohraniti rezultat operacije. Slično kao i kod pristupa memoriji svaki spremnik procesora ima svoju adresu. Kako je broj spremnika relativno mali 16, 32 ili 64 za njihovo adresiranje potrebno je svega 4, 5 ili 6 bita. Ako se uzme model naredbe s 3 adrese i umjesto pristupa glavnoj memoriji ograniči se samo na pristup spremnicima (kojih neka ima npr. 32) tada naredba mora imati 7 bita opcode, 5 bita adresa spremnika 1. operanda, 5 bita adresa spremnika 2. operand i 5 bita adresa spremnika za pohranu rezultata, odnosno za naredbu je potrebno 22 bita. Ovim pristupom zadržana je dobrim dijelom fleksibilnost u programiranju, međurezultati mogu ostati pohranjeni unutar procesora pa se umanjuje broj pristupa memoriji, a i naredba je znatno manje dužine. Ovakve "male" adrese kojima se specificiraju spremnici obično se nazivaju pola adrese pa se i ove naredbe opisuju kao naredbe s 1 | adresom.

Na slici 2.8. je prikazan primjer procesora s 8 spremnika opće namjene.



***Slika 2.8.** Naredba s 1 | adresom, procesor sa spremnicima opće namjene.*

Za primijetiti je da je naredba dužine svega 17 bita, te da joj se pristupa u jednom memorijskom ciklusu za raliku od **load** naredbe koja je dužine 35 bita pa zahtjeva dva pristupa memoriji za njen dohvat i jedan pristup za prijenos operanda.

Ovakvi procesori omogućavaju programeru najveću moguću fleksibilnost pa je praktički svaki moderniji procesor (nakon 1980 g.) projektiran po ovom principu. Ovdje se nameće pitanje zašto već za vrijeme procesora s akumulatorom nije se odmah prešlo na koncept sa spremnicima opće namjene. Razlog je u tome što je tada cijena jednog bita iznosila preko 25$ pa je s ekonomskog aspekta bilo neprihvatljivo imati više spremnika.

Provedena razmatranja odnose se na hipotetske procesora. Praktično procesori koji koriste naredbe s četiri adrese ne postoje. Većina procesora koristi neku od kombinacija preostalih modela. Tako npr. VAX11 praktički koristi sve preostale tipove naredbi s 3-, 2-, 1, 0 i 1 | adrese.

Suvremeni procesori, naročito RISC temeljeni su na naredbama s 1 | adrese. Sve aritmetičke i logičke naredbe kao i grananja koriste isključivo spremnike, dok je pristup memoriji ograničen **load** i **store** naredbama. Ovakvi procesori obično se nazivaju spremnik – spremnik procesori (*register to register machine*). Osnova ideje je da je prebacivanje podataka između memorije i procesora skupa operacija te je potrebno destimulirati limitirajući ovakvu operaciju na svega dvije naredbe.

Drugi tip procesora su tzv. spremnik – memorija procesori (*register – memory machine*) kod kojih se operandi i rezultati pohranjuju bilo u spremnike ili u memoriju. Oni se obično opisuju kao procesori s s naredbama s 1 ili 1 | adrese, jer jedan operand ili rezultat obavezno su pohranjeni u akumulatoru ili spremniku opće namjene.

Treći tip procesora su memorija – memorija procesori (*memory to memory machine*) koji dozvoljavaju da oba operanda i rezultat budu pohranjeni u memoriji. Oni se opisuju kao procesori s naredbama s 2 ili 3 adrese.

Kompromisi između skupa naredbi i strukture spremnika procesora

Prilikom odabira skupa naredbi kao i strukture spremnika procesora kompromis je između fleksibilnosti u pohrani operanada i rezultata operacija nasuprot količini informacije koju mora naredba sadržavati kako bi se uspješno obavila zadana operacija.

- Naredbe s 3 adrese omogućavaju najkraću dužinu koda za realizaciju određene obrade ali zahtijevaju nepotrebno veliki broj bita po naredbi.
- Naredbe s 0 adresa zahtijevaju najduži kod za realizaciju određene obrade ali imaju najkraću dužinu naredbe.
- Čak i kod procesora koji koriste naredbe s 0 adresa moraju postojati i naredbe s 1 adresom **push** i **pop**.
- Procesori sa spremnicima opće namjene mijenjaju način pristupa operandima i rezultatu na način da se specificiraju kratke (5 ili 6 bita) adrese jednog od spremnika opće namjene. Kao jedna od modifikacija ovakvog tipa procesora je korištenje naredbi koje specificiraju dva spremnika (dvije kratke adrese) i jednu memorijsku adresu (duga adresa). Ovo je modifikacija naredbe s 3 adrese.

- Procesori temeljeni na principu spremnik – spremnik koriste **load** i **store** naredbe za pristup glavnoj memoriji. Sve ostale naredbe imaju pristup samo spremnicima opće namjene.
- Današnja tehnologija je na takovoj razini da je pristup spremnicima znatno brži od pristupa glavnoj memoriji. To je dodatni razlog da se favoriziraju procesori sa spremnicima opće namjene i kratkim naredbama.

2.2.5. Način pristupa operandima, modovi adresiranja

Proces obrade podataka obuhvaća kontinuiran prijenos naredbi i podataka iz glavne memorije u procesor i pohranu rezultata obrade iz procesora u glavnu memoriju. Današnji programi obrađuju različite tipove i strukture podataka. Adresa podatka može biti poznata za vrijeme prevođenja programa, tzv. **varijabla** ili je njenu adresu potrebno proračunavati tijekom izvođenja tzv. **pokazivač**. Navedeno postavlja poseban zahtjev na projektanta arhitekture računala da razvije različite sofisticirane načine pristupa operandima u glavnoj memoriji i spremnicima, tzv. **adresne modove** (*addressing modes*).

Procesor pristupa operandima u glavnoj memoriji na način da generira adresu lokacije na kojoj je ili operand pohranjen ili se namjerava upisati. Ovo se naziva **efektivna adresa**. Ta adresa može se dobit na različite načine od direktnog preuzimanja adrese koja je sastavni dio naredbe do složenijeg postupka njenog proračuna. Neki tipični adresni modovi prikazani su na slici 2.9.

a) **Neposredno adresiranje**, (*immediate addressing*) koristi se da bi se pristupilo konstanti koja je sastavni dio naredbe. Koristi se za dobavu operanda naredbe, a ne može se koristiti za pohranu rezultata.

b) **Direktno adresiranje** (*direct addressing*) je kada je adresa operanda sastavni dio naredbe. Ovaj način adresiranja može se koristiti i za dohvat operanda i pohranu rezultata.

c) **Indirektno adresiranje** (*indirect addressing*), konstanta koja je sastavni dio naredbe je adresa memorijske lokacije na kojoj je upisana adresa memorijske lokacije s koje se operand namjerava očitati ili na koju se rezultat želi upisati. Ovim modom implementiraju se pokazivači. Za manipulaciju s podatkom potrebna su dva pristupa memoriji, prvo dohvat pokazivača tj. adrese gdje je pohranjen podatak s kojim se manipulira, a zatim dohvat operanda ili upis rezultata u memoriju.

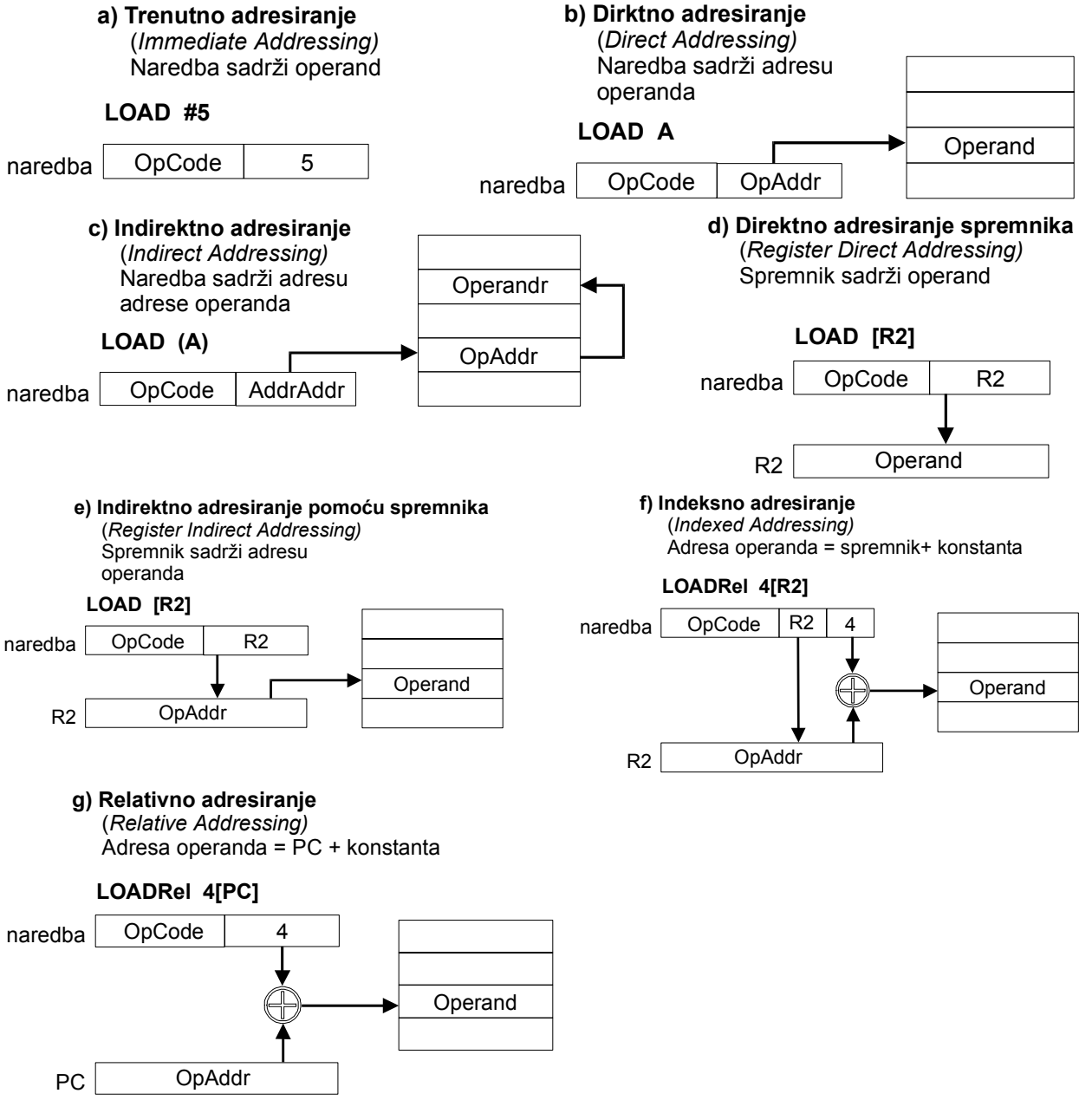
d) **Direktno adresiranje spremnika** (*register direct mode*), podatak je upisan ili se upisuje u spremnik čija adresa je sastavni dio naredbe.

e) **Indirektno adresiranje pomoću spremnika** (*register indirect mode*), adresa operanda pohranjena je u spremniku čija adresa je sastavni dio naredbe. Ovaj adresni mod koristi se obično za sekvencijalni pristup elementima polja pohranjenim u memoriji. Početna adresa polja pohranjuje se u spremnik te se nakon svakog pristupa elementu polja sadržaj spremnika poveća za dužinu elementa.

f) **Indeksno adresiranje** (*indexed mode*). Adresa se dobiva zbrajanjem konstante koja je obično sastavni dio naredbe i sadržaja spremnika adresiranog naredbom. Konstanta koja je sastavni dio naredbe obično se naziva **baza**, a sadržaj spremnika **pomak** ili

offset. To je razlog da se ponekad ovaj način adresiranja naziva i **posmačno** ili **bazno adresiranje**.

g) **Relativno adresiranje** (*relative addressing mode*) je sličan indeksnom adresiranju uz razliku da se konstanti (bazi) koja je sastavni dio naredbe pridodaje sadržaj programskog brojila te sa na taj način formira efektivna adresa.



Slika 2.9. *Različiti adresni modovi.*

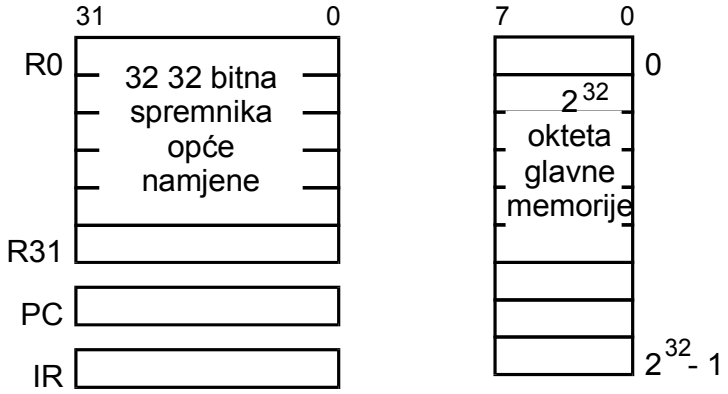
Ova razmatranja ukazuju na složenost adresnih modova. Projektant arhitekture kao i projektant logičkih sklopova imaju težak zadatak projektirati i implementirati opisane modove.

2.3. Neformalan opis jednostavnog RISC-a, SRC-a

U sljedećem poglavlju neformalno (tekstualno i grafički) opisati će se jednostavan procesor sa smanjenim skupom naredbi, SRC-a. Ova razmatranja poslužiti će kao podloga za njegov formalan opis pomoću kojeg je moguće i projektirati i realizirati jedan takav procesor.

2.3.1. Struktura spremnika i memorije

Slika 2.10. prikazuje kako programer vidi računalo, tzv. programerski model računala.

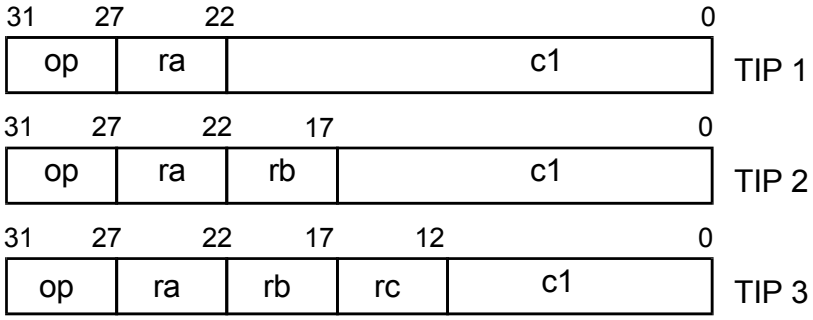


Slika 2.10. *Programerski model računala.*

Jednostavan procesor sa smanjenim skupom naredbi ima 32 32 bitovna spremnika opće namjene, programsko brojilo PC i spremnik naredbe IR. Memorija je kapaciteta 2^{32} okteta, ali procesor u jednom pristupu memoriji čita ili upisuje četiri okteta ili 32 bita. Prvi oktet, tj. oktet s najmanjom adresom sadrži oktet najvećeg značaja dok onaj oktet s najvećom adreso oktet najmanjeg značaja. Ovaj procesor organiziran je kao procesor sa spremnicima opće namjene tako da se memoriji može pristupati samo preko naredbi **load** i **store**. Efektivna memorijska adresa je dužine 32 bita.

2.3.2. Format naredbe

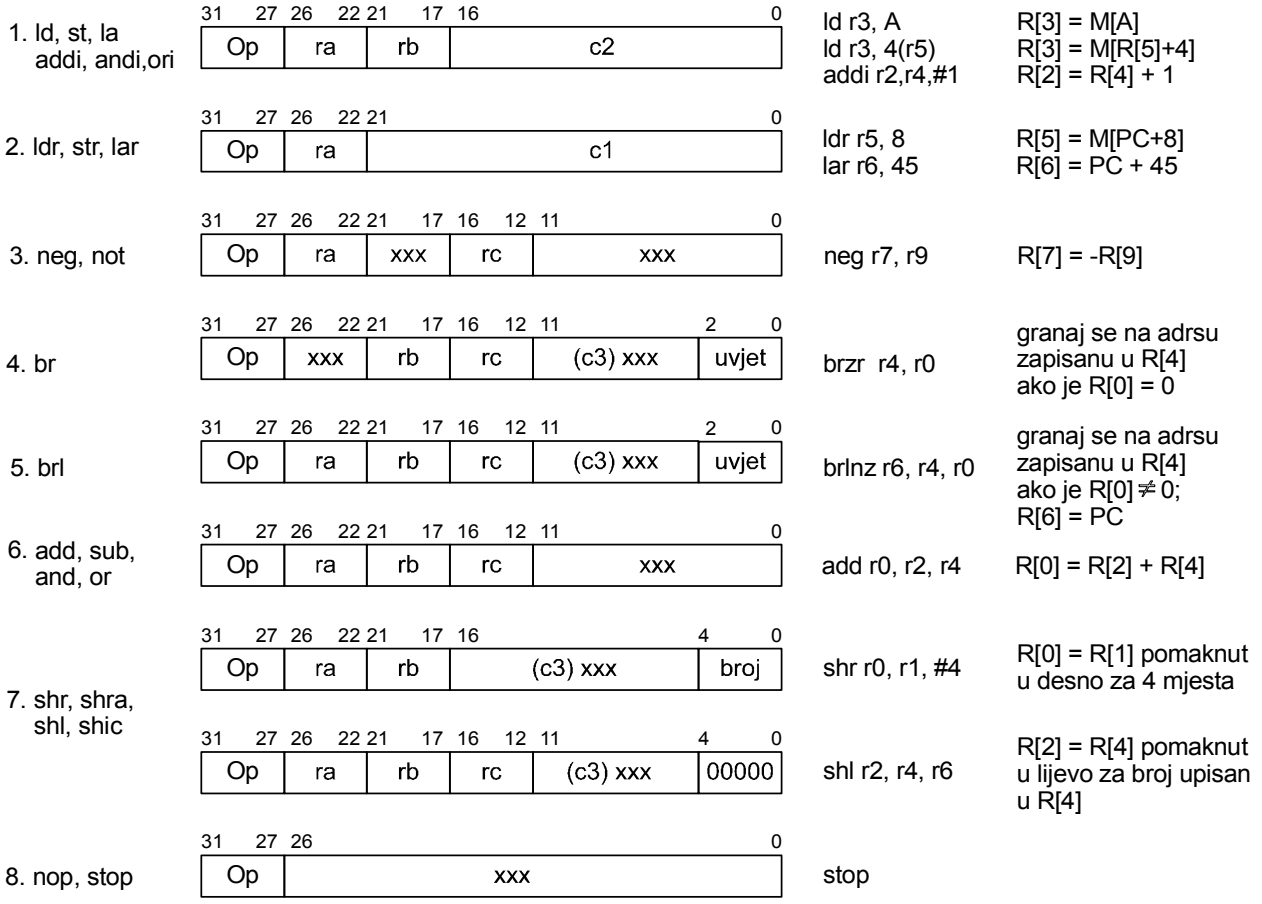
U osnovi se razlikuju tri tipa naredbi: TIP 1 koji uz operacijski kod ima specificiran jedan spremnik i konstantu, TIP 2 koji uz operacijski kod ima specificirana dva spremnika i konstantu, TIP 3 koji uz operacijski kod ima specificirana tri spremnika i konstantu. Pretpostavlja se da će procesor izvoditi svega 32 naredbe pa je za operacijski kod potrebo 5 bitova kao i za specifikaciju spremnika kojih je 32 (slika 2.11.).



Slika 2.11. *Formati naredbi.*

U spremnik ra pohranjuje se rezultat operacije dok je operand odnosno operandi smješteni u spremnicima rb i rc.

Slika 2.12. prikazuje 21 naredbu (koju može izvesti hipotetički SRC) u 7 različitih formata koji se izvode iz gore navedenih osnovnih formata.



Slika 2.12. *Formati naredbi SRC-a.*

- Četiri naredbe za upis u spremnike (**load**) – **ld**, **ldr**, **la** i **lar**, te dvije naredbe za upis podataka iz spremnika u glavnu memoriju **(store**): **st** i **str**.
- Dvije naredbe za grananje **br** i **brl** koje omogućavaju bezuvjetno i uvjetno grananje na adresu određenu specificiranim spremnikom. Kod uvjetnog grananja u ovisnosti o tro-bitovnom uvjetu uspoređuje se sadržaj spremnika rc da li je = 0, $\neq$ 0, $\geq$ 0 i <0 te se u ovisnosti o rezultatu ispitivanja grana na adresu određenu spremnikom rb. Kod naredbe **brl** ujedno se i sadržaj programskog brojila upisuje u spremnik ra. Ovo je interesantno kod realizacije potprograma kako bi se po izvršenju potprograma moglo vratiti u glavni program.
- Aritmetičke naredbe. Postoje četiri aritmetičke naredbe: **add**, **addi**, **sub** i **neg**. Sve ove naredbe osim **addi** zahtijevaju dva operanda, sadržaj spremnika rb i rc, te rezultat pohranjuju u spremnik ra. Naredba **addi** pribraja konstantu c2 koja je sastavni dio naredbe sadržaju spremnika rb i rezultat pohranjuje u spremnik ra.

- Logičke naredbe i posmaci. Postoji devet ovakvih naredbi: **and**, **andi**, **or**, **ori**, **not**, **shr**, **sha**, **shl** i **shc**. Logičke naredbe **and**, **andi**, **or** i **ori** imaju sličan oblik kao i aritmetičke naredbe **add** i **addi**. Naredbe za posmak pomiču sadržaj spremnika rb za ili iznos koji je sastavni dio naredbe (bitovi 0 – 4) ili za iznos upisan u spremniku rc i rezultat pohranjuju u spremnik ra.
- Dvije posebne naredbe bez operanada su **nop** i **stop**.

Sve naredbe su dužine 32 bita. Projektirani SRC može obrađivati samo operande pohranjene u spremnicima, a memoriji se pristupa preko **load** i **store** naredbi. Sve naredbe imaju pet bita najvećeg značaja rezervirane za operacijski kod naredbe. Na ovaj način moguće je kodirati 32 naredbe. SRC ima samo 21 naredbu. Polja ra, rb i rc su dužine po 5 bita i specificiraju po jedan od 32 32 bitovna spremnika opće namjene. Konstante c1, c2, c3, uvjet i broj koriste se na način koji će biti naknadno opisan kako se budu objašnjavale pojedine naredbe.

Za primijetiti je da neke naredbe imaju neka neiskorištena polja. Ovo se može smatrati kao loše korištenje memorije. Ali ipak ovo je kompromis uobičajen kod modernih RISC procesora između iskoristivosti memorije i prednosti iste dužine svih naredbi.

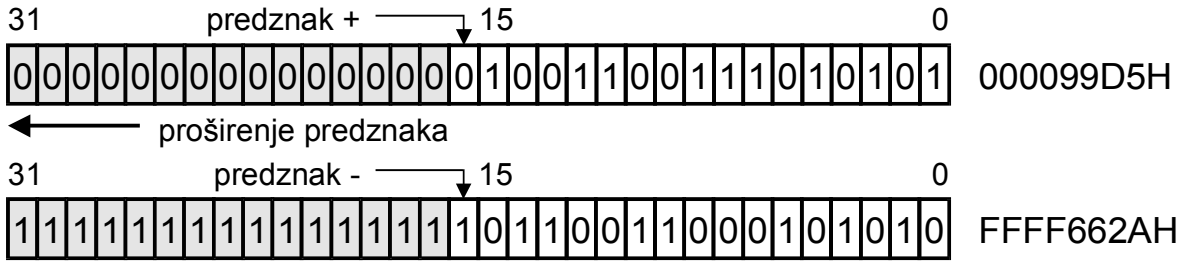
2.2.3. Adresiranje memorije: Naredbe za pristup memoriji load *i* store

Memoriji se može pristupiti samo pomoću naredbi **load** i **store**. SRC koristi sljedeće naredbe za pristup memoriji:

```
ld   ra,  c2        ; direktno adresiranje R[ra] = M[c2]
ld   ra,  c2(rb)    ; indeksno adresiranje (rb ≠ 0); R[ra] = M[c2 + R[rb]]
st   ra,  c2        ; direktno adresiranje M[c2] = R[ra]
st   ra,  c2(rb)    ; indeksno adresiranje (rb ≠ 0); M[c2 + R[rb]] = R[ra]
la   ra,  c2        ; upiši vrijednost adrese u spremnik R[ra] = c2
la   ra,  c2(rb)    ; upiši vrijednost adrese u spremnik R[ra] = c2+ R[rb]
```

Ove naredbe koriste format br.1 s prethodne slike. Spremnik u koji se pohranjuje rezultat specificiran je 5 bitovnim poljem ra, a adresa sa 17 bitovnom konstantom u c2 polju. Polje rb ima dvojako značenje. Ako je rb = 0, odnosno svih 5 bita ovog polja su 0 te specificirajući spremnik r0, upravljačka jedinica procesora ovo tumači na način da je polje c2 ili konstanta koju treba upisati u spremnik ili adresa memorijska lokacije čiji sadržaj je potrebno upisati u spremnik. Tada od 17 bita konstante c2 16 bita predstavlja njenu vrijednost a 17 bit predznak, odnosno konstanta c2 je cjelobrojna veličina prikazana u dvostrukom komplementu.

Ako je polje rb $\neq$ 0 i specificira neki od preostalih 31 spremnika opće namjene tada se adresa varijable koju je potrebno upisati u spremnik, odnosno vrijednost koju se direktno upisuje u spremnik dobiva zbrajanjem konstante c2 (2'ki komplement s proširenjem predznaka) i sadržaja spremnika rb. Konstanta c2 može poprimiti bilo koju cjelobrojnu vrijednost iz intervala -2^{16} do $2^{16} - 1$, tj –65536 do 65535.



Slika 2.13. *Postupak proširenja predznaka.*

$000099D5 = 5 + 13 \cdot 16 + 9 \cdot 16^2 + 9 \cdot 16^3 = 39381$
$FFFF662A = -(000099D5 + 1) = -39382$
2'ki komplement = $-(FFFF662A + 1) \Rightarrow FFFF662A = 00099D5$ (zamjena $1 \rightarrow 0$ i $0 \rightarrow 1$)

Tako naredba **ld** upisuje u spremnik ra sadržaj memorijske lokacije na koju pokazuje konstanta c2 ako je rb = 0 ili c2 + R[rb] ako je rb $\neq$ 0, odnosno sadržaj spremnika ra na memorijsku lokaciju na koju pokazuje konstanta c2 ako je rb = 0 ili c2 + R[rb] ako je rb $\neq$ 0 kod **st** naredbe.

Naredba **la** (*load address*) upisuje u spremnik ra vrijednost konstante c2 ako je rb = 0, odnosno c2 + R[rb] ako je rb $\neq$ 0.

Za navedene naredbe potrebno je napraviti nekoliko napomena. Prvo, kako je konstanta c2 16 bitovna (17 bit predznak) tako je direktno moguće adresirati samo prvih 64k (00000000H - 0000FFFFH), odnosno posljednjih 64k memorije (FFFF0000H – FFFFFFFFH). Kod naredbe **la** moguće je direktno u spremnik upisati samo konstantu od (-65356 do 65355). Za adresiranje preostalog memorijskog prostora kao i unos konstanti izvan navedenog intervala potrebno je koristiti indeksno adresiranje, odnosno pribrajanje sadržaja jednog spremnika ($\neq$ R[0]) konstanti.

Relativno adresiranje proračunava efektivnu adresu pribrajajući konstanti sadržaj PCa.

ldr	ra,	c1	; upis u spremnik s adrese PC + c1	R[ra] = M[PC + c1]
str	ra,	c1	; upis na adresu PC + c1 iz spremnika	M[PC + c1] = R[ra]
lar	ra,	c1	; upiši vrijednost relativne adrese u spremnik	R[ra] = PC + c1

Efektivna adresa izračunava se za vrijeme izvođenja programa pribrajanjem sadržaja programskog brojila konstanti c1, PC + c1. Ovaj mod adresiranja omogućava da kod programa, kao i podaci budu *realokatibilni*. Naredbe i podaci specificiraju se kao konstantna udaljenost od sadržaja PC. Tako program i podaci mogu se premjestiti kao cjelina bilo gdje u memoriji bez ikakvih promjena u kodu. Samo je potrebno pri početku izvođenja programa u PC upisati početnu vrijednost koda. Kako je konstanta koja određuje pomak dužine 22 bita (21 bit vrijednost i 1 bit predznak) tako je moguće specificirati naredbe i podatke unutar $\pm 2^{21}$ oko trenutne vrijednosti programskog brojila.

Sljedeća tablica prikazuje neke od naredbi za prebacivanje podataka, kao i njima pripadajući kod.

Naredba	Op	ra	rb	c1	Značenje	Mod adresiranja
ld r1,32	1	1	0	32	R[1]←M[32]	Direktno
ld r22, 24(r4)	1	22	4	24	R[22]←M[24+R[4]]	Indeksno
st r4, 0(r9)	3	4	9	0	M[R[9]]←R[4]	Indirektno spremnikom
la r7, 32	5	7	0	32	R[7]←32	Trenutno
ldr r12, -48	2	12	-	-48	R[12]←M[PC-48]	Relativno
lar r3, 0	6	3	-	0	R[3]←PC	Adresa u spremnik

Primjer: **ld** r22, 24(r4)

op=1 ra=22 rb=4 c1=24

00001 10110 00100 00000000000011000 = 0D880018H

2.3.4. Aritmetičke i logičke naredbe

Ove naredbe koriste aritmetičku i logičku jedinicu procesora kako bi se provela obrada nad operandima. Prvo će biti opisane naredbe koje koriste samo jedan operand.

Naredbe koje imaju samo jedan operand

neg ra, rc ; negiraj tj. 2'ki komplement R[ra] = -R[rc]
not ra, rc ; NE logička operacija R[ra] = $\overline{R[rc]}$

Ove naredbe koriste format broj 3 sa slike formata naredbi a imaju samo jedan operand pa tako je potrebno specificirati samo jedan spremnik za operanda, rc, i jedan spremnik za rezultat. Naredba za proračun negativne vrijednosti broja **neg** (op = 15) određuje 2'ki komplement od vrijednosti upisane u spremnik ra i upisuje je u spremnik rc. Naredba za logičku negaciju **not** (op = 24) zamjenjuje za operand upisan u spremnik rc nule jedinicama i jedinice nulama i rezultat pohranjuje u spremnik ra. Preostala polja naredbe ostaju neiskorištena.

Sljedeće naredbe specificiraju dvije varijable: **add** (op = 12), **sub** (op = 14), **and** (op = 20) i **or** (op = 22). Oba operanda i rezultat nalaze se u spremnicima opće namjene. Ove naredbe koriste format broj 6 sa slike formata naredbi. Dvanaest bita najmanjeg značaja ostaju neiskorišteni jer su varijable (operandi) pohranjeni u spremnicima opće namjene.

Naredbe koje koriste tri spremnika

add ra, rb rc ; zbrajanje u aritmetici 2'ki komplement R[ra] = R[rb]+R[rc]
sub ra, rb rc ; oduzimanje u aritmetici 2'ki komplement R[ra] = R[rb]-R[rc]
and ra, rb rc ; logičko I R[ra] = R[rb]$\wedge$R[rc]
or ra, rb rc ; logičko ILI R[ra] = R[rb]$\vee$R[rc]

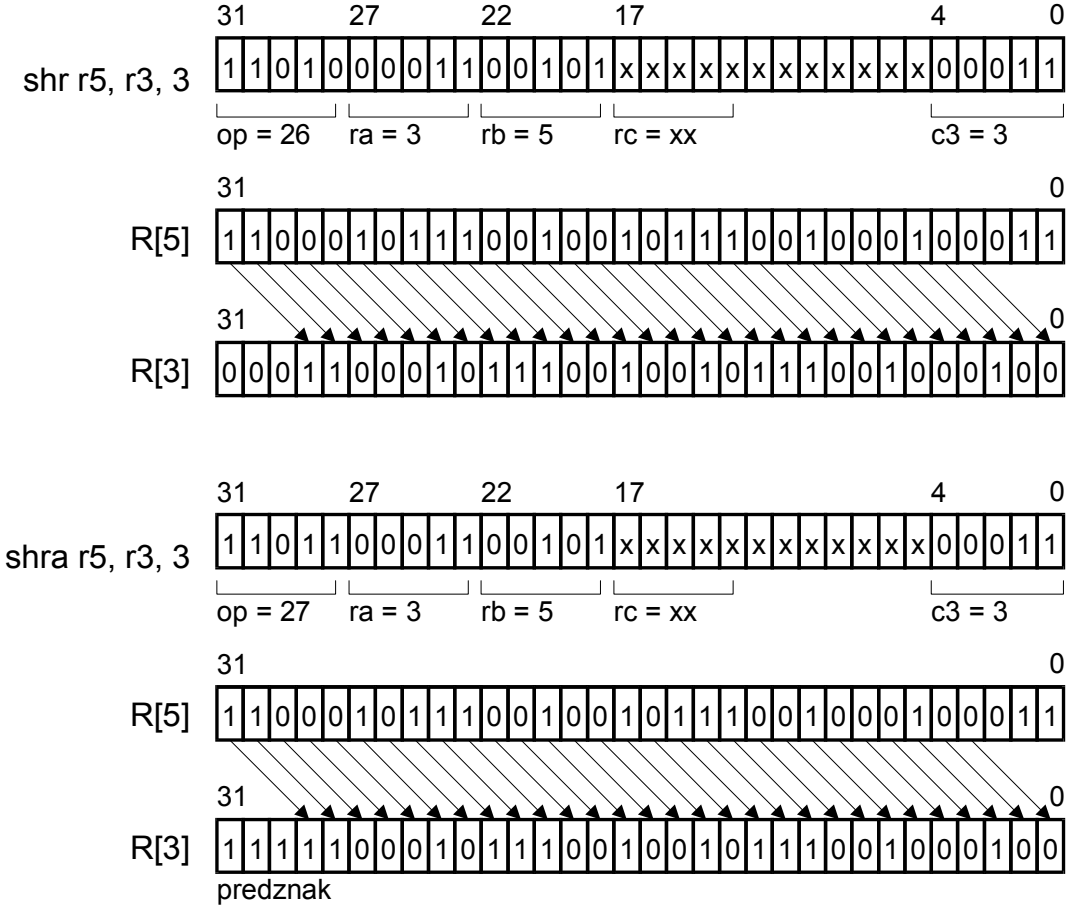
Tri ALU naredbe koriste neposredno adresiranje, odnosno jedan operand je konstanta koja je sastavni dio naredbe. To su naredbe: **addi** (op = 13), **andi** (op = 21) i **ori** (op = 23). Konstanta je upisana u polje c2 i dužine je 17 bita (16 bita vrijednost i 1 bit predznak). Ove naredbe koriste format broj 1 sa slike formata naredbi.

Naredbe koje koriste dva spremnika i konstantu kao dio naredbe

addi	ra,	rb	c2	; zbrajanje u aritmetici 2'ki komplement	R[ra] = R[rb]+c2
andi	ra,	rb	c2	; logičko I	R[ra] = R[rb]$\wedge$c2
ori	ra,	rb	c2	; logičko ILI	R[ra] = R[rb]$\vee$c2

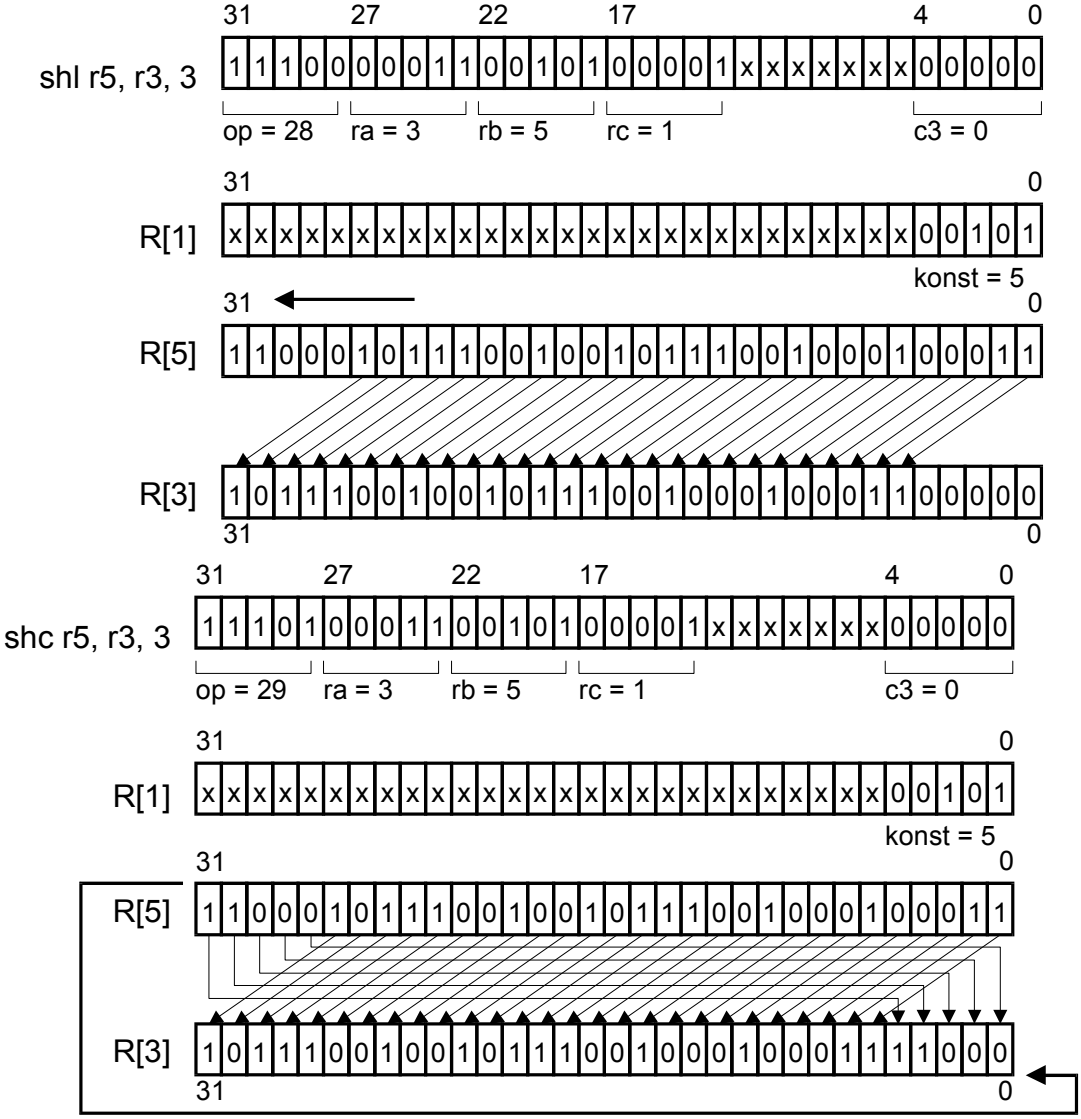
Naredbe za posmak izvode pomak bitova spremnika rb u lijevo, desno ili kružno za 1 do 31 mjesto i rezultat pohrannjuju u spremnik ra. Za koliko mjesta je potrebno izvršiti posmak određeno je s pet bita najmanjeg značaja naredbe (format naredbe 7a) odnosno ukoliko su bitovi naredbe 0-4 postavljeni u 0 tada broj je mjesta za koji je potrebno izvršiti pomak određen s pet bita najmanjeg značaja spremnika rc (format naredbe 7b).

Postoje dva oblika posmaka u desno, **shr** (op = 26) i **shra** (op = 27). Prvi oblik ubacuje nule na upražnjena mjesta od lijeva na desno, a drugi oblik upisuje na ista mjesta predznak izvorne vrijednosti tj. 0 za pozitivni broj i 1 za negativni. Drugim oblikom naredbe očuvan je predznak izvorne vrijednosti. Na slici 2.14. prikazana su primjeri ove dvije naredbe.



Slika 2.14. *Primjeri naredbi za posmak.*

Za posmak u lijevo postoji samo jedna naredba **shl** (op = 28) koja unosi nule na upražnjena mjesta s desna na lijevo. Cirkularni posmak **shc** (op = 29) pomiče bit najvećeg značaja na mjesto bita najmanjeg značaja i taj postupak ponavlja za onaj broj puta koliko specificira ili konstanta u sastavu naredbe ili operand u sastavu spremnika rc. (slika 2.15.).



Slika 2.15. *Primjer naredbi za cirkularni posmak.*

Naredbe za posmak:

shr ra, rb, rc ; posmak u desno R[rb] za iznos R[rc] (bitovi 0-4) a rezultat u R[ra]
shr ra, rb, *broj* ; posmak u desno R[rb] za *broj* (bitovi 0-4 naredbe) a rezultat u R[ra]
shra ra, rb, rc ; ar. posmak u desno R[rb] za iznos R[rc] (bitovi 0-4) a rez. u R[ra]
shra ra, rb, *broj* ; ar. posmak u desno R[rb] za *broj* (bitovi 0-4 naredbe) a rez. u R[ra]
shl ra, rb, rc ; posmak u lijevo R[rb] za iznos R[rc] (bitovi 0-4) a rezultat u R[ra]
shl ra, rb, *broj* ; posmak u lijeco R[rb] za *broj* (bitovi 0-4 naredbe) a rezultat u R[ra]
shc ra, rb, rc ; cirk. posmak u lijevo R[rb] za iznos R[rc] (bitovi 0-4) a rez. u R[ra]
shc ra, rb, *broj* ; cirk. posmak u lijeco R[rb] za *broj* (bitovi 0-4 naredbe) a rez. u R[ra]

Sve naredbe za posmak koriste format broj 7 sa slike 2.12. Ukoliko je polje naredbe *broj* $\neq$ 0 tada ono određuje za koliko mjesta će se posmak izvršiti, a ukoliko je *broj* = 0 tada 5 bita najmanjeg značaja spremnika rc određuje broj mjesta za koji će se posmak izvršiti.

2.3.5. Naredbe za grananje

Naredbe za grananje **br** (op = 8) i **brl** (op = 9) dekodiraju se koristeći format broj 4 i 5. Format broj 4 koristi se za naredbu za grananje **br** i izvodi zamjenu sadržaja PCa sadržajem spremnika rb, dok format broj 5 je za naredbu za grananje i vezivanje (*branch & link*) **brl** koja prvobitno pohranjuje sadržaj Pca u jedan spremnik (ra) a zatim zamjenjuje njegov sadržaj sadržajem spremnika rb. Ove aktivnosti izvode se samo ukoliko sadržaj spremnika rc zadovoljava uvjet određen s prva tri bita naredbe (bitovi 0-2).

Naredba za grananje i vezivanje koristi se za skokove u potprograme. Naime tom naredbom sačuvan je sadržaj PCa prije skoka pa je moguće nakon završetka potprograma ponovo se vratiti na sljedeću naredbu glavnog programa.

Postoji pet različitih uvjeta koji se ispituju prije skoka. Oni su određeni bitovima 0-2 naredbe, a njihovo značenje opisano je sljedećom tablicom:

naredba	c3<2..0>	uvjet grananja
brnv, brlnv	0	nikada
br, brl	1	uvijek (bezuvjetno)
brzr, brlzr	2	ako je R[rc] = 0
brnz, brlnz	3	ako je R[rc] $\neq$ 0
brpl, brlpl	4	ako je R[rc] $\geq$ 0, bit 31 R[rc] = 0 pozitivan
brmi, brlmi	5	ako je R[rc] < 0, bit 31 R[rc] = 1 negativan

U prethodnim razmatranjima spomenuto je da mnogi procesori u svrhu grananja imaju poseban spremnik (statusni spremnik) čiji pojedini bitovi (zastavice) koriste naredbe za grananje. SRC koji se ovdje razmatra na primjenjuje takvu koncepciju već ALU direktno uspoređuje sadržaj spremnika rc prema postavljenim uvjetima.

Sljedeća tablica prikazuje primjere nekih naredbi za grananje:

Naredba	Primjer	Značenje	Op	ra	rb	rc	c3<2..0>	Uvjet
brlnv	brlnv r6	R[6]←PC	9	6	-	-	000	nikada
br	br r4	PC←R[4]	8	-	4	-	001	uvijek
brl	brl r6,r4	R[6]←PC PC←R[4]	9	6	4	-	001	uvijek
brzr	brzr r5,r1	ako je R[1]=0 PC←R[5]	8	-	5	1	010	nula
brlzr	brzr r7,r5,r1	R[7]←PC ako je R[1]=0 PC←R[5]	9	7	5	1	010	nula
brnz	brnz r1,r0	ako je R[0]≠0 PC←R[1]	8	-	1	0	011	nije nula
brlnz	brnz r2,r1,r0	R[2]←PC ako je R[0]≠0 PC←R[1]	9	2	1	0	011	nije nula
brpl	brpl r3,r2	ako je R[2]>=0 PC←R[3]	8	-	3	2	100	plus
brlpl	brzr r4,r3,r2	R[4]←PC ako je R[2]>=0 PC←R[3]	9	4	3	2	100	plus

brmi	brmi r0,r1	ako je R[1]<0 PC←R[0]	8	-	0	1	101	minus
brlmi	brmi r3,r0,r1	R[3]←PC ako je R[1]<0 PC←R[0]	9	3	0	1	101	minus

2.3.6. Posebne naredbe

Uz spomenute naredbe koriste se još dvije posebne naredbe **nop** (op = 0) i **stop** (op = 31). Naredba **nop** ne izvodi nikakvu operaciju nego se koristi za razdjeljivanje nekih naredbi ili za potrošnju vremena procesora što je važno kod implementacije cjevovoda što će biti naknadno opisano. Naredba **stop** zaustavlja rad procesora i korisna je kod testiranja programa (programa *debugger*). Na kritičnim mjestima programa umetne se **stop** naredba te se ispituje stanje procesora i memorije.

Primjer: Prevedite dio koda napisanog u Cu u simbolički jezik SRCa:

```
#define  konst    125
if  (x<0)  x = -x;
```

Simbolički dio koda izgledao bi na sljedeći način:

```
          konst   .equ    125
                  .org    1000
1000      x:      .dw     1
                  .org    5000
5000              lar     r0, skok
5001              ld      r1, x
5002              brpl    r0, r1
5003              neg     r1, r1
5004      skok:   ...
```

Simbolički jezik uz naredbe (**add**, **lar**, **ld**, itd.) ima i tzv. upute ili direktive (*statements*). Tako .equ pridružuje simboličkoj konstanti vrijednost. .org određuje poziciju podataka i programa u memoriji. Tako u primjeru druga linija programa .org 1000 određuje da konstanta, simbolički nazvana x se nalazi na lokaciji 1000 dek. .dw rezervira za varijablu x 1 dvostruku riječ. Sljedeća linija .org 5000 određuje da programski kod koji slijedi započinje na adresi 5000 dek. Svaka naredba je duljine 1 dvostruke riječi. Program prevodilac proračunava vrijednosti labela (skok) kako bi mogao odrediti adresu lokacije na koju se potrebno granati. Ovo se rješava na način da program prevodilac prolazi dva puta kroz program. U prvom prolazu formira tablicu simbola u koju upisuje vrijednost svakog pojedinog simbola. U drugom prolazu kodiraju se naredbi i izračunavaju konstante. Tako npr. labela skok ima vrijednost 5004. Kako je prva naredba relativno adresiranje, tako konstanta c1 je razlika iznosa labele skok (5004) i trenutne vrijednosti programskog brojila (PC = 5001).

```
lar   r0, skok        00110  00000  00000000000000000011
                      op = 6 ra = r0            c1 = 3
```

Nakon izvođenja ove naredbe u spremniku r0 je upisana vrijednost 5001 + 3 = 5004.

Umjesto ove naredbe moglo se koristiti naredba

la r0, skok 00101 00000 00000 0 0001 0011 1000 1000
op = 5 ra = r0 rb = 0 c1 = 5004 =1388H

2.4. Formalan opis SRCa primjenom Notacije za prijenos među spremnicima (Register Transfer Notation skr RTN)

Tekstualni i slikoviti opis karakteristika procesora, njegovih naredbi i sklopova za njihovu realizaciju koristan je opis općih svojstava procesora ali za njegovu izradu potrebna je precizna definicija svih njegovih karakteristika. Takva specifikacija zahtijeva primjenu matematičke notacija kao i prirodnog jezika procesora. Značaj prijenosa podataka između spremnika i memorijskih lokacija te između spremnika procesora uz obradu podataka, rezultirao je nazivom ovog opisa kao jezik za opis prijenosa među spremnicima (*Register Transfer Language* skr. RTL). Ova jezik ima dovoljnu razinu preciznosti u specifikaciji svojstava procesora. Ovaj jezik razvijen je na temelju jezika za opis skupa naredbi procesora (*Instruction Set Processor Language*) razvijenog od Gordona Bella i Alana Newella sličnih svojstava kao RTL koji će se koristiti u ovom kolegiju.

2.4.1. RTN opis SRCa

Kao uvod RTN će se primijeniti za opis komponenata i operacija SRCa. Prvo će se opisati spremnici i memorijske lokacije koje sadrže podatke i kojima se može manipulirati određenim naredbama.

Kako je već spomenuto stanje procesora (*Processor State*) predstavlja stanje svih spremnika procesora.

PC<31..0>:	32 bitovni spremnik nazvan PC (*Program Counter*),
IR<31..0>:	32 bitovni spremnik nazvan IR (*Instruction Register*),
Run:	1 bit indikator radi/stani (*run/halt*),
Strt:	Start signal,
R[0..31]<31..0>	32 32 bitovna spremnika opće namjene.

Spremnici se označavaju alfanumeričkim imenima (PC, IR, itd.). Bitovi spremnika označeni su u zagradama <> iza naziva spremnika. U zagradama se navode ili pojedinačni bitovi ili niz bitova odijeljen s dvije točke (..). S lijeva se nalazi bit najvećeg značaja, a s desna bit najmanjeg značaja. Ukoliko uz naziv nema zagrada radi se o pojedinačnom bitu. Grupa identičnih spremnika može imati isto ime, a razlikuju se indeksom upisanim u uglate zagrade [] iza imena spremnika. Ukoliko se izostave <> zagrade smatra se da se ime odnosi na cjelokupni prethodno definirani opseg spremnika.

RISC procesori opisuju se stanjem procesora koje obuhvaća veći skup spremnika opće namjene i nekoliko spremnika posebne namjene. Kod stvarnih procesora postoje i dodatni spremnici koji određuju stanje procesora kao što su npr. spremnik stanja procesora, spremnik prekida, U/I spremnici, itd. Navedene spremnike obraditi će se naknadno.

Iz navedenoga, stanje SRCa opisano je 32 bitovnim programskim brojilom PC, spremnikom naredbe IR, bitovima Run i Strt, te s 32 32 bitovna spremnika opće namjene. U spremnik naredbe pohranjuje se naredba u cilju dekodiranja njenih pojedinih polja. Bit Run smatra se flip-flop koji se postavlja u jedinicu kada procesor radi, odnosno u nulu kada

je zaustavljen. Strt bit razlikuje se od Run bita utoliko što njega postavlja neki vanjski događaj (npr. uspostavljen je napon napajanja) i on signalizira da su ispunjeni uvjeti za rad procesora, odnosno za postavljanje Run bita.

Sljedeći element koji je značajan za određivanje stanja računala je memorija. Ona je polje istovrsnih spremnika koji se mogu adresirati (pristupati) po pojedinim oktetima. Ali kod SRCa određeno je da se naredbama i podacima pristupa samo kao 32 bitovnoj riječi.

Mem[0..2^{32}-1]<7..0>: 2^{32} adresabilnih okteta glavne memorije.

M[x]<31..0>:=Mem[x]# Mem[x+1]# Mem[x+2]# Mem[x+3]:

Memorijska riječ definira se kao četiri uzastopna memorijska okteta. Kod RTNa ovo je definirano operatorom pridruživanja := i operatorom povezivanja ili konkatencije #. Tako npr. prilikom pristupanja memorijskoj lokaciji 40, SRC bi automatski pristupio adresi 40, 41, 42 i 43, gdje je na adresi 40 smješten oktet najvećeg značaja, a na adresi 43 oktet najmanjeg značaja:

M[40]<31..0>:=Mem[40]# Mem[x+1]# Mem[x+2]# Mem[x+3]:

Sve naredbe SRCa dužine su 32 bita te uz operacijski kod naredbe sadrže i druga polja značajna prilikom njenog dekodiranja. Razlikovanje dijelova spremnika naredbe značajno je za interpretaciju naredbe. Zato RTL uvodi posebne oznake za dijelove spremnika naredbe:

op<4..0>:=IR<31..27>:	Polje operacijskog koda
ra<4..0>:=IR<26..22>:	Polje ciljnog spremnika
rb<4..0>:=IR<21..17>:	Operand, indeks adrese ili lokacija za grananje
rc<4..0>:=IR<16..12>:	Drugi operand, uvjet grananja ili broj posmaka
c1<21..0>:=IR<21..0>:	Polje dugog pomaka (*displacement*)
c2<16..0>:=IR<16..0>:	Polje kratkog pomaka ili neposredna konstanta
c3<11..0>:=IR<11..0>:	Polje uvjeta ili posmaka

Kako se pojedina polja preklapaju te imaju različita značenja u ovisnosti o tipu naredbe tako ona dobivaju i različite nazive.

Naročito značajno je precizno definirati adresne modove, odnosno polja spremnika koja se koriste prilikom proračuna efektivne adrese. Adresiranje može biti apsolutno (pomak ili *displacement*) ili relativno u odnosu na trenutnu vrijednost PCa.

disp<31..0> := ((rb=0) $\rightarrow$ c2<16..0) {proširenje predznaka}:
(rb$\neq$0) $\rightarrow$ R[rb]+ c2<16..0) {proširenje predznaka i aritmetika 2'gi komp.}):

Za primijetiti je da kod opisa direktnog adresiranja nije R[0] = 0 nego je odabran spremnik R[0], r0 = 0. Naravno isti rezultat dobio bi se ukoliko se specificira indeksno adresiranje uz sadržaj indeksnog spremnika jednakog nuli R[rb] = 0. Za primijetiti je kad se specificira rb = 0 tada se odnosi na spremnik broj 0, a R[rb] = 0 znači da je sadržaj spremnika rb jednak nuli.

Simbol $\rightarrow$ simbolizira aktivnost koja slijedi uvjet ukoliko je on ispunjen. Ukoliko nije prelazi se na sljedeći dio izraza. Vitičaste zagrade koje slijede izraz opisuju kako se kratke varijable ili konstante proširuju te kako se izvodi aritmetička operacija.

Drugi oblik adresiranja je relativno adresiranje. Efektivna adresa generira se kao pomak (*displacement, offset*) u odnosu na trenutnu vrijednost PCa.

rel<31..0> := PC<31..0> + c1<21..0) {proširenje predznaka i aritmetika 2'gi komp.}:

Kako ovaj adresni mod zahtjeva samo jedno polje za specifikaciju spremnika moguće je koristiti dugu konstantu za pomak. Ovaj mod naročito je pogodan kod modularnog programiranja kada se moduli primjenjuju u različitim programima. Tada bez obzira gdje se modul upiše u memoriji, sva grananja ostaju nepromjenjena.

Interpretacija naredbe. Do sada se analiziralo samo opis statičkih svojstava procesora, stanje procesora i memorije kao i modovi adresiranja. Slijedi opis dinamičkih svojstava procesora, odnosno opis ciklusa dohvati i izvedi. Nakon što je procesor iz memorije dohvatio naredbu i pohranio je u spremnik naredbe IR slijedi njeno dekodiranje. Aktivnosti koje procesor izvodi prije nego izvede naredbu naziva se interpretacija naredbe. Interpretacija te izvođenje naredbe je slijed aktivnosti koji se naizmjence odvija od trenutka kad se procesor uključi u rad. Kod stvarnih procesora situacija je dosta složenija zbog mogućih prekida i drugih posebnih aktivnosti procesora. Ali u ovom trenutku one će biti zanemarene i razmatrati će se jednostavan ciklus interpretacije i izvođenja naredbe. RTN opis interpretacije naredbe je sljedeći:

interpretacija_naredbe:=(¬Run∧Strt→Run←1:
Run→(IR←M[PC]: PC←PC+4; izvođenje_naredbe));

Prijenos podataka iz memorije u spremnik opisano je strelicom ←. Lijeva strana označava memorijsku lokaciju ili spremnik u koji se podatak pohranjuje (odredište), a desna strana odakle podatak dolazi (izvorište).

Dva ili više operacija odvojenih dvotočkom (:) odvijaju se istovremeno. Ovo je posebno značajno prilikom projektiranja logičkih sklopova za realizaciju navedenih funkcija. Istovremeno za projektanta logičkih sklopova znači da se navedene aktivnosti odvijaju u jednom taktu (jednom *clock* impulsu). Vrijednosti na desnoj strani operatora operacija odvojenih dvotočkom izračunavaju se s istom ulaznom vrijednosti spremnika te se istovremeno pridružuju vrijednostima na lijevoj. Tako u izrazu za interpretaciju naredbe ista vrijednost PCa se koristi za proračun M[PC] i PC + 4. Tek kada se proračun provede ažuriraju se vrijednosti s lijeve strane IR i nova vrijednost PCa. Za zaključiti je da redosljed odvijanja operacija odvojenih dvotočkom nije bitan. Isti rezultat interpretacije naredbe dobio bi se u slučaju sljedećeg izraza:

interpretacija_naredbe:=(¬Run∧Strt→Run←1:
Run→(PC←PC+4: IR←M[PC]; izvođenje_naredbe));

Ipak ovaj način opisa nije uobičajen jer nije "prirodan" procesu koji se odvija.

Aktivnosti odvojene točkom-zarez (;) moraju se izvoditi sekvencijalno, odnosno jedna za drugom. Tako se izvođenje_naredbe izvodi tek nakon što je naredba dohvaćena i PC inkrementiran. Ovakav opis dosta je sličan tekstualnom opisu aktivnosti koje se odvijaju, ali je unio i dodatnu preciznost u redoslijed njihovog izvođenja.

Simbol ¬ je znak za logičku negaciju, simbol ∧ je znak za logičku funkciju I, a simbol ∨ je znak za logičku funkciju ILI. Tako je prvi uvjet koji se ispituje kod interpretacije_naredbe je da li je potrebno postaviti zastavicu ili bit Run u jedinicu. Naime ako Run nije postavljen a

Strt je u jedinici tada je potrebno postaviti Run u jedinicu tj. ispunjeni su uvjeti da procesor krene u izvođenje. U tom slučaju ne izvodi se sljedeća naredba jer je predhodna vrijednost Run bita nula. Procesor tada mora izvršiti inicjalizaciju sklopovlja. Ako pak Strt bit nije postavljen (Strt = 0), a procesor nije u stanju izvođenja (Run = 0) nisu ispunjeni uvjeti za nastavak rada. Tek ako je procesor u stanju izvođenja, tj. Run bit je u jedinici može se pristupiti dohvatu naredbe i njenom izvođenju.

Izvođenje naredbe. Slijedi opis što svaka pojedina naredba radi. Kako je opisano kada se dohvati naredba ujedno se inkremetira programsko brojilo, a potom se izvodi naredba. RTN opis procesa izvođenja naredbi započinje s podužom listom uvjeta koje je potrebno ispitati. Naime prvi korak je odrediti o kojoj se naredbi radi pa je potrebno ispitati operacijski kod naredbe. Broj uvjeta jednak je broju naredbi koje procesor može izvesti. Tako npr. SRCa koji se projektira ima 21 različitu naredbu. Nakon svakog uvjeta slijedi popis aktivnosti koje je potrebno izvesti kako bi se izvela naredba.

Kako je već spomenuto naredbe se mogu podijeliti u četiri osnovne skupine: naredbe za prijenos podataka između memorije i spremnika, ALU naredbe, naredbe za upravljanje programskim tokom (naredbe za grananja) i posebne naredbe.

Naredbe za pristup memoriji. Kod SRCa memoriji se pristupa preko *load* i *store* naredbi. Za ove naredbe vrijedi sljedeći opis procedure izvođenja_naredbe

izvođenje_naredbe:=((

ld (:=op = 1) → R[ra] ← M[disp]:	upiši u spremnik s apsolutne adrese
ldr (:=op = 2) → R[ra] ← M[rel]:	upiši u spremnik s relativne adrese
st (:=op = 3) → M[disp] ← R[ra]:	upiši na apsolutnu adresu iz spremnika
str (:=op = 4) → M[rel] ← R[ra]:	upiši na relativnu adresu iz spremnika
la (:=op = 5) → R[ra] ← disp:	upiši u spremnik apsolutnu adresu
lar (:=op = 6) → R[ra] ← rel:	upiši u spremnik relativnu adresu

Ove naredbe omogućavaju pristup memoriji bilo preko apsolutne ili relativne adrese. Naredbe za upis adrese u spremnik **la** i **lar** praktički omogućavaju upis konstante u spremnik nad kojom se naknadnim aritmetičkim operacijama mogu realizirati i složeniji adresni modovi.

Naredbe za kontrolu programskog toka ili naredbe za grananje. Ove naredbe koriste dio polja c3 u sastavu naredbe za specifikaciju uvjeta koji treba biti ispunjen da bi se realiziralo grananje. Sljedeći je RTN opis naredbi za grananje:

uvjet := (c3<2..0> = 0 → 0:	nikad
(c3<2..0> = 1 → 1:	uvijek
(c3<2..0> = 2 → R[rc] = 0:	ako je sadržaj spremnika rc jednak 0
(c3<2..0> = 3 → R[rc] ≠ 0:	ako je sadržaj spremnika rc različit od 0
(c3<2..0> = 4 → R[rc]<31> = 0:	ako je sadržaj spremnika rc pozitivan ili 0
(c3<2..0> = 5 → R[rc]<31> = 1:);	ako je sadržaj spremnika rc negativan
br (:= op = 8) → (uvjet → PC ← R[rb]):	uvjetno grananje
brl (:= op = 9) → (R[ra] ← PC: uvjet → (PC← R[rb])):	uvjetno s vezivanjem

Ove naredbe prenose izvođenje s postojećeg na drugi niz naredbi koji počinje na adresi određenoj sadržajem spremnika ra ukoliko je postavljeni uvjet zadovoljen. Naredba za

grananje i povezivanje pohranjuje sadržaj programskog brojila u spremnik ra te ukoliko je uvjet za grananje zadovoljen izvodi grananje. Za primijetiti je da se sadržaj PCa pohranjuje u spremnik ra bez obzira da li će doći do grananja. Bit (zastavica) uvjet definirana je jednim od šest postavljenih uvjeta te poprima vrijednost 0 (nije istina) 1 (istina). Adresa grananja određena je sadržajem spremnika rb. Mnogi komercijalni procesori za naredbe za grananje koriste slične modove kao i kod pristupa memoriji. Postoje višestruke pogodnosti ukoliko je adresa grananja sastavni dio naredbe te se time mogu poboljšati performanse procesora. Ipak ovo bi značajno zakompliciralo izvedbu procesora tako da će biti isključeno iz razmatranja.

Aritmetičke i logičke naredbe. Kod obrade aritmetičkih naredbi ograničiti će se na naredbe za zbrajanje, oduzimanje i negativnu vrijednost nad 32 bitovnim brojevima u aritmetici 2'kog komplementa. U jednom od sljedećih poglavlja kod razmatranja realizacije aritmetičkih funkcija ova razmatranja proširiti će se na realne brojeve, odnosno brojeve s pomičnim zarezom.

add (:= op = 12) $\rightarrow$ R[ra] $\leftarrow$ R[rb] + R[rc]:
addi (:= op = 13) $\rightarrow$ R[ra] $\leftarrow$ R[rb] + c2<16..0> {proširenje predznaka}

sub (:= op = 14) $\rightarrow$ R[ra] $\leftarrow$ R[rb] - R[rc]:
neg (:= op = 15) $\rightarrow$ R[ra] $\leftarrow$ -R[rc]:

Razlikuju se dvije naredbe za zbrajanje: zbrajanje sadržaja dvaju spremnika i zbrajanje sadržaja spremnika i kratke konstante koja je sastavni dio naredbe. Prije zbrajanja predznak kratke konstante proširuje se na bitove 17..31 (proširenje predznaka) te se potom izvodi zbrajanje u aritmetici 2'kog komplementa. Naredba za oduzimanje konstante od sadržaja spremnika je nepotrebna jer je istu moguće realizirati kao pribrajanje sadržaju spremnika negativne vrijednosti te konstante.

Nenumerički podaci obrađuju se logičkim operacijama I, ILI i NE. Ove operacije simboliziranje su $\wedge$, $\vee$ i $\neg$ znakovima.

and (:= op = 20) $\rightarrow$ R[ra] $\leftarrow$ R[rb] $\wedge$ R[rc]:
andi (:= op = 21) $\rightarrow$ R[ra] $\leftarrow$ R[rb] $\wedge$ c2<16..0> {proširenje predznaka}

or (:= op = 22) $\rightarrow$ R[ra] $\leftarrow$ R[rb] $\vee$ R[rc]:
ori (:= op = 23) $\rightarrow$ R[ra] $\leftarrow$ R[rb] $\vee$ c2<16..0> {proširenje predznaka}

not (:= op = 23) $\rightarrow$ R[ra] $\leftarrow \neg$R[rc]

Logičke operacije korisne su pri maskiranju pojedinih polja varijabli kao i pri postavljanju i poništavanju njihovih pojedinih bitova.

Naredbe za posmak koriste za manipulaciju podacima koji su obično kraći od jedne riječi. Posmaci se također koriste i kod nekih aritmetičkih operacije kod kojih je naročito kod posmaka u desno potrebno i sačuvati informaciju o predznaku varijable. Posmak za više od 32 mjesta nema smisla kako se radi o 32 bitovnim varijablama. Broj mjesta za koji potrebno posmak izvršiti može biti konstanta koja je dio naredbe ili varijabla koja je sastavni dio nekog od spremnika opće namjene.

```
n :=  ((c3<4..0> = 0 → R[rc]<4..0>:    broj mjesta posmaka je sastavni dio spremnika
       (c3<4..0> ≠ 0 → c3<4..0>));     broj mjesta posmaka je sastavni dio naredbe
```

shr (:= op = 26) → R[ra]<31..0> ← (n @ 0) # (R[rb]<31..n>):
shra (:= op = 27) → R[ra]<31..0> ← (n @ R[rb]<31>) # (R[rb]<31..n>):
shl (:= op = 28) → R[ra]<31..0> ← (R[rb]<31-n..0>) # (n @ 0):
shc (:= op = 29) → R[ra]<31..0> ← (R[rb]<31-n..0>) # (R[rb]<31..32-n>):

Operator @ ponavlja vrijednost s desne strane onoliko puta koliko specificira konstanta s lijeve strane tvoreći niz istovrsnih vrijednosti dužine konstante s lijeve strane. Npr. izraz 5@0 jednak je nizu 00000.

Posebne naredbe. Kod SRCa definirane su dvije posebne naredbe **nop** i **stop**. Naredba **nop** ne izvršava nikakvu aktivnost te jednostavno zauzima mjesto u memoriji (nizu naredbi) kao i vrijeme procesora. Naredba **stop** zaustavlja rad procesora.

nop (:= op = 0) →: nikakva operacija

stop (:= op = 31) → Run ← 0; zaustavi rad
}; interpretacija_naredbe. kraj dijela izvođenje_naredbe

Ovim je završeno formalno definiranje SRCa. Može se postaviti pitanje kako ovaj procesor ostvaruje vezu s vanjskim svijetom, odnosno kako se izvode U/I operacije. Kad većine procesora pristup U/I međusklopovima identičan je pristupu memoriji pa se zato i nazivaju memorijski mapiran U/I prostor.

Sljedeća tablica prikazuje opis RTN naredbi

oznaka	značenje
←	prijenos između spremnika te spremnika i memorije
[]	indeks riječi, odabire opseg spremnika ili memorijskih lokacija
<>	indeks bita, odabire bit ili niz bita riječi
n..m	opseg bitova, od lijeva na desno odabire niz bitove riječi
→	uvjet, ako je uvjet zadovoljen lijeva strana određuje vrijednost
:=	definicija, izraz koji se pridružuje varijabli
#	konkatencija, bitovi s lijeva se povezuju s bitovima s desna
:	separator operacija koje se mogu izvoditi paralelno
;	separator operacija koje se moraju izvoditi sekvencijalno
{}	informacija o prethodnoj operaciji, tekstualni opis
()	grupiranje operacija
= ≠ < ≤ > ≥	operatori usporedbe
+ - * /	aritmetički operatori
∧∨¬⊕≡	logički operatori, I, ILI, NE, EXOR, JEDNAKO

Osnovni koncept opisa skupa naredbi procesora

Osnovne veličine potrebne za opis skupa naredbi procesora mogu se podijeliti u sljedeće četiri kategorije:

1. *Stanje procesora i memorije*. Veličina i broj spremnika procesora, struktura memorije i U/I prostor. Ovo se obično naziva programerski model računala.

2. *Format i interpretacija podataka u spremnicima*. Ovdje spadaju tipovi podataka, format naredbi i proračun efektivne adrese. Aritmetički podaci su obično znatno jednostavniji u usporedbi s interpretacijom naredbi.
3. *Sekvenca interpretacije naredbe*. Ciklus dohvati i izvedi je srce računala i podrazumjeva izvođenje svih naredbi.
4. *Opis pojedinih naredbi*. Naredbe se mogu svrstati u sljedeće četiri skupine: za naredbe prijenos podataka, naredbe za grananja, ALU naredbe i posebne naredbe.

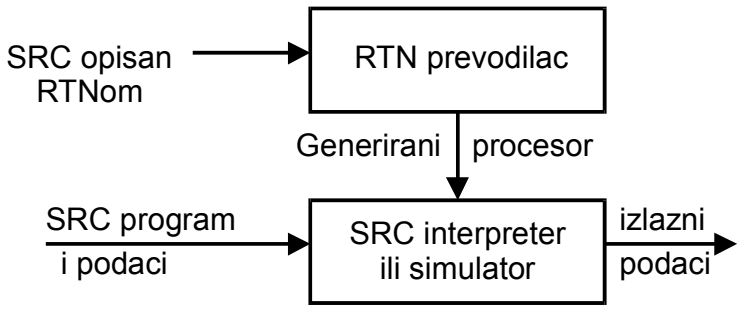
- Naredbe za prijenos podataka određuju fleksibilnost računala da prebacuje podatke između različitih jedinica za njihovo skladištenje (spremnici, memorija, U/I).
- Naredbe za grananje utječu na sekvencu ciklusa dohvati i izvedi tako da određuju koja je sljedeća naredba koju je potrebno izvesti.
- Iako prevodilac simboličkog jezika (*assembler*) može pridjeljivati vrijednosti simbolima i labelama ipak svaka simbolička naredba odgovara jednoj binarnoj strojnoj naredbi.

2.4.2. Usporedba formalnog i neformalnog opisa

Formalan i neformalan opis računala imaju značaj pri njegovom projektiranju i analizi rada. Neformalan opis daje više intuitivni osjećaj kako procesor radi i što treba raditi ali je neprecizan, nepotpun a ponekad i zbunjujući. Formalan opis je s druge strane precizan i točan, ali ponekad težak za shvaćanje i suhoparan. U usporedbi sa strojnim jezikom, RTN se može smatrati kao *metajezik* ili jezik koji se koristi da opiše jezike.

RTN je koristan kako bi se smanjila mogućnost pogrešaka prilikom projektiranja i realizacije procesora. RTN sprječava greške zbog pogrešna interpretacija neformalnog opisa računala te greške u projektiranju i implementaciji.

U prvom slučaju RTN osigurava precizan i nedvojben opis strukture i funkcija računala. U drugom slučaju mogu se programirati prevodioci ili projektirati interpreteri RTNa koji automatski generiraju simulator ili pak sklopovlje računala. TAko RTN program prevodilac može automatski generirati C simulator za procesor koji se projektira ili pak imati izlaz za uređaj za izradu maski za integrirane sklopove. Dobiveni integrirani sklopovi formiraju mikroprocesor koji može izvoditi zadani SRC strojni jezik.



Istina je da može biti i pogrešaka u RTN programu prevodiocu, ali iskustva pokazuju da jezici za formalan opis i automatski generatori iskazuju superiorne rezultate te da postaju univerzalni alati za projektiranje suvremenih računala. Tako npr. VHDL (*Very high-speed integrated circuits Hardware Description Language*) je industrijski standard temeljen na

Ada programskom jeziku. Također postoje postoje posebni jezici koji dozvoljavaju specificiranje komponenata računala na razini logičkih vrata, logičkih blokova i podsustava.

2.5. Opis adresnih modova pomoću RTNa

Načini na koji procesor može pristupati operandima i pohranjivati rezultate značajan je za performanse cjelovitog sustava. Zato je važno adresne modove precizno i formalno opisati. Različiti adresni modovi temeljeni su na uobičajenom strukturiranju podataka u računalnim sustavima. Varijabli u memoriji računala može se pristupiti preko konstantne adrese ili preko pomaka (*offset*) od trenutne vrijednosti PCa. Elementu polja pristupa se pomoću indeksa koji je tek poznat za vrijeme izvođenja programa. Za vezane liste podatak u memoriji predstavlja adresu (pokazivač) sljedećeg elementa liste. Pristup stogu također zahtjeva pokazivač koji se mijenja kako se elementi stavljaju na odnosno skidaju sa stoga. Konačno operandi i rezultat mogu biti pohranjeni kako u memoriji tako i u spremnicima. Sljedeća tablica prikazuje RTN opis adresnih modova.

Naziv moda	Simbolička sintaksa	RTN opis	Upotreba
Spremnik	Ra	R[t] ← R[a]	Privremena varijabla
Spremnik indirektno	(Ra)	R[t] ← M[R[a]]	Pokazivač na strukturu
Neposredno	#x	R[t] ← x	Konstanta
Direktno apsolutno	x	R[t] ← M[x]	Globalna varijabla
Indirektno	(x)	R[t] ← M[M[x]]	Pokazivač na varijablu
Indeksno s pomakom	x(Ra)	R[t] ← M[x+R[a]]	Polje, struktura
Relativno	x(PC)	R[t] ← M[x+PC]	Konst. u programskoj memoriji
Autoinkrement	(Ra)+	R[t] ← M[R[a]] R[a] ← Ra + 1	Sekvencijalni pristup, Stavljanje na stog
Autodekrement	-(Ra)	R[t] ← M[R[a]] R[a] ← Ra - 1	Sekvencijalni pristup, Skidanje sa stog

Programsko brojilo PC, spremnici opće namjene i memorija *n* bitovnog računala s *m* bitovnom memorijskom adresom definira se na sljedeći način:

PC<m-1..0>: Programsko brojilo
R[0..2^q-1]<n-1..0>: Spremnici opće namjene
M[0..2^m-1]<n-1..0>: Memorija

Temeljem ovih oznaka može se zaključiti da je *q* bita potrebno za odabir spremnika opće namjene te *m* bita za adresiranje memorijske lokacije. Oznake a, t i x označavaju polja spremnika naredbe. Polja a i t selektiraju spremnike opće namjene pa su ova polja dužine *q*. Polje x koristi se za pristup memoriji pa mora biti ili dužine *m* bita ili se mora proširiti na tu dužinu uz uvjet da se sačuva informacija o predznaku (proširenje predznaka). Za primijetiti je da prvi adresni mod ne rezultira u memorijskoj adresi pošto se specificiraju spremnici opće namjene.

Posebno kod CISC procesora postoje dodatne varijacije adresnih modova prikazanih u tablici. Tako neki procesori uz konstantu i sadržaj spremnika adresu generiraju i dodavanjem sadržaja drugog spremnika (x + R[a] + R[b]) formirajući tzv. bazno indeksno adresiranje (*based indexed addressing*). Također kod autoinkrementiranja i

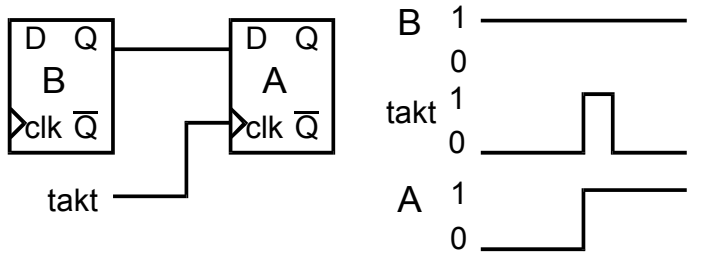
autodekrementiranja moguće je povećavati odnosno umanjivati pokazivač za proizvoljnu vrijednost k određenu naredbom. Ovim se olakšava pristup nizovima čiji elementi su duljine k.

2.6. Prijenos podataka i logički sklopovi: Od opisa do sklopova

U prethodnim razmatranjima objašnjeno je kao procesor izvodi naredbe. Slijedi opis kao se prijenos podataka između spremnika odvija na sklopovskoj razini. Projektanti digitalnih sklopova koriste I, ILI i NI logička vrata za realizaciju funkcija kombinacijske logike te flip-flopove ili bistabili kao memorijske elemente sekvencijalnih logičkih sklopova. Kod računala značajni su i putovi prijenosa podataka kao i elementi za uskladištenje podataka.

2.6.1. Logički sklopovi za realizaciju prijenosa podataka

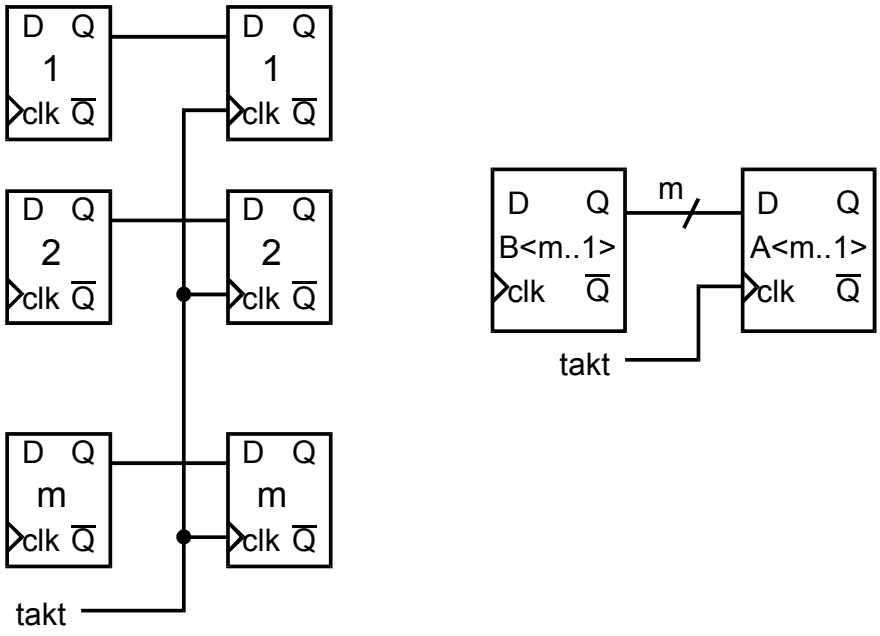
Na ovoj razini razmatra se sklopovska realizacija prijenosa podataka s jednog na drugi element za njihovo uskladištenje. Ako su A i B 1 bitovni spremnici, prijenos podatka iz spremnika B u Spremnik A, A ← B, može se implementirati pomoću dva D bistabila prikazana na slici.



a) Sklopovska realizacija b) Vremenski dijagram

Kod sklopovske realizacije nije samo bitno gdje i odakle se podaci prenose nego i kada. U primjeru se podaci iz spremnika B prenose u spremnik A kada je aktivan signal takt koji je spojen na clk ulaz bistabila. U primjeru je pretpostavljeno da je takt signal kratak impuls mada se u praksi razlikuju bistabili koji se aktiviraju na logičku razinu (0 ili 1) ili na brid impulsa(uzlazni ili padajući).

U računarski sustavima rjeđi su slučajevi kada se manipulira samo s jednim bitom. Češći je slučaj manipulacija s oktetima, riječima, odnosno većim brojem bitova istovremeno. To je razlog da je potrebno odrediti prikladan način prikaza prijenosa podataka koji su prikazani s više bitova (slika).

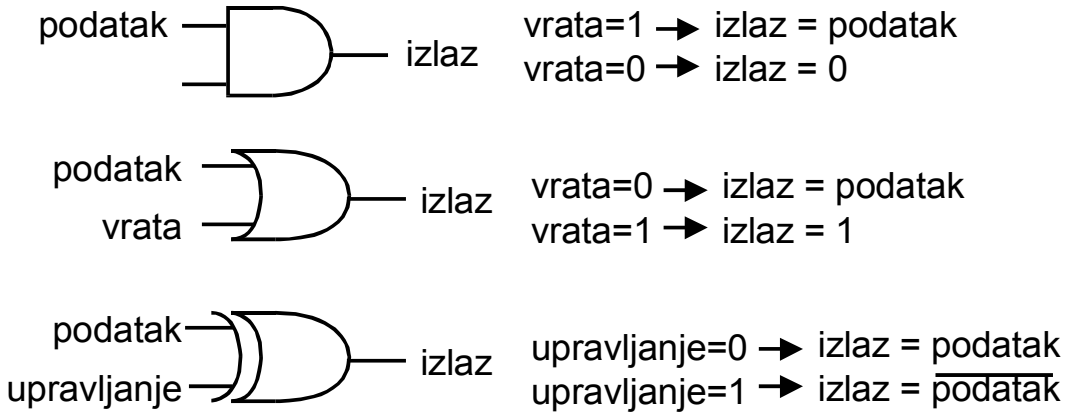


a) realizacija s m bistabila b) skraćeni prikaz

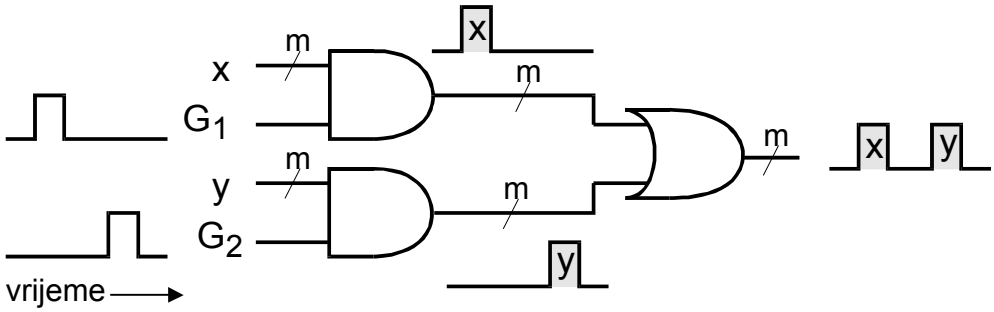
Kako se radi o m bistabila koji predstavljaju izvorište podatka, te m bistabila u koji se taj isti podatak upisuje u skraćenom prikazu bistabili su označeni s B>m..1> i A<m..1> te linija povezivanja izlaza Q bistabila B i ulaza D bistabila A kao m/ što znači da se radi o m linija. Takt signal je samo jedna linija zajednička za sve bistabile pa je tako prikazana i na skraćenom prikazu.

Prijenos podataka u računalima. Logička vrata i bistabili mogu se promatrati s aspekta njihovih prijenosnih svojstava. Prema sljedećoj slici ako jedan od ulaza I vrata nazove *vrata* (*gate*) a drugi *podatak* (*data*) tada jedan od mogućih pogleda na I vrata je da je izlaz jednak *podatku* ukoliko je *vrata* = 1, inače ukoliko je *vrat* = 0 izlaz je 0 bez obzira na vrijednost *podatka*. Slično razmatranje može se provesti i za ILI vrata. Ukoliko je *vrata* = 0 tada je izlaz jednak *podatku*, a ukoliko je *vrata* = 1 izlaz je uvijek 1 bez obzira na vrijednost *podatka*. Kod ova dva sklopa ulaz *vrata* može se promatrati kao upravljački ulaz koji određuje hoće li *podatak* biti proslijeđen na izlaz.

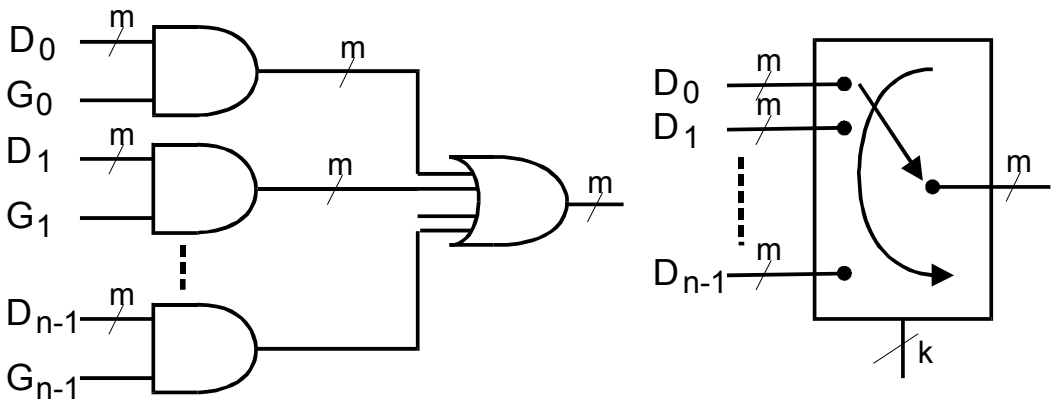
Ekskluzivni ILI (EXOR) sklop može se koristiti za negaciju odnosno logički komplement. Ukoliko je ulaz *upravljanje* = 0 tada se *podatak* prenosi nepromijenjen na izlaz. Obrnuto, ukoliko je *upravljanje* = 1 tada je izlaz komplementiran *podatak* tj. ako je *podatak* = 0 izlaz = 1, odnosno ako je *podatak* = 1 izlaz = 0.



Sjedinjavanje (multipleksiranje) i razdvajanje (demultipleksiranje) podataka. U računalnim sustavima stvarna situacija je takva da je više jedinica povezane su međusobno sabirničkim sustavom ili sustavima. Dva višebitovna podatka vezana su svaki preko i logičkih sklopova čiji izlazi su upravljani nezavisnim signalima G_1 i G_2. Izlazi I sklopova vezani su na ili sklop koji sjedinjava podatke u jedinstveni podatak vremenski razdijeljen. Uvjet ispravnog djelovanja ovog sklopa je da signali G_1 i G_2 nisu nikad istovremeno u logičkoj jedinici.

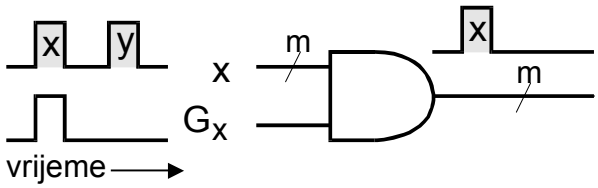


Općeniti n kanalni multipleksor za podatke dužine m bita prikazan je na slici. Uvjet funkcioniranja ovog sklopa je $G_i \wedge G_j$ za svaki $i \neq j$ za cijelo vrijeme. Ovo znači da je u jednom trenutku aktivan samo jedan upravljački signal. Kod realizacije multipleksora, kako je prikazano njegovim simbolom upravljanje se odvija pomoću k linija, gdje je $n = 2^k$. k ulaza preko dekodera selektiraju jedan od n ulaza. Naravno ovo rješenje umanjuje broj upravljačkih linija mada u ovisnosti o samoj aplikaciji nije isključeno ni rješenje gdje se odabir izvodi preko n nezavisnih linija.

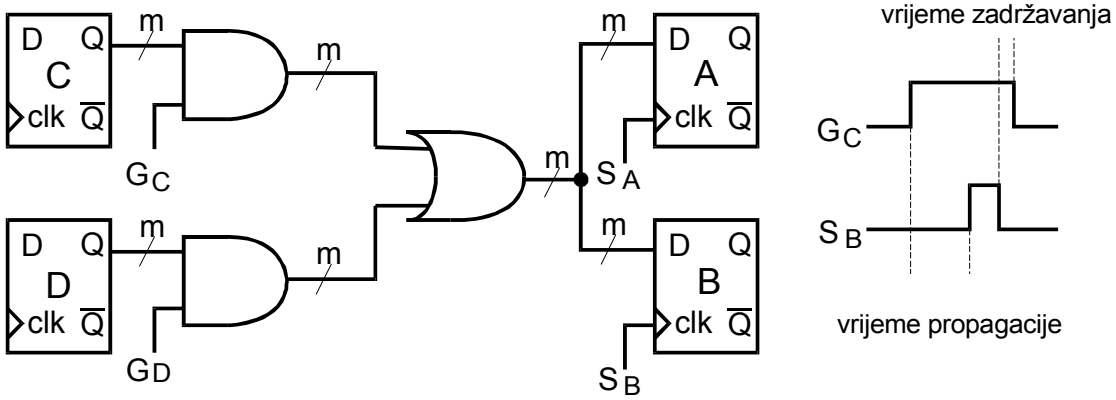


a) multipleksor realiziran logičkim vratima b) simbolički prikaz multipleksora

Sklop za razdvajanje (demultipleksor). Sklop sličan multipleksoru može se koristiti i za razdvajanje multipleksiranih podataka. Postavljanjem upravljačkog signala u odgovarajućem vremenskom trenutku može se iz slijeda podataka izdvojiti određeni podatak (slika).



Prijenos između više spremnika. Pitanje je kako riješiti prijenos podataka iz jednog od više različitih polaznih spremnika u jedan od više odredišnih spremnika.



Neka se namjerava sadržaj spremnika D prenijeti u spremnik B. Postavljanjem signala G_D u logičku jedinicu (G_D = 1) sadržaj spremnika D se preko I i ILI vrata prenosi na sabirnicu koja je spojena na ulaze spremnika A i B. Nakon kratkog vremena koje je potrebno da bi se signal propagirao kroz I i ILI logička vrata, postavljanjem kratkotrajno takt signal S_B u logičku jedinicu upisuje se sadržaj sa sabirnice (sadržaj spremnika D) u spremnik B. Ovaj prijenos, koliko god jednostavan izgledao, zahtijeva strogo poštivanje određenog vremenskog slijeda događaja. Naime stvarni logički sklopovi imaju uvijek određeno vrijeme

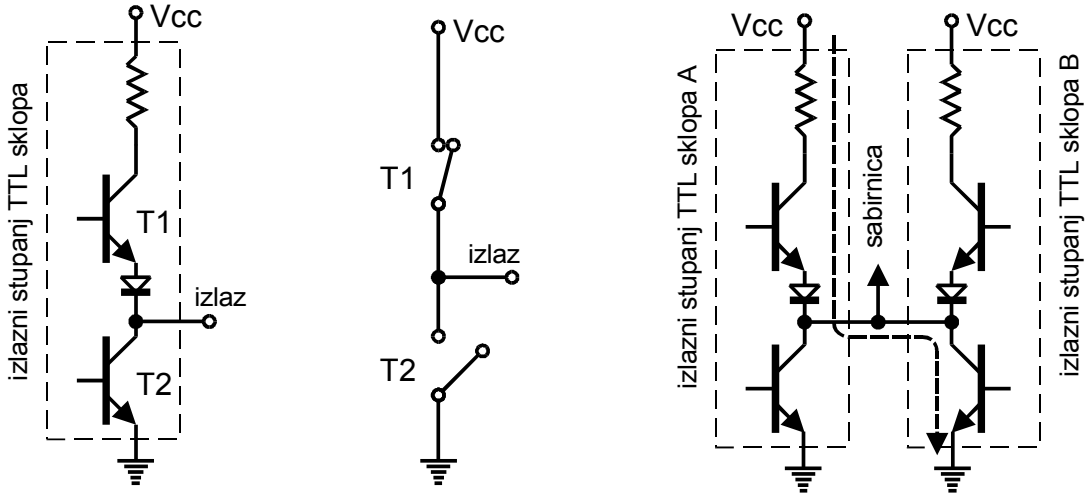
potrebno da signal s ulaza prenesu na izlaz (vrijeme propagacije eng. *propagation time*) te prenesu preko sabirnice do odredišta. Također bistabili zahtijevaju da je signal na ulazu stabilan određeno vrijeme prije nego se aktivira takt signal, a i nakon njegova prestanka, kako bi podatak mogao biti pravilno upisan u bistabil. Ovo je tzv. vrijeme postavljanje eng. *setup time* i vrijeme zadržavanja eng. *hold time*. Obično je vrijeme zadržavanja znatno kraće od vremena propagacije i vremena postavljanja.

U praksi stvarno vremensko vođenje procesa prijenosa podataka iz jednog u drugi spremnik znatno ovisi o njihovoj sklopovskoj realizaciji. Naime spremnici se mogu realizirati pomoću različitih sklopova izvedenih u različitim tehnologijama. Tako npr. upis u spremnik može se izvesti ukoliko je takt signal u određenoj logičkoj razini (1 ili 0) tzv. upis na razinu eng. *level sensitive* ili pak na brid impulsa bilo silazni ili uzlazni eng. *edge triggered*. Različite tehnologije izrade (TTL, CMOS, ECL itd.) imaju različita vremena propagacije, vremena postavljanja i vremena zadržavanja. Svi ovi podatci dani su u uputstvima proizvođača komponenata.

2.6.2. Sabirnice za prijenos podataka

Već na prethodnoj shemi prikazan je način spajanja više različitih ulaza (A i B spremnici) na isti izlaz (ili sklopa). Sustav vodiča kojima je realiziran ovaj spoj naziva se **sabirnica** eng. *bus*. Na sabirnice se spajaju sklopovi koji će određivati u nekom trenutku stanje na sabirnici te sklopovi koji će očitavati stanja sa sabirnice. Shematsko spajanje ulaza i izlaza modula na isti sabirnički sustav izgleda krajnje jednostavan, ali na razini implementacijske domene potrebno je voditi računa o mnogim posebnostima sklopova kojima su realizirani moduli. Jedan od načina spajanja više izlaza na zajednički sabirnički sustav je preko multipleksora (preko I i ILi vrata) kako je već objašnjeno prethodnom slikom.

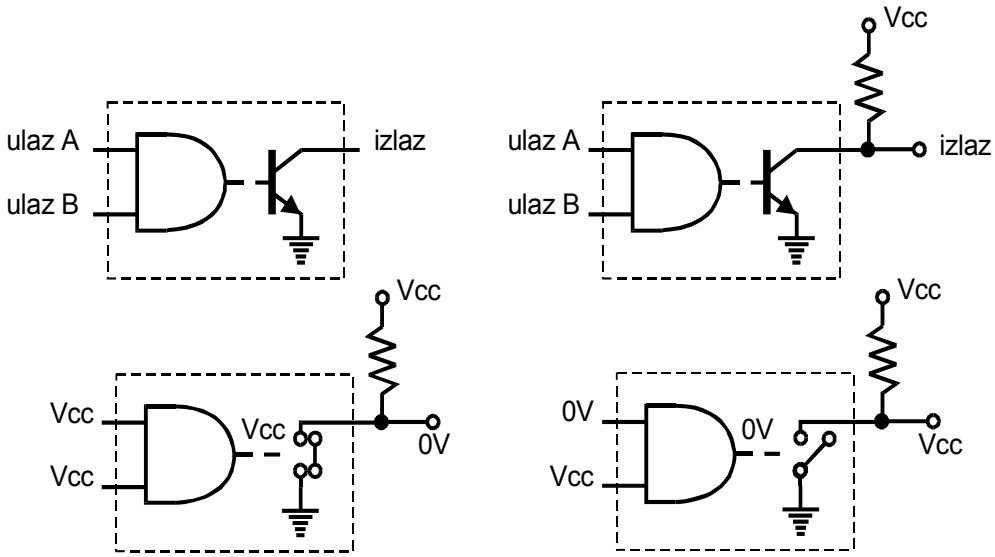
Logički sklopovi s otvorenim kolektorom (*Open Collector*). Logički sklopovi na svojim izlazima koriste tranzistore (bipolarne ili unipolarne) kao sklopke koje spajaju u ovisnosti o stanju na ulazima izlaz ili u logičku jedinicu (napon napajanja) ili logičku nulu (referentni napon).



a) shema izlaza TTLa b) princip djelovanja TTLa c) paralelni spoj izlaza TTLa

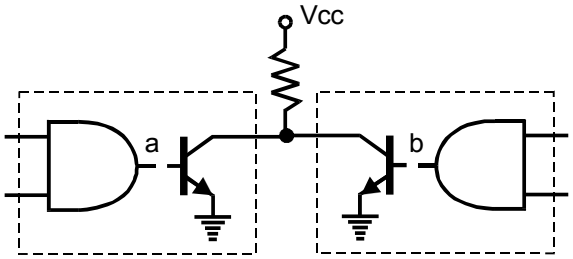
Na slici je prikazan izlazni stupanj TTL sklopa. Kada vodi tranzistor T1 ne vodi tranzistor T2. Izlaz je preko diode i otpornika male vrijednosti spojen na napon napajanja. Izlaz sklopa je u logičkoj jedinici. Obratno kada vodi tranzistor T2 ne vodi tranzistor T1. Izlaz sklopa je spojen na referentni potencijal odnosno u logičkoj je jedinici. Paralelnim spajanjem izlaza TTL sklopa na istu sabirnicu u slučaju kada je izlaz jednog sklopa u logičkoj jedinici a drugog u logičkoj nuli dolazi do kratkog spoja između tranzistora T1 jednog TTL sklopa i tranzistora T2 drugog TTL sklopa. Osim povećane struje i mogućeg oštećenja sklopova stanje na sabirnici je neodređeno.

Ovaj problem moguće je riješiti ukoliko se koriste sklopovi na čijem se izlazu nalazi tranzistor s otvorenim kolektorom.

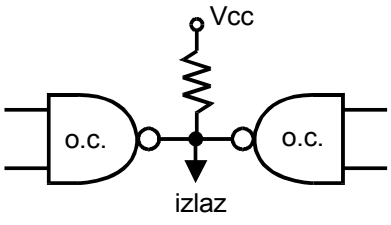


Da bi sklop s otvorenim kolektorom mogao ispravno raditi potrebno je njegov izlaz preko otpornika (obično 10k do 100k) pritegnuti prema naponu napajanja (*pull – up resistor*). Kada su oba ulaza I sklopa u logičkoj jedinici tada je i njegov izlaz u logičkoj jedinici, odnosno na naponu napajanja. Tranzistor na izlazu tada vodi i priteže izlaz sklopa na 0V ili na logičku 0. Ako je jedan od ulaza u logičkoj nuli (0V) tada je i izlaz I sklopa u logičkoj nuli (0V) i izlazni tranzistor ne vodi. Izlaz sklopa je tada preko izlaznog otpora pritegnut na napon napajanja, odnosno je u logičkoj jedinici.

Paralelno spajanje izlaza dva sklopa s otvorenim kolektorom prikazano je na sljedećoj slici.



a) Spajanje izlaza o.c. logičkih sklopova



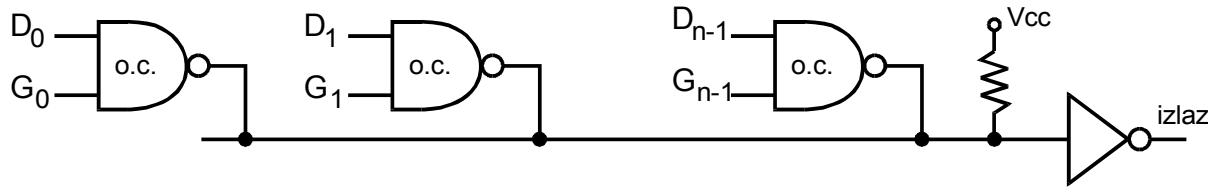
b) Shematski prikaz

a	b	izlaz
0	0	0
0	1	0
1	0	0
1	1	1

c) Tablica istine

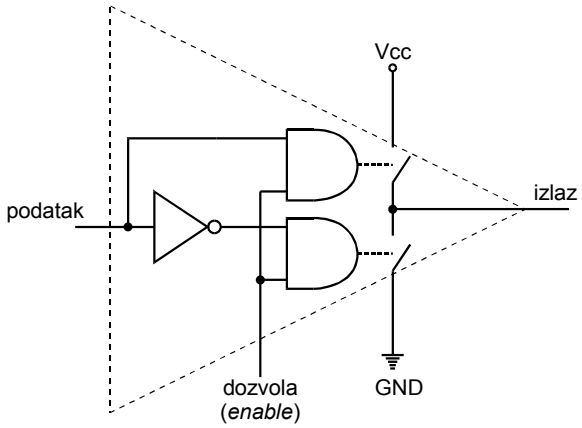
Iz tablice istine ovakvog spoja može se zaključiti da je realizirana logička funkcija I, a zbog načina realizacije ove funkcije (spajanjem izlaza I vrata i priteznim otpornikom) ovaj spoj se naziva **vezano I** (*wired AND*). De Morganovi teoremi govore da komplement ove funkcije je ILI logička funkcija pa u negativnoj logici ovaj sklop postaje **vezano ILI** (*wired OR*). Realizacija logičkih I i ILI funkcija paralelnim spajanjem više izlaza i s jednim priteznim otpornikom ima veliku važnost pri realizaciji različitih računalnih arhitektura.

Standardna realizacija vezane ILI funkcije s n logičkih vrata prikazana je sljedećom slikom. Kako će naknadno biti pokazano ovo rješenje primjenjuje se kod spajanja više uređaja na isti prekidni ulaz procesora.



Zamjena multipleksora logikom s tri stanja (*Tri-State Gates*)

Izlazni stupanj logike s tri stanja realiziran je pomoću dvije sklopke (tranzistora). Upravljanje sklopkama realizirano je na način da se izlaz može spojiti na napon napajanja (logička jedinica), referentni potencijal (logička nula) odnosno da obje sklopke ostanu otvorene. Treće stanje naziva se stanje visoke impedancije(*high impedance)*. Logički sklop s tri stanja prikazan je na sljedećoj slici.



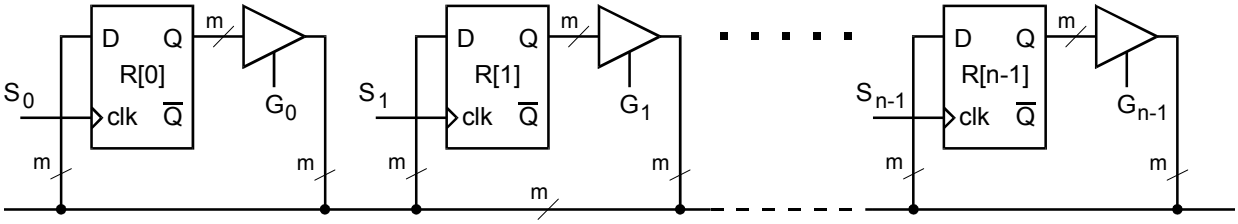
dozvola	podatak	izlaz
0	0	visok. imp.
0	1	visok. imp.
1	0	0
1	1	1

Posebnim ulazom dozvola (*enable*) podatak s ulaza se ili prosljeđuje na izlaz (dozvola = 1) ili se izlaz postavlja u stanje visoke impedancije. Danas postoje sklopovi koji realiziraju različite logičke funkcije a izlazni stupanj im je riješen u logici s tri stanja.

Spajanje spremnika pomoću logičkih sklopova s tri stanja na zajedničku sabirnicu prikazano je sljedećom slikom.

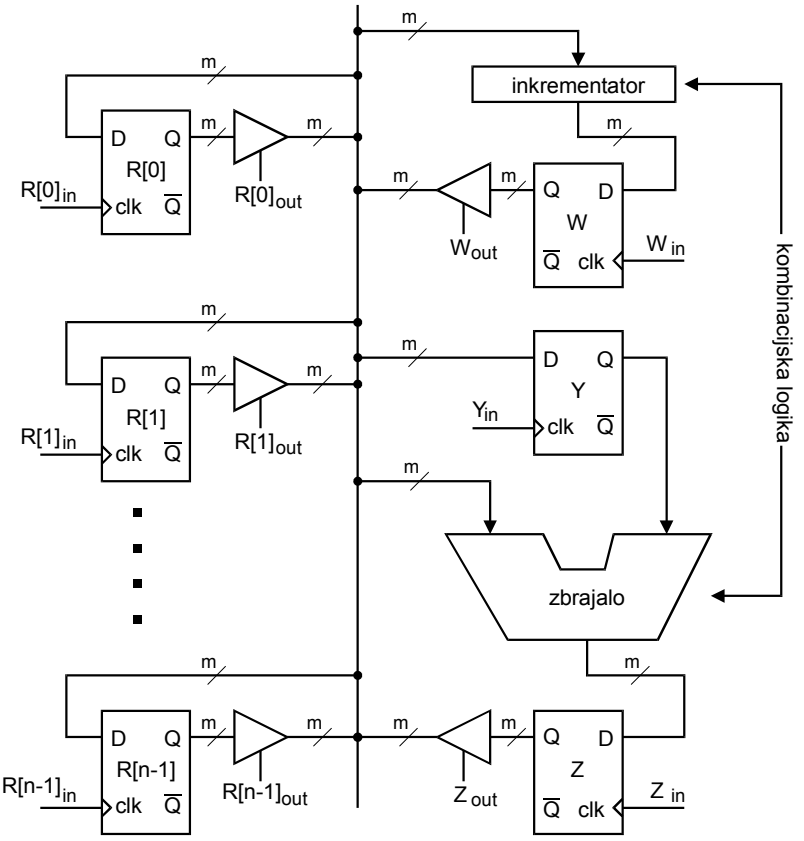
Aktiviranjem samo jednog signala dozvola G_i, i $\in$ 0, n-1 postavlja se sadržaj i-tog spremnika na sabirnicu. Važno je napomenuti da je istovremeno dozvoljeno da bude aktivan samo jedan signal dozvola kako ne bi došlo do kratkih spojeva na izlazima aktivnih sklopova te kako ne bi stanje na sabirnici ostalo nedefinirano. Sadržaj sa sabirnice sada se

može upisati u jedan ili više spremnika aktivirajući (logička razina ili brid) njihove signale vrata.



Prijenos među spremnicima i aritmetička obrada

Jednosabirnički sustav može se jednostavno koristiti za prijenos podataka među spremnicima kako je već opisano. Ukoliko se jednosabirnički sustav namjerava proširiti na način da se omogući i aritmetička obrada podataka potrebne su određene dopune. Npr. za zbrajanje dvaju brojeva potrebno je osigurati jedan privremeni spremnik za prvi operand, a drugi operand aritmetička jedinica može dohvatiti sa sabirnice. Izlaz aritmetičke jedinice nije moguće odmah postaviti na sabirnicu jer je na njoj prisutan drugi operand, već ga je potrebno privremeno pohraniti u jedan izlazni međuspremnik. Slično razmatranje vrijedi i za sklop za inkrementiranje. Shema jednosabirničkog sustava prikazana je na sljedećoj slici.



Ukoliko se namjerava inkrementirati sadržaj k-tog spremnika te rezultat pohraniti u j-ti spremnik (R[k] ← R[j] + 1) potrebna su dva koraka: W ← R[k] + 1; R[j] ← W. Upravljački signali ove sekvence su: $R[k]_{out}$, W_{in}; W_{out}, $R[j]_{in}$.

Slično razmatranje može se provesti za zbrajanje dvaju brojeva (R[3] ← R[1] + R[2]). Prvi korak je upisati prvi operand (sadržaj spremnika R[2]) u privremeni spremnik Y (Y ← R[2]). Slijedi postavljanje drugog operanda (sadržaj spremnika R[1]) na sabirnicu te pohrana rezultata zbrajanje u privremeni izlazni spremnik Z (Z ← R[1] + Y). U trećem koraku se sadržaj privremenog izlaznog spremnika prebacuje u odredišni spremnik R[3] (R[3] ← Z). RTN opis ove operacije je: Y ← R[2]; Z ← R[1] + Y; R[3] ← Z;, a upravljačka sekvenca: $R[2]_{out}$, Y_{in}; $R[1]_{out}$, Z_{in}; Z_{out}, $R[3]_{in}$.

Apstraktan RTN, konkretan RTN, upravljačka sekvenca. Analiza operacije zbrajanja dvaju varijabli pokazala je da postoje dva RTN opisa operacije: prvi koji opisuje što operacija radi

R[3] ← R[1] + R[2]; apstraktan opis,

odnosno kao se operacija izvodi

Y ← R[2]; Z ← R[1] + Y; R[3] ← Z; konkretan opis.

Za realizaciju ove operacije potrebno je generirati sljedeću upravljačku sekvencu:

$R[2]_{out}$, Y_{in}; $R[1]_{out}$, Z_{in}; Z_{out}, $R[3]_{in}$; upravljačka sekvenca.

Apstraktni opis opisuje efekte transfera podatak, dok konkretan opis opisuje kako se transfer podataka između spremnika ostvaruje na konkretnom sklopovlju. Upravljačka sekvenca je niz signala koje je potrebno generirati kako bi se realizirao konkretan prijenos podataka. Za primijetiti je da operacija opisana samo jednim apstraktnim opisom se izvodi određenom sklopovlju u tri koraka.

2.6.3. Operacije transfera podataka i putovi podataka

Sklopovska struktura koja povezuje spremnike, uključujući i aritmetičku jedinicu i ostale sklopove za obradu podataka, tvore putove podataka računala. Prethodno su već prikazane određene međuveze između ponašanja računala određenog RTNom i strukture putova podataka. Oznaka A ← B implicira postojanje veze između spremnika A i B ali ne opisuje o kakvoj se vezi radi (zajednička sabirnica, multipleksor i sl.). Prijenos podataka između različitih spremnika može se odvijati preko jedinstvene sabirnice uz uvjet da u svakom trenutku samo jedan spremnik određuje stanje na sabirnici. Prijenos podatka iz jednog u više spremnika (A ← B: C ← B) preko zajedničke sabirnice može se realizirati istovremeno generiranjem odgovarajuće upravljačke sekvence.

RTN naredbe koje uključuju i transformaciju podataka (aritmetičke i logičke operacije) impliciraju postojanje aritmetičkih i logičkih jedinica u putovima podataka. Primjeri su A ← A + 1 i A ← B + C. Prvu operaciju moguće je realizirati jednostavnim sklopom za inkrementiranje, dok za drugu operaciju je potrebno potpuno zbrajalo za m bita. Opisani primjer pokazaje da se zbrajanje sadržaja dvaju spremnika, R[3] ← R[1] + R[2], mora izvesti u više koraka mijenjajući pri tome sadržaje privremenih spremnika Y i Z. Ovaj primjer pokazao je da se i RTN opis može podijeliti na dvije razine. Apstraktan opis opisuje utjecaj operacije na spremnike vidljive programeru, dok konkretan RTN opis opisuje utjecaj na sve spremnike računala. Kod konkretnog RTN opisa uzima se u obzir i sve specifičnosti putova podataka.

RTN, ponašanje i implementacija

RTN specificira ponašanja operacija računala i opisuje sklopovlje za implementaciju specificiranih operacija.

- U projektiranju računala odjeljuju se operacije nad podacima i upravljačke operacije iako se obije realiziraju logičkim sklopovima
- Apstraktan RTN opis specificira sveukupni rezultat jednog ili više transfera podataka i njihove transformacije kao npr. izvođenje cjelovite naredbe.
- Konkretan RTN opis specificira izvođenje operacije na konkretnom sklopovlju.
- Konkretan prijenos među spremnicima implementira se jednostavnom kombinacijskom logikom i bistabilima.
- Prijenos iz jednog izvora na više odredišta temelji se na primjeni multipleksora.
- Povezivanje više izvora podataka i više njihovih odredišta preko zajedničke sabirnice zahtijeva primjenu posebnim sklopova, sklopovi s otvorenim kolektorom ili logika s tri stanja.

Zaključak

- S aspekta programera i projektanta arhitekture računala, računala i njihove naredbe se klasificiraju prema načinu kako procesor pristupa operandima. Broj eksplicitno zadanih adresa operanada u okviru naredbe varira od tri do nula. Procesori temeljeni na akumulatoru bili su posebno popularni u prvim računalima i mikroprocesorima. Danas su ustaljene arhitekture temeljene na spremnicima opće namjene te pristupu memoriji preko *load*/*store* naredbi.
- Instrukcijski skup je skup svih naredbi koje procesor može interpretirati i izvesti. Naredbe su podijeljene u tri skupine: prijenos podataka, aritmetičke i logičke operacije, upravljanje tokom programa ili naredbe za grananja.
- Većina računala pristupa memoriji na više načina, odnosno posjeduje različite adresne modove. Ovi modovi omogućavaju pristup različitim strukturama podataka koje se susreću u višim programskim jezicima. Tako se razlikuje neposredno, direktno, indirektno relativno, indeksno itd.
- Na jednostavnom računalu sa smanjenim skupom naredbi, SRC, prikazani su temljni principi arhitekture računala. SRC je određen kao računalo sa spremnicima opće namjene koje pristupa memoriji samo preko *load* i *store* naredbi. Arhitektura skupa naredbi ISA SRCa opisana je neformalno, tekstom i slikom, i formalnom primjenom jezika za opis prijenosa podataka između spremnika, RTN. RTN je primjenjen i na opis adresnih modova procesora. RTN je alat za formalan i precizan opis procesora i njegovih operacija.
- RTN može se jednostavno preslikati u sklopovsku realizaciju. I, ILI, i NE vrata kao i bistabili koriste se pri uskladištenju i prijenosu podataka. Multipleksori usmjeravaju podatke iz izvora prema odredištu. Sklopovi s otvorenim kolektorom, kao i logika s tri stanja omogućavaju prijenos podataka preko zajedničke sabirnice.

Zadaci

2.1. Procesor Motorola 6800 temeljen je na osam bitovnom akumulatoru. Naredba za prebacivanje podatka iz memorije u akumulator je **lda** adr, odnosno da se sadržaj akumulatora prebaci u memoriju **sta** adr. Kako izgled kod programa za prebacivanje 128 bitovnog realnog broja s jedne memorijske adrese na drugu?

2.2. Napišite kod za implementaciju izraza A = (B + C)*(D + E) sa naredbama sa 3, 2, 1 i 0 adresa. Uobičajeno je u programerskoj praksi da rezultat ne prepisuje operand.

2.3. Napišite kod za implementaciju izraza A = (B + C)*(D + E) na SRCu. Operandi se prvobitno nalaze u memoriji te je konačni rezultat također potrebno pohraniti u memoriju.

2.4. Usporedite broj pristupa memoriji za realizaciju izraza A = (B + C)*(D + E) na spomenutih pet računala.

2.5. Neka operacijski kod zauzima 8 bita, memorijske adrese su dužine 24 bita, a broj spremnika je određen s 5 bita te naredba može zauzimati bilo koji cijeli broj okteta. Izračunajte koliko memorije zauzima program za proračun izraza A = (B + C)*(D + E) na spomenutih pet računala.

2.6. Kako je moguće kodirati na procesoru sa spremnicima opće namjene uvjetno grananje koje uspoređuje dva broja i izvodi grananje ukoliko su oni jednaki?

2.7. Napišite SRC kode za implementaciju sljedećeg segmenta napisanog u Cu uz pretpostavku da su sve varijable 32 bitovni integeri.

a) **if** (a == 0) a = a + 1; **else** a = a – 1;

b)
```
for  (i = 0; i < 10; i++)
        znamenka[i] = 0;
```

2.8. Modificirajte RTN SRC opis uvođenjem zastavice za *Single-Step* funkciju. Kada je zastavica *Run* postavljena u nulu, tj. procesor je zaustavljen, postavljenjem *single-Step* zastavice procesor izvodi jednu naredbu i ponovo se zaustavlja.

2.9. Modificirajte SRC na način da se instrukcijski skup proširi naredbom **swap** (op = 7) koja zamjenjuje sadržaje spremnika ra i rb te napišite RTN nove naredbe.

2.10. Modificirajte SRC RTN na način da postoji naredba za izvođenje uvjetnog grananja **jpr** (op = 29). Naredba mora koristiti format za naredbe br. 2 (op = 5 bita, ra = 5 bita i c1 = 22 bita). **jpr** koristi relativno grananje. Grananje se izvodi samo ako je ra = 0.

2.11. Opišite riječima razliku između sljedeća dva adresna moda opisana RTNom:

a) M[M[X + R[a]]]
b) M[M[X] + R[a]]

2.12. Napišite SRC naredbe za upis podatka iz memoriju u spremnik koristeći sve spomenute adresne modove. Za različite adresne modove potrebno je koristiti više naredbi.

2.13. Za sklopovsku realizaciju procesora sa sklopom za inkrementiranje i zbrajanje napišite RTN opis i upravljačku sekvencu za implementaciju sljedećih operacija:

a) R[3] ← R[4] + R[6] + 1
b) R[2] ← R[3] + R[4] + R[5]

2.14. Projektirajte putove podataka koji omogućavaju sljedeći prijenos podataka u jednom koraku te odredite upravljačke signala potrebne za njegovu realizaciju.

A<m-1..0> ← B<m-1..0>;
C<m-1..0> ← B<m-1..0> + 1;
A<m-1..0> ← C<m-1..0>;
B<m-1..0> ← C<m-1..0>;

2.15. Projektirajte putove podataka koji omogućavaju da se bilo koja dva specificirana prijenosa koja ne zahtijevaju isto odredište realiziraju u jednom koraku:

A ← B:
A ← C:
C ← B:
B ← C: