4. PROJEKTIRANJE PROCESORA

4.1. postupak projektiranja procesora

Cilj ovog poglavlja je prikaz postupka projektiranja procesora kao pristup projektanta procesu projektiranja. Prvi koraci procesa projektiranja koji opisuju što procesor treba raditi prikazani su u drugom poglavlju. Tada su neformalno i formalno opisane naredbe koje procesor kojeg se želi realizirati mora izvoditi kao i dio sklopovlja potrebnog za njihovu realizaciju. Radi se o sklopovima koji su vidljivi programeru. Sljedeći korak je definiranje tzv. putova podataka. Putovi podataka su spremnici i međuveze koji se koriste u procesu promjena podatka kojima se realiziraju naredbe. Za opis promjena koje se zbivaju s podacima u putovima podataka koristiti će se RTN. Pri tome je potrebno voditi računa i o posebnostima sklopova, što sveukupno predstavlja podlogu za projektiranje logičkih sklopova za realizaciju putova podataka.

Sljedeći korak je projektiranje sklopova. Kako bi se pojedine aktivnosti realizirale projektant razmatra koje upravljačke signale je potrebno generirati, kao što je signal za upis u spremnik, upiši, i signal za postavljenje sadržaja spremnika na sabirnicu, postavi. Posljednji korak je realizacija upravljačke jedinice koja određenim redoslijedom generira upravljačke signale koji upravljaju aktivnostima na putovima podataka.

Važno je napomenuti da svaki korak sadrži specifikacije što je potrebno napraviti u sljedećem koraku. Naravno, uvijek je prisutno pitanje da li je moguće navedene specifikacije sklopovski realizirati. To je razlog da je ponekad potrebno vratiti se na prethodne korake i izvesti određene preinake, odnosno proces projektiranja zahtijeva više iteracija.

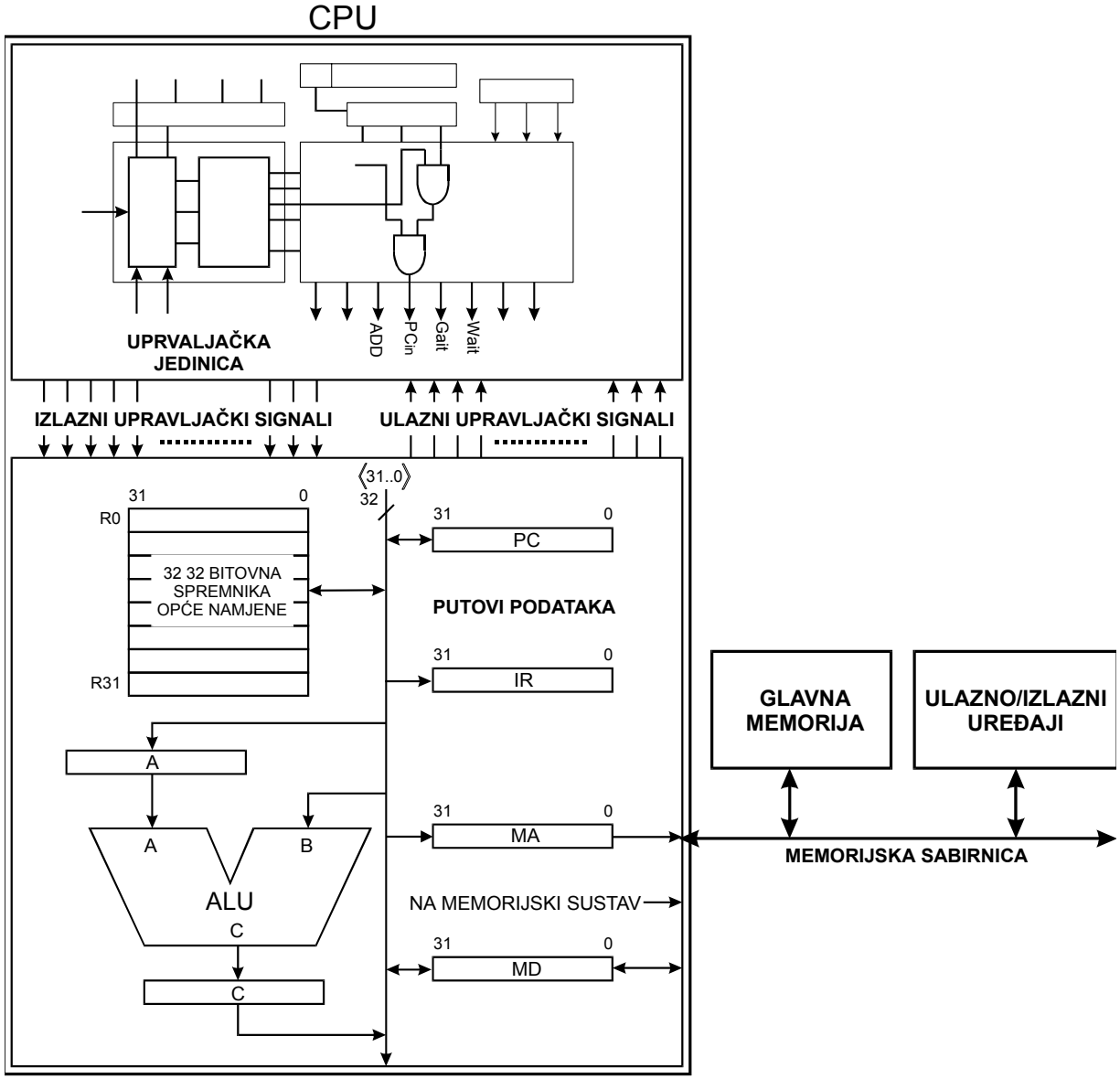
Važan početni korak u procesu projektiranja procesora bio je apstraktan te kasnije precizan opis pomoću RTNa njegovog skupa naredbi, odnosno arhitekture skupa naredbi. Apstraktan opis je potpuno neovisan o domeni implementacije i definira statička svojstva procesora, spremnike i memoriju, te dinamička svojstva, operacije. Konkretan RTN opisuje detalje prijenosa podataka između spremnika, odnosno radnje koji se događaju na putovima podataka i rezultiraju konačnim efektom obrade. Proces izvođenja naredbe je osnovna jedinica apstraktnog opisa, dok koraci konkretnog RTNa odgovaraju aktivnostima sinkroniziranim s procesorskim taktom. Tako je konkretan RTN vezan uz domenu implementacije, pa za jedan apstraktan opis može postojati više konkretnih RTN opisa.

Različite sklopovske arhitekture mogu podržavati istu arhitekturu skupa naredbi što slijedi iz prethodnih razmatranja. Tako npr. SPARC International specificira apstraktan procesor s apstraktnom arhitekturom skupa naredbi. Mnogi proizvođači realizirali su navedene specifikacije na različite načine. Razlike su u čitavom nizu praktičkih detalja kao što je struktura sabirnica, vrijeme izvođenja naredbi, veličina cjevovoda, stupanj paralelne obrade više naredbi, itd. Važno je naglasiti da svi ti procesori izvode iste naredbe iako se značajno razlikuju u pojedinostima realizacije procesora. Temeljen navedenog za SRC koji se namjerava projektirati i koji je opisan apstraktnim RTN opisom slijedi razrada mogućih konkretnih RTN opisa.

4.2. Jedno-sabirnička mikroarhitektura SRCa

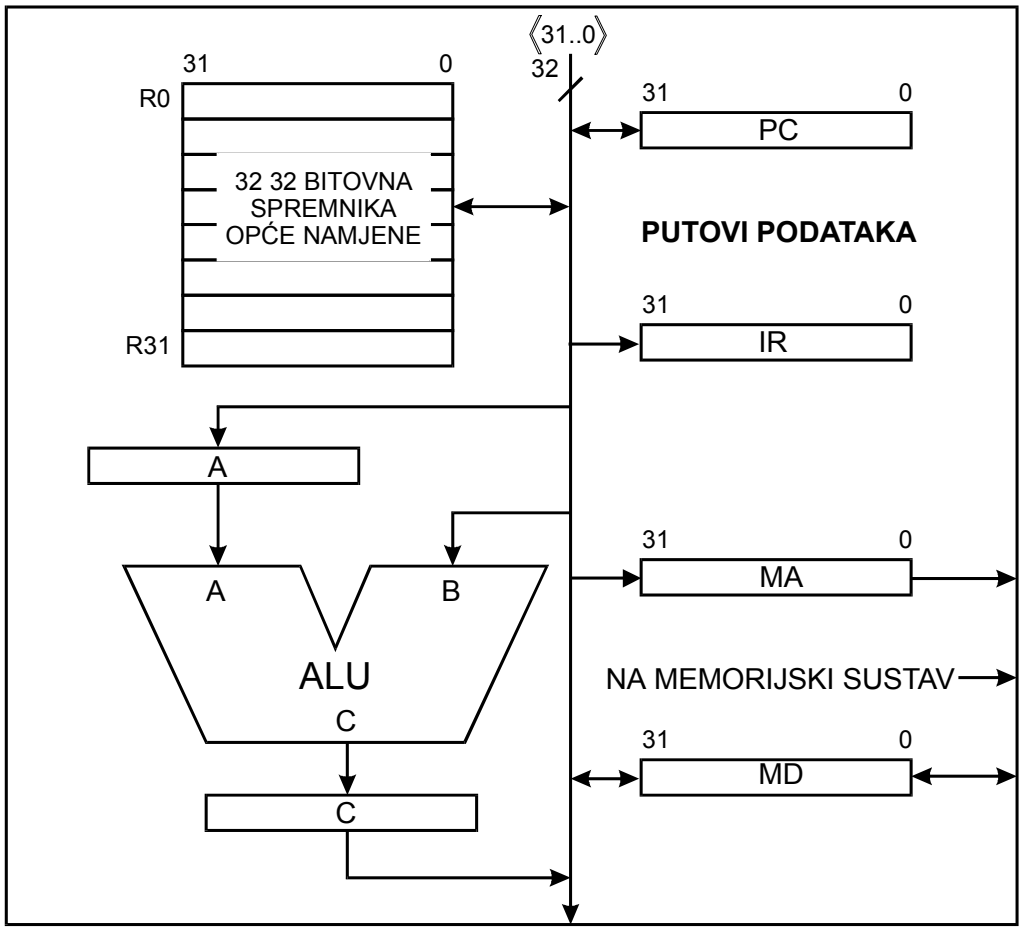
Konkretan RTN opis slijedi iz određene mikroarhitekture procesora kojom se namjerava realizirati arhitektura skupa naredbi. Pojam mikroarhitekture odnosi se na spremnike

procesora, sustav sabirnica i ostale funkcionalne jedinice kao što je npr. ALU, upravljačka jedinica, brojila. Sljedeća slika prikazuje kako su međusobno povezane pojedine funkcionalne jedinice procesora.



Ova slika predstavlja temelj daljnjim detaljnijim prikazima koji opisuju komponente putova podataka i upravljačke jedinice. Osnovni podsustavi računala, prikazani na slici su centralna procesna jedinica (CPU), glavna memorija i ulazno/izlazni uređaji. Putovi podataka (*data path*) povezani su s upravljačkom jedinicom. Upravljačka jedinica dobiva informacije, signale, od putova podataka temeljem kojih generira upravljačke signale. Upravljački signali upravljaju tokovima podataka unutar CPUa, između CPUa i glavne memorije te između CPUa i ulazno/izlaznih uređaja. Sljedeća razmatranja fokusirana su na projektiranje putova podataka i upravljačke jedinice.

Sljedeća slika prikazuje dio jedno-sabirničke mikroarhitekture SRCa. Zbog jednostavnosti prikaza izostavljeni su detalji realizacije spremnika kao i upravljački signali. Oni će biti naknadno dodani. Za primijetiti je da postoje dodatni spremnici koji nisu opisani apstraktnim RTN opisom, A, C, MA, MD. Prva dva su potrebna u mikroarhitekturi za privremenu pohranu operanda i rezultata prilikom obavljanja ALU naredbi. Potreba uvođenja privremenih spremnika opisana je u prethodnom poglavlju, a rezultat je zahtijeva da u određenom trenutku samo jedan podatak smije biti prisutan na sabirnici. MA (*memory address*) i MD (*memory data*) koriste se kao međuveza s memorijom i ulazno/izlaznim uređajima. MA sadrži memorijsku adresu operanda, dok MD je međuspremnik podataka koje ulaze u CPU ili izlaze iz njega.



4.2.1. Apstraktan i konkretan RTN SRC naredbe za zbrajanje, add

Kao primjer apstraktnog i konkretnog RTN opisa uzeti će se naredba za zbrajanje **add**:

Apstraktni RTN (IR ← M[PC]: PC ← PC + 4; izvođenje_naredbe);
izvođenje_naredbe := (
•••
add (:= op = 12) → R[ra] ← R[rb] + R[rc]:
•••

Apstraktan RTN kaže da se naredba s adrese na koju pokazuje sadržaj programskog brojil dohvata i upisuje u spremnik naredbe. Ujedno se sadržaj programskog brojila poveća za četiri kako bi pokazivao na sljedeću naredbu. Slijedi izvođenje naredbe. Ako je operacijski kod 12 radi se o naredbi za zbrajanje koja zbraja sadržaje spremnika rb i rc te rezultat upisuje u spremnik ra. Za primijetiti je da nije opisano kako procesor izvodi zbrajanje. On ovisi isključivo o samoj realizaciji putova podataka.

Konkretan RTN opis naredbe za zbrajanje za procesor temeljen na jednosabirničkoj mikroarhitekturi prikazanoj na prethodnoj slici prikazan je sljedećom tablicom:

Korak	RTN
T0	MA ← PC: C ← PC +4;
T1	MD ← MMA: PC ← C;
T2	IR ← MD;
T3	A ← R[rb];
T4	C ← A + R[rc];
T5	R[ra] ← C;

Apstraktan opis dohvata i izvođenja naredbe za zbrajanje zahtijeva samo dva koraka. Na konkretnom sklopovlju dohvat i izvođenje iste naredbe zahtijeva šest koraka, tri za dohvat naredbe i tri za njeno izvođenje. Sljedeći su konkretni koraci u dohvatu i izvođenju naredbe za zbrajanje:

1. Postavi sadržaj PCa i upiši u MA. Ujedno inkrementiraj sadržaj PCa pomoću ALU i rezultat pohrani u privremeni spremnik C.
2. Očitaj sadržaj memorijske lokacije na adresi na koju pokazuje sadržaj MA i upiši u međuspremnik MD. Postavi sadržaj privremenog spremnika C na sabirnicu i upiši u PC.
3. Postavi sadržaj MDa ma sabirnicu i upiši u IR. Sklopovi sada dekodiraju naredbu i zaključuju da se radi o zbrajanju.
4. Postavi sadržaj spremnika rb na sabirnicu i upiši ga u privremeni spremnik A.
5. Postavi sadržaj spremnika rc na sabirnicu, naredi ALU da izvede zbrajanje sadržaja privremenog spremnika A i sadržaja na sabirnici te rezultat upiši u privremeni spremnik C.
6. Postavi sadržaj spremnika C na sabirnicu i upiši ga u spremnik ra.

Za primijetiti je kako se u prva dva koraka izvode paralelno dvije operacije. One su u konkretnom RTNu odvojene dvotočkom. ALU inkrementira sadržaj PC za četiri što je i sasvim prihvatljiv zahtjev na ALU.

Prva tri koraka predstavljaju dohvat naredbe. Kako je potrebno prvo dohvatiti naredbu prije nego što se ona dekodira, a tek nakon dekodiranja moguće je znati što je potrebno raditi, slijedi zaključak da su prva tri koraka identična za svaku naredbu kod ovakve mikroarhitekture.

Ukoliko se svaki korak poistovjeti s jednim taktom, naredba za zbrajanje traje šest taktova. Naravno, intencija je da se naredba izvede u što kraćem vremenu pa sukladno tom zahtjevu projektanti prilagođavaju mikroarhitekturu.

4.2.2. Konkretan RTN za tipične naredbe SRCa

U ovom dijelu konkretno će se opisati po jedna naredba karakteristična za pojedinu grupu. Opis će započeti s **addi** naredbom koja zbraja sadržaj spremnika rb s konstantom c2 koja je sastavni dio naredbe i rezultat upisuje u spremnik ra. Zatim će se opisati naredba za prijenos podatka iz memorije u spremnik, **ld**, i naredba za prijenos podatka iz spremnika u memoriju, **st**. Sljedeće naredbe su naredbe za upravljanje programskim tokom. Iz te skupine naredbi obraditi će se **br** naredba. Konačno će se obraditi naredba za posmak, **shr**.

Aritmetičke naredbe: naredba za neposredno zbrajanje, addi. Nredba za neposredno zbrajanje pribraja sadržaju spremnika rb konstantu c2 koja je sastavni dio naredbe.

addi (:= op = 13) $\rightarrow$ R[ra] $\leftarrow$ R[rb] + c2$\langle$16..0$\rangle$ {2'ki komplement, proširenje predznaka}:

Konkretan RTN opis prikazan je sljedećom tablicom u kojoj su prva tri koraka izostavljena iz razloga što su identični svim naredbama.

Korak	RTN
T0 –T2	Dohvat naredbe
T3	A $\leftarrow$ R[rb];
T4	C $\leftarrow$ A + c2 {proš. pred.}
T5	R[ra] $\leftarrow$ C;

Za primijetiti je veliku sličnost ove i naredbe za zbrajanje sadržaja dvaju spremnika, **add**. One se razlikuju samo u četvrtom koraku kada se drugi operand na sabirnicu postavlja direktno iz spremnika naredbe, IR, uz proširenje predznaka bitova 17..31. Sklopovsko rješenje proširenja predznaka biti će naknadno opisano.

Naredbe za prebacivanje podataka, čitaj i piši. Sljedeći je apstraktan RTN opis naredbi čitaj i piši:

ld (:= op = 1) $\rightarrow$ R[ra] $\leftarrow$ M[disp]: Upiši u spremnik iz memorije
st (:= op = 2) $\rightarrow$ M[disp] $\leftarrow$ R[ra]: Upiši u memoriju iz spremnika

gdje je sljedeća definicija pomaka, disp:

disp$\langle$31..0$\rangle$:= ((rb = 0) $\rightarrow$ c2$\langle$16..0$\rangle$ {proširenje predznaka}:
(rb $\neq$ 0) $\rightarrow$ R[rb] + c2$\langle$16..0$\rangle$ {proširenje predznaka}):

Konkretan RTN opis ovih naredbi prikazan je sljedećom tablicom

Korak	**RTN** za **ld**	**RTN** za **st**
T0 –T2	Dohvat naredbe	Dohvat naredbe
T3	A $\leftarrow$ ((rb = 0) $\rightarrow$ 0: rb $\neq$ 0) $\rightarrow$ R[rb];	A $\leftarrow$ ((rb = 0) $\rightarrow$ 0: rb $\neq$ 0) $\rightarrow$ R[rb];
T4	C $\leftarrow$ A + (16@IR$\langle$16$\rangle$#IR$\langle$15..0$\rangle$);	C $\leftarrow$ A + (16@IR$\langle$16$\rangle$#IR$\langle$15..0$\rangle$);
T5	MA $\leftarrow$ C;	MA $\leftarrow$ C;
T6	MD $\leftarrow$ M[MA];	MD $\leftarrow$ R[ra];
T7	R[ra] $\leftarrow$ MD;	M[MA] $\leftarrow$ MD;

Baza adrese određuje se u četvrtom koraku, T3, kada se u privremeni spremnik A upisuje 0 ako je rb = 0, odnosno sadržaje spremnika rb ako je rb $\neq$ 0. U petom koraku, T4, izračuna se efektivna adresa pribrajanjem konstante, pomaka, koja je sastavni dio naredbe uz proširenje njenog predznaka, s bazom koja se nalazi upisana u privremenom spremniku A.

U šestom koraku, T5, upisuje se memorijska adresa u spremnik MA. Sedmi i osmi korak razlikuje se kod ovih dviju naredbi u smjeru prijenosa podatka. Kod **ld**, iz memorije u međuspremnik MD u sedmom koraku, T6, te iz MD u odredišni spremnika ra u osmom koraku, T7. Podatak iz spremnika ra prebacuje se u MD u sedmom koraku, T6, kod naredbe **st**, te u osmom koraku iz MD u memoriju. MD je poseban spremnik koji može čitati sa i pisati na dvije sabirnice, unutarnju sabirnicu procesora i memorijsku sabirnicu.

Naredba za grananje. Naredba za grananje, **br**, koristi polje uvjeta, c3⟨2..0⟩, koje je sastavni dio naredbe za specifikaciju uvjeta grananja. Ako je uvjet zadovoljen grananje se izvodi upisom odredišne adrese grananja u programsko brojilo:

br (:= op = 8) → (uvjet → PC ← R[rb]): uvjetno grananje

gdje je uvjet zadan kao:

uvjet :=	(c3⟨2..0⟩ = 0 → 0:	nikad
	c3⟨2..0⟩ = 1 → 1:	uvijek
	c3⟨2..0⟩ = 2→ R[rc] = 0:	ako je sadržaj rc jednak nuli
	c3⟨2..0⟩ = 3→ R[rc] ≠ 0:	ako je sadržaj rc različit od nule
	c3⟨2..0⟩ = 4→ R[rc]⟨31⟩ = 0:	ako je sadržaj rc pozitivan
	c3⟨2..0⟩ = 5→ R[rc]⟨31⟩ = 1:	ako je sadržaj rc negativan

Za primijetiti je da je c3⟨2..0⟩ = IR⟨2..0⟩.

Sljedeća tablica prikazuje konkretan RTN opis naredbe **br**.

Korak	RTN
T0 –T2	Dohvat naredbe
T3	CON ← uvjet(R[rc]);
T4	CON → PC ← R[rb]:

Proračun u četvrtom, T3, i petom, T4, koraku možda je nejasan u ovom trenutku. Uvjet CON proračunava sklopovlje na temelju prva tri bita spremnika naredbe, c3⟨2..0⟩ = IR⟨2..0⟩, i sadržaja spremnika rc. Ukoliko je uvjet zadovoljen, odnosno zastavica CON postavljana, u petom koraku mijenja se sadržaj programskog brojila. Sklopovlje potrebno za realizaciju navedenih funkcija biti će naknadno opisano.

Naredba za posmak. SRC ima mogućnost pomicati sadržaj spremnika za više mijesta u oba smjera. Broj mjesta za koji je potrebno izvršiti posmak specificiran je ili poljem naredbe ili sadržajem spremnika rb. ALU specificirana je na način da može izvoditi posmak samo za jedno mjesto. Zato se posmak za n mjesta izvodi u n prolaza kroz ALU. Apstraktan opis naredbe za posmak u desno je:

shr (:= op = 26) → R[ra]⟨31..0⟩ ← (n@0#R[rb]⟨31..n⟩):

n := (	(c3⟨4..0⟩ = 0) → R[rc]⟨4..0⟩:	broj n je u spremniku
	(c3⟨4..0⟩ ≠ 0) → c3⟨4..0⟩):	broj n je u naredbi

Sljedeća tablica prikazuje kako se realizira ova naredba.

Korak	RTN
T0 –T2	Dohvat naredbe
T3	$n \leftarrow IR\langle 4..0\rangle$;
T4	$(n = 0) \rightarrow (n \leftarrow R[rc]\langle 4..0\rangle)$;
T5	$C \leftarrow R[rb]$;
T6	Shr (:= $n \neq 0 \rightarrow (C\langle 31..0\rangle \leftarrow 0\#C\langle 31..0\rangle$: $n \leftarrow n - 1$; Shr));
T7	$R[ra] \leftarrow C$;

Ponovo sklopovi u četvrtom, T3, i petom, T4, koraku određuju broj n. Također kako se sklopovski realizira n posmaka za jedno mjesto u desno naknadno će biti opisano.

Apstraktan RTN opisuju broj n na sličan način kao što je opisan uvjet kod naredbe za grananje, ali konkretan RTN uvodi dodatni spremnik, $n\langle 4..0\rangle$, koji služi i kao brojilo koraka posmaka.

4.3. Implementacija putova podataka

Dok projektanti razvijaju konkretan RTN opis naredbi ujedno razmatraju detalje realizacije putova podataka potrebnih za implementaciju specificiranih naredbi. Slijedi detaljan razvoj putova podataka počevši od određenih općih napomena potrebnih u postupku projektiranja.

Projektiranje putova podataka zahtijeva odluke na više razina. Na višoj razini projektira se mikroarhitektura, odnosno definiraju se spremnici i međuveze. Također potrebno je donijeti odluku koji tip spremnika i kakve međuveze primijeniti. Općenito gledajući viša razina odluka utječe na tip prijenosa podataka između spremnika te koliko i kakvih prijenosa je moguće istovremeno obavljati. Niža razina odnosi se na brzinu kojom se prijenos može ostvariti, odnosno utječe na izbor frekvencije takta procesora.

Na višoj razini osnova je implementacija putova podataka koja uključuje znatno više od odabira broja spremnika i međuveza, iako oni određuju temeljnu karakteristiku mikroarhitekture. U tom smislu koristiti će se izrazi: jedno-, dvo- i tro-sabirnički sustavi. Daljnje odluke u projektiranju su odabir međuveze između procesora i memorije, uključenje nezavisnog sklopa za inkrementiranje programskog brojila, itd. Ove odluke utječu na mogućnost implementacije cjevovoda te drugih paralelnih aktivnosti koje se događaju u procesoru.

Niža razina vezuje se uz odabir implementacijske domene spremnika i sabirnica. Ovdje se određuje kojim signalima se aktiviraju pojedini sklopovi kako bi izveli zadane radnje. Ovi upravljački signali postavljaju sadržaj spremnika na sabirnicu, upisuju sadržaj sa sabirnice u spremnik, određuju ALU koju operaciju mora izvesti, upravljaju aktivnostima vezanim uz pristup memoriji, itd. Upravljački signali su međuveza između upravljačke jedinice (*control unit*) i putova podataka (*data paths*). Iz tog razloga značajni su u projektiranju putova podataka.

U cilju jednostavnosti razmatranja ograničiti će se na slučaj implementacije putova podataka pomoću flip-flopova s izlazima s tri stanja. Različite varijante implementacije funkcija unutar i izvan integriranog kruga neće se analizirati u detalje.

Domena implementacije spremnika. Na razini implementacije spremnika temeljna odluka je odabir između flip-flopova upravljanih razinom i upravljanih bridom. Sklopovi upravljani

razinom zauzimaju manje mjesta u integriranom sklopu od istih upravljanih bridom. Jedina iznimka su TTL sklopovi koji obično bez obzira na izvedbu imaju približno jednak broj komponenti. Značajna prednost primjene spremnika upravljanih bridom je jednostavnije vremensko vođenje (*timming*). Uzlazni ili silazni brid takt signala tada se koristi za upis sadržaja u spremnik.

Izbor tipa flip-flopa utječe i na konkretne RTN korake. Flip-flopovi upravljani razinom, obično se nazivaju "*latch*" obično ne dozvoljavaju promjenu stanja na ulazima za vrijeme aktivne razine na ulazu za upisivanje. Konkretno, stanje upisano u navedeni spremnik tada nije u funkciji njegovog prethodnog stanja. Tako prijenos C $\leftarrow$ A + C; implicira da je C spremnik realiziran flip-flopom upravljan bridom. "*Master-slave*" flip-flopovi mogu se također koristiti za ovu primjenu. Neznatna razlika između upravljanja bridom i "*master-slave*" principa rada nevažna je u slučaju D flip-flopa koji će se primarno koristiti za implementaciju putova podataka.

4.1. Logičko projektiranje jednosabirničkog SRC

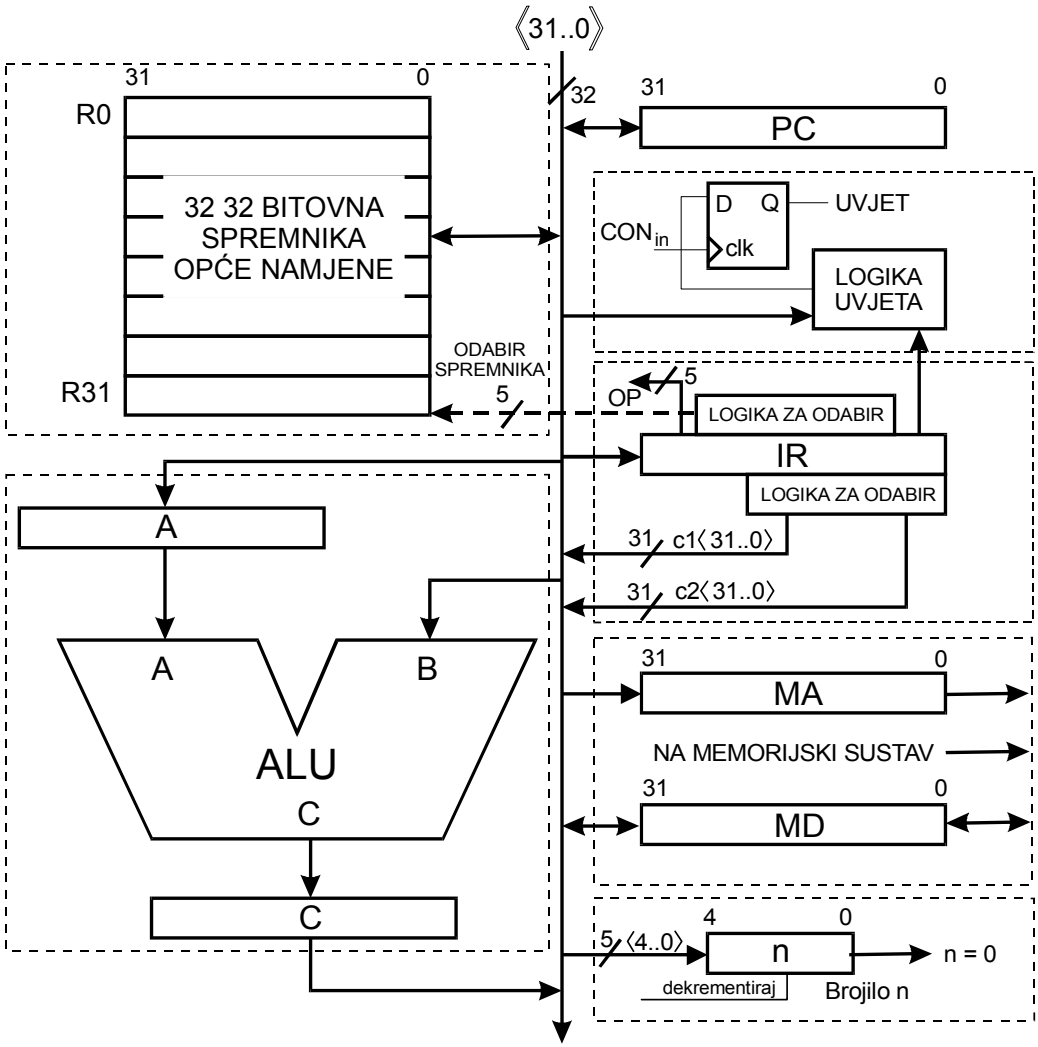
Prethodna razmatranja odnosila su se na projektiranje SRCa na općenitoj razini. Slijedi projektiranje na razini logičkih vrata. Temeljen blokovskog prikaza procesora slijedi razrada pojedinih blokova na razini logičkih vrata. Započeti će se projektiranjem sklopovlja putova podataka, međuveze s memorijom, spremnika naredbe temeljn čijeg sadržaja se generira većina upravljačkih signala. Sljedeći korak je projektiranje upravljačke jedinice.

4.4.1. Uvod u postupak projektiranja jednosabirničkog SRCa

Sljedeća slika prikazuje detaljnije predloženu jednosabirničku strukturu putova podataka SRCa. Slika je slična prethodnim prikazima uz dodatne detalje. Programsko brojilo sadrži adresu sljedeće naredbe, a spremnik naredbe tekuću naredbu. Temeljen sadržaja spremnika naredbe definira se operacija, odabiru se spremnici, generiraju se konstante potrebne za izvođenje naredbe. Spremnici MA (*memory address*) i MD (*memory data*) i njima pripadajući signali koriste se kao međuveza procesora i memorije. MA je jednosmjerni spremnik u koji se upisuje sadržaj sa interne sabirnice procesora i postavlja memorijska adresa na vanjsku adresnu sabirnicu. MD je dvosmjerni spremnik u koji se upisuje s interne sabirnice procesora i vanjske podatkovne sabirnice. U 5-bitovni spremnik n upisuje se informacija o posmaku, odnosno broj mjesta za koje je potrebno posmak izvršiti. On je projektiran kao spremnik čiji sadržaj se dekrementira. Uvjetna logika koristi posljednja tri bita spremnika naredbe kao i sadržaj sa sabirnice kako bi odredila da li je uvjet grananja ispunjen. Uvjet se zapisuje u jednobitovni spremnik, D flip-flop.

4.4.2. Projektiranje SRCa na razini logičkih vrata

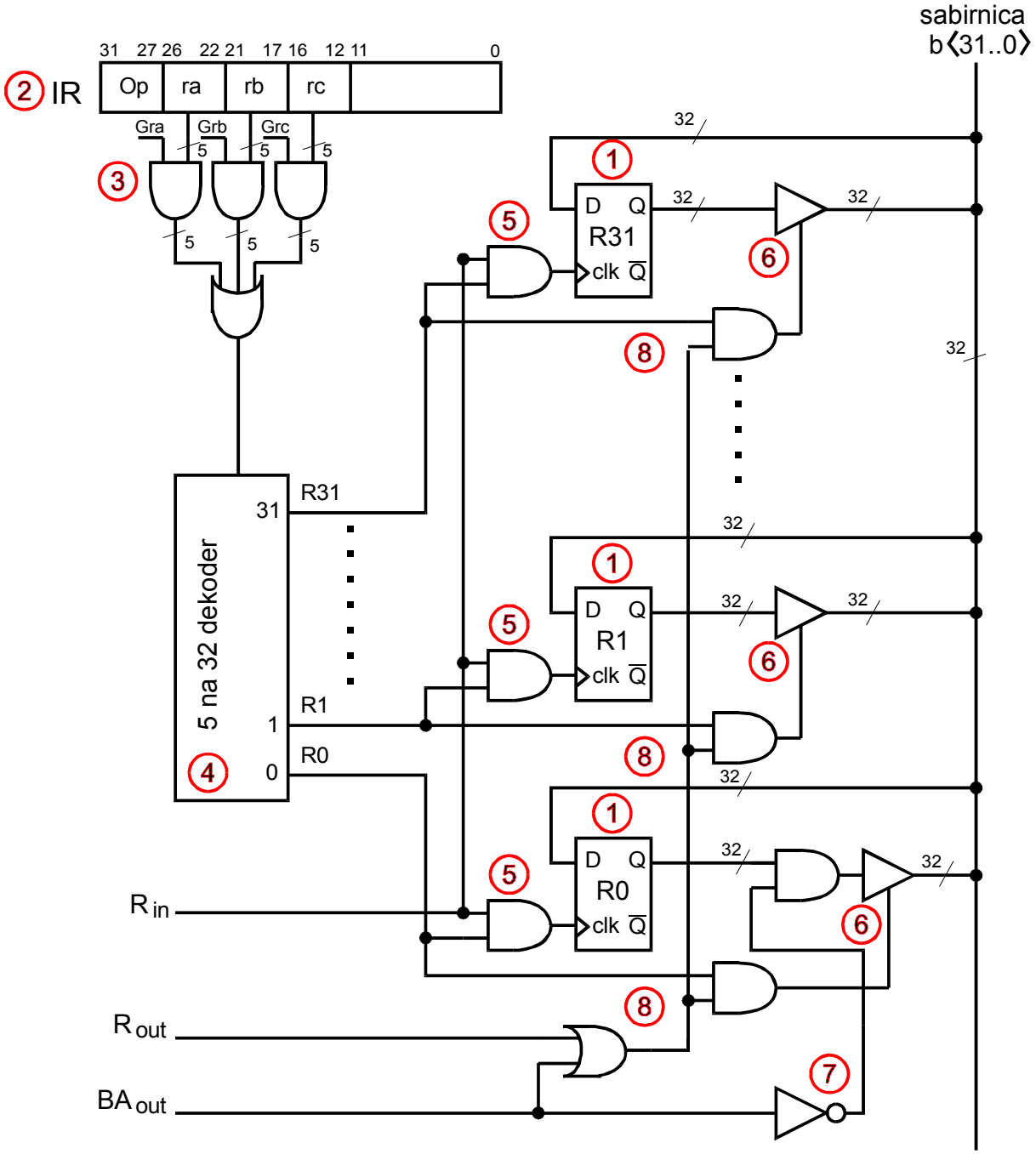
U ovom dijelu detaljnije će se razraditi blokovi prikazani prethodnom slikom, opisati upravljački signali kojima se implementiraju naredbe opisane u prethodnim poglavljima. Projektiranje će započeti sa spremnicima opće namjene, zatim će se obraditi postupak proširenja na 32 bita konstanti c1 i c2 koje su sastavni dio naredbe kao i izdvajanje operacijskog koda naredbe. U sljedećem koraku projektirati će memorijski međusklop. Temeljen projektiranih sklopova opisati će se postupak dohvata i izvođenja naredbi, **add**, **addi**, **ld** i **st**. Nakon što se projektira n spremnik koji se koristi kod naredbi za posmak, opisati će se postupak dohvata i izvođenja naredbe **shr**. Ovo razmatranje zaključiti će se projektiranjem spremnika uvjeta i opisom postupka dohvati i izvedi naredbe **br**.



spremnici opće namjene. Sljedeća slika prikazuje detaljniji prikaz spremnika opće namjene R[0..31]⟨31..0⟩ ①. Polja ra, rb i rc u spremniku naredbe, ②, specificiraju izvorišne i odredišne spremnike. Upravljački signali Gra, Grb i Grc, ③, spajaju jedno od polja na 5 na 32 dekoder, ④, koji određuje koji je spremnik odabran. Ovaj signal koristi se kod upisa sa sabirnice u odabrani spremnik, ⑤, kao i kod postavljanja sadržaja odabranog spremnika na sabirnicu, ⑥. Kod upisa u spremnik koristi se i signal R_{in}, a kodpostavljanja sadržaja spremnika na sabirnicu signal R_{out}. Upravljački signal BA_{out} (BA – *Base Address*) koristi prilagodbi proračuna bazne adrese iz koje se proračunava adresa:

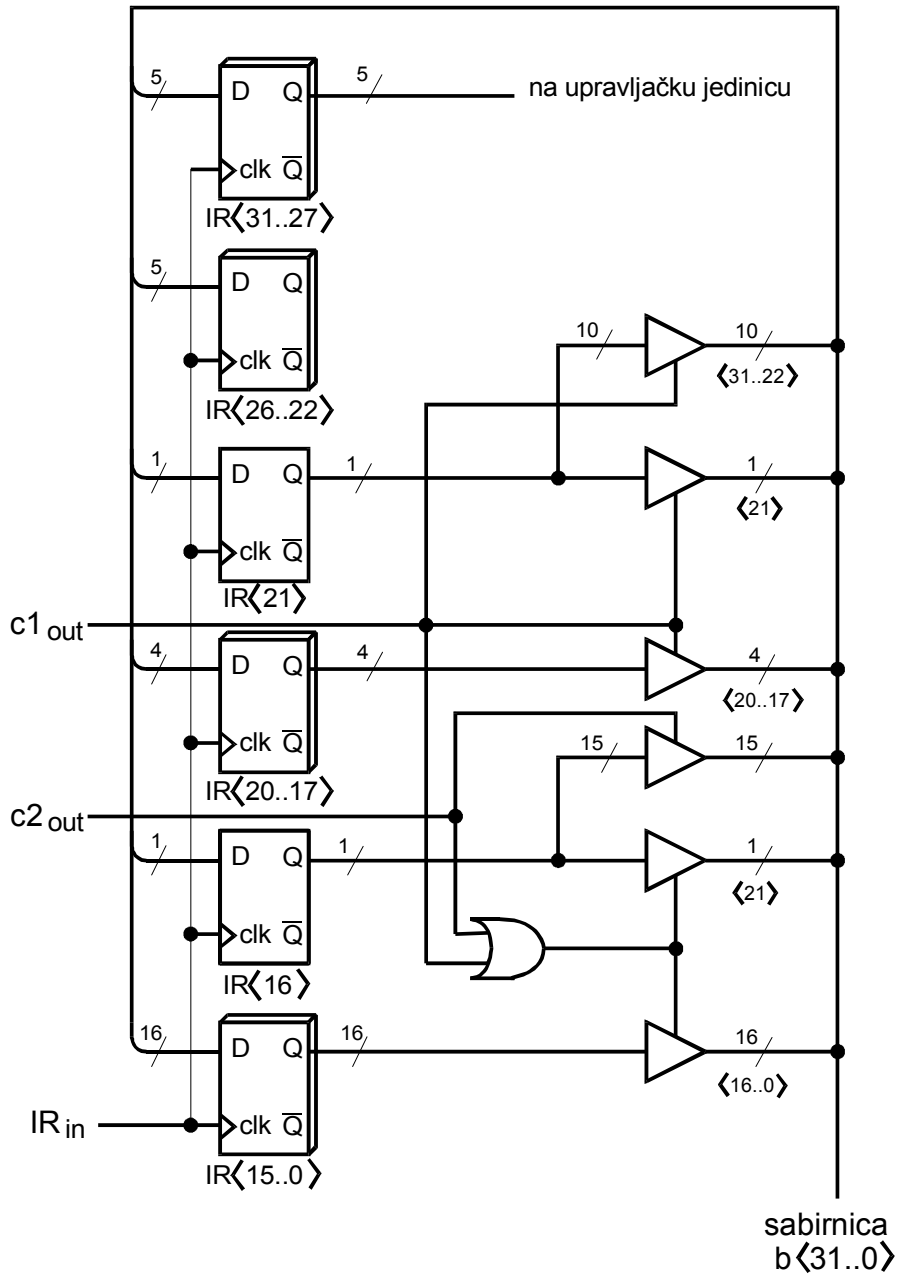
disp⟨31..0⟩ := ((rb = 0) → c2⟨16..0⟩ {proširenje predznaka}:
(rb ≠ 0) → R[rb] + c2⟨16..0⟩ {proširenje predznaka, 2'gi komplement});

BA_{out} postavlja nule na sabirnicu ako je R0 odabran, ⑦, ili sadržaj odabranog spremnika ukoliko je selektiran jedan od spremnika R1..R31, ⑧.



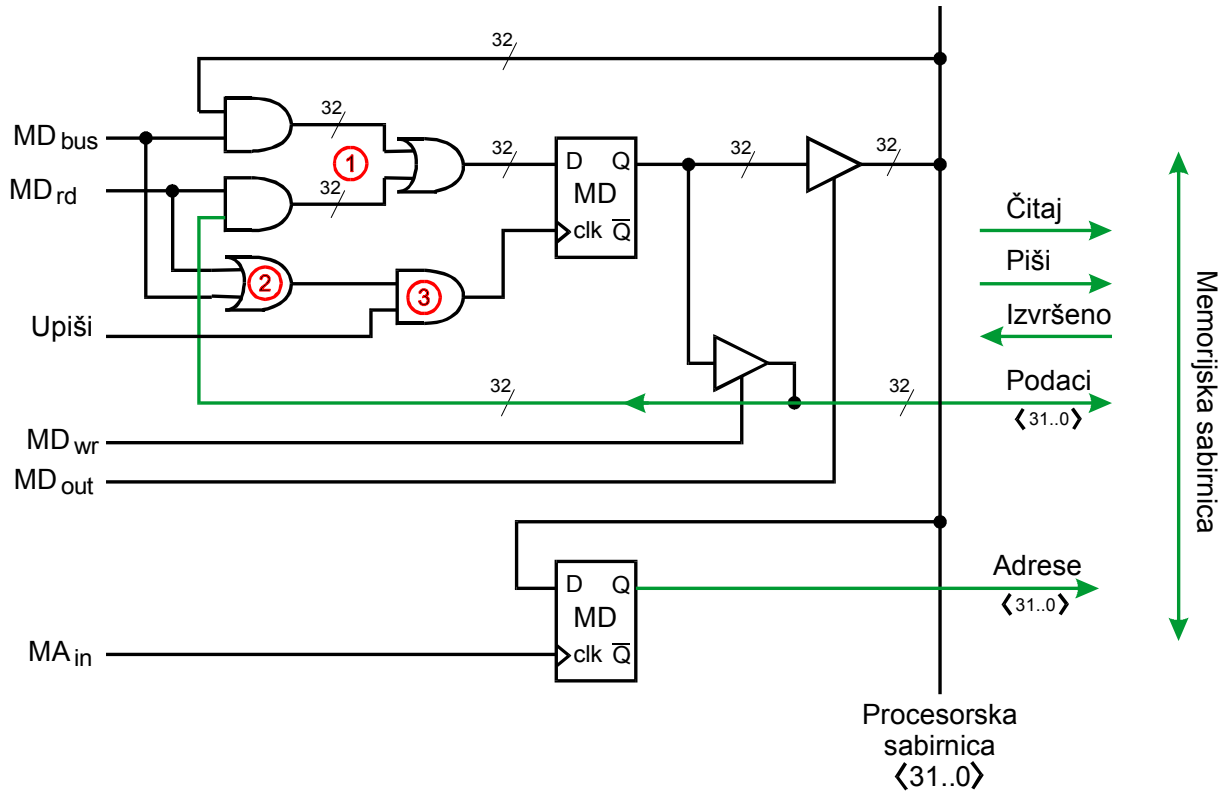
Selektiranje Op i konstanti c1 i c2 iz spremnika naredbe. Sljedeća slika prikazuje postupak izdvajanja operacijskog koda, IR⟨31..27⟩ i konstanti c1 i c2 iz spremnika naredbe. Na slici je odstupljeno od standardnog prikaza pa su zbog preglednosti višebitovni D flip-flpovi prikazani trodimenzionalno kako bi ih se razlikovalo od jednobitovnih. Prema definiciji konstante c1 i c2 su:

c1⟨21..0⟩ := IR⟨21..0⟩	dugo polje pomaka
c1⟨16..0⟩ := IR⟨16..0⟩	kratko polje pomaka, neposredna konstanta



Konstantama c1 i c2 potrebno je proširiti predznak na 32 bita kako bi se mogle koristiti u aritmetičkim operacijama. U prikazu 2'gog komplementa ovo znači kopirati msb na preostale bitove većeg značenja. Sklopovi prikazani na prethodnoj slici rješavaju navedeno spajanjem 22 bita na linije podatkovne sabirnice ⟨21..31⟩ odnosno 17 bita na linije podatkovne sabirnice ⟨16..31⟩. Operacijski kod naredbe direktno se vodi na upravljačku jedinicu.

Memorijski međusklop. Sljedeća slika prikazuje izvedbu memorijskog međusklopa na razini logičkih vrata.



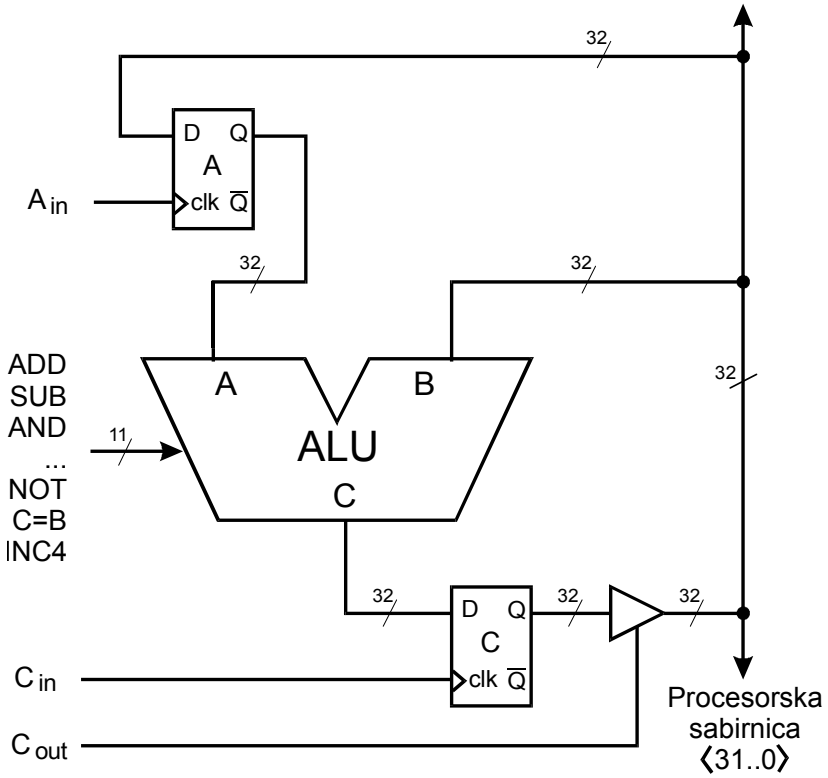
U spremniku MA (*memory address*) upisana je adresa memorijske lokacije kojoj se pristupa bilo radi dohvata naredbe ili operanda iz memorije ili za upis rezultata obrade u memoriju. U spremnik MA upisuje se sadržaj sa procesorske sabirnice aktiviranjem signala MA_{in}. Prema slici sadržaj spremnika je stalno prisutan na memorijskoj adresnoj sabirnici. Ukoliko postoji mogućnost da neki drugi vanjski uređaji određuju sadržaj memorijske adresne sabirnice, npr. DMA sklop, tada se na izlaz spremnika MA spaja jedan međusklop s izlazom s tri stanja.

Spremnik MD (*memory data*) je nešto složeniji. Namjena ovog spremnika je privremena pohrana podatka koji se iz memorije prenosi u procesor ili iz procesora u memoriju. Ovo je dvosmjerni spremnik. Signali MD_{bus} i MD_{rd} odabiru smjer iz kojeg se podatak upisuje u spremnik MA ①, prvi s procesorske sabirnice, a drugi s memorijske podatkovne sabirnice. U jednom trenutku samo jedan od navedenih signala smije biti aktivan. Signali se preko ILI vrata, ②, dovode na I vrata kojima se upis u spremnik sinkronizira sa signalom “upiši”, ③. Izlaz spremnika može se također spojiti ili na procesorsku sabirnicu aktiviranjem signala MD_{out}, ili na memorijsku podatkovnu sabirnicu aktiviranjem signala MD_{wr}.

Tri dodatna signala vezana su uz memorijski međusklop: Čitaj, Piši i Izvršeno. Ovi signali ne završavaju na navedenoj shemi ali su sastavni dio memorijskog međusklopa. Njihov izvor ili odredište je upravljačka jedinica. Čitaj (*Read*) signalizira memoriji da postavi sadržaj memorijske lokacije određene sadržajem MA spremnika na podatkovnu memorijsku sabirnicu. Piši (*Write*) signalizira memoriji da učita sadržaj s podatkovne memorijske sabirnice u memorijsku lokaciju određene sadržajem MA spremnika. Izvršeno (*Done*) signalizira procesoru da je memorijski sustav završio tekuću transakciju. Kod

sustava koji imaju dovoljno brzu memoriju ili je memorija poznate brzine odziva ovaj signal nije neophodan.

ALU i njoj pripadajući spremnici. Sljedeća slika prikazuje ALU i njoj pripadajuće privremene spremnike zajedno s potrebnim upravljačkim signalima.



ALU je realizirana u kombinacijskoj logici, što znači da je vrijednost na izlazu samo funkcija vrijednosti na ulazu. Odabir funkcije ostvaruje se aktiviranjem jednog od 11 upravljačkih signala. Za realizaciju stvarne ALU možda nije potrebno svih 11 upravljačkih signala budući da npr. C = B naredba samo kopira sadržaj sa sabirnice u C spremnik. Signal INC4 inkrementira vrijednost s B ulaza za 4. Ovaj signal koristi se za inkrementiranje sadržaja programskog brojila. Nakon što se stanje na izlazu ALU stabilizira (vrijeme odziva) rezultat operacije se upisuje u privremeni izlazni spremnik C. Sam izgled ALU biti će naknadno opisan u jednom od sljedećih poglavlja.

4.4.3. Upravljačka sekvenca jednosabirničkog SRCa

Nakon što je opisana sklopovska struktura jednosabirničkog SRCa slijedi definiranje upravljačke sekvence kojom se realizira procedura dohvata i izvođenja naredbe. Upravljačka sekvenca je slijed signala kojim se upravlja putovima podataka. Upravljački signali upisuju sadržaj u spremnike (*strobe*) te postavljaju sadržaj spremnika na određenim linijama (*gate*). Prethodno opisano sklopovlje polazište je za definiranje iz konkretnog RTNa upravljačke sekvence. U prvom koraku opisati će se samo dohvat naredbe, a zatim izvođenje naredbi add, addi, ld, shift i branch.

Dohvat naredbe. Sljedeća tablica prikazuje konkretan RTN i upravljačku sekvencu dohvata naredbe.

Korak	RTN	Upravljačka sekvenca
T0	MA ← PC: C ← PC + 4;	PC_{out}, MA_{in}, INC4, C_{in}
T1	MD ← M[MA]: PC ← C;	Čitaj, C_{out}, PC_{in}, Čekaj
T2	IR ← MD;	MD_{out}, IR_{in}
T3	izvođenje_naredbe	

Signal Čitaj (*Read*) dojavljuje memoriji da postavi sadržaj lokacije određene sadržajem adresne sabirnice na podatkovnu sabirnicu. Signal Čekaj (*Wait*) u koraku T1 umetnut je iz razloga što postoji mogućnost da je memorija sporija od procesora pa je potrebno čekati određeni broj taktova za njen odaziv. Sporija memorija dojavljuje da je postavila sadržaj odabrane memorijske lokacije na podatkovnu sabirnicu aktiviranjem signala Izvršeno (*Done*). Po završetku ovog signala potrebno je upisati sadržaj s podatkovne sabirnice u MD spremnik aktiviranjem signala MD_{rd} i Upiši.

Naredba za zbrajanje, add. Slijedi opis naredbe za zbrajanje sadržaja dvaju spremnika

add (:= op = 12) → R[ra] ← R[rb] + R[rc]:

Korak	RTN	Upravljačka sekvenca
T0	MA ← PC: C ← PC + 4;	PC_{out}, MA_{in}, INC4, C_{in}
T1	MD ← M[MA]: PC ← C;	Čitaj, C_{out}, PC_{in}, Čekaj
T2	IR ← MD;	MD_{out}, IR_{in}
T3	A ← R[rb];	Grb, R_{out}, A_{in}
T4	C ← A + R[rc];	Grc, R_{out}, ADD, C_{in}
T5	R[ra] ← C;	C_{out}, Gra, R_{in}, Kraj

Za primijetiti je u koraku T3 kako se signalima Grb i R_{out} odabire spremnik rb i postavlja njegov sadržaj na sabirnicu. Isti sadržaj se zatim upisuje u privremeni spremnik A aktiviranjem signala A_{in}. Slično u koraku T5 se nakon postavljanja sadržaja spremnika C na sabirnicu aktiviranjem signala C_{out}, odabire spremnika ra signalom Gra i upisuje u njega sadržaj sa sabirnice aktiviranjem signala R_{in}. Sve aritmetičke i logičke naredbe koriste sličnu upravljačku sekvencu.

U ovom tekstu koristi se indeks “out” za postavljanje sadržaja spremnika na sabirnicu ili određeni ulaz (npr. ALU) dok se indeks “in” koristi za upis u spremnik. Signal Kraj signalizira upravljačkoj jedinici da je naredba izvedena.

Zbrajanje sadržaja spremnika i konstante, addi. Naredba za zbrajanje sadržaja spremnika i konstante koja je sastavni dio naredbe

addi (:= op = 13) → R[ra] ← R[rb] + c2⟨16..0⟩ {2’gi komplement, proširenje predznaka}:

opisana je sljedećom tablicom:

Korak	RTN	Upravljačka sekvenca
T0	MA ← PC: C ← PC + 4;	PC_{out}, MA_{in}, INC4, C_{in}
T1	MD ← M[MA]: PC ← C;	Čitaj, C_{out}, PC_{in}, Čekaj
T2	IR ← MD;	MD_{out}, IR_{in}
T3	A ← R[rb];	Grb, R_{out}, A_{in}
T4	C ← A + c2 {proš. predz.}	$c2_{out}$, ADD, C_{in}

T5	R[ra] ← C;	C_{out}, Gra, R_{in}, Kraj

U koraku T4 drugi operand se dohvata iz dijela spremnika naredbe aktiviranjem signala $c2_{out}$. Konstanti se sklopovski proširuje predznak, 16 bit, na preostale bitove 17..31.

Upis u spremnik iz memorije, load. Naredba za upis sadržaja memorijske lokacije u spremnik R[ra] ← M[disp] opisana je sljedećom tablicom:

Korak	RTN	Upravljačka sekvenca
T0	MA ← PC: C ← PC + 4;	PC_{out}, MA_{in}, INC4, C_{in}
T1	MD ← M[MA]: PC ← C;	Čitaj, C_{out}, PC_{in}, Čekaj
T2	IR ← MD;	MD_{out}, IR_{in}
T3	A ← ((rb = 0) → 0: (rb ≠ 0) → R[rb];	Grb, BA_{out}, A_{in}
T4	C ← A + c2 {proš. predz.}	$c2_{out}$, ADD, C_{in}
T5	MA ← C;	C_{out}, MA_{in}
T6	MD ← M[MA];	Čitaj, Čekaj
T7	R[ra] ← MD;	MD_{out}, Gra, R_{in}, Kraj

Za analizu upravljačke sekvence potrebno je ponovo se osvrnuti na sheme odabira i realizacije spremnika opće namjene kao i memorijskog međusklopa. U koraku T3 odabire se iz dijela spremnika naredbe spremnik rb a njegov sadržaj postavlja se na sabirnicu ne pomoću signala R_{out}, nego pomoću signala BA_{out}. Ovim rješenjem se na sabirnicu postavlja sadržaj odabranog spremnika ukoliko nije odabran R[0] kada se na sabirnicu postavljaju sve 0. Ova vrijednost se upisuje u privremeni spremnik A. U sljedećem koraku, T4, na sabirnicu se postavlja iz spremnika naredbe konstanta c2 kojoj se proširi predznak i ona se pribroji sadržaju privremenog spremnika A. Na kraju ovog koraka u privremenom spremniku C upisana je efektivna adresa operanda iz memorije. Posljednja tri koraka slična su dohvatu naredbe.

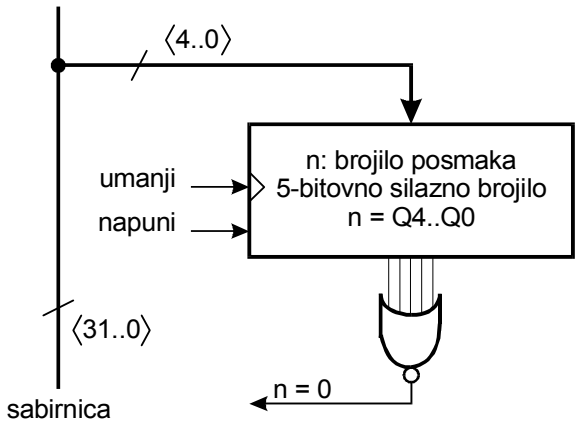
Naredba za upis sadržaja spremnika na memorijsku lokacije, store, slična je opisanoj naredbi upisa sadržaja memorijske lokacije u spremnik. Razlika je u koracima T6 i T7. U koraku T6 upisuje se sadržaj odabranog spremnika u MD spremnik aktiviranjem signala MD_{bus} i Upiši. U sljedećem koraku T7 generira se signal Piši (*Write*), a signal Čekaj podrazumijeva odgovor memorije da je upisan sadržaj s podatkovne sabirnice postavljanjem signala Izvršeno.

Naredba za posmak, shift. Kod naredbi za posmak broj mjesta posmaka određen je ili s posljednjih pet bita naredbe IR⟨4..0⟩ ili specificiranog spremnika rc, u ovisnosti da li je IR⟨4..0⟩ = 0, odnosno n jr specificiran sljedećim apstraktnim RTNom:

n := ((c3⟨4..0⟩ = 0) → R[rc]⟨4..0⟩: broj n je sastavni dio spremnika R[rc]
((c3⟨4..0⟩ ≠ 0) → c3⟨4..0⟩); broj n je sastavni dio spremnika naredbe

Sklopovsko rješenje ALU dozvoljava samo posmak za jedno mjestu u lijevo ili desno, shr, shra, shl, shla. Za realizaciju posmaka potrebno je imati brojilo, odnosno spremnik koji će se dekrementirati nakon svakog pojedinog posmaka. Sljedeća slika prikazuje izgled brojila posmaka.

Brojilo se može napuniti bilo sadržajem 5 bita najmanjeg značenja programskog brojila odnosno ukoliko su svi bitovi nula 5 bita najmanjeg značenja specificiranog spremnika.



Upravljačka sekvenca naredbe za posmak u desno, shr, opisana je sljedećom tablicom:

Korak	RTN	Upravljačka sekvenca
T0	MA ← PC: C ← PC + 4;	PC_{out}, MA_{in}, INC4, C_{in}
T1	MD ← M[MA]: PC ← C;	Čitaj, C_{out}, PC_{in}, Čekaj
T2	IR ← MD;	MD_{out}, IR_{in}
T3	n ← IR⟨4..0⟩;	$c1_{out}$, napuni
T4	n = 0 → (n ← R[rc]⟨4..0⟩);	n = 0 → (Grc, R_{out}, napuni)
T5	C ← R[rb];	Grb, R_{out}, C = B, C_{in}
T6	Shr (:= n ≠ 0 → (C⟨31..0⟩ ← 0#C⟨31..1⟩: n ← n – 1; Shr));	n ≠ 0 → C_{out}, SHR, C_{in}, umanji, GotoT6
T7	R[ra] ← C;	C_{out}, Gra, R_{in}, Kraj

U prvom koraku izvođenja naredbe, T3, postavlja se na sabirnicu konstanta c1 iz spremnika naredbe te se njenih prvih pet bita upisuje u brojilo. Ukoliko je upisni broj jednak nuli, signalizira se upravljačkoj jedinici te se u sljedećem koraku postavlja na sabirnicu sadržaj spremnika specificiranog poljem naredbe rc te se njegovih pet bita najmanjeg značaja upisuje u brojilo. U sljedećem koraku se sadržaj spremnika rb upisuje preko ALU u privremeni spremnik C. U koraku T6 izvodi se posmak sadržaja privremenog spremnika C za jedno mjesto, a rezultat se upisuje u isti spremnik. Ujedno se dekrementira brojilo. Ovaj korak ponavlja se n puta, odnosno sve dok sadržaj brojila nije jednak nuli. U posljednjem koraku naredbe prebacuje se sadržaj privremenog spremnika C u odredišni spremnik ra.

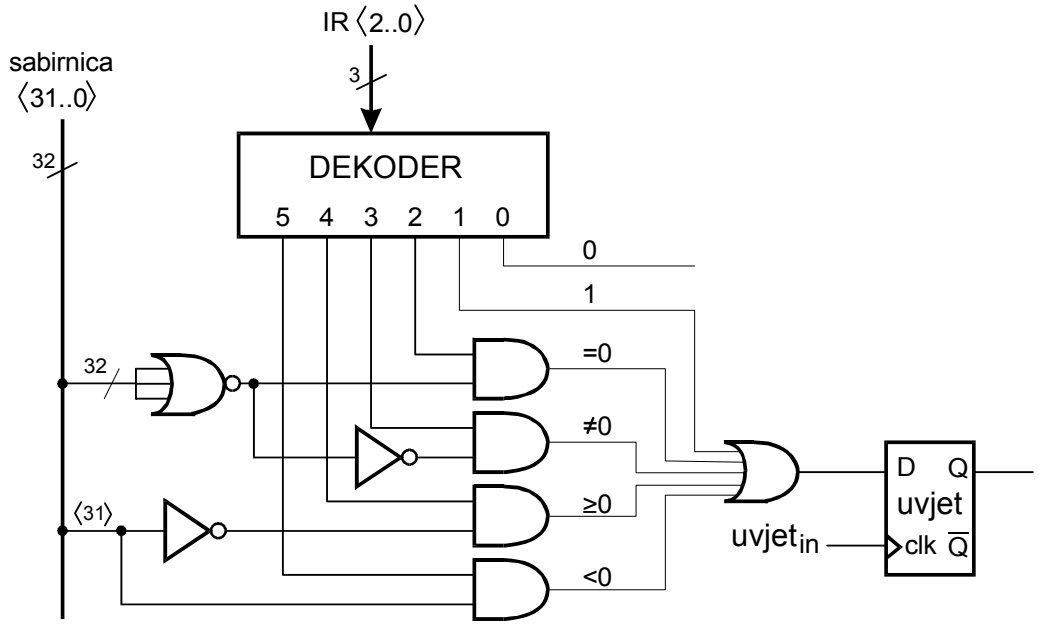
Kod realizacije ove naredbe potrebno je obratiti osobitu pažnju na ispravno vremensko vođenje koraka T6. Naime dekrementiranje sadržaja brojila mora se realizirati na samom kraju ciklusa kako se na bi prijevremeno promijenio signal n. Posebnom sklopovskom realizacijom moguće je riješiti posamk za više mjesta u jednom taktu.

Grananja. Uvjet grananja određen je s prva tri bita spremnika naredbe c3⟨2..0⟩, a sadržaj rc spremnika se uspoređuje prema uvjetu grananja.

uvjet := (c3⟨2..0⟩ = 0 → 0: nikad
c3⟨2..0⟩ = 1 → 1: uvijek

c3⟨2..0⟩ = 2 → R[rc] = 0:	nula
c3⟨2..0⟩ = 3 → R[rc] ≠ 0:	različit od nule
c3⟨2..0⟩ = 4 → R[rc]⟨31⟩ = 0:	veći od nule
c3⟨2..0⟩ = 5 → R[rc]⟨31⟩ = 1:	negativan

Na sljedećoj slici prikazan je sklop koji određuje da li je zadovoljen uvjet grananja.



Logika prikazana na slici automatski određuje da li je zadovoljen uvjet i rezultat pohranjuje u D flip-flop nazvan “uvjet”. Dekoder odabire temeljen prva tri bita spremnika naredbe koji je uvjet odabran. Izlaz 1 dekodera direktno se preko ILI vrata spaja na “uvjet” flip-flop te je on obavezno postavljen ukoliko je vrijednost IR⟨2..0⟩ = 1, odnosno bezuvjetno grananje. Ukoliko je IR⟨2..0⟩ = 2 tada se odabire prva I vrata preko 2 izlaza dekodera. Na drugi ulaz I vrata spojen je NILI sklop s 32 ulaza čiji izlaz je jednak jedinici samo ako su sva 32 bita jednaka nuli, odnosno ovaj sklop uspoređuje sadržaj sabirnice s nulom. Ulazi drugih I vrata spojeni su na izlaz broj 3 dekodera i na izlaz invertora čiji ulaz je izlaz NILI sklopa koji uspoređuje sadržaj sabirnice s nulom. Ovaj dio sklopa uspoređuje da li je sadržaj sabirnice različit od nule. Usporedba da li je sadržaj sabirnice pozitivan ili negativan ostvaruje se usporedbom bita najvećeg značenja sabirnice. Sinkronizacija postavljanja sadržaja flip-flopa “uvjet” realizira se pomoću signala $uvjet_{in}$.

Naredba za grananje ima sljedeći apstraktni RTN opis:

br (:= op = 8) → (uvjet → PC ← R[rb]): uvjetno grananje

Sljedeća tablica prikazuje i konkretan RTN naredbe za grananje:

Korak	RTN	Upravljačka sekvenca
T0	MA ← PC: C ← PC + 4;	PC_{out}, MA_{in}, INC4, C_{in}
T1	MD ← M[MA]: PC ← C;	Čitaj, C_{out}, PC_{in}, Čekaj
T2	IR ← MD;	MD_{out}, IR_{in}

T3	uvjet ← uvjet(R[rc]);	Grc, R_{out}, $uvjet_{in}$
T4	uvjet → PC ← R[rb];	Grb, R_{out}, uvjet → PC_{in}, Kraj

Upravljačka sekvenca naredbe za grananje je relativno jednostavna s obzirom da sklopovlje izračunava uvjet. U prvom koraku izvođenja naredbe, T3, postavlja se na sabirnicu sadržaj spremnika rc i rezultat usporedbe se pohranjuje u "uvjet" flip-flop. U sljedećem koraku na sabirnicu se postavlja sadržaj spremnika rb u kojem je upisana adresa grananja. Ukoliko je uvjet grananja zadovoljen, odnosno uvjet = 1, tada se sadržaj sabirnice upisuje u programsko brojilo. Ovu funkciju jednostavno je realizirati jednim I vratima.

4.4.4. Pregled postupka projektiranja procesora

Nakon analiza provedenih u ovom i prethodnim poglavljima može se napraviti pregled koraka u projektiranju procesora. Proces projektiranja počinje s opisom zadataka procesora, odnosno neformalnim opisom njegovih funkcija, a završava projektom logičkih sklopova kojima se realiziraju navedene funkcije. Sljedeći su koraci u procesu projektiranja jednog procesora:

1. Neformalni opisa procesora i njegovih naredbi.
2. Formalan opis naredbi na programerskoj razini, apstraktan RTN.
3. Specifikacija putova podataka na razini višeg jezika kojim je moguće ostvariti apstraktno opisan procesor iz prethodne točke.
4. Definiranje konkretnog RTNa za svaku pojedinu naredbu polazeći od određene konfiguracije putova podataka.
5. Projektiranje logičkih sklopova polazeći od konkretnog RTN opisa naredbi i specifikacija potrebnih upravljačkih signala.
6. Definiranje upravljačke sekvence svake pojedine naredbe.
7. Projektiranje upravljačke jedinice koja generira prema zadanom vremenskom dijagramu upravljačku sekvencu.

Prva dva koraka ovog procesa opisana su u drugom poglavlju. Točke 3, 4, 5, 6 i 7 obrađene su u četvrtom poglavlju, 3. i 4. točka u potpoglavlju 2, točke 5 i 6 u potpoglavlju 4 do će se projektiranje upravljačke jedinice obraditi u sljedećem 5. potpoglavlju.

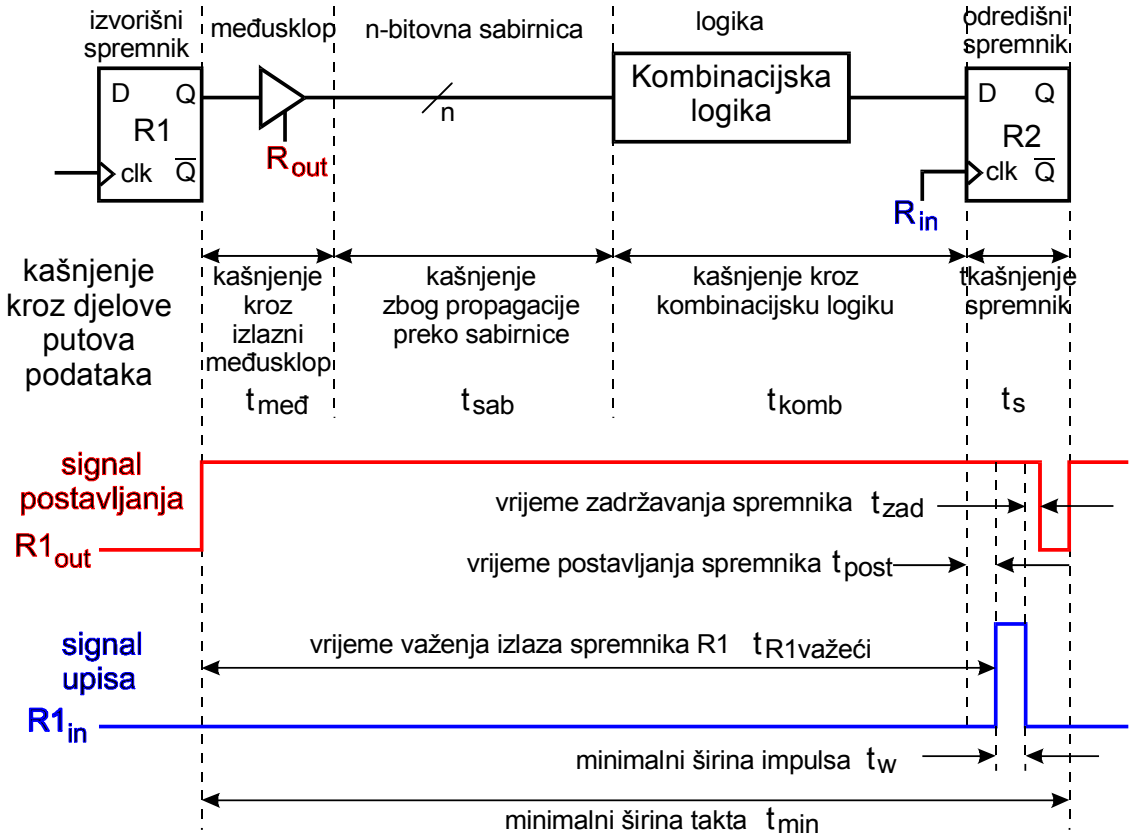
I ovdje je potrebno napomenuti da rješenja u jednom koraku značajno utječu na moguća rješenja sljedećeg koraka. Iz toga razloga ponekad je potrebno preinačiti rješenja prethodnog koraka ukoliko se pokaže da to rješenje značajno usložnjava sljedeći korak. Tako npr. sklopovlje pojedinih spremnika može se bitno pojednostavniti uz manje promjene konkretnog RTN opisa naredbe. Slično ukoliko se predefinira skup naredbi može se značajno utjecati na vremensko vođenje ciklusa izvođenja naredbi.

4.5. Upravljačka jedinica

Kroz prethodna razmatranja definirana je upravljačka sekvenca. U ovom potpoglavlju slijedi projektiranje logičkog sklopa koji ima zadatak generirati projektiranu upravljačku sekvencu. U prvom koraku opisati će se općenita problematika vezana uz vremenski raspored slijeda impulsa koji mijenjaju stanja u procesoru.

4.5.1. Upravljački signali i njihov vremenski raspored (*clocking & timing*)

Podaci se obrađuju na način da se prenose iz jednog spremnika preko sabirnice potencijalno kroz određenu kombinacijsku logiku u drugi spremnik. Navedene aktivnosti prijenosa podatka zahtijevaju određeno vrijeme, odnosno vrijeme postavljanja na sabirnicu, prijenosa kroz kombinacijsku logiku i upisa u odredišni spremnik. Jedan od zadatak projektanta logičkih sklopova je da što preciznije odredi navedena vremena kako bi se u što kraćem vremenu realizirala obrada podatka. Sljedeća slika prikazuje primjer vremenskog rasporeda aktivnosti koji se događaju duž putova podataka.



Za objašnjenje događanja na putovima podataka pretpostaviti će se da se radi u spremnicima u koje se upisuje na razinu signala, logička jedinica, iako se prema oznakama na sklopovima radi o spremnicima u koje se upisuje na brid. Ovo odstupanje napravljeno je iz razloga što je na ovaj način prikladnije opisati vremenski slijed događanja na putovima podataka. Slika prikazuje kašnjenje podatka koji se iz spremnika R1 pomoću sabirničkog međusklopa (sklopa s tri stanja) preko sabirnice prenosi na obradu kombinacijskoj logici. Rezultat obrade pohranjuje se u odredišni spremnik R2. Svaka od navedenih dijelova unosi određeno vremensko kašnjenje:

1. Kašnjenje kroz sabirnički međusklop, odnosno izlazni međusklop spremnika R1, $t_{međ}$.
2. Kašnjenje zbog propagacije signala preko sabirnice, t_{sab}.
3. Vrijeme propagacije kroz kombinacijsku logiku, t_{komb}.
4. Vrijeme potrebno da se signal upiše u odredišni spremnik, t_s.

Procjena minimalnog takt perioda. Minimalni period takta za zadani prijenos podatka određen je vremenom propagacije signala preko putova podataka:

$$t_{min} = t_{među} + t_{sab} + t_{komb} + t_s$$

Procjena najkraćeg mogućeg signala za upis. Podatak je prisutan i spreman za upis u odredišni spremnik R2 nakon vremena potrebnog da se propagira kroz izlazni sabirnički međusklop spremnika R1, preko sabirnice i kroz kombinacijsku logiku, te vremena potrebnog da spremnik signal postavljen na ulaz prihvati (*setup time*):

$$t_{R2važeći} = t_{među} + t_{sab} + t_{komb} + t_{post}$$

Nakon što protekne navedeno vrijeme potrebno je postaviti signal za upis u spremnik koji mora trajati minimalno, t_w. Nakon vremena odziva spremnika R2 obrađeni podatka je upisna u odredišni spremnik.

Minimalno vrijeme spajanja izlaza spremnika R1 na sabirnicu. U intervalu od početka sabirničkog ciklusa pa do okončanja upisa obrađenog podatka u odredišni spremnik potrebno je držati sabirnički međusklop spremnika R1 otvoren, odnosno signal R_{out} aktivan

$$t_{R1out} = t_{među} + t_{sab} + t_{komb} + t_{post} + t_{zad}$$

Ipak ovakva prosudba je dosta konzervativna. U praksi ovaj signal se dosta ranije deaktivira iz razloga što je potrebno uzimati u obzir efekte disperzije signala kroz sustav.

Primjeri vremena kašnjenja i propagacije. U sljedećoj tablici dani su brojčani podaci vremena kašnjenja i propagacije TTL brze logike (*Fast*) proizvođača National Semiconductor te GaAs logičkih sklopova proizvođača VITESSE:

Naziv	Parametar	F TTL	GaAs
Kašnjenje kroz međusklop	$t_{među}$	5 ns	150 ps
Vrijeme propagacije preko sabirnice	t_{sab}	5 ns	500 ps
Propagacija kroz kombinacijsku logiku	t_{komb}	14 ns	400 ps
Kašnjenje spremnika	t_{spr}	6 ns	440 ps
Vrijeme postavljanja spremnika	t_{post}	2 ns	146 ps
Vrijeme zadržavanja spremnika	$t_{zadrž}$	3 ns	104 ps
Širina signala upisa	t_w	4 ns	-

Tablica pokazuje da su GaAs logički sklopovi za red veličine brži od TTL sklopova.

Primjer: Izračunaj maksimalnu frekvencija takta za sklop prikazan na prethodnoj slici prema podacima iz tablice.

Za F TTL sklopove minimalni period takta iznosi:

$$t_{min} = t_{među} + t_{sab} + t_{komb} + t_s = 5 + 5 + 14 + 6 = 30\,\text{ns}.$$

Ovoj vrijednosti praktički se obično doda još 10% radi sigurnosti što ukupno čini 33 ns, odnosno maksimalna frekvencija takta je 30MHz.

Za GaAs VITESSE logiku minimalni period takta iznosi:

$$t_{min} = t_{među} + t_{sab} + t_{komb} + t_s = 0.15 + 0.5 + 0.4 + 0.44 = 1.49\,\text{ns}.$$

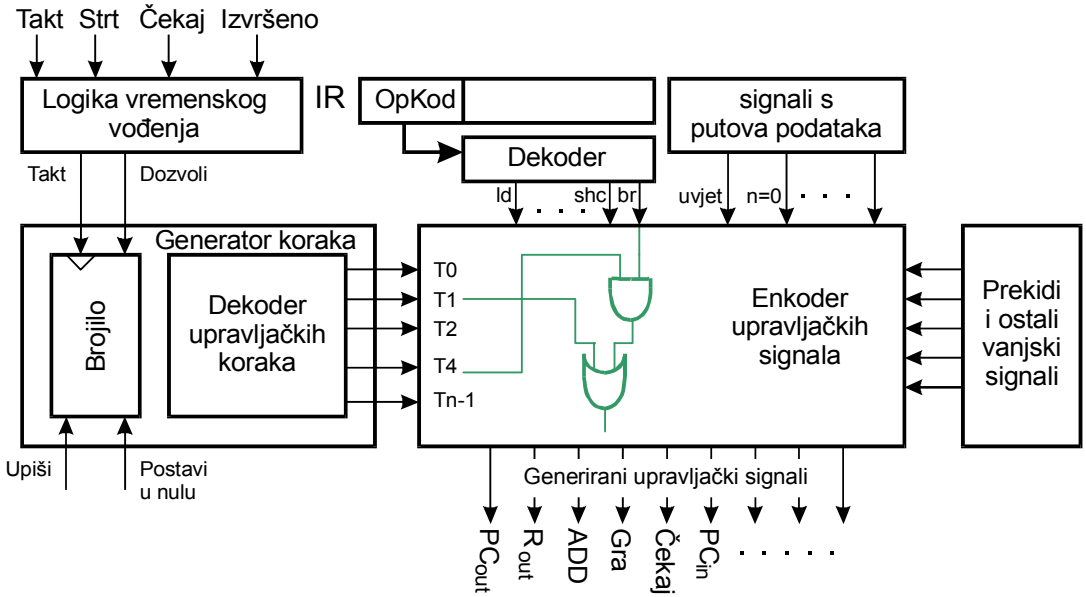
Ukoliko se i ovoj vrijednosti doda 10% radi sigurnosti dobiva se minimalni period takta 1.6 ns, odnosno maksimalna frekvencija 625 MHZ.

Prijenos podataka između spremnika primjenom logike upravljane razinom (*level triggered*). Prema opisanom vremenskom dijagramu vođenja procesa prijenosa podatka iz spremnika u spremnik nije moguće podatak iz jednog spremnika postaviti na sabirnicu propustiti kroz kombinacijsku logiku i upisati u isti spremnik. Ovo je npr. potrebno pri realizacije naredbe za posmak gdje se sadržaj privremenog C spremnika propusti kroz kombinacijsku logiku i rezultat ponovo upiše u C spremnik. Naime, nakon upisa nove vrijednosti u spremnik postoji vjerojatnost da nova vrijednost ponovo prođe kroz krug obrade i bude upisana u spremnik. Projektanti ovaj problem rješavaju uporabom spremnika u koje se upisuje na brid umjesto na razinu (*edge triggered*) ili primjenom dvofaznog takta.

Prijenos podataka između spremnika primjenom logike upravljane bridom (*edge triggered*). Primjena spremnika u koje se upisuje bridom pojednostavnjuje projektiranje sklopovlja za vremensko vođenje procesa prijenosa podatka. Naime, isti signal može se koristiti za postavljanje sadržaja spremnika na sabirnicu i upis rezultata obrede natrag u spremnik ukoliko se upis izvodi na padajući brid signala.

Upravljačka jedinica jednosabirničkog SRCa

Upravljačka jedinica je u svakom pogledu srce procesora. Ona prihvaća kao ulaz signale koje opisuju što procesor mora raditi te u kojem se trenutno stanju nalazi, te generira signale potrebne za vođenje rada procesora. Sljedeća slika prikazuje na blokovskoj razini upravljačku jedinicu SRCa s ulaznim i izlaznim signalima.



Izlazni signali iz upravljačke jedinice su upravljačka sekvenca naredbe koja se izvodi. Oni se generiraju temeljen četiri osnovna izvora informacija:

1. Polja operacijskog koda naredbe, odnosno temeljen pet bita najvećeg značenja spremnika naredbe, IR.

2. Preostalih signala s putova podataka. Tu spada rezultat usporedbe uvjeta, uvjet, posmačno brojilo n, tj uvjetni kodovi kod procesora sa statusnim i uvjetnim spremnikom.
3. Podatak o koraku naredbe koji se trenutno izvodi, odnosno izlaz iz generator koraka.
4. Vanjski signali kao što su Strt, Čekaj, Izvršeno, kao i zahtjev za prekidom, Int, i zahtjev za postavljanjem u početno stanje, Reset.

Enkoder upravljačkih signala. Enkoder upravljačkih signala koristi navedene ulazne signale kako bi generirao ispravnu upravljačku sekvencu. Enkoder je izrađen u kombinacijskoj logici i specificira se logičkim, Booleovim, funkcijama ovisnosti izlaza o ulazu. Sklopovska realizacija moguća je čvrsto povezanim sklopovima (*hardwired controller*) ili pomoću zasebnog mikroprogramiranog upravljačkog sklopa. Posljednje rješenje spada u skupinu zasebnih procesora koji će se posebno analizirati.

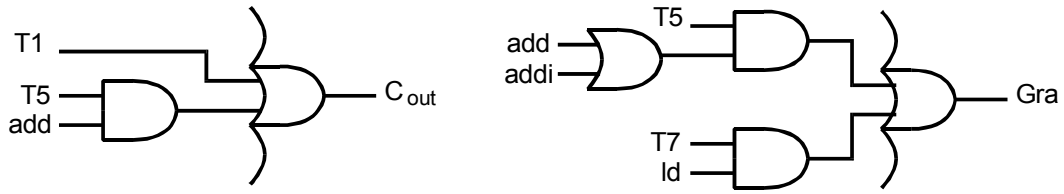
Upravljački signali generiraju se temeljen analize upravljačkih sekvenci svih naredbi koje procesor može izvesti. Neke od sekvenci opisane su prethodnim tablicama. Za svaki pojedini izlazni signal piše se logički izraz koji uvjetuje njegovo postavljanje. Tako npr. signal C_{out} pojavljuje se u svakom prvom koraku, T0, te u šestom koraku naredbe za zbrajanje itd., odnosno:

$$C_{out} = T1 + T5 \cdot add + ...$$

Sličan izraz može se izvesti za signal Gra koji se pojavljuje u koraku T5 naredbi add i addi, u koraku T7 naredbe ld itd., odnosno

$$Gra = T5 \cdot (add + addi) + T7 \cdot ld + ...$$

Logika kojom se realiziraju spomenuti upravljački signali prikazana je sljedećom slikom:



Za procesore s većim brojem naredbi posljednja ILI vrata mogu imati značajni broj ulaza što postavlja posebne zahtjeve na njihovu sklopovsku realizaciju. Naime faktor grananja na ulazu (*fan in*) je velik te je potrebno sklop realizirati s više ulaznih razina ili s vezanom logikom.

Pojedini signali koji se generiraju na putovima podataka zahtijevaju poseban tretman. Tako npr. signal Kraj koji označava kraj izvođenja tekuće naredbe praktički predstavlja signal za postavljanje u nulu (*Reset*) brojila koraka. Ovaj signal mora biti generiran na kraju koraka u kojem se javlja.

Signal uvjet, koji se dobiva kao rezultat usporedbe uvjeta grananja koristi se za eventualnu promjenu sadržaja programskog brojila u T4 koraku. Ovo se postiže pomoću I funkcije signala uvjet i T4 i operacijskog koda naredbe za grananje:

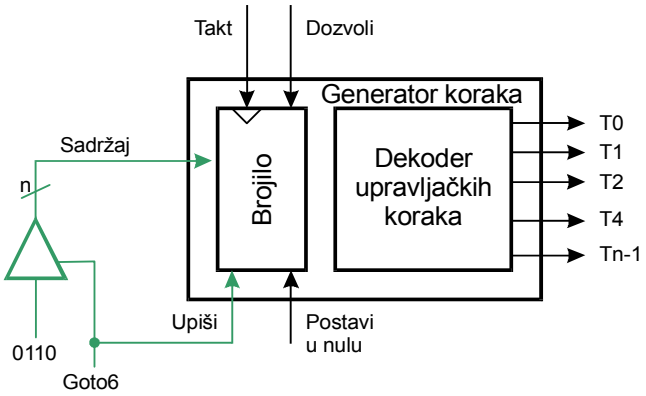
$$PC_{in} = ... + uvjet \cdot T4 \cdot br + ...$$

Posebno razmatranje vrijedi za naredbu za posmak. Kod ove naredbe u T4 koraku postavljen je uvjet, n = 0 $\rightarrow$ (Grc, R_{out}, napuni), pa npr. signal Grc ima u izrazu:

Grc = ... + (n = 0)·shr·T4 + ...

Signali R_{out} i upiši imaju sličan oblik. Vremenski raspored navedenih signala može biti problematičan budući da upiši signal upisuje sadržaj u brojilo koji istovremeno generira signal n = 0.

Generator koraka. Generator upravljačkih koraka prikazan je na sljedećoj slici.



Radi se o sinkronom brojilu s mogućnošću direktnog upisa početne vrijednosti kao i postavljanja početne vrijednosti u nulu. Brojilo povećava sadržaj prema osnovnom takt signalu, uz uvjet da je signal dozvoli postavljen. Postavljanjem signala postavi u nulu postavlja brojilo u sljedećem taktu na početni korak T0. Slično postavljanjem signala Upiši, koji je ekvivalentan signalu Goto6 naredbe za posmak, upisuje se u brojilo na sljedeći takt impuls broj 6, odnosno brojilo pokazuje na korak T6 naredbe. Naime naredba za posmak, sh, u koraku T6 koristi uvjet n = 0 za ponovno vraćanje na istu naredbu kako bi se realizirao višestruki posmak. Broj mjesta za koje je posmak potrebno izvršiti upisuje se u brojilo posmaka bilo iz c3 polja naredbe ili iz sadržaja spremnika na koji pokazuje naredba. Nakon svakog posmaka za jedno mjesto umanjuje se sadržaj istog brojila uspoređuje s nulom i ako rezultat nije jednak nuli ponavlja se posmak, odnosno korak T6. Ovo se realizira upisom u brojilo koraka vrijednosti 6 ili binarno 0110.

Logika vremenskog vođenja. Logika vremenskog vođenja prikazana je na sljedećoj slici. Kao što je slikom pokazano vanjski signali, npr. Strt, ili neki signali generirani unutar sklopovlja, npr. Run, odnosno signali generirani unutar upravljačke logike, npr. Čitaj, Piši, Stop, i interno generirani signali kao npr. Takt, koriste se za vremensko vođenje. Sklopovi izvode sljedeće četiri različite funkcije:

1. Generiranje Run signala temeljen Strt signala. Flip-flop označen ① koristi sa za sinkronizaciju promjena Run signala s vodećim taktom. Ulazna logika slijedi iz RTN opisa Run signala:

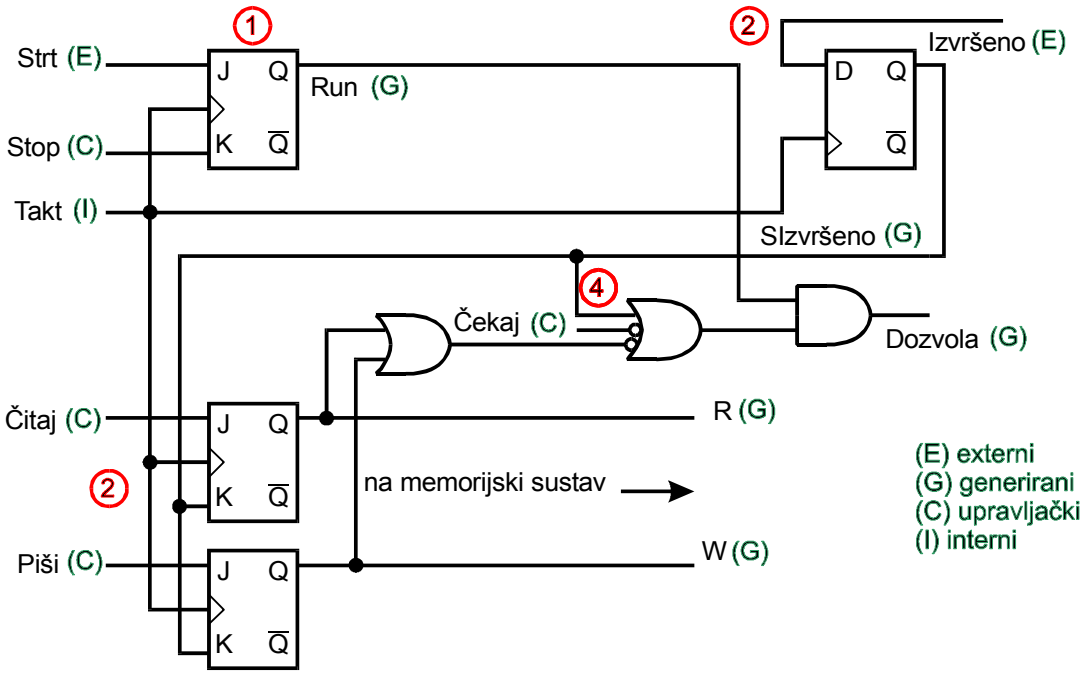
   $\neg$Run $\wedge$ Strt $\rightarrow$ Run $\leftarrow$ 1

   Stop signal postavlja JK flip-flop u nulu i generira se prema stop operacijskom kodu:

   stop (:= op = 31) $\rightarrow$ Run $\leftarrow$ 0

2. Signal SIzvršeno, ②, sinkronizira asinkroni Izvršeno signal koji generira memorijski sustav kako bi dojavio procesoru da je memorijska operacija uspješno okončana. Ukoliko je rad memorijskog sustava sinkroniziran s radom procesora onda ovaj sklop je nepotreban.
3. Generiranje R i W signala (*Read, Write*), ③, dolazi od upravljačkih signala Čitaj, Piši. Ovi signali su neophodni s obzirom da memorijski sustav zahtjeva da signali R i W budu aktivni za tijekom upisa ili čitanja, dok se signali Čitaj, Piši generiraju samo u takt cikusu u kojem se pojavljuju.
4. Generiranje signala Dozvola (*Enable*), ④, dozvoljava rad generatora upravljačkih koraka. Na slici ovaj signal se koristi da bi zaustavio brojilo upravljačkih koraka za vrijeme dok procesor čeka na odaziv sporijeg memorijskog sustava. Brojilo upravljačkih koraka mora biti aktivno kada je aktivan signal Run i kada je memorijska operacija izvršena ili kada signal Čekaj je neaktivan ili su neaktivni signali R i W:

$$\text{Dozvola} = \text{Run} \cdot \left(\text{SIzvršeno} + \overline{\text{Čekaj}} + \overline{(\text{R} + \text{W})} \right)$$



Inicjalizacija sustava. Kada se procesor prvi put priključi na napon napajanja, ili kada se narine signal za postavljanje u početno stanje (*Reset*) pojedini sklopovi moraju se postaviti u poznato početno stanje. Generator upravljačke sekvence mora biti inicjaliziran na prvi korak, a programsko brojilo mora pokazivati na adresu prve naredbe programa, npr 0000H ili FFFEH. Specifikacija ovog postupka izostavljena je u RTN opisu procesora.

Realizacija upravljačke jedinice. Upravljačku jedinicu moguće je realizirati zasebnim integriranim sklopovima, PAL ili PLA ili drugim programabilnim poljima logičkih vrata. Temeljni problem u realizaciji upravljačke jedinice je veliki broj ulaza, izlaza i međučlanova. Također neki od signala koji se dosta koriste zahtijevaju veliki ulazni faktor grananja što

dodatno otežava sklopovsku implementaciju. Tako npr. kod SRCa za očekivati je ulazne faktore grananja i do 32, a kod složenijih CISCova i preko 100.

Područje koje može rezultirati u značajnom poboljšanju je vremensko vođenje upravljačke sekvence. Očito je iz opisa naredbi da one zahtjeva različit broj koraka, a također iz vremenskog dijagrama pojedinog koraka može se zaključiti da svi koraci ne zahtjevaju isto vrijeme za uspješnu realizaciju. Tako će jednostavan prijenos između dva spremnika zahtijevati kraće vrijeme od ALU operacije. Jedan od načina da se iskoristi različito trajanje pojedinih koraka je da se generira više takt signala ili da se koristi samookidajuća logika. Ali ova razmatranja prelaze okvire ovog kolegija.

Dvo- i tro-sabirnička SRC arhitektura

Ukoliko se poveća broj sabirnica, odnosno broj međuveza, moguće je istovremeno prenositi više podataka. Tako je broj i konfiguracija sabirnica jedan od odlučujućih faktora u povećanju performansi procesora. S druge strane povećanje broja sabirnice, neovisno o razini implementacije, rezultira u povećanju cijene procesora. Ponovo potrebno je pronaći kompromis između dva oprečna zahtjeva, performanse – cijena.

U nastavku analizirati će se dvo - i tro – sabirnička arhitektura koje ujedno rezultiraju i različitim konkretnim RTN opisom.

Dvosabirnički SRC

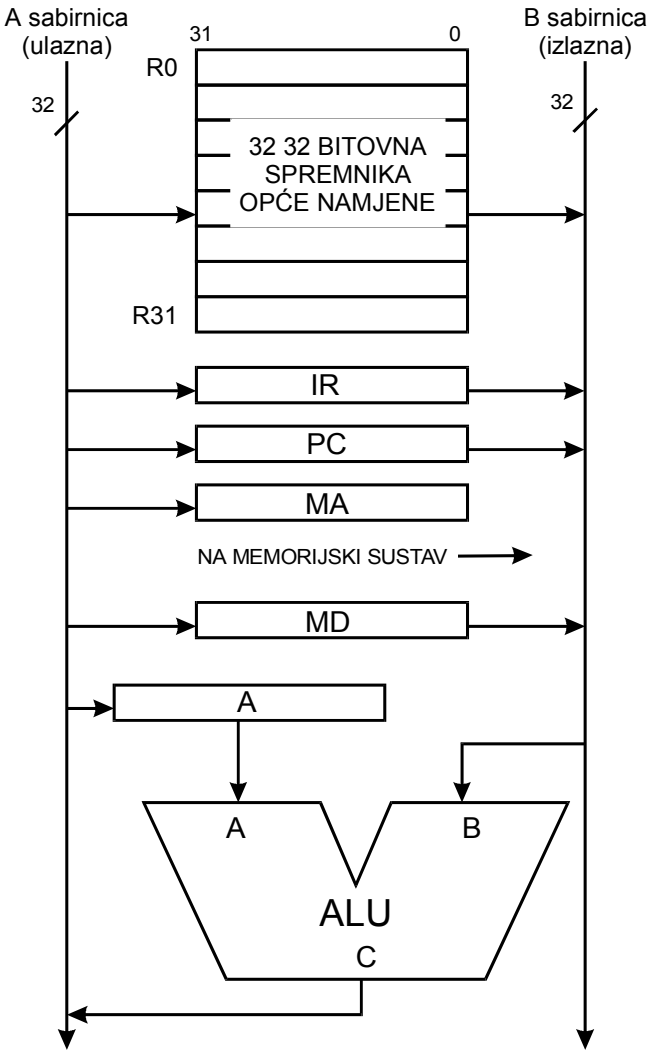
Sljedeća slika prikazuje dvosabirničku SRC mikroarhitekturu. Informacije, podaci, prenose se iz spremnika preko B sabirnice, nazvane izlazna sabirnica, a upisuju se u spremnike preko A sabirnice, nazvane ulazna sabirnica. Ova arhitektura na zahtijeva izlazni međuspremnik ALU s obzirom da se rezultat može direktno preko A sabirnice upisati u odredišni spremnik.

Prijenos podataka između spremnika u ovoj arhitekturi ostvaruje se postavljanjem podatka na izlaznu sabirnicu i upisom podataka s ulazne sabirnice. Vezu između ovih dviju sabirnica ostvaruje ALU. Jedna od funkcija ALU je i kopiranje sadržaja sabirnice B na sabirnicu A koja se skraćeno označava C = B (izlaz C je jednak sadržaju sabirnice B). U ovoj arhitekturi naredba za zbrajanje imala bi sljedeći oblik:

Korak	RTN	Upravljačka sekvenca
T0	MA ← PC:	PC_{out}, C = B, MA_{in}
T1	PC ← PC + 4: MD ← M[MA];	PC_{out}, INC4, PC_{in}, Čitaj, Čekaj
T2	IR ← MD;	MD_{out}, C = B, IR_{in}
T3	A ← R[rb];	Grb, R_{out}, C = B, A_{in}
T4	R[ra] ← A + R[rc];	Grc, R_{out}, ADD, Sra, R_{in}, Kraj

Kako nije potrebna pohrana rezultata zbrajanja u međuspremnik napravljena je ušteda u jednom koraku. Tako korak T4 zamjenjuje dva koraka u jednosabirničkoj arhitekturi T4 i T5. Ali za realizaciju navedenog potrebno je uz dodatnu sabirnicu i proširenje upravljačkog sklopovlja. Naime, u posljednjem koraku potrebno je istovremeno odabrati dva spremnika, rb i ra. Logika za odabir spremnika temeljen sadržaja spremnika naredbe kod jednosabirničke arhitekture može odabrati samo jedan spremnik u jednom koraku. Kod dvosabirničke arhitekture istu logiku je potrebno proširiti s posebnim odabirom spremnika

čiji se sadržaj postavlja na sabirnicu Gra, Grb i Grc, te logikom koja odabire spremnik u koji se upisuje, Sra, Srb i Src. Ovo znači da je potrebno imati dva nezavisna 5-32 dekodera.



Slično razmatranje može se provest za naredbu za upis sadržaja memorije u spremnik, ld. Ona je konkretno za ovu arhitekturu opisana na sljedeći način:

Korak	RTN	Upravljačka sekvenca
T0	MA ← PC:	PC_{out}, C = B, MA_{in}
T1	PC ← PC + 4: MD ← M[MA];	PC_{out}, INC4, PC_{in}, Čitaj, Čekaj
T2	IR ← MD;	MD_{out}, C = B, IR_{in}
T3	A ← ((rb = 0) → 0: (rb ≠ 0) → R[rb]);	Grb, BA_{out}, B = C, A_{in}
T4	MA ← A + c2 {proš. predz.};	$c2_{out}$, ADD, MA_{in}
T5	MD ← M[MA];	Čitaj, Čekaj
T6	R[ra] ← MD;	MD_{out}, C = B, Sra, R_{in}, Kraj

Poboljšanje performansi i složenost izvedbe. Moguće je napraviti procjenu poboljšanja performansi procesora prelaskom s jedno- na dvo-sabirničku arhitekturu. Na prvi pogled očito je da će se performanse poboljšati s obzirom na prosječno manji broj taktova po naredbi. Ipak to poboljšanje smanjeno s obzirom da zbog dva sustava sabirnica je povećano vrijeme propagacije između izvora i odredišta, odnosno smanjena frekvencija takta. Prvo će se analizirati ubrzanje zbog smanjenja prosječnog broja taktova (ciklusa) po naredbi:

$$\%\text{ubrzanje} = \frac{T_{1-\text{sabirnička}} - T_{2-\text{sabirnička}}}{T_{2-\text{sabirnička}}} \cdot 100,$$

gdje je

$$T = \text{vrijeme izvođenja} = IC \cdot CPI \cdot \tau .$$

Za pretpostaviti je da se broj naredbi, IC (*Instruction Count*), ne mijenja prelaskom na dvo-sabirnički sustav. U nedostatku preciznijih podataka napraviti će se pretpostavka da su prosječne naredbe slične naredbi ld te da imaju ubrzanje s osam na sedam ciklusa. Tada je:

$$\%\text{ubrzanje} = \frac{IC \cdot 8 \cdot \tau - IC \cdot 7 \cdot \tau}{IC \cdot 8 \cdot \tau} \cdot 100 = 14\% .$$

Ako se pretpostavi da je zbog uvođenja druge sabirnice bilo potrebno povećati trajanje takta za 10%, odnosno da je $\tau_2 = 1.1 \cdot \tau_1$, tada je:

$$\%\text{ubrzanje} = \frac{IC \cdot 8 \cdot \tau_1 - IC \cdot 7 \cdot 1.1 \cdot \tau_1}{IC \cdot 8 \cdot \tau} \cdot 100 = 3.9\% .$$

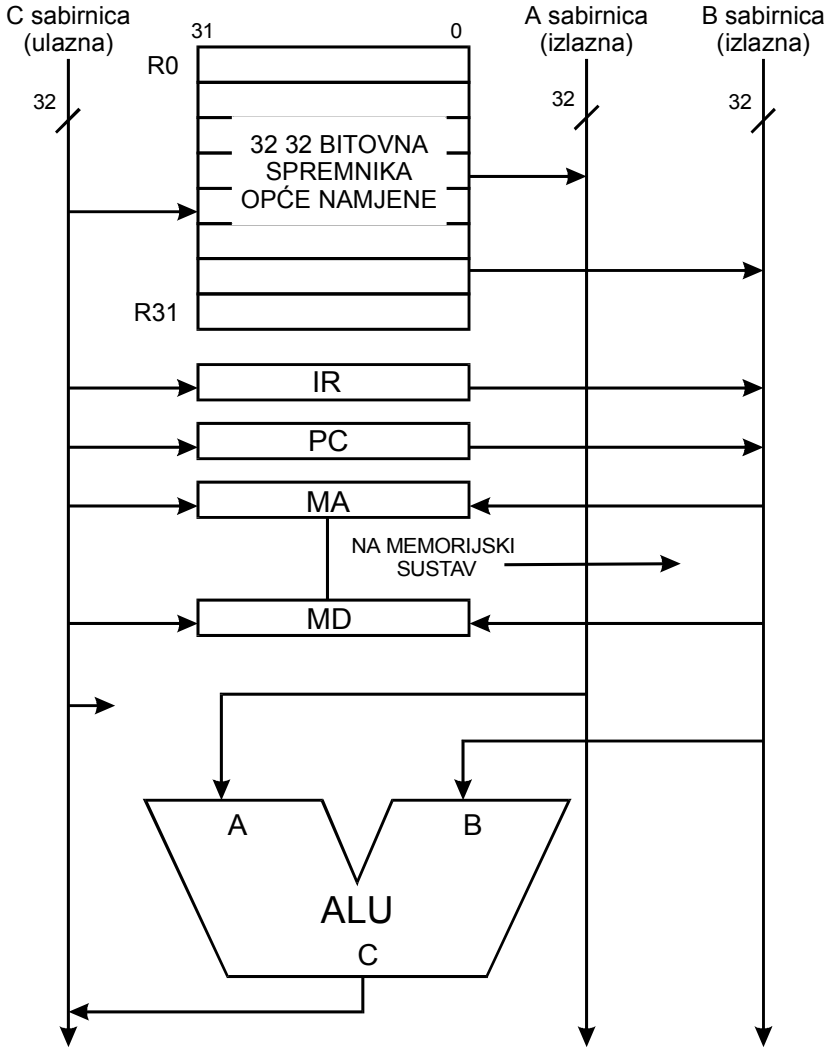
Provedene analize ukazuju kako stvarno poboljšanje performansi je znatno ispod očekivanih. Uzimajući u obzir dodatnu složenost sklopova proizlazi zaključak da se praktički ne isplati uvoditi drugu sabirnicu.

Trosabirnički SRC

Uvođenjem treće sabirnice u arhitekturu SRCa može se istovremeno na ALU dovesti oba operanda i rezultat upisati u odredišni spremnik. Sljedeća slika prikazuje trosabirnički SRC, a tablica prikazuje konkretan RTN opis naredbe za zbrajanje na trosabirničkoj arhitekturi.

Korak	RTN	Upravljačka sekvenca
T0	MA ← PC: MD ← M[MA]: PC ← PC + 4;	PC_{out}, MA_{in}, INC4, PC_{in}, Čitaj, Čekaj
T1	IR ← MD;	MD_{out}, C = B, IR_{in}
T2	R[ra] ← R[rb] + R[rc];	GArc, RA_{out}, GBrb, RB_{out}, ADD, Sra, R_{in}, Kraj

Ovom arhitekturom smanjen je ukupan broj koraka na svega tri te je eliminiran i privremeni spremnik A. U koraku T0 izvode se paralelno dvije aktivnosti: inicjalizacija očitanja naredbe i inkrementiranje programskog brojila. Aritmetička operacija se izvodi u jednom, T2, koraku. U ovoj arhitekturi bilo je potrebno uvesti dodatnu odabirnu logiku u odnosu na dvosabirnički SRC. Naime dva spremnika se istovremeno odabiru za postavljanje sadržaja na sabirnicu te jedan spremnik se odabire za upis rezultata, U realizaciji ovo znaći tri 5-32 dekodera te signale GAra, GArb, GArc, RA_{out}, GBra, GBrb, GBrc, RB_{out}, Sra, Srb, Src i R_{in}.



U ovoj arhitekturi potrebni su također određeni zahvati i na memorijskom međusklopu. Memorijsku adresu potrebno je postaviti već na početku prvog ciklusa T0, što nije izvedivo flip-flopovima upravljanim bridom. Ukoliko se navedeno realizira moguće je postaviti Čitaj/Piši signal na vanjsku sabirnicu istovremeno s MA_{in} signalom. Ako je memorija dovoljno brza podatak je moguće pročitati u jednom taktu. Adresa se može na više načina postaviti na adresnu sabirnicu na samom početku T0 ciklusa što uključuje i rješenja sa standardnim spremnicima ili posebne arhitekture koje preskaču MA spremnik i sadržaj spajaju na adresnu sabirnicu na samom početku T0 ciklusa.

Sljedeća tablica prikazuje konkretan RTN opis naredbe za upis sadržaja memorijske lokacije u spremnik, ld.

Korak	RTN	Upravljačka sekvenca
T0	MA ← PC: MD ← M[MA]: PC ← PC + 4;	PC_{out}, MA_{in}, INC4, PC_{in}, Čitaj, Čekaj
T1	IR ← MD;	MD_{out}, C = B, IR_{in}
T2	MA ← ((rb = 0) → 0: (rb ≠ 0) → R[rb]) + c2 {proš. predz.}: MD ← M[MA];	GArb, BA_{out}, $c2_{out}$, ADD, MAC_{in}, Čitaj, Čekaj

T3	R[ra] ← MD;	MD_{out}, C = B, Sra, R_{in}, Kraj

Kako se broj sabirnica povećava, tako se i povećava broj mogućnosti povezivanja pojedinih spremnika. Pronalaženje optimalne konfiguracije u tom slučaju zahtjeva detaljnu analizu i simulacije.

Povećanje performansi. I ovom slučaju moguće je procijeniti poboljšanje performansi prelaskom s jedno- na tro-sabirničku arhitekturu. Analiza naredbe ld pokazala je smanjenje broja koraka s osam na četiri. Uz istu grubu pretpostavku da se radi o prosječnom smanjenju broja koraka te da uvođenje novih sabirnica rezultira u povećanju perioda takta za 10% dobiva se ubrzanje od:

$$\%\text{ubrzanje} = \frac{\text{IC} \cdot 8 \cdot \tau_1 - \text{IC} \cdot 4 \cdot 1.1 \cdot \tau_1}{\text{IC} \cdot 8 \cdot \tau} \cdot 100 = 82\% .$$

Neka je nova pretpostavka da je potrebno tri ciklusa za pristup memoriji umjesto jedan kako je prvobitno pretpostavljeno, odnosno potrebna su dva ciklusa za čekanje. Nadalje, neka je 20% naredbi koje upisuju sadržaj iz memorije u spremnik, ld, te da kod upisa u memoriju nema čekanja. Kako svaka naredba mora izvest dohvat naredbe iz memorije, te još 20% naredbi čita iz memorije prosječan broj ciklusa po naredbi iznosi:

$$\text{CPI}_{1-\text{sabirnica}} = 8 + 2(\text{dohvat naredbe}) + 0.2 \cdot 2(\text{naredba ld}) = 10.4 ,$$

$$\text{CPI}_{3-\text{sabirnice}} = 4 + 2(\text{dohvat naredbe}) + 0.2 \cdot 2(\text{naredba ld}) = 6.4 .$$

Sada ubrzanje iznosi:

$$\%\text{ubrzanje} = \frac{\text{IC} \cdot 10.4 \cdot \tau_1 - \text{IC} \cdot 6.4 \cdot 1.1 \cdot \tau_1}{\text{IC} \cdot 8 \cdot \tau} \cdot 100 = 48\% ,$$

što opet predstavlja značajno povećanje performansi.

Osnovni koncept: Tehnika projektiranja procesora

Projektiranje procesora vrlo je zahtjevan poduhvat. U provedenim razmatranjima opisan je samo dio tog procesa. Bez obzira moguće je napraviti kratku rekapitulaciju procesa projektiranja.

- Projektiranje počinje apstraktnim RTN opisom procesora. Opis obuhvaća i spremnike vidljive projektantu.
- Sljedeći korak je projektiranje mikroarhitekture, posebice projektiranjem putova podataka. Ovaj korak obuhvaća sve spremnike koje zahtjeva arhitektura, a ne samo one vidljive programeru.
- Struktura sabirnica ima značajan utjecaj na performanse procesora. Povećani broj sabirnica rezultira smanjenjem prosječnog trajanja naredbe, ali i povećanjem trajanja takta.
- Nakon što su projektirani putovi podataka, projektant opisuje konkretno, RTNom, sve naredbe. Ovim korakom definiraju se aktivnosti u svakom koraku izvođenja naredbe.

- Operacije određene konkretnim RTNom zahtijevaju da se putovima podataka generiraju određeni upravljački signali koji rezultiraju u aktivnostima potrebnim da se izvedu koraci naredbe. Ovi signali generiraju se temeljen stanja u putovi podataka.
- Upravljačka jedinica, izvedena pomoću kombinacijske logike, implementira funkcije opisane konkretnim RTNom.
- U stvarnosti projektant prolazi više puta kroz pojedine korake procesa projektiranja, vraćajući se i korigirajući prethodne korake s ciljem povećanja performansi sustava i smanjenja njegove cijene. Utjecaj i učinkovitost pojedinih rješenja rezultat je pažljive analize i simulacija sa ciljem dobivanja što boljeg rješenja.

4.7. Postavljanje procesora u početno stanje (*Reset*)

Slijede razmatranja koja obuhvaćaju proces postavljanja procesora u početno stanje nakon prikapčanja na napon napajanja ili nakon što se postavi zaseban "Reset" signal. Sukladno provedenim razmatranjima očito je da se nakon prikapčanja napona napajanja minimalno generator upravljačkih koraka postaviti u nulu, kao i programsko brojilo da pokazuje na poznatu lokaciju na kojoj se mora nalaziti prva naredba programa.

4.7.1. Sklopovi koje je potrebno postaviti u početno stanje

Uz postavljanje generatora upravljačkih koraka i programskog brojila u početno stanje potrebno je prilikom inicjalizacije, odnosno postavljanja sustava u početno stanje poništiti sve zastavice prekida te zabraniti prekide za vrijeme trajanja inicjalizacije. Također, kod procesora sa spremnikom stanja ili statusa potrebno je ili poništiti zastavice stanja ili ih postaviti u neko poznato stanje. Nakon postavljanja u početno stanje neki procesori prolaze kroz proceduru ispitivanja sklopovlja. Kako prilikom detekcije signala za postavljanje u početno stanje obično se trenutno obustavljaju sve aktivnosti, tako ovaj signal ako se pojavi za vrijeme sklopovskog ispitivanja i promijeni stanja sklopovlja može se krivo interpretirati kao neka sklopovska pogreška. Zato je potrebno obratiti posebnu pažnju na realizaciju unutarnjih i vanjskih sklopova za postavljanje u početno stanje.

Neki procesori razlikuju dvije vrste postavljanja u početno stanje, tvrdo (*hard*) i mekano (*soft*). Mekano postavljanje u početno stanje podrazumijeva inicjalizaciju samo programskog brojila i generatora upravljačkih koraka, dok tvrdo podrazumijeva inicjalizaciju cjelovitog stanja procesora, sistemskih spremnika te vanjskih perifernih uređaja.

Inicjalizacija programskog brojila. U nadležnosti procesorske upravljačke jedinice je inicjalizacija programskog brojila. Programsko brojilo može se inicjalizirati s vrijednošću početne adrese procedure koja izvodi daljnji postupak inicjalizacije ostalih sklopova, tzv. direktni pristup, ili može se inicjalizirati s adresom gdje se nalazi adresa prve naredbe iste procedure, tzv. indirektni pristup. Npr. Intel 8086 nakon postavljanja u početno stanje inicjalizira programsko brojilo na vrijednost FFFF0H. Naredba zapisana na toj adresi obično je naredba za bezuvjetno grananje JMP, a adresa grananja je početna adresa ROMa u kojem je upisan inicjalizacijski program, *bootstrap*.

Proces postavljanja MC68000 u početno stanje. Procesor MC68000 u procesu postavljanja u početno stanje upisuje u sistemsko kazalo stoga, SSP, sadržaj s memorijske lokacije 0H, tzv. vektor posebnih stanja, a u programsko brojilo sadržaj memorijske lokacije 4H. Na tim memorijskim lokacijama mora biti upisana odgovarajuća vrijednost te inicjalizacijski program mora se postaviti na lokacijama na koje pokazuju te vrijednosti.

Slijedi ponovljeni opis procedure postavljanja u početno stanje procesora MC68000:

```
Reset:                                  Reset ulaz
exc_req:                                zastavica: zahtjev za posebnim stanjem
exc_lev⟨2..0⟩:                          razina posebnog stanja
vect⟨7..0⟩:                             broj vektora posebnog stanja
exc := exc_req ∧ (exc_lev > INT):       postoji zahtjev s prioritetom većim od trenutnog
tmp⟨15..0⟩:                             privremeni spremnik za spremnik stanja

Interpretacija_naredbe := (
    Run ∧ ¬(Reset ∨ exc) → (IR ← Mw[PC]: PC ← PC + 2);    Normalno izvođenje
        Izvođenje_naredbe);

    Reset → (INT⟨2..0⟩ ← 7: S ← 1: T ← 0                  Reset
        SSP ← Ml[0]: PC ← Ml[4]:
        Reset ← 0: Run ← 1);

    Run ∧ ¬Reset ∧ exc →
    (Tmp ←Status: S ← 1: T ← 0;
        (SSP ← SSP - 4; Ml[SSP] ← PC;                     PC na stog
        SSP ← SSP - 2; Mw[SSP] ← Tmp;                     spremnik stanja na stog
        INT⟨2..0⟩ ← exc_lev⟨2..0⟩:
        PC ← Ml[vect⟨7..0⟩#00₂]);
Interpretacija_naredbe:
```

Za primijetiti je da procesor kod postavljanja u početno stanje prvo zabrani sve prekide postavljanjem tekuće razine prioriteta na najveću, 7, poništavanja zastavicu praćenja, postavljenja bit nadzornog stanja, upisuju u sistemsko kazalo stoga sadržaj memorijske adrese 0H te u programsko brojilo sadržaj memorijske adrese 4. Po završetku ove procedure poništava se zastavica postavljanja u početno stanje i postavlja se Strt zastavica. Nakon završetka procesa postavljanja u početno stanje slijedi interpretacija naredbe koja počinje dohvatom naredbe.

Izvori signala postavljanja u početno stanje. Svaki procesor ima poseban ulaz za signal za postavljanje u početno stanje, *Reset Pin*. Neki procesori imaju i više takvih ulaza vezanih uz funkcije tvrdog i mekog postavljanja u početno stanje. Ovi vanjski signali općenito su asinkroni i moraju se sinkronizirati s radom procesora, odnosno sa sistemskim taktom. Najjednostavniji i mnogim korisnicima najpoznatiji izvor signala za postavljanje u početno stanje je posebno “Reset” tipkalo. Ovaj ulaz koristi i za druge namjene vezane uz sklopovsku ispravnost sustava kao i ispravno funkcioniranje programa. Tako npr. pad napona napajanja ispod dozvoljene razine trebao bi rezultirati prekidom rada procesora i njegovim postavljanjem u početno stanje. U nekim aplikacijama u ovom slučaju potrebno je kontrolirati prekid rada, odnosno upisati sve bitne informacije u memoriju s pomoćnim napajanjem, a tek onda dozvoliti postavljanje u početno stanje. Drugi slučaj je kad se program zatvori u beskonačnu petlju kao posljedica pogreške u programiranju. Jedno od rješenja je takozvani nadzorni sat (*watch-dog timer*) kojega mora program svako određeno vrijeme postaviti u početno stanje. Ukoliko se to ne napravi ovo brojilo postavlja procesor u

početno stanje. Mnogi proizvođači imaju specijalizirane sklopove za ovu primjenu. Primjer je nadzorni sklop MAX690 proizvođača Maxim.

4.7.2. Inicjalizacija i postavljanje u početno stanje SRCa

Slijedi dopuna SRCa mogućnošću postavljanja u početno stanje pojavom signala Strt, tvrdo postavljanje u početno stanje i meko postavljanje u početno stanje pojavom signala Rst nakon što je postavljena zastavica Run. Tvrdo postavljanje u početno stanje inicjalizira programsko brojilo i spremnike procesora, dok mekano postavljanje u početno stanje podrazumijeva samo inicjalizaciju programskog brojila. VAnjski signal za postavljanje u početno stanje je asinkron što je i realna pretpostavka.

Apstraktni RTN operacije postavljanja u početno stanje. Izvorni RTN opis SRCa mora se izmijeniti kako bi obuhvatio tvrdo i meko postavljanje u početno stanje. Stanje procesora mora se proširiti signalom:

Rst: Vanjski signal postavljanja u početno stanje

Interpretacija naredbe mora se izmijeniti na dva mjesta: na početnim aktivnostima i za vrijeme izvođenja naredbe. Izvorni RTN opis bio je:

interpretacija_naredbe: := ($\neg$Run $\wedge$ Strt $\rightarrow$ Run $\leftarrow$ 1:
Run $\rightarrow$ (IR $\leftarrow$ M[PC]: PC $\leftarrow$ PC + 4; izvođenje_naredbe)):
izvođenje_naredbe := (Id (:= op = 1 ...);

Na početku rada procesora, , potrebno je izvršiti inicjalizaciju procesora i spremnika opće namjene, a za vrijeme izvođenja, meko postavljanje u početno stanje, samo poništiti zastavicu Rst i inicjalizirati programsko brojilo:

interpretacija_naredbe: := ($\neg$Run $\wedge$ Strt $\rightarrow$ Run $\leftarrow$ 1: PC, R[0..31] $\leftarrow$ 0):
Run $\wedge$ $\neg$Rst $\rightarrow$ (IR $\leftarrow$ M[PC]: PC $\leftarrow$ PC + 4; izvođenje_naredbe)):
Run $\wedge$ Rst $\rightarrow$ (Rst $\leftarrow$ 0: PC $\leftarrow$ 0; interpretacija_naredbe)):

Temeljen ovog opisa naredba se u potpunosti izvede prije usporedbe uvjeta postavljanja u početno stanje, što praktički i nije u potpunosti praktično. Ipak apstraktan RTN opis ne omogućava prekid naredbe prije njenog okončanja. Ovo je moguće samo opisati konkretnim RTN opisom.

Konkretan RTN opis postavljanja u početno stanje SRCa. Konkretan opis jednosabirničkog SRCa proširiti će se mogućnošću postavljanja u početno stanje. Ispitivanje zastavice Rst izvodi se na početku svakog takt ciklusa. Tako konkretan RTN opis naredbe za zbrajanje s mekim postavljanjem u početno stanje izgleda:

Korak	RTN	Upravljačka sekvenca
T0	$\neg$Rst $\rightarrow$ (MA $\leftarrow$ PC: C $\leftarrow$ PC + 4): Rst $\rightarrow$ (Rst $\leftarrow$ 0: PC $\leftarrow$ 0: T $\leftarrow$ 0*);	(PC_{out}, MA_{in}, INC4, C_{in}): Rst $\rightarrow$ (ClrPC,Goto0);
T1	$\neg$Rst $\rightarrow$ (MD $\leftarrow$ M[MA]: PC $\leftarrow$ C): Rst $\rightarrow$ (Rst $\leftarrow$ 0: PC $\leftarrow$ 0: T $\leftarrow$ 0);	$\neg$Rst $\rightarrow$ (Čitaj, C_{out}, PC_{in}, Čekaj): Rst $\rightarrow$ (ClrPC,Goto0);
T2	(IR $\leftarrow$ MD): Rst $\rightarrow$ (Rst $\leftarrow$ 0: PC $\leftarrow$ 0: T $\leftarrow$ 0);	$\neg$Rst $\rightarrow$(MD_{out}, IR_{in}): Rst $\rightarrow$ (ClrPC,Goto0);
T3	(A $\leftarrow$ R[rb]): Rst $\rightarrow$ (Rst $\leftarrow$ 0: PC $\leftarrow$ 0: T $\leftarrow$ 0);	$\neg$Rst $\rightarrow$(Grb, R_{out}, A_{in}): Rst $\rightarrow$ (ClrPC,Goto0);

T4	(C ← A + R[rc]): Rst → (Rst ← 0: PC ← 0: T ← 0);	¬Rst →(Grc, R_{out}, ADD, C_{in}): Rst → (ClrPC,Goto0);
T5	¬Rst → (R[ra] ← C): Rst → (Rst ← 0: PC ← 0: T ← 0);	¬Rst →(C_{out}, Gra, R_{in}, Kraj): Rst → (ClrPC,Goto0);

*Postavlja se generator upravljačkih koraka u nulu

Signal ClrPC postavlja sadržaj programskog brojila u nulu. Jedan od načina realizacije je postavljanje nula na sabirnicu i signala upisa u programsko brojilo, PC_{in}. Signal Goto0 postavlja brojilo upravljačkih koraka u nulu, da pokazuje na korak T0, na sličan način kao što se realizira povrat na korak T6 pomoću Goto6. Tvrdo postavljanje u početno stanje realizira se na sličan način.

Stvarna realizacija sklopovlja kojom se ostvaruje postavljanje u početno stanje može se napraviti kao vježba. Za naglasiti je kako je praksa da uzlazni brid signala za postavljanje u početno stanje zaustavlja aktivnosti procesora, a silazni brid postavlja procesor u poznato početno stanje. Tako pritisak na taster “Reset” rezultira u trenutnom prekidu naredbe koja se izvodi i postavljanju procesora u početno stanje

4.8. Posebna stanja

Slijedi analiza obrade posebnih stanja s aspekta procesora. Razmotriti će se potreba uvođenja različitih tipova posebnih stanja, te pristup njihove obrade. Kod obrade ulazno/izlaznih operacije, jedno od sljedećih poglavlja, posebice će se analizirati prekidi s aspekta ulazno/izlaznog podsustava i obavljanja ulazno/izlaznih operacija. Kako je već spomenuto posebna stanja mogu se podijeliti na unutarnja i vanjska, sinkrona i asinkrona. Prva podjela polazi od lokacije izvora posebnog stanja, unutar procesora ili iz njegovog okruženja, dok druga podjela se temelji na vremenskoj pojavi posebnog stanja. Općenito, unutarnja posebna stanja su sinkrona dok vanjski izazvana posebna stanja iako mogu biti sinkrona obično su asinkrona. Razlog je što su posljednja uzrokovana nekim događajem izvan procesora te nemaju direktnu spregu s procesorskim taktom. Primjerice dijeljenje s nulom je unutarnje izazvano posebno stanje i ono je očito sinkrono procesorskom taktu dok je pritisak tipke na tipkovnici vanjski prekid asinkron procesorskom taktu.

4.8.1. Proces obrade posebnih stanja

Pojava posebnog stanja, kao što je dijeljenje s nulom ili pritisak tipke na tipkovnici, može rezultirati u promjeni normalnog toka izvođenja programa. Posebna stanja tako prekidaju normalan tok izvođenja programa, čime su i dobili dodatni naziv prekidi. Važno je napomenuti da ova terminologija nije standardizirana tako da se mogu kod različitih autora naći različita tumačenja ovih pojmova, ali tendencija je da se pod pojmom posebnih stanja naziva sve što mijenja normalan tok izvođenja programa, uključujući pogrešno funkcioniranje sklopova, dok pod nazivom prekidi podrazumijeva se posebna stanja generirana van procesora kao što su ulazno/izlazni zahtjevi. Obrada posebnih stanja i njihovo prepoznavanje također spada u jednu od temeljnih točaka procesa projektiranja procesora. U sljedećim analizama obraditi će se prvo općenito obrada posebnih stanja, a zaključiti će se proširenjem projektiranog SRCa ovom funkcijom.

U prethodnom poglavlju opisana je obrada posebnih stanja kod procesora MC68000 na apstraktnoj razini. Iako različiti procesori imaju nešto različit pristup obradi posebnih stanja imaju i dosta zajedničkoga te je moguće uopćiti trendove u njihovoj obradi.

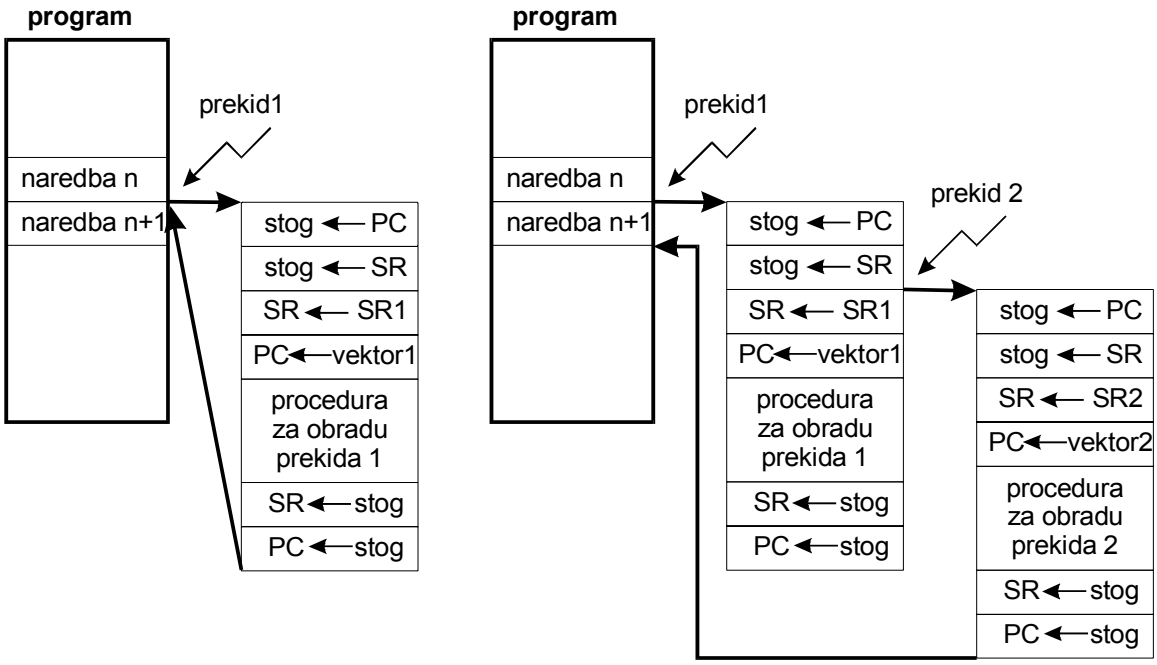
Procesor mora biti u stanju obraditi posebna stanja. Neki procesori kao što je MC68000 posvetili su nekoliko bitova da bi opisali razinu prioriteta tekućeg posebnog stanja. Smo posebna stanja većeg prioriteta od tekućeg mogu se prihvatiti i uzeti u obradu. Drugačiji pristup je zamjena razina prioriteta s jedinstvenom zastavicom dozvoli/zabrani. U prvom slučaju upravljanje razinama prioriteta posebnih stanja je u nadležnosti upravljačke jedinice za interno generirana posebna stanja jer u tom slučaju postoji i mogućnost fizičke veze između uzroka posebnog stanja i upravljačke jedinice. Za vanjski generirana posebna stanja, prekide, određivanje prioriteta i razrješavanje mogućih konflikata je u nadležnosti zasebnog sklopa za upravljanje prekidima (*Interrupt Controller*).

Procesor mora identificirati izvor posebnog stanja. Izvor posebnog stanja ili prekida mora se identificirati na način da predstavi vrstu posebnog stanja ili preko adrese procedure za obradu tog posebnog stanja. U praksi se susreću dva načina rješavanja navedenog problema: pomoću vektora posebnog stanja ili primjenom zasebnog spremnika koji sadrži informaciju o vrsti posebnog stanja. Vektor se definira kao početna adresa procedure za obradu prekida ali može imati i dodatnu riječ koja opisuje posebno stanje. Kod ovakvog pristupa rješavanju identifikacije posebnog stanja, slanje vektora posebnog stanja procesoru je u nadležnosti njegovog izvora koji treba ovaj podatak proslijediti nakon što procesor dojavi da je spremna prihvati posebno stanje na obradu. Prednost ovog pristupa je u tome što svaki izvor posebnog stanja ima svoj vektor, odnosno proceduru za njegovu obradu. Nedostatak je što se dosta memorijskog prostora troši na vektore posebnih stanja i kod procedura za njihovu obradu.

Alternativni pristup je primjena samo jedne procedure za obradu posebnog stanja. Pojavom posebnog stanja sadržaj programskog brojila pohrani se u alternativni spremnik i izvođenje se prebaci na opću proceduru za obradu posebnog stanja. Proces koji je izazvao posebno stanje mora proslijediti proceduri za obradu dovoljno informacija kako bi se zaključilo o kojoj vrsti posebnog stanja se radi. Ovaj pristup koristi se kod MIPS RISC procesora.

Stanje procesora se mora sačuvati. U trenutku prekida svog normalnog izvođenja procesor mora sačuvati stanje procesora koje će se obnoviti nakon obrade posebnog stanja. Time je omogućen nesmetan nastavak prekinutog izvođenja. Obično je potrebno sačuvati sadržaj programskog brojila kao i spremnike koji sadrže informaciju o prioritetu tekućeg izvođenja. Sadržaji ovih spremnika stavljaju se ili na sistemski stog ili u posebno za to namijenjene spremnike procesora. U ove spremnike se zatim upisuje adresa i prioritet procedure za obradu novog posebnog stanja.

Posebna stanja moraju se zabraniti tijekom kritičnih aktivnosti. Za vrijeme pohrane tekućeg stanja procesora i prelaska na proceduru za obradu novog posebnog stanja, tzv. promjena konteksta, novo-pridošli zahtjevi za obradom posebnih stanja moraju se ili odbaciti ili odgoditi. U suprotnom dolazi do gubitka informacija potrebnih za naknadan normalan nastavak prekinutih obrada.



Procedura za obradu posebnog stanja mora sačuvati i sadržaje spremnika koje koristi. Procedura za obradu posebnog stanja koristi neke spremnike opće namjene. Za nesmetani nastavak prekinutog programa potrebno je sačuvati i sadržaje tih spremnika koji će se obnoviti po završetku obrade posebnog stanja. Uobičajeno je da se sadržaji spremnika stave na stog.

Procedura obrade posebnih stanja može se rezimirati na sljedeći način:

1. Postavlja se zahtjev za prekidom, vanjski ili unutarnji.
2. Ako postoji više prioriteta prekida procesor uspoređuje prioritet pridošlog s tekućim prekidom sa ciljem donošenja odluke hoće li se prekid prihvatiti na obradu ili ne. Tako npr. prekid sistemskog sata može imati veći prioritet od tipkovnice. Ukoliko nema više razina prioriteta ispituje se da li je dozvoljeno ili ne postavljanje zahtjeva za prekidom ispitivanjem zastavice prekida.
3. Istovremeno procesor završava s izvođenjem tekuće naredbe ukoliko je to moguće. Naime npr. kod dijeljenja s nulom zaustavlja se tekuća naredba prije nego što je završilo njeno izvođenje.
4. Ukoliko je zahtjev za prekidom prihvaćen procesor sprema dio trenutnog stanja, minimalno sadržaj programskog brojila i trenutnu razinu prioriteta, na sistemski stog ili u posebne interne spremnike. Novi prekidi su zabranjeni tijekom ovih aktivnosti.
5. Izvor prekida (ALU, pisač, tipkovnica, itd.) šalje procesoru informaciju o vrsti prekida bilo u vidu odgovarajućeg koda, vektora prekida ili adrese procedure za obradu prekida. Kod vanjskih prekida obično se ova informacija prenosi preko podatkovnih linija nakon što se zahtjev za prekidom prihvati na obradu u odgovarajući spremnik ili u slučaju internog prekida direktno se informacija prosljeđuje u isti spremnik.

6. Ako je potrebno procesor pretvara kod u vektor prekida obično jednostavnom operacijom posmaka za određeni broj bita u lijevo.
7. Procesor upisuje vektor prekida u programsko brojilo i postavlja novu vrijednost prioriteta. Obrada se zatim nastavlja s adrese na koju pokazuje programsko brojilo, odnosno izvodi se procedura za obradu prekida. U početku ova procedura mora sačuvati sadržaje svih spremnika koje koristi na način da ih stavi na stog.
8. Izvodi se obrada posebnog stanja.
9. Po završetku obrade obnavlja se sadržaj spremnika koje je procedura koristila skidanjem sadržaja sa stoga te se izvodi naredba povrat iz prekida (*return from interrupt*), rti. Ova naredba obnavlja stanje procesora, odnosno skida sa stoga prethodni prioritet prekida i sadržaj programskog brojila. Nakon toga nastavlja se izvođenje prekinutog programa.

4.8.2. Vrste posebnih stanja

Suvremeni procesori imaju brojne vrste posebnih stanja spojene na njihov sustav za obradu posebnih stanja. Ovo je posljedica trenda proizvođača procesora i računala da što više “love” i otklanjaju nepravilnosti koje se javljaju tijekom rada sustava. U daljnjem tekstu opisati će se brojni različiti izvori posebnih stanja podijeljeni prema vrstama uobičajeno korišteni u današnjim procesorima.

Sistemsko postavljanje u početno stanje. Pojedini sustavi tretiraju postavljenje u početno stanje kao posebno stanje. Tako MC68000 ima ovaj pristup. SPARC je napravi jedan korak unaprijed uvodeći više različitih vrsta postavljanja u početno stanje.

Pogreška u provjeri memorije. Novije generacije procesora Intel Pentium i Motorola PPC601 imaju ugrađen mehanizam provjere i korekcije sadržaja upisanog u svaku pojedinu memorijsku lokaciju (*machine check exception*). Memorijski podsustav temeljen dodatnih bitova (pariteta ili korekcijskih kodova) ispituje ispravnost sadržaja memorijske lokacije i ukoliko se pojavi greška dojevljuje se procesoru preko zasebnog ulaza.

Pogrešan pristup kodu i podacima. Ova vrsta posebnog stanja dojavljuje se ukoliko se pristupa memorijskim lokacijama koje nisu fizički prisutne ili je tom programu zabranjen pristup tim lokacijama. Ova posebna stanja obično dojavljuje sustav za upravljanje memorijom (*memory management unit*). U prvom slučaju kod modernijih računala radi se o sustavu s virtualnom memorijom koji će biti naknadno opisan kod obrade memorijskog sustava. Drugi slučaj je u višekorisničkim operacijskim sustavima kada jedan korisnički proces neovlašteno nastoji koristiti memorijsko područje ili drugog procesa ili operacijskog sustava.

mnogi suvremeni sustavi razlikuju memorijska područja za program i podatke pa tako i posebna stanja neovlaštenog pristupa podacima i kodu.

Pogreška položaja podatka. Kod procesora koji koriste tvrdo smještanje podataka ukoliko dođe do nedozvoljenog pristupa podatku, npr. pristup riječi na neparnoj lokaciji, generira se ova posebna vrsta posebnog stanja. Ukoliko je memorija podijeljena na stranice, obično memorijske blokove veličine 1 do 4 kokteta, ova pogreška se dojavljuje ukoliko podatak prelazi iz jedne u drugu stranicu odnosno podatka je smješten na kraju jedne i početku druge stranice. Podjela memorije po stranicama obraditi će se u zasebnom poglavlju.

Programski uvjetovana posebna stanja. Postoji više vrsta posebnih stanja koja su posljedica izvođenja programa. Prvo od njih je izvođenje nepostojeće (*illegal instruction*) ili neimplementirane naredbe (*unimplemented instruction*). Ovu pogrešku detektira upravljačka jedinica prilikom dekodiranja naredbe. Sljedeća podvrsta ovih posebnih stanja je pokušaj izvođenja privilegiranih naredbi dok je izvođenje u korisničkom modu (*privilege instruction*). Tako npr. samo operacijski sustav smije mijenjati trenutnu razinu privilegija ili maskirati zahtjeve za prekidom. Ukoliko to pokuša izvesti korisnik procesoru se to dojavljuje kao posebna vrsta prekida. Kao posebna podvrsta ovih posebnih stanja su i pogreške koje su rezultat aritmetičkih operacija, kao npr. dijeljenje s nulom.

Različita sklopovska posebna stanja. Brojna su sklopovska posebna stanja. Neki procesori imaju jedno ili više brojila. Njihova vrijednost se dekrementira sinkrono sa sistemskim taktom i kad se dosegne nula postavi se procesoru zahtjev za prekidom. Uobičajena primjena ovakvih brojila je da nadziru vremensko izvođenje programa ili njegovih dijelova. Npr. pokušaj ispisa na pisač koji nije spojen može rezultirati beskonačnim čekanjem na ispis. Ali pomoću brojila se definira maksimalno vrijeme čekanja i ukoliko pisač ne odgovori u tom intervalu prekida se čekanje i ispisuje poruka korisniku. Ovakva brojila nazivaju se nadzorna brojila ili popularno “Watchdog timer”.

Praćenje i otkrivanje pogrešaka (*trace and debugging*). Mogućnost praćenje izvođenja programa i otkrivanje pogrešaka bilo u programu ili radu sklopovlja važna je funkcija računala. Jedan od načina njene realizacije je izvođenje programa naredbu po naredbu (*single stepping*). Ovo se realizira na način da se nakon svake naredbe procesoru generira posebno stanje. Postavljanje procesora ovakav mod rada obično realizira se postavljanjem jedne zastavice, bita praćenja (*trace flag*). Procedura za obradu ovog posebnog stanja obično korisniku prikazuje stanje procesora (sadržaj spremnika procesora) i dijela memorije, a po potrebi dozvoljava i izmjene sadržaja. Ovom funkcijom programer dobiva važne informacije o procesu izvođenja njegovog programa.

Nemaskirani prekid (*Nonmaskable Interrupt NMI*). Većina procesora ima posebnu vrst prekida koju nije moguće maskirati, odnosno zabraniti. Ova vrsta prekida ima najveći prioritet i koristi se za dojavu stanja koja ozbiljno ugrožavaju nastavak rada sustava, kao što je npr. nizak napon napajanja. U ovom slučaju procedura za obradu prekida mora u što kraćem vremenu, dok još uvijek postoji dovoljno rezervne energije, pohraniti važne podatke ili na disk ili u memoriju s baterijskim napajanjem.

Vanjski prekidi (*Interrupts*). Sustav za obradu vanjski izazvanih prekida omogućava programeru komunikaciju s ulazno/izlaznim uređajima, kao što je npr. slanje ili primanje znaka preko serijske komunikacije. Ovim se realizira jedan od načina izvođenja ulazno/izlaznih operacija.

4.8.3. Obrada posebnih stanja kod SRCa

Naredbe opisane apstraktnim i konkretnim RTNom nisu sadržavale opis obrade posebnih stanja. Sljedeći primjer sadrži sva svojstva obrade posebnih stanja uz neophodnu razinu jednostavnosti koja je potrebna razumijevanju problematike. Zbog jasnoće izostaviti će se postavljanje u početno stanje. U primjeru je primjetna mješavina dvaju pristupa obradi posebnih stanja: vektorski pristup i pomoću spremnika opisa posebnog stanja. Slijedi statički opis dijela procesora potreban za obradu posebnih stanja:

ireq:	Signal zahtjev za prekidom (*Interrupt request*)
iack:	Signal odobravanja prekida (*Interrupt acknowledge*)
IE:	1-bitovna zastavica dozvole prekida (*Enable flag*)
IPC⟨31..0⟩:	Lokacija pohrane PCa nakon pojave prekida
II⟨31..0⟩:	Informacija o prekidu (izvoru prekida)
Isrc_info⟨15..0⟩:	Informacija od izvora prekida
Isrc_vec⟨7..0⟩:	Vektor ili kod poslan od izvora prekida
Ivect⟨31..0⟩:=20@0#Isrc_vect⟨7..0⟩#4@0	Vektor prekida

Zahtjev za prekidom signalizira se vanjskim asinkronim signalom ireq. Postoji jedinstvena zastavica dozvole prekida, odnosno sve prekide moguće je samo ili dozvoliti ili zabraniti, a ne postoji mehanizam prioriteta prekida. Umjesto pohrane sadržaja programskog brojila na stog nakon prihvaćanja zahtjeva za prekidom, ono se upisuje u zaseban spremnik IPC. Po potrebi je moguće ovaj sadržaj programski prebaciti na stog. II (*Interrupt Information*) je spremnik vidljiv programeru u koji je upisana informacija o izvoru prekida. Isrc_info (*Interrupt Source Information*)je informacija koju šalje izvor prekida nakon što je prekid prihvaćen na obradu od strane procesora. Ova se informacija upisuje u II spremnik.

Izvor prekida dužan je poslati procesoru osam bitovni kod prekida koji se upisuje u spremnik Isrc_vek (*Interrupt Source Vector*). Posmakom sadržaja ovog spremnika za četiri mjesta u lijevo i postavljajući prvih dvadeset i posljednja četiri bita u nulu dobiva se vektor prekida koji se upisuje u spremnik Ivect. Ovdje je potrebno naglasiti da je 32 bitovni spremnik Ivect uveden u apstraktnom opisu procesora, ali u konkretnoj realizaciji možda nije potreban.

Izmijenjena sekvenca interpretacije naredbe izgleda:

interpretacija_naredbe :=

(¬Run ∧ Strt → Run ← 1:	Start
Run ∧ ¬(ireq ∧ IE) → (IR ← M[PC]: PC ← PC + 4; izvođenje naredbe):	Normalno izvođenje naredbe
Run ∧ (ireq ∧ IE) → (IPC ← PC⟨31..0⟩: II⟨15..0⟩ ← Isrc_info⟨15..0⟩: iack ← 1: IE ← 0: PC ← Ivect⟨31..0⟩; iack ← 0: interpretacija_naredbe);	Prekid

Ako postoji zahtjev za prekidom i prekidi su dozvoljeni tada:

- Programsko brojilo, PC, kopira se u spremnik IPC.
- Informacija o izvoru prekida, Isrc_info upisuje se u spremnik II.
- Bit odobravanja prekida, iack, se postavlja. Procesor ne mora odmah prihvatiti prekid na obradu. Zato izvor prekida mora držati aktivnu liniju za postavljanje zahtjeva za prekidom aktivnom sve dok procesor ne prihvati zahtjev za prekidom. Tada izvor prekida mora deaktivirati ovu liniju.
- Prekidi se zabranjuju postavljanjem maske ili dozvole prekida, IE.
- U programsko brojilo upisuje se vektor prekida, Ivect.

Nakon ovih aktivnosti započinje procedura za obradu prekida. Kako je već spomenuto u nadležnosti je ove procedure, programera, da sačuva sadržaje spremnika opće namjene

koje ona koristi stavljajući ih na stog. Također, ukoliko procedura dozvoljava da se postave novi zahtjevi za prekidom tijekom njenog izvođenja, mora se sačuvati i sadržaj IPCa i IIa prije nego se dozvoli prekidanje. Ukoliko se ovo ne napravi, novi prekid bi prepisao informaciju o tekućem prekidu i adresu povrata iz prekida te se ne bi moglo neometano nastaviti izvođenje prekinutog programa ili procedure. Naredbe **svi** i **ri** koriste se spremanje, svi, sadržaja IPCa i IIa u spremnike opće namjene kao i za obnovu njihovog sadržaja iz spremnika, ri:

svi (:= op = 16) →	(R[ra]⟨15..0⟩ ← II⟨15..0⟩: R[rb]⟨31..0⟩ ← IPC⟨31..0⟩);	Spremi II i IPC
ri (:= op = 17) →	(II⟨15..0⟩ ← R[ra]⟨15..0⟩: IPC⟨31..0⟩ ← R[rb]⟨31..0⟩);	Obnovi II i IPC

Programer mora imati također mogućnost dozvoliti ili zabraniti prihvaćanje zahtjeva za prekidom. Ovo se realizira dvjema naredbama, **een** i **edi**, dozvola posebnog stanja (*exception enable*) i zabrana posebnog stanja (*exception disable*):

een (:= op = 10) →	(IE ← 1);	Dozvoli prekidanje
edi (:= op = 11) →	(IE ← 0);	Zabrani prekidanje

Naredba, povrat iz prekida, rfi (*return from interrupt*), vraća izvođenje u prekinuti program ili proceduru kopiranjem sadržaja IPC u PC i postavljanjem dozvole prekidanja, IE:

rfi (:= op = 30) →	(PC ← IPC: IE ← 1);	Povrat iz procedure za obradu prekida

Važno je proces povrata realizirati kao jedinstvenu aktivnost koja se mora izvesti u cjelini. Kombinacija naredbe za grananje, br, i naredbe postavljanja zastavice prekida, een, nije prikladna zbog jedino mogućeg redoslijeda izvođenja ovih naredbi, a to je prvo een a zatim br. U tom slučaju ako već postoji zahtjev za prekidom koji čeka grananje u novu prekidnu proceduru bi se izvelo bez povrata u prekinuti program, a adresa povratka u prekinuti program bi se izgubila.

Sljedeća tablica prikazuje konkretnu realizaciju obrade prekida na jednosabirničkom SRCu.

Korak	RTN	Upravljačka sekvenca
T0	(¬ireq ∧ IE) → MA ← PC: C ← PC + 4;	(¬ireq ∧ IE) → (IPC ← PC: II ← Isrc_info: IE ← 0: PC ← 20@0#Isrc_vec⟨7..0⟩#0000: iack ← 1; iack ← 0: Kraj);
T1	MD ← M[MA]: PC ← C;	
T2	IR ← MD;	
T3	Izvođenje naredbe	

U skladu s apstraktnim opisom ispitivanje ulaza, zastavice, prekida obavlja se samo u prvom taktu naredbe, T0. Nakon ovog ispitivanja izvođenje se normalno odvija sve do početka dohvata nove naredbe. Također, oznaka iack ← 1; iack ← 0: u skladu s apstraktnim opisom naredbi ne podrazumijeva da se prihvat prekida postavlja u jednom taktu i poništava u sljedećem, već podrazumijeva da se u taktu T0 na ovom izlazu generira impuls koji izvor prekida mora prepoznati. U cilju realizacije specificiranog prijenosa podataka u samo jednom taktu potrebne su odgovarajuće preinake i putovima podataka, odnosno dodatne veze među spremnicima. Prvenstveno moraju se dodati spremnici II i

IPC. Kako informacije od izvora prekida direktno se upisuju u II spremnik, odnosno sadržaj PC direktno se upisuje u IPC moguće je posebnim vezama mimoići procesorsku sabirnicu.

U razmatranjima vezanim uz postavljanje procesora u početno stanje napomenuto je da prihvaćanje zahtjeva za prekidom mora biti zabranjeno neposredno nakon postavljanja u početno stanje. Prema tome postavljanje u početno stanje i posebna stanja usko su vezani. Slijedi izmjena opisa SRC koja obuhvaća postavljanje u početno stanje i posebna stanja:

interpretacija_naredbe :=

(¬Run ∧ Strt → Run ← 1: PC, R[0..31] ← 0; interpretacija_naredbe):	Tvrdi "Reset"
(Run ∧ Rst → Rst ← 0: IE ← 0: PC ← 0; interpretacija_naredbe):	Meki "Reset"
Run ∧ ¬Rst ∧ (ireq ∧ IE) → (IPC ← PC⟨31..0⟩: II⟨15..0⟩ ← Isrc_info⟨15..0⟩: iack ← 1: IE ← 0: PC ← Ivect⟨31..0⟩; iack ← 0: interpretacija_naredbe);	Prekid
Run ∧ ¬Rst ∧ ¬(ireq ∧ IE) → (IR ← M[PC]: PC ← PC + 4; izvođenje naredbe):	Normalno izvođenje

4.8.4. Problemi u realizaciji obrade posebnih stanja

Postoji više aspekta koji usložnjavaju realizaciju obrade posebnih stanja, a vezane su uz posebna stanja koja se pojavljuju tijekom izvođenja naredbe koja je potrebno ih odmah obraditi. Kod projektiranog SRCa ulaz za postavljanje prekida ispituje se samo u prvom koraku naredbe, T0, a svi prekidi koji se pojave tijekom izvođenja naredbe čekaju na obradu dok se naredba ne izvede. U stvarnosti ovo je neprihvatljivo iz dva razloga. Prvo, u nekim slučajevima prikladno je obraditi posebno stanje prije nego se pogrešan rezultat operacije upiše u odredište i promjeni zastavice stanja. Drugo, kod procesora sa cjevovodom praktički je uvijek jedna ili više naredbi u fazi obrade, odnosno ne postoji trenutak kad su sve naredbe izvedene.

Posebna stanja koja zaustavljaju izvođenje naredbe. Postoji nekoliko razloga zbog kojih je dobro da se može prekinuti izvođenje naredbe u bilo kojem ciklusu. Jedan od razloga je što CISC procesori imaju naredbe koje mogu trajati jako dugo. Npr. Intel 80x86 ima naredbu za obradu nizova dužine do 2^{32} riječi. Slične naredbe imaju i drugi CISC procesori. Kod računala s virtualnom memorijom moguće je da naredba koja koristi više operanada postoji vjerojatnost da se jedan operand nalazi u memorijskoj stranici koja trenutno nije u glavnoj memoriji. Ovaj tip posebnog stanja poznat je pod nazivom promašaj stranice (*page fault*). Naredba se ne može izvesti dok se ne obradi ovo posebno stanje. Složene aritmetičke operacije mogu također generirati posebno stanje tijekom izvođenja jer naredbu nije moguće ispravno izvršiti.

U svim navedenim slučajevima naredba se mora zaustaviti usred izvođenja i izvođenje se prebaciti na proceduru za obradu posebnog stanja. Procedura za obradu posebnog stanja mora obnoviti stanje koje je prethodilo naredbi koja se nije uspješno izvela. Jedno od rješenja je povratak naredbe na početak i obnova stanja procesora i memorije iz tog

trenutka. Drugi pristup je da se prati promjena trenutnih stanja procesora u svakom taktu, mikrostanja, te da se zapamti stanje prije pojave posebnog stanja, a povratak se ostvaruje obnovom prekinutog mikrostanja. Pamćenje mikrostanja zahtjeva pohranu svih spremnika procesora, kao što su privremeni spremnici A i C, vrijednost brojila upravljačkih koraka, brojilo posmaka, i sl. Naravno, u ovom slučaju značajno se usložnjava sklopovska realizacija procesora.

Posebna stanja kod procesora sa cjevovodom. U slučaju procesora sa cjevovodom obrada posebnih stanja posebno se usložnjava. Razlog je što se istovremeno obrađuje više naredbi od kojih je svaka u različitom stadiju izvođenja. Njihovo prekidanje zahtijeva složeno obnavljanje stanja procesora i obično projektanti postavljaju posebne procedure za obradu posebnih stanja s kojima programeri moraju se detaljno upoznati.

Zaključci

- U ovom poglavlju obrađeno je projektiranje procesora na razini logičkih vrata, kao i postavljanje u poznato početno stanje i obrada posebnih stanja. Proces projektiranja temeljen je na apstraktnom opisu procesora.
- Proces projektiranja započinje projektiranjem putova podataka. Polazeći od pretpostavljene, grubo predložene, arhitekture putova podataka postavi se konkretan opis skupa naredbi procesora. Pomoću ovog opisa projektant nadopunjava putove podataka, detaljima vezanim uz spremnike, upravljačke signale, sabirnice i memorijske međusklopove. Nakon što su sklopovski detaljno određeni putovi podataka, projektant definira upravljačku sekvencu za sve korake naredbi. Ovaj korak obično predstavlja kritičnu točku u procesu projektiranja koja rezultira korekcijom prethodno odabranih rješenja.
- Nakon što je definira upravljačku sekvencu, projektant može nastaviti projektiranjem upravljačke jedinice koja mora generirati upravljačke signale u zadanim vremenskim trenutcima. Naravno i u ovom slučaju ponekad je potrebno korigirati rješenja iz prethodnih koraka. U ovom poglavlju opisana je samo realizacija upravljačke jedinice s fiksnom funkcijom (*hardwired*), dok realizacija programabilnim sklopom (*microcoded*) nije razmatrana.
- CPU može imati jedan ili više sustava sabirnica. S koliko sabirnica i na koji način će one povezivati spremnike i ALU ovisi o odluci projektanta, a ona je temeljena na povećanju performansi uz prihvatljivu cijenu i složenost realizacije. Više sabirnica obično rezultira poboljšanjem performansi procesora, uz povećanje cijene i složenosti izvedbe kao i produživanjem osnovnog takta.
- Grube procjene performansi pomažu projektantu u procjeni povećanja performansi vezanih uz pojedina rješenja. Precizniji podaci dobivaju se simulacijama ili preciznim proračunima.
- Procesor mora imati mogućnost postavljanja u poznato početno stanje. Neki procesori ima više mogućnosti postavljanja u poznato početno stanje, a vezana su uz koje spremnike se inicjalizira. Tako se razlikuje tvrdo i meko postavljenje u poznato početno stanje. Tvrdo postavljanje inicjalizira veći broj spremnika procesora.
- Obrada posebnih stanja je prisutna kod svakog procesora. Posebna stanja omogućavaju da neplanirani događaj ili događaj kojem nije vremenski određen prekine

izvođenje tekućeg programa. Takvi događaji se mogu javljati unutar ili izvan procesora. Posebna stanja su svrstana po tipovima i nadograđen je opis procesora kako bi se navedena posebna stanja mogla obraditi.