6.PROCESOR I SABIRNICE

6.1.Pojam i struktura procesora

Procesor ili centralna procesorska jedinica (CPU) je jedinica koja obavlja najveći dio obrade unutar računara. On je složeni sekvencijalni sklop koji komunicira sa memorijom i ulazno-izlaznim uređajima. Procesor dohvata instrukcije iz memorije, jednu ili više u pojedinom trenutku i izvršava ih. Izvršenje instrukcija se odražava na stanje memorije, samog procesora i ulazno-izlaznih uređaja.

Četiri najvažnije komponente svakog procesora su;

Upravljačka jedinica izdvaja instrukcije iz memorije i dekodira ih te na bazi instrukcija uz pomoć sekvencijalnih sklopova šalje upravljačke signale drugim dijelovima procesora. Mašinske instrukcije uglavnom rade vrlo primitivne stvari, poput čitanja i upisa podataka iz memorije, vršenja najprostijih operacija sa podacima, poput sabiranja i oduzimanja, itd. Upravljačka jedinica se može realizovati na dva načina: kao ožičena i mikroprogramirana. Ožičena upravljačka jedinica implementira se pomoću kombinacione strukture, koja na bazi instrukcije generiše upravljačke signale. Ožičene upravljačke jedinice su obično brže od mikroprogramiranih. Njihov dizajn koristi fiksnu arhitekturu - on zahtijeva promjene u ožičenju ako je skup instrukcija modificiran ili

promijenjen. Ova arhitektura je poželjna u kompjuterima sa smanjenim skupom instrukcija (RISC) jer oni koriste jednostavniji skup instrukcija. Upravljački sklop koji koristi ovaj pristup može raditi na velikoj brzini; međutim, ima malo fleksibilnosti, a složenost skupa instrukcija koje može implementirati je ograničena.

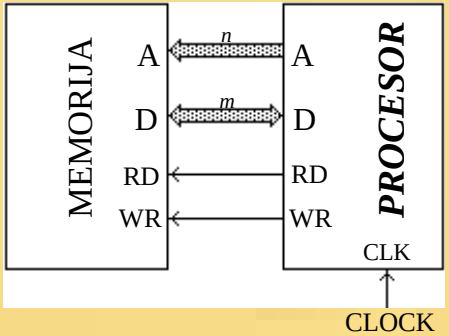
Upravljačka jedinica za mikroprogramom uvedena kao srednji nivo za izvršavanje instrukcija za računarske programe. Mikroprogrami su organizovani kao niz mikroinstrukcija i čuvani u posebnoj kontrolnoj memoriji. Algoritam za kontrolnu jedinicu mikroprograma, za razliku od ožičene kontrolne jedinice, obično je određen opisom dijagrama toka. Glavna prednost upravljačke jedinice mikroprograma je jednostavnost njegove strukture. Izlazi upravljačke jedinice su organizovani u mikrostrukturi i mogu se lako zamijeniti.

Aritmetička logička jedinica (ALU): upravlja aritmetičkim i logičkim operacijama. Za razliku od upravljačke jedinice, ona je često kombinaciona struktura. Unutar ALU obavljaju se cjelobrojne operacije poput sabiranja, oduzimanja, logičkog AND, OR, XOR i cjelobrojnog množenja. Mnogi procesori imaju i operacije cjelobrojnog dijeljenja, pa i operacije sa realnim brojevima, ali njihova realizacija zahtijeva veći angažman upravljačke jedinice.

Registri služe za privremeno prihvatanje mašinske instrukcije koja treba da se izvrši, za prihvatanje raznih podataka koji se obrađuju, kao i za vođenje evidencije o tome sa koje se adrese treba dobaviti sljedeća instrukcija. Registri su najbrža memorija, jer čuvaju podatke unutar centralnog procesora. Broj i namjena registara variraju među različitim procesorima. Jedan od njih se zove programski brojač (PC) i sadrži memorijsku lokaciju gdje se nalazi instrukcija koja se treba izvršiti. Statusni registar sadrži podatke u rezultatima aritmetičkih operacija (npr. da li je pri sabiranju brojeva došlo do prijenosa). Registri opšte namjene čuvaju trenutne podatke koji se obrađuju. Na jednostavnijim procesorima postoji akumulator koji je registar specijalizovan za aritmetičke operacije, ali na iole složenijim procesorima registri opšte namjene uglavnom mogu ravnopravno obavljati sve aritmetičke operacije. Neki procesori imaju i registre za pamćenje brojeva u pokretnom zarezu.

Veza sa sabirnicama je veza između procesora, memorije i periferijskih . Da bi ispravno funkcionirao, CPU se oslanja na sistemski sat, memoriju, sekundarnu pohranu, te podatkovne i adresne sabirnice. Sabirnica je skup linija koje koriste, procesor, memorija i ulazno-izlazni uređaji. Procesor postavlja zahtjeve prema memoriji šaljući odgovarajuće signale prema sabirnici: adresne signale koji određuju memorijsku lokaciju , upravljačke

signale koji govore da li je riječ o čitanju ili pisanju podatka i signale podataka koji predstavljaju sadržaje memorijskih lokacija koje se žele mijenjati. Slično, preko signala podataka se prima sadržaj izabrane memorijske lokacije. Stoga se memorija i procesor neposredno povezuju kao na sljedećoj slici.



Slika 115 *Spoj memorije i procesora*

Uloge memorije i procesora se bitno razlikuju. Naime, uloga memorije je *pasivna*. Jedine operacije koje se mogu obavljati nad memorijom su *čitanje iz memorije* i *upis u memoriju*. Kao što je ranije objašnjeno, čitanje odnosno upis vrše se pomoću *linija podataka*, dok se izbor memorijske lokacije čiji se sadržaj čita odnosno u koju se sadržaj upisuje vrši pomoću *adresnih linija*. Nalog za čitanje odnosno upis vrši se pomoću dva upravljačka ulaza koji su nazvani RD odnosno WR. S druge strane, uloga procesora je *aktivna*. Procesor je, na izvjestan način, gospodar nad čitavim sistemom koji određuje kad će se i odakle podaci čitati, a kad će se i gdje upisivati. Čitavo vrijeme

svog rada procesor neposredno komunicira sa memorijom, čita iz nje podatke, obrađuje ih i vraća nazad u memoriju. Da bi se ostvarila ova komunikacija, procesor također posjeduje adresne priključke i priključke za podatke, kao i naloge za čitanje i pisanje RD i WR. Međutim, za razliku od memorija, adresni priključci kao i nalozi za čitanje i pisanje kod procesora su *izlazi*, a ne *ulazi*. Preko ovih izlaza, procesor memoriji *izdaje naloge* za čitanje odnosno upis podataka na određene adrese. Priključci za podatke su kod procesora *dvosmjerni*, tj. mogu biti i ulazi i izlazi, slično kao u posljednjoj izvedbi RAM memorije koja je opisana u prethodnom poglavlju.

Na procesorima namijenjenim za izvršavanje više procesa, između upravljačke jedinice i veze sa sabirnicama dodaje se i peta jedinica: **jedinica za upravljanje memorijom**, MMU. Ova jedinica obavlja transformaciju memorijskih lokacija koje zahtijeva upravljačka jedinica procesora i memorijskih lokacija koje se stvarno prosljeđuju memorijskim čipovima kako bi svaki proces dobio svoje privatno memorijsko područje i omogućilo izvršavanje procesa koji ne mogu cijeli da stanu u fizičku memoriju.

6.2.Ciklus izvršenja instrukcije

Veoma je važno shvatiti da procesor obradu podataka koje dobavlja iz memorije *ne vrši po nekom fiksnom unaprijed zadanom pravilu*. Naime, u tom slučaju bi procesor mogao obavljati samo jednu unaprijed zadanu vrstu obrade podataka i nikakvu drugu. Cjelokupna filozofija rada procesora se zapravo zasniva na tome da procesor instrukcije o tome šta treba raditi sa podacima *također preuzima iz memorije*. Drugim riječima, u memoriji nisu pohranjeni samo podaci koji se obrađuju, već i upute *šta treba raditi sa tim podacima*. Te upute se nazivaju ***program***. U skladu sa Von Neumannovom arhitekturom računara koja je još uvijek najrasprostranjenija arhitektura po kojoj se grade računarski sistemi, podaci i programi su pohranjeni u *istoj memoriji* (arhitekture kod kojih to vrijedi nazivaju se Princeton arhitekture) mada postoje i arhitekture računara (npr. Harvard arhitekture) kod koje se za smještanje podataka odnosno programa koriste *posebne memorije*. U nastavku će biti pretpostavljeno da se koristi Von Neumannova arhitektura.

Poznato je da memorija ne može pamtiti ništa drugo nego *nizove nula i jedinica*. Stoga je jasno da i programi ne mogu u memoriji biti zapisani ni na kakav drugi način osim kao nizovi nula i jedinica. Drugim riječima, svi programi

su u memoriji zapisani kao nizovi nula i jedinica, pri čemu *različite kombinacije nula i jedinica imaju različita značenja*. U tom smislu, svaka kombinacija nula i jedinica koja za procesor ima neko tačno određeno značenje naziva se ***mašinska instrukcija*** (ili ***instrukcijski kôd***). Skupina mašinskih instrukcija formira ***mašinski program***. Ovako zapisani programi, koji su samo sekvence nula i jedinica, *jedino su što računar zaista razumije*. Cijelo vrijeme rada računar izvršava samo mašinske instrukcije i nikakve druge.

Sad će se razmotriti malo detaljnije šta se tačno odvija između procesora i memorije prilikom rada računara. Kako su instrukcije pohranjene u memoriji, procesor mora stalno dobavljati instrukcije iz memorije da vidi šta treba raditi. Stoga, prilikom rada računara, procesor čita mašinsku instrukciju sa neke adrese iz memorije, izvršava akciju predviđenu njenim značenjem, zatim čita narednu instrukciju (najčešće sa sljedeće adrese) i tako unedogled. Drugim riječima, procesor prvo dobavi instrukciju iz memorije, zatim je izvrši, zatim dobavi sljedeću, zatim nju izvrši, itd. Stoga se svaka instrukcija izvršava u tri faze: *faza* ***dobavljanja instrukcije,*** faza **dekodiranja** i *faza* ***izvršavanja*** . Većina procesora započinje dobavljanje instrukcija sa adrese 0, a dalji tok dobavljanja zavisi od samog programa.

U fazi dobavljanja instrukcije, procesor obično radi sljedeće:

1. Na adresnu sabirnicu se pošalje vrijednost programskog brojača.
2. Pošalje se kontrolni signal za čitanje iz memorije.
3. Memorija pošalje sadržaj tražene memorijske lokacije.
4. Sadržaj lokacije se upiše u poseban instrukcijski registar.

U fazi dekodiranja se prema sadržaju instrukcijskog registra određuje šta uraditi sa navedenom instrukcijom. Mašinske instrukcije uglavnom rade vrlo primitivne stvari, poput čitanja i upisa podataka iz memorije, vršenja najprostijih operacija sa podacima, poput sabiranja i oduzimanja, itd, ali često imaju više načina za pristup memorijskim lokacijama. U ovoj fazi se obično obavi sljedeće:

1. Prema sadržaju instrukcijskog registra provjeri da li su svi podaci potrebni za izvršenje instrukcije u procesoru
2. Ako nisu, obavlja se niz dodatnih operacija čitanja iz memorije da se svi podaci dovedu u procesor

U fazi izvršenja instrukcije se pripremljeni podaci proslijede na izvršavanje. U većini instrukcija izvršenje obavlja aritmetičko logička jedinica i dobiveni rezultati se upišu u memoriju. Tipične aktivnosti u ovoj fazi su:

1. Pošalji ulazne podatke aritmetičko-logičkoj jedinici
2. Sačekaj rezultat i upiši ga u odredišni registar ili memorijsku lokaciju koja je saznata tokom dekodiranja
3. Za vrlo složene instrukcije smještaju se međurezultati u interne registre i ponavlja faza izvršenja.

Izvršenje instrukcija se diriguje sistemskim satom koji osciluje određenom frekvencijom, stalno se ponavljajući Zavisno od organizacije procesora i složenosti instrukcija vrijeme izvršenja instrukcije može da varira od 0 ciklusa sata (instrukcije skokova u sistemima kod kojih se faze izvršenja instrukcija izvršavaju istovremeno za različite instrukcije) do oko 1100 ciklusa (stepenovanje u pokretnom zarezu na procesoru Intel 8087).

6.3.Instrukcijski skup

Koje tačno mašinske instrukcije procesor zna da izvršava, zavisi od konkretnog procesora, ali tipično se radi o *izuzetno primitivnim* instrukcijama koje se mogu podijeliti u pet osnovnih skupine: ***instrukcije za prenos podataka***, ***aritmetičko-logičke instrukcije*** , ***instrukcije za upravljanje tokom programa i sistemske instrukcije***. *Instrukcije za prenos podataka* nalažu procesoru da premjesti neki podatak sa jednog mjesta na drugo, na primjer da pročita sadržaj neke memorijske lokacije u neki od svojih registara, da upiše neku vrijednost ili sadržaj nekog registra na određenu memorijsku lokaciju, ili da prepiše sadržaj jednog od svojih internih registra u neki drugi registar. *Aritmetičko-logičke instrukcije* nalažu procesoru da izvrši neku od elementarnih računskih operacija nad sadržajima registara ili memorijskih lokacija. Ove operacije mogu biti ***aritmetičke*** (poput sabiranja i oduzimanja) ili ***logičke*** (poput poređenja, konjunkcije, disjunkcije i negacije). *Instrukcije za upravljanje tokom programa* su veoma značajne, jer omogućavaju da se izvođenje programa nastavi od proizvoljne memorijske lokacije. Mogu se podijeliti na ***bezuvjetne***, kod kojih se izvođenje programa bezuvjetno nastavlja od zadane memorijske lokacije, kao i ***uvjetne***, kod kojih se prelazak na

zadanu memorijsku lokaciju izvršava samo pod određenim uvjetom (na primjer, ukoliko je rezultat posljednje aritmetičko-logičke instrukcije bio jednak nuli). Upravo uvjetne instrukcije za upravljanje tokom programa omogućavaju računaru da *donosi odluke*, tj. da planira buduće ponašanje u ovisnosti od rezultata prethodnih operacija. Kod nekih procesora se susreće i četvrta grupa mašinskih instrukcija, koje se nazivaju ***ulazno-izlazne instrukcije***. One su slične instrukcijama za prenos podataka, samo što ostvaruju komunikaciju između registara procesora i ulaznih odnosno izlaznih jedinica umjesto memorije. S druge strane, mnogi procesori ne posjeduju posebne ulazno-izlazne instrukcije, nego se za komunikaciju sa ulaznim i izlaznim jedinicama koriste *iste instrukcije* kao za komunikaciju sa memorijom. Napredniji procesori imaju i **sistemske instrukcije**, koje prebacuju procesor u specijalni režim rada, na primjer režim koji uključuje ili isključuje predmemoriju (keš) za brže čitanje iz memorije, režim zaštite pristupa memoriji, režim mapiranja memorije, simuliranje drugih procesora, reakcije na nepostojeće instrukcije itd.

Procesori mogu uključiti kompleksne instrukcije u svoj skup instrukcija. Jedna "kompleksna" instrukcija radi nešto što može uzeti mnogo instrukcija na drugim računarima. Takve instrukcije su tipizirane instrukcijama koje uzimaju više koraka, kontrolišu višestruke funkcionalne jedinice ili se na drugi način

pojavljuju u većem opsegu od većine jednostavnih instrukcija koje implementira dati procesor. Neki primjeri "složenih" instrukcija uključuju:

- prebacivanje više registara u memoriju ili iz memorije (posebno stog) odjednom
- pomicanje velikih blokova memorije (npr. kopiranje niza ili DMA prijenos)
- komplikovane aritmetičke cjelobrojne i operacije nad realnim brojevima (npr. kvadratni korijen , ili transcendentne funkcije kao što su logaritam , sinus , kosinus , itd.)
- SIMD instrukcije ,pojedinačne instrukcije koje izvršavaju operaciju na više homogenih vrijednosti paralelno
- obavljanje atomske test-i-set instrukcije
- instrukcije koje izvršavaju ALU operacije s operandom iz memorije, a ne registra

Ovakve instrukcije znatno olakšavaju programiranje, ali komplikuju dizajn procesora. Tipično, vrijeme izvršavanja instrukcije kao i broj bajtova koje ona zauzima zavisi od toga koja je instrukcija u pitanju, što otežava istovremeno izvršavanje više instrukcija i onemogućuje realizaciju upravljačke jedinice na

ožičeni način. Procesori koji omogućavaju ovakve instrukcije se zovu CISC procesori (Complex instruction Set Computer). Za razliku od toga, procesori RISC tipa (Reduced Instruction Set Computer) obično nemaju takve instrukcije, pa su jednostavnije građe i postižu veću brzinu uz manju potrošnju energije zbog manjeg broja tranzistora.

Na tradicionalnim arhitekturama, instrukcija uključuje operacioni kod koji specificira operaciju koju treba izvršiti, kao što je dodavanje sadržaja memorije registru i nula ili više specifikatora operanda , koji mogu specificirati registre , memorijske lokacije ili doslovne podatke. Specifikatori operanda mogu imati načine adresiranja koji određuju njihovo značenje ili mogu biti u fiksnim poljima.

U principu, najosnovije operacije obrade podatka unutar jedne instrukcije predstavljaju operacije nad binarnim podacima. Dosta operacija ima dva operanda (npr. sabiranje, množenje, XOR itd.). Nazovimo ih SRC1 i SRC2. Rezultat operacije potrebno je smjestiti u određenu memorijsku lokaciju u memoriji ili registru koja predstavlja adresu odredišnog operanda. Odredišni operand označavamo sa oznakom DEST. Zavisno od toga kako se želi iskoristiti predviđeni broj bita za instrukciju, operandi se mogu kodirati na četiri različita načina.

Kod formata instrukcija **sa tri operanda**, instrukcija je podijeljena na četiri dijela prikazanoj na slici

OP	DEST	SRC1	SRC2

Slika 116: Format instrukcije sa 3 operanda

Prvi dio instrukcije označen sa OP predstavlja kod instrukcije. Na osnovu koda instrukcije, procesoru se zadaje tip obrade izvršavanja tekuće instrukcije. Drugi dio instrukcije označen sa DST predstavlja niz bitova koji navode memorijsku lokaciju ili oznaku registra na kojoj će se smjestiti rezultat operacije instrukcije. Ako tip instrukcije koja za obradu nema rezultat operacije, onda se ovaj dio instrukcije zanemaruje tokom izvršavanja. Treći dio instrukcije označen sa SRC1 predstavlja memorijsku adresu, vrijednost ili oznaku registra prvog izvorišnog operanda. Ako instrukcija ne koristi nikakve podatke za obradu, onda se ovaj dio zanemaruje tokom izvršavanja. Četvrti dio instrukcije označen sa SRC2 predstavlja memorijsku adresu, vrijednost ili oznaku registra drugog izvorišnog operanda. Ako instrukcija ne koristi nikakve podatke za obradu, ili koristi samo jedan operand, onda se ovaj dio instrukcije zanemaruje.

Kod formata instrukcije sa **dva operanda**, drugi dio (DST/SRC1) predstavlja u isto vrijeme lokaciju prvog izvorišnog operanda i lokaciju

odredišnog operanda. Ako imamo instrukciju operacije koja zahtijeva dva parametra, onda se tokom izvršavanja prvo sa lokacije koju pokazuje drugi dio instrukcije (DST/SRC1) uzima prvi operand, obavi operacija između njega i drugog operanda pa se rezultat operacije skladišti na istu lokaciju.

OP	DEST/SRC1	SRC2

Slika 117 : Format instrukcije sa 2 operanda

Kod formata instrukcije sa **jednim operandom** unutar same instrukcije nemamo drugi izvorišni operand SRC2. U ovom slučaju, drugi izvorišni operand predstavlja sadržaj posebnog registra, akumulatora., koji služi za aritmetičke operacije

OP	SRC

Slika 118: Format instrukcije sa jednim operandom

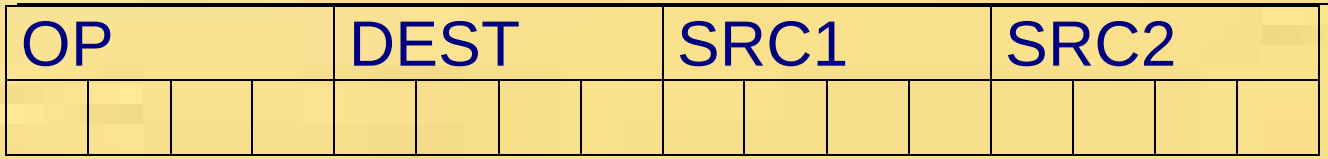
Kod načina adresiranja **sa nula operanda**, u instrukciji stoji samo kod o tipu obrade . Kad imamo instrukciju koja zahtjeva oba izvorišna operanda, onda prve dvije popunjene lokacije dijela memorije, koja se zove stek, gledano od pozicije na koju pokazuje specijalni registar koji se zove SP, predstavljaju adrese izvorišnih operanda. Da bi se podaci obrađivali na osnovu koda operacije, potrebno je da se nalaze na vrhu steka. Zbog toga

moramo imati instrukcije koje stavljaju podatak na stek ili ga skidaju sa njega. To su instrukcije PUSH i POP koje imaju format sa jednim operandom, dok ostale instrukcije sadrže samo kod

OP

6.4.Programski model nastavnog procesora SVEU16

Da bi se projektirao konkretan procesor, odnosno da bi se utvrdila građa njegove upravljačke jedinice i odredilo kakva mu je aritmetičko-logička jedinica potrebna, neophodno je prvo utvrditi kakve će uopće instrukcije raspoznavati ovaj procesor i koliku će količinu memorije moći da opsluži. Što se tiče količine memorije koju će procesor moći da opsluži, ovdje će biti pretpostavljeno da će procesor komunicirati sa memorijom koja može imati kapacitet od najviše 128 KB, i koja je organizirana kao skup od 65536 (2^{16}) 16-bitnih lokacija (adresa). Iz ovoga slijedi da razmatrani procesor mora posjedovati 16 adresnih priključaka i 16 priključaka za podatke. Svi registri trebaju imati kapacitet od 16 bita. Ukupno će biti 16 registara, od kojih je jedan programski brojač, a procesor će imati 16 instrukcija. Zbog ovih osobina, procesor će se zvati SVEU16.Registri R0-R14 su registri opšte namjene, dok registar R15 predstavlja programski brojač. Neka procesor ima arhitekturu sa tri operanda. Stoga se 16 bitna instrukcija može podijeliti na 4 polja po 4 bita.



Slika 119: Format instrukcije procesora SVEU16

Polje OP predstavlja operacioni kod instrukcije, dok polja DEST, SRC1 i SRC2 sadrže kodove registara operanda (osim u slučaju ako je OP=1001, kada SRC1 i SRC2 zajedno sadrže osmobitnu konstantu). Instrukcije su date u sljedećoj tabeli.

OP	OPERACIJA	MNEMONIK	KOMENTAR
0000	DEST=MEM[SRC2]	LOD R1,R2,(R3)	Čitanje iz memorije, u odredišni registar se upisuje vrijednost sa memorijske lokacije koju pokazuje registar SRC2.
0001	DEST=SRC1+SRC2	ADD R1,R2,R3	Sabiranje
0010	DEST=SRC1-SRC2	SUB R1,R2,R3	Oduzimanje
0011	DEST=SRC1 & SRC2	AND R1,R2,R3	Logičko AND
0100	DEST=SRC1 | SRC2	ORA R1,R2,R3	Logičko OR
0101	DEST=SRC1 ^ SRC2	XOR R1,R2,R3	Logičko XOR
0110	DEST=SRC1 >>> $SRC2_{0-3}$ DEST=SRC1 >> $SRC2_{0-3}$ DEST=SRC1 << $SRC2_{0-3}$ DEST=SRC1 rot $SRC2_{0-3}$	SHR R1,R2,R3	Pomjera ili rotira SRC1 ulijevo ili udesno onoliko puta koliko je navedeno u bitima 0-3 registra SRC2, dok biti 4- 5 određuju vrstu pomjeranja.

0111	DEST=SRC1 *SRC2	MUL R1,R2,R3	Množenje
1000	MEM[SRC2]=SRC1; DEST=SRC1	STO R1,R2,(R3)	U odredišni registar i na memorijsku lokaciju na koju pokazuje registar SRC2 upisuje se sadržaj registra SRC1.
1001	DEST=CONST	LDC R1,22	U odredišni registar se upisuje osmobitna konstanta predznačn o proširena iz bitova unutar SRC1 i SRC2 polja a ne iz pripadnih registara
1010	DEST=SRC1 >SRC2	GTU R1,R2,R3	Vrijednost 1 se upiše u DEST ako je SRC1>SRC2, kao nepredznačeni broj, inače 0
1011	DEST=+-SRC1 >+-SRC2	GTS R1,R2,R3	Vrijednost 1 se upiše u DEST ako je SRC1>SRC2, kao predznačeni broj, inače 0
1100	DEST=SRC1 <SRC2	LTU R1,R2,R3	Vrijednost 1 se upiše u DEST ako je SRC1<SRC2, kao nepredznačeni broj, inače 0
1101	DEST=+-SRC1 <+-SRC2	LTS R1,R2,R3	Vrijednost 1 se upiše u DEST ako je SRC1<SRC2, kao predznačeni broj, inače 0
1110	DEST=SRC1 == SRC2	EQU R1,R2,R3	Vrijednost 1 se upiše u DEST ako je SRC1 jednak SRC2, , inače 0
1111	DEST=SRC1 ; PC=SRC2	MAJ R1,R2,R3	U registar DEST se upiše sadržaj registra SRC1 a u programski brojač registra SRC2.

Instrukcijski skup je odabran kako bi se upravljačka jedinica realizovala na što jednostavniji način. Pojedini biti instrukcije će se direktno voditi na

multipleksere koji usmjeravaju podatke prema ALU i registrima. Tako se sve instrukcije se izvršavaju u jednom ciklusu. Pored ovoga, heksadekadni zapis ovako formiranih instrukcija omogućava njihovo lako prepoznavanje. Npr. instrukcija sa heksadekadnim kodom 513A simbolički se piše kao XOR R1,R3,R10 tj, obavi ekskluzivnu disjunkciju između bit po bit sadržaja registara R3 i R10 i rezultat smjesti u R1. Na primjer, ako je binarna vrijednost registara prije ove instrukcije R3=0000 1011 0000 1111 R10=1100 0000 0000 0001, nakon izvršenja instrukcije R1 će imati vrijednost R1=1100 1011 0000 1110.

Instrukcija sa kodom 1001 (heksadekadno 9) namijenjena je za unos osmobitne konstante u odredišni registar. Na primjer instrukcija sa heksadekadnim kodom 923A smješta u registar R2 predznačno proširenu konstantu 3A.

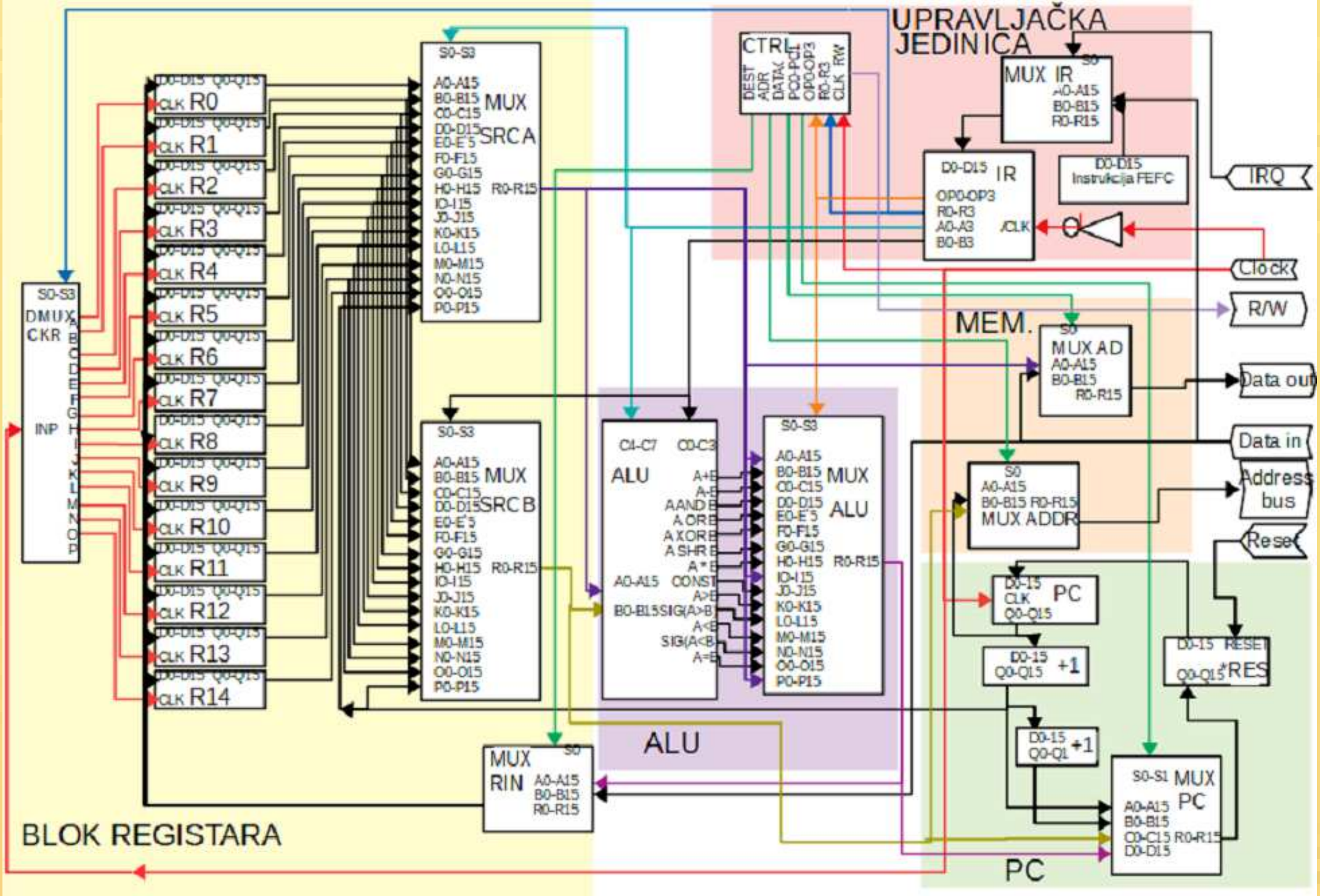
Kako je programski brojač PC registar sa kodom 1111 (drugo ime za registar R15) korištenje ovog registra u kao odredištu omogućava bezuslovih i uslovnih skokova. Neobična osobina ovog procesora je što nema statusnog registra, nego koristi instrukcije poređenja. Nakon instrukcija sa kodovima između binarno 1010 i 1110, rezultat poređenja može biti 0 ili 1. Npr. instrukcija s

heksadekadnim kodom E423 će u registar R4 upisati vrijednost 1 ako su registri R2 i R3 međusobno jednaki.

Memorijske instrukcije imaju kodove 0000 i 1000. Prva od njih u odredišni registar čita vrijednost sa memorijske lokacije na koju pokazuje registar SRC2, dok polje SRC1 nema značenja. Druga upisuje sadržaj registra navedenog u polju SRC1 i na memorijsku lokaciju na koju pokazuje SRC2 i u registar naveden u polju DEST. Kada je programski brojač izvor on pokazuje na instrukciju iza instrukcije koja se trenutno izvršava. To omogućava čitanje šesnaestobitnih konstanti koje su zakodirane u narednoj riječi iza instrukcije. Na primjer, ako se u jednoj memorijskoj riječi nalazi instrukcija sa heksadekadnim kodom 033F, a u narednoj riječi je kod 125A, posljedica je da će u odredišni registar 3 biti smještena vrijednost sa lokacije na koju pokazuje PC, a to je 125A.Instrukcija sa kodom 1111 korisna je za realizaciju potprograma, skokova i prebacivanja podataka između registara. Ona upiše registar SRC1 u DEST, a registar SRC2 u PC. Ako je SRC1=PC (kod 1111), tada će se obaviti skok na lokaciju na koju pokazuje SRC2, uz pamćenje sadržaja programskog brojača u SRC1. To će omogućiti povratak iz potprograma, skokom na lokaciju u zapamćenom registru.Instrukcija sa kodom 0110 omogućava četiri vrste pomjeranja bitova. Argument SRC2 pokazuje na

registar koji određuje koliko puta se pomjera SRC1 i na koji način. Najniža četiri bita registra navedenog u SRC2 određuju broj pomjeranja a kombinacije bita na pozicijama 4 i 5 određuju na koji način , kombinacija 00 predstavlja pomjeranje udesno sa ubacivanjem bita predznaka sa lijeve strane, kombinacija 01 predstavlja pomjeranje udesno sa ubacivanjem 0 bita sa lijeve strane, kombinacija 10 predstavlja rotiranje bita ulijevo sa prebacivanjem bita na drugu stranu, a kombinacija 11 predstavlja pomjeranje bita ulijevo sa ubacivanjem nula sa strane.

6.5.Princip rada procesora SVEU16


UPRAVLJAČKA JEDINICA
CTRL
MUX IR
D0-D15 IR
D0-D15 Instrukcija FEFC
IRQ
Clock
R/W
MEM.
MUX AD
Data out
Data in
Address bus
Reset
MUX ADDR
PC
+1
RESET
MUX PC
ALU
MUX SRC A
MUX SRC B
MUX ALU
MUX RIN
DMUX CKR
INP
R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
BLOK REGISTARA

Uočava se šest većih grupa logičkih blokova koji realizuju rad ovog procesora. To su blok registara, aritmetičko logička jedinica, upravljačka jedinica, veza ka memoriji, sklop za ažuriranje programskog brojača i priključci.

Ciklus izvršenja instrukcije sastoji se od dvije faze. U fazi kada je signal sata na visokom nivou, pripremi se dohvaćanje instrukcije u instrukcijski registar. Kada je signal sata na niskom nivou, obavlja se dekodiranje, izvršenje i priprema upisa.

Upravljačka jedinica sastoji se od instrukcijskog registra IR generatora kontrolnih signala CTRL i multipleksera prije ulaska u instrukcijski registar. Ukoliko je ovaj multiplekser u normalnom stanju, instrukcijski registar se puni na silaznu ivicu signala sata i dobije vrijednost sa memorijske sabirnice ulaznih podataka. Te podatke šalje memorija sa lokacije na koju je u tom trenutku pokazivao programski brojač. Alternativno, ako je signal IRQ bio na vrijednosti 1, tada se umjesto vrijednosti sa memorijske lokacije podmeće instrukcija koja zapamti programski brojač u registru R12 i skače na lokaciju na koju pokazuje registar R13. Ovo je predviđeno za realizaciju prekida (interrupt).

Iz instrukcijskog registra izlazi 16 bita, podijeljenih u grupe 4x4: operacioni kod instrukcije, odredišni registar, izvorni registar 1 i izvorni registar 2. Signal

oznake operacionog koda se prosljeđuje na aritmetičko logičku jedinicu, dok se oznake izvornih registara poslјeđuju multiplekserima u bloku registara i sporedno na aritmetičko logičku jedinicu. Signal oznake odredišnog registra se proslijedi u demultiplekser u bloku registara. Generator kontrolnih signala upravlja ostalim multiplekserima koristeći relativno jednostavne logičke funkcije koje računaju signale iz operacionog koda instrukcije.

U **bloku registara** se nalazi 15 registara realizovanih kao blok od po 8 sinhronih flip flopova koji se pune na uzlazu ivicu. Samo jedan od njih može u tom trenutku primiti vrijednost sa multipleksera MUX RIN, a koji će to biti određuju biti operacionog koda instrukcijskog registra. Demultiplekser DMUX CKR sa 16 jednobitnih izlaza prosljeđuje signal sata ka izabranom od 15 registara, dok se programski brojač puni na drugi način. Šesnaest linija iz izlaza MUX RIN povezano je na ulaze podataka svakog od 15 registara istovremeno. Iako svi registri imaju izlazne vrijednosti, samo sva izabrana izvorna registra prosljeđuju svoje vrijednosti aritmetičko logičkoj jedinici, izlazu na sabirnice i programskom brojaču. Koji su to registri određuje instrukcijski registar. Njihove vrijednosti prosljeđujuju multiplekseri MUX SRCA i MUX SRCB. Svaki od njih ima 16x16 ulaza, tj. po 16 linija od svakog od 15 registara plus linije programskog brojača uvećane za 1. Svaki ima i 16 izlaznih linija koje

se prosljeđuju na ALU,. Kako ulaz u blok registara može biti ALU ili sabirnica podataka, multiplekser MUX RIN određuje da li je ulaz u registre iz ALU ili DataIn memorijskog ulaza. To se određuje signalom DEST upravljačke jedinice CTRL koji u suštini znači da se trenutno izvršava instrukcija sa kodom 1000.

Dva izlaza iz registara ulaze u aritmetičko-logičku jedinicu ALU. Ona istovremeno obavlja većinu aritmetičko-logičkih operacija, a koja će biti proslijeđena dalje određuje MUX ALU sa 16x16 ulaza koji se biraju prema OP polju.

Aritmetičko logička jedinica obavlja 13 operacija nad dva šesnaestbitna broja koji dolaze iz SRCA i SRCB i 3 propuštanja ulaza iz SRCA. Koji rezultat će se proslijediti dalje određuje MUX ALU čiji kontrolni kodovi su identični polju OP instrukcijskog registra.

Najviše funkcija unutar ALU obavlja jedan šesnaestbitni sabirač koji je sastavljen od četiri četverobitna.. Četverobitni sabirači se mogu realitovati pomoću četiri jednobitna puna sabirača, ali u cilju ubrzanja rada koristi se složeniji sabirač sa propagacijom prijenosa koji zahtijeva više logičkih kola. Sabirač ima dva šesnaestbitna ulaza i jedan bit ulazni prijenos, a izlaz ima šesnaest bita i izlazni prijenos. Ovim sabiračem se realizuju operacije sabiranja (na prvi šesnaestbitni ulaz se dovede prvi sabirak, a na drugi

šesnaestbitni ulaz se dovede drugi sabirak, dok se na ulazni prijenos dovede 0), oduzimanja (na prvi šesnaestbitni ulaz se dovede prvi sabirak, na drugi šesnaestbitni ulaz se dovede invertovani drugi sabirak, dok se na ulazni prijenos dovede 1) i svih šest operacija poređenja (ulazi se povežu kao kod oduzimanja). Izlaz iz sabirača se koristi kao rezultat sabiranja i oduzimanja. Za operacije poređenja izlaz se formira tako da se svi biti izlaza osim najnižeg postave na nula. Najniži bit se kreira zavisno od operacije. Za operaciju testiranja jednakosti on se dobija operacijom NOR između svih bita rezultata oduzimanja. Tako, ako su SRC1 i SRC2 jednaki, rezultat oduzimanja je 0, i NOR između svih bitova rezultata je 1. Za operaciju testiranja da li je SRC1<SRC2, nepredznačeno najniži bit izlaza se dobije kao negacija izlaznog prijenosa iz rezultata oduzimanja. Za operaciju testiranja da li je SRC1<SRC2, predznačeno najniži bit izlaza se dobije iz najvišeg bita izlaza iz oduzimanja, jer on indicira da je rezultat oduzimanja negativan broj. Za operaciju testiranja da li je SRC1>SRC2, nepredznačeno najniži bit izlaza se dobije kao NOR operacija između negacije izlaznog prijenosa iz rezultata oduzimanja i rezultata testiranja jednakosti. Za operaciju testiranja da li je SRC1<SRC2, predznačeno najniži bit izlaza se dobije NOR operacijom između najvišeg bita izlaza iz oduzimanja i rezultata testiranja jednakosti.

Operacije AND, OR i XOR se realizuju jednostavnim odgovarajućim blokovima sa po dva šesnaestbitna ulaza i izlaza.

Operacije pomijeranja i rotiranja se realizuju za svaku od njih četiri posebno uz pomoć 16 multipleksera, za svaki bit rezultata po jedan. Svaki multiplekser ima 16 jednobitnih ulaza i jedan jednobitni izlaz, kako je prikazano u poglavlju barrel shifter. Četiri kontrolna bita na ulazima u multipleksere (koji se dovode iz četiri najniža bita SRC2) omogućuju pravilno miješanje bitova ulaza SRC1 na datim pozicijama. Rezultati četiri vrste pomijeranja se dalje prosljeđuju u novi multiplekser koji ima četiri šesnaestbitna ulaza i jedan šesnaestbitni izlaz. Izbor iz ovog multipleksera obavlja se sa dva bita registra SRC2, na pozicijama 4 i 5.

Najsloženiji dio aritmetičko logičke jedinice je množač. On je realizovan uz pomoć 15 šesnaestobitnih sabirača koji obavljaju operaciju

(((A+B)+(C+D))+((E+F)+(G+H)))+(((I+J)+(K+L))+((M+N)+(O+P))). Osam sabirača na prvom nivou sabiraju dva po dva sabirka. Rezultati ovih sabirača se prosljeđuju, dva po dva rezultata na četiri sabirača na drugom nivou. Rezultati sabirača na drugom nivou se prosljeđuju na dva sabirača na trećem nivou. Izlazi iz sabirača na trećem nivou se proslijede na ulaz sabirača na četvrtom nivou. Sabirci za sabirač na prvom nivou su ili nule ili vrijednost

SRC1 pomjerena ulijevo n puta zavisno od toga da li je bit na odgovarajućoj poziciji ulaza SRC2 jednak 0 ili 1.

Operacija konstante na izlaz prosljeđuje osam bita sa alternativnog ulaza u ALU. Preostalih osam bita se kopira iz najvišeg bita ulaza, što predstavlja predznačno proširenje.

Preostale tri operacije samo prosljeđuju ulaz SRC1 na izlaz, ali glavni dio njihovog izvršenja se obavlja izvan aritmetičko-logičke jedinice.

Blok programskog brojača se sastoji iz registra programskog brojača, dva inkrementera, multipleksera za izbor novog stanja programskog brojača i množača sa reset signalom. Registra programskog brojača se puni na ulaznu ivicu. Njegov izlaz se prosljeđuje bloku za upravljanje memorijom i inkrementeru. Inkrementer obavlja aritmetičku operaciju uvećanja za 1. Izlaz iz prvog inkrementera se dalje prosljeđuje multiplekserima za ulaz u ALU (jer je korisnija ta vrijednost radi čitanja šesnaestbitnih konstanti), na drugi inkrementer i multiplekser za izbor novog stanja programskog brojača. Drugi inkrementer sadrži vrijednost programskog brojača uvećanu za 2. Multiplekser za izbor nove vrijednosti programskog brojača bira jedan od četiri ulaza na bazi signala koje generiše upravljačka jedinica. Nova vrijednost će biti PC+2 ako je bila instrukcija za čitanje ili pisanje memorije (kod 0000 ili 1000) a SRC2

je imao kod 1111 (programski brojač) jer je potrebno preskočiti konstantu. Ako je bila instrukcija koja kao DES T ima programski brojač, bira se ulaz koji je rezultat iz ALU. Ako je bila instrukcija sa kodom 1111 (move and jump), bira se ulaz iz multipleksera SRCB. U svim ostalim slučajevima, bira se PC+1. Nova vrijednost programskog brojača koja je izlaz iz multipleksera se operacijom AND pomnoži sa negiranom vrijednošću RESET signala čime se može postaviti početna vrijednost progarmskog brojača, kada je ovaj signal na vrijednosti 1. Na uzlaznu ivicu rezultantna vrijednost programskog brojača se upiše.

Kontrola memorije se sastoji od dva multipleksera, adresnog i multipleksera izlaza podataka. Adresni multiplekser bira između programskog brojača i ulaza SRC2 u ALU. Adresni multiplekser uglavnom propušta programski brojač prema adresnoj sabirnici, osim kada je instrukcija čitanja ili pisanja u memoriju i signal sata je na niskom nivou (faza izvršenja). Multiplekser izlaza podataka obično prosljeđuje stanje sabirnice ulaznih podataka ka sabirnici izlaznih podataka, osim ako su ispunjena tri uslova: signal sata na niskom nivou, proteklo izvjesno vrijeme (otprilike vrijeme potrebno za prolazak signala kroz četiri logička kola) od spuštanja signala sata na niski nivo i u toku je instrukcija pisanja u memoriju. U tom slučaju se prosljeđuje ulaz u ALU SRC1. Kada su

ovi uslovi ispunjeni i izlazni signal RW se postavlja na 0, koji bira u memoriji između operacija čitanja ili pisanja.

6.6.Detaljna logička šema procesora SVEU16

Ovaj procesor ima oko 6000 osnovnih logičkih kola. To je prilično malo u poređenju sa ostalim šesnaestbitnim procesorima, ali bi crtanje takve detaljne šeme na papiru bilo veoma sporo i nerazumljivo zbog ogromnog broja veza, pogotovo kada jedne prelaze preko drugih. Stoga će se blokovi predstaviti tekstualno, jezikom za opis hardvera. Opisaćemo notaciju koja će se koristiti. Dosta je slična jeziku Verilog koji se koristi za ove svrhe, ali je sintaksa nešto izmjenjena da bi se simulacija procesora obavila običnim kompajlerom za C.

Opis logičkog bloka počinje ključnom riječju *module* iza koga slijedi ime bloka i unutar zagrada lista njegovih ulaznih i izlaznih priključaka, označenih riječima input i output. Preporučljivo je prvo navesti izlazne, a zatim ulazne priključke. Svaki od priključaka ima svoje ime koje se koristi unutar logičkog bloka. Priključci koji se sastoje od više srodnih žica navode se sa brojem tih žica navedenim u uglastim zagradama iza imena priključka.

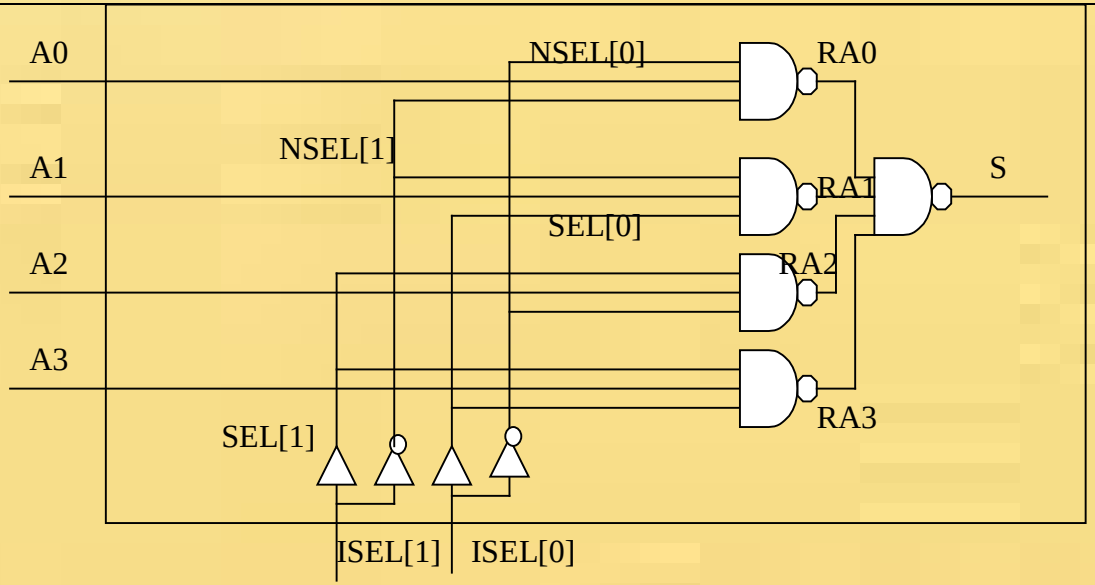
Nakon liste priključaka, unutar vitičastih zagrada opisuje se struktura bloka. Unutar te strukture navode se blokovi od kojih se on sastoji,. Svaki blok

sastojak se navodi sa nabrojanim priključcima unutar zagrada koji moraju biti navedeni u istom broju i redoslijedu kao u opisu bloka. Izlazi iz blokova sastojaka koji se ne prosljeđuju izvan bloka koji se opisuje, trebaju se nabrojati unutar deklaracije *wire*.

Ako je takav izlaz sastavljen od više srodnih žica (npr. 16), navode se unutar deklaracije bus, čiji prvi parametar predstavlja ime skupa žica, a drugi njihov broj. Pojedinačna žica iz skupa žica se onda referiše njenim imenom i rednim brojem navedenim u uglastoj zagradi. Ako se treba povezati više žica, ali ne sve, iz skupa prema drugom bloku koristi se riječ *lines* iza koje se navodi u zagradi redni broj početne žice i koliko se žica prosljeđuje. Za povezivanje svih žica dovoljno je navesti ime skupa.

Osnovna logička kola and, or,, nand, nor i xor imaju uvijek 1 izlaz i do 32 ulaza. Invertor not i bafer buf imaju po jedan ulaz i jedan izlaz. Baferi sa tri stanja bufif0 i bufif1 imaju jedan izlaz i dva ulaza.

Za primjer će se predstaviti sljedeća šema multipleksera sa 4 jednobitna ulaza.



Nakon označavanja pojedinih žica, navedenu šemu bloka možemo predstaviti sljedećim opisom.

```
/* Ulazi 4 linija podataka(4 ulaza, svaki 1 bit). Kombinacijom 2 bita u sel ulazu bira se 1 bit za s izlaz */
module mux4(output s,input a0,input a1,input a2,input a3,input isel[2]) {
    bus(nsel,2);
    bus(sel,2);
    wire(ra0,ra1,ra2,ra3);
    not(nsel[0],isel[0]);
    not(nsel[1],isel[1]);
    buf(sel[0],isel[0]);
    buf(sel[1],isel[1]);
    nand(ra0,a0,nsel[1],nsel[0]);
    nand(ra1,a1,nsel[1], sel[0]);
    nand(ra2,a2,sel[1],nsel[0]);
    nand(ra3,a3, sel[1], sel[0]);
    nand(s,ra0,ra1,ra2,ra3);
}
```

Sada slijedi opis cijelog procesora počevši od opisa gradivnih blokova do bloka koji predstavlja cijeli procesor. Počeće se od multipleksera sa dva ulaza.

```
module mux2(output s,input a0,input a1,input isel) {
    wire(nsel,sel,ra0,ra1,nh);
    buf(sel,isel);
    not(nsel,isel);
    nand(ra0,a0,nsel);
    nand(ra1,a1,sel);
    nand(s,ra0,ra1);
}
```

Šesnaest ovakvih multipleksera čini blok za izbor između dva šesnaestbitna ulaza. Svaki od bita podataka upravljan je posebnim multiplekserom sa jednobitnim ulazom.

```
module mux2x16(input s[16], input a0[16], input a1[16],input sel) {
    mux2(s[0],a0[0],a1[0],sel);
    mux2(s[1],a0[1],a1[1],sel);
    mux2(s[2],a0[2],a1[2],sel);
    mux2(s[3],a0[3],a1[3],sel);
    mux2(s[4],a0[4],a1[4],sel);
    mux2(s[5],a0[5],a1[5],sel);
    mux2(s[6],a0[6],a1[6],sel);
    mux2(s[7],a0[7],a1[7],sel);
    mux2(s[8],a0[8],a1[8],sel);
    mux2(s[9],a0[9],a1[9],sel);
    mux2(s[10],a0[10],a1[10],sel);
    mux2(s[11],a0[11],a1[11],sel);
    mux2(s[12],a0[12],a1[12],sel);
    mux2(s[13],a0[13],a1[13],sel);
    mux2(s[14],a0[14],a1[14],sel);
    mux2(s[15],a0[15],a1[15],sel);
}
```

Nakon ovoga može se napraviti multiplekser sa 4 jednobitna ulaza

```
/* Ulazi 4 linija podataka(4 ulaza, svaki 1 bit). Kombinacijom 2 bita u sel ulazu bira se 1 bit za s izlaz */
module mux4(output s,input a0,input a1,input a2,input a3,input isel[2]) {
    bus(nsel,2);   bus(sel,2);
    wire(ra0,ra1,ra2,ra3);
    not(nsel[0],isel[0]);
    not(nsel[1],isel[1]);
    buf(sel[0],isel[0]);
    buf(sel[1],isel[1]);
    nand(ra0,a0,nsel[1],nsel[0]);
    nand(ra1,a1,nsel[1], sel[0]);
    nand(ra2,a2,sel[1],nsel[0]);
    nand(ra3,a3, sel[1], sel[0]);
    nand(s,ra0,ra1,ra2,ra3);
}
```

Ovi multiplekseri su gradivni elementi multipleksera sa4 šesnaestbitna ulaza.

```
module mux4x16(output s[16], input a0[16], input a1[16], input a2[16], input a3[16],input sel[2]) {
    mux4(s[0],a0[0],a1[0],a2[0],a3[0],sel);
    mux4(s[1],a0[1],a1[1],a2[1],a3[1],sel);
    mux4(s[2],a0[2],a1[2],a2[2],a3[2],sel);
    mux4(s[3],a0[3],a1[3],a2[3],a3[3],sel);
    mux4(s[4],a0[4],a1[4],a2[4],a3[4],sel);
    mux4(s[5],a0[5],a1[5],a2[5],a3[5],sel);
    mux4(s[6],a0[6],a1[6],a2[6],a3[6],sel);
    mux4(s[7],a0[7],a1[7],a2[7],a3[7],sel);
    mux4(s[8],a0[8],a1[8],a2[8],a3[8],sel);
    mux4(s[9],a0[9],a1[9],a2[9],a3[9],sel);
    mux4(s[10],a0[10],a1[10],a2[10],a3[10],sel);
    mux4(s[11],a0[11],a1[11],a2[11],a3[11],sel);
    mux4(s[12],a0[12],a1[12],a2[12],a3[12],sel);
    mux4(s[13],a0[13],a1[13],a2[13],a3[13],sel);
    mux4(s[14],a0[14],a1[14],a2[14],a3[14],sel);
    mux4(s[15],a0[15],a1[15],a2[15],a3[15],sel);
}
```

Zatim slijedi izgled multipleksera sa 16 jednobitnih ulaza

```
module mux16(output s,input a0,input a1,input a2,input a3,input a4,
input a5,input a6,input a7,input a8,input a9,input a10,
input a11,input a12,input a13,input a14,input a15,input isel[4]) {
    bus(nsel,4);    bus(sel,4);
    wire(ra0,ra1,ra2,ra3,ra4,ra5,ra6,ra7,ra8,ra9,ra10,
ra11,ra12,ra13,ra14,ra15);
    not(nsel[0],isel[0]);    not(nsel[1],isel[1]);
    not(nsel[2],isel[2]);    not(nsel[3],isel[3]);
    buf(sel[0],isel[0]);    buf(sel[1],isel[1]);
    buf(sel[2],isel[2]);    buf(sel[3],isel[3]);
    nand(ra0,a0,nsel[3],nsel[2],nsel[1],nsel[0]);
    nand(ra1,a1,nsel[3],nsel[2],nsel[1], sel[0]);
    nand(ra2,a2,nsel[3],nsel[2], sel[1],nsel[0]);
    nand(ra3,a3,nsel[3],nsel[2], sel[1], sel[0]);
    nand(ra4,a4,nsel[3], sel[2],nsel[1],nsel[0]);
    nand(ra5,a5,nsel[3], sel[2],nsel[1], sel[0]);
    nand(ra6,a6,nsel[3], sel[2], sel[1],nsel[0]);
    nand(ra7,a7,nsel[3], sel[2], sel[1], sel[0]);
    nand(ra8,a8, sel[3],nsel[2],nsel[1],nsel[0]);
    nand(ra9,a9, sel[3],nsel[2],nsel[1], sel[0]);
    nand(ra10,a10,sel[3],nsel[2], sel[1],nsel[0]);
    nand(ra11,a11,sel[3],nsel[2], sel[1], sel[0]);
    nand(ra12,a12, sel[3], sel[2],nsel[1],nsel[0]);
    nand(ra13,a13, sel[3], sel[2],nsel[1], sel[0]);
    nand(ra14,a14, sel[3], sel[2], sel[1],nsel[0]);
    nand(ra15,a15, sel[3], sel[2], sel[1], sel[0]);
    nand(s,ra0,ra1,ra2,ra3,ra4,ra5,ra6,ra7,ra8,ra9,ra10,ra11,
ra12,ra13,ra14,ra15);
}
```

Od 16 jednobitnih multipleksera sa 16 ulaza pravi se multiplekser sa 16 šesnaestobitnih ulaza.

```
/* Ulazi 256 linija podataka(16 ulaza, svaki 16 bita). Kombinacijom 4 bita u sel ulazu bira se 16 bita za s izlaz */
module mux16x16(output s[16], input a0[16], input a1[16], input a2[16],
 input a3[16], input a4[16],input a5[16], input a6[16], input a7[16],
 input a8[16], input a9[16], input a10[16], input a11[16],input a12[16],
 input a13[16], input a14[16], input a15[16], input sel[4]) {
mux16(s[0],a0[0],a1[0],a2[0],a3[0],a4[0],a5[0],a6[0],a7[0],a8[0],a9[0],a10[0],a11[0],a12[0],a13[0],a14[0],a15[0],sel);
mux16(s[1],a0[1],a1[1],a2[1],a3[1],a4[1],a5[1],a6[1],a7[1],a8[1],a9[1],a10[1],a11[1],a12[1],a13[1],a14[1],a15[1],sel);
mux16(s[2],a0[2],a1[2],a2[2],a3[2],a4[2],a5[2],a6[2],a7[2],a8[2],a9[2],a10[2],a11[2],a12[2],a13[2],a14[2],a15[2],sel);
mux16(s[3],a0[3],a1[3],a2[3],a3[3],a4[3],a5[3],a6[3],a7[3],a8[3],a9[3],a10[3],a11[3],a12[3],a13[3],a14[3],a15[3],sel);
mux16(s[4],a0[4],a1[4],a2[4],a3[4],a4[4],a5[4],a6[4],a7[4],a8[4],a9[4],a10[4],a11[4],a12[4],a13[4],a14[4],a15[4],sel);
mux16(s[5],a0[5],a1[5],a2[5],a3[5],a4[5],a5[5],a6[5],a7[5],a8[5],a9[5],a10[5],a11[5],a12[5],a13[5],a14[5],a15[5],sel);
mux16(s[6],a0[6],a1[6],a2[6],a3[6],a4[6],a5[6],a6[6],a7[6],a8[6],a9[6],a10[6],a11[6],a12[6],a13[6],a14[6],a15[6],sel);
mux16(s[7],a0[7],a1[7],a2[7],a3[7],a4[7],a5[7],a6[7],a7[7],a8[7],a9[7],a10[7],a11[7],a12[7],a13[7],a14[7],a15[7],sel);
mux16(s[8],a0[8],a1[8],a2[8],a3[8],a4[8],a5[8],a6[8],a7[8],a8[8],a9[8],a10[8],a11[8],a12[8],a13[8],a14[8],a15[8],sel);
mux16(s[9],a0[9],a1[9],a2[9],a3[9],a4[9],a5[9],a6[9],a7[9],a8[9],a9[9],a10[9],a11[9],a12[9],a13[9],a14[9],a15[9],sel);
mux16(s[10],a0[10],a1[10],a2[10],a3[10],a4[10],a5[10],a6[10],a7[10],a8[10],a9[10],a10[10],a11[10],a12[10],a13[10],a14[10],a
15[10],sel);
mux16(s[11],a0[11],a1[11],a2[11],a3[11],a4[11],a5[11],a6[11],a7[11],a8[11],a9[11],a10[11],a11[11],a12[11],a13[11],a14[11],a
15[11],sel);
mux16(s[12],a0[12],a1[12],a2[12],a3[12],a4[12],a5[12],a6[12],a7[12],a8[12],a9[12],a10[12],a11[12],a12[12],a13[12],a14[12],a
15[12],sel);
mux16(s[13],a0[13],a1[13],a2[13],a3[13],a4[13],a5[13],a6[13],a7[13],a8[13],a9[13],a10[13],a11[13],a12[13],a13[13],a14[13],a
15[13],sel);
mux16(s[14],a0[14],a1[14],a2[14],a3[14],a4[14],a5[14],a6[14],a7[14],a8[14],a9[14],a10[14],a11[14],a12[14],a13[14],a14[14],a
15[14],sel);
mux16(s[15],a0[15],a1[15],a2[15],a3[15],a4[15],a5[15],a6[15],a7[15],a8[15],a9[15],a10[15],a11[15],a12[15],a13[15],a14[15],a
15[15],sel);
}
```

Demultiplekser ima samo jedan ulaz podataka i 16 izlaza, te četiri kontrolna bita.

```
/* Ulazi 1 linija podataka. Kombinacijom 4 bita u sel ulazu bira se 16 izlaza bit za s ulaz */
module demux16(output outlines[16],input inputline,input select[4]) {
  bus(nselect,4);
  not(nselect[0],select[0]);
  not(nselect[1],select[1]);
  not(nselect[2],select[2]);
  not(nselect[3],select[3]);
 and (outlines[0],nselect[3],nselect[2],nselect[1],nselect[0],inputline);
  and (outlines[1],nselect[3],nselect[2],nselect[1],select[0],inputline);
  and (outlines[2],nselect[3],nselect[2],select[1],nselect[0],inputline);
  and (outlines[3],nselect[3],nselect[2],select[1],select[0],inputline);
  and (outlines[4],nselect[3],select[2],nselect[1],nselect[0],inputline);
  and (outlines[5],nselect[3],select[2],nselect[1],select[0],inputline);
  and (outlines[6],nselect[3],select[2],select[1],nselect[0],inputline);
  and (outlines[7],nselect[3],select[2],select[1],select[0],inputline);
  and (outlines[8],select[3],nselect[2],nselect[1],nselect[0],inputline);
  and (outlines[9],select[3],nselect[2],nselect[1],select[0],inputline);
  and (outlines[10],select[3],nselect[2],select[1],nselect[0],inputline);
  and (outlines[11],select[3],nselect[2],select[1],select[0],inputline);
  and (outlines[12],select[3],select[2],nselect[1],nselect[0],inputline);
  and (outlines[13],select[3],select[2],nselect[1],select[0],inputline);
  and (outlines[14],select[3],select[2],select[1],nselect[0],inputline);
  and (outlines[15],select[3],select[2],select[1],select[0],inputline);
}
```

Sada slijedi dizajn aritmetičko logičke jedinice. Počinje se od jednostavnih blokova koji realizuju 16 bitne logičke operacije. Blok koji realizuje AND povezuje pojedinačne bite oba ulaza prema izlazima.

```
module and16b(output s[16],input a[16],input b[16]) {
    and(s[0],a[0],b[0]);
    and(s[1],a[1],b[1]);
    and(s[2],a[2],b[2]);
    and(s[3],a[3],b[3]);
    and(s[4],a[4],b[4]);
    and(s[5],a[5],b[5]);
    and(s[6],a[6],b[6]);
    and(s[7],a[7],b[7]);
    and(s[8],a[8],b[8]);
    and(s[9],a[9],b[9]);
    and(s[10],a[10],b[10]);
    and(s[11],a[11],b[11]);
    and(s[12],a[12],b[12]);
    and(s[13],a[13],b[13]);
    and(s[14],a[14],b[14]);
    and(s[15],a[15],b[15]);
}
```

Blok za realizaciju funkcije OR je slično jednostavan.

```
module or16b(output s[16],input a[16],input b[16]) {
    or(s[0],a[0],b[0]);
    or(s[1],a[1],b[1]);
    or(s[2],a[2],b[2]);
    or(s[3],a[3],b[3]);
    or(s[4],a[4],b[4]);
    or(s[5],a[5],b[5]);
    or(s[6],a[6],b[6]);
    or(s[7],a[7],b[7]);
    or(s[8],a[8],b[8]);
    or(s[9],a[9],b[9]);
    or(s[10],a[10],b[10]);
    or(s[11],a[11],b[11]);
    or(s[12],a[12],b[12]);
    or(s[13],a[13],b[13]);
    or(s[14],a[14],b[14]);
    or(s[15],a[15],b[15]);
}
```

Naredni blok realizuje ekskluzivnu disjunkciju.

```
module xor16b(output s[16],input a[16],input b[16]) {
    xor (s[0],a[0],b[0]);
    xor (s[1],a[1],b[1]);
    xor (s[2],a[2],b[2]);
    xor (s[3],a[3],b[3]);
    xor (s[4],a[4],b[4]);
    xor (s[5],a[5],b[5]);
    xor (s[6],a[6],b[6]);
    xor (s[7],a[7],b[7]);
    xor (s[8],a[8],b[8]);
    xor (s[9],a[9],b[9]);
    xor (s[10],a[10],b[10]);
    xor (s[11],a[11],b[11]);
    xor (s[12],a[12],b[12]);
    xor (s[13],a[13],b[13]);
    xor (s[14],a[14],b[14]);
    xor (s[15],a[15],b[15]);
}
```

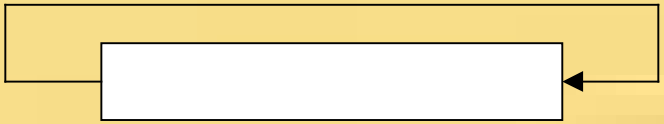
Naredni blok predstavlja brzi četverobitni sabirač. Koristi dosta veliku logičku funkciju za račun prijenosa kako bi se smanjio broj nivoa logičkih kola. Naime, lančano vezani puni sabirači mogu zahtijevati prolazak kroz veći broj kaskadno vezanih logičkih kola dok se prijenos ne prebaci do najvišeg bita. S druge strane, po cijenu većeg broja logičkih kola, broj njihovih nivoa se može smanjiti i samim tim povećati brzinu i frekvenciju rada procesora.

```
module fadder4(output S[4],output cout,input a[4],input b[4],input cin) {
    bus(G,4);
    bus(P,4);
    bus(C,4);
    wire(PG,GG,p3g2,p3p2g1,p3p2p1g0,p3p2p1p0cin,
p0cin,p1g0,p1p0cin,p2p1g0,p2p1p0cin,p2g1);
    and(G[0],a[0],b[0]);
    and(G[1],a[1],b[1]);
    and(G[2],a[2],b[2]);
    and(G[3],a[3],b[3]);
    xor(P[0],a[0],b[0]);
    xor(P[1],a[1],b[1]);
    xor(P[2],a[2],b[2]);
    xor(P[3],a[3],b[3]);
    xor(S[0],P[0],cin);
    xor(S[1],P[1],C[1]);
    xor(S[2],P[2],C[2]);
    xor(S[3],P[3],C[3]);
    and(p3g2,P[3],G[2]);
    and(p3p2g1,P[3],P[2],G[1]) ;
    and(p3p2p1g0,P[3],P[2],P[1],G[0]);
    and(p3p2p1p0cin,P[3],P[2],P[1],P[0],cin);
    and(p0cin,P[0],cin);
    and(p1g0,P[1],G[0]);
    and(p1p0cin,P[1],P[0],cin);
    and(p2p1g0,P[2],P[1],G[0]);
    and(p2p1p0cin,P[2],P[1],P[0],cin);
    and(p2g1,P[2],G[1]);
    or(C[1],G[0],p0cin);
    or(C[2],G[1],p1g0,p1p0cin);
    or(C[3],G[2],p2g1,p2p1g0, p2p1p0cin);
    or(cout,G[3], p3g2, p3p2g1, p3p2p1g0,p3p2p1p0cin);
}
```

Četiri puna sabirača se spajaju u jedan šesnaestbitni sabirač.

```
/* Sabira 2 16 bitna broja (ukupno 33 ulazne linije i 17 izlaznih) */
module adder16b(output s[16],output cout,
input a[16],input b[16],input cin) {
    wire(c0,c1,c2);
    fadder4(lines(s,0,4),c0,lines(a,0,4),lines(b,0,4),cin);
    fadder4(lines(s,4,4),c1,lines(a,4,4),lines(b,4,4),c0);
    fadder4(lines(s,8,4),c2,lines(a,8,4),lines(b,8,4),c1);
    fadder4(lines(s,12,4),cout,lines(a,12,4),lines(b,12,4),c2);
}
```

Shema operacije rotiranja data je na slici.



Ako se ova operacija obavlja jedan put, bit koji je bio na najvišoj poziciji pojavljuje se na najnižoj, dok se ostali pomjere za jedno mjesto ulijevo. Na primjer, ako je ulazna vrijednost bila binarno 0001001010010011, rezultujuća vrijednost operacije će biti 0010010100100110. Ako se operacija obavlja pet puta, rezultantna vrijednost će biti 0101001001100010.

Operacija rotiranja realizuje se pomoću 16 multipleksera. Svaki od njih predstavlja po jedan bit izlaza, a izbor ulaza u multiplekser se odabira prema broju koliko puta treba rotirati.

```
/* Rotira a b puta ulijevo */
module rotatorl16(output s[16],input a[16],input sel[4]) {
    mux16 (s[0],a[0],a[15],a[14],a[13],a[12],a[11],a[10],
a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1],sel);
    mux16 (s[1],a[1],a[0],a[15],a[14],a[13],a[12],a[11],a[10],
a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2],sel);
    mux16 (s[2],a[2],a[1],a[0],a[15],a[14],a[13],a[12],a[11],a[10],
a[9],a[8],a[7],a[6],a[5],a[4],a[3],sel);
    mux16 (s[3],a[3],a[2],a[1],a[0],a[15],a[14],a[13],a[12],a[11],
a[10],a[9],a[8],a[7],a[6],a[5],a[4],sel);
    mux16 (s[4],a[4],a[3],a[2],a[1],a[0],a[15],a[14],a[13],a[12],
a[11],a[10],a[9],a[8],a[7],a[6],a[5],sel);
    mux16 (s[5],a[5],a[4],a[3],a[2],a[1],a[0],a[15],a[14],a[13],
a[12],a[11],a[10],a[9],a[8],a[7],a[6],sel);
    mux16 (s[6],a[6],a[5],a[4],a[3],a[2],a[1],a[0],a[15],a[14],
a[13],a[12],a[11],a[10],a[9],a[8],a[7],sel);
    mux16 (s[7],a[7],a[6],a[5],a[4],a[3],a[2],a[1],a[0],a[15],
a[14],a[13],a[12],a[11],a[10],a[9],a[8],sel);
    mux16 (s[8],a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1],a[0],
a[15],a[14],a[13],a[12],a[11],a[10],a[9],sel);
    mux16 (s[9],a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1],
a[0],a[15],a[14],a[13],a[12],a[11],a[10],sel);
    mux16 (s[10],a[10],a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2],
a[1],a[0],a[15],a[14],a[13],a[12],a[11],sel);
    mux16 (s[11],a[11],a[10],a[9],a[8],a[7],a[6],a[5],a[4],a[3],
a[2],a[1],a[0],a[15],a[14],a[13],a[12],sel);
    mux16 (s[12],a[12],a[11],a[10],a[9],a[8],a[7],a[6],a[5],a[4],
a[3],a[2],a[1],a[0],a[15],a[14],a[13],sel);
    mux16 (s[13],a[13],a[12],a[11],a[10],a[9],a[8],a[7],a[6],a[5],
a[4],a[3],a[2],a[1],a[0],a[15],a[14],sel);
    mux16 (s[14],a[14],a[13],a[12],a[11],a[10],a[9],a[8],a[7],a[6],
a[5],a[4],a[3],a[2],a[1],a[0],a[15],sel);
    mux16 (s[15],a[15],a[14],a[13],a[12],a[11],a[10],a[9],a[8],a[7],
a[6],a[5],a[4],a[3],a[2],a[1],a[0],sel);
}
```

Shema operacije pomicanja udesno data je na slici. Ova operacija predstavlja dijeljenje sa 2, a ako se ponavlja više puta, onda predstavlja dijeljenje bilo kojim stepenom broja 2.

Ako se ova operacija obavlja jedan put, bit koji je bio na najnižoj poziciji se gubi, dok se ostali pomjere za jedno mjesto udesno. Na primjer, ako je ulazna vrijednost bila binarno 0001001010010011, rezultantna vrijednost operacije će biti 0000100101001001. Ako se operacija obavlja pet puta, rezultantna vrijednost će biti 0000000010010100.

```
/* Pomjera a b puta desno */
module shifterr16(input s[16],input a[16],input sel[4]) {
    mux16 (s[0],a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],
a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15],sel);
    mux16 (s[1],a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],
a[9],a[10],a[11],a[12],a[13],a[14],a[15],0,sel);
    mux16 (s[2],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],
a[10],a[11],a[12],a[13],a[14],a[15],0,0,sel);
    mux16 (s[3],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],
a[11],a[12],a[13],a[14],a[15],0,0,0,sel);
    mux16 (s[4],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],
a[12],a[13],a[14],a[15],0,0,0,0,sel);
    mux16 (s[5],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],
a[13],a[14],a[15],0,0,0,0,0,sel);
    mux16 (s[6],a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[13],
a[14],a[15],0,0,0,0,0,0,sel);
    mux16 (s[7],a[7],a[8],a[9],a[10],a[11],a[12],a[13],a[14],
a[15],0,0,0,0,0,0,0,sel);
    mux16 (s[8],a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15],
0,0,0,0,0,0,0,0,sel);
    mux16 (s[9],a[9],a[10],a[11],a[12],a[13],a[14],a[15],0,0,
0,0,0,0,0,0,0,sel);
    mux16 (s[10],a[10],a[11],a[12],a[13],a[14],a[15],0,0,0,0,
0,0,0,0,0,0,sel);
    mux16 (s[11],a[11],a[12],a[13],a[14],a[15],0,0,0,0,0,0,0,
0,0,0,0,sel);
    mux16 (s[12],a[12],a[13],a[14],a[15],0,0,0,0,0,0,0,0,0,0,0,0,sel);
    mux16 (s[13],a[13],a[14],a[15],0,0,0,0,0,0,0,0,0,0,0,0,0,sel);
    mux16 (s[14],a[14],a[15],0,0,0,0,0,0,0,0,0,0,0,0,0,0,sel);
    mux16 (s[15],a[15],0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,sel);
}
```

Shema operacije pomjeranja ulijevo data je na slici. Ova operacija predstavlja množenje sa 2, a ako se ponavlja više puta, onda predstavlja množenje bilo kojim stepenom broja 2.

Ako se ova operacija obavlja jedan put, bit koji je bio na najvišoj poziciji se gubi, dok se ostali pomjere za jedno mjesto ulijevo. Na primjer, ako je ulazna vrijednost bila binarno 0001001010010011, rezultantna vrijednost operacije će biti 0010010100100110. Ako se operacija obavlja pet puta, rezultantna vrijednost će biti 0101001001100000.

```
/* Pomjera a b puta ulijevo */
module shifterl16(output s[16],input a[16],input sel[4]) {
    mux16 (s[0],a[0],0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,sel);
    mux16 (s[1],a[1],a[0],0,0,0,0,0,0,0,0,0,0,0,0,0,sel);
    mux16 (s[2],a[2],a[1],a[0],0,0,0,0,0,0,0,0,0,0,0,0,sel);
    mux16 (s[3],a[3],a[2],a[1],a[0],0,0,0,0,0,0,0,0,0,0,0,sel);
    mux16 (s[4],a[4],a[3],a[2],a[1],a[0],0,0,0,0,0,0,0,0,0,0,sel);
    mux16 (s[5],a[5],a[4],a[3],a[2],a[1],a[0],0,0,0,0,0,0,0,0,0,sel);
    mux16 (s[6],a[6],a[5],a[4],a[3],a[2],a[1],a[0],
0,0,0,0,0,0,0,0,sel);
    mux16 (s[7],a[7],a[6],a[5],a[4],a[3],a[2],a[1],a[0],
0,0,0,0,0,0,0,0,sel);
    mux16 (s[8],a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1],
a[0],0,0,0,0,0,0,0,sel);
    mux16 (s[9],a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1],
a[0],0,0,0,0,0,0,sel);
    mux16 (s[10],a[10],a[9],a[8],a[7],a[6],a[5],a[4],a[3],
a[2],a[1],a[0],0,0,0,0,0,sel);
    mux16 (s[11],a[11],a[10],a[9],a[8],a[7],a[6],a[5],a[4],
a[3],a[2],a[1],a[0],0,0,0,0,sel);
    mux16 (s[12],a[12],a[11],a[10],a[9],a[8],a[7],a[6],a[5],
a[4],a[3],a[2],a[1],a[0],0,0,0,sel);
    mux16 (s[13],a[13],a[12],a[11],a[10],a[9],a[8],a[7],a[6],
a[5],a[4],a[3],a[2],a[1],a[0],0,0,sel);
    mux16 (s[14],a[14],a[13],a[12],a[11],a[10],a[9],a[8],
a[7],a[6],a[5],a[4],a[3],a[2],a[1],a[0],0,sel);
    mux16 (s[15],a[15],a[14],a[13],a[12],a[11],a[10],
a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1],a[0],sel);
}
```

Aritmetičko pomjeranje ulijevo je isto kao i obično pomjeranje ulijevo. To nije slučaj sa pomicanjem udesno. Pomicanje udesno predstavlja predznačeno dijeljenje sa stepenom broja 2. U tom slučaju se na mjesta novoformiranih

bitova sa lijeve strane rezultata ubacuje ne nula, nego najviši bit izvora. Na primjer, ako je ulazna vrijednost bila binarno 1001001010010011, rezultantna vrijednost operacije će biti 1100100101001001. Ako se operacija obavlja pet puta, rezultantna vrijednost će biti 1111110010010100.

```
/* Pomjera aritmetički a b puta desno */
module ashifterr16(output s[16],input a[16],input sel[4]) {
    mux16 (s[0],a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],
a[10],a[11],a[12],a[13],a[14],a[15],sel);
    mux16 (s[1],a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],
a[11],a[12],a[13],a[14],a[15],a[15],sel);
    mux16 (s[2],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],
a[12],a[13],a[14],a[15],a[15],a[15],sel);
    mux16 (s[3],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],
a[13],a[14],a[15],a[15],a[15],a[15],sel);
    mux16 (s[4],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[13],
a[14],a[15],a[15],a[15],a[15],a[15],sel);
    mux16 (s[5],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[13],a[14],
a[15],a[15],a[15],a[15],a[15],a[15],sel);
    mux16 (s[6],a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15],
a[15],a[15],a[15],a[15],a[15],a[15],sel);
    mux16 (s[7],a[7],a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15],
a[15],a[15],a[15],a[15],a[15],a[15],a[15],sel);
    mux16 (s[8],a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15],a[15],
a[15],a[15],a[15],a[15],a[15],a[15],a[15],sel);
    mux16 (s[9],a[9],a[10],a[11],a[12],a[13],a[14],a[15],a[15],a[15],
a[15],a[15],a[15],a[15],a[15],a[15],a[15],sel);
    mux16 (s[10],a[10],a[11],a[12],a[13],a[14],a[15],a[15],a[15],a[15],
a[15],a[15],a[15],a[15],a[15],a[15],a[15],sel);
    mux16 (s[11],a[11],a[12],a[13],a[14],a[15],a[15],a[15],a[15],a[15],
a[15],a[15],a[15],a[15],a[15],a[15],a[15],sel);
    mux16 (s[12],a[12],a[13],a[14],a[15],a[15],a[15],a[15],a[15],a[15],
a[15],a[15],a[15],a[15],a[15],a[15],a[15],sel);
    mux16 (s[13],a[13],a[14],a[15],a[15],a[15],a[15],a[15],a[15],a[15],
a[15],a[15],a[15],a[15],a[15],a[15],a[15],sel);
    mux16 (s[14],a[14],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],
a[15],a[15],a[15],a[15],a[15],a[15],a[15],sel);
    mux16 (s[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],
a[15],a[15],a[15],a[15],a[15],a[15],a[15],sel);
}
```

Četiri moguća načina pomicanja bitova biraju se novim multiplekserom sa 4 ulaza koristeći bitove na pozicijama 4 i 5 kao kontrolnim bitima za izbor operacije.

```
module allshift(output s[16],input a[16],input sel[16]) {
    bus(ashifterrout,16);    bus(shifterout,16);
    bus(rotatorlout,16);    bus(shifterlout,16);
    ashifterr16(ashifterrout,a,lines(sel,0,3));
    shifterr16(shifterout,a,lines(sel,0,3));
    rotatorl16(rotatorlout,a,lines(sel,0,3));
    shifterl16(shifterlout,a,lines(sel,0,3));
    mux4x16(s,ashifterrout,shifterout,rotatorlout,shifterlout,
lines(sel,4,2));
}
```

Blok aritmetičko logičke jedinice za račun predznačene konstante samo prosljeđuje ulaze prema izlazu, pri čemu se najviši bit multiplicira

```
module signedconst(output s[16],input lonible[4],input hinible[4]) {
    buf(s[0],lonible[0]);    buf(s[1],lonible[1]);
    buf(s[2],lonible[2]);    buf(s[3],lonible[3]);
    buf(s[4],hinible[0]);    buf(s[5],hinible[1]);
    buf(s[6],hinible[2]);    buf(s[7],hinible[3]);
    buf(s[8],hinible[3]);    buf(s[9],hinible[3]);
    buf(s[10],hinible[3]);    buf(s[11],hinible[3]);
    buf(s[12],hinible[3]);    buf(s[13],hinible[3]);
    buf(s[14],hinible[3]);    buf(s[15],hinible[3]);
}
```

Da bi se sabirač mogao iskoristiti i za oduzimanje dodaje se blok koji računa prvi komplement kod broja ako je kontrolni bit jednak 1.

```
/* Modul koji vrati 1. komplement broja ako je cond=1 */
module complementer (output s[16],input a[16],input cond) {
    xor (s[0],a[0],cond);    xor (s[1],a[1],cond);
    xor (s[2],a[2],cond);    xor (s[3],a[3],cond);
    xor (s[4],a[4],cond);    xor (s[5],a[5],cond);
    xor (s[6],a[6],cond);    xor (s[7],a[7],cond);
    xor (s[8],a[8],cond);    xor (s[9],a[9],cond);
    xor (s[10],a[10],cond);    xor (s[11],a[11],cond);
    xor (s[12],a[12],cond);    xor (s[13],a[13],cond);
    xor (s[14],a[14],cond);    xor (s[15],a[15],cond);
}
```

Množač je najveći dio aritmetičko logičke jedinice. Množač šesnaestbitnog sa jednobitnim brojem je samo skup od AND blokova.

```
module mulwith1or0(output s[16],input a0,input a1,input a2,input a3,
input a4,input a5,input a6,input a7,input a8,input a9,
input a10,input a11,input a12,input a13,input a14,input a15,input sel) {
    and(s[0],a0,sel);    and(s[1],a1,sel);
    and(s[2],a2,sel);    and(s[3],a3,sel);
    and(s[4],a4,sel);    and(s[5],a5,sel);
    and(s[6],a6,sel);    and(s[7],a7,sel);
    and(s[8],a8,sel);    and(s[9],a9,sel);
    and(s[10],a10,sel);    and(s[11],a11,sel);
    and(s[12],a12,sel);    and(s[13],a13,sel);
    and(s[14],a14,sel);    and(s[15],a15,sel);
}
```

Množač radi na sljedećem principu. Prvi faktor se nepomjeren pomnoži za nultim bitom drugog faktora. Prvi faktor pomjeren 1 put se pomnože sa prvim bitom drugog faktora i tako se formira 16 sabiraka, na kraju se prvi faktor pomjeren 15 puta pomnoži sa 15-im bitom drugog faktora. Dobiveni sabirci se ubace u 8 sabirača, njihovi rezultati u 4, pa u 2 i na kraju u finalni sabirač.

```
module mul16b(output s[16],input a[16],input b[16]) {
    bus(s0,16);    bus(s1,16);    bus(s2,16);
    bus(s3,16);    bus(s4,16);    bus(s5,16);
    bus(s6,16);    bus(s7,16);    bus(s8,16);
    bus(s9,16);    bus(s10,16);    bus(s11,16);
    bus(s12,16);    bus(s13,16);    bus(s14,16);
    bus(s15,16);    bus(p0,16);    bus(p1,16);
    bus(p2,16);    bus(p3,16);    bus(p4,16);
    bus(p5,16);    bus(p6,16);    bus(p7,16);
    bus(q0,16);    bus(q1,16);    bus(q2,16);
    bus(q3,16);    bus(r0,16);    bus(r1,16);
    wire(cout);
    mulwith1or0(s0,a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],
a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15],b[0]);
    mulwith1or0(s1,0,a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],
a[8],a[9],a[10],a[11],a[12],a[13],a[14],b[1]);
    mulwith1or0(s2,0,0,a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],
a[8],a[9],a[10],a[11],a[12],a[13],b[2]);
    mulwith1or0(s3,0,0,0,a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],
a[8],a[9],a[10],a[11],a[12],b[3]);
    mulwith1or0(s4,0,0,0,0,a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7]
,a[8],a[9],a[10],a[11],b[4]);
    mulwith1or0(s5,0,0,0,0,0,a[0],a[1],a[2],a[3],a[4],a[5],a[6],
a[7],a[8],a[9],a[10],b[5]);
    mulwith1or0(s6,0,0,0,0,0,0,a[0],a[1],a[2],a[3],a[4],a[5],a[6],
a[7],a[8],a[9],b[6]);
    mulwith1or0(s7,0,0,0,0,0,0,0,a[0],a[1],a[2],a[3],a[4],a[5],a[6],
a[7],a[8],b[7]);
    mulwith1or0(s8,0,0,0,0,0,0,0,0,a[0],a[1],a[2],a[3],a[4],a[5],a[6],
a[7],b[8]);
    mulwith1or0(s9,0,0,0,0,0,0,0,0,0,a[0],a[1],a[2],a[3],a[4],a[5],
a[6],b[9]);
    mulwith1or0(s10,0,0,0,0,0,0,0,0,0,0,a[0],a[1],a[2],a[3],a[4],
a[5],b[10]);
    mulwith1or0(s11,0,0,0,0,0,0,0,0,0,0,0,a[0],a[1],a[2],a[3],
a[4],b[11]);
    mulwith1or0(s12,0,0,0,0,0,0,0,0,0,0,0,0,a[0],a[1],a[2],a[3],b[12]);
```

```
    mulwith1or0(s13,0,0,0,0,0,0,0,0,0,0,0,0,0,a[0],a[1],a[2],b[13]);
    mulwith1or0(s14,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a[0],a[1],b[14]);
    mulwith1or0(s15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,a[0],b[15]);
    adder16b(p0,cout,s0,s1,0);    adder16b(p1,cout,s2,s3,0);
    adder16b(p2,cout,s4,s5,0);    adder16b(p3,cout,s6,s7,0);
    adder16b(p4,cout,s8,s9,0);    adder16b(p5,cout,s10,s11,0);
    adder16b(p6,cout,s12,s13,0);    adder16b(p7,cout,s14,s15,0);
    adder16b(q0,cout,p0,p1,0);    adder16b(q1,cout,p2,p3,0);
    adder16b(q2,cout,p4,p5,0);    adder16b(q3,cout,p6,p7,0);
    adder16b(r0,cout,q0,q1,0);    adder16b(r1,cout,q2,q3,0);
    adder16b(s,cout,r0,r1,0);
}
```

Operacije koje za rezultat imaju samo jedan bit, a ostalih 15 su nule se realizuju sljedećim blokom.

```
module onebitresult(output s[16],input last) {
    buf(s[0],last);
    zero(s[1]);    zero(s[2]);    zero(s[3]);
    zero(s[4]);    zero(s[5]);    zero(s[6]);
    zero(s[7]);    zero(s[8]);    zero(s[9]);
    zero(s[10]);    zero(s[11]);    zero(s[12]);
    zero(s[13]);    zero(s[14]);    zero(s[15]);
}
```

Konačno možemo sklopiti cijelu aritmetičko-logičku jedinicu od pojedinih blokova i veza.

```
module alu(output s[16],input a[16],input b[16],input hinible[4],
input lonible[4],input op[4]) {
    wire(cout);
    bus(adderout,16);    bus(andout,16);    bus(orout,16);
    bus(xorout,16);    bus(shiftout,16);    bus(multout,16);
    bus(constout,16);    bus(gtunsout,16);    bus(gtsinout,16);
    bus(ltunsout,16);    bus(ltsinout,16);    bus(eqout,16);
    bus(negop,4);    bus(adderin,16);
    wire(carryin,zeroflag,less,greater,greaters);
    not(negop[0],op[0]);    not(negop[1],op[1]);
    not(negop[2],op[2]);    not(negop[3],op[3]);
    nand(carryin,op[0],negop[1],negop[2],negop[3]);
    complementer(adderin,b,carryin);
    adder16b(adderout,cout,a,adderin,carryin);
    and16b(andout,a,b);   or16b(orout,a,b);    xor16b(xorout,a,b);
    mul16b(multout,a,b);   allshift(shiftout,a,b);
    nor(zeroflag,adderout[15],adderout[14],adderout[13],
adderout[12],adderout[11],adderout[10],adderout[9],
adderout[8],adderout[7],        adderout[6],adderout[5],adderout[4],adderout[3],
adderout[2],adderout[1],adderout[0]);
    not(less,cout);    nor(greater,less,zeroflag);
    nor(greaters,adderout[15],zeroflag);     ž
    onebitresult(gtunsout,greater);    onebitresult(gtsinout,greaters);
    onebitresult(ltunsout,less);   onebitresult(ltsinout,adderout[15]);
    onebitresult(eqout,zeroflag);
    signedconst(constout,lonible,hinible);
    mux16x16(s,a,adderout,adderout,andout,orout,xorout,
shiftout,multout,a,constout,gtunsout,gtsinout,ltunsout,
ltsinout,eqout,a,op);
}
```

Sljedeći blok služi za realizaciju kašnjenja signala. Signal prosto prolazi kroz šest komponenti.

```
module delay(output out,input inp) {
    wire(stage1,stage2,stage3,stage4,stage5,stage6);
    buf(stage1,inp);
    buf(stage2,stage1);
    buf(stage3,stage2);
    buf(stage4,stage3);
    buf(stage5,stage4);
    buf(stage6,stage5);
    buf(out,stage6);
}
```

Za realizaciju registara najprije treba spojiti D flip flop okidan na uzlaznu ivicu.

```
module dff(output q,input d,input clk) {
    wire(n0,n1,n3,nq,n5);
    nand(n0,n5,n1);
    nand(n1,n0,clk);
    nand(q,nq,n1);
    nand(n3,n1,clk,n5);
    nand(nq,q,n3);
    nand(n5,d,n3);
}
```

Registri se sastoje od 16 D flip flopova sa zajedničkim signalom sata.

```
module register16(output q[16],input d[16],input clk) {
    dff(q[0],d[0],clk);
    dff(q[1],d[1],clk);
    dff(q[2],d[2],clk);
    dff(q[3],d[3],clk);
    dff(q[4],d[4],clk);
    dff(q[5],d[5],clk);
    dff(q[6],d[6],clk);
    dff(q[7],d[7],clk);
    dff(q[8],d[8],clk);
    dff(q[9],d[9],clk);
    dff(q[10],d[10],clk);
    dff(q[11],d[11],clk);
    dff(q[12],d[12],clk);
    dff(q[13],d[13],clk);
    dff(q[14],d[14],clk);
    dff(q[15],d[15],clk);
}
```

Petnaest registara opšte namjene povezani su sa tri multipleksera i jednim demultiplekserom. Izlazni multiplekseri idu na ALU. Ulazni demultiplekser povezuje signal sata prema registrima. Ulazni multiplekser određuje da li registar prima vrijednost iz memorije ili iz ALU.

```
module registerblock(output ALUA[16],output ALUB[16],input ALUOut[16],
input datain[16],input Regsinput[16],input PCplus1[16],
input dest[4],input src1[4],input src2[4],input destgate,input clk) {
  // Opšti registri, pune se na ulaznu ivicu
   bus(R0output,16);
   bus(R1output,16);
   bus(R2output,16);
   bus(R3output,16);
   bus(R4output,16);
   bus(R5output,16);
   bus(R6output,16);
   bus(R7output,16);
   bus(R8output,16);
   bus(R9output,16);
   bus(R10output,16);
   bus(R11output,16);
   bus(R12output,16);
   bus(R13output,16);
   bus(R14output,16);
   bus(clkreg,16);
   mux2x16(Regsinput, ALUOut, datain, destgate);
   register16(R0output,Regsinput,clkreg[0]);
   register16(R1output,Regsinput,clkreg[1]);
   register16(R2output,Regsinput,clkreg[2]);
   register16(R3output,Regsinput,clkreg[3]);
   register16(R4output,Regsinput,clkreg[4]);
   register16(R5output,Regsinput,clkreg[5]);
   register16(R6output,Regsinput,clkreg[6]);
   register16(R7output,Regsinput,clkreg[7]);
   register16(R8output,Regsinput,clkreg[8]);
   register16(R9output,Regsinput,clkreg[9]);
   register16(R10output,Regsinput,clkreg[10]);
   register16(R11output,Regsinput,clkreg[11]);
   register16(R12output,Regsinput,clkreg[12]);
   register16(R13output,Regsinput,clkreg[13]);
   register16(R14output,Regsinput,clkreg[14]);
   mux16x16(ALUA,R0output,R1output,R2output,R3output,R4output,
R5output,R6output,R7output,  R8output,R9output,R10output,
```

```
R11output,R12output,R13output,R14output,PCplus1,src1);
   mux16x16(ALUB,R0output,R1output,R2output,R3output,R4output,
R5output,R6output,R7output, R8output,R9output,R10output,R11output,
R12output,R13output,R14output,PCplus1,src2);
   demux16(clkreg,clk,dest);
}
```

Upravljačka jedinica sadrži instrukcijski registar koji se puni na silaznu ivicu. i generiše na bazi njegovog sadržaja signale za sve multipleksere u procesoru. Multiplekseri vezani za ALU ulaz i izlaz, kao i demultiplekser za ulaz dobivaju kontrolne signale direktno iz instrukcijskog registra. Preostala četiri multipleksera zahtijevaju male operacije između bitova tog registra, zavisno od operacionog koda, odredišnog registra i stanja sata.

```
module controlunit(output opcode[4],output dest[4],output src1[4],
output src2[4],output destgate, output addrgate,output dataoutgate, output pcgate[2],output memreadwr,input
datain[16],input irq,input clk) {
   wire(nclk);
   not(nclk,clk);
// Instrukcijski registar, na silaznu ivicu,
   bus(IRinput,16);   bus(IRoutput,16);
   bus(doint,16); /*  Napuni instrukciju interapta */
   mux2x16(IRinput,datain, doint, irq);
   register16(IRoutput,IRinput,nclk);
   buf(opcode[3],IRoutput[15]);   buf(opcode[2],IRoutput[14]);
   buf(opcode[1],IRoutput[13]);   buf(opcode[0],IRoutput[12]);
   buf(dest[3],IRoutput[11]);   buf(dest[2],IRoutput[10]);
   buf(dest[1],IRoutput[9]);    buf(dest[0],IRoutput[8]);
   buf(src1[3],IRoutput[7]);   buf(src1[2],IRoutput[6]);
   buf(src1[1],IRoutput[5]);   buf(src1[0],IRoutput[4]);
   buf(src2[3],IRoutput[3]);   buf(src2[2],IRoutput[2]);
   buf(src2[1],IRoutput[1]);   buf(src2[0],IRoutput[0]);
   printbus(IRoutput,0,16);
   wire(op0,op8,op15);  // tri posebne instrukcije
   bus(nopcode,4);
   not(nopcode[0],opcode[0]);   not(nopcode[1],opcode[1]);
   not(nopcode[2],opcode[2]);   not(nopcode[3],opcode[3]);
   and(op0,nopcode[0],nopcode[1],nopcode[2],nopcode[3]);
   and(op8,opcode[0],nopcode[1],nopcode[2],nopcode[3]);
   and(op15,opcode[0],opcode[1],opcode[2],opcode[3]);
   /* Upravljanje programskim brojačem   */
   wire(pcdest,pcsrc,njal);
   wire(memop); /* memorijska instrukcija */
   wire(NextPC0a,NextPC0b);
   or(memop,op0,op8);
   not(njal,op15);
   and(pcdest,dest[0],dest[1],dest[2],dest[3]);
   and(pcsrc,src2[0],src2[1],src2[2],src2[3]);
   or(pcgate[1],op15,pcdest);
   and(NextPC0a,njal,pcdest);
   and(NextPC0b,memop,njal,pcsrc);
   or(pcgate[0],NextPC0a,NextPC0b);
```

```
  /*     Memorijski signali */
   or(memop,op0,op8);
   wire(delayed);
   delay(delayed,nclk);  // Pošto je memorija, ubacuje se malo kašnjenje
   and(addrgate,memop,nclk,delayed);  /* clock nizak i memop operacija */
   nand(memreadwr,op8,delayed,nclk);
   buf(dataoutgate,op8);
   buf(destgate,op0);
}
```

Veza na sabirnicu je veoma jednostavan blok koji se sastoji od dva multipleksera, za izbor registra koji ide na adresnu sabirnicu i uzbor izlaza podataka.

```
module meminterface(output addr[16],output dataout[16],output memreadwr,input PCoutput[16],input ALUA[16],input
ALUB[16],input datain[16],input addrgate,input dataoutgate) {
   mux2x16(addr, PCoutput, ALUB, addrgate);
   mux2x16(dataout, datain, ALUA, dataoutgate);
}
```

Inkrementer je sklop koji uvećava broj za 1. Nešto je jednostavniji i brži od sabirača opštih cijelih brojeva. Zasnovan je na XOR operaciji između pojedinačnog bita i AND operacije između bita koji su na nižoj poziciji od njega.

```
module incrementer(output q[16],input d[16]) {
    bus(c,16);
    buf(c[1],d[0]);
    and(c[2],d[1],d[0]);
    and(c[3],d[2],d[1],d[0]);
    and(c[4],d[3],d[2],d[1],d[0]);
    and(c[5],d[4],d[3],d[2],d[1],d[0]);
    and(c[6],d[5],d[4],d[3],d[2],d[1],d[0]);
    and(c[7],d[6],d[5],d[4],d[3],d[2],d[1],d[0]);
    and(c[8],d[7],d[6],d[5],d[4],d[3],d[2],d[1],d[0]);
    and(c[9],d[8],d[7],d[6],d[5],d[4],d[3],d[2],d[1],d[0]);
    and(c[10],d[9],d[8],d[7],d[6],d[5],d[4],d[3],d[2],d[1],d[0]);
    and(c[11],d[10],d[9],d[8],d[7],d[6],d[5],d[4],d[3],d[2],d[1],d[0]);
    and(c[12],d[11],d[10],d[9],d[8],d[7],d[6],d[5],d[4],
d[3],d[2],d[1],d[0]);
    and(c[13],d[12],d[11],d[10],d[9],d[8],d[7],d[6],d[5],d[4],
d[3],d[2],d[1],d[0]);
    and(c[14],d[13],d[12],d[11],d[10],d[9],d[8],d[7],d[6],d[5],
d[4],d[3],d[2],d[1],d[0]);
    and(c[15],d[14],d[13],d[12],d[11],d[10],d[9],d[8],d[7],d[6],
d[5],d[4],d[3],d[2],d[1],d[0]);
    not(q[0],d[0]);
    xor(q[1],d[1],c[1]);    xor(q[2],d[2],c[2]);
    xor(q[3],d[3],c[3]);    xor(q[4],d[4],c[4]);
    xor(q[5],d[5],c[5]);    xor(q[6],d[6],c[6]);
    xor(q[7],d[7],c[7]);    xor(q[8],d[8],c[8]);
    xor(q[9],d[9],c[9]);    xor(q[10],d[10],c[10]);
    xor(q[11],d[11],c[11]);    xor(q[12],d[12],c[12]);
    xor(q[13],d[13],c[13]);    xor(q[14],d[14],c[14]);
    xor(q[15],d[15],c[15]);}
```

Blok programskog brojača sadrži registar programskog brojača, dva inkrementera koji daju vrijednost PC+1 i PC+2, multiplekser za izbor naredne vrijednosti programskog brojačai množača te vrijednosti sa negiranim

jednobitnim reset signalom. Taj signal započinje izvršenje programa od lokacije 0.

```
module pcblock(output PCoutput[16],output PCplus1[16],input ALUB[16],input ALUOut[16],input dest[4],input src2[4],input
pcgate[2],input clk,input reset) {
   bus(PCinput,16);
   bus(PCplus2,16);
   register16(PCoutput,PCinput,clk);
   incrementer(PCplus1,PCoutput);
   incrementer(PCplus2,PCplus1);
   bus(NewPC,16);
   mux4x16(NewPC, PCplus1, PCplus2, ALUB, ALUOut,pcgate);
   wire(nreset);
   not(nreset,reset);
   mulwith1or0(PCinput,NewPC[0],NewPC[1],NewPC[2],
NewPC[3],NewPC[4],NewPC[5],NewPC[6],NewPC[7],   NewPC[8],NewPC[9],NewPC[10],NewPC[11],NewPC[12],
NewPC[13],NewPC[14],NewPC[15],nreset);
}
```

Konačno, svi blokovi se mogu spojiti u jedinstveni procesor.

```
module cpu(output addr[16],output dataout[16],output memreadwr,input datain[16],input clk,input irq,input reset) {
   bus(opcode,4);
   bus(dest,4);
   bus(src1,4);
   bus(src2,4);
   bus(Regsinput,16);
bus(ALUOut,16);  // Izlaz iz ALU i ulazi
   bus(ALUA,16);
   bus(ALUB,16);
   bus(PCoutput,16);
   bus(PCplus1,16);
   wire(destgate,addrgate,dataoutgate);
   bus(pcgate,2);
  registerblock(ALUA,ALUB,ALUOut,datain,Regsinput,PCplus1,
dest,src1,src2,destgate, clk);
controlunit(opcode,dest,src1,src2,destgate,addrgate,dataoutgate,pcgate,memreadwr,datain,irq,clk);
   pcblock(PCoutput,PCplus1,ALUB,ALUOut,dest,src2, pcgate, clk, reset);
   meminterface(addr,dataout,memreadwr,PCoutput,ALUA,ALUB,
datain,addrgate,dataoutgate);
   alu(ALUOut,ALUA,ALUB,src1,src2,opcode);

}
```

6.7.Primjer programa za procesor SVEU16

Sljedeći primjer računa faktorijel broja 8 podijeljen sa 2 i upisuje ga u memorijsku lokaciju50. Ovdje će biti navedenai stvarni kodovi instrukcija i njihova simbolička predstava.

Adresa	Heks Kod instrukcije	Simbolička instrukcija	EFEKAT
0000	9101	LDC R1,1	R1=1
0001	02FF	LDM R2,PC,(PC)	R2=50
0002	0032	WRD 50	
000	9301	LDC R3,1	R3=1

3			
0004	9508	LDC R5,8	R5=8
0005	9606	LDC R6,6	R6=6
0006	7335	MUL R3,R3,R5	R3=R3*R5
0007	2331	SUB R2,R3,R1	R2=R3-R1
0008	E451	EQU R4,R5,R1	R4=1 ako je R5=R1
0009	1FF4	ADD PC,PC,R4	Preskoči sljedeću naredbu za R4=1
000A	3F66	AND PC,R6,R6	Idi na adresu koju sadrži R6
000B	6331	RSH R3,R3,R1	Pomjeri R3 za R1 mjesta (aritmetički

			udesno, jer su biti 4 i 5 registra R1=00)
000C	8332	STM R3,R3,(R2)	Upiši R3 na MEM[R2]
000D	2FF4	SUB PC,PC,R1	HALT