



Elektrotehnički fakultet
BEOGRAD

MASTER RAD

**SIMULATOR MODELA *PIPELINE* PROCESORA ZA NASTAVU
ARHITEKTURE I ORGANIZACIJE RAČUNARA**

kandidat:
dipl. ing. Stanisavljević Žarko

profesor:
dr Jovan Đorđević

Beograd
27.11.2008.

Sadržaj

1. UVOD.....	5
2. MOTIVACIJA I PREDLOG REŠENJA.....	7
2.1. MOTIVACIJA I CILJEVI RADA	7
2.2. PROCESOR SA PIPELINE ORGANIZACIJOM	8
2.2.1. Opis pipeline procesora	8
2.2.2. Strukturalni hazardi	9
2.2.3. Hazardi podataka.....	9
2.2.4. Upravljački hazardi	12
2.3. PREGLED POSTOJEĆIH SIMULATORA	14
2.3.1. Simulatori fiksnih računarskih sistema.....	18
2.3.2. Simulatori konfigurabilnih računarskih sistema.....	19
2.3.3. Pregled postojećih simulatora pipeline procesora	21
2.4. PREDLOG JEDNOG REŠENJA.....	24
3. ARHITEKTURA I ORGANIZACIJA PIPELINE PROCESORA	26
3.1. ARHITEKTURA	26
3.1.1. Skup programski dostupnih registara	26
3.1.1.1. Registri opšte namene	26
3.1.1.2. Registri posebne namene.....	26
3.1.2. Tipovi podataka	27
3.1.3. Formati instrukcija	27
3.1.4. Načini adresiranja	28
3.1.5. Skup instrukcija.....	28
3.1.5.1. Instrukcije za pristup memoriji	28
3.1.5.2. ARITMETIČKE INSTRUKCIJE	29
3.1.5.3. LOGIČKE INSTRUKCIJE	29
3.1.5.4. INSTRUKCIJE ZA PUNJENJE KONSTANTOM.....	30
3.1.5.5. RELACIONE INSTRUKCIJE	30
3.1.5.6. POMERAČKE INSTRUKCIJE.....	31
3.1.5.7. USLOVNI SKOKOVI	32
3.1.5.8. BEZUSLOVNI SKOKOVI.....	32
3.1.5.9. PROGRAMSKI PREKID	33
3.1.5.10. INSTRUKCIJE ZA PRISTUP POSEBNIM REGISTRIMA	33
3.1.6. KONVENCIJE U VEZI STEKA.....	34
3.1.7. Mehanizam prekida.....	34
3.1.7.1. TIPOVI PREKIDA	34
3.2. ORGANIZACIJA	36

3.2.1. Pipeline posmatranog procesora	36
3.2.1.1. Osnovni elementi idealizovanog pipeline-a	36
3.2.1.2. Aktivnosti u stepenima idealizovanog pipeline-a	37
3.2.1.3. Aktivnosti u realnom pipeline-u	40
4. ALAT IGOVSOEDS.....	42
4.1. KORISNIČKI INTERFEJS ALATA IGOVSoEDS.....	42
4.2. OSNOVNI PROZOR ALATA.....	43
4.2.1. Podloga za prikaz elektronske digitalne strukture	43
4.2.2. Glavni meni i grafički skup poziva funkcija	43
4.2.2.1. Glavni meni.....	43
4.2.2.2. Grafički skup poziva funkcija	44
4.2.3. Biblioteke kola i hijerarhijsko stablo modula	45
4.3. FUNKCIJE GRAFIČKOG EDITORA ELEKTRONSKE DIGITALNE STRUKTURE	46
4.3.1. Skup funkcija.....	46
4.3.1.1. Osnovni skup funkcija	46
4.3.1.2. Dodatni skup funkcija	47
4.3.1.2.1. Meni za rad sa podlogom za prikaz strukture (Canvas Menu)	47
4.3.1.2.2. Meni za rad sa objektima (Object Menu).....	48
4.3.1.2.3. Meni za rad sa signalima (Signal Menu).....	48
4.3.1.2.4. Meni za rad sa grupama objekata (Special Menu)	49
4.3.2. Detaljni pregled izabranih funkcija	49
4.3.2.1. Portovi i simboli modula (Port Manager...).....	49
4.3.2.2. Memorijski moduli (Memory Module)	51
4.3.2.3. Osobine signala i kreiranje Clone/PartClone signala (Properties...).....	52
4.3.2.4. Povezivanje signala i modula.....	54
4.3.2.4.1. Povezivanje signala i modula sa portovima (Port Module).....	54
4.3.2.4.2. Povezivanje signala i modula bez portova (NonPort Module).....	55
5. PROJEKTOVANJE PROCESORA I SIMULATORA	57
5.1. BLOK–ŠEMA PROCESORA	57
5.2. STEPEN IF PIPELINE PROCESORA	57
5.2.1. Jedinica NewPC.....	57
5.2.2. Jedinica PCache	59
5.2.3. Jedinica Instruction Memory	60
5.2.4. Pipeline registar RID.....	61
5.3. STEPEN ID PIPELINE PROCESORA	61
5.3.1. Jedinica DecodID	61
5.3.2. Jedinica Registers	62

5.3.3. Jedinica Extend.....	63
5.3.4. Pipeline registar REX.....	64
5.4. STEPEN EX PIPELINE PROCESORA.....	64
5.4.1. Jedinica EXP1.....	64
5.4.2. Jedinica EXP2.....	66
5.4.3. Jedinica EX1.....	66
5.4.4. Jedinica EX2.....	66
5.4.5. Jedinica ALU.....	66
5.4.5.1. Blok ADD.....	66
5.4.5.2. Blok LOG.....	67
5.4.5.3. Blok CMP.....	68
5.4.5.4. Blok SHIFT.....	68
5.4.5.5. Blok OUT.....	69
5.4.6. Jedinica Zero.....	69
5.4.7. Jedinica DecodeX.....	69
5.4.8. Pipeline registar RMEM.....	71
5.5. STEPEN MEM PIPELINE PROCESORA.....	72
5.5.1. Jedinica MEMP.....	72
5.5.2. Jedinica MEMSD.....	73
5.5.3. Jedinica INTERRUPT.....	73
5.5.4. Jedinica Data Memory.....	75
5.5.5. Pipeline registar RWB.....	76
5.6. STEPEN WB PIPELINE PROCESORA.....	76
5.6.1. Jedinica WBSD.....	76
5.6.2. Pipeline registar RTMP.....	76
5.7. OSTALE JEDINICE.....	76
5.7.1. Jedinica Stall.....	77
6. IZVRŠAVANJE SIMULACIJE.....	79
6.1. FUNKCIONALNOSTI SIMULATORA.....	79
6.1.1. Definisanje događaja za zaustavljanje simulacije.....	79
6.1.2. Rad sa listom signala za pregled promena vremenskih oblika signala.....	81
6.1.3. Upravljanje simulatorom.....	82
6.2. SIMULACIJA: HAZARDI PODATAKA I UPRAVLJAČKI HAZARD.....	86
6.3. SIMULACIJA: HAZARDI PODATAKA I VIŠESTRUKO PROSLJEĐIVANJE.....	89
6.4. SIMULACIJA: UPRAVLJAČKI HAZARDI (INSTRUKCIJE SKOKA).....	92
7. ZAKLJUČAK.....	98
8. LITERATURA.....	101

1. UVOD

Oblast arhitekture i organizacije računara predstavlja jednu od osnovnih oblasti u savremenoj nastavi računarske tehnike i informatike na svim svetskim fakultetima. Na Elektrotehničkom fakultetu u Beogradu postoji grupa predmeta, koja sistematski pokrivaju sve celine od značaja iz ove oblasti. U skladu sa savremenim načinima učenja, već nekoliko godina unazad, na Elektrotehničkom fakultetu u Beogradu, pored predavanja i vežbi na tabli, postoji i praktični deo predmeta, koji se obično izvodi ili kao samostalni projekat ili kao laboratorijske vežbe [1]. Kada govorimo o predmetima iz oblasti arhitekture i organizacije računara, to su obično laboratorijske vežbe na kojima studenti korišćenjem simulatora na praktičnim primerima mogu da provere koncepte, koje su učili na predavanjima. Iako je ovakvo izvođenje nastave u skladu sa IEEE preporukama [2], još jedan stav, koji se iznosi u ovim preporukama je da bi studentima trebalo omogućiti i da sami učestvuju u vežbama na kreativan način, a ne da samo pasivno prate tok vežbe.

Na Elektrotehničkom fakultetu u Beogradu razvijen je veliki broj simulatora sa kojima je u potpunosti pokriveno gradivo, koje se izlaže na predmetima iz oblasti arhitekture i organizacije računara. Sa druge strane, svi ovi simulatori su tzv. simulatori fiksni računarskih sistema, koji studentima ne omogućavaju da upotrebe svoju kreativnost, već ih ostavlja da prate tok izvršavanja simulacije i da proveravaju naučene koncepte. Kako bi se ispoštovala najnovije preporuke IEEE, koje zahtevaju da student na aktivan način koristi simulatore i da mu se omogući da koristi kreativnost, autor je postavio cilj da razvije jedan simulator, koji bi omogućio pomenuto.

Jedna od složenijih struktura koja se izučava iz oblasti arhitekture i organizacije računara je procesor sa *pipeline* organizacijom. Ovakva organizacija procesora je dominantna kod savremenih procesora, jer poboljšava vreme izvršavanja instrukcija, pa samim tim i ukupne performanse procesora. Sama struktura je prilično složena, ali i sa dovoljno modularnih elemenata, koji se mogu realizovati na različite načine, nezavisno u odnosu na druge module i kao celina funkcionisati na isti način. Kod *pipeline* procesora se istovremeno može izvršavati više različitih instrukcija, podelom instrukcija na faze u izvršavanju i preklapanjem različitih faza, tako da je sam osnovni model jednostavan. Iako je ovo jako korisno, ovakva organizacija procesora dovodi do situacija u kojima za neku instrukciju ne može da se izvrši faza predviđena stepenom *pipeline*-a i to se naziva hazard. Postoje strukturalni hazardi, hazardi podataka i upravljački hazardi. Za svaki od ovih problema postoje tehnike za rešavanje, koje omogućavaju različite realizacije rešenja i koje povećavaju složenost osnovnog modela, čineći *pipeline* procesor jednom od najsloženijih struktura.

Nakon izbora odgovarajuće strukture, autor je imao zadatak da izabere i odgovarajući alat u kome je najpogodnije realizovati simulator. Da bi došao do najpogodnijeg alata, autor je napravio pregled postojećih simulatora iz oblasti arhitekture i organizacije računara, sa

posebnim osvrtom na postojeće simulatore *pipeline* procesora. Nakon toga izabran je alat, koji je u fazi razvoja na Elektrotehničkom fakultetu u Beogradu, pod nazivom *IgoVSoEDS* (*Interaktivni generator vizuelnih simulatora elektronskih digitalnih struktura*) [3] [4] [5] [6]. Ovaj alat je već u upotrebi u nastavi na Elektrotehničkom fakultetu u Beogradu na nižim godinama, gde se koristi za pravljenje jednostavnijih struktura. U ovom alatu je napravljen i određeni broj složenijih struktura, kako bi se proverile mogućnosti alata. Autor je dobio zadatak da nakon korišćenja ovog alata da svoju procenu alata, opiše koliko je alat pogodan za razvoj ovakve strukture i da iznese uočene prednosti i nedostatke.

U drugoj glavi daje se motivacija i predlog jednog rešenja. Najpre se daje motivacija za izradu rada, sa nacrtom planiranih ciljeva. Zatim se razmatra procesor sa *pipeline* organizacijom [7] i situacije koje čine *pipeline* procesor složenom strukturom, sa posebnim osvrtom na delove koji mogu biti lako izmenljivi, kao što je keš za predikciju skokova [8]. Daje se i pregled postojećih simulatora iz oblasti arhitekture i organizacije računara [9], kako bi se prikazale mogućnosti tih simulatora, sa posebnim osvrtom na postojeće simulatore *pipeline* procesora [10], da bi se napravio pregled podržanih mogućnosti i radi procene realizovanog simulatora u skladu sa tim. Na kraju se daje predlog rešenja postavljenog problema.

U trećoj glavi je prikazana arhitektura i organizacija *pipeline* procesora, koji je realizovan. Najpre se razmatra arhitektura procesora, a zatim organizacija. Kompletne arhitekture i organizacije preuzeta je iz literature [11] uz saglasnost autora.

U četvrtoj glavi je prikazan alat koji je korišćen za izradu *pipeline* simulatora, kao i način korišćenja alata za projektovanje digitalnih struktura. Opis alata preuzet je iz literature [3] uz saglasnost autora.

U petoj glavi prikazan je način projektovanja procesora, uporedo sa prikazom projektovanja simulatora pomoću korišćenog alata. Ovakav način prikaza projektovanja procesora, daje mogućnost da se na konkretnom primeru projektovanja verifikuje korišćeni alat, kao i sam procesor. Šema je preuzeta iz literature [11] uz saglasnost autora i dopunjena u potrebnoj meri kako bi bilo moguće realizovati simulator u korišćenom alatu. Ovo podrazumeva da su sve komponente realizovane do nivoa prekidačkih mreža.

U šestoj glavi prikazuje se način korišćenja simulatora i funkcionalne mogućnosti simulatora. Najpre je prikazan način korišćenja alata za izvršavanje simulacija. Nakon toga opisan je rad simulatora kroz simulacije koje pokrivaju najveći deo specifičnosti *pipeline* procesora i veliki broj različitih hazarda podataka i upravljačkih hazarda, kao i primene raznovrsnih tehnika za njihovo otklanjanje.

U sedmoj glavi daje se zaključak, kao kritički osvrt na ispunjenje ciljeva postavljenih na početku ovog rada, kao i rezime svega urađenog. Takođe se daje i procena korišćenog alata i iznose se uočene prednosti i nedostaci.

2. MOTIVACIJA I PREDLOG REŠENJA

U ovoj glavi daje se motivacija i predlog jednog rešenja. Najpre se daje motivacija za izradu rada, sa nacrtom planiranih ciljeva. Zatim se razmatra procesor sa *pipeline* organizacijom i situacije koje čine *pipeline* procesor složenom strukturom, sa posebnim osvrtom na delove koji mogu biti lako izmenljivi, kao što je keš za predikciju skokova. Daje se i pregled postojećih simulatora iz oblasti arhitekture i organizacije računara, kako bi se prikazale mogućnosti tih simulatora, sa posebnim osvrtom na postojeće simulatore *pipeline* procesora, da bi se napravio pregled podržanih mogućnosti i radi procene realizovanog simulatora u skladu sa tim. Na kraju se daje predlog rešenja postavljenog problema.

2.1. MOTIVACIJA I CILJEVI RADA

U nastavi iz oblasti arhitekture i organizacije računara na Elektrotehničkom fakultetu u Beogradu već dugi niz godina koriste se simulatori računarskih sistema, koji pomažu studentima da vide kako funkcionišu sistemi, koje uče na predavanjima. Razvijeni su simulatori, koji pokrivaju sve teme koje se obrađuju u nastavi i to su simulatori fiksni računarskih sistema. Ovaj praktičan pristup izvođenju nastave je u skladu sa preporukama IEEE [2], u kojima osim pomenutog, stoji i preporuka da studenti ne bi trebalo da samo pasivno posmatraju šta se događa u simulatorima, već da bi im trebalo omogućiti da upotrebe svoju kreativnost. Kao odgovor na ovakav zahtev, mogu se upotrebiti simulatori konfigurabilnih računarskih sistema.

Da bi kreativnost studenata zaista mogla da dođe do izražaja, potrebno je napraviti neku od složenijih struktura iz oblasti arhitekture i organizacije računara, ali koja ima delova, koji se mogu realizovati na različite načine, a u samoj strukturi menjati jednostavno (*test bed*).

Jedna od veoma složenih struktura iz oblasti arhitekture i organizacije računara, koja odgovara prethodno pomenutom je procesor sa *pipeline* organizacijom. Ovakva organizacija procesora podrazumeva paralelno izvršavanje više instrukcija, pa samim tim i poboljšanje performansi računarskih sistema. U današnjim procesorima, ovakva organizacija se masovno koristi, kako bi se ubrzao rad procesora.

Pipeline organizacija podrazumeva da se izvršavanje operacije može razbiti na više logičkih celina, koje se nazivaju fazama (čitavanje instrukcije, formiranje adresa operanada, čitanje operanada, izvršavanje operacije, itd...). Zatim za svaku od faza postoji posebna kombinaciona mreža i prihvatni registar, koji se koriste za izvršavanje te faze. Na taj način u zavisnosti od broja faza možemo imati veći broj instrukcija, koje se istovremeno izvršavaju u procesoru, tako što se nalaze u različitim fazama izvršavanja. Tipični *pipeline* ima pet faza izvršavanja: IF (*instruction fetch*), ID (*instruction decode and register fetch*), EX (*execute and*

effective address calculation), MEM (*memory access and branch completion*) i WB (*write back*). Po redosledu izvršavanja faza instrukcije mogu se razlikovati statički i dinamički *pipeline*. Ako je redosled izvršavanja faza instrukcije uvek isti, takav *pipeline* naziva se statički (linearni). Ako se redosled stepeni kroz koje instrukcija prolazi može razlikovati od instrukcije do instrukcije, takav *pipeline* naziva se dinamički (nelinearni). U zavisnosti od toga kako se upravlja slanjem informacija iz stepena u stepen *pipeline*-a, mogu se razlikovati asinhroni i sinhroni *pipeline*. U asinhronom *pipeline*-u tok podataka između susednih stepeni kontroliše se *handshaking* protokolom. U sinhronom *pipeline*-u upis u prihvatne registre svih stepeni realizuje se na isti signal takta. U nastavku rada biće razmatran sinhroni statički *pipeline* procesor. Ovakva organizacija procesora dovodi do situacija u kojima za neku instrukciju ne može da se izvrši faza predviđena redosledom izvršavanja, zbog međusavisnosti instrukcija, koje se javljaju uvođenjem ovakve organizacije, i to se naziva hazard. Postoje strukturalni hazardi, hazardi podataka i upravljački hazardi. Za svaki od ovih hazarda postoje tehnike za eliminisanje ili uklanjanje posledica, koje omogućavaju različite realizacije rešenja i koje u mnogome povećavaju složenost, čineći *pipeline* procesor jednom od najsloženijih struktura.

Nakon svega pomenutog, može se postaviti cilj ovog rada, a to je:

- napraviti pregled koncepata procesora *pipeline* organizacije, radi izbora odgovarajuće strukture za pravljenje simulatora,
- napraviti pregled postojećih simulatora računarskih sistema, sa posebnim osvrtom na simulatore *pipeline* procesora, radi izbora alata za rad i radi prikaza karakterističnih mogućnosti postojećih simulatora,
- realizovati simulator *pipeline* procesora u nekom od simulatora konfigurabilnih računarskih sistema, ali tako da se neki delovi mogu lako realizovati na različite načine (*test bed*) i
- dati analizu mogućnosti alata korišćenog za realizaciju.

2.2. PROCESOR SA PIPELINE ORGANIZACIJOM

U ovom odeljku će biti opisani osnovni koncepti *pipeline* procesora [7] [8], zatim hazardi koji nastaju usled paralelnog izvršavanja instrukcija i tehnike koje se koriste za prevazilaženje ovih hazarda.

2.2.1. OPIS PIPELINE PROCESORA

Pipeline je jedna od tehnika realizacije izvršavanja operacija po kojoj se preklapa izvršavanje više operacija. Izvršavanje operacije se može razbiti na više logičkih celina koje će se nazivati fazama (čitanje instrukcije, formiranje adresa operanada, čitanje operanada, izvršavanje operacije, itd...). Za izvršavanje neke faze postoji poseban deo koji će se nazivati

stepen. Stepen se sastoji od kombinacione mreže i prihvatnog registra. Kombinaciona mreža izvršava aritmetičke i logičke operacije. Prihvatni registar sadrži informacije potrebne za izvršavanje faze u datom stepenu i preostalih faza u stepenima koji slede. Istovremeno se može naći k operacija u k različitih faza izvršavanja. Prihvatni registar uz stepen i sadrži informacije potrebne za izvršavanje faze u stepenu i , ali i u svim preostalim stepenima.

Prilikom *pipeline* izvršavanja instrukcija u *pipeline*-u mogu da se stvore takve situacije da za neku instrukciju ne može da se izvrši faza predviđena stepenom *pipeline*-a u kome se ona nalazi. Ove situacije se nazivaju hazardima. Postoje tri vrste hazarda:

- strukturalni hazardi,
- hazardi podataka i
- upravljački hazardi.

U narednim poglavljima biće dat kratak osvrt na ove hazarde i na tehnike, koje se koriste za razrešavanje hazarda kod *pipeline* procesora, jer upravo ove tehnike čine *pipeline* organizaciju procesora jednom od najsloženijih struktura u oblasti arhitekture i organizacije računara.

2.2.2. STRUKTURALNI HAZARDI

Strukturalni hazard u *pipeline* procesoru nastaje kada dve instrukcije koje se nalaze u različitim stepenima *pipeline*-a treba da pristupe istom resursu. Tipičan primer za ovo je sistem u kome se koristi ista memorija i za instrukcije i za podatke. Ovaj hazard bi mogao da se javi u slučaju razmatranog procesora ukoliko ne bi postojale posebne memorije za instrukcije i podatke. Hazard bi se desio kada bi *load* ili *store* instrukcija došla u stepen MEM u kome bi se čitao ili upisivao podatak u memoriju. Tada očitavanje nove instrukcije i ubacivanje u stepen IF ne bi moglo da se realizuje i bilo bi odloženo za jednu periodu signala takta. Sve ostale instrukcije bi se normalno izvršavale. Način za kompletno eliminisanje ovog strukturalnog hazarda je podela memorije na posebne memorije za instrukcije i podatke.

Postoje i drugi primeri strukturalnih hazarda kod *pipeline* procesora, koji se ne mogu rešiti bez primene složenih tehnika za rešavanje. U zavisnosti od toga koliko se često javlja neki strukturalni hazard procenjuje se da li se isplati ubacivanje nekog rešenja kojim se ovaj hazard izbegava ili smanjuje. U nekim situacijama zaustavljanje *pipeline*-a zbog hazarda može da bude prihvatljivo rešenje.

2.2.3. HAZARDI PODATAKA

Hazard podataka se javlja kod procesora sa *pipeline* organizacijom zbog izmenjenog redosleda pristupa podacima u odnosu na redosled pristupa podacima kod procesora bez *pipeline* organizacije. Ovo je posledica činjenice da se kod *pipeline* procesora preklapa izvršavanje različitih faza više instrukcija. Kod procesora koji nemaju *pipeline* organizaciju

instrukcije se izvršavaju sekvencijalno, pa se tek po izvršavanju svih faza jedne instrukcije kreće sa izvršavanjem prve faze sledeće. Kao ilustracija hazarda podataka koji može da nastane kod *pipeline* izvršavanja instrukcija može se uzeti sledeći primer:

```
add  R1, R2, R3
sub  R4, R5, R1
and  R6, R1, R7
```

Obe instrukcije posle instrukcije *add* koriste rezultat instrukcije *add* koji se nalazi u registru R1. Situacija u stepenima *pipeline*-a prilikom izvršavanja tri instrukcije prikazana je na slici 1. Vidi se da instrukcija *add* upisuje podatak u R1 u stepenu WB, dok instrukcija *sub* čita podatak u stepenu ID. Instrukcija *add* završava upis tek posle tri periode signala takta u odnosu na trenutak kad instrukcija *sub* počinje čitanje. Ovakva situacija se naziva hazard podataka. Ukoliko se nešto ne preduzme da se ovakva situacija izbegne, instrukcija *sub* će očitati pogrešnu vrednost i koristiće je. Ova vrednost čak i ne mora da bude uvek postavljena od iste instrukcije. Nekada to može da bude neka od instrukcija pre instrukcije *add*. Ukoliko, pak, stigne prekid između instrukcija *add* i *sub*, a obrada prekida je tako realizovana da se skače na prekidnu rutinu po kompletiranju instrukcije *add*, u R1 će biti vrednost koju je postavila instrukcija *add*.

	1	2	3	4	5	6	7	8	9
add R1, R2, R3	IF	ID	EX	MEM	WB*				
sub R4, R5, R1		IF	ID**	EX	MEM	WB			
and R6, R1, R7			IF	ID**	EX	MEM	WB		

Legenda:

*—**add** upisuje u R1

—sub**, **and** čitaju pogrešnu vrednost iz R1

Slika 1. Situacija u pipeline-u sa prisutnim hazardom podataka

Ovo je samo jedna i to dosta česta vrsta hazarda podataka. Hazarda podataka ima više i oni se mogu svrstati u tri grupe u zavisnosti od toga po kom redosledu bi trebalo čitati i upisivati od strane instrukcija u *pipeline*-u da bi se sačuvalo regularno izvršavanje programa. Posmatraće se dve instrukcije *i* i *j*, pri čemu se instrukcija *i* javlja pre instrukcije *j*. Mogući hazardi podataka su:

- 1) RAW (*read after write*)—instrukcija *j* pokušava da čita podatak pre nego što instrukcija *i* upiše, tako da instrukcija *j* dobija nekorektno staru vrednost. Ovo je najčešći tip hazarda podataka.
- 2) WAR (*write after read*)—instrukcija *j* pokušava da upiše pre nego što instrukcija *i* čita, pa instrukcija *i* nekorektno dobija novu vrednost. U posmatranom *pipeline*-u ovo ne može da se desi zato što se čitanje obavlja u jednom od početnih stepeni *pipeline*-a i to stepenu ID, a upis u jednom od poslednjih stepeni *pipeline*-a i to stepenu WB.

3) WAW (*write after write*)—instrukcija *j* pokušava da upiše neki podatak pre nego što je on najpre upisan instrukcijom *i*. Ovde se upisivanje izvršava po pogrešnom redosledu pa u operandu ostaje vrednost upisana instrukcijom *i* umesto instrukcijom *j*. Ovaj hazard postoji u *pipeline*-ovima u kojima se može upisivati iz više stepeni, što nije slučaj u posmatranom *pipeline*-u u kome se može upisivati samo iz stepena WB.

Treba zapaziti da slučaj RAR (*read after read*) nije hazard.

Postoje tehnike koje se koriste za izbegavanje hazarda podataka i neke od njih će biti opisane u nastavku. Tehnike, koje će biti razmatrane su:

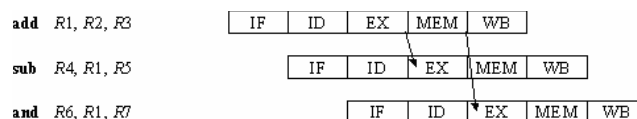
- korektno izvršavanje instrukcija zaustavljanjem *pipeline*-a,
- korektno izvršavanje instrukcija bez zaustavljanja *pipeline*-a prosleđivanjem i
- zakašnjeno punjenje.

Korektno izvršavanje instrukcija zaustavljanjem *pipeline*-a je tehnika, koja podrazumeva da se registruju situacije u kojima bi došlo do hazarda i da se zaustavi učitavanje novih i protok postojećih instrukcija, koje bi mogle dovesti do hazarda u *pipeline*-u, sve dok se ne ispune uslovi da se izvršavanje instrukcija nastavi na korektan način. Najjednostavniji način da se hazard podataka prisutan u primeru sa slike 1 reši je zaustavljanje instrukcije *sub* u *pipeline*-u za tri periode signala takta, da bi se instrukcija *add* normalno izvršavala (slika 2). Time bi se instrukciji *sub* omogućilo da pređe u stepen ID u kome čita podatak iz registra R1 tek pošto instrukcija *add* završi fazu WB u kojoj upisuje podatak u registar R1.

	1	2	3	4	5	6	7	8	9	10
add R1, R2, R3	IF	ID	EX	MEM	WB					
sub R4, R5, R1		IF	<i>stall</i>	<i>stall</i>	<i>stall</i>	ID	EX	MEM	WB	
and R6, R1, R7						IF	ID	EX	MEM	WB

Slika 2. Situacija u *pipeline*-u kao posledica hazarda podataka

Korektno izvršavanje instrukcija bez zaustavljanja *pipeline*-a prosleđivanjem, podrazumeva rešavanje hazarda podataka, prosleđivanjem vrednosti kasnijih stepena *pipeline*-a ranijim, kako bi naredne instrukcije, koje bi mogle da dovedu do hazarda, imale ispravne vrednosti podataka u svim fazama, iako prethodne instrukcije, nisu završile ciklus upisa podataka. Hazard tipa RAW se može hardverski eliminisati tehnikom prosleđivanja (*forwarding, bypassing, short-circuiting*). Rezultat sa izlaza jedinice ALU stepena EX se zajedno sa još nekim informacijama iz *pipeline* registra ID/EX stepena EX na signal takta šalje u *pipeline* registar EX/MEM stepena MEM (slika 3).



Slika 3. Situacija u *pipeline*-u pri korišćenju hardvera za prosleđivanje rezultata jedinice

Situacija može postati još složenija, ukoliko instrukcija koja se izvršava "prva", treba da prosledi svoj rezultat, ne samo sledećoj, već i narednim instrukcijama. U tom slučaju potrebno je prosleđivanje rezultata prve instrukcije iz nekoliko faza u fazu kada je njen rezultat potreban narednim instrukcijama.

Postoje neke situacije RAW hazarda podataka kada je jedini način da se obezbedi korektno izvršavanje instrukcija u *pipeline*-u zaustavljanje *pipeline*-a. Ovakve situacije nastaju, kada je potrebno prosleđivanje između faza, koje se za određene instrukcije izvršavaju u istom ciklusu signala takta i tada ne postoji drugo rešenje osim zaustavljanja.

Postoji i rešenje koje kombinuje prethodne dve tehnike i naziva se zakašnjeno punjenje (*delayed load*). Zakašnjeno punjenje rešava situacije kada je potrebno prosleđivanje između faza, koje bi se izvršavale u istom ciklusu signala takta, tako što zaustavi protok instrukcija kroz *pipeline* za jedan takt, kako bi bilo moguće prosleđivanje.

2.2.4. UPRAVLJAČKI HAZARDI

Upravljački hazardi predstavljaju situacije koje se javljaju u *pipeline* procesorima prilikom izvršavanja instrukcija uslovnog skoka, kada treba zaustaviti *pipeline* određen broj signala takta dok se ne odluči da li će biti skok ili ne i time se dobije vrednost *PC*-ja sa koje treba očitati sledeću instrukciju. Broj perioda signala takta koji se mora sačekati da bi se utvrdila nova vrednost *PC*-ja se naziva *branch delay*.

Postoje tehnike koje se koriste za rešavanje upravljačkih hazarda i neke od njih će biti opisane u nastavku. Tehnike, koje će biti razmatrane su:

- korektno izvršavanje instrukcija zaustavljanjem *pipeline*-a,
- izbor stepena u *pipeline*-u u kome se izvršava skok i
- smanjivanje posledica zbog skokova u *pipeline*-u (statičko i dinamičko predviđanje).

Korektno izvršavanje instrukcija zaustavljanjem *pipeline*-a podrazumeva da se zaustavlja protok instrukcija unutar *pipeline*-a, kao i učitavanje novih instrukcija, sve dok se ne izračuna da li će skoka biti ili ne. U tom slučaju ukoliko ima skoka, učitavaju se instrukcije sa nove adrese (adrese skoka), a u suprotnom se učitavaju instrukcije sa sledeće sekvencijalne adrese i nema hazarda.

Gubitak nekoliko taktova za svaki skok koji se izvrši, kao posledica primene prethodno opisane tehnike, dovodi do značajnog usporavanja *pipeline* procesora i samim tim do gubitka performansi. Broj izgubljenih taktova se može smanjiti ukoliko se proverava da li će biti skoka realizuje u što ranijoj fazi *pipeline*-a. Izborom organizacije procesora može se primeniti ova tehnika i proverava da li će skoka biti se može izvršiti čak i u stepenu ID.

Ima više metoda za smanjivanje posledica koje mogu da nastanu u *pipeline*-u zbog skokova. Tim metodama se pravi predviđanje kako će se program ponašati kod instrukcija

skoka. Na osnovu toga se kao prva instrukcija posle instrukcije skoka očitava ili instrukcija sa adrese skoka ili prva sledeća instrukcija. Metode predviđanja se svrstavaju u statičke i dinamičke. U slučaju statičkih metoda predviđanje za svaki skok je uvek isto za vreme kompletnog izvršavanja programa. U slučaju dinamičkih metoda predviđanje se menja u toku izvršavanja programa, na osnovu prethodnog ponašanja programa. U nastavku se razmatraju četiri statičke metode ponašanja pri skokovima u *pipeline*-u: zaustavljanje, predviđanje nema skoka (*not taken*), predviđanje ima skoka (*taken*) i zakašnjen skok (*delayed branch*). Prve tri metode su hardverske, dok je četvrta metoda softverska.

Zaustavljanje *pipeline*-a je najjednostavnija metoda ponašanja pri skoku čime se očitavanje prve sledeće instrukcije iza instrukcije skoka zaustavlja onoliko perioda signala takta koliko je potrebno da se utvrdi sledeća vrednost *PC*-ja.

Predviđanje nema skoka je metoda ponašanja pri skoku u *pipeline*-u koja podrazumeva da se uvek predvidi da neće biti skoka. Kod ove metode hardver produžava sa očitavanjem i izvršavanjem instrukcija iza instrukcije skoka kao da skoka neće biti, pri čemu se mora voditi računa o tome da instrukcije iza instrukcije skoka ne smeju da menjaju stanje u procesoru sve dok ishod instrukcije skoka nije poznat. U slučaju da nema skoka produžava se sa izvršavanjem očitanih instrukcija, jer se u *pipeline*-u nalaze korektne instrukcije. U slučaju da ima skoka treba zaustaviti *pipeline* i isprati ga (*flush*) od pogrešnih instrukcija. To se postiže restartovanjem *pipeline*-a od koraka IF instrukcije na koju se skočilo.

Predviđanje ima skoka je metoda ponašanja pri skoku u *pipeline*-u koja podrazumeva da se uvek predvidi da će biti skoka. Kod ove metode čim se dekoduje instrukcija skoka i sračuna adresa skoka, predviđa se da će biti skok, pa se počinje sa očitavanjem i izvršavanjem instrukcija od instrukcije na koju se predviđa da će se skočiti. Kada se utvrdi da li je zaista skok napravljen ili ne izvršavanje se produžava dvojako. Ako je skok napravljen u *pipeline*-u su korektne instrukcije, pa treba produžiti sa očitavanjem i izvršavanjem instrukcija. Ako skok nije napravljen, treba očistiti *pipeline* od instrukcija iza instrukcije skoka, jer su pogrešne, i krenuti sa korakom IF prve instrukcije iza instrukcije skoka.

Kod nekih procesora negativni efekti upravljačkog hazarda se ublažavaju ili eliminišu korišćenjem softverske metode zakašnjeni skok (*delayed branch*). Kod ove metode se tokom prevođenja iza instrukcije skoka stavlja onoliko instrukcija koliko bi perioda signala takta trebalo zaustaviti *pipeline* dok se ne donese odluka o tome koja će se sledeća instrukcija iza instrukcije skoka izvršavati. To moraju da budu instrukcije koje treba izvršiti bez obzira na to da li je kao rezultat izvršenja instrukcije skoka skok napravljen ili ne. Ova metoda, slično kao i tehnika zakašnjenog punjenja (*delayed load*), zahteva da u vreme prevođenja prevodilac nađe instrukcije koje može da smesti iza instrukcije skoka.

Dinamičko predviđanje skokova zahteva postojanje posebnog hardvera koji će dinamički predviđati da li će skok biti napravljen ili ne. Za razliku od predhodnih tehnika koje to rade

statički, uvek predviđajući da će skok biti napravljen (*taken*) ili da neće biti napravljen (*not taken*), hardver za dinamičko predviđanje skokova menja svoje predviđanje u toku samog izvršavanja programa.

Najjednostavnija tehnika je tehnika sa baferom predviđanja (*branch prediction buffer*). Ona podrazumeva postojanje male memorije u koju se ulazi na osnovu nekoliko najmlađih bitova adrese instrukcije skoka. Memorija u svakoj lokaciji sadrži samo jedan bit koji se naziva bit predviđanja i koji vrednostima 1 i 0 određuje da li je pri poslednjem izvršavanju instrukcije skoka na toj adresi skok napravljen ili ne. Ako je vrednost 1 očitavanje i izvršavanje instrukcije se produžava sa adrese skoka. U suprotnom slučaju produžava se sekvencijalno. Ako se kasnije u stepenu *pipeline*-a u kome se odlučuje da li treba napraviti skok ili ne utvrdi da je predviđanje bilo pogrešno, sve instrukcije u *pipeline*-u iza instrukcije skoka se ispiraju i kreće se sa očitavanjem korektne instrukcije. Pritom se bit predviđanja u memoriji invertuje.

Ova tehnika predikcije je jednostavna za realizaciju. Ona, međutim, ima nedostatak u petljama gde se skok, sem kada se izlazi iz petlje, stalno pravi. Da bi se ovakve situacije efikasnije rešavale veoma često se koristi šema predviđanja sa 2 umesto sa 1 bitom.

Nedostatak tehnika predviđanja sa baferom predviđanja je da se odlučivanje na osnovu bafera predviđanja može realizovati tek kada se utvrdi da je reč o instrukciji grananja. To je moguće uraditi tek u stepenu *pipeline*-a u kome se dekoduje instrukcija. Ako je stepen dekodovanja instrukcije prvi stepen (ID) iza stepena čitanja instrukcije (IF) onda se sa očitavanjem predviđene instrukcije posle instrukcije skoka uvek kasni jednu periodu signala takta. Da bi se ovo izbeglo potrebno je još dok se instrukcija skoka čita u stepenu IF utvrditi da li se čita instrukcija skoka, napraviti predviđanje da li će se skok realizovati ili ne, i ako je predviđanje da će se skok realizovati utvrditi adresu skoka. Time će biti omogućeno da se na isti takt, kojim se očitana instrukcija skoka prebacuje iz stepena IF u stepen ID u kome se tek vrši njeno dekodovanje, krene u stepenu IF sa očitavanjem instrukcije sa adrese prema predviđenom ishodu skoka. Za realizaciju ovoga se koristi keš za predikciju skokova koji može biti realizovan na veliki broj različitih načina.

2.3. PREGLED POSTOJEĆIH SIMULATORA

U literaturi [9] se može pronaći veliki broj raznovrsnih simulatora iz oblasti arhitekture i organizacije računara, koji su pogodni za nastavu arhitekture i organizacije računara. Osnovne karakteristike odabranih simulatora se nalaze u tabeli 1 i obuhvataju autore, ciljne korisnike, operativne sisteme na kojima se pokreću, programske jezike u kojima su razvijeni, dostupnost i ciljnu arhitekturu računara.

Simulator	Autor	Korisnici	Operativni sistem	Programski jezik	Dostupnost	Ciljna arhitektura računara
ANT	Harvard University, USA	Studenti	Windows, Linux, MacOS	C, Java	Besplatan	MIPS R2000
CASLE	Purdue University, USA	Studenti	Windows, Linux, MacOS	Java	Nepoznato	Proizvoljna
CCSTUDIO	Texas Instruments Inc.	Inženjeri, studenti i testeri	Windows Vista/XP/2000	nepoznato	Komercijalan	ARM, C2000, C5000, C6000, OMAP
CodeWarrior	Freescale Semiconductor, Inc.	Inženjeri, studenti i testeri	Windows, Linux, Solaris, MacOS	nepoznato	Komercijalan	HC(S)08/RS08, 56800/E, 5xx/55xx, 68K, ColdFire, OSEK, Power Architecture, PPC, StarCore, SDMA
CPU Sim	Colby College, USA	Studenti	Windows, Linux, MacOS	Java	Besplatan	RISC
DigLC2	University Paris-Sud, France	Studenti	Windows, UNIX	C	Nepoznato	LC-2
DLXview	Purdue University, USA	Studenti	Solaris, SunOS, HP-UX, and Linux	C	Nepoznato	DLX
Easy CPU	Holon Institute of Technology, Israel	Studenti i učenici srednjih škola	Windows	Java	Nepoznato	Intel 80X86
EDCOMP	University of Belgrade, Serbia	Studenti	Windows, Linux, MacOS	Java	Besplatan	CISC
ESCAPE	Ghent University,	Studenti	Windows	Delphi	Besplatan	von Neumann

	Belgium					
FastCache	University of Wisconsin Madison, USA	Studenti	Solaris 2.5.1	C	Besplatan	
HASE	University of Edinburgh, UK	Studenti, inženjeri, testeri, nastavnici	Solaris, Linux, Windows	Java	Besplatan	
HASE-dinero	University of Edinburgh, UK	Studenti	Solaris, Linux, Windows	HASE++	Besplatan	
ISE Design Suite	Xilinx Inc.	Inženjeri, studenti i testeri i analitičari performansi	Windows, Red Hat Enterprise Linux, SUSE Enterprise Linux	nepoznato	Komercijalan	
JCachesim	University of Siena, Italy	Studenti	Windows, Linux, MacOS	Java	Besplatan	MIPS R3000
JHDL	Brigham Young University, USA	Studenti, inženjeri i testeri	Windows, Linux, MacOS	Java	Nepoznato	
Logisim	Hendrix College, USA	Studenti	Windows, Linux, MacOS	Java	Nepoznato	
M5	The University of Michigan, USA	Inženjeri, studenti i testeri i analitičari performansi	Linux, MacOS X, Solaris, OpenBSD, Cygwin	Python, C++	Nepoznato	Alpha, SPARC, MIPS, ARM ISAs
Quartus II	Altera Corporation	Inženjeri, studenti i testeri i analitičari performansi	Windows, Red Hat Enterprise Linux, SUSE Enterprise Linux, HP-UX	nepoznato	Komercijalan	
RM	University of Catalonia (UPC), Spain	Studenti	Windows, Linux	nepoznato	Besplatan	RISC

RSIM	Rice University Houston Texas, USA	Studenti i istraživači	UNIX	C, C++		Proizvoljna
SIMCA	University of Minnesota, USA	Studenti, inženjeri i tester	SUN SunOS 5.6, IRIX 6.2	C	Besplatan	Proizvoljna
SimFlex	Carnegie Mellon University, USA	Studenti, inženjeri i tester	Linux	C++	Besplatan	
Simics	Virtutech AB Stockholm, Sweden	Studenti, inženjeri i tester	Linux (x86, PowerPC, and Alpha), Solaris/UltraSparc, True64/Alpha, and Windows 2000/x86	C	Komercijalan	PowerPC, x86, ARM, MIPS
SimOS	Stanford University, USA	Studenti, inženjeri i analitičari performansi	Irix, Solaris, Linux	C	Besplatan	MIPS
SimpleScalar	University of Wisconsin- Madison, USA	Studenti, inženjeri i analitičari performansi	Windows, UNIX	C	Besplatan	Alpha, PISA, ARM, x86
SMOK	University of Washington, USA	Studenti	Windows	nepoznato	Besplatan	
Virtual Vulcan	YOERIC Corporation	Studenti	Windows	nepoznato	Komercijalan	

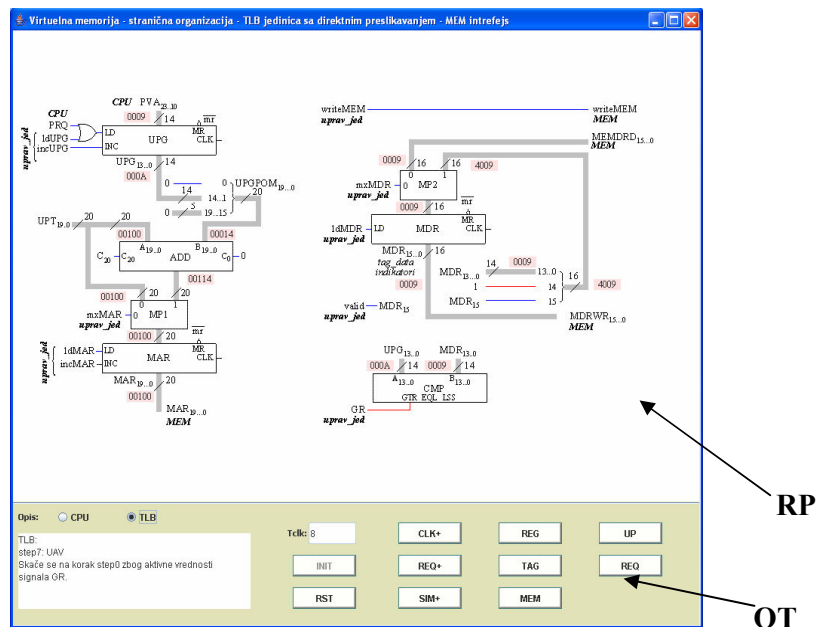
Tabela 1. Karakteristike izabranih simulatora

Navedeni simulatori bi se u opštem slučaju mogli podeliti u dve grupe:

- simulatori fiksnih računarskih sistema,
- simulatori konfigurabilnih računarskih sistema.

2.3.1. SIMULATORI FIKSNIH RAČUNARSKIH SISTEMA

Prva grupa simulatora sadrži odgovarajuće alate, koji omogućavaju korisniku da simulira, već napravljen računarski sistem. Simulator [12] koji je razvio autor (slika 4) je uzet kao primer ove grupe i na njemu će biti prikazane karakteristične mogućnosti ovih simulatora.



Slika 4. Prozor prve grupe simulatora

Prozor simulatora podeljen je u dve grupe: RP - radna površina (gornji deo ekrana) i OT - opcije za testiranje (donji deo ekrana).

Ova vrsta simulatora podrazumeva proceduru, koja se sastoji od pet koraka i u kojoj korisnik:

1. inicijalizuje simulator sa podrazumevanim (*default*) vrednostima,
2. specificira konfiguraciju za testiranje,
3. kreira test vektore i dijagrame, koje treba posmatrati,
4. izvršava simulaciju,
5. analizira rezultate simulacije za izabranu konfiguraciju.

Ova vrsta simulatora obično ima neke konfigurable parametre, koje treba podesiti, pre svakog izvršavanja simulacije. Ovi parametri su obično broj perifernih uređaja, inicijalna vrednost registara i memorijskih lokacija, itd. Ovi parametri omogućavaju da se kreiraju različite konfiguracije za testiranje. Izveštaji testiranja su obično dati tablično u vidu sadržaja registara i memorije, vremenskih dijagrama, tekstualnih fajlova sa različitim logovima i sl.

Najreprezentativniji iz ove grupe simulatora su: ANT [13], CASLE [14], CCSTUDIO [15], CodeWarrior [16], CPU Sim [17], DigLC2 [18], DLXview [19], Easy CPU [20], EDCOMP,

ESCAPE [21], FastCache [22], HASE-Dinero, JCacheSim, RM, RSIM, SIMCA, SimFlex, SimOS i SimpleScalar. ANT je virtualna mašina, bazirana na RISC arhitekturi. CASLE (*Compiler/Architecture Simulation for Learning and Experimenting Simulator*) je simulator na nivou mašinskog jezika. CCSTUDIO (*Code Composer Studio*) je integrisano razvojno okruženje za razvoj DSP koda za MS320 DSP, ARM i OMAP familije procesora. *CodeWarrior Development Studio* sa integrisanim razvojnim okruženjem obezbeđuje alate za kreiranje, kompajliranje, uvezivanje (*linking*), asembliranje i debugovanje 8-bitne i 16-bitne familije mikrokontrolera sa simulatorom sa setom instrukcija (*Instruction Set Simulator*). CPU Sim je paket za simuliranje računarskog sistema, koji omogućuje korisniku da specificira detalje procesora. DigLC2 je simulator na nivou gejta (*gate-level simulator*), koji pruža detaljan opis svih komponenti procesora. DLXview je simulator *pipeline* procesora, koji koristiti DLX set instrukcija za tri verzije DLX *pipeline*-a. Easy CPU je simulator Intel 80X86 procesora sa pojednostavljenim setom instrukcija i sa animiranom reprezentacijom internih računarskih operacija. EDCOMP (*EDucation COMPUter*) je fleksibilni računarski sistem sa simulacijom na nivou logičkih kola. ESCAPE (*Environment for the Simulation of Computer Architectures for the Purpose of Education*) je okruženje koje podržava simulaciju dve proizvoljne arhitekture procesora. FastCache obezbeđuje osnovu (*framework*) za implementaciju simulatora memorijskih sistema. HASE Dinero predstavlja okruženje za simulaciju keš memorije. JCacheSim je simulator za procenu performansi MIPS baziranih računarskih sistema sa keš memorijom. RM (*Rudimentary Machine*) simulira osnovni RISC sistem. RSIM (*Rice Simulator for ILP Multiprocessor*) je simulator, koji radi sa događajima i koji se koristi za analizu multiprocesora sa deljenom memorijom i sistema sa jednim procesorom, koji primenjuju instrukcijski nivo paralelizma. SIMCA (*The Simulator for Multi-threaded Computer Architecture*) podržava procenu performansi arhitektura sa podrškom za rad sa nitima. SimFlex je osnova za simulatore (*simulation framework*) koja koristi dizajn baziran na komponentama. SimOS je simulator kompletnog računarskog sistema dizajniran za efikasnu i preciznu analizu sistema sa jednim i sa više procesora. SimpleScalar je skup alata sa kompajlerom, linkerom, alatom za simulaciju i alatom za vizualizaciju.

2.3.2. SIMULATORI KONFIGURABILNIH RAČUNARSKIH SISTEMA

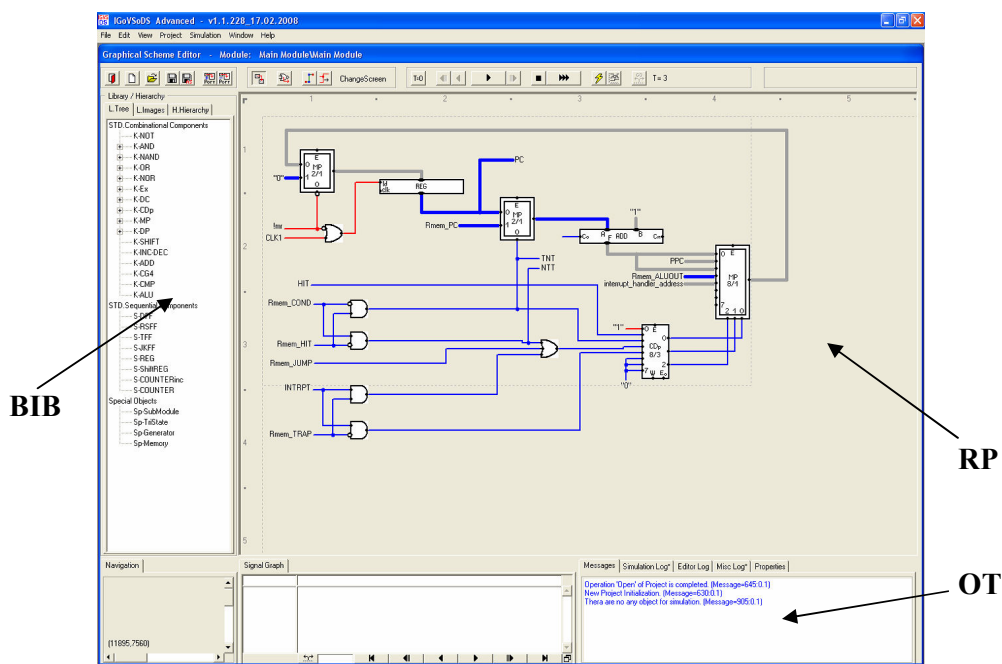
Simulatori fiksnih računarskih sistema sadrže odgovarajuće alate i funkcionalnosti, koje omogućavaju korisniku da najpre napravi specifičnu računarsku konfiguraciju, a zatim da je simulira. Simulator koji je trenutno u fazi razvoja i koji se koristi kao alat u okviru ovog rada (slika 5) biće uzet kao predstavnik ove grupe simulatora, kako bi se prikazale uobičajene mogućnosti, koje ova vrsta simulatora nudi. Glavni prozor simulatora podeljen je na tri oblasti: BIB - dostupne biblioteke (gore levo), RP - radna površina (gore desno) i OT - opcije za testiranje (dole).

Ova vrsta simulatora podrazumeva proceduru od pet koraka, u kojoj korisnik:

1. bira komponente iz biblioteke,
2. postavlja izabrane komponente na radnu površinu,
3. povezuje međusobno odabrane komponente, kako bi napravio nove,
4. pravi interfejs za novu kreiranu komponentu,
5. testira kreiranu komponentu.

Komponenta koja se napravi na ovaj način, pojavice se u BIB delu ekrana i kasnije će moći da se koristi kao gradivna komponenta većeg sistema.

Najkarakterističniji predstavnici ove grupe simulatora su: HASE [23], ISE Design Suite [24], JHDL [25], Logisim [26], M5 [27], Quartus II [28], Simics, SMOK i Virtual Vulcan. HASE (*Hierarchical Computer Architecture design and Simulation Environment*) predstavlja skup alata za pravljenje hijerarhijskih modula, konfigurisanje modula, simuliranje dobijenih sistema i analiziranje skalabilne arhitekture. ISE Design Suite omogućava dizajn uzimajući u obzir vremensku komponentu, pa se koristi kako bi se dobile bolje performanse i manja potrošnja energije, takođe omogućuje dizajn integrisanih sistema (*embedded systems*). JHDL (*Hardware Description Language*) je skup FPGA CAD alata koji omogućavaju korisniku da dizajnira strukturu i izgled kola, da debuguje kolo i da izvrši sintezu. Logisim je edukacioni alat za dizajniranje i simuliranje digitalnih logičkih kola. M5 obezbeđuje opcije neophodne za determinističko simuliranje mrežnih host-ova, uključujući mogućnosti celog sistema, kao i detalje ulazno izlaznog podsistema.



Slika 5. Glavni prozor predstavnika druge grupe simulatora

Quartus II je okruženje za dizajn FPGA, CPLD i struktuiranih ASIC. Simics je platforma za simulaciju celokupnog sistema, koja može da pokrene postojeći firmware i u potpunosti ne modifikovano jezgro i drajvere. SMOK (*SLOOP Machine Organization Kit*) je skup alata koji pruža interfejs za konstrukciju i debugovanje modela mašina. *Virtual Vulcan* je softverski simulator sa podrškom za više od 75 digitalnih kola.

2.3.3. PREGLED POSTOJEĆIH SIMULATORA PIPELINE PROCESORA

U cilju razvoja simulatora izvršena je analiza postojećih simulatora *pipeline* procesora i u nastavku ovog odeljka se prezentuju rezultati te analize. U tabeli 2 dat je pregled implementacionih karakteristika analiziranih simulatora, dok je u tabeli 3 dat pregled karakteristika simulacionog okruženja analiziranih simulatora. Neki delovi analiziranih simulatora nisu bili javno dostupni, pa neke njihove karakteristike nisu mogle biti analizirane, pa je na takvim mestima u tabelu ubačen znak pitanja. U tabeli 2 nema kolone, koja bi predstavljala rešavanje strukturalnih hazarda, jer svi analizirani simulatori koriste slične tehnike za rešavanje tih hazarda.

Termini koji se koriste u tabeli 2 za procenu analiziranih simulatora su "Arhitektura zadovoljava" i "Organizacija zadovoljava". "Arhitektura zadovoljava" daje ocenu da li je arhitektura konzistentna, ortogonalna i jednostavna za implementaciju. "Organizacija zadovoljava" daje ocenu da li je organizacija razumljiva, jednostavna i uniformna.

Većina analiziranih simulatora je bazirana ili na DLX ili na MIPS arhitekturama. Kompletan DLX set instrukcija (kakav se simulira u DLXview i WinDLX) implicira završetak u pogrešnom redosledu (*out-of-order completion*), upis nakon upisa i još neke strukturalne hazarde. Ove karakteristike nisu neophodne za razumevanje osnovnih principa *pipeline* procesora, a osim toga značajno komplikuju implementaciju i samim tim smanjuju jasnoću koncepata. Dodatno, DLX arhitektura nije u potpunosti konzistentna zbog toga što specifikacija operacija i operanada nisu na istom mestu u različitim formatima instrukcija. Isti problem postoji u simulatorima koji implementiraju kompletnu MIPS arhitekturu (MipsIt [29], WinMIPS64 i SPIM) ili podskup DLX seta instrukcija sa instrukcijama sa pokretnim zarezom (*floating-point*) i celobrojnim *mult* i *div* instrukcijama (JavaHASE i Superscalar DLX).

Većina simulatora kod kojih su detalji organizacije procesora sakriveni imaju problema sa netačnom organizacijom, koja funkcioniše manje ili više različito u odnosu na opisani hardver. Na primer, kada se izvršavaju test programi, koji zahtevaju višestruka prosleđivanja ESCAPE i WinMIPS64 veštački proizvode korektne rezultate koristeći tehnike, koje bi zahtevale hardver, koji ne postoji u stvarnom procesoru. ESCAPE prikazuje bit koji predstavlja uslov skoka, kao deo MEM faze izvršavanja, iako skokove izvodi u fazi EX. SPIM ne simulira izvršavanje sa protočnom obradom, jer se sve instrukcije završavaju u jednom ciklusu takta. U JavaHASE simulatoru, hazardi podataka se mogu dogoditi prilikom

upisivanja u registar R0. WinDLX izračunava adresu skoka tako što prepisuje niži deo niže reči sa pomerajem, umesto da vrši sabiranje. U WebMIPS [30] i MipsIt model 2 simulatorima, prikaz njihove organizacije je suviše komplikovan. WinDLX, WinMIPS64 i MIPS simulatori izvode neobičajene operacije zaustavljanja, kao i ispiranja. U MIPSim i ESCAPE simulatorima, *mult* i *div* instrukcije se završavaju u jednom ciklusu takta u EX fazi izvršavanja.

Simulator	Arhitektura zadovoljava	Organizacija zadovoljava	Rešavanje hazarda podataka	Rešavanje upravljačkih hazarda	Obrada prekida
DARC2	delimično	?	prosleđivanje ili zaustavljanje	predikcija skoka ili zaustavljanje	ne
DLXide	delimično	?	prosleđivanje ili zaustavljanje	predikcija skoka, zakašnjeni skok ili zaustavljanje	ne
DLXview	ne	delimično	prosleđivanje	zakašnjeni skok	ne
ESCAPE	delimično	ne	prosleđivanje ili zaustavljanje	predikcija skoka	ne
MIPSim	da	delimično	nema	predikcija skoka	ne
MipsIt	ne	?	prosleđivanje	zakašnjeni skok	da
RaVi	da	da	prosleđivanje	zakašnjeni skok	ne
WebMIPS	delimično	da	Fwd	zakašnjeni skok	ne
JavaHASE	ne	ne	zaustavljanje	predikcija skoka	ne
MIPS	delimično	ne	prosleđivanje i zaustavljanje	predikcija skoka	ne
mmmix	ne	ne	?	predikcija skoka	?
SuperSim	da	ne	?	predikcija skoka	ne
WinDLX	ne	delimično	prosleđivanje ili zaustavljanje	predikcija skoka	ne
WinMIPS64	ne	ne	prosleđivanje ili zaustavljanje	predikcija skoka ili zakašnjeni skok	ne
DLXsim	delimično	?	prosleđivanje	zakašnjeni skok	ne
DLXVsim	ne	?	prosleđivanje	zakašnjeni skok	ne
J-MIPS	delimično	?	?	?	ne
SPIM	ne	ne	?	?	da
SSVisual601	ne	delimično	prosleđivanje i zaustavljanje	predikcija skoka	ne
HDLDLX	delimično	delimično	?	?	?
Superscalar DLX	ne	ne	prosleđivanje i zaustavljanje	?	da

Tabela 2. Pipeline implementacione karakteristike analiziranih simulatora

Samo dva od nabrojanih simulatora (SuperSim and Superscalar DLX) podrazumevaju

dinamičku predikciju skoka, ali obadva su simulatori superskalarnih procesora. Potpuna obrada prekida je podržana samo u MipsIt, SPIM i Superscalar DLX, dok u HDL DLX korisnici mogu sami da definišu mehanizam za obradu prekida. Opisi obrade prekida kod postojećih simulatora *pipeline* procesora su suviše kratki i ne obuhvataju sve vrste prekida koje se javljaju u pravim procesorima.

Simulator	Vizuelna prezentacija	RTL detalji	Tok instrukcija jasan	Specifični događaji jasni	Dobra kontrola simulacije	Prikazane vrednosti	Web baziran
DARC2	da	ne	ne	?	delimično	reg. i memorije	ne
DLXide	da	ne	?	?	ne	da	ne
DLXview	da	ne	delimično	da	da	mogu se pronaći	ne
ESCAPE	da	ne	delimično	ne	da	da	ne
LSE	da	ne	ne	ne	?	ne	ne
MIPSim	da	ne	da	?	ne	da	ne
MipsIt	da	ne	delimično	ne	ne	da	ne
RaVi	da	ne	delimično	ne	ne	da	da
WebMIPS	da	ne	ne	delimično	ne	da	da
JavaHASE	blokovi	ne	ne	delimično	ne	reg. i memorije	da
MIPS	blokovi	ne	da	delimično	ne	da	da
mmmix	blokovi	ne	delimično	ne	ne	ne	?
SuperSim	blokovi	ne	delimično	ne	ne	registara i memorije	ne
WinDLX	blokovi	ne	da	delimično	ne	da	ne
WinMIPS64	blokovi	ne	ne	ne	ne	ne	ne
DLXsim	ne	ne	ne	ne	ne	mogu se pronaći	ne
DLXVsim	ne	ne	ne	ne	ne	mogu se pronaći	ne
J-MIPS	ne	ne	ne	ne	ne	registara i memorije	da
SPIM	ne	ne	ne	ne	ne	registara i memorije	ne
SSVisual601	ne	ne	ne	ne	da	registara	ne
HDL DLX	da	većinom	delimično	delimično	da	da	ne
Superscalar DLX	da	većinom	delimično	delimično	da	da	ne

Tabela 3. Karakteristike simulacionog okruženja analiziranih simulatora

Neki od simulatora prikazuju samo komponente procesora na visokom nivou kao što su: ALU, registarski fajl, registri. Takvi su: DARC2, DLXide, DLXview, ESCAPE, LSE, MIPSim, MipsIt, RaVi i WebMIPS. Drugi prikazuju samo blokove korespondentne fazama u izvršavanju *pipeline* procesora i možda neke od važnijih jedinica (JavaHASE, MIPS Simulator, mmmix, SuperSim, WinDLX i WinMIPS64). Neki simulatori nemaju grafički prikaz, već su tekstualni (DLXsim [31], DLXVsim, J-MIPS, SPIM i SSVisual601).

Samo dva od nabrojanih simulatora prikazuju komponente do nivoa prekidačkih mreža (*RTL*) i to su: HDLXL i Superscalar DLX. Obadva su VHDL modeli DLX baziranih procesora. VHDL omogućava ne samo vizuelni prikaz organizacije do nivoa prekidačkih mreža, već i eksperimentisanje sa organizacijom procesora, kako na nivou prekidačkih mreža, tako i na višim nivoima. Mana ovakvog pristupa je ta što je VHDL komercijalni alat, pa nije pogodan za učenje.

Samo nekolicina simulatora prikazuje tok instrukcija na jednostavan i razumljiv način (MIPSim, MIPS Simulator i WinDLX). Među njima, samo DLXview omogućava da se da naglasak važnijim događajima. Mogućnost povratka simulacije na prethodni ciklus takta je implementirana samo u DLXview, ESCAPE i donekle u SSVisual601 i HDLXL. Manje od polovine analiziranih simulatora može da prikazuje vrednosti u *pipeline*-u.

Neki od simulatora (HASE [32], LSE i RaVi) su simulatori hardvera opšte namene. LSE i RaVi funkcionalni modeli veoma dobro odgovaraju stvarnim procesorima i stoga su pogodni za eksperimentisanje na visokom nivou.

Samo pet analiziranih simulatora su *Web* bazirani (RaVi, WebMIPS, JavaHASE, MIPS, and J-MIPS).

2.4. PREDLOG JEDNOG REŠENJA

Nakon svega navedenog u prethodnim poglavljima, sada je moguće na osnovu definisanih ciljeva sa početka ove glave, izneti predlog jednog rešenja problema.

Kao što smo videli iz analize postojećih simulatora, nijedan od pobrojanih simulatora ne odgovara u potpunosti zahtevima ovoga rada. Iako neki od alata imaju predispozicije, takvi alati su ili komercijalni ili nisu dostupni. Zbog toga alat, koji će biti korišćen u nastavku ovog rada je alat koji je razvijen na Elektrotehničkom fakultetu u Beogradu (*IGoVSoEDS*) i koji je već u upotrebi u laboratoriji na nižim godinama, gde se koristi za razvoj jednostavnijih struktura.

Potrebno je napomenuti i da postoji simulator *pipeline* procesora, koji u potpunosti zadovoljava potrebe za shvatanjem principa *pipeline* procesora (*PipeSim*), koji se koristi na Elektrotehničkom fakultetu u Beogradu. Pošto je cilj autora, da pored shvatanja koncepata, omogući studentima i da upotrebe svoju kreativnost, u pravljenju ovog simulatora, biće iskorišćena već postojeća arhitektura i organizacija, uz saglasnost autora.

Predlog jednog rešenja podrazumeva da će autor uraditi sledeće:

- realizovati simulator *pipeline* procesora u alatu *IGoVSoEDS*, ali tako da se neki delovi mogu lako realizovati na različite načine (*test bed*),
- napraviti pregled tehnika *pipeline* organizacije procesora,
- napraviti pregled postojećih simulatora računarskih sistema, sa posebnim osvrtom na simulatore *pipeline* procesora i
- dati analizu mogućnosti alata korišćenog za realizaciju (*IGoVSoEDS*).

3. ARHITEKTURA I ORGANIZACIJA *PIPELINE* PROCESORA

U ovoj glavi je prikazana arhitektura i organizacija *pipeline* procesora, koji je realizovan. Najpre se razmatra arhitektura procesora, a zatim organizacija. Samom organizacijom implicirana je RISC arhitektura, sa svim prednostima i nedostacima.

3.1. ARHITEKTURA

U ovom poglavlju se razmatraju elementi arhitekture procesora i to skup programski dostupnih registara, tipovi podataka, formati instrukcija, načini adresiranja, skup instrukcija i mehanizam prekida.

3.1.1. SKUP PROGRAMSKI DOSTUPNIH REGISTARA

Skup programski dostupnih registara čine registri opšte namene i registri posebne namene.

3.1.1.1. REGISTRI OPŠTE NAMENE

Posmatrana arhitektura sadrži 32 registra opšte namene, veličine po 32 bita. Ti registri označeni su od R0 do R31. Biti tih registara označeni su od 31 do 0, pri čemu je 31 bit najveće težine (MSB – *Most Significant Bit*).

Registri R0 i R31 imaju posebne uloge. Vrednost registra R0 je uvek 0 i ne može biti promenjena. R0 može biti određišni operand instrukcija, ali takve instrukcije zapravo ne mogu da izvrše upis u R0. To omogućava jednostavnije ostvarivanje nekih operacija i načina adresiranja.

U registar R31 implicitno se upisuje adresa povratka iz prekida prilikom pokretanja opsluživanja prekida. Adresa tog registra fiksirana je u mehanizmu prekida. Zato treba izbegavati upotrebu tog registra u druge svrhe.

Registri R29 i R30 takođe imaju posebne uloge, ali za razliku od R0 i R31 njihove uloge nisu fiksirane već su stvar dogovora u pogledu pisanja mašinskih programa. U R30 se upisuje adresa povratka prilikom poziva potprograma. R29 se koristi kao stek pointer. Radi međusobne kompatibilnosti programa, treba izbegavati upotrebu tih registara u druge svrhe.

3.1.1.2. REGISTRI POSEBNE NAMENE

Registri posebne namene su programski brojač PC i programska statusna reč PSW.

Programski brojač PC (Program Counter) je 32-bitni registar u kome se nalazi adresa instrukcije. U svakom taktu vrednost ovog registra se menja na jedan od sledećih načina: 1) PC se inkrementira, ili 2) u PC se upisuje adresa zadata instrukcijom skoka ili adresa

prekidne rutine. Dakle, PC je implicitni odredišni argument za instrukcije uslovnih i bezuslovnih skokova, a menja ga i mehanizam prekida.

Programska statusna reč PSW (*Program Status Word*) je 10-bitni registar koji služi za prihvatanje spoljašnjih zahteva za prekid, maskiranje (zabranu pokretanja opsluživanja) prekida, režim prekid posle svake instrukcije (*trap mode*) i čuvanje hardverski postavljenih indikatora koji se koriste u prekidnoj rutini. Biti registra PSW, od najstarijeg do najmlađeg, su:

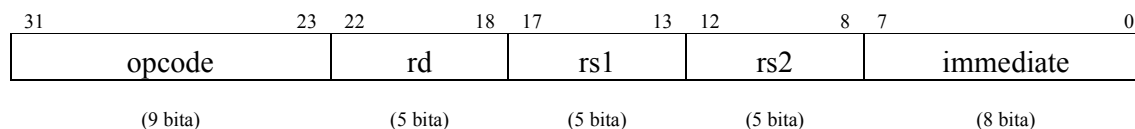
- GM (9) – globalna maska za dozvolu ili zabranu svih prekida
- INI (8) – indikator inicijalizacije procesora
- TRAP (7) – indikator prekida izazvanog programski, instrukcijom *trap*
- UOE (6) – indikator prekida izazvanog pogrešnim kodom operacije
- ARE (5) – indikator prekida izazvanog prekoračenjem pri aritmetičkoj operaciji
- NMI (4) – indikator zahteva za nemaskirajući prekid
- MI (3) – indikator zahteva za maskirajući prekid
- MASK (2) – maska za dozvolu ili zabranu maskirajućih prekida
- TRM (1) – indikator režima prekid posle svake instrukcije
- INC (0) – indikator da prekidna rutina treba da inkrementira adresu povratka

3.1.2. TIPOVI PODATAKA

U posmatranom procesoru postoje dva tipa podataka: celobrojne veličine sa i bez znaka, dužine 32 bita. Podaci kraći od 32 bita (neposredni argumenti instrukcija, posebni registri kraći od 32 bita, rezultati relacionih operacija) prilikom upisa u 32-bitna polja ili registre smeštaju se u bite najmanje težine i proširuju do 32 bita znakom ili nulom, u zavisnosti od operacije.

3.1.3. FORMATI INSTRUKCIJA

Sve instrukcije su fiksne dužine 32 bita. Postoji samo jedan format instrukcija, prikazan na slici 6.



Slika 6. Format instrukcija

Polje *opcode* određuje kod operacije, *rd* je oznaka odredišnog registra opšte namene, *rs1* i *rs2* su oznake izvorišnih registara opšte namene, a polje *immediate* sadrži neposredni argument. Pošto postoje 32 registra opšte namene, za njihovo adresiranje potrebno je 5 bita. Zato su polja *rd*, *rs1* i *rs2* veličine po 5 bita, a preostalih 17 bita raspodeljeno je na polja *opcode* i *immediate*. Neke instrukcije koriste argumente određene poljima *rd*, *rs1* i *rs2*, neke koriste *rd*, *rs1* i *immediate*, instrukcija *store* koristi *rs1*, *rs2* i *immediate*, a neke instrukcije

imaju manje od 3 argumenta. Dakle, nijedna instrukcija ne koristi sva 4 argumenta, ali pošto se u svim instrukcijama isti argument nalazi u istim bitima, pojednostavljena je implementacija ove arhitekture. Redosled argumenata je usklađen sa uobičajenom notacijom u mašinskom jeziku po kojoj se odredišni argument prvi navodi.

3.1.4. NAČINI ADRESIRANJA

S obzirom da su instrukcije posmatranog procesora 32-bitne i da su oba tipa podataka 32-bitni, adresiranje je na nivou 32-bitnih reči. Adresa je dužine 32 bita. Programski brojač inkrementira se za 1.

Memorijskim lokacijama pristupa se jedino pomoću instrukcija *load* i *store*. Adresa memorijske lokacije dobija se sabiranjem sadržaja registra opšte namene (određenog poljem *rs1*) i pomeraja (određenog neposrednim argumentom *immediate*). Dakle, instrukcije *load* i *store* koriste registarsko indirektno adresiranje sa pomerajem. Ako je vrednost registra nula (a za to se koristi R0), adresiranje se pretvara u memorijsko direktno. Ako je vrednost pomeraja nula, dobija se registarsko indirektno adresiranje.

Najveći broj operacija obavlja se nad sadržajem dva registra, ili jednog registra i neposrednim podatkom, i rezultat se upisuje u registar. Dakle, koriste se registarsko direktno adresiranje i neposredno adresiranje. Najmlađi bit u polju *opcode* određuje da li je drugi izvorišni operand registar ili neposredni podatak, a ostatak polja *opcode* specificira samu operaciju.

3.1.5. SKUP INSTRUKCIJA

U ovom odeljku se daju opisi instrukcija.

3.1.5.1. INSTRUKCIJE ZA PRISTUP MEMORIJI

Instrukcije za pristup memoriji su:

lw	<i>rd, rs1, imm</i>	load word	$\text{Regs}[rd] \leftarrow \text{Mem}[\text{Regs}[rs1] + ((imm_7)^{24} \parallel imm)]$
sw	<i>rs1, rs2, imm</i>	store word	$\text{Mem}[\text{Regs}[rs1] + ((imm_7)^{24} \parallel imm)] \leftarrow \text{Regs}[rs2]$

Instrukcija *lw* (*load word*) očitava sadržaj zadate memorijske lokacije i upisuje ga u registar opšte namene određen poljem *rd*. Adresa memorijske lokacije dobija se sabiranjem sadržaja registra opšte namene određenog poljem *rs1* i neposrednog argumenta. Neposredni argument se pre sabiranja proširuje znakom. Rezultat sabiranja se tretira kao ceo broj bez znaka, a eventualno prekoračenje (*overflow*) se ignoriše.

Instrukcija *sw* (*store word*) upisuje sadržaj registra opšte namene određenog poljem *rs1* u zadatu memorijsku lokaciju. Adresa memorijske lokacije dobija se na isti način kao u slučaju instrukcije *lw*.

3.1.5.2. ARITMETIČKE INSTRUKCIJE

Aritmetičke instrukcije su:

add	<i>rd, rs1, rs2</i>	add	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] + \text{Regs}[rs2]$
addi	<i>rd, rs1, imm</i>	add immediate	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] + ((imm_7)^{24} \parallel imm)$
addu	<i>rd, rs1, rs2</i>	add unsigned	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] + \text{Regs}[rs2]$
addui	<i>rd, rs1, imm</i>	add unsigned immediate	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] + (0^{24} \parallel imm)$
sub	<i>rd, rs1, rs2</i>	subtract	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] - \text{Regs}[rs2]$
subi	<i>rd, rs1, imm</i>	subtract immediate	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] - ((imm_7)^{24} \parallel imm)$
subu	<i>rd, rs1, rs2</i>	subtract unsigned	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] - \text{Regs}[rs2]$
subui	<i>rd, rs1, imm</i>	subtract unsigned immed.	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] - (0^{24} \parallel imm)$

Instrukcija *add* vrši sabiranje sadržaja registara opšte namene određenih poljima *rs1* i *rs2* i upisuje rezultat u registar opšte namene određen poljem *rd*. Rezultat se tretira kao ceo broj sa znakom, predstavljen u komplementu dvojke. Pri sabiranju može nastati prekoračenje, ako rezultat nije u opsegu $[-2^{31}..2^{31}-1]$.

Instrukcija *addi* (*add immediate*) vrši sabiranje sadržaja registra opšte namene određenog poljem *rs1* i neposrednog argumenta i upisuje rezultat u registar opšte namene određen poljem *rd*. Neposredni argument se pre sabiranja proširuje znakom. Način izvršavanja operacije je isti kao kod instrukcije *add*.

Instrukcija *addu* (*add unsigned*) vrši sabiranje sadržaja registara opšte namene određenih poljima *rs1* i *rs2* i upisuje rezultat u registar opšte namene određen poljem *rd*. Rezultat se tretira kao ceo broj bez znaka. Pri sabiranju može nastati prekoračenje, ako rezultat nije u opsegu $[0..2^{32}-1]$. Jedina razlika u načinu izvršavanja instrukcija *add* i *addu* je u načinu prepoznavanja prekoračenja.

Instrukcija *addui* (*add unsigned immediate*) vrši sabiranje sadržaja registra opšte namene određenog poljem *rs1* i neposrednog argumenta i upisuje rezultat u registar opšte namene određen poljem *rd*. Neposredni argument se pre sabiranja proširuje nulom. Način izvršavanja operacije je isti kao kod instrukcije *addu*.

Instrukcije *sub* (*subtract*), *subi* (*subtract immediate*), *subu* (*subtract unsigned*) i *subui* (*subtract unsigned immediate*) su analogne instrukcijama *add*, *addi*, *addu* i *addui*, respektivno, po načinu tretiranja operanada, rezultata i prepoznavanju prekoračenja; razlikuju se samo po tome što se vrši operacija oduzimanja a ne sabiranja.

3.1.5.3. LOGIČKE INSTRUKCIJE

Logičke instrukcije su:

and	<i>rd, rs1, rs2</i>	and	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] \& \text{Regs}[rs2]$
andi	<i>rd, rs1, imm</i>	and immediate	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] \& (0^{24} \parallel imm)$

or	<i>rd, rs1, rs2</i>	or	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] \mid \text{Regs}[rs2]$
ori	<i>rd, rs1, imm</i>	or immediate	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] \mid (0^{24} \parallel imm)$
xor	<i>rd, rs1, rs2</i>	exclusive or	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] \oplus \text{Regs}[rs2]$
xori	<i>rd, rs1, imm</i>	exclusive or immediate	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] \oplus (0^{24} \parallel imm)$

Instrukcija *and* vrši logičku operaciju "i" na nivou bita (*bitwise logical and*) nad sadržajima registara opšte namene određenih poljima *rs1* i *rs2* i upisuje rezultat u registar opšte namene određen poljem *rd*.

Instrukcija *andi* vrši logičku operaciju "i" na nivou bita (*bitwise logical and*) nad vrednostima registra opšte namene određenog poljem *rs1* i neposrednog argumenta i upisuje rezultat u registar opšte namene određen poljem *rd*. Neposredni argument se pre logičke operacije proširuje nulom.

Instrukcije *or* i *xor* (*exclusive or*) s jedne, i *ori* (*or immediate*) i *xori* (*exclusive or immediate*) s druge strane, analogne su instrukcijama *and* i *andi*, respektivno, po načinu tretiranja operanada i rezultata; razlikuju se samo po tome što se vrše logičke operacije "ili" odn. "ekskluzivno ili" na nivou bita. Zato nisu posebno opisane.

3.1.5.4. INSTRUKCIJE ZA PUNJENJE KONSTANTOM

Instrukcije za punjenje konstantom su:

li1	<i>rd, rs1, imm</i>	load immediate to byte 1	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] \mid (0^{16} \parallel imm \parallel 0^8)$
li2	<i>rd, rs1, imm</i>	load immediate to byte 2	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] \mid (0^8 \parallel imm \parallel 0^{16})$
li3	<i>rd, rs1, imm</i>	load immediate to byte 3	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1] \mid (imm \parallel 0^{24})$

Instrukcija *li1* proširuje neposredni argument nulom sa desne strane za 8 bita i sa leve strane za 16 bita. Zatim vrši logičku operaciju "ili" na nivou bita (*bitwise logical or*) nad vrednostima registra opšte namene određenog poljem *rs1* i tako proširenog neposrednog argumenta i upisuje rezultat u registar opšte namene određen poljem *rd*. (Pod pretpostavkom da je bajt 1 izvorišnog registra opšte namene imao vrednost 0, kao rezultat instrukcije *li1* polazni neposredni argument upisuje se u bajt 1 odredišta.)

Instrukcije *li2* i *li3* analogne su instrukciji *li1* i dovode do upisivanja polaznog neposrednog argumenta u bajt 2 i 3 odredišta, respektivno. Nema potrebe za instrukcijom *li0*, jer bi bila identična instrukciji *ori* (*or immediate*). Preporučuje se da vrednost izvorišnog registra prve od instrukcija za punjenje konstantom bude 0, za šta se može koristiti registar R0.

3.1.5.5. RELACIONE INSTRUKCIJE

Relacione instrukcije su:

slt	<i>rd, rs1, rs2</i>	set if less than	if $\text{Regs}[rs1] < \text{Regs}[rs2]$ then $\text{Regs}[rd] \leftarrow (0^{31} \parallel 1)$
------------	---------------------	------------------	--

			$\text{else Regs}[rd] \leftarrow 0^{32}$
slti	$rd, rs1, imm$	set if less than immediate	if $\text{Regs}[rs1] < ((imm_7)^{24} \parallel imm)$ then ...
sle	$rd, rs1, rs2$	set if less or equal	if $\text{Regs}[rs1] \leq \text{Regs}[rs2]$ then ...
slei	$rd, rs1, imm$	set if less or equal immedi.	if $\text{Regs}[rs1] \leq ((imm_7)^{24} \parallel imm)$ then ...
seq	$rd, rs1, rs2$	set if equal to	if $\text{Regs}[rs1] = \text{Regs}[rs2]$ then ...
seqi	$rd, rs1, imm$	set if equal to immediate	if $\text{Regs}[rs1] = ((imm_7)^{24} \parallel imm)$ then ...
sne	$rd, rs1, rs2$	set if not equal to	if $\text{Regs}[rs1] \neq \text{Regs}[rs2]$ then ...
snei	$rd, rs1, imm$	set if not equal to immedi.	if $\text{Regs}[rs1] \neq ((imm_7)^{24} \parallel imm)$ then ...
sgt	$rd, rs1, rs2$	set if greater than	if $\text{Regs}[rs1] > \text{Regs}[rs2]$ then ...
sgti	$rd, rs1, imm$	set if greater than immedi.	if $\text{Regs}[rs1] > ((imm_7)^{24} \parallel imm)$ then ...
sge	$rd, rs1, rs2$	set if greater or equal	if $\text{Regs}[rs1] \geq \text{Regs}[rs2]$ then ...
sgei	$rd, rs1, imm$	set if greater or eq. imm.	if $\text{Regs}[rs1] \geq ((imm_7)^{24} \parallel imm)$ then ...

Instrukcije *slt* (*set if less than*), *sle* (*set if less then or equal to*), *seq* (*set if equal to*), *sne* (*set if not equal to*), *sgt* (*set if greater than*) i *sge* (*set if greater than or equal to*) vrše poređenje sadržaja registara opšte namene određenih poljima *rs1* i *rs2*. Odredišni registar opšte namene određen je poljem *rd*. Ako je zadovoljen uslov poređenja, u najmlađi bit odredišnog registra upisuje se 1, a ako nije, 0. U ostale bite odredišnog registra upisuje se 0. Operandi se tretiraju kao celi brojevi sa znakom, predstavljeni u komplementu dvojke.

Instrukcije *slti* (*set if less than immediate*), *slei* (*set if less than or equal to immediate*), *seqi* (*set if equal to immediate*), *snei* (*set if not equal to immediate*), *sgti* (*set if greater than immediate*) i *sgei* (*set if greater than or equal to immediate*) vrše poređenje sadržaja registra opšte namene određenog poljem *rs1* i neposrednog argumenta. Neposredni argument se pre poređenja proširuje znakom. Način tretiranja operandi i rezultati su isti kao kod instrukcija *slt*, *sle*, *seq*, *sne*, *sgt* i *sge*, respektivno.

3.1.5.6. POMERAČKE INSTRUKCIJE

Pomeračke isstrukcije su:

shl	$rd, rs1$	shift left	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1]_{30..0} \parallel 0$
shrl	$rd, rs1$	shift right logical	$\text{Regs}[rd] \leftarrow 0 \parallel \text{Regs}[rs1]_{31..1}$
shra	$rd, rs1$	shift right arithmetic	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1]_{31} \parallel \text{Regs}[rs1]_{31..1}$
rotl	$rd, rs1$	rotate left	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1]_{30..0} \parallel \text{Regs}[rs1]_{31}$
rotr	$rd, rs1$	rotate right	$\text{Regs}[rd] \leftarrow \text{Regs}[rs1]_0 \parallel \text{Regs}[rs1]_{31..1}$

Instrukcija *shl* (*shift left*) vrši pomeranje sadržaja registra opšte namene određenog poljem *rs1* za jedno mesto ulevo, pri čemu se najmlađi bit rezultata postavlja na 0. Rezultat se upisuje u registar opšte namene određen poljem *rd*.

Instrukcija *shrl* (*shift right logical*) vrši pomeranje sadržaja registra opšte namene određenog poljem *rs1* za jedno mesto udesno, pri čemu se najstariji bit rezultata postavlja na

0. Rezultat se upisuje u registar opšte namene određen poljem *rd*.

Instrukcija *shra* (*shift right arithmetic*) vrši pomeranje sadržaja registra opšte namene određenog poljem *rs1* za jedno mesto udesno, pri čemu se u najstariji bit rezultata prepisuje bit znaka početne vrednosti. Rezultat se upisuje u registar opšte namene određen poljem *rd*.

Instrukcija *rotr* (*rotate left*) vrši rotiranje sadržaja registra opšte namene određenog poljem *rs1* za jedno mesto ulevo i upisuje rezultat u registar opšte namene određen poljem *rd*.

Instrukcija *rotr* (*rotate right*) vrši rotiranje sadržaja registra opšte namene određenog poljem *rs1* za jedno mesto udesno i upisuje rezultat u registar opšte namene određen poljem *rd*.

3.1.5.7. USLOVNI SKOKOVI

Instrukcije uslovnih skokova su:

beqz *rs1, imm* if $\text{Regs}[\text{rs1}] = 0$ then $\text{PC} \leftarrow \text{PC} + ((\text{imm}_7)^{24} \parallel \text{imm})$

bnez *rs1, imm* if $\text{Regs}[\text{rs1}] \neq 0$ then $\text{PC} \leftarrow \text{PC} + ((\text{imm}_7)^{24} \parallel \text{imm})$

Instrukcija *beqz* (*branch if equal to zero*) vrši sabiranje sadržaja programskog brojača i neposrednog argumenta. Neposredni argument se pre sabiranja proširuje znakom. Rezultat sabiranja se tretira kao ceo broj bez znaka, a eventualno prekoračenje se ignoriše. Ako je sadržaj registra opšte namene određenog poljem *rs1* jednak nuli, rezultat sabiranja se upisuje u programski brojač.

Instrukcija *bnez* (*branch if not equal to zero*) upisuje odredišnu adresu u programski brojač ako sadržaj registra opšte namene određenog poljem *rs1* nije jednak nuli. Odredišna adresa se određuje na isti način kao u slučaju instrukcije *beqz*.

3.1.5.8. BEZUSLOVNI SKOKOVI

Instrukcije bezuslovnih skokova su:

jr *rs1, imm* $\text{PC} \leftarrow \text{Regs}[\text{rs1}] + ((\text{imm}_7)^{24} \parallel \text{imm})$

jsr *rd, rs1, imm* $\text{Regs}[\text{rd}] \leftarrow \text{PC}; \text{PC} \leftarrow \text{Regs}[\text{rs1}] + ((\text{imm}_7)^{24} \parallel \text{imm})$

Instrukcija *jr* (*jump register*) vrši sabiranje sadržaja registra opšte namene određenog poljem *rs1* i neposrednog argumenta, i upisuje rezultat u programski brojač. Neposredni argument se pre sabiranja proširuje znakom. Rezultat sabiranja se tretira kao ceo broj bez znaka, a eventualno prekoračenje se ignoriše. Ova instrukcija koristi se i za povratak iz potprograma i povratak iz prekida.

Instrukcija *jsr* (*jump to subroutine*) izvršava se na isti način kao instrukcija *jr*, a osim toga upisuje staru vrednost programskog brojača u registar opšte namene određen poljem *rd*. Radi međusobne kompatibilnosti programa, preporučuje se da polje *rd* u slučaju ove instrukcije uvek ima vrednost 30, to jest da upućuje na registar R30. Za povratak iz potprograma koristi

se instrukcija *jr*.

3.1.5.9. PROGRAMSKI PREKID

Instrukcija programskog prekida je:

trap *rs1, imm* $PC \leftarrow \text{Regs}[rs1] + ((imm_7)^{24} \parallel imm); \text{prekid}$

Instrukcija *trap* izvršava se na isti način kao instrukcija *jr*, a osim toga izaziva pokretanje opsluživanja prekida.

Povratak iz prekida izvodi se instrukcijom *jr*. S obzirom da se adresa povratka iz prekida smešta u registar R31, za korektan rad se preporučuje da vrednost polja *rs1* u slučaju instrukcije *jr* koja se koristi za povratak iz prekida bude 31. Takođe, za korektan rad je neophodno da se neposredno pre instrukcije *jr* koja se koristi za povratak iz prekida izvrši instrukcija koja postavlja najstariji bit registra PSW na 1. Taj bit označen je GM (*Global Mask*) i služi za dozvolu ili zabranu pokretanja opsluživanja svih prekida. U završnom delu prekidne rutine bit GM treba da bude isključen, da ne bi ugnježdjeni prekid poremetio adresu povratka iz prekida i/ili vrednost registra PSW, a neposredno pre instrukcije *jr* kojom se završava prekidna rutina treba uključiti taj bit, čime se dozvoljava pokretanje opsluživanja prekida.

3.1.5.10. INSTRUKCIJE ZA PRISTUP POSEBNIM REGISTRIMA

Instrukcije za pristup posebnim registrima su:

intm	<i>imm</i>	interrupt mask	$PSW_2 \leftarrow imm_0$
intgm	<i>imm</i>	interrupt global mask	$PSW_9 \leftarrow imm_0$
trm	<i>imm</i>	trap mode	$PSW_1 \leftarrow imm_0$
movs2i	<i>rd</i>	move spec. reg. to int. reg.	$\text{Regs}[rd] \leftarrow 0^{22} \parallel PSW$
movi2s	<i>rs2</i>	move int. reg. to spec. reg.	$PSW \leftarrow \text{Regs}[rs2]_{9..0}$

Instrukcija *intm* (*interrupt mask*) upisuje najmlađi bit neposrednog argumenta u bit 2 registra PSW. Taj bit označen je MASK i služi za dozvolu ili zabranu pokretanja opsluživanja maskirajućih prekida.

Instrukcija *intgm* (*interrupt global mask*) upisuje najmlađi bit neposrednog argumenta u najstariji bit registra PSW. Taj bit označen je GM (*Global Mask*) i služi za dozvolu ili zabranu pokretanja opsluživanja svih prekida.

Instrukcija *trm* (*trap mode*) upisuje najmlađi bit neposrednog argumenta u bit 1 registra PSW. Taj bit označen je TRM i služi za aktiviranje ili deaktiviranje režima prekid posle svake instrukcije.

Instrukcija *movs2i* (*move special register to integer register*) upisuje sadržaj registra PSW u najmlađih 10 bita registra opšte namene određenog poljem *rd*, a ostatak se popunjava

nulom.

Instrukcija *movi2s* (*move integer register to special register*) upisuje pojedine bite registra opšte namene određenog poljem *rs2* u odgovarajuće bite registra PSW. Biti 7, 6, 5 i 0 (TRAP, UOE, ARE i INC) registra PSW ne mogu biti programski postavljeni i ignorišu se pri upisu, a u bite 9, 8, 4, 3, 2 i 1 (GM, INI, NMI, MI, MASK i TRM) registra PSW se upisuju biti 9, 8, 4, 3, 2 i 1 vrednosti Regs[*rs2*], respektivno.

3.1.6. KONVENCIJE U VEZI STEKA

Stek se realizuje softverski, u memoriji, a ulogu stek pointera obavlja jedan od registara opšte namene. Pristup steku vrši se pomoću instrukcija za pristup memoriji (*lw*, *sw*), a ažuriranje stek pointera pomoću aritmetičkih instrukcija (npr. *addui*, *subui*).

Radi međusobne kompatibilnosti programa neophodno je poštovati određene konvencije u vezi korišćenja steka. Npr. ako kod jednog potprograma stek raste na gore a kod drugog na dole, mogu uništiti podatke jedan drugom. Uvedene su sledeće konvencije:

- Preporučuje se da se kao stek pointer koristi registar R29. Treba izbegavati upotrebu tog registra u druge svrhe. Taj registar koristi se kao stek pointer i u prekidnim rutinama.
- Stek raste na gore (ka višim adresama).
- Stek pointer pokazuje na poslednju zauzetu lokaciju.
- Početnu vrednost stek pointera postavlja operativni sistem prilikom inicijalizacije procesora. Po potrebi, korisnički program može da modifikuje vrednost stek pointera. Početna vrednost stek pointera mora biti postavljena pre prvog poziva potprograma. Naročito treba voditi računa o tome ako korisnički program koristi neki drugi registar kao stek pointer.
- Nije obavezno da se stek pointer ažurira posle svakog upisa na stek ili čitanja sa steka, već se može ažurirati npr. jednom *addui* instrukcijom posle niza uzastopnih *sw* instrukcija. Međutim, da bi se obezbedio korektan rad u slučaju prekida, takav niz instrukcija može sadržati maksimalno 32 uzastopna upisa na stek pre ažuriranja stek pointera, ili treba najpre uvećati stek pointer pa tek onda upisivati podatke na stek. Pri skidanju sa steka važi analogno ograničenje ako se najpre umanjuje stek pointer pa tek onda očitavaju podaci. Naime, prekidna rutina, po dogovoru, na svom početku uvećava stek pointer za 33 i na kraju ga umanjuje za istu vrednost, da ne bi poremetila stek prekinutog programa.

3.1.7. MEHANIZAM PREKIDA

U ovom odeljku izneti su tipovi prekida koji se mogu javiti.

3.1.7.1. TIPOVI PREKIDA

Prekid je vanredna situacija u kojoj se prekida izvršavanje programa i pokreće novi

program, tzv. prekidna rutina, koji vrši određene aktivnosti u zavisnosti od uzroka prekida, a nakon toga se, u većini slučajeva, nastavlja izvršavanje prekinutog programa. Svi prekidi mogu se po izvoru zahteva za prekid podeliti na unutrašnje i spoljašnje. Unutrašnji prekidi mogu biti izazvani programski ili problemima pri izvršavanju instrukcija. Zahteve za spoljašnje prekide generišu periferni uređaji.

U posmatranom procesoru mogu se javiti sledeći prekidi:

- prekid izazvan programski, instrukcijom *trap*,
- prekid zbog pogrešnog koda operacije (označen UOE – *Unexistent Opcode Exception*),
- prekid zbog prekoračenja (*overflow*) pri aritmetičkoj operaciji (ARE – *ARithmetic Exception*),
- nemaskirajući prekid – spoljašnji prekid čije opsluživanje ne može biti softverski zabranjeno (NMI – *Non-Maskable Interrupt*),
- maskirajući prekid – spoljašnji prekid čije opsluživanje može biti softverski zabranjeno (MI – *Maskable Interrupt*),
- režim prekid posle svake instrukcije (TRM – *TRap Mode*, neki autori ga nazivaju *tracing instruction execution*), koristi se za *debugging*.

TRAP, UOE, ARE i TRM su unutrašnji prekidi, a NMI i MI spoljašnji. TRAP i TRM su izazvani programski, a UOE i ARE problemima pri izvršavanju instrukcija.

Za slučaj da se u istom taktu javi više zahteva za prekid, mora postojati prioritet prekida. Najviši prioritet ima TRAP (6), zatim UOE (5), ARE (4), NMI (3), MI (2), a najniži TRM (1).

Po dogovoru, prekid tipa UOE je terminirajući, što znači da prekidna rutina prijavljuje grešku i ne nastavlja izvršavanje programa. Ostali tipovi prekida nisu terminirajući – nakon završetka prekidne rutine nastavlja se izvršavanje programa.

U slučaju posmatranog procesora prekoračenje (prekid ARE) se može javiti prilikom izvršavanja instrukcija sabiranja ili oduzimanja, sa ili bez znaka. Radi pojednostavljenja mehanizma prekida, usvojeno je da se kod svih drugih instrukcija koje obavljaju aritmetičke operacije prekoračenje ignoriše. To su instrukcije koje obavljaju aritmetičke operacije radi izračunavanja memorijske adrese (instrukcija *trap*, instrukcije uslovnih i bezuslovnih skokova, instrukcije za pristup memoriji podataka).

Treba primetiti da se prekidi TRAP, UOE i ARE uzajamno isključuju, tj. bilo koja dva od tih zahteva za prekid ne mogu se javiti u istom taktu. Da bi se javio prekid ARE ili TRAP mora biti prepoznat kod aritmetičke ili *trap* operacije, respektivno, pa se ne može javiti prekid zbog neprepoznatog koda operacije (UOE). Ako kod operacije nije prepoznat (što izaziva prekid UOE), instrukcija koja se izvršava nije ni aritmetička ni *trap* instrukcija, pa ne može dovesti do prekida ARE ni TRAP. Aritmetičke instrukcije, naravno, nisu *trap* i ne mogu izazvati taj tip prekida. Pri izvršavanju instrukcije *trap* eventualno prekoračenje se ignoriše pa se ne može javiti prekid ARE.

U posmatranom procesoru se ne mogu javiti prekidi prilikom pristupa instrukcijskoj memoriji ili memoriji podataka, usled jednostavne organizacije memorije. Na primer, pošto se ne koristi tehnika virtuelne memorije, ne može se javiti *page fault*, *memory protection error* se ne može javiti jer se ne koristi zaštita memorijskih segmenata, *misaligned memory access* se ne može javiti jer je adresiranje na nivou reči, a ne bajtova. Ovo pojednostavljuje mehanizam prekida.

Svi zahtevi za maskirajući prekid stižu po zajedničkoj liniji. Prekidna rutina očitava statusne registre jednog po jednog perifernog uređaja, po redosledu prioriteta tih uređaja, i na taj način utvrđuje koji periferni uređaj je generisao zahtev. Nakon toga isključuje indikator zahteva na tom uređaju. Pretpostavlja se da periferni uređaj koji je generisao zahtev drži aktivnu vrednost signala zahteva sve dok prekidna rutina ne isključi indikator zahteva na tom uređaju. Isto važi za nemaskirajuće prekide, s tim što ne koriste istu liniju kao maskirajući.

3.2. ORGANIZACIJA

U ovom poglavlju se razmatraju elementi organizacije procesora i to osnovni koncepti *pipeline* organizacije procesora, idealizovana *pipeline* organizacija procesora, problemi specifični za *pipeline* procesore (hazardi i prekidi) i rešenja usvojena u posmatranom procesoru.

3.2.1. PIPELINE POSMATRANOG PROCESORA

U ovom odeljku opisani su osnovni elementi pipeline organizacije i osnovne aktivnosti u stepenima pipeline-a posmatranog procesora. Ovaj odeljak zadržava se na idealizovanoj pipeline organizaciji.

3.2.1.1. OSNOVNI ELEMENTI IDEALIZOVANOG PIPELINE-A

U posmatranom procesoru primenjena je sinhrona statička pipeline organizacija. Izvršavanje instrukcija podeljeno je na pet faza: 1) IF (*instruction fetch*) – čitanje instrukcije, 2) ID (*instruction decode and register fetch*) – dekodovanje instrukcije i čitanje registara opšte namene, 3) EX (*execution and effective address calculation*) – izvršavanje ALU operacije, izračunavanje adrese skoka ili memorijske adrese i proveru uslova skoka, 4) MEM (*memory access and branch completion*) – pristup memoriji i izvršavanje skoka, i 5) WB (*write back*) – upis rezultata instrukcije u registar opšte namene. Samim tim procesor se sastoji od pet stepeni, koji imaju iste nazive kao navedene faze.

Organizacija posmatranog procesora i njegove osnovne jedinice prikazane su na slici 7. U stepenu IF nalaze se registar PC (*Program Counter*) – programski brojač, logika za određivanje sledeće vrednosti programskog brojača, i memorija za instrukcije (*Instruction Memory*). U stepenu ID nalaze se registri opšte namene, organizovani kao registar fajl (*Registers*), i jedinica *Extend*, koja proširuje neposredni argument nulom ili znakom u zavisnosti od instrukcije.

U stepenu EX nalaze se aritmetičko-logička jedinica (ALU), i jedinica Zero, koja proverava da li je sadržaj datog registra jednak nuli, tj. da li je ispunjen uslov za uslovni skok. U stepenu MEM nalazi se memorija za podatke (*Data Memory*), a u stepenu WB multiplexer za izbor vrednosti koja se upisuje u registar fajl. Kasnije će biti pomenute još neke jedinice i biće dati detaljniji opisi navedenih jedinica.

Slika 7. *Pipeline* organizacija procesora.

Aktivnosti jedinica koje se nalaze u istom stepenu *pipeline*-a obavljaju se paralelno. U stepenu IF, iz memorije za instrukcije čita se instrukcija određena vrednošću programskog brojača, i paralelno sa tim se određuje sledeća vrednost programskog brojača. U stepenu ID paralelno se obavljaju pristup registar fajlu i proširivanje neposrednog argumenta u jedinici Extend. U stepenu EX jedinice ALU i Zero takođe paralelno obavljaju svoje aktivnosti.

Na ulazu u svaki stepen *pipeline*-a postoji prihvatni registar. Registri R_{ID} , R_{EX} , R_{MEM} i R_{WB} su prihvatni registri za stepene ID, EX, MEM i WB, respektivno. Ti registri sadrže informacije potrebne za izvršavanje instrukcije u stepenu kome registar pripada i u preostalim stepenima. Izlazni prihvatni registar, čija uloga će kasnije biti opisana, nosi oznaku R_{TMP} . Registri R_{ID} , R_{EX} , R_{MEM} , R_{WB} i R_{TMP} nazivaju se *pipeline* registri. Treba primetiti da registar PC sadrži informacije potrebne za izvršavanje faze IF, i stoga predstavlja prihvatni registar za stepen IF.

3.2.1.2. AKTIVNOSTI U STEPENIMA IDEALIZOVANOG PIPELINE-A

Aktivnosti u stepenima idealizovanog *pipeline*-a opisane su na slici 8. U stepenu IF se čita instrukcija iz instrukcijske memorije, sa adrese određene vrednošću registra PC, i na sledeći signal takta upisuje u polje IR registra R_{ID}. Na isti signal takta se u registar R_{ID} upisuje vrednost registra PC. Time se instrukcija prebacuje iz stepena IF u stepen ID. Na isti signal takta se u registar PC upisuje nova vrednost.

Polje IR sastoji se od potpolja opcode, rd, rs1, rs2 i Imm. Radi sažetijeg označavanja u opisu stepena IF prikazan je upis cele instrukcije u polje IR, a u opisima kasnijih stepeni koriste se oznake potpolja, pošto nisu sva potpolja potrebna u svim stepenima.

Stepen	Instrukcije	
IF	sve instrukcije $R_{ID}.IR \leftarrow Mem[PC]; R_{ID}.PC \leftarrow PC$ if $R_{MEM}.COND$ then $PC \leftarrow R_{MEM}.ALUOUT$ else $PC \leftarrow PC + 1$	
ID	sve instrukcije $R_{EX}.A \leftarrow Regs[R_{ID}.rs1]; R_{EX}.B \leftarrow Regs[R_{ID}.rs2]; R_{EX}.rd \leftarrow R_{ID}.rd$ $R_{EX}.PC \leftarrow R_{ID}.PC$	
	logičke instrukcije, addui i subui $R_{EX}.Imm \leftarrow 0^{24} \parallel R_{ID}.Imm$	sve ostale instrukcije $R_{EX}.Imm \leftarrow (R_{ID}.Imm_7)^{24} \parallel R_{ID}.Imm$
EX	sve instrukcije $R_{MEM}.B \leftarrow R_{EX}.B; R_{MEM}.rd \leftarrow R_{EX}.rd; R_{MEM}.PC \leftarrow R_{EX}.PC$ $R_{MEM}.ENDIS \leftarrow R_{EX}.Imm_0$	
	ALU instrukcije $R_{MEM}.ALUOUT \leftarrow R_{EX}.A \text{ func } R_{EX}.B, \text{ ili}$ $R_{MEM}.ALUOUT \leftarrow R_{EX}.A \text{ func } R_{EX}.Imm$ $R_{MEM}.COND \leftarrow 0$	instrukcije za pristup memoriji $R_{MEM}.ALUOUT \leftarrow R_{EX}.A + R_{EX}.Imm$ $R_{MEM}.COND \leftarrow 0$
	instrukcije uslovnog skoka $R_{MEM}.ALUOUT \leftarrow R_{EX}.PC + R_{EX}.Imm$ $R_{MEM}.COND \leftarrow R_{EX}.A \text{ func } 0$	instr. bezuslovnog skoka i prekida $R_{MEM}.ALUOUT \leftarrow R_{EX}.A + R_{EX}.Imm$ $R_{MEM}.COND \leftarrow 1$
MEM	sve instrukcije $R_{WB}.rd \leftarrow R_{MEM}.rd$	
	ALU instrukcije $R_{WB}.ALUOUT \leftarrow R_{MEM}.ALUOUT$	instrukcije trm, intm i intgm $PSW_x \leftarrow R_{MEM}.ENDIS$
	instrukcija sw $Mem[R_{MEM}.ALUOUT] \leftarrow R_{MEM}.B$	instrukcija movi2s $PSW \leftarrow R_{MEM}.B_{9..0}$
	instrukcija lw $R_{WB}.LMD \leftarrow Mem[R_{MEM}.ALUOUT]$	instrukcija movs2i $R_{WB}.ALUOUT \leftarrow 0^{22} \parallel PSW$
	instrukcija jsr $R_{WB}.ALUOUT \leftarrow R_{MEM}.PC$	
WB	sve instrukcije $R_{TMP}.rd \leftarrow R_{WB}.rd$	
	ALU instrukcije, jsr i movs2i $Regs[R_{WB}.rd], R_{TMP}.RES \leftarrow R_{WB}.ALUOUT$	instrukcija lw $Regs[R_{WB}.rd], R_{TMP}.RES \leftarrow R_{WB}.LMD$

Slika 8. Aktivnosti u stepenima pipeline-a posmatranog procesora

Vrednost registra PC vodi se uz instrukciju da bi mogla biti iskorišćena npr. za izračunavanje adrese skoka u slučaju instrukcije uslovnog skoka, ili kao adresa povratka u slučaju instrukcije skoka u potprogram.

Nova vrednost registra PC određuje se na sledeći način:

- Ako je polje $R_{MEM}.COND$ nula, to znači da u stepenu MEM nije instrukcija skoka, ili jeste instrukcija uslovnog skoka ali uslov za skok nije ispunjen. U tom slučaju treba nastaviti sa sekvencijalnim očitavanjem instrukcija, pa se u registar PC upisuje uvećana vrednost tog registra.
- Ako je polje $R_{MEM}.COND$ jedan, to znači da je u stepenu MEM instrukcija bezuslovnog skoka, ili instrukcija uslovnog skoka i uslov za skok je ispunjen. U tom slučaju treba u registar PC upisati vrednost određenu instrukcijom koja se nalazi u stepenu MEM, tj. vrednost polja $R_{MEM}.ALUOUT$.

U stepenu ID se iz registar fajla čitaju sadržaji dva registra, određeni poljima $rs1$ i $rs2$ *pipeline* registra R_{ID} , i na sledeći signal takta upisuju u polja A i B *pipeline* registra R_{EX} , respektivno. U jedinici Extend se neposredna veličina proširuje do 32 bita nulom ili znakom, i na isti signal takta upisuje u polje Imm *pipeline* registra R_{EX} . Na isti signal takta se iz *pipeline* registra R_{ID} u R_{EX} prepisuje sadržaj polja rd i PC.

Proširenje neposredne veličine nulom vrši se u slučaju logičke instrukcije i aritmetičke instrukcije bez znaka (*addui* ili *subui*). Za sve druge instrukcije vrši se proširenje znakom.

U stepenu EX jedinica ALU u slučaju ALU instrukcije izvršava odgovarajuću operaciju nad sadržajima polja $R_{EX}.A$ i $R_{EX}.B$, ili $R_{EX}.A$ i $R_{EX}.Imm$. U slučaju instrukcije uslovnog skoka, jedinica ALU određuje adresu skoka sabiranjem sadržaja polja $R_{EX}.PC$ i $R_{EX}.Imm$. U slučaju instrukcije bezuslovnog skoka ili instrukcije prekida, određuje adresu skoka sabiranjem sadržaja polja $R_{EX}.A$ i $R_{EX}.Imm$. Na isti način se određuje memorijska adresa u slučaju instrukcije za pristup memoriji. Jedinica Zero uporedo sa tim poredi sadržaj polja $R_{EX}.A$ sa nulom.

Sadržaj sa izlaza jedinice ALU se na sledeći signal takta upisuje u polje $ALUOUT$ *pipeline* registra R_{MEM} . Na isti signal takta se u polje $COND$ upisuje izlaz jedinice Zero u slučaju uslovnog skoka, jedan u slučaju bezuslovnog skoka i prekida, ili nula u slučaju svih ostalih instrukcija. Na isti signal takta se iz *pipeline* registra R_{EX} u R_{MEM} prepisuje sadržaj polja rd , PC i B, a najmlađi bit polja Imm prepisuje se u polje $ENDIS$.

U stepenu MEM instrukcije *lw* i *sw* pristupaju memorijskoj lokaciji čija adresa je određena poljem $ALUOUT$ *pipeline* registra R_{MEM} . Instrukcija *sw* upisuje u tu lokaciju sadržaj polja B *pipeline* registra R_{MEM} . Instrukcija *lw* čita sadržaj te lokacije i na sledeći signal takta upisuje ga u polje LMD (*Loaded Memory Data*) *pipeline* registra R_{WB} .

Na taj signal takta, u polje $ALUOUT$ *pipeline* registra R_{WB} se prepisuje:

- u slučaju ALU instrukcije, sadržaj polja ALUOUT *pipeline* registra R_{MEM} ,
- u slučaju instrukcije *jsr*, sadržaj polja PC *pipeline* registra R_{MEM} , i
- u slučaju instrukcije *movs2i*, sadržaj registra PSW proširen nulama.

Na isti taj signal takta vrši se upis u posebni registar PSW u slučaju instrukcije *movi2s*, *intgm*, *intm* ili *trm*. U slučaju instrukcije *movi2s*, u PSW se prepisuju biti 9 do 0 polja $R_{MEM}.B$. U slučaju instrukcija *intgm*, *intm* i *trm*, sadržaj polja $R_{MEM}.ENDIS$ se prepisuje u bit 9, 2 ili 1 registra PSW, respektivno.

Na isti signal takta se iz *pipeline* registra R_{MEM} u R_{WB} prepisuje sadržaj polja rd.

U stepenu WB vrši se upis rezultata instrukcije u registar fajl. Odredišni registar opšte namene određen je poljem rd *pipeline* registra R_{WB} . U slučaju ALU instrukcije, *jsr* ili *movs2i*, u registar se upisuje vrednost polja ALUOUT, a u slučaju instrukcije *lw* vrednost polja LMD *pipeline* registra R_{WB} . U oba slučaja, vrednost koja se upisuje u registar fajl se na sledeći signal takta upisuje i u polje RES *pipeline* registra R_{TMP} . Na isti signal takta se iz *pipeline* registra R_{WB} u R_{TMP} prepisuje sadržaj polja rd.

Neke aktivnosti ovde su, radi lakšeg razumevanja, opisane pojednostavljeno. To se pre svega odnosi na određivanje sledeće vrednosti programskog brojača i na instrukcije uslovnih i bezuslovnih skokova i prekida.

Da bi se obezbedio korektan tok podataka kroz stepene *pipeline*-a, treba generisati odgovarajuće vrednosti upravljačkih signala za multipleksere. U stepenu IF nalazi se multiplekser koji propušta ili uvećanu vrednost registra PC, ili sračunatu adresu skoka koja se nalazi u polju $R_{MEM}.ALUOUT$. Ovim multiplekserom upravlja polje COND *pipeline* registra R_{MEM} . Multiplekser EX1 u stepenu EX propušta na ulaz jedinice ALU vrednost polja $R_{EX}.PC$ u slučaju instrukcije uslovnog skoka, a u slučaju svih drugih instrukcija vrednost polja $R_{EX}.A$. Multiplekser EX2 u stepenu EX propušta na ulaz jedinice ALU vrednost polja $R_{EX}.Imm$ ili $R_{EX}.B$ u zavisnosti od toga da li instrukcija koristi neposredni argument ili ne, respektivno. Multiplekser u stepenu WB propušta za upis u registar fajl vrednost polja $R_{WB}.LMD$ u slučaju instrukcije *lw*, a $R_{WB}.ALUOUT$ u slučaju ALU instrukcije, *jsr* ili *movs2i*. Upravljački signali za ova tri multipleksera generišu se na osnovu koda operacije.

3.2.1.3. AKTIVNOSTI U REALNOM PIPELINE-U

U ovom odeljku opisane su aktivnosti koje se obavljaju u stepenu IF u realnom *pipeline*-u posmatranog procesora. Za sve instrukcije vrše se aktivnosti prikazane na slici 9.

Operacije u kešu za predikciju mogu se opisati na sledeći način:

- ako je predikcija za instrukciju koja se nalazi u stepenu MEM bila *not taken* a ishod je *taken*, u *PCache* ulaz koji odgovara toj instrukciji (tj. adresi $R_{MEM}.PC$) se upisuje odredišna adresa te instrukcije ($R_{MEM}.ALUOUT$),
- ako je predikcija za instrukciju koja se nalazi u stepenu MEM bila *taken* a ishod je *not*

- *taken*, PCache ulaz koji odgovara toj instrukciji (tj. adresi $R_{MEM}.PC$) se invaliduje,
- inače, vrši se čitanje iz keša sa ulaza koji odgovara vrednosti PC.

Sve instrukcije

Operacije u kešu za predikciju

```
if not  $R_{MEM}.CHIT$  and  $R_{MEM}.COND$  then PCache[ $R_{MEM}.PC$ ]  $\leftarrow R_{MEM}.ALUOUT$ 
elseif  $R_{MEM}.CHIT$  and not  $R_{MEM}.COND$  then PCache[ $R_{MEM}.PC$ ].invalidate()
else PCache[PC].read()
```

Upis u *pipeline* registar R_{ID}

```
 $R_{ID}.IR \leftarrow Mem[PC]$ ;  $R_{ID}.PC \leftarrow PC$ ;  $R_{ID}.CHIT \leftarrow PCache[PC].HIT$ ;  $R_{ID}.VALID \leftarrow 1$ 
```

Određivanje nove vrednosti programskog brojača

```
if PREKID and not  $R_{MEM}.TRAP$  then PC  $\leftarrow$  adresa prekidne rutine
elseif (PREKID and  $R_{MEM}.TRAP$ ) or  $R_{MEM}.JUMP$  or (not  $R_{MEM}.CHIT$  and  $R_{MEM}.COND$ )
    then PC  $\leftarrow R_{MEM}.ALUOUT$ 
elseif  $R_{MEM}.CHIT$  and not  $R_{MEM}.COND$  then PC  $\leftarrow R_{MEM}.PC + 1$ 
elseif STALL then PC  $\leftarrow PC$ 
elseif PCache[PC].HIT then PC  $\leftarrow PCache[PC]$ 
else PC  $\leftarrow PC + 1$ 
```

Slika 9. Aktivnosti u stepenu IF realnog procesora

Nova vrednost programskog brojača određuje se po sledećim pravilima:

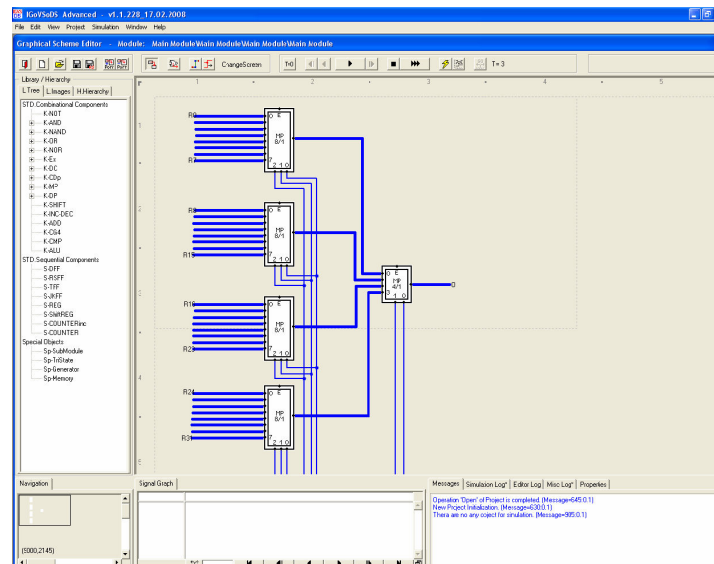
- ako se javio prekid koji nije programski, u PC se upisuje *hardcoded* adresa prekidne rutine,
- inače, ako se javio prekid izazvan instrukcijom *trap*, ili se u stepenu MEM nalazi instrukcija bezuslovnog skoka, ili je predikcija za instrukciju bila *not taken* a ishod je *taken*, u PC se upisuje odredišna adresa te instrukcije ($R_{MEM}.ALUOUT$),
- inače, ako je predikcija za instrukciju koja se nalazi u stepenu MEM bila *taken* a ishod je *not taken*, u PC se upisuje adresa instrukcije koja sekvencijalno sledi tu instrukciju ($R_{MEM}.PC + 1$),
- inače, ako se *pipeline* zaustavlja, vrednost PC se ne menja (zaustavljanje se ne može javiti u slučajevima obuhvaćenim prethodnim granama IF konstrukcije; u slučaju zaustavljanja zapravo se ne vrši bilo kakav upis u PC, jer PC tada ne dobija signal takta),
- inače, ako se javio pogodak u kešu za predikciju, u PC se upisuje vrednost pročitana iz keša,
- inače, PC se inkrementira za 1 (izvršavanje se nastavlja od sekvencijalno sledeće instrukcije).

4. ALAT *IGOVSOEDS*

U ovoj glavi se opisuje alat koji je korišćen za izradu *pipeline* simulatora, kao i način korišćenja alata. Opisan je način projektovanja digitalnih struktura uz pomoć alata. Detaljno su prikazane mogućnosti koje su često korišćene, dok su ostale samo pomenute.

4.1. KORISNIČKI INTERFEJS ALATA *IGOVSOEDS*

Interaktivni generator vizuelnih simulatora elektronskih digitalnih struktura (IGoVSoEDS) pruža intuitivan korisnički interfejs za kreiranje i modifikaciju elektronskih digitalnih struktura, omogućava detaljno podešavanje značajnih parametara svih elektronskih digitalnih kola i modula, i obezbeđuje jednostavno upravljanje simulacijom i efikasan sistem pregleda rezultata rada simulacije trenutno aktivne elektronske digitalne strukture (u daljem tekstu *projekat*). Na slici 10 prikazan je osnovni prozor alata *IGoVSoEDS*, koji se prilagođava trenutnoj rezoluciji radne površine, i može da se koristi za sve rezolucije.



Slika 10. Osnovni prozor alata *IGoVSoEDS*, sa realizacijom multipleksera 32/1
Glavne komponente alata *IGoVSoEDS* su:

- Osnovni prozor softverskog paketa, čiji su sastavni delovi:
 - podloga za prikaz elektronske digitalne strukture,
 - glavni meni i grafički skup poziva funkcija,
 - biblioteke kola i hijerarhijsko stablo modula,
 - navigacija i pregled rezultata rada simulatora,
 - pregled poruka i sadržaja strukture podataka,

- sistem za pomoć pri korišćenju softverskog paketa.
- Funkcije grafičkog editora elektronske digitalne strukture
- Rad sa parametrima simulacije i upravljanje simulatorom

4.2. OSNOVNI PROZOR ALATA

U poglavljima koja slede daje se pregled svih komponenti osnovnog prozora alata *IGoVSoEDS*, od kojih su neke koje su intenzivno korišćenje u realizaciji simulatora detaljno opisano.

4.2.1. PODLOGA ZA PRIKAZ ELEKTRONSKE DIGITALNE STRUKTURE

Deo podloge na kojoj se realizuju topologija šeme elektronskih digitalnih struktura u daljem tekstu nazivaće se *podloga za prikaz strukture*. Prikazan je deo podloge koji je trenutno vidljiv (slika 10), dok je ostatak podloge dostupan pomeranjem podloge pomoću strelica na tastaturi, u smerovima levo–desno i gore–dole. Na svim rubovima podloge prikazan je lenjir za lakše određivanje pozicije na podlozi. Lenjir ima po 10 podeoka na svim rubovima podloge, i može da se ukloni aktiviranjem odgovarajuće funkcije glavnog menija.

4.2.2. GLAVNI MENI I GRAFIČKI SKUP POZIVA FUNKCIJA

Na slici 11 prikazan je gornji deo osnovnog prozora alata *IGoVSoEDS*, na kome su informacija o verziji softverskog paketa, glavni meni, naziv modula elektronske digitalne strukture koji je trenutno prikazan i grafički skupovi poziva funkcija (*toolbars*).



Slika 11. Gornji deo osnovnog prozora alata *IGoVSoEDS*

U poglavljima koja slede daju se pregledi glavnog menija i grafičkih skupova poziva funkcija.

4.2.2.1. GLAVNI MENI

Na slici 12 prikazan je glavni meni alata *IGoVSoEDS*. Na raspolaganju su grupe funkcija:

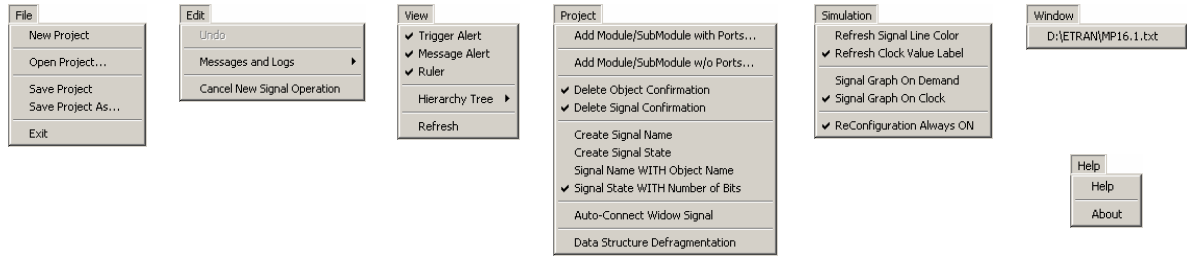
File – skup standardnih funkcija za aktiviranje novog projekta, kao i za rad sa datotekama, koji ima sledeće opcije: *New Project*, *Open Project...*, *Save Project*, *Save Project As...* i *Exit*.

Edit – funkcije za poništavanje akcija korisnika (*Undo* i *Cancel New Signal Operation*), kao i funkcije za rad sa objektima za obaveštavanje korisnika o radu softverskog paketa, koji ima sledeće opcije: *Undo* (*Undo Alignment*, *Undo Signal*), *Messages and Logs*, *Cancel New Signal Operation*.

View – funkcije za definisanje vrsta i načina prikaza poruka (*Trigger Alert*, *Message Alert*

i *Ruler*), i prikaza hijerarhijskog stabla modula (skup funkcija *Hierarchy Tree*), kao i funkcija za restauraciju sadržaja podloge za prikaz strukture (*Refresh*), koji ima sledeće opcije: *Trigger Alert*, *Message Alert*, *Ruler*, *Hierarchy Tree*, *Refresh*.

Project – funkcije za otvaranje dijaloga za učitavanje modula iz datoteka, kao i za definisanje vrsta i načina komunikacije korisnika i softverskog paketa, koji ima sledeće opcije: *Add Module/SubModule with Ports...*, *Add Module/SubModule w/o Ports...*

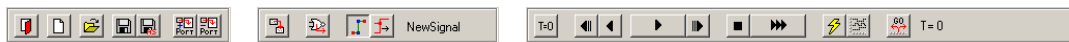


Slika 12. Glavni skup funkcija alata *IGoVSoEDS* – Glavni meni

Skup parametara kojima se definiše ponašanje alata čine opcije: *Delete Object Confirmation*, *Delete Signal Confirmation*, *Create Signal Name*, *Create Signal State*, *Signal Name WITH Object Name*, *Signal State WITH Number of Bits*, *Auto-Connect Widow Signal*, *Data Structure Defragmentation*, *Simulation*, *Refresh Signal Line Color*, *Refresh Clock Value Label*, *Signal Graph On Demand*, *Signal Graph On Clock*, *ReConfiguration Always ON*, *Window*, *Help*.

4.2.2.2. GRAFIČKI SKUP POZIVA FUNKCIJA

Grafički skupovi funkcija obezbeđuju prečice ka najznačajnijim funkcijama realizovanim u glavnom meniju alata *IGoVSoEDS*, kao i skup funkcija za upravljanje radom simulatora trenutno aktivne digitalne strukture (*Toolbars*, slika 13).



Slika 13. Grafički skup funkcija za rad sa datotekama, za izbor režima rada grafičkog editora i za rad sa funkcijama simulatora

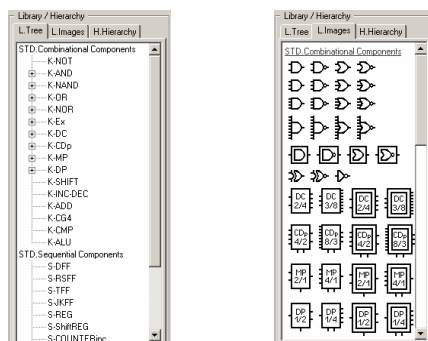
Grafički skupovi funkcija obezbeđuju:

- rad sa datotekama (*File Toolbar*), odnosno pozive najznačajnijih funkcija glavnog menija: *File Menu*, *Project Menu*
- izbor režima rada grafičkog editora elektronske digitalne strukture (*Editor Mode Toolbar*): *Change Screen*, *Move Object*, *New Signal*, *Move Signal*, *Current Mode Indicator*
- skup funkcija za rad sa simulatorom elektronske digitalne strukture (*Simulation Toolbar*): *Reset Simulation*, *Fast Backward Simulation*, *Backward Simulation*,

Forward Simulation, Next Clock, Fast Forward Simulation, Stop Simulation, Forward Simulation, Run Simulation, Signal Graph, Go To Tcl i druge

4.2.3. BIBLIOTEKE KOLA I HIJERARHIJSKO STABLO MODULA

Na slici 14 prikazana su dva raspoloživa sadržaja dela osnovnog prozora za izbor kola koje treba staviti na podlogu za prikaz strukture (*Library: L.Tree* i *L.Images*, respektivno).



Slika 14. Biblioteke za izbor kola, tekstualna *L.Tree* i grafička *L.Images*

Na raspolaganju se sledeći tipovi kola i modula:

Standardna kombinaciona kola, sa označenim brojem ulaznih i izlaznih konektora: K-NOT, K-AND_i, K-NAND_i, K-OR_i, K-NOR_i, K-ExOR2, K-ExNOR2, K-DC_{k-m}, K-CDP_{k-m}, K-MP_{k-m}, K-DP_{k-m}, K-SHIFT, K-INC-DEC, K-ADD, K-CG4, K-CMP i K-ALU. Nakon izbora iz biblioteke, na ulazne konektore ovih kola mogu da se povežu signali širine 1 bit, dok su izlazni signali širine 1 bit. Primenom funkcije *Expand Connectors...* menjaju se širine svih konektora.

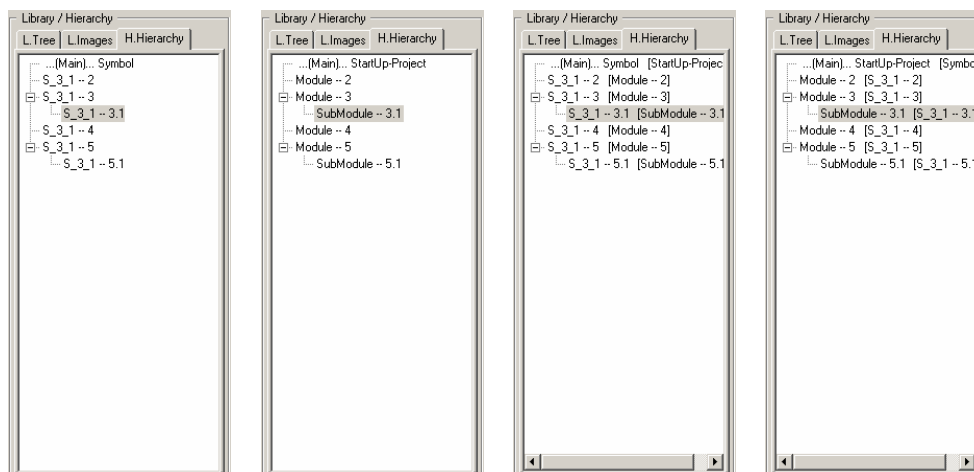
Standardna kombinaciona kola, sa modifikovanim ulaznim konektorima: K-gAND, K-gNAND, K-gOR, K-gNOR, K-gDC_{k-m}, K-gCDP_{k-m}, K-gMP_{k-m} i K-gDP_{k-m}. Sva kola iz ove grupe imaju po jedan ulazni konektor na koji može da se poveže signal sa širinom od 1 do 32 bita, dok je izlazni signal uvek širine 1 bit. Nakon izbora iz biblioteke, ulazni konektori ovih kola prihvataju signale širine 16 bita. Primenom funkcije *Expand Connectors...* menjaju se širine samo ulaznih konektora.

Standardna sekvencijalna kola: S-DFF, S-RSFF, S-TFF, S-JKFF, S-REG, S-ShiftREG, S-COUNTERinc i S-COUNTER. Nakon izbora flip-flopova iz biblioteke kola, na ulazne konektore mogu da se povežu signali širine 1 bit, dok su izlazni signali širine 1 bit. Primenom funkcije *Expand Connectors...* menjaju se širine svih konektora. Nakon izbora registara i brojača iz biblioteke kola, ulazni i izlazni signali za podatke su širine 16 bita, dok su svi kontrolni ulazni i izlazni signali širine 1 bit. Primenom funkcije *Expand Connectors...* menjaju se širine samo ulaznih i izlaznih signala za podatke.

Specijalni moduli: *Sp-SubModule*, *Sp-TriState*, *Sp-Generator* i *Sp-Memory*.

Na stablu za prikaz hijerarhijske strukture modula elektronske digitalne strukture (slika 15)

može da se vidi organizacija i raspored raspoloživih modula, a može i da se izabere modul koji će se prikazati na podlozi za prikaz strukture. Izbor modula koji treba da se prikaže obavlja se postavljanjem strelice pokazivača (*mouse pointer*) na odabranu stavku stabla *H.Hierarchy* (slika 15), i aktiviranjem levog tastera miša izabrani modul pojavljuje se na podlozi za prikaz strukture. Aktiviranjem odgovarajućih funkcija glavnog menija izgled stabla za prikaz hijerarhijske strukture modula može da se menja.



Slika 15. Stablo hijerarhijske organizacije modula elektronske digitalne strukture

4.3. FUNKCIJE GRAFIČKOG EDITORA ELEKTRONSKE DIGITALNE STRUKTURE

U zavisnosti od odabranog režima rada grafičkog editora elektronske digitalne strukture (u daljem tekstu *grafički editor*), obezbeđeni su postupci za rad sa:

- objektima koji predstavljaju standardna kola i podređene module,
- signalima i linijama signala,
- natpisima za prikaz naziva signala i za prikaz stanja na signalima širim od jednog bita,
- natpisima opšte namene.

U nekoliko narednih poglavlja daje se detaljni pregled izabranih funkcija koje mogu da se aktiviraju nakon otvaranja padajućih menija u grafičkom editoru.

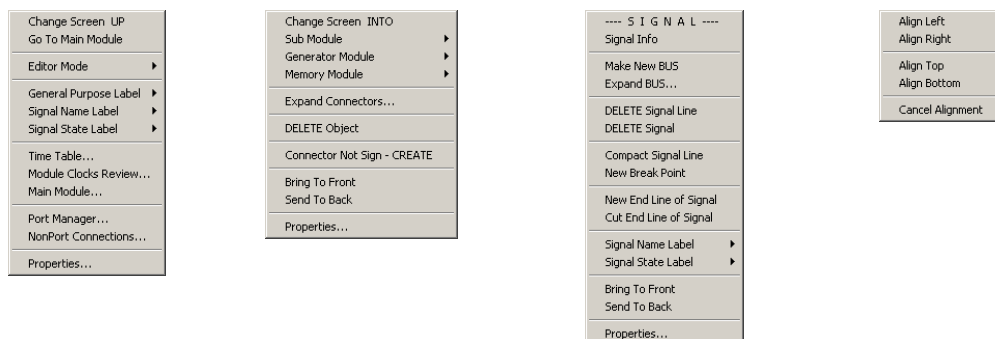
4.3.1. SKUP FUNKCIJA

U poglavljima koja slede daju se pregledi osnovnog skupa funkcija i izabranih funkcija koje se mogu aktivirati korišćenjem osnovnog skupa funkcija. Neke od funkcija su samo pomenute, a neke, koje su korišćene za realizaciju simulatora su detaljno opisane.

4.3.1.1. OSNOVNI SKUP FUNKCIJA

U zavisnosti od vrste objekta na podlozi iznad koga se aktivira otvaranje menija, kao i režima rada grafičkog editora, otvaraju se različiti padajući meniji (slika 16): *Canvas Menu*,

Object Menu, Signal Menu, Special Menu.



Slika 16. Osnovni skup funkcija za kreiranje i modifikaciju elektronske digitalne strukture (*PopUp Menus*): *Canvas Menu*, *Object Menu*, *Signal Menu* i *Special Menu*, respektivno

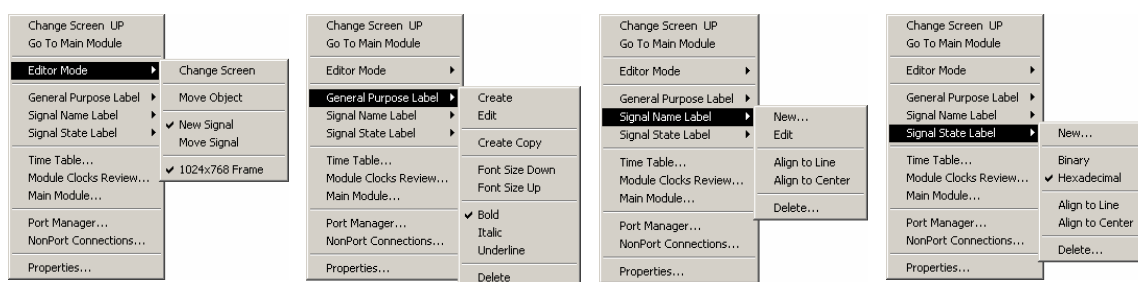
U zavisnosti od konteksta u kome se otvaraju padajući meniji, neke od funkcija nisu dostupne.

4.3.1.2. DODATNI SKUP FUNKCIJA

U poglavljima koja slede daje se pregled svih raspoloživih funkcija koje mogu da se aktiviraju otvaranjem padajućih menija *Canvas Menu*, *Object Menu*, *Signal Menu* i *Special Menu*.

4.3.1.2.1. Meni za rad sa podlogom za prikaz strukture (*Canvas Menu*)

Na slici 17 prikazane su sve funkcije koje su dostupne u okviru grupe funkcija *Canvas Menu*.



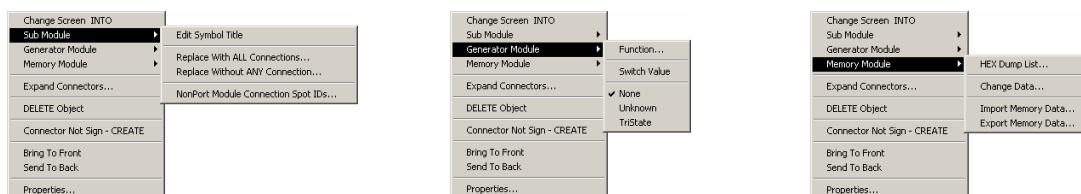
Slika 17. *Canvas Menu*: Skup funkcija *Editor Mode*, *General Purpose Label*, *Signal Name Label* i *Signal State Label*, respektivno

Skup funkcija *Canvas Menu* dostupan je kada se pokazivač miša postavi na slobodnu površinu podloge za prikaz strukture i aktivira desni taster miša. Slobodna površina podloge za prikaz strukture je ona površina na kojoj se ne nalazi kolo, modul ili linija signala. Meni za rad sa podlogom za prikaz strukture (*Canvas Menu*) obezbeđuje funkcije: *Change Screen UP*, *Go To Main Module*, *Editor Mode* (*Change Screen*, *Move Object*, *New Signal*, *Move Signal*, *1024x768 Frame*), *General Purpose Label* (*Create*, *Edit*, *Create Copy*, *Font Size Down*, *Font Size Up*, *Bold*, *Italic*, *Underline* i *Delete*), *Signal Name Label* (*New...*, *Edit*, *Align to Line*, *Align to Center*, *Delete...*), *Signal State Label* (*New...*, *Binary*, *Hexadecimal*, *Align to Line*, *Align to Center*, *Delete...*).

Align to Center, Delete...), Signal State Label (New..., Binary, Hexadecimal, Align to Line, Align to Center, Delete...), Time Table..., Module Clocks Review..., Main Module..., Port Manager..., NonPort Connections..., Properties...

4.3.1.2.2. Meni za rad sa objektima (Object Menu)

Na slici 18 prikazane su sve funkcije koje su dostupne u okviru grupe funkcija *Object Menu*.



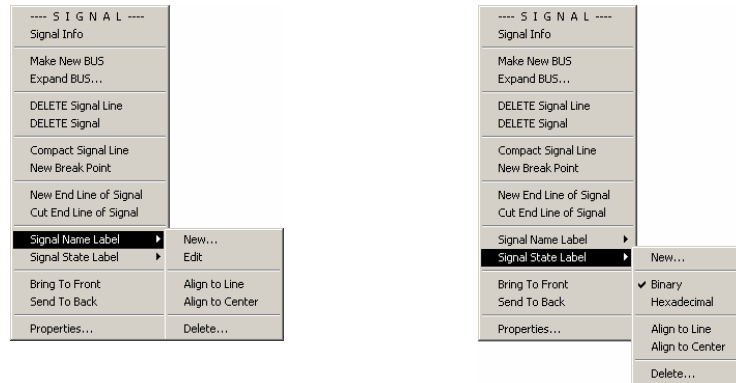
Slika 18. *Object Menu*: Skup funkcija *Sub Module*, *Generator Module* i *Memory Module*

Skup funkcija *Object Menu* dostupan je kada se pokazivač miša postavi na neko kolo ili modul, i aktivira desni taster miša. Meni za rad sa objektima (*Object Menu*) obezbeđuje funkcije: *Change Screen INTO*, *Sub Module* (*Edit Symbol Title*, *Replace With ALL Connections...*, *Replace Without ANY Connection...*, *NonPort Module Connection Spot IDs...*), *Generator Module* (*Function...*, *Switch Value*, *None*, *Unknown* i *TriState*), *Memory Module* (*HEX Dump List...*, *Change Data...*, *Import Memory Data...* i *Export Memory Data...*), *Expand Connectors...*, *DELETE Object*, *Connector Not Sign* (*CREATE* i *REMOVE*), *Bring To Front*, *Send To Back*, *Properties...*

Funkcije *Sub Module*, *Generator Module* i *Memory Module* međusobno se isključuju i uvek je dostupna samo jedna od ovih funkcija, u zavisnosti od modula iznad koga je postavljen pokazivač miša prilikom otvaranje skupa funkcija *Object Menu*.

4.3.1.2.3. Meni za rad sa signalima (Signal Menu)

Na slici 19 prikazane su sve funkcije koje su dostupne u okviru grupe funkcija *Signal Menu*. Skup funkcija *Signal Menu* dostupan je kada se pokazivač miša postavi na neku liniju signala i aktivira desni taster miša.



Slika 19. *Signal Menu*: Skup funkcija *Signal Name Label* i *Signal State Label*, respektivno

Meni za rad sa signalima (*Signal Menu*) obezbeđuje funkcije: *Signal Info*, *Make New BUS*, *Expand BUS...*, *DELETE Signal Line*, *DELETE Signal*, *Compact Signal Line*, *New Break Point*, *New End Line of Signal*, *Cut End Line of Signal*, *Signal Name Label* (*New...*, *Edit*, *Align to Line*, *Align to Center*, *Delete...*), *Signal State Label*, *New...*, *Binary*, *Hexadecimal*, *Align to Line*, *Align to Center*, *Delete...*), *Bring To Front*, *Send To Back*, *Properties...*

4.3.1.2.4. Meni za rad sa grupama objekata (*Special Menu*)

Na raspolaganju su funkcije: *Align Left*, *Align Right*, *Align Top* i *Align Bottom*, kada se poravnavanje radi prema levoj, desnoj, gornjoj i donjoj ivici referentnog objekta, respektivno, gde je referentni objekat, objekat nad kojim se otvara *Special Menu*. Funkcija *Cancel Alignment* obustavlja operaciju poravnavanja i uklanja okvir za grupisanje objekata.

4.3.2. DETALJNI PREGLED IZABRANIH FUNKCIJA

U poglavljima koja slede daju se detaljni pregledi nekih od:

- funkcija koje se aktiviraju pozivima iz menija:
 - *Canvas Menu* – rad sa podlogom za prikaz strukture,
 - *Object Menu* – rad sa objektima,
 - *Signal Menu* – rad sa signalima,
 - *Special Menu* – poravnavanje objekata.
- skupa postupaka za kreiranje i modifikaciju delova struktura neophodnih za realizaciju složenijih digitalnih struktura:
 - pregled postupaka za rad sa signalima,
 - pregled postupaka za rad sa objektima.

4.3.2.1. PORTOVI I SIMBOLI MODULA (*PORT MANAGER...*)

Na slici 20 prikazan je dijalog za kreiranje i modifikaciju podataka o portovima, koje bi trenutno aktivna digitalna struktura imala ukoliko se integriše u neku drugu digitalnu

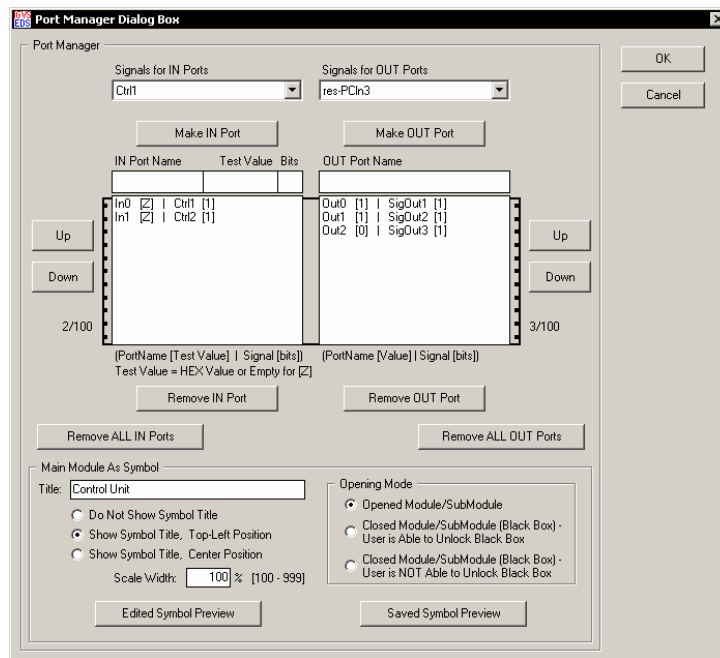
strukturu kao modul. U gornjem delu su dati *Combo Box* objekti sa listama kandidata za ulazne i izlazne portove (*Signal for IN ports* i *Signal for OUT ports*). Aktiviranjem tastera *Make IN Port* i *Make OUT Port* signal koji je trenutno odabran u odgovarajućoj listi kandidata promoviše se u ulazni (*IN*), odnosno izlazni (*OUT*) port, respektivno. Signali određeni za ulazne i izlazne portove pojavljuju se u listama ispod tastera za promociju signala u portove. Pored lista ulaznih i izlaznih portova nalaze se tasteri za promenu redosleda portova (*Up* i *Down*), kao i za brisanje odabranog ulaznog ili izlaznog signala (*Remove IN Port* i *Remove OUT Port*, respektivno), odnosno za brisanje svih ulaznih ili izlaznih portova (*Remove ALL IN Ports* i *Remove ALL OUT Ports*, respektivno). Redosled portova je veoma važan, jer je time određen redosled crtanja portova na simbolu koji će predstavljati modul. U donjem delu dijaloga dat je deo za podešavanje izgleda simbola (*Main Module As Symbol*). Obezbeđena je modifikacija natpisa na simbolu, mesta gde će biti napisan naziv, procentualne vrednosti za deformaciju veličine simbola, kao i parametra koji određuje da li će korisnik moći da uđe i pregleda strukturu modula, ili to neće biti dopušteno (*Opening Mode*). Omogućen je izbor tri vrednosti parametra *Opening Mode*:

- *Opened Module/SubModule* – dozvoljeno je da se uđe i pregleda struktura modula,
- *Closed Module/SubModule (Black Box), User is Able to Unlock Black Box* – nije dozvoljeno da se uđe i pregleda struktura modula, ali korisnik može da promeni dozvolu za ulazak u modul promenom vrednosti parametra *Open for Entering Into* u dijalogu *Canvas Menu/Properties...*,
- *Closed Module/SubModule (Black Box), User is NOT Able to Unlock Black Box* – nije dozvoljeno da se uđe i pregleda struktura modula.

Vrednost *Opening Mode* parametra označena je na simbolu modula bojom malog pravougaonika u gornjem levom uglu, gde su boje zelena, crna i crvena, respektivno prema redosledu mogućih vrednosti parametra koji je naveden.

U donjem delu dijaloga nalaze se tasteri *Edited Symbol Preview* i *Saved Symbol Preview* čijim aktiviranjem se otvara dijalog za pregled izgleda simbola modula. Ovim dijalogom može da se prikaže kako bi izgledao simbol modula ukoliko se tasterom OK prihvate učinjene promene (*Edited...*), kao i trenutni izgled simbola modula (*Saved...*).

Postupak integracije modula u drugu elektronsku digitalnu strukturu obavlja se aktiviranjem funkcija *Add Module/SubModule with Ports...* i *Add Module/SubModule w/o Ports...*. Ukoliko je aktivirana funkcija *Add Module/SubModule with Ports...* modul će biti integrisan kao simbol sa portovima samo ako su portovi definisani, u suprotnom, modul će biti integrisan kao simbol bez portova. Ukoliko je aktivirana funkcija *Add Module/SubModule w/o Ports...* modul će biti integrisan kao simbol bez portova, bez obzira da li su portovi definisani ili nisu definisani.



Slika 20. Funkcija *Canvas/Port Manager...* – dijalog za kreiranje i modifikaciju podataka o portovima

Pored definisanja portova koji se koriste za povezivanje signala i modula, moguće je i povezivanje signala i modula bez definisanih portova. Moduli na koje su povezani signali mogu da se zamene drugim modulima tako da se ostvarene konekcije sačuvaju ili da se raskinu. Zamena modula sa portovima (*Port Module*) ostvaruje se korišćenjem informacija o portovima, ali takve informacije ne postoje za module bez portova (*NonPort Module*). Za potrebe ostvarivanja zamene modula bez portova, sa mogućnošću da se očuvaju uspostavljene konekcije, obezbeđena je mogućnost numeracije signala.

4.3.2.2. MEMORIJSKI MODULI (MEMORY MODULE)

Memorijski modul je jedan od posebnih modula koji su na raspolaganju u biblioteci kola. Memorijski modul je realizovan kao modul sa standardnim kontrolnim ulazima, ulazima za adresu i podatak, kao i izlazom za podatak koji je pročitao. Ulazne linije za adrese i podatke imaju po 16 bita. Na raspolaganju je 65 memorijskih modula, sa memorijskim rečima širine 16 bita, sa punim kapacitetom za prihvatanje vrednosti sadržaja svih memorijskih lokacija. Širine ulaznih i izlaznih linija za podatke mogu da se podešavaju korišćenjem funkcije *Object/Expand Connectors...*, sa dozvoljenim vrednostima broja linija na ulaznim i izlaznim linijama za podatke 8, 4, 2 i 1. Na ovaj način definišu se širine memorijskih reči memorijskih modula, i broj raspoloživih memorijskih modula može da se kreće od 65 do 1025.

Skup funkcija za rad sa sadržajem memorijskih modula (*Object/Memory Module*), obezbeđuje funkcije:

HEX Dump List... – funkcija za otvaranje dijaloga za pregled sadržaja memorijskih

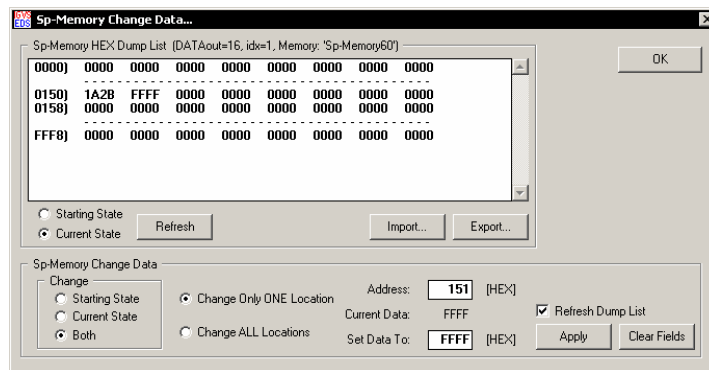
lokacija memorijskog modula. Dijalog ima izgled kao gornji deo dijaloga prikazanog na slici 21. Taster *Change* (taster *Refresh* na dijalogu prikazanom na slici 21) transformiše *HEX Dump List...* dijalog u *Change Data...* dijalog. Tasteri *Import...* i *Export...* pozivaju iste dijaloge kao i funkcije *Import Memory Data...* i *Export Memory Data...*, respektivno.

Change Data... – funkcija za otvaranje dijaloga za pregled i modifikaciju sadržaja memorijskih lokacija memorijskog modula (slika 21).

Import Memory Data... – dijalog za učitavanje sadržaja memorijskih lokacija iz datoteke.

Export Memory Data... – dijalog za snimanje sadržaja memorijskih lokacija u datoteku.

Dijalozi *Import...* i *Export...* imaju identični izgled kao i dijalozi *Open Project...* i *Save Project As...*, respektivno, sa odgovarajućim zaglavlјima.



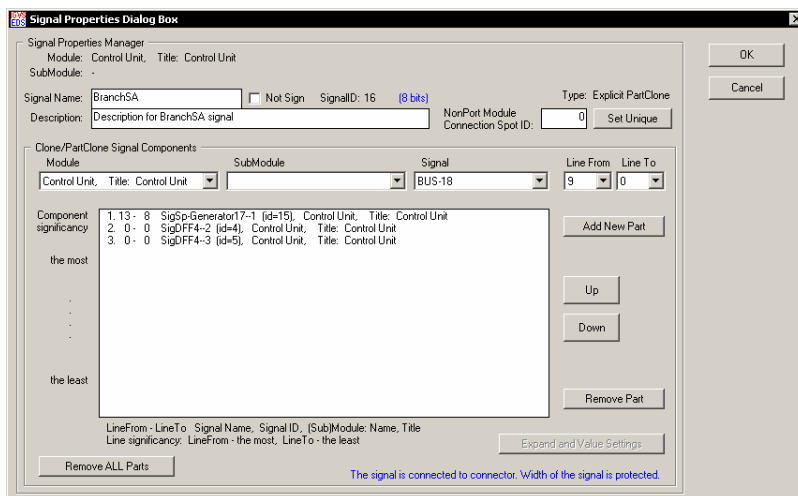
Slika 21. Funkcija *Object/Memory Module/Change Data...*

Vrednost *Optional Box* objekta unutar okvira *Sp-Memory HEX Dump List*, *Starting State* i *Current State*, definiše da li se u *Text Box* objektu prikazuju sadržaji početnog stanja memorijskih lokacija ili sadržaj trenutnog stanja memorijskih lokacija, respektivno. Korišćenjem objekata unutar okvira *Sp-Memory Change Data* može da se modifikuje sadržaj memorijskih lokacija samo početnog stanja (*Starting State*), samo trenutnog stanja (*Current State*), ili i početnog i trenutnog stanja (*Both*), kao i da se promeni sadržaj memorijske lokacije na koju ukazuje adresa u polju *Address* (*Change Only ONE Location*) ili sadržaj svih memorijskih lokacija (*Change ALL Locations*). Vrednost na koju treba da se postavi sadržaj odabrane memorijske lokacije ili sadržaj svih memorijskih lokacija treba da se unese u polje *Set Data To*. Tasterom *Apply* aktivira se operacija modifikacije sadržaja memorijske lokacije, odnosno memorijskih lokacija prema vrednostima polja čija namena je opisana. Taster *OK* zatvara dijalog.

4.3.2.3. OSOBINE SIGNALA I KREIRANJE CLONE/PARTCLONE SIGNALA (PROPERTIES...)

Na slici 22 prikazan je dijalog za kreiranje i modifikaciju osnovnih podataka o odabranom signalu (*Signal Properties Dialog*). Osnovni skup polja za pregled i modifikaciju podataka o signalu dostupan je za sve vrste signala i sadrži sledeća polja: naziv signala (*Signal Name*),

kraći opis namene signala (*Description*), parametar koji određuje da li iznad natpisa naziva signala treba da se nalazi horizontalna linija kao indikator negacije signala (*Not Sign*), kao i jedinstveni broj u sistemu numeracije signala za potrebe rada funkcije *Replace With ALL Connections...* (*NonPort Module Connection Spot ID*).

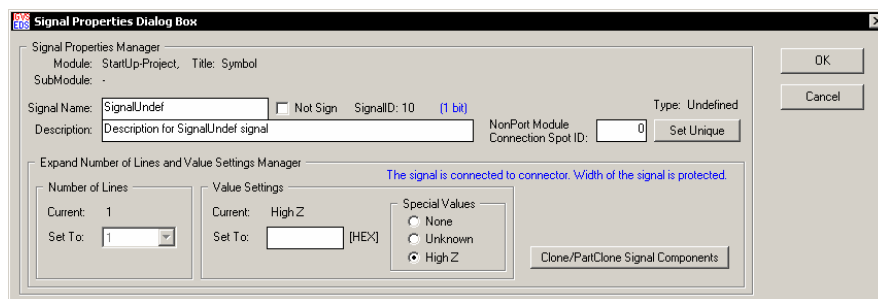


Slika 22. Funkcija *Signal/Properties...*

Za tipove signala *Undefined*, *Clone* i *PartClone* dostupan je i donji deo dijaloga *Signal Properties Dialog (Clone/PartClone Signal Components)*. Obezbeđen je sistem sa listama modula (*Module*), podređenih modula (*SubModule*) i signala (*Signal*), preko kojih je dostupan svaki signal koji je vidljiv na modulu koji je aktivan prilikom otvaranja padajućeg menija *Signal Menu*. Izborom signala, korišćenjem sistema za pregled signala (*Module/SubModule/Signal*), izborom pojedinih linija odabranog signala, korišćenjem objekata *Line From/Line To*, i aktiviranjem tastera *Add New Part*, odabrane linije signala postaju komponente kojima se kreira zavisni signal za koji je otvoren *Signal Properties Dialog*. Neposredno levo (*Component significance*) i ispod (*Line significance*) liste komponenti signala označene su koordinate važnosti pojedinih komponenti i pojedinih linija svake komponente. Obezbeđeni su tasteri za modifikaciju redosleda komponenti signala (*Up* i *Down*) i za brisanja jedne ili svih komponenti (*Remove Part* i *Remove ALL Parts*, respektivno). Na ovaj način obezbeđena je mogućnost kreiranja signala sastavljenih od proizvoljnih delova svih vidljivih signala. Ukoliko zavisni signal ima samo jednu komponentu u vidu svih linija jednog signala, dobija status *Clone Signal*, dok u svim ostalim slučajevima dobija status *PartClone Signal*. Signali čije se linije koriste kao komponente u kreiranju *Clone* ili *PartClone* signala dobijaju status *Master Signal*. Signali u statusu *Clone* i *PartClone* mogu da budu *Master* signali za druge *Clone* i *PartClone* signale.

Ukoliko je signal nad kojim je otvoren padajući meni *Signal Menu* tipa *Undefined*, dostupan je taster *Expand and Value Settings*, u donjem desnom uglu dijaloga prikazanog na slici 22. Aktiviranjem ovog tastera, dijalog prikazan na slici 22 transformiše se u dijalog

prikazan na slici 23. Aktiviranjem tastera *Clone/PartClone Signal Components*, u donjem desnom uglu dijaloga prikazanog na slici 23, dijalog se ponovo vraća u oblik prikazan na slici 22. Ukupan broj bitova *Clone/PartClone* signala mora da bude identičan pri otvaranju i zatvaranju dijaloga, ukoliko je *Clone/PartClone* signal povezan na neki ulazni konektor ili port. U tom slučaju vidljiva je plava napomena *The signal is connected...*, na dnu dijaloga, i svaka promena koja narušava navedeno pravilo onemogućava prihvatanje promena aktiviranjem tastera *OK*.



Slika 23. Funkcija *Signal/Properties...* – *Expand and Value Settings*

Donji deo dijaloga prikazanog na slici 23 omogućava da se odabranom *Undefined* signalu, signalu bez izvora vrednosti, dodeli broj bitova od 1 do 32, u koracima po 1 bit (*Number of Lines*), i da se postavi vrednost (*Value Settings*). Modifikacija broja bitova nije dostupna ukoliko je signal povezan na neki konektor ili port, jer je time broj bitova signala definisan. U tom slučaju vidljiva je plava napomena *The signal is connected...*. Vrednost *Undefined* signala postavljena u ovom dijalogu postoji sve do trenutka kada se kroz grafički editor dodeli izvor vrednosti za signal. Tada signal gubi status *Undefined* signala, kao i postavljenu vrednost u *Value Settings* dijalogu. Taster *OK* zatvara dijalog sa prihvatanjem promena, dok taster *Cancel* zatvara dijalog bez prihvatanja promena.

4.3.2.4. POVEZIVANJE SIGNALA I MODULA

Povezivanje signala i modula je jedna od najznačajnijih funkcija alata *IGoVSoEDS*. Ovom funkcijom omogućava se razvoj hijerarhijske organizacije modula aktivne digitalne strukture. Realizovane su dve vrste modula: moduli sa portovima (*Port Module*) i moduli bez portova (*NonPort Module*). Stoga je povezivanje signala i modula realizovano kroz dva postupka:

- povezivanje signala i modula sa portovima (*Port Module*),
- povezivanje signala i modula bez portova (*NonPort Module*).

U paragrafima koji slede daju se detaljni opisi mogućnosti za povezivanje signala i modula.

4.3.2.4.1. Povezivanje signala i modula sa portovima (*Port Module*)

Korišćenjem funkcije *Canvas/Port Manager...* modulu se pridružuju neophodni podaci za definisanje ulaznih i izlaznih portova, koji se prikazuju kada se taj modul integriše u drugu

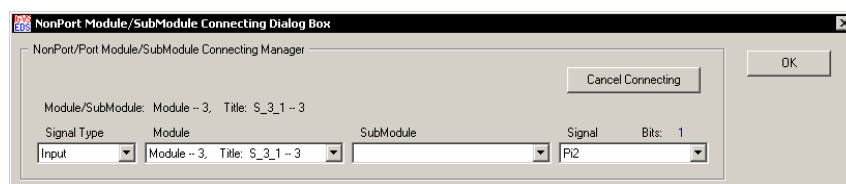
digitalnu strukturu. Dodavanje *Port Module* objekta digitalnoj strukturi ostvaruje se korišćenjem funkcije *Add Module/SubModule with Ports...*, koja pripada grupi funkcija glavnog menija *Project*. Nakon dodavanja *Port Module* objekta drugoj digitalnoj strukturi, prikazuju se ulazni i izlazni portovi. Kreiranje linija signala koje će povezivati ulazne i izlazne portove, obavlja se na identičan način kao i za konektore kola i specijalnih modula. Jedino ograničenje u odnosu na konektore je da nije moguće kreirati kružić za negaciju signala na portu.

4.3.2.4.2. Povezivanje signala i modula bez portova (*NonPort Module*)

NonPort Module objekti nemaju unapred definisane portove za povezivanje signala. Stoga je omogućeno da se ostvari veza svakog signala van modula sa odgovarajućim signalom unutar modula, koji je vidljiv van modula. Neophodno je da su ispunjeni standardni uslovi za povezivanje signala, odnosno da su signali odgovarajućeg tipa i da imaju isti broj bitova. U zavisnosti od početne i krajnje tačke linije signala, sa signalom unutar *NonPort Module* objekta omogućeno je povezivanje:

- *Undefined* signala, odnosno signala povezanih na ulazne portove *Port Module*–a,
- *Allocated*, *Clone* i *PartClone* signala, odnosno signala sa izlaznih portova *Port Module*–a,
- signala unutar drugog *NonPort Module*–a,
- magistrala.

Prilikom povezivanja *Undefined* signala, odnosno signala povezanih na ulazne portove *Port Module*–a, signal dobija izvor vrednosti od *Allocated*, *Clone* ili *PartClone* signala unutar *NonPort Module*–a. U ovom slučaju, otvara se dijalog prikazan na slici 24, gde je *Signal Type*=*Output*, koji obezbeđuje izbor odgovarajućeg signala unutar *NonPort Module*–a.

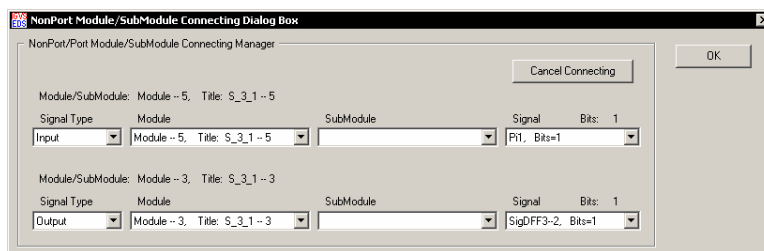


Slika 24. Funkcija za povezivanje signala i *NonPort Module* objekta

Kada se povezuju *Allocated*, *Clone* i *PartClone* signali, odnosno signali sa izlaznih portova *Port Module*–a, tada signal postaje izvor vrednosti za *Undefined* signal unutar *NonPort Module*–a. I u ovom slučaju otvara se dijalog prikazan na slici 24, gde je *Signal Type*=*Input*, koji obezbeđuje izbor odgovarajućeg signala unutar *NonPort Module*–a.

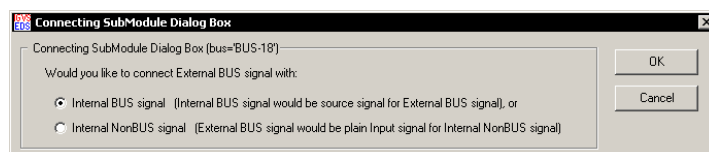
Prilikom povezivanja signala unutar jednog *NonPort Module*–a sa signalom unutar drugog *NonPort Module*–a, otvara se dijalog prikazan na slici 25, koji omogućava izbor signala za povezivanje. Vrednosti lista vrsta signala *Signal Type* moraju da budu različite, jer je

obavezno da signal iz jednog *NonPort Module*–a bude izvor vrednosti za signal iz drugog *NonPort Module*–a.



Slika 25. Funkcija za povezivanje signala dva *NonPort Module* objekta

Za potrebe povezivanja magistrala i signala unutar *NonPort Module*–a, na raspolaganju su dve vrste povezivanja. Stoga se prvo otvara dijalog za izbor vrste povezivanja (slika 26), kada se određuje da li se magistrala povezuje bez prenošenja osobina magistrale na signal unutar *NonPort Module*–a (*Internal NonBUS signal*), ili se magistrala povezuje sa prenošenjem osobina magistrale na signal unutar *NonPort Module*–a (*Internal BUS signal*). Taster *OK* zatvara dijalog sa prihvatanjem vrste povezivanja, dok taster *Cancel* zatvara dijalog sa poništavanjem operacije povezivanja. Nakon izbora vrste povezivanja, otvara se dijalog sa listama odgovarajućih signala unutar *NonPort Module*–a. Ukoliko je odabrana vrsta povezivanja *Internal NonBUS signal*, postavlja se *Signal Type=Input*, i prikazuje se lista *Undefined* signala unutar *NonPort Module*–a. Na ovaj način, magistrala postaje izvor vrednosti za *Undefined* signala unutar *NonPort Module*–a. Ukoliko je odabrana vrsta povezivanja *Internal BUS signal*, postavlja se *Signal Type=Output*, i prikazuje se lista *BUS* signala unutar *NonPort Module*–a. Nakon povezivanja, magistrala unutar *NonPort Module*–a postaje sastavni deo magistrale koja se povezuje na *NonPort Module*. Magistrala van *NonPort Module*–a dobija dodatni status *External BUS*, dok magistrala unutar *NonPort Module*–a dobija dodatni status *Internal BUS*. Obe magistrale se prikazuju i ponašaju kao da postoji samo *External BUS* magistrala.



Slika 26. Dijalog za izbor vrste povezivanja magistrale i *NonPort Module* objekta

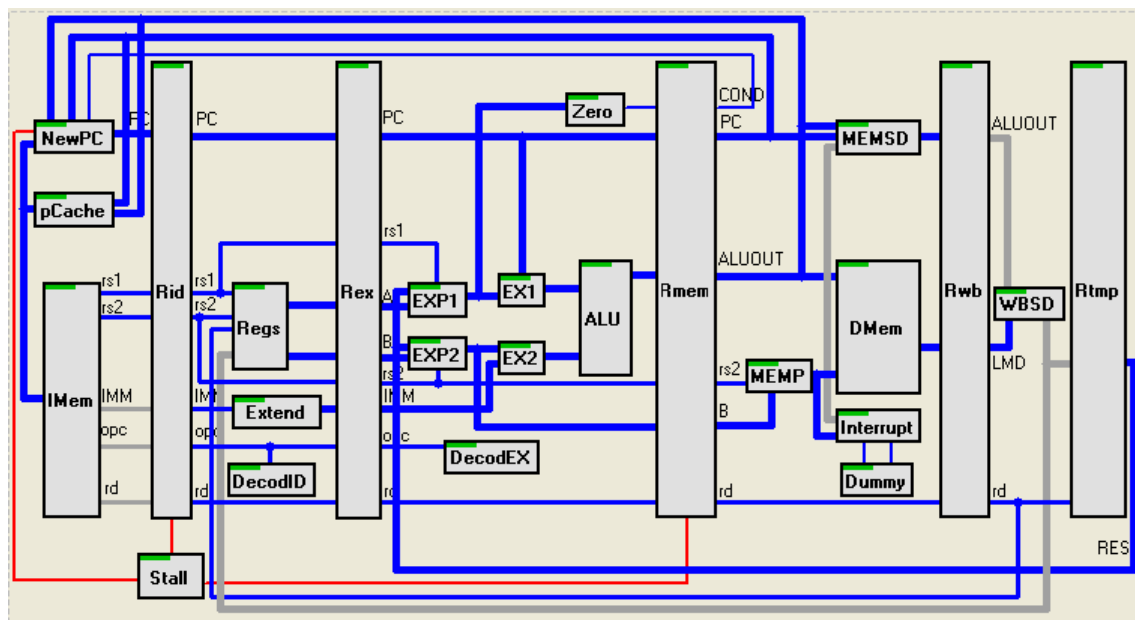
Dijalozi prikazani na slikama 24 i 25 obezbeđuju liste odgovarajućih signala sa kojima je moguće ostvariti povezivanje. Liste signala formiraju se na taj način da su prikazani samo signali sa kojima je moguće ostvariti povezivanje. Stoga, prilikom korišćenja ovih dijaloga, nije potrebno da se vodi računa o tipovima signala, brojevima bitova signala i slično.

5. PROJEKTOVANJE PROCESORA I SIMULATORA

U ovoj glavi prikazan je način projektovanja procesora, uporedo sa prikazom projektovanja simulatora pomoću korišćenog alata. Ovakav način prikaza projektovanja procesora, daje mogućnost da se na konkretnom primeru projektovanja verifikuje korišćeni alat, kao i sam procesor. Sve komponente su realizovane do nivoa prekidačkih mreža.

5.1. BLOK–ŠEMA PROCESORA

U ovom odeljku data je blok-šema procesora. Slika 27 prikazuje jedinice posmatranog procesora i najvažnije veze između njih. Šema se sastoji iz pet *pipeline* registara koji razdvajaju stepene procesora i u kojima se čuva sve što je potrebno za izvršavanje narednog stepena procesora i pet stepeni procesora, koji predstavljaju pet faza u izvršavanju instrukcije: IF, ID, EX, MEM, WB.



Slika 27. Dopunjena blok-šema procesora

5.2. STEPEN IF PIPELINE PROCESORA

Stepen IF (instruction fetch) *pipeline* procesora sastoji se od sledećih jedinica: New PC, PCache, Instruction Memory. *Pipeline* registar koji razdvaja stepene IF i ID označen je sa Rid. U sledećim sekcijama opisane su ove jedinice.

5.2.1. JEDINICA NEWPC

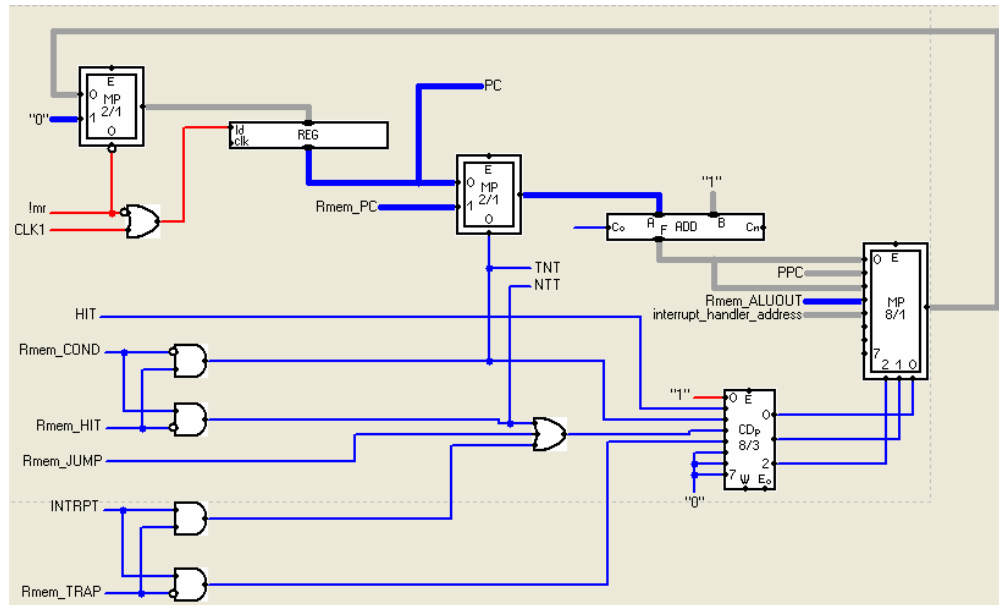
Programski brojač PC i jedinica NewPC, koja vrši izbor nove vrednosti programskog

brojača, prikazani su na slici 28. 32-bitni registar PC kao signal takta koristi signal CLK1 generisan u jedinici Stall. Jedinica NewPC sastoji se od multipleksera MPNEWPC i MPOPC, sabirača ADD, koda prioriteta CD i pratećih logičkih kola. Multiplekser MPNEWPC je 32-bitni multiplekser 8/1, pri čemu se koristi 5 od 8 ulaza. Izlaz ovog multipleksera se na sledeći signal takta upisuje u registar PC. Koder prioriteta CD generiše upravljačke signale za multiplekser MPNEWPC. To je koder 8/3, pri čemu se koristi 5 od 8 ulaza. Neiskorišćeni ulazi koda CD fiksirani su na 0. Što je veća oznaka ulaza u koder to je viši prioritet tog ulaza. Multiplekser MPOPC je 32-bitni multiplekser 2/1, koji propušta na ulaz sabirača ADD vrednost registra PC ili polja $R_{MEM.PC}$, u zavisnosti od signala TNT. Sabirač ADD inkrementira izlaznu vrednost tog multipleksera za 1.

Ukoliko je signal PREKID aktivan a signal $R_{MEM.TRAP}$ neaktivan (pokreće se opsluživanje prekida koji nije izazvan instrukcijom **trap**), na izlaz multipleksera MPNEWPC propušta se adresa prekidne rutine i na sledeći signal takta upisuje u programski brojač. Ulaz 3 koda CD biće aktivan ako se pokreće opsluživanje prekida izazvanog instrukcijom **trap** (aktivni su signali PREKID i $R_{MEM.TRAP}$), ili ako se u stepenu MEM nalazi instrukcija bezuslovnog skoka (aktivno je polje $R_{MEM.JUMP}$), ili instrukcija uslovnog skoka za koju je predviđanje bilo *not taken* a ishod je *taken* (NTT). (Prognoza ishoda skoka prikazana je vrednošću polja $R_{MEM.CHIT}$, a stvarni ishod skoka vrednošću polja $R_{MEM.COND}$.) Tada se na izlaz multipleksera MPNEWPC propušta vrednost $R_{MEM.ALUOUT}$; to je odredišna adresa uslovnog ili bezuslovnog skoka ili instrukcije **trap**. Može se primetiti da mehanizam prekida zahteva vrlo malo dodatnog hardvera u ovoj jedinici, jer se pokretanje prekidne rutine sa stanovišta ove jedinice izvodi na isti način kao bezuslovni skokovi.

Ako je polje $R_{MEM.CHIT}$ aktivno a $R_{MEM.COND}$ neaktivno, to znači da je u stepenu MEM instrukcija uslovnog skoka za koju je predviđanje bilo *taken* a ishod je *not taken* (TNT). U registar PC treba upisati adresu instrukcije koja se nalazi u stepenu MEM ($R_{MEM.PC}$) uvećanu za 1. Ako je aktivan signal CHIT, tj. ako u kešu za predikciju postoji ulaz za instrukciju određenu vrednošću registra PC, na izlaz multipleksera MPNEWPC propušta se vrednost pročitana iz keša za predikciju, PPC – prognozirana odredišna adresa skoka.

Za slučaj da nijedan od ulaza višeg prioriteta nije aktivan, što odgovara sekvencijalnom izvršavanju programa, ulaz 0 koda CD fiksiran je na aktivnu vrednost. Tada je neaktivan signal TNT, multiplekser MPOPC propušta vrednost registra PC. Sabirač ADD inkrementira tu vrednost za 1, a multiplekser MPNEWPC propušta tu inkrementiranu vrednost.



Slika 28. Registar PC i jedinica $NewPC$

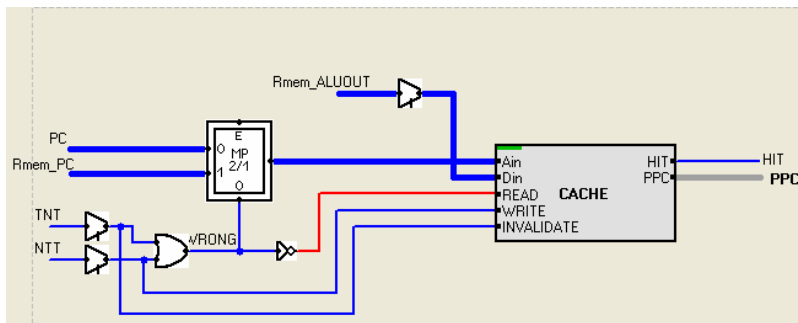
Pravila na osnovu kojih je određen redosled ulaza u koder CD i multiplekser MPNEWPC su sledeća: 1) Koder CD mora davati prednost skokovima i prekidima u odnosu na sekvencijalno izvršavanje programa. U suprotnom bi, umesto da u njega bude upisana npr. odredišna adresa skoka, programski brojač samo bio inkrementiran. 2) Prognozirana odredišna adresa skoka (PPC) mora imati prednost u odnosu na sekvencijalno izvršavanje programa. 3) Instrukcija **trap** sa stanovišta jedinice NewPC ekvivalentna je instrukcijama bezuslovnog skoka. 4) Prekid, izvršena instrukcija skoka (bezuslovni ili uslovni sa pogrešnom predikcijom) i **trap** moraju imati viši prioritet od prognozirane odredišne adrese skoka (PPC) jer se u tim situacijama ispira *pipeline*. 5) Prekid koji nije izazvan instrukcijom **trap** mora imati viši prioritet od instrukcija skoka, jer se može javiti prilikom izvršavanja tih instrukcija. U suprotnom bi umesto adrese prekidne rutine u programski brojač bila upisana odredišna adresa skoka. 6) Međusobni prioritet instrukcija skoka i instrukcije **trap** nije bitan jer se ne mogu istovremeno naći dve takve instrukcije u stepenu MEM.

5.2.2. JEDINICA PCACHE

Jedinica PCache (slika 29) je keš za predikciju odredišta i ishoda uslovnih skokova. Osim same keš memorije, ova jedinica sadrži 32-bitni multiplekser 2/1 MPPCac čiji izlaz se vodi na adresni ulaz keša. Jedinica PCache koristi signale NTT i TNT generisane u jedinici NewPC.

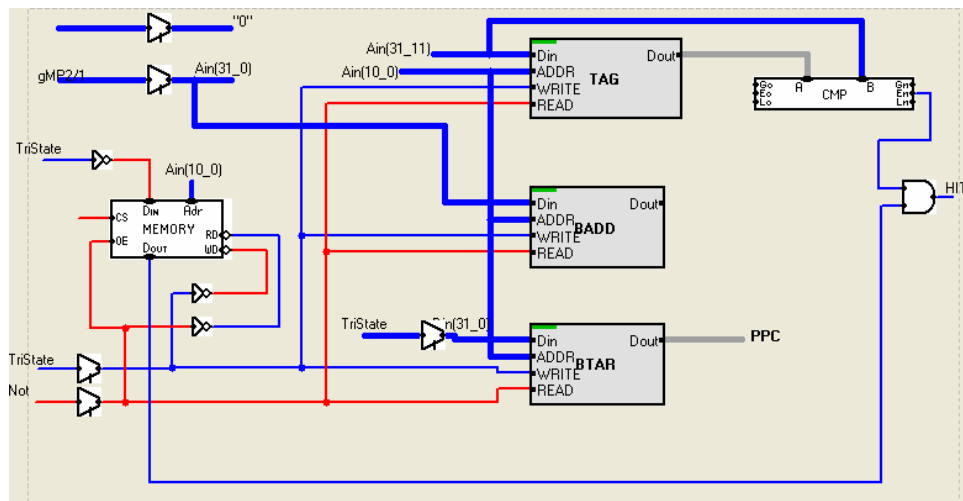
Ako je predviđanje ishoda skoka bilo *taken* a ishod je *not taken*, potrebno je poništiti ulaz u keš memoriji. Zato se signal TNT koristi za invalidaciju ulaza na adresi $R_{MEM.PC}$. Ako je predviđanje ishoda skoka bilo *not taken* a ishod je *taken*, potrebno je kreirati novi ulaz u keš memoriji. Signal NTT koristi se za upis u keš, a adresa ulaza u kešu takođe je $R_{MEM.PC}$. U RAM deo keša upisuje se odredišna adresa skoka iz polja $R_{MEM.ALUOUT}$. Ako nisu aktivni signali TNT ni NTT, proverava se da li u keš memoriji postoji ulaz koji odgovara vrednosti

registra PC. Ako se ostvari pogodak u kešu, biće aktivan signal CHIT (*cache hit*), a prognozirano odredište skoka naći će se na izlazu RAM dela keša – PPC (*predicted PC*).

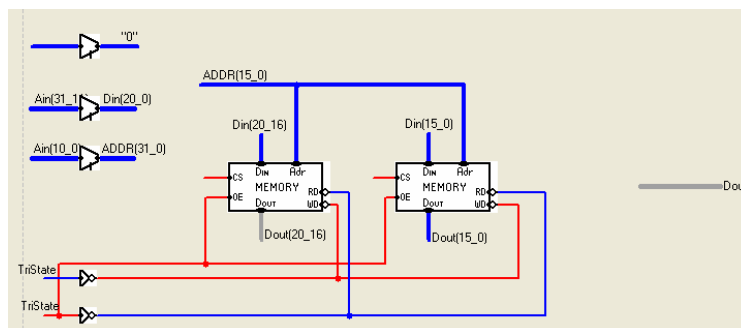


Slika 29. Jedinica *PCache*

Na slici 30 prikazana je jedna od mogućih realizacija keš memorije. Prikazana realizacija predstavlja keš memoriju sa direktnim preslikavanjem, koja ima TAG, BADD i DATA deo. Svi delovi su realizovani uz pomoć modula MEM, koji je prikazan na slici 31.



Slika 30. Jedinica *CACHE*



Slika 31. Jedinica *MEM*

5.2.3. JEDINICA INSTRUCTION MEMORY

Jedinica Instruction Memory (memorija za instrukcije) je analogna jedinici Data Memory. Kao adresa u memoriji za instrukcije se koristi vrednost registra PC. 32-bitni izlazni podatak

nosi oznaku IR; odgovarajući delovi tog podatka upisuju se u polja opcode, rd, rs1, rs2 i IMM *pipeline* registra R_{ID}. Signal za čitanje iz memorije za instrukcije je stalno aktivan. Upis u memoriju za instrukcije je izvan opsega ove knjige.

5.2.4. PIPELINE REGISTAR R_{ID}

Pipeline registar R_{ID} sadrži 64 bita. Signal takta za taj registar je CLK1, a signal za brisanje $\overline{mr2}$, koje generiše jedinica Stall. Polja od kojih se sastoji *pipeline* registar R_{ID} opisana su u tabeli 4. IR je oznaka za izlazni podatak jedinice Instruction Memory, pročitana sa adrese određene vrednošću programskog brojača.

Polje	Broj bita	Biti	Izvor	Opis
VALID	1	0	Fiksno 1	Sadržaj <i>pipeline</i> registra je validan
CHIT	1	1	PCache	Predikcija ishoda uslovnog skoka
PC	32	2-33	Registar PC	Vrednost programskog brojača
rs1	5	34-38	IR _{17..13}	Adresa prvog izvorišnog registra
rs2	5	39-43	IR _{12..8}	Adresa drugog izvorišnog registra
IMM	8	44-51	IR _{7..0}	Neposredni argument
opcode	7	52-58	IR _{29..23}	Kod operacije
rd	5	59-63	IR _{22..18}	Adresa odredišnog registra

Tabela 4. Opis polja koja sačinjavaju *pipeline* registar R_{ID}

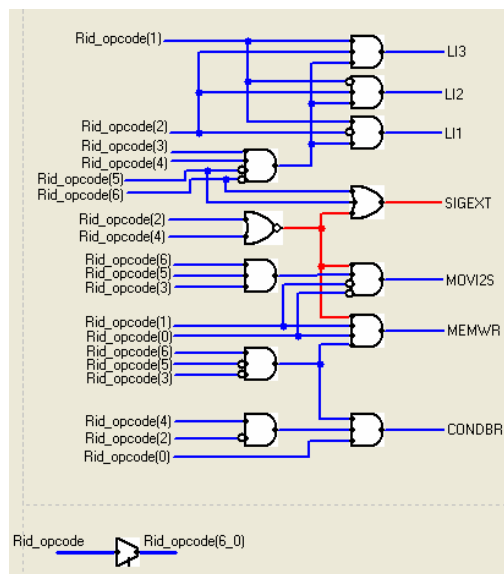
5.3. STEPEN ID PIPELINE PROCESORA

U procesorima klasične organizacije dekodovanje operacija najčešće obavlja jedna jedinica, koja na osnovu koda operacije generiše potrebne upravljačke signale. Međutim, kod *pipeline* procesora pri takvom načinu dekodovanja uz svaku instrukciju bi se u *pipeline* registrima morali voditi svi upravljački signali generisani pri dekodovanju operacije. Zato se u posmatranom procesoru dekodovanje vrši jednim delom u stepenu ID a drugim u stepenu EX.

Stepen ID (instruction decode and register fetch) *pipeline* procesora sastoji se od sledećih jedinica: DecodID, Registers, Extend. *Pipeline* registar koji razdvaja stepene ID i EX označen je sa Rex. U sledećim sekcijama opisane su ove jedinice.

5.3.1. JEDINICA DECODID

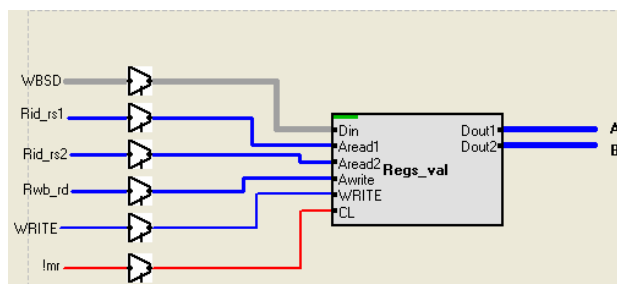
Jedinica DecodID (slika 32) vrši dekodovanje operacija u stepenu ID. Pri tome generiše upravljačke signale potrebne u fazi ID i upravljačke signale potrebne na početku faze EX, takve da bi generisanje tih signala u fazi EX moglo zahtevati produženje periode signala takta. Pošto treba da generiše mali broj signala, jedinica DecodID realizovana je kao kombinaciona mreža koja prepoznaje odgovarajuće vrednosti kodova operacije.



Slika 32. Jedinica *DecodID*

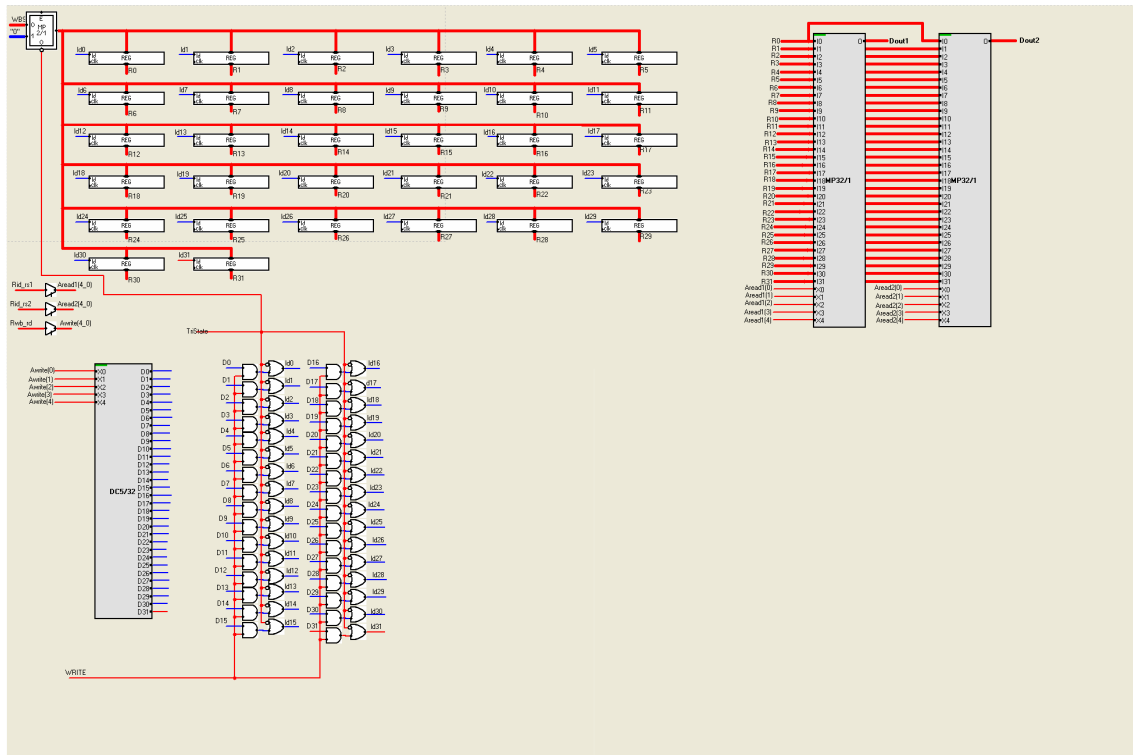
5.3.2. JEDINICA REGISTERS

Jedinica Registers (slika 33) je registar fajl koji se sastoji od 32 registra veličine po 32 bita. Ova jedinica omogućava istovremeno čitanje iz dva registra i upis u treći. Adrese registara su veličine 5 bita. Podaci pročitani iz registara čije se adrese nalaze u poljima $R_{ID}.rs1$ i $R_{ID}.rs2$ se upisuju u polja $R_{EX}.A$ i $R_{EX}.B$, respektivno. Signal za čitanje iz registar fajla je stalno aktivan. Podatak za upis dobija se iz jedinice WBSD, kao i signal za upis WRITE, a adresa registra u koji se taj podatak upisuje nalazi se u polju $R_{WB}.rd$. Interna organizacija registar fajla je izvan opsega ove knjige.



Slika 33. Jedinica *Registers*

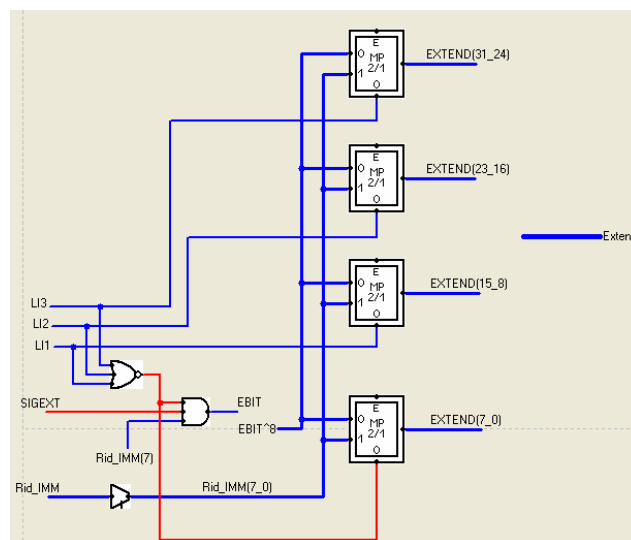
Kako bi realizacija ovakve jedinice bila moguća u simulatoru konfigurabilnih računarskih sistema, kao što je ovaj u kome se realizuje *pipeline* simulator, neophodno je realizovati mrežu od 32 registra, kao što je prikazano na slici 34, kako bi bila omogućena zahtevana funkcionalnost da se upisuje u jedan registar i istovremeno čita iz dva registra. Selekcija registara vrši se pomoću odgovarajućih kombinacionih mreža, koje su na slici 34 označene sa DC5/32 i MP32/1, a koji su takođe morali biti realizovani pomoću osnovnih kombinacionih kola.



Slika 34. Jedinica *Regs_val*

5.3.3. JEDINICA EXTEND

Jedinica Extend (slika 35) vrši proširivanje 8-bitnog neposrednog argumenta instrukcije do 32 bita, nulom ili znakom. Prilikom izvršavanja instrukcija za punjenje registra opšte namene konstantom (*load immediate*) ova jedinica prepisuje ulazni neposredni argument u zadati bajt izlazne veličine, uz popunjavanje ostalih bajtova nulom. Jedinica Extend sastoji se od 8-bitnih multipleksera 2/1 MP0, MP1, MP2 i MP3 i pratećih logičkih kola.



Slika 35. Jedinica *Extend*

5.3.4. PIPELINE REGISTAR R_{EX}

Pipeline registar R_{EX} sastoji se od 155 bita. Signal takta za taj registar je globalni takt CLK, a signal za brisanje $\overline{mr3}$ generiše jedinica Stall. Polja tog registra opisana su u tabeli 5.

Polje	Broj bita	Biti	Izvor	Opis
VALID	1	0	R _{ID} .VALID	Sadržaj <i>pipeline</i> registra je validan
CHIT	1	1	R _{ID} .CHIT	Predikcija ishoda uslovnog skoka
PC	32	2-33	R _{ID} .PC	Vrednost programskog brojača
rs1	5	34-38	R _{ID} .rs1	Adresa prvog izvorišnog registra
A	32	39-70	Registers.Dout1	Vrednost prvog izvorišnog registra
B	32	71-102	Registers.Dout2	Vrednost drugog izvorišnog registra
rs2	5	103-107	R _{ID} .rs2	Adresa drugog izvorišnog registra
IMM	32	108-139	Extend	Prošireni neposredni argument
opcode	7	140-146	R _{ID} .opcode	Kod operacije
MEMWR	1	147	DecodID	Upis u Data Memory
MOVI2S	1	148	DecodID	Upis u registar PSW
CONDBR	1	149	DecodID	Izvršava se instr. uslovnog skoka
rd	5	150-154	R _{ID} .rd	Adresa odredišnog registra

Tabela 5. Opis polja koja sačinjavaju *pipeline* registar R_{EX}

5.4. STEPEN EX PIPELINE PROCESORA

Stepen EX (execute and effective address calculation) *pipeline* procesora sastoji se od sledećih jedinica: EXP1, EXP2, EX1, EX2, ALU, Zero i DecodEx. *Pipeline* registar koji razdvaja stepene EX i MEM označen je sa R_{mem}. U sledećim sekcijama opisane su ove jedinice.

5.4.1. JEDINICA EXP1

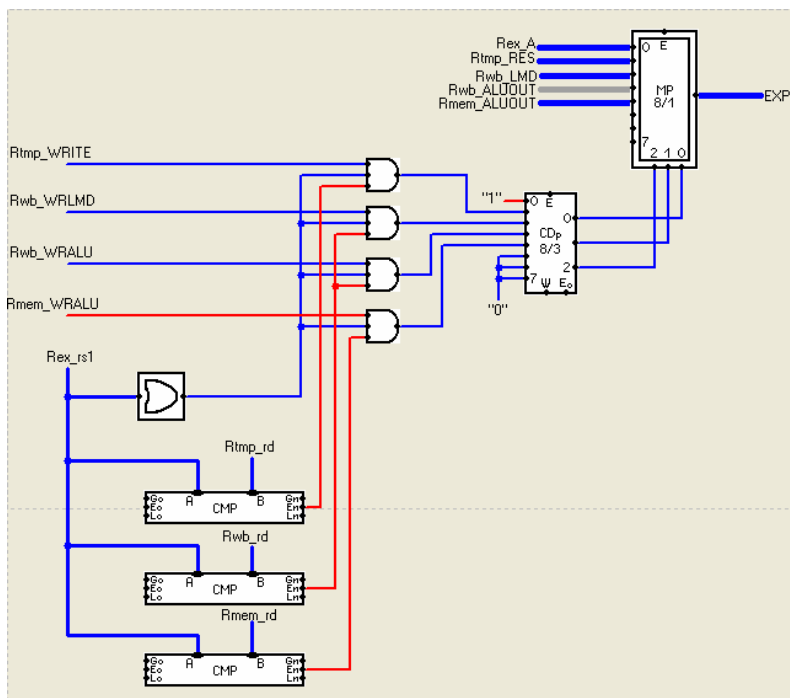
Jedinica EXP1 (slika 36) je prvi deo logike za prosleđivanje u stepenu EX. Sastoji se od multipleksera MPEXP1, koda prioriteta CD, komparatora CMP1, CMP2 i CMP3 i pratećih logičkih kola. Prosleđivanje podatka iz *pipeline* registara R_{MEM}, R_{WB} ili R_{TMP} na izlaz jedinice EXP1 se vrši ako su zadovoljeni svi sledeći uslovi: 1) Adresa odredišnog registra instrukcije u odgovarajućem *pipeline* registru (R_{MEM}.rd, R_{WB}.rd, odnosno R_{TMP}.rd) je jednaka adresi prvog izvorišnog registra instrukcije u *pipeline* registru R_{EX} (R_{EX}.rs1). 2) R_{EX}.rs1 je različito od nule. 3) U odgovarajućem polju *pipeline* registra R_{MEM}, R_{WB}, odnosno R_{TMP} nalazi se vrednost koja se upisuje u registar fajl, to jest konačni rezultat odgovarajuće instrukcije. Ako bilo koji od ovih uslova nije zadovoljen, nema prosleđivanja i na izlaz jedinice EXP1 propušta se vrednost R_{EX}.A.

Multiplekser MPEXP1 je 32-bitni multiplekser 8/1, pri čemu se koristi 5 od 8 ulaza. Koder

prioriteta CD generiše upravljačke signale za multiplekser MPEXP1. To je koder 8/3, pri čemu se koristi 5 od 8 ulaza. Neiskorišćeni ulazi kodera CD fiksirani su na 0. Što je veća oznaka ulaza u koder to je viši prioritet tog ulaza. Komparatori CMP1, CMP2 i CMP3 proveravaju jednakost vrednosti $R_{EX}.rs1$ sa $R_{TMP}.rd$, $R_{WB}.rd$ i $R_{MEM}.rd$, respektivno. Izlazni signali ovih komparatora nose oznake EQ1, EQ2 i EQ3, respektivno. 5-ulazno ILI-kolo proverava da li je vrednost $R_{EX}.rs1$ različita od 0, i u tom slučaju aktivan je signal RNEZ.

Ako su aktivni signali RNEZ, EQ3 i $R_{MEM}.WRALU$ (što znači da je $R_{EX}.rs1 \neq 0$, $R_{MEM}.rd = R_{EX}.rs1$, i u stepenu MEM se nalazi neka instrukcija koja će upisati vrednost $R_{MEM}.ALUOUT$ u registar fajl), multiplekser će propustiti vrednost $R_{MEM}.ALUOUT$. Ako je $R_{EX}.rs1 \neq 0$, $R_{WB}.rd = R_{EX}.rs1$, i u stepenu WB se nalazi neka instrukcija koja upisuje vrednost $R_{WB}.ALUOUT$ u registar fajl, biće aktivni signali RNEZ, EQ2 i $R_{WB}.WRALU$, i multiplekser će propustiti vrednost $R_{WB}.ALUOUT$. Ako su aktivni signali RNEZ, EQ2 i $R_{WB}.WRLMD$ (tj. ako je $R_{EX}.rs1 \neq 0$, $R_{WB}.rd = R_{EX}.rs1$, i u stepenu WB se nalazi instrukcija **lw**), multiplekser će propustiti vrednost $R_{WB}.LMD$.

Ako je $R_{EX}.rs1 \neq 0$, $R_{TMP}.rd = R_{EX}.rs1$, i u *pipeline* registru R_{TMP} se nalazi neka instrukcija koja je upisala rezultat u registar fajl, biće aktivni signali RNEZ, EQ1 i $R_{TMP}.WRITE$. Zbog toga će multiplekser propustiti vrednost $R_{TMP}.RES$. Za slučaj da nijedan od prethodnih uslova nije zadovoljen (što znači da nema prosleđivanja), ulaz 0 kodera CD fiksiran je na vrednost 1. To dovodi do propuštanja vrednosti $R_{EX}.A$ na izlaz mutipleksera MPEXP1.



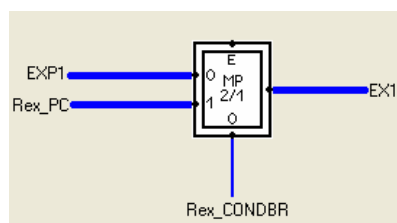
Slika 36. Jedinica EXP1

5.4.2. JEDINICA EXP2

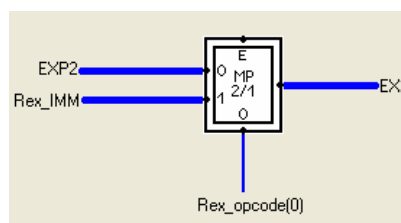
Jedinica EXP2 je drugi deo logike za prosleđivanje u stepenu EX. Ova jedinica je organizovana na potpuno isti način kao jedinica EXP1. Razlika je samo u tome što se umesto $R_{EX.rs1}$ koristi $R_{EX.rs2}$, i umesto $R_{EX.A}$ koristi se $R_{EX.B}$. Izlazni 32-bitni podatak nosi oznaku EXP2, koristi se u jedinici EX2 i upisuje u polje $R_{MEM.B}$.

5.4.3. JEDINICA EX1

Jedinica EX1 (slika 37) omogućuje izbor vrednosti za ulaz A jedinice ALU. Ovu jedinicu čini multiplexer MPEX1, 32-bitni multiplexer 2/1. Upravljački signal tog multiplexera, $R_{EX.CONDBR}$, aktivan je kada se u stepenu EX nalazi instrukcija uslovnog skoka. U tom slučaju će multiplexer propustiti vrednost $R_{EX.PC}$, a u suprotnom vrednost EXP1.



Slika 37. Jedinica EX1



Slika 38. Jedinica EX2

5.4.4. JEDINICA EX2

Jedinica EX2 (slika 38) omogućuje izbor vrednosti za ulaz B jedinice ALU. Ovu jedinicu čini 32-bitni multiplexer 2/1 MPEX2. Upravljački signal tog multiplexera je $R_{EX.opcode_0}$, najmlađi bit koda operacije instrukcije u stepenu EX. To je *immediate* bit, koji je aktivan ako instrukcija koristi neposredni argument.

5.4.5. JEDINICA ALU

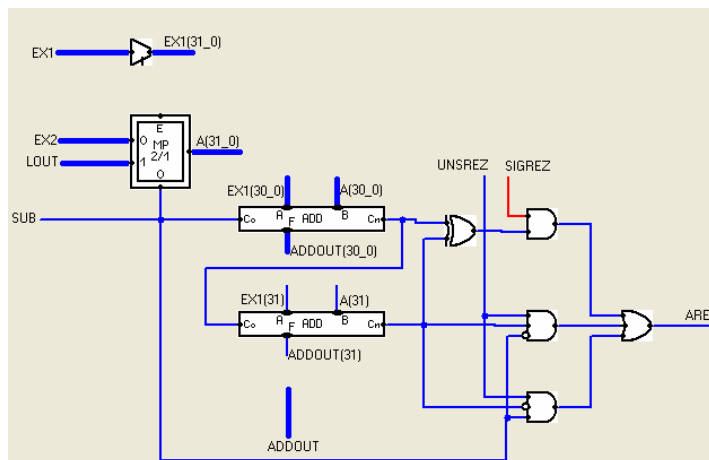
Jedinica ALU (aritmetičko-logička jedinica) se sastoji od blokova ADD, LOG, CMP i SHIFT, koji obavljaju aritmetičke, logičke, relacione i pomeračke operacije, respektivno, i bloka OUT, koji bira izlaznu vrednost jedinice ALU između izlaznih vrednosti ostala 4 bloka.

5.4.5.1. BLOK ADD

Blok ADD (slika 39) sastoji se od 32-bitnog sabirača ADD, 32-bitnog multiplexera 2/1 MPADD i pratećih logičkih kola. Ako je aktivan signal SUB (vrši se oduzimanje), multiplexer MPADD propušta rezultat bloka LOG, koji u tom slučaju izračunava komplement jedinice za vrednost EX2. Tada ulazni prenos sabirača (C_{in}) ima vrednost 1. Treba primetiti da se na isti način izvršava oduzimanje i sa i bez znaka. Ako signal SUB nije aktivan, multiplexer propušta vrednost EX2, a ulazni prenos sabirača ima vrednost 0.

Aktivan signal SIGREZ pokazuje da treba detektovati prekoračenje prilikom sabiranja ili oduzimanja sa znakom. U tom slučaju prekoračenje se prepoznaje po tome što se prenos iz

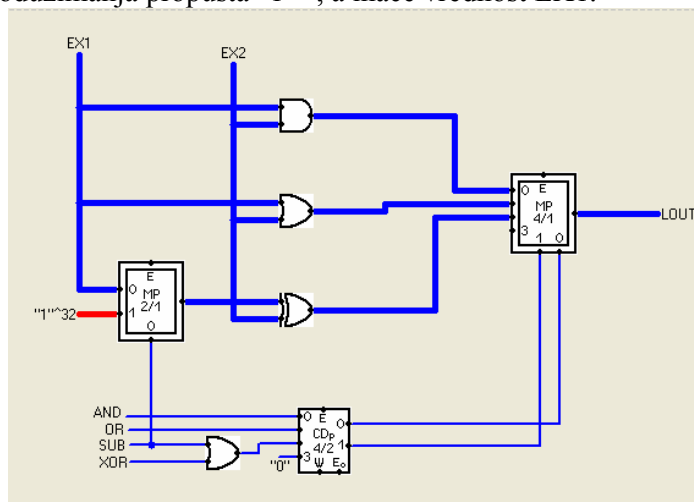
najstarijeg razreda (Cout) razlikuje od prenosa iz prvog mlađeg razreda (C30). Ako je aktivan signal UNSREZ, treba detektovati prekoračenje prilikom sabiranja ili oduzimanja bez znaka. Kod sabiranja bez znaka prekoračenje se prepoznaje po tome što postoji prenos iz najstarijeg razreda. U slučaju oduzimanja bez znaka prekoračenje se prepoznaje po tome što NE postoji prenos iz najstarijeg razreda. Ukoliko se prepozna prekoračenje na bilo koji od opisanih načina, generiše se signal zahteva za prekid ARE.



Slika 39. Blok *ADD* jedinice *ALU*

5.4.5.2. BLOK LOG

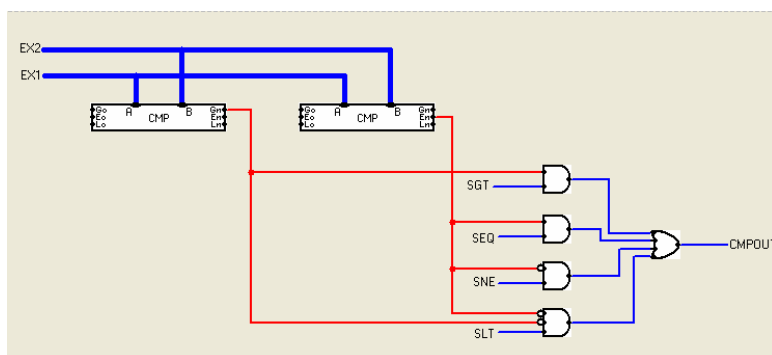
Blok LOG (slika 40) realizuje logičke operacije "i", "ili" i "ekskluzivno ili" na nivou bita, a osim toga učestvuje u operaciji oduzimanja pošto generiše komplement jedinice za veličinu koja se oduzima. Ovaj blok sastoji se od mreža AND, OR i XOR, koje obavljaju logičke operacije "i", "ili" i "ekskluzivno ili", respektivno, na nivou bita, nad 32-bitnim veličinama. 32-bitni multiplekser 4/1 MPLOG bira izlaznu veličinu ovog bloka. 32-bitni multiplekser 2/1 MPXOR u slučaju oduzimanja propušta "1"³², a inače vrednost EX1.



Slika 40. Blok *LOG* jedinice *ALU*

5.4.5.3. BLOK CMP

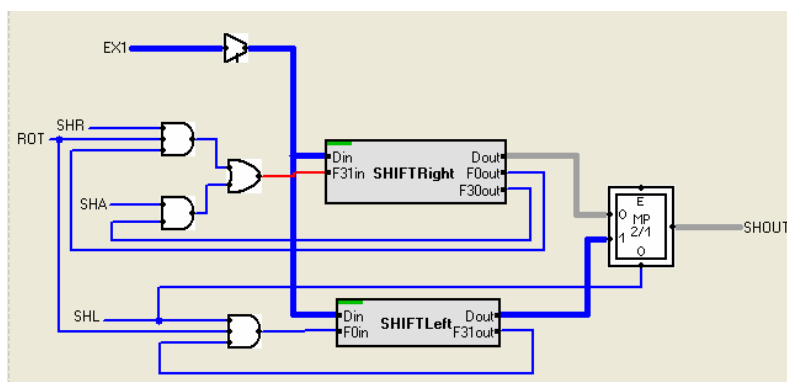
Blok CMP (slika 41) određuje rezultate relacionih operacija (operacija poređenja). Sastoji se od komparatora CMPGT i CMPEQ i pratećih logičkih kola. Ne postoje upravljački signali za svih 6 relacija podržanih u arhitekturi, već se relacije "manje ili jednako" i "veće ili jednako" realizuju kombinovanjem relacije "jednako" sa relacijama "manje" i "veće", respektivno. Osim toga, relacija "nije jednako" predstavlja negaciju relacije "jednako", a relacija "manje" negaciju unije relacija "veće" i "jednako", tako da se umesto 4 koriste samo 2 komparatora.



Slika 41. Blok *CMP* jedinice *ALU*

5.4.5.4. BLOK SHIFT

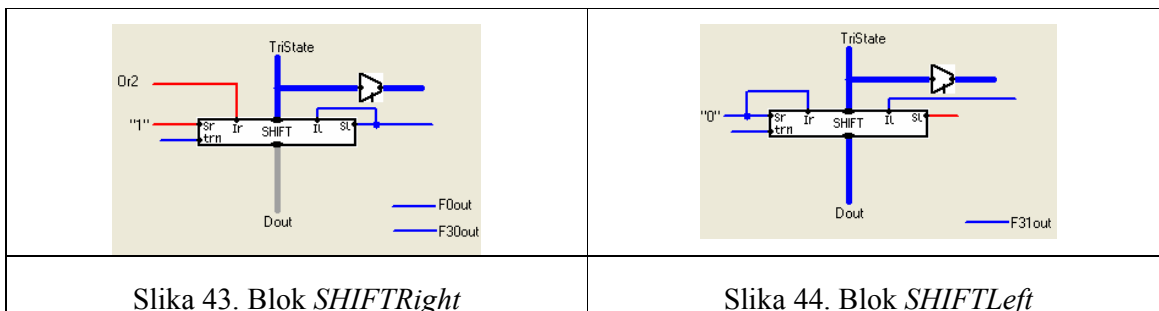
Blok SHIFT (slika 42) realizuje operacije pomeranja i rotiranja za jedno mesto ulevo ili udesno. Sastoji se od mreža SHIFLeft i SHIFTRight, multipleksera MPSH i pratećih logičkih kola. SHIFLeft i SHIFTRight su jednostavne ožičene mreže koje realizuju pomeranje 32-bitne veličine za jedno mesto ulevo ili udesno, respektivno. Na ulaze obe mreže dovodi se vrednost EX1. Multiplekser MPSH je 32-bitni multiplekser 2/1 koji na izlaz bloka SHIFT propušta rezultat mreže SHIFLeft ili SHIFTRight u zavisnosti od smeru pomeračke operacije.



Slika 42. Blok *SHIFT* jedinice *ALU*

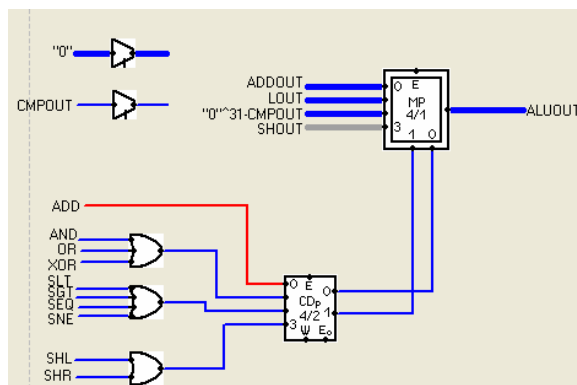
Jedinice SHIFLeft (slika 43) i SHIFTRight (slika 44) su morale biti realizovane u ovom simulatoru na određeni način pomoću osnovnih logičkih kola. Autor je kao jednostavno

rešenje izabrao da upotrebi gotovu kombinacionu komponentu, koja ima mogućnost pomeranja u levo ili u desno i koja je označena sa SHIFT. Da bi ove dve jedinice ispravno funkcionisale, povezivanje jedinice SHIFT je urađeno, kao na slikama 43 i 44, respektivno, za SHIFTRight, odnosno SHIFTRight.



5.4.5.5. BLOK OUT

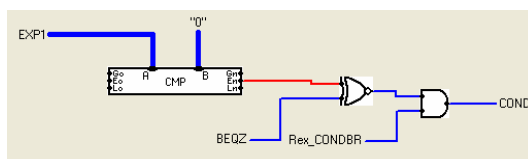
Blok OUT (slika 45) propušta rezultat bloka ADD, LOG, CMP ili SHIFT na izlaz jedinice ALU u zavisnosti od operacije koja se izvršava. Blok OUT sastoji se od multipleksera MPOUT, koodera prioriteta CD i pratećih logičkih kola.



Slika 45. Blok *OUT* jedinice *ALU*

5.4.6. JEDINICA ZERO

Jedinica Zero (slika 46) proverava da li je ispunjen uslov za uslovni skok. Sastoji se od kombinacione mreže EQ0, koja proverava jednakost 32-bitnog podatka sa nulom, i pratećih logičkih kola.

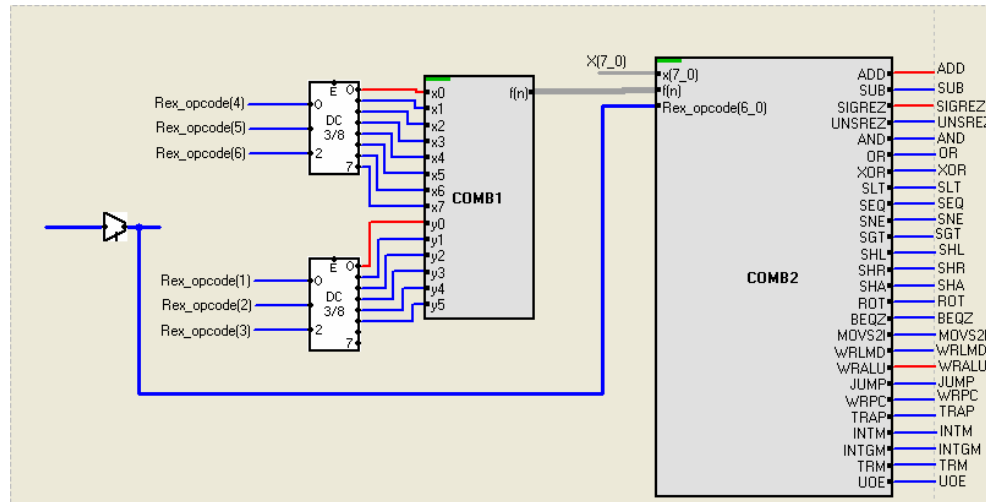


Slika 46. Jedinica *Zero*

5.4.7. JEDINICA DECODEX

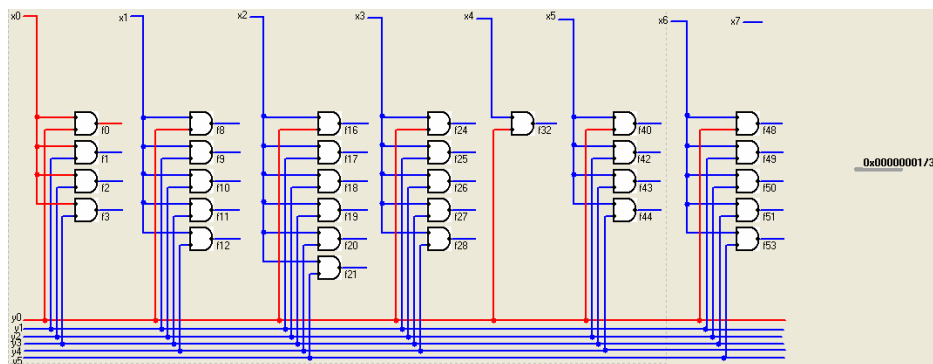
Jedinica DecodEX (slika 47) obavlja dekodovanje operacije u stepenu EX i pritom

generiše upravljačke signale koji se koriste u fazama EX, MEM i WB. Ta jedinica sastoji se od dva dekodera 3/8 i dve kombinacione mreže. Dekoder DCX generiše signale x_i , gde je $i = 0, 1, \dots, 7$; signal x_i aktivan je ako i samo ako je i decimalna vrednost polja $R_{EX}.opcode_{6..4}$. Dekoder DCY generiše signale y_i , gde je $i = 0, 1, \dots, 5$; signal y_i aktivan je ako i samo ako je i decimalna vrednost polja $R_{EX}.opcode_{3..1}$. Kombinaciona mreža KOMB1 na osnovu x_i i y_i generiše signale f_n ; signal f_n aktivan je samo ako je n decimalna vrednost najstarijih 6 bita koda operacije, tj. $R_{EX}.opcode_{6..1}$. Kombinaciona mreža KOMB2 na osnovu x_i , f_n i koda operacije generiše izlazne upravljačke signale.



Slika 47. Jedinica *DecodeX*

Kako bi Ova jedinica ispravno funkcionisala, autor je u sklopu simulatora, morao da dizajnira kombinacione module COMB1 (slika 48) i COMB2 (slika 49). Modul COMB1 je realizovan korišćenjem odgovarajućeg broja dvoulaznih I kola, koja su povezana u kombinacionu mrežu, tako da od x_i i y_i generiše signale f_n .



Slika 48. Blok *COMB1* jedinice *DecodeX*

Modul COMB2 predstavlja kombinacionu mrežu, realizovanu na način prikazan na slici 49, koja na osnovu x_i , f_n i koda operacije generiše izlazne upravljačke signale.

ostati nepromenjen; u tom delu su vrednost programskog brojača ($R_{MEM}.PC$) i rezultat jedinice ALU ($R_{MEM}.ALUOUT$) za instrukciju koja se nalazila u stepenu MEM u taktu u kome je došlo do prekida. Vrednosti koje se na taj signal takta upisuju u deo 2 će dovesti do upisa adrese povratka iz prekida u registar R31. Upis u R31 dogodiće se kada se ta fiktivna "instrukcija" nađe u stepenu WB.

Polje	Broj bita	Biti	Izvor	Opis
VALID	1	0	$R_{EX}.VALID$	Sadržaj <i>pipeline</i> registra je validan
CHIT	1	1	$R_{EX}.CHIT$	Predikcija ishoda uslovnog skoka
COND	1	2	Zero	Da li je ispunjen uslov za uslovni skok
ARE	1	3	ALU	Zahtev za prekid tipa ARE
rs2	5	4-8	$R_{EX}.rs2$	Adresa drugog izvorišnog registra
B	32	9-40	EXP2	Vrednost drugog izvorišnog registra
ENDIS	1	41	$R_{EX}.IMM_0$	Najmlađi bit neposrednog argumenta
MEMWR	1	42	$R_{EX}.MEMWR$	Upis u Data Memory
MOVI2S	1	43	$R_{EX}.MOVI2S$	Upis u registar PSW
MOVS2I	1	44	DecodEX	Upisati u registar fajl sadržaj registra PSW
WRLMD	1	45	DecodEX	Upisati u reg. fajl podatak iz Data Memory
JUMP	1	46	DecodEX	Bezuslovni skok
TRAP	1	47	DecodEX	Zahtev za prekid tipa TRAP
INTM	1	48	DecodEX	Postavljanje bita INTM
INTGM	1	49	DecodEX	Postavljanje bita INTGM
TRM	1	50	DecodEX	Postavljanje bita TRM
UOE	1	51	DecodEX	Zahtev za prekid tipa UOE

Tabela 8. Opis polja koja sačinjavaju deo 3 *pipeline* registra R_{MEM}

5.5. STEPEN MEM PIPELINE PROCESORA

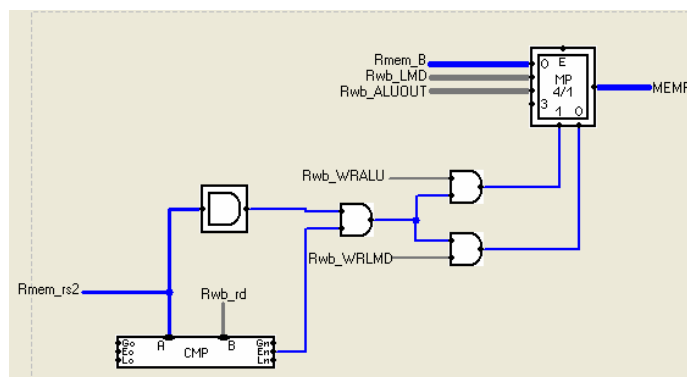
Stepen MEM (memory access and branch completion) *pipeline* procesora sastoji se od sledećih jedinica: MEMP, MEMSD, Interrupt i Data Memory. *Pipeline* registar koji razdvaja stepene MEM i WB označen je sa Rwb. U sledećim sekcijama opisane su ove jedinice.

5.5.1. JEDINICA MEMP

Jedinica MEMP (slika 50) je logika za prosleđivanje u stepenu MEM. Sastoji se od 32-bitnog multipleksera 4/1 MPMEMP, komparatora CMP i pratećih logičkih kola. Prosleđivanje iz *pipeline* registra R_{WB} u stepen MEM se vrši ako su zadovoljeni svi sledeći uslovi: 1) Adresa odredišnog registra instrukcije u *pipeline* registru R_{WB} ($R_{WB}.rd$) je jednaka adresi drugog izvorišnog registra instrukcije u *pipeline* registru R_{MEM} ($R_{MEM}.rs2$). 2) $R_{MEM}.rs2$ je različito od nule. 3) U odgovarajućem polju *pipeline* registra R_{WB} nalazi se vrednost koja se upisuje u registar fajl, to jest konačni rezultat instrukcije koja se nalazi u tom *pipeline*

registru. Ako bilo koji od ovih uslova nije zadovoljen, nema prosleđivanja i na izlaz jedinice MEMP propušta se vrednost $R_{MEM}.B$.

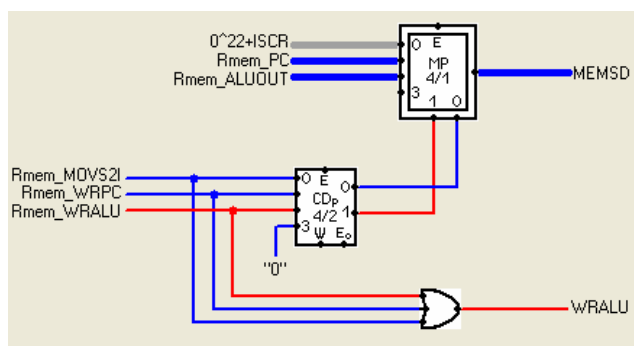
Ako je $R_{MEM}.rs2 \neq 0$, $R_{WB}.rd = R_{MEM}.rs2$ i aktivan je signal $R_{WB}.WRALU$, multiplexer će propustiti vrednost $R_{WB}.ALUOUT$. Ako je $R_{MEM}.rs2 \neq 0$, $R_{WB}.rd = R_{MEM}.rs2$ i aktivan je signal $R_{WB}.WRLMD$, multiplexer će propustiti vrednost $R_{WB}.LMD$. Ako nijedna od prethodnih kombinacija uslova nije zadovoljena (nema prosleđivanja), na izlaz multiplexera će biti propuštena vrednost $R_{MEM}.B$. Pri normalnom radu ne može se dogoditi da signali $R_{WB}.WRALU$ i $R_{WB}.WRLMD$ budu istovremeno aktivni, tako da međusobni prioritet ta dva signala u ovoj jedinici nije bitan.



Slika 50. Jedinica *MEMP*

5.5.2. JEDINICA MEMSD

Jedinica MEMSD (slika 51) određuje vrednost koja se upisuje u polje $R_{WB}.ALUOUT$. Time ova jedinica omogućava izvršavanje instrukcija jsr i movs2i bez povećavanja pipeline registra R_{WB} . Ovu jedinicu čini 32-bitni multiplexer 4/1 MPMEMSD i prateća logička kola. U jedinici MEMSD se signali $R_{MEM}.WRPC$ i $R_{MEM}.MOVS2I$ spajaju sa signalom $R_{MEM}.WRALU$, i pod zajedničkom oznakom $WRALU$ upisuju u $R_{WB}.ALUOUT$.

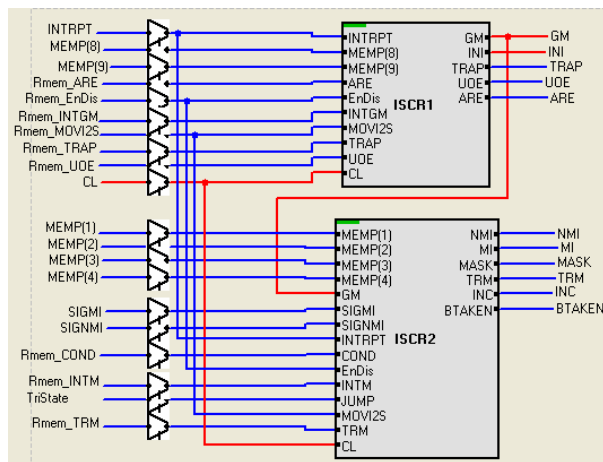


Slika 51. Jedinica *MEMSD*

5.5.3. JEDINICA INTERRUPT

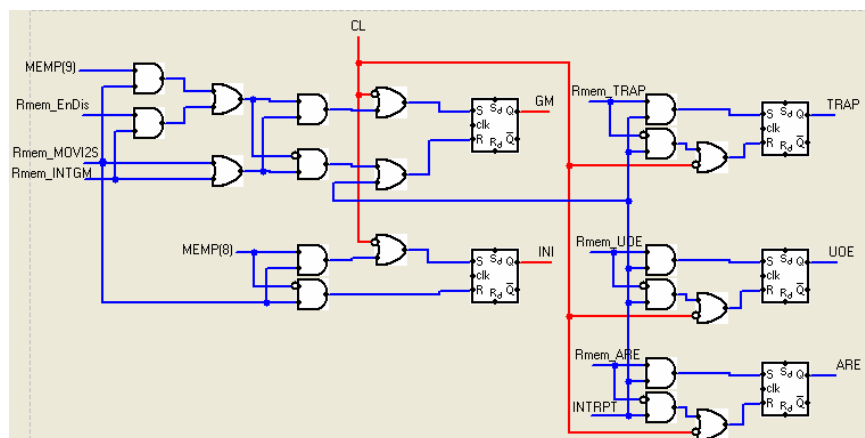
Jedinica Prekid (slika 52) sastoji se od registra *PSW* (*Program Status Word* – programska statusna reč) i logike koja postavlja vrednost signala *PREKID*.

U slučaju simulatora, koji je razvio autor, bilo je potrebno realizovati registar ISCR, kao što je prikazano na slici 53.

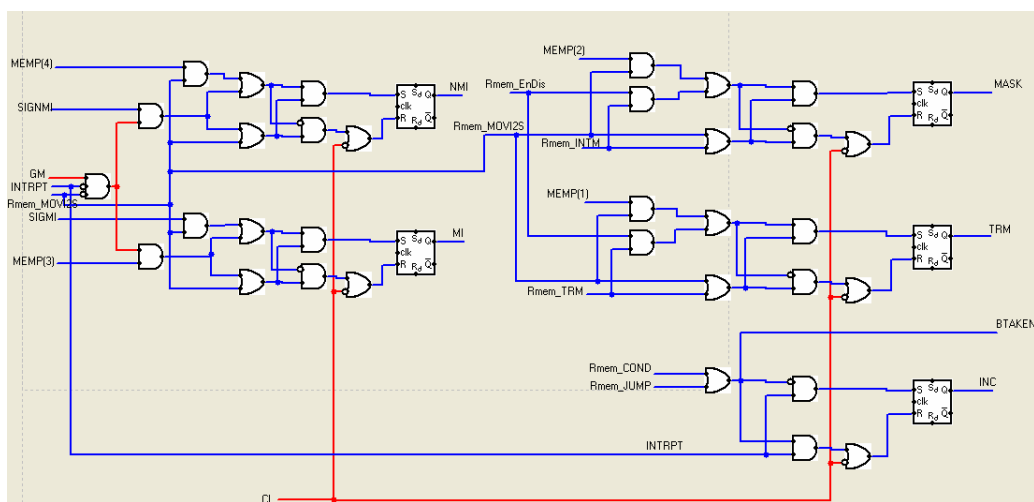


"Registar" PSW realizovan je kao blok od 10 flip-flopova sa pratećim logičkim kolima (slike 54 i 55). Time je omogućeno zasebno postavljanje svakog od bita registra PSW, posebnim instrukcijama i/ili spoljnim signalima. Pošto svih 10 flip-flopova ima isti signal takta CLK i signal za brisanje \overline{mr} , i postoje instrukcije za čitanje svih bita i upis svih programski upisivih bita odjednom, može se smatrati da tih 10 flip-flopova logički

sačinjavaju registar.



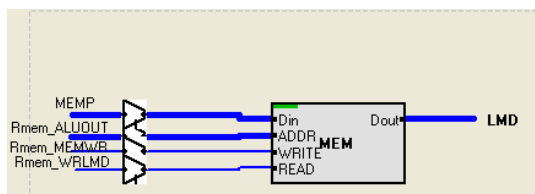
Slika 54. Blok *ISCR1* jedinice Prekid



Slika 55. Blok *ISCR2* jedinice Prekid

5.5.4. JEDINICA DATA MEMORY

Jedinica Data Memory (memorija za podatke, slika 56) omogućava čitanje ili upis 32-bitnog podatka na zadatu 32-bitnu adresu. Ulazni podatak za upis u memoriju dobija se iz logike za prosledjivanje MEMP, a pročitani izlazni podatak upisuje u polje $R_{WB.LMD}$. Adresa za čitanje ili upis se nalazi u polju $R_{MEM.ALUOUT}$. Ako je uključeno polje $R_{MEM.MEMWR}$ biće izvršen upis, a ako je uključeno polje $R_{MEM.WRLMD}$ biće izvršeno čitanje. Interna organizacija memorije za podatke je izvan opsega ove knjige.



Slika 56. Jedinica *Data Memory*

5.5.5. PIPELINE REGISTAR RWB

Pipeline registar R_{WB} sadrži 71 bit. Signal takta za taj registar je globalni signal takta CLK, a signal za brisanje globalni *master reset* signal \overline{mr} . Polja od kojih se sastoji *pipeline* registar R_{WB} opisana su u tabeli 9.

Polje	Broj bita	Biti	Izvor	Opis
ALUOUT	32	0-31	MEMSD	Rezultat jed. ALU ili vrednost pos. registra
WRALU	1	32	MEMSD	Upisati u reg. fajl podatak iz polja ALUOUT
LMD	32	33-64	Data Memory	Podatak pročitani iz jedinice Data Memory
WRLMD	1	65	$R_{MEM}.WRLMD$	Upisati u reg. fajl podatak iz polja LMD
rd	5	66-70	$R_{MEM}.rd$	Adresa odredišnog registra

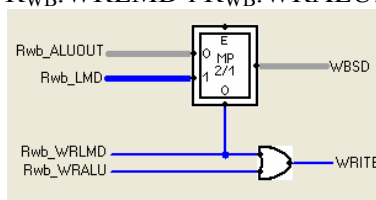
Tabela 9. Opis polja koja sačinjavaju *pipeline* registar R_{WB}

5.6. STEPEN WB PIPELINE PROCESORA

Stepen MEM (memory access and branch completion) *pipeline* procesora sastoji se od jedinice: WBSD. *Pipeline* registar koji se nalazi na izlazu stepena WB označen je sa Rtmp. U sledećim sekcijama opisane su ove jedinice.

5.6.1. JEDINICA WBSD

Jedinica WBSD (slika 57) vrši izbor vrednosti koja se upisuje u registar fajl. Sastoji se od 32-bitnog multipleksera 2/1 MPWBSD i jednog dvoulaznog ILI-kola, pomoću koga generiše signal WRITE kao uniju signala $R_{WB}.WRLMD$ i $R_{WB}.WRALU$.



Slika 57. Jedinica WBSD

5.6.2. PIPELINE REGISTAR RTMP

"Izlazni" *pipeline* registar R_{TMP} sadrži 38 bita. Signal takta za taj registar je CLK, a signal za brisanje \overline{mr} . Polja od kojih se sastoji *pipeline* registar R_{TMP} opisana su u tabeli 10.

Polje	Broj bita	Biti	Izvor	Opis
RES	32	0-31	WBSD	Podatak upisan u registar fajl
WRITE	1	32	WBSD	Da li je izvršen upis u registar fajl
rd	5	33-37	$R_{WB}.rd$	Adresa odredišnog registra

Tabela 10. Opis polja koja sačinjavaju *pipeline* registar R_{TMP}

5.7. OSTALE JEDINICE

U ovom odeljku opisane su jedinice koje ne spadaju u prethodno navedene grupe.

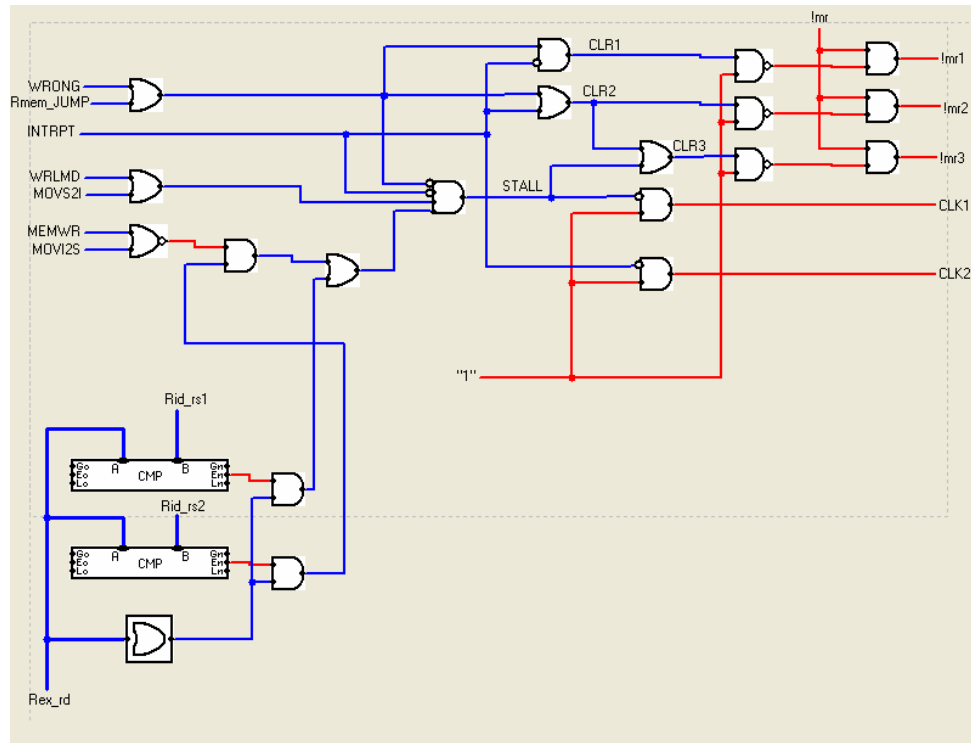
5.7.1. JEDINICA STALL

Jedinica Stall (slika 58) je kombinaciona mreža koja realizuje zaustavljanje i ispiranje *pipeline*-a. Zaustavljanje posmatranog *pipeline*-a vrši se samo kada su dve uzastopne instrukcije i i $i+1$ takve da i generiše rezultat u stepenu MEM, a $i+1$ koristi taj rezultat u stepenu EX. Instrukcija i može biti **lw** ili **movs2i**, a $i+1$ bilo koja instrukcija koja koristi jedinicu ALU ili Zero (to važi za sve instrukcije osim instrukcija za pristup posebnim registrima). Zaustavljanje instrukcije $i+1$ za jedan takt se vrši kada se ta instrukcija nalazi u stepenu ID. (Instrukcija **jsr** i mehanizam prekida takođe u stepenu MEM generišu vrednosti koje se upisuju u registar fajl. Međutim, nakon instrukcije **jsr** ili prekida se *pipeline* ispira, tako da se ne može javiti potreba za zaustavljanjem.)

Zaustavljanje *pipeline*-a se vrši na taj način što se ne dovodi signal takta *pipeline* registru R_{ID} i registru PC, tako da vrednosti tih registara ostaju nepromenjene, a sadržaj *pipeline* registra R_{EX} se briše, čime se instrukcija koja bi se nalazila u njemu pretvara u instrukciju bez dejstva. Signal STALL, koji izaziva zaustavljanje *pipeline*-a, biće aktivan ako su ispunjeni svi sledeći uslovi: 1) Aktivan je signal IDEXcmp1; ili, aktivan je signal IDEXcmp2 i nije aktivan signal MEMWR ni MOVI2S. 2) Aktivan je signal WRLMD ili MOVS2I, tj. u stepenu EX nalazi se instrukcija **lw** ili **movs2i**, respektivno. 3) Nije aktivan signal PREKID, WRONG ni R_{MEM} .JUMP.

Signali IDEXcmp1 i IDEXcmp2 su aktivni kada je adresa odredišnog registra instrukcije u stepenu EX jednaka adresi prvog ili drugog, respektivno, izvorišnog registra instrukcije u stepenu ID, a različita od nule. Signal IDEXcmp2 je uslovljen neaktivnošću signala MEMWR i MOVI2S da bi se izbegla nepotrebna zaustavljanja kada je u stepenu ID instrukcija **sw** ili **movi2s**, respektivno. Te instrukcije koriste vrednost registra opšte namene određenog argumentom rs2, ali ne u stepenu EX već u stepenu MEM.

Umesto globalnog signala takta CLK, za *pipeline* registar R_{ID} i registar PC koristi se signal takta CLK1, generisan u ovoj jedinici. Ako je signal STALL aktivan, CLK1 je neaktivan, a u suprotnom se CLK1 menja na isti način kao CLK. Brisanje sadržaja *pipeline* registra R_{EX} u slučaju zaustavljanja *pipeline*-a obuhvaćeno je delom jedinice Stall koji generiše signale za ispiranje *pipeline*-a.



Slika 58. Jedinica *Stall*

Specijalan slučaj zaustavljanja javlja se prilikom pokretanja opsluživanja prekida. Da bi se pojednostavilo čuvanje stanja procesora, deo 1 *pipeline* registra R_{MEM} se prilikom prekida zaustavlja za jedan takt i podaci iz tog dela ponovo prolaze kroz stepen MEM. Za taj deo se koristi signal takta CLK2, generisan u ovoj jedinici. Ako je signal PREKID aktivan, CLK2 je neaktivan, a u suprotnom se CLK2 menja na isti način kao CLK.

Ispiranje posmatranog *pipeline*-a vrši se u slučaju prekida (bez obzira na uzrok), безусловnog skoka, ili pogrešnog predviđanja ishoda uslovnog skoka. Ispiranje *pipeline*-a vrši se generisanjem signala za brisanje (*master reset*) za *pipeline* registre R_{ID} , R_{EX} i R_{MEM} . Ukoliko je aktivan signal WRONG (pogrešno predviđanje ishoda uslovnog skoka) ili signal R_{MEM_JUMP} (bezuslovni skok), a nije aktivan signal PREKID, biće aktivan signal CLR1. Ukoliko je aktivan signal WRONG ili R_{MEM_JUMP} ili PREKID, biće aktivan signal CLR2. Ukoliko je aktivan signal CLR2 ili STALL, biće aktivan signal CLR3.

Ispiranje *pipeline*-a na osnovu signala CLR1, CLR2 i CLR3 mora se uskladiti sa globalnim signalom takta CLK. Treba koristiti invertovanu logiku, što je običaj kod signala za brisanje. Osim toga, mora se omogućiti brisanje odgovarajućih *pipeline* registara globalnim *master reset* signalom \overline{mr} , nezavisno od takta. Zato se na osnovu signala CLR1, CLR2 i CLR3 generišu signali $\overline{mr1}$, $\overline{mr2}$ i $\overline{mr3}$, respektivno. Signal $\overline{mr1}$ se koristi za brisanje delova 1 i 2 *pipeline* registra R_{MEM} . Signal $\overline{mr2}$ se koristi za brisanje *pipeline* registra R_{ID} i dela 3 *pipeline* registra R_{MEM} . Signal $\overline{mr3}$ se koristi za brisanje *pipeline* registra R_{EX} .

6. IZVRŠAVANJE SIMULACIJE

U ovoj glavi prikazuje se način korišćenja simulatora, odnosno funkcionalne mogućnosti simulatora. Najpre je prikazan način korišćenja alata za izvršavanje simulacija. Nakon toga opisan je rad simulatora kroz simulacije koje pokrivaju najveći deo specifičnosti *pipeline* procesora, tj. veliki broj različitih hazarda podataka i upravljačkih hazarda, kao i primene raznovrsnih tehnika za njihovo otklanjanje.

6.1. FUNKCIONALNOSTI SIMULATORA

U poglavljima koja slede daju se detaljni pregledi postupaka za rad sa parametrima simulacije i za upravljanje simulatorom. Predstavljaju se pregledi postupaka za:

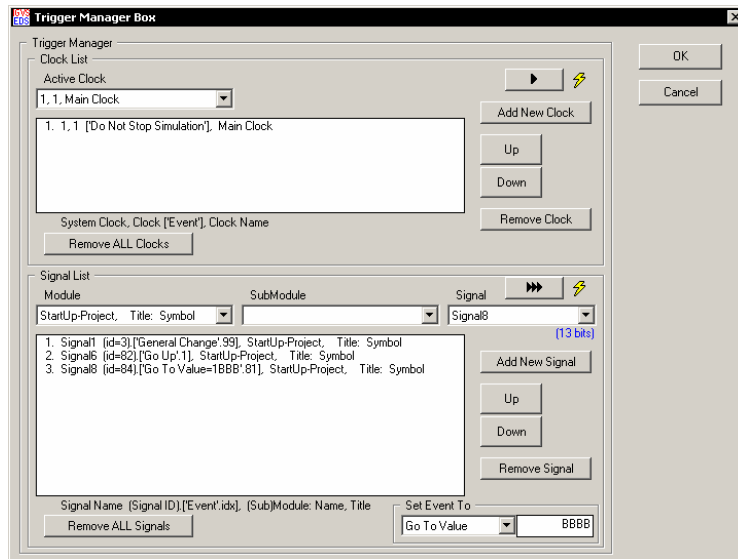
- definisanje događaja za zaustavljanje simulacije,
- rad sa listom signala za pregled promena vremenskih oblika signala i
- upravljanje simulatorom.

6.1.1. DEFINISANJE DOGAĐAJA ZA ZAUSTAVLJANJE SIMULACIJE

Pored zaustavljanja simulacija rada aktivnih digitalnih struktura, saglasnih ugrađenim karakteristikama funkcija za rad sa simulatorom, realizovan je i sistem postavljanja dodatnih kriterijuma za zaustavljanje rada simulatora tokom obrade standardnih funkcija za pokretanje i rad simulatora. Ovi dodatni kriterijumi nazivaju se događaji, i ispunjavanje uslova za aktiviranje događaja dovodi do zaustavljanja simulacije uz kreiranje odgovarajuće poruke u jedinstvenom sistemu za prikaz i arhiviranje poruka. Poruka se kreira samo ako je postavljen parametar *Trigger Alert*. Provere ostvarivanja uslova obavljaju se pri radu funkcija simulatora:

- *Forward Simulation, Next Clock*,
- *Forward Simulation, Run Simulation*,
- *Go To Tclk*, ukoliko je u polju *New Tclk Value* postavljena vrednost veća od maksimalne dostignute vrednosti, za tekuću simulaciju. Tada funkcija *Go To Tclk* radi na isti način kao i funkcija *Forward Simulation, Run Simulation*, uz dodatni uslov da se simulacija zaustavi u vremenskom trenutku postavljenom u polju *New Tclk Value*.

Na slici 59 prikazan je dijalog za definisanje događaja za zaustavljanje simulacije (*Trigger Manager Box*), koji se otvara aktiviranjem tastera grafičkog skupa funkcija *Triggers Dialog Box*. Lista signala takta u gornjem delu dijaloga određuje signale takta čije uzlazne ivice neće biti kriterijumi za zaustavljanje simulacije pri radu funkcije *Forward Simulation, Next Clock*. Lista signala u donjem delu dijaloga određuje signale kojima su dodeljeni kriterijumi za zaustavljanje simulacije pri radu funkcije *Forward Simulation, Run Simulation* i *Go To Tclk*.



Slika 59. Dijalog za definisanje događaja za zaustavljanje simulacije (*Trigger Manager Box*)

U gornjem delu dijaloga dat je spisak svih aktivnih signala takta, i tasterom *Add New Clock* označeni signal takta smešta se u listu signala taktova čije uzlazne ivice neće biti kriterijumi za zaustavljanje simulacije pri radu funkcije *Forward Simulation*, *Next Clock*. Obezbeđeni su tasteri za promenu redosleda signala takta u listi (*Up* i *Down*), kao i za brisanje jednog ili svih signala takta iz liste (*Remove Clock* i *Remove ALL Clocks*, respektivno). U donjem delu dijaloga, obezbeđen je sistem lista modula (*Module*), podređenih modula (*SubModule*) i signala (*Signal*), preko kojih je dostupan svaki signal aktivne digitalne strukture. Izborom signala, korišćenjem sistema za pregled signala (*Module/SubModule/Signal*), i aktiviranjem tastera *Add New Signal*, odabrani signal postaje deo liste signala kojima su dodeljeni kriterijumi za zaustavljanje simulacije, pri radu funkcija *Forward Simulation*, *Run Simulation* i *Go To Tclk*. Nije dozvoljeno da se jedan signal više puta pojavi u ovoj listi signala, za isti kriterijum za zaustavljanje simulacije. Obezbeđeni su tasteri za promenu redosleda signala u listi (*Up* i *Down*), kao i za brisanje jednog ili svih signala iz liste (*Remove Signal* i *Remove ALL Signals*, respektivno). Prilikom aktiviranja tastera *Add New Signal*, signalu koji se dodaje na listu pridružuje se vrsta događaja koja je odabrana u objektu *Set Event To*. Na raspolaganju su dva skupa događaja u zavisnosti od toga da li je širina signala jedan bit, ili je širina signala veća od jednog bita. Raspoloživi događaju su:

Za sve signale:

- *Go To TriState* – promena signala iz bilo kog stanja u stanje visoke impedanse,
- *Go To Unknown* – promena signala iz bilo kog stanja u nepoznato stanje,
- *Go To TriState/Unknown* – promena signala iz bilo kog stanja u stanje visoke impedanse ili nepoznato stanje,
- *Back From TriState* – promena signala u bilo koje stanje iz stanja visoke impedanse,
- *Back From Unknown* – promena signala u bilo koje stanje iz nepoznatog stanja,

- *Back From TriState/Unknown* – promena signala u bilo koje stanje iz stanja visoke impedanse ili nepoznatog stanja,
- *General Change* – promena signala bilo koje vrste.

Samo za signale širine jedan bit:

- *Go Up* – promena signala iz bilo kog stanja u stanje logičke jedinice,
- *Go Down* – promena signala iz bilo kog stanja u stanje logičke nule,
- *Go Up/Down* – promena signala iz bilo kog stanja u stanje logičke jedinice ili nule,
- *Back From Up* – promena signala u bilo koje stanje iz stanja logičke jedinice,
- *Back From Down* – promena signala u bilo koje stanje iz stanja logičke nule,
- *Back From Up/Down* – promena signala u bilo koje stanje iz stanja logičke jedinice ili nule.

Samo za signale širine veće od jednog bita:

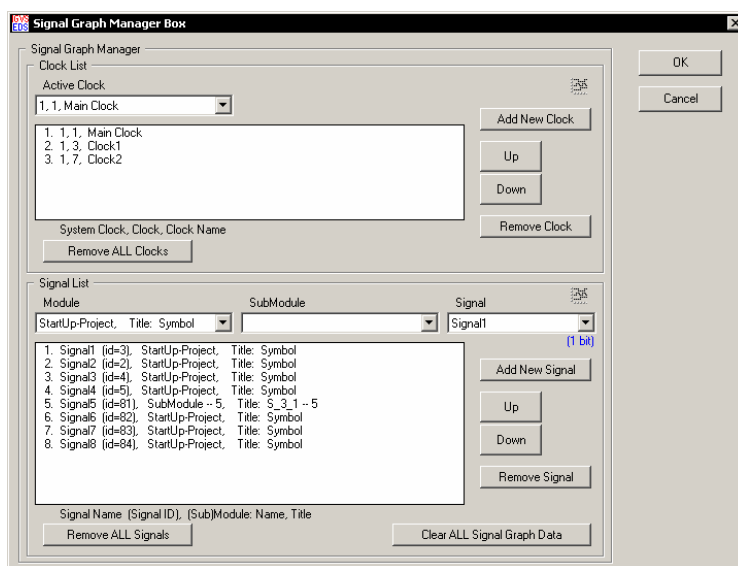
- *Go To Value* – promena stanja signala šireg od jednog bita tako da stanje signala bude jednako postavljenoj vrednosti. Događaj se odnosi samo na trenutak promene vrednosti, a ne i na period kada je vrednost signala konstantna i jednaka postavljenoj vrednosti,
- *Back From Value* – promena stanja signala šireg od jednog bita tako da stanje signala bude različito od postavljene vrednosti. Događaj se odnosi samo na trenutak promene vrednosti, a ne i na period kada je vrednost signala konstantna i različita od postavljene vrednosti.

Kada se definišu događaji *Go To Value* i *Back From Value*, pojavljuje se *Text Box* polje u koje su upisuje vrednost na koju se odnosi događaj (slika 59). Taster *OK* zatvara dijalog sa prihvatanjem promena, dok taster *Cancel* zatvara dijalog bez prihvatanja promena.

6.1.2. RAD SA LISTOM SIGNALA ZA PREGLED PROMENA VREMENSKIH OBLIKA SIGNALA

U opštem slučaju, aktivna digitalna struktura može da ima nekoliko desetina hiljada signala čije promene bi mogle da se prate. Deo strukture podataka koji je rezervisan za čuvanje promena signala, u opštem slučaju ne može da bude dovoljno veliki da prihvati promene svih signala, za svaki ciklus, svih aktivnih signala takta. Zbog toga je određena manja vidljiva oblast u prostoru koji određuju glavni signal takta *Main Clock*, koji ima maksimalno dozvoljenu učestalost jednaku učestalosti sistemskog signala takta *System Clock*, i niz odabranih signala, kojih može biti maksimalno 24. Čuvaju se promene odabranih signala za maksimalno 8000 ciklusa pojavljivanja uzlazne ivice glavnog signala takta *Main Clock*. Promene aktivnih signala takta se ne pamte, jer se na osnovu vrednosti učestalosti može u svakom vremenskom trenutku simulacije jednoznačno utvrditi da li signal takta ima uzlaznu ivicu ili nema uzlaznu ivicu.

Na slici 60 prikazan je dijalog za rad sa listom signala za pregled promena vremenskih oblika signala (*Signal Graph Manager*), koji se otvara aktiviranjem tastera grafičkog skupa funkcija *Signal Graph*.



Slika 60. Dijalog za rad sa listom signala za pregled promena vremenskih oblika signala (*Signal Graph Manager Box*)

Dijalog za definisanje liste signala koji će biti prikazivani na vremenskom dijagramu radi na isti način kao i dijalog za definisanje događaja za zaustavljanje simulacije pri radu funkcija simulatora (*Trigger Manager Box*). Razlika je u tome što nije na raspolaganju postavljanje vrednosti parametra *Set Event To*, i dozvoljeno je da se jedan signal više puta pojavi na listi signala, uz upozorenje da se signal već nalazi na listi. Pored toga, na raspolaganju je i funkcija za brisanje svih podataka sačuvanih u strukturi za čuvanje promena signala. Ova funkcija aktivira se korišćenjem tastera *Clear ALL Signal Graph Data*, kao i implicitno pri svim akcijama koje remete integritet sačuvanih podataka. Taster *OK* zatvara dijalog sa prihvatanjem promena, dok taster *Cancel* zatvara dijalog bez prihvatanja promena.

6.1.3. UPRAVLJANJE SIMULATOROM

Upravljanje simulatorom alata *IGoVSoEDS* obuhvata kontrolu režima rada simulatora i upravljanje simulacijom. U cilju pogodnijeg prikaza detaljnog pregleda mogućnosti za upravljanje simulatorom neophodno je da se definišu odgovarajuća stanja simulatora. Značajna stanja simulatora i odgovarajuće oznake su:

- T_0 – početno stanje simulatora. Ovo stanje može da se dobije aktiviranjem funkcije za vraćanje simulacije u početno stanje, kao i višestrukim aktiviranjem funkcija za vraćanje simulacije unazad.
- T_{sim} – stanje simulatora koje se trenutno prikazuje u softverskom paketu. Aktiviranjem bilo koje od funkcija za upravljanje simulacijom, u opštem slučaju, stanje simulatora

T_{sim} se menja.

- T_{max} – maksimalno dostignuti vremenski trenutak simulacije za trenutno aktivnu digitalnu strukturu. Upotrebom funkcija simulatora za pokretanje simulacije unapred, u opštem slučaju, vremenski trenutak T_{max} pomera se ka većim vrednostima.

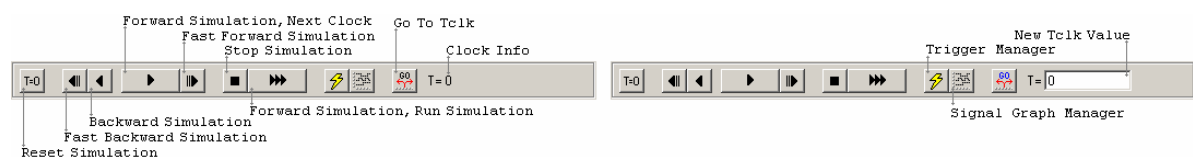
Simulator alata ima dva režima rada, koji se označavaju kao režimi rada:

- *Play* – režim rada kada se stanje simulatora i pregledi rezultata rada simulatora prikazuju za vremenski trenutak $T_{sim}=T_{max}$,
- *RePlay* – režim rada kada se stanje simulatora i pregledi rezultata rada simulatora prikazuju za vremenski trenutak $T_{sim}<T_{max}$.

Režim rada *Play* je režim rada kada se simulacija ostvaruje samo aktiviranjem tastera za obradu jednog ili više narednih signala takta, bez korišćenja funkcija za vraćanje simulacije unazad. U režimu rada *Play* vremenski trenutak simulacije je uvek T_{max} . U ovom režimu rada na raspolaganju su pregled vremenskih oblika odabranih signala, kao i funkcije za promenu vrednosti svih memorijskih elemenata digitalne strukture.

Aktiviranjem neke od funkcija za vraćanje simulacije u vremenske trenutke koji su već obrađeni, prelazi se u režim rada *RePlay*, kada se kombinacijom memorisanih stanja simulatora, i simulacije rada digitalne strukture za taktove između odabranog memorisanog stanja simulatora i vremenskog trenutka simulacije koji treba prikazati, stanje simulatora dovodi u traženi vremenski trenutak. Režim rada *RePlay* koristi se za prikaz stanja simulatora za vremenske trenutke simulacije $T_{sim}<T_{max}$. Na raspolaganju je maksimalno 10 memorisanih stanja simulatora, koji čuvaju do 8 stanja simulatora, približno ravnomerno raspoređenih od T_0 do T_{max} , kao i 2 stanja simulatora, veoma bliskih vremenskom trenutku T_{max} . Prva grupa memorisanih stanja simulatora koristi se za velika pomeranja simulacije unazad, dok se druga grupa koristi za pomeranja simulacije za jedan takt unazad. Sva memorisana stanja simulatora imaju važeća stanja samo za simulacije koje prelaze više desetina hiljada taktova, dok se za simulacije sa manje obrađenih taktova koriste samo neka od memorisanih stanja. U režimu rada *RePlay* nije moguće pregledati vremenske oblike signala, i nisu na raspolaganju funkcije za promenu vrednosti memorijskih elemenata digitalne strukture. Korišćenjem funkcija za pomeranje stanja simulatora unapred, u trenutku kada se simulator postavi na vremenski trenutak simulacije T_{max} , režim rada *RePlay* implicitno prelazi u režim rada *Play*.

Upravljanje simulacijom ostvaruje se korišćenjem tastera grafičkog skupa funkcija *Simulation Toolbar*, koji je prikazan na slici 61.



Slika 61. Grafički skup funkcija za rad sa funkcijama simulatora (*Simulation Toolbar*)

Skup tastera za upravljanje radom simulacije (slika 61–levo), obezbeđuje funkcije za:

- postavljanje početnog stanja simulatora, odnosno stanja T_0 , za $T_{\max}=0$ (*Reset Simulation*)
- postavljanje simulatora na prethodno dostignute vremenske trenutke, čime se pokreće *RePlay* režim rada simulatora (*Fast Backward Simulation* i *Backward Simulation*)
- pokretanje simulatora za jedan takt unapred (*Forward Simulation, Next Clock*)
- postavljanje simulatora na vremenske trenutke koji se nalaze iza vremenskih trenutaka koji se posmatraju u *RePlay* režimu rada simulatora (*Fast Forward Simulation*)
- zaustavljanje simulacije (*Stop Simulation*)
- pokretanje simulatora za više taktova unapred (*Forward Simulation, Run Simulation*)
- postavljanje simulatora na proizvoljan vremenski trenutak pre ili iza trenutno dostignutog vremenskog trenutka simulacije T_{\max} (*Go To Tclk*).

Na krajnjoj desnoj strani nalazi se natpis na kome se prikazuje vrednost dostignutog trenutka simulacije T_{\max} (*Clock Info*). U režimu rada *RePlay* vrednost parametra *Clock Info* prikazuje se kao T_{sim}/T_{\max} . Na raspolaganju su i tasteri (slika 61–desno) za otvaranje dijaloga za definisanje događaja za zaustavljanje simulacije (*Trigger Manager*) i za rad sa listom signala za pregled promena vremenskih oblika signala (*Signal Graph Manager*).

Reset Simulation funkcija vraća stanje simulatora na početno stanje T_0 , odnosno na stanje $T_{\max}=0$. Aktiviranjem funkcije *Reset Simulation* brišu se sva memorisana stanja simulatora i sačuvane promene signala za potrebe prikaza vremenskih oblika odabranih signala. Ova funkcija je jedina funkcija koja vraća simulaciju unazad, ali ne postavlja simulator u *RePlay* režim rada.

Fast Backward Simulation postavlja stanje simulatora vremenski unazad, na vremenski najbliže memorisano stanje simulatora, u odnosu na trenutno prikazani vremenski trenutak simulacije. Ukoliko postoji odgovarajuće memorisano stanje simulatora, simulator se prevodu u to stanje i nastavlja da radi u *RePlay* režimu rada. Ukoliko ne postoji odgovarajuće memorisano stanje simulatora, simulator se ne postavlja u drugo stanje, i režim rada ostaje isti kao i pre aktiviranja funkcije.

Backward Simulation postavlja stanje simulatora vremenski unazad, na vremenski trenutak za 1 manji od trenutno prikazanog vremenskog trenutka, i prevodi simulator u *RePlay* režim rada. Ova funkcija ostvaruje simulaciju za jedan takt unazad, i može da dovede simulator u stanje T_0 , ali ostaje $T_{\max}>0$.

Forward Simulation, Next Clock aktivira simulaciju jednog narednog sistemskog signala takta, ali i svih narednih sistemskih signala takta do trenutka pojavljivanja uzlazne ivice prvog narednog signala takta, za koji u sistemu definisanja događaja za zaustavljanje simulacije (*Trigger Manager*) nije postavljeno *Do Not Stop Simulation*. Funkcija je dostupna u oba

režima rada simulatora.

Fast Forward Simulation postavlja stanje simulatora vremenski unapred, na vremenski najbliže memorisano stanje simulatora, u odnosu na trenutno prikazani vremenski trenutak simulacije. Ukoliko postoji odgovarajuće memorisano stanje simulatora, simulator se prevodu u to stanje. Ukoliko ne postoji odgovarajuće memorisano stanje, simulator se ne postavlja u drugo stanje, i režim rada ostaje isti kao i pre aktiviranja funkcije. Samo u *RePlay* režimu rada postoje odgovarajuća memorisana stanja simulatora za rad funkcije *Fast Forward Simulation*.

Stop Simulation funkcija zaustavlja simulaciju, za sve simulacije koje obuhvataju obradu više sistemskih signala takta. Ova funkcija ne utiče na promenu režima rada simulatora.

Forward Simulation, *Run Simulation* funkcija koristi se za startovanje simulacije, odnosno aktiviranje simulacije za obradu proizvoljnog broja sistemskih signala takta do prvog sledećeg kriterijuma za zaustavljanje simulacije (*Trigger Manager*), odnosno do aktiviranja funkcije *Stop Simulation*. Funkcija je dostupna u oba režima rada simulatora.

Go To Tclk funkcija stanje simulatora dovodi u stanje jednako postavljenoj vrednosti sistemskog signala takta u polju *New Tclk Value*. Prvo aktiviranje funkcije *Go To Tclk* kreira polje *New Tclk Value* (slika 61–desno). Naredno aktiviranje izvršava funkciju postavljanja simulatora na vremenski trenutak koji je postavljen u polju *New Tclk Value*, i potom uklanja polje *New Tclk Value*. Ukoliko je vrednost polja *New Tclk Value* manja od vrednosti trenutno prikazanog vremenskog trenutka, simulacija se vremenski pomera unazad, a u suprotnom simulacija se vremenski pomera unapred. Bez obzira na režim rada simulatora, koji je aktivan neposredno pre izvršavanje funkcije *Go To Tclk*, nakon izvršavanja funkcije, režim rada zavisi od vremenskog trenutka u koji je doveden simulator. Ukoliko je traženi vremenski trenutak stanja simulatora *New Tclk Value* manji od T_{\max} , nakon završetka rada funkcije postavlja se režim rada simulatora *RePlay*, a ako je jednak ili veći od T_{\max} , postavlja se režim rada simulatora *Play*, i T_{\max} dobija vrednost postavljenu u polju *New Tclk Value*. Izvršavanje funkcije *Go To Tclk* sastoji se u pronalaženju memorisanog stanja simulatora koje čuva stanje simulatora za najbliži vremenski trenutak manji od traženog vremenskog trenutka postavljenog u *New Tclk Value*, i izvršavanja simulacije rada sistemskih signala takta od memorisanog stanja do traženog stanja simulatora.

Za potrebe efikasnijeg kreiranja i modifikacije veoma složenih digitalnih struktura, realizovan je parametar *ReConfiguration Always ON*. Ukoliko ovaj parametar nije potvrđen, sa promenom digitalne strukture ne ažurira se struktura podataka koja čuva opis strukture, i ne ažurira se stanje simulatora. U ovom režimu rada, skup funkcija za upravljanje radom simulacije nije dostupan, i pokriven je porukom *ReConfiguration is Stopped. Some values likely are incorrect*. Stanja signala ne moraju da budu odgovarajuća, kao i vrednosti u svim ostalim pregledima rezultata rada simulatora.

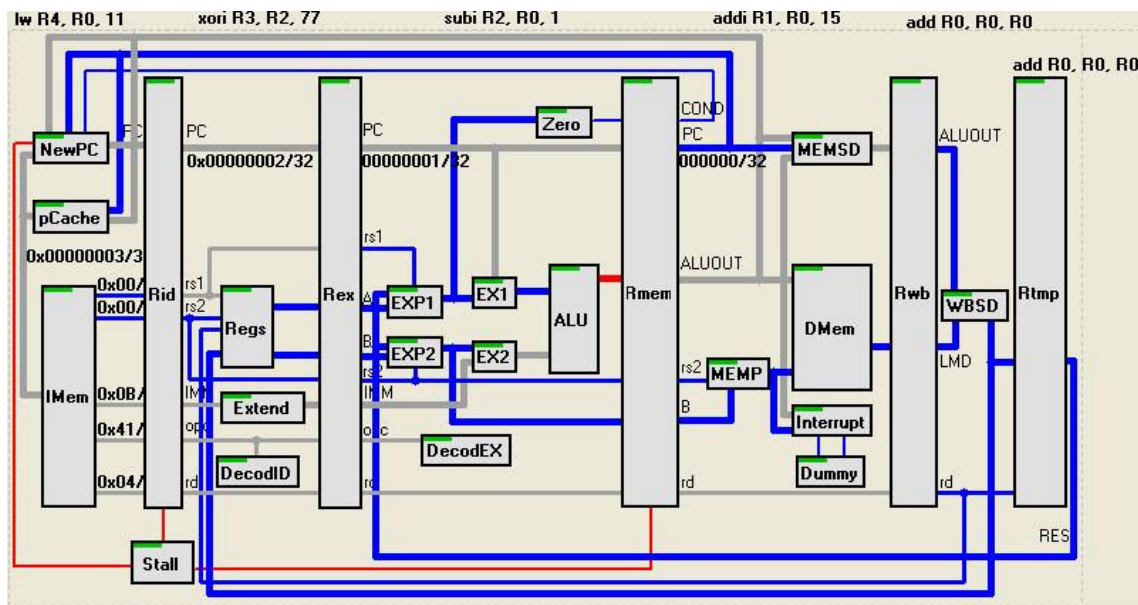
6.2. SIMULACIJA: HAZARDI PODATAKA I UPRAVLJAČKI HAZARD

U narednom primeru biće prikazano nekoliko situacija sa hazardima podataka, kao i načini njihovog rešavanja u realizovanom *pipeline*-u. Takođe biće prikazana i jedna moguća situacija promašene predikcije skoka i oporavka od iste. Program primera nalazi se u tabeli 11.

Address	Heksadecimalna vrednost	Instrukcija	Efekat instrukcije
0x0000	0x0084000f	addi R1, R0, 15	R1 = 15
0x0001	0x01880001	subi R2, R0, 1	R2 = -1
0x0002	0x0a8c404d	xori R3, R2, 77	R3 = not 77
0x0003	0x2090000b	lw R4, R0, 11	R4 = DMem [11]
0x0004	0x2880800a	beqz R4, 10	if (R4 == 0) PC = PC + 10
0x0005	0x00142200	add R5, R1, R2	R5 = R1 + R2
0x0006	0x0998600c	ori R6, R3, 12	R6 = R3 or 12
0x0007	0x081ca600	and R7, R5, R6	R7 = R5 and R6
0x000e	0x02108100	sub R8, R4, R1	R8 = R4 - R1

Tabela 11. Kod primera

Simulacija započinje sa punjenjem prve četiri instrukcije u *pipeline*, respektivno, u prva četiri ciklusa takta (slika 62).



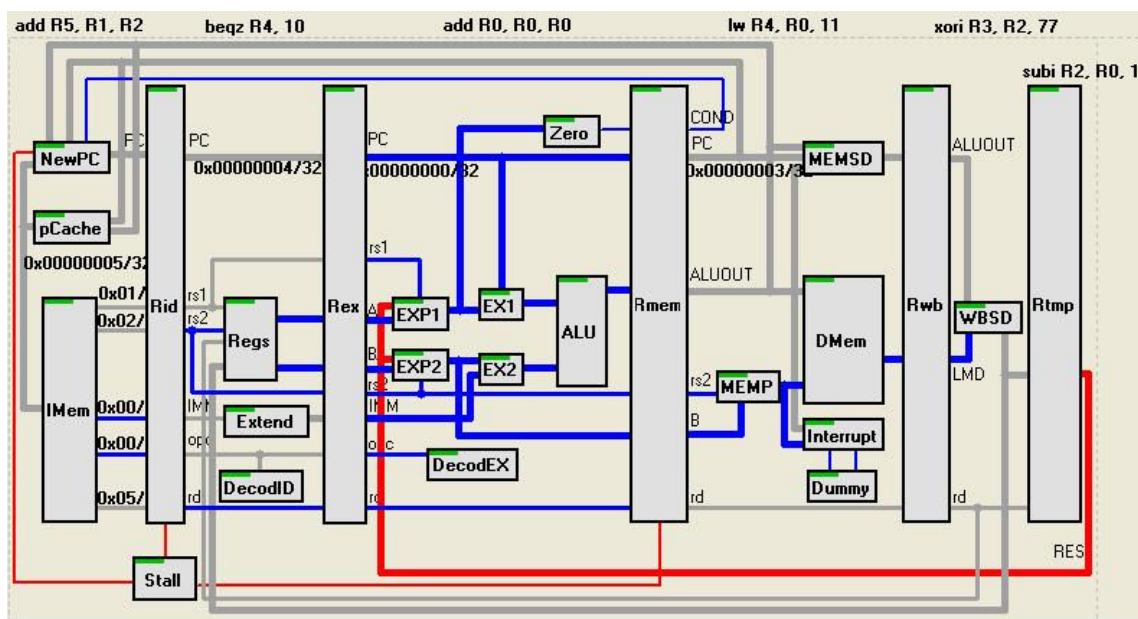
Slika 62. Situacija u *pipeline*-u nakon prva četiri ciklusa takta

U petom ciklusu signala takta dolazi do prvog hazarda podataka. Instrukcija *xori R3, R2, 77* se u ovom ciklusu signala takta nalazi u EX stepenu *pipeline*-a i potreban joj je kao operand rezultat instrukcije *subi R2, R0, 1*, koja još uvek nije upisala svoj rezultat i nalazi se u stepenu MEM, kao što je prikazano u tabeli 12. Ovaj hazard je rešen prosleđivanjem iz faze MEM u fazu EX.

	1	2	3	4	5
addi R1, R0, 15	IF	ID	EX	MEM	WB
subi R2, R0, 1		IF	ID	EX	MEM
xori R3, R2, 77			IF	ID	EX
lw R4, R0, 11				IF	ID
beqz R4, 10					IF

Tabela 12. Situacija u *pipeline*-u nakon petog ciklusa signala takta

U šestom ciklusu signala takta, ponovo dolazi do hazarda podataka, ovaj put zbog instrukcije *beqz R4, 10*, koja se nalazi u ID stepenu *pipeline*-a i kojoj je potrebna vrednost registra R4, u koji instrukcija *lw R4, R0, 11* još uvek nije upisala vrednost. Ovaj hazard je rešen zaustavljanjem *pipeline*-a jedan ciklus signala takta i ispiranjem EX stepena u narednom ciklusu signala takta, kao što je prikazano na slici 63.



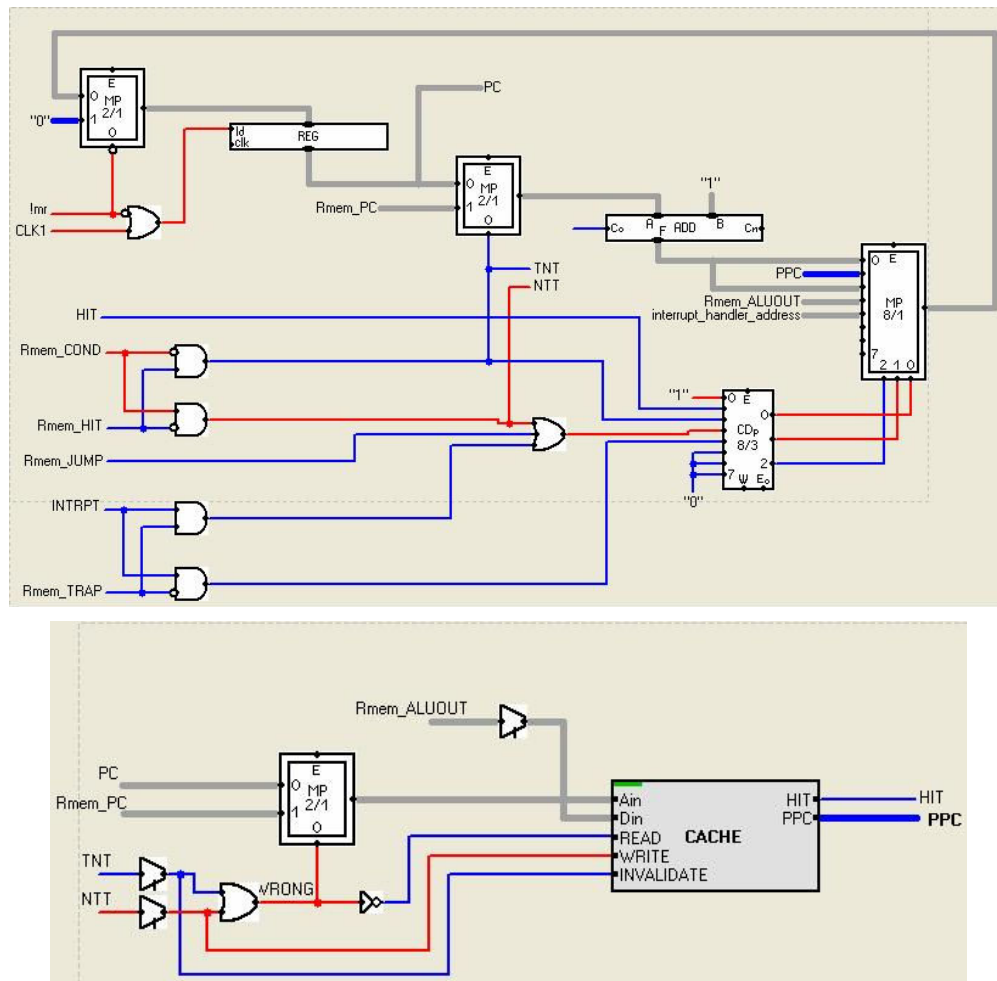
Slika 63. Situacija u *pipeline*-u nakon zaustavljanja (sedmi ciklus takta)

Kako je nakon ove intervencije u *pipeline*-u situacija u osmom ciklusu signala takta, kao što je prikazano u tabeli 13. Odnosno instrukcija *beqz R4, 10* se sada nalazi u EX stepenu, a instrukcija *lw R4, R0, 11* u WB stepenu, potrebno je izvršiti i prosleđivanje iz stepena WB u stepen EX, kako bi instrukcija *beqz R4, 10* bila korektno izvršena.

	3	4	5	6	7	8
xori R3, R2, 77	IF	ID	EX	MEM	WB	
lw R4, R0, 11		IF	ID	EX	MEM	WB
beqz R4, 10			IF	ID	stall	EX
add R5, R1, R2				IF	stall	ID
ori R6, R3, 12						IF

Tabela 13. Situacija u *pipeline*-u nakon osmog ciklusa signala takta

U devetom ciklusu signala takta, signal *WRONG* u jedinici *pCache* postaje aktivan (slika 64) i svojom aktivnom vrednošću, signalizira da je došlo do greške u predikciji skoka. Kako je signal *NTT*, koji se generiše u jedinici *NewPC* (slika), taj koji je doveo do aktivne vrednosti signala *WRONG*, znači da je predikcija bila da neće biti skoka (*not taken*), a da je stvarno stanje da ima skoka (*taken*).



Slika 64. Generisanje signala *NTT* i *WRONG* u devetom ciklusu signala takta

Zbog pogrešne predikcije skoka u desetom ciklusu signala takta dolazi do ispiranja ID, EX i MEM stepena *pipeline*-a, a u stepen IF dolazi prava instrukcija sa adrese skoka, kao što je

prikazano u tabeli 14.

	5	6	7	8	9	10
beqz R4, 10	IF	ID	<i>stall</i>	EX	MEM	WB
add R5, R1, R2		IF	<i>stall</i>	ID	EX	<i>Flush</i>
ori R6, R3, 12				IF	ID	<i>Flush</i>
and R7, R5, R6					IF	<i>Flush</i>
sub R8, R4, R1						IF

Tabela 14. Situacija u *pipeline*-u nakon desetog ciklusa signala takta

Simulacija se završava u petnaestom ciklusu signala takta, bez ikakvih hazarda do kraja izvršavanja.

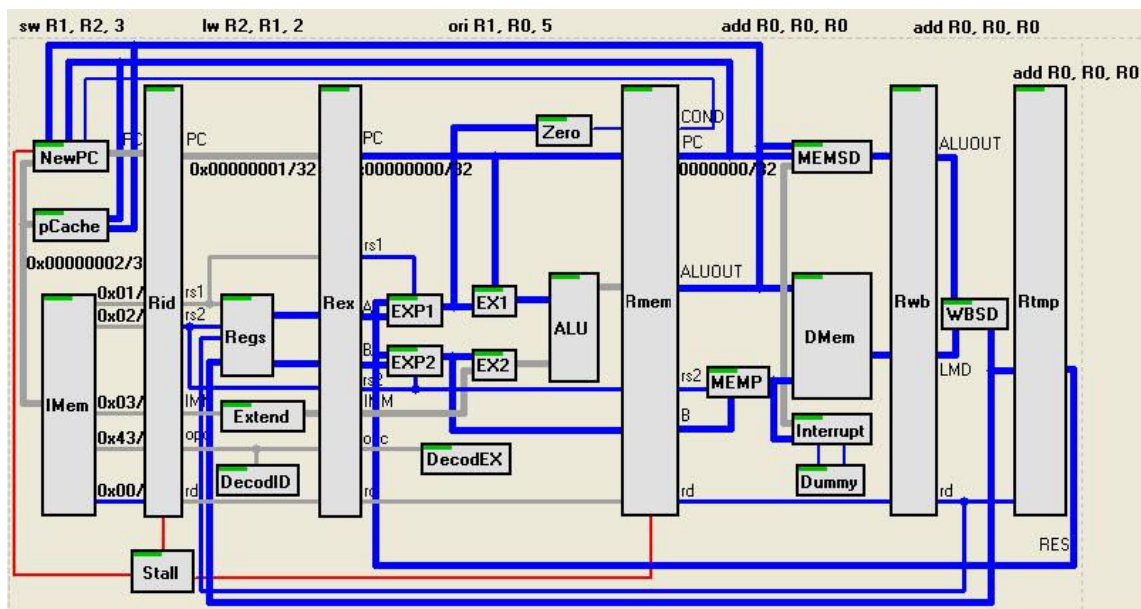
6.3. SIMULACIJA: HAZARDI PODATAKA I VIŠESTRUKO PROSLEĐIVANJE

U narednom primeru biće prikazano nekoliko situacija sa hazardima podataka, kao i načini njihovog rešavanja u realizovanom *pipeline*-u. Takođe biće prikazana i situacija sa višestrukim prosleđivanjem. Program primera nalazi se u tabeli 15.

Address	Heksadecimalna vrednost	Instrukcija	Efekat instrukcije
0x0000	0x09840005	ori R1, R0, 5	R1 = R0 or 12
0x0001	0x20882002	lw R2, R1, 2	R2 = DMem [R1+2]
0x0002	0x21802203	sw R1, R2, 3	DMEM[R1+3] = R2
0x0003	0x010c2200	sub R3, R1, R2	R3 = R1 - R2
0x0004	0x35100000	movs2i R4	R4 = PSW
0x0005	0x21802404	sw R1, R4, 4	DMEM[R1+4] = R4

Tabela 15. Kod primera

Simulacija započinje sa punjenjem prve tri instrukcije u *pipeline*, respektivno, u prva tri ciklusa takta (slika 65).



Slika 65. Situacija u *pipeline*-u nakon prva tri ciklusa takta

U četvrtom ciklusu signala takta dolazi do prvog hazarda podataka. Instrukcija *lw R2, R1, 2* se u ovom ciklusu signala takta nalazi u EX stepenu *pipeline*-a i potreban joj je kao operand rezultat instrukcije *ori R1, R0, 5*, koja još uvek nije upisala svoj rezultat i nalazi se u stepenu MEM, kao što je prikazano u tabeli 16. Ovaj hazard je rešen prosleđivanjem iz faze MEM u fazu EX.

	1	2	3	4
ori R1, R0, 5	IF	ID	EX	MEM
lw R2, R1, 2		IF	ID	EX
sw R1, R2, 3			IF	ID
sub R3, R1, R2				IF

Tabela 16. Situacija u *pipeline*-u nakon četvrtog ciklusa signala takta

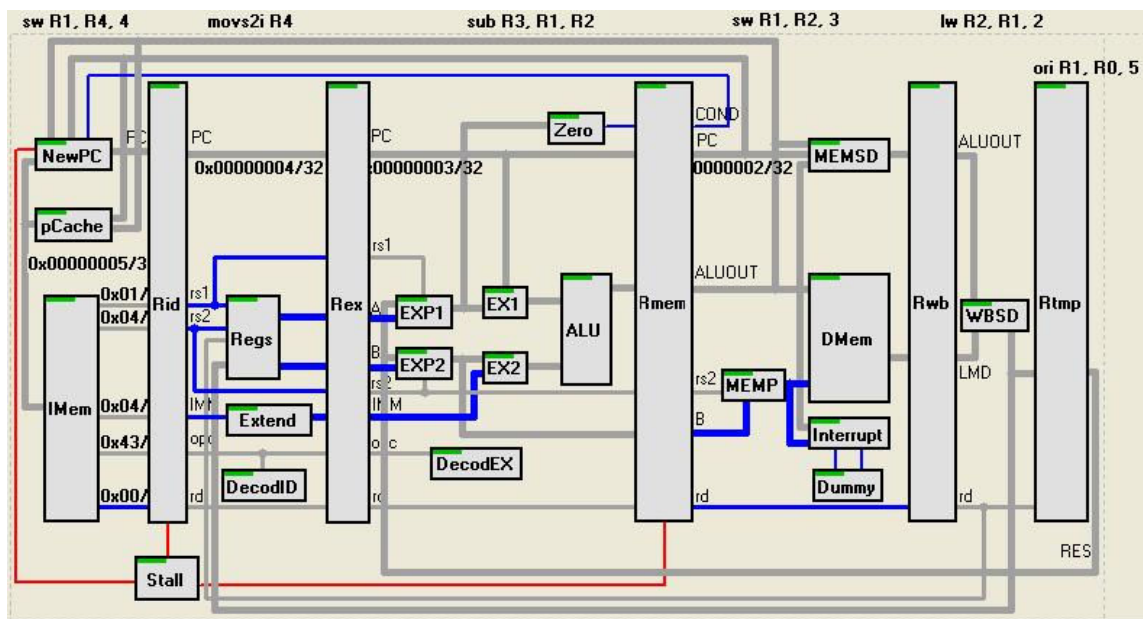
U petom ciklusu signala takta dolazi do drugog hazarda podataka. Instrukcija *sw R1, R2, 3* se u ovom ciklusu signala takta nalazi u EX stepenu *pipeline*-a i potreban joj je kao operand rezultat instrukcije *ori R1, R0, 5*, koja još uvek nije upisala svoj rezultat i nalazi se u stepenu WB, kao što je prikazano u tabeli 17. Ovaj hazard je rešen prosleđivanjem iz faze WB u fazu EX.

	1	2	3	4	5
ori R1, R0, 5	IF	ID	EX	MEM	WB
lw R2, R1, 2		IF	ID	EX	MEM
sw R1, R2, 3			IF	ID	EX

sub R3, R1, R2				IF	ID
movs2i R4					IF

Tabela 17. Situacija u *pipeline*-u nakon petog ciklusa signala takta

U šestom ciklusu signala takta, ponovo dolazi do hazarda podataka, i to ovaj put višestrukog. Situacija u šestom ciklusu signala takta je sledeća: instrukcija *sub R3, R1, R2* se nalazi u stepenu EX *pipeline*-a i kao što se može videti u tabeli, ovoj instrukciji kao operandi trebaju rezultati instrukcija *lw R2, R1, 2* (trenutno u fazi WB) i *ori R1, R0, 5* (čiji se rezultat nalazi u registru Rtmp). Takođe, instrukciji *sw R1, R2, 3*, koja se nalazi u MEM stepenu *pipeline*-a, potreban je kao operand rezultat operacije *lw R2, R1, 2* (trenutno u fazi WB). Ova situacija razrešava se prosleđivanjem iz faze WB u faze MEM i EX, kao i prosleđivanjem iz *pipeline* registra Rtmp u fazu EX i na taj način se dobija korektno izvršavanje programa, kao što je to prikazano na slici 66.



Slika 66. Situacija u *pipeline*-u nakon šestog ciklusa takta

Poslednji hazard podataka u ovom primeru javlja se u devetom ciklusu signala takta za koji je stanje u *pipeline*-u dato u tabeli 18. U ovom taktu instrukciji *sw R1, R4, 4* koja se nalazi u MEM stepenu *pipeline*-a, potreban je kao operand rezultat instrukcije *movs2i R4*, koja se nalazi u stepenu WB. Ovaj hazard razrešava se prosleđivanjem iz stepena WB u stepen MEM.

	5	6	7	8	9
movs2i R4	IF	ID	EX	MEM	WB
sw R1, R4, 4		IF	ID	EX	MEM

Tabela 18. Situacija u *pipeline*-u nakon osmog ciklusa signala takta

Simulacija se završava u jedanaestom ciklusu signala takta, bez ikakvih hazarda do kraja izvršavanja.

6.4. SIMULACIJA: UPRAVLJAČKI HAZARDI (INSTRUKCIJE SKOKA)

U narednom primeru, biće demonstriran rad dela za predikciju skoka, kao i ponašanje dizajniranog *pipeline* procesora kod situacija pogrešne predikcije skoka, odnosno, upravljačkih hazarda. Programski kod primera nalazi se u tabeli 19.

Address	Heksadecimalna vrednost	Instrukcija	Efekat instrukcije
0x0000	0x20840000	lw R1, R0, 0	R1 = DMem [0]
0x0001	0x20880001	lw R2, R0, 1	R2 = DMem [1]
0x0002	0x0df400f8	li1 R29, R0, 248	R29 = 0 0x0000f800
0x0003	0x0dc400e0	li1 R17, R0, 224	R17 = 0 0x0000e000
0x0004	0x2bfa2000	jsr R30, R17, 0	R30 = PC; PC = R17
0x0005	0x00142200	add R5, R1, R2	R5 = R1 + R2
0x0006	0x0a98a0ff	xori R6, R5, 255	R6 = R5 \oplus 0x000000ff
0x0007	0x081ca600	and R7, R5, R6	R7 = R5 + R6
0xe000	0x2183a101	sw R29, R1, 1	DMEM[R29+1] = R1
0xe001	0x02f7a001	addui R29, R29, 1	R29 = R29 + 1
0xe002	0x000c0000	add R3, R0, R0	R3 = R0 + R0
0xe003	0x000c6200	add R3, R3, R2	R3 = R3 + R2
0xe004	0x01842001	subi R1, R1, 1	R1 = R1 - 1
0xe005	0x298020fe	bnez R1, -2	if R1 \leq 0 PC = PC-2
0xe006	0x2087a000	lw R1, R29, 0	R1 = DMem [R29]
0xe007	0x03f7a001	subui R29, R29, 1	R7 = R5 + R6
0xe008	0x2a83c001	jr R30, 1	PC = R30 + 1

Tabela 19. Kod primera

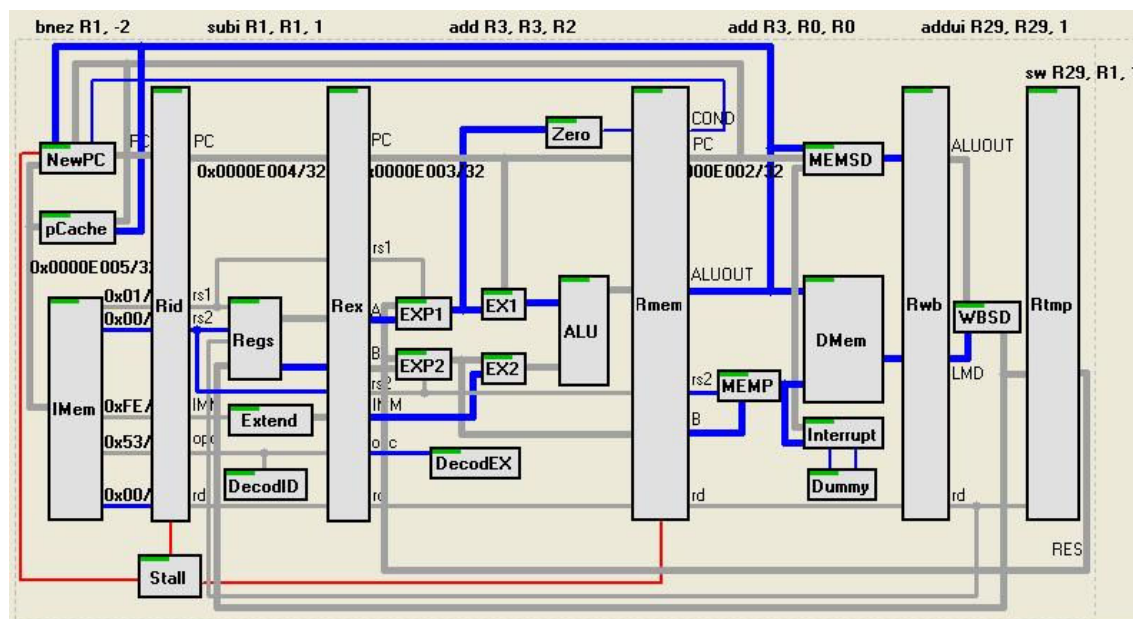
Simulacija započinje sekvencijalnim učitavanjem prvih pet instrukcija u *pipeline*. U petom ciklusu signala takta instrukcija *jsr R30, R17, 0* ulazi u IF stepen *pipeline*-a. Nakon toga nastavlja se sa sekvencijalnim učitavanjem narednih instrukcija u *pipeline*, sve do osmog ciklusa takta, kada je izračunata adresa skoka i ispunjen uslov za skok u potprogram

instrukcije *jsr R30, R17, 0*. Zbog toga u narednom taktu registri Rid, Rex i Rmem u kojima se trenutno nalaze pogrešne instrukcije, biće isprani (*flushed*), a u IF stepen *pipeline*-a će biti učitana prva instrukcija potprograma sa adrese *0xe000*. Situacija u *pipeline*-u u devetom ciklusu signala takta prikazana je u tabeli 20.

	5	6	7	8	9
li1 R17, R0, 224	ID	EX	MEM	WB	
jsr R30, R17, 0	IF	ID	EX	MEM	WB
add R5, R1, R2		IF	ID	EX	Flush
xori R6, R5, 255			IF	ID	Flush
and R7, R5, R6				IF	Flush
sw R29, R1, 1					IF

Tabela 20. Situacija u *pipeline*-u u devetom ciklusu signala takta

Nakon skoka na potprogram nastavlja se sa sekvencijalnim ubacivanjem instrukcija u *pipeline*, sve do četrnaestog ciklusa signala takta, kada se u IF stepenu pojavljuje instrukcija *bnez R1,-2*. U četrnaestom ciklusu signala takta počinje rad sa prediktorom skoka, koji u prvom slučaju pojavljivanja instrukcije *bnez R1,-2*, daje predikciju da neće biti skoka (*not taken*), kao što je prikazano na slici 67.



Slika 67. Situacija u *pipeline*-u u četrnaestom ciklusu signala takta

Zbog takve predikcije u narednim taktovima nastavlja se sa sekvencijalnim učitavanjem instrukcija u *pipeline* sve do sedamnaestog ciklusa signala takta u kome se zaista izračunava uslov za skok instrukcije *bnez RI, -2*, koja se u ovom ciklusu signala takta nalazi u MEM

stepenu *pipeline*-a. Pošto se utvrdi da će skoka biti (*taken*), instrukcije koje su u međuvremenu učitane i nalaze se u IF, ID i EX stepenima *pipeline*-a su pogrešne i moraju biti poništene. To će biti urađeno tako što će u sledećem ciklusu signala takta biti isprani *pipeline* registri Rid, Rex i Rmem. Ono što će se takođe dogoditi, jeste da će u kešu za predikciju biti ažuriran ulaz, koji je nastao za instrukciju *bnez R1, -2* i predikcija skoka za ovu instrukciju će biti promenjena sa nema skoka na ima skoka. Prikaz rasporeda instrukcija u *pipeline*-u nakon sedamnaestog ciklusa signala takta nalazi se u tabeli 21.

	14	15	16	17	18
subi R1, R1, 1	ID	EX	MEM	WB	
bnez R1, -2	IF	ID	EX	MEM	WB
lw R1, R29, 0		IF	ID	EX	<i>Flush</i>
subui R29, R29, 1			IF	ID	<i>Flush</i>
jr R30, 1				IF	<i>Flush</i>
add R3, R3, R2					IF

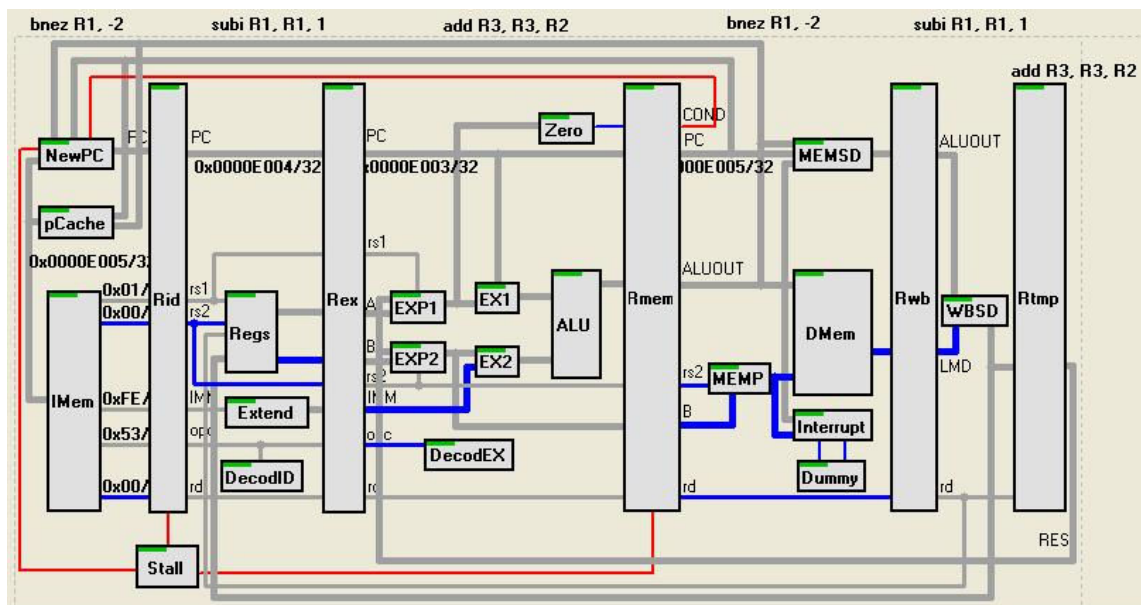
Tabela 21. Situacija u *pipeline*-u nakon sedamnaestog ciklusa signala takta

U dvadesetom ciklusu signala takta u IF stepenu *pipeline*-a ponovo je učitana instrukcija *bnez R1, -2*. Ovoga puta predikcija za ovu instrukciju, koju dobijamo iz prediktora skoka je da će skoka biti (*taken*). Zbog toga se u sledećem ciklusu signala takta u IF stepen učitava prva instrukcija sa adrese skoka, a ne naredna sekvencijalna instrukcija. Situacija u dvadesetprvom ciklusu signala takta prikazana je u tabeli 22.

	18	19	20	21
add R3, R3, R2	IF	ID	EX	MEM
subi R1, R1, 1		IF	ID	EX
bnez R1, -2			IF	ID
add R3, R3, R2				IF

Tabela 22. Situacija u *pipeline*-u u dvadesetprvom ciklusu signala takta

Ovaj put kada instrukcija *bnez R1, -2* dođe u MEM stepen *pipeline*-a uslov skoka je ponovo ispunjen, što znači da je predikcija ovoga puta bila dobra i da se u stepenima IF, ID, i EX nalaze korektne instrukcije, kao što je to prikazano na slici 68.



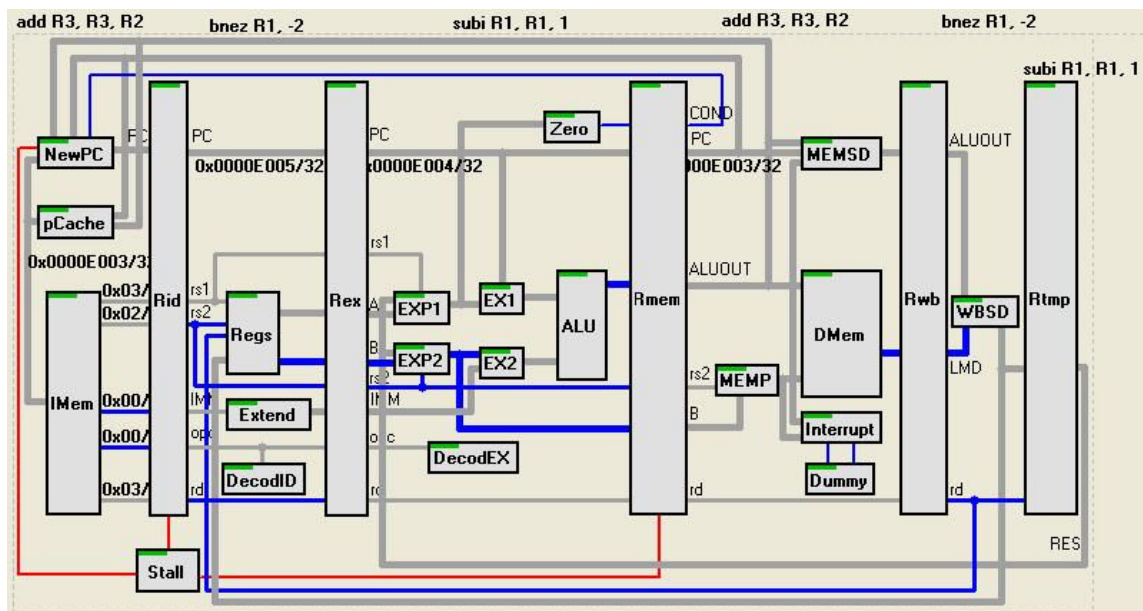
Slika 68. Situacija u *pipeline*-u u 23. ciklusu signala takta

U istom taktu, u ovom primeru se u IF stepenu *pipeline*-a ponovo pojavljuje instrukcija *bnez R1,-2*, za koju je predikcija i dalje da će biti skoka (*taken*). Raspored instrukcija u *pipeline*-u u 23. ciklusu signala takta prikazan je u tabeli 23.

	18	19	20	21	22	23
add R3, R3, R2	IF	ID	EX	MEM	WB	
subi R1, R1, 1		IF	ID	EX	MEM	WB
bnez R1, -2			IF	ID	EX	MEM
add R3, R3, R2				IF	ID	EX
subi R1, R1, 1					IF	ID
bnez R1, -2						IF

Tabela 23. Situacija u *pipeline*-u u 23. ciklusu signala takta

Kako je predikcija skoka ponovo da će ga biti, u narednom taktu se u IF stepen opet učitava prva instrukcija sa adrese skoka, umesto prve sekvencijalne instrukcije, kao što je to prikazano na slici 69.



Slika 69. Situacija u *pipeline*-u u 24. ciklusu signala takta

Međutim, ovaj put kada instrukcija *bnez R1, -2* dođe u MEM stepen *pipeline*-a, uslov za skok nije ispunjen. To znači da smo imali skok, koji nije bio izvršen i da se sada moraju preduzeti operacije kako bi se otklonile posledice neispravnog skoka. Pošto su posledice neispravne instrukcije u stepenima IF, ID i EX, to znači da ćemo ih ispraviti ukoliko u sledećem ciklusu signala takta isperemo *pipeline* register Rid, Rex i Rmem. Takođe će u ulazu keš memorije za predikciju skoka, koji je korespondentan sa instrukcijom *bnez R1, -2* biti promenjena predikcija sa ima skoka, na nema skoka. Prikaz rasporeda instrukcija u *pipeline*-u nakon 27. ciklusa signala takta dat je u tabeli 24.

	22	23	24	25	26	27
subi R1, R1, 1	IF	ID	EX	MEM	WB	
bnez R1, -2		IF	ID	EX	MEM	WB
add R3, R3, R2			IF	ID	EX	Flush
subi R1, R1, 1				IF	ID	Flush
bnez R1, -2					IF	Flush
lw R1, R29, 0						IF

Tabela 24. Situacija u *pipeline*-u u 27. ciklusu signala takta

Nakon toga izvršavanje potprograma se nastavlja sekvencijalno sve do 32. ciklusa signala takta u kome instrukcija *jr R30, 1*, koja predstavlja povratnik iz prekidne rutine, dolazi u MEM stepen *pipeline*-a. Kako je ovo instrukcija bezuslovnog skoka u narednom ciklusu signala takta dolazi do ispiranja *pipeline* registara Rid, Rex i Rmem i do učitavanja instrukcija sa

adrese skoka. Povratak iz potprograma je realizovan tako da se izvršavanje nastavlja sa prvom instrukcijom koja sledi poziv potprograma u glavnom programu. Sve ovo je realizovano softverski. Situacija u *pipeline*-u u 33. ciklusu signala takta je prikazana u tabeli 25.

	28	29	30	31	32	33
subui R29, R29, 1	IF	ID	EX	MEM	WB	
jr R30, 1		IF	ID	EX	MEM	WB
add R5, R1, R2						IF

Tabela 25. Situacija u *pipeline*-u u 33. ciklusu signala takta

Nakon povratka iz potprograma primer se do kraja izvršava sekvencijalno i završava se u 40. ciklusu signala takta. Stanje od 33. ciklusa signala takta do 40. ciklusa signala takta prikazano je u tabeli 26.

	33	34	35	36	37	38	39	40
add R5, R1, R2	IF	ID	EX	MEM	WB			<i>End</i>
xori R6, R5, 255		IF	ID	EX	MEM	WB		<i>End</i>
and R7, R5, R6			IF	ID	EX	MEM	WB	<i>End</i>

Tabela 26. Situacija u *pipeline*-u od 33. do 40. ciklusa signala takta

7. ZAKLJUČAK

U zaključku se daje kritički osvrt na ispunjenje ciljeva postavljenih u ovom radu. Analizira se šta je urađeno i daju se predlozi šta bi još moglo biti urađeno.

Napravljen je pregled koncepta procesora *pipeline* organizacije. Ova struktura je izabrana, kao često korišćena organizacija u savremenim procesorima, koja služi za poboljšanje njihovih performansi. Dat je kompletan opis svih koncepta *pipeline* organizacije, na koncizan i sažet način, pogodan za njihovo razumevanje. Objašnjeni su hazardi, koji ovu strukturu čine složenom, kao i tehnike za njihovo razrešavanje. Prikazane su i različite mogućnosti realizacije nekih elemenata, kao što je hardver za predikciju skoka.

Napravljen je pregled postojećih simulatora računarskih sistema, sa posebnim osvrtom na simulatore *pipeline* procesora. Pregled je kratak i bez mnogo detalja, ali su pokriveni svi dominantni simulatori u upotrebi za svrhu nastave, a i za komercijalne upotrebe. Prikaz postojećih simulatora *pipeline* procesora, poslužio je kao odrednica za izbor odgovarajućeg alata za realizaciju simulatora.

Realizovan je simulator *pipeline* procesora u alatu *IGoVSoEDS*, ali tako da se neki delovi mogu lako realizovati na različite načine (*test bed*). Realizovani simulator *pipeline* procesora je potpuno verodostojan i ispravno funkcioniše. Takođe, ovaj simulator je realizovan najpre na blokovskom nivou, pa zatim sve do nivoa prekidačke mreže, s tim da alat *IGoVSoEDS* dozvoljava da se bilo koji od blokova realizuje na bilo koji drugi način, kao poseban modul i da se jednostavno zameni umesto postojećeg. Na taj način se postiže efekat *test bed*-a za dalja istraživanja studenata. Ono što je autor uočio kao manu, jeste da, iako složena struktura i iako modularno realizovan, *pipeline* procesor ima jedan blok (keš za predikciju skoka), koji ima smisla realizovati na druge načine. Za sve ostale blokove, nije veliki izazov isprobavati druge realizacije, jer su funkcionalnosti ovih blokova utvrđene postaljenom arhitekturom i organizacijom procesora. Naravno, kreativnim studentima bi ovakva situacija mogla biti podstrek, da umesto na nivou modula, izvrše promene i u organizaciji, isprobavajući tako različite varijante prikazanog modela *pipeline* procesora.

Na kraju, autor nakon upotrebe *IGoVSoEDS* alata, može dati njegovu procenu, kao i prednosti i nedostatke ovog alata. Opšti utisak autora je da je alat veoma dobar za realizaciju, kako jednostavnijih, tako i složenijih struktura. Takođe, alat je prilično jednostavan za upotrebu i ima dosta detaljno i dobro napisano uputstvo, koje pokriva sve mogućnosti ovog alata. Nedostaci alata, koje je uočio autor, mogu se podeliti u dve grupe: konceptualni i tehnički nedostaci. Pod konceptualnim nedostacima podrazumevaju se funkcionalnosti, koje nedostaju, a koje bi bile korisne, dok se pod tehničkim nedostacima podrazumevaju greške u izvršavanju programa i ograničenja, koja otežavaju korišćenje. Autor je uočio sledeće

konceptualne nedostatke:

- ne postoji tabelarni pregled važnijih signala (trebalo bi obezbediti neki sličan vremenskom dijagramu signala), u koji bi se mogli dodati željeni signali i vrednosti registara, koje korisnik želi da prati,
- ne postoji podrška za dovođenje izlaza jednog kola na ulaz drugog, ukoliko je već izlaz drugog kola doveden na ulaz prvog (npr. kao kod flip-flopova) i nema rešenja za ovakvu situaciju,
- nema podrške za rotiranje elemenata na šemi, što je jako korisno kod ovakve vrste simulatora,
- nema podrške za pomeranje grupe elemenata.

Autor je uočio sledeće tehničke nedostatke:

- širina ulaza za podatke memorije je ograničena na maksimalno 16 bita, što može predstavljati otežavajuću okolnost u određenim situacijama,
- takođe, širina adresnog ulaza memorije je fiksirana na 16 bita i zbog ovakvih ograničenja, autor je morao da improvizuje memorije većeg kapaciteta, koristeći postojeće.
- na izlazu memorijskog modula se pojavljuje sadržaj sa adrese dovedene na adresni ulaz, na sledeći signal takta, što je dovelo do poteškoća u korektnom izvršavanju primera (morala se veštački dovoditi prava vrednost na izlaz),
- nema podrške za postavljanje Undefined signala na IN port modula, što je logično kod pravljenja složenijih struktura, ali isto tako i moguće rešiti zaobilaznim putem.

Pored ove dve grupe nedostataka, koje imaju veze sa samim simulatorom, postoji još jedan, koji je vezan za programsku realizaciju simulatora. Naime, kada se dogodi neka greška u projektovanju šeme ili izvršavanju simulacije, simulator ispiše poruku o grešci, koja je čisto tehničke prirode i nerazumljiva je običnom korisniku, ali još važnije, simulator se nakon toga samo ugasi, bez mogućnosti da korisnik sačuva svoj prethodni rad, što je jako loše, jer kod ovakvih vrsta programa, korisnik retko razmišlja o tome da često sačuva svoj rad, pa može doći do gubitka podataka i vraćanja korak u nazad u realizaciji nekog projekta.

Kada govorimo o dobrim stranama, to je već pomenuta jednostavnost u korišćenju, zatim dobar korisnički interfejs, sa dobrim rasporedom najčešće korišćenih komandi. Takođe, jako dobra strana je biblioteka sa osnovnim logičkim elementima, kao i mogućnost dodavanja novih modula u biblioteku. Zatim, u šemama je moguće prikazivati module sa ili bez portova, ali ipak povezivati "nevidljivim" vezama ovakve module, kao što se to obično radi u šemama logičkih struktura, kada je preveliki broj linija na šemi. Takođe, moguće je ograničiti pregled na nivou blokova, sa mogućnošću dozvole ili zabrane ulaženja u strukturu nekog bloka, pa se

na taj način može napraviti simulator sa specifičnom namerom da demonstrira koncepte i/ili realizaciju koncepata. Prostorna uređenost je odlična, tako da jako složene šeme, mogu biti prikazane veoma jasno i pregledno. Simulacija se izvodi veoma jednostavno i po subjektivnom utisku autora precizno i korektno. Autor je uspeo da izvrši simulacije, koje su dale ispravne rezultate na procesoru, koji je realizovan u okviru ovog rada, kao što je i opisano ranije.

Krajnja ocena autora je da je korišćeni alat jako dobar i da uz ispravljanje nekih ili svih nedostataka može postati još bolji. Želja autora je da napravljeni simulator *pipeline* procesora, pomogne studentima da lakše shvate koncepte *pipeline* organizacije i da eksperimentišući sa ovim simulatorom dođu do novih ideja za poboljšanje performansi procesora.

8. LITERATURA

1. J. Đorđević, *Arhitektura i organizacija računara: učenje pomoću računara*, Elektrotehnički fakultet Beograd, 2004.
2. IEEE Computer Society/ACM Computing Curriculum - Computer Engineering, "Computing Curricula - Computer Engineering (CCCE) Task Force," Available from: <http://www.eng.auburn.edu/ece/CCCE/> (accessed August 2005).
3. N. Grbanović, *Interaktivni generator vizuelnih simulatora digitalnih sistema (IGoVSoDS)*, doktorska disertacija u izradi, Elektrotehnički fakultet, Beograd, 2008.
4. N. Grbanović, B. Nikolić, J. Đorđević, *THE VSDS ENVIRONMENT BASED LABORATORY IN COMPUTER ARCHITECTURE AND ORGANISATION*, Elektrotehnički fakultet Beograd, 2008.
5. B. Nikolić, J. Đorđević, N. Grbanović, *Vizuelni simulatori u nastavi arhitekture i organizacije računara*, Zbornik radova ETRAN 2006, Beograd, Srbija, 2006.
6. N. Grbanović, B. Nikolić, J. Đorđević, *Vizuelni simulatori digitalnih modula*, Zbornik radova ETRAN 2006, Beograd, Srbija, 2006.
7. J. L. Hennessy, D. A. Patterson, *Computer Architecture: A Quantitative Approach*, second edition, Morgan Kaufmann, 1996.
8. J. Đorđević, *Arhitektura i organizacija računara: Pipeline*, Elektrotehnički fakultet Beograd, 1997.
9. B. Nikolić, Z. Radivojević, J. Đorđević, V. Milutinović, *A Survey and Evaluation of Simulators Suitable for Teaching Courses in Computer Architecture and Organization*, Elektrotehnički fakultet u Beogradu, 2008.
10. A. Stojković, J. Đorđević, B. Nikolić, *WASP – A WEB BASED SIMULATOR FOR AN EDUCATIONAL PIPELINED PROCESSOR*, Elektrotehnički fakultet Beograd, 2006.
11. A. Stojković, J. Đorđević, *Arhitektura i organizacija računara: processor RISC arhitekture i pipeline organizacije*, magistarski rad, Elektrotehnički fakultet, 2007.
12. Ž. Stanisavljević, *GRAFIČKI SIMULATOR VIRTUELNE MEMORIJE STRANIČNE ORGANIZACIJE SA JEDINICOM ZA DIREKTNO PRESLIKAVANJE*, diplomski rad, Elektrotehnički fakultet u Beogradu, 2007.
13. Harvard University, "Welcome to the Ant Home Page," Available from: <http://www.ant.harvard.edu/> (accessed August 2005).
14. G. B. Adams III, "Casle," Available from: <http://shay.ecn.purdue.edu/~casle/> (accessed

June 2004).

15. Texas Instruments, "Code Composer Studio User's Guide," Available from: <http://focus.ti.com/lit/ug/spru328b/spru328b.pdf> (accessed August 2008), February 2000.

16. Freescale Semiconductor, "CodeWarrior Development Tools," Available from: www.freescale.com/codewarrior (accessed August 2008).

17. D. Skrien, "CPU Sim Home Page," Available from: <http://www.cs.colby.edu/djskrien/CPUSim/> (accessed April 2008).

18. A. Cohen, and O. Temam, "DigLC2 - Gate-level LC-2 Simulator," Available from: <http://www-rocq.inria.fr/~acohen/teach/diglc2.html.en> (accessed June 2005).

19. G. Adamas, "DLXview v0.9 - Home Page," Available from: <http://yara.ecn.purdue.edu/~teamaaa/dlxview/> (accessed August 2005).

20. H.I.T.- Holon Institute of Technology, "Easy CPU," Available from: <http://www.hit.ac.il/EasyCPU/> (accessed May 2008).

21. P. Verplaetse, and J. V. Campenhout, "ESCAPE v1.1 Homepage," Available from: <http://trappist.elis.ugent.be/~hvdieren/escape/> (accessed May 2008).

22. A. R. Lebeck, and D. A. Wood, "FastCache," Available from: <http://www.cs.duke.edu/~alvy/fast-cache/> (accessed August 2005).

23. Institute for Computing Systems Architecture, School of Informatics, University of Edinburgh, "HASE - a computer architecture simulation environment," Available from: <http://www.icsa.inf.ed.ac.uk/research/groups/hase/> (accessed August 2005).

24. Xilinx Incorporated, "ISE Design Suite 10.1," Available from: http://www.xilinx.com/products/design_resources/design_tool/index.htm (accessed May 2008).

25. Brigham Young University, "JHDL Overview," Available from: <http://www.jhdl.org/overview.html> (accessed August 2005).

26. C. Burch, "Logisim," Available from: www.cburch.com/logisim/ (accessed May 2007).

27. N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "Main Page - M5," Available from: <http://www.m5sim.org> (accessed September 2007).

28. Altera Corporation, "Quartus II Software," Available from: <http://www.altera.com/products/software/products/quartus2/qts-index.html> (accessed May 2008).

29. M. Brorsson, MipsIt – A Simulation and Development Environment Using Animation for Computer Architecture Education, Proc. WCAE 2002, pp. 65-72

30. I. Branovic, R. Giorgi, E. Martinelli, WebMIPS: A New Web-Based MIPS Simulation Environment for Computer Architecture Education, Proc. WCAE 2004, pp. 93-98
31. L. B. Hostetler and B. Mirtich, *DLXsim – A Simulator for DLX*, Technical report, University of California at Berkeley, 1990
32. R. N. Ibbett, *HASE DLX Simulation Model*, IEEE Computer Society, IEEE Micro, Special Issue on Computer Architecture Education, Vol. 20, No. 3, pp. 57-65, May-June 2000