MREŽNO DOSTUPAN SISTEM ZA UČENJE ARHITEKTURE I ORGANIZACIJE PIPELINE PROCESORA

Aleksandar Stojković, *Institut "Mihajlo Pupin" – RJ "Računarstvo", Beograd*
Jovan Đorđević, Boško Nikolić, *Elektrotehnički fakultet u Beogradu*

Sadržaj – *U ovom radu opisan je softverski sistem za simulaciju procesora RISC arhitekture i pipeline organizacije, namenjen za laboratorijske vežbe na Elektrotehničkom fakultetu u Beogradu. Procesor je namenski projektovan, sa ciljem da arhitektura bude dovoljno ilustrativna a ipak jednostavna za implementaciju. Organizacija procesora razvijena je do RTL nivoa. Softverski sistem omogućava prikaz detalja organizacije na globalnom i RTL nivou, uz praćenje relevantnih vrednosti i dinamičko osvežavanje prikaza. Program koji je korisnik uneo ili zadao može se izvršavati za po jedan takt unapred ili unazad, u celini ili do zadate granice.*

1. UVOD

U savremenoj nastavi računarske tehnike bitna je uloga softverskih sistema za simulaciju, jer omogućavaju opažajni, praktični uvid u izučavane teme iz oblasti za koje je demonstriranje na stvarnim uređajima neostvarljivo. Time na jednostavan, prilagodljiv, jeftin i široko dostupan način omogućavaju povezivanje teorije i prakse. Koristeći takve softverske pakete studenti mogu primeniti i produbiti stečena znanja. Razvitkom učenja na daljinu sistemi za simulaciju naročito dobijaju na značaju, a s druge strane postavlja se zahtev da takvi sistemi budu prilagođeni korišćenju na daljinu putem Internet mreže.

Na Elektrotehničkom fakultetu u Beogradu izučavaju se brojne teme iz oblasti arhitekture i organizacije računara. U okviru Edukacionog računarskog sistema razvijeno je nekoliko softverskih paketa koji se već više godina koriste za laboratorijske vežbe. To su sistem za simuliranje procesora CISC arhitekture i klasične organizacije, za simuliranje hijerarhijskog memorijskog sistema, sistem za proveru znanja, kao i okruženje za projektovanje digitalnih sistema ([1], [2], [3]).

Bila je očevidna potreba da se ovaj skup dopuni sistemom za simulaciju procesora RISC arhitekture i *pipeline* organizacije, koji bi omogućio vizuelno demonstriranje svih bitnih tema u vezi takvih procesora koje se izučavaju u okviru nastave. Sistem treba da koriste studenti različitih smerova, koji izučavaju različite aspekte *pipeline* procesora: za studente računarskog smera, koji izučavaju projektovanje hardvera, bitni su detalji organizacije procesora na nivou logičkih kola, a za studente ostalih smerova bitan je globalni prikaz *pipeline* izvršavanja i mogućnost zadavanja asemblerskih programa.

Analiza nekoliko najpoznatijih javno dostupnih sistema za simulaciju *pipeline* procesora ([4]) pokazala je da ti sistemi ne zadovoljavaju postavljene zahteve. Najpre, nijedan od analiziranih sistema ne omogućava uvid u detalje organizacije. Analizirani sistemi ne bave se nekim mehanizmima bitnim za *pipeline* procesore (dinamičkom predikcijom uslovnih skokova, mehanizmom prekida itd.). Događaji specifični za *pipeline* izvršavanje, npr. zaustavljanje i brisanje, kao i uzroci tih događaja, tipično nisu dovoljno uočljivo prikazani. Nijedan od analiziranih sistema u aktuelnoj verziji nema potpuno funkcionalni *web* interfejs. Uočen je prostor za unapređenja i u pogledu arhitekture, koja za edukativne namene treba da bude što čistija, saglasna sa principima RISC arhitektura.

U daljem tekstu u poglavljima 2 i 3 dat je opis arhitekture i organizacije procesora, respektivno, u poglavlju 4 navedene su osnovne osobine softverskog sistema, u poglavlju 5 dat je primer simulacije, a poglavlje 6 je zaključak.

2. ARHITEKTURA

Arhitekturu procesora čine skup programski dostupnih registara, tipovi podataka, formati instrukcija, načini adresiranja, skup instrukcija i programski vidljivi aspekti mehanizma prekida. U projektovanoj arhitekturi definisana su 32 registra opšte namene, veličine po 32 bita. Vrednost registra R0 uvek je 0, što se koristi za jednostavnije ostvarivanje nekih operacija i načina adresiranja. U registar R31 upisuje se adresa povratka iz prekida. Registri posebne namene su 32-bitni programski brojač PC, i 10-bitni statusno-upravljački registar prekida ISCR (*Interrupt Status and Control Register*).

Podržane su 32-bitne celobrojne veličine sa i bez znaka. Adresiranje je na nivou 32-bitnih reči, a veličina memorijskog adresnog prostora je 2^{32} reči. Zastupljeni su svi načini adresiranja tipični za RISC arhitekture, npr. registarsko direktno, neposredno i registarsko indirektno sa ili bez pomeraja. Definisan je samo jedan format instrukcija, sa tri 5-bitna registarska argumenta i 8-bitnim neposrednim argumentom. Time je omogućeno jednostavnije dekodovanje instrukcija.

Arhitektura je *load-store* tipa, što znači da se memoriji pristupa samo pomoću instrukcija *load* i *store*. Pri definisanju sintakse instrukcija insistiralo se da instrukcije sa sličnim efektima imaju i sličnu sintaksu, kako bi se omogućila jednostavnija implementacija. U okviru grupe ALU instrukcija definisane su aritmetičke (sabiranje i oduzimanje sa i bez znaka), logičke (**and**, **or**, **xor**), relacione (svih 6 osnovnih relacija) i pomeračke (aritmetička i logička pomeranja, rotiranja) instrukcije. Za aritmetičke, logičke i relacione instrukcije postoji registarska i neposredna varijanta, u zavisnosti od toga da li se drugi izvorišni operand nalazi u registru opšte namene ili je zadat neposrednim argumentom. Ovoj grupi pridružene su i instrukcije punjenja konstantom, jer su slične logičkoj operaciji *ili* sa neposrednim argumentom.

Uslovni skokovi su relativni u odnosu na programski brojač, a uslov je jednakost ili nejednakost sa nulom. U grupu bezuslovnih skokova spadaju registarski bezuslovni skok (**jr**) i registarski skok na potprogram (**jsr**), a takođe i programski prekid (**trap**) jer, kao i bezuslovni skokovi, dovodi do upisa nove vrednosti u programski brojač. Pošto sve instrukcije skoka imaju sličnu sintaksu i izračunavaju novu vrednost programskog brojača na sličan način, može se koristiti zajednički mehanizam za upis nove vrednosti u programski brojač. Za povratak iz potprograma i povratak iz prekida ne postoje posebne instrukcije već se koristi instrukcija **jr**.

Instrukcije za pristup posebnom registru ISCR su po sintaksi slične instrukcijama za pristup memoriji, i zato se pri implementaciji mogu uklopiti u već postojeće mehanizme. Stek se realizuje softverski. Po dogovoru, preporučuje se da se kao pokazivač steka koristi registar opšte namene R29, tako da pokazuje na poslednju zauzetu lokaciju, i da stek raste na gore. Adresu povratka iz potprograma, po dogovoru, treba upisivati u registar opšte namene R30.

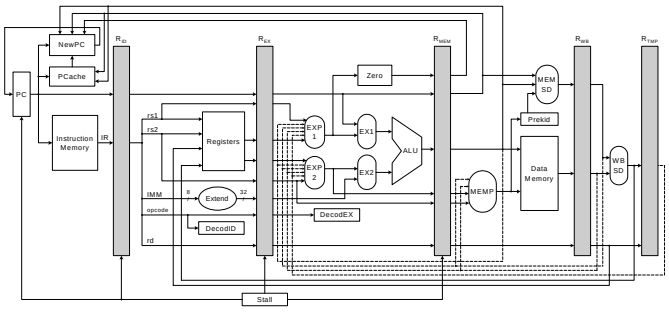
Podržani su svi osnovni tipovi prekida kao kod procesora CISC tipa: programski prekid (TRAP, izazvan istoimenom instrukcijom), prekid zbog pogrešnog koda operacije (UOE – *Unexistent Opcode Exception*), prekoračenje (ARE – *Arithmetic Exception*), spoljašnji nemaskirajući prekid (NMI – *Non-Maskable Interrupt*), spoljašnji maskirajući prekid (MI – *Maskable Interrupt*) i režim prekid posle svake instrukcije (TRM – *TRap Mode*). Tipovi prekida ovde su navedeni po opadajućem redosledu prioriteta.

3. ORGANIZACIJA

Procesor je organizovan kao sinhroni statički *pipeline* (sl. 1) koji se sastoji od tradicionalnih 5 stepeni: IF (*Instruction Fetch*), ID (*Instruction Decode & register fetch*), EX (*Execution & address calculation*), MEM (*MEMory access & branch completion*) i WB (*Write Back*). Najvažnije jedinice u stepenu IF su memorija za instrukcije (*Instruction Memory*) i jedinice za određivanje sledeće vrednosti programskog brojača; u stepenu ID registar fajl (*Registers*) i jedinica *Extend* koja proširuje neposredni argument nulom ili znakom; u stepenu EX aritmetičko-logička jedinica *ALU* i jedinica *Zero* za proveru uslova skoka; u stepenu MEM memorija za podatke (*Data Memory*) i jedinica *Prekid*; u stepenu WB multiplekser *WBSD* za izbor vrednosti koja se upisuje u registar fajl.

Podaci se prenose iz stepena u stepen kroz *pipeline* registre R_{ID}, R_{EX}, R_{MEM}, R_{WB} i R_{TMP}. Svaki *pipeline* registar sadrži podatke potrebne za izvršavanje instrukcije u tekućem i narednim stepenima. Programski brojač PC predstavlja *pipeline* registar za stepen IF.

Činioci koji otežavaju projektovanje *pipeline* organizacije su hazardi i prekidi. Hazardi mogu biti strukturalni, hazardi podataka i upravljački. Strukturalni hazardi su u projektovanom procesoru izbegnuti time što se koriste posebne memorije za instrukcije i za podatke, i time što registar fajl dozvoljava istovremeno čitanje iz dva i upis u jedan registar.



Sl. 1. *Organizacija procesora*

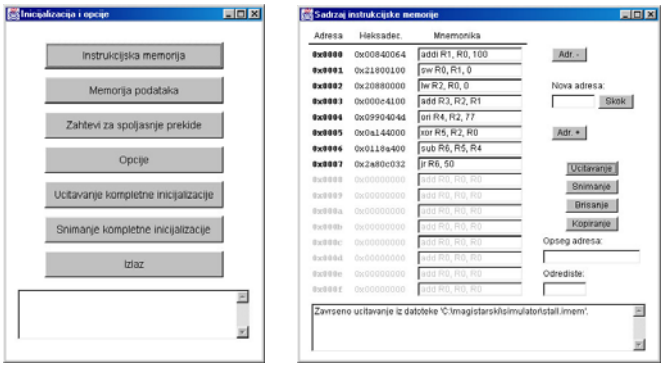
Od tri vrste hazarda podataka, u projektovanom procesoru može se javiti samo RAW (*read after write*) hazard, dok se WAR (*write after read*) i WAW (*write after write*) hazardi ne mogu javiti. Za razrešavanje RAW hazarda koriste se tehnike prosleđivanja i zaustavljanja. Prosleđivanjem se podatak vodi iz *pipeline* registara na ulaz jedinice u kojoj je potreban. Prosleđivanje u stepen EX može se vršiti iz *pipeline* registara R_{MEM}, R_{WB} i R_{TMP}, a u stepen MEM iz *pipeline* registra R_{WB}. Jedinica *EXP1* omogućava prosleđivanje na ulaz jedinice *Zero* i ulaz A jedinice *ALU*, jedinica *EXP2* na ulaz B jedinice *ALU* i u polje $R_{MEM}.B$, a jedinica *MEMP* na ulaz podataka jedinice *Data Memory* i na ulaz jedinice *Prekid*. Ako podatak još nije generisan u taktu u kome je potreban nekoj instrukciji, ta instrukcija mora se zaustaviti. Zaustavljanje za jedan takt se vrši kada je ta instrukcija u stepenu ID, jer zaustavljanjem u stepenu EX ne bi mogao da se obezbedi korektan rad u svim situacijama ([4]).

Postoje brojne tehnike za razrešavanje i/ili ublažavanje efekata upravljačkih hazarda. U projektovanom procesoru primenjena je tehnika dinamičke predikcije ishoda i odredišta skoka. Koristi se keš za predikciju, u kome se čuvaju adrese instrukcija uslovnog skoka za koje se pri poslednjem izvršavanju skok dogodio, i odredišne adrese skokova. U stepenu IF proverava se da li u kešu postoji ulaz za tekuću instrukciju, i u slučaju pogotka na sledeći signal takta u PC se upisuje odredišna adresa skoka. Ako se u stepenu MEM ispostavi da je predikcija bila pogrešna, keš se ažurira a *pipeline* ispira.

4. SOFTVERSKI SISTEM

Zbog zahteva da bude mrežno dostupan i nezavisan od platforme, softverski sistem je realizovan kao aplet u programskom jeziku Java. Sistem se sastoji od obrazaca za inicijalizaciju i simulacionog okruženja. Pomoću obrasca *Inicijalizacija i opcije* (sl. 2 levo) mogu se pokrenuti obrasci za zadavanje sadržaja instrukcijske memorije, memorije podataka, zahteva za spoljašnje prekide i opcija. Kompletna inicijalizacija sistema može se učitati iz datoteke ili snimiti u datoteku, što omogućava da se simulacija kasnije ponovi ili nastavi od takta u kome je prekinuta.

Obrazac *Sadržaj instrukcijske memorije* (sl. 2 desno) omogućava zadavanje jedne po jedne instrukcije uz automatsku proveru i unifikaciju sintakse i pretvaranje u heksadecimalni oblik. Sadržaj kompletne instrukcijske memorije ili nekog njenog dela može se učitati, snimiti, obrisati ili iskopirati u neki drugi deo memorije. Obrazac *Sadržaj memorije podataka* (sl. 4 levo) funkcioniše na sličan način, a omogućava još i popunjavanje memorijskih lokacija zadatom ili slučajnim vrednostima. Pomoću obrasca *Zahtevi za spoljašnje prekide* može se zadati u kojim taktovima će se javiti zahtevi za prekide tipa NMI i/ili MI. Obrazac *Opcije* (sl. 4 desno) omogućava zadavanje početne adrese programa, kriterijuma za završetak programa, adrese podrazumevane prekidne rutine i izbor situacija u kojima se prikazuju upozorenja. Oba obrasca omogućavaju učitavanje i snimanje.



Sl. 2. *Obrasci Inicijalizacija i opcije i Sadržaj instr. memorije*

Pokretanjem simulacije prikazuju se prozori *Ceo sistem* i *Glavni prozor* (sl. 5). U prozoru *Ceo sistem* prikazuje se globalna organizacija sistema. Svakoj jedinici vidljivoj na globalnom nivou, uključujući i *pipeline* registre, dodeljeno je dugme pomoću koga se mogu prikazati detalji realizacije te jedinice. Svakoj instrukciji dodeljena je boja, koju ona zadržava tokom celog kretanja kroz *pipeline*. Sve linije koje potiču od neke instrukcije prikazane su istom bojom kao ta instrukcija. Instrukcije bez dejstva, koje odgovaraju praznim stepenima *pipeline*-a, prikazane su manjim slovima i sivom bojom. Uz većinu magistralnih linija prikazana je njihova vrednost. Linije za signale prikazane su plavom ili crvenom bojom ako je signal aktivan ili neaktivan, respektivno. Izuzetak su linije za signale takta (zelenom ili crvenom) i signale za brisanje (sivom ili roze bojom, respektivno). Obaveštenja o karakterističnim događajima prikazuju se kao

upozorenja (*alert*, sl. 5, 6) i u polju *Informacije* na *Glavnom prozoru*. Prosleđivanja su dodatno naglašena promenom boje odgovarajućih jedinica i njihovih izlaznih linija.

Glavni prozor omogućava navigaciju i kontrolu simulacije. U polju *Informacije* prikazuju se obaveštenja. U koloni *Status* prikazan je broj takta i vrednosti registara PC i ISCR. Klikom na dugme **Hijerarhija** prikazuje se hijerarhija sistema u novom prozoru. Dugme **Ceo sistem** omogućava prikaz prozora *Ceo sistem*. Dugme **Gore** omogućava prelazak u jedinicu višeg nivoa po hijerarhiji.

Klikom na dugme **Takt +** pokreće se izvršavanje programa za jedan takt unapred. Nakon izračunavanja novih vrednosti, svi otvoreni prozori se automatski osvežavaju. Dugme **Program +** pokreće izvršavanje programa do kraja. Prikazuje se samo završno stanje. Dugme **Takt –** omogućava povratak za jedan takt unazad. Dugme **Do granice** omogućava izvršavanje programa do zadatog graničnog takta i/ili vrednosti programskog brojača.

Klikom na dugme **Tabela** prikazuje se tabela situacije u *pipeline*-u, u kojoj su označena i zaustavljanja, brisanja i prosleđivanja. Klikom na dugme **Inic. i opcije** otvara se već opisani obrazac *Inicijalizacija i opcije*; korisnik može da menja sadržaj memorija, opcije i zahteve za spoljašnje prekide, da snimi tekuću ili učita novu simulaciju čak i dok je simulacija u toku. Dugme **Reset** omogućava vraćanje simulatora u stanje koje je zapamćeno kao početno ili u "nulto" stanje. Dugme **Izlaz** omogućava izlaz iz programa.

5. PRIMER SIMULACIJE

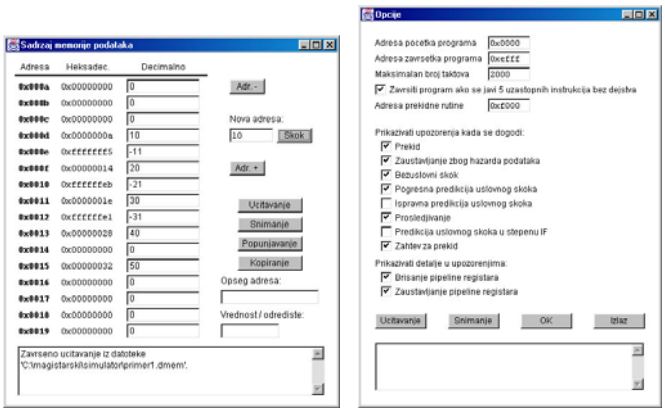
Ovaj primer demonstrira najpre normalan tok instrukcija u *pipeline*-u, zatim razrešavanje hazarda podataka prosleđivanjem, zaustavljanje *pipeline*-a, i na kraju posledice pogrešne predikcije uslovnog skoka. Delovi programa koji se izvršava prikazani su na sl. 3. Program i početni sadržaj memorije podataka (sl. 4 levo) se mogu uneti na odgovarajućim obrascima ili učitati iz datoteke. Koriste se podrazumevane vrednosti opcija (sl. 4 desno). Na početku simulacije registri opšte namene i keš za predikciju uslovnih skokova su prazni.

Adresa	Instrukcija	Komentar
0x0000 (0)	**addi** R1, R0, 15	R1 ← 15
0x0001 (1)	**subi** R2, R0, 1	R2 ← –1
0x0002 (2)	**xori** R3, R2, 77	R3 ← *not* 77
0x0003 (3)	**lw** R4, R0, 20	R4 ← DMem [20]
0x0004 (4)	**beqz** R4, 120	R4 = 0 ⇒ PC ← PC + 120
0x0005 (5)	**add** R5, R1, R2	R5 ← R1 + R2
0x0006 (6)	**ori** R6, R3, 12	R6 ← R3 | 12
0x0007 (7)	**and** R7, R5, R6	R7 ← R3 & 12
	...	
0x007C (124)	**sub** R8, R4, R1	R8 ← R4 – R1
	...	

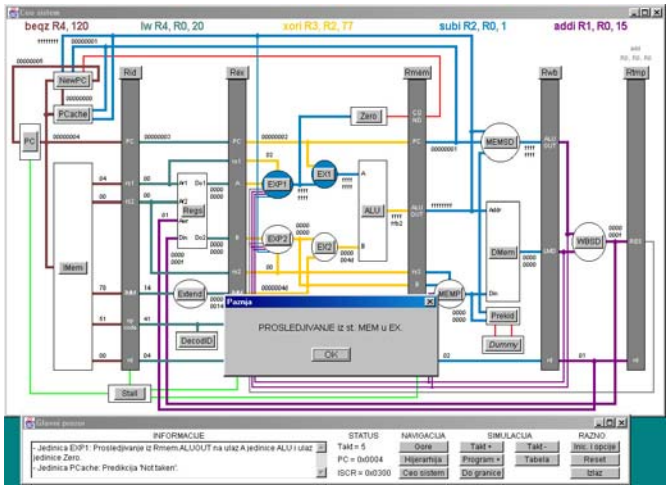
Sl. 3. *Delovi programa koji se izvršava*

Pipeline je na početku prazan. U taktu 1 prva instrukcija ulazi u *pipeline*. Izvršavanje najpre teče bez vanrednih događaja, u svakom taktu nova instrukcija ulazi u *pipeline*, i u taktu 5 (sl. 5) ceo *pipeline* je popunjen. U tom taktu vrši se prosleđivanje iz stepena MEM u EX radi razrešavanja hazarda podataka. U istom taktu u *pipeline* ulazi instrukcija uslovnog skoka, i predikcija ishoda je da se skok neće dogoditi.

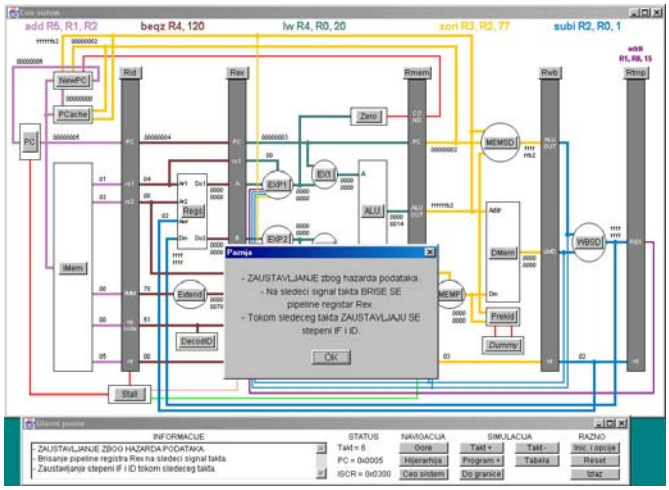
U taktu 6 (sl. 6) detektovan je hazard podataka koji se ne može razrešiti samo prosleđivanjem već zahteva zaustavljanje: neposredno iza instrukcije **lw** je instrukcija **beqz** kojoj će rezultat instrukcije **lw** biti potreban u stepenu EX. U toj situaciji može biti od interesa da se pogledaju detalji realizacije jedinice *Stall*, koja kontroliše zaustavljanje i brisanje *pipeline* registara. Klikom na dugme **Stall** prikazuju se detalji te jedinice (sl. 7). Stanje signala označeno je bojama linija.



Sl. 4. *Obrasci Sadržaj memorije podataka i Opcije*



Sl. 5. *Takt 5: Razrešavanje hazarda podataka prosleđivanjem*

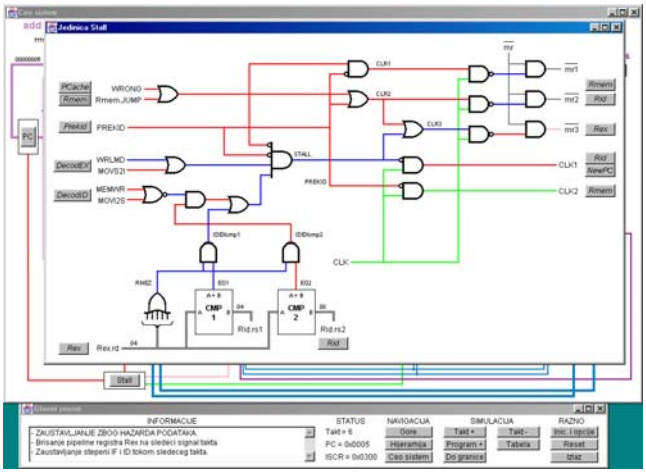


Sl. 6. *Takt 6: Situacija u kojoj je neophodno zaustavljanje*

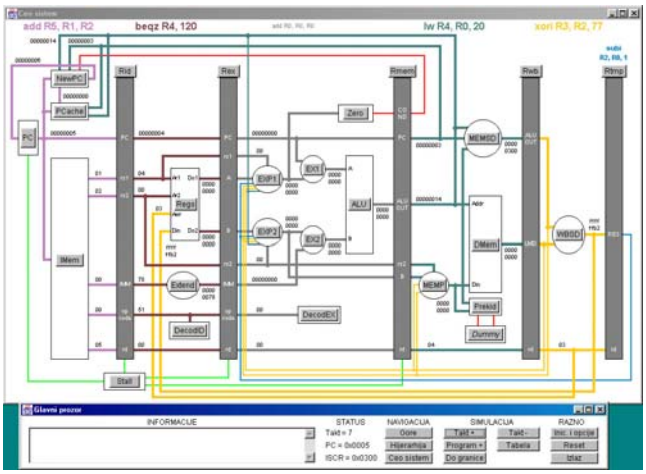
Zaustavljanje se izvodi na taj način što registar PC i *pipeline* registar R_{ID} ne dobijaju signal takta 7, a *pipeline* registar R_{EX} se briše na taj signal takta. Zato se u taktu 7 (sl. 8) instrukcija **lw** nalazi u stepenu MEM, instrukcije **beqz** i **add** ostale su u stepenima ID i IF, respektivno, a stepen EX je prazan. U taktu 8 rezultat instrukcije **lw** prosleđuje se instrukciji **beqz**. Prethodno izvršeno zaustavljanje omogućilo je da se hazard podataka razreši prosleđivanjem.

U taktu 9 (sl. 9) instrukcija **beqz** je u stepenu MEM. Uslov za skok je ispunjen, a predikcija u stepenu IF (u taktu

5) bila je da se skok neće dogoditi. Instrukcije u stepenima IF, ID i EX su pogrešno učitane i zato se brišu na sledeći signal takta. U taktu 10 u stepen IF ulazi instrukcija sa odredišne adrese skoka (0x007C), a tri stepena ispred te instrukcije (ID, EX i MEM) su prazna. To je poslednji takt koji se posmatra u ovom primeru. Neke od prikazanih instrukcija izvršile su upis u registre opšte namene. Kao posledica izvršenog uslovnog skoka u keš za predikciju dodat je zapis za instrukciju sa adrese 0x0004. Sadržaj registara opšte namene i keša za predikciju u taktu 10 prikazan je na sl. 10.



Sl. 7. *Detalji realizacije jedinice Stall*

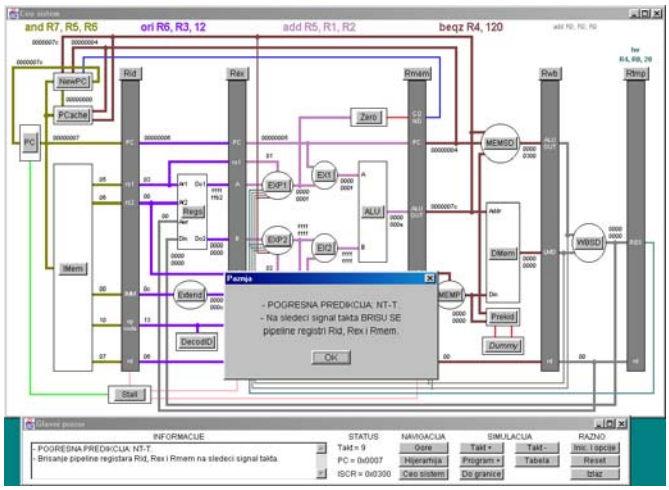


Sl. 8. *Takt 7: Instrukcije u stepenima IF i ID zaustavljene*

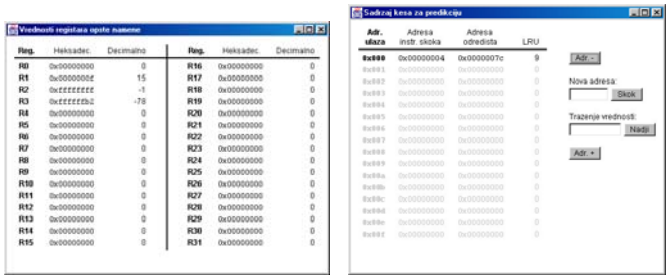
6. ZAKLJUČAK

Razvojem opisanog softverskog sistema omogućena je praktična demonstracija svih osnovnih principa *pipeline* procesora koji se predaju na ETF u Beogradu. Softverski sistem omogućava jednostavnu inicijalizaciju i simulaciju, uz praćenje rada procesora na globalnom nivou ili nivou standardnih logičkih kola i kombinacionih i sekvencijalnih modula. Namenski projektovana arhitektura je jednostavna a ipak dovoljno ilustrativna, što je ostvareno insistiranjem na pravilnosti. Detaljno razvijena organizacija je potpuno funkcionalna a ipak jednostavna za praćenje. Time je potvrđeno da je konzistentna arhitektura preduslov za efikasnu organizaciju.

Slični softverski sistemi koriste se u nastavi na ETF već više od deset godina, i vrlo dobro su prihvaćeni od studenata. Korišćenje takvih sistema rezultiralo je u kvalitetnijim odgovorima na laboratorijskim vežbama i boljim ocenama na ispitu ([1]). Postojeći sistem može biti unapređen povezivanjem sa sistemom za proveru znanja, tako da se tokom rada mogu postavljati pitanja u vezi trenutne situacije ili generalna pitanja u vezi RISC i *pipeline* procesora. Može se izraditi složenija varijanta sistema koja bi podržavala operacije čije izvršavanje traje više taktova, raznovrsne tehnike za razrešavanje hazarda i složeniju organizaciju memorije.



Sl. 9. *Takt 9: Pogrešna predikcija uslovnog skoka*



Sl. 10. Sadržaj registar fajla i keša za predikciju u taktu 10

LITERATURA

[1] J. Djordjevic, A. Milenkovic, N. Grbanovic, *An Integrated Environment for Teaching Computer Architecture*, IEEE Computer Society, IEEE Micro, Vol. 20, No. 3, pp. 66-74, May-June 2000.

[2] J. Djordjevic, A. Milenkovic, S. Prodanovic, *A Hierarchical Memory System Environment*, IEEE Computer Society Technical Committee on Computer Architecture Newsletter, pp. 60-62, March 1999.

[3] A. Milenkovic, B. Nikolic, J. Djordjevic, *CASTLE: Computer Architecture Self-Testing and Learning System*, Proc. IEEE/ACM HPCA-02 Workshop on Computer Architecture Education, Anchorage, May 2002.

[4] A. Stojković, *Mrežno dostupan sistem za učenje arhitekture i organizacije procesora sa preklopljenim izvršavanjem instrukcija*, Magistarski rad, ETF Beograd, 2005.

Abstract – This paper describes a software system for simulation of a RISC pipeline processor, used in computer architecture and organization courses. Designed processor architecture tends to be illustrative yet easy to implement. Organization was developed down to register transfer level. Software system displays global organization as well as details on RT level, with relevant values and dynamic refreshment. Programs can be executed clock by clock forward or backward, completely or until given condition.

WEB-BASED SYSTEM FOR TEACHING ARCHITECTURE AND ORGANIZATION OF PIPELINE PROCESSORS

Aleksandar Stojković, Jovan Đorđević, Boško Nikolić