Univerzitet u Kragujevcu
Tehnički fakultet u Čačku

MAGISTARSKI RAD

PRILOG SIMULACIJI RAČUNARSKIH ARHITEKTURA

Kandidat:
Stanković Nebojša, dip. ing. el.

Mentor:
prof. dr Siniša Ranđić

Čačak, 2009.

SADRŽAJ

1

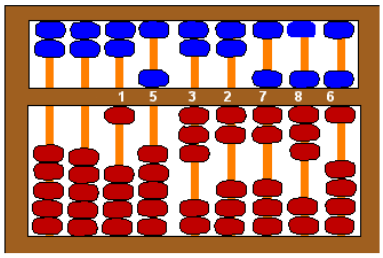
1. UVOD

Jedna od osnovnih karakteristika 20. i 21. veka je ekspanzija računara i računarske tehnike uopšte. Ovaj razvoj kretao se i kao evolucioni i kao revolucionarni proces. Evolucija se ogledala u stalnoj promeni komponenata računarskog sistema i to od jednostavnijih ka složenijim. Ova promena je omogućila pre svega brži rad računara kao i mogućnosti obrade različitih tipova podataka. Revolucija se ogleda u primeni računarske tehnike u raznim multidisciplinarnim oblastima nauke i tehnike gde se, zahvaljujući računarima, preko noći menjalo shvatanje i otvarali novi pravci za proučavanje i primenu, a takođe na bazi toga nastale su i nove naučne i stručne discipline.

Sa razvojem računara javila se i potreba da se osoblje na različitim nivoima (od korisničkog pa do dizajnerskog) upoznaje sa funkcionalnostima koje računari poseduju. Radi toga, paralelno sa razvojem računara razvijala se i tehnika simulacije računarskog sklopa i/ili pojedinih njegovih delova sa ciljem da se ilustrativno dočara «kako to računar radi». Simulacije računarskog sistema su obično imale sledeće dve uloge: da edukuju osoblje kako bi ono razumelo šta se u sistemu dešava i da omogući istraživanja za dalji razvoj korišćenjem simulacija određenih situacija koje su od interesa za računarski sklop. Tako je istorijat razvoja simulacije računarskih sistema tesno vezan za razvoj samih računara.

Od samih davnina ljudi su imali potrebu za računanjem. Čak i u vreme kada se trgovina svodila na prostu razmenu dobara (bez novca) ljudi su nalazili načine da prikažu vrednosti svojih dobara po principu poređenja. Takođe vršili su osnovne operacije koje su se svodile na sabiranje, oduzimanje i sl. Da bi te operacije ubrzali i na neki način dugoročnije zabeležili težili su da pronađu uređaj koji bi to i omogućio. Kao jedan od

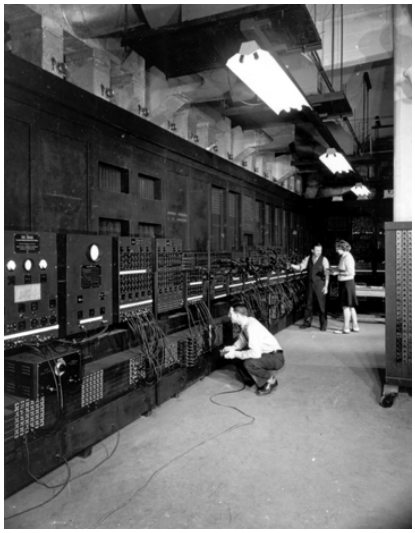
najstarijih "uređaja", koji su drevni narodi koristili za pamćenje brojeva, smatra se „kipu" i on je radio pomoću kanapa sa čvorovima. Sledeći korak bilo je pravljenje prvog "personalnog kalkulatora" - abakusa (abacus) - koji koristi isti koncept, da jedan skup objekata zamenjuje drugi skup, ali i koncept da jedan objekat zamenjuje kolekciju objekata - poziciono označavanje (slika 1.1).



Slika 1.1 Abakus

Abakus potiče još od 2600. godine pre Hrista i napravili su ga Kinezi. Sastojao se od kuglica nanizanih na žice, nalik današnjoj računaljci. Od pojave logaritama 1621. godine, on je bio isključivi alat matematičara dok tu ulogu nije preuzeo elektronski kalkulator u ranim sedamdesetim godinama prošlog veka. Svi ovi rani pokušaji manipulisanja brojevima imali su dve zajedničke osobine: oni su bili mehanički i bili su u ravni prosečnih ljudskih znanja. Blejz Paskal (Blaise Pascal) [20], poznati matematičar, naučnik i teolog, smatra se ocem prve računske mašine (1642.), koja je koristila sistem zupčanika koji su se ručno pokretali. Početkom devetnaestog veka, poznati engleski matematičar Čars Bebidž (Charles Babbage) je osmislio prvu računarsku mašinu, čiji koncept podseća na modernu računarsku arhitekturu.

Prava prekretnica u razvoju modernih računara dogodila se 1904. godine kada je Džon Ambroz Fleming (John Ambrose Fleming), napravio prvu komercijalnu diodnu vakumsku cev. Značaj vakumske cevi je u tome što je njome načinjen prvi korak u stvaranju mašina koje prevazilaze prosečna ljudska znanja. Do tada, izračunavanja su se obavljala pomoću zupčanika, a kasnije pomoću prekidača. Posle mehaničkih mašina koje su služile za računanje vremenom se prešlo na takozvane elektromehaničke. Oni su u sebi imali i odgovarajući elektromotor. Prvi potpuno elektronski računar bio je ENIAC (Electronic Numerical Integrator Analyzer and Computer), koji je američka vojska napravila, između 1943. i 1945. godine, za izračunavanje tabela trajektorija raketa (slika 1.2).



Slika 1.2 ENIAC

ENIAC je obavljao 5.000 sabiranja u sekundi, mada je za neki problem, čije je rešavanje trajalo dve sekunde, zahtevao pripremu koja je trajala i do dva dana. ENIAC je koštao 500.000$, težio je 30 tona i bio je 30 metara dugačak i 3 metra visok. On je imao 1.500 releja i 17.468 vakumskih cevi. Godine 1945. Džon fon Nojman (John von Neumman) konsultant ENIAC projekta [16], je predložio računar koji bi imao neka poboljšanja u odnosu na prethodnike: memorija bi prihvatala i programe i podatke i imao bi binarnu obradu podataka. Na fon Nojmanovom principu konstruisana su dva računara: EDVAC na univerzitetu u Pelsilvaniji i IAS na Princenton univerzitetu. Oba računara su završena 1951.-1952. godine. U ovim računarima je prvi put primenjena memorija sa direktnim pristupom (RAM) za memorisanje programa i podataka, a programi su pisani u mašinskom jeziku.

Posle razvoja prvih elektronskih računara koji su bili namenjeni u vojne svrhe počinje razvoj i novih koji imaju komercijalnu upotrebu. Nastaje veliki bum u razvoju računara koji se do sada može podeliti u pet generacija[13].

Prvu generaciju predstavljaju računari proizvedeni pedesetih godina. Za njih je karakteristično da su koristili vakumske cevi kao aktivne elemente, a memorije su im bile u obliku magnetnih traka i doboša. Računari ove generacije su bili glomazni, trošili su veliku količinu električne energije i stvarali toplotu. Programiranje ovih mašina bilo je na mašinskom jeziku.

Za **drugu generaciju** računara je karakteristično da su proizvedeni krajem pedesetih i u prvoj polovini šezdesetih godina. Ovi računari se zasnivaju na tranzistorima. Iako je tranzistor pronađen 1948. godine, tek 1959. se stvorila tehnološka sitacija za njegovu primenu. Ovi računari su sadržali i do 10.000 tranzistora, trošili su manje

električne energije, razvijali manje toplote. Prvi od ovih računara je bio Philco Transac S-2000, a posebno veliki značaj ima IBM-ov računar 1401. Počeli su da se koriste i programski jezici kao što su: COBOL, FORTRAN, ALGOL i LISP.

Treća generacija računara ima novu tehničku inovaciju – primenu integrisanih kola. Tako je npr. 1961. godine bilo napravljeno integrisano kolo od četiri tranzistora, a 1968. integrisano kolo od 160 tranzistora. Uvođenjem novih tehnologija za integraciju omogućeno je kreiranje čipova i sa hiljade tranzistora, tako da je broj aktivnih komponenata u računaru prerastao sa oko 10.000 na preko pola miliona. Računari su postali manjih dimenzija, visoke pouzdanosti i nižih cena. Magnetna traka je zamenjena sa magnetnim diskom. Karakteristike i mogućnosti korišćenja su se znatno poboljšale. Karakterističan računar ove generacije je IBM 360 kao i prvi mini računar PDP-1 firme Digital Equipment Corporation.

Četvrta i peta generacija računara se ne mogu jasno odvojiti ali za njih je karakteristično da predstavljaju nadogradnju na treću generaciju i da se počela primenjivati tehnika vrlo visoke integracije (VLSI – Very Large Scale Integration). Ova tehnika omogućila je stvaranje mikroprocesora. On je nastao 1971. godine od strane kompanije Intel, a cilj je bio da sve osobine koje je posedovao centralni procesor jednog računara budu smeštene u jedan čip. Prvi mikroprocesor je bio četvorobitni i nosio je oznaku Intel 4004.

Slika 1.3 Procesor Intel 4004

Od tada pa do današnjih 64-bitnih procesora došlo je do mnogih promena u arhitekturi računara i njihovoj strukturi. Računari postaju komercijalni proizvod dostupan širokom krugu korisnika, a najznačajnije godine u razvoju personalnih računara su sledeće [14]: 1983. (IBM predstavlja računar IBM PC/XT sa procesorom 8086, memorijom do 256kB i diskom od 10MB), 1984. (IBM proizvodi računare AT bazirane na Intelovom procesoru 80286. Memorija je do 512kB, a disk do 20MB. Ovaj računar ima disketnu jedinicu od 360kB i 1,2MB), 1986 (proizvode se PC računari na bazi procesora Intel 80386), 1987. (IBM proizvodi takozvane PS/2 računare i prodaje milionski broj primeraka), 1989. (počinje prodaja računara bazirana na Intelovom procesoru 80486), 1993. (počinje isporuka računara bazirana na procesoru Pentium) i tako dalje. Savremeni

personalni računari su vrlo moćni i u poređenju sa onima od pre samo nekoliko godina imaju duplirane brzine, duplirane kapacitete RAM memorije i hard diskova i to za pristupačnu cenu. Sve više su u upotrebi višeprocesorske arhitekture (sa dva, četiri ili više procesora) koje mogu da rade brže i mnogo pouzdanije od klasičnih rešenja. Može se očekivati da će dalji razvoj računarskih sistema biti vrlo progresivan pri čemu već sada sve veću ulogu dobija nano tehnologija.

Današnji čovek se smatra „pismen“ ukoliko je kompjuterski obrazovan, tj. ako je u stanju da koristi računar na poslu i u svakodnevnom životu. Intezivno korišćenje informacionih tehnologija u raznim oblastima ljudske delatnosti, uticalo je na to da računari postanu sve prisutniji i u procesima edukacije. U uslovima «eksplozije znanja» računar sa svojim hardverskim i softverskim mogućnostima predstavlja jedno od najboljih, najbržih, najpreciznijih i najpouzdanijih sredstava dolaženja do informacije i njihove primene u radu. Savremeni računari pružaju mogućnost simultanog gledanja slike, slušanja govora i korišćenja multimedijskih izvora saznanja, što svakako, doprinosi bržem i potpunijem usvajanju gradiva, trajnijem pamćenju naučenog, efikasnijem korišćenju i kreativnijoj primeni dobijenih znanja. Svoj status i ulogu u obrazovanju računar je posebno učvrstio sa pojavom veštačke inteligencije (artifical intelligence) i ekspertnih sistema. Zbog toga postoji velika potreba da se učenici/studenti edukuju kako računar funkcioniše sa aspekta hardvera i softvera. Posebno je interesantno razmotriti kako hardverske komponente međusobno funkcionišu i kako primaju/šalju podatke od/ka spoljašnjim uređajima. Zato je jedan broj nastavnih predmeta iz računarske tehnike opredeljen ka obučavanju učenika/studenata o tome šta je u srži računarskog sklopa. Korišćenjem poznate mudrosti: „Jedna slika je vrednija od hiljadu reči“ predavači se trude da vizuelno dočaraju procese koji se dešavaju unutar računara, a pre svega rad CPU-a, memorija, magistrala i U/I jedinica. U tom cilju se i kreiraju simulatori računarske arhitekture koji, na slikovit način, vodeći obučavane korak po korak, dočaravaju složene postupke koji su karakteristični za funkcionisanje sistema.

U ovom radu prikazaće se simulatori računarske arhitekture koji na principu 16-bitnog procesora realizuju različite aritmetičke i logičke operacije, a takođe učitavaju/izvršavaju različite programe koji su smešteni u obliku niza komandi u unutrašnjoj (RAM) memoriji. Razvijeni simulatori omogućava da studenti kroz određeni broj vežbi spoznaju funkcionisanje unutrašnjosti računarskog sklopa na principu kretanja od prostijeg simulatora ka složenijem. Na taj način kroz logički niz elementarnih koraka

dolazi se do fundamentalnih znanja koja pokrivaju ovu, inače vrlo složenu oblast. Realizovani simulatori su definisani kao zasebne programske celine i to:

1. SIM 1 – koji omogućava osnovne aritmetičke operacije,
2. SIM 2 – omogućava izvršavanje aritmetičkih i logičkih operacija,
3. SIM 3 – objedinjuje aritmetiče i logičke operacije sa U/I uređajima,
4. SIM 4 – zaokružuje aritmetičke i logičke operacije sa korišćenjem memorije i
5. TFaCo simulator – koji objedinjuje potpunu arhitekturu mikroračunarskog sistema.

Druga glava ovog rada opisuje računarsku simulaciju uopšteno. Tu se razmatra postupak modeliranja, istorijski razvoj simulacije, potreba za računarskom simulacijom i slično. Takođe, tradicionalni i teoretski narativni pristup računarskoj simulaciji ovde je izložen.

U trećoj glavi objašnjen je edukacioni značaj simulatora, njegova primena u nastavnom radu kao i problemi koji se time iniciraju. Posebno su istaknuti simulatori u računarskoj tehnici, kao i kategorizacija simulatora.

Svetska iskustva u pogledu korišćenja simulatora i simulacioni softveri izloženi su u četvrtoj glavi. Posebno treba istaći simulatore tipa ECS, HASE, ESCAPE, DLXview i sl.

Peta glava daje osnovne postavke arhitekture računara, polazeći od podataka koji se u računaru obrađuju, preko njihove predstave, pa preko aritmetičkih i logičkih operacija, registara, mašinskih komandi, pristupu memoriji i slično. U ovoj glavi su prikazane i funkcije i arhitektura procesora koji se simulira – TFaCo-a.

Najvažniji deo rada opisan je u šestoj glavi gde su detaljno izloženi svi realizovani simulatori sa posebnim akcentom na TFaCo simulatoru. U njoj su detaljno objašnjeni arhitektura, osnovne postavke i softverska rešenja na bazi kojih je izvršena realizacija simulatora kroz niz koraka (od prostijih ka složenijim).

Rezultati dobijeni ovim radom, kao i mogućnosti daljeg usavršavanja (razvijanja dodatnih funkcionalnosti za simuliranje računarske arhitekture) dati su u završnoj - sedmoj glavi.

Prilog sadrži listinge procedura i fukcija za koje autor smatra da mogu biti od koristi čitaocima ovog rada.

2

2. RAČUNARSKA SIMULACIJA

Sa razvojem računara stvorila se i mogućnost da se različiti procesi iz realnog sveta repliciraju u virtuelnom svetu i da se modifikovanjem njihovih parametara dođe do novih ili potvrde već postojećih saznanja. U tom smislu razvila se i računarska simulacija koja pre svega podrazumeva korišćenje računara u cilju formiranja modela koji će oponašati realan sistem i nad kojim će se vršiti odgovarajuća analiza. Od posebnog je interesa i računarska simulacija samog računara, tj. delova računarskog sistema kojoj je u ovom radu i posvećeno centralno mesto.

Da bi uspešno mogli izvršiti simulaciju potrebno je najpre pristupiti procesu modeliranja. To je kreativan proces ljudskog uma kojim se na klasičan način formiraju funkcije (koje su od interesa) nekog realnog sistema. Na bazi modela realizuje se i simulacija koja se može posmatrati i kao relacija između modela i samog računara na kome se odvijaju odgovarajuća izračunavanja.

Sa aspekta teorije simulacije postoji mnogo pristupa i radova na tu temu. Najintersantniji su oni koji se zasnivaju na tradicionalnom i na narativnom pristupu o čemu će biti više reči u nastavku ovog rada.

2.1. MODELIRANJE I SIMULACIJA

U današnje vreme vrlo često se pristupa kreiranju modela za određeni realni sistem kao i simulaciji funkcija tog sistema. Postoji dosta razloga zbog kojih se pristupa ovim procesima i oni će biti izloženi u daljem delu ovog rada zajedno sa ostalim aspektima koji prate ove procese.

2.1.1. Modeliranje i modeli

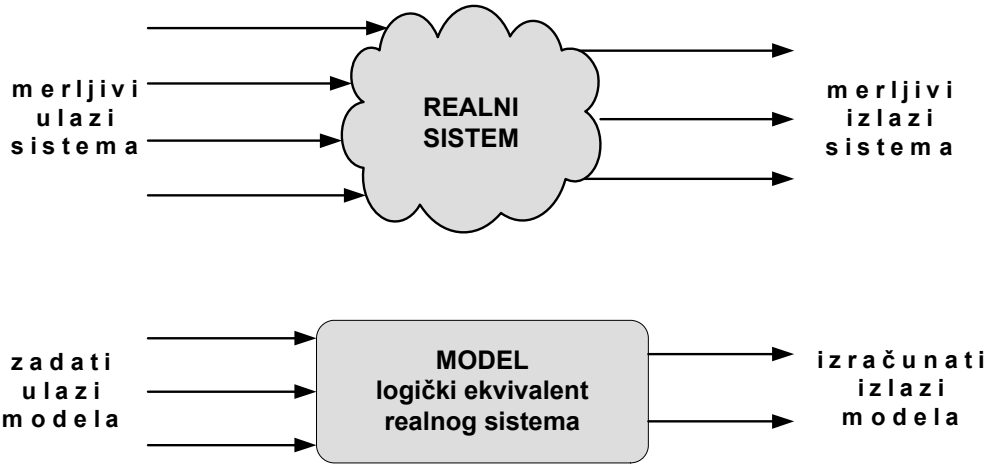
Modeliranje predstavlja jedan od osnovnih procesa ljudskoga uma. Ono je usko vezano za način ljudskog razmišljanja i rešavanja problema. Kao rezultat procesa koji nazivamo inteligentno ljudsko ponašanje, modeliranje predstavlja svakodnevnu aktivnost i veliki deo onoga što nas čini ljudskim (inteligentnim) bićima. Modeliranje izražava našu sposobnost da mislimo i zamišljamo, da koristimo simbole i jezike, komuniciramo, da vršimo generalizacije na osnovu iskustva, da se suočavamo sa neočekivanim. Ono nam omogućava da uočavamo obrasce, da procenjujemo i predviđamo, da upravljamo procesima i objektima, da izlažemo značenje i svrhu. Upravo zato, modeliranje se najčešće posmatra kao najznačajnije konceptualno sredstvo koje čoveku stoji na raspolaganju [17].

U praksi često nije tako. Posledice korišćenja neadekvatnog modeliranja mogu biti katastrofalne za procese koji se modeliraju. U osnovi svake računarske simulacije je proces modeliranja. Zadatak modeliranja je da se dođe do određenog saznanja o nekom realnom sistemu korišćenjem nečega (modela) a da to bude isplativo. U toku postupka modeliranja nekog sistema treba od velikog broja elemenata i karakteristike tog sistema izabrati samo one koje su od značaja za istraživanje sistema. Kao rezultat modeliranja nastaje **model** [15]. Model predstavlja uprošćenu sliku realnog sistema. Cilj modela je da uobliči na vidljiv način ono što je suštinsko za razumevanje nekog aspekta njegove strukture, a ne da precizno reprodukuje stvarnost u svoj njenoj složenosti.

Zbog toga što se zadržavaju samo one karakteristike originala koje su bitne za svrhu njegovog izučavanja, može se reći da su modeli uvek apstrakcije realnog sistema. Nivo apstrakcije u procesu modeliranja utiče na ispravnost modela, odnosno na uspešnost predstavljanja realnog sistema modelom. Zbog pojednostavljenog pogleda na realni sistem problem ispravnosti modela javlja se u svakom procesu modeliranja. Modeli koji daju iste izlazne vrednosti kao i realni sistemi suviše su složeni i preskupi i neadekvatni su za eksperimentisanje. Ako je pak, model suviše pojednostavljen, onda on ne odslikava na pravi način posmatrani sistem, pa rezultati koji se dobijaju primenom takvih modela mogu

da budu neadekvatni i pogrešni. Zato treba izabrati model koji će što vernije da odslikava posmatrani sistem, ali i da njegova složenost i cena ne budu ograničavajući faktori.

Na slici 2.1 [15] prikazan je odnos realnog sistema i jednog njegovog modela. Može se izvesti zaključak da prikazani model nije savršen u smislu kako je to prethodno objašnjeno.

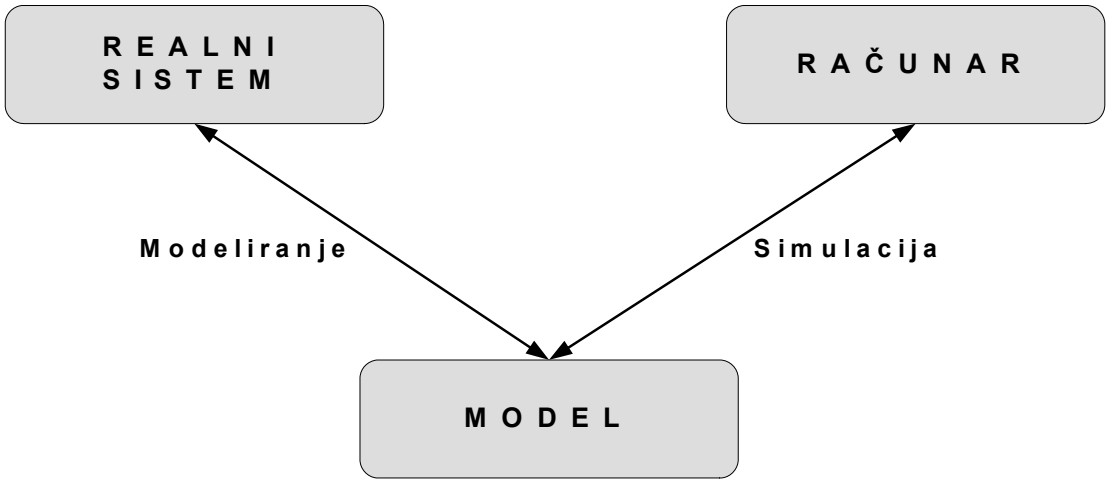


Slika 2.1 Odnos realnog sistema i njegovog modela

Ако se za dati ograničeni skup opisnih promenljivih jednog modela, njegovo ponašanje može odrediti na praktičan način: analitički, numerički ili putem eksperimenta, tada se može reći da je model koristan. Tada se za izvesne, uglavnom slučajne ulaze, posmatraju odgovarajući izlazi. Takav proces naziva se **simulacija** [15].

Reč simulacija u svakodnevnoj upotrebi može da označi veći broj različitih aktivnosti. Ako se proces izgradnje apstraktnih modela za neke sisteme realnog sveta i obavljanje eksperimenata nad njima odvijaju na računaru, tada se govori o računarskom modeliranju i simulaciji. Danas je modeliranje nezamislivo bez računara. Računar se u modeliranju koristi za dve svrhe: za razvoj modela i za izvođenje proračuna na osnovu stvorenog modela.

Modeliranje i simulacija predstavljaju složenu aktivnost koja uključuje tri elementa: **realni sistem**, **model** i **računar**. Na slici 2.2 [15] dat je uprošćen prikaz ove aktivnosti.



Slika 2.2 Relacije modeliranja i simulacije

Realni sistem je izvor podataka za specifikaciju model, tj. izvor podataka o ponašanju. Pojavljivanje tih podataka se može izarziti funkcijom **X(t)**, gde je **X** bilo koja promenljiva koja interesuje istraživača, a **t** je vreme mereno u odgovarajućim jedinicama.

Model je apstraktni prikaz sistema i daje njegovu strukturu, njegove komponente i njihovo uzajamno delovanje. Njegovi objekti se opisuju atributima ili promenljivim. Kako se za simulaciju najčešće koristi računar, to se pod modelom može podrazumevati skup instrukcija (program) koji služi da se generiše ponašanje simuliranog sistema. Ponašanje modela ne mora da bude u potpunosti jednako ponašanju simuliranog sistema, već samo u onom domenu koji je od interesa.

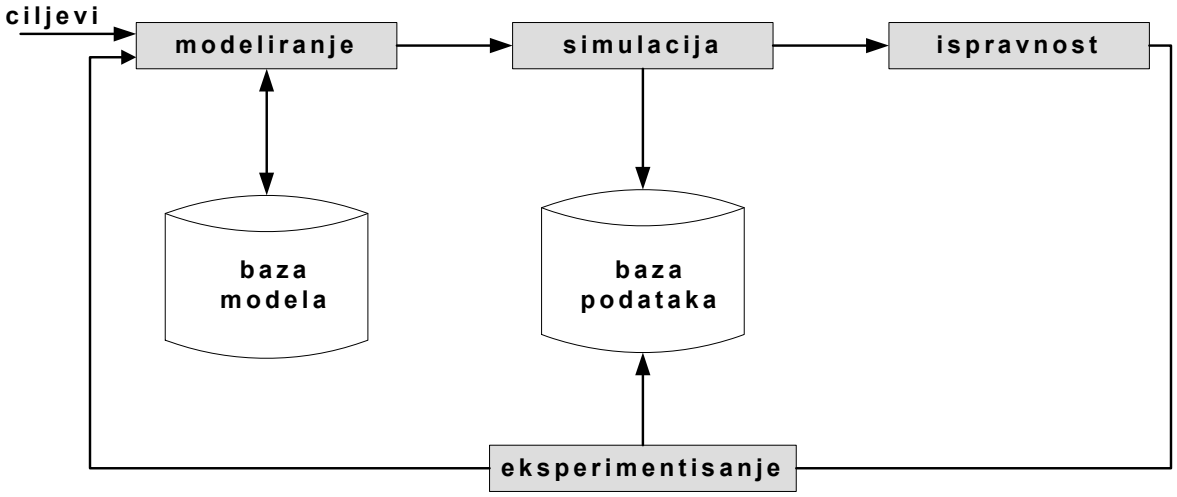
Računar je uređaj koji je sposoban da izvrši instrukcije modela, koje na bazi ulaznih podataka generišu razvoj modela u vremenu. Uz pomoć različitih metoda i programskih alata, računari pružaju pogodno okruženje za stvaranje složenih modela i efikasan rad nad njima.

Posebnu pažnju treba usmeriti na otkrivanje i definisanje **relacije modeliranje** i **relacije simulacija**, koje postoje između ova tri modela. Modeliranje je proces kojim se uspostavlja veza između realnog sistema i modela, dok je simulacija proces koji uspostavlja relaciju između modela i računara.

Relacija modeliranja odnosi se proces utvrđivanja stepena slaganja podataka o realnom sistemu sa podacima modela, odnosno ispravnost modela. Na osnovu ispravnosti modela donose se odluke o upotrebljivosti rezultata simulacije, izmeni modela, izmeni podataka (ulaznih promenljivih, parametara), daljem nastavku simulacije, ponavljanju simulacije, itd.

Relacija simulacije odnosi se na proveru da li simulacioni program verno prenosi model na računar kao i na tačnost kojom računar izvršava instrukcije modela. Da bi mogli da se uporede stvarni podaci sa podacima koje generiše simulator, mora se izvršiti **verifikacija**, tj. mora da se utvrdi tačnost, odnosno korektnost simulatora.

Slika 2.3 [15] prikazuje aktivnosti procesa modeliranja i simulacije sa bazom modela kao centralnim objektom. Na osnovu ciljeva koji se generišu van granica sistema upravlja se procesom **modeliranja**. Uloga **baza modela** i **baza podataka** je da čuvaju i organizuju prikupljene podatke o realnom sistemu. Svaki novi cilj inicira aktivnost znanje iz baze. Faze **simulacije** (eksperimentisanje sa modelom) i **ispravnosti** slede fazu izgradnje modela.



Slika 2.3 Proces modeliranja i simulacije

Nakon faze **ispravnosti** ponavlja se postupak **eksperimentisanja** nad realnim sistemom. Mogu se javiti zahtevi dodatne modifikacije ili reinicijalizacije prvobitnog modela. U tom procesu, kao rezultat nedostatka podataka u bazi znanja mogu se formulisati novi ili izmeniti postojeći **ciljevi**. Na kraju, kao rezultat javlja se jedan ili više modela koji vode ka ispunjenju eksternih ciljeva (ukoliko proces ne "upadne" u petlju iz koje ne može da izađe). Kreirane modele koristi donosilac odluke. Pored toga, oni se mogu memorisati u bazi modela i koristiti u nekoj narednoj fazi aktivnosti.

U osnovi računarske simulacije je model sistema. Sistemi mogu biti [15]:

- proizvodni pogon sa mašinama, ljudima, transportnim uređajima, pokretnim trakama i skladišnim prostorom,
- računarska mreža sa serverima, klijentima, diskovima, printerima, mrežnim karakteristikama i operatorima,
- preduzeće za osiguranje gde stiže obimna dokumentacija koja se sortira, pregleda, dopunjava, kopira i šalje itd.

2.1.2. Karakteristike simulacionog modeliranja

Od onog trenutka kada je čovekov um počeo da shvata svu složenost pojava i stvari koje su ga u prirodi okruživale, on je nesvesno vršio fizičko modeliranje.

Naučni smisao modeliranje i simulacija dobijaju tek pojavom prvih digitalnih računara. Računar se tom prilikom koristi kao pomoćno sredstvo kako u fazi izgradnje modela tako i u samom simulacionom eksperimentu. U tabeli 2.1 [15] dat je prikaz istorijskog pregleda razvoja simulacije.

Tabela 2.1 Istorijski pregled razvoja simulacije

1600.	Fizičko modeliranje
1940.	Pojava elektronskih računara
1955.	Simulacija u avio – industriji
1960.	Simulacija proizvodnih procesa
1970.	Simulacija velikih sistema uključujući ekonomske, društvene i ekološke
1975.	Sistemski pristup u simulaciji
1980.	Simulacija diskretnih stohastičkih sistema i viši nivo učešća u sistemima za podršku odlučivanju
1990.	Integracija računarske simulacije, veštačke inteligencije, računarskih mreža i multimedijalnih tehnologija

Reč simulacija, u sadašnjem značenju, potiče s krajem četrdesetih godina prošlog veka, kada je Džon fon Nojman (John von Neumman) rešavajući određene probleme veće složenosti ustanovio da se rezultati ne mogu dobiti analitičkim putem. Kako je izvođenje neposrednih eksperimenata bilo veoma skupo, on je za rešavanje tih problema koristio Monte Karlo tehniku.

Cilj simulacije je da ce prouči ponašanje sistema koji ce simulira, ali i da se ustanovi kako bi se taj isti sistem ponašao kada bi na njega delovao neki drugi skup ulaznih veličina i parametara.

Simulacioni modeli prikupljaju podatke o promenama stanja sistema i izlaza, fokusirajući se na ponašanje individualnih komponenti sistema, dok analitički modeli sveukupno ponašanje sistema tretiraju direktno. Međutim, samo mali broj kompleksnih realnih sistema može se adekvatno opisati preko analitičkih jednačina.

Rešenje simulacionog modeliranja ne može da se dobije u analitičkom obliku, u kojem su zavisne promenljive funkcije nezavisnih promenljivih, već se rešenje problema dobija eksperimentisanjem nad modelom.

Simulacioni modeli najčešće su modeli sistema koji se menjaju u vremenu, tj. dinamičkih sistema. Može se reći da su simulacioni modeli modeli sistema koji se ne mogu opisati niti rešavati matematičkim sredstvima. Oblasti u kojima se najviše koriste su: inženjerstvo, menadžment i ekonomija, medicina, biologija i druge prirodne nauke.

Simulacija je proces koji zahteva multidisciplinarni pristup. Po svojoj prirodi to je proces koji je blisko vezan za alate i tehnike računarskih nauka i sistemske analize. Simulacija je danas zasebna naučna disciplina i predstavlja koherentnu i dobro razvijenu oblast.

2.1.3. Potreba za simulacijom

Koji su **razlozi** zbog čega se jedan sistem zamenjuje modelom, a zatim vrši simulacija? Najvažniji su [15]:

- *Visoka cena eksperimenata nad realnim sistemom*. Eksperimentisanje sa realnim sistemom, uglavnom je neisplativo ili suviše složeno. Modeliranje može da ukaže na to da li je dalje ulaganje u eksperiment ekonomski opravdano ili ne.
- *Simulacione metode lakše je primeniti nego analitičke metode*. Stoga je krug potencijalnih korisnika simulacionih metoda znatno širi.
- *Nemogućnost eksperimenata u nekim sferama*. Npr., ponekad treba simulirati uslove pod kojima nastupa razaranje sistema. Naravno, razaranje realnog sistema najčešće nije dopustivo. U takvim prilikama, simulacija predstavlja pravo rešenje.
- *Kada se vrši realni eksperiment, uvek postoji izvesna greška pri merenju usled nesavršenosti mernih uređaja*. Pri simulaciji ove greške nema. Postoji samo greška "zaokruživanja" usled konačne dužine reči u računaru, ali se ona sa malo truda može učiniti zanemarljivom.
- U realnom sistemu je teško moguće zaustaviti dalje odvijanje eksperimenta, kako bi se ispitale vrednosti svih promenljivih u tom trenutku. U toku izvršavanja simulacije je moguće.
- Sa realnim sistemom teško je izvodljivo iznalaženje optimalnog funkcionisanja nekog sistema, menjanjem raznih parametri. Razlog je što bi takav eksperiment bio preskup. Tada su gradnja modela i njegova simulacija moguće rešenje.
- *Pri simulaciji, vreme se može sažeti*. To je značajno kod simulacije dugotrajnih procesa. U drugim slučajevima, vreme se može znatno produžiti. Na primer, sa ciljem da se prati postepeno odvijanje vrlo brzih procesa, što je u realnom sitemu nemoguće.

Osnovni **nedostaci** korišćenja simulacije su sledeći [15]:

- Simulacioni modeli za digitalne računare mogu biti skupi i mogu zahtevati značajno vreme za izgradnju i validaciju.
- Zbog statističkog karaktera simulacije potrebno je izvođenje većeg broja simulacionih eksperimenata kako bi se dobio odgovarajući uzorak rezultata simulacije, a već i pojedinačno izvođenje eksperimenta može zahtevati dosta vremena i memorije računara.
- Nekada se ne dobijaju zavisnosti izlaznih od ulaznih promenljivih modela niti optimalna rešenja.
- Za ispravno korišćenje simulacionog modeliranja potrebno je poznavanje više različitih metoda i alata.
- Vrednovanje modela je dosta složeno i zahteva dodatne eksperimente.

2.2. SIMULACIJA KAO METOD I NARACIJA

Kada se radi o razmatranju računarske simulacije onda se obično koriste dva pristupa: tradicionalni i teoretski iskazan narativno.

- **Tradicionalan pristup** podrazumeva radove o računarskoj simulaciji kao istraživačkom metodu,
- **Teoretski iskazan narativno** (naracija – pričanje, prepričavanje) pomaže nam da upoznamo i vidimo strukturu i proces računarske simulacije različito zavisno od auditorijuma, kao i sa različitih gledišta.

Računarska simulacija može biti prikazana na poseban način tako da postane jasnija povezanost računarske simulacije i teorije organizacije.

2.2.1. Simualcija kao istraživački metod

O računarskoj simulaciji se piše već više od pola veka, a cilj je da se razumeju postojeće vrste računarske simulacije, da se prikažu prednosti i mane računarske simulacije u odnosu na druge metode i tehnike koje istraživači mogu da koriste da bi usavršili istraživanje simulacija.

Akcenat će biti stavljen isključivo na adaptivne i dinamičke računarske modele (modeli koji opisuju promenu u vremenu kao što su agent-bazirani modeli i modeli dinamičkih sistema), a radi postizanja veće jasnoće i pravljenja odgovarajućih simulatora koji mogu da se prilagode većini statičkih simulatora.

Komponente i tipovi računarskog modela

Istraživači simulacija su se vremenom složili sa osnovnom tezom Cohena i Cyerta [2] koji su 1965. godine dali definiciju računarske simulacije: "*Računarska simulacija uključuje kreiranje pretpostavki o relacijama između koncepata i implikacija koje proističu iz tih pretpostavki".*

Međutim, istraživači simulacija nisu saglasni po pitanju komponenata računarske simulacije, zbog različitog pristupa istraživanju simulacija. Osnova ovih razlika može biti nađena u razlikama između agent-baziranog modela i modela dinamičkih sistema.

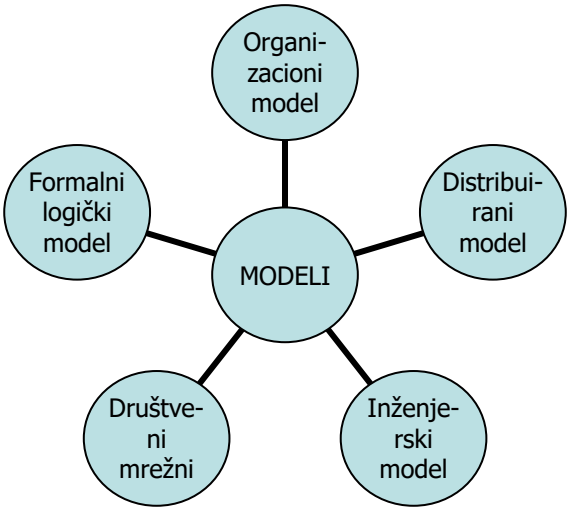
Naučnici koji koriste **agent-bazirane modele** da bi došli do implikacija o njihovim pretpostavkama kreiraju modele sa spojenim konceptima (npr. zakup, naklonost učenju, potrebna energija, zahtevana energija, verovanja) u apstraktnim agentima (npr. ljudi, organizacije, problemi, izbori, verovanja) koji imaju uticaj tokom vremena saglasno sa skupom unutrašnjih jednačina i logičkih (npr. ako…onda sledi…) pravila.

Naučnici koji koriste **model dinamičkog sistema** pokušavaju da model relacija između koncepata prikažu kao različite jednačine koje opisuju kako promene jedne promenljive utiču na promenu druge promenljive u vremenu. Ovi pristupi se razlikuju od agent-baziranih modela koji tretiraju učesnike kao izdvojene „agente“ (celine) koje deluju u diskretnim vremenskim intervalima između više stepeni slobode, dok dinamički sistemi, generalno ne razmatraju uticaj individualnih faktora, tretiraju vreme kao kontinualnu veličinu, i imaju manji broj stepeni slobode za opis sistema.

Razlike između agent-baziranih modela i dinamičkih sistema su u načinu na koji istraživači razdvajaju tipove simulacija i identifikuju njihove komponente. Tipovi simulacije se razlikuju i na osnovu metodologije i na osnovu svrhe, tj. namene simulacije.

Postoje pet metodoloških pristupa za matematičko modeliranje (koji se mogu primeniti, ali nisu ograničeni samo za računarske simulacije) [29]:

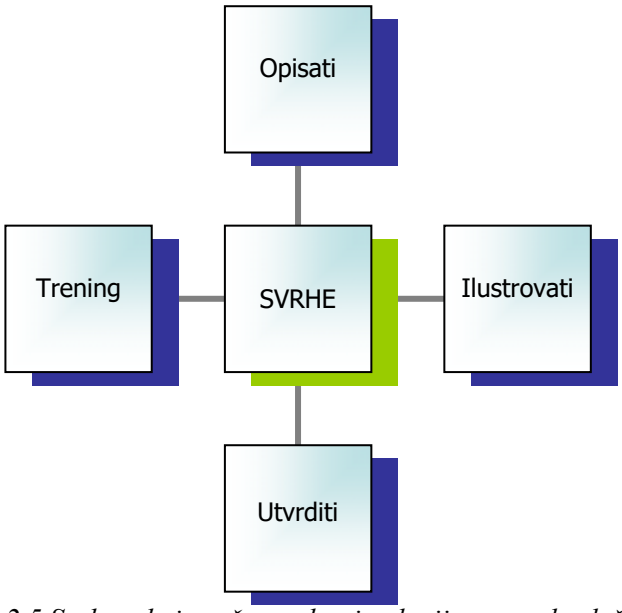
- organizacioni model koji opisuje „idealni tip“ organizacionog ponašanja,
- distribuirani model veštačke inteligencije koji omogućava dovoljnu jasnoću ili testiranje konceptualne adekvatnosti,
- organizacioni inženjerski model koji simulira specifične sisteme za vođenje intervencija,
- društveni mrežni model i
- formalni logički model.



Slika 2.4 Metodološki pristupi (modeli) za matematičko modeliranje računarskih simulacija

Cohen i Cyert [2] razvrstavaju računarske simulacije na osnovu namene tj. svrhe. Simulacije mogu da služe za sledeće četiri namene:

- da opišu zašto se pojedinačne organizacije ponašaju na specifičan način,
- da ilustruju posledice odgovarajućih logičkih organizacionih pretpostavki,
- da utvrde koji metodi ili dizajni se najbolje uklapaju za postizanje odgovarajućih ciljeva (normativa) ili
- za trening ljudi koji rade u organizacijama (npr. čovek-računar)



Slika 2.5 Svrhe u koje računarske simulacije mogu da služe

Da bi se postigli neki od ovih ciljeva, moguće je simulirati sa ili bez različitih organizacionih agenata, sa diskretnim ili kontinualnim modelima vremena, sa mnogo ili nekoliko stepeni slobode – zavisno od cilja koji se želi postići.

Metodološke implikacije

Radi poređenja računarskih simulacija sa drugim istraživačkim metodama i radi formulisanja preporuka za poboljšanje kvaliteta simulacije, koriste se posebni opisi računarskih simulacija. Računarske simulacije porede se sa širokom klasom istraživačkih metoda, kao što su laboratorijska istraživanja, pregledi i studijske oblasti. Računarska simulacija je nenametljiv način za ispitivanje ponašanja pojedinačnih sistema tako da radi dosta dobro na postizanju opštih osobina i realnog sadržaja, ali nije dovoljno dobra za merenje preciznog ponašanja. Postoji i suprotni koncept po kome prednosti i mane računarske simulacije u odnosu na druge metode najmanje zavise od tipa simulacije koja je primenjena.

Naučnici su takođe svesni potencijalnih slabosti ovih istraživanja i preporučuju metode za kompenzovanje tih slabosti. Npr. da bi obezbedili da zaključci izvedeni iz računarskih simulacija budu prihvaćeni kao rezultati određenog modela, a ne samo kao proizvod jednog programskog jezika, naučnici preporučuju „prebacivanje“ modela u drugi programski jezik ili sistem modeliranja. A da bi uzeli u obzir nedostatak spoljne provere koja često prati računarske simulacije, naučnici su razvili kvalitativni metod koji uzima podatke za kreiranje modela i kvantitativne metode za poređenje ponašanja modela u odnosu na realnost. Da bi se ovo kompenzovalo, preporučuje se povećanje aktivnosit u realnom vremenu, skupljanje zabeleški o snimljenim odlukama modela, i programiranje modela na taj način da su interno ispravni, upotrebljivi i proširivi.

2.2.2. Teoretski pristup simulaciji

Ideja da istraživači simulacija treba da pažljivo objašnjavaju svoja istraživanja verbalno dovodi do razmišljanja kako teoretski izneti pristup računarskoj simulaciji.

Ne postoje eksplicitne norme, standardi ili pravila za održavanje društvenih, naučnih istraživanja koja kažu da naučnik ne treba da testira hipoteze simulacionog istraživanja. Međutim, testiranje hipoteza i u tu svrhu korišćenja matematičkih modela kao što su računarska simulacija bez naglašavanja da rezultati nisu empirijski – je teorija koje može da zbuni mnogo čitalaca, zato što matematika, formalna logika i formalne hipoteze imaju tendenciju da nose relativno nisku objektivnost [29].

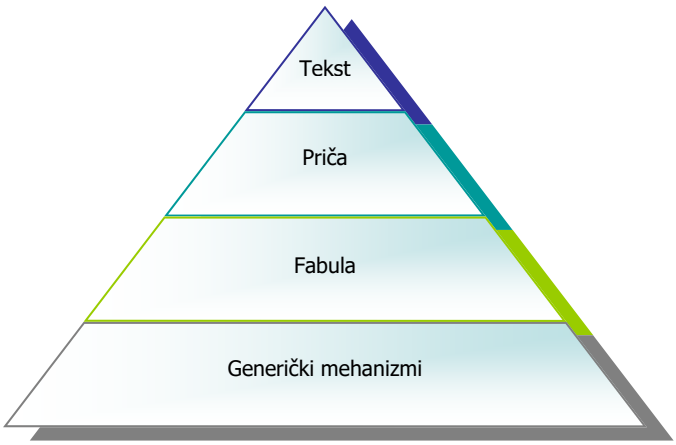
Računarska simulacija ima takođe i druga teoretska svojstva. Da bi se prikazalo kako je računarska simulacija proces izlaganja teorija jezikom koji se naziva matematički – opisan je Penltandov model [12]. Primenjujući ovaj model na računarske simulacije mogu

se uočiti unutrašnje relacije između računarske simulacije i teorije organizacije i njihova transparentnost.

Narativnost u računarskoj simulaciji

Naracija prema Pentlandu, ima strukturne osobine koje nam omogućuju da koristimo narativne podatke (elemente priče) ili tekstove, da bi otkrili kako organizacija funkcioniše ili kako se sistem menja iz jednog vremenskog intervala u drugi. Drugačije rečeno, po Pentlandu [12], naracija ne uključuju samo tekstove koje opisuju uzorak događaja koji nastaje kao posledica učesnika ili primene sistema na poseban način, već uključuje takođe implicitno ili eksplicitno objašnjenja o tome zašto se promene događaju u pojedinačnim slučajevima. Ovi opisni uzorci i objašnjenja čine osnovnu strukturu naracije. Zato što osnovna struktura pojedinačnog teksta nije obično direktno uočljiva, Pentland preporučuje okvir koji naučnici treba da koriste za analizu tekstova da bi otkrili narativnu strukturu.

Pentland predlaže da naracija treba da imaju 4 nivoa strukture.



Slika 2.6 Nivoi naracije po Pentlandu

Najdublji nivo naracije su **generički mehanizmi**. Generički mehanizmi su apstraktni procesi koji omogućavaju i ograničavaju pojavu pojedinačnih događaja. Generički mehanizmi uključuju evolucione procese (manifestovane kao varijacija-selekcija-čuvanje-varijacija), procese životnog ciklusa (manifestovane kao startovanje-rast-rod-završetak-startovanje) ili dijalektičke procese (manifestovane kao teze-antiteze-sinteze-teze), gde naučnici koriste različite uzročne mehanizme za objašnjenje zašto se svaki od generičkih koraka u procesu pojavljuje. Ovi procesi mogu, naravno, delovati nezavisno ili u zajedništvu sa drugim procesima, mogu uzeti oblik radnih rutina, koordinacije prometa, ili drugih praktičnih pojedinačnih poslova. Pentland tretira generičke mehanizme kao neprimetne, ali ipak kao objektivnu realnost. Drugi analitičari naracije,

međutim više vole da tretiraju ove mehanizme kao tekstove koji „objektivno“ postoje samo zato da naglase da učesnici deluju na ovaj način jedino ako su mehanizmi realni, ali i pored toga oni nisu društveno konstruisani, kontinualno menjani i samo su primenljivi lokalno za vreme njihovog postojanja (egzistencije).

Ostala 3 nivoa narativne strukture su progresivno uočljiva tokom naracije.

Sledeći nivo iznad generičkih mehanizama je **fabula**, koja predstavlja generički opis redosleda događaja koje generički mehanizmi omogućavaju ili ograničavaju (npr. varijacija-selekcija-čuvanje-varijacija). Iako fabula obično nije eksplicitno identifikovana tokom izlaganja priče, ljudi mogu često reći da različite priče imaju istu fabulu zato što su zaključci priče uglavnom isti bez obzira ko priča priču, koji pogled na priču je uzet, koji jezik ili medijum je korišćen. Prema tome, kada je fabula pričana sa određenog stanovišta (npr. stanovište nekog zaposlenog, ili nekog menadžera) ta verzija fabule se naziva **priča** (ona je nivo viši od fabule), a pojedinačno pričanje ove priče naziva se **tekst** (koji je jedan nivo više od priče). Prema tome priče zavise od stanovišta koje ima osoba koja priča („izbacuje“ fabulu), a tekstovi variraju u zavisnosti od toga koja ličnost priča priču (npr. u zavisnosti kom slušaocu osoba priča), iako stanovište ne mora da bude uopšte promenjeno.

Tekstovi su podaci sa kojima teoretičari naracije počinju kada pokušavaju da razumeju razloge zašto nastaju promene na određeni način. Tako, ako je istraživački cilj kao što Pentland sugeriše [12], da se objasni zašto nastaju promene na određen način, onda naučnik mora analizirati tekstove na takav način da otkrije stanovišta od kojih verzija je priča sastavljena, generičku sekvencu (redosled) događaja koja je osnova različitih priča, i relacije ili mehanizme koji povezuju događaje u pojedinačnu sekvencu. Istraživači mogu raditi ovo po Pentlandu, prateći kako tekstovi markiraju (poklapaju) sekvence događaja u vremenu, identifikujući ko je glavni akter ili akteri u tekstu, identifikujući glas kojim je pričan tekst, identifikujući evaluacione okvire i reference koje su govornici ili pisci teksta koristili, i prateći druge pomoćne identifikatore sadržaja i dodataka.

„Pričanje“ simulacije

Istraživači naracije često koriste različite slušaoce, različita stanovišta, različite sekvence događaja itd. da bi razumeli neku naraciju (priču). Istraživači simulacija sa druge strane, prave pretpostavke (usvajaju) o tome šta su generički mehanizmi (zato što su generički mehanizmi relacije između koncepata), i oni razmišljaju o tome koje vrste tekstova (izlaza) ovi generički mehanizmi će kreirati kada simulacija bude realizovana na

digitalnom računaru. Zaista, računarska simulacija ima istu strukturu koju je Pentland opisao za govornu naraciju.

Generički mehanizmi računarske simulacije su skup relacija između koncepata u strukturi modela kao što su jednačine i pravila kreiranja.

Fabula računarske simulacije je zaključna sekvenca događaja koju model kreira tokom izražavanja jednog skupa unapred definisanih relacija za taj model.

Priče računarske simulacije su uzroci događaja koji se pojavljuju kada su parametri modela isti ali se ulazne promenljive menjaju (npr. mi možemo zamisliti da su parametri ono što se definiše kao „stanovište“ modela koji se uzima prilikom izlaganja priče).

Tekstovi računarske simulacije su izlazne promenljive generisane za svaki skup različitih ulaznih promenljivih bez obzira koji su parametri modela (npr. može se misliti o ovome kao simulaciji o pričanju različitih verzija jedne priče u zavisnosti od ulaznih promenljivih koje su date za tu priču).

Stoga, svaki put kada naučnik izvršava računarsku simulaciju tada koristi skup generičkih mehanizama (pretpostavki) da bi generisao tekst (izlaz) koji je verzija priče (npr. koristeći pojedinačne ulazne promenljive) pričana sa pojedinačnog stanovišta (npr. koristeći specifične parametre).

Prvu stvar koju treba da primetimo, kada koristimo računarsku simulaciju koja ima strukturu naracije, je da računarska simulacija omogući i ograniči organizacione fenomene koje vidimo, i kako ih vidimo. Drugim rečima, generički mehanizmi računarske simulacije omogućavaju nam da vidimo posledice pretpostavki da su organizacioni koncepti u međusobnim relacijama na određen način, ali nas ograničava da vidimo mogućnosti koje postoje ako su drugi koncepti ili relacije uključeni; parametri nam omogućuju da vidimo efekte ispitujući pojedinačna stanovišta kada istražujemo posledice pojedinačnih generičkih mehanizama; i ulazne promenljive nam omogućuju da vidimo različite verzije priča koje računarska simulacija može da „priča“.

Modeli računarske simulacije obezbeđuju da izlazni tekstovi sugerišu nove uvide u svaku individualnu računarsku simulaciju primenjujući strategije narativne analize. Npr., jedna strategija koja je odmah očigledna je primena Pentlandovih saveta za ispitivanje frekvencije događaja, glavnih aktera, narativnog glasa, evolucionog okvira za reference i dodatnih indikatora za sadržaj i dodatke radi otkrivanja korišćenih slušalaca sa stanovišta i odnosa snaga i za model i za izlaz. Druge tehnike narativne analize takođe postoje kao što

je analiza transfera diskusionih objekata ili analiza retoričkih uređaja kao što su kredibilitet i defamiliriacije ili stilski izbori kao što su forma, neodređenost, dijalektična rekonstrukcija i prisutnost. Ove analize pomažu da se vidi šta omogućuje istraživaču da napravi teoretski doprinos, a takođe šta istraživač može propustiti kao rezultat svog gledišta relacija i opredeljenih slušalaca.

2.2.3. Analize „računarskih naracija“

Prikupljeni istraživački članci o računarskim simulacijama su objavljeni u toku 40-tak godina (1970.-2008.) u 4 ravnopravna časopisa: *Academy of Management Journal (AMJ), Administrative Science Quarterly (ASQ)*, *Strategic Management Journal (SMJ)*, *Organizational Behavior and Human Decision Processes (OBHDP)*. Ovi časopisi su izabrani zbog procene da su objavljeni članci doprinos teoriji, da postoji njihov bar minimalni naučni doprinos, da su fokusirani su na organizacione tokove i orijentisani ka širokom krugu čitalaca (nasuprot, npr. uskom fokusu koji ima *System Dynamics* or *Computational and Mathematical Organization Theory)* [29].

Razmatrani radovi o simulaciji su:

- Model o organizovanoj anarhiji nazvan „prazna konzerva“,
- Model o preciznoj ravnoteži i
- Model o istraživanju i iskorišćenju.

Cilj pregleda ovih simulacija je bio da se daju čvrsti primeri o svakom tipu naracija koje istraživači računarskih simulacija mogu reći. Pokušano je da se prikaže da naučnici mogu koristiti dekonstruktivne simulacije da bi pronašli i ispravili logičke tokove, a takođe da istraživači mogu koristiti ilustrativne simulacije da daju dovoljno objašnjenja za organizacione fenomene. Uz to istraživači mogu koristiti normativne simulacije radi poređenja performansi organizacija ili organizacionih učesnika, a sve to bazirano na promenama parametara od interesa ili promenama modela. Data je pretpostavka da i model i njegov izlaz su narativni tekstovi, da oboje ograničavaju i omogućavaju ono što su u stanju da vide istraživači, i da izbori o tome kako opisati modele verbalnim jezikom treba da se poklapaju sa vrstom doprinosa koje modeli daju, razmišljajući pri tome o mogućnostima da se svet vidi i na drugi način i koristeći autorski stil koji priznaje (prihvata) sadržajnu prirodu posledica istraživanja simulacija.

Istraživači računarskih simulacija koriste i objašnjavaju tri tipa naracija:

- **Dekonstruktivne simulacije** koriste se za utvrđivanje i korigovanje logičkih tokova,

- **Ilustrativne simulacije** koriste se da objasne organizacione fenomene i
- **Normativne simulacije** koriste se radi poređenja performansi organizacija ili učesnika.

2.2.4. Računarska simulacija i teorija organizacije

Centralna pretpostavka je da je računarska simulacija deo teorije organizacije (pre nego jednostavan istraživački metod primenjen na teoriju) zato što i model i rezultat računarske simulacije – slično verbalnoj teoriji – su oboje narativni tekstovi koje naučnici koriste da opišu i objasne organizacione fenomene. Kada razmišljamo o računarskim modelima kao narativnim tekstovima mi zaključujemo da ovi tekstovi i omogućavaju i ograničavaju ono što vidimo i razumemo i kod organizacionih fenomena: tako se mogu otkriti podrazumevana publika, pogledi, privremene strukture, posledične veze u modelima i njihovim izlazima (rezultatima) primenjujući tehnike narativne analize na računarske simulacije. Takođe, preporuke koje su dali naučnici za poboljšanje istraživanja simulacije mogu se prihvatiti kao saveti za poboljšanje retorike matematičke relacije. Identifikovana su tri tipa „priča" koje istraživači računarskih simulacija koriste: **ilustrativne sa dovoljno objašnjenja, normativne testove osetljivosti i dekonstruktivne testove logičke koherentnosti** [29].

Naučnici mogu koristiti ova tri tipa teoretskih doprinosa da pojasne jezik njihovih verbalnih teorija i da nađu komplementarnost sa drugim tipovima istraživanja (posebno kvalitativno istraživanje koje se takođe fokusira na procese koji su promenljivi u vremenu).

Kvalitativni podaci

Trenutno, najčešće korišćenje kvalitativnih podataka u istraživanju simulacije je u formi pričica (anegdota) koje se koriste da opravdaju relacije u modelu ili da ilustruju „validnost" rezultata modela. Međutim, kvalitativno istraživanje može generisati teoriju koja ima osobine da može dopuniti dobro simulaciono-baziranu teoriju. Dekonstruktivni i normativni primeri nude slična poređenja.

Jedan od razloga što su možda ovakve studije retke je zato što se kvalitativno istraživanje i računarska simulacija shvataju kao da su različite po prirodi, i istraživači su pisali vrlo malo kako da se pristupi takvom istraživanju.

Statistički podaci

Istraživači simulacija su uradili mnogo više na integrisanom statističkom istraživanju računarskih simulacija nego na integrisanom kvalitativnom istraživanju. Npr., naučnici savetuju, sa određenom verovatnoćom, uključivanje relacija u računarski model koji ima empirijsku potvrdu. Takođe, istraživači su razvili statističke testove za poređenje vremenskih nizova generisanih od strane modela sa aktuelnim vremenskim nizovima podataka realnog sistema koji se modelira. Drugi simulatori razvijaju testove koji su dizajnirani za posebne namene i to za poređenje rezultata računarskog modela sa jedinstvenim podacima koji se generišu i koji razmatraju neki fenomen od interesa. Prednosti korišćenja ova dva metoda su jasni: računarska simulacija omogućava naučnicima da uključe povratnu spregu i nelinearnost u modele koji razmatraju organizacione fenomene, dok statistički testovi povećavaju kredibilitet – verovatnoću računarskog modela. Dodatni metodi za integraciju statističkih testova u računarskoj simulaciji bi sigurno bili od koristi – posebno za različite forme agent-baziranih modela. Istraživači dinamičkih sistema su napredovali najviše u ovoj oblasti, efikasno mešajući i kvalitativno i statističko istraživanje sa računarskim simulacijama.

Poređenje i odmeravanje metoda

Kombinovanje kvalitativnog i/ili statističkog istraživanja sa računarskom simulacijom može biti moćan koncept za razvijanje prihvatjive i razumljive teorije. Trenutno najveći broj istraživača simulacija koji koriste više od jednog metoda za razvijanje teorije rade to u odvojenim studijama– što je dobar način da se vrši istraživanje. Ideja o kombinovanju metoda u istoj studiji može biti moćna, ne samo zbog uobičajene koristi od triangulacije, već takođe i od preporuka (verbalnih saveta) koje iz toga proističu. Već su pomenute neke od ovih dobiti: kao što je dodatna kredibilnost postignuta kroz višestruke metode, mogućnost balansiranja vrednosti kao što su defamiliarizacija i jasnoća, fokus i multidimenzionalnost i shvatljivost i pamtljivost. Druga prednost međutim, je da istraživači „vagaju“ između metoda u istoj studiji, oni gledaju iste fenomene sa različitih tačaka gledišta, razmišljaju o perspektivama različitog auditorijuma, identifikuju više detalja u sekvencama događaja i razmatraju uključivanje ili isključivanje različitih koncepata i relacija u objašnjenjima. Ovo su implicitne tehnike narativnih analiza, koje sve pomažu da se identifikuju različite priče koje mogu biti pričane ili da se ocrtaju osnovni mehanizmi fenomena koji se razmatraju sa više detalja. Stoga jedna preporuka – od mnogo mogućih preporuka koje mogu proizići iz ovih analiza – je da istraživači traže mogućnosti da kombinuju računarske simulacije sa drugim istraživačkim metodama.

2.2.5. Sumarno o računarskoj simulaciji

Naučnici koji su pisali o računarskoj simulaciji dali su brojne komentare o prirodi računarske simulacije i izvrsne sugestije kako da se poboljša istraživanje. Ideja je da se predstavi nova slika u literaturi koja vidi računarsku simulaciju manje kao istraživački metod na koji može da se primeni teorija, a više kao narativni proces koji je deo teorije. Proces je započet pokazujući kako su i računarski modeli i njihovi izlazi - narativni tekstovi, pisani jezikom matematike, sa formulama koje variraju zavisno od čitalaca/slušalaca kojima su namenjeni, stanovišta sa koga se priča i relacija između ljudi koji su uključeni. Zatim je pokazano da posmatrajući računarsku simulaciju u ovom svetlu, mi ne samo da smo u stanju da vidimo obuhvaćene slušaoce, stanovišta i relacije primenjujući proces narativne analize za dobijanje simulacionih tekstova, već takođe i narativna svojstva o savetima koje su naučnici dali za sprovođenje računarske simulacije. Mi ne koristimo računarsku simulaciju da bi pričali svaku priču nego istraživači koriste računarsku simulaciju da bi pričali više vrsta priča. Stoga se postavlja pitanje: **„Koje vrste priča istraživači koriste da se ispriča računarska simulacija?“**

Preporuka koja se ovde nameće je da se istraživanje simulacije sprovodi sa eksplicitnom jasnoćom o narativnoj prirodi računarske simulacije. Potrebno je identifikovati i istražiti tip verbalnog sadržaja (priče, naracije), identifikovati i istražiti karaktere, radnje, sadržaj, glas, potencijalnu publiku, gledišta, hijerarhijske odnose i moral.

3

3. EDUKACIONI ZNAČAJ SIMULATORA

Intezivan tehnološki razvoj, «eksplozija» protoka informacija u svim sektorima privređivanja i posebno razvoj informacionih tehnologija, razvoj novih naučnih disciplina i naučnih metodologija kao i izuzetna dinamika promena u sistemima i njihovom okruženju nisu mogli mimoići ni područje **obrazovanja**. Nova saznanja i nove tehnologije utiču, neposredno ili posredno, na reformu i usavršavanje sistema obrazovanja, i izmene u nastavnim sadržajima, unapređivanjem tehnike i tehnologije nastave. Međutim, dok je tehnologija u drugim sferama i oblastima ljudskog rada i delovanja značajno napredovala, obrazovne institucije su, kao jedan od inertnijih sistema društvenog razvoja, u priličnoj meri ostale na nivou klasične organizacije rada. One su uglavnom zadržale starije obrazovne tehnologije pa postoji potreba da se to unapredi kako ne bi došlo do zaostajanja za dešavanjima u drugim oblastima.

S obzirom na to da su informacione tehnologije nesumljivo postala civilizacijsko obeležje našeg doba kao i neophodan uslov društvenog razvoja, to i za sistem vaspitanja i obrazovanja znači da treba da se uvedu novi obrazovni sadržaji. Ovi sadržaji bi značili promene u načinu izlaganja, obrade i savladavanja gradiva, a samim tim i modernizaciju nastavnog rada. Kao posledica brzog razvoja automatizacije i elektronike stvorili su se nove obrazovne potrebe, ali i otvorile nove i neslućene mogućnosti u didaktici.

Informatičku epohu i elektronsku kulturu komunikacija simbolizuje pojava računara, te stoga računarskoj tehnologiji i pripada posebno mesto pri razmatranju aktuelnih promena u nastavi i učenju. Skoro svuda u svetu u obrazovnim ustanovama, kao deo nastavnih sredstava, koristi se računar. Sa svojim performansama računar je postao interaktivni medij, pa njegova primena u obrazovanju, nastavi i učenju postaje i najveća i

najznačajnija promena u sistemu inovacija[1]. I pored dosadašnjeg iskustva u obrazovanju i dalje se istražuje racionalnost primene računara i posebno njeni efekti na obrazovne ishode nastave.

Organizacija i arhitektura računara je važan sadržaj kojim se bave računarske nauke. Ali, ovaj važan sadržaj može biti i težak za nastavu. Čest je slučaj da studenti imaju teškoće da uspostave vezu između teorijskih znanja i praktičnog iskustva. Zbog prirode ovog sadržaja koja je vezana za hardver, predavači mogu da biraju da li će izlaganje biti na apstraktnom nivou (koristeći samo udžbenike) ili na specifičnom nivou (fokusirajući se samo na jednu specifičnu arhitekturu računara). Ali ni ovo ne mora da bude uvek slučaj. Grafički računarski simulatori omogućavaju vizuelizaciju unutrašnjih operacija računara, takvu da predavači imaju moćno aktivno nastavno sredstvo pomoću koga studenti mogu biti u praktičnoj interakciji sa širokim opsegom ranijih, sadašnjih i budućih arhitektura računara.

Većina obrazovnih institucija ne može da obezbedi laboratorije koje bi sadržale računarske sistema reprezentativne za različite arhitekture. Simulatori su stoga prirodna alternativa pošto su relativno jeftini ili čak i besplatni. Takođe može se koristiti istovremeno veći broj instanci jednog simulatora što je značajno kod procesa obučavanja. Svi ovi razlozi povećavaju zainteresovanost za korišćenje simulatora u nastavi ARS-a (Arhitekture Računarskog Sistema) [27].

3.1. PRIMENA RAČUNARA U OBRAZOVNOM PROCESU

U izvornom značenju reč tehnika označava veštinu kako se nešto radi, a tehnologija[2] skup znanja i veština o različitim postupcima i procesima. S obzirom na to da se značenje ovih reči tokom evolucije ljudskog društva proširilo, danas možemo govoriti o različitim tehnikama, odnosno različitim tehnologijama. Pod pojmom **nastavna tehnika** danas se, uglavnom, podrazumeva skup raznovrsnih sredstava koje predavač i studenti primenjuju u nastavnom procesu[3], mada se ponekad ovim terminom označava i sama umešnost predavača u praktičnom izvođenju nastavnog procesa [1]. Za razliku od nastavne tehnike kod koje je naglasak na sredstvima, **nastavna tehnologija** se, kao širi pojam, pretežno odnosi na organizaciju nastavnog procesa. Zapravo nastavna tehnologija podrazumeva skup mera, postupaka i metoda organizacije nastavnog procesa. Upotreba

[1] Računar sa na univerzitetima primenjivao i pre nego što se razvio do nivoa interakcije
[2] Potiče od grčkih reči – *tehne* (veština) i *logos* (reč, znanje, nauka)
[3] Npr. audio-vizuelna nastavna tehnika; računarska tehnika (računar u nastavi)

termina **obrazovna tehnologija** podrazumeva još šire sagledavanje i shvatanje ovog pojma koje pored organizacije nastave, uključuje i realizaciju i verifikaciju procesa nastave i učenja [3]. Može se reći da se pod obrazovnom tehnologijom podrazumevaju procesi koji su usmereni na ostvarivanje ciljeva obrazovanja i stoga uključuje ljude (predavače, saradnike, studente), ideje, organizaciju, nastavne baze, sredstva, oblike i metode rada, te postupke i sredstva vrednovanja onoga što je u procesu obrazovanja ostvareno [7].

3.1.1. Uloga i značaj primene računara u nastavnom procesu

Intezivna primena informacionih tehnologija u raznim oblastima ljudske delatnosti, uticala je na to da računari postanu sve prisutniji i u procesu nastave i učenja. Oni su uslovili i uzrokovali promene u konceptu obrazovanja, nastavnim sadržajima, tehnologiji nastave i odnosima između predavača i studenata. Računaru, zahvaljujući prednostima koje ima nad ostalim sredstvima, pripad vodeće mesto u procesu uvođenja inovacija u nastavi.

Dijapazon primene računara u obrazovanju praktično je neograničen (nastava, istraživački rad, upravljanje, administracija i dr.). Može se reći da u uslovima «eksplozije znanja» računar sa svojim hardverskim i softverskim mogućnostima predstavlja jedno od najboljih, najbržih, najpreciznijih i najpouzdanijih sredstava za dolaženja do informacija i njihove primene. U obrazovanju njegova primena praktično može doprineti rešavanju problema «informacione barijere», tj. podići nastavni rad na viši, kvalitetniji nivo; učiniti ga efikasnijim, delotvornijim i savremenijim. Kao tehnička baza savremene nastave računari imaju značajnu ulogu u njenom razvoju, bogaćenju, prilagođavanju duhu vremena, potrebama i interesovanjima korisnika. U istraživačkom domenu primenom računara takođe se ostvaruju izuzetno značajni rezultati.

Primena računara u nastavi i obrazovanju implicira korenite i značajne promene u organizaciji nastavnog procesa, položaju i ulogama učesnika u ovom procesu. Savremena obrazovna tehnologija kreira nove uslove učenja i poučavanja, nove poglede na ulogu sadržaja nastave ka razvoju ličnosti i posebno ka razvoju mišljenja studenata.

Računarski uređaji omogućavaju potpuno drugačiju organizaciju nastavnog rada, primerenu individualnim sposobnostima i interesovanjima samih studenata. Osim što računar obezbeđuje kontrolu, regulisanje i upravljanje nastavom i učenjem putem stalne povratne veze koja predstavlja snažan motivacioni podsticaj i čini osnovu sistema vrednovanja i objektivnog ocenjivanja, njegova primena, takođe, osigurava, zahvaljujući mogućnosti povezivanja sa bazom podataka, bržu i efikasniju emisiju, transmisiju i

apsorbciju znanja što svakako, doprinosi većoj aktivnosti, samostalnosti i kreativnosti učesnika nastavnog procesa.

Nastavno gradivo obrađeno na računaru, nazvano „**courseware",** pojavilo se u raznim nastavnim predmetima: matematika, fizika, medicina, tehničke nauke, biologija, strani jezici, pravo i dr.

U visokoškolskom obrazovanju, prednosti nastave uz pomoć računara takođe su sadržane u mogućnostima veće misaone mobilnosti, aktiviranja i samostalnog rada studenata. Savremeni računari pružaju mogućnost simultanog gledanja slike, slušanja govora i korišćenja multimedijskih izvora saznanja, što, svakako, doprinosi bržem i potpunijem usvajanju gradiva, trajnijem pamćenju naučenog, efikasnijem korišćenju i kreativnijoj primeni usvojenih znanja.[4] Računar sa studentom komunicira pismeno i usmeno, vodi dijalog, pruža neophodne informacije, predstavlja grafikone, slike, filmove, stranice knjiga, projekcije, simulacije, daje objašnjenja pokazanog, upućuje na rešavanje problema, po potrebi daje dopunska uputstva, ispravlja greške i ocenjuje rezultate učenja.

Svoj status i ulogu u visokoškolskom obrazovanju računar je posebno učvrstio s pojavom veštačke inteligencije (*artifical intelligence*) i ekspertnih sistema. Od ključne je važnosti primena veštačke inteligencije u nastavnom procesu za demonstriranje teorema i formula, zajedno s automatski dobijenim rešenjima problema; u modeliranju kognitivnih procesa i drugo. S druge strane ekspertni sistemi zajedno sa simulacijskim modelima, mogu predstavljati značajnu podršku u nastavi na univerzitetskom nivou, između ostalog, za aplikacije u prirodnim naukama, za elaboraciju studija slučaja, u nastavi egzaktnih predmeta, za upoznavanje pojava iz stvarnog života itd.

Primenu računara u nastavi valja kombinovati i s drugim nastavnim medijima, jer upravo takva komplementarna primena izvora znanja s njihovim optimalnim mogućnostima može bitno doprineti povećanju efikasnosti visokoškolske nastave. Zapravo, radi se o potrebi kombinovanja različitih metoda, oblika i sredstava u «*jedan celoviti didaktički višestruko funkcionalan miltimedijalni sistem»* koji će osavremeniti nastavu i učiniti je primerenom zahtevima i potrebama vremena u kojem živimo.

Računar ne vrši samo funkciju određenog nastavnog sredstva, već sve šire vrši i funkciju metode učenja.[5] Pri tom, predavač nije zamenjen mašinom, već timom programera koji su u «funkciji» predavača i efikasnije realizacije nastavnog procesa.

[4] Posebna vrednost multimedijske prezentacije je transfer znanja.

[5] Zamenjujući brojne aktivnosti predavača i neke aktivnosti studenata te postaje specifično sredstvo - metoda

U nizu razmatranih mogućnosti i prednosti primene računara u visokoškolskom obrazovanju i obrazovanju uopšte, od posebnog, moglo bi se reći i neprocenjivog značaja je i njegova uloga u zadovoljavanju potrebe savremenog čoveka za samoobrazovanjem i permanentnim obrazovanjem kao neophodnim pretpostavkama i činiocima ličnog razvoja i društvenog progresa uopšte.

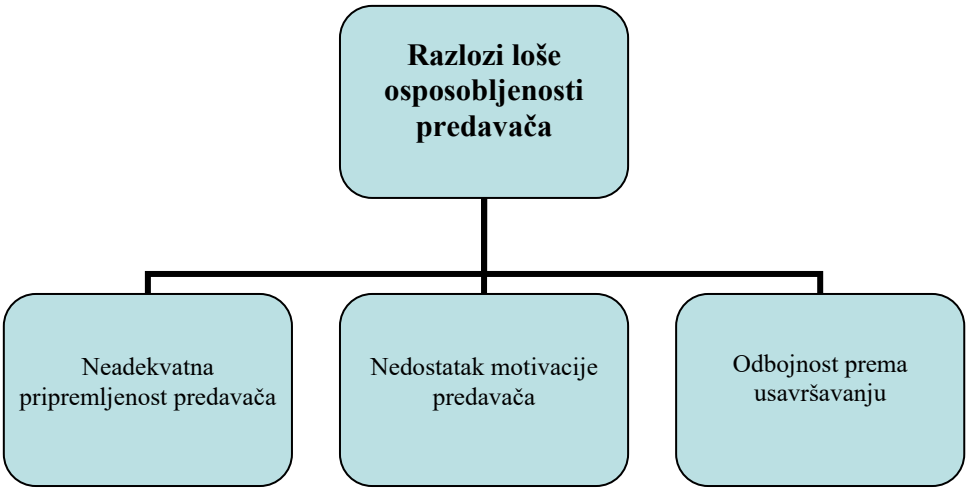
3.1.2. Problemi vezani za primenu računara u nastavnom procesu

I pored toga što primena računara u visokoškolskoj nastavi krije u sebi značajan saznajni i didaktički potencijal, moraju se uzeti u obzir određeni problemi, ograničenja i nedostaci koji se pojavljuju prilikom njegovog korišćenja u nastavnom radu. Samu dinamiku primene računara u nastavi uslovljava niz objektivnih činilaca: nedostatak finansijskih sredstava za njihovu nabavku, nedostatak odgovarajućih softvera, skupi programi «**courseware»** i potreba njihovog stalnog menjanja, tj. osavremenjivanja, nemogućnost ostvarivanja jedinstvenog jezika za sve vrste mikroračunara, jer hardver nije standardizovan i tako dalje.

Posebno su interesantni i važni problemi koji se odnose na osposobljenost predavača i saradnika za korišćenje i primenu računara u nastavnom radu. Kad se govori o osposobljenosti predavača, misli se, u prvom redu, na kognitivno osposobljavanje (usvajanje potrebnih znanja, veština, sposobnosti itd.). Računarska pismenost još uvek, u većem broju slučajeva, nije postala sastavni deo tzv. opšte pismenosti, te predavači vrlo često ispoljavaju nezainteresovanost, odbojnost ili otpor prema primeni računara u nastavi. Takođe često pokušavaju da u novi tehnološki sistem smeštaju stari sadržaj, stari način poučavanja ili učenja, čime smanjuju rezultate i efikasnost nove obrazovne tehnologije ili, pak, doživljavaju sopstveni neuspeh koji zatim pripisuju računarima. Suštinski posmatrano, ovi problemi nastaju, upravo, **zbog nedostatka motivacije predavača, neadekvatne pripremljenosti i nepoznavanja pedagoškog i didaktičkog smisla i značaja računara za nastavu i vežbe** (slika 3.1).

U velikom broju zemalja preduzimaju se značajne mere da se budući predavači tokom svog školovanja pripreme za efikasnu upotrebu računara u nastavi, učenju, vrednovanju, istraživanju i drugim poslovima u obrazovanju. Tako se razvijaju forme i oblici posle školskog usavršavanja i permanentnog obrazovanja predavača iz ove oblasti. Predavače ne bi trebalo puno angažovati na pitanjima programiranja, već ih osposobiti kako da kompjuterizuju metode poučavanja, otkrivaju nove mogućnosti nastave i učenja i izgrađuju sopstvenu strategiju pedagoškog rada. Predavači ne moraju biti eksperti u

programiranju, ali se zato za kratko vreme mogu osposobiti da koriste preimućstva koja pružaju računari, mogu proširivati i bogatiti svoju pedagošku funkciju, uneti više smisla u obrazovni proces i učiniti ga prihvatljivijim. S druge strane, neophodno je, međutim, osigurati takvu kadrovsku bazu, odnosno, stručnjake koji bi bili angažovani za rukovođenje opremom i izradu kvalitetnih programa za nastavu i učenje uz pomoć računara.



Slika 3.1 Razlozi loše osposobljenosti predavača za korišćenje i primenu računara u nastavi

Primena računara u nastavi još uvek nije na zadovoljavajućem nivou upravo zbog nedovoljnog naučno verifikovanog iskustva u ovoj oblasti. Računari mogu unaprediti nastavu i obrazovanje ukoliko se primene na pravom mestu, u pravo vreme, sa adekvatnim sadržajem i metodičkim osmišljenim tehnikama i postupcima. To, zapravo, znači da bi bila potrebna posebna metodika primene računara u nastavi, a nova funkcija predavača u savremenim tehničko-tehnološkim uslovima pretpostavlja i radikalnu promenu filozofije obrazovanja, korenite promene psihološkog i pedagoškog obrazovanja, osavremenjivanje metodičke spreme, potpunije poznavanje i adekvatnu upotrebu savremenih obrazovnih tehnologija.

3.2. KORIŠĆENJE RAČUNARSKE SIMULACIJE U NASTAVI

Korišćenje računarske simulacije u naučnim istraživanjima veoma značajno se povećalo tokom poslednjih decenija. „Računarska simulacija sada udružuje teoriju i eksperiment kao treći deo naučnog saznanja. Simulacije u svim oblastima nauke i tehnike igraju sve veću ulogu. Međutim, kako se proširuje korišćenje simulacija, raste i potreba za proračunima koji imaju sve veću moć, fleksibilnost i korisnost.“ (NSF Advanced Computational Research program announcement, NSF 98-168)

Sistemi za simulaciju i vizuelizaciju su tradicionalno “stand-alone” [28] (nezavisne) računarske aplikacije. Poslednjih godina, sa sve rasprostranjenijim korišćenjem web-a i naprednih tehnologija, razvija se sve više sistema za simulaciju i vizuelizaciju zasnovanih na web-u. Mnogi od ovih razvijenih sistema koriste objektne računarske arhitekture, kao što su CORBA, RMI, EJB, i autonomne agente, kako bi obezbedili efektivno širenje aplikacija preko web-a. Na dosta univerziteta, a posebno u USA, (npr. University of Pennsylvania), sistem za vizuelizaciju u realnom vremenu zasnovan na web-u (Web based realtime visualization system) je tako razvijen da omogućava korisniku da interaktivno i dinamički manipuliše modelima neuronskih mreža i da podaci budu generisani pomoću simulacije eksperimenata. Glavna prednost postavljanja “stand-alone” aplikacija na Web-u je širenje dostupnosti web pretraživača, koji mogu da budu korišćeni kao “front-end” korisnički interfejs aplikacije za udaljenog korisnika koji može potencijalno da bude lociran bilo gde na zemaljskoj kugli. Razvoj sistema neuronskih mreža omogućava postavljanje simulacija i generisanje podataka na web-u i obezbeđuje udaljenim korisnicima da konfigurišu i posmatraju simulacije u realnom vremenu.

Međutim, postoje i iskustveni problemi u identifikovanju kvalifikovanih studenata, programera-istraživača koji imaju odgovarajući nivo znanja i veština da učestvuju u istraživačkim projektima. Nažalost, na većini fakulteta, kako u svetu, tako i kod nas, ne postoje kursevi iz računarskih nauka u oblasti računarske simulacije. Studenti prirodnih nauka možda imaju dovoljno primenljivog znanja, ali obično nemaju zahtevane veštine u razvoju nizova softverskih sistema za simulaciju i vizuelizaciju pa se na tom problemu mora intezivno raditi.

U naučnim simulacijama uobičajeno je da se koriste alati za vizuelnu analizu podataka, kao “stand-alone” aplikacije. Ranije je posmatrač vizuelizacije morao biti fizički prisutan uz računar na kome je pokrenuta vizuelizacija (na primer u učionici ili laboratoriji). Aktuelna prednost tehnologije zasnovane na web-u je u tome što omogućava

razvoj „on-line" sistema za simulaciju i vizuelizaciju. Sistem vizuelizacije zasnovan na web-u razvijen je da bi omogućio udaljenim posmatračima da interaktivno i dinamički manipulišu modelima i generišu podatke.

Sistem za vizuelizaciju zasnovan na web-u omogućava značajne prednosti u učenju. Predavač može da pokazuje interaktivan i „real-time" naučni model studentima, kako u učionici, tako i na udaljenoj lokaciji, npr. zumiranjem u okviru i van modela i rotiranjem modela u različite posmatračke perspektive. Pošto je sistem za vizuelizaciju povezan sa mašinom za simulaciju, predavač može da pokaže studentima dinamičke sklopove koji se menjaju tokom vremena, da daje nove podatke itd. Sa sistemom za vizuelizaciju u realnom vremenu u prostoru, posmatrači, bilo da su to istraživači ili studenti, mogu da vide dinamičke promene sistema u realnom vremenu.

Integrisani simulacioni i vizuelizacioni sistem omogućava udaljenim korisnicima da budu u interakciji sa mašinom za simulaciju, te da rekonfigurišu mrežnu arhitekturu i posmatraju efekte promene posle ponovnog pokretanja eksperimenta. U distribuiranoj sredini za učenje, ova mogućnost za udaljenim manipulisanjem virtuelnim eksperimentom će povećati mogućnost uključivanja eksperimenata i demonstracija u predmete kakvi su fizika, hemija, biološke nauke itd.

3.2.1. Potrebna znanja i veštine za razvijanje sistema za simulaciju i vizuelizaciju

Da bi student programer/istraživač uspešno učestvovao u istraživačkom projektu kakav je opisan, potrebna su mu sledeća znanja i veštine [28]:

1) Iz oblasti računarskog programiranja:

Završeni kursevi iz računarskih nauka kao što su Programiranje, Objektno-orijentisano programiranje i Internet programiranje. Poseban akcent se stavlja na Java programiranje.

2) Iz oblasti matematike:

Uspešno programiranje aplikacija za simulaciju i vizuelizaciju zahteva dovoljno znanja iz matematike, koja se mogu steći na kursevima: Analiza i Linearna algebra. Studenti mogu dobiti potrebna znanja od naprednih kurseva koji su usmereni ka Modelima simulacije.

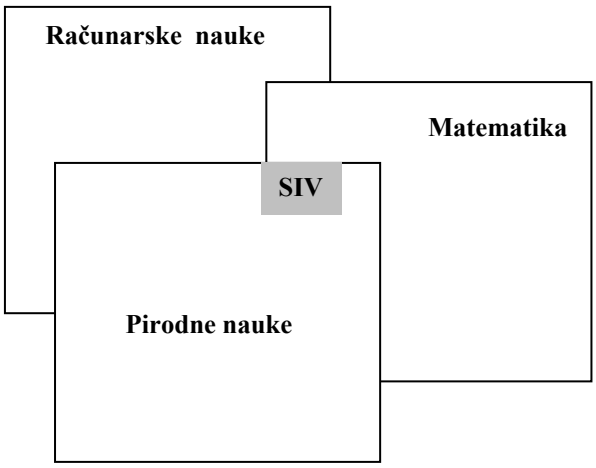
3) Iz oblasti prirodnih nauka:

Sa obzirom na širok sperktar predmeta u prirodnim naukama, nepraktično je tražiti od studenata da pohađaju kurseve za svaku oblast. Međutim, važno je da

studenti detaljno izuče najmanje jedan kurs iz osnovnih naučnih područja, na primer fizike. Uobičajena je praksa u oblasti razvoja softvera da osobe koje razvijaju softver prođu analitički stadijum da bi razumeli oblast aplikacije, pre nego što pristupe neposrednom razvoju softvera. Zahtevana fundamentalna naučna znanja iz kurseva prirodnih nauka, anticipiraju da će student biti sposoban da izabere znanja specifično potrebna za novu aplikaciju.

Interdisciplinarni kursevi su često potrebni da bi se ostvarili ciljevi usavršavanja studenata programera/istraživača da budu sposobni da razvijaju programe simulacije i/ili vizuelizacije ili da postavljaju istraživačke eksperimente koristeći razvijene alate. Ilustrativno, na slici 3.2, Simulacija I Vizuelizacija (SIV) je predstavljena kao koncentracija specifičnih znanja iz tri predmetne oblasti (presek znanja iz ovih oblasti). Interdisciplinarnost vodi studente ka usvajanju znanja i/ili veština iz više oblasti tako da postanu kvalifikovani programeri i/ili istraživači u projektima simulacije i vizuelizacije.

Kombinovanje fundamentalnih znanja i rezultata istraživanja iz više disciplina, omogućava studentima suštinski trening u računarskom programiranju, matematici, prirodnim naukama, kao i napredni kursevi iz različitih oblasti povezani sa naučnim pristupom simulaciji i vizuelizaciji.



Slika 3.2 Simulacija i vizuelizacija u interdisciplinarnoj oblasti studija (SIV – Simulacija I Vizuelizacija)

3.3. SIMULATORI U RAČUNARSKOJ TEHNICI

Studenti osamdesetih godina su imali papir i olovku kao sredstva da kreiraju CPU komponente koristeći Bulovu algebru i Karnaove mape dok danas oni mogu koristiti CPU simulatore. Simulatori im omogućavaju da proučavaju operacije koje računar izvodi tokom izvršavanja programa i da vrše vizuelno upoređivanjem različitih događaja.

Kako se složenost i varijacije hardvera računara svakodnevno povećavaju, to se za direktno proučavanje određene arhitekture (realnog sistema) smanjuju mogućnosti i interes. Zbog toga umesto ka realnom sistemu, mnogi se okreću ka simulatorima kao pomoćnim sredstvima u podučavanju o arhitekturi i funkcijama računara. Postoje mnogi simulatori koji su dostupni na Internetu, ali prava je umetnost pronaći pravi simulator za kurs ili seminar koji se želi izvesti.

3.3.1. Proučavanje arhitekture računara

Simulatori, kao pomoćno sredstvo u izučavanju arhitekture računara, pomažu studentima da bolje razumeju npr. von Neumann-ovu arhitekturu i njenu vezu sa asemblerskim jezikom, da shvate interakciju između arhitekture, asemblerskog jezika i operativnog sistema. Oni takođe, dozvoljavaju studentima da uče operacije računara posmatrajući i/ili utičući na osnovne događaje tokom izvršenja softvera simulatora. Na taj način studenti dobijaju vredno iskustvo o važnim idejama uključujući u koncept memorisanog programa i ciklusa tipa uzmi-izvrši.

Razumevanje ARS-a, (ili CSA-a – Computer Systems Architecture na engleskom) je suštinsko za razumevanje računarske nauke. Postoji tendencija, da se u nastavi informacione i komunikaciona tehnologije (IKT-a) zanemaruje ARS. Međutim, poučavanje IKT-a bez ARS-a je kao učenje srpskog jezika bez učenja ćiriličnog pisma. Dva su glavna razloga što se zanemaruje ARS u nastavi IKT-a [30]:

- postoje pogrešna shvatanja o efektima tehnoloških promena, i
- postoji tendencija da se koriste neprilagođena didaktička sredstva.

Uloga tehnoloških promena

Postavlja se pitanje: „Sa brzim razvojem informacionih tehnologija (IT-a), da li je pogrešno učiti nešto što će zastareti za veoma kratko vreme?“ Odgovor je: „Ne!”. Iako je razvoj IT aplikacija veoma brz i promene nagle, osnovni principi ARS-a su isti kao i pre 50 godina. Učenje tih osnovnih principa omogućava studentima da razviju sopstveno shvatanje uticaja IT aplikacija na znanja o bazičnim principima digitalnih računara.

Sa razvojem računara adekvatne laboratorije koje bi pratile taj trend bi za većinu univerziteta bile preskupe. Stoga su logična zamena za njih simulatori arhitekture računara, koji su jeftiniji ili čak i besplatni, a i studenati ih mogu koristiti istovremeno. To je jedan od razloga zašto je došlo do povećanog interesovanja za korišćenje simulatora.

U cilju boljeg pripremanja studenata mnogi univerziteti su integrisali korišćenje simulatora u nastavni plan. Ima studenata koji uče samo osnove o računarima i oni su više zaokupljeni pokretanjem simulatora (running) nego razumevanjem i modifikovanjem implementacija, dok sa druge strane ima i onih naprednijih studenata koji bi učestvovali u razvijanju novih arhitekturnih modela. U nekim slučajevima studenti mogu i da redizajniraju npr. procesor koliko god puta hoće bez bojazni da će se napraviti «ozbiljnija» greška.

Interaktivni simulatori omogućavaju aktivno učenje dozvoljavajući studentima da dizajniraju sopstvene hipotetičke mašine, da ih programiraju, izvršavaju softver na njima i da koriste simulacije kako bi razumeli operacije stvarnih mašina. Kao pomoćno sredstvo simulatori su atraktivni jer studenti uče npr. osnovne detalje računarskih operacija na različitim nivoima imajući pristup kad i gde žele sa malo ili čak nimalo ulaganja.

Vrlo su interesantni i Internet dostupni simulatori koji omogućavaju studentima eksperimente počev od programiranja modela iz prošlosti (historical machines) do kreiranja njihovih ličnih novih arhitektura.

Konačno, kreiranje softverske simulacije računara je edukaciono iskustvo slično pravljenju pravog računara sa hardverskim komponentama ali znatno jeftinije, fleksibilnije u smislu dozvoljavanja studentu da napravi grešku i proširivije u dodavanju dodatnih funkcionalnosti.

3.3.1.1. Unapređenje nastave o arhitekturi računara

Cilj svakog dobrog predavanja o nekoj temi mora da bude uvođenje studenata u nove pojmove u najnaprednijem obliku. Predmet izlaganja mora da bude analiziran i mora da postoji razvijen plan realizacije tako da ga studenti što lakše prihvate i usvoje. Ukoliko se kroz učenje studentima omogući i realno iskustvo, onda je sigurno da će očekivanja i studenta i predavača biti postignuta. Ciljevi nastave moraju da budu tako podešeni da predavanja budu tako struktuirana kako bi omoćila studentima da profitiraju do maksimuma sopstvenih sposobnosti [30].

Dobro struktuirana predavanja omogućavaju da studenti dosegnu do znanja i veština na različitim nivoima. Predavanja bi trebalo da imaju dovoljno dodatnog materijala

kao izazov za sposobnije studente, dajući im mogućnosti da razviju svoje sposobnosti, dok istovremeno treba da obezbede bazične pojmove za one koji su manje sposobni dajući im osnovna znanja koja se od njih očekuju.

Unapređivanje nastave je ideal kome se težilo oduvek. Zbog toga, sa razvojem IT-a, predavači se trude da u nastavne programe uvrste i arhitekturu računara sa programiranjem [30]. Iako je u ovoj vrsti materije zastupljen visok nivo apstraktnosti ipak to i predstavlja izazov „novog doba" kako za predavače tako i za studente. Sadržaj nastave vezane za arhitekturu računara (na određenom nivou detaljnosti) može biti i težak za određeni broj studenta iz sledećih razloga:

Prvo, potrebno je ovladati širokim opsegom veština i detalja – od primene tranzistora kao logičkog gate-a do arhitekture računarskog sistema kao celine.

Drugo, arhitektura aktuelnih računarskih sistema sadrži karakteristike kakve su:

- jedinstvenost izrade,
- složenost za razumevanje i
- kriptička dokumentovanost.

  Ovo određuje da laboratorijski eksperimenti za studente predstavljaju izazov koji zahteva široku pripremu, a rezultat toga mogu da budu veštine i znanja koja nisu prenosiva na druge mašine.

Treće, arhitektura računara je za većinu studenata kompleksna – dok studenti mogu imati visok nivo računarske pismenosti o softverskim aplikacijama, jezicima i periferijama, mnogi imaju nekompletne, nerealne i ponekad nezasnovane predstave o tome kako računar radi.

Četvrto, studenti su preplavljeni sa marketinškim informacijama o računarima iz reklama na Internetu, televiziji, radiju i časopisima. Sve dok studenti ne dobiju kvalitetne informacije postoji opasnost o nedovoljnom poznavanju arhitekture računara, a takođe i od pogrešnog usvajanja pojedinih ideja.

Simulator arhitekture računara može da rasvetli ključne karakteristike računara sa odgovarajućom pedagoškom vrednošću i eliminiše nepotrebne detalje. Napokon, softversko kreiranje simulatora računara je oslobađajuće pedagoško iskustvo slično sa izgrađivanjem hardvera realnog računara, ali manje košta, mnogo fleksibilnije omogućava studentima da prave greške i otkrivaju nova znanja, i lako se proširuje njegova funkcionalnost dodajući nove komponente. Simulatori odgovarajuće obrazovne vrednosti mogu biti napravljeni za različite aktuelne arhitekture mašina ili to mogu biti

generalizovane/eksperimentalne mašine i mogu ih kreirati studenti kao svoje projekte. Tako npr. LMC (Little Man Computer) simulatori predstavljaju klasu eksperimentalnih mašina obrazovne vrednosti.

3.4. KATEGORIZACIJA SIMULATORA

Postoje različite klasifikacije simulatora računarske arhitekture. Jedna od podela koja se često koristi je podela po tipu simulatora i podela prema predznanju korisnika (studenata).

3.4.1. Kategorizacija simulatora prema tipu

Simulatori računarskog sistema se mogu podeliti u sedam različitih kategorija (za svaku od ovih kategorija su navedeni Internet dostupni besplatni simulatori sa odgovarajućom Internet adresom sa koje se mogu preuzeti). Te kategorije su [19]:

- Simulatori ranijih mašina (Historical Machine Simulators)
- Simulatori digitalnih kola (Digital Logic Simulators)
- Simulatori jednostavnih hipotetičkih mašina (Simple Hypotetical Machine Simulators)
- Simulatori posredovani setom instrukcija (Intermediate Instruction Set Simulators)
- Simulatori napredne mikroarhitekture (Advanced Microarchitecture Simulators)
- Simulatori multiprocesora (Multi-processor Simulators)
- Simulatori memorije (Memory Subsystem Simulators)

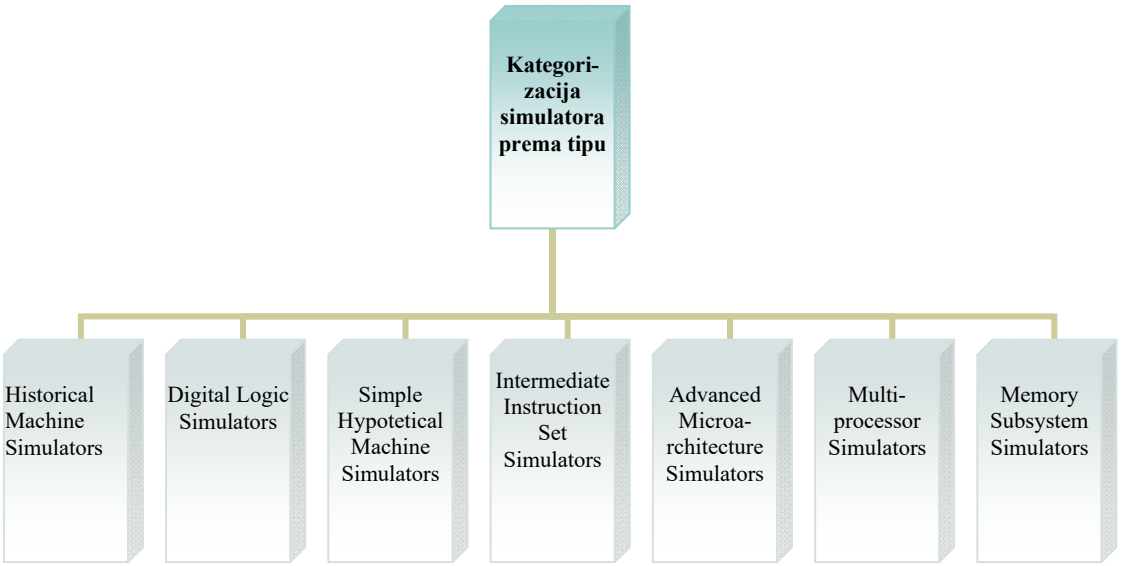
Simulatori iz 1. i 2. kategorije imaju zadatak, u edukativnom smislu, da objasne osnovne koncepte uvoda u arhitekturu računara koji su već odavno istraženi.

Simulatori iz 3., 4. i 5. kategorije generalno pokrivaju uvod, arhitekturu računara i kurseve arhitekture računara koji se nalaze u mnogim nastavnim planovima računarske tehnike.

Multiprocesorski simulatori proširuju kategoriju napredne mikroarhitekture fokusirajući se na paralelne arhitekture sa više procesora.

Simulatori memorije obuhvataju simulatore koji se fokusiraju na interakciju procesora i njegove keš memorije.

Na slici 3.3 dat je grafički prikaz kategorizacije simulatora prema tipu.



Slika 3.3 Kategorizacija simulatora prema tipu

3.4.1.1. Simulatori ranijih mašina

Učenje operacija računara se često može uvećati korišćenjem primeraka mašina koje više uopšte i ne postoje (sem onih u muzejima) ili ako postoje onda su preskupe da bi se opravdalo njihovo korišćenje jedino u edukacione svrhe.

Korišćenje simulacija arhitekture računara omogućava predavaču da izlaže koncepte bilo koje mašine za koju su simulatori dostupni. Često su mašine koje su u prošlosti korišćene najbolji primer koncepta arhitekture. Korišćenjem simulatora zastarelost tih mašina se prevazilazi jer se one mogu virtuelno ponovo kreirati. Internet dostupni simulatori ove kategorije dati su u tabeli 3.1 [19].

Tabela 3.1 Internet dostupni simulatori ranijih mašina

Simulator	**Internet adresa**	**Operativni sistem pod kojim radi**
Analytical Engine	www.fourmilab.ch/babbage/applet.html	Windows
Apple IIe	www.quark.netfront.net:6502	Unix, Windows
Atari ST	www.complang.tuwien.ac.at/nino/stonx.html	Unix, MSDOS, Windows
Commodore Amiga	www.freiburg.linux.de/~uae	Unix, Mac
Commodore 64	www.unimainz.de/~bauec002/FRMain.html	Unix, Windows
DEC PDP-8	www.cs.uiowa.edu/~jones/pdp8/	Unix, Windows
DEC PDP-11	http://www.update.uu.se/pub/ibmpc/emulators/	DOS
EDSAC	http://www.dcs.warwick.ac.uk/~edsac/	Win32, Mac
Sinclair QL	www.geocities.com/SiliconValley/Heights/1296	Windows, Mac
Turing Machine	www.cs.brandeis.edu/paulq/Turing/TuringAppletMac.html	Unix, Windows

3.4.1.2. Simulatori digitalnih kola

Moderni računari se sastoje od velikog broja veoma prostih struktura. U ovoj kategoriji simulatori digitalnih kola opisuju osnovne elemente na kojima počiva hardver: osnovne logičke prekidačke elemente, analizu kola (inplementaciju i minimizaciju), kašnjenja, flip-flopove, registre, logičke strukture (multipleksere, dekodere, komparatore), memorijske elemente (ROM, PROM, RAM). Internet dostupni simulatori ove kategorije dati su u tabeli 3.2 [19].

Tabela 3.2 Internet dostupni simulatori digitalnih kola

Simulator	Internet adresa	Opis
6.111 Digital Simulator	www.mit.edu/people/eichin/thesis/usrdoc.html	skup makroa koji mogu biti korišćeni za pravljenje programa koji simulira kolo
Digital Logic Simulator	www.cs.gordon.edu/courses/module7/logic-sim/example1.html	internet-bazirani klikni i prevuci simulator logičkih kola koji ilustruje operacije prostog kola
Digital Workshop	www.cise.ufl.edu/~fishwick/dig/DigSim.html	web-zasnovana Java aplikacija koja prikazuje izvršavanja ranije napravljenih logičkih kola
esim Simulator	www.cse.ucsc.edu/~elm/Software/Esim/index.html	napredan alat za pravljenje komplesnih digitalnih kola, radi pod Unix-om
Interactive Full-Adder	www.acs.ilstu.edu/faculty/javila/acs254/fullAdder/FullAdder.html)	web-zasnovana interaktivna demonstracija implementacije kompletnog brojača
Iowa Logic Simulator	www.cs.uiowa.edu/~jones/logicsim/	simulator napravljen u Pascalu za digitalne sisteme od tranzistora do CPU-a
MIT Digital Logic	web.mit.edu/ara/www/ds.html	grafički simulator za flip-flopove sa kompletnom analizom
Multimedia Logic Kits	www.softronix.com/logic.html	vizuelni sistem za dizajniranje i testiranje prostih kola
Simcir circuit simulator	www.tt.rim.or.jp/~kazz/simcir/	web-zasnovana Java aplikacija za analiziranje prekidačkih elemenata

3.4.1.3. Simulatori jednostavnih hipotetičkih mašina

Sa povećanjem kompleksnosti realnih mašina one ujedno i postaju manje podesne za izučavanje na kursevima arhitekture računara. Simulatori jednostavnih hipotetičkih mašina tu mogu odigrati značajnu ulogu omogućavajući studentima pristup internim operacijama sistema (što nije moguće sa stvarnim procesorima).

Ovi simulatori ilustruju veoma bitne koncepte kao što su: koncept memorisanog programa, princip sekvencijalnog izvršenja, komplikovanost predstavljanja podataka, skup suštinskih instrukcija, proces prenosa instrukcija tipa uzmi i izvrši ciklus, kao i korišćenje registara.

Ukratko, ovi simulatori omogućavaju predavaču da selektivno usmeri pažnju na važne koncepte. Internet dostupni simulatori ove kategorije dati su u tabeli 3.3 [19].

Tabela 3.3 Internet dostupni simulatori jednostavnih hipotetičkih mašina

Simulator	Internet adresa	Opis
CASLE	shay.ecn.purdue.edu/~casle/	eksperimentiše sa registrima
CPU Sim	www.cs.colby.edu/~djskrien/	emulator na nivou transfera registra Mas OS
EasyCPU	www.cteh.ac.il/departments/ education/cpu.htm	animira osnovne i složene Intel 80x86 operacije
Little Man Computer	www.acs.ilstu.edu/faculty/javila/lmc/	vizuelizacija LMC
PIPPIN	www.cs.gordon.edu/courses/cs111/ module6/cpu-sim/cpusim.html)	binarni i simbolički mod CPU-a, označava kretanje podataka
oirsc&urisc	www.pdc.kth.se/~jas/retro/retromuseum.html	ekctremni RISC kompjuter sa jednom instrukcijom
Simple Computer Emulator	www.Beachstudios.com/sc/	emulator sa memorijskim ćelijama i I/O jedinicama

3.4.1.4. Simulatori posredovani setom instrukcija

Dosad opisani simulatori su dizajnirani da koriste samo jednostavno adresiranje, ograničen set instrukcija i veoma jednostavan memorijski model. Nasuprot tome, simulatori posredovani setom instrukcija teže da sadrže realnije modele adresiranja, mnogo ozbiljniju memorijsku hijararhiju, skoro kompletan set instrukcija i poneki mehanizam prekida. Kao rezultat toga na njima se mogu izvršavati mnogo realnije programske aplikacije. Internet dostupni simulatori ove kategorije dati su u tabeli 3.4 [19].

Tabela 3.4 Internet dostupni simulatori posredovani setom instrukcija

Simulator	Internet adresa	Operativni sistem pod kojim radi
LC2	www.mhhe.com/patt	Unix , Windows
Relative Simple Computer System Simulator	www.awl.com/carpinelli	Unix , Windows
SIMHC12	www.aracnet.com/tomalmy/68hc.html	Unix, Mac
AMD SimNow!	www.x86-64.org/downloads	Unix
SPIM	www.cs.wisc.edu/~larus/spim.html	Unix , DOS , Windows
SPIMSAL	www.cs.wisc.edu/~larus/spim.html	Windows , Mac

3.4.1.5. Simulatori napredne mikroarhitekture

Simulatori napredne mikroarhitekture su dizajnirani tako da dozvoljavaju nadgledanje izvršenja mašinskog jezika na nivou mikrokoda. Napredni simulatori se mogu koristiti za ispitivanje i utvrđivanje prednosti i mana raznih tehnika kao što su „pipelining", „branch prediction" i „parallelism" na nivou instrukcija.

Neki od ovih simulatora su mikroprogramabilni dozvoljavajući tako studentima da eksperimentišu sa nizom instrukcija. Za većinu simulatora iz ove kategorije su već

napisane knjige i bili bi veoma podesni za više kurseve arhitekture računara. Internet dostupni simulatori ove kategorije dati su u tabeli 3.5 [19].

Tabela 3.5 Internet dostupni simulatori mikroarhitekture

Simulator	Internet adresa	Operativni sistem pod kojim radi
DLX	www.max.stanford.edu/pub/max/ pub/hennessy-patterson.software	Unix
DLXview	www.yara.ecn.purdue.edu/~teamaaa/dlxview	Unix
Mic-1 Simulator	www.ontko.com/mic1/	Unix , Windows
Micro Architecture	ww.kagi.com/fab/msim.html	Mac
MipSim	www.mouse.vlsivie.tuwien. ac.at/lehre/rechnerarchitekturen/ download/Simulatoren	Unix , Windows
SimpleScalar	www.simplescalar.org	Unix
SuperScalarDLX	www.rs.etechnik. tu-darmstadt.de/TUD/res/	Unix
WinDLX	www.ftp.mkp.com/pub/dlx/	Windows

3.4.1.6. Simulatori multiprocesora

Simulatori multiprocesora se značajno razlikuju od jednoprocesorskih simulatora. Jedna od razlika je u tome što se kod multiprocesorskih simulatora zahtevaju funkcije koje uopšte ne postoje kod jednoprocesorskih, kao što je npr. deljenje memorije. Druga razlika je rezultat istovremenih izvršavanja, ispravna simulacija mora odslikavati činjenicu da se instrukcije na različitim procesorima izvršavaju istovremeno.

Napomenimo da je vreme izvršenja simulacije svojevrstan tehnički izazov budući da vreme simulacije raste proporcionalno sa brojem procesora u simulaciji. Samo korišćenje ovih simulatora je mnogo komplikovanije u poređenju sa korišćenjem jednoprocesorskih simulatora tako da su ovi simulatori podesni za korišćenje na višim kursevima arhitektura računara.

Simulatori multiprocesora su veoma precizno i detaljno razvijani i kao rezultat toga su dobijeni simulatori sa dobrim performansama koji se koriste u istraživačke svrhe. Neki od simulatora navedeni u tabeli 3.6 [19] se i danas koriste u istraživanju.

Tabela 3.6 Internet dostupni multiprocesorski simulatori

Simulator	Internet adresa
ABSS	www.arithmetic.Stanford.edu/~lemon/abss.html
MINT	www.cs.rochester/u/veenstra/
Proteus	www.ee.lsu.edu/koppel/proteus.html
RSIM	rsim.cs.uiuc.edu/rsim/
SimOS	simos.stanford.edu/introduction.html
Wisconsin Simulator Page	www.cs.wisc.edu/arch/www/tools.html

3.4.1.7. Simulatori memorije

Simulatori memorije su namenjeni za više kurseve arhitekture računara na kojima se vrše ozbiljna izučavanja, analize i upoređivanja. Simulatori ove kategorije se koriste za modelovanje i analizu velikog broja različitih memorijskih hijerarhija.

U tabeli 3.7 [19] su prikazani simulatori korišćeni u eksperimentima na različitim nivoima keš memorije, sa različitim veličinama memorije i stepenom povezanosti.

Tabela 3.7 Internet dostupni simulatori memorije

Simulator	Internet adresa	Operativni sistem pod kojim radi
Cacheprof	www. Cacheprof.com	Unix
Cashe simulator	www.ece.gatech.edu/ research/labs/research	Unix , Windows
CACTI	www.research.compaq.com/wrl/people/jouppi/CACTI.html	Unix , Windows
Dinero IV	www.cs.wisc.edu/~markhill/DineroIV/	Unix
PRIMA	www.dsi.unimo.it/staff/st36/imagelab/prima.html	Unix

3.4.2. Kategorizacija simulatora prema predznanju studenata

Savremena softverska tehnologija omogućava obogaćivanje okruženja za učenje vizuelizacijom. Na kursevima ARS-a ovaj oblik aktivnog učenja se unapređuje korišćenjem animacionih simulatora. Studenti mogu da uče osnove ARS-a pomoću vizuelne opservacije i u interakciji sa animiranim podacima na posebnim virtuelnim ili realnim mašinama. Obično su kursevi za upoznavanje sa računarima podeljeni na dva dela: organizacija i arhitektura računara. U zavisnosti od predznanja koje studenti poseduju definišu se i sredstva kojima će se postići odgovarajući nivo znanja. U tom smislu bira se i odgovarajuća kompleksnost simulatora sa kojima se izvodi nastava.

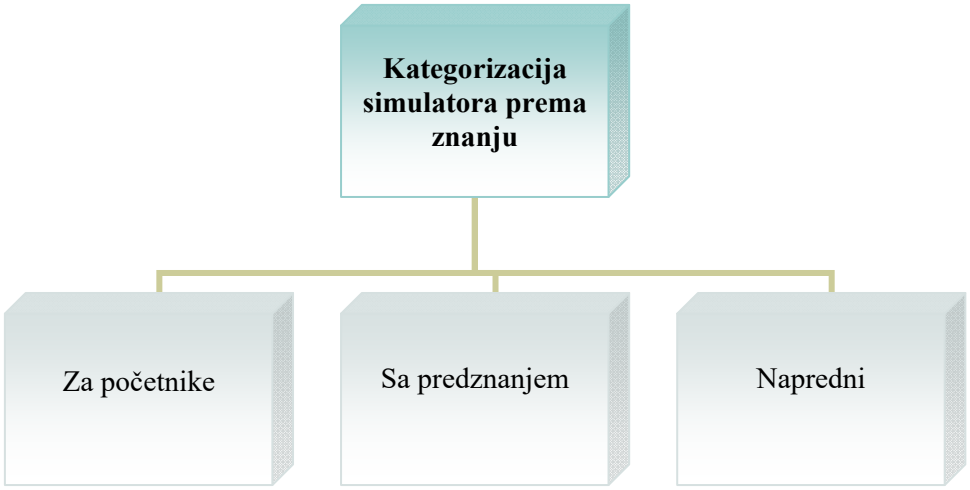
Ako bi se razmatrao određen vremenski period (npr. zadnja decenija 20-tog veka) i izvršila analiza simulatora korišćenih u edukativne svrhe onda bi se mogli istaći sledeći zanimljivi primeri [27]:

- Skrien i Hosack 1991. godine prikazuju jedan Apple-Macintosh miltilevel simulator (CPU SIM) za priručnu upotrebu u učionici obezbeđujući studentima da kreiraju različite konfiguracije hardvera za poređenje.
- Foley 1992. godine opisuje korišćenje projekta programiranja simulatora Paskala za učenje mikrokodiranja.
- Coey je 1993. godine razvio je Hypercard and Hypertalk interaktivni simulacioni tutorijal namenjen kursevima poučavanja MC68000 mirkoprocesorskog asemblerskog jezika. Utvrdili su da njihov simulator omogućava laku demonstraciju i interakciju studenata tokom časova.

- Magagnosc je 1994. godine dokumentovao svoj Apple Macintosh Micro simulator koji je kreiran 1989. godine i testiran 4 godine od strane nastavnika na osnovnim i master studijama.
- Coe i ostali su 1996. godine diskutovali upotrebu DLX i DASH arhitekture simulatora zasnovane na HASE (Hierarchical Architecture design and Simulation Environment), rasvetljavajući moć nastavne animacije u vizuelizaciji namenjenoj učenju i predstavljanju na različitim nivoima apstrakcije.
- Knox 1997. godine preporučuje korišćenje simulacionog projekta na kursevima iz arhitekture računara.
- Fuente je 1999. godine implementirao superskalarni simulator i set laboratorijskih vežbi za nastavu/poučavanje asemblerskog jezika, keš (cache) ponašanja i kanalisanje.
- Bruschi je 1999. godine opisao ASiA (Ambiente de Simulacao Automatico) sistem koji koristi modularni pristup (GUI front'end aplikacioni generator i izlazne analitičke module) koji mogu da se prošire na nova područja.
- Đorđević i ostali su 2000. godine podelili sa drugima petogodišnje iskustvo korišćenja integrisanog okruženja za učenje arhitekture računara koji se sastoji od SPIECS (Softvare Package Integrated Educational Computer System – Softverski paket integrisanog obrazovnog računarskog sistema) i CALKAS (Computer Architecture Learning and Knowledge Assessment System – Sistem za učenje arhitekture računara i procenu znanja). SPIECS grafički simulira RISC/CISC procesore i hijerarhijsku memoriju. CALKAS se sastoji od kviza – testa znanja sa zadacima višestrukog izbora o operacijama računara.
- Frank je 2000. godine modelovao računarske operacije u objektno orijentisanoj arhitekturi sa objektima (i njihovim povezanim karakteristikama i kontrolnim signalima) mapiranim da biraju komponente računara.

Kod ovih nabrojanih radova postavljeni su opšti ciljevi u dizajniranju računarskih simulatora. Ti ciljevi su: bliskost korisniku, mapiranje pomoću udžbenika, realnost, kompletnost, fleksibilnost i od svega najvažnije vizuelizacija.

Prema predznanju koje studenti poseduju u trenutku kada se sreću sa problemima arhitekture računara možemo ih podeliti na **početnike**, **korisnike sa određenim predznanjem** i **napredne korisnike** (slika 3.4).



Slika 3.4 Kategorizacija simulatora prema predznanju

3.4.2.1. Simulatori za početnike

Simulatori za početnike imaju osnovnu ulogu da omoguće realizaciju bazičnih ideja o organizaciji računara sa relativno malo detalja ili kompleksnosti.

Uobičajene karakteristike simulatora ove kategorije su: vizuelizacija suštinskih komponenti i njihovih interacija, pojednostavljeni set instrukcija za asemblerski jezik, dostupnost i lakoća korišćenja. Opšta namena ovih simulatora je da vode studente ka korišćenju srednjih i naprednih simulatora, mada studenti koji ne studiraju računarske nauke mogu imati potrebu da se zaustave na ovom nivou saznanja. Tabela 3.8 [27] daje pregled ovih simulatora sa njihovim osnovnim informacijama.

Tabela 3.8 Opis simulatora za početnike

Naziv	**Opis**
Babbage's Analytical Engine http://www.fourmilab.ch/babbage/applet.html (John Walker/Fourmilab Switzerland)	Java aplet koji emulira mehanički računar nazvan Analitička mašina koju je izgradio Čarls Bebejdž (Charles Babbage). Emulacija je na nivou mašinskog jezika.
CASLE (Compiler/Architecture Simulation for Learning and Experience) http://shay.ecn.purdue.edu/~casle/ (Purdue University)	HTML oblici i CGI skript povezani sa kompajlerom i simulatorom na nivou mašinskog jezika. Ulaz je u C jeziku koji je kompajliran u mašinski jezik. Sadržaj memorije i programerski vidljivi registri su prikazani kao programsko izvršenje.
CPU/Disk Subsystem http://www.cis.ufl.edu/~fishwick/CPUDisk/ (Paul Fishwick/U Florida)	Web-based simulacija zatvorenog mrežnog modela sa jednim serverom koji predstavlja CPU i nekoliko servera za predstavljanje diskova. Izvorni kod i MPEG video on-line.
CPU SIM http://www.cs.colby.edu/~djskrien/ (Dale Skrien/Colby College)	Kontaktirati autora za dobijanje besplatne kopije za upotrebu u učionici. Macintosh (MC680X0 ili PowerPC) platforma.

Naziv	**Opis**
CPU Simulator applet http://www.cs.gordon.edu/courses/cs111/module6/cpusim/cpusim.html (Irvin Levy/Gordon College)	Automatizovana simulacija vizuelno prikazuje signale između ključnih komponenti računara. Dostupni su i simbolički i binarni mod reprezentovanja.
EasyCPU http://www.cteh.ac.il/departments/education/cpu.html (Cecile Yehezkel/Weizman Institute of Science)	Na Windows platformi zasnovan pojednostavljeni set instrukcija Intel80X86 simulatora koji ima i bazični i napredni mod.
Little Man Computer www.acs.ilstu.edu/faculty/javila/lmc/ (Bill Yurcik and Larry Brumbaugh/Illinois State)	Simulira Little Man Computer model od MIT's (Madnick, 1965). Demonstrira izvršavanje programa na asemblerskom jeziku i obezbeđuje izvršavanje po koracima u odgovarajućem mašinskom kodu. Sadržaj memorije i registri su prikazani tokom izvršavanja.
Simple Computer http://beachstudios.com/sc/ (Beach Studios) (used at Saddleback College and Cal State Fullerton)	Simulacija zasnovana na Java skript programu. Aplet pokazuje sadržaj akumulatora, IR (instrukcioni registar), PC (programski brojač) i memoriju. Izvorni kod simulatora je veoma čitljiv i jednostavno dostupan posmatranjem izvora na web stranici.
CPU Simulator for Windows http://www.spasoft.co.uk/cpusim.html (SPA Corp.)	Na windows-u zasnovana vizuelno animirana simulacija akumulatorskog CPU na instrukcionom nivou. Pokazuje akumulator, IR, PC i memorijske adrese kao insturkcono izvršenje.

3.4.2.2. Simulatori za korisnike sa predznanjem

Ovi simulatori su namenjeni za studente koji imaju neko predznanje o arhitekturi računara i žele da koriste simulator da bi otkrivali principe na detaljnijem nivou. Ovi simulatori su pogodni za studente koji nemaju široko predznanje iz arhitekture računara, ali imaju zrelost i interesovanje da započnu sa detaljnijim simulatorima. Oni još uvek nisu spremni za simulatore koji prikazuju sva svojstva aktuelnih saznanja u istraživanjima arhitekture računara.

Ovi simulatori imaju za cilj da ilustruju i pouče dva opšta koncepta: arhitekturu instrukcionog seta i mikroarhitekturu. Simulatori arhitekture instrukcinog seta simuliraju npr. sadržaj memorije, programeru vidljive registre, instrukcioni registar (IR) i programski brojač (program counter - PC) kao i redosled kako se instrukcije izvršavaju. Simulatori na instrukcionom nivou, namenjeni studentima sa određenim predznanjem, razlikuju se od onih koji su namenjeni početnicima u tome što su procesori koje simuliraju mnogo realističniji i set instrukcija koje predstavljaju mnogo kompletniji.

Simulatori mikroarhitekturnog tipa fokusirani su na prikazivanje mehanizma u kome mikroarhitektura implementira izvršavanje svake instrukcije. U tabeli 3.9 [27] je data lista simulatora za studente sa određenim predznanjem.

Tabela 3.9 Opis simulatora namenjenih za korisnike sa određenim predznanjem

Naziv	**Opis**
Simulatori instrukcionog nivoa	
LC2 Simulator http://www.mhhe.com/patt/	LC2 je proceros opisan u udžbeniku Uvod u računarske sisteme (Patt and Patel, 2000). Ovaj simulator prati udžbenik a dostupan je i poseban priručnik on-line. Dostupne su i UNIX i Windows verzija.
SPIM http://www.cs.wisc.edu/~larus/spim.html (James Larus/U Wisconsin-Microsoft Research)	Simulator asemblerskog jezika za MIPS (R2000/R3000) procesor koji ima i jednostavni terminalni interfejs, i vizuelni interfejs zansovan na Windows-u. Dostupne su i UNIX, LINUX, MS/DOS. Široko se koristi sa udžbenikom (Pattersen and Hennessz, 2000) i ima mnogo dokumentacije on-line.
SPIMSAL www.cs.wisc.edu/~larus/spim.html (James Larus/U Wisconsin-Microsoft Research)	SPIMSAL je starija verzija od SPIM koji se pokreće pod Windows 3.1 i Macintosh-em. SPIM web stranica ima veze za SPIMSAL zipfile. SPIMSAL je simulator za proširenu verziju MIPS (R2000/R3000) asemblerskog jezika, koji uključuje memorija-memorija instrukcije i obezbeđuje uvod u MIPS koji je korišćen u (Goodman and Miller, 1993).
THRSim11 http://www.hc11.demon.nl/thrsim11/thrsim11.htm (Harry Broeders/Rijswijk Inst. of Technology)	Simulator nivoa seta instrukcija za Motorola 68HC11A8 mikrokontroler. Simulira sve registre, memoriju, pinove, i portove. Pokreće se sa Windows-a grafičkim interfejsom. THRSim11 potiče od marta 1999 iz časopisa Dr. Dobbs. Skup je, ali je dostupna i besplatna demo verzija.
Microprocessor Simulator http://www.softwareforeducation. (Software for Education)	Ovo je shareware simulator Intel 8086 instrukcionog seta. Podržava podset instrukcionog seta. Može da napiše jednostavan progam u asemblerskom jeziku. Izvršavanje ovog programa prikazuje na nivou asemblerskog jezika, registre (uključujući instrukcione pointere) i glavnu memoriju (256 bytes). Besplatan je za korišćenje u obrazovanju u učionici.
Simulatori mikroarhitekture	
Mic-1 Simulator http://www.ontko.com/mic1/ (written by Ray Ontko and Dan Stone with advice from Andrew S. Tanenbaum)	Mic-1 je smiluator zasnovan na Javi, i implelentira Mic-1 mirkoakrhitekturu opisanu u 4. poglavlju udžbenika Tanenbauma iz 1998. Postoje Unix i Windows verzije. Kompletan vodič za korisnike dostupan je online.
Micro Architecture http://www.kagi.com/fab/msim.html (Fabrizio Oddone) (used at Earlham College)	Micro-Architecture simulator modeluje mirkoprogramirani procesor sličan onom koji se koristi u (Tanenbaum, 1999). Ove hardverske komponente i instrukcioni set su fiksirani, u potpunosti je moguće

Naziv	Opis
	editovati mikroporgram na način blizak korisniku. Ova aplikacija se pokreće na svakom Apple Macintosh sa instaliranim System 7.
MIPSim http://mouse.vlsivie.tuwien.ac.at/lehre/rechnerarchitekturen/download/Simulatoren/ (Institut fur Technische Informatik)	Jednostavni simulator za kanalni procesor /kanalisanje procesora, zasnovan na radu Henesija i Petersona (Hennessy and Patterson, 1996). Dostupan MIPSim.txt readme fajl za objašnjavanje izvornih fajlova. Komentari na nemačkom jeziku.

3.4.2.3. Simulatori za napredne korisnike

Ovi simulatori su namenjeni za studente sa značajnim predznanjem u oblasti arhitekture računara. Kao grupa, ovi simulatori realistično odražavaju stanje dizajna procesora, a neki se koriste i kao alati za proučavanje arhitekture.

Dva su tipa simulatora koji se smatraju naprednim. Simulatori mikroarhitekture su mnogo sofisticiraniji nego oni koji su namenjeni za studente sa određenim predznanjem. Ovi simulatori su ugradili u sebe današnje složene procesore visoke performanse, na primer, detalje «pipelined» izvršavanja i «parallelisam» instrukcionog nivoa. Zbor složenosti, oni su mnogo pogodniji za napredni nivo kurseva (kurs arhitekture računara) nego za osnovni kurs (kurs organizacije računara). Drugi tip naprednih simulatora su logički simulatori, koji obezbeđuju virtualni elektronski radni sto (work-bench). Oni obuhvataju svojstva rangiranja od jednostavnog logičkog crtanja do projektovanja i editovanja kola, i često uključuju simulaciju vremena. Mada su logički simulatori napredni, oni su fokusirani na sadržaje nižeg nivoa: tranzistore, logičke kapije (gates), digitalna kola itd. Prikaz primera ovih simulatora dat je u tabeli 3.10 [27].

Tabela 3.10 Opis simulatora za naprednije korisnike

Naziv	Opis
Napredni simulatori mikroarhitekture	
DLXsim www.mkp.com/books_catalog/ca/hp2e_res.htm (DLX Processor of Computer Architecture book)	Pipeline simulator DLX instrukcionog seta korišćen u udžbeniku Arhitektura računara (Hennessy and Patterson, 1996). Zadržava DLX asemblerski kôd (kreiran pomoću cross- kompajlera) i simulira njegovo izvršavanje na DLX računaru.
DLXview http://yara.ecn.purdue.edu/~teamaaa/dlxview/ (Purdue)	Modifikacija i proširenje DLXsim koja daje grafički, interaktivni interfejs za pipeline simulator. Uključuje dodatne pipeline protokole i vremenske parametre za realističko modelovanje i evaluaciju izvršavanja.
RSIM	RSIM simulira mašine koje odražavaju dva aktuelna trenda:

http://www-ece.rice.edu/~rsim/ (Rice)	procesore koji koriste paralelizam instrukcionog nivoa (ILP) i pojavu multiprocesora sa podeljenom memorijom sa proizvodnim sistemima. RSIM se izvršno pokreće i modeluje state-of-the-art ILP procesore i SMPs korišćenjem veoma detaljnih simulatora. Pokreću se različite komercijalne verzije za Unix (na primer Solaris, Irix).
SimOS http://simos.stanford.edu/ (Stanford)	SimOS je okruženje simulacije mašine koje simulira CPU, keš, multiprocesorske memorijske magistrale, drajvere/diskove, ethernet, konzole, i druga sredstva. Može da se koristi i kao komercijalni operativni sistem.
SimpleScalar http://www.simplescalar.org/ (Todd Austin, Wisconsin)	SimpleScalar je set alata: kompajler, linker, simulator i vizuelizacioni alat. Postoje simulatori od onih koji su brzo funkconalni na visokom nivou, do detaljnih procesora koji podržavaju neblokirajući keš, spekulativno izvršavanje i branch prediction (predviđanje skoka). Pogodni su za modifikovanje i proširivanje (na primer korisnička platforma). Zasnovan je na Unix-u.
Logički simulatori	
LogicWorks http://www.capilano.com/LogicWorks	Alat za kreiranje, editovanje i testiranje projektovanja kola. Interaktivne operacije koje obuhvataju merenu (clocked) simulacionu kontrolu sredstava i kola. Verzije za Windows i Macintosh. Kupuje se.
Multimedia Logic Kits www.softronix.com/logic.html	Ovo je vizualni logički projektni sistem. Pogodan za projektovanje i testiranje jendostavnih kola ali ne i adresnu organizaciju računara. Postoji samo Windows verzija.
WinBreadboard http://www.yoeric.com/ (YOERIC software)	ATTL simulator koristi se kao zamena na kursevima iz digitalne elektronike. Uključuje simulator vremena. Postoje Windows i Macintosh verzija. Kupuje se.

3.4.3. Sumarno poređenje

Može se zaključiti da priroda simulatora varira zavisno do ciljne grupe korisnika. Za studente početnike, ovi simulatori su usmereni ka jednostavnoj arhitekturi insturkcionog seta, na hipotetskim mašinama. Na primer: ove mašine često imaju samo jedan akumulator umesto seta generalno namenjenih registara. Neki od ovih simulatora samo pokazuju izvršavanje instrukcija mašinskog jezika. Drugi pokazuju i verzije programa u assemblerskom jeziku i verzije prevedenog mašinskog jezika. Pošto je ilustrovana jednostavnost izvršavanja, relativna prednost ovih simulatora je direktno izvršavanje na web-u kao Java apleti ili CGI skriptovi. Svi ovi simulatori su vizuelni.

Simulatori za studente sa određenim preznanjem su značajno sofisticiraniji. U ovoj kategoriji su simulatori mikroarhitekture. Ovi tipovi simulatora ilustruju kako se izvršavanje pojedinačne instrukcije mašinskog jezika implementira preko «data path-a» i

kontrolne jedinice (mikrokôd) procesora. Međutim, ilustrovani putevi podataka su relativno jednostavni. Simulatori nivoa instrukcija u ovoj kategoriji su značajno složeniji nego simulatori za početnike, pošto ovi simulatori imaju složeniji set instrukcija. Oni simuliraju realne procesore [27].

Simulatori za napredne studente se dele na napredne simulatore mikroarhitekture i logičke simulatore. Napredni simulatori mikroarhitekture ilustruju putanju podataka i kontrolne jedinice modernih procesora visokih performansi. Zbog toga, oni odražavaju takve karakteristike kakve su kanalisanje (pipeling) i paralelizan nivoa instrukcija. Logički simulatori obezbeđuju sredstva za ilustrovanje kako se primenjuje mikroakrhitektura pomoću logičkih kola. Stoga su oni uvod za sledeći nivo hardvera računara.

4

4. SVETSKA ISKUSTVA KORIŠĆENJA SIMULACIJA U ARS OBRAZOVANJU

Arhitektura i organizacija računara (ARS) je jedna od najvažnijih oblasti računarskih nauka i u školovanju studenata kurs iz ove oblasti zauzima visoko mesto. Zato je veoma važno pitanje na koji način organizovati takav jedan kurs.

Arhitektura i organizacija računara je sadržaj težak za učenje i nastavu:

- Prvo, potrebno je ovladati širokim opsegom veština i detalja – od primene tranzistora kao logičkog gate-a do arhitekture računarskog sistema
- Drugo, arhitektura aktuelnih računarskih sistema sadrži karakteristike kakve su:
    - jedinstvenost izrade,
    - složenost za razumevanje, i
    - kriptička dokumentovanost.

  Ovo određuje da laboratorijski eksperimenti za studente predstavljaju izazov koji zahteva široku pripremu a rezultat toga mogu da budu veštine i znanja koja nisu prenosiva na druge mašine.
- Treće, arhitektura računara je za većinu studenata – dok studenti mogu imati visok nivo računarske pismenosti o softverskim aplikacijama, jezicima, i periferijama, mnogi imaju nekompletne, nerealne i ponekad nezasnovane pojmove o tome kako računar radi. Stoga većina studenata su otvoreni da otkrivaju računar
- Poslednje, studenti su preplavljeni sa marketinškim informacijama o računarima iz reklama na televiziji, radiju i časopisima. Dok studenti nisu informisani korisnici, razumevanje arhitekture računara ima dva efekta:
    - nedostatak informacija o činjenicama o arhitekturi računara i
    - studenti usvajaju pogrešne ideje

Da bi se ostvarili odgovarajući rezultati trebalo bi zadovoljiti određene kriterijume. U okviru kursa studente bi trebalo upoznati sa osnovnim konceptima arhitektura računarskih sistema, kao i izvršavanjem osnovnih operacija. Tokom predavanja, trebalo bi, gde god je to moguće, uspostavljati vezu sa drugim oblastima računarske nauke. Tako, pri razmatranju registarskog indirektnog adresiranja trebalo bi pomenuti i koncept pokazivača kod viših programskih jezika, na primer kod C programskog jezika.

Tokom kursa predavači bi morali da ukažu na osnovne principe koji se moraju poštovati pri današnjem projektovanju složenijih sistema. Na osnovu ovih principa studenti bi trebalo da nauče da projektuju računarske sisteme. Najbolji način da steknu takvo iskustvo je da neprekidno tokom kursa, paralelno sa predavanjima i vežbama na tabli, koriste laboratoriju i specijalne simulatore.

Za svakog predavača idealno rešenje bi bilo kada bi postojao jedinstveni softverski sistem koji bi mogao da se koristi kao podrška predavanjima tokom celog kursa iz arhitekture i organizacije računara.

Vrednost nastave arhitekture računara i programiranja u assemblerskom jeziku je stara debata koja se nastavlja kao spoljašnji pritisak na obrazovne institucije da pokrenu veštine zasnovane na visokom nivou apstraktnosti. Konkretno i striktno pedagoško iskustvo na nižem nivou neophodno kao osnova izgrađivanja apstrakcija višeg nivoa. Dr C. Revišenker (Ravishanker), profesor, je rekao "Skrivanje informacija je veoma dobro, ali student treba da ima neke informacije pre nego što može da započne da ih skriva".

Da bi se analizirala postojeća rešenja, prvo, ćemo razmotriti teme koje se obrađuju tokom kursa iz arhitekture i organizacije računara, a potom ćemo dati opis nekih simulatora koji se koriste u nastavi iz ove oblasti.

4.1. KAKO PRISTUPITI PREDAVANJU ARS-A

Jedan od načina organizovanja kursa iz arhitekture i organizacije računara je definisan od strane IEEE Computer Society i ACM Computer Engineering Task Force. Sledi kratak opis njihovog predloga.

4.1.1. Osnovni principi

Studenti bi se prvo upoznali sa osnovama arhitekture i organizacije računara. U okviru ovog dela kursa bi se razmatrali registri i fajl registri, tipovi podataka, kao i različiti pristupi projektanata, što se može uočiti sa različitim načinima adresiranja, različitim skupovima insrtukcija, varijabilni ili fiksni formati instrukcija, itd.

Zatim bi se objašnjavali postupci dohvatanja instrukcije iz memorije i faze dekodovanja i izvršavanja instrukcije. U okviru ovih tema razmatraju se i organizacije osnovnih računarskih sistema, kao i njihove osnovne funkcionalne jedinice. Studentima se objašnjava veza između mašinskog formata instrukcije na binarnom nivou i prezentacije iste te instrukcije u asembleru posmatranog računara. Daje im se mogućnost da pišu manje programe i delove asemblerskog kôda da bi bolje shvatili operacije na mašinskom nivou. Takođe, na ovaj način mogu da implementiraju neke osnovne principe viših programskih jezika, na mašinski nivo [25].

Na kraju ovog dela kursa studenti se upoznaju sa različitim ulazno-izlaznim tehnikama i pojmovima vezanim za prekide.

4.1.2. Računarska aritmetika

Sledeći deo kursa bi razmatrao računarsku aritmetiku. Ovaj deo kursa bi doprineo da studenti razumeju kako se numeričke vrednosti prezentuju u digitalnim sistemima, na taj način što bi se upoznali sa načinima prezentacije celobrojnih veličina, celobrojnih veličina sa znakom i bez znaka, i prezentacije realnih brojeva. Nakon razmatranja načina prezentacije, proučavali bi se osnovni algoritmi za operacije sa celobrojnim veličinama, kao što su sabiranje, oduzimanje, množenje i deljenje, i osnovni algoritmi za operacije sa realnim brojevima. Razmatrali bi se načini konverzije između celobrojnih veličina i realnih brojeva. Studentima bi se objasnila i ograničenja računarske aritmetike i efekti grešaka pri različitim operacijama.

Na kraju ovog dela kursa razmatra se povezanost dizajna jedinice za aritmetiku i perfomansi pri računanju različitih izraza.

4.1.3. Glavna memorija

Nakon ovih osnovnih pojmova počelo bi proučavanje naprednijih tehnika. Prvo se razmatraju pitanja vezana za glavnu memoriju. Prva tema je hijerahijski memorijski sistem, gde se objašnjavaju tehnike za smanjenje memorijskih kašnjenja. Zatim se objašnjavaju principi upravljanja memorijom. Sledeća tema je organizacija memorije, gde se studenti upoznaju sa najvažnijim memorijskim tehnologijama. Nakon toga se razmatraju osnovni pojmovi vezani za memoriju, kao što su kašnjenje, širina memorijske reči, vreme ciklusa, performanse, itd. Sledeći deo kursa bi obuhvatio složenije načine realizacije memorije kao što su virtuelna memorija, keš memorije i preklapanje memorijskih modula.

Studenti bi se upoznali i sa postojećim tehnologijama izrade memorija (SRAM, DRAM, EPROM, Flash).

Na kraju ovog dela kursa bi se razmatrali pitanje grešaka kod memorija, gde bi se studenti upoznali na koji način nastaju greške u memorijskim sistemima i kako mogu ovi problemi da se reše.

4.1.4. Ulaz-izlaz i komunikacija

Sledeća oblast koja se razmatra je način realizacije i osnovni pojmovi vezani za ulazno-izlazni podsistem i komunikaciju između modula računarskog sistema. Studentima se prvo objašnjava način na koji se prekidi koriste za realizaciju ulazno-izlazne kontrole i transfera podataka. U ovom delu kursa se upoznaju sa pojmovima handshake-a[6] i baferovanja[7]. Nakon toga se proučavaju osnovne ulazno-izlazne tehnike: programirani ulaz-izlaz, ulaz-izlaz pomoću mehanizma prekida i ulaz-izlaz pomoću DMA kontrolera. Studenti imaju mogućnost da pišu manje prekidne rutine i kôd za inicijalizaciju periferijskih uređaja pomoću definisanog asemblera. Razmatra se vektorski mehanizam prekida i prekidi sa prioritetima. U okviru razmatranja načina komunikacije između modula računarskog sistema proučavaju se različite vrste magistrale, kao i koncepti koji se primenjuju pri arbitraciji pristupa magistrali.

4.1.5. Organizacija procesora

Poslednja oblast koja se razmatra je organizacija procesora. Ona obuhvata upoređivanje različitih alternativa implementacije procesora. Razmatraju se prednosti i mane korišćenja jedne ili više magistrala. Studenti se zatim upoznaju sa pojmom pipeline (kanalisanje) i sistema koji koriste pipeline organizaciju. Objašnjavaju se osnove ILP (Instruction Level Parallelism) pomoću pipeline-a i najčešći hazardi koji se mogu pojaviti. Razmatraju se efekti skokova kod ovakvih sistema. Diskutuje se na koji način skup instrukcija može da utiče na perfomanse sistema, na primer predikcija izvršavanja instrukcije. U nastavku kursa se proučavaju različiti načini realizacije upravljačke jedinice: hardverska realizacija i mikroprogramska realizacija. Razmatra se generisanje upravljačkih signala pomoću ove dve implementacije.

Na kraju kursa se proučava implementacija aritmetičkih jedinica.

[6] Handshake (rukovanje) – procedura kojom se reguliše način razmena podataka i tajming, uobičajeno se koristi kod paralelnog interfejsa.

[7] Bafer (Buffer) – obično predstavlja deo računarske memorije, izdvojen da bi se u njega beležili podaci potrebni za međukorake pri izvođenju određenih radnji.

Pregled navedenih tema naveden je tabelarno u tabeli 4.1 [25].

Tabela 4.1 Osnovne teme iz oblasti ARS-a

Osnovni principi	**Računarska aritmetika**	**Glavna memorija**	**Ulaz-izlaz i komunikacija**	**Organizacija CPU**
▪ Registri ▪ Tipovi podataka ▪ Tipovi instrukcija ▪ Načini adresiranja ▪ Formati instrukcija ▪ Dohvatanje, dekodovanje i izvršavanje instrukcije ▪ Tehnike ulaz-izlaza i mehanizam prekida	▪ Prezentacija celobrojnih veličina - sa i bez znaka ▪ Osnovni aritmetički algoritmi za celobrojne veličine: o sabiranje, o oduzimanje, o množenje i o deljenje ▪ Prezentacija realnih brojeva ▪ Osnovni aritmetički algoritmi za realne brojeve ▪ Konverzija između realnih i celobrojnih veličina	▪ Hijerarhijski memorijski sistemi ▪ Organizacija glavne memorije ▪ Kašnjenje i perfomanse ▪ Virtuelne memorije ▪ Keš memorije ▪ Preklapanje memorijskih modula ▪ Tehnologije izrade memorija (SRAM, DRAM, EPROM, Flash) ▪ Otkrivanje i oporavak od grešaka	▪ Osnove ulaza/izlaza: handschaking, baferovanje ▪ Tehnike ulaza-izlaza: programirani ulazi-zlaz, zasnovan na mehanizmu prekida, pomoću DMA ▪ Mehanizam prekida: vektorisani i sa prioritetom ▪ Magistrale: ciklusi, kontrolne, adresne i magistrale podataka, arbitracija	▪ Analiza sistema sa jednom i više magistrala ▪ Sistemi sa i bez pipeline-a ▪ Upravljačka jedinica: ožičena i mikroprogramska realizacija ▪ Implementacija aritmetičke jedinice

Pored teorijskog predznanja student koga zanimaju računarske nauke treba da stekne i praktično znanje iz navedenih tema. Iz tog razloga kursevi iz oblasti arhitekture i organizacije računara treba da obuhvate i laboratorijske vežbe.

Kao i bilo koji drugi inženjerski kurs, veoma je je važno da studenti pohađajući ove kurseve imaju mogućnosti da proučavaju i istražuju karakteristike i ponašanje različitih uređaja, sistema i procesa. Potrebno je da projektuju, implementiraju i testiraju hardverske i softverske komponente, stvaraju eksperimente i primere za analiziranje projektovanih sistema, i u nekim slučajevima, u zavisnosti od dobijenih rezultata da imaju mogućnost da menjaju sam dizajn. Sve ove operacije mnogo je efikasnije izvoditi u okviru laboratorijskih vežbi, nego kao integralni deo predavanja ili u obliku odvojenog kursa.

Uvodne laboratorijske vežbe treba da studente direktno upute na tehnike projektovanja različitih sistema, objašnjene u okviru predavanja i vežbi na tabli. Ove vežbe služe za demonstraciju karakteristika i ponašanja komponenti sistema, kao i analiziranje osobina manjih sistema. Laboratorijske vežbe na srednjem i krajnjem stepenu treba da

uključe složenije tehnike i koncepte. U okviru ovih vežbi treba da se proučavaju složeni sistemi, njihove komponente i međusobna povezanost ovih komponenti.

Idealan simulator treba da ima mogućnost izvršavanja praktičnih primera za veoma širok opseg različitih tema. Studentima treba omogućiti rad sa alatima za pisanje sopstvenih asemblerskih programa i mogućnost vizuelne simulacije izvršavanja napisanog programa na više hijerarhijskih nivoa. Edukacioni simulatori treba da budu jednostavni za korišćenje, jer studenti koji ih koriste imaju različito predznanje korišćenja računara. Ovi simulatori treba da olakšaju predavačima stvaranje konkretnih primera i eksperimenata za različite predmete iz ove oblasti. Svi ovi zahtevi mogu da budu i kontradiktorni, čime je razvoj simulatora za potrebe celokupne oblasti arhitekture i organizacije računara veoma otežan

4.2. SIMULATORI IZ ARS-A

U literaturi može se pronaći veći broj simulatora računarskih sistema, koji se mogu koristiti kao podrška kursevima iz oblasti arhitekture i organizacije računara. Analizom dostupnih radova dolazi se do zaključka da su postojeći simulatori projektovani u različite svrhe i za različite kurseve.

4.2.1. SPECS - Obrazovno okruženje u nastavi kursa Arhitektura i organizacija računara (AOR) – ETF BEOGRAD[8]

Na ETF u Beogradu razvijeno je obrazovno okruženje za nastavu iz predmeta Arhitektura i organizacija računara. Okruženje obuhvata obrazovni računarski sistem, priručnik, softverski paket i niz/skup laboratorijskih eksperimenata. Obrazovni računarski sistem je osmišljen da obuhvati osnovnu strukturu računarskog sistema. Referentni priručnik je detaljan. Softverski paket uključuje programske razvojne alate i grafički simulator. Niz laboratorijskih eksperimenata demonstrira funkcionisanje obrazovnog računarskog sistema.

Kurs Arhitektura i organizacija računara je predmet u II godini osnovnih studija na ETF, Univerziteta u Beogradu, namenjen studentima Odseka za komunikacije, automatizaciju i elektroniku i Odseka za računarske nauke. Kurs obuhvata osnovne pojmove iz arhitekture i organizacije računara pod pretpostavkom da studenti Odeljenja za komunikacije, automatizaciju i elektroniku primarno koriste relativno jednostavne konfiguracije računarskih sistema kakvi su kontroleri u komunikacionim delovima opreme,

[8] Jovan Đorđević, Nenad Grbanović, Boško Nikolić, Elektrotehnički fakultet u Beogradu

automatizovani kontrolni sistemi i druga računarska kontrolna elektronska sredstva. Za studente odseka za računarske nauke ovo je upravo uvodni kurs u oblast AOR [23].

Kurs AOR obuhvata procesorsku arhitekturu i organizaciju, memoriju, input/output podsisteme i magistralu (bus). Procesorska arhitektura se razmatra kroz proučavanje programski kontrolisanih registara, tipova podataka, formata instrukcija, adresnih modova, instrukcionih sistema i mehanizama za prekid. Proučavanje procesorske organizacije je skoncentrisano/usmereno na različite implementacije procesnih i kontrolnih jedinica. Dizajn memorije se razmatra preko problema organizacije memorijskih modula. Input/output sistemi deluju sa I/O programskim tehnikama i strukturama perifernih sredstava (izlaza) i direktnom memorijskim procenjivačkim kontrolerima. Razmatranja bus-ova obuhvataju bus arbitražu i transfere podataka od asinhronih na sinhrone bus-ove.

Veliki problem u nastavi iz predmeta AOR je naći načine kako pomoći studentima da naprave kognitivni prelaz od opisa arhitekture i organizacije računara na tabli do korišćenja toga kao programibilnog sredstva. Ovaj problem je proučavan, i prema prikazima u literaturi, obično je rešavan davanjem određenih računarskih simulacija. Međutim, pošto nijedno od ovih rešenja nije neposredno primenjeno na kursu AOR, autori su odlučili da ih primene i razviju sopstveno obrazovno okruženje koje obuhvata ECS (Educational Computer System) i referentni priručnik (uputstvo) za to, softverski paket za ECS (SPECS) i niz laboratorijskih eksperimenata.

4.2.1.1. Softverski paket za ECS

ECS se sastoji od procesora, memorije, input/output podsistema i sistemske magistrale.

Arhitektura procesora je load/store tipa. Arhitektura procesora od programski distupnih registara ima 16 registara opšte namene. Tipovi podataka podržavaju 16-bitne označene i neoznačene cele brojeve. Instrukcioni format je promenljiv. Načini adresiranja uključuju memorijsko direktno, registarsko indirektno, registarsko indirektno sa pomerajem i registarsko direktno adresiranje. Instrukcioni set obuhvata instrukcije prenosa, aritmetičke instrukcije, logičke instrukcije, instrukcije pomeranja (shift) i rotiranja, instrukcije skoka i kontrolne instrukcije. Podržana je tehnika vektorisanog mehanizma prekida.

Organizacija procesora je tako odabrana da je čine dve odvojene jedinice za upravljanje i kontrolu. Jedinica za upravljanje (operaciona jedinica) ima registar fajl (RFU), izvršavanje operacija (EXU), servis prekida (ISU) i bus interfejs (BIU) i oni su međusobno povezani sa 16-bitnim internim magistralom. Za prikazanu operacionu jedinicu

procesora date su četiri moguće realizacije upravljačke jedinice. Jedna od njih koristi hardverske tehnike, dok preostale tri koriste mikroprogramske tehnike sa mešovitim formatom mikroinstrukcija i mikroprogramska realizacija sa vertikalnim formatima mikroinstrukcija i nanoprogramiranjem [23].

Memorija ECS ima kapacitet od 64 kilo-reči, dužina memorijske reči je 16-bitna i procesorski generisane adrese su za 16-bitne reči.

Input/output podsistem je napravljen od tri I/O jedinice i jednog DMA kontrolera. Svaka I/O jedinica je napravljena od perifernih uređaja, kontrolera i samih periferija. Periferni uređaji i kontroleri imaju registre kontrole, statusa, podataka i vektorski registar prekida podržavaju prenos između periferalnih sredstava i memorije i obratno. DMA kontroler je jedna od I/O jedinica. Sastoji se od adrese kontrole, statusa , podataka i izvora, ciljne adrese (adresa destinacije), blok brojčanika i vektorskih registara prekida i podrške, ne samo transfera između perifernih sredstava i memorije, već, takođe i između memorije ka transferu memorije. Prenosi mogu biti u cikličnim skrivenim (stealing) ili raspršenim (burst) modovima. IO adresni prostor je memorijski mapiran.

Asinhroni bus je napravljen od 16 adresnih linija, 16 linija podataka i RDBUS (read - čita), WRBUS (write – piše) i FCBUS (function completed – funkcionalno kompletne) kontrolnih linija.

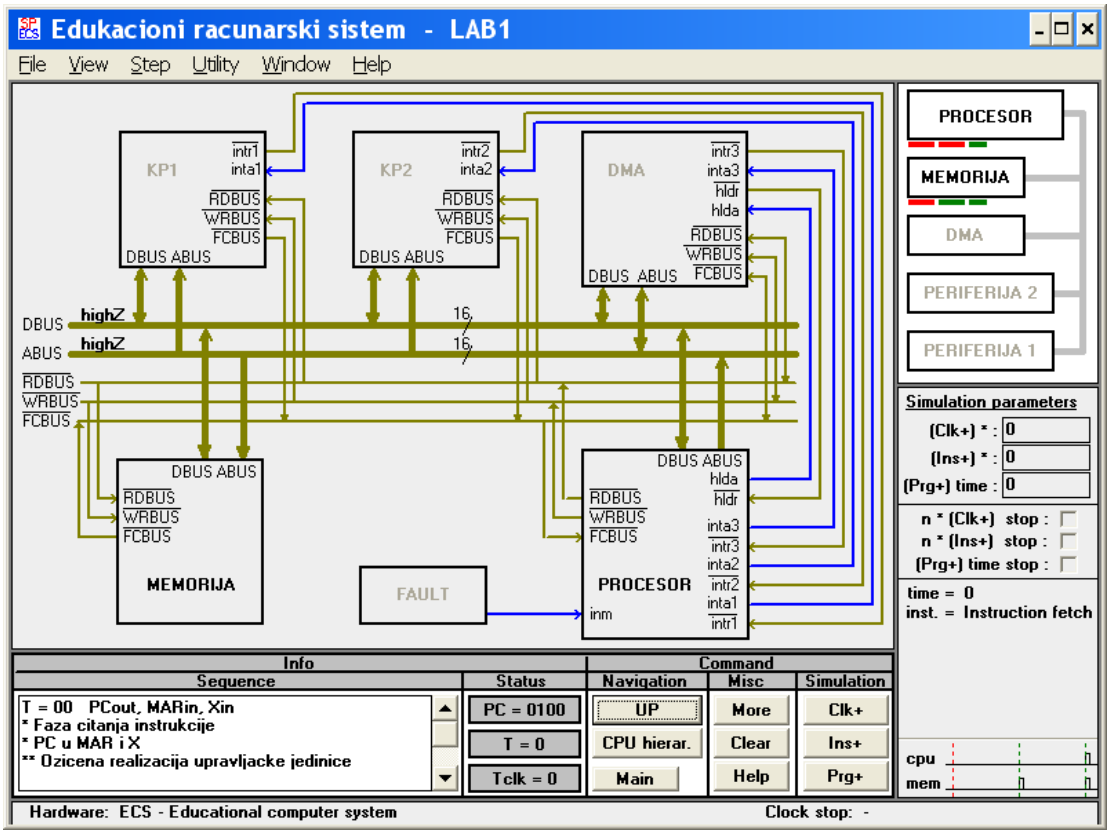
Organizacija laboratorijskih eksperimenata do sada je podrazumevala da studenti prvo uče posebne delove o ECS i onda pristupaju laboratorijskom eksperimentu. Takav pristup zahteva zavoj nekoliko vrsta simulacija ECS. Mnogo pogodniji izbor je korišćenje hardverskog deksriptorskog jezika ISP i paketa ENDOT. Stoga, simulator ECS'a je razvijen i startovan da bi se koristio za laboratorijske eksperimente.

Simulator i priručnik ECS-a [24], [6], [8] korišćeni su kao pomoć u nastavi o radu računarskog sistema i za ispunjavanje početnih ciljeva projekta. Međutim, tekstualna umesto grafičke prezentacije signala i mogućnost za interaktivnim pisanjem, modifikovanjem, transliranjem i izvršenjem programa su nedostatak. Stoga je odlučeno da se započne sa razvojem novog grafički orijentisanog simulatora i prilagodi okruženje za ECS, nazvano SPECS. SPECS je razvijen korišćenjem MS Visual BASIC 3.0. Radi pod Windows95 ili WinNT operativnim sistemima sa minimalnom zahtevanom konfirguracijom od PC 486 sa 8MB RAM memorije. SPECS omogućava višestrane olakšice koje se mogu grupisati u one koje se koriste za inicijalizaciju ECS simulatora i one koje se koriste za rad simulatora.

Prvi korak u inicijalizaciji ECS simulatora je selekcija kontrolne jedinice. Sledeći korak je loading procesorske, memorijske i IO jedinice sa inicijalnim vrednostima. Ovo može da deluje bilo interaktivno, bilo pozivanjem fajla. U prvom slučaju korisnik može da napiše sopstveni program na simbolički način, prevede ga, linkuje i load-uje. Takođe postoji mogućnost da piše/čita u i iz memorijskih lokacija, memorijskih registara i IO jedinica. Aktuelno stanje simulatora može da bude sačuvano u fajlu. U drugim slučaju, korisnik može da selektuje jedan od primera za proveru ili pozove jedan od fajlova sa kojima je prethodno sačuvane stanje simulatora.

Sistem za simulaciju, prethodno inicijalizovanog edukacionog računarskog sistema, dobija se pritiskom na taster Simulation u uvodnom ekranu softverskog paketa SPECS. Prvim ulaskom u sistem za simulaciju dobija se ekran sa sadržajem kao na slici 4.1 [24].

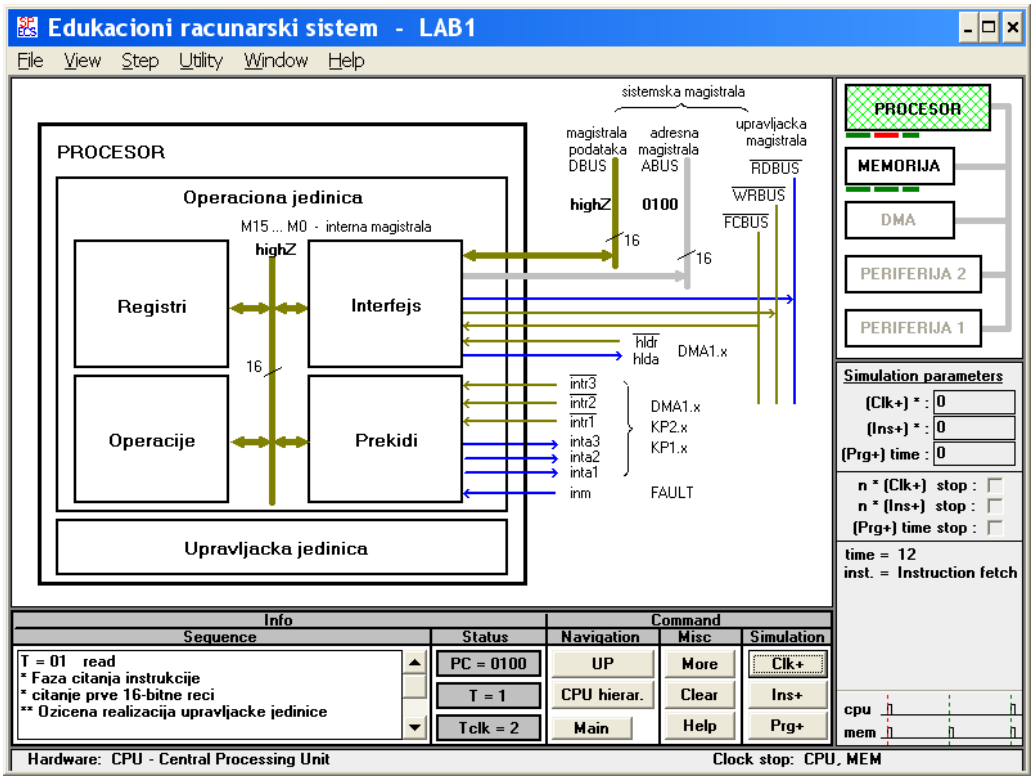
Tokom rada simulatora on može da prati vrednosti ECS signala na različitim nivoima. Organizacija celog ekrana kada je aktivan sistem za simulaciju uvek je ista i saglasna je skupu navedenih funkcija ovog sistema. Svaki prozor (ekran) je sačinjen od većeg prozora nazvanog Block-diagram, i manjih prozora.



Slika 4.1 Edukacioni računarski sistem

Osnovne oblasti na koje je podeljen ekran simulacije mogu se ovako grupisati (slika 4.1):

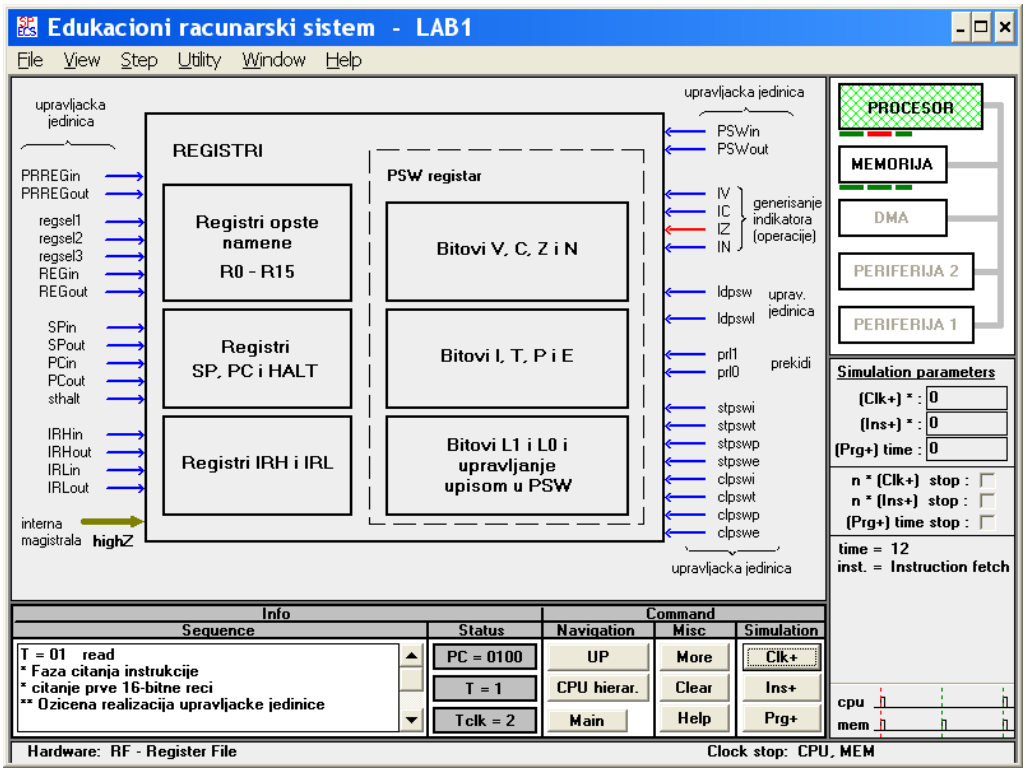
- prozor za prikaz hardvera, odnosno sekvencijalnih i kombinacionih mreža posmatranog računarskog sistema, kao i navigacija među njima (*Block-diagram*, levo-gore)
- prozor za pristupa komandama za upravljanje radom simulatora, za prikaz osnovnih informacija o toku i rezultatima rada simulatora, kao i za poziv dodatnih ekrana za detaljniji pregled rezultata rada simulatora (*Info i Command*, levo-dole)
- prozor za prikaz topologije posmatranog računarskog sistema, za navigaciju između modula sistema i simbolički prikaz trenutno vidljivog modula na prozoru za prikaz hardvera (gore-desno)
- višenamenski prozor za navigaciju između ekrana za prikaz hardvera unutar procesora, odnosno za postavljanje parametara simulacije i prikaz trenutnog, aktivnog, ali i nekoliko narednig signala takta (*Simulation parametars*..., desno-dole)
- prozor za prikaz statusnih informacija o simulaciji i trenutno vidljivom ekranu softverskog paketa SPECS (dole) i
- glavni meni



Slika 4.2 Prikaz strukturne šeme procesora

Ako pravougaona oblast u Block-diagram-u predstavlja deo hardvera koji nije sekvencijalna ili kombinaciona mreža, može da se pozove prikaz šeme tog hardvera. Na slici 4.2 data je strukturna šema procesora, koja se dobija kada se klikne pokazivačem, na

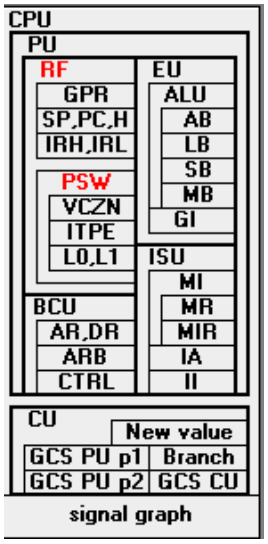
slici 4.1, na pravougaoni objekat *PROCESOR*. Na opisani način može da se ulazi u hijerarhijski niži nivo prikaza hardvera. Na slici 4.3 data je strukturna šema skupa registara koja se dobija kada se klikne pokazivačem, na slici 4.2, na pravougaonik *Registri*.



Slika 4.3 Prikaz strukturne šema skupa registara

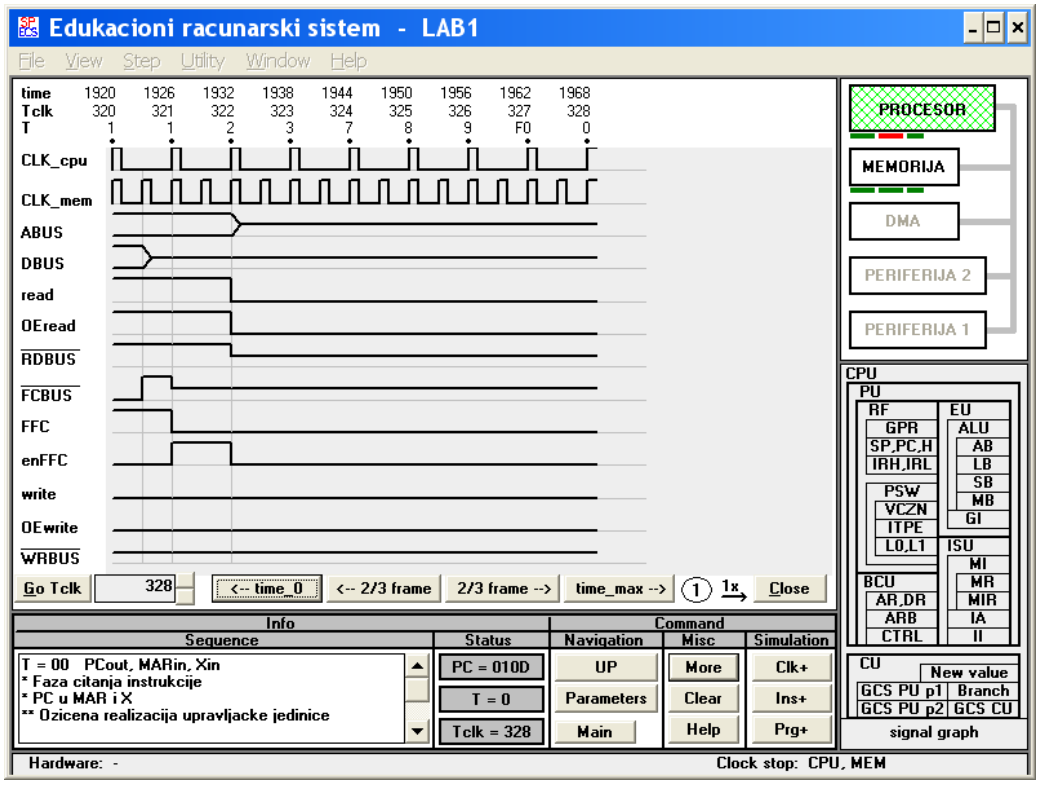
Status dugme **PC** i **Tclk** u *Info* prozoru, prikazuju aktuelne vrednosti programskog brojača i broj izvršenih taktova. *Info* prozor *Sequence* daje ili vrednost mikroprogramskog brojača mPC ili vrednost step brojača. *Command* prozor se sastoji od *Navigation*, *Simulation* i *Miscellaneous* komandih dugmadi.

Navigationo dugme **UP** dozvoljava da se pokrene jedan od nivoa screena, **CPU hierarchy** omogućava da se u višenamenskom prozoru za navigaciju (dole-desno) prikaže CPU hierarchy screen (slika 4.4). Od CPU hierarchy screen-a može da se ide direktno na neki drugi screen niže u hijerarhiji. Komandna dugmad za simulaciju **Clk+**, **Ins+** i **Prg+** dozvoljavaju nastavak simulacije jedan takt ili onaj broj taktova potreban za aktuelnu instrukciju ili program. Miscellaneous komandna dugmad su **More**, **Clear** i **Help**. Dugme More otvara prozor koji dozvoljava prikaz i postavljanje vrednosti memorijske lokacije, procesorske registra i crtanje vremenskog dijagrama selektovanih signala (slika 4.5). Dugme Clear čisti aktuelno stanje simulacije i vraća na početak. Dugme Help aktivira help sistem u kome su dostupni svi detalji o funkcionisanju ECS i SPECS.



Slika 4.4 CPU hierarchy screen

Takva organizacija simulatora omogućava simulaciju ECS na različitim hijerarhijskim nivoima. Na višim nivoima, mogu se pratiti signali razmenjivani između blokova, dok na nižim nivoima, signali u registrima, flip-flopovi, itd. Grupe linija su date sa tankim sivim linijama i njihovim heksadecimalnim vrednostima. Za linije koje predstavljaju zaseban provodnik usvojeno je sledeće obeležavanje: plave linije - logička nula, crvene linije - logička jedinica, žute linije – stanje visoke impedanse.



Slika 4.5 Pregled vremenskog oblika signala

4.2.1.2. Organizacija laboratorijskih eksperimenata

Praktičan rad u laboratoriji je organizovan u 5 eksperimenata. Od svakog studenta se traži da radi sa ECS izvršavajući primere programa i odgovarajući na pitanja o nekim tipičnim situacijama. U poslednja dva laboratorijska eksperimenta, svakom studentu je dat problem koji treba da reši, napiše deo programa i testira ga. U pojedinim slučajevima studenti su, takođe, upitani da nacrtaju vremenske dijagrame nekih interesantnih signala i uporede ih sa onim dobijenim sa alatima sa prikazivanje signala u SPECS (SPECS Show signals facility) [6], [8] .

- Eksperiment izvršavanja instrukcija demonstrira kako se izvode postavljanje instrukcije, izračunavanje operand adrese i operaciono izvršavanje faza za neke tipične instrukcije i adresne modove. Da bi se ovo postiglo, od studenata se zahteva da pokrenu, takt po takt, 4 puta svaki put sa istom procesnom jedinicom i sa jednim od četiri tipa kontrolnih jedinica.
- Eksperiment bus arbitracije i prenosa podataka, demonstrira arbitraciju između procesora i DMA kontrolera, i čitanje memorije, pisanje i prekid vektorskog broja ciklusa akvizicije.
- Eksperiment mehanizma prekida pokazuje tipične razmatrane situacije prekida. Baratanje prekidom (interrupt) i prekid i vraćanje od prekidnih instrukcija demonstrirani su na nivou takta. Selektivno i potpuno maskiranje zahteva za prekidom, servisiranje višestrukih zahteva za prekidom su demonstrirani na instrukcionom nivou.
- Eksperiment programski kontrolisanog input-a/output-a demonstrira strukturu perifernih kontrolerskih uređaja, njihovu inicijalizaciju i prenos podataka sa programski kontrolisanim input/output korišćenjem i pooling i prekidnih mehanizama.
- Eksperiment input-a/output-a sa DMA kontrolerom demostrira strukturu DMA kontrolera, njegovu inicijalizaciju i prenos podataka korišćenjem DMA kontrolera i u stealing (skrivenom) modu operacija i u burst modu operacije.

U ovom poglavlju je izloženo obrazovno okruženje koje se koristi desetak godina na ETF u Beogradu za laboratorijske eksperimente iz AOR. Uspešno kompletiranje laboratorijskih eksperimenta ja preduslov za ispit. Na osnovu brojnih diskusija sa studentima i rezultata sa ispita, autori su uvereni da ovakva obrazovna sredina može biti snažna pomoć u nastavi AOR. Ovi rezultati u razvoju jednostavnog okruženja mogu se proširiti i biti korisni i za druge kurseve u oblasti AOR.

Simulator SPECS je dostupan na adresi (verzija 26. 03. 2007.) - http://rti.etf.bg.ac.yu/rti/ef2ar/labvezbe/index.html [24].

4.2.2. HASE - Primena simulacije računarske arhitekture u nastavi – UNIVERZITET U EDINBURGU[9]

Vizuelno predstavljanje aktivnosti koje se dešavaju u računaru je važan aspekt u podučavanju računarske aritekture. Na Univerzitetu u Edinburgu koristi se Hierarchical Computer Architecture i Simulation Environment (HASE) da bi se napravilo nekoliko arhitektonskih modela za primenu u istraživanju i nastavi. Ovaj simulator postoji u različitim formama od 1992. godine [32]. Glavni cilj mu je da obezbedi onima koje interesuje arhitektura računarskih sistema skup alata koji dozvoljavaju brz razvoj i istraživanje dizajna sistema. HASE omogućava dizajneru da se kreće između nivoa i da bira najprikladniji nivo simulacije za različite delova sistema. Nova osobina u okviru HASE-a, Java HASE, dozvoljava da se modeli prevode u aplete (applet[10]) kojima se može pristupiti putem www. Java HASE apleti su modeli simulacije koji se mogu programirati i u okviru kojih se sadržaji «code and data» memorije mogu promeniti, simulacija se može restartovati u apletu, a rezultati se mogu koristiti da se vizualno predstave aktivnosti koje se dešavaju u modelu (pomeranja podataka, promene stanja, promene u sadržaju memorije, itd.). Ovi apleti se koriste u okviru više nastavnih metoda.

Trenutna verzija ima mogućnosti za prikazivanje simulacionog modela hijerarhijski strukturiranog i za vizuelno prikazivanje ponašanja modela tokom animacije preko prozora simulatora (slika 4.6). Simulator omogućava prikazivanje raznih nivoa apstrakcije arhitekture ili problema. Ostale mogućnosti uključuju snabdevanje sa simulacionim šablonima i komponentama koje predstavljaju idealan mehanizam za brz razvoj prototipa i proučavanje skalabilnih[11] arhitektura. Postoje kolekcije procesor, memorija i mrežnih entiteta, koji imaju ime, ikonu (u obliku bitmape), tekstualni opis, listu parametara, listu ulaza i pointera na programski kôd.

HASE koristi svoju sopstvenu simulacionu biblioteku diskretnih događaja (pod imenom HASE ++). U HASE ++ entiteti se izvršavaju u odvojenim nitima (threads) i svi su sinhronizovani u centralnoj klasi, gde se i čuva red budućih događaja. Koristi se standardni simulacioni algoritam diskretnih događaja (pokupi prvi sledeći događaj u redu budućih događaja, omogući povezane entitete, čekaj dok se ne završi, ponovi korake dok još postoji događaja u redu).

[9] Roland Ibbet and Frederic Mallet, Institute for Computing Systems Architecture, University of Edinburgh

[10] Apleti su mali programi pisani Java jezikom koji se mogu ugnezditi u HTML stranicu i aktivirati veb čitačem

[11] Skalabilnost – sposobnost da proces može da prihvati povećan obim

Implementacija HASE podrazumeva da je redosled kojim će entiteti završavati svoje aktivnosti nedeterministički, ali redosled kojim se događaji menjaju između entiteta moraju biti deterministički.

Postoji pet metoda rada:

- dizajn,
- validacija modela,
- pravljenje simulacije,
- simulacija sistema i
- eksperiment.

Operacije moguće u **dizajn modu** su:

- dodavanje i brisanje komponenata,
- podešavanje veza između komponenata i
- dodavanje i brisanje parametara i portova na pojedinačnim komponentama.

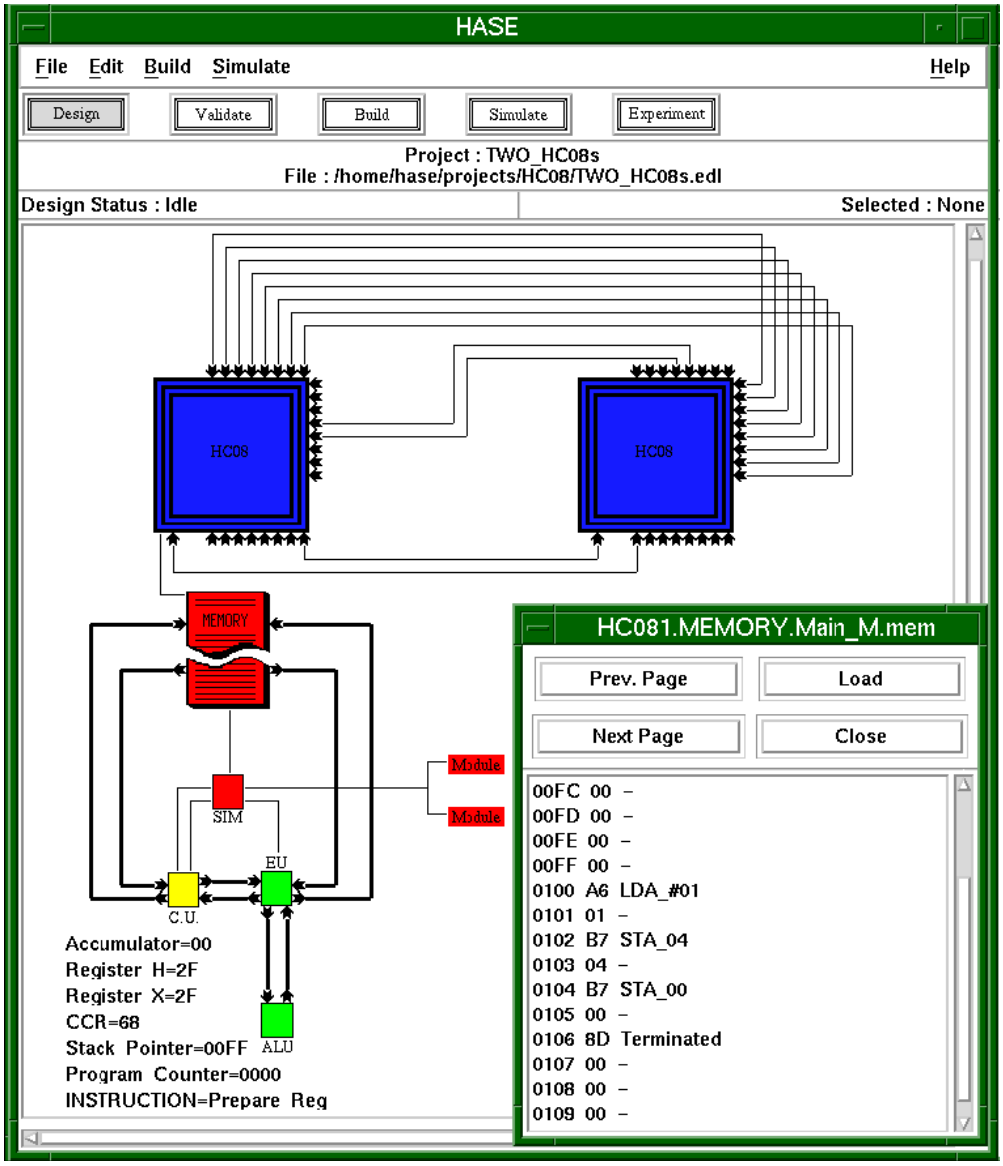
Validacija modela dozvoljava provere korektnosti dizajniranog sistema, na primer da li svi entiteti na kraju veze očekuju isti tip paketa.

Pravljenje simulacije obuhvata kreiranje izvršne simulacije za dizajniranu arhitekturu. Prisutne opcije uključuju korišćeni simulacioni jezik i tip kreiranja simulacije, na primer real time animacija ili trace file generation.

Simulacija sistema je izvršavanje same aplikacije i animiranje grafičkog prikaza dizajna. U ovom modu je takođe dozvoljeno menjanje parametara sistema i prikaz vremenskih oblika signala.

U **eksperiment modu** moguće je automatsko mnogostruko izvršavanje simulacije sa različitim parametrima definisanim za svako izvršavanje [31].

Tokom simulacije se automatski ispisuje sekvenca događaja u poseban dokument i moguće je iscrtavanje grafova neke izlazne veličine u funkciji promenljivog ulaznog parametara, pri čemu korisnik sam definiše svoje okruženje, kao i vremenske dijagrame. Aktivnosti u simulacionom modelu mogu se vizuelizirati na različite načine, na primer preko kretanja ikona koje prikazuju pakete podataka kroz procesor u mreži ili promenom ikone komponente koja prikazuje trenutno stanje. Korisnik na ovaj način može utvrditi da li model stvara korektne rezultate. Takođe korisniku su ponuđene opcije za ponovno pokretanje simulacije, korišćenje različitih ulaznih parametara za svako izvršenje i iscrtavanje grafikona neke izlazne vrednosti u funkciji različitih ulaznih parametara. Da bi se ovaj posao automatizovao kreiran je niz skriptova, zajedno sa grafičkim korisničkim interfejsom i dobijeno je okruženje lakše za kontolu parametara i izvršavanje.



Slika 4.6 Simulacija procesora u okviru simulacionog paketa HASE

Ceo simulator obuhvata skladište programskih podataka (za simulacione entitete, kôd simulacije, itd), simulatorsku mašinu diskretnih događaja, predstavljanje simulacionih modela, grafički mehanizam za prikazivanje i promenu parametara, mehanizam vizuelizacije i alati za podešavanje eksperimenata i prikupljanje rezultata. Projekti se čuvaju u dokumentima tipa edl (Entity Description Language) i elf (Entity Layout File). HASE čita dokumente tipa edl i od njih formira arhitekturu sistema. Dokumenti tipa elf sadrže informacije vezane za fizičko prikazivanje, na primer gde će se komponente i njihovi portovi nalaziti na ekranu, i sve promenljive koje se prikazuju. Moguće je edl opis generisati direktno iz HASE ++ simulacionog kôda.

Modeli koji su trenutno dostupni sadrže demonstraciju Tomasulovog algoritma, nekoliko verzija DLX arhitekture i modele Stanford DASH arhitekture. Svaki model podržava web sajt koji opisuje arhitekturu i model. DLX modeli sadrže jednu jednostavnu

(integer arithmetic) mrežu pipeline, jednu verziju sa višestrukim aritmetičkim jedinicama i semafora sa rezultatima i jednu verziju sa dve mreže i mehanizmom predviđanja. DASH modeli, koji sadrže verziju sa jednim čvorom sa primarnim i sekundarnim keš memorijama i verziju sa jednim klasterom sa četiri čvora međusobno povezana magistralom, se koriste da se demonstriraju protokoli keš memorije. Verzija sa višestrukim klasterom i protokolom direktorijuma je u procesu razvoja.

4.2.2.1. WebHASE i JavaHASE

HASE se trenutno pokreće u Linuksu i koristi se u nekim istraživanjima i studentskim projektima za razvoj novih arhitektonskih modela. Za potrebe nastave, ipak, učenicima je samo potreban pristup modelima simulacije, a ne sve mogućnosti koje nudi HASE, a nekoliko mehanizama za ponudu HASE-a putem www-a se još uvek istražuju. Prvi od ovih je bio SimJava, paket diskretni događaj simulacije baziran na procesu za Java-u koja se bazira na Hase ++ sa mogućnošću animacije. SimJava je postala uspešna sama po sebi, ali iz HASE perspektive ona nudi alternativni način kreiranja modela, pre nego način predstavljanja postojećih HASE modela putem www-a. WebHASE je zbog toga razvijen da omogući pristup HASE simulacijama putem pretraživača [32].

WebHASE koristi kao svoj unos HASE fajlove korišćene za neki projekat i daje kao rezultat XML fajl koji sadrži aplet i sve informacije neophodne za prikazivanje i animiranje simulacije. Ovom fajlu se može pristupiti putem www-a. Prvobitno, korisnici su mogli da menjaju parametre modela (npr. instrukcije u memoriji računara) i pošalju nove parametre nazad i time bi pokrenuli simulaciju koristeći nove parametre. Kada se simulacija završi, vraćaju se rezultati ove simulacije nazad u aplet da bi se animirao. Delom zbog toga što se ovo pokazalo kao nestabilno i delom zbog mogućnosti preopterećenja servera, ove interaktivne mogućnosti WebHASE-a su ukinute i jedan alternativni interaktivni sistem JavaHASE je razvijen umesto njega. WebHASE se još uvek koristi kao sistem za demonstraciju koji omogućava da se demo simulacije sagledaju putem weba, pošto on zahteva ne tako napredne osobine pretraživača.

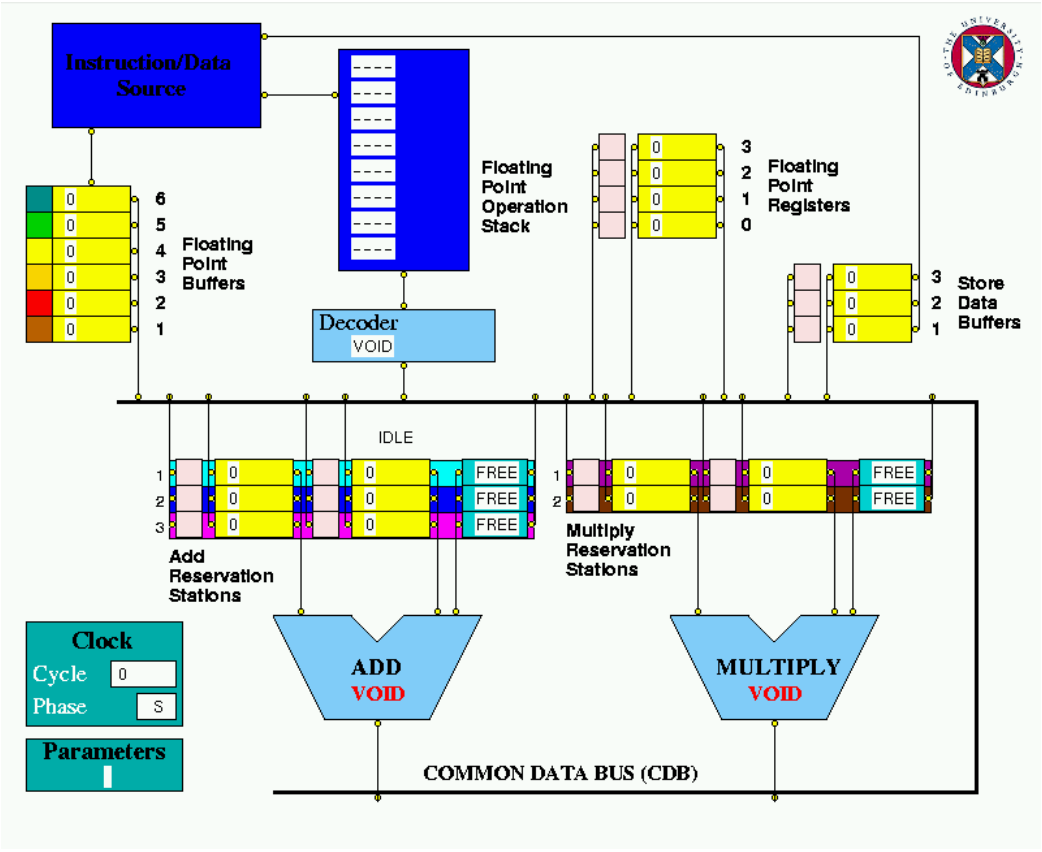
JavaHASE omogućava da se HASE projekti prevedu u potpuno razvijene modele simulacije koji sadrže aplete, a baziraju se na SimJava. Pošto je SimJava izvedena iz Hase ++, ona može da dâ kao rezultat kompatibilne datoteke, a sve prethodne verzije Hase ++ imaju direktne ekvivalente u SimJava. JavaHASE apleti [32] mogu da se preuzimaju putem www-a, a simulacije pokrenu na računarima klijenata pre nego na serveru. Da bi studenti bili u mogućnosti da izvrše vežbanja koristeći modele, apletima je neophodan pristup da seku i lepe (cut i paste) osobine koristeći klipbord na računaru klijenta. Iako

standarni menadžeri za bezbednost apleta ne dozvoljavaju pristup klipbordu, menadžer za bezbednost može biti konfigurisan koristeći Java policy fajl tako da dozvoli apletima iz određenih URL-a da pristupe klipbordu.

4.2.2.2. Tomasulov algoritam

Tomasulov algoritam (TA) je prvi put korišćen u IBM System/360 Model 91. TA je kreiran da kontroliše protok podataka između seta registra koji se mogu programirati i grupe paralelnih aritmetičkih jedinica. Kreiran je da osigura da se ograničenja koja nameću zavisnost od instrukcija ispoštuju. TA pokušava da umanji zastoj između između davanja rezultata jedne operacije i početka naredne operacije koja zahteva taj rezultat kao unos. Danas se još uvek koristi u procesorima kao što su Power PC 620 i često se koristi u nastavi računarske arhitekture.

Algoritam funkcioniše tako što koristi dodatne registre, poznate kao Reservation Stations, kod unosa ka aritmetičkim jedinicama i sistem tag-ova koji usmeravaju operande rezultata do mesta gde su potrebni, pre nego do mesta gde bi otišli kada su instrukcije koje su ih proizvele bile izdate. Algoritam je težak za objašnjavanje učenicima bez dinamične demonstracije.



Slika 4.7 Hase model simulacije Tomasulovog algoritma

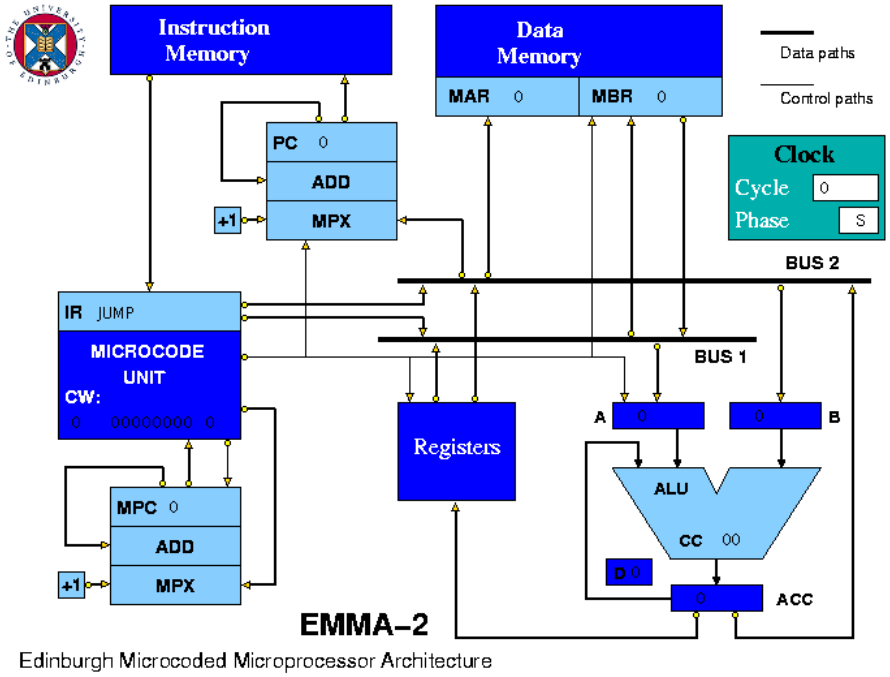
HASE model simulacije [32] napravljen je da demonstrira TA (slika 4.7). Procesor 360 i memorija su predstavljeni u modelu putem Instruction/Data Source Unit koji čuva niz instrukcija i set vrednosti podataka. Kao u modelu 91, instrukcije su prevedene u pseudo registar-registar instrukcije pre nego što su poslate u Floating point Operation Stack (FLOS). Za instrukcije koje određuju adresu čuvanja, odgovarajući operandi su poslati u okviru Source Unit-a (jednak po dužini očekivanoj memorijskoj pripravnosti) do Floating point Buffers-a (FLBs). Ovi baferi se dodeljuju ciklično i odgovarajući FLB broj se ubacuje u pseudo instrukciju izdatu Flos-u. Tagovi su predstavljeni u modelu različito obojenim ikonama; svaki FLB je predstavljen bojom svog taga pored njega i stanice rezervacije imaju boje svojih tagova kao pozadinu.

4.2.2.3. Procesor EMMA-2

Simulator EMMA (Edinburgh Microcoded Microprocessor Applet) [32] realizuje simulaciju projektovanog simulacionog modela procesora EMMA arhitekture i mikroprogramirane organizacije i operativne memorije, koja je realizovana kao posebna instrukcijsku memoriju i memoriju podataka. Arhitektura procesora podržava sve bitne elemente arhitekture procesora DLX.

Procesor se sastoji od operacione i upravljačke jedinice (Slika 4.8). Operacionu jedinicu čine programski dostupni registar PC, registri opšte namene (Registers), aritmetičko logička jedinica (ALU) i implementacioni registri A, B, ACC, MBR (Memory Buffer Register), MAR (Memory Address Register) i IR (Instruction Register) povezani sa dve interne magistrale BUS1 i BUS2. Sadržaji registara opšte namene (Registers) se puštaju na obe magistrale, a u njih se upisuje jedino preko registra ACC. Registar MAR je povezan na adresne linije, a registar MDR na ulazne i izlazne linije podataka memorije podataka (Data Memory).

Aritmetičko logička jedinica (ALU) izvršava operacije nad sadržajima registara A i B, a rezultat upisuje u registar ACC. Sadržaji sa magistrala BUS1 i BUS2 se vode u registre MAR i MBR i registre A i B. Sadržaji registara IR i MBR se šalju na magistralu BUS1, a registra ACC na magistralu BUS2. Nova vrednost se upisuje u registar PC preko magistrale BUS2. Registar PC je povezan na adresne linije, a registar IR na izlazne linije podataka instrukcijske memorije (Instruction Memory). Upravljačka jedinica (MICROCODE UNIT) je mikroprogramska i generiše upravljačke signale za sve delove operacione jedinice i instrukcijsku memoriju i memoriju podataka na osnovu sadržaja mikroinstrukcija (CW) očitanih iz mikroprogramske memorije.



Slika 4.8 Struktura EMMA-2 procesora

Prva 32 ulaza mikroprogramske memorije sadrže početne adrese mikroprograma svih instrukcija, a preostali ulazi same mikroprograme. Iz mikroprogramske memorije se čita jedanput za svaku očitanu instrukciju na osnovu polja kôda operacije iz registra IR i očitana vrednost upisuje u mikroprogramski brojač MPC, dok se posle toga iz nje čita na osnovu sadržaja mikroprogramski brojač MPC. Šema organizacije procesora i memorije je, radi preglednijeg praćenja promena koje se dešavaju tokom simulacije, prilagođena tako da cela stane na jedan ekran (slika 4.8).

Kao okruženje opšte namene, HASE ne prikazuje nikakve detalje implementacije operacione i upravljačke jedinice procesora i instrukcijske memorije i memorije podataka, već samo funkcionalne blokove (entitete). Ti entiteti su instrukcijska memorija (Instruction Memory), memorija podataka (Data Memory), upravljačka jedinica (MICROCODE UNIT), registri opšte namene (Registers), aritmetičko logička jedinica (ALU) i registri PC, MPC, A, B, ACC, MBR, MAR i IR itd. Tok podataka prikazan je razmenom paketa između entiteta. U posebnom entitetu je prikazan i takt (Clock). Takt je podeljen u dve faze: u fazi P0 vrši se izračunavanje vrednosti, a u fazi P1 komunikacija između entiteta.

4.2.2.4. HASE DLX model

DLX simulator [32] realizuje simulaciju projektovanog simulacionog modela procesora DLX arhitekture i pipeline organizacije. Arhitektura procesora podržava sve bitne elemente arhitekture procesora DLX, slično ESCAPE projektu. Organizacija pipeline-a je takva da postoji istih pet stepeni za istih pet faza izvršavanja instrukcije. Radi

izbegavanja strukturalnih hazarda[12], postoji posebna memorija instrukcija i memorija podataka, a registri opšte namene su tako realizovani da se istovremeno mogu čitati sadržaji dva registra i upisivati u treći registar. Radi izbegavanja hazarda podataka koristi se tehnika zaustavljanja. Radi smanjivanja posledica upravljačkih hazarda koristi se hardverska statička predikcija da neće biti skoka.

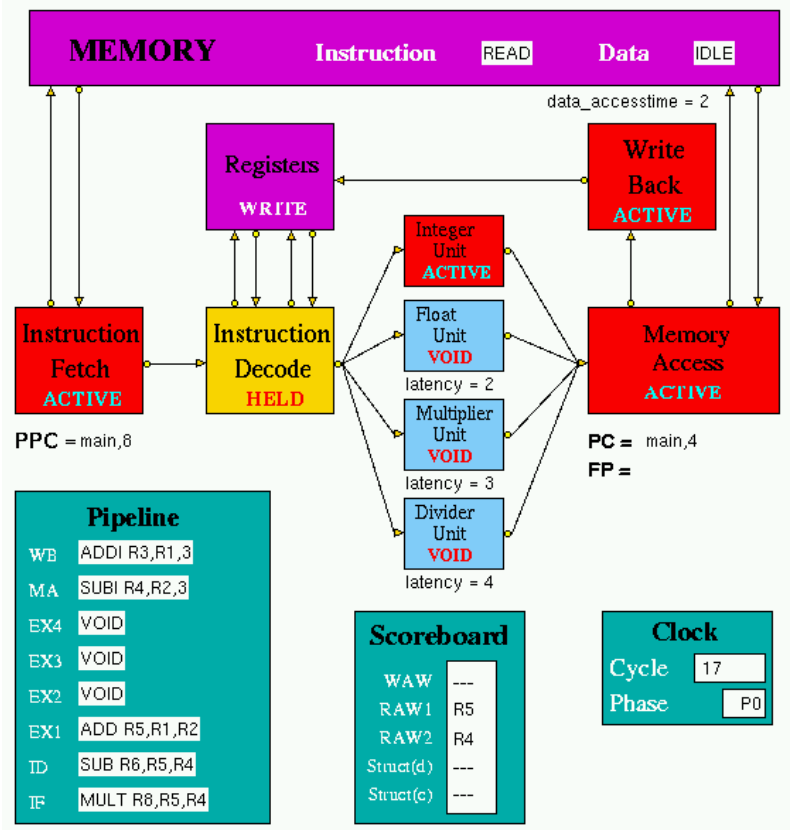
Šema organizacije računarskog sistema je, radi preglednijeg praćenja promena koje se dešavaju tokom simulacije, prilagođena tako da cela stane na jedan ekran (slika 4.9). Kao okruženje opšte namene, HASE ne prikazuje nikakve detalje implementacije pipeline-a, već samo funkcionalne blokove (entitete) koji odgovaraju stepenima pipeline-a.

Ti entiteti su:

- (1) IF (**Instruction Fetch**) u kome se realizuje dohvatanje instrukcije iz memorije, .
- (2) ID (**Instruction Decode**) koji je zadužen za dekodovanje instrukcije i čitanje sadržaja dva registra opšte namene,
- (3) EX (**Integer Unit**) u kome se relizuje izvršavanje operacije, sračunavanje adrese i realizacija skokova,
- (4) MA (**Memory Access**), zadužen je za čitanje/upis podatka iz/u memoriju i
- (5) WB (**Write Back**) koji realizuje upis rezultata u registar opšte namene.

Memorija (MEMORY) i registri opšte namene (Registers) su takođe predstavljeni kao posebni entiteti. Tok podataka kroz pipeline prikazan je razmenom paketa između entiteta, a u paketima su prikazane instrukcije i vrednosti njihovih argumenata. Tok podataka između entiteta stepena ID (Instruction Decode) i entiteta registara opšte namene (Registers) takođe je prikazan razmenom paketa između entiteta, a u paketima su prikazani adresa i podatak. Status svakog od entiteta je jasno prikazan i to tekstualno i promenom boje. Statusi stepena pipeline-a mogu biti ACTIVE, HELD (prilikom zaustavljanja) i VOID (ako u entitetu nema instrukcije), a statusi memorije i registar fajla IDLE, READ i WRITE. Instrukcije koje se nalaze u pipeline-u nisu stalno prikazane uz stepene, već u posebnom entitetu (Pipeline). U posebnim entitetima prikazani su i takt (Clock) i hazardi (Data Hazards). Takt je podeljen u dve faze: u fazi P0 vrši se izračunavanje vrednosti, a u fazi P1 komunikacija između entiteta koji odgovaraju stepenima pipeline-a.

[12] Hazard - situacija kada izvršenje jedne instrukcije biva privremeno zaustavljeno zato što operandi koje ona zahteva još uvek nisu izračunati od strane prethodne instrukcije

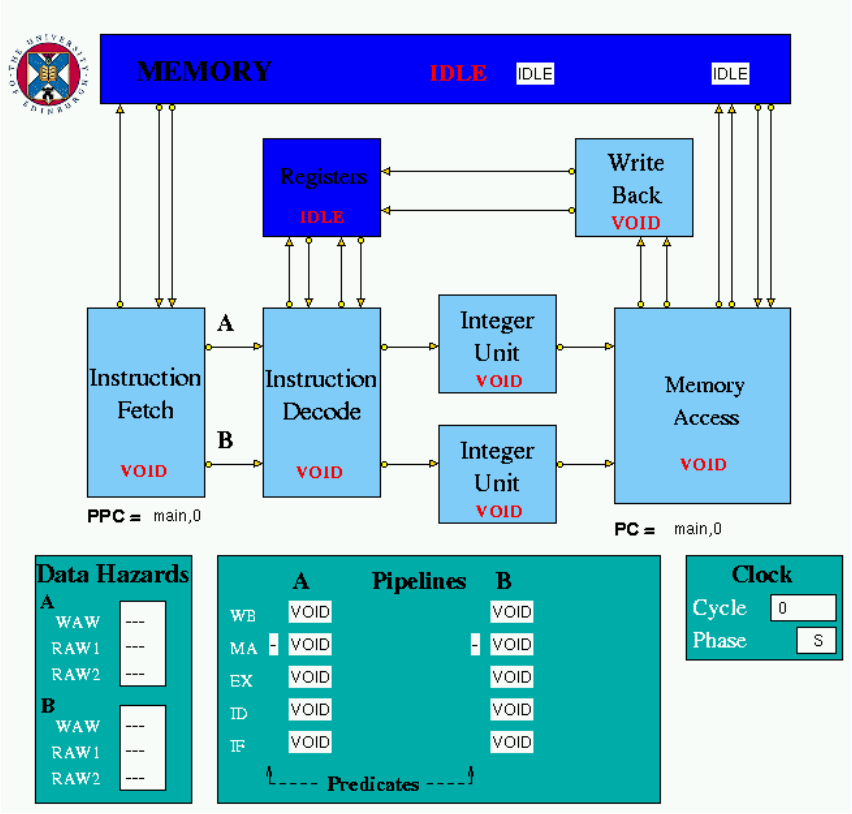


Slika 4.9 Struktura DLX procesora

JAVAHASE Predication Applet

JAVAHASE Predication Applet (slika 4.10) [32] se bazira na modifikovanom DLX modelu prvobitno razvijenom kao učenički projekat. On ima dvostruku mrežu sa dve jednostavne intedžer izvršne jedinice i dvostruke staze podataka (A i B) u svim drugim jedinicama. Set instrukcija je u suštini isti kao i kod standardnog DLX, ali instrukcijama se uvek pristupa u parovima, stimulišući efekat duple dužine VLIW, i svaka instrukcija u paru ima dodatno Predicate Tag polje koje se koristi da se izabere jedan od osam predikatskih registara (Predicate Registers). Predikatski register 1 je stalno podešen na 1. Grane (branches) se mogu pojaviti samo kao prva instrukcija para gde je druga instrukcija ekvivalent za „Delay Slot“ u tradicionalnom DLX.

Jedinica Instruction Fetch šalje instrukcije kojima je pristupila iz memorije duž dve staze podataka jedinici Instruction Decode. Jedinica Instruction Decode pristupa registrima zbog potrebnih operanda i šalje jednu instrukciju svakoj od dve izvršne jedinice. Izvršne jedinice šalju svoje rezultate jedinici Memory Access koja pristupa predikatskim registrima da bi utvrdila da li će se svaka instrukcija izvršiti. Ako je predikat za neku instrukciju tačan, instrukcija se normalno nastavlja; ako je predikat netačan, instrukcija se prekida, tj. ne čita ili ne piše memoriju niti obnavlja registar.



Slika 4.10 Struktura DLX - Predication Pipeline

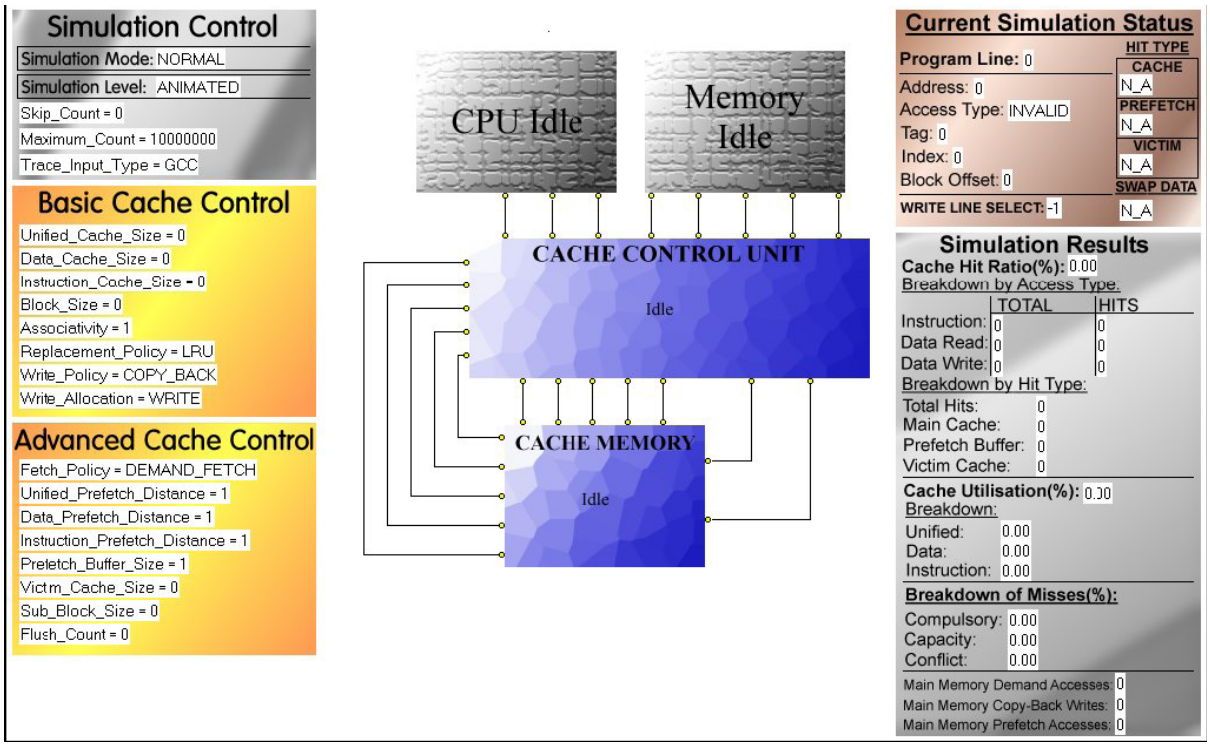
4.2.2.5. HASE Dinero

HASE Dinero je realizovan tokom 1999. god. u okviru projekta na četvrtoj godini odseka Computer Science na Edinburgh University. Ovaj simulator [32] predstavlja alat za učenje i dizajniranje različitih arhitektura keš memorije. Realizovan je simulacioni model prvog nivoa keš memorije u memorijskoj hijerarhiji. HASE DINERO je nastao integracijom dva postojeća simulatora: The Hierarchical computer Architecture design and Simulation Environment (HASE) i Dinero. U suštini ovaj simulator predstavlja implementaciju Dinero sistema unutar HASE. Originalna verzija Dinero je bila veoma teška za korišćenje, ali je predstavljala odličan osnov za učenje osnovnih keš arhitektura i operacija. Zato je iskorišćen vizuelni prikaz komponenata HASE sistema.

Dinero je keš simulator koji se izvršava sa komandne linije. To je trace-driven simulator, što znači da se simulacioni rezultati dobijaju na osnovu velikih ulaznih trace fajlova. Ulazni fajlovi se sastoje od oko milion memorijskih referenci, koje su dobijene tokom kursa o izvršavanju aplikacija. Svaka referenca unutar fajla sadrži i heksadecimalnu memorijsku adresu i tip pristupa referenci, koji može biti čitanje podatka, upis podatka ili dohvatanje instrukcije. Većinu osnovnih parametara keš arhitekture je moguće menjati za potrebe simulacije.

Moguće je koristiti i dve napredne tehnike arhitekture keš memorija sub-block zamena i prefetching. Dinero simulator je realizovan za UNIX operativni sistem. Glavne karakteristike HASE Dinero simulatora su animacija keš aktivnosti, veći broj različitih izveštaja sa rezultatima simulacije, mogućnost pokretanja simulatora u tutorijal modu sa iscrpnim objašnjenjima. Takođe, tokom simulacije moguće je pratiti i menjati veći broj parametara. Unified, Data and Instruction Cache Size predstavlja veličinu pojedinačnih keš memorija u bajtovima. Block Size predstavlja veličinu ulaza u keš memoriji u bajtovima. Associativity predstavlja broj setova kod set-asocijativnog preslikavanja. Ako je ovaj parametar postavljen na jedan, tada se simulira keš memorija sa direktnim mapiranjem. U slučaju da je ovaj parametar postavljen na nulu, simulira se keš memorija sa asocijativnim preslikavanjem, Maksimalan broj setova je Cache Size / Block Size. The replacement policy definiše na koji način se bira ulaz u keš memoriji koja se zamenjuje u slučaju da se novi blok učitava, a ne postoji slobodan ulaz.

Mogući algoritmi su Least Recently Used (LRU), First-In First-Out (FIFO) ili Slučajan izbor. The write policy definiše da li se pri upisu podatka koristi Write-Through tehnika ili Copy-Back. Ako je parametar Write Allocation postavljen na Write tada se svi upisi podataka od strane procesora upisuju u keš memoriji. Ako je ovaj parametar postavljen na No Write tada se nijedan od podataka pri upisu od strane procesora ne upisuju u keš memoriju.



Slika 4.11 Simulacija sa HASE DINERO simulatorom

Simulacija se može izvršavati na dva načina: Animated i Fast.

U **Animated modu**, sve aktivnosti tokom simulacije se mogu vizuelno pratiti, ali limit ulaznog fajla je 500 linija. Ovaj mod je predviđen za bolje razumevanje keš operacija, jer se na jednostavan način može uočiti pogodak i promašaji u keš memoriji, kao i efekti promena keš konfiguracija.

Fast mod je dosta efikasniji način simulacije. Kod ovog moda ulazni fajl može sadržati i do 10 000 000 linija. Nedostatak je što pri simulaciji ne postoji nikakva animacija, samo se prikazuju rezultati za određeni objekat. Ovaj mod je predviđen za analiziranje kako različite keš konfiguracije utiču na perfomanse računarskih sistema. Pomoću HASE DINERO simulatora studenti mogu da rade vežbe sa sledećim temama: keš memorije sa direktnim preslikavanjem, keš memorije sa asocijativnim preslikavanjem, keš memorije sa set-asocijativnim preslikavanjem, prednosti povećanja veličine bloka, operacija sa Split instukcijom, demonstracija algoritma LRU zamene ulaza, demonstracija algoritma FIFO zamene ulaza, demonstracija algoritma Slučajan izbor zamene ulaza, demonstracija algoritma Copy-Back upisa, demonstracija algoritma Write-Thorugh upisa, prednosti Victim keš tehnologije.

Apleti simulacije nude način vizualnog predstavljanja akcija koje se dešavaju unutar računarskog sistema dok se programi izvršavaju i virtualnu laboratoriju u kojoj učenici mogu da rade vežbanja koja će i testirati i povećati njihovo razumevanje koncepta računarske arhitekture. Nekoliko apleta opisanih ovde se koriste i lokalno na Edinburškom univerzitetu i šire na „Institute for System Level Integration“, zajednički poduhvat između 4 škotska univerziteta smešten u Livingstonu, 30-ak km od Edinburga. Učenici, takođe, mogu da koriste ove aplete kod kuće, a njihova upotreba za učenje na daljinu se razmatra. U početku je bilo nekoliko operativnih teškoća, ali su ove uglavnom uzrokovane problemima oko kompatibilnosti pretraživača i instalacija. „Web start“ tehnologija se trenutno ispituje kao način da se izbegnu ove teškoće. Postojećim JavaHASE apletima se može pristupiti sa HASE web sajta na www.icsa.informatics.ed.ac.uk/research/groups/hase [32]. Kreiranje apleta i web sajtova povezanih sa njima čini učeničke projekte interesantnim i novi apleti se stalno razvijaju.

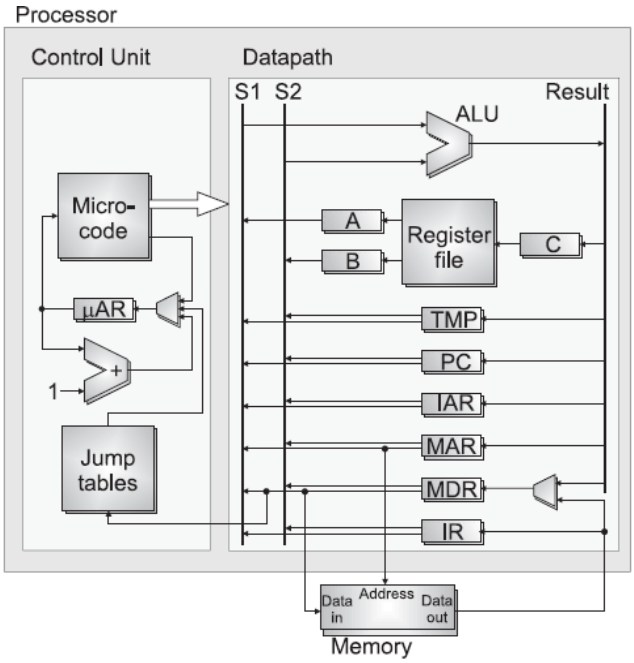
4.2.3. ESCAPE - Simulaciono okruženje za kurseve iz arhitekture računara - Univerziteta Ghent, Belgija[13]

ESCAPE (Environment for the Simulation of Computer Architectures for the Purpose of Education) je simulator razvijen na Odseku za elektroniku i informacione sisteme univerziteta Ghent, Belgija. To je lako za korišćenje, interaktivno, grafičko, PC bazirano simulaciono okruženje napravljeno sa ciljem da podrži kurseve iz arhitekture računara.

ESCAPE okuženje se sastoji od simulatora dva računarska sistema sa istim skupom instrukcija. Arhitektura ovog skupa instrukcija inspirisana je Ilennessy i Patersonovim DLX-om [35]. Nasuprot DLX arhitekturi veličina polja kod kodiranja nije fiksna, ali zavisi od maksimalnog broja instrukcija i veličine registarskog fajla. Instrukcija za čitanje mogu biti u jednom od 6 formata, pa je ostavljena mogućnost implementiranja mnogih naprednih operacija u mikroprogramiranoj arhitekturi. Razlika ova dva simulatora je u organizaciji.

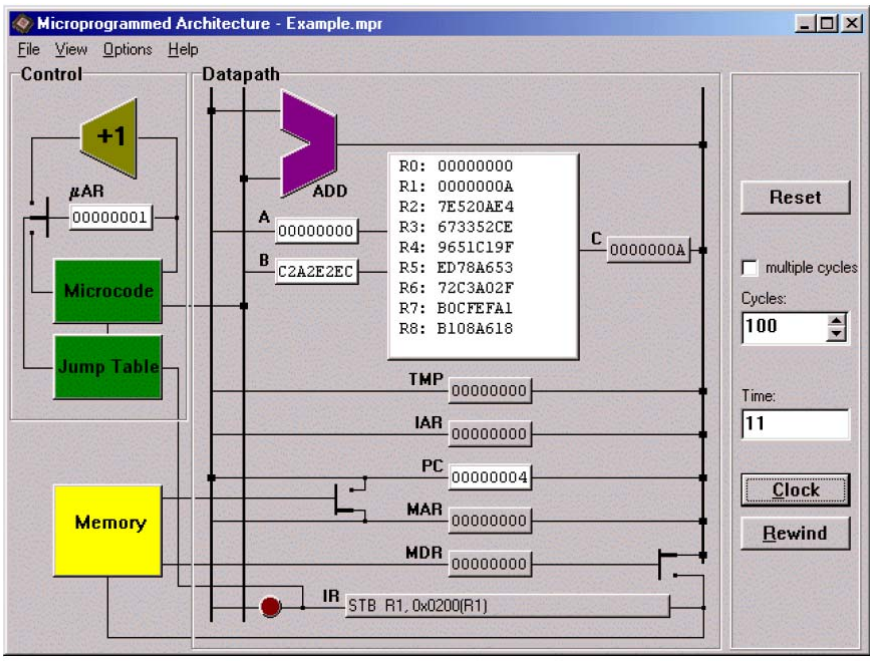
Prvi je mikroprogramirani procesor Von Neumannove organizacije. Na slici 4.12 dat je mikrokod arhitekture procesora, dok je na slici 4.13 dat primer izgleda ekrana (screenshoot) ESCAPE simulatora za mikroprogramirani procesor.

Drugi je pipeline procesor sa jednostavnim pipeline-om. Na slici 4.14 dat je pajplan arhitektura, dok slika 4.17 prikazuje izgled ekrana ESCAPE simulatora za pajplan procesor



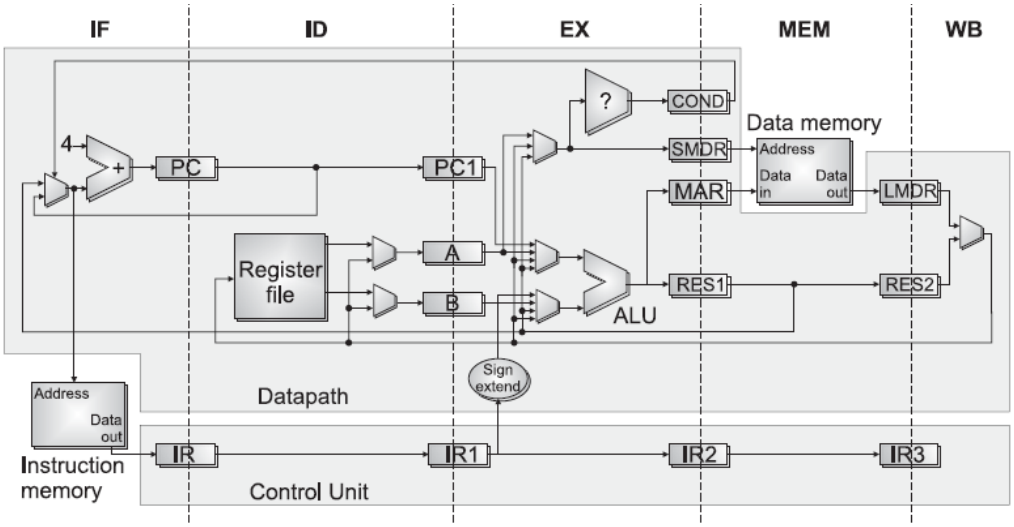
Slika 4.12 Mikrokod arhitekture procesora

[13] Jan Van Campenhout, Peter Verplaetse, Henk Neefs, Department of Electronics and Information Systems, University of Ghent, Belgium

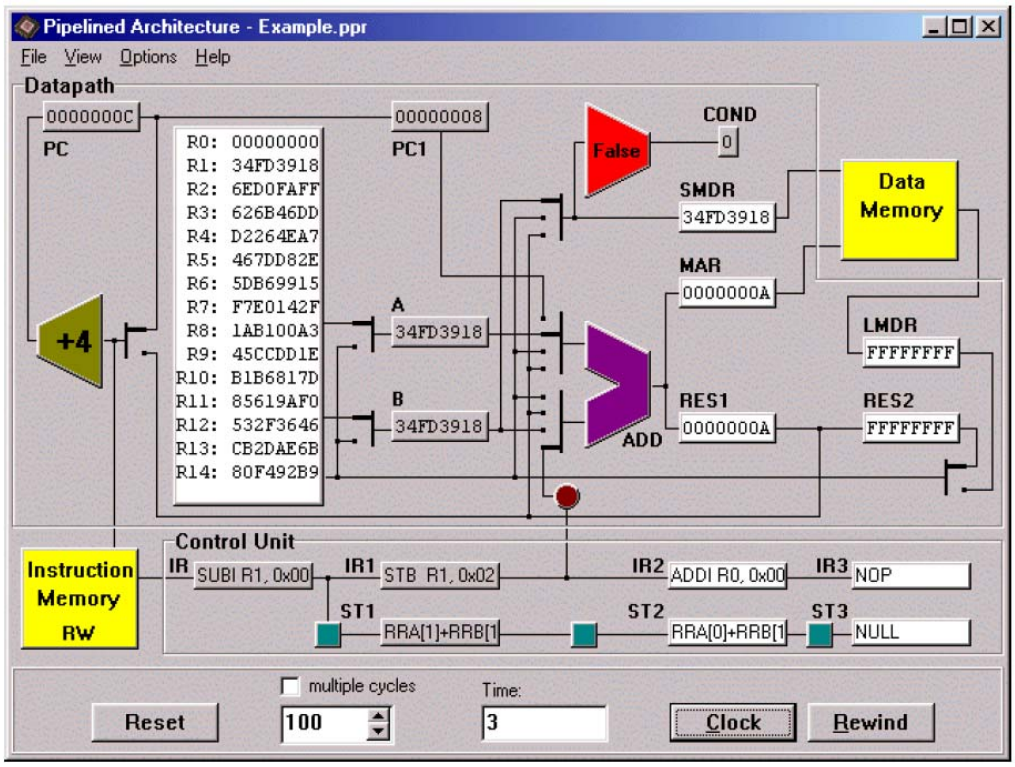


Slika 4.13 Primer izgleda ekrana pri simulaciji ESCAPE simulatora- mikroprogramirani procesor

Mikroprogramirani procesor se sastoji od operacione i kontrolne jedinice. Operaciona jedinica se sastoji iz skupa opštih registara i skupa organizacionih registara i ALU jedinice koja može da simulira jedan broj osnovnih operacija u jednom taktu. Postoje indikatori nule i znaka. Kontrolna jedinica je mikrokodirana. Svakog takta mikrokod adresa se inkrementira ili zamenjuje sa apsolutnom vrednošću. Ova vrednost, koja može biti u mikrokodu ili se čita iz tabele skokova, indeksira se iz polja koda instrukcije. Adresa skoka se čita iz tabele skoka, pri čemu je indeks definisan poljem koda instrukcije. Sa memorijom se obavlja transfer bajta, pola reči (16 bita) ili cele reči (32 bita). Instrukcije i podaci su smešteni u memoriji.



Slika 4.14 Pipeline arhitektura



Slika 4.15 Primer izgleda ekrana pri simulaciji ESCAPE simulatora - pipeline procesor

Kod pipeline organizacije procesora pipeline se izvršava u pet faza - instruction fetch, instruction decode, execute and efective address calculation, memory i write back. Postoje dva, odvojena memorijska interfejsa, jedan je za podatke, drugi za instrukcije. Simulacija se može pratiti ciklus po ciklus ili više ciklusa odjednom. Pomoću prekidnih tačaka može se zaustaviti izvršavanje simulacije. Postoji i opcija za vraćanje unazad određen broj taktova, ali ne više od 1024.

Postoje još i forme za pregled ili promenu stanja memorije (stanje memorije moguće je menjati i preko unosa asemblerskog koda), za pregled mikroinstrukcija i tabele skoka (mikrokod forma) i za konfiguraciju sistema. Moguće je prilagoditi broj registara, za svaku instrukciju je prikazan njen kod, tip i simboličko prikazivanje. Memorijski interfejs može upravljati sa bajtom, pola reči ili celom reči pristupa memoriji. Takođe korisnik definiše pristup memoriji u celom broju taktova - bira broj između 1 i 9. Postoje i asembler/disembler i alati za analizu pipeline aktivnosti i dijagrami postupka, koji doprinose boljem praćenju simulacije.

Može se uočiti i nekoliko nedostataka sistema. Trenutno sve operacije su sa znakom, ne postoji podrška ALU operacijama bez znaka. Takođe u ALU jedinici ne postoji detekcija prekoračenja. Ne postoji bilo kakva opcija sa korišćenjem spoljnih prekida. Korisnik može da upravlja sa vrlo malo elemenata, posebno za pipeline arhitekturu. Prozori ne izgledaju lepo sa većim fontovima. Okruženje je implementirano u

alatu Borland Delphi. Kako je kod kompatibilan sa verzijom Delphi 1.0 postoje i 16-to i 32-bitne verzije, pa je moguće pokrenuti simulaciju na bilo kom Windows operativnom sistemu.

4.2.4. DLXview - Interaktivni pipeline simulator - Univerzitet Purdue, Indijana[14]

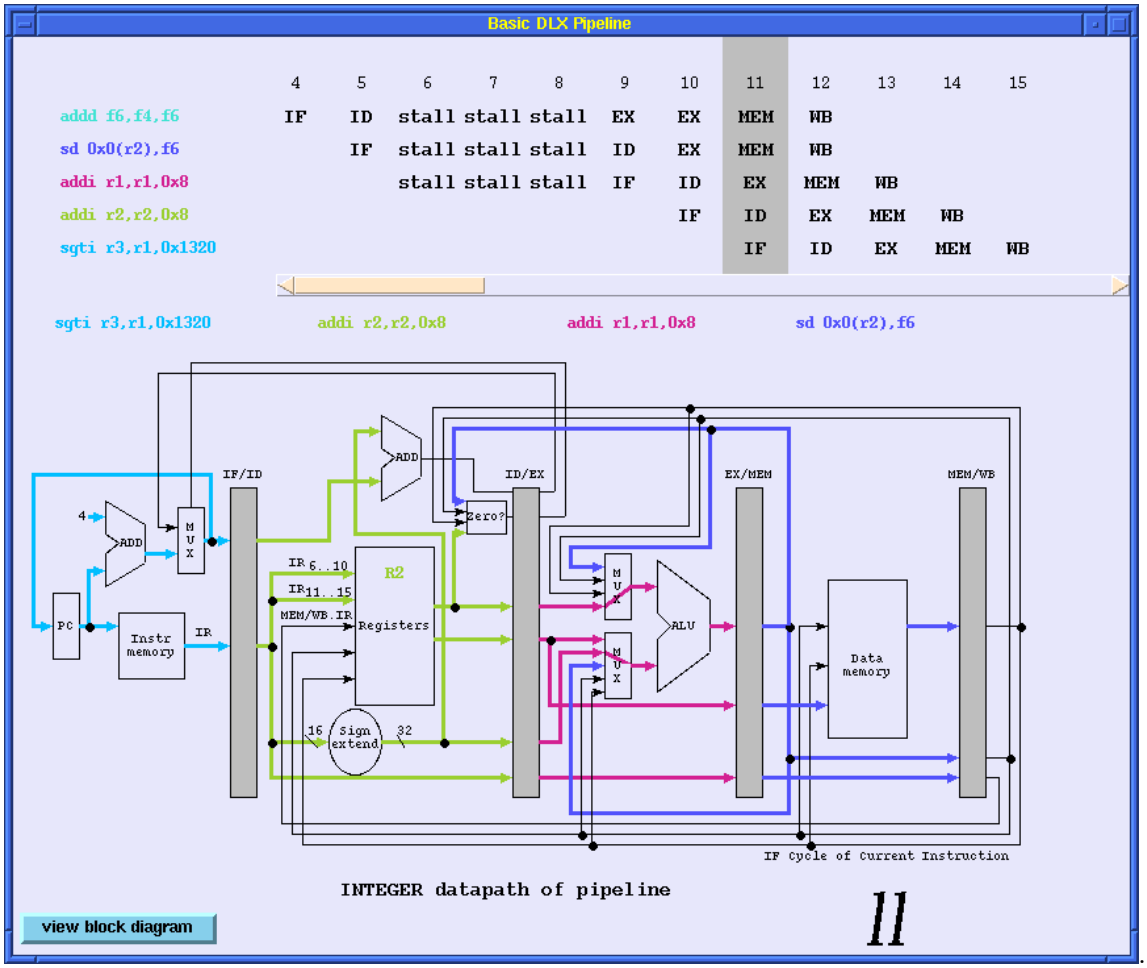
Trenutna verzija DLXview simulatora je komponenta CASLE (Compiler/Architecture Simulation for Learning and Experimenting) [33] projekta na univerzitetu Purdue, School of Electrical and Computer Engineering. To je interaktivni pipeline simulator koji koristi DLX skup instrukcija.

Glavni cilj DLXview-a je da obezbedi vizuelno, interaktivno okruženje, gde bi se operacije pipeline procesora mnogo lakše razumele, nego čitanjem običnog tekstualnog opisa. Pomoću ovog simulatora bolje se razume i DLX skup instrukcija, debagiranje i perfomanse procesora.

U okviru simulatora razmatraju se tri verzije DLX pipeline-a: osnovna (slika 4.16), scoreboard (slika 4.17) i Tomasulo (slika 4.18). Za svaku verziju parametri, kao što su broj funkcionalnih jedinica, njihovo kašnjenje i da li je nad njima primenjen pipeline, mogu se menjati i tokom same simulacije. Samo se konfiguracija samog procesora ne može više menjati kada se jednom simulacija startuje, već mora biti konfigurisana pre čitanja dokumenata. Dokumenata ima tri tipa i to *.s su sa asemblerskim kodom, *.d sa podacima, a *.i sa komandama za inicijalizaciju registara koji se koriste.

Korisnik može pratiti sledeće informacije: uobičajena vremenska tabela pipeline-a pokazuje taktove, instrukcije u pipeline-u i njihov tok u vremenu i svaki stall ciklus; šemu pipeline-a bilo operacione jedinice jedinice za rad sa celobrojnim veličinama, ili operacione jedinice jedinice za rad sa veličinama u pokretnom zarezu sa obojenim aktivnim stepenom; za integer instrukcije posebnu aktivnu operacionu jedinicu, obojenu, sa imenima aktivnih registara. Tokom same simulacije moguće je vratiti se jedan takt ili jednu instrukciju unazad, da bi se bolje proučilo ponašanje sistema.

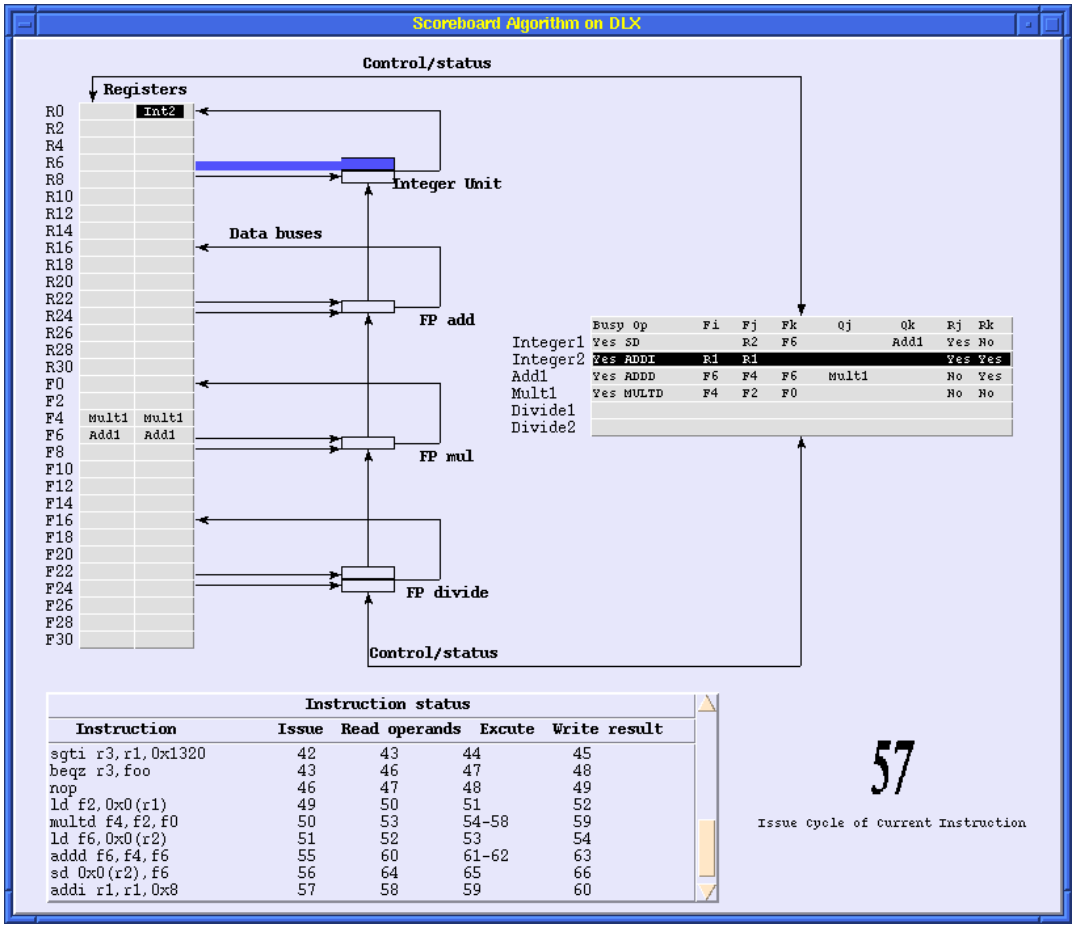
[14] John Hennessy and David Patterson, School of Electrical and Computer, Purdue University's



Slika 4.16 Primer izgleda ekrana simulatora DLXview - osnovna verzija pipeline-a

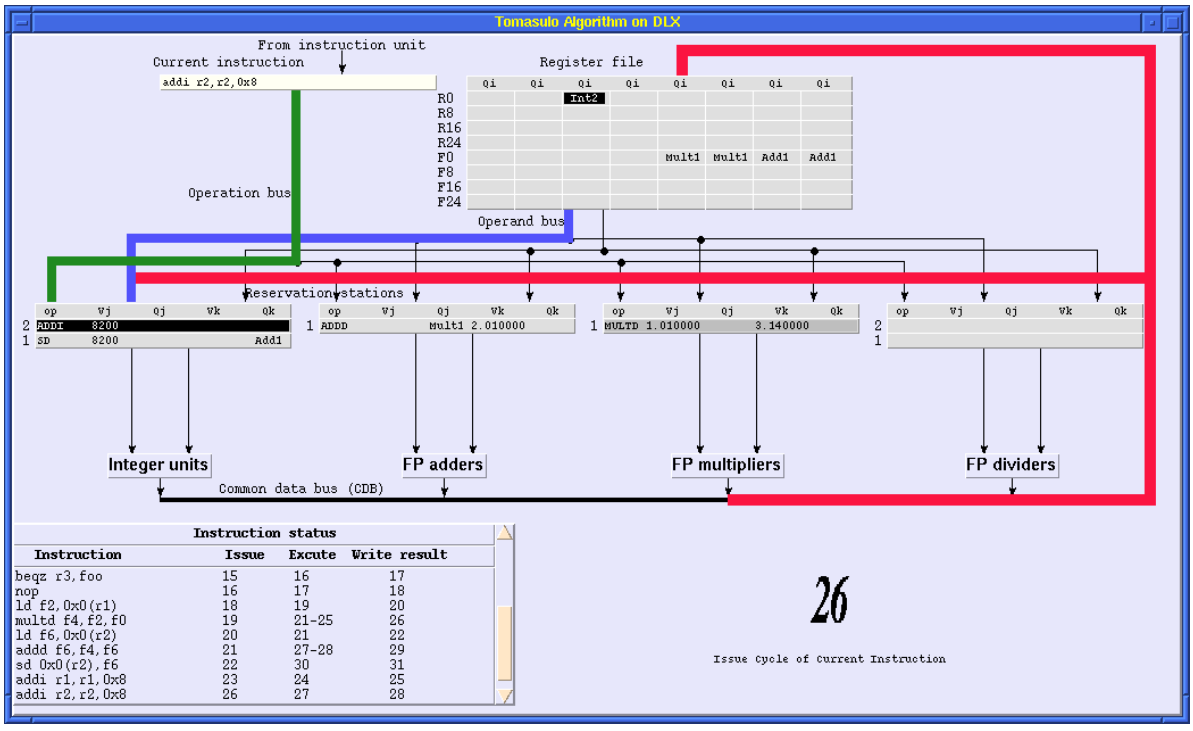
DLXview je modifikovana i proširena verzija DLXsim-a, ne-grafičkog DLX pipeline simulatora, koji je nastao od MIPS simulatora. Pomoću ovog alata se želi obezbediti sveobuhvatan pedagoški alat za učenje koncepata arhitektura, tehnologije kompajliranja i njihove međusobne veze. Cilj nastavka rada je da se pomoću iskustava sa DLXview-om, izgradi mnogo jači simulator, koji će pokriti generalne izvršne mikroparalelne modele, kao što su superskalar i VLIW, sa različitim konfiguracijama.Takođe bi taj budući alat obuhvatio i neke popularne RISC procesore, kao što su Alpha i PowerPC.

Za DLXview softver potreban je sistem sa Unix i X11, sa instaliranim Tcl/Tk paketima i preporučuje se korišćenje GNU gcc kompajlera. Za prikazivanje dijagrama preporučuje se ekran sa 256 boja i rezolucijom 1024x768 piksela. Sistem je testiran na Solaris 2.3, SunOS 4.1, HP-UX 9.0, DEC OSF/1 4.0, i Linux kernel 1.2.1.



Slika 4.17 Primer izgleda ekrana simulatora DLXview - scoreboard verzija pipeline-a

Osnovnoj verziji DLX simulatora može se pristupiti preko adrese http://cobweb.ecn.purdue.edu/~teamaaa/dlxview/ [33].



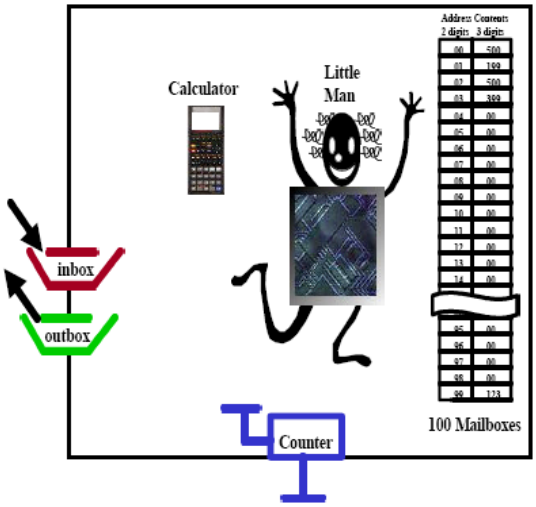
Slika 4.18 Primer izgleda ekrana simulatora DLXview - Tomasulo verzija pipeline-a

4.2.5. LMC - Vizuelna računarska simulaciona nastavna sredstva

U ovom poglavlju se opisuje korišćenje posebnog tipa simulatora računara kao sredstva za nastavu/poučavanje arhitekture računara. Paradigmu Little Man Computer (LMC) razvio je Stjuart Mednik (Stuart Madnics) sa MIT-a (Massachusetts Institute of Technology) šezdesetih godina XX veka i prošao je vremenski test kao konceptualno sredstvo koje pomaže da student razume osnove rada računara. Uspehom LMC paradigme, LMC simulatori su se proširili. Uporediće se aktuelno mnoštvo LMC simulatora rasvetljavajući vizuelne karakteristike.

4.2.5.1. LMC paradigma

LMC paradigma se sastoji od uokvirene mail'sobe, 100 mail'boksova numerisanih od 00 do 99, kalkulatora, dvo-digitalnog lokacionog brojača, input korpe, i output korpe. Svaki mail'boks je projektovan da zadrži papirnu traku na kojoj je napisan trodigitalni decimalni broj. Svaki mail'boks ima jedinstvenu adresu i sadržaj svakog mail'boksa je različit od adrese. Kalkulator može da se koristi za input/output, povremeno skladištenje, i dodavanje. Dvodigitalni lokacioni brojač se koristi za inkrementiranje proračuna vremena za izvršavanje Little Man instrukcija. Lokacioni brojač ima reset lociran van mail'sobe. Konačno, postoji „Little Man", koji izvršava zadatke u okviru mail-sobe.



Slika 4.19 LMC paradigm

Nakon resetovanja switch-a za lokacioni brojač, komunikacija sa spoljnim korisnikom se ostvaruje pomoću Little Man-a preko kartice papira sa trodigitalnim brojevima postavljenim u input korpu ili pronađenim iz output korpe. Algoritam je sledeći: [26]

- LM gleda u brojač, memoriše brojeve, i inkrementira brojač,
- Pronalazi instrukciju iz mail'boksa specificiranu pomoću memorisanog broja,

- Izvršava se instrukcija.

Instrukcija može biti:

- Aritmetička instrukcija (dodavanje ili oduzimanje),
- Memorijska (load/store) u kojoj se podaci prenose između mail-boksa i kalkulatora,
- I/O instrukcija (input/output) u kojoj se podaci prenose između kalkulatora i input/output trase,
- Program prati instrukciju (skoči, razdvoj, preskoči) koja uslovno ili bezuslovno menja vrednost u lokacionom brojaču,
- Pauza koja pravi pauzu u izvršavanju programa.

LMC paradigma je ilustrovana slikom 4.19.

Analogija između LMC i realnih računara nije savršena. Kod realnih računara memorija (mail-boks) je odvojena i fizički i funkcionalno od CPU. Kod većine računara registri su upotrebljivi za zadržavanje podataka povremeno dok su u procesu. Mada LMC paradigma nema registre, displej kalkulatora služi i kao slabo povezan akumulator. Vremenski sat i prekidi nisu deo LMC paradigme. Konačno, LMC instrukcioni set je zasnovan na decimalnom sistemu, ne binarnom kao realni računari. Uprkos ovim odstupanjima od realnosti, korišćenje ovog jednostavnog ali moćnog modela obezbeđuje studentima da se fokusiraju na razumevanje zadataka obavljanja izvršnih instrukcija pre nego na kompleksne detalje specifičnih implementacija.

4.2.5.2. Višestruki LMC simulatori

Sa uspehom LMC paradigme, LMC simulatori su se umnožavali. Aktuleno širenje LMC'a nije samo funkcionalno u procesiranju informacija, već takođe koristi vizuelne tehnike za rasvetljavanje LMC operacija. Cilj ovoga je da obezbedi studentima vizualizaciju različitih delova LMC simultano za pokazivanje odnosa i koristi i intuiciju studenata [26].

Neočekivani uvidi se mogu dobiti upoređivanjem aktuelnih LMC simulatora koji rade pod istom paradigmom, ali se svaka implementacija razlikuje po svojim slabostima i prednostima. Startovaćemo sa opisom najsofisticiranijeg LMC simuatora i vratićemo se na opis manje razvijenih LMC-a. Nezavisno od sofisticiranosti, svi LMC simulatori su uspešni u fokusiranju na različite aspekte obrazovanja u poučavanju/nastavi arhitekture računara.

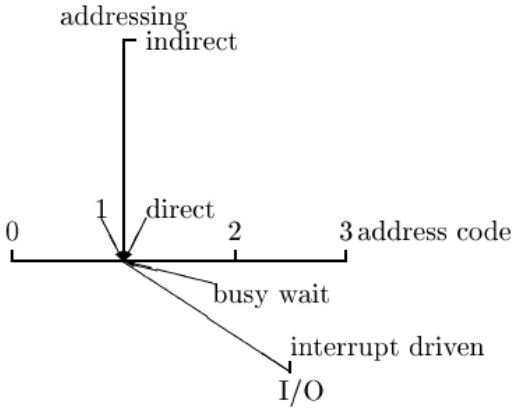
Poostrom računar

Postroom računar je najravzijenije proširenje LMC modela, projektovano da uvede pojedine aspekte arhitekture računara i programiranje nižeg nivoa na inkrementalni način. Proširenja su projektovana da prošire računarske modele u okviru LMC paradigme. Mogu da budu povezani i sa LMC paradigmom i sa „realnim“ računarima.

Glavna proširenja su:

- opseg seta instrukcija arhitekture – 0, 1, 2, i 3-adresne mašine,
- izbor adresne arhitekture – neposredna ili indirektna,
- izbor I/O arhitekture – (implicitno) čekanje ili prekid pogona,
- veliki broj mail-boksova, veliki adresni prostor i veliki (višestruko digitalni) lokacioni brojač,
- neki opkodovi su različiti.

Proširenje Postrom računara je u visokom stepenu ortogonalno, inspirisano arhitekturom PDP11. Ova ortogonalnost znači da je korektnije govoriti o 16 odvojenih Postroom računara, nego o jednom. Osnovni Postroom računari su veoma slični sa originalnim LMC, dok su napredniji modeli sličniji realnim računarima, posebno PDP11. Dizajn prostora Postroom računara je pokazan na slici 4.20. [26]



Slika 4.20 Dizajn Postroom računara

Proširenje instrukcionog seta Postroom

Prvo proširenje paradigme je bilo uvođenje različitih arhitektura instrucionog seta. Postoje 4 arhitekrute insturkcionog seta Postroom računara:

- 0-adresne stack-based mašine,
- 1-adresne mašine zasnovane na akumulatoru,
- 2-adresne mašine, sa jednim izvornim operanda i jednim destinacionim operandom,
- 3-adresa mašine, sa 2 izvorna operanda i 1 destinacionim operandom.

Osnovni dizajn seta instrukcija je uobičajen za sve mašine. OPCODE se sastoji od jednog digitalnog i adresnog polja sa tri digitalne dužine. Reč n-adresne mašine je 3 (n+1) digitalne dužine. Ovo znači da svaka mašina može da ima do 1000 trodigitalnih opkodova, koji omogućavaju uvođenje CISC arhitekture, ali je u didaktičkoj praksi bolje postaviti konceptualno jednostavnije jednodigitalne opkodove. Postoje minimalne razlike u nastavi – na primer, nulto-adresne mašine imaju PUSH (push to stack) i POP (pop from stack) instrukcije, pre nego LDA (load accumulator) i STA (store accumulator), dok dvo i troadresne mašine koriste ove opkodove za MOV (move value) i MEA (move effective address). Dvo i troadresne mašine menjaju JMP (unconditional jump) i SKP (conditional skip) instrukcije za nultu ili jednoadresnu mašinu sa pojedinačnim JMP instrukcijama, i koriste extra opkôd za MSK instrukciju (logical mask).

Proširenje moda Postroom adresiranja

Drugo proširenje bazičnog modela uvodi ekstra fleksibilnost u adresiranju. U osnovnom modelu (direktni adresni model), trodigitalna adresa je jednostavno adresa u memoriji. U indirektnom adresnom modelu, trodigitalnost je podeljena na jednodigitalni adresni mod i dvodigitalni registarski broj. Veliki broj registara se većinom koristi za implementaciju mikrokoda, koje je zasnovana na RISC arhitekturi. Postoji 10 korisničkih registara (registri opšte namene) i 20 „administrativnih“ registara – programski brojač, stack pointer, flag registar, MAR, MDR, I/O, itd. Adresnih modovi su:

- Direktni i indirektni,
- Prosečni direktni i indirektni,
- Osnovni direktni i indirektni,
- Predekrementacioni direktni i indirektni,
- Postinkrementacioni direktni i indirektni.

Proširenje Postroom Input/Output-a

Treća dimenzija u dizajniranju prostora je data Postroom računarskom IO modelu. U osnovnom Postoom Computer izvršava se neka IO instrukcija, kao i svaka druga pojedinačna instrukcija. Ovo je implicitno forma čekanja zbor zauzetosti I/O. U prekidu I/O modela, I/O instrukcija će izazvati prekid. Prekid mora da omogući da se prekid i prenos nužnih podataka do I/O modula vrši prema zahtevanim procedurama. Kompletiranjem I/O, I/O modul će uzrokovati drugi prekid, itd.

Izgrađivanje opštenamenskog računarskog sistema zasnovanog na Postroom računaru

Postroom računarski sistem dozvoljava korisniku da asemblira, loaduje i trasira izvršenje programa Postroom računara. Takođe se modeluje različita brzina komponenti Postroom računara.

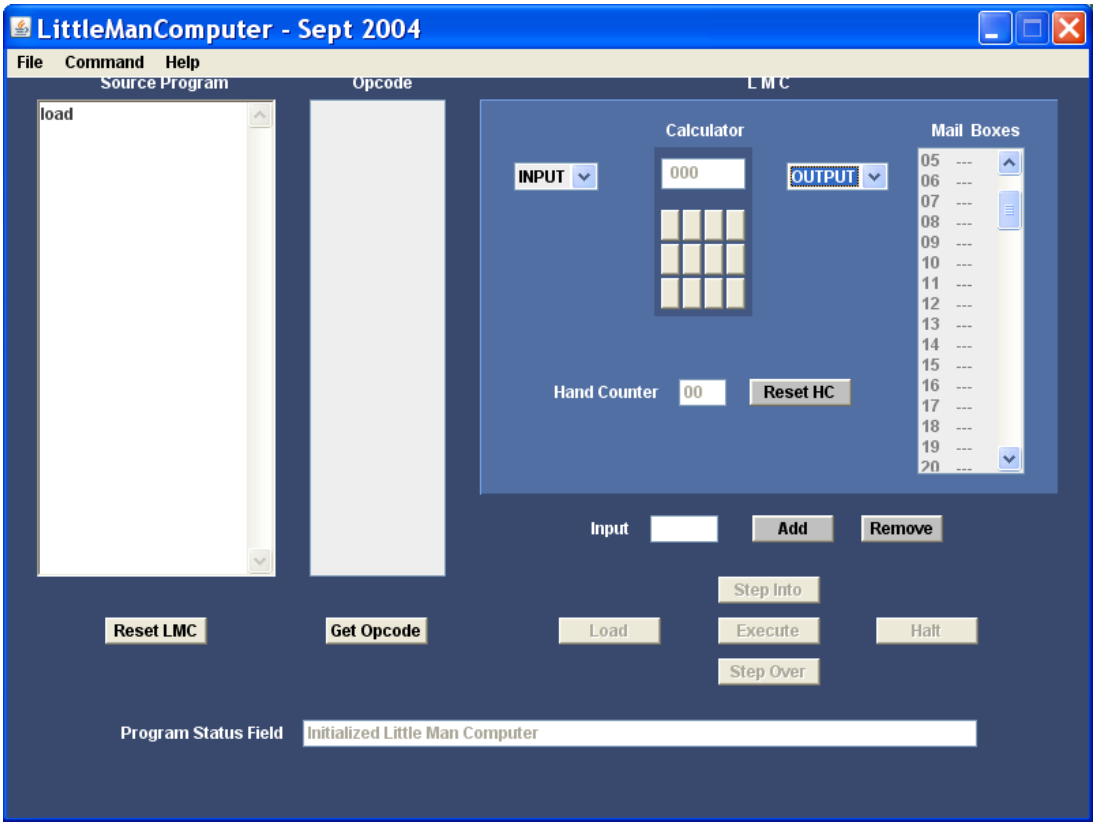
Postroom računarski asember ima sva uobičajena svojstva asemblera: leksičku i sintaksičku listu, simboličke oznake, pamćenje opkodova, registre, adresne modove uslove kodiranja. Podržani su i integer, i karakterni vrednosni podaci, a stringovi su sekvencijalnog karaktera. Postoje, takođe, i makromehanizmi, uključujući makroparametre, dozvolu da korisnik programa ima sopstvene makroe, na primer MUL.xy makro za multipliciranje. Posrednički fajlovi su stvoreni na svakom stadijum asembliranja, i mogu da služe za inspekciju, i za ilustrovanje asemblerskih procesa – na primer makro ekspanzije, oznaka generacija – za studente.

Loader će zadržati fajl mašinskog koda Postroom računara i start izvršenja. Za program će izveštaj biti neki broj za izvršavanje instrukcije, i može da bude konfigurisan takođe da izveštava o vremenu izvršenja. Naprednije konfiguracije omogućavaju korisniku da modeluje različite nivoe efikasnosti za različite komponente Postroom računara, definišući (virtuelno) vreme za to, procenjivati registre i keš, glavnu i sporednu memoriju, i bazične I/O operacije.

LMC-1 zasnovan na web-u

Pod rukovodstvom profesora Lerija Bramboa i Viljema Jurika (Larry Brumbaugh and William Yurick of Illinois State University), razvijen je interaktivni LMC simulator (slika 4.21) [34] tako da studenti mogu da vizualizuju simultane događaje tokom izvršavanja svog EMC assemblerskog programa. Ova interaktivna vizuelizaciji postavljena je kao web-based aplikacija implementirana u Java okruženju u formi apleta. Jedinstvene karakteristike za ovaj web-zasnovan LMC-1 simulator obuhvataju:

- LMC asemblerski kodni editor sa čeking listom instrukcione sintakse,
- Jednoprolazni asembler za vizuelizaciju mnemoničkog asemblerskog jezika za konverzaciju sa mašinskim kodom,
- Status programskog polja koje je prikazano aktuelnim statusom LMC operacija koje uključuju grešku poruke i flagove (flags zasnovane na vrednosti kalkulatora ako je pozitivna, negativna, nulta, ili pogrešna),

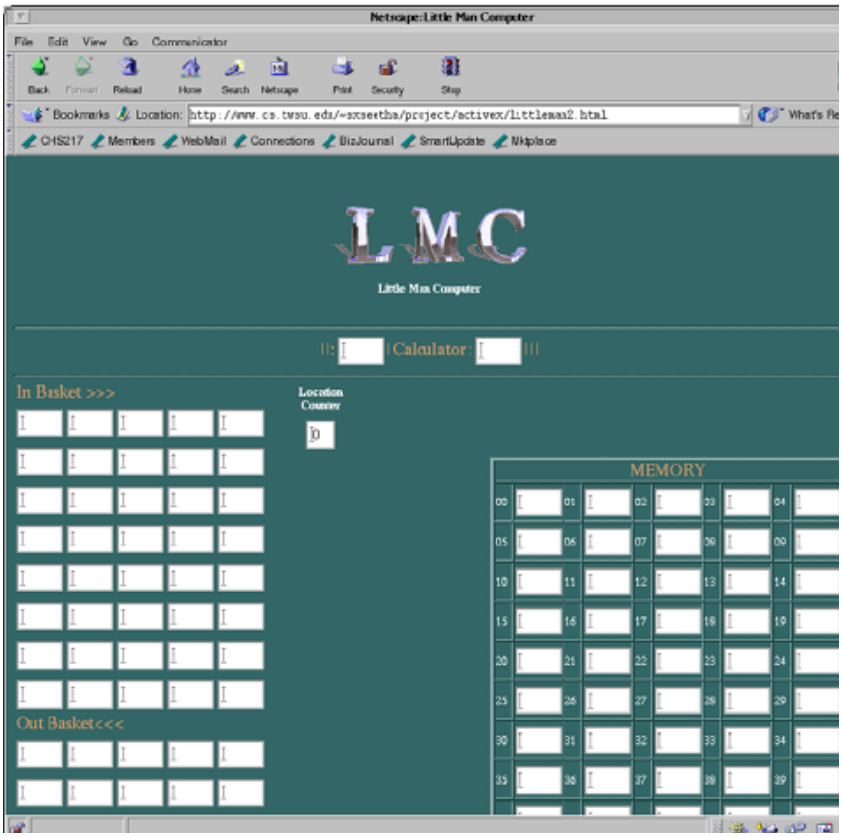


Slika 4.21 ActiveXWeb-Based LMC-1 simulator

- Vizuelizacija procesa loadovanja iz mašinskog kôda,
- Različito izvršavanja modova uključujući korak po korak (pojedinačni koraci), izborni mod, step-over (kretanje kroz kondicionalnu logiku),
- Lokaciju reseta brojača,
- LMC asemblerski programski jezik input/output za lokalni fajl system, koji dozvoljava da se sačuva program i zadrži na lokalnoj platformi.

LMC-2 zasnovan na web-u

Satianarajana Sithasridhar (Satyanarayana Seethasridhar), pod rukovodstvom profesora M. Dadašeda (Dadashzadeh) sa Wichita državnog univerziteta, razviola je ActiveX LMC-2 simulator zasnovan na web-u, koji se pokreće na Internet Explorer-u (preferira se) ili Netscape Navigator-u. Ova simulator ne pokušava da vizuelizuje FE cikluc već vizuelizuje kalkulator, input/output boksove, lokacioni brojač, i memoriju sa jednostavnim i interaktivnim interfejsom. Inicijalni skrin simulatora je na slici 4.22.



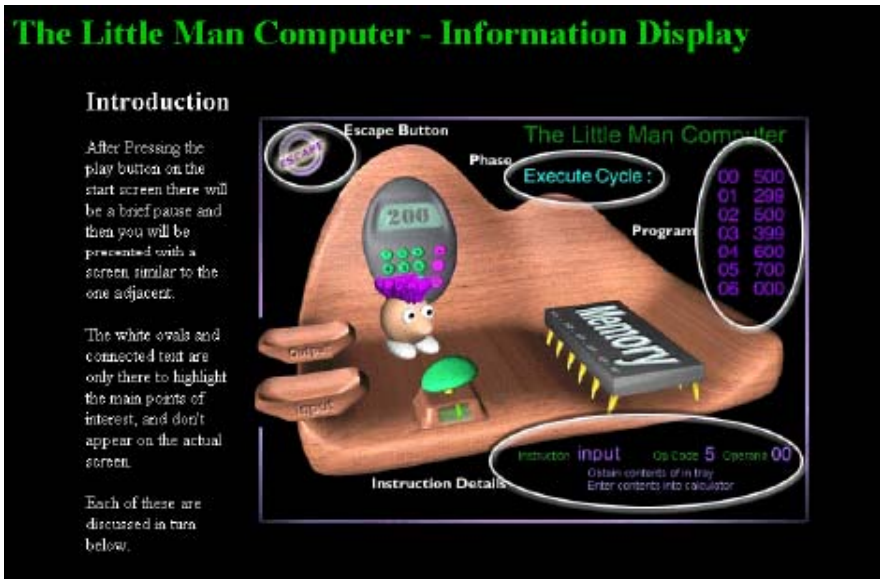
Slika 4.22 ActiveXWeb-Based LMC-2 simulator

Son-of-LMC

Alan Pink sa Alkonkvon koledža (Alqonquin college Canada) je razvio Son-of-LMC simulator za windows 95/98. Son-of-LMC je razvijen u Visual Basic-u i dat je dinamički link za biblioteke. Jedinstvenost Son-of-LMC je vizuelizacija procesa, pozivanje potprograma i povezivanje/likovanje procesa.

Shockwave LMC

Tim Univerziteta Hertfordshire (Engleska) je razvio schockwave animacionu vizuelizaciju LMC paradigme da "proveri" koliko studenti razumeju puteve koje rutinski preduzimaju. Pokazano je da studenti dobijaju potpunije razumevanje fetch ciklusa pomoću vizuelizacije individualnih komponenti i njihove instrukcije. Uvodna stranica Shockwave LMC-a je prikazana na slici 4.23.



Slika 4.23 Shockwave LMC - Informacioni diplej

Ovo poglavlje rezimira korišćenje LMC simulatora kao vizuelnog alata/sredstva za poučavanje/nastavu arhitekture računara. Naglašeno je da postoji mnoštvo različitih LMC simulatora i poredili smo neke.

Aplet simulatora LMC 1 dostupan je preko adrese: http://www.acs.ilstu.edu/faculty/javila/lmc/. [34]

5

5. ARHITEKTURA RAČUNARA

U ovoj glavi biće izloženo kako se podaci predstavljaju u računaru, sinteza arhitekture računara i konkretna specifikacija za procesor TFaCo.

5.1. PREDSTAVLJANJE PODATAKA

Podaci su osnovni elementi koje obrađuju računari. Predstavljaju se u diskretnom obliku (pomoću nula i jedinica) i u tom obliku nad njima se primenjuju različite aritmetičko – logičke operacije.

5.1.1. Obrada podataka

Podaci se mogu posmatrati kao nešto što objektivno postoji u prirodi, jer predstavljaju registrovane činjenice o nekom objektu posmatranja, dok je informacija subjektivna, jer postoji samo u odnosu na primaoca kome saopštava nešto novo. U procesu donošenja odluka na osnovu dobijenih informacija primalac preduzima određene upravljačke akcije. [40]

Podaci su činjenice, pojmovi ili događaji predstavljeni (zapisani, registrovani) na unapred dogovoren, formalizovan način. Formalizovan način predstavljanja je neki standardizovan, uobičajeni način koji se primenjuje u radu sa podacima. Predstavljeni podaci imaju dva osnovna oblika:

- analogni podaci i
- diskretni podaci.

Podatak je predstavljen u analognom obliku ako je zadat pomoću fizičke veličine koja se menja neprekidno (kontinualno), a čija vrednost funkcionalno zavisi od podataka.

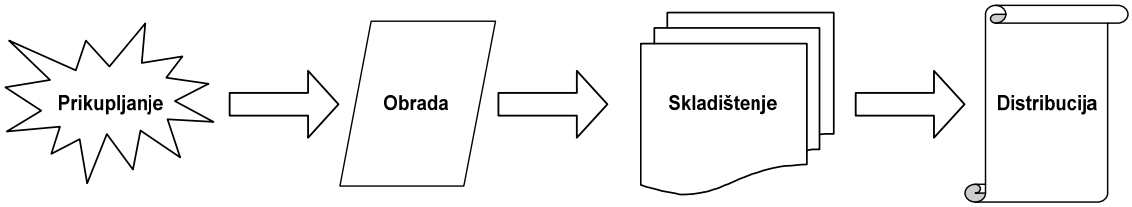
Podatak je predstavljen u digitalnom obliku ako je zadat pomoću prekidnih fizičkih veličina koje imaju određene, oštro odvojene vrednosti.

Informacija (u obradi podataka) predstavlja smisao dodeljen podacima na osnovu dogovora za njihovo predstavljanje.

Sistematska primena operacija nad podacima naziva se obrada podataka ili obrada informacija. Obradom se vrši transformacija (pretvaranje) podataka i informacija u oblik koji će moći da se koristi za određenu namenu. Ako se obrada podataka vrši automatskim sredstvima, pre svega računarima, ona se naziva automatska obrada podataka (AOP).

U procesu obrade podataka vrši se:

- prikupljanje,
- obrada,
- skladištenje (pamćenje) i
- dostavljanje podataka radi njihovog korišćenja (distribucija).



Dijagram 5.1 Proces obrade podataka

Ulaz se sastoji u prikupljanju i registrovanju podataka radi dalje obrade. Obrada ima osnovni cilj da ulazne podatke transformiše tako da se dobije informacija koja se može koristiti za predviđene namene. Etapa dostavljanja ili komunikacije sastoji se u pripremi čitljivih dokumenata (izveštaja, pregleda, tabela,...), njihovom prenosu i dostavljanju korisnicima. Uneti podaci, kao i podaci i informacije dobijeni obradom, pamte se na masovnim memorijama radi daljeg korišćenja, arhiviranja, reprodukcije ili pripreme za kasnije nove obrade. Na osnovu primljene informacije korisnik (čovek ili mašina) donosi odluke i preduzima određene akcije. Odzivom objekta na te akcije nastaje generisanje novih podataka koji se ponovo uključuju u proces obrade.

Uređaji za obradu podataka - obavljaju operacije sa podacima (ručni, poluautomatski ili automatski). Automat (mašina) je uređaj koji bez neposrednog učešća čoveka izvršava određene operacije. Programski upravljana mašina ili mašina sa programskim upravljanjem funkcioniše prema unapred zadatom programu. Program je sastavljen od konačnog skupa instrukcija ili naredbi, pri čemu svaka instrukcija opisuje elementarnu operaciju koju automatski uređaj može direktno da izvrši.

5.1.2. Diskretno predstavljanje podataka

Radi predstavljanja podataka dogovorno se usvaja skup znakova ili apstraktna azbuka -konačan neprazan skup različitih elemenata (mala i velika slova azbuke, decimalne cifre, znaci interpukcije, specijalni znaci,...). Konačan broj redom napisanih znakova iz skupa znakova naziva se niska, niz znakova ili reč nad tim skupom znakova. Broj znakova niske naziva se dužina niske.

Da bi se podaci mogli obrađivati, moraju se predstaviti primenom odgovarajućeg postupka. Nad usvojenim skupom znakova formira se potreban broj niski i svakom elementu stavi se jednoznačno u vezu po jedna niska. Postupak uspostavljanja ove veze naziva se kodiranje ili kodovanje, a skup pravila kojima se opisuje način predstavljanja naziva se kôd. Kodom se opisuje veza između svakog elementa polaznog skupa i određene kodne reči. Ovakvo predstavljanje podataka naziva se takođe diskretno ili azbučno predstavljanje.

U računarskim sistemima podaci se prikazuju signalima (naponima ili strujama) konačnih vrednosti, a obrada podataka se izvodi uređajima koji pravilno rade sa ograničenim brojem konačnih stanja. Nepostojanje elemenata koji bi pouzdano radili sa više od dva konačna stanja dovelo je do najšire primene binarnih elemenata, tj. elemenata koji prikazuju signale i stanja pomoću dve vrednosti, tj. za predstavljanje podataka i informacija isključivo se koristi azbuka od dva elementa. Ova azbuka se naziva binarna azbuka, a njeni elementi se označavaju znacima " 0 " i " 1 ". Kodiranje podataka rečima binarne azbuke naziva se binarno kodiranje, a odgovarajući kôd binarni kôd.

Svi podaci, bez obzira šta znače, zapisuju se u računaru u obliku brojeva, i to brojeva binarnog brojevnog sistema. Binarni brojevi se upotrebljavaju zbog toga što je za njihov prikaz dovoljno imati elemente sa samo dva stanja.

Ako su podaci zadati pomoću brojnih vrednosti kaže se da su to brojčani, brojni ili numerički podaci. Za zapisivanje brojeva koriste se brojni sistemi - skupovi znakova i pravila njihovog korišćenja za predstavljanje brojeva. Najčešće korišćeni brojni sistemi za predstavljanje podataka u računarima su:

- Binarni brojni sistem je brojni sistem sa osnovom q = 2 (koristi samo "0" i "1"). Broj napisan u ovom brojnom sistemu naziva se binarni broj.
- Oktalni brojni sistem je brojni sistem sa osnovom q = 8 (koriste se cifre 0, 1, ..., 7, koje imaju iste vrednosti kao i u decimalnom brojnom sistemu).
- Heksadecimalni brojni sistem je brojni sistem sa osnovom q = 16. Potrebno je da postoji 16 različitih znakova za cifre, tako da se koriste cifre od 0 do 9, a kao znaci

za cifre sa količinskim ekvivalentima 10, 11, ..., 15 koriste se velika slova A, B, ...,F.

5.1.3. Predstavljanje celih brojeva

U ovom radu biće korišćeni celi brojevi za potrebu simulacije rada procesora.

Celi brojevi zapisuju u *n*-bitnoj reči u obliku binarnog broja $a_{n-1}\ a_{n-2}...a_1a_0$, na sledeći način: [36], [37]

Tabela 5.1 Zapis celog broja sa n binarnih cifara

Binarna cifra	a_{n-1}	a_{n-2}			a_1	a_0.
Pozicija	n-1	n-2	…	…	1	0

5.1.3.1. Neoznačeni i označeni brojevi

Broj čiji zapis ne sadrži znak se naziva *neoznačen*. Zapis neoznačenih celih brojeva je identičan njihovoj reprezentaciji u binarnom brojnom sistemu. Ako je *A* neoznačen ceo (decimalni) broj zapisan u obliku binarnog broja $a_{n-1}a_{n-2}...a_1a_0$ tada važi $A \in [0, 2^n\text{-}1]$.

Ako se niz od *n* uzastopnih binarnih cifara $a_{n-1}a_{n-2}...a_1a_0$ interpretira kao neoznačeni ceo broj A, tada je njegova decimalna vrednost

$$A = \sum_{i=0}^{n-1} 2^i a_i \tag{5.1}$$

Tabela 5.2 Primeri zapisa nekih neoznačenih brojeva

Binarni zapis	**Decimalna vrednost**
0000000000000000	0
0000000000000001	1
0000000000001111	15
0000000000100000	32
0001000000001111	4111
1000000000000000	32768
1111111111111111	65535
11111111111111111111111111111111	4294967295

Označeni brojevi se mogu napisati na sledeće načine:

- **Zapis pomoću znaka i apsolutne vrednosti -** u ovom zapisu cifra najveće težine označava znak broja, dok ostale cifre predstavljaju apsolutnu vrednost broja. Uobičajeno je da najmanja cifra brojčanog sistema označava pozitivne, a najveća negativne brojeve.
- **Zapis uz korišćenje *komplementa* broja -** komplement se takođe koristi i za konverziju iz pozitivnih u negativne brojeve i obrnuto. Neka je *XN* pozitivan broj zapisan u brojčanom sistemu sa osnovom *N* pomoću *n* cifara, pri čemu je znak

broja zapisan na mestu najveće težine pomoću najmanje cifre brojčanog sistema. Broj - *XN* se može zapisati pomoću:

- N-1-og komplementa koji se dobija tako što se svaka cifra u zapisu broja XN oduzme od N -1,
- N-tog komplementa ili komplementa u odnosu na osnovu sistema (radix komplement) koji se dobija tako što se na zapis broja u N-1-om komplementu doda 1 na poziciju najmanje težine. Radix komplement broja XN se može izračunati i direktno, bez određivanja N -1-og komplementa, kao razlika Nn -X. Pri tome se oduzimanje izvodi u brojčanom sistemu sa osnovom N.
- Zapisom uz dodavanje uvećanja koji se dobija dodavanjem uvećanja k na N-ti komplement broja. Pojava eventualnih prenosa pri sabiranju se ignoriše. Ovaj način zapisa se još naziva i zapis u kodu višak k.

Primeri zapisa pozitivnih i negativnih brojeva u različitim brojčanim sistemima su prikazani u tabeli 5.3.

Tabela 5.3 Zapis označenih celih brojeva u različitim brojčanim sistemima

Broj	Znak i apsolutna vrednost	*N*−1-vi komplement	*N*-ti komplement	Uvećanje za $(6)_{10}$
$(+127)_{10}$	0127	0127	0127	0133
$(-127)_{10}$	9127	9872	9873	9879
$(+64)_8$	064	064	064	072
$(-64)_8$	764	713	714	722
$(+AB)_{16}$	0AB	0AB	0AB	0B1
$(-AB)_{16}$	FAB	F54	F55	F5B
$(+101)_2$	0101	0101	0101	1011
$(-101)_2$	1101	1010	1011	0001

U *n*-bitnoj reči krajnje levi bit označava znak, a ostalih *n*-1 bitova apsolutnu vrednost broja. Vrednost broja *A* je

$$A_{ZA} = (-1)^{a_{n-1}} \sum_{i=0}^{n-2} 2^i a_i \tag{5.2}$$

Ako se niz od *n* uzastopnih binarnih cifara $a_{n-1}a_{n-2}...a_1a_0$ interpretira kao ceo broj A zapisan pomoću znaka i apsolutne vrednosti, tada je njegova decimalna vrednost $A \in [-2^{n-1}+1, 2^{n-1}-1]$.

5.1.3.2. Prvi komplement

Broj A u prvom komplementu[15] (one's complement) se zapisuje na sledeći način:

- Krajnje levi bit označava znak broja

[15] U pojedinim literaturama se za I (prvi) komplement koristi termin nepotpuni komplement, dok se za II (drugi) komplement koristi termin potpuni komplement.

- Ostalih *n*-1 bitova se zapisuje:
  - za pozitivne brojeve kao apsolutna vrednost broja, i
  - za negativne brojeve kada se u zapisu apsolutne vrednosti broja A (bez znaka broja) svaka cifra zameni njenim komplementom do najveće cifre brojnog sistema.

Vrednost broja $A = a_{n-1}a_{n-2}...a_1a_0$ zapisanog u binarnom sistemu u prvom komplementu je u decimalnom sistemu data pomoću zbira

$$A_{ZA} = (-1)^{a_{n-1}} \sum_{i=0}^{n-2} 2^i a_i \tag{5.3}$$

Ako se niz od *n* uzastopnih binarnih cifara $a_{n-1}a_{n-2}...a_1a_0$ interpretira kao ceo broj A zapisan u prvom komplementu, za njegovu decimalnu vrednost važi $A \in [-2^{n-1}+1, 2^{n-1}-1]$.

5.1.3.4. Drugi komplement

Broj A u drugom komplementu (two's complement) se zapisuje na sledeći način:

- Krajnje levi bit u *n*-bitnoj reči označava znak broja, a ostalih *n*-1 bitova označavaju vrednost broja.
- Vrednost broja *A* se zapisuje na sledeći način:
  - za pozitivne brojeve kao apsolutna vrednost tog broja, i
  - za negativne brojeve kao broj koji se dobija kada se na zapis broja A (bez znaka broja) u prvom komplementu doda jedinica na mesto najmanje težine.

Vrednost broja A zapisanog u binarnom sistemu u drugom komplementu je u decimalnom sistemu

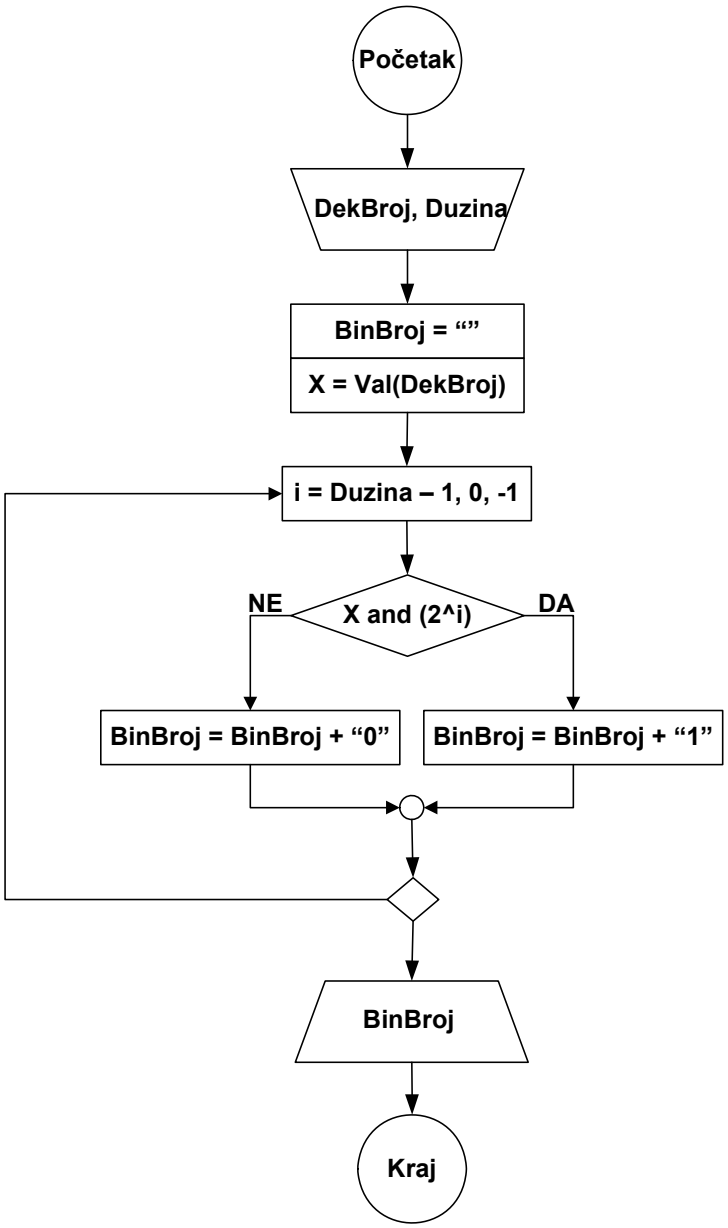
$$A_{PK} = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i \tag{5.4}$$

Ako se niz od *n* uzastopnih binarnih cifara $a_{n-1}a_{n-2}...a_1a_0$ interpretira kao ceo broj A zapisan u drugom komplementu, tada za njegovu decimalnu vrednost važi $A \in [-2^{n-1}, 2^{n-1}-1]$.

Tako, npr. za n = 32 važi:

$-2^{31} \le x \le +2^{31}-1$, односно
$-2147483648 \le x \le +2147483647$

Prevođenje zapisa celih brojeva iz decimalnog sistema u drugi komplement data je blok dijagramom 5.1.



Blok dijagram 5.1 Prevođenje celih decimalnih brojeva u drugi komplement

5.1.3.4.1. Konverzija između zapisa različitih dužina

Neka je ceo broj $A = a_{n-1}a_n2...a_1a_0$ zapisan u binarnoj reči dužine n i neka ga treba upisati u binarnu reč dužine m.

- Ako je m < n upisivanje nije moguće izvesti korektno zbog mogućeg gubitka značajnih cifara.
- Ako je m = n konverzija ne postoji.
- Ako je m > n tada se upisuje a_{n-1} na sve pozicije i u zapisu broja, gde važi $n \leq i < m$.

5.1.4. Celobrojna aritmetika

Izvršavanje aritmetičkih operacija u računaru izvršava se kroz promenu znaka, sabiranje, oduzimanje, množenje i deljenje.

5.1.4.1. Promena znaka

U zavisnosti kako je broj zapisan za promenu znaka imamo sledeće slučajeve: [37]

- Znak i apsolutna vrednost – kod ovog slučaja komplementira se bit za znak broja (tabela 5.4)

Tabela 5.4 Primer promene znaka

Decimalna vrednost	Binarni zapis
+9	00001001
-5	10000101

- Prvi komplement – kod ovog slučaja vrši se komplementiranje svake cifre u binarnom zapisu broja, uključujući i mesto za znak (tabela 5.5).

Tabela 5.5 Primer promene znaka

Decimalna vrednost	Binarni zapis
+9	00001001
-5	11111010

- Drugi komplement – kod ovog slučaja vrši se promena znaka broja u dva koraka:

  1. U prvom se izvrši komplementiranje svake cifre do najveće cifre brojnog sistema (u binarnom sistemu do jedinice), uključujući i mesto za znak.

  2. U drugom se dobijeni broj sabere sa jedinicom, pri čemu se sabiranje obavlja po pravilima za sabiranje neoznačenih brojeva (tabela 5.6).

Tabela 5.6 Primer promene znaka

Decimalna vrednost		Binarni zapis	Koraci
+9	=	00001001	drugi komplement
		11110110	1. korak
		+00000001	2. korak
-9		11110111	rezultat
-5	=	11111011	drugi komplement
		00000100	1. korak
		+00000001	2. korak
+5	=	00000101	rezultat

5.1.4.2. Sabiranje i oduzimanje brojeva u drugom komplementu

Prilikom ovih aritmetičkih operacija može doći do prekoračenja, tako npr. ako se kao rezultat operacije sabiranja brojeva *A* i *B* koji su zapisani sa po *n* cifara dobije broj *C* za čiji je tačan zapis potrebna *n* +1 cifra tada se kaže da je došlo do prekoračenja pri izvođenju operacije. U računaru se prekoračenje otkriva upotrebom modifikovanog oblika broja. [41], [11]

$$\begin{aligned} A &= a_n a_{n-1} a_{n-2} \ldots a_1 a_0 \\ B &= b_n b_{n-1} b_{n-2} \ldots b_1 b_0 \\ C &= c_n c_{n-1} c_{n-2} \ldots c_1 c_0 \end{aligned} \tag{5.5}$$

Ako se **sabiraju** dva broja istog znaka (bilo oni pozitivni ili negativni), a kao rezultat se dobija zbir suprotnog znaka onda je došlo do prekoračenja.

Neka su dati brojevi $A = a_{n-1}a_{n-2}\ldots a_1 a_0$ i $B = b_{n-1}b_{n-2}\ldots b_1 b_0$ (u drugom komplementu). Tada se izračunavanje njihovog zbira vrši u dva koraka:

- Označi se međurezultat koji se dobija sabiranjem *A* i *B* sa *C'*:

A	=		a_{n-1}	a_{n-2}	...	a_1 a_0
B	=		b_{n-1}	b_{n-2}	...	b_1 b_0
C'	=	c'_n	c'_{n-1}	c'_{n-2}	...	c'_1 c'_0

- Konačan rezultat $C = A + B$ se dobija uklanjanjem c'_n iz međurezultata C' i proverom pojave prekoračenja.

Postupak **oduzimanja** dva broja (A, B) C = A – B se svodi na sabiranje uz promenu znaka drugom operandu: C = A+(-B).

Sledeći primeri pokazuju različite situacije sa i bez prekoračenja (tabela 5.7).

Tabela 5.7 Primeri sabiranja/oduzimanja

c) +14 + 10

A	= +14	=		00001110
B	= +10	=		00001010
C'		=	0	00011000
C	= +24	=		00011000

a) +127 -10

A	= +127	=		01111111
B	= -10	=		11110101
C'		=	1	01110100
C	= +117	=		01110101

d) +100 + 65

A	= +100	=		01100100
B	= +65	=		01000001
C'		=	0	10100101
C	= ***	=		10100101

Prekoračenje, jer se sabiranjem dva pozitivna dobija negativan broj.

b) -100-(+65)=-100+(-65)

A	= -100	=		10011100
B	= -65	=		10111111
C'		=	1	01011011
C	= ***	=		01011011

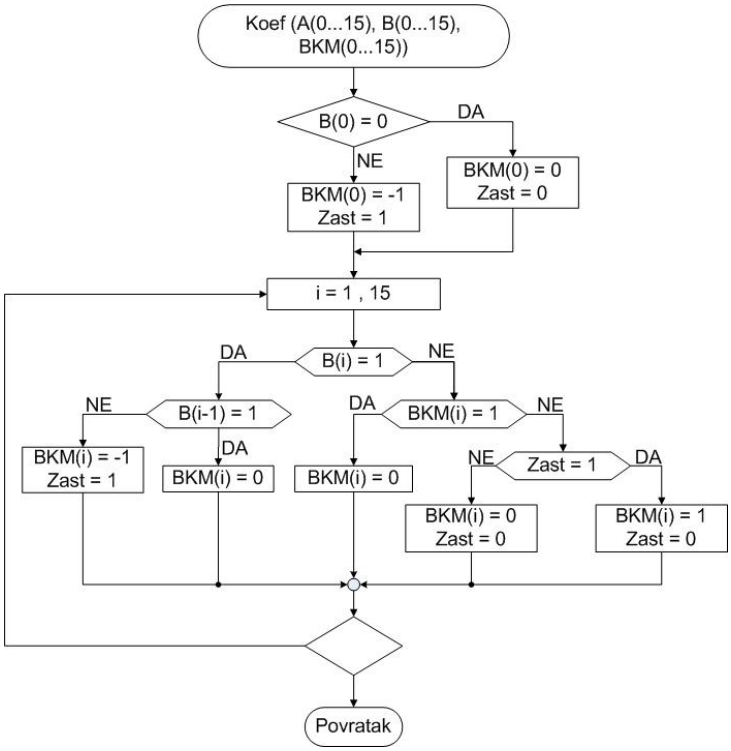
Prekoračenje, jer se sabiranjem dva negativna dobija pozitivan broj.

5.1.4.3. Množenje brojeva u drugom komplementu

Klasični način množenja označenih brojeva podrazumeva da se množenik i množilac pretvaraju u pozitivne brojeve pamteći njihove originalne predznake. Kako su brojevi 16-bitni, tada algoritam za množenje treba da uradi 31-nu iteraciju ostavljajući bite predznaka neobuhvaćene izračunavanjem, a rezultat treba da bude negativan ukoliko su predznaci množenika i množioca različiti ili da bude pozitivan, ukoliko su predznaci isti.

Elegantniji pristup množenju označenih brojeva je primenom modifikovanog Booth-ovog algoritma. Ovaj algoritam polazi od zapažanja da, uz korišćenje mogućnosti i sabiranja i oduzimanja, postoji više načina da se izračuna proizvod. On se može realizovati kroz sledeće korake: [37]

- Najpre se formira Booth-ov Kodirani Množilac (BKM) tako što se pretražuje originalni množilac zdesna ulevo i upisuje se -1 na svakoj poziciji gde je 1 na početku niske (niska se smatra celina od trenutnog položaja pa do krajnje leve pozicije množioca), +1 kada se naiđe na prvu sledeću 0, dok se na ostalim mestima upisuje 0. Već u ovom koraku vidi se uspešnost optimizacije (pojava -1 znači da se primenjuje oduzimanje, +1 da se primenjuje sabiranje, a 0 da nema akcije). Blok dijagram 5.2 se koristi za određivanje BKM-a za operande A i B.

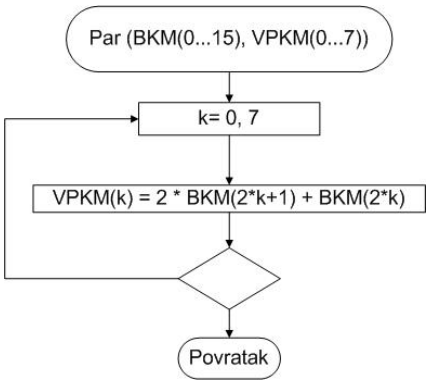


Blok dijagram 5.2 Određivanje Booth-ovog Kodiranog Množioca

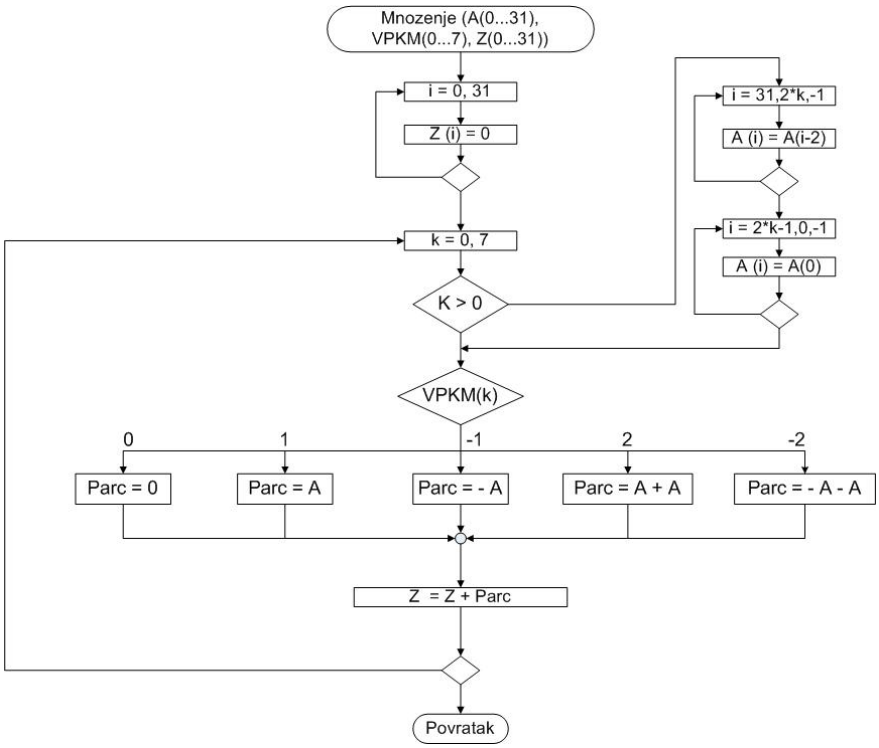
- Zatim se izdvajaju parovi oblika (a_{2k+1}, b_{2k}) iz Booth-ovog Kodiranog Množioca, gde su u indeksu označene pozicije na kojima se nalaze vrednosti a i b, pri čemu važi $k \in [0, n/2-1]$, a n je dužina operanda u bitima. Težina a_{2k+1} je dvostruko veća

od težine b_{2k}. Na osnovu težine se određuje zajednička vrednost svakog para korišćenjem blok dijagrama 5.3.

- Za svaki par koji se pojavi u kodiranom množiocu izvrši se pomeranje množenika za 2k mesta ulevo. Tako dobijena vrednost se množi sa vrednošću para i dodaje se proizvodu Z (blok dijagram 5.4.).



Blok dijagram 5.3 Određivanje Vrednosti Parova Booth-ovog Kodiranog Množioca (VPKM)



Blok dijagram 5.4 Određivanje parcijalnih zbirova i konačnog proizvoda (Z)

Sam proces množenja binarnih brojeva korišćenjem modifikovanog Booth-ovog algoritma se odvija kroz više koraka, pri čemu svaki od njih daje mogućnost da korisnik jasno vidi šta se sa unetim podacima dešava sve do dobijanja krajnjeg rezultata. Koraci koji se jasno mogu identifikovati su sledeći :

1. Unos 16-bitnih binarnih brojeva (operandi A i B);
2. Nalaženje Booth-ovog Kodiranog Množioca (BKM);

3. Nalaženje Vrednosti Parova Kodiranog Množioca (VPKM);
4. Dopunjavanje operanda A bitima 0 (ako je pozitivan) ili 1 (ako je negativan) (jer se vrednost proizvoda Z zapisuje 32-bitnim brojem);
5. Pomeranje množenika za 2k mesta ulevo i množenje sa VPKM-om, kreiranje parcijalnih proizvoda;
6. Prikaz krajnjeg rezultata - proizvoda Z.

Kao primer su izabrani negativni brojevi A = -46, i B = -57. Operandi se zapisuju kao 16-bitni označeni binarni brojevi, pri čemu se za predznak operanda unosi "+" za pozitivne, predznak "-" za negativne brojeve na najtežoj poziciji. U predviđena polja (od 1 do 15 manje težine) moguć je jedino unos cifara 0 ili 1, dok u najteže polje moguć je jedino unos predznaka "+" ili "-". Tako, za napred navedene decimalne vrednosti binarni zapis opernada A i B bio bi: A = -101110, B = -111001.

5.1.4.4. Deljenje brojeva u drugom komplementu

Implementacija deljenja neoznačenih brojeva koristi tri registra *P, A* i *M* i brojač čija je inicijalna vrednost jednaka broju bitova u registrima. Inicijalno se u registar *P* upisuje deljenik, u registar *M* delilac, a u registar *A* nula. Deljenje se vrši na sledeći način:

1. U prvom koraku se pomera se sadržaj registara *A* i *P* (posmatranih kao jedna binarna reč) ulevo za jedan bit. [37]
2. Sadržaj registra *M* se oduzima od sadržaja registra *A* i dobijena vrednost se upisuje u regist. *A*.
3. Ispituje se da li je vrednost u registru $A \geq 0$, tj. da li može da se izvrši oduzimanje i dobije delimični ostatak. Ako može, tada se u bit najmanje težine registra *P* upisuje 1. U suprotnom se u bit najmanje težine registra *P* upisuje 0 i sadržaj registra *M* se sabira sa sadržajem registra *A* radi restauracije prethodnog sadržaja registra *A*.
4. Vrednost brojača se smanjuje za 1; ukoliko je vrednost brojača veća od nule, izvršavanje se vraća na korak 1.
5. Na kraju deljenja količnik se nalazi u registru *P*, ostatak u registru *A* (slika 5.1).

Slika 5.1 Primer deljenja neoznačenih celih brojeva

Što se tiče deljenja brojeva u drugom komplementu potrebno je uraditi sledeće:

- Na početku se u registre A i P upisuje deljenik posmatran kao broj u drugom komplementu dužine $2n$. Delilac se upisuje u registar M. Sadržaj registara A i P se pomera (aritmetičkim pomeranjem) za jedno mesto ulevo. Ako M i A imaju isti znak tada se izvršava operacija oduzimanja: $A = A - M$; u suprotnom se izvršava sabiranje $A = A + M$. Izvršena operacija se smatra uspešnom ako je znak registra A nepromenjen posle njenog izvršavanja.
- Ako je operacija uspešna ili ($A = 0 \wedge P = 0$) tada se bit najmanje težine registra P postavlja na 1. U suprotnom (ako je operacija neuspešna i ($A \neq 0 \vee P \neq 0$)) bit najmanje težine registra P se postavlja na 0 i restaurira se prethodna vrednost registra A.
- Prethodna dva koraka se ponavljaju n puta, gde je n dužina registra P. Na kraju procesa registar A sadrži ostatak. Ako je znak deljenika i delioca isti tada je vrednost količnika zapisana u registru P. U suprotnom, za vrednost količnika treba uzeti vrednost u registru P sa promenjenim znakom.

Primer realizacije deljenja brojeva u drugom komplementu dat je u tabeli 5.8.

Tabela 5.8 Primer deljenja dva broja kod kojih je deljenik negativan broj (- 24)/9

M	A	P	Komentar
00001001	11111111	11101000	Početno stanje, -24/9
00001001	11111111	11010000	Pomeranje ulevo
	00001001		Sabiranje A ← A - M
00001001	11111111	11010000	$P_0 \leftarrow 0$, restauracija sadržaja A
00001001	11111111	10100000	Pomeranje ulevo
	00001001		Sabiranje A ← A - M
00001001	11111111	10100000	$P_0 \leftarrow 0$, restauracija sadržaja A
00001001	11111111	01000000	Pomeranje ulevo
	00001001		Sabiranje A ← A - M
00001001	11111111	01000000	$P_0 \leftarrow 0$, restauracija sadržaja A
00001001	11111110	10000000	Pomeranje ulevo
	00001001		Sabiranje A ← A - M
00001001	11111110	10000000	$P_0 \leftarrow 0$, restauracija sadržaja A
00001001	11111101	00000000	Pomeranje ulevo
	00001001		Sabiranje A ← A - M
00001001	11111101	00000000	$P_0 \leftarrow 0$, restauracija sadržaja A
00001001	11111010	00000000	Pomeranje ulevo
	00001001		Sabiranje A ← A - M
00001001	11111010	00000000	$P_0 \leftarrow 0$, restauracija sadržaja A
00001001	11110100	00000000	Pomeranje ulevo
	00001001		Sabiranje A ← A - M
00001001	11111101	00000001	$P_0 \leftarrow 1$
00001001	11111010	00000010	Pomeranje ulevo
	00001001		Sabiranje A ← A - M
00001001	11111010	00000010	$P_0 \leftarrow 0$, restauracija sadržaja A
	11111010	00000010	Znak deljenika i delioca nije isti $\Rightarrow$
			$\Rightarrow$ količnik = - P = -2, ostatak = -6

5.1.5. Logička algebra i logički sklopovi

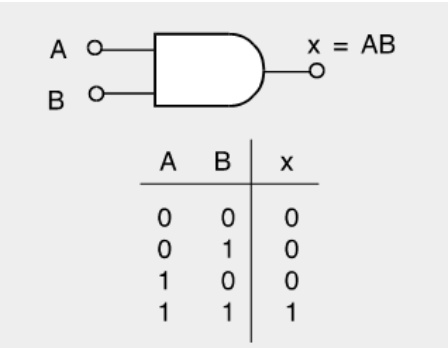
Logička ili Bulova algebra je sistem teorema koji koriste simboličku logiku da bi opisali skupove elemenata i odnose među njima. Naziv je dobila po engleskom matematičaru George Boole-u (1815.-1864.). On je u svom delu "Matematička analiza logike" želeo da obradi postupke deduktivnog logičkog razmišljanja, pri čemu ulazni podaci imaju samo dva stanja – istina ili laž. Razvojem digitalnih računara otkriveno je da je Bulova algebra vrlo dobro primenljiva u konstruisanju i analizi rada računara jer takvi računari takođe mogu da imaju samo dva stanja (uključen - isključen, tj. ima - nema napona). [39]

Znači, koliko god računar izgleda složeno, njegov rad može da se prikaže kombinacijom dva stanja binarnog brojnog sistema, tj. digitalni signali su predstavljeni binarno sa dva naponska, odnosno logička, nivoa. Nad takvim signalima mogu da se izvode razne operacije koje se nazivaju logičke operacije ili logičke funkcije. Ovaj naziv potiče iz matematičke discipline koja se naziva matematička logika, gde se rezultati logičkog razmišljanja iskazuju sa dva iskaza: tačno i pogrešno.

U Bulovoj algebri definisane su tri osnovne operacije nad logičkim promenljivama. To su I operacija (engl. AND), koja se označava simbolom ”.”, ILI operacija (engl. OR), koja se označava simbolom ”+” i NE operacija (engl. NOT) ili komplementiranje, koja se označava crticom iznad simbola promenljive ”—”. I i ILI operacija se izvode nad najmanje dve promenljive, dok je NE operacija unarna, tj. izvodi se nad jednom promenljivom. [38]

5.1.5.1. I operacija (logičko množenje)

Posmatrajmo prvo I funkciju dve logičke promenljive A i B. Rezultat I operacije najčešće se prikazuje u vidu tzv. kombinacione tablice ili tablice istinitosti koja je prikazana na slici 5.2. Na istoj slici prikazan je i najčešće korišćeni grafički simbol za predstavljanje I operacije.



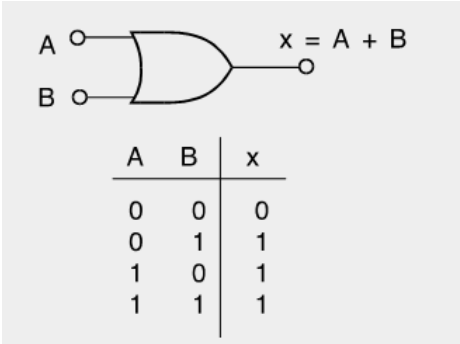
A	B	x
0	0	0
0	1	0
1	0	0
1	1	1

Slika 5.2 Kombinaciona tablica i grafički simbol za I operaciju.

Kao što se vidi, osnovna osobina I operacije nad dve promenljive je da se kao rezultat dobija logička jedinica, ako i samo ako obe promenljive imaju vrednost logičke jedinice. Zato se ponekad I operacija naziva i logičko množenje ili konjunkcija. Kolo koje realizuje I operaciju naziva se I (AND) kolo.

5.1.4.2. ILI operacija (logičko sabiranje)

ILI operacija nad dve logičke promenljive A i B prikazana je kombinacionom tablicom na slici 5.3. Na istoj slici prikazan je i najčešće korišćeni grafički simbol za predstavljanje ILI operacije. Vidi se da se kao rezultat dobija logička jedinica ako bar jedna promenljiva ima vrednost logičke jedinice. Zato se ponekad ILI operacija naziva i logičko sabiranje ili disjunkcija. Kolo koje realizuje ILI operaciju naziva se ILI (OR) kolo.

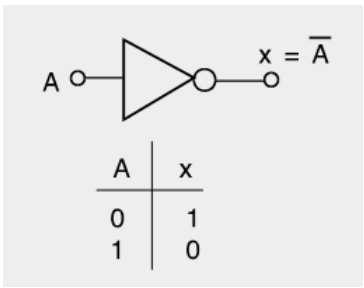


A	B	x
0	0	0
0	1	1
1	0	1
1	1	1

Slika 5.3 Kombinaciona tablica i grafički simbol za ILI operaciju.

5.1.4.3. NE operacija (komplementiranje)

Za razliku od I i ILI operacija, NE operacija se definiše nad jednom logičkom promenljivom ili izrazom. Kombinaciona tablica za NE operaciju i grafički simbol za predstavljanje kola koje obavlja NE operaciju prikazani su na slici 5.4. Često se NE operacija naziva i komplementiranje ili negacija. Kolo koje realizuje NE operaciju naziva se NE (NOT) kolo, ili još češće, invertor.



A	x
0	1
1	0

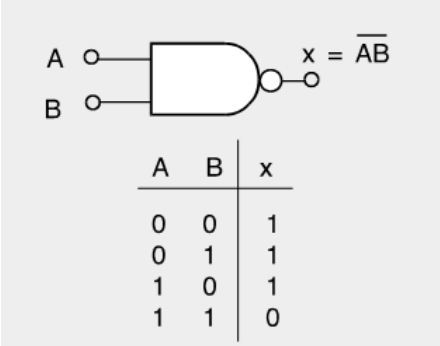
Slika 5.4 Kombinaciona tablica i grafički simbol za NE operaciju.

Kombinacijom tri osnovne logičke operacije mogu se dobiti još neke vrlo važne i korisne logičke operacije. Kombinacijom I i NE operacije dobija se NI (engl. NAND)

operacija, a kombinacijom ILI i NE operacije dobija se NILI (engl. NOR) operacija. Osim njih praktičnu primenu imaju još i operacija isključivo-ILI i operacija koincidencije.

5.1.4.4. NI operacija

Već je rečeno da se NI operacija dobija kombinacijom I i NE operacije. Prema tome, kombinaciona tablica za NI operaciju dobija se tako što se u kombinacionoj tablici za I operaciju komplementira izlazna kolona. Rezultat je prikazan na slici 5.5. Na istoj slici je prikazan i grafički simbol za NI operaciju koji je takođe kombinacija simbola za I i NE operaciju. Kolo koje realizuje NI operaciju naziva se NI (NAND) kolo.



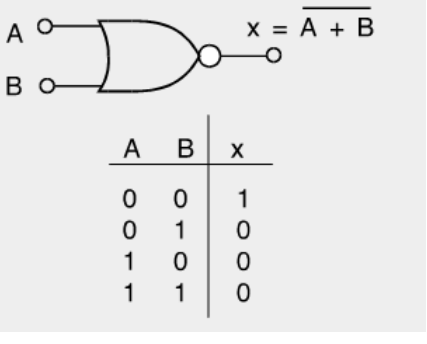
A	B	x
0	0	1
0	1	1
1	0	1
1	1	0

Slika 5.5 Kombinaciona tablica i grafički simbol za NI operaciju.

U Bulovoj algebri se može definisati tzv. potpun skup operacija. To je skup operacija pomoću kojih se može iskazati bilo koja logička funkcija. Pokazano je da takav potpun skup čine I i NE odnosno ILI i NE operacije. Dakle, NI operacija takođe čini potpun skup operacija, odnosno, proizvoljna logička funkcija se može izraziti samo pomoću NI operacije. Ova činjenica daje veliku važnost NI operaciji.

5.1.4.5. NILI operacija

NILI operacija dobijena je komplementiranjem rezultata ILI operacije. Kombinaciona tablica i grafički simbol za NILI operaciju prikazani su na slici 5.6. Treba reći da i NILI operacija predstavlja potpun skup za realizaciju logičkih funkcija. Kolo koje realizuje NILI operaciju naziva se NILI (NOR) kolo.



A	B	x
0	0	1
0	1	0
1	0	0
1	1	0

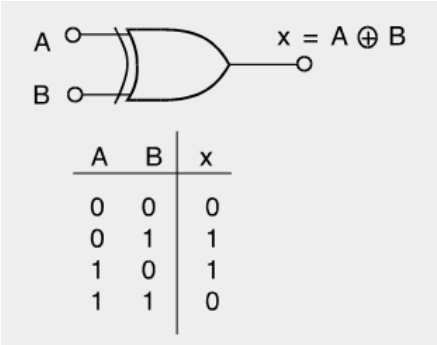
Slika 5.6 Kombinaciona tablica i grafički simbol za NILI operaciju.

5.1.4.6. Isključivo-ILI operacija

Isključivo-ILI operacija (engl. Exclusive-OR, EX-OR) razlikuje se od obične ILI operacije po tome što daje kao rezultat logičku nulu i u slučaju kada su obe promenljive logičke jedinice. Kombinaciona tablica i grafički simbol za isključivo-ILI operaciju prikazani su na slici 5.7. U jednačinama se za označavanje isključivo-ILI operacije najčešće koristi simbol "⊕". Na osnovu kombinacione tablice može se napisati logička jednačina za isključivo-ILI funkciju:

$$x = A \cdot \overline{B} + \overline{A} \cdot B = A \oplus B$$

Isključivo-ILI operaciju realizuje isključivo-ILI (EX-OR) kolo.



A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

Slika 5.7 Kombinaciona tablica i simbol za isključivo-ILI operaciju.

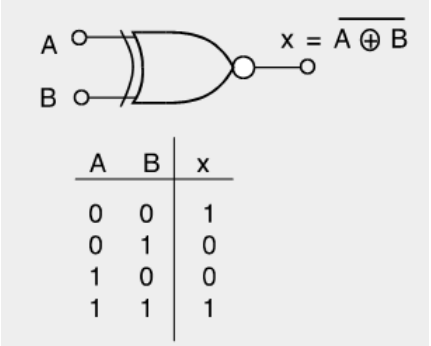
5.1.4.7. Operacija koincidencije (isključivo-NILI)

Operacija koincidencije daje kao rezultat logičku jedinicu ako su obe promenljive identične. Na osnovu toga se može napisati kombinaciona tabela koja je prikazana na slici 5.8.

Na osnovu logičke jednačine koja definiše operaciju koincidencije:

$$x = A \cdot B + \overline{A} \cdot \overline{B} = \overline{A \oplus B}$$

vidi se da je rezultat ustvari komplement isključivo-ILI operacije. Zbog toga se operacija koincidencije često naziva i isključivo-NILI operacija (engl. exclusive-NOR). Kolo koje realizuje isključivo-NILI operaciju naziva se isključivo-NILI (EX-NOR) kolo.



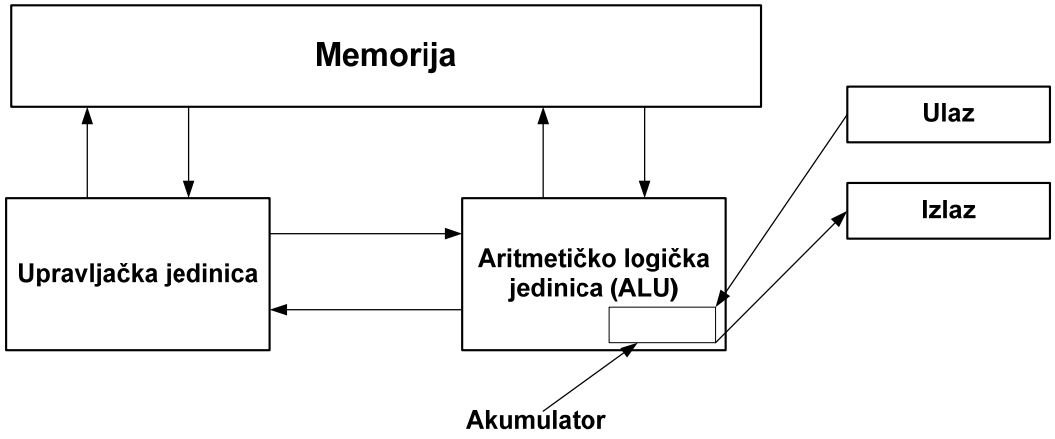
A	B	x
0	0	1
0	1	0
1	0	0
1	1	1

Slika 5.8 Kombinaciona tablica i simbol za isključivo-NILI operaciju.

5.2. SINTEZA ARHITEKTURE RAČUNARA

Problem sinteze arhitekture računara podrazumeva definisanje potrebnih aritmetičkih i logičkih operacija, realizacije ulaza i izlaza, rada sa memorijskim lokacijama, registrima i definisanje ostalih funkcija koje omogućavaju manipulaciju podacima na nivou bita.

U narednom delu ovog rada biće pojašnjeni osnovni zahtevi kreiranja arhitekture računara idući od prostijih ka složenijim sistemima.



Slika 5.9. Originalna fon Nojmanova mašina [42]

Da bi se analizirao rad arhitekture računara može se poći od ograničenog skupa instrukcija koje su prikazane u tabeli 5.9 i koje omogućavaju osnovne operacije. [21]

Tabela 5.9 Ograničen skup instrukcija

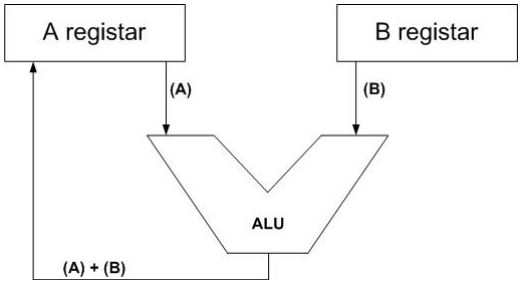
Operacija	Kôd operacije	Komentar
NOP	0000	Operacija bez efekta
ADD	0001	Sabiranje
ADC	0010	Sabiranje sa prenosom
SUB	0011	Oduzimanje
SBB	0100	Oduzimanje sa pozajmljivanjem
OR	0101	Logička ILI operacija
AND	0110	Logička I operacija
NOT	0111	Negacija
XOR	1000	Isključivo ILI operacija
IN	1001	Operacija Ulaz
OUT	1010	Operacija Izlaz
MOV	1011	Kopiranje
MVI	1100	Neposredno kopiranje
HLT	1101	Zaustavljanje
Jxx	1110	Uslovno grananje
JMP	1111	Bezuslovno grananje

5.2.1. Osnovne operacije

Na osnovu tabele 5.9 mogu se uočiti:

- Aritmetičke operacije
  - ADD, ADC, SUB i SBB
- Logičke operacije
  - OR, AND, NOT i XOR
- Operacije ulaza/izlaza
  - IN, OUT
- Operacije kopiranja
  - MOV, MVI
- i ostale operacije (NOP, HLT, Jxx, JMP)

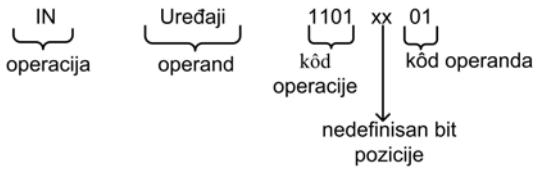
Za realizaciju **aritmetičko-logičkih operacija** (slika 5.10) potrebna su dva binarna broja nad kojim se izvršavaju ove operacije, dva registra i aritmetičko-logička jedinica (ALU) opšte namene.



Slika 5.10 Sabiranje sadržaja dva registra

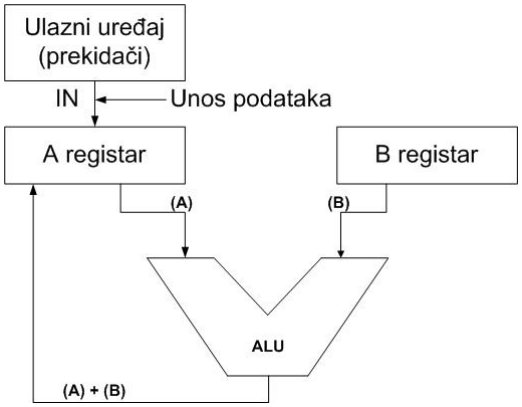
Na slici 5.10 sa (A) i (B) su označeni sadržaji registara A i B.

Da bi se podatak uneo u sistem potrebno je koristiti mašinsku instrukciju **IN**, čiji je format prikazan na slici 5.11. Instrukcija IN je 8-bitna, kôd operanda definiše adresu ulaznog uređaja, a polje bitova xx je definisano za kasniju upotrebu.



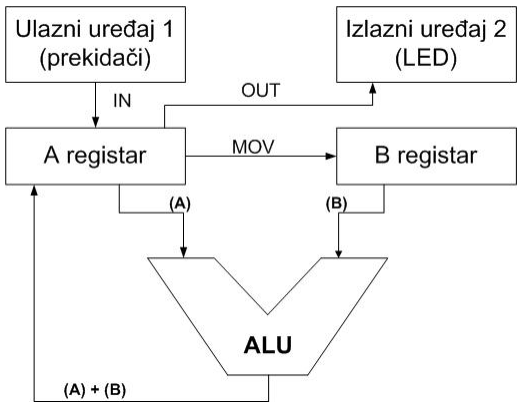
Slika 5.11 Format mašinske instrukcije IN

Neophodna hardverska struktura sistema pomoću koje se obavlja operacija IN prikazana je na slici 5.12. Registar A je određen za odredište uzlazne informacije.



Slika 5.12 Šema za unošenje podataka u sistem

Prilikom obavljanja operacije sabiranja potrebno je obezbediti mehanizam za prenos podataka u registar B (kao što je prikazano na slici 5.13). Ovakav prenos ostvaruje se operacijom **MOV** kojom se podatak premešta iz registra A u registar B.



Slika 5.13 Šema za unošenje podataka, pomeranje, sabiranje i generisanje rezultata

Prikaz programske sekvence pomoću koje je realizovan unos, pomeranje, sabiranje i prikaz dat je u tabeli 5.10.

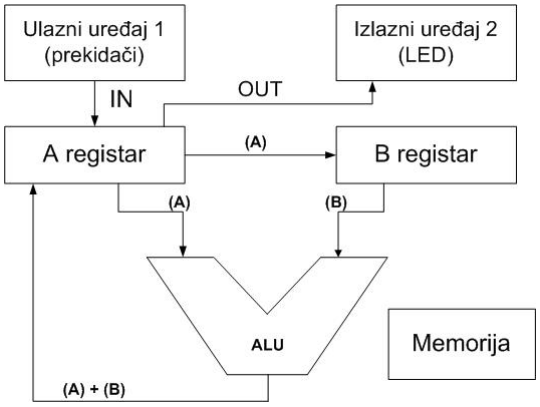
Tabela 5.10 Realizacija programske sekvence

IN 1	(A) ← (Prekidači)	Sadržaj registra A održava stanje prekidača
MOV	(B) ← (A)	B prima sadržaj A
IN 1	(A) ← (Prekidači)	
ADD	(A) ← (A) + (B)	Sadržaj A čini sadržaj B plus stari sadržaj A
OUT 2	(LED) ← (A)	Sadržaj LED čini sadržaj A

Svaki ulazno-izlazni uređaj ima svoj broj koji specificira njegovu adresu. U ovom primeru ulaznom uređaju je dodeljena adresa 1, a izlaznom adresa 2.

5.2.2. Memorija

Memorija je prostor gde se upisuju podaci i programi. (slika 5.14). Svaka memorijska lokacija ima svoju adresu, pri čemu prva memorijska lokacija počine sa adresom „0“.



Slika 5.14 Šema sistema sa memorijom

Da bi se program zapisan u tabeli 5.10 izvršio potrebno je upisati u memorijske lokacije odgovarajući sadržaj, kao što je npr. dato tabelom 5.11. [21]

Tabela 5.11 Upis programske sekvence u memoriju

Memorijska lokacija (adresa)	Sadržaj memorije	Tip operacije
0	10010001	IN 1
1	10110100	MOV B,A
2	10010001	IN 1
3	00010001	ADD A,B
4	10101000	OUT 2

U datom primeru uzeto je da važi:

- Svaka operacija je tipa bajt (osmobitna), dok je memorija bajtovski organizovana.
- Internim registrima A i B dodeljene su adrese 0 i 1. U instrukciji **MOV B, A**, registar A je izvorni, a registar B odredišni ((B) ← (A)).
- Polje zadnje četiri bit pozicije instrukcije **MOV B, A**, podeljeno je na dva jednaka dela. Pomoću dve bit pozicije u svakom delu specificira se izbor izvorišnog i odredišnog operanda. Prve dve bit pozicije, veće težine, specificiraju izbor odredišnog, a druge dve, manje težine, izbor izvornog operanda.

1 0 1 1	0 1	0 0
kôd operacije MOV	odredišni operand B	izvorni operand A

- Polje zadnje četiri bit pozicije, kod instrukcije IN, podeljeno je na dva dela. Prve dve bit pozicije, veće težine, specificiraju izbor izvornog operanda, a druge dve, manje težine, izbor odredišnog operanda.

1 0 0 1	0 0	0 1
kôd operacije IN	odredišni operand, registar A	izvorni operand, ulazni uređaj 1

- Polje zadnje četiri bit pozicije, kod instrukcije OUT, podeljeno je na dva dela. Prve dve bit pozicije, veće težine, specificiraju izbor izvornog operanda, a druge dve, manje težine, izbor odredišnog operanda.

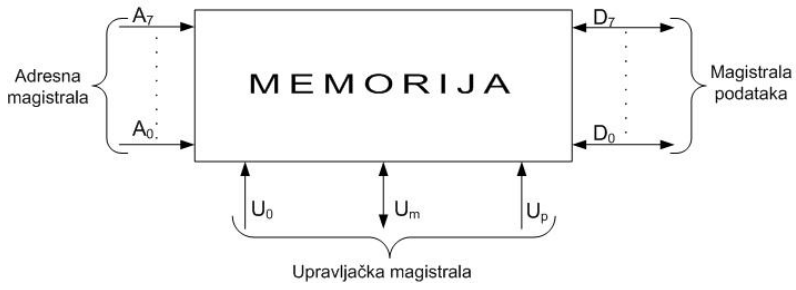
1 0 1 0	1 0	0 0
kôd operacije OUT	odredišni operand, izlazni uređaj 2	izvorni operand, registar A

Kod instrukcije IN, načinom povezivanja (hardverski) je definisano (slika 5.13) da odredište bude registar A, pa je zbog toga nepotrebno pisati **IN A, 1** jer se podrazumeva da će registar A biti odredište, nego je dovoljno samo **IN 1**.

- Kod instrukcije OUT, načinom povezivanja, je definisano da izvorni registar bude A, a da se odredište možde specificirati. Zbog toga suvišno bi bilo pisati **OUT 2, A**, dovoljno je samo **OUT 2**.
- Kod instrukcije **ADD A, B** ((A) ← (A) + (B)), registri A i B u prvoj fazi izvršavanja instrukcije sabiranja si izvorišni, a nakon dobijanja rezultata registar A je odredišni

0 0 0 1	0 0	0 1
kôd operacije ADD	odredišni operand	izvorni operand

Da bi se objasnio način rada memorije potrebno je uvesti pojmove: adresna magistrala, magistrala podataka i upravljačka magistrala. Magistralu predstavlja skup veza (linija) po kojima se prenosi informacija. Uloga i tip informacije koja se prenosi po određenoj liniji na magistrali je unapred definisan. Memorija se povezuje na način kao što je prikazano na slici 5.15.



Slika 5.15 Povezivanja memorije

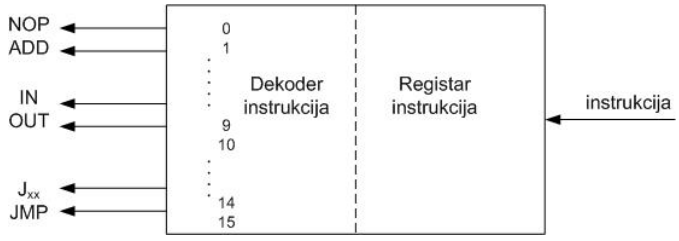
Na bazi slike 5.15 može se uočiti sledeće:

- Prenos signala po adresnoj magistrali vrši se u jednom smeru; procesor → memorija. Broj adresnih linija je u direktnoj vezi sa brojem lokacija koje se mogu selektovati.
- Magistrala podataka je dvosmerna, što znači da je u memoriju moguće upisivati podatke ili iz nje čitati.

- Smer prenosa podataka na magistrali za podatke, nakon što je izvršen izbor adresirane lokacije, određen je stanjem signala na upravljačkoj magistrali. Neke od linija na upravljačkoj magistrali se koriste za jednosmerni prenos, a neke za dvosmerni.

5.2.3. Registar instrukcija i dekoder

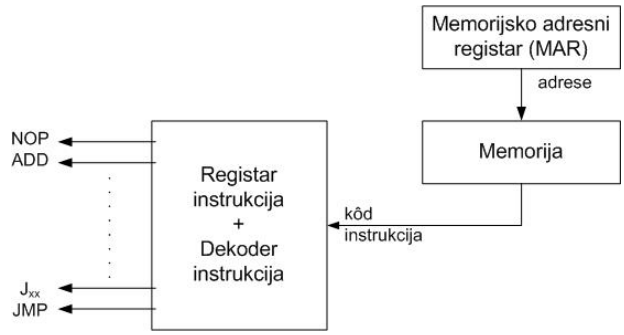
Postoji mehanizam pomoću koga se instrukcije iz memorije prenose u registar instrukcija. Njemu se dodeljuje dekoder instrukcija i oni zajedno čine „mozak“ centralne procesorske jedinice (CPU-a). Slika 5.16 [21] prikazuje blok koga čine registar instrukcija i dekoder instrukcija. Registar instrukcija prihvata instrukciju, a dekoder instrukcija u zavisnosti od sadržaja na njegovim ulazima aktivira odgovarajući izlaz. Svaki dekodirani izlaz iz dekodera instrukcija, odgovara jednoj od instrukcija definisanih u tabeli 5.9.



Slika 5.16 Registar instrukcija i dekoder

5.2.4. Memorijsko adresni registar

Tokom pristupa memorijskim lokacijama potrebno je znati njihove adrese. Na ovim adresama smeštene su instrukcije a zadatak za čuvanje memorijskih adresa ima Memorijsko Adresni Registar (MAR), koji je prikazan na slici 5.17. [21]

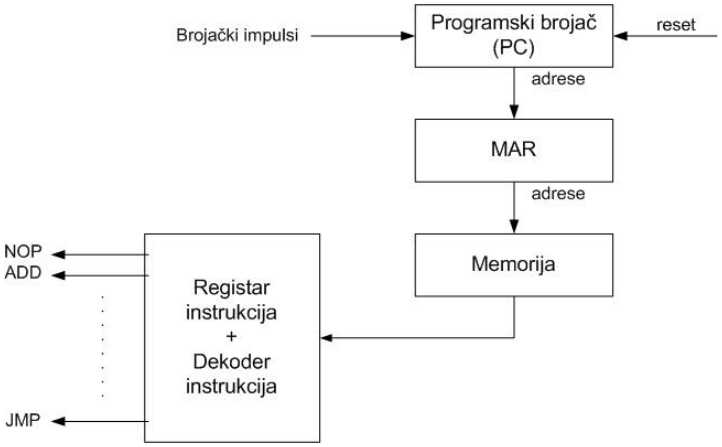


Slika 5.17 Mesto i uloga MAR-a

5.2.5. Programski brojač

Da bi se izvršio odgovarajući program koji se sastoji od progamske sekvence (npr. programska sekvenca iz tabele 5.11potebno je slati odgovarajuće komande jednu za drugom. Ova aktivnost liči na brojanje pa se u sistem uključuje i blok koji se naziva programski brojač (slika 5.18). Ovom brojaču se dovode impulsi za, plus reset impuls koji

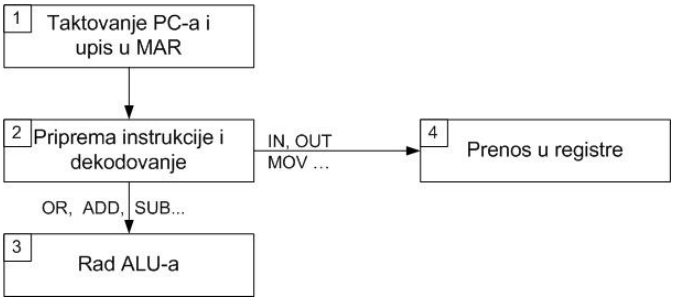
omogućava restartovanje programa od početka, tj. brojanje od memorijske lokacije sa adresom „0“. [21]



Slika 5.18 Uloga programskog brojača

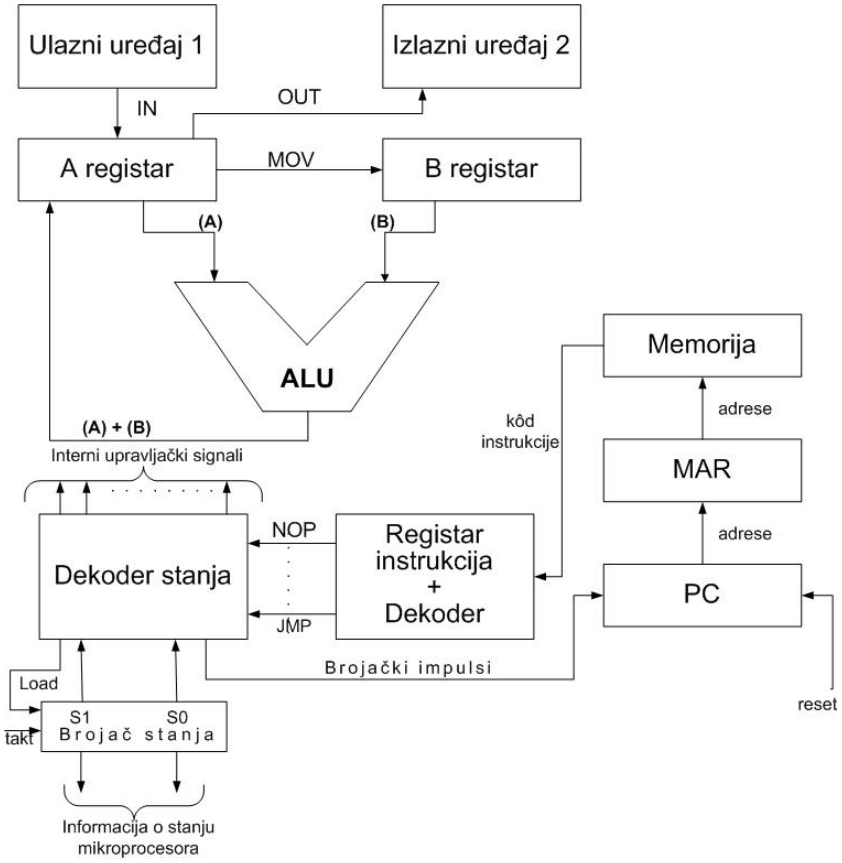
5.2.6. Sinhronizacija sistema i sekvencijalna stanja mašine

Kako se neke operacije sporije izvršavaju od drugih (npr, ADD se sporije izvršava od MOV), potrebno je impulse za inkrementiranje programskog brojača dovoditi u vremenskim intervalima koji su u saglasnosti sa trajanjem mašinskih operacija. Problem se rešava uvođenjem mašina sa sekvencijalnim stanjima. Da bi se ovo realizovalo, potrebno je analizirati ponavljanja sekvencijalnog rada mašine za vreme svake instrukcije. Na slici 5.19 dat je prelaz sekvencijalnih stanja. [21]



Slika 5.19 Mašina sa četiri sekvencijalna stanja

Realizacija sekvencijalnog rada ostvaruje se ugradnjom brojača stanja i dekodera stanja, što je prikazano na slici 520.



Slika 5.20 Sekvencijalni rad sa dekoderom i brojačem stanja

Dekoder stanja generiše upravljačke signale u strogo definisanim vremenskim intervalima. Njima se upravlja funkcijom pojedinih blokova sistema kao što su: tip operacije (ADD, SUB, OR, AND itd.), čitanje ili upis podataka u registre A, B ili MAR, čitanje ili upis podataka u memoriju itd.

Brojač stanja ukazuje u kom se stanju nalazi mašina (slika 5.19). Izlazi brojača se vode na ulaz dekodera stanja. Ulaz „Load“ postaje aktivan samo kada mašina iz stanja 2 prelazi u stanje 4, tj. kod izvršenja instrukcija koje traju kraće, kao što su i MOV, NOP i dr.. Tokom izvršenja instrukcija tipa ADD, SUB i dr. mašina sekvencijalno prelazi kroz sva četiri stanja.

5.2.7. Instrukcija za obraćanje memoriji

Postoji potreba da se procesor direktno obraća memoriji. Jedna od tipičnih instrukcija za neposredno kopiranje je **MVI**. To je dvobajtna instrukcija gde prvi bajt predstavlja operacioni kôd, a drugi bajt neposredni podatak. Dekoder instrukcija se konstruiše tako da može drugi bajt MVI instrukcije smestiti iz programske memorije direktno u registar. U tabeli 5.12 je data programska sekvenca koja koristi instrukciju MVI.

Tabela 5.12 Korišćenje instrukcije MVI

Lokacija	Sadržaj	Mnemonik		Komentar
0	10010001	IN	1	(A) ← (prekidači)
1	10110100	MOV	B, A	(B) ← (A)
2	10010001	IN	1	(A) ← (prekidači)
3	00010001	ADD	A, B	(A) ← (A) + (B)
4	10110100	MOV	B, A	(B) ← (A)
5	11000000	MVI	A, 65	
6	01100101			(A) ← 65
7	00010001	ADD	A, B	(A) ← (A) + (B)
8	10101000	OUT	2	LED ← (A)
9	?????????			

5.2.8. Bezuslovna izmena toka programa

Izmena toka programa može se klasifikovati u dve grupe: **bezuslovne** i **uslovne.**

Često se tokom izvršavanja programske sekvence javlja potreba bezuslovnog grananja. U tu svrhu može se koristi instrukcija **JMP** koja je trobajtna. Prvi bajt predstavlja operacioni kod instrukcije, a druga dva adresu sledeće instukcije koja treba da se izvrši.

Tabela 5.13 prikazuje izvršavanje JMP instrukcije kojom se tok programa vraća na adresu "0". [21]

Tabela 5.13 Korišćenje instrukcije JMP

```
        0:   IN    1
        1:   MOV   B, A
        2:   ADD   A, B
        4:   MOV   B, A
     5: 6:   MVI   A, 65
        7:   ADD   A, B
        8:   OUT   2
9: 10 : 11:  JMP   0000      ; Adresa skoka je 16-bitna
                             ; nakon izvršenja ove
                             ; instrukcije skače se na adresu 0
```

5.2.9. Uslovna izmena toka programa

U registar **Uslovni markeri** koji se dodeljuje ALU-u upisuju se stanja rezultata dobijena nakon izvršenja aritmetičkih i logičkih operacija. Stanja rezultata ukazuju da li se u toku izvršenja instrukcija javio prenos, da li je rezultat nula, da li je pozitivan i dr.. Instrukcije o uslovnoj izmeni toka vrše testiranje stanja uslovnih markera (određeni bit ćelije u registru Uslovni markeri), i u zavisnosti od rezultata testa izvršava se određena akcija. Instrukcija uslovnog grananja ima oblik Jxx, čiji je format :

1110	xxxx	yyyyyyyy	zzzzzzzz
Operacioni kôd instrukcije	Tip testa	Drugi bajt instrukcije: MS adresa skoka	Treći bajt instrukcije: LS adresa skoka

Postoje različite vrste instrukcija uslovnog grananja u zavisnosti od testa koje obavljaju (tabela 5.14) [21]

Tabela 5.14 Vrste instrukcija uslovnog grananja

Sadržaj prvog bajta instrukcije Jxx	Mnemonik	Komentar
11100000	JC	grananje ako ima prenosa
11100001	JNC	grananje ako nema prenos
11100010	JZ	grananje ako je rezultat nula
11100011	JNZ	grananje ako rezultat nije nula
11100100	JS	grananje ako je rezultat pozitivan
11100101	JNS	grananje ako je rezultat negativan
11100110	JP	grananje ako je parna parnost
11100111	JNP	grananje ako je neparna parnost

Primer programske sekvence koja koristi instrukciju uslovnog grananja JC, dat je u tabeli 5.15.

Tabela 5.15 Korišćenje instrukcije JC

0:	IN	1	
1:	MOV	B, A	
2:	IN	1	
3:	ADD	A, B	
4: 5: 6:	JC	0011	uslovni skok ako se javio prenos na adresi 11
7:	MOV	B, A	
8: 9:	MVI	A, 65	
10:	ADD	A, B	
11:	OUT	2	
12: 13: 14:	JMP	0000	

5.2.10. Registri

Kod računarskih sistema registri se koriste za čuvanje simbola, a predstavljaju logički skup ili uvođenje komponenata koje imaju dva ili veći broj stabilnih stanja. Registri obično mogu da služe obično za pomeranje informacija, brojanje, inkrementiranje, dekrementiranje kao i čuvanje podataka. Računar je efikasniji ukoliko ima veći broj različitihtipova registara pod uslovom da su optimalno projektovani i povezani.

Uloga CPU registara sagleda se njegove aktivnosti koje su: [5]

- Pribavljanje instrukcije – CPU mora da pročita instrukcije iz memorije,
- Interpretiranje instrukcije – instrukcija mora da se dekodira da bi se odredila koja se akcija zahteva,
- Pribavljanje podataka – izvršenje instrukcije zahteva čitanje podataka iz memorije ili U/I modula,
- Upis podataka – rezultat koji se dobija nakon izvršenja instrukcije u zavisnosti od tipa instrukcija potrebno je nekad ipisati u memoriju ili U/I modul.

Uspešno obavljanje ovih aktivnosti podrazumeva da CPU ima ugrađene registre u kojima se podaci privremeno smeštaju kao i da pamti lokacije zadnjih instrukcija, i

operacioni kod. To zajedno predstavlja malu internu memoriju što i jeste cilj implementacije ovog skupa registara.

5.2.10.1. Funkcionalna podela registara

Osnovna poedla registara prema funkcijama je na:

- Korisnički vidljive koji obezbeđuju programeru da na mašinskom ili asemblerskom jeziku minimizira obraćanje glavnoj memoriji putem optimiziranog korišćenja ovih registara,
- Upravljačke i statusne registri koji se koriste od strane upravljačke jedinice da upravljaju radom CPU-a, a takođe i od strane privilegovanih operativno-sistemskih programa da upravljaju izvršenjem programa.

Postoje sledeće kategorije korisnički vidljivih registara: [22]

- Registri opšte namene – njima programer dodeljuje različite finkcije, ali ipak postoje ograničenja..
- Registri za podatke – koriste se za čuvanje podataka, ali se ne mogu koristiti za izračunavanje adresa operanada.
- Adresni registri koji mogu biti opšte namene, ili su namenjeni za realizaciju nekog od sledećih adresnih načina rada:
- Pokazivači segmenta – kod mašina koje koriste segmentno adresiranje, u segmentni registar se smešta bazna adresa segmenta. Obično postoji veći broj ovih registara, na primer, jedan za operativni sistem, a drugi za tekući proces.
- Indeksni registri – koriste se za indeksno adresiranje, a mogu biti i autoindeksirani.
- Pokazivač magacina – ako postoji korisničko vidljivo adresiranje magacina, tada je, tipično, magacin deo memorije i postoji namenski registar koji pokazuje na vrh magacina. Ovaj način rada obezbeđuje implicitno adresiranje, što znači da PUSH, POP i druge instrukcije koje manipulišu sa magacinom ne treba eksplicitno da sadrže operand magacina (podrazumeva se koji je).
- Marker registar koji je delimično vidljiv korisniku. U njemu se čuvaju markeri (flags ili condition code) koje CPU hardver postavlja kao rezultat izvršenja operacije. Uslovni markeri se obično grupišu u jedan, ili veći broj registara. Obično markeri čine deo statusno/upravljačke grupe registara.

Prilikom projektovanja arhitekture računara za korisničko vidljive registre treba obratiti pažnju na sledeće:

- da li su registri specijalizovani ili ne,
- broj registara koji se ugrađuje,
- način korišćenja registara sa stanovišta obima podataka (8-, 16-, 32-, 64-bitnih) itd..

Upravljačkii statusni registri obično nisu vidljivi korisniku. Nekima od njih se može pristupiti jedino preko mašinskih instrukcija koje se izvršavaju u sistemskom režimu rada. Najvažnija 4 registra koja se koriste za programsko izvršavanje su:

- Programski brojač (PC) – sadrži adresu instrukcije koja čeka na izvršenje.
- Registar instrukcija (IR) – čuva instrukciju koja je zadnje doneta.
- Memorijsko adresni registar (MAR) – čuva adresu memorijske lokacije koja se adresira.
- Memorijsko baferski registar (MBR) – čuva podatak koji se upisuje u memoriju ili podatak koji je poslednji pročitan.

Postoji registar poznat kao PSW (Program Status Word) čiji je zadatak da čuva statusne informacije. Ovaj registar sadrži uslovne markere i informacije kao što su:

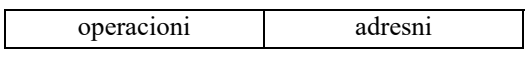
- SIGN (znak) – sadrži bit znaka rezultata zadnje aritmetičke operacije.
- ZERO (nula) – postavlja se kada je rezultat jednak 0.
- CARRY (prenos) – postavlja se izvršenjem operacije javio prenos (sabiranja), ili pozajmljivanje (oduzimanje) na mesto bita najveće težine.
- EQUAL (jednako) – postavlja se ako logička komparacija rezultira jednakost.
- OVERFLOW (premašaj) – koristi se da ukaže na aritmetički promašaj.
- INTERRUPT ENABLE/DISABLE (dozvola rada prekida ili zabrana) – koristi se da dozvoli ili zabrani prekid.
- SUPERVISOR (supervizor) – ukazuje kada CPU izvršava instrukcije u supervizorskom načinu rada.

5.2.11. Rad sa instrukcijama

Na standardnom mašinskom nivou svaka instrukcija je predstavljena sekvencom bita. Instrukcija se sastoji od odgovarajućih elemenata koji su potrebni radi izvršavanja, a ovi elementi definišu: [22]

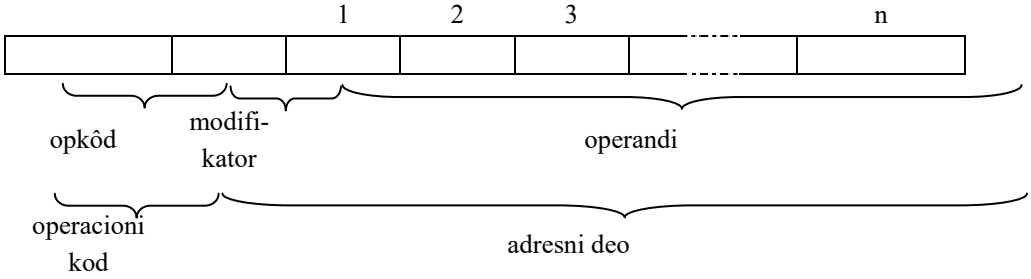
- tip operacije koji treba da se izvrši,
- implicitnu ili eksplicitnu specifikaciju jednog ili većeg broja operanada nad kojima se izvršava operacija,
- adresu gde treba da smestiti rezultat operacije i
- adresu naredne instrukcije koja treba da se izvrši

Slika 5.21 daje prikaz instrukcije koja sadrži dva osnovna dela: operacioni i adresni, ge se operacioni deo zove operacioni kôd (opkôd), dok adresni deo sadrži informaciju o stavkama b), c) i d).



Slika 5.21 Osnovni delovi instrukcije

Svaka instrukcija ima odgovrajući format koji se obično sastoji iz više polja. Za većinu instrukcija postoji više od jednog formata. Osnovn n-operandski format instrukcije dat je na slici 5.22.



Slika 5.22 Osnovni format instrukcije

5.2.11.1. Adresni deo instrukcije

Kao što se slike 5.22 može videti adresni deo instrukcije se sastoji od operanada i modifikatora. Informacije koje modifikator sadrži su:

- opis načina adresiranja
- dodatni uslovi koji se odnose na pristup podacima ili na način izvršenja operacije

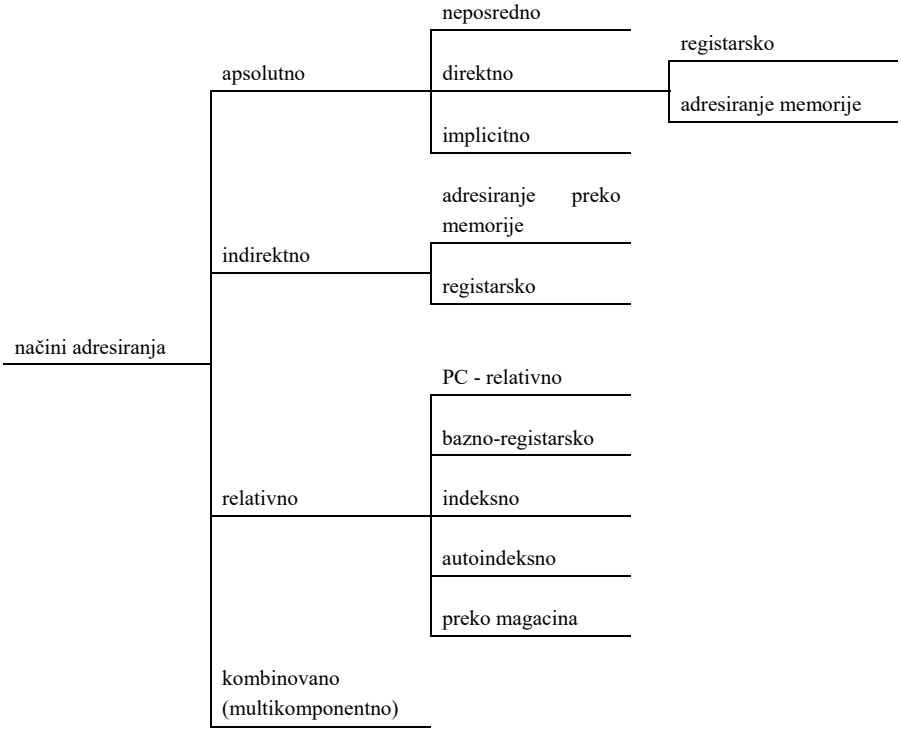
Najveći broj operanada se zahteva tokom izvršenja aritmetičkih i logičkih instrukcija. Kod njih su potrebne četiri adrese: dve adrese za obraćanje operandima, jedna za pamćenje rezultata, i jedna koja ukazuje odakle treba pribaviti narednu instrukciju. Umesto četvoro-adresne specifikacije može se koristiti i tro-adresna, pri čemu se adresa naredne instrukcije dobija implicitno preko programskog brojača.

Da bi se smanjio obim instrukcije, a time i memorijski prostor, neophodno je da se instrukcijom eksplicitno odredi m operanada, gde je m<n. Za ostale operande se smatra da su implicitno određeni. Tako eksplicitna adresna polja se odnose na obraćanje memoriji, a implicitna na obraćanje na adrese registra. Za procesor kažemo da je m-to adresna mašina ako je m maksimalan broj eksplicitnih adresa glavne memorije. Implicitni ulazni operandi moraju se prethodno smestiti u registre koji su procesoru poznati pre nego što se instrukcija počne izvršavati.

5.2.11.2. Načini adresiranja

CPU koristi različite metode za pristupanje adresama. Pošto adresno polje kod tipičnog formata instrukcija je ograničeno to programeri koriste različite tehnike da bi pristupili glavnoj i/ili virtuelnoj memoriji. Ove tehnike predstavljaju komprimis između adresnog prostora i/ili adresne fleksibilnosti sa jedne strane i broja obraćanja memoriji i/ili kompleksnosti izračunavanja adresa sa druge strane. Neki procesori koriste veći broj različitih registara koji služe za adrese, a drugi uopšte ne koriste adresne registre.

Cilj adresnog načina rada je da se odredi precizna lokacija željenog podatka tj. njegova efektivna adresa. U tom smislu razlikujemo nekoliko vrsta adresiranja (slika 5.23).



Slika 5.23 Načini adresiranja

Apsolutna adresa je ona adresa koja određuje memorijsku lokaciju operanda bez korišćenja bilo koje druge dodatne informacije.

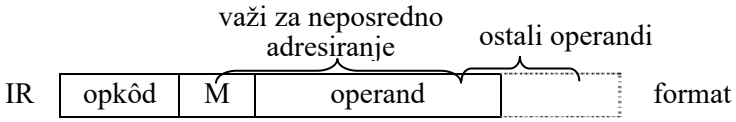
Indirektna adresa ne pokazuje memorijsku lokaciju operanda već memorijsku lokaciju na kojoj se nalazi adresa operanda.

Kod **relativnog adresiranja** polje operanda sadrži relativnu adresu ili razmeštaj, a kompletna adresa operanda formira se od strane CPU-a.

Neposredno adresiranje

Neposredno adresiranje je najjednostavniji opblik adresiranja. Kod njega adresni deo instrukcije određuje konstanta koja ukazuje na adresu gde se podatak nalazi (slika 5.24). Koristi kod definisanja konstanti ili za postavljanje početne vrednosti promenljivih.

OPERAND = A



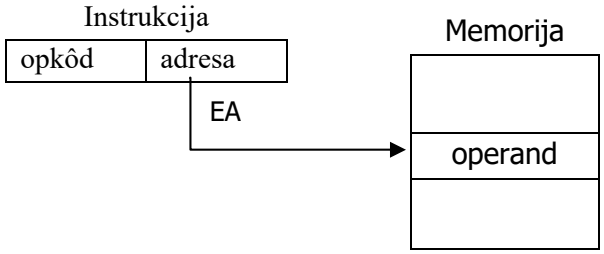
Slika 5.24 Neposrednog adresiranje

Prednost ovog adresiranja je u uštedi vremena, jer ne postoji obraćanje memoriji sem onog koje je potrebno za pribavljanje instrukcije. Nedostatak ovog adresiranja je ograničenost adresnog polja.

Direktno adresiranje

Adresno polje kod ovog adresiranja sadrži efektivnu adresu operanda (EA) na osnovu koje se vrši obraćanje memoriji. Zbog toga ova adresa zove direktna adresa.

EA = A



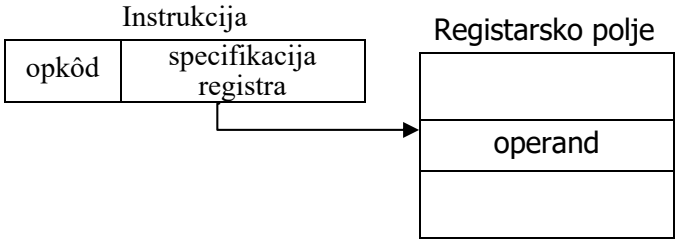
Slika 5.25 Direktno adresiranje

Prednost ovog adresiranja je što postoji samo jedno obraćanje memoriji, odnosno ne postoji potreba za posebno izračunavanje adrese. Mana je što se manipuliše sa ograničenim adresnim prostorom.

Registarsko adresiranje

Registarsko adresiranje (slika 5.26) za razliku od direktnog adresiranja u adresnom polju daje specifikaciju registra kome se pristupa a ne na adresu glavne memorije.

EA=R



Slika 5.26 Registarsko adresiranje

Adresno polje kojim se određuju registri je, obično, 3 do 4 bita, tako da se maksimalno može pristupiti registarskom polju veličine od 8 do 16 registara.

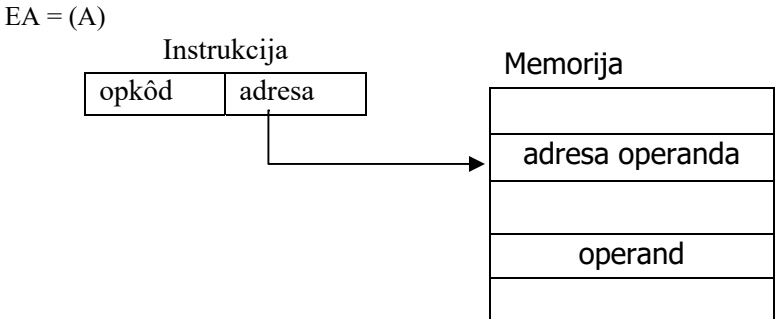
Prednosti registarskog adresiranja su:

- adresno polje instrukcije kojim se specificira registar je mali i obima je nekoliko bitova
- nije potrebno obraćanje memoriji
- instrukcije koje manipulišu sa sadržajem registra brzo se izvršavaju.

Nedostatak registarskog adresiranja je je obično mali broja CPU-ovih registara.

Indirektno adresiranje preko memorije

Direktno adresiranje ima problem ograničenog adresnog prostora kome se može pristupiti zbog dužine adresnog polja koje je malo. Zbog toga se pristupa indirektnom adresiranju koje adresno polje koristi da bi se pristupilo memorijskoj adresi u kojoj je smeštena potpuna adresa operanda (slika 5.27).



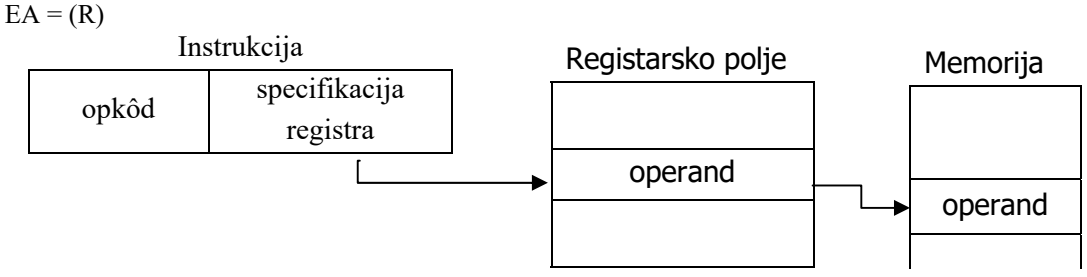
Slika 5.27 Indirektnog adresiranja preko memorije

Prednost indirektnog afresiranja je dostupnost velikog adresnog prostora (ako je dužina reči N, biće dostupan adresni prostor obima 2^N).

Nedostatak se ogleda u tome što se zahtevaju dva memorijska obrađanja: jedan prilikom pribavljanja adrese, a drugi prilikom dobijanja vrednosti operanda.

Indirektno registarsko adresiranje

Ovo adresiranje analogno je indirektnom, s tom razlikom što se adresno polje naredbe odnosi na određeni registar, a ne na memorijsku lokaciju (slika 5.28).



Slika 5.28 Indirektno registarsko adresiranje

Prednosti i nedostatci ovog adresiranja su isti kao kod indirektnog adresiranja. Kod indirektno registarskog adresiranja imamo jedno obraćanje memoriji manje, u odnosu na indirektno adresiranje.

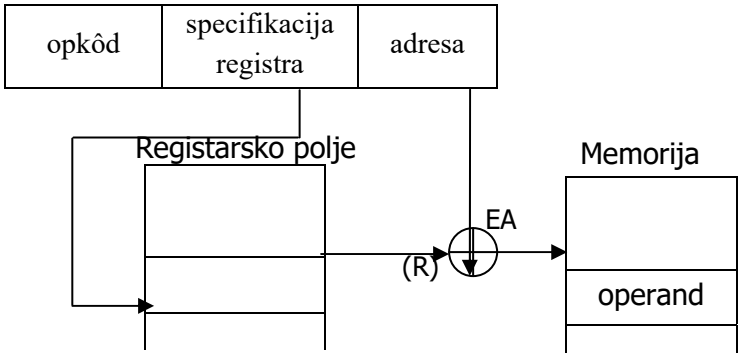
Relativno adresiranje

Ovo adresiranje je moćno zbog toga što kombinuje mogućnosti direktnog i indirektno registarskog adresiranja (slika 5.29). Naredbe kod ovog adresiranja imaju dva adresna polja pri čemu je jedno eksplicitno.

EA = A + (R)

Jedno adresno polje (vrednost = A) se koristi kao direktno, dok se drugom adresnom polju implicitno pbraća na bazi opkôda. Vrednosti se sabiraju da bi se dobila efektivna vrednost. Najčešće korišćeni načini relativnog adresiranja su:

- PC relativno adresiranje
- Bazno registarsko adresiranje
- Indeksno adresiranje



Slika 5.29 Relativno adresiranje

PC relativno adresiranje

Ovo adresiranje koristi registar programski brojač (PC). Efektivna adresa (EA) se formira tako što se tekuća adresa instrukcije sabira sa razmeštajem i izračunava se na sledeći način:

EA = (PC) + D

gde je D razmeštaj specificiran instrukcijom.

Bazno registarsko adresiranje

Kod ovog adresiranja registar kome se obraćamo, R, čuva memorijsku adresu, a adresno polje sadrži razmeštaj, D, (obično neoznačena celobrojna vrednost) u odnosu na tu adresu. Obraćanje (baznom) registru može biti eksplicitno ili implicitno.

Metod izračunavanja EA je:

EA = R + D

Indeksno adresiranje

Kod indeksnog adresiranja adresno polje, A, sadrži adresu glavne memorije, a u registru, R, kome se obraćamo smešten je pozitivni razmeštaj u odnosu na tu adresu. Metod izračunavanja EA isti kao kod bazno registarskog adresiranja:

EA = R + D

U tabeli 5.16 su date su glavne prednosti i nedostaci različitih načina adresiranja.

Tabela 5.16 Karakteristike načina adresiranja

Način rada	Algoritam	Glavna prednost	Glavni nedostatak
neposredno adresiranje	Operand=A	ne postoji obraćanje memoriji	ograničeni iznos operanda
direktno adresiranje	EA = A	jednostavnost	ograničeni radni prostor
indirektno adresiranje	EA = (A)	veliki adresni prostor	veći broj obraćanja memoriji
registarsko adresiranje	EA = R	ne postoji obraćanje memoriji	ograničeni adresni prostor
registarsko indire-ktno adresiranje	EA = (R)	veliki adresni prostor	dodatno obraćanje memoriji
relativno adresiranje	EA = A + (R)	fleksibilno	kompleksno

**Napomena: EA – aktuelna (efektivna) adresa lokacije koja sadrži operand kome se obraćamo*

5.2.12. Blok šema mikroračunara

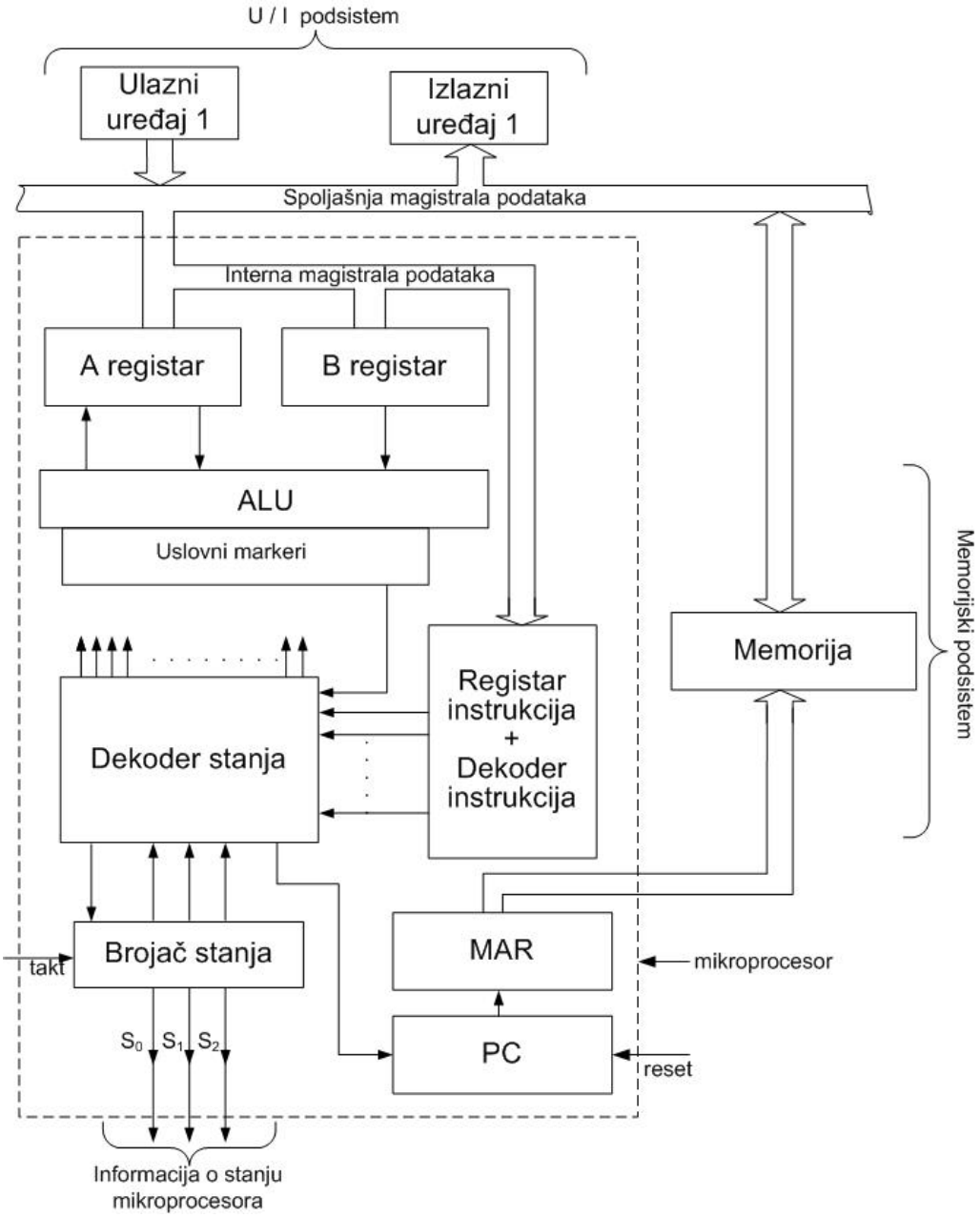
Opšta blok šema arhitekture jedne realizacije mikroračunara data je na slici 5.30.

Mikroračunarski sistem čine sledeće tri celine:

- Mikroprocesor (CPU)
- Memorija
- U/I.

Sa slike se, takođe, može uočiti da spoljašnje magistrale služe za prenos podataka i instrukcija iz memorije i podataka za ulazno-izlazne (U/I) uređaje.

Mikroprocesor generiše upravljačke signale kojima se određuje smer prenosa podataka.



Slika 5.30 Blok šema arhitekture mikroračunarskog sistema

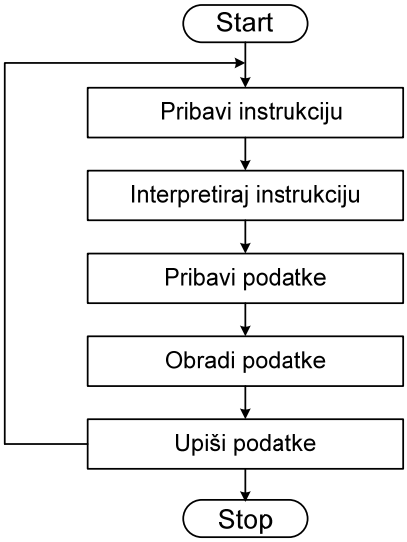
5.3.SPECIFIKACIJA ARHITEKTURE PROCESORA TFACO

U prethodnom delu ovog rada objašnjeni su osnovni zahtevi kreiranja arhitekture računara idući od prostijih ka složenijim sistemima. U narednom delu biće izložena specifikacija arhitekture procesora, pod imenom TFaCo, čija je arhitektura kreirana od strane autora ovog rada.

5.3.1. Funkcije procesora

Procesor TFaCo obavlja sledeće funkcije (slika 5.31): [44]

- pribavlja instrukcije – čita instrukciju koja se nalaze u memoriji
- interpretira instrukcije – instrukcija mora da se dekodira da bi se odredilo koja se akcija zahteva
- pribavlja podatke – izvršenje instrukcije može da zahteva čitanje podataka iz memorije
- obrađuje podatke – izvršenje instrukcije može da predviđa obavljanje neke aritmetičke ili logičke operacije nad podacima
- upisuje podatke – rezultati izvršenja se upisuje u memoriju

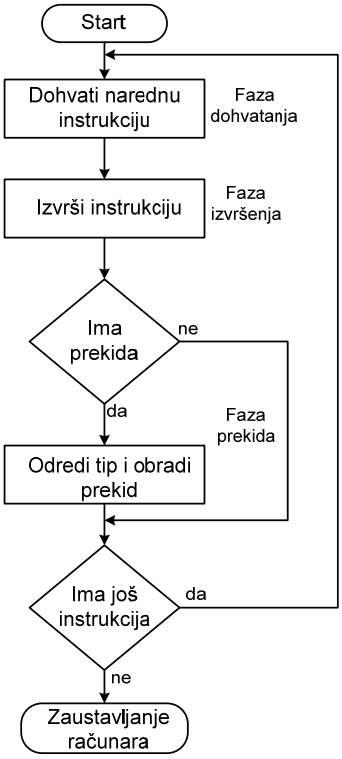


Slika 5.31 Funkcije procesora TFaCo

Obrada koja je predviđena jednom instrukcijom naziva se ciklus instrukcije. Faze ciklusa instrukcije procesora su (slika 5.32): [43], [45]

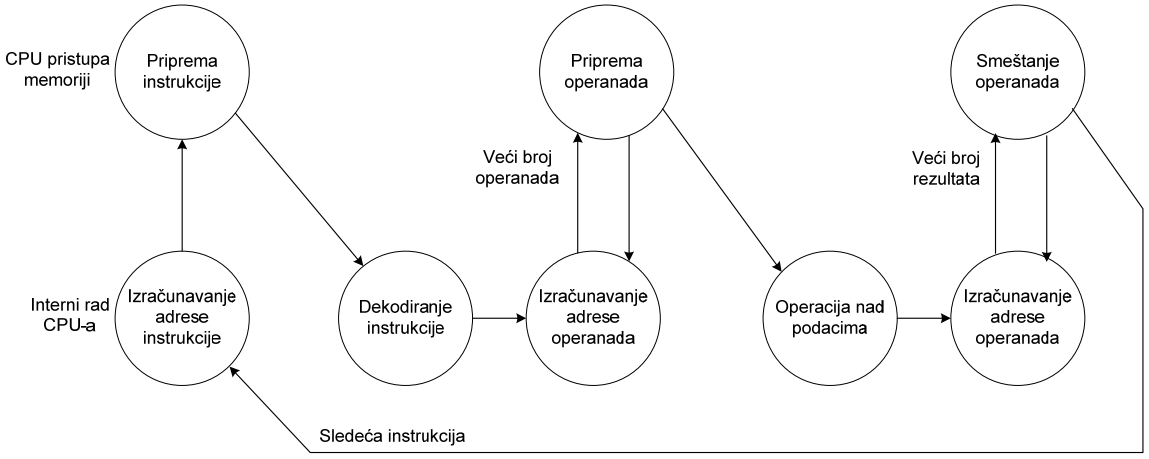
- Faza dohvatanja, u kojoj se izvršava:
- Izračunavanje adrese instrukcije, pri čemu se određuje adresa naredne instrukcije koja treba da se izvrši.
- Priprema instrukcije. U ovom koraku se čita instrukcija iz memorije i prenosi u CPU.

- Dekodiranje operacionog koda instrukcije. Na osnovu analize operacionog koda određuje se tip operacije i broj argumenata instrukcije.
- Izračunavanje adrese operanada. Za svaki od eventualnih operanada instrukcije se određuje gde je smešten (registar ili memorija).
- Priprema operanada u kome se vrši dohvatanje operanada iz memorije.
- Faza izvršenja. U njoj se:
- Izvršava operacija koja je zahtevana operacionim kodom.
- Izračunavaju adrese operanada
- Smeštaju operandi (čuvanje rezultata u memoriji ili na U/I uređajima).
- Faza prekida, koja:
- Proverava da li se u međuvremenu desio prekid.
- Ako je bilo prekida, tada se vrši njegova obrada.



Blok dijagram 5.32. Faze ciklusa instrukcija procesora TFaCo

Procesor izvršava svaku instrukciju kroz niz određenih koraka. Na slici 5.33 dat je dijagram stanja ciklusa instrukcija procesora



Slika 5.33 Dijagram stanja ciklusa instrukcija procesora TFaCo

5.3.2. Opšti format instrukcija

Instrukcije procesora TFaCo su fiksne dužine 16 bita. Osnovni format instrukcije ima sledeći izgled.

COP	ADRESNI DEO

Dužina operacionog dela instrukcije (COP) je **4 bita** ($\mathbf{I_{12} - I_{15}}$). Funkcija ostalih bitova instrukcije ($\mathbf{I_0 - I_{11}}$) zavisi od vrste instrukcije i primenjenog načina adresiranja. Dužina procesorske reči je **16 bita**, a istu dužinu ima i **memorijska reč**, čija je dužina prilagođena dužini instrukcije, koja se na taj način može dohvatati u okviru jednog memorijskog ciklusa.

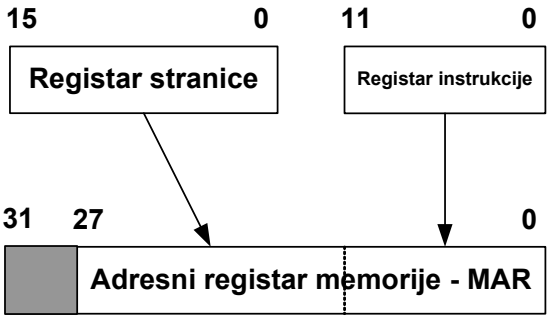
5.3.3. Načini adresiranja

Procesor TFaCo omogućava sledeće načine adresiranja

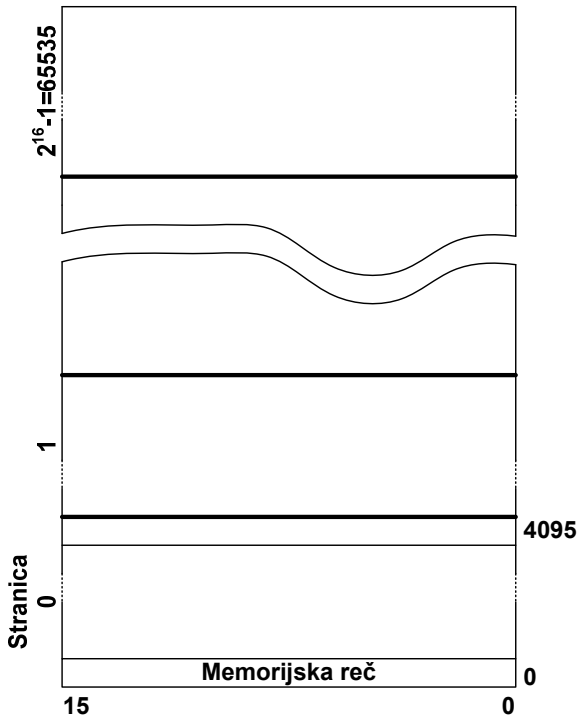
- **Direktno memorijsko adresiranje**[16] – Kod ovog načina adresiranja bitovi instrukcije **0 – 11** predstavljaju adresu adresibilne jedinice (memorijske reči dužine 16 bita) unutar stranice od $\mathbf{2^{12}}$ (4096) lokacija. Adresa stranice je definisana sadržajem **Registra stranice** (R_{11} – RC i R_{12} - RD) i memorijska adresa se dobija spajanjem sadržaja segment registra i adresnog dela instrukcije. Na taj način memorijska adresa TFaCo ima dužinu **16** (definiše jednu od 2^{16} stranica) plus **12** (definiše memorijsku reč unutar stranice) bita, što ukupno iznosi **28 bita**. Način formiranja izvršne adrese prikazan je na slici 5.34. Ukupan broj adresibilnih jedinica (memorijskih reči dužine 16 bita) memorije sistema TFaCo iznosi $\mathbf{2^{28}}$ memorijskih reči ili **256MB**. S obzirom da je memorijska reč dugačka 2 bajta onda je maksimalni kapacitet ove memorije $\mathbf{2^{29}}$ (**512MB**) bajta. Struktura memorije

[16] Ovde je promenjen koncept veličine operativne memorija, koja je inicijalno bila kapaciteta – 4096 reči ili 8192 bajta. Uvođenjem *registra stranice* maksimalni kapacitet se povećava na $2^{16}\times2^{12} = 2^{28}$ adresibilnih jedinica (reči).

sistema TFaCo je prikazana na slici 5.35. Shodno ovome dužina **memorijskog adresnog registra** (MAR) iznosi 28 bitova. Ovaj način adresiranja primenjuje se kod instrukcija kojima se vrši prenos podataka između **akumulatora** i **memorije**.

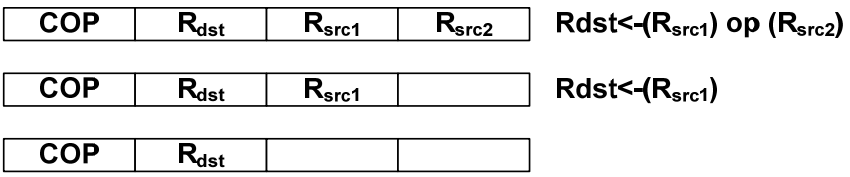


Slika 5.34 Način formiranja izvršne memorijske adrese



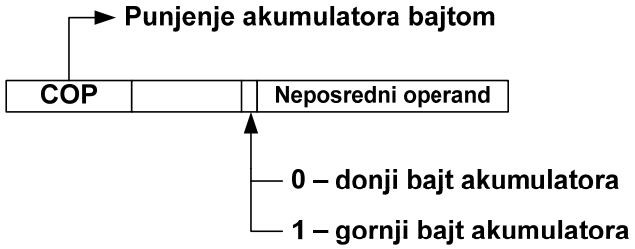
Slika 5.35 Struktura operativne memorije

- **Registarsko adresiranje –** Ovaj način adresiranja implicira da se sve aritmetičko – logičke instrukcije izvršavaju nad sadržajima registara. Ovde postoje dva podformata, jedan se odnosi na troadresne instrukcije, a drugi su dvoadresne instrukcije u koje spadaju instrukcije kojima se vrši prenos podataka između **akumulatora** i **registara.** Formati instrukcije kod ovog načina adresiranja prikazani su na slici 5.36.



Slika 5.36 Registarski načini adresiranja

- **Neposredno adresiranje** – Ovaj način adresiranja za punjenje akumulatora neposrednom vrednošću operanda. Usvojeno je da dužina neposrednog operanda bude 8 bita (1 bajt). U tom slučaju bitovi instrukcije $I_8 – I_{11}$ ostaju neiskorišćeni. S obzirmo, da je dužina akumulatora 16 bita ova instrukcija može da se iskoristi da se bira u koji bajt akumulatora će se upisati neposredni operand. Za tu namenu se koristi bit 8. Ako je $I_8 = 0$ tada se neposredni operand upisuje u donji bajt akumulatora, a ako je $I_8 = 1$ tada se neposredni operand upisuje u gornji bajt akumulatora, kao što je prikazano na slici 5.37.

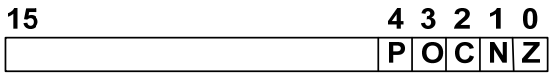


Slika 5.37 Neposredno adresiranje

5.3.4. Skup registara i njihova funkcija

Arhitektura procesora TFCo je RISC orijentisana pa procesor ima registar fajl od **16 registara**, koji se svi programski dostupni i svaki je dužine 16 bita. Skup registara je podeljen u dva bloka registara

- Registri opšte namene (**$R_0 – R_7$**)
- Registri specijalne namene (**$R_8 – R_{15}$**) – svaki registar ima određenu namenu. Registri su programski dostupni tako da u određenim slučajevima mogu da budu korišćeni i kao registri opšte namene. Spisak specijalnih registara koje poseduje procesor TFaCo dat je u tabeli 5.17.
- **R_8 – Akumulator (RA)** – Registar se koristi kao implicitno odredište/izvorište podatka za potrebe razmene informacija sa memorijom.
- **R_9 – Statusni registar (RS)** – U ovaj registar se po okončanju svake operacije upisuje odgovarajuća informacija o statusu izvršene operacije u skladu sa pravilima datim u tabeli 5.18. Struktura statusnog registra je data na slici 5.38.



Slika 5.38 Struktura statusnog registra RS (R9)

- **R_{10}** – **Registar skoka (RJ)** – U ovom registru se nalazi adresa (12 bita) po kojoj se vrši prelazak (skok) u programu. Punjenje odgovarajućim sadržajem (adresom skoka) može da se realizuje:
  - Unosom neposrednih operanada
  - Prenosom sadržaja nekog od registara
  - Čitanjem sadržaja memorije preko akumulatora

- **R_{11} – Registar programske stranice (RC)** – Registar koji sadrži početnu adresu jedne od stranica u memoriji u kojima se nalaze instrukcije. Puni se na jedan od načina kao i registar skoka ili izvršavanjem određene aritmetičke operacije.
- **R_{12} – Registar podataka stranice (RD)** – Registar koji sadrži početnu adresu jedne od stranica u memoriji u kojima se nalaze podaci. Puni se na jedan od načina kao i registar skoka ili izvršavanjem određene aritmetičke operacije.
- **R_{13} – Indeks registar (RX)** – Registar koji se koristi za indeksiranje sadržaja nizova pri relativnom adresiranju
- **R_{14} – Bazni registar (RB)** – Koristi se kod relativnog adresiranja, kao registar baze.
- **R_{15} – Programski brojač (PC)** – Definiše adresu instrukcije u slučaju konsekutivnog izvršavanja instrukcija.

Tabela 5.17 Funkcije specijalnih registara

R_8	Akumulator (RA)
R_9	Statusni registar (RS)
R_{10}	Registar skoka (RJ)
R_{11}	Registar programske stranice (RC)
R_{12}	Registar stranice podataka (RD)
R_{13}	Indeks registar (RX)
R_{14}	Bazni registar (RB)
R_{15}	Programski brojač (PC)

Tabela 5.18 Značenje bitova u registru statusa – R_9

Bit	Oznaka	Značenje
R_9 (0)	**Z – Zero Bit**	
Z=0		Rezultat operacije različit od 0
Z=1		Rezultat operacije jednak 0
R_9 (1)	**N – Negative Bit**	
N=0		Rezultat operacije pozitivan
N=1		Rezultat operacije negativan
R_9 (2)	**C – Carry Bit**	
C=0		Nema prenosa iz najstarijeg razreda rezultata
C=1		Postoji prenos iz najstarijeg razreda rezultata
R_9 (3)	**O – Overflow Bit**	
O=0		Nema prekoračenja opsega računanja
O=1		Došlo je do prekoračenja opsega računanja
R_9 (4)	**P – Parity Bit**	
P=0		Broj 1 u rezultatu paran
P=1		Broj 1 u rezultatu neparan

5.3.5. Instrukcije TFaCo i opis njihovog izvršavanja

Skup instrukcija procesora TFaCo dat je u tabeli 5.19.

Tabela 5.19 Skup instrukcija TFaCo i opis njihovog izvršavanja

COP	Simboli-čki kôd	Sintaksa	Opis
0000	**NOP**	NOP	**Operacija bez efekta**
0001	**LI**	LI lb, podatak	**Operacija punjenja akumulatora neposrednim operandom** ▪ **lb** – Jednobitna informacija, koji se nalazi na lokaciji $\mathbf{I_8}$, i koja definiše u koji bajt se upisuje neposredni operand, koji je dužine 1 bajt. o **lb=0** – podatak se upisuje u donji bajt akumulatora o **lb=1** – podatak se upisuje u gornji bajt akumulatora ▪ **podatak** – 8 – bitna informacija. Pomoću ove naredbe, u opštem slučaju, mogu da se formiraju sledeći podaci u akumulatoru: o Neoznačena veličina dužine 16 bita o Označena veličina dužine 16 bita o Binarni broj dužine 16 bita o Četiri heksadecimalna broja o Dva ASCII karaktera o Jedan Unicode karakter
0010	**LOAD**	LOAD adresa	**Puni akumulator podatkom iz memorije** ▪ **adresa** - memorijska adresa sa koje se čita podatak
0011	**STORE**	STORE adresa	**Pamti sadržaj akumulatora** ▪ **adresa** - memorijska adresa na koju se upisuje podatak
0100	**ADD**		**Sabiranje** r3←(r1)+(r2) – Funkcionalni format instrukcije za sabiranje ▪ r3 – Broj odredišnog registra ▪ r2 – Broj registra drugog operanda ▪ r1 – Broj registra prvog operanda
	ADDR	ADDR r3, r1, r2	r3←(r1)+(r2) – Pun format instrukcije za sabiranje. Registar može da bude bilo koji registar iz skupa registara $\mathbf{R_0 - R_{15}}$.
	ADCR	ADCR r1, r2	acc←(r1)+(r2) – Odredišni registar je akumulator, registri operanada može da bude bilo koji iz skupa registara.
	ADCC	ADCC r	acc←(acc)+(r2) – Odredišni i registar prvog operanda je akumulator, drugi operand se može uzeti iz bilo kog registra iz skupa registara.
0101	**SUB**		**Oduzimanje** r3←(r1)-(r2) – Funkcionalni format instrukcije za oduzimanje ▪ r3 – Broj odredišnog registra ▪ r2 – Broj registra drugog operanda ▪ r1 – Broj registra prvog operanda

COP	Simboli-čki kôd	Sintaksa	Opis
	SUBR	SUBR r3, r1, r2	r3←(r1)-(r2) – Pun format instrukcije za oduzimanje. Registar može da bude bilo koji registar iz skupa registara $\mathbf{R_0 – R_{15}}$.
	SBCR	SBCR r1, r2	acc←(r1)-(r2) – Odredišni registar je akumulator, registri operanada mogu da budu bilo koji iz skupa registara.
	SBCC	SBCC r	acc←(acc)-(r2) – Odredišni i registar prvog operanda je akumulator, drugi operand se može uzeti iz bilo kog registra iz skupa registara.
0110	**MVAC**	MVAC r	**Kopiranje sadržaja registra u akumulator** acc←(r)
0111	**MVR**	MVR r	**Kopiranje sadržaja akumulatora u registar** r←(acc)
1000	**NOT**	NOT r1	**Logička n**egacija acc←not(r1)
1001	**OR**		**Logičko ILI** r3←(r1) or (r2) – Funkcionalni format instrukcije za logičko ili ▪ r3 – Broj odredišnog registra ▪ r2 – Broj registra drugog operanda ▪ r1 – Broj registra prvog operanda
	ORR	ORR r3, r1, r2	r3←(r1) or (r2) – Pun format instrukcije za logičko ili. Registar može da bude bilo koji registar iz skupa registara $\mathbf{R_0 – R_{15}}$.
	ORCR	ORCR r1, r2	acc←(r1) or (r2) – Odredišni registar je akumulator, registri operanada mogu da budu bilo koji iz skupa registara.
	ORAC	ORAC r2	acc←(acc) or (r2) – Odredišni i registar prvog operanda je akumulator, drugi operand se može uzeti iz bilo kog registra iz skupa registara.
1010	**AND**		**Logičko I** r3←(r1) or (r2) – Funkcionalni format instrukcije za logičko i ▪ r3 – Broj odredišnog registra ▪ r2 – Broj registra drugog operanda ▪ r1 – Broj registra prvog operanda
	ANDR	ANDR r3, r1, r2	r3←(r1) and (r2) – Pun format instrukcije za logičko i. Registar može da bude bilo koji registar iz skupa registara $\mathbf{R_0 – R_{15}}$.
	ANCR	ANCR r1, r2	acc←(r1) and (r2) – Odredišni registar je akumulator, registri operanada mogu da budu bilo koji iz skupa registara.
	ANAC	ANAC r2	acc←(acc) and (r2) – Odredišni i registar prvog operanda je akumulator, drugi operand se može uzeti iz bilo kog registra iz skupa registara.
1011	**XOR**		**Ekskluzivno ILI** r3←(r1) or (r2) – Funkcionalni format instrukcije za ekskluzivno ili ▪ r3 – Broj odredišnog registra ▪ r2 – Broj registra drugog operanda ▪ r1 – Broj registra prvog operanda

COP	Simboli-čki kôd	Sintaksa	Opis
	XORR	XORR r3,r1,r2	r3←(r1) xor (r2) – Pun format instrukcije za ekskluzivno ili. Registar može da bude bilo koji registar iz skupa registara **R_0 – R_{15}**.
	XRCR	XRCR r1, r2	acc←(r1) xor (r2) – Odredišni registar je akumulator, registri operanada mogu da budu bilo koji iz skupa registara.
	XRAC	XRAC r2	acc←(acc) xor (r2) – Odredišni i registar prvog operanda je akumulator, drugi operand se može uzeti iz bilo kog registra iz skupa registara.
1100	**SHFT**	SHIFT ind, r ,np	**Pomeranje podatka levo/desno** acc←shift(r) ▪ **ind** – Jednobitna informacija koja se nalazi na poziciji **I8** i koja definiše smer pomeranja o **ind=0** - pomeranje levo, o **ind=1** - pomeranje desno ▪ **r** – Broj registra čiji se sadržaj pomera, pri čemu je odredište rezultata uvek akumulator ▪ **np** – Broj bitova za koji se vrši pomeranje. Četvorobitna informacija koja zauzima pozicije **I_0 – I_3**.
1101	**JUMP**	JUMP maska	**Instrukcija skoka** Adresa skoka se nalazi u registru skoka (RJ). ▪ **maska** – Definiše test za proveru uslova za skok u programu. Dužina maske je 6 bitova o **maska=000001** – provera na nulu (Z=1) o **maska=000010** – provera na negativno (N=1) o **maska=000100** – provera na prenos (C=1) o **maska=001000** – provera na prekoračenje (O=1) o **maska=010000** – provera na parnost (P=1) o **maska=1xxxxx** – bezuslovni skok
1110	**CLR**	CLR r3	**Upisivanje nula u registar r** r3 ← 0000...000. Odredišni registar i registar prvog operanda je isti.
1111	**HALT**	HALT	**Zaustavljanje programa**

5.3.6. Opisi binarnog formata instrukcija TFaCo

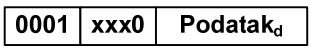
Pri objašnjavanju načina kodovanja instrukcija koristiće se sledeća pravila za označavanje:

- **0 –** Vrednost bita jednaka 0
- **1 –** Vrednost bita jednaka 1
- **x –** Vrednost bita nije bitna

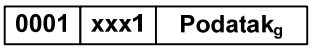
NOP – Bez operacije

Slika 5.39 Format instrukcije NOP

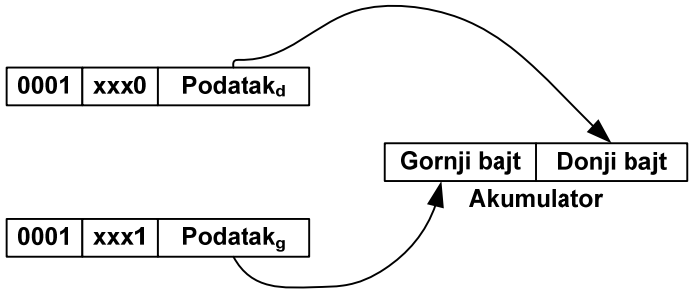
LI – Punjenje akumulatora neposrednim operandom



a) Upis podatka u donji bajt akumulatora



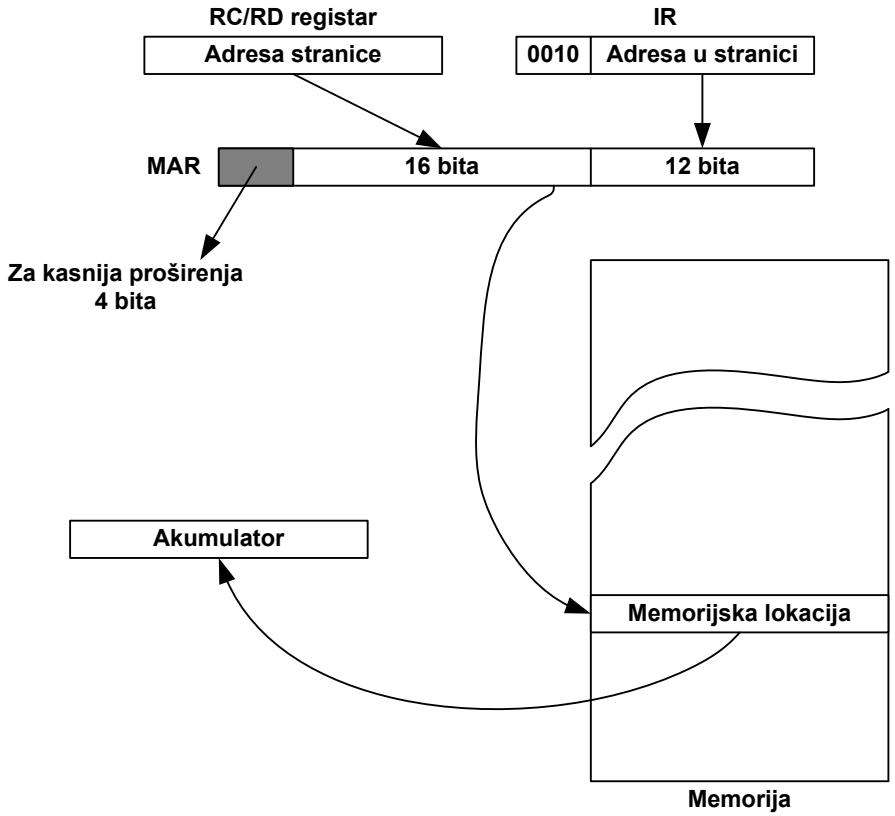
b) Upis podatka u gornji bajt akumulatora



Slika 5.40 Način izvršenja LI instrukcije

LOAD – Punjenje akumulatora sadržajem memorije

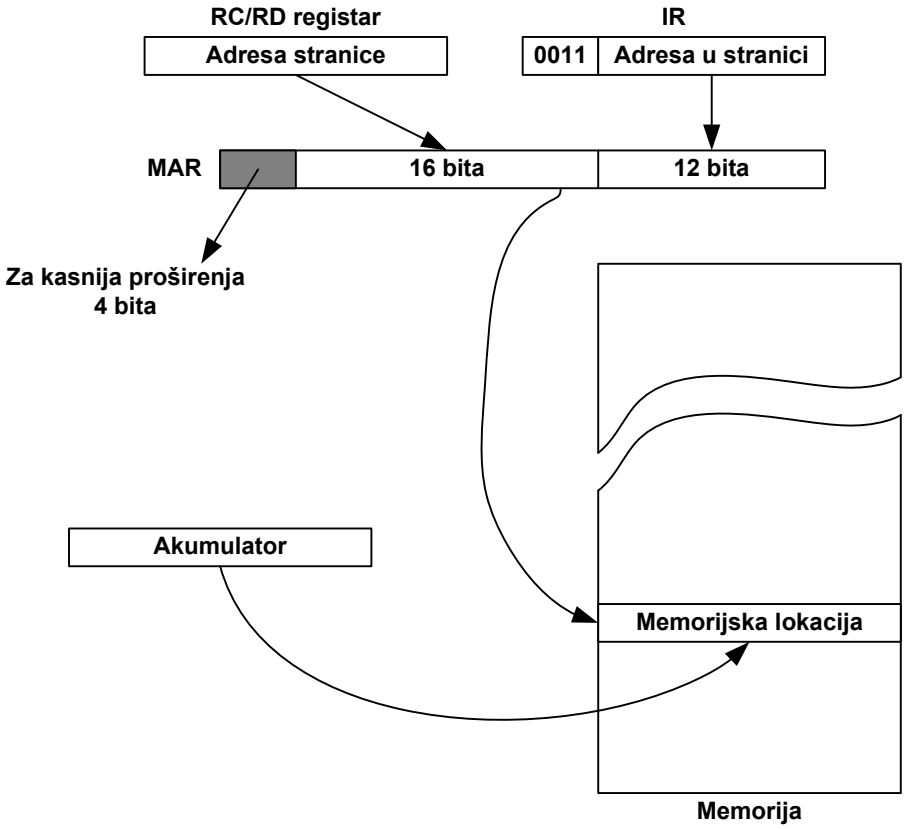
Slika 5.41 Punjenje akumulatora sadržajem adresirane memorijske lokacije specificirana adresa odgovara memorijskoj lokaciji sa koje se čita podatak



Slika 5.42 Način izvršavanja instrukcije LOAD

STORE – Pamćenje sadržaja akumulatora u memoriji

Slika 5.43 Pamćenje sadržaja akumulatora u adresiranoj memorijskoj lokaciji specificirana adresa odgovara memorijskoj lokaciji na kojoj se pamti podatak



Slika 5.44 Način izvršenja instrukcije STORE

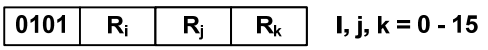
ADD – Sabiranje

Slika 5.45 Opšti format instrukcije ADD

Slika 5.46 Format instrukcije ADD kad je odredište akumulator[17]

Slika 5.47 Format instrukcije ADD kad su odredište i registar prvog operanda akumulator[18]

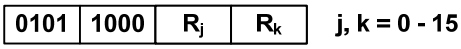
SUB – Oduzimanje



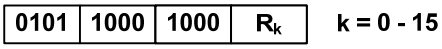
Slika 5.48 Opšti format instrukcije SUB

[17] Ovaj format ima svoju posebnost samo na nivou simboličkog mašinskog jezika i može se u programu bez greške koristiti i kao opšti format samo što u polje odredišta treba staviti oznaku za akumulator

[18] Ovaj format ima svoju posebnost samo na nivou simboličkog mašinskog jezika i može se u programu bez greške koristiti i kao opšti format samo što u polje odredišta i prvog operanda treba staviti oznaku za akumulator

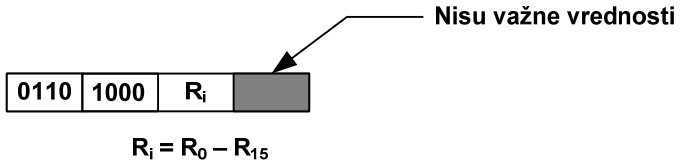


Slika 5.49 Format instrukcije SUB kad je odredište akumulator[2]

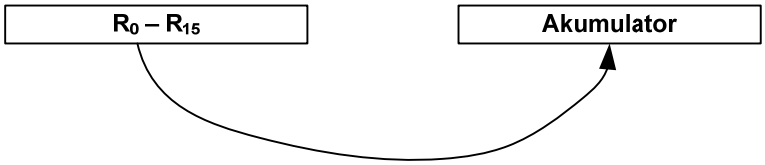


Slika 5.50 Format instrukcije SUB kad su odredište i registar prvog operanda akumulator[3]

MVAC – Kopiranje sadržaja registra u akumulator

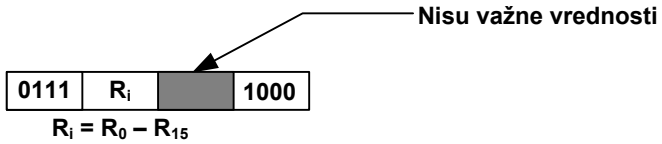


Slika 5.51 Format instrukcije MVAC

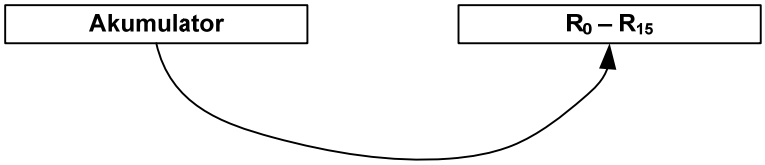


Slika 5.52 Šema izvrešenja instrukcije MVAC

MVR – Kopiranje sadržaja akumulatora u registar

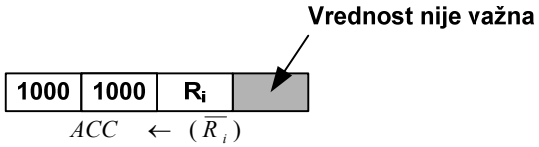


Slika 5.53. Format instrukcije MVR

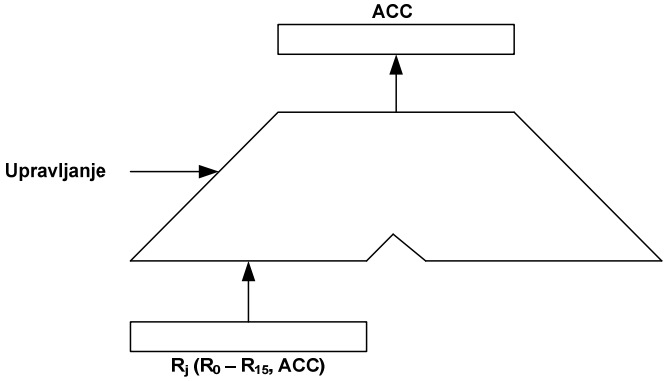


Slika 5.54 Šema izvršavanja instrukcije MVR

NOT – Logička negacija



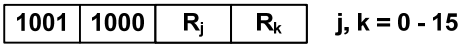
Slika 5.55 Opšti format instrukcije NOT



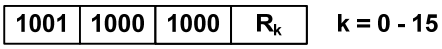
Slika 5.56 Šema izvršavanja instrukcije NOT

OR – Logičko ILI

Slika 5.57 Opšti format instrukcije OR



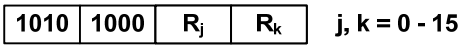
Slika 5.58 Format instrukcije OR kad je odredište akumulator



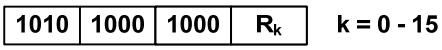
Slika 5.59 Format instrukcije OR kad su odredište i registar prvog operanda akumulator

AND – Logičko I

Slika 5.60 Opšti format instrukcije AND



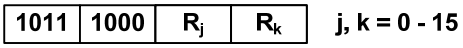
Slika 5.61 Format instrukcije AND kad je odredište akumulator



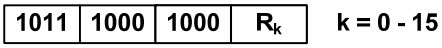
Slika 5.62 Format instrukcije AND kad su odredište i registar prvog operanda akumulator

XOR – Ekskluzivno ILI

Slika 5.63 Opšti format instrukcije XOR

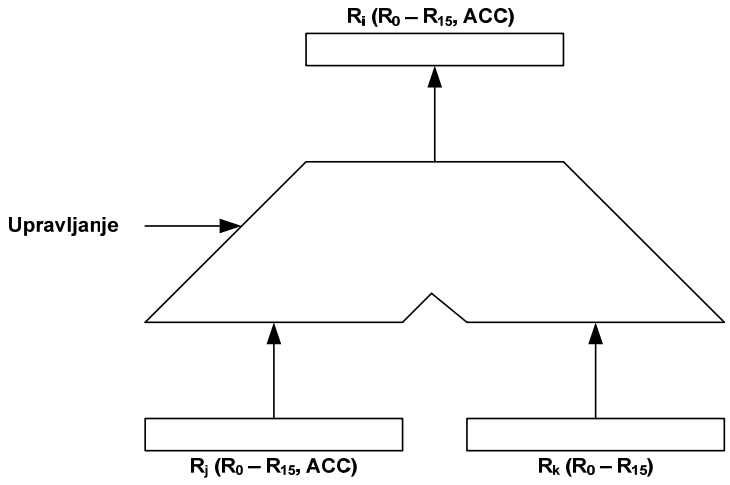


Slika 5.64 Format instrukcije XOR kad je odredište akumulator



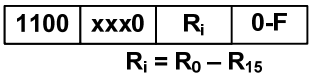
Slika 5.65 Format instrukcije XOR kad su odredište i registar prvog operanda akumulator

Aritmetičko – logičke instrukcije ADD, SUB, OR, AND i XOR izvršavaju se prema šemi na sledećoj slici:

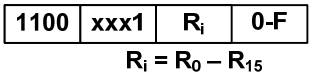


Slika 5.66 Šema izvršavanja aritmemtičko – logičkih operacija

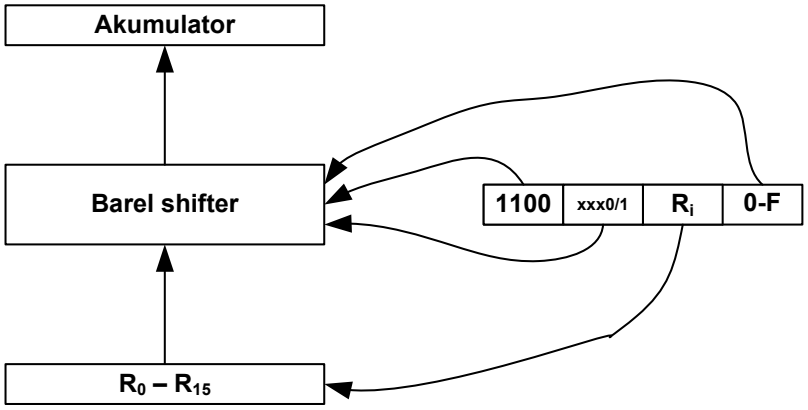
SHIFT – Pomeranje sadržaja registra u levo/desno



Slika 5.67 Format instrukcije SHIFT kod pomeranja sadržaja u levo



Slika 5.68 Format instrukcije SHIFT kod pomeranja sadržaja u desno



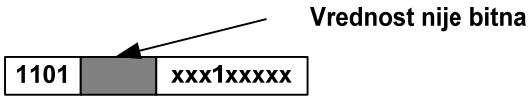
Slika 5.69 Šema izvršenja instrukcije SHIFT

JUMP – Uslovni/bezuslovni skok

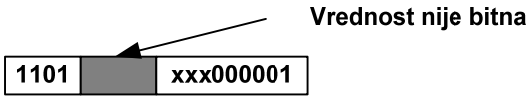
Skokovi kod TFaCo mogu biti bezuslovni ili uslovni. Uslovni skokovi se realizuju na bazi statusnog registra RS (R9) u koji se po izvršavanju operacija upisuje odgovarajuća statusna informacija, prema tabeli 5.18 i slici 5.38.

Instrukcija skoka ima jedinstveni COP, a test za proveru uslova za izvršenje skoka definiše se postavljanjem odgovarajućeg bita u adresnom formatu u skladu sa sadržajem statusnog registra, kao što je prikazano na slici 5.70. U slučaju da se postavi bit **J**, tj. **J=1** instrukcija će se izvršiti kao bezuslovni skok. Izvor adrese skoka uvek je sadržaj **registra skoka (R_{10}).**

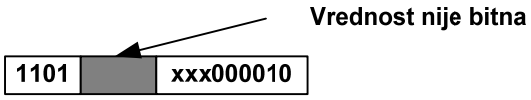
Slika 5.70. Format instrukcije skoka



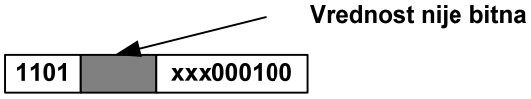
a) Format instrukcije bezuslovnog skoka



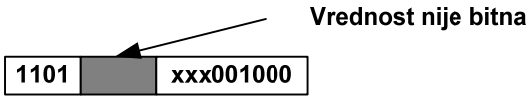
b) Skok ako je rezultat jednak 0



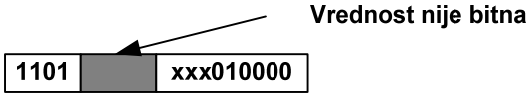
c) Skok ako je rezultat negativan



d) Skok ako je generisan prenos

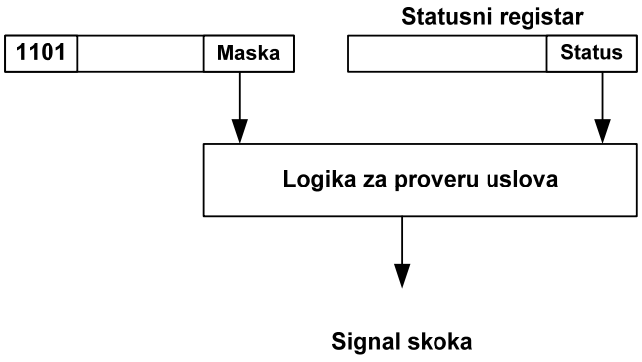


e) Skok ako je došlo do prekoračenja opsega računanja



f) Skok po parnosti

Slika 5.71 Formati uslovnog skoka



Slika 5.72 Šema izvršenja instrukcije skoka

CLR – Upisivanje nula u registar

Slika 5.73 Opšti format instrukcije CLR

HALT – zaustavljanje programa

Slika 5.74 Opšti format instrukcije HALT

5.3.7. Uticaj instrukcija na indikatore u registru RS

Instrukcije LI, LOAD, STORE, MVAC, MVR, CLR

Njihovo dejstvo je sledeće:

- indikator Z (0/1) – na uobičajen način; indikator Z se postavlja na 1 ako je preneseni sadržaj nula, inače je 0.
- indikator N (0/1) – na uobičajen način; indikator N se postavlja na 1 ako je preneseni sadržaj posmatran kao celobrojna veličina sa znakom negativan, inače je nula.
- indikator C (0/1) se ne menja.
- indikator O (0/1) je nula.
- indikator P (0/1) – na uobičajen način; postavlja se 0 ako je broj jedinica u rezultatu paran, inače je 1.

Aritmetičke instrukcije ADD, SUB

Njihovo dejstvo je sledeće:

- indikator Z – postavlja se na 1 ako je rezultat nula, inače je nula.
- indikator N – postavlja se na 1 ako je rezultat posmatran kao celobrojna veličina sa znakom negativan, inače je 0.
- indikator C – postavlja se na 1 ako je bilo prenosa iz najstarijeg razreda u instrukciji ADD, odnosno pozajmice u najstariji razred u instrukciji SUB, inače je 0.
- indikator O – postavlja se na 1 ako je došlo do prekoračenja, inače je 0.
- indikator P – na uobičajen način; postavlja se 0 ako je broj jedinica u rezultatu paran, inače je 1.

Logičke instrukcije NOT, AND, OR, XOR

Njihovo dejstvo je sledeće:

- indikator Z (0/1) – na uobičajen način; indikator Z se postavlja na 1 ako je preneseni sadržaj nula, inače je 0.
- indikator N (0/1) – na uobičajen način; indikator N se postavlja na 1 ako je preneseni sadržaj posmatran kao celobrojna veličina sa znakom negativan, inače je nula.
- indikator C (0/1) se ne menja.
- indikator O (0/1) je nula.
- indikator P – na uobičajen način; postavlja se 0 ako je broj jedinica u rezultatu paran, inače je 1.

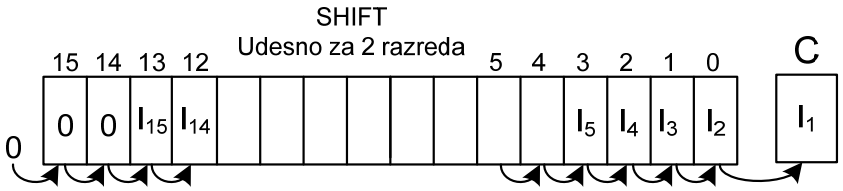
Instrukcija SHIFT

Dejstvo instrukcije SHIFT zavisi od vrednosti 9-tog bita.

- ako je 9-ti bit (I8) jednak 0, onda se sadržaj registra pomera ulevo. Neka se vrši pomeranje npr. 3 bita sadržaja registra ulevo. U najmlađa 3 razreda (I0, I1, I2) se upisuju 0. U indikator C registra RS upisuje se treći najveći razred (I13) (slika 5.75).

Slika 5.75 Sadržaj registra nakon izvršenja instrukcije SHIFT ulevo za 3 razreda

- ako je 9-ti bit (I8) jednak 1, onda se sadržaj registra pomera udesno. Uzmimo da se vrši pomeranje npr. 2 bita sadržaja registra udesno. U najstarija 2 razreda (I15, I14) se upisuju 0. U indikator C registra RS upisuje se drugi najmanji razred (I1) (slika 5.76).



Slika 5.76 Sadržaj registra nakon izvršenja instrukcije SHIFT udesno za 2 razreda

Na ostale indikatore u registru RS instrukcija SHIFT utiče na sledeći način:

- indikator Z (0/1) – na uobičajen način; indikator Z se postavlja na 1 ako je preneseni sadržaj nula, inače je 0.
- indikator N (0/1) – na uobičajen način; indikator N se postavlja na 1 ako je preneseni sadržaj posmatran kao celobrojna veličina sa znakom negativan, inače je nula.
- indikator O (0/1) je nula.
- indikator P – na uobičajen način; postavlja se 0 ako je broj jedinica u rezultatu paran, inače je 1.

Instrukcija JUMP

U zavisnosti od MASKE instrukcija skoka može biti:

- instrukcija bezuslovnog skoka (MASKA=1xxxxx)

Instrukcija bezuslovnog skoka JUMP skače na adresu koja se nalazi u registru RJ i ne utiče na indikatore Z, N, C, O, P u registru RS.

- instrukcija uslovnog skoka, koja ne utiče na indikatore Z, N, C, O, P u registru RS.

Instrukcije NOP i HALT ne utiču na indikatore u registru RS.

6

6. REALIZACIJA PROJEKTA SIMRA

Projekat **SIMRA** je razvijen kao skup pet aplikacija koje, koristeći princip od jednostavnijeg ka složenijem (na osnovu teorije koja je izložena u **Glavi 5**), vizuelizuju rad CPU-a.

Prve dve aplikacije su simulatori namenjeni su za učenike/studente elementarnog znanja i oni predstavljaju vizuelizaciju sinteze arhitekture računara opisane u prethodnoj glavi, **Glavi 5**. Treća i četvrta aplikacija su demonstracije rada programske sekvence bez, odnosno sa memorijom. „Glavna" aplikacija je simulator, nazvan TFaCo, namenjen je za naprednije studente, i predstavlja ponuđeno rešenje jedne arhitekturu procesora, koji je autor ovog rada projektovao. Sve aplikacije nude grafički interfejs za korisnika i daju niz komentara i vizuelnih efekata.

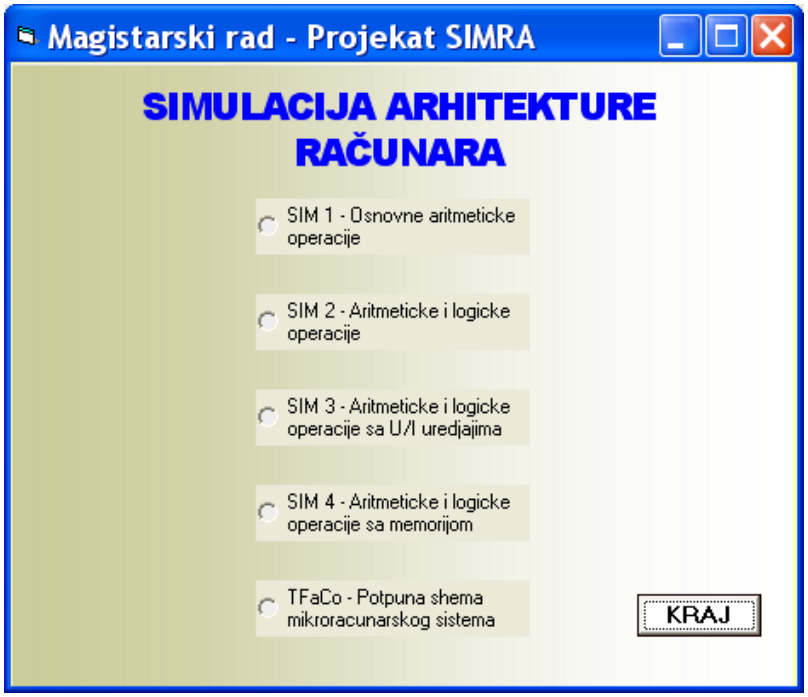
Program je urađen na operativnom sistemu Microsoft Windows XP Professional korišćenjem programskog jezika Microsoft Visual Basic 6 [4], [9], [10], [18]. Hardver na kome je simulacija urađena i testirana je: računar tipa Pentium 4 sa radnom memorijom od 512 MB i integrisanom grafičkom karticom. Programska rešenja, za koje autor smatra da mogu biti korisna za čitaoce ovog rada, data su pomoću listinga procedura/funkcija u glavi **Prilog**.

Prozori simulatora sadrže veliki broj komandnih dugmadi/tastera (CommandButton), opcionih dugmadi (OptionButton), lista izbora (ComboBox), tekstboksova (TextBox), labela (Label), okvira (Frame), slika (Picture).

Radi lakšeg objašnjavanja rada simulatora, u tekstu koji sledi, ovi elementi se označavaju na sledeći način:

- naziv prozora simulatora je velikim slovima i završava uskličnikom, npr. prozor **ARITMETICKE OPERACIJE!**
- imena komandnih, opcionih dugmadi i lista izbora se nalazi između simbola < i simbola > , npr. komandni taster **<Kraj>.**
- imena tekstboksova, labela i okvira su stavljena pod navodnike, npr. „**Izaberite opciju**“ je labela.

  Projekat **SIMRA** startuje se pokretanjem dokumenta **Pocetna,** pri čemu se dobija prozor **MAGISTARSKI RAD – PROJEKAT SIMRA!** koji je dat na slici 6.1. Prozor sadrži 5 opcionih dugmadi i taster **<KRAJ>.** Biranjem opcionog dugmeta bira se jedan od 5 simulatora.



Slika 6.1 Osnovni prozor projekta SIMRA

Biranjem opcionog dugmeta **<SIM 1>,** na veoma uprošćen način, prikazuje se kako CPU obrađuje unešene binarne operande, ne objašnjavajući princip rada. CPU izvršava osnovne računske operacije (sabiranje, oduzimanje, množenje i deljenje), koristeći dva registra i ALU.

Opciono dugme **<SIM 2>** pokreće napredniji simulator, koji vizuelizuju rad ALU-a sa osnovnim aritmetičko – logičkim operacijama, bit po bit.

Nakon izbora opcionog dugmeta **<SIM 3>** na primeru sabiranja dva broja, demonstrira se tok podataka (ulaznih uređaj-ALU-izlaznih uređaj), ali bez korišćenja memorije.

Četvrti simulator se pokreće izborom opcionog dugmeta **<SIM 4>,** gde takođe na primeru sabiranja dva broja, daje detaljniju vizuelizaciju rada CPU-a, uvodeći memoriju, registar instrukcija (IR), programski brojač (PC) i memorijski adresni registar (MAR).

Simulator TFaCo je centralni deo ovog rada i predstavlja detaljni prikaz rada ponuđene arhitekture procesora. On se pokreće potvrđivanjem opcionog dugmeta **<TFaCo>.** Specifikacija projektovanog procesora TFaCo data je u **poglavlju 5.3**, a ovde je izložen prikaz vizuelizacije rada tog procesora.

Svaki simulator je opisan kako radi i za svaki simulator dat je rezime (prednosti i nedostaci, kao i koja poboljšanja treba izvršiti).

6.1. SIM 1 – OSNOVNE ARITMETIČKE OPERACIJE

Ovo je simulator koji ne zahteva posebno predznanje. On treba da približi korisnicima, a to su prvenstveno učenici osnovne škole, binarni jezik računara. Učenik unosi neoznačene cele binarne brojeve, bira jednu od četiri računske operacije i dobija rezultat.

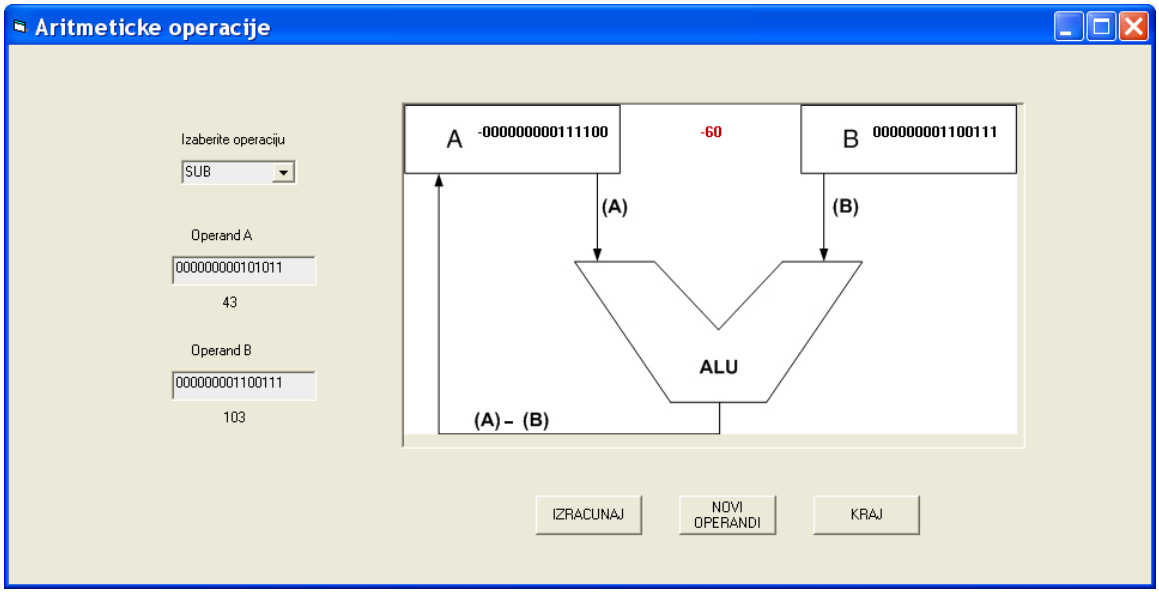
6.1.1. Realizacija SIM 1

Potvrđivanjem dugmeta **<SIM 1>** na slici 6.1 dobija se prozor **ARITMETICKE OPERACIJE!**. Prozor ove simulacije sadrži:

- Dva tekstualna polja (**„Operand A“** i **„Operand B“**), gde korisnik unosi binarne vrednosti za operande A i B, što znači da korisnik ne može da unese druge simbole osim 0 ili 1. Maksimalna dužina operanada A i B je 15-bita, što znači da maksimalan broj koji može da se unese, je decimalni broj 32767 (2^{15}-1). Korisnik može da unese i manji broj cifara, u tom slučaju će operandima biti dodate nebitne nule, počev od najvišeg bita.
- Listu izbora <**Izaberi operaciju**>, gde korisnik bira jednu od aritmetičkih operacija, **ADD** – sabiranje, **SUB** – oduzimanje **MUL** - množenje, **DIV** – deljenje. Moguće je za iste operande birati različite operacije.
- Dva registra **A** i **B**, u koje se upisuju podaci nakon izvršenja izabrane računske operacije. Registri su 16-bitni, gde je bit najveće težine predviđen za znak (pošto rezultat može da bude negativan). Ako je rezultat negativan za znak broja će biti upisan znak „-„
- **ALU** opšte namene čiji je zadatak da izvrši izabranu računsku operaciju nad unešenim operandima. Sadržaji registara **(A)** i **(B)** se prenose u ALU i na njima se primenjuje izabrana aritmetička operacija.

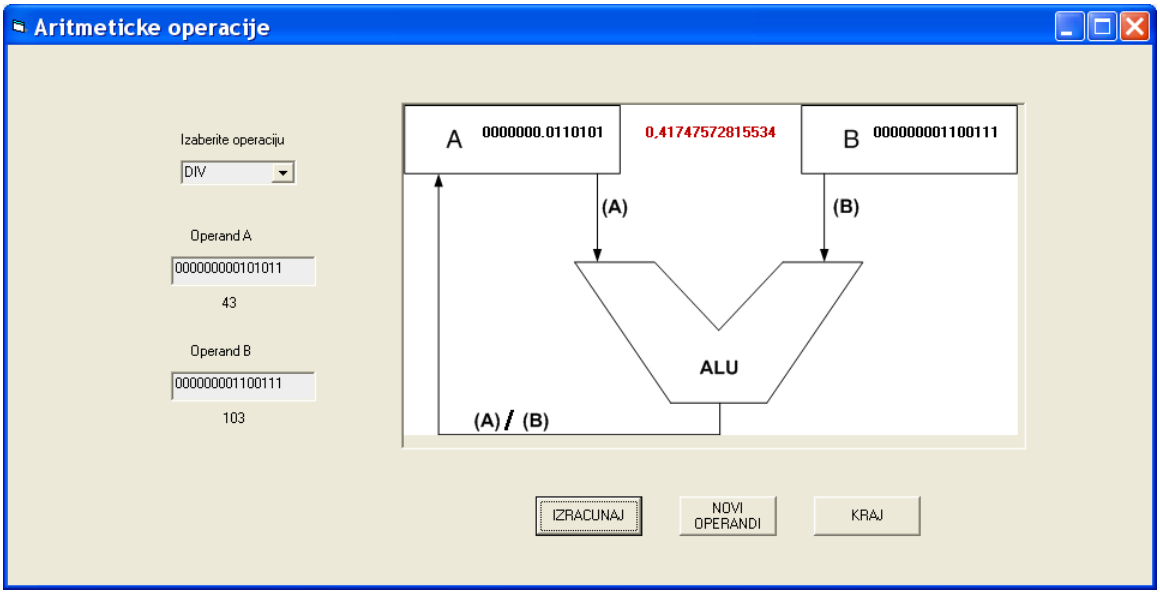
- Taster **<IZRAČUNAJ>.** Potvrđivanjem ovog tasterta se izvršava računska operacija nad operandima A i B. U registar B se upisuje vrednost operanda B, dok se u registar A upisuje rezultat operacije. Ispod polja **„Operand A“** i **„Operand B“** ispisuju se decimalne vrednosti unešenih operanada, dok se decimalni rezultat ispisuje između registara A i B.
- Taster **<NOVI OPERANDI>.** Potvrđivanjem ovog tasterta brišu se vrednosti operanada A i B, kao i sadržaj registara A i B. Time se može izvršiti unos novih operanada i biranje nove računske operacije.
- Taster **<KRAJ>** omogućuje povratak na osnovni prozor projekta (na sliku 6.1)

Na slici 6.2 dat je izgled prozora ovog simulatora za jedan test primer. Uneti su binarni brojevi 101011 **(“Operand A”)** i 11001111010 **(“Operand B”)** (decimalno 43 i 103). Potom je iz liste izbora **<Izaberite operaciju>** izabrana računska operacija **SUB** (oduzimanje) i potvrđen je taster **<IZRACUNAJ>**. Rezultat -000000000111100 (-60) upisan je u registar **A**. U slučaju da dođe do prekoračenja (prilikom sabiranja ili množenja) u registar **A** će biti upisano „*prekoracenje*”.



Slika 6.2 Prozor simulatora SIM 1 – Oduzimanje dva broja

Ako se na iste operande primeni aritmetička operacija deljenja (**DIV**), prozor će imati izgled kao na slici 6.3.



Slika 6.3 Prozor simulatora SIM 1 – Deljenje dva broja

6.1.2. Rezime

Nivo simulatora: **Početni (osnovna škola)**

Prednosti	Nedostaci	Poboljšanja
▪ Upoznavanje binarnog brojnog sistema (sistem koji ima samo dva simbola). ▪ Upoznavanje osnovnih aritmetičkih operacija u asemblerskom zapisu. ▪ Saznanje da i računar ima ograničenja.	▪ Brojevi, koji se unose, su neoznačeni (pozitivni). ▪ Rezultat, koji je negativan, zapisuje se kao označeni broj, pomoću znaka i apsolutne vrednosti. ▪ Simuliraju se samo aritmetičke operacije ▪ Korisniku ove simulacije ne prikazuje se kako se u ALU izvršavaju operacije, na nivou bita, korak po korak.	▪ Omogućiti rad sa označenim brojevima, uz korišćenje drugog komplementa. ▪ Simulirati i logičke operacije. ▪ Treba na nivou bita pokazati kako se izvršavaju osnovne aritmetičko – logičke operacije.

Broj linija kôda: **300**

6.2. SIM 2 – ARITMETIČKO - LOGIČKE OPERACIJE

Ovo je napredniji simulator i on treba da pojasni korisniku kako CPU izvršava aritmetičko – logičke operacije, na nivou bita. Kada se na slici 6.1 potvrdi taster SIM 2, dobiće se prozor **ARITMETICKE I LOGICKE OPERACIJE!** (slika 6.4).

Slika 6.4 Osnovni prozor simulatora SIM 2

Dobijeni prozor omogućuje sledeće:

- Opcija **<Aritmetičke operacije>** simulaciju aritmetičkih operacija na nivou bita
- Opcija **<Logičke operacije>** simulaciju logičkih operacija na nivou bita
- Taster **<KRAJ>** povratak na osnovni prozor projekta SIMRA

6.2.1. Realizacija SIM 2 - Aritmetičke operacije

Potvrđivanjem opcije **<Aritmeticke operacije>**, dobija se prozor kao na slici 6.5. Prozor sadrži sledeće elemente:

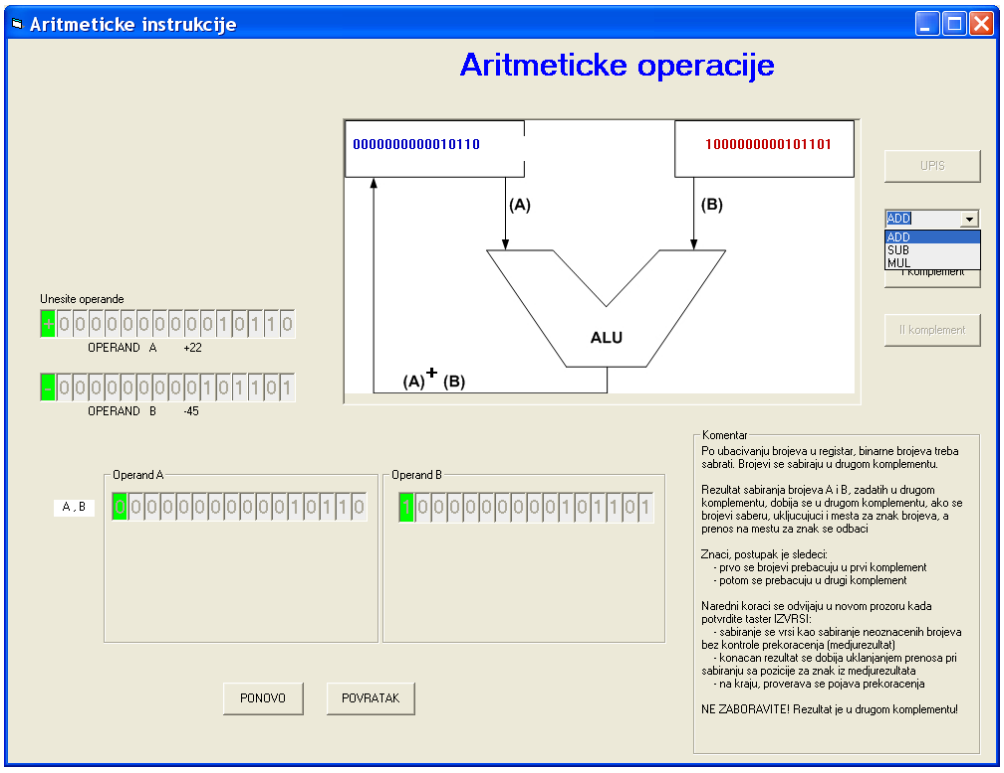
- **„Unesite operande“** – Omogućava unos označenih 16-bitnih brojeva (**„OPERAND A“** i **„OPERAND B“**). Unos cifara je ograničen sa 0 ili 1. Znak broja se unosi na mesto bita najveće težine (bit osenčen zelenom bojom). Za pozitivne brojeve se unosi znak „+“ ili se ne unosi ništa, dok se za negativne brojeve unosi znak „-„.
- Taster **<UPIS>**. Ako su za operande A i B unešeni binarni brojevi koji imaju manje od 15 cifara (što je slučaj u ovom primeru), operandi se dopunjuju sa nevažećim nulama. Potom se dopunjeni operandi upisuju u registre **A** i **B** (polja **„Operand A“** i **„Operand B“** ispod slike), tako što se za znak broja, na mestu bita najveće težine, za pozitivan broj upisuje 0, a za negativne 1.

- Lista izbora **<I komplement>** omogućuje izbor aritmetičke operacije**.**
- Okvir **„Komentar“.** Posle izbora aritmetičke operacije ispisuje se prateći komentar za izabranu aritmetičku operaciju.
- Taster **<I komplement>**. Kako se operacije izvode nad brojevima u drugom komplementu, potrebno je simulirati konverziju u prvi, a potom u drugi komplement. Potvrđivanjem ovog tastera brojevi se prebacuju u prvi komplement i upisuju se u polja **„Operand A“**, odnosno **„Operand B“**.
- Taster **<II komplement>.** Nakon prebacivanja brojeva u prvi komplement, izborom ovog tastera brojevi se prebacuju u drugi komplement.
- Polja **„Operand A“** i **„Operand B“** u koja se redom upisuju vrednosti operanada **A** i **B**, potom vrednosti prvog komplementa (**C1(A)** i **C1(B)**), i na kraju drugog komplementa (**C2(A)** i **C2(B)**) operanada A i B.

Sabiranje/oduzimanje binarnih brojeva

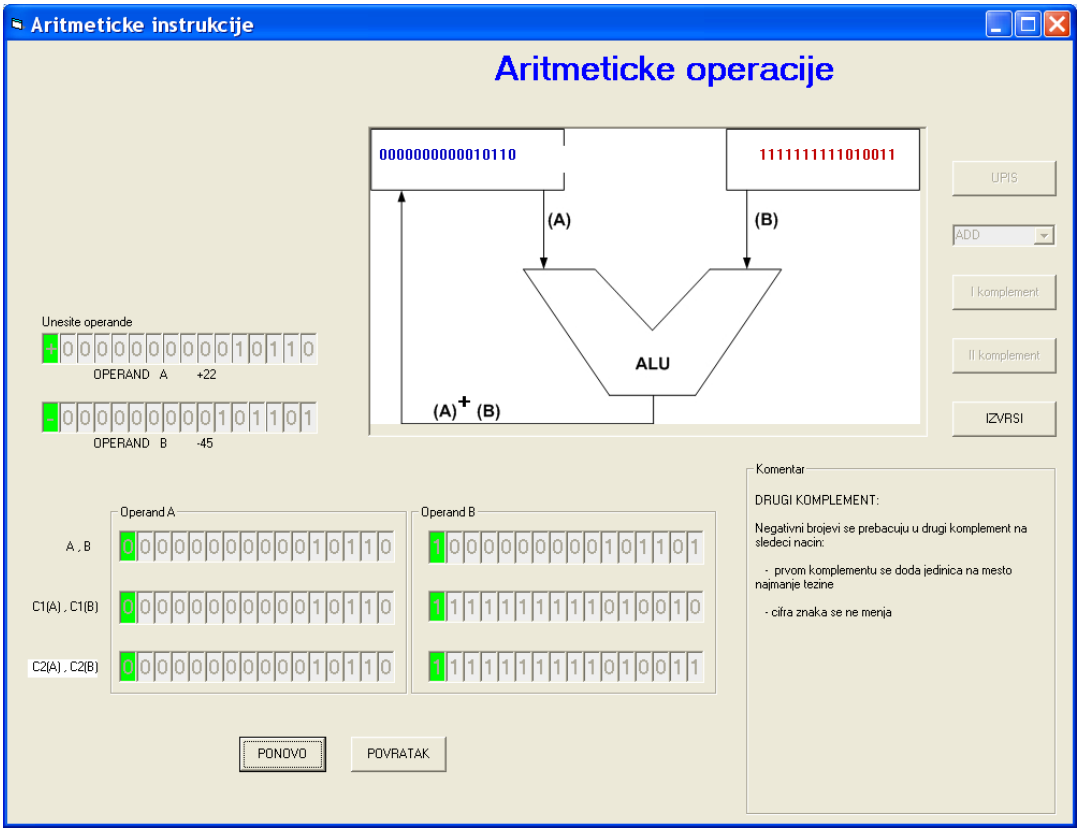
Kako se binarni brojevi sabiraju (oduzimaju) objašnjeno je u **Glavi 5**. Kao primer, uneti su binarni brojevi **+22** i **-45**. Potvrđeno je taster **<UPIS>** i izabrana je aritmetička operacija **ADD** (sabiranje). Slika 6.5 daje prikaz nakon izbora operacije sabiranja.

Slika 6.6 daje prikaz nakon konverzije broja u prvi, odnosno drugi komplement. Postupak prebacivanja brojeva u prvi, odnosno drugi komplement, takođe je dat u **Glavi 5**.

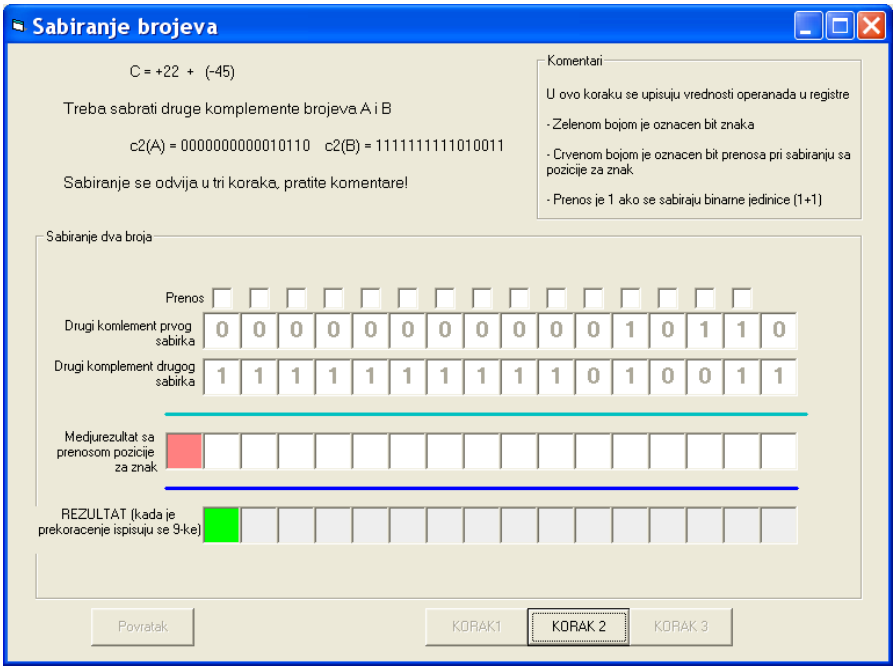


Slika 6.5 Simulator SIM 2, Aritmetičke operacije – Unos i upis brojeva

Sada su operandi spremni za da se nad njima izvrši operacija sabiranja. Nakon potvrđivanja tasterta **<IZVRSI>** otvoriće se prozor **SABIRANJE BROJEVA!**. Sabiranje dva broja zapisana u drugom komplementu, simulirano je u tri koraka. U prvom koraku (**<KORAK 1>**) brojevi se upisuju u registre (slika 6.7).

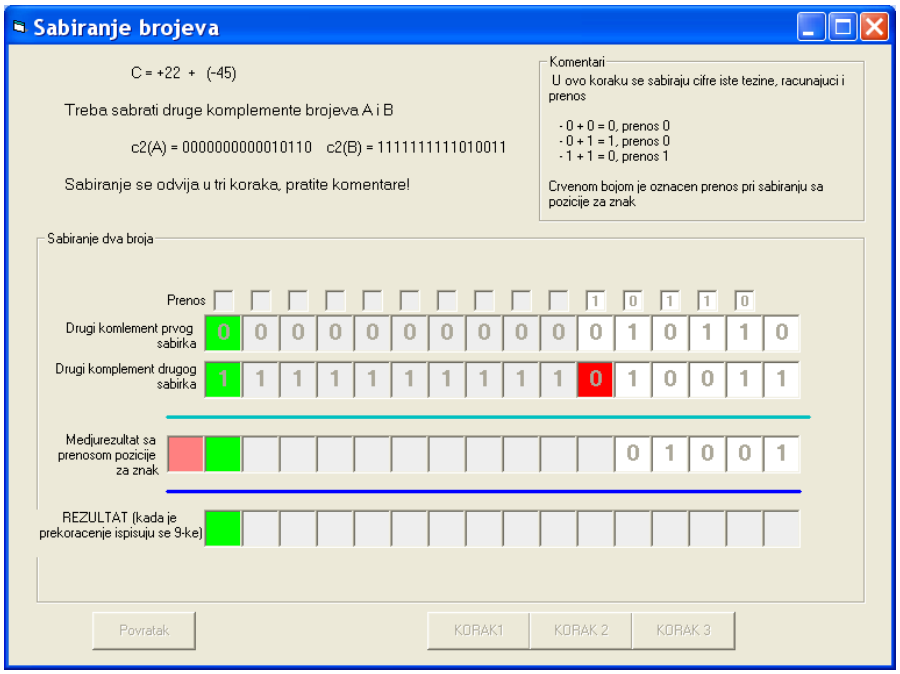


Slika 6.6 Simulator SIM 2, Aritmetičke operacije – Komplementiranje unetih brojeva



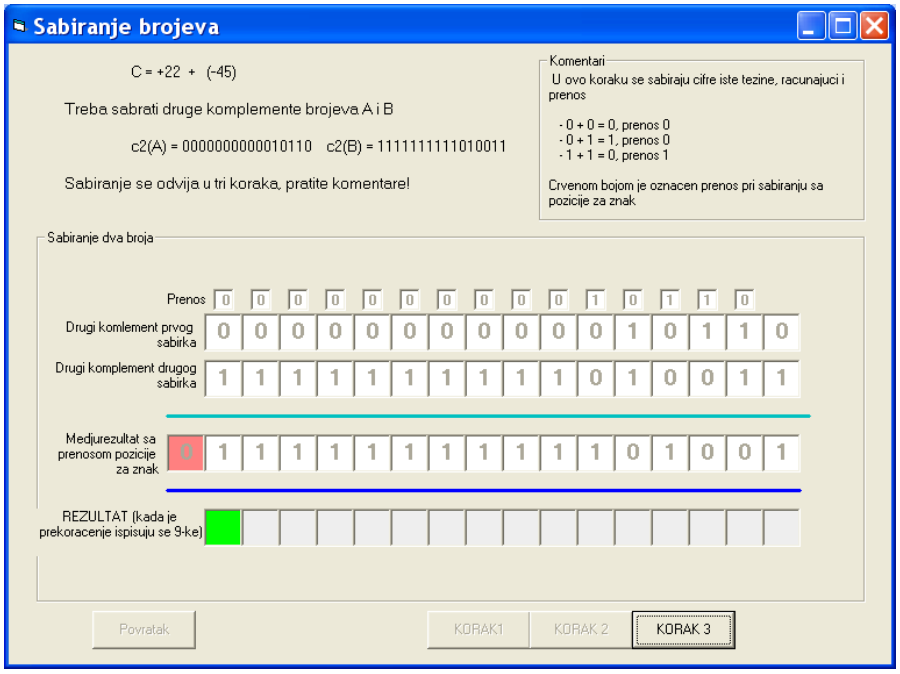
Slika 6.7 Simulator SIM 2, Aritmetičke operacije – Sabiranje brojeva, KORAK 1

U drugom koraku (**<KORAK** 2>) sabiraju se cifre iste težine računajući i prenos. Simulacija teče tako što se prvo osvetli bit prvog operanda, potom bit iste težine drugog operanda, rezultat se upisuje u bit iste težine međurezultata i upisuje se i ostvareni prenos („**Prenos”**). Slika 6.8 daje prikaz „zamrznute” simulacije sabiranja bitova na mestu težine pet.



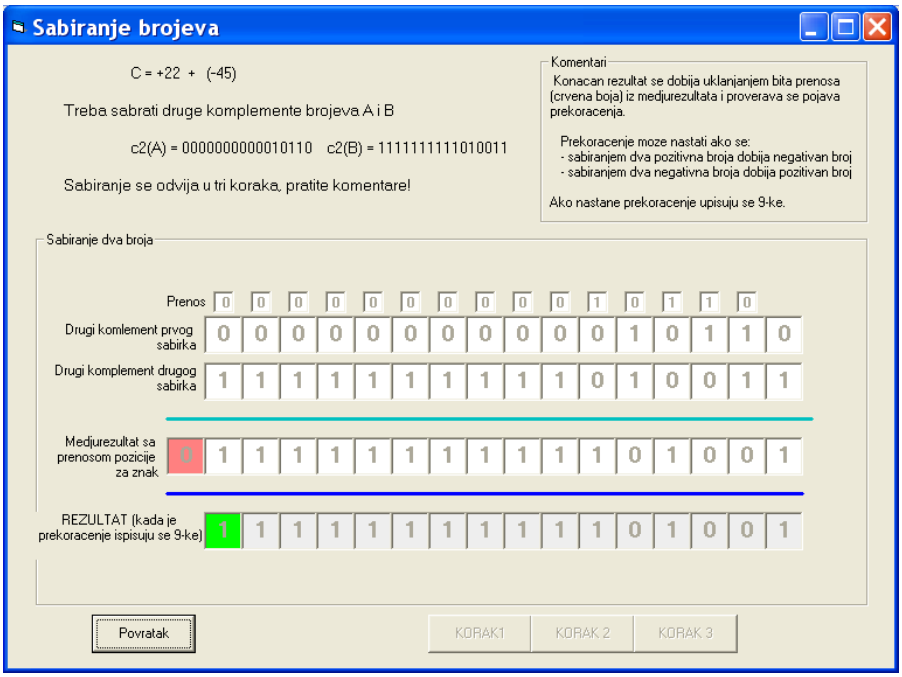
Slika 6.8 Simulator SIM 2, Aritmetičke operacije – Sabiranje brojeva, KORAK 2 (međukorak)

Slika 6.9 daje izgled prozora po završetku drugog koraka. Crvenom bojom je označen prenos pri sabiranju sa pozicije za znak.



Slika 6.9 Simulator SIM 2, Aritmetičke operacije – Sabiranje brojeva, KORAK 2 (kraj)

Po završetku drugog koraka potrđuje se taster **<KORAK 3>**, koji daje prikaz konačnog rezultata. Konačan rezultat se dobija uklanjanjem bita prenosa (crvena boja) iz međurezultata i proverava se pojava prekoračenja. Prekoračenje može nastati ako se sabiranjem dva pozitivna broja dobije negativan broj ili sabiranjem dva negativna broja dobije pozitivan broj. U slučaju ovog primera koji se razmatra nije došlo do prekoračenja (slika 6.10).



Slika 6.10 Simulator SIM 2, Aritmetičke operacije – Sabiranje brojeva, KORAK 3

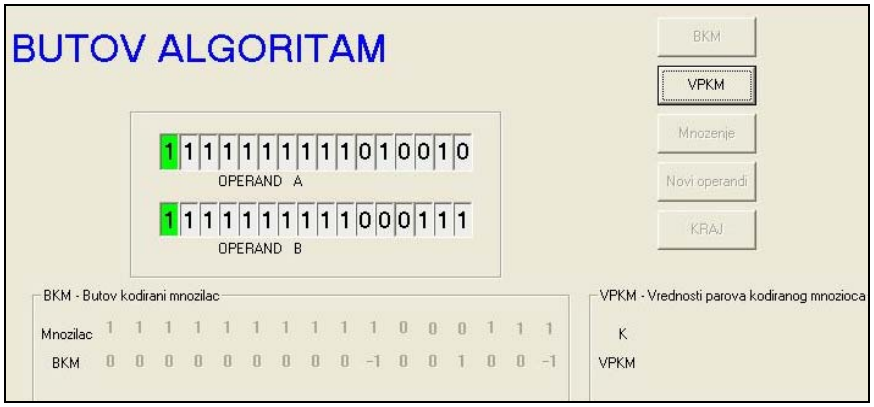
Taster **<Povratak>**, omogućava povratak u prozor **ARITMETICKE INSTRUKCIJE!** (slika 6.5).

Množenje binarnih brojeva

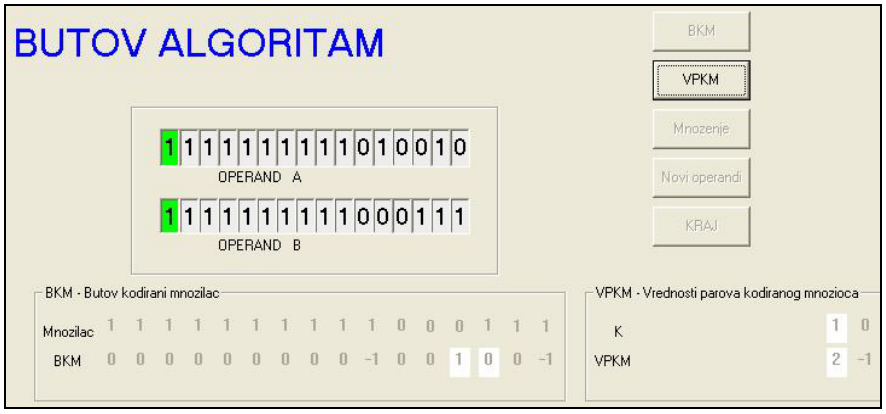
Nakon povratka u prozor **ARITMETICKE INSTRUKCIJE!** (slika 6.5), potvrđivanjem tastera **<Ponovo>**, postupak se ponavlja za nove operande i izbor nove aritmetičke operacije. Za nove operande A = -101110 (-46) i B = -111001 (-57), izabrana je operacija množenja binarnih brojeva (MUL). Postupak množenja označenih brojeva, primenom modifikovanog Booth-ovog algoritma, dat je u **poglavlju 5.1**.

Analogno predhodnom primeru sabiranja dva broja, nakon upisa operanada A i B, bira se operacija MUL, potom se brojevi prebacuju u prvi i drugi komplement. Nakon toga potvrđuje se taster **<IZVRSI>**. Otvara se prozor **MNOZENJE BROJEVA!**. U dobijenom prozoru se aktivira taster **<BKM>** čime se određuje **„Butov Kodirani Mnozilac“** (BKM = 000000000-100100-1). Rezultat ovog postupka prikazan je na slici 6.11.

Sledi aktiviranje tastera **<VPKM>** koji omogućava određivanje **„Vrednosti Parova Kodiranog Mnozioca“**. Pošto je dužina zapisa 16, vrednosti za k pripadaju intervalu [0, 7]. Slika 6.12 vizuelno prikazuje nalaženje vrednosti para po formuli: $VPKM_k = 2*BKM_{2k+1} + BKM_{2k}$. Za k = 1 dobija se: $VPKM_1 = 2*BKM_3 + BKM_2 = 2*1+0 = 2$.

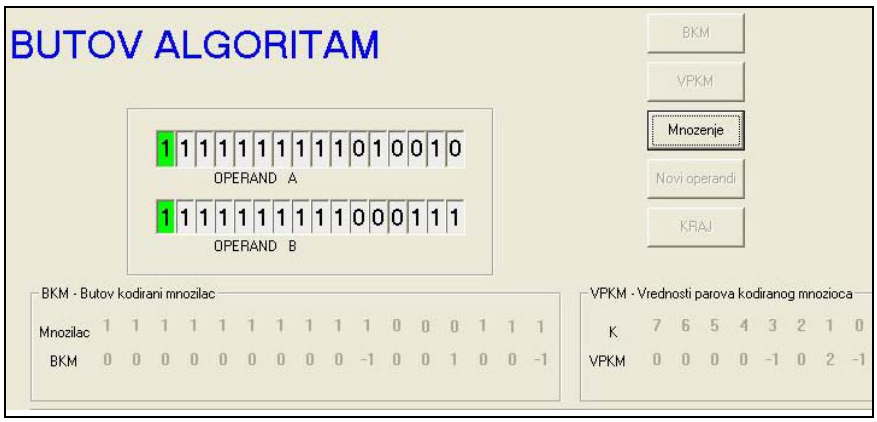


Slika 6.11 Simulator SIM 2, Aritmetičke operacije – Množenje, Određivanje BKM



Slika 6.12 Simulator SIM 2, Aritmetičke operacije – Množenje, Nalaženje VPKM za k=2

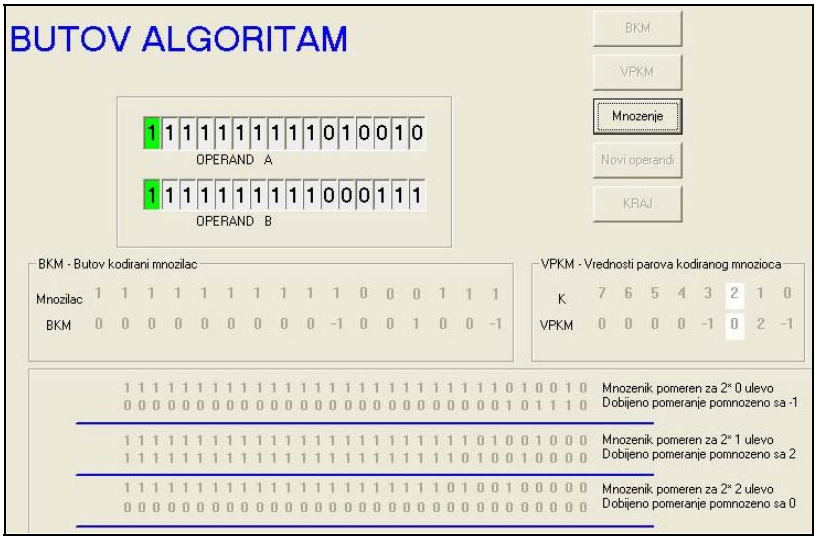
Na istovetan način (dato na slici 6.13) određuju se i ostale vrednosti **„VPKM“** za **„K“** iz intervala [0, 7].



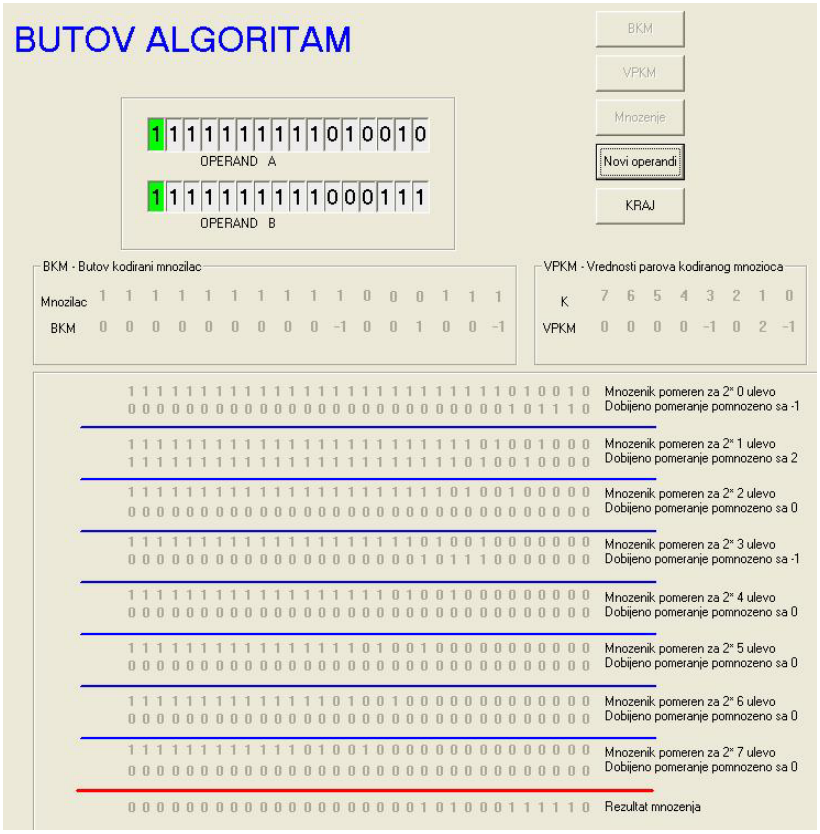
Slika 6.13 Simulator SIM 2, Aritmetičke operacije – Množenje, Vrednosti VPKM za k∈ [0, 7]

Rezultujući proizvod se zapisuje kao 32-bitni binarni broj. Pritiskom na taster **<Mnozenje>** započinje ovaj proces. Vrednost množenika se u svakom koraku (0, 1, ..., 7) pomera za 2k mesta ulevo i množi se VPKM-om i na taj način se dobija parcijalni sabirak za taj korak.

Slika 6.14 prikazuje nalaženje parcijalnog sabirka za k = 2, dok slika 6.15 daje konačan rezultat množenja sabiranjem parcijalnih sabiraka kroz sve korake.



Slika 6.14 Simulator SIM 2, Aritmetičke operacije – Množenje, Parcijalni sabirak za k = 2



Slika 6.15 Simulator SIM 2, Aritmetičke operacije – Množenje,Konačan rezultat množenja

Može se zaključiti da modifikovani Booth-ov algoritam korektno radi za date označene brojeve i dobija se:

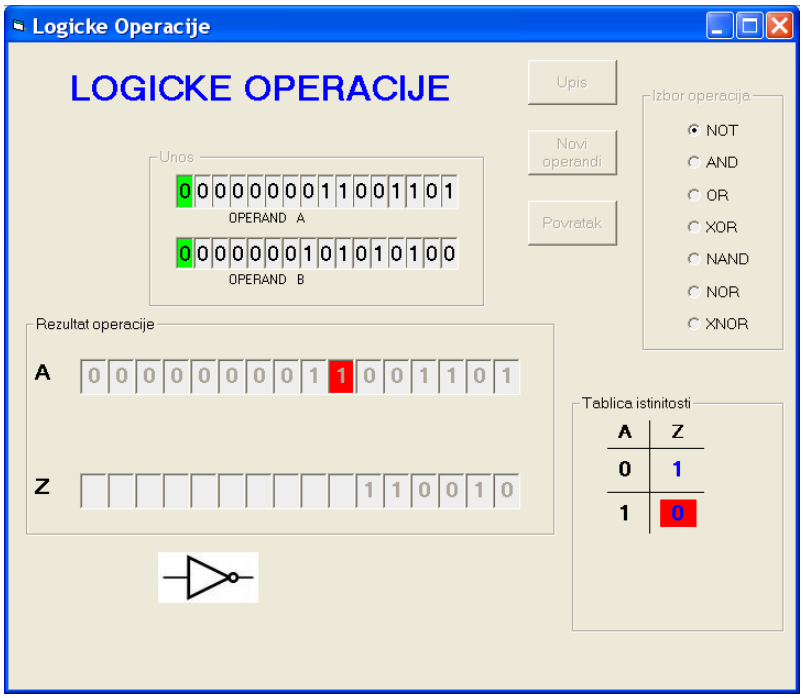
$$Z = (-46) * (-57) = (1111111111010010)_2 * (1111111111000111)_2$$
$$= (00000000000000000001010001111110)_2$$

6.2.2. Realizacija SIM 2 - Logičke operacije

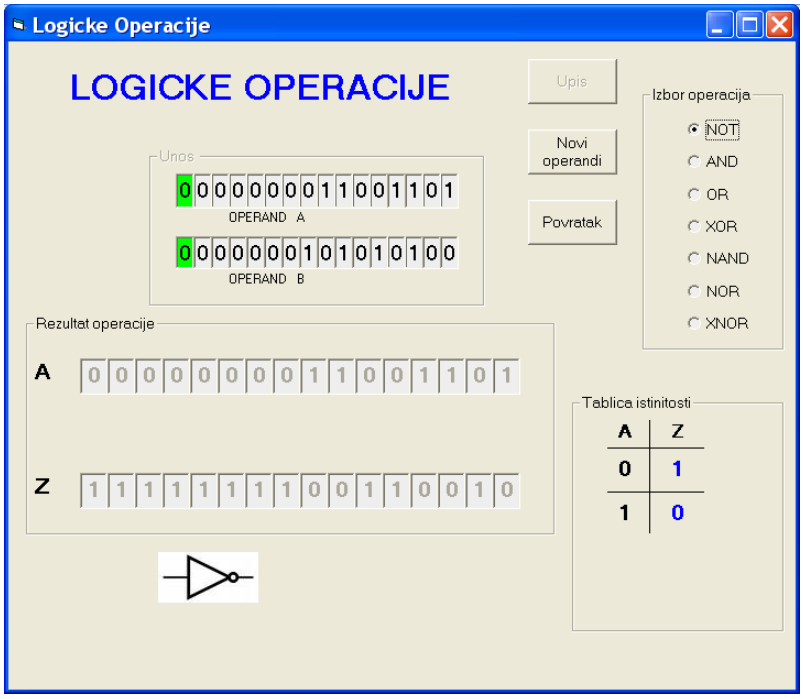
Posle izvršenih aritmetičkih operacija i povratka u prozor **ARITMETICKE I LOGICKE OPERACIJE!** (slika 6.4), potvrđivanjem opcije **<Logicke operacije>**, dobija se prozor **LOGICKE OPERACIJE!**, koji omogućuje sledeće:

- Unos operanada A i B (**"Unos"**), na već poznat način
- Upis operanada u registre (**<Upis>**)
- Izbor logičke operacije (**„Izbor operacija"**)
- Operacije sa novim operandima (**<Novi operandi>**)
- Povratak u glavni prozor simulacije SIM 2 (**<Povratak>**)
- Prikaz operanada koji učestvuju u opeaciji i rezultata operacije (**„Rezultat operacije"**)

Kao primer unešena su dva pozitivna broja. Nakon izbora operacije (**<NOT>** – negacija), prikazuje se **„Tablica istinitosti"** za tu operaciju, grafički simbol za tu operaciju (ispod **„Rezultat operacije"**) i započinje proces negacije. Slika 6.16 prikazuje postupak negacije u toku procesa, dok slika 6.17 prikazuje završetak procesa negacije. U registar „**Z**" se upisuje rezultat izabrane logičke operacije.

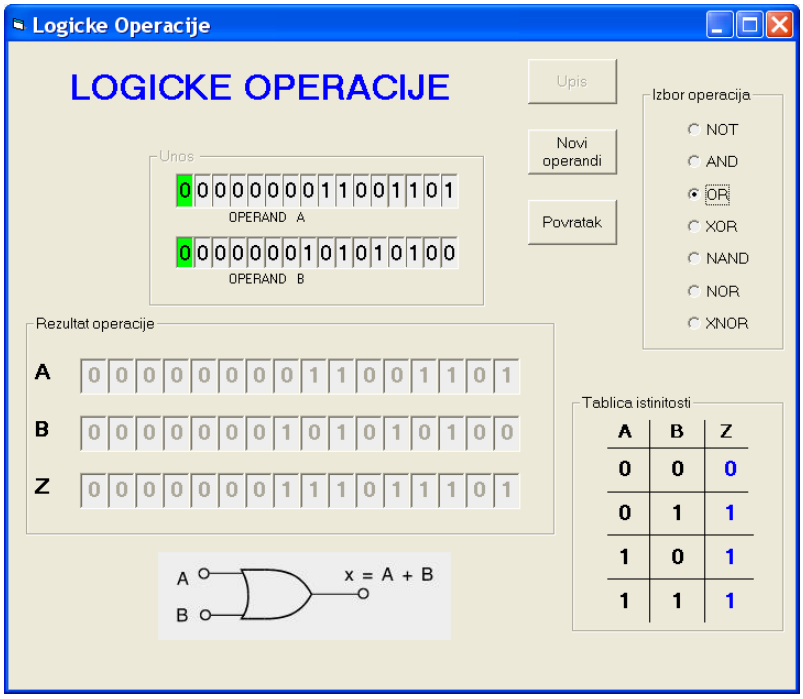


Slika 6.16 Simulator SIM 2, Logičke operacije – NOT (Postupak u toku)

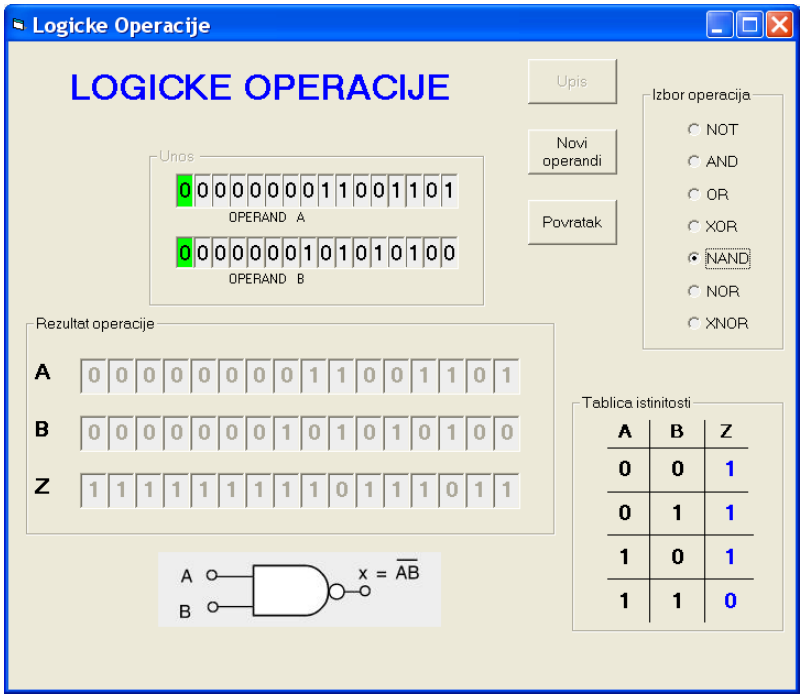


Slika 6.17 Simulator SIM 2, Logičke operacije – NOT (Kraj)

Slike 6.18 i 6.19 prikazuju rezultat logičkih operacija **<OR>** (logičko ILI) i **<NAND>** (NI kolo) za iste operande A i B.



Slika 6.18 Simulator SIM 2, Logičke operacije – OR



Slika 6.19 Simulator SIM 2, Logičke operacije – NAND

Kao što se primećuje, logička operacija NOT (komplementiranje-negacija) se definiše (primenjuje) nad jednom logičkom promenljivom (operandom), dok se ostale logičke operacije primenjuju nad dve logičke promenljive.

6.2.3. Rezime

Nivo simulatora: **Srednji (srednja škola)**

Prednosti	Nedostaci	Poboljšanja
▪ Aritmetičke operacije se izvode sa brojevima u drugom komplementu ▪ Korisnik na nivou bita može da vidi kako se izvršavaju osnovne aritmetičko – logičke operacije. ▪ Detaljna vizuelizacija aritmetičkih i logičkih operacija	▪ Korisnik nije objašnjeno odakle se unose podaci sa kojima se operiše ▪ Korisnik ne zna gde se u toku operacije sa podacima čuvaju njihovi međurezultati. ▪ Nije prikazano kako se operacije izvršavaju u ALU. ▪ Nije objašnjeno gde se prikazuje konačni rezultat.	▪ Uvesti U/I uređaje ▪ Uvesti aritmetičko-logičku jedinicu ▪ Pojasniti rad sa regisitrima

Broj linija kôda: **3300**

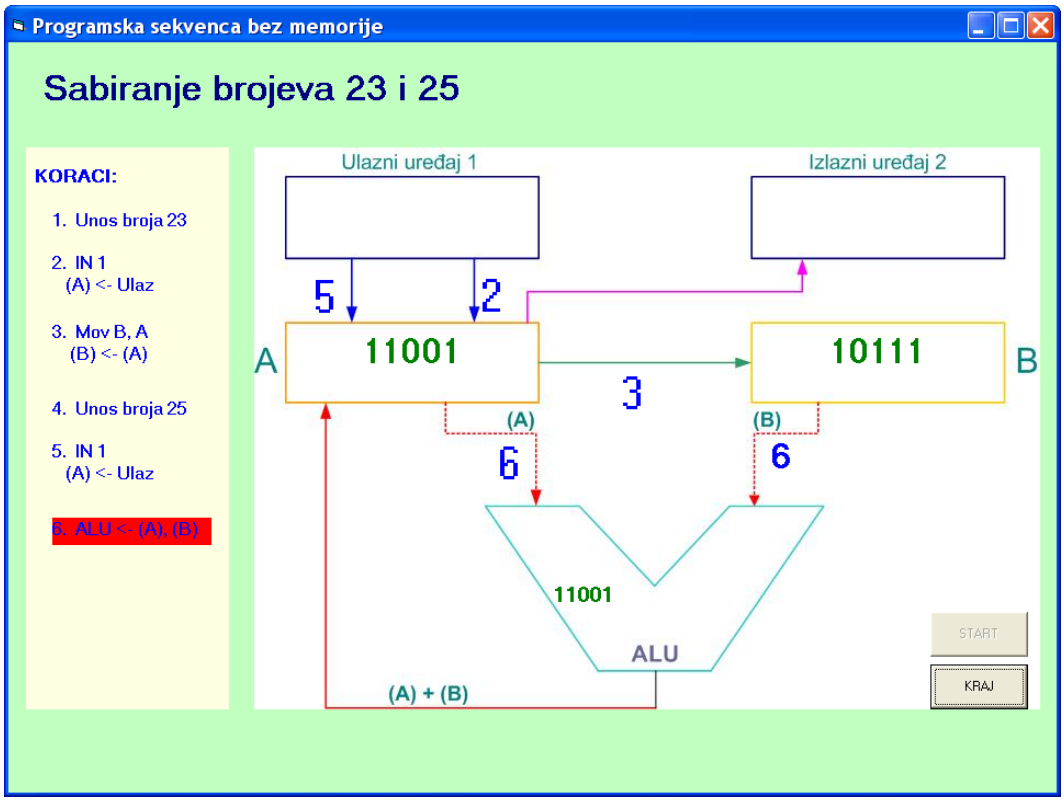
6.3. SIM 3 – PROGRAMSKA SEKVENCA BEZ MEMORIJE

Ovaj simulator treba da pokaže korisnicima na primeru sabiranja dva broja, simulirajući korak po korak, tok podataka od ulaznih uređaja, preko ALU, do izlaznih uređaja, ali bez korišćenja memorije.

6.3.1. Realizacija SIM 3

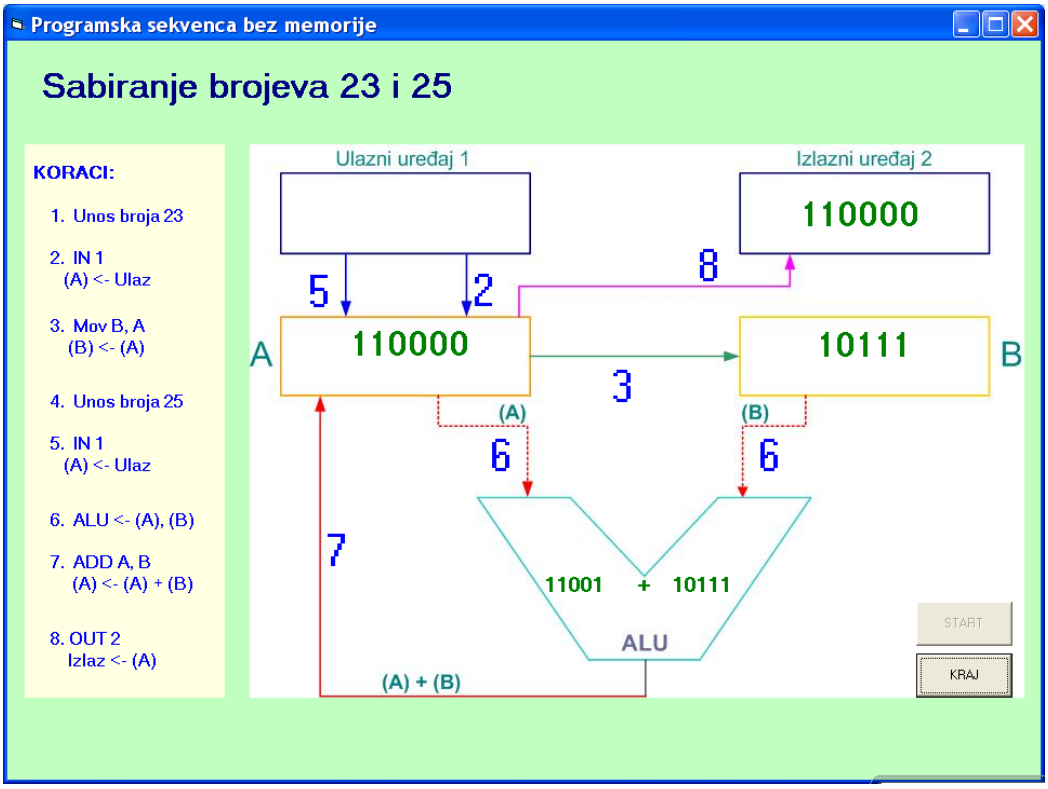
Potvrđivanjem **<SIM 3>** na slici 6.1 dobija se prozor **PROGRAMSKA SEKVENCA BEZ MEMORIJE!**. Prozor ove demonstracije sadrži samo dva komandna tastera, **<START>**, koji startuje simulaciju i **<KRAJ>**, koji omogućuje povratak na prozor projekta **SIMRA**.

Za demonstraciju sabiranja dva broja uzeti su brojevi, 23 i 25. Njihovi binarni zapisi su 10111 i 11001. Nakon potvrđivanja tastera **<START>** demonstracija kreće i odvija se u 8 koraka. Akcija u svakom koraku se prati u okviru **„KORACI:“**, dok se na slici demonstracije svaki korak prikazuje brojevima **1, 2..**. Tako npr. u 6. koraku brojevi se unose u ALU, pa se u sledećem 7. koraku brojevi sabiraju i rezultat se smešta u registar A. Slika 6.20 prikazuje 6. korak, koji nije do kraja izvršen, jer je prvi broj iz registra A unešen u ALU, dok drugi (iz registra B) nije.



Slika 6.20 Demonstrator SIM 3, Korak unošenja drugog broja u ALU

Slika 6.21 daje konačni izgled nakon izvršenja svih koraka.



Slika 6.21 Demonstrator SIM 3, Konačan rezultat sabiranja

6.3.2. Rezime

Nivo simulatora: **Srednji (srednja škola)**

Prednosti	Nedostaci	Poboljšanja
▪ Da bi se sabrala dva broja nije dovoljna samo jedna instrukcija (ADD), već 4 (još IN, MOV i OUT). ▪ Korisniku se, kroz korake, vizuelno pokazuje gde sve podaci treba da „prođu“, da bi se nad njima izvršila operacija sabiranja. ▪ Svaki korak je dokumentovan a na slici simulatora numerisan. ▪ Ako želimo da pojasnimo neki korak, vizuelizacija se može pauzirati (potvrđivanjem tastera Alt), a potom nastaviti (potvrđivanjem istog tastera).	▪ Ovo je demonstracija (animacija) sabiranja dva broja, tj. korisnik ne može da unese brojevi i upisuju instrukcije ▪ U ovom primeru koriste se četiri instrukcije (IN, MOV, ADD, OUT). Nije objašnjeno gde je zabeležen program i koji je redosled tih instrukcija? ▪ Češći su slučajevi da se brojevi učitavaju iz memorije, odnosno da se rezultat smešta u memoriju, nego preko U/I uređaja.	▪ Uvesti memoriju. ▪ Uvesti programski brojač (PC). ▪ Uvesti memorijsko adresni registar (MAR). ▪ Uvesti registar instrukcija (IR).

Broj linija kôda: **300**

6.4. SIM 4 – PROGRAMSKA SEKVENCA SA MEMORIJOM

Na primeru sabiranja istih brojeva, kao kod treće aplikacije (demonstrator), ovo je takođe demonstracija koja daje detaljniju vizuelizaciju rada CPU-a, uvodeći memoriju, registar instrukcija (IR), programski brojač (PC) i memorijski adresni registar (MAR).

6.4.1. Realizacija SIM 4

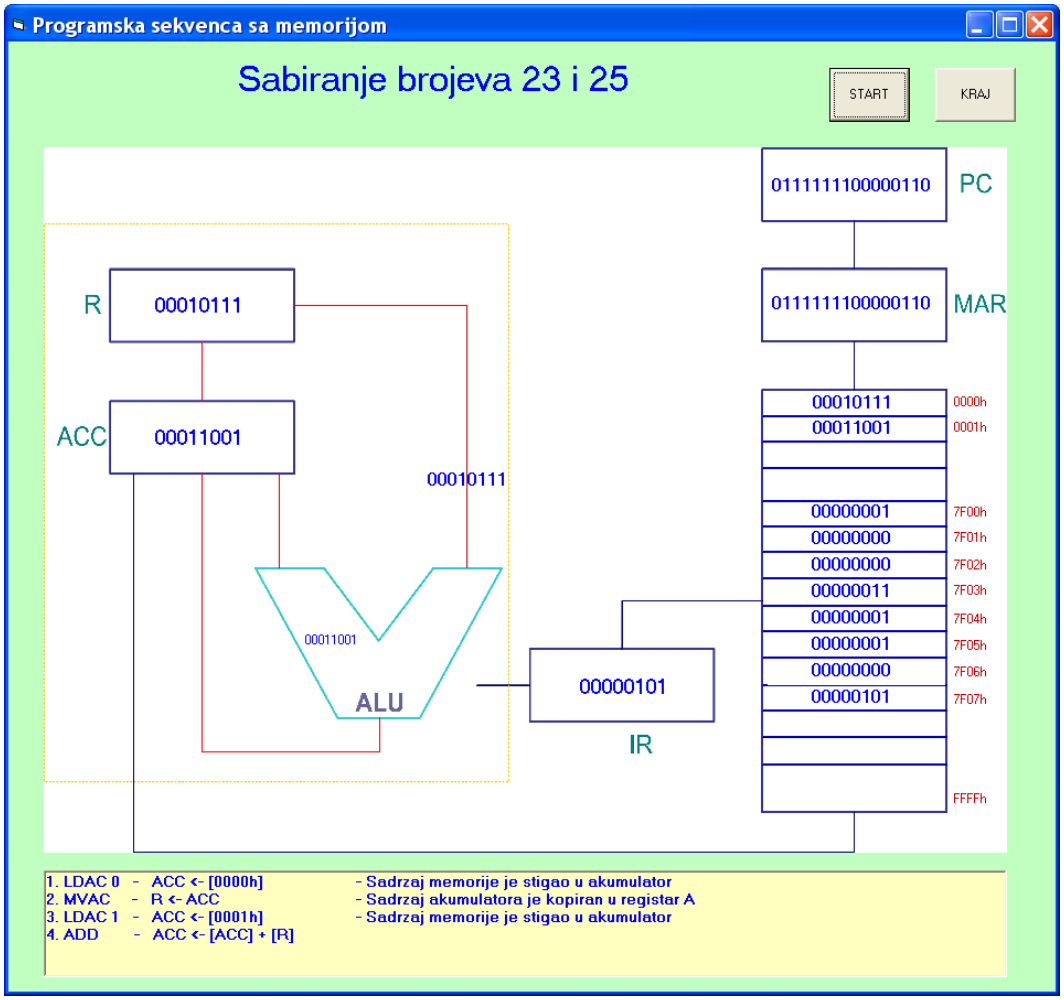
Potvrđivanjem **<SIM 4>** na slici 6.1 dobija se prozor **PROGRAMSKA SEKVENCA SA MEMORIJOM!**. Prozor ove demonstracije sadrži samo dva tastera, **<START>** koji startuje simulaciju i **<KRAJ>** koji omogućuje povratak na prozor projekta **SIMRA**. Simulator sadrži (slika 6.22):

- Dva 8-bitna registra: akumulator ACC i registar R.
- Memoriju, koja je podeljena na dva segmenta. U prvom segmentu, počev od adrese 0000h (h-heksadecmalno) nalaze se vrednosti podataka, osmobitnih brojeva 23 (00010111) i 25 (00011001). U drugom segmentu, počev od adrese 7F00h (0111111100000000), u memoriju su upisane 16-bitne instrukcije. Kako je dužina memorijske reči 8 bita (1 bajt), to se instrukcije moraju dohvatiti u okviru 2 memorijska ciklusa (2 bajta). Koriste se dva tipa instrukcija:
- instrukcije sa operandom (LDAC i STAC). Ove instrukcije koriste direktno memorijsko adresiranje (obraćaju se memoriji) i prvi bajt ovih instrukcija je kôd instrukcije, dok je drugi bajt rezervisan za adresni deo instrukcije.
- instrukcije bez operanada (MVAC i ADD). Ove instrukcije koriste registarsko adresiranje i prvi bajt ovih instrukcija je instrukcija bez dejstva, dok je drugi bajt kôd instrukcije.

  U memoriji je zapisan sledeći program:
- LDAC 0 – iz memorijske lokacije 0, učitava se broj i šalje u akumulator (ACC <- [0000h]), kôd instrukcije je 00000001
- MVAC – sadržaj akumulatora se kopira u registar (R <- ACC), kôd instrukcije je 00000011
- LDAC 1 - iz memorijske lokacije 1, učitava se broj i šalje u akumulator (ACC <- [0001h])
- ADD – sadržaj akumulatora se sabira sa sadržajem registra i smešta se u akumulator (ACC <- [ACC] + [R]), kôd instrukcije je 00000101
- STAC 2 – rezultat sabiranja koji je u akumulatoru, smešta se u memoriju, u memorijsku lokaciju 2 ([0002] <- [ACC]), kôd instrukcije je 00000010
- 16-bitni programski brojač PC, početne adrese 7F00h, adresa prve instrukcije. Uvećava za 1 za svaki novi memorijski ciklus i sadrži adresu instrukcije koja čeka na izvršenje.

- Memorijsko adresni registar MAR, koji čuva adresu memorijske lokacije koja se adresira. Njegova dužina je 16 bita, što znači postoji mogućnost adresiranja 2^{16}-1 KB, tj. kapacitet memorije je 64 KB.
- 8-bitni registar instrukcija IR interpretira instrukciju koja je zadnja doneta, tako što prihvata prvi (drugi) bajt instrukcije i u zavisnosti od sadržaja aktivira odgovarajući izlaz.
- Aritmetičko – logičku jedinicu (ALU), koja u ovom primeru, izvršava operaciju sabiranja ADD.

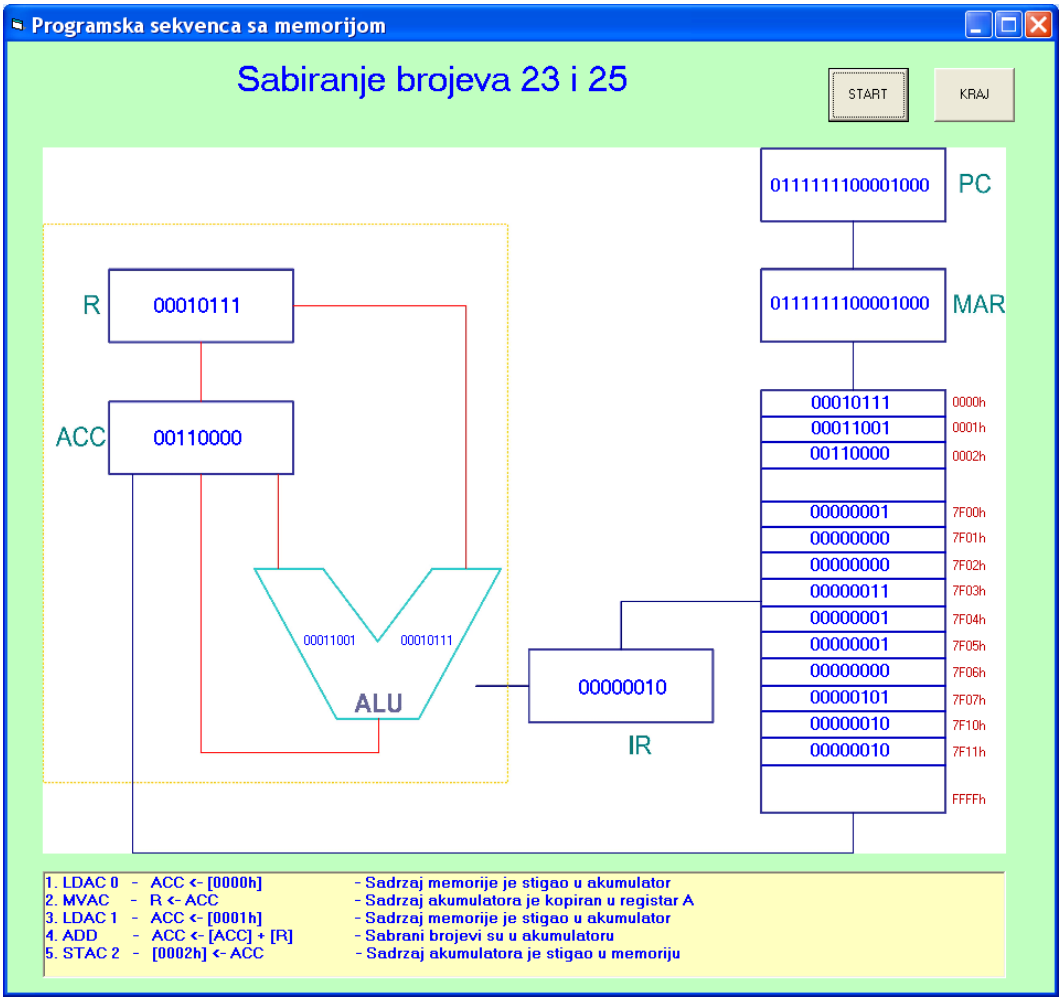
Nakon potvrđivanja tastera **<START>,** kreće izvršavanje simulacije. Početna vrednost programskog brojača je adresa prve instrukcije (**LDAC 0**). MAR prihvata adresu iz PC-a i pronalazi odgovarajuću memorijsku lokaciju. Sadržaj memorijske lokacije se šalje u IR, dekodira se i u zavisnosti od instrukcije izvršava se akcija. Kako je prva instrukcija **LDAC 0** treba memorijski sadržaj iz lokacije 0 proslediti u akumulator. Instrukcija se izvršava u dva memorijska ciklusa. Nakon završetka ove instrukcije prvi broj (23) je u akumulatoru.



Slika 6.22 Demonstrator SIM 4, Izvršavanje instrukcije ADD

Da bi sabrali dva broja koristi se akumulator ACC i registar R. Prvi broj, koji je u akumulatoru, treba sačuvati i sabrati sa drugim brojem. Zato se prvi broj prosleđuje registru R, instrukcijom MVAC. Potom se akumulator puni drugim brojem (25), instrukcijom LDAC 1. Na taj način, prvi broj se nalazi u registru R, dok je drugi broj u akumulatoru. Adresira se sledeća instrukcija (**ADD**), IR šalje signal ALU-u da treba da sabere brojeve koji se nalaze u registrima (ACC i R). Na slici 6.22 dat trenutak izvršavanja instrukcije sabiranja ADD, kada se ALU-u prosleđuju sadržaji akumulatora ACC i registra R, a potom se izvršava operacija sabiranja i rezultat će nakon toga biti smešten u akumulator.

Nakon izvršenja instrukcije sabiranja rezultat treba poslati, instrukcijom **STAC 2**, u memoriju na lokaciju 2. Ovo je poslednja instrukcija programa zapisanog u memoriji. Slika 6.23 daje završni prikaz nakon realizacije programa. I u ovom slučaju, simulacija se može privremeno prekidati (pauzirati) potvrđivanjem na tastaturi tastera Alt.



Slika 6.23 Demonstrator SIM 4, Program je izvršen

6.4.2. Rezime

Nivo simulatora: **Napredni (srednja škola/fakultet)**

Prednosti	Nedostaci	Poboljšanja
▪ Uvodi se memorija, programski brojač PC, memorijsko adresni registar MAR i registar instrukcija IR. ▪ Podaci koji se obrađuju su u memoriji. ▪ Program (niz instrukcija) koji se izvršava nad podacima je, takođe, zapisan u memoriji. ▪ Rezultati obrade se smeštaju u memoriju. ▪ Vizuelno se demonstrira postupak učitavanja instrukcija iz memorije, dekodiranje i izvršavanje istih. ▪ Mogućnost zaustavljanja/nastavka simulacije, radi diskusija.	▪ Korisnik, i dalje, ne može da unosi podatke i da upisuju instrukcije (program). ▪ Operativna memorija je malog kapaciteta. ▪ Ograničen broj instrukcija (samo 4). ▪ Kôd instrukcija (dužina operacionog dela instrukcije) je 8 bita, i to je nepotrebno trošenje prostora, jer sa 8 bita može se kodirati 2^8 (256) instrukcija. ▪ Koriste se samo dva registra. ▪	▪ Smanjiti kôd instrukcije na 4 bita. Time se može kodirati 16 instrukcija. ▪ Omogućiti upisivanje instrukcija i podataka. ▪ Mogućnost učitavanja programa iz datoteke. ▪ Povećati kapacitet memorije. ▪ Povećati dužinu memorijske reči na 2 bajta, kako bi se dohvatanje instrukcije obavilo u jednom memorijskom ciklusu. ▪ Povećati broj registara. ▪ Uvesti indikatore i status registar RS. ▪ Uvesti magistrale.

Broj linija kôda: **650**

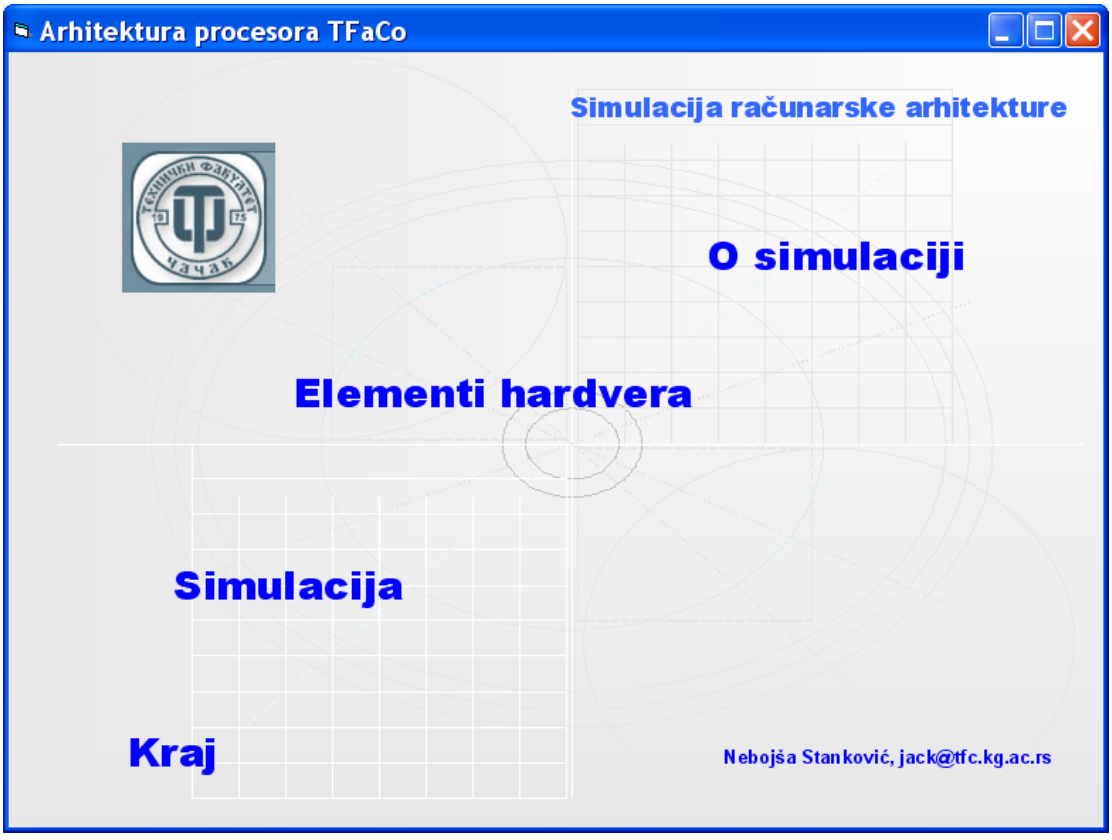
6.5. SIMULACIJA RADA PROCESORA TFACO

Simulator TFaCo je namenjen naprednijim studentima, i predstavlja ponuđeno rešenje jedne arhitekturu procesora, koji je autor ovog rada projektovao. Arhitektura ovog simulatora data je u **poglavlju 5.3**. Sve simulacije nude grafički interfejs za korisnika i daju niz komentara i vizuelnih efekata.

6.5.1. Realizacija TFaCo

Potvrđivanjem tasterta **<TFACO>** na slici 6.1 dobija se prozor **ARHITEKTURA PROCESORA TFACO!** (slika 6.24). U ovom prozoru moguće je:

- dobiti osnovne informacije o simulaciji, potvrđivanjem **„O simulaciji“.** U poglavlju **5.3** data je specifikacija arhitekture ovog procesora.
- potvrđivanjem **„Elementi hardvera“**, dobiti prozor **POTPUNA SHEMA MIKRORACUNARSKOG SISTEMA - OBJASNJENJE!** (slika 6.25) gde su objašnjeni osnovni delovi procesora TFaCo kao i elementi prozora koji omogućavaju vođenje simulacije, tako što se klikne na neki deo prozora i u predviđenom prostoru za objašnjenje će se pojaviti opis tog dela.
- pokrenuti simulaciju, potvrđivanjem „**Simulacija**“.
- potvrđivanjem „**Kraj**“, vratiti se u osnovni prozor projekta **SIMRA**.

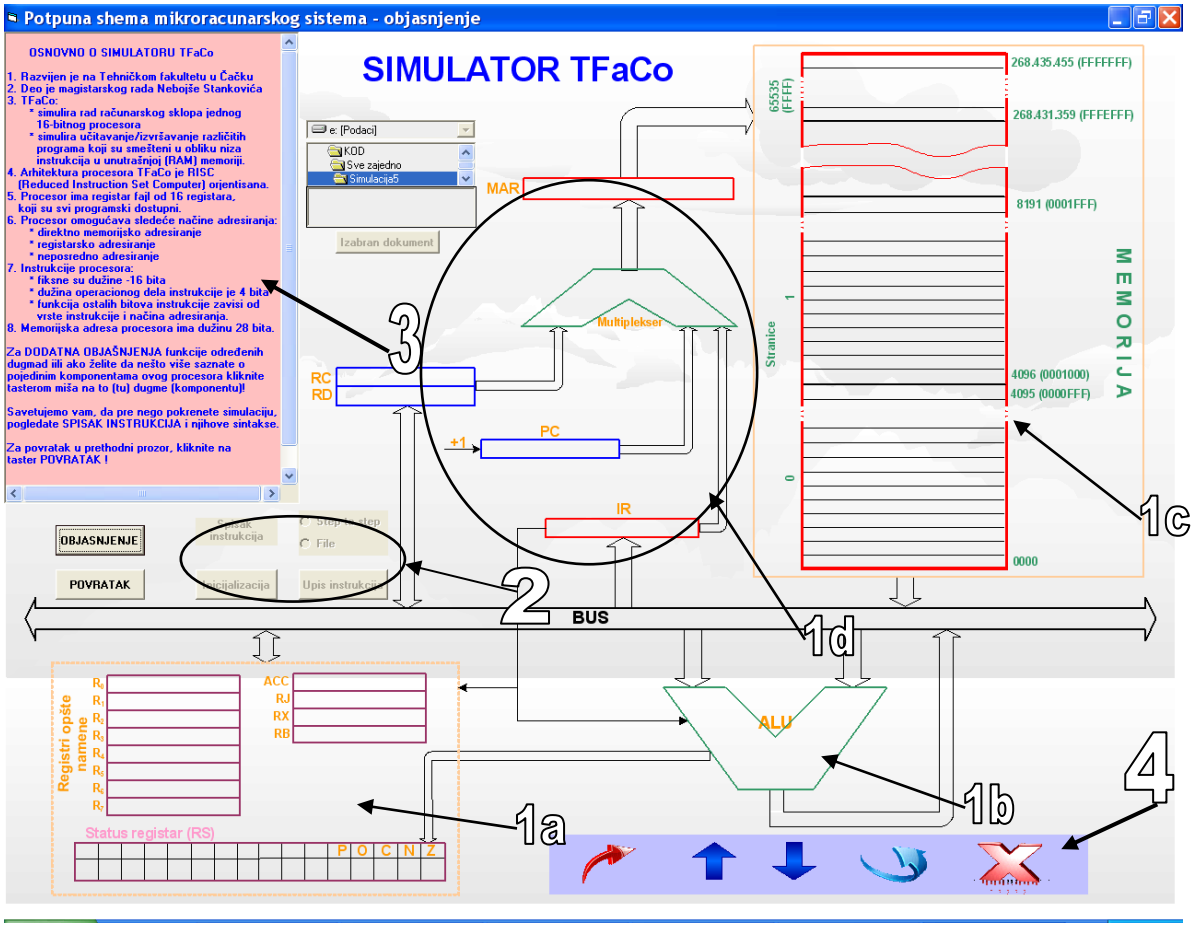


Slika 6.24 Simulator TFaCo, Početni prozor

Objašnjenje elemenata arhitekture procesora TFaCo

Nakon ulaska u prozor **POTPUNA SHEMA MIKRORACUNARSKOG SISTEMA - OBJASNJENJE!** aktivna su samo dva tastera **<POVRATAK>** i **<OBJASNJENJE>.** Potvrđivanjem ostalih elemenata prozora dobija se objašnjenje tog elementa.

Prvi taster omogućava povratak u prethodni prozor aplikacije (slika 6.24), dok potvrđivanjem drugog tastera dobja se prikaz dat na slici 6.25.



Slika 6.25 Simulator TFaCo, Prozor objašnjenja kako se koristi simulator

Na slici 6.25, pored napomenutih tastera, vidljivi su sledeći elementi:

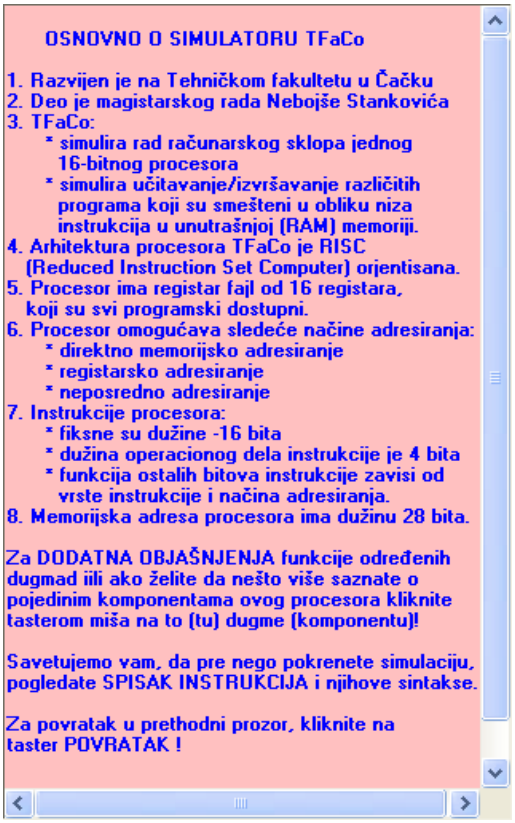
- šema mikroračunarskog sistema procesora TFaCo (centralni deo prozora). Kao što je već rečeno, detaljno objašnjenje elemenata arhitekture ovog procesora dato je u poglavlju 5.3. Četiri osnovne celine arhitekture su obeležene sa:
- 1a – skup registara
- 1b – aritmetičko - logička jedinica (ALU)
- 1c – memorija

- 1d – deo za adresiranje memorije i interpretiranje instrukcija (registri RD i RC, programski brojač PC, multiplekser, memorijsko adresni registar MAR i registar instrukcija IR)
- tasteri za incijalizaciju sistema, unos instrukcija i upis u memoriju (2 – pored tastera za objašenjenje i povratak)
- prozor u kome se daju objašnjenja onih delova na koje se klikne levim tasterom miša (3 – gornji levi ugao prozora).
- alatke za upravljanje simulacijom - pokretanje simulacije, promene brzine simulacije, brisanje i povratak u prethodni prozor (4 – donji desni ugao prozora)

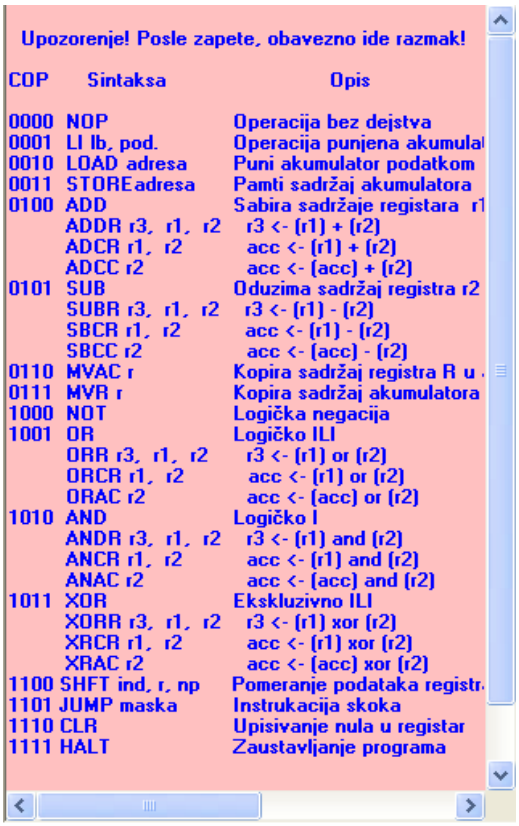
Slike 6.26 i 6.27 predstavljaju uvećani deo prozora (označenog brojem 3) u kome se date osnovne informacije o simulatoru TFaCo (slika 6.26), odnosno su date sintakse instrukcija TFaCo i opis njihovog izvršavanja (slika 6.27).

Pre nego se pokrene simulacija, savetuje se da se pogleda spisak instrukcija i njihove sintakse (slika 6.26).

Ako se instrukcije unose korak-po-korak (step-to-step) treba voditi računa o pravilnom unosu (npr. u objašnjenju na slici 6.27 je i dato upozorenje da prilikom unosa instrukcija posle zapete obavezno mora da ide razmak).



Slika 6.26 Simulator TFaCo, Objašnjenje - osnovno o simulatoru



Slika 6.27 Simulator TFaCo, Objašnjenje – sintakse i opis instrukcija

Simulacija arhitekture procesora TFaCo

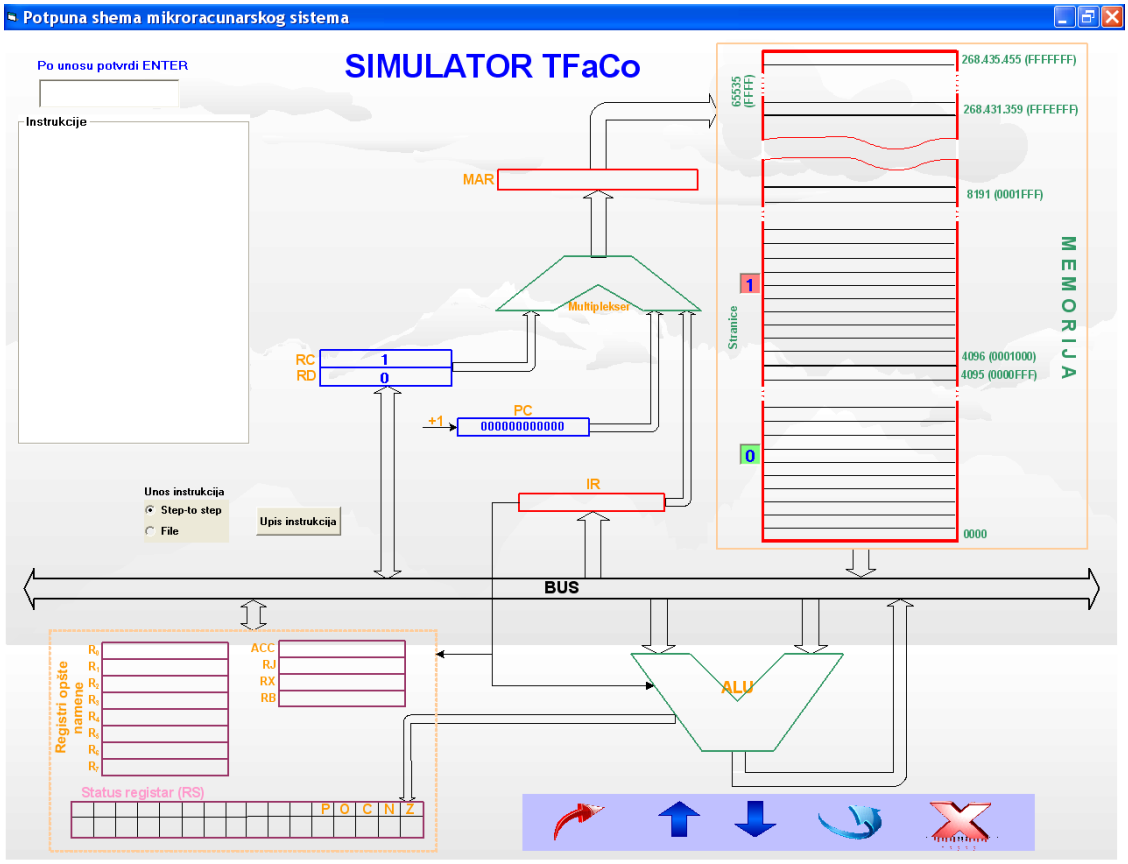
Potvrđivanjem „**Simulacija**“ u prozoru **ARHITEKTURA PROCESORA TFACO!** (slika 6.24) dobija se prozor **POTPUNA SHEMA MIKRORACUNARSKOG SISTEMA!** U ovom prozoru su aktivna jedino dva tastera, taster za inicijalizaciju sistema **<INICIJALIZACIJA>** i alatka **Kraj** koja omogućuje povratak u početni prozor simulatora. Da bi se pokrenula simulacija treba izvršiti inicijalizaciju sistema, tj. postaviti početne vrednosti za:

- startnu adresu registra stranice podataka RD,
- startnu adresu registra stranice programa RC,
- programski brojač PC,

Radi lakšeg praćenja simulacije početne vrednosti za PC i RD su nula, dok je početna vrednost registra stranice programa jednaka jedan.

Nakon inizijalizacije sistema, dobija se slika 6.28. Da bi simulator mogao da radi potrebno je uneti instrukcije (program) koje procesor treba da izvrši i upisati ih u memoriju. Ponuđena su dva načina unosa:

- Unos instrukcija po instrukcija (**<STEP TO STEP>)** ili
- Učitavanje programa iz tekstualne datoteke (**<FILE>)**



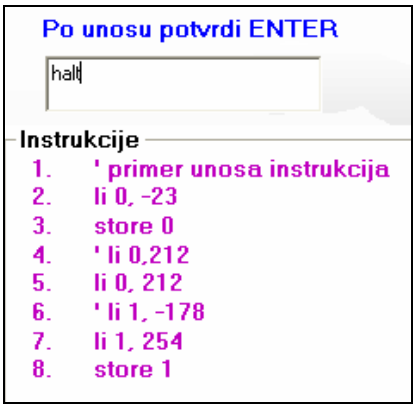
Slika 6.28 Simulator TFaCo, Simulacija – unošenje instrukcija (programa)

Upis instrukcija korak po korak

Unošenje instrukcija korak korak obavlja se uz pomoć polja „**Po unosu potvrdi ENTER**“. Nakon unosa potvrđuje se taster **ENTER** i, ako je programska linija korektno unešena, upisuje se u prozor **„Instrukcije“**, u suprotnom briše se iz polja unosa.

Za unošenje instrukcija „korak po korak“ postupak će biti objašnjen na unosu dva negativna broja (-23 i -300) u akumulator i njihov upis u memoriju. Slika 6.29 daje prikaz pravilno unešenih 8 programskih linija (9-ta čeka napotvrdu), gde linije koje počinju apostrofom su komentari, koje će procesor kasnije ignorisati :

- Linija 1 – Ovo je komentar, koji će biti ignosrisan od strane procesora.
- Linija 2 - Kako se broj -23 može zapisati pomoću 8 bita, za njegov zapis dovoljno je koristi donji (niži) bajt akumulatora i to se postiže instrukcijom **LI 0, -23**. Broj se zapisuje u drugom komplementu.
- Linija 3 - Broj iz akumulatora instrukcijom **STORE 0** se upisuje u memoriju na lokaciju „nula”.



Slika 6.29 Simulator TFaCo, Unošenje instrukcija step-to-step

Da bi broj -300 zapisali u memoriju, potrebno je koristiti 16 bita. Kako je dužina adresnog dela instrukcija procesora TFaCo 12 bita (4 su rezervisana za OPKOD), to se ovaj broj mora zapisati pomoću dve instrukcije LI, prva koja će napuniti donji bajt akumulatora, a druga za gornji deo akumulatora. Broj -300 prikazan u drugom komplementu je 1111111011010100. U gornji bajt akumulatora upisuje se vrednost 11111110 (decimalno 254), dok se u donji bajt upisuje vrednost 11010100 (212). To je postignuto pomoću programskih linija 5 i 7.

- Linija 4 – Primer loše sintakse instrukcije, jer posle zapete nema razmaka **LI 0,212.**
- Linija 5 – Upis broja 212 u donji bajt akumulatora pomoću instrukcije **LI 0, 212.**
- Linija 6 – Greška LI 1, -178; u gornji bajt akumulatora ne mogu se upisivati negativni brojevi.
- Linija 7 – Upis broja 254 u gornji bajt akumulatora pomoću instrukcije **LI 1, 254.**

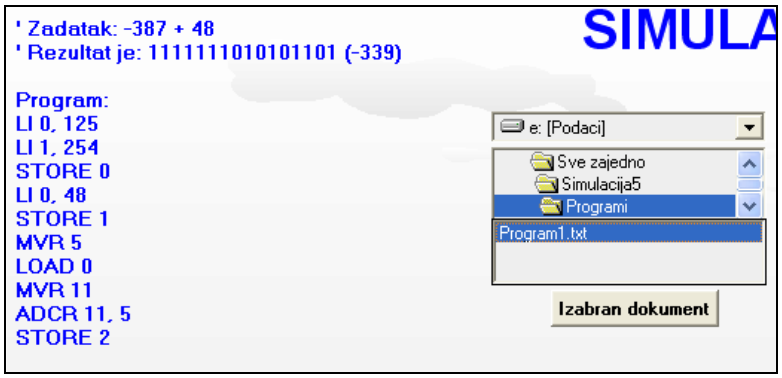
- Linija 8 – Upisivanje vrdnosti iz akumulatora u memoriju – **STORE 1**
- Linija 9 – Čeka na potvrđivanje, a to je instrukcija **HALT**, koja označava kraj programa.

Nakon unosa svih instrukcija i potvrđivanja tastera **<UPIS INSTRUKCIJA>** program se upisuje u memoriju. Znači, u memoriju će biti upisane samo programske linije 2, 3, 5, 7, 8 i 9. Izgled memorije nakon unešenog programa dat je na slici 6.30. Ukupno ima 6 upisanih instrukcija. Prva instrukcija je na dnu stranice jedan, a naredne se ređaju jedna iznad druge. Prva četiri bita (gledano sleva u desno) označavaju vrstu instrukcije, a narednih 12 su adresni deo instrukcija.

Slika 6.30 Simulator TFaCo, Izgled memorije nakon upisa instrukcija step-to-step

Učitavanje programa iz datoteka

Ako se korisnik odluči za učitavanje programa iz nekog dokumenta, treba da potvrdi taster (**<FILE>).** Na slici 6.31 dat je prikaz učitavanja dokumenta **Programi1.txt**. Dokument se nalazi na CD-u. Sadržaj dokumenta je desno.



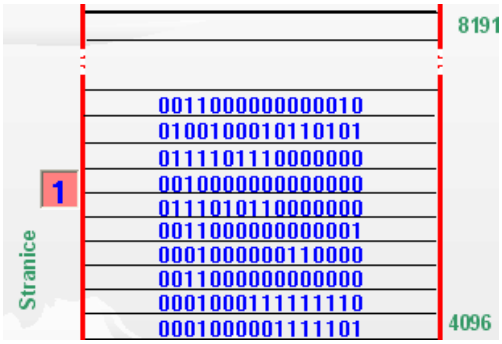
Slika 6.31 Simulator TFaCo, Unošenje programa iz dokumenta

Na slici je dat programa za sabiranje dva cela broja **-387 i 48**, koji sadrži sledeće instrukcije:

- instrukcije **LI 0, 125** i **LI 1, 254** omogućavaju unos broja -387 u akumulator
- instrukcija **STORE 0** upisuje broj iz akumulatora (-387) u memoriju
- četvrta instrukcija **LI 0, 48** upisuje broj 48 u akumulator

- potom se taj broj pomoću pete instrukcije **STORE 1** upisuje u memoriju
- pomoću instrukcije **MVR 5** se sadržaj akumulatora kopira u registar br. 5.
- instrukcijom **LOAD 0** se sadržaj memorije sa adresom 0 upisuje u akumulator
- instrukcija **MVR 11** sadržaj akumulatora se kopira u registar br. 11
- sadržaje registara 11 i 5 treba sabratii rezultat smestiti u akumulator. To se postiže instukcijom **ADCR 11, 5**
- rezultat sabiranja se iz akumulatora šalje u memoriju instrukcijom **STORE 2**.

Da bi se program učitao u memoriju, potrebno je potvrditi taster **<IZABRAN DOKUMENT>** a potom taster **<UPIS INSTRUKCIJA>**. Izgled popunjene memorije instrukcijama (ima ih ukupno 10) dat je na slici 6.32.



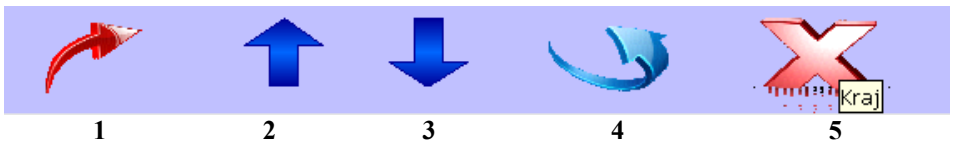
Slika 6.32 Simulator TFaCo,Izgled memorije nakon upisa programa

Izvršavanje programa (instrukcija)

Nakon upisa instrukcija u memoriju, treba te instrukcije pribaviti i interpretirati (faza dohvatanja) [43]. To je zadatak procesora. Na dnu prozora nalaze se alatke za upravljanje simulacijom (slika 6.33). Funkcija ovih alatki je:

- Alatka br. 1 – **Startuj simulaciju**, simulacija faze dohvatanja.
- Alatka br. 2 – **Ubrzaj**, potvrđivanjem ove alatke ubrzava se simulacija.
- Alatka br. 3 – **Uspori**, potvrđivanjem ove alatke usporava se simulacija.
- Alatka br. 4 – **Obrisi**, da bi se započeo unos novih instrukcija (programa), treba očistiti memoriju i registre.
- Alatka br. 5 – **Kraj**, završetak i povratak na početni prozor.

U toku izvođenja simulacije, simulacija se može zaustaviti (pauzirati) tasterom Alt na tastaturi. Nastavak simulacije se takođe izvodi potvrđivanjem ovog tastera.



Slika 6.33 Simulator TFaCo, Alatke za upravljanje simulacijom

Nakon potvrđivanja **Startuj simulaciju** simulacija počinje i njeno izvođenje se može opisati u nekoliko koraka:

- **Izračunavanje adrese instrukcije**. Da bi se dobila adresa u kojoj je smeštena instrukcija, potrebno je u multiplekseru „sabrati“ vrednost iz registra instrukcije RC i vrednost programskog brojača. U poglavlju 5.33 objašnjeno je zbog čega su uvedeni registri stranica (registri RC i RD). Kako svaki registar stranica ima 4096 memorijskih ćelija (reči), to se izvršna adresa formira po formuli rc*4096+pc, gde su rc i pc vrednosti adresa u RC, odnosno PC. U slučaju instrukcija STORE i LOAD, izvršna adresa se formira na osnovu vrednosti iz registra podataka RD i adrese operanda iz registra instrukcije IR.
- **Čitanje instrukcije iz memorije**. Iz multipleksera izvršna adresa se prosleđuje Memorijsko adresnom registru i instrukcija se čita iz memorije sa te adrese.
- **Dekodiranje operacionog kôda instrukcije**. Instrukcija se prosleđuje registru instrukcije (IR) i vrši se njeno dekodiranje. Instrukcija se dekodira na osnovu četiri bita najveće težine. Na osnovu analize operacionog kôda određuje se tip operacije i broj argumenata instrukcije Na slici 6.34 dat je trenutak kada je IR dekodirao instrukciju **LI 0, 125**, čiji je kôd 0001.
- **Izvršava se operacija koja je zahtevana operacionim kôdom**. U ovom slučaju u donji bajt akumulatora treba upisati broj 125 u binarnom zapisu. U ovom primeru procesor treba da izvrši 5 različitih tipova instrukcija: **LI, STORE, LOAD, MVR i ADCR**.
- **Određivanje vrednost indikatora u status registru**. U status registar RS se upisuju vrednosti za indikatore P, O, C, N i Z. Uticaj instrukcija na indikatore u registru RS dato je u poglavlju 5.37

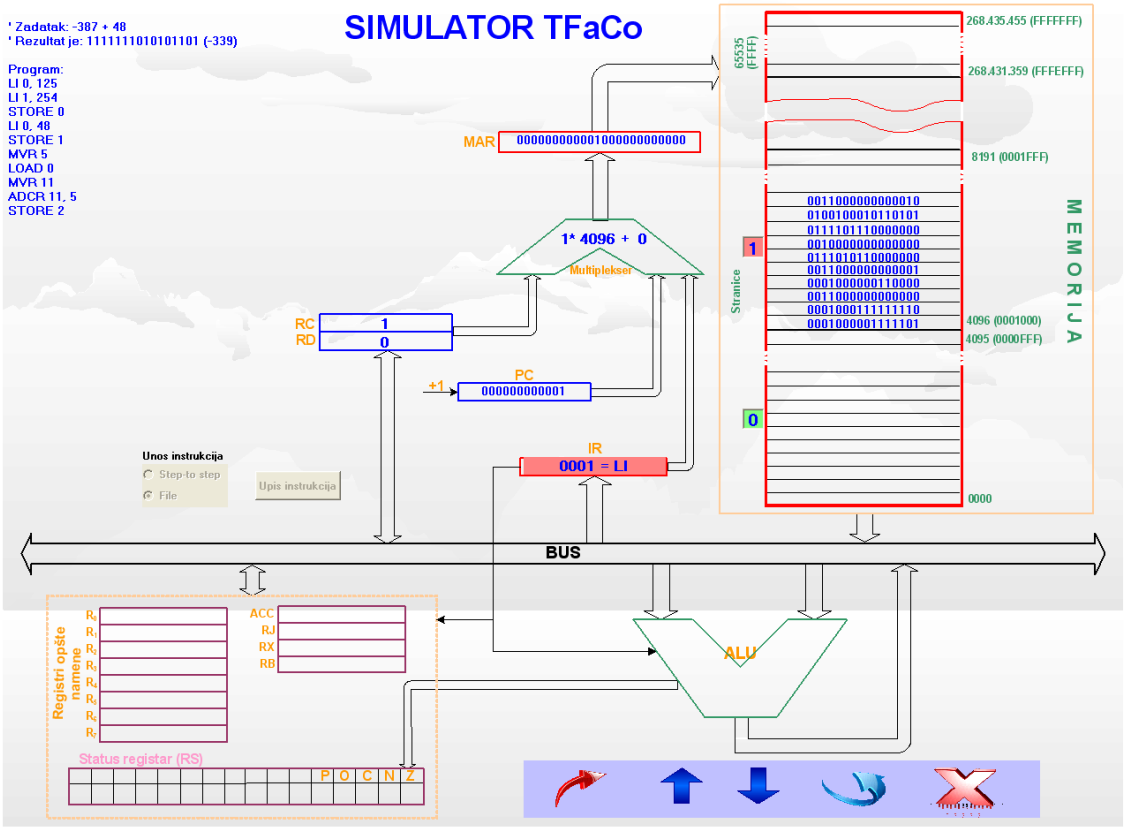
Po izvršenoj operaciji i upisu indikatora u registar RS, postupak se ponavlja za sledeću instrukciju (čitanje, dekodiranje, izvršavanje, upis). Simulacija se odvija po redosledu upisanih instrukcija u memoriji.

Slika 6.35 prikazuje trenutak izvršenja instrukcije **STORE 0**, kada se vrednost iz akumulatora **1111111001111101** (-387) upisuje u memoriju na adresu **0000** (na slici je osenčena) u registru podataka RD = 1.

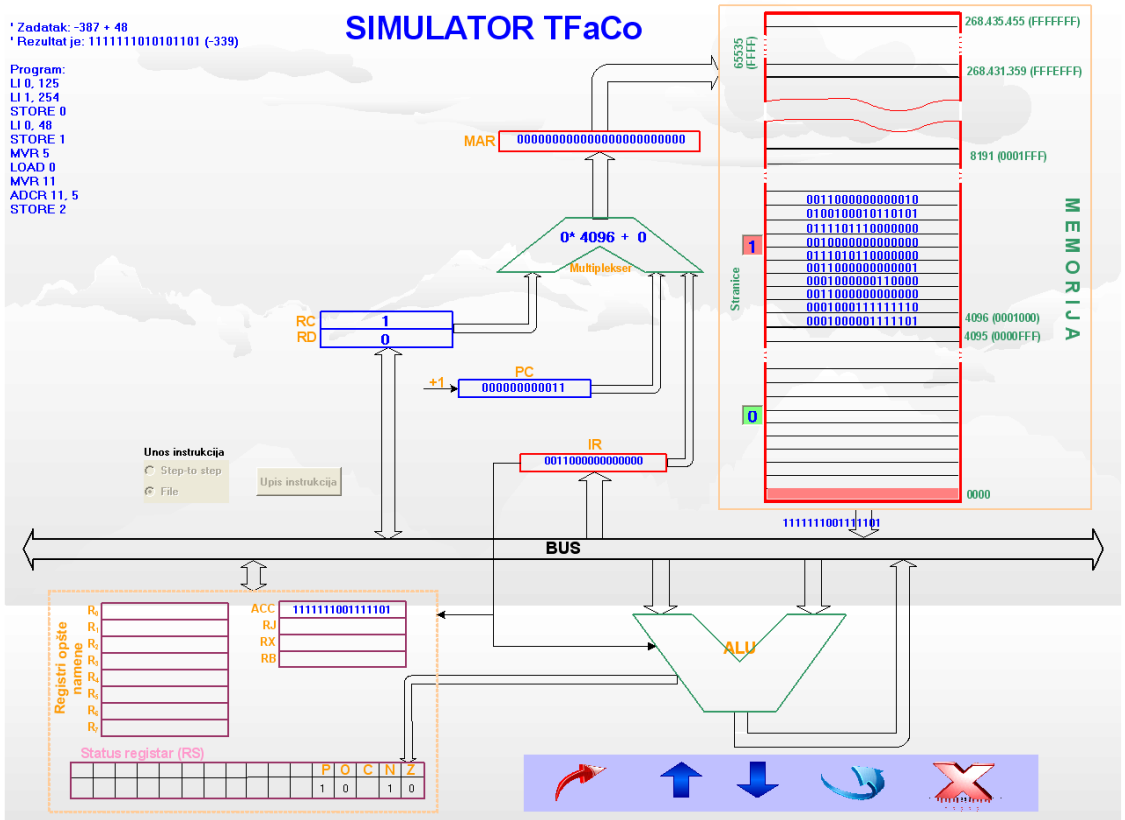
Trenutak izvršenja instrukcije **ADCR 11, 5**, kada se rezultat sabiranja sadržaja registra 11 i registra 5 prosleđuje u akumulator ACC dat je na slici 6.36.

Izvršenjem instrukcije **STORE 2** rezultat iz akumulatora upisuje se u memoriju na adresu **0010** i to je poslednja instrukcija u programu. Izgled prozora nakon izvršenja poslednje instrukcije dat je na slici 6.37.

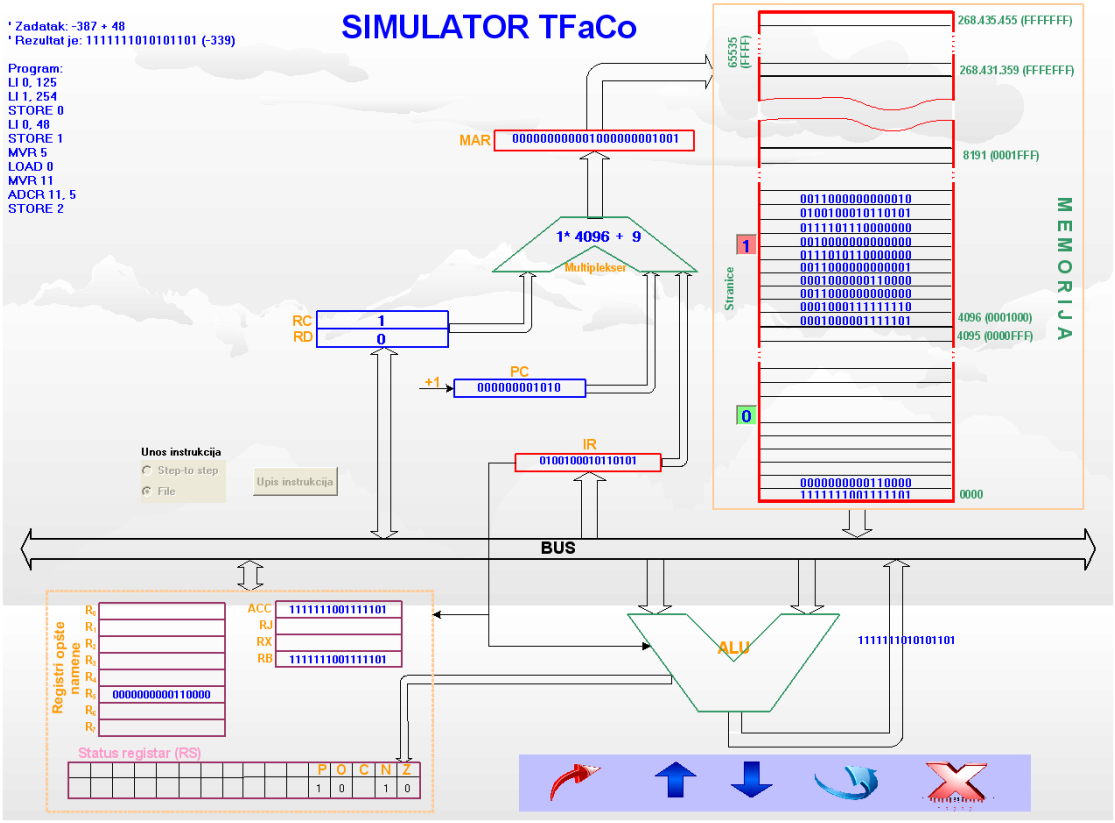
Ako se želi upis novih instrukcija, prethodono treba očistiti memoriju i registre alatkom 4 **Obrisi**, a izlazak iz simulatora se postiže potvrđivanjem alatke 5 **Kraj**.



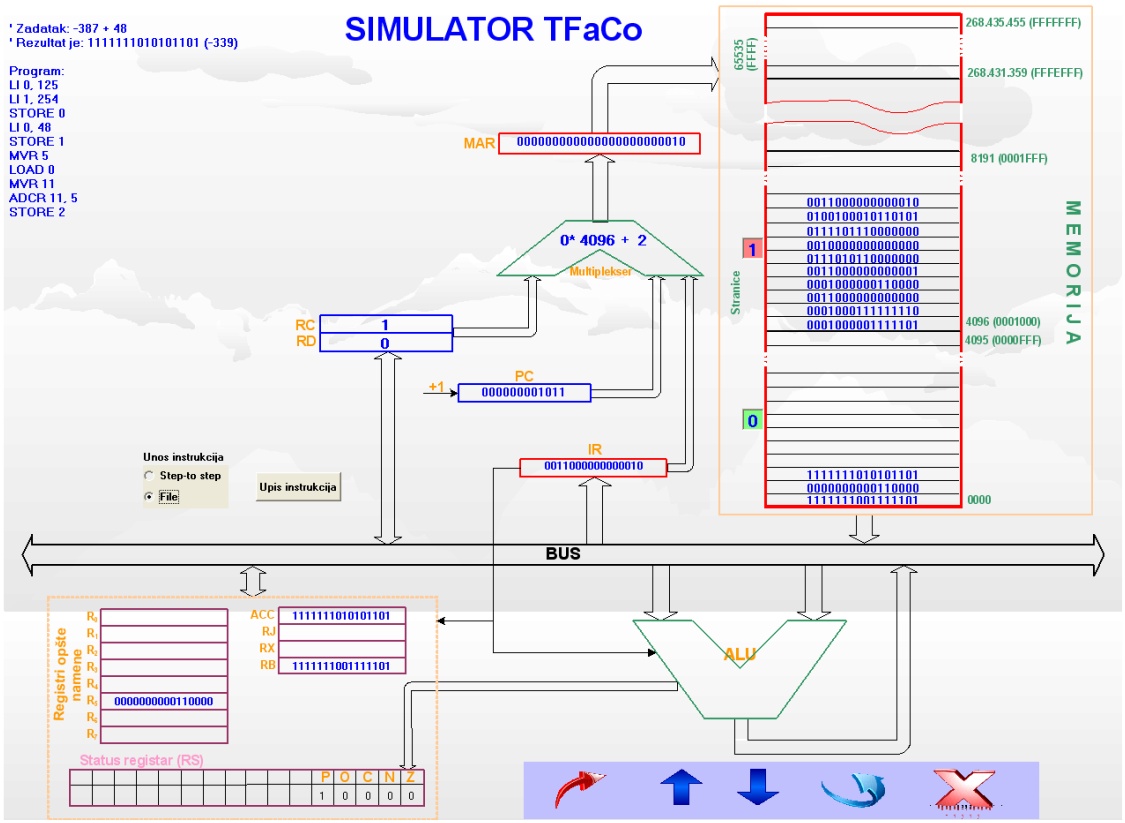
Slika 6.34 Simulator TFaCo, Prva instrukcija je dekodirana (0001=LI)



Slika 6.35 Simulator TFaCo, Trenutak upisa vrednosti iz akumulatora u memoriju (STORE 0)



Slika 6.36 Simulator TFaCo,Prosleđivanje rezultata iz ALU u akumulator ACC (ADCR 11, 5)



Slika 6.37 Simulator TFaCo, Izgled prozora nakon izvršenja programa

6.5.2. Rezime

Nivo simulatora: **Napredni (fakultet)**

Broj linija kôda: **2600**

Prednosti:

Hardverske prednosti arhitekture procesora TFaCo, kreiranog od strane autora ovog rada, izložene su u poglavlju 5.3. Ovde se navode prednosti aplikacije koja simulira taj procesor. Simulator TFaCo otklanja većinu nedostataka prethodnih simulatora i pruža velike mogućnosti njegovim korisnicima:

- Mogućnost izbora objašnjenja osnovnih delova simulatora.
- Povećan je broj instrukcija koje se simuliraju.
- Mogućnost simuliranja samo jedne instrukcije.
- Asemblerski program se može unositi instrukcija po instrukcija ili se može učitati iz datoteka.
- Podaci koji se obrađuju korisnik sam zadaje (unosi).
- Mogućnost rada sa 16-bitnim podacima, upoznavanje uloge akumulatora.
- Vizuelizacijom faze izračunavanja adrese instrukcija korisnik se upoznaje sa ulogama registara RD, RC, PC i MAR.
- Zahvaljujući vizuelizaciji rada memorije, korisnik upoznaje principe direktno memorijskog adresiranja, registarskog adresiranja i neposrednog adresiranja.
- Korisnik upoznaje postupak dekodiranja instrukcije i ulogu registra IR.
- U toku vizuelizacije prenosa podataka od memorije ka registrima i obratno, korisnik se upoznaje sa ulogom magistrala.
- U toku izvršavanja različitih instrukcija simulira se dvosmerni transfer podataka između registara i ALU-a i korisniku se pojašnjava njihova uloga.
- Korisnik saznaje kako instrukcije utiču na indikatore u registru RS.
- Mogućnost privremenog zaustavljanja simulacije.
- Mogućnost ubrzavanja/usporavanja vizuelizacije.

Moguća poboljšanja

Napraviti robusniji simulator koji će obuhvatiti još veći broj instrukcija, povećati broj izvršnih linija programa koje korisnik može da unese (zbog preglednosti vizuelizacije rada memorije, broj linija je ograničen na deset).

7

7. ZAKLJUČAK

Brz razvoj IT-a u zadnjih pola veka doveo je do sveukupnih promena u životu čoveka. Ove promene su pre svega diktirane razvojem računarskih nauka u čijem centru se nalazi mašina (računar) kao subjekt koji je eksplicitno i implicitno uticao na korenitost promena. Sa razvojem računarskih sistema javila se i potreba da se kompleksne operacije unutar samog računara približe što većem broju korisnika. Zato je zadužena oblast nauke koja se naziva arhitektura računarskog sistema (ARS). Ona zbog složenosti samih procesa i funkcija unutar računara pribegava specifičnim metodama modeliranja, vizuelizacije i simulacije, kako bi obučavani stekli što potpunija znanja iz ove oblasti.

Korišćenje simulatora ima veliki značaj u obučavanju učenika/studenata jer omogućava da se složeni procesi unutar računarskog sklopa prikažu učenicima/studentima slikovito, na vrlo uverljiv način. U ovom radu istaknut je značaj simulatora računarske arhitekture i sa aspekta ekonomske isplativosti, a takođe i sa aspekta mogućnosti simultanog korišćenja od strane većeg broja učenika/studenata. Uz to, dati su i problemi vezani za uvođenje računara i njihovih simulatora u nastavni proces. Vrlo je bitno da simulatori budu izabrani prema predznanju učenika/studenata. Simulatori realizovani kroz ovaj rad pokrivaju različite nivoe predznanja korisnika.

Sa stanovištva svetskih iskustava dati su pregledi nekih simulatora ARS-a i to pre svega ECS, HASE, ESCAPE, DLXview, LMC. Svaki od ovih simulatora sa svojim specifičnostima objašnjen je kroz slikovite prikaze arhitekture i osnovnih funkcionalnih komponenata. Oni su predstavljali osnovu za razmatranje dizajna TFaCo simulatora koji predstavlja najpotpuniji deo softverske realizacije ovog rada. Suštinske pretpostavke za realizaciju ovog simulatora objašnjeni su kroz prikaz: osnovnih aritmetičkih i logičkih

operacija, korišćenja mašinskih instrukcija (unošenje, donošenje, dekodiranje), rada sa registrima, pristupanja memoriji, korišćenja PC-a, korišćenja ALU-a i slično.

Korišćenjem programskog jezika Microsoft Visual Basic 6.0 kreiran je projekat SIMRA (SIMulacija Računarske Arhitekture), koji se sastoji od 5 aplikacija: 3 simulatora i 2 demonstratora. Njihov zadatak je prikaz određenih f-ja računarske arhitekture kao i kompletna realizacija jednog virtuelnog procesora (TFaCo).

Prva aplikacija ovog projekta je simulator SIM1. On omogućava korisniku da se upozna sa binarnim brojnim sistemom i asemblerskim zapisom osnovnih aritmetičkih operacija. Predviđen je za korisnike bez početnog znanja.

U drugoj aplikaciji realizovan je jedan od ključnih simulatora ovog projekta SIM2. Korisnicima se simulira rad osnovnih aritmetičkih i logičkih operacija nad brojevima u drugom komplementu. Data je detaljna vizuelizaciju na nivou bita. Namenjen je korisnicima sa predznanjem.

Aplikacija SIM3 je demonstracija, koja na primeru dva binarna broja korisniku, kroz korake, vizuelno pokazuje gde sve podaci treba da „prođu“, da bi se nad njima izvršila operacija sabiranja. Takođe, namenjena je korisnicima sa predznanjem.

I četvrta aplikacija je demonstracija, koja daje vizuelni prikaz učitavanja programa (instrukcija) iz memorije, njihovo dekodiranje i izvršavanje. Namenjena je naprednim korisnicima.

Krucijalni doprinos ovoga rada predstavlja peta aplikacija, a to je simulacija arhitekture procesora TFaCo. Ovaj simulator omogućava: unos asemblerskog programa i njegovo izvršavanje; vizuelizaciju stanja odgovarajućih registara; vizuelizaciju rada memorije, korišćenjem direktnog, registarskog i neposrednog adresiranja; upoznavanje sa postupkom dekodiranja instrukcije, kao i upoznavanje sa tokom podataka na magistralama između registara, unutar ALU-a i slično. Simulator daje mogućnost izvršavanja simulacije korak po korak (privremenog zaustavljanja i pokretanja), kao i ubrzanje/usporenje određenih aktivnosti. Kompletan proces rada simulatora je praćen sa odgovarajućim help sistemom kojem korisnik može da pristupi radi detaljnijeg upoznavanja elemenata arhitekture procesora i funkcije pomoćnih alata. Ovaj simulator je, prvenstveno, namenjen u edukativne svrhe za korisnike sa određenim predznanjem koji u našim visokoškolskim ustanovama pohađaju predmet tipa "arhitektura računara".

Doprinos ovog magistarskog rada je i to što su sve aplikacije realizovane na srpskom jeziku, što omogućuje našim učenicima/studentima da lakše izučavaju oblast ARS-a.

Dalje smernice unapređivanja ovog rada išle bi u dva smera:

1. Kreiranje kompletnog skupa simulatora i demonstratora koji bi pokrivali prostor od osnovnog do visokoškolskog obrazovanja.
2. Povećanje robusnosti TFaCo procesora, uvođenjem dodatnih instrukcija, poboljšanjem vizuelizacije memorije, rad sa registrima većeg kapaciteta od 16 bita, spuštanje na nivo mikroinstrukcija, razmatranje višeprocesorske arhitekture itd.

Simulatori bi bili realizovani i u Web okruženju.

LITERATURA

Knjige, časopisi, članci

[1] Bezdanov, S. (2000): Sistem kvaliteta i standardizacija u obrazovanju prema zahtevima serije standarda JUS-ISO 9000, *Inovacije u nastavi*, časopis za savremenu nastavu, Beograd.

[2] Cohen, K. J., & Cyert, R. M. (1965): Simulation of Organizational Behavior. J. G. March (Editor), *Handbook of Organizations* (pp. 305-334). Chicago: Rand McNally & Company.

[3] Danilović, M. (2003): Uticaj i mogućnost informaciono-komunikacionih medija i tehnologija u realizaciji savremenih oblika učenja i nastave, *Zbornik radova, "Komunikacija i mediji u savremenoj nastavi"*, Učiteljski fakultet, Jagodina.

[4] Đorđević, D., Trifković, S. (1995): *Visual Basic, programiranje kroz primere*, M&G Inženjering, Beograd.

[5] Đorđević, J. (1998): *Priručnik iz arhitekture i organizacije računara*, ETF, Beograd.

[6] Đorđević, J. (2003): *Arhitektura računara, Edukacioni računarski sistem, Arhitektura i organizacija računarskog sistema*, ETF, Beograd.

[7] Đorđević, J. (2003): Nastava kao proces poučavanja, učenja i komunikacije, *Zbornik radova, "Komunikacija i mediji u savremenoj nastavi"*, Učiteljski fakultet, Jagodina.

[8] Đorđević, J., Grbanović, N., Nikolić, B. (2002): *Arhitektura računara, Edukacioni računarski sistem, Priručnik za simulaciju sa zadacima*, ETF, Beograd.

[9] Maxvell, T., Scott, B. (1992): *Visual Basic Super Bible*, Corte Madera, California.

[10] Milovanović, J., Perić, D. (1995): *Visual Basic 3.0 profesional – razvoj aplikacija i programiranje*, Viša elektrotehnička škola, Beograd.

[11] Parezanović, N. (1983): *Računari i programiranje*, Naučna knjiga, Beograd.

[12] Pentland, B. (1999). Building process theory with narrative: From description to explanation. *Academy of Management Review*, 24(4): 711-724.

[13] Peter Norton (2000): *Nadgradnja i popravka PC-a*, Kompjuter Biblioteka, Čačak.

[14] Peter Norton (2000): *Unutrašnjost PC-a*, Kompjuter Biblioteka, Čačak.

[15] Radenković, B., Stanojević M. i Marković A. (1999): *Računarska simulacija*, udžbenik, Fakultet organizacionih nauka i Saobraćajni fakultet, Beograd.

[16] Ron White (2004): *Kako rade računari*, CET Computer Equipment and Trade, Beograd.

[17] Rothenberg J. (1989): *Tutorial: artificial intelligence and simulation*, Winter Simulation Conference (pp. 33-39).

[18] Smith, E., Whisler, V., Maruis, H. (1999): *Visual Basic 6 biblija*, Mikro knjiga, Beograd.

[19] Spasić I. (2006): *Simulatori arhitektura računara*, Diplomski rad, Tehnički fakultet, Čačak, 2006.

[20] Stephen Bigelow (2000): *Rešavanje problema, popravka i nadogradnja PC-a*, Kompjuter Biblioteka, Čačak.

[21] Stojčev, M. (1990): *Savremeni 16-bitni procesori*, Ei Istraživačko-razvojni Institut, Naučna Knjiga, Beograd.

[22] Stojčev, M. (1997): *RISC, CISC i DSP procesori*, Elektronski Fakultet, Niš.

WEB stranice (navedeno po datumu posećivanja)

[23] Djordjevic, J., Milenkovic, A., Grbanovi, N., Bojovic, M. (1999). An Educational Environment for Teaching a Course in Computer Architecture and Organization, *IEEE TC Computer Architecture Newsletter*, July 1999. **posećeno aprila 2007**., http://tab.computer.org/tcca/NEWS/jul99/jovan98.pdf

[24] Đorđević, J. (2007): Arhitektura računara, Edukacioni računarski sistem**,** Simulator SPECS, **posećeno aprila 2007.**, http://rti.etf.bg.ac.yu/rti/ef2ar/labvezbe/index.html.

[25] Đorđević J., Bošković, N. (2007)*:*Nastava iz arhitekture i organizacije računara problemi i rešenja, Nulta verzija materijala o simulatorima, ETF, Beograd, **posećeno aprila 2007**., http://rti.etf.bg.ac.yu/rti/ir3ar2/projekat/Simulacija.pdf

[26] Yurcik, W., Osborne, Hugh. (2001): A crowd of little man computers: visual computer simulator teaching tools, *Proceedings of the 2001 Winter Simulation Conference*, New York, **posećeno aprila 2007**., www.informs-sim.org/wsc01papers/224.PDF

[27] Yurcik, W., Wolffe, G. S., Holiday, M. A. (2001): A Survey of Simulators Used in Computer Organization/Architecture Courses, *Summer Computer simulation Conference,* Orlando FL, July 2001., **posećeno aprila 2007**., http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/y/Yurcik:William.html

[28] Yang, T. A. (2001). Integration of Computer Simulation and Visualization Research into Undergraduate Degree Programs, in: B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds., *Proceedings of the 2001 Winter Simulation Conference,* 1593-1596., **posećeno maja 2007**., www.informs-sim.org/wsc01papers/217.PDF

[29] Ryan W. Quinn (2001): Tales of the microprocessor: the narrative properties of computer simulation, Olin School of Business Washington University in St. Louis , **posećeno maja 2007.**, http://www.apps.olin.wustl.edu/workingpapers/pdf/2004-08-001.pdf

[30] Osborne, H., Crossley, S., Mencak, J., Yurick, W. (2002): PECTOPAH: Promoting Education in Computer Technology using an Open-ended Pedagogically Adaptable Hierarchy, **posećeno maja 2007.**, www.ncsu.edu/wcae/ISCA2002/submissions/osborne-pectopah.pdf

[31] Ibbett, R., Mallet, F. (2003): Computer architecture simulation applets for use in teaching, *33rd ASEE/IEEE Frontiers in Education Conference*, **posećeno maja 2007.**, http://www.fie-conference.org/fie2003/papers/1545.pdf

[32] *Computer Architecture Simulation & Visualisation,* HASE ProjectInstitute for Computing Systems Architecture, School of Informatics, University of Edinburgh, 1989, **posećeno oktobra 2007**., http://www.icsa.inf.ed.ac.uk/research/groups/hase

[33] DLXViev simulator, Compiler/Architecture Simulation for Learning and Experimenting, **posećeno oktobra 200**7., http://cobweb.ecn.purdue.edu/~teamaaa/dlxview/

[34] Simulation of Little Man Computer, Illinois State Univesity, 2004, posećeno **oktobra 200**7., http://www.acs.ilstu.edu/faculty/javila/lmc/

[35] Campenhout, J., Verplaetse, P., Neefs, H. (1999): ESCAPE: Environment for the Simulation of Computer Architectures for the Purpose of Education, Department of Electronics and Information Systems, University of Ghent, Belgium, **posećeno oktobra 2007**., ww.ncsu.edu/wcae/ISCA1998/verplaetse.pdf

[36] Stojčev M.: Predstavljanje brojeva u računaru, Elektronski Fakultet Niš, **posećeno novembra 2007.**, www.elfak.ni.ac.yu/phptest/new/html/informacije/vesti/resenja/mps/poglavlja/stojcev/digitalna%20elektronika/Glava3.pdf

[37] Filipović, V. (2007): Celi brojevi i celobrojna aritmetika, Matematički fakultet, Beograd, **posećeno novembra 2007.**, www.matf.bg.ac.yu/~vladaf/Courses/ORS/Predavanja/4.celibrojevi.pdf

[38] Аритметичке и логичке операције, Prirodno matematički fakultet, Niš, **posećeno novembra 2007**., www.pmf.ni.ac.yu/pmf/predmeti/1231/Computer%20Systems/CS16.ppt

[39] Popović, M. (2004): Osnovi elektronike, ETF, Beograd, **posećeno decembra 2007**., http://default.co.yu/~trooper/sietf/osnovi_elektronike.pdf

[40] Predstavljanje brojeva, Tehnička škola Užice, **posećeno decembra 2007** http://www.tehnickaue.edu.rs/srp/cas/?conid=600

[41] Čajić, B.: Građa digitalnih sklopova i uređaja, **posećeno marta 2007.**, http://free-zg.t-com.hr/HrvojeCajic/Grada_digitalnoh_sklopova.pdf

[42] Malbaša, V. (2001): Mikroprocesorska elektronika, Fakultet tehničkih nauka, Novi Sad, **posećeno decembra 2007.**, http://www.kel.ftn.ns.ac.yu/predmeti/3/mpe/predavanja/MPE%20Skripta%20-%202002-1.pdf

[43] Filipović, V. (2007): Upravljačka jedinica, Matematički fakultet, Beograd, **posećeno decembra 2007.**, http://poincare.matf.bg.ac.yu/~vladaf/Courses/ORS/Predavanja/12.upravljacka%20jedinica.pdf

[44] Struktura CPU-a, Prirodno matematički fakultet, Niš, **posećeno decembra 2007.**, http://www.pmf.ni.ac.rs/pmf/predmeti/1231/Knjiga%20-%20Vol2/GLAVA2.pdf

[45] Arhitektura CPU, Udruženje studenata Visoke Tehničke škole, Kragujevac, **posećeno marta, 2008.**, www.usvts.net/skripte/Arhitektura%20Racunara.ppt

PRILOG

U ovom delu rada dati su listinzi procedura i fukcija za koje autor smatra da mogu biti od koristi čitaocima ovog rada.

Listing 1 Procedura koja ne dozvoljava da se u tekstualno polje unesu drugi simboli osim 0 ili 1

```
Sub txtOperand_KeyPress(KeyAscii As Integer)
  Dim Brojevi As String
        Brojevi = "01"                    'definisemo sta moze da se unosi
        If KeyAscii = vbKeyBack Then Exit Sub
        If InStr(Brojevi, Chr(KeyAscii)) = 0 Then
              KeyAscii = 0
        End If
End Sub
```

Listing 2 Procedura koja konvertuje binarni broj u prvi komplement

```
Sub Prvikomplement(niz() As Byte)
        'Svaka cifra se zamenjuje njenim komplementom do 1
  Dim i As Integer
        If niz(15) = 1 Then
              For i = 0 To 14
                    If niz(i) = 0 Then
                          niz(i) = 1
                    Else
                          niz(i) = 0
                    End If
              Next i
        End If
End Sub
```

Listing 3 Procedura koja konvertuje binarni broj iz prvog u drugi komplement

```
Sub Drugikomplement(niz() As Byte)
        'Dodaje se jedinica na mesto najmanje tezine
  Dim i As Integer
  Dim prenos As Byte
        prenos = 0
        If niz(15) = 1 Then
              niz(0) = 1 + niz(0)
              If niz(0) = 2 Then
                    niz(0) = 0
                    prenos = 1
              End If
              For i = 1 To 14
                    niz(i) = niz(i) + prenos
```

```
                    If niz(i) = 2 Then
                        niz(i) = 0
                        prenos = 1
                    Else
                        prenos = 0
                    End If
                Next i
        End If
End Sub
```

Listing 4 Funkcija za konvertovanje decimalnog broja (i pozitivnog i negativnog) u binarni

```
Function DekToBin(Decimalni As String, Duzina As Integer) As String
 'Duzina predstavlja broj bitova (duzinu reci 8-bitna, 16-bitna...)
   Dim i As Long, X As Long, bin As String
        bin = ""                        'Pocetna vrednost za binarni broj, pomocna promenljiva
        X = Val(Decimalni)              'Konvertovanje decimalnog stringa u ceo broj
        For i = Duzina - 1 To 0 Step -1 'Ukupno broj ima,  n-1 cifru
            If X And (2 ^ i) Then       ' Koristi se logicki "AND" operator
                bin = bin + "1"
            Else
                bin = bin + "0"
            End If
        Next
        DekToBin = bin
End Function
```

Listing 5 Funkcija za konvertovanje neoznačenog binarnog broja u decimalni

```
Function BinToDec1(BinBroj As String) As Long
   Dim rezultat As Long, i As Integer, eksponent As Integer, neg As Integer
        eksponent = 0
        For i = Len(BinBroj) To 1 Step -1
            Select Case Asc(Mid$(BinBroj, i, 1))
                Case 48                 ' ako je cifra "0" ne radi nista
                Case 49                 ' ako je cifra "1"
                    rezultat = rezultat + 2 ^ (eksponent)
            End Select
            eksponent = eksponent + 1
        Next
        BinToDec1 = rezultat
End Function
```

Listing 6 Funkcija za konvertovanje označenog binarnog broja u decimalni

```
Function BinToDec2(BinBroj As String) As Long
   Dim rezultat As Long, i As Integer, eksponent As Integer, duz As Integer
        duz = Len(BinBroj)
        If Mid$(BinBroj, 1, 1) = 1 Then         'ako je broj negativan (drugi komplement)
            rezultat = -1 * Val(Mid$(BinBroj, 1, 1)) * 2 ^ (duz - 1)
            For i = duz - 1 To 1 Step -1
                rezultat = rezultat + Val(Mid$(BinBroj, duz - i + 1, 1)) * 2 ^ (i - 1)
            Next i
        Else    'ako je broj pozitivan
            rezultat = 0
            For i = duz To 1 Step -1
```

```
                    rezultat = rezultat + Val(Mid$(BinBroj, duz - i + 1, 1)) * 2 ^ (i - 1)
                Next i
        End If
        BinToDec2 = rezultat
End Function
```

Listing 7 Procedura koja "secka" rečenici i pravi niz reči

```
Private Sub Iseckaj(Recenica As String, Reci() As String, brReci As Byte)
        Reci() = Split(Recenica)          ' indeks niza pocinje od nula
        brReci = UBound(Reci)             ' pravi broj reci je brReci+1
End Sub
```

Listing 8 Procedura koja simulaira pauzu

```
Public Sub Pauza(NbSec As Single)
  Dim Finish As Single
        Finish = Timer + NbSec / 50
        DoEvents
        Do Until Timer >= Finish
        Loop
End Sub
```

Listing 9 Procedura koji sa diska čita tekstualni dokument

```
Private Sub procitaj(dok As String)
  Dim fn As Integer
  Dim sText As String
  Dim ocitred As String
        sText = ""
        fn = FreeFile
        Open dok For Input As #fn
        Do While Not EOF(fn)
                Line Input #fn, ocitred
                sText = sText & vbCrLf & ocitred
        Loop
         Close #fn
End Sub
```

SKRAĆENICE

ALU	-	Arithmetic and Logic Unit
AOR	-	Arhitektura i Organizacija Računara
ARS	-	Arhitektura Računarskog Sistema
BKM	-	Butov Kodirani Množilac
CALKAS	-	Computer Architecture Learning and Knowledge Assessment System
CASLE	-	Compiler/Architecture Simulation for Learning and Experince
CAT	-	Computer Aided Teaching
CPU	-	Central Processing Unit
CSA	-	Computer Systems Architecture
EA	-	Efektivna Adresa
EMMA	-	Edinburg Microcoded Microprocessor Applet
ESCAPE	-	Environment for the Simulation of Computer Architectures for the Purpose of Education
HASE	-	Hierarchical Architecture design and Simulation Environment
IKT	-	Informacione i Komunikacione Tehnologije
ILP	-	Instruction Level Paralelism
IR	-	Instruction Register
IT	-	Information Technology
LMC	-	Little Man Computer
MAR	-	Memory Address Register
MBR	-	Memory Buffer Register
PC	-	Program Counter
PSW	-	Program Status Word
SIMRA	-	SIMulacija Računarske Arhitekture
SIV	-	Simulacija I Vizuelizacija
SPIECS	-	Software Package Integrated Educational Computer System
VPKM	-	Vrednost Para Kodiranog Množioca