Elektronski Fakultet
Univerzitet u Nišu

Simulacija konačnih automata kontrolisanih mikroprocesorom 8086

Arhitekture i projektovanje ugrađenih računarskih sistema

Student:
Saša Stojković
br. indexa: 11332

prof. dr. Mile Stojčev

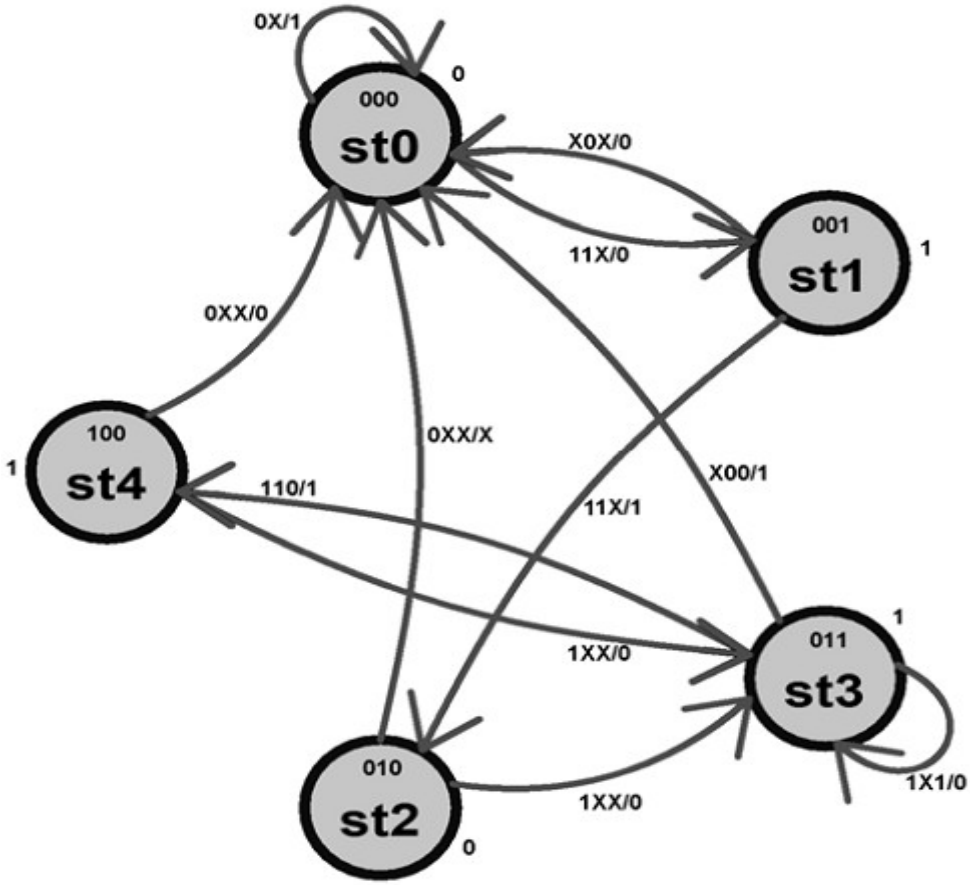
Sadržaj:

1. Uvod

U ovom poglavlju razmotrićemo osnovne termine i definiciji koji su nam neophodni za razumevanje i rešavanje problema vezanih za implementaciju automata kao i konkretnih problema koje srećemo u praksi a koji se lako mogu rešiti primenom teorije automata i njihovih osobina. Najznačajnije mesto u ovom poglavlju zauzeće sami automati i njihovi osnovni elementi, njihove definicije i vrste.

1.1 Konačni Automati

Automat prectavlja sekvencijalnu mašinu, definisanu svojim stanjima (stanja u kojima mašina može da se nađe) i funkcijama ili zakonima promene stanja (uslovi prelaska iz jednog stanja u drugo). Pošto bi sekvencijalna mašina zauvek ostala u istom stanju ukoliko nebi došlo do uticaja iz spoljne sredine, to moramo automatu pridružiti i potencijalne ulaze (svi signali spolja koji pod određenim uslovima mogu uticati na promenu stanja) a pošto automat u najvećem broju slučajeva mora da interaguje sa spoljnom sredinom, to automatu moramo pridružiti i izlaz karakterističan za svaki od njegovih stanja.



1.1 Šema konačnog automata

Automati se matematički mogu prectaviti kao unija skupova stanja u kojima se automat (sekvencijalna mašina) može naći, skupa funkcija promena stanja, skupa ulaza i skupa izlaza.

Konačni automati prectavljaju automate sa konačnim brojem stanja. Konačni automati se u praksi najčešće prectavljaju grafički, pomoću grafova

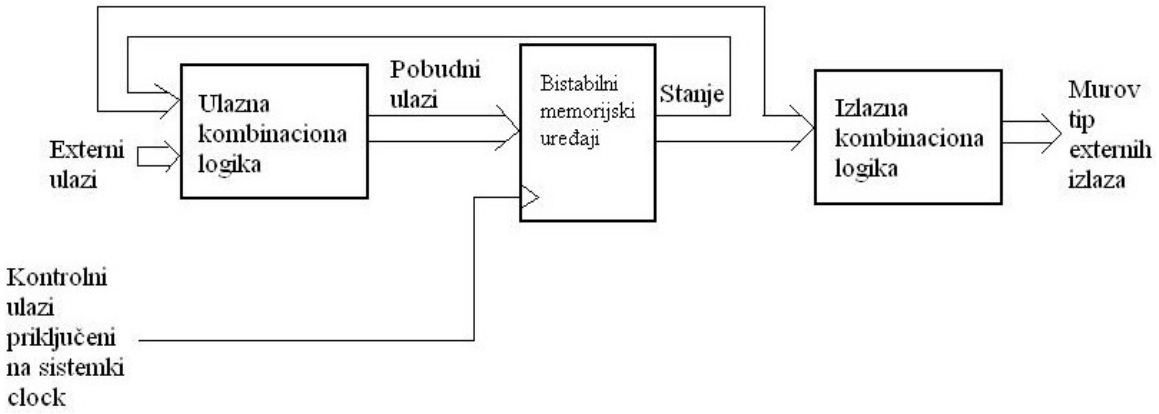
Čvorovi grafa prectavljaju stanja automata. Grane grafa, koje povezuju dva različita stanja prectavljaju prelaz automata između ta dva stanja pri čemu se kraj grane vezan za stanje iz koga automat prelazi ne obeležava posebno, a kraj grane koji se završava u stanju u koje automat prelazi, obeležava se strelicom. Uslov ili funkcija pod kojim se aktivira ta grana, tj. automat menja stanje, nalazi se ispisan neposredno pored grane. Ukoliko grana izvire i uvire u isti čvor, ta grana prectavlja uslov pod kojim automat ostaje u tom stanju tj. ne menja stanje.

Konačni automati se mogu sresti u dve varijante: Murov i Milijev konačni automat.

1.2 Murov automat

Murov automat se definiše skupom stanja u kojima se automat može naći, skupom funkcija prelaza iz stanja u stanje kao i skupom izlaza koje će generisati automat za svako stanje.

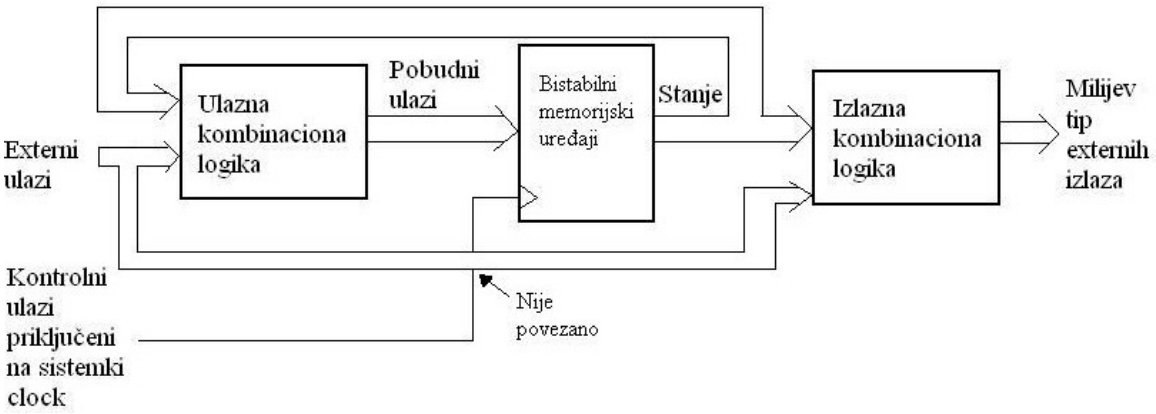
Naime, kod Murove mašine (murov automat) trenutni izlaz iz automata je funkcija samo stanja u kome se u tom trenutku nalazi automat, nezavisno od trenutnog stanja na ulazu automata.



1.2 Konačni automat Murovog tipa

1.3 Milijev automat

Za razliku od Murove mašine, Milijeva mašina (Milijev konačni automat) definiše se skupom stanja u kojima se automat može naći, funkcijama prelaza između stanja koje za razliku od Murovog automata zavise i od stanja u kome se automat trenutno nalazi, kao i skupom izlaza.



1.3 Konačnia automat Milijevog tipa

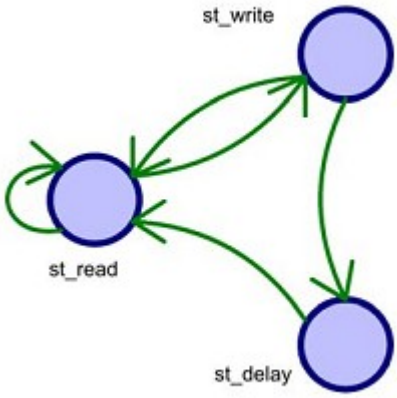
Razlika između Murovog i Milijevog automata jeste u tome što je kod Murovog automata, trenutni izlaz iz automata određen isključivo stanjem u kome se automat nalazi u tom trenutku a kod Milijevog automata, izlaz je određen trenutnim stanjem automata kao i stanjem na ulazu automata.

2. Implementacija

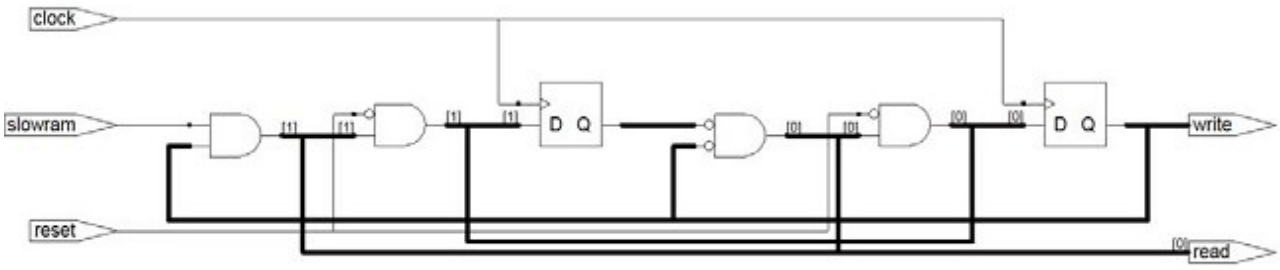
U praksi, konačni automati se najčešće projektuju i realizuju primenom elektronskih komponenti i logičkih kola. Iz ovog možemo zaključiti da je za realizaciju automata neophodan hardver, koji od slučaja do slučaja tj. od automata do automata, može biti složeniji ili jednostavniji. Direktno zavisna od primenjenog hardvera jeste i cena. Što više komponenti upotrebimo, veća će biti i cena realizacije automata.

Prednosti ovako realizovanih automata jeste velika brzina i relativno mala cena.

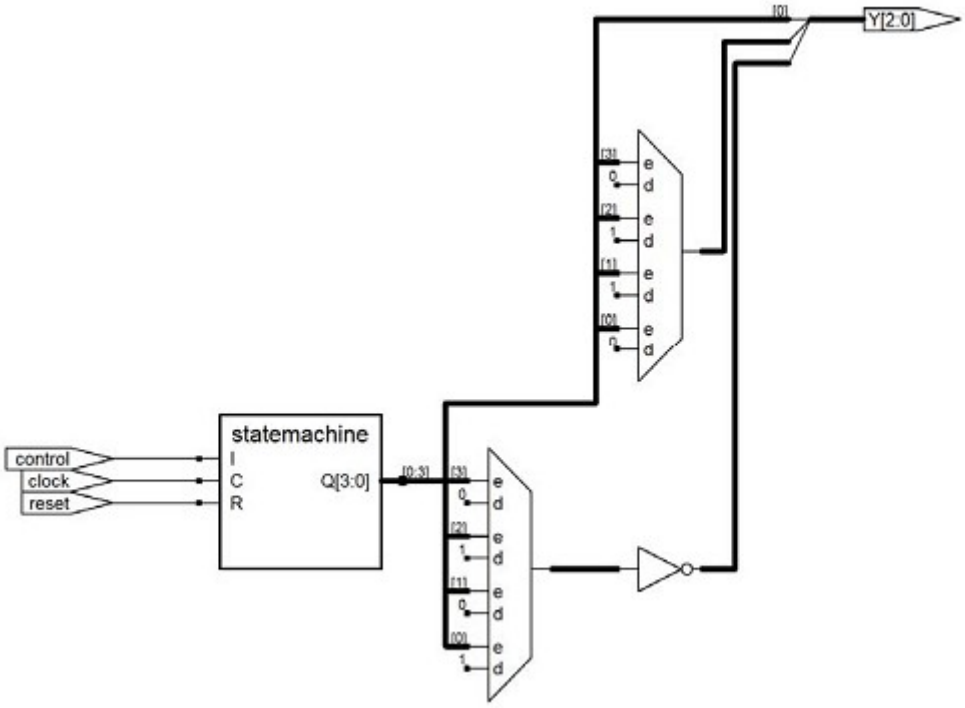
Na sledećoj slici je prikazan graf relativno jednostavnog automata. Na sledeće dve slike, prikazane su dve elektronske šeme. Na prvoj se nalazi šema realizacije automata a na drugoj šema elektronskog uređaja u koni je ugrađen konačni automat.



2.1 Šema automata prectavljenog grafom



2.2 Elektronska šema za realizaciju gore prikazanog automata



2.3 Šema uređaja u koji je ugrađen automat

2.1 Softverska implementacija

Čisto softverska implementacija konačnih automata koristi ovaj opisani model automata a nalazi svoju primenu u raznim programskim jezicima i problemima u programiranju i projektovanju. Model automata je dosta pogodan za rastavljanje i uprošćavanje složenih problema.

Jedna konkretna i najočiglednija primena modela konačnog automata u programiranju jeste problem traženja reči u zadatom tekstu. Ovo je najočigledniji primen koji se u potpunosti podudara sa modelom konačnog automata koji ima onoliko stanja koliko karaktera ima trežena reč.

Svaki programer se pre ili kasnije sretne sa ovim problemom. U najkraćim crtama, problem je naći odgovarajuću podsekvencu znakova u sekvenci znakova.

Opis rešenja je, opet u najkraćim crtama, prvo naći prvi karakter iz sekvence. Zatim, počevši od te pozicije, nastaviti sekvencijalno traženje svih znakova iz tražene sekvence, respektivno. Ukoliko bar jedan znak nije na mestu tj. nije onaj koji očekujemo, prekida se traženje, i nastavlja dalje sa traženjem sledećeg pojavljivanja prvog karaktera, naravno pri čemu se preskače prethodno upoređeni deo.

Ako pažljivo pogledamo, ovo rešenje prectalja realizaciju konačnog automata. Znači, imamo stanje S0, koje označava da znak na ulazu nije prvi znak iz tražene sekvence. U trenutku kada na ulaz dođe znak koji prectavlja prvi karakter tražene sekvence, automat prelazi u prvo pobuđeno stanje S1. Ukoliko sada na ulazu naiđe drugi karakter iz tražene sekvence, automat prelazi u stane S2 i tako redom dok ne pređe u zadnje stanje kada signalizira da je našao reč.

Iz bilo kog pobuđenog stanja, automat može preći u S0 stanje ukoliko se na ulazu ne pojavi očekivani karakter.

Primetićemo da je ovaj opisani automat Milijevog tipa, jer izlaz tj. signal i sledeće stanje zavise i od karaktera koji će se pojaviti na ulazu, a ne samo od stanja koje je aktivno.

2.2 Mikroprocesorsko upravljani konačni automat

Mikroprocesori predstavljaju jedan alternativni pristup za sintezu sinhronih sekvencijalnih mašina. Primena mikroprocesora obezbeđuje sledećih nekoliko prednosti u odnosu na konvencionalne sinhrone sekvencijalne mašine, tzv. sinhrone FSM-ove ili sinhrone konačne automate:

1. Mikroprocesor ima izvršnu jedinicu koja pruža značajnu fleksibilnost kod sinteze sekvencijalnih mašina jer u svom repertoaru poseduje bogat skup aritmetičkih i logičkih instrukcija. Tako na primer, moguće je da se naredno stanje mašine odredi pomoću neke od aritmetičkih operacija, kakva je recimo, sabiranje ili oduzimanje, a zatim da se dobijeni rezultat komparira sa unapred određenom vrednošću sa ciljem da se aktivira prelazak mašine iz jednog stanja u drugo.

2. Druga prednost se sastoji u sledećem: Mikroprocesor može putem korišćenja softverskih kašnjenja da unese promenljiva vremenska kašnjenja između stanja mašine. Drugim rečima, trajanje stanja mašine nije više ograničeno, tj. vezano sa trajanjem taktnog intervala. Definisanje promenljivog vremena trajanja između dva stanja, kod standardnog pristupa, zahteva uvođenje eksternog uređaja (logike) tipa brojač koji se mora prethodno postaviti na vrednost željenog kašnjenja što u suštini usložnjava i poskupljuje rešenje.

3. Treća prednost je veoma značajna i sastoji se u sledećem: U odnosu na iznos odgovarajućeg hardvera moguće je generisati veliki broj stanja. Drugim rečima, dodavanjem izlaznih portova, broj stanja koji je dostupan spoljnim uređajima može drastično da se poveća. Ipak kod najvećeg broja sistema, dodatna stanja uglavnom se odnose samo na povećanje softvera a ne i hardvera.

4. Četvrta prednost se tiče uvođenja promena (izmene) u radu mašine. Ako se specifikacijama mašine zahtevaju promene u dijagramu stanja tada se te modifikacije obično implementiraju softverski. To je u suprotnosti sa konceptom realizacije *hardwired* mašina kod kojih promene mogu da zahtevaju izbacivanje ili dodavanje novih veza (ožičenje) u kolu. To znači da je mikroprocesorsko upravljana sekvencijalna mašina značajno fleksibilnija u odnosu na *hardwired* mašinu.

5. Peta prednost se sastoji u sledećem: U okviru jednog mikroprocesorskog sistema moguće je konfigurisati veći broj mašina. Svaka mašina se upravlja od strane različitog programa koji je smešten u memoriji mikroprocesorskog sistema. Takođe, svaka mašina je povezana sa odgovarajućim spoljnim kolom preko jedinstvenih U/I portova. Pri ovome prvenstveno zbog konačnog vremena izvršenja programa postoje i neka ograničenja, a ona se odnose na broj (posebnih) mašina koji se može implementirati od strane jednog procesora.

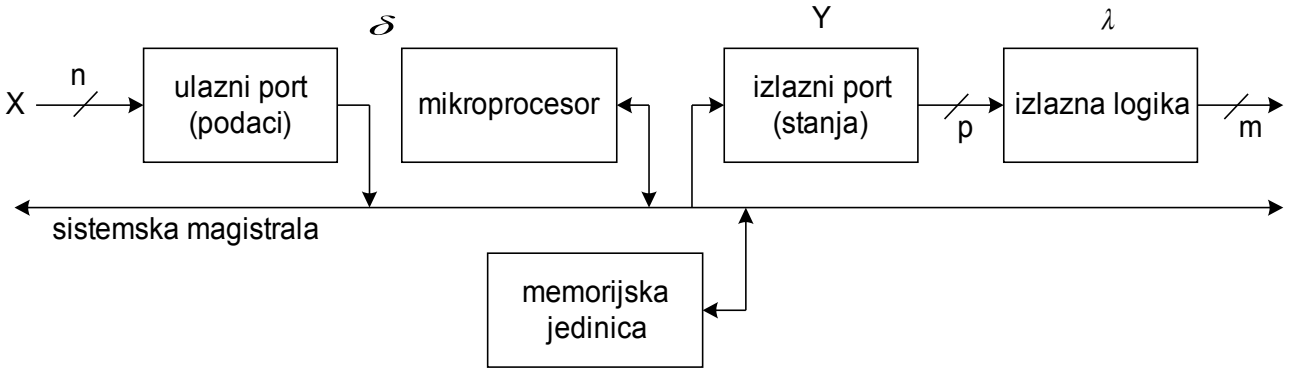
6. Šesta prednost je sledeća: Operacije mašine mogu se jedinstveno definisati od strane softvera. Nakon što je već jedanput hardver izveden, funkcionalnost mašine je određena od načina programiranja hardvera pri čemu moraju biti zadovoljene odgovarajuće specifikacije mašine. Oba tipa FSM mašine, Moore-ov i Mealy-ev automat, se mogu implementirati od strane mikroprocesora.

7. Sedma prednost se ogleda u sledećem: Moguće je aplicirati velike (složene) mašine na koje je moguće povezati veći broj perifernih uređaja. Struktura prekida kod mikroprocesora omogućava spoljnjem uređaju da inicira U/I operaciju pomoću generisanja zahteva za prekid. Svakom uređaju se dodeljuje jedinstveni prekidni vektor koji usmerava procesor da iz memorije izvrši odgovarajuću prekidno-uslužnu rutinu sa ciljem da se opsluži uređaj koji je zahtevao uslugu.

Mikroprocesor mora prvo da pročita ulazne signale sa ulaznog porta, zatim obavi operacije nad ulazima, i na kraju upiše rezultat u izlazni port kako bi definisao stanje na izlazu. Na slici je prikazan opšti blok dijagram mikroprocesorski kontrolisane sekvencijalne mašine kod koje se jasno identifikuju karakteristike narednog stanja, memorijskih elemenata, i izlazne logike.

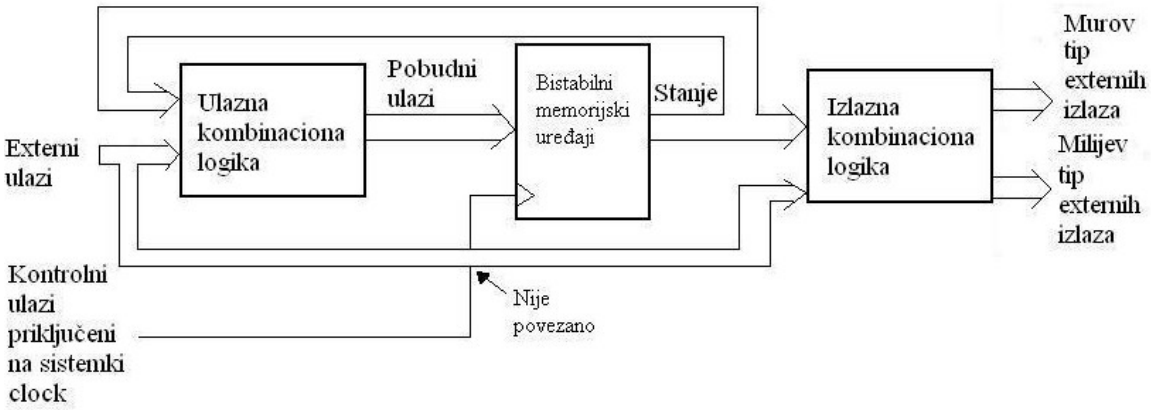
Treba pri ovome naglasiti da ne postoji potreba za hardverski izvedenom povratnom spregom od registra-koji-čuva-tekuće-stanje (izlazni port) do ulaznog-registra-za-podatke (ulazni port) koji generiše naredno stanje. Softver je taj koji generiše naredno-stanje u funkciji samo tekućeg stanja, ili generiše naredno stanje u zavisnosti od tekućeg stanja i prisutnih ulaza.

Mikroprocesor prvo čita stanje ulaznog porta. Nakon toga sledi programska sekvenca u toku koje on testira stanje ulaznih signala. Ako je određeni uslov ispunjen softver će generisati novo stanje i upisati to stanje u izlazni port.



2.4 Strukturna šema automata kontrolisanih mikroprocesorom

Tri tipa modela kola kanačnih automata dati su na naredne tri slike. Svaki model se razlikuje od načina postavljanja izlaza. Kolo na slici 1.2 se zove konačni automat Murovog tipa, kod koga se može primetiti da izlazi zavise jedino od trenutnog stanja pa se izlazi kod ovog kola nazivaju Murovi izlazi. Na slici 1.3 je prikazano kolo Milijevog konačnog automata kod koga izlazi zavise i od ulaza i od konačnog stanja pa se izlazi kod ovog kola nazivaju Milijevi izlazi. Na slici 2.5 je prizan konačni automat koji ima izlaze oba tipa, Milijev i Murov, pa se zbog toga ovi automati nazivaju konačni automati mešovitog tipa.

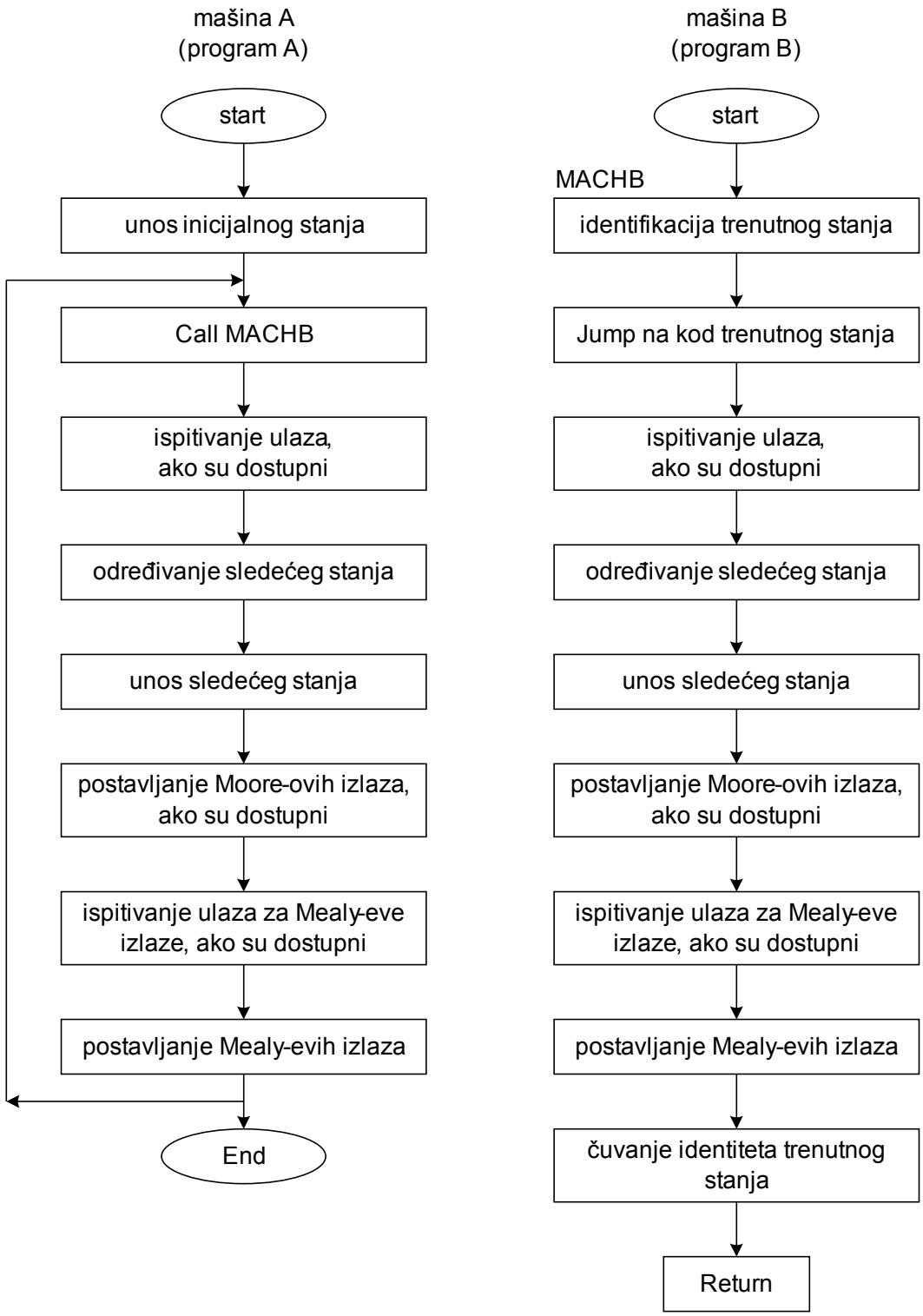


2.5 Konačni automat mešovitog tipa

2.3 Veći broj mašina

Prednost, moć i univerzalnost mikroprocesorski baziranih rešenja se može efikasno iskoristiti za sintezu većeg broja FSM-ova pomoću jednog mikroračunara. I pored toga što mikroprocesor u datom trenutku upravlja radom samo jednim FSM-om, FSM-ovi u suštini rade konkurentno. Na slici 17 prikazani su ulazni i izlazni registri za dve mašine A i B pri čemu se svaka upravlja od strane istog mikroprocesora.

Kod ove organizacije se pretpostavlja da postoje izdvojeni portovi za izlaze mašina. Svakom od šest portova se pristupa različitom adresom. Prelaz između programskih segmenata koji upravljaju radom svake mašine se ostvaruje izvršenjem intersegmentne instrukcije CALL ili pomoću prekida. Intersegmentna CALL instrukcija prenosi pravo upravljanja između kodnog segmenta mašine A i kodnog segmenta mašine B. Na slici 18 prikazan je pojednostavljeni dijagram toka za programe koji upravljaju radom mašina A i B. Aplikacioni program mašine A (program A) počinje ulaskom u inicijalno stanje. Program zatim izvršava intersegentnu instrukciju CALL kojom se poziva procedura MACHB čime počinje izvršenje programa za mašinu B (program B). Lokacija tekućeg stanja u programu B je prethodno bila zapamćena u *Jump* tabeli. Program B zatim skače na sekvencu instrukcija koje se odnose na tekuće stanje.



2.6 Strukturni blok dijagram upravljanja dva automata

Program A nakon toga ponovo poziva program B pomoću druge intersegmentne CALL instrukcije i proces se ponavlja.

Struktura prekida mikroprocesorskog sistema predstavlja alternativn i pristup za prenos upravljanja između aplikacionih programa.

Važni kriterijum kod prekidno upravljanih sistema predstavlja određivanje prioriteta mašina. Tako na primer, mašini A može biti dodeljen viši prioritet opsluživanja u odnosu na mašinu B. Nakon uključenja sistema na napajanje obe mašine se inicijaliziraju na neko unapred određeno stanje, da bi se u tom trenutku generisala dva zahteva za prekid. Arbitar određuje koja će mašina prva biti opslužena. U suštini, arbitar je kolo koje rešava probleme dodele prioriteta u opsluživanju kada se istovremeno javi zahtev za opsluživanjem od strane većeg broja mašina. Kada se INTR signal primi od strane mašine koaj je zahtevala uslugu, na magistrali podataka se u trenutku prihvatanja zahteva za prekid postavlja vektor-broj. Vektor-broj ukazuje na početnu adresu rutine za obradu prekida.

Druga tehnika koja se često koristi za prihvatanje zahteva za prekid (preko jedinstvene grupne linije INTR) i identifikacije izvora za prekid se bazira na tehnici kružnog ispitivanja (*pooling technique*). Kod ova tehnike metodom kružnog ispitivanja testira se status svih potencijalnih inicijatora prekida. Treći pristup prihvatanja zahteva za prekid se može realizovati tehnikom lančanja (*daisy chain*).

3. Opis programa

U narednom delu reći ću nešto o korišćenoj tehnologiji u izradi programa, arhitekturi i hierarhiji klasa koje pripadaju projektu, kao i omehanizmima koji se interno koriste da zamene hardverske (fizičke) komponente potencijalne hardverske implementacije ovakvog sistema.

3.1 Tehnologija korišćena u izradi

Program FSM x86 prectavlja Windows desktop aplikaciju, razvijenu u Microsoft Visual Studio 2008 Professional Edition razvojnom okruženju, tj. u Framework-u 2.0. Ovo okruženje podržava i Framework 3.5 ali smatram da nije potreban tako masivan mehanizam i podrška za ovakav projekat.

Rešenje se sastoji iz jednog projekta, windows desktop aplikacije tj. projekta, koji u sebi sadrži četiri forme, za svaki od prikazanih primera po jedan. Takođe sadrži i četiri klase automata i jednu klasu interpretatora instrukcija 80x86 mikroprocesora.

3.2 Arhitektura programa

Program se sastoji iz dva glavna dela. To su korisnički interfejs i za korisnika nevidljivi mehanizam mikroprocesora sa četiri pridružena automata.

Korisnički interfejs prectavljaju četiri forme koje korisnik vidi (prozori) i na kojima preduzima svoje akcije i vidi rezultat svoih akcija. Na njima se nalaze prikazani i registri mikroprocesora, kao i njegov segment za kod, i graf automata kojim on upravlja tj. njegov model koji korisnik treba da realizuje.

Instrukcije ispisane sa leve strane na svakom prozoru, izvršavaju se pritiskom na F10 taster na tastaturi. Moguće je proizvoljno izvršenje odabrane instrukcije iz lista tj. klikom na neku od instrukcija i njenim markiranjem, odredili ste koju ćete instrukciju izvršiti kada pritisnete F10.

Program omogućava korišćenje 16 bitnih registar procesora 80x86 kao i njihovih 8 bitnih delova.

Rezultat svake instrukcije možete videti na bilo kom od registara, kao i promene na flegovima ukoliko postoje.

Ukoliko neko stanje na automatu postane aktivo, biće obojeno crveno.

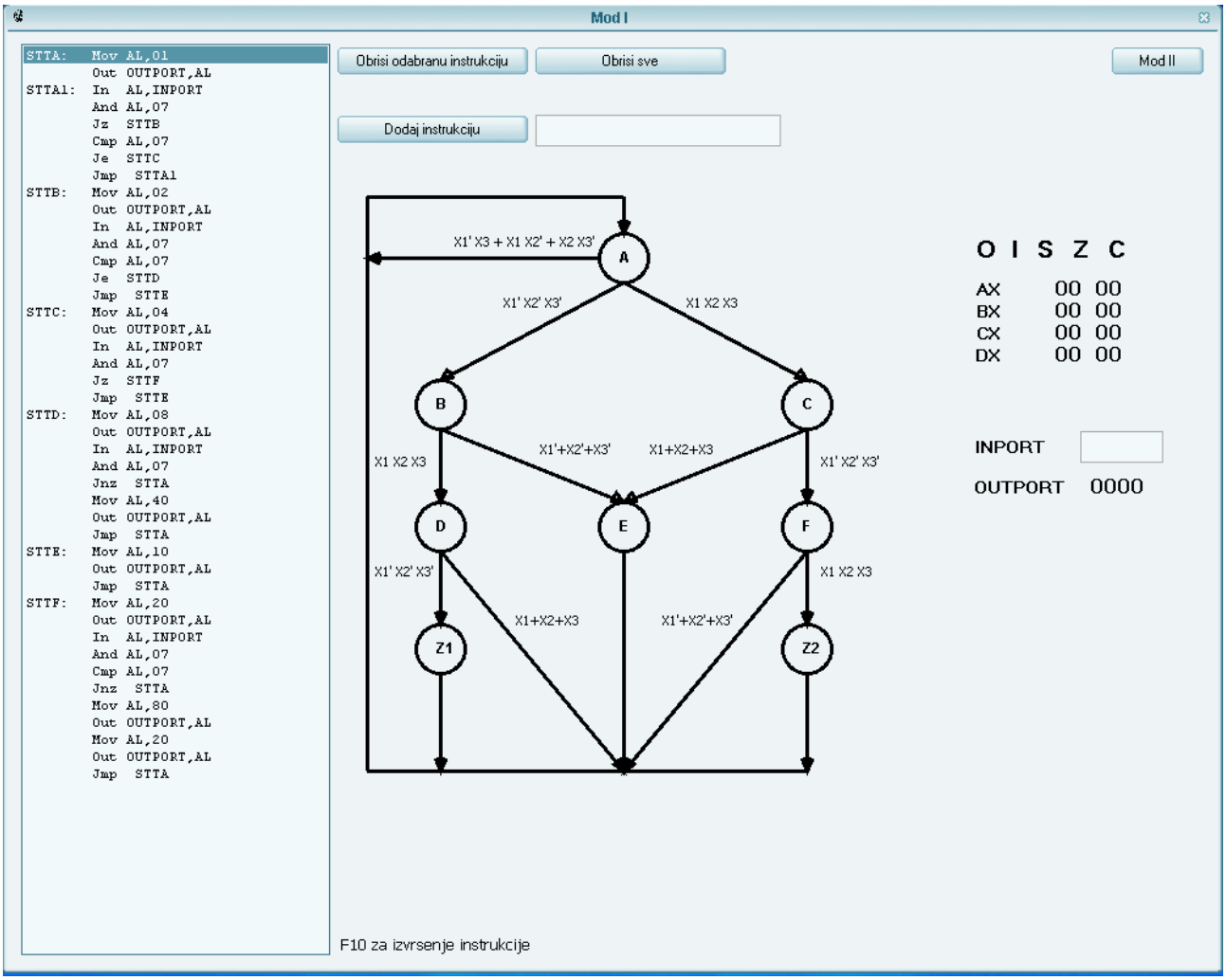
Deo koji korisnik ne vidi prectavlja interpretator instrukcija mikroprocesora 80x86 koji direktno i koristi modele prikazanih na prozorima i njima prividno upravlja, preko INPORT i OUTPORT, koji prectavljaju ulazni i izlazni port, respektivno. Interpretator ima ugrađen validator za instrukcije.

Postoje četiri klase konačnih automata, zbog njihovih karakteristika. Svaki automat prikazan na formama prectavlja posebnu implementaciju, tj. posebnu klasu.

4. Opis primera

U narednom delu objasniću svaki od primera koji se mogu naći u programu za simulaciju FSM x86.

Primer 4.1



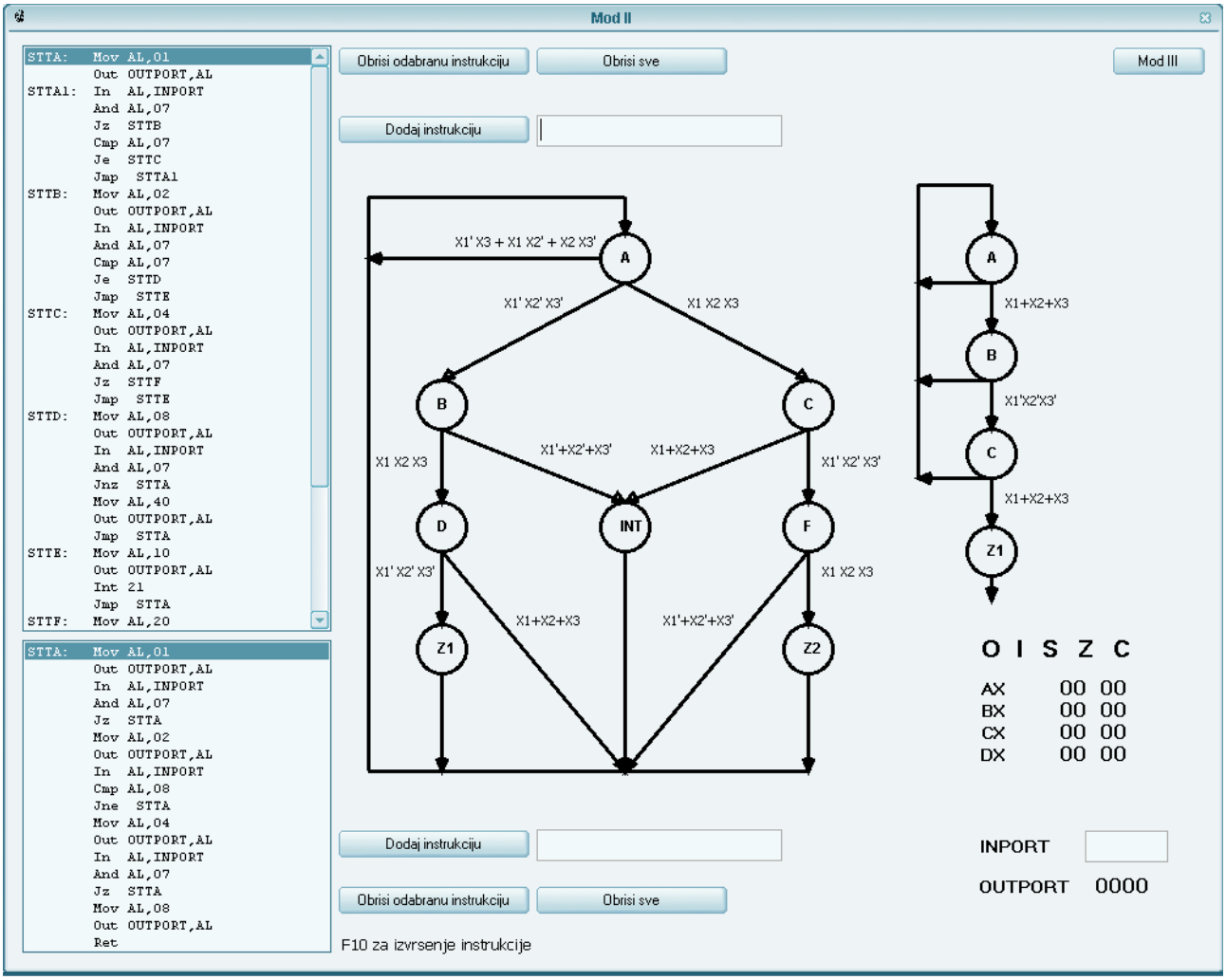
4.1 Slika primera 1 iz programa

Ovo je primer koji je opisan u delu teksta posvećenom konačnim automatima kontrolisanim pomoću mikroprocesora. Radi se o automatu koji na ulaznom portu prihvata promenljivu dužine tri bita tj. X1 X2 X3. Ukoliko se na ulazu pojavi niz bitova 100 odnosno 04h automat prelazi u prvo pobuđeno stanje. Da bi automat prešao u neko od sledećih pobuđenih stanja, na ulazu mora da se pojavi 07h. Ukoliko se pojavi bilo šta drugo, automat ponovo prelazi u osnovno stanje. Ukoliko se na ulazu ponovo pojavi 04h, automat prelazi u ciljno stanje tj. Z.

U primeru vidimo da se ulaznom portu pristupa preko adrese INPORT a izlaznom preko adrese OUTPORT. Ovaj pristup se moze koristiti i za ostale automate, kao što je korišćen i u ostalim primerima navedenim ovde.

Izvršavanje instrukcije je pomoću tastera F10. Celokupno rešenje je moguće obrisati i napisati novo, po sopstvenoj ideji. Automat će se ponašati u skladu sa napisanim programom.

Primer 4.2



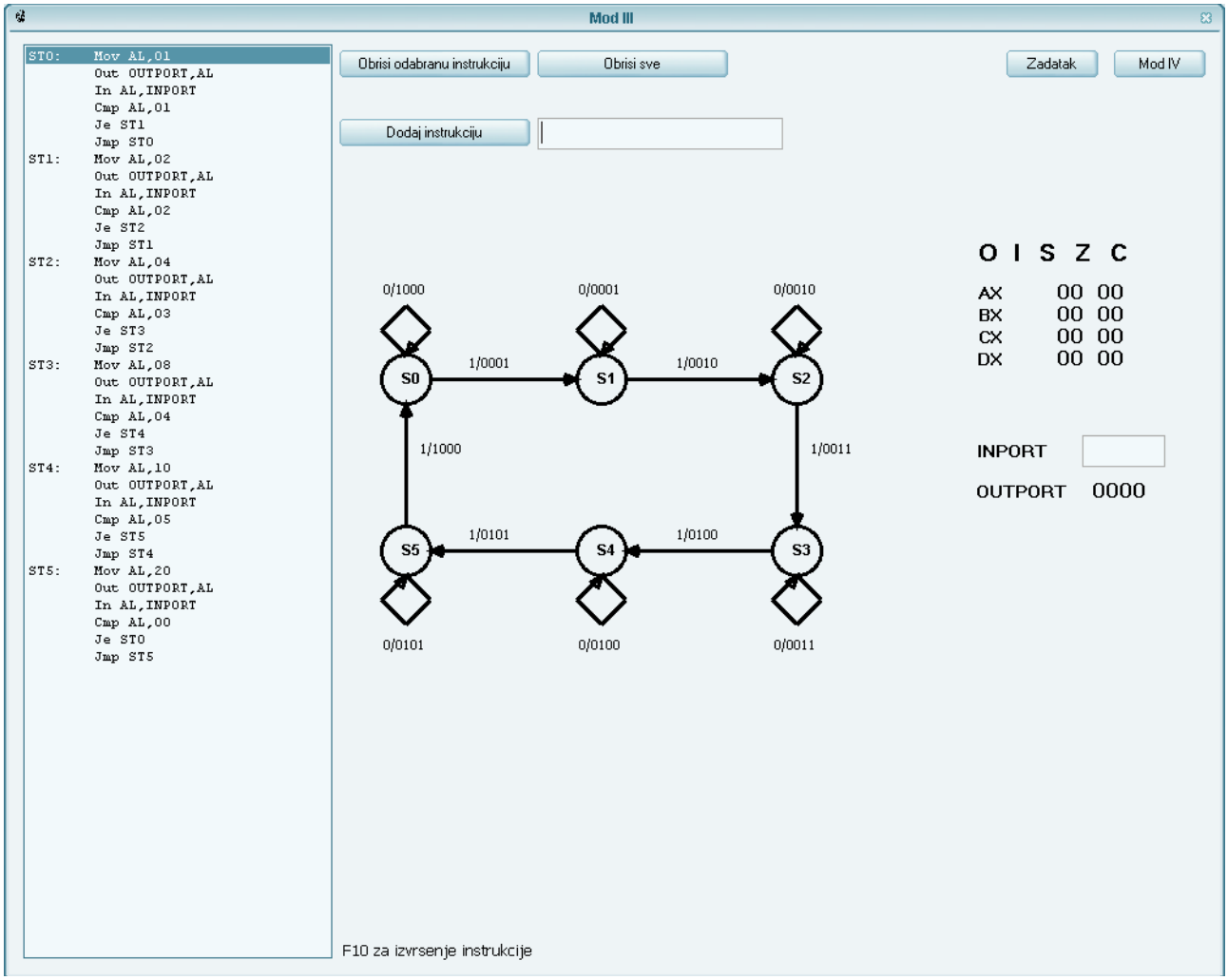
4.2 Slika primera 2 iz programa

Drugi primer je varijacija na prethodni. Jedina specifičnost ovog automata jeste što prectavlja spregu dva automata.

Komunikacija ova dva automata tj. prenošenje kontrole sa jednog na drugi se obavlja pomoću prekida (Int 21h u ovom primeru). Osnovni automat je isti kao u prethodnom primeru, pri čemu kada automat dođe u stanje E iz prethodnog primera, kontrola se prenosi na drugi automat. Upravljanjem drugim automatom bavi se prekidni pod program koji je prikazan u donjem levom uglu prozora. I njega možemo u potpunosti izmeniti. Oba primera programa su ispisana da implementiraju automate sa slike.

Povratak u glavni program se obezbeđuje instrukcijom Ret u prekidnom programu. Nastavak izvršenja programa se nastavlja od selektovane naredbe. Ukoliko se korisnik ne meša u izvršenje, program će nastaviti od sledeće instrukcije, tj. odmah nakon prekida.

Primer 4.3

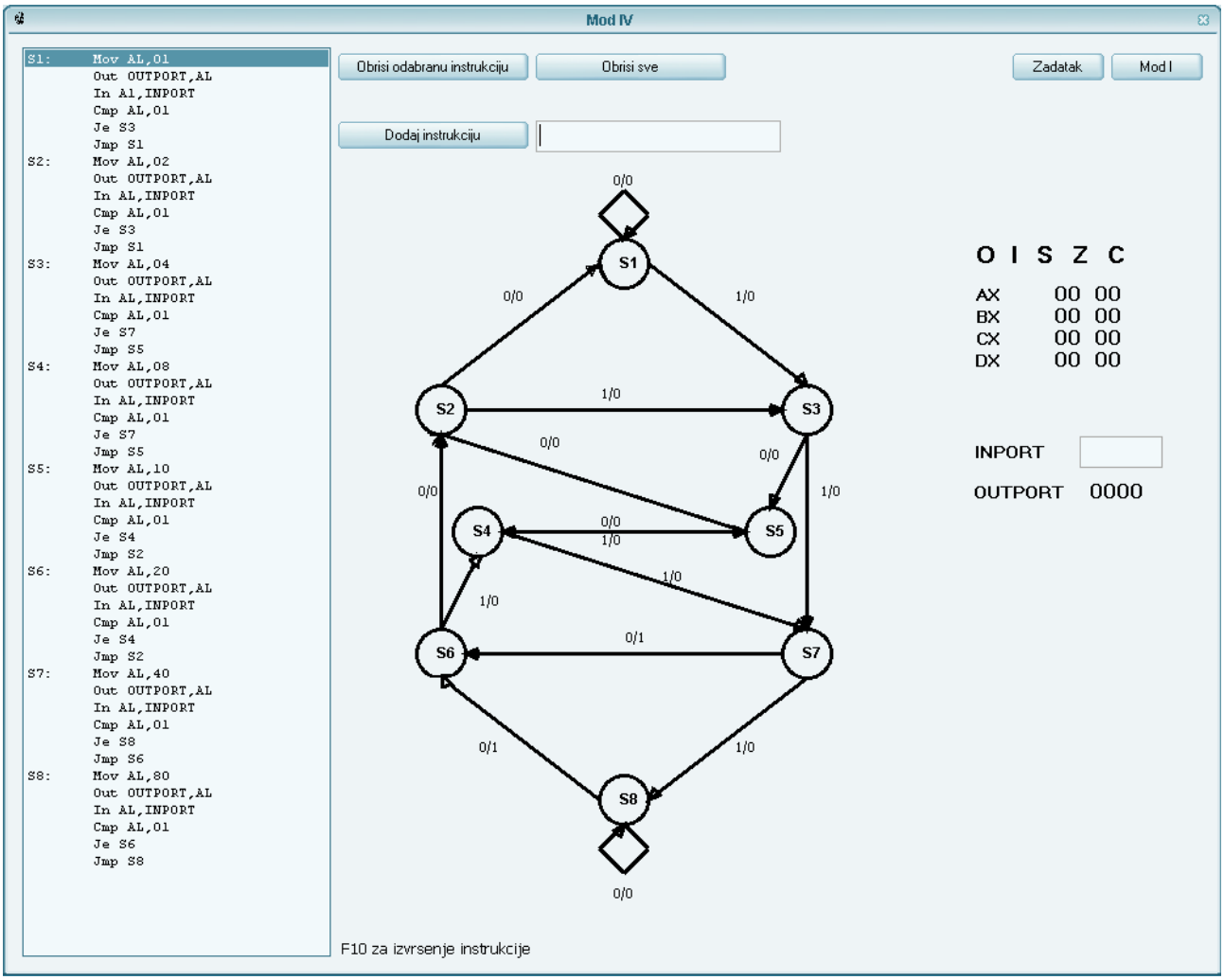


4.3 Slika primera 3 iz programa

Treći primer prectavlja automat brojač po modulu šest. Ovaj automat očekuje najpre 01h na svom ulazu. Nakon toga prelazi u prvo pobuđeno stanje i tako dalje, prihvatajući brojeve od 1 do 5 redom. Kada prihvati broj 5 očekuje broj 0 kako bi se opet vratio u osnovno stanje. Ukoliko u bilo kom trenutku automat na ulazu dobije cifru koju ne očekuje, ostaje u tekućem stanju.

Ovaj automat je u primeru inplementiran kao i prethodna dva.

Primer 4.4



4.4 Slika primera 4 iz programa

Automat prikazan na slici prectavlja bit komparator. On u suštini traži niz bitova 110 i u tom trenutku postavlja prva dva stanja aktivna tj. njihove izlaze. U bilo kom drugom slučaju ne postavljamo ni jedno stanje aktivnim. Iz svakog od prikazanih stanja, automat može preći u neko od dva raspoloživa, tj. zavisno dali mu se na ulazu pojavi 0 ili 1.

Ovaj sistem nije u stanju da detektuje prva tri bita, ali je nakon toga sposoban da do beskonačnosti testira ulaze. Ovaj problem možemo rešiti tako što ćemo pre početka rada, sistem postaviti u neko od stanja. Na taj način možemo pokriti taj jedninstveni slučaj.

Ovaj automat se može realizovati na više načina i po sopstvenim idejama. Prikazani primer poštuje načine implementacije prikazane u prethodnim primerima u ovom programu.

5. Zaključak

Na osnovu prethodno nabrojanih prednosti, čitaocu se na prvi pogled može učiti da mikroprocesorsko upravljane sekvencijalne mašine predstavljaju znatno bolje tehničko rešenje u odnosu na mašine koje se implementiraju sa diskretnim komponentama. Ipak kod ovakvog pristupa postoje i neki značajni nedostaci koji se tiču primene mikroprocesora kod sinteze sekvencijalnih mašina. Kao prvo, mikroprocesorski sistem je veoma skup za implementaciju kod malih mašina.

Za korektan rad mikroprocesorskih sistema potreban je rad i podrška drugih kola kakva su taktni generator, RAM i ROM memorija, U/I portovi, i odgovarajuća adresno dekoderska logika. Cena ovakvog sistema je suviše velika ako mašina treba da ima samo nekoliko stanja. Jedno od rešenja je da se koriste mikrokontroleri koji u sebi imaju ugrađen ograničen iznos RAM-a, ROM-a i U/I portova.

Kao što smo već pomenuli, značajnu prednost mikroprocesorski baziranih sekvencijalnih mašina predstavlja to što se one mogu programirati. To sa jedne strane predstavlja prednost jer ovakva mogućnost pruža fleksibilnost u radu, ali sa druge sttrane je nedostatak jer zahteva da se koristi još jedan uređaj koji se naziva ROM programator. Ovaj programator se koristi za upis programa mašine u ROM memoriju sistema. Relativno posmatrano cena ROM programatora nije visoka, ali će cena cele FSM biti visoka ako ona ima samo nekoliko stanja. Kod novijih verzija mikrokontrolera postoji mogućnost loadovanja programa preko JTAG interfejsa što u značajnoj meri smanjuje cenu proizvoda a time i čini ovakav proizvod konkurentnim.

Treći nedostatak je verovatno najznačajniji. Nasuprot sekvencijalnih mašina sintetizovanih pomoću diskretnih komponenti, mikroprocesorski kontrolisane sekvencijalne mašine su suviše spore. Po nekoliko ciklusa instrukcija je potrebno da se izvrše u toku svakog prelaza iz jednog stanja u drugo. U toku ovog vremena, sa ciljem da se odredi naredno stanje, ulazni signali se čizaju preko ulaznog porta. Sa druge strane, izlazi su dostupni samo kada su odgovarajući bitovi na izlaznom portu postavljeni. Ulazno/izlazne operacije se realizuju pomoću relativno sporog aplikacionog programa, pri čemu U/I uređaji mogu sa aspekta adresibilnosti da pripadaju memorijsko preslikanom ili izdvojenom U/I prostoru. Kod izdvojenog U/I prostora U/I operacije se obavljaju IN i OUT naredbama, dok kod memorijsko preslikanog koncepta prenos podataka ostvaruje se pomoću instrukcija za obraćanje memoriji kakve su MOV, LOAD, STORE, i druge.

Četvrti nedostatak, problem gličeva. Za kratak vremenski period oba stanja su potvrđena, ali ipak treba naglasiti da neće doći do gliča - tj. mašina ne

prelazi kroz drugo stanje iz razloga što se stanja flip-flopova kod ovakvog rešenja ne dekodiraju sa ciljem da se odredi drugo stanje kakav je to slučaj kod FSM-ova sa diskretnim elementima. Naime, korišćenjem mikroprocesorske implementacije kod sekvencijalnih mašina, stanje se predstavlja jedinstvenim bitom u *state register*, a ne dobija se putem dekodiranja stanja na izlazima flip-flopova.

Mikroprocesor je idealan za realizaciju velikih sekvencijalnih konačnih automata pod uslovom ako mašina može da toleriše relativno dugo trajanje između stanja.

6. Vežbe

6.1 Vežba 1

Zadatak za samostalni rad ili vežbu: *Automat prikazan na slici u primeru 4.1 programa FSM x86 realizovati softverski primenom instrukcija iz skupa instrukcija arhitektura familije mikroprocesora x86. Ulazni port mikroprocesora dat je na adresi INPORT a izlazni na adresi OUTPORT. Izvršiti sintezu automata u verziji:*

a) Murov konačni automat
b) Milijev konačni automat.

6.2 Vežba 2

Zadatak za samostalni rad ili vežbu: *Automat prikazan na slici u primeru 4.2 programa FSM x86 realizovati softverski primenom instrukcija iz skupa instrukcija arhitektura familije mikroprocesora x86. Ulazni port mikroprocesora dat je na adresi INPORT a izlazni na adresi OUTPORT. Izvršiti sintezu automata u verziji:*

a) Murov konačni automat
b) Milijev konačni automat.

6.3 Vežba 3

Zadatak za samostalni rad ili vežbu: *Automat prikazan na slici u primeru 4.3 programa FSM x86 realizovati softverski primenom instrukcija iz skupa instrukcija arhitektura familije mikroprocesora x86. Ulazni port mikroprocesora dat je na adresi INPORT a izlazni na adresi OUTPORT. Izvršiti sintezu automata u verziji:*

a) Murov konačni automat
b) Milijev konačni automat.

6.4 Vežba 4

Zadatak za samostalni rad ili vežbu: *Automat prikazan na slici u primeru 4.4 programa FSM x86 realizovati softverski primenom instrukcija iz skupa instrukcija arhitektura familije mikroprocesora x86. Ulazni port mikroprocesora dat je na adresi INPORT a izlazni na adresi OUTPORT. Izvršiti sintezu automata u verziji:*

a) Murov konačni automat
b) Milijev konačni automat.