



UNIVERZITET SINGIDUNUM

FAKULTET ZA INFORMATIKU I RAČUNARSTVO

Emulator arhitekture računara Milantex Tx16

- Diplomski rad -

Mentor:

Prof. dr Mladen Veinović

Student:

Milan Tair

Br. indeksa: 2008213514

Beograd, 2012



UNIVERZITET SINGIDUNUM

FAKULTET ZA INFORMATIKU I RAČUNARSTVO

UNIVERZITET SINGIDUNUM

FAKULTET ZA INFORMATIKU I RAČUNARSTVO

Beograd, Zorana Đinđića 44

Kandidat: _____

Broj indeksa: _____

Smer: _____

Tema: _____

Datum odobrenja rada:

Beograd, _____._____._____.

MENTOR

Prof. dr Mladen Veinović

DEKAN

Prof. dr Mladen Veinović

Emulator arhitekture računara Milantex Tx16

Sažetak: U ovom diplomskom radu je opisan princip kompjuterske emulacije sa primerom kompjuterskog programa za emuliranje rada računarskog hardvera arhitekture računara Milantex Tx16. Emulator je realizovan sa procesorom u vidu interpretera programskog koda i sa virtualnim periferijama za ulaz i izlaz, kao i sa virtualnim uređajima za trajno skladištenje podataka.

Ključne reči: Emulacija, programiranje, arhitektura računara

Milantex Tx16 computer architecture emulator

Abstract: This paper describes the principles of computer emulation with aid from a computer program which emulates the Milantex Tx16 computer architecture. The emulator includes a processor that interprets the program byte code and includes virtual peripheral input and output devices, as well as a virtual permanent data storage device.

Keywords: Emulation, programming, computer architecture

SADRŽAJ

1	UVOD	1
2	KOMPJUTERSKA EMULACIJA	2
2.1	RAZLIKA IZMEĐU EMULACIJE I SIMULACIJE	2
2.2	EMULATOR	3
2.3	KORISTI OD EMULACIJE	3
2.4	STRUKTURA EMULATORA	4
2.5	TEHNIKE EMULACIJE	9
3	EMULATOR MILANTEX TX16 RAČUNARA	12
3.1	BLOK ŠEMA RAČUNARA MILANTEX TX16	12
3.2	REALIZACIJA EMULATORA	13
3.3	KLASA PROCESORA (CPU KLASA)	13
3.4	SET INSTRUKCIJA PROCESORA MILANTEX TX16 RAČUNARA	20
3.5	INTERAPTI PROCESORA MILANTEX TX16 RAČUNARA	21
3.6	KLASA ZA PRIKAZ REGISTRA (REGISTERDISPLAY KLASA)	22
3.7	KLASA MEMORIJE (KLASA MEMORY)	25
3.8	KLASA ULAZA I IZLAZA (I/O KLASA)	30
3.9	KLASA DISPLEJA (DISPLAY KLASA)	33
3.10	KLASA DISKA (DISK KLASA)	38
3.11	KLASA ZA EDITOR TEKSTA SA PAMETNIM OBELEŽAVANJEM	39
3.12	KLASA ZA PREVOĐENJE IZVORNOG KODA (INSTRUCTIONCOMPILER KLASA)	44
3.13	GLAVNA KLASA PROGRAMA (KLASA MAIN)	48
4	OPIS RADA EMULATORA MILANTEX TX16 RAČUNARA	53
4.1	KORISNIČKI INTERFEJS EMULATORA	53
4.2	PISANJE IZVORNOG KODA PROGRAMA	55
4.3	PROGRAM KORIŠĆEN U DEMONSTRACIJI	55
4.4	IZVORNI KÔD PROGRAMA ZA DEMONSTRACIJU	56
4.5	PREVOĐENJE IZVORNOG KODA PROGRAMA	61
4.6	POKRETANJE EMULATORA	61
4.7	PRAĆENJE STATUSA EMULACIJE	62
4.8	DODAVANJE VIRTUALNOG DISKA I PRIKAZ NJEGOVOG SADRŽAJA	62
4.9	DODAVANJE DISPLEJA I UPRAVLJANJE NJEGOVIM IZGLEDOM	63
5	ZAKLJUČAK	64
6	LITERATURA	65

1 Uvod

U ovom diplomskom radu je opisan princip emuliranja računarskog hardvera određene arhitekture na računaru potpuno drugačije arhitekture, sa akcentom na izradu procesora emulatora po principu interpretera.

Pored teorijskog opisa principa i načina realizacije procesa emulacije hardvera, rad prati i praktična realizacija programa za emuliranje arhitekture računara Milantex Tx16. Program je u radu detaljno opisan i izvorni kod programa je priložen u celosti, dok su pojedini delovi posebno objašnjeni.

U radu je ukratko opisan i koncept izrade procesora u vidu simulatora, kao teorijsko objašnjenje metoda za implementaciju, dok je praktično procesor realizovan u vidu interpretera kao sastavni deo emulatora.

Opisana je i veza između procesora emulatora i virtualnih periferijskih ulaznih i izlaznih uređaja, kao i veza sa virtualnim uređajem za trajno skladištenje podataka koji su u programu realizovani kao moduli emulatora.

Rad je podeljen u tri dela prema tematici koja se obrađuje:

- Prvi deo predstavlja kratak pregled istorije kompjuterske emulacije, razliku između simulacije i emulacije u računarstvu, objašnjenje osnovnih pojmova koji se koriste u ostatku rada, kao i teorijski pregled metoda realizacije emulatora.
- Drugi deo predstavlja detaljnu razradu programa koji je sastavni deo diplomskog rada, strukturu arhitekture računara Milantex Tx16, analogno tome šemu modula i veza arhitekture u emuliranom okruženju, algoritme pod-rutina programa korišćenog za demonstraciju rada emulatora, kao i kompletan izvorni kod emulatora.
- Treći deo sadrži opis i demonstraciju rada emulatora sa prikazanim i opisanim *screenshot*-ovima programa u procesu emuliranja Milantex Tx16 računara dok izvršava program za demonstraciju rada emulatora opisanog i predstavljenog u drugom delu rada.

Računar Milantex Tx16 je osmišljen sa svim svojim delovima u potpunosti za potrebe izrade diplomskeg rada i kao takav nikada nije fizički sastavljen i korišćen. Set instrukcija procesora ovog računara je takođe osmišljen po modelu RISC (Reduced instruction set computing), odnosno procesor sa smanjenim skupom naredbi koje su u radu takođe detaljno opisane. Cilj osmišljanja ove arhitekture računara je bio da se u potpunosti spreči mogućnost dovođenja u pitanje originalnosti rada, jer je na Internetu u trenutku izrade ovog rada bio dostupan veliki broj primera emulatora postojećih arhitektura računara koji bi bili realizovani na sličan ili identičan način kao emulator u ovom radu. Prema tome, osmišljanjem kompletne arhitekture računara i seta instrukcija za procesor tog računara je sprečena moguća sumnja da je u pitanju plagijat.

2 KOMPJUTERSKA EMULACIJA

Reč „emulacija“ je nastala 1963. godine u IBM-u tokom razvoja IBM 360 serije računara. Programeri su razvijali softver kojim bi simulirali rad starijih tipova IBM računara u cilju obezbeđivanja kompatibilnosti programa pisanih za te platforme sa novom arhitekturom NPL serije.

2.1 RAZLIKA IZMEĐU EMULACIJE I SIMULACIJE

U poslednje vreme se reč emulacija koristi isključivo u kontekstu softvera, dok je ranije ta reč označavala samo jednu pomoćnu tehniku u optimizaciji rada simulatora. Kako emulatori nisu bili u mogućnosti da rade ništa više od onoga za šta su napravljeni i šta je programirani interpreter bio sposoban da obradi, emulacija se duže vreme smatrala samo za jedan privremeni korak u optimizaciji rada simulatora.

Pošto je simuliranje jednog kompletnog računarskog sklopa oduzimalo mnogo resursa novog računara, inženjeri su razvili program koji je interpretirao kod programa pisanih za stare modele računara i izvršavao semantički isti kod kako bi ostvario isti rezultat. Obzirom da ovaj način rada nije spadao u definiciju simulacije jednog sistema, već njegovog oponašanja, ali ne na identičan način kako on funkcioniše u fizičkom svetu, kao ni u logičkom smislu, inženjeri su osmislili termin emulacija za ovaj pristup izvršavanja koda koji nije napisan za ciljanu arhitekturu na kojoj se izvršava.

Za razliku od simulacije gde se kompjuterskim programima proračunavaju stanja i ponašanje modela ili sistema u okolnostima gde je ono nepredvidivo, nepoznato ili jednostavno nije moguće praktično realizovati, kompjuterska emulacija je postupak izvršavanja kompjuterskog programa na taj način da se ponašanje nekog sistema ili modela oponaša u virtualnom okruženju na isti način kako je poznato da se taj sistem ili model ponaša u realnom svetu, do određene tačnosti, sa ciljem da se to ponašanje imitira ili unapredi virtualnim poboljšanjima koja kasnije mogu, ukoliko se pokažu kao dobro rešenje, da se implementiraju u realan sistem ili model.

Najpreciznije rečeno, kompjuterska simulacija je postupak koji omogućava imitaciju ponašanja sistema u određenom vremenskom periodu korišćenjem kompjuterskih programa za proračun stanja sistema.

Kompjuterske simulacije se često koriste kao deo matematičkog modelovanja raznih delova prirodnih sistema u fizici, astrofizici, hemiji, biologiji, ekonomiji, psihologiji, sociologiji, inženjerstvu itd.

Postoje razni oblici kompjuterskih programa koji se koriste za simulacije. Ranije su to bili jednostavniji programi koji su serijski proračunavali stanja modela za određene ulazne parametre i koji su unapred definisani u vidu nizova ulaznih vrednosti ili prenešenih vrednosti iz perthodnih proračunatih stanja. Savremeniji kompjuterski programi za simuliranje obrađuju velike količine ulaznih podataka koji su često kompleksne vrednosti, kao i rezultati poračuna. Neke vrste kompjuterskih simulacija predstavljaju rezultate grafički u vidu animacije korišćenjem CGI (computer generated imagery) tehnologija.

Suština kompjuterske simulacije je da se ispita ponašanje nekog sistema ili modela u okolnostima i uslovima koji su najčešće malo verovatni ili skoro nemogući za praktičnu realizaciju sa tehnologijom ili resursima koji su na raspolaganju u datom momentu.

2.2 EMULATOR

Emulator je kompjuterski program koji se izvršava na jednoj arhitekturi računara sa ciljem da oponaša rad druge kako bi se izvršavao softver pisan za arhitekturu koja se emulira.

Emulacije se neretko realizuju sa ciljem da se očuva mogućnost korišćenja sistema ili modela u virtualnom okruženju kada više ne postoji ili je sve teže u realnom fizičkom okruženju korišćenje istog. Najčešći primer su stare igračke konzole koje se više ne proizvode ili više ne postoje ispravni primerci ili sistemi u mainframe-terminal vezama koji se više ne održavaju u praksi, ali su potrebni zbog nekih starih podataka i programa.

2.3 KORISTI OD EMULACIJE

Jedan od čestih motivacija za razvoj emulatora je želja za očuvanjem starih i prevaziđenih sistema i arhitektura koje se više ne koriste ili ne postoje ispravni primerci uređaja koji mogu da pokreću softver napisan za te arhitekture. Često se takvi emulatori izrađuju radi istorijskih istraživanja razvoja softvera ili kao projektni zadaci na fakultetima gde studenti u timovima rade na realizaciji emulatora, čime pokazuju poznavanje arhitekture računara, kao i programiranja na nivou potrebnom da se program kao što je emulator napiše.

Jedna od ključnih koristi emulacija je da softver koji se pokreće u emuliranom okruženju zadržava formu i izgled koji je imao na originalnom hardveru u vremenu kada je korišćen, tako da korisnicima pruža što bliži ugođaj onom koji su imali tokom korišćenja softvera u realnom okruženju.

1. Softver u emuliranom okruženju se izvršava približno istom brzinom, na isti način i sa istim hardverskim karakteristikama i tehničkim mogućnostima u virtualnom smislu kao što je slučaj u realnom sistemu.
2. Kada se softver razvija za platformu na kojoj nije moguće direktno pisati i izvršavati kod, već je potrebno kompajlirati softver preneti nekim kanalom na ciljani hardver ili uređaj, tada je lakše i brže softver razvijati na platformi kao što je PC gde je moguće tokom razvoja pokretati softver u emulatoru i u realnom vremenu uklanjati greške, pa tak kada je verzija softvera stabilna prebaciti je na ciljanu platformu.
3. Razvoj emulatora pod GNU licencom što obezbeđuje izradu boljih i bržih emulatora za određene arhitekture i sisteme lakšom kolaboracijom ljudi širom sveta.
4. Igranje igara programiranih specijalno za jednu hardversku platformu na drugima.

Razvoj softvera u emuliranom okruženju

Programi za hardverske platforme se često razvijaju na drugim operativnim sistemima za koje su napisana bolja razvojna okruženja. Najbolji primer je Eclipse IDE za Java programiranje sa dodatkom Android SDK-a za razvoj Android aplikacija za mobilne telefone i tablete. Nekada je u nedostatku uređaja određenih karakteristika, na primer tableta sa velikim ekranom, na kojem bi se testirala aplikacija neophodno imati emulator.

Razvoj softvera za hardver koji još uvek ne postoji

Neretko se softver razvija paralelno sa hardverom, tako da kada u jednom trenutku hardver bude spreman, kako se ne bi čeklo dugo na izradu adekvatnog pratećeg softvera, koristi se emulator hardvera izrađen po specifikacijama po kojima će raditi hardverska platforma.

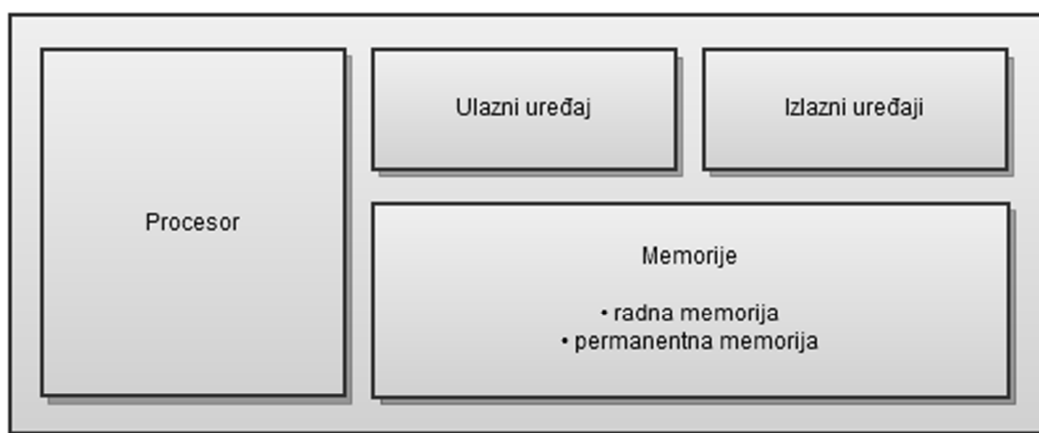
Ovo je jedan od najčešćih komercijalnih vidova emulacije, kada se zanemare emulatori za industriju video igrica koji obezbeđuju igranje igara pravljenih za jednu platformu na nekoj drugoj.

Emulatori se koriste i kod programiranja softvera za mikrokontrolere, gde programeri nekada razvijaju softver za uređaje koji još uvek nisu sastavljeni i spremni.

2.4 STRUKTURA EMULATORA

Emulator je najčešće podeljen u module koji okvirno odgovaraju strukturi sistema, tj. hardverske platforme koja se emulira. To su najčešće sledeći moduli:

- Procesor
- Memorija
- Ulazno-izlazni uređaji



Slika 1 - Blok šema jednostavnog emulatora

Magistrale i delovi arhitekture emulirane platforme koji služe za koordinaciju uređaja i komunikaciju između delova sistema se ne emuliraju, najčešće zbog brzine, ali i zbog toga što je jednostavnije razviti emulator kod kojeg virtualni moduli sistema direktno komuniciraju sa procesorom i međusobno, nego što bi bilo kada bi se programirali svi komunikacioni protokoli i veze koje postoje u realnom sistemu, a njihov nedostatak neće uticati na rad emulirane platforme i preciznost izvršavanja softvera koji se pokreće u emulatoru.

Modul za upravljanje memorijom

Postoji nekoliko pristupa za emuliranje rada systemske memorije. Najjednostavniji je da se memorija emuliranog sistema smesti u memoriju sistema na kojem se emulira u niz elemenata veličine identične kao što je veličina reči sistema koji se emulira.

Ovo je način koji je moguć samo kod veoma jednostavnih sistema koji nemaju napredno upravljanje memorijom pomoću eksternog modula zaduženog za upravljanje memorijom, već je procesor taj koji direktno upravlja i alocira memoriju. Često to nije slučaj, jer većina platformi ima neki vid naprednog upravljanja memorijom za koju je zadužen poseban čip, tj. deo sistema za čije se ponašanje u programiranju emulatora mora kompenzovati nekim vidom virtualnog raspoređivanja resursa što unosi složenost u dve osnovne funkcije za rad sa memorijom, a to su čitanje i pisanje vrednosti.

U situaciji kada se jedan zajednički opseg za pristup memorijskim lokacijama segmentira na više tipova memorija, kao što su ROM, RAM i proširena memorija, potrebno je programski rešiti logiku koja će na osnovu adrese efikasno i brzo odlučiti u kom nizu elemenata da iščita ili da upiše vrednost.

Bitno je da se proces preračunavanja tačne memorijske lokacije traženog elementa, kao i odluka o tome kojoj memoriji da se pristupi na osnovu segmenta, mora da bude napisana tako da radi izuzetno brzo, jer se funkcija čitanja iz memorije koristi mnogo češće nego funkcija pisanja u memoriju i ne sme da unosi nepotrebno kašnjenje između zahteva za iščitavanjem i dobijanja iščitane vrednosti, jer se onda ceo sistem usporava.

Iz ovog razloga, modul upravljanja memorijom treba pisati na što nižem nivou, možda čak i na nivou assembler-a ili C programskog jezika kako bi se obezbedila brza i efikasna funkcionalnost modula.

Modul procesora

Modul procesora je jedna od najznačajnijih elemenata emulatora. Često su svi ostali moduli takvi da mogu da se koriste za druge emulatorne raznih arhitektura bez mnogo promena, ali modul procesora je jedan koji je specifičan za određenu arhitekturu zbog toga što skoro svaka ima jedinstvene setove instrukcija, načine adresiranja, dužinu instrukcija, registre i njihove adrese, pristupe upravljanju stekovima itd.

Obzirom da svaki procesor ima setove instrukcija koji su jasno definisani i obavljaju tačno utvrđene poslove i transformacije, izrada modula procesora nije komplikovana za programiranje, osim kada je u pitanju arhitektura za koju nije dostupna dokumentacija.

Tada je potrebno obaviti prvo obrnuti inžinejring seta instrukcija, načina adresiranja, dužine instrukcija i varijanti svake instrukcije, uticaja izvršavanja instrukcija na flegove i registre, pa tek onda programirati modul procesora.

Često je moguće u emulator ugraditi simulator procesora umesto pisanja modula koji vrši emuliranje izvršavanja instrukcija. Ovaj pristup je dobar onda kada postoji napisan dobar i brz simulator, pa nije logično pisati iznova ceo modul kada nije sigurno da će performanse tog modula biti bolje od onih koje obezbeđuju već razvijeni simulatori.

Kada ne postoji simulator procesora za određenu arhitekturu računara, tada se pristupa razvoju modula.

Elementi modula procesora

Procesor emulatora ima skoro sve logičke komponente kao i fizički procesor u nekom računarskom sklopu, kao što su registri, stekovi, registar flagova, registri stanja, aritmetička, logička i kontrolna jedinica, programska memorija u koju je upisan program koji procesor izvršava po pokretanju.

Jedan element koju procesor emulatora ima, a koju fizički procesor nema kao jedinstvenu celinu, već kao elemente koji su protkani kroz celu njegovu strukturu, je deo koji je zadužen za tumačenje naredbi i njihovo izvršavanje.

Ovaj deo procesora emulatora može da bude realizovan na mnogo načina. U nekim slučajevima on u sebi uključuje aritmetičku, logičku i kontrolnu jedinicu, a u nekim slučajevima samo upravlja njima. Logički gledano, to je centralna jedinica procesora.

Na neki način, ovaj deo bi mogao da se shvati kao procesor procesora, jer je njegova svrha da koordinira funkcionisanjem procesora u emulatoru. Iako tako nešto ne postoji u konstruisanim fizičkim procesorima, u realizaciji emulatora ovo je veoma važan deo.

Najjednostavnije rečeno, ovaj deo je zadužen da obezbedi da postupak dobijanja rezultata tokom izvršavanja instrukcija u emuliranom procesoru bude semantički ekvivalentan postupku dobijanja rezultata tokom izvršavanja instrukcija u fizičkom procesoru i da sva stanja koja bi nastala u registrima, stekovima, statusnim registrima i registrima flagovima, nastanu i u procesoru emulatora. Način realizacije ovog dela je određen isključivo načinom realizacije modula procesora, tj. od principa na koji će procesor izvršavati instrukcije iz programske memorije.

Modul procesora kao interpreter

Prvi i najjednostavniji pristup kod izrade modula procesora je interpreterski. U ovom pristupu program iščitava binarni kôd programa uzimajući instrukciju po instrukciju iz radne memorije sa adrese u pokazivaču instrukcija i izvršava kôd koji je semantički identičan onom koji emulirani procesor obavlja u realnom sistemu. Na ovaj način je rezultat instrukcije ispravan, a odmah po izvršavanju se prelazi na sledeću instrukciju i tako u krug.

Kod ovog pristupa se za svaki registar procesora dodeljuje jedna varijabla ili element niza u kojima se smeštaju vrednosti registara, a isti princip se primenjuje za stekove, bafere, flagove i ostale oblike interne memorije procesora. Interpreterski model oponaša logiku rada instrukcije, a ne tačan način na koji bi instrukcija bila izvršena. Kako je akcenat najviše na tome da rezultat ulaznih podataka dâ izlaz koji je očekivan, kako u emuliranom okruženju, tako i u realnom.

Interpreterski modeli emuliranja rada procesora su jednostavni i dovoljno efikasni da se koriste za emuliranje rada računara starih oko deset godina, a na modernim računarima, dok za emuliranje računara čija je brzina reda veličine računara na kojem se emulacija izvršava, ovaj pristup često nije praktičan, pa se tada koriste drugi pristupi emulaciji procesora.

Tehnika binarnog prevođenja u modulu procesora

Kada interpreterski model za izradu modula procesora nije moguć, alternativni pristup je često tehnika binarnog prevođenja, koja podrazumeva da se instrukcije ne izvršavaju semantički jednakim kodom, već da se koriste setovi instrukcija procesora računara na kojem se izvršava emulacija za izvršavanje instrukcija procesora koji se emulira.

Ovaj pristup nije jednostavan, jer često setovi instrukcija ove dve arhitekture nisu isti ili se u mnogo toga razlikuju. Na primer, ARM procesori čiji su setovi instrukcija realizovani po RISC modelu i imaju sasvim drugačiji princip adresiranja, rada sa konstantama, dužinu pokazivača na adresne lokacije ili imaju veći ili manji broj registara od procesora na kojem se ARM procesor emulira.

U tom slučaju nije dovoljno samo transformisati redosled argumenata u instrukciji ili proširiti dužinu binarnog zapisa konstante, adrese ili izmeniti adresu registara, već je nekada potrebno jednu instrukciju razložiti na dve ili više kako bi procesor na kojem se emulira ARM mogao to da obradi, a nekada čak da se više uzastopnih instrukcija sastave u jednu složenu, ukoliko procesor računara na kojem se ARM procesor emulira tako nešto omogućava, a sa ciljem postizanja boljih performansi i nadoknade za usporavanja dobijena razlaganjem nekih instrukcija.

Sve ovo čini izradu efikasnog modula procesora znatno težim u situacijama kada su brzine emulirane platforme i platforme na kojoj se emulira istih reda veličina.

Tehnika dinamičkog prevođenja u modulu procesora

Kada je akcenat na postizanju veće brzine emuliranja, tada tehnika binarnog prevođenja mora da bude poboljšana tako što se neće prevoditi jedna ili nekoliko uzastopnih instrukcija programa za emulirani procesor, već celi blokovi instrukcija.

Tipičan školski primer ovog slučaja je dat u sledećem prikazu koda:

cpystr:

```

mov A,[adresa 1. stringa] ; U reg. A smesti adresu (pokazivač) na prvi string
mov B,[adresa 2. stringa] ; U reg. B smesti adresu (pokazivač) na drugi string
loop:
mov C,[A]                  ; Kopiraj bajt sa adrese koja je u reg. A u reg. C
mov [B],C                  ; Kopiraj bajt koji je u reg. C na adresu koja je u reg. B
inc A                      ; Povečaj vresnot u reg. A (pomeri pokazivač na sledeće slovo)
inc B                      ; Povečaj vresnot u reg. B (pomeri pokazivač na sledeće slovo)
cmp C,#0                   ; Uporedi bajt u reg. C sa 0 (služi za markiranje kraja stringa)
jnz loop                   ; Ako vrednost bajta nije 0, nastavi dalje sa izvršavanjem petlje
end:                       ; Ako je vrednost bila 0, nastaviće dalje sa izvršavanjem programa

```

Ovaj podprogram **strcpy** služi za kopiranje karaktera jednog stringa u drugi string i to radi tako što svaki bajt prvog stringa kopira u registar C, a zatim iz registra C na sledeće mesto u drugom stringu.

Razlog zbog kojeg ovaj program prvo mora bajt da smesti u registar C, pa da ga onda prebaci na drugu memorijsku adresu je to što procesor za koji je ovaj program pisan ne podržava direktno prenošenje vrednosti sa jedne memorijske adrese na drugu.

Ako predpostavimo da proceor na kojem ćemo pokretati emulator ima instrukciju za kopiranje vrednosti koja podržava da joj oba argumenta budu adrese memorijskih lokacija smeštene u registre, tada prevedeni blok ovog podprograma za izvršavanje na toj platformi može da izgleda ovako:

cpystr:

```

mov A,[adresa 1. stringa] ; U reg. A smesti adresu (pokazivač) na prvi string
mov B,[adresa 2. stringa] ; U reg. B smesti adresu (pokazivač) na drugi string
loop:
movs [B],[A]               ; Kopiraj vrednost sa adrese u reg. B na adresu u reg. A i
                           ; zatim povečaj vrednosti u A i B za jedan
jnz loop                   ; Ako Zero flag nije setovan tokom kopiranja, znači da nismo
                           ; stigli do vrednosti gde je karakter 0, pa ponovi petlju.
end:                       ; Ako je Zero flagsetovan, nastaviće dalje izvršavanje programa

```

U ovom slučaju vidimo da je sa šest instrukcija u petlji program sveden na dve instrukcije u petlji, korišćenjem instrukcije **movs** koja ne postoji u setu instrukcija procesora koji se emulira, ali je distupna emulatoru na računaru na kojem se emulator izvršava. Ova instrukcija je specifična, jer pored toga što može direktno da kopira vrednosti sa jedne lokacije u memoriji na drugu, ujedno povećava adrese, tj. pokazivače na memorijske lokacije za jedan, tako da sledeći put kada se izvrši, nastavi da kopira sa prve sledeće lokacije.

Takođe, procesor na kojem se izvršava emulator, za razliku od ovog koji se emulira, kod koga smo morali da proveravamo vrednost u registru kako bismo odlučili da li da nastavimo dalje ili da skočimo nazad na petlju, ima tu osobinu da kada su registri sa kojima je poslednja instrukcija radila imaju vrednost nula, setuje automatski Zero flag, tako da nakon kopiranja odmah možemo, bez potrebe za instrukcijom za poređenje da odlučimo da li nastavljamo izvršavanje programa ili se vraćamo na izvršavanje instrukcija u petlji.

Iako je ovaj pristup efikasniji, jer koristi direktno procesor računara na kojem se emulacija izvršava na najnižem mogućem nivou, on je znatno teži za realizaciju, ali ima i jednu posebno problematičnu manu, a to je što je emulator pisan na ovom principu često neupotrebljiv na bilo kojem drugom računaru sa drugim procesorom, tako da ako, na primer, imamo emulator napisan za Intel procesore, on možda neće moći da radi na računaru sa AMD procesorom.

Ulazno-izlazni uređaji

Obzirom da emulatori skoro nikada ne emuliraju rad magistrala i kanala komunikacije i kontrole između komponenata u računaru, nije uvek lako razviti standardizovani model za programiranje ulaznih i izlaznih uređaja.

Zbog toga se kod programiranja emulatora često ulazne i izlazne jedinice programiraju iznova uz sam emulator i prilikom kompajliranja koda programa za emulaciju oni postaju sastavni deo emulatora, tj. ne programiraju se u vidu dodataka za emulator, kao što bi to mogao da bude slučaj da je u pitanju simulacija. Najčešće se ti ulazni i izlazni uređaji pišu kao komponente celog emulatora.

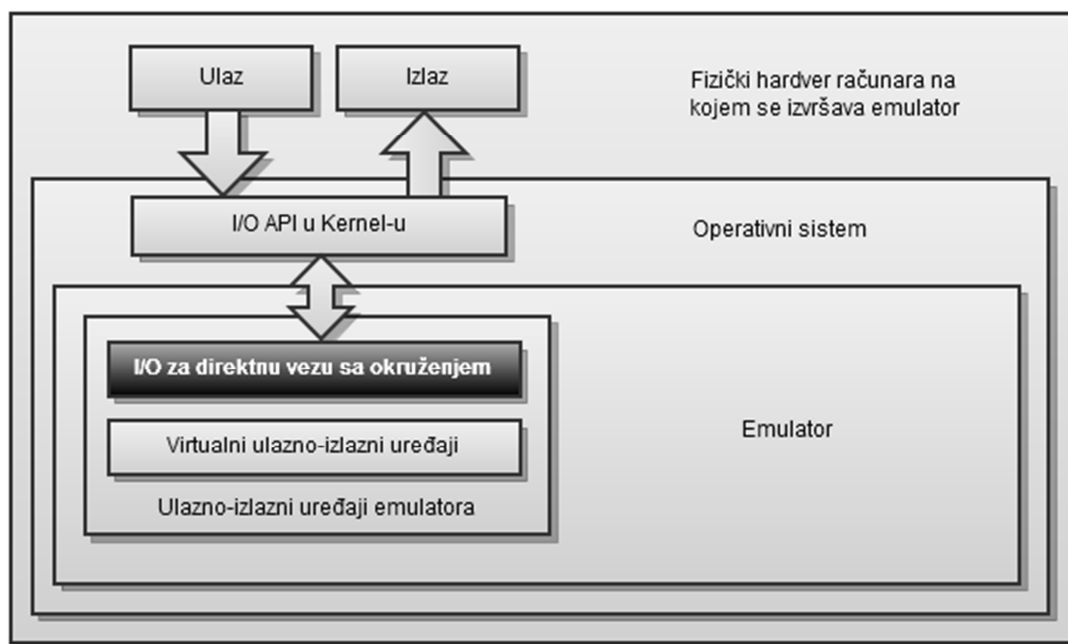
Kako nije uvek potrebno oponašati baš svaki najmanji deo računara, neki ulazni ili izlazni uređaji mogu da budu izostavljeni iz emulatora. Prema tome, prave se samo najosnovniji uređaji, kao što su na primer:

- **Tastatura** (sa standardnim rasporedom tastera ili se nekada razvije nekoliko varijanti tastature)
- **Miš** (najčešće su pokriveni svi vidovi ulaza dobijenih sa miša, kao što su skrolovi, pomoćni dugmići itd.)
- **Hard diskovi** (obično se programira podrška za PATA i SATA, a nekada i za SCSI kontrolere)
- **Flopy disk** (pošto su jednostavni za programiranje, većina emulatora ih podržava)
- **Optički mediji** (komunikacioni protokoli su slični za CD, DVD i BlueRay diskova, pa podrška ne izostaje)
- **Monitor** (često se preskaču grafičke kartice, a njihov rad se delegira modulu za prikaz slike)
- I drugi...

Komunikacija sa spoljašnjim svetom

Često ulazne i izlazne periferije nisu dovoljne za komunikaciju sa okruženjem, pa je prema tome potrebno obezbediti vezu između programa u emulatoru i spoljašnosti emulatora.

Obzirom da se emulator pokreće kao program na drugom računaru, potrebno je napraviti još jedan skok između tih čvorišta i realnog sveta i komunikacionih portova u spoljašnjem okruženju.



Slika 2 - Veze između emulatora i spoljašnjeg sveta putem virtualnih I/O uređaja za direktnu vezu sa hardverom

Neki od modula koji se razvijaju za ove potrebe su:

- Virtualni komunikacioni portovi (UART, COM i LPT dobijaju i podršku za otvaranje stream-ova ka drugim aplikacijama ili ka mreži za upravljanje i slanje iz daljine)
- Direktna veza sa periferijama računara na kojem se emulira (najčešće zbog podrške za USB, kako bi se omogućilo korišćenje USB uređaja)
- Mrežna kartica (princip rada je sličan kao kod virtualnih mašina)

2.5 TEHNIKE EMULACIJE

Emulacija arhitekture se koristi kada je potrebno program pisan u višem programskom jeziku pokretati na više različitih arhitektura, a čiji izvorni kod na nivou mašinskog koda ne postoji, nije isplativo menjati nakon automatskog prevođenja ili uopšte nije moguće menjati.

U takvom slučaju je lakše oponašati računarski sklop ciljane arhitekture i u okviru takvog virtualnog računara pokretati program. Postoji mnogo teorijskih pristupa za emulaciju fizičkog računara. Neki podrazumevaju delimičnu emulaciju, uz pomoć simuliranih modula, dok neki pristupi predlažu kompletno semantičko prevođenje sa ciljem uspešnog izvršavanja na ciljanoj arhitekturi bez potrebe za emulatorom.

U praksi se ne koriste pristupi kao što je potpuna simulacija sistema, jer je previše zahtevna po pitanju resursa i performansi računara na kojem se simulacija vrši, tako da je na kraju rezultujuća brzina emuliranog programa značajno redukovana u odnosu na realne mogućnosti.

Najčešće korišćena tehnika gde je potrebno napraviti brz emulator je ona gde se procesor simulira, dok se ostali delovi sistema emuliraju, tj. semantički prevode.

Takođe, gde nije neophodno obezbediti brzinu emuliranog računara reda veličine računara na kojem se emulacija vrši, koristi se pristup potpune emulacije.

Taj pristup je jednostavniji od simulacije, jer je potrebno semantički prevesti samo određene delove sistema, među kojima nisu magistrale, kontroleri, fizičko ponašanje registara i stekova, već samo postizanje rezultata koji su ekvivalentni onima koje postiže fizički računar.

Tehnike emulacije procesora

Kada je u pitanju modul procesora koji je centralni deo svakog emulatora, u praksi se koristi nekoliko tehnika emulacije procesora.

Jedna od starijih naprednih tehnika emulacije je binarno prevođenje, odnosno morfining ili pretapanje koda za jedan set instrukcija u semantički ekvivalentan kod za drugi set instrukcija i arhitekturu procesora.

Savremeniji pristup programiranju emulatora je takav da se emulacija vrši mnogo transparentnije, odnosno da se registri, stekovi, statusni registar sa flagovima i ostali elementi procesora arhitekture koja se emulira ne čuvaju kao promenljive, nizovi elemenata i blokovi alocirane memorije u okviru radne memorije računara na kojem se emulacija vrši, već da se iskoriste fizički registri, stek, baferi, flagovi i drugi elementi procesora na računaru na kojem se emulacija vrši za potrebe bržeg i efikasnijeg iskorišćenja resursa za potrebe emulacije.

Ovakav pristup je blizak konceptu virtualizacije, koji je i nastao iz ideje da je moguće emulirati procesor, ali ne na taj način da su resorsi potpuno posvećeni jednom procesu, već da se procesi konkurentno izvršavaju i na taj način veći broj emulatora procesora pokreću istovremeno, da dele resurse i da je moguće izvršavanje, ne više programa, već više virtualnih okruženja sa sopstvenim operativnim sistemima i mnoštvom programa na jednoj fizičkoj mašini.

Direktnim poređenjem performansi ove dve osnovne tehnike, gde se jedna oslanja na simulaciju fizičkog rada procesora, a druga na emulaciju interpretiranjem koda programa koji se u procesoru izvršava, dolazi se do zaključka da je simulacija znatno skuplji pristup po pitanju resursa i brzine i da često programi u simuliranom okruženju rade između 10 i 1000 puta sporije nego na fizičkoj mašini.

Sa druge strane, emulacijom nekada može da se postigne brzina koja je približna brzini fizičke mašine koja se emulira, a u nekim slučajevima i da postigne bolje performanse, kada se vrši prevođenje mašinskog koda programa i kada procesor arhitekture na kojoj se emulacija vrši podržava instrukcije sa više argumenata koje u jednom izvršavanju mogu da obave mnogo više posla nego instrukcije procesora arhitekture koja se emulira.

Kod tehnike emuliranja modula procesora nije potrebno oponašati sve delove procesora, već samo one koje program može da vidi, tj. čiji status može da očita ili koji može da promeni. Ostale fizičke komponente koje u procesoru obavljaju funkciju kontrole, veze, komunikacije, kontrole napajanja elemenata, slojeva, kao i senzore temperature i slično, nije potrebno emulirati.

Komponente kao što su registri, stekovi, flagovi, ali i upravljači interpreterima, tajmeri i slične komponente nije potrebno, kao kod simulacije, oponašati na identičan način kako se u fizičkom računaru ponašaju, već ih treba posmatrati kao crne kutije koje imaju ulaze i izlaze, kao i neka prenešena i nasleđena stanja i programirati ih tako da na osnovu ulaznih stanja daju određeni rezultat koji je identičan onom koji bi dala ista komponenta u fizičkom računaru u datim okolnostima i sa datim vrednostima. Način na koji je programski rešeno ponašanje komponente, kao i program koji vrši obradu podataka u cilju obezbeđivanja rezultata treba samo semantički da odgovara algoritmu fizičkog ponašanja komponente.

Prednosti i mane u realizaciji modula procesora

Kako sam naveo, u praksi postoji više pristupa realizacije modula procesora. Ovde se fokusiram na četiri osnovna pristupa i to na simulaciju procesora, emulaciju procesora interpretiranjem, emulaciju korišćenjem postojeće arhitekture (transparentno emuliranje) i emulaciju procesora binarnim previđenjem koda programa.

Određene prednosti i mane pomenutih pristupa su očigledne i bez potrebe za posebnim istraživanjima, merenjima i testiranjima, ali su ona bez obira na to često vršena u cilju ispitivanja, a neki od rezultata do kojih se došlo istraživanjima je uporedni prikaz prednosti i mana pomenutih pristupa koji je prikazan u tabeli ispod.

Tehnika	Implementacija	Gubitak performansi	Ciljani set instrukcija
Simulacija	Jednostavna	Veoma visok	Bilo koji
Interpretacija	Jednostavna	Visok	Bilo koji
Transparentno emu.	Teška	Mali	Na kojem se izvršava
Binarno prevođenje	Veoma teška	Jako mali	Bilo koji

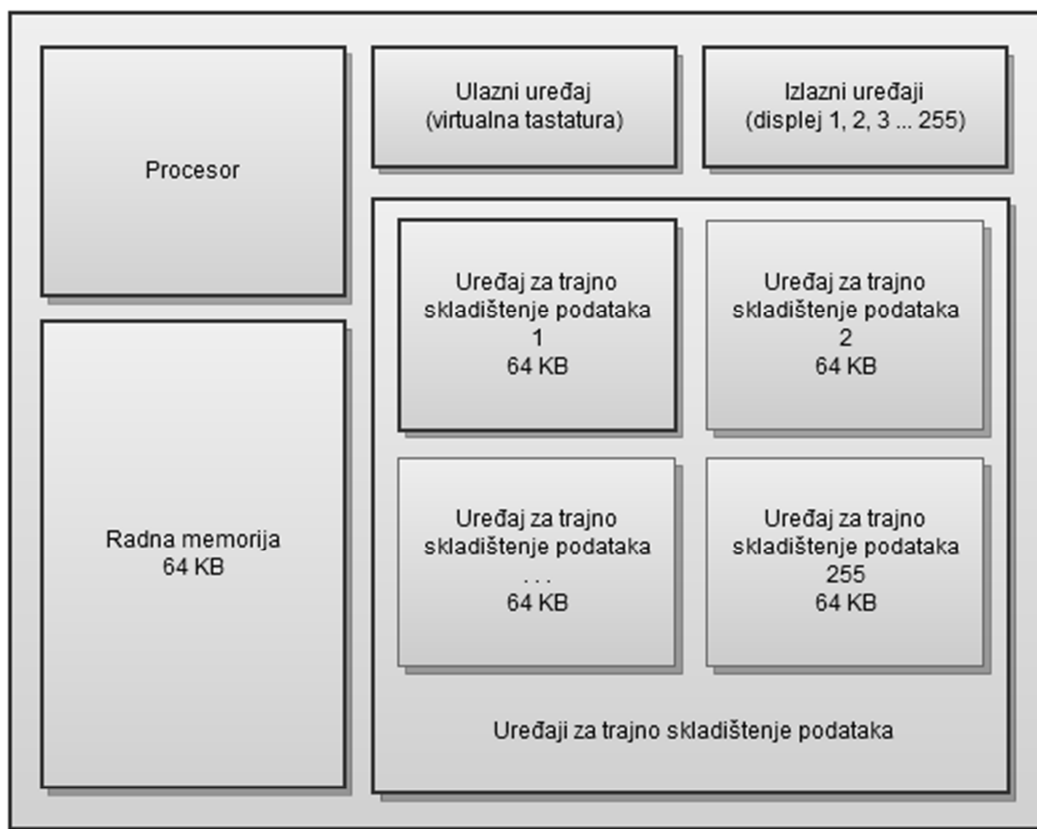
3 EMULATOR MILANTEX Tx16 RAČUNARA

Emulator računara Milantex Tx16 čiji je celokupan izvorni kod priložen u ovom delu rada je razvijen za potrebe praktične demonstracije koncepta emulacije arhitekture računara Milantex Tx16.

Računar Milantex Tx16 nije nikada konstruisan i korišćen, već je u potpunosti osmišljen sa svim svojim modulima za potrebe ovog rada. Procesor računara za potrebe demonstracije je osmišljen tako da ima set instrukcija po ugledu na neke starije RISC (Reduced instruction set computing) procesore, odnosno procesore sa smanjenim skupom naredbi. Sve naredbe iz osmišljenog seta instrukcija su detaljno opisane u daljem tekstu.

3.1 BLOK ŠEMA RAČUNARA MILANTEX Tx16

Kako računar Milantex Tx16 nikada nije konstruisan i ne postoji detaljna dokumentacija o njegovim sklopovima, vezama, kao ni detaljna električna šema kola koje ga čine, sastavljena je jednostavna blok šema sa elementima koji predstavljaju virtualne module, koji takođe nisu detaljno osmišljeni, kako bi bilo moguće predstaviti električne šeme i slično, pošto realizacija nije u vidu simulatora, gde bi ovaj vid tehničke razrade bio neophodan, već emulacija, gde je dovoljno realizovati logičke jedinice koje izvršavaju ekvivalentne poslove i daju rezultate kakve bi realan sistem ili isti takav simuliran sistem dali.



Slika 3 - Blok šema računara Milantex Tx16

3.2 REALIZACIJA EMULATORA

Emulator je pisan u programskom jeziku C#, a u integrisanom razvojnom okruženju (IDE) Visual Studio 2010 Express Edition kompanije Microsoft.

Svi blokovi u šemi računara su u emulatoru realizovani kao klase i u daljem radu će biti opisane pojedinačne klase koje čine emulator, sa izvornim kodom svake klase pojedinačno.

3.3 KLASA PROCESORA (CPU KLASA)

Klasa CPU je apstraktna statička klasa koja predstavlja centralni modul emulatora. Metodi CPU klase upravljaju stanjima ostalih modula, kako statičkih i apstraktnih, tako i instanciranih objekata određenih klasa, ako što su memorije i diskovi.

Klasa CPU sadrži sedam metoda i to:

- void ExecuteNextInstruction()
- void Jump(ushort instruction_number)
- ushort ShiftLeft(ushort value, int count)
- ushort ShiftRight(ushort value, int count)
- void Interrupt(byte code)
- string u_binarni_string(int n, int size)
- UInt32 MilantexDateTime()

Svaki od navedenih metoda je statičan, ali su samo dva metoda javna i to:

- void ExecuteNextInstruction()
- UInt32 MilantexDateTime()

Prvi javni metod je najsloženiji i obavlja centralnu funkciju ovog modula, a to je da iz radne memorije preuzima instrukciju sa sledeće pozicije na koju pokazuje pokazivač instrukcija i da na osnovu koda naredbe koja se nalazi na toj memorijskoj lokaciji odluči koji set naredbi da izvrši, tj. koji slučaj (case) u okviru grananja (switch).

Izvršavanje ovog metoda utiče na stanje flagova, registara i radne memorije, a kada je u pitanju izvršavanje interapta, na kraju izvršavanja ovog metoda moguće su promene i na spoljašnjoj memoriji, kao i na periferijama, tj. na displejima u slučaju ovog emulatora.

Drugi javni metod generiše četiri bajta koje procesor smešta u unapred utvrđene registre koji zajedno predstavljaju razložen podatak o trenutnom datumu i vremenu u formatu godina-mesec-dan sat-minut-sekund.

Ovaj podatak se u programu koristi za dobijanje datuma i vremena emuliranog BIOS-a, koji kao poseban modul nije napravljen, jer za time ne postoji realna potreba, već se koristi grupa metoda raznih klasa, tj. modula emulatora, razvijenih da nadoknade nepostojanje centralizovane jedinice za testiranje konfiguracije hardver, obzirom da je i takav proces nepotreban u slučaju emulacije.

Ostali metodi klase su privatni i mogu se izvršavati samo po pozivu iz ExecuteNextInstruction() metoda, a funkcije koje obavljaju u procesu emulacije su:

Metod Jump je jednostavan metod koji samo postavlja novu vrednost u pokazivač memorijske lokacije sledeće instrukcije u radnoj memoriji. Nova vrednost je prosleđena kao jedini argument koji ovaj metod uzima.

Metodi ShiftLeft i ShiftRight emuliraju operaciju pomeranja bitova unutar registra, tj. vrednost memorijske lokacije na levo, odnosno na desno, respektivno, a za određeni broj mesta, a da pritom zadržavaju, odnosno rotiraju stanje bitova, tako da preneti bitovi nisu izgubljeni u postupku pomeranja.

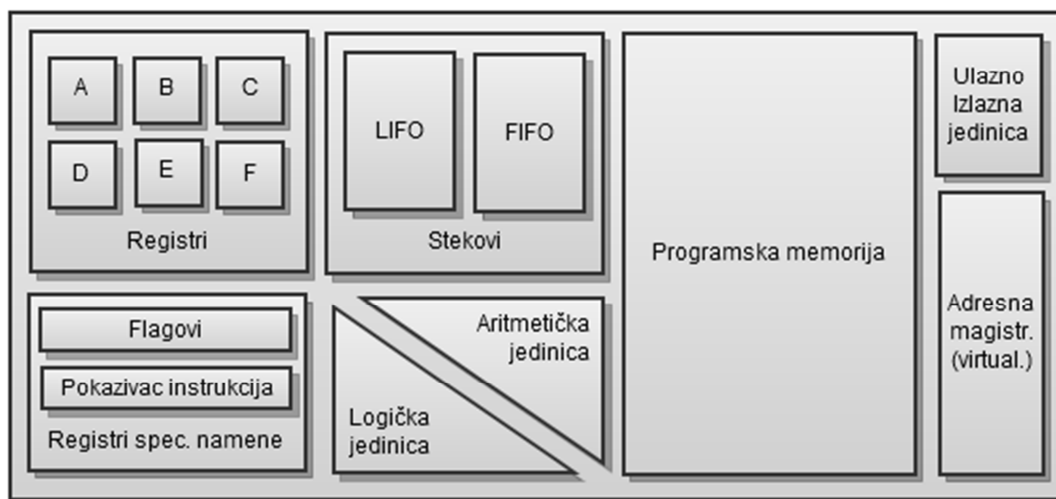
Metod Interrupt je privatni metod kojim ima posebnu ulogu u emulatoru. On sam predstavlja zamenu za hardverski upravljač interaptima koji bi bio prisutan u slučaju simulatora, a postoji kao sastavni deo realnog uređaja, tj. procesora u ovom slučaju.

Njegova osnovna uloga je da na osnovu prosleđenog argumenta, odnosno kôda interapta izvrši odabrani set naredbi koje semantički odgovaraju operacijama interapta u realnom sistemu.

Poslednji metod ove klase je *u_binarni_string()* koji uzima dva argumenta, celobrojnu vrednost koja se transformiše u string i dužinu stringa, tj. broj karaktera u njemu. Ovaj metod se koristi primarno u okviru javnog metoda MilantexDateTime.

Pošto je najbitniji deo emulatora računara Milantex Tx16 je njegov procesor, koji je, isto kao i računar osmišljen za potrebe demonstracije i za njega kao takvog ne postoji detaljna električna šema niti posebna šema veza i blokova, već isključivo malo detaljniji šematski prikazan spisak elemenata koji čine ovaj procesor u logičkom smislu, a koji u određenoj meri preslikava kakva bi bila fizička struktura procesora da je ikada konstruisan.

Sledeća slika predstavlja blok šemu procesora Milantex Tx16 računara:



Slika 4 - Blok šema procesora Milantex Tx16 računara

Izvorni kod klase CPU

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5
6 namespace Milantex_Tx16_Emulator {
7     public abstract class CPU {
8         public static int sleep_time = 0;
9
10        public static void ExecuteNextInstruction() {
11            byte Instruction = Memory.ReadNextByteFromProgramCode();
12            byte reg, reg1, reg2, literal, byte_A, byte_B;
13            byte[] bytes;
14            ushort value_A, value_B;
15            int value_I;
16            double value_D;
17
18            switch ( Instruction ) {
19                case 0x00:
20
21                    break;
22                case 0x01:
23                    reg = Memory.ReadNextByteFromProgramCode();
24                    byte_A = Memory.ReadNextByteFromProgramCode();
25                    byte_B = Memory.ReadNextByteFromProgramCode();
26                    bytes = new byte[] { byte_B, byte_A };
27                    value_B = BitConverter.ToInt16( bytes, 0 );
28                    Memory.SetRegisterValue( reg, value_B );
29                    break;
30                case 0x02:
31                    reg = Memory.ReadNextByteFromProgramCode();
32                    value_A = Memory.GetRegisterValue( reg );
33                    if ( ( ( int ) value_A ) + 1 > ushort.MaxValue ) {
34                        value_A = 0;
35                        Memory.SetFlag( 5 );
36                    }
37                    else value_A++;
38                    Memory.SetRegisterValue( reg, value_A );
39                    break;
40                case 0x03:
41                    reg = Memory.ReadNextByteFromProgramCode();
42                    value_A = Memory.GetRegisterValue( reg );
43                    if ( ( ( int ) value_A ) - 1 < 0 ) {
44                        value_A = ushort.MaxValue;
45                        Memory.SetFlag( 3 );
46                    }
47                    else value_A--;
48                    Memory.SetRegisterValue( reg, value_A );
49                    break;
50                case 0x04:
51                    reg1 = Memory.ReadNextByteFromProgramCode();
52                    reg2 = Memory.ReadNextByteFromProgramCode();
53                    Memory.SetRegisterValue( reg1, Memory.GetRegisterValue( reg2 ) );
54                    break;
55                case 0x05:
56                    reg1 = Memory.ReadNextByteFromProgramCode();
57                    reg2 = Memory.ReadNextByteFromProgramCode();
58                    Memory.WriteToMemory( Memory.GetRegisterValue(reg1), Memory.GetRegisterValue(reg2));
59                    break;
60                case 0x06:
61                    reg1 = Memory.ReadNextByteFromProgramCode();
62                    reg2 = Memory.ReadNextByteFromProgramCode();
63                    Memory.SetRegisterValue(reg1, Memory.ReadFromMemory( Memory.GetRegisterValue(reg2)));
64                    break;
65                case 0x07:
66                    Interrupt( Memory.ReadNextByteFromProgramCode() );
67                    break;
68                case 0x08:
69                case 0x09:
70                case 0x0A:
71                case 0x0B:
72                    reg = Memory.ReadNextByteFromProgramCode();
73                    value_A = Memory.GetRegisterValue( reg );
74                    literal = Memory.ReadNextByteFromProgramCode();
75                    if (Instruction == 0x08) Memory.SetRegisterValue( reg, (ushort)(value_A << literal));

```

```

76         else if (Instruction==0x09)Memory.SetRegisterValue(reg,(ushort)(value_A >> literal));
77         else if (Instruction==0x0A)Memory.SetRegisterValue(reg,ShiftLeft(value_A, literal));
78         else if (Instruction==0x0B)Memory.SetRegisterValue(reg,ShiftRight(value_A, literal));
79         break;
80     case 0x0C:
81         byte_A = Memory.ReadNextByteFromProgramCode();
82         byte_B = Memory.ReadNextByteFromProgramCode();
83         Jump( BitConverter.ToInt16( new byte[] { byte_B, byte_A }, 0 ) );
84         break;
85     case 0x0D:
86         byte_A = Memory.ReadNextByteFromProgramCode();
87         byte_B = Memory.ReadNextByteFromProgramCode();
88         if ( Memory.LoadedFlag ) Jump( BitConverter.ToInt16(new byte[]{byte_B,byte_A}, 0));
89         break;
90     case 0x0E:
91         byte_A = Memory.ReadNextByteFromProgramCode();
92         byte_B = Memory.ReadNextByteFromProgramCode();
93         if ( !Memory.LoadedFlag ) Jump( BitConverter.ToInt16(new byte[]{byte_B,byte_A}, 0));
94         break;
95     case 0x0F:
96         byte_A = Memory.ReadNextByteFromProgramCode();
97         if ( byte_A > 15 ) byte_A = 15;
98         Memory.LoadFlagForTesting( byte_A );
99         break;
100    case 0x10:
101        Memory.ClearFlags();
102        break;
103    case 0x11:
104        Memory.ClearLoadedFlag();
105        break;
106    case 0x12:
107        reg1 = Memory.ReadNextByteFromProgramCode();
108        reg2 = Memory.ReadNextByteFromProgramCode();
109        value_A = Memory.GetRegisterValue( reg1 );
110        value_B = Memory.GetRegisterValue( reg2 );
111        if ( value_A < value_B ) Memory.SetFlag( 2 );
112        else if ( value_A > value_B ) Memory.SetFlag( 1 );
113        else if ( value_A == value_B ) Memory.SetFlag( 0 );
114        break;
115    case 0x13:
116        reg1 = Memory.ReadNextByteFromProgramCode();
117        reg2 = Memory.ReadNextByteFromProgramCode();
118        value_A = Memory.GetRegisterValue( reg1 );
119        value_B = Memory.GetRegisterValue( reg2 );
120        Memory.SetRegisterValue( reg1, ( ushort ) ( value_A & value_B ) );
121        break;
122    case 0x14:
123        reg1 = Memory.ReadNextByteFromProgramCode();
124        reg2 = Memory.ReadNextByteFromProgramCode();
125        value_A = Memory.GetRegisterValue( reg1 );
126        value_B = Memory.GetRegisterValue( reg2 );
127        Memory.SetRegisterValue( reg1, ( ushort ) ( value_A | value_B ) );
128        break;
129    case 0x15:
130        reg1 = Memory.ReadNextByteFromProgramCode();
131        reg2 = Memory.ReadNextByteFromProgramCode();
132        value_A = Memory.GetRegisterValue( reg1 );
133        value_B = Memory.GetRegisterValue( reg2 );
134        Memory.SetRegisterValue( reg1, ( ushort ) ( value_A ^ value_B ) );
135        break;
136    case 0x16:
137        reg = Memory.ReadNextByteFromProgramCode();
138        value_A = Memory.GetRegisterValue( reg );
139        Memory.SetRegisterValue( reg, ( ushort ) ~value_A );
140        break;
141    case 0x17:
142        reg1 = Memory.ReadNextByteFromProgramCode();
143        reg2 = Memory.ReadNextByteFromProgramCode();
144        value_A = Memory.GetRegisterValue( reg1 );
145        value_B = Memory.GetRegisterValue( reg2 );
146        value_I = value_A + value_B;
147        if ( value_I > ushort.MaxValue ) {
148            value_A = ( ushort ) ( ( value_I - ushort.MaxValue ) - 1 );
149            Memory.SetFlag( 5 );
150        } else value_A = ( ushort ) value_I;
151        Memory.SetRegisterValue( reg1, value_A );
152        break;

```

```

153     case 0x18:
154         reg1 = Memory.ReadNextByteFromProgramCode();
155         reg2 = Memory.ReadNextByteFromProgramCode();
156         value_A = Memory.GetRegisterValue( reg1 );
157         value_B = Memory.GetRegisterValue( reg2 );
158         value_I = value_A - value_B;
159         if ( value_I < 0 ) {
160             value_A = ( ushort ) ( ushort.MaxValue + value_I + 1 );
161             Memory.SetFlag( 3 );
162         } else value_A = ( ushort ) value_I;
163         Memory.SetRegisterValue( reg1, value_A );
164         break;
165     case 0x19:
166         byte_A = Memory.ReadNextByteFromProgramCode();
167         byte_B = Memory.ReadNextByteFromProgramCode();
168         Memory.PutAddress( Memory.GetIBP() );
169         Jump( BitConverter.ToUInt16( new byte[] { byte_B, byte_A }, 0 ) );
170         break;
171     case 0x1A:
172         Memory.SetIBP( Memory.GetAddress() );
173         ExecuteNextInstruction();
174         break;
175     case 0x1B:
176         Memory.Push( Memory.GetRegisterValue( Memory.ReadNextByteFromProgramCode() ) );
177         break;
178     case 0x1C:
179         reg = Memory.ReadNextByteFromProgramCode();
180         Memory.SetRegisterValue( reg, Memory.Pull() );
181         break;
182     case 0x1D:
183         Memory.Put( Memory.GetRegisterValue( Memory.ReadNextByteFromProgramCode() ) );
184         break;
185     case 0x1E:
186         reg = Memory.ReadNextByteFromProgramCode();
187         Memory.SetRegisterValue( reg, Memory.Get() );
188         break;
189     case 0x1F:
190         reg1 = Memory.ReadNextByteFromProgramCode();
191         reg2 = Memory.ReadNextByteFromProgramCode();
192         value_A = Memory.GetRegisterValue( reg1 );
193         value_B = Memory.GetRegisterValue( reg2 );
194         value_I = value_A * value_B;
195         if ( value_I > ushort.MaxValue ) {
196             value_A = ( ushort ) ( ( value_I - ushort.MaxValue ) - 1 );
197             Memory.SetFlag( 5 );
198         }
199         else value_A = ( ushort ) value_I;
200         Memory.SetRegisterValue( reg1, value_A );
201         break;
202     case 0x20:
203         reg1 = Memory.ReadNextByteFromProgramCode();
204         reg2 = Memory.ReadNextByteFromProgramCode();
205         value_A = Memory.GetRegisterValue( reg1 );
206         value_B = Memory.GetRegisterValue( reg2 );
207         if ( value_B == 0 ) {
208             value_A = 0xFFFF;
209             Memory.SetFlag( 6 );
210             Memory.SetRegisterValue( reg1, value_A );
211         }
212         else {
213             value_D = value_A / value_B;
214             if ( value_D == 0 ) {
215                 value_A = 0;
216                 Memory.SetFlag( 4 );
217                 Memory.SetRegisterValue( reg1, value_A );
218             }
219             else {
220                 value_A = ( ushort ) value_D;
221                 if ( ( ( int ) value_A % ( int ) value_B ) > 0 ) Memory.SetFlag( 7 );
222                 Memory.SetRegisterValue( reg1, value_A );
223             }
224         }
225         break;
226     case 0x21:
227         reg1 = Memory.ReadNextByteFromProgramCode();
228         reg2 = Memory.ReadNextByteFromProgramCode();
229         value_A = Memory.GetRegisterValue( reg1 );

```

```

230         value_B = Memory.GetRegisterValue( reg2 );
231         if ( value_B == 0 ) {
232             value_A = 0xFFFF;
233             Memory.SetFlag( 6 );
234             Memory.SetRegisterValue( reg1, value_A );
235         }
236         else {
237             value_A = ( ushort ) ( value_A % value_B );
238             Memory.SetRegisterValue( reg1, value_A );
239         }
240         break;
241     case 0x22:
242         byte_A = Memory.ReadNextByteFromProgramCode();
243         byte_B = Memory.ReadNextByteFromProgramCode();
244         if ( Memory.LoadedFlag ) {
245             Memory.PutAddress( Memory.GetIBP() );
246             Jump( BitConverter.ToInt16( new byte[] { byte_B, byte_A }, 0 ) );
247         }
248         break;
249     case 0x23:
250         byte_A = Memory.ReadNextByteFromProgramCode();
251         byte_B = Memory.ReadNextByteFromProgramCode();
252         if ( ! Memory.LoadedFlag ) {
253             Memory.PutAddress( Memory.GetIBP() );
254             Jump( BitConverter.ToInt16( new byte[] { byte_B, byte_A }, 0 ) );
255         }
256         break;
257     }
258     IO.ProveriPersistantFlagove();
259 }
260
261 protected static void Jump( ushort instruction_number ) {
262     ushort instruction_first_byte_location=(ushort)(Memory.byte_locations[instruction_number]+1);
263     Memory.SetIBP( instruction_first_byte_location );
264     ExecuteNextInstruction();
265 }
266
267 protected static ushort ShiftLeft( ushort value, int count ) {
268     bool last_bit = false;
269     int tmp1 = value;
270     if ( ( ( ushort ) tmp1 & ( ushort ) 32768 ) == ( ushort ) 32768 ) last_bit = true;
271     else last_bit = false;
272     int tmp = tmp1 << count;
273     if ( last_bit ) tmp = tmp | 1;
274     return ( ushort ) tmp;
275 }
276
277 protected static ushort ShiftRight( ushort value, int count ) {
278     bool last_bit = false;
279     int tmp1 = value;
280     if ( ( ( ushort ) tmp1 & ( ushort ) 1 ) == ( ushort ) 1 ) last_bit = true;
281     else last_bit = false;
282     int tmp = tmp1 >> count;
283     if ( last_bit ) tmp = tmp | 32768;
284     return ( ushort ) tmp;
285 }
286
287 protected static void Interrupt( byte code ) {
288     byte KK, LL, WW, XX, YY, ZZ, OO, PP, QQ, RR;
289     ushort A, B, C;
290     byte[] tmp;
291
292     switch ( code ) {
293     case 0xA0:
294         Memory.SetRegisterValue( 0x01, IO.NumberOfDisplays );
295         break;
296     case 0xA1:
297     case 0xA2:
298         A = Memory.GetRegisterValue( 0x01 );
299         tmp = BitConverter.GetBytes( A );
300         KK = tmp[ 1 ];
301         LL = tmp[ 0 ];
302         if ( code == 0xA1 ) IO.SendToDisplay( KK, LL );
303         else if ( code == 0xA2 ) IO.ClearDisplayWith( KK, LL );
304         break;
305     case 0xA3:
306         A = Memory.GetRegisterValue( 0x01 );

```

```

307         tmp = BitConverter.GetBytes( A );
308         KK = tmp[ 1 ];
309         LL = tmp[ 0 ];
310         Memory.SetRegisterValue( 0x04, IO.GetDisplayCurrentPosition( LL ) );
311         break;
312     case 0xA4:
313         A = Memory.GetRegisterValue( 0x01 );
314         tmp = BitConverter.GetBytes( A );
315         KK = tmp[ 1 ];
316         LL = tmp[ 0 ];
317         B = Memory.GetRegisterValue( 0x02 );
318         IO.SetDisplayCurrentPosition( LL, B );
319         break;
320     case 0xD0:
321         Memory.SetRegisterValue( 0x01, IO.NumberOfDisks );
322         break;
323     case 0xD1:
324     case 0xD2:
325         A = Memory.GetRegisterValue( 0x01 );
326         B = Memory.GetRegisterValue( 0x02 );
327
328         tmp = BitConverter.GetBytes( A );
329         KK = tmp[ 1 ];
330         LL = tmp[ 0 ];
331
332         tmp = BitConverter.GetBytes( B );
333         WW = tmp[ 1 ];
334         XX = tmp[ 0 ];
335
336         if ( code == 0xD1 ) IO.WriteToDisk( LL, B, KK );
337         else if ( code == 0xD2 ) Memory.SetRegisterValue( 0x04, IO.ReadFromDisk( LL, B ) );
338
339         break;
340     case 0xE0:
341         Memory.SetRegisterValue( 0x04, ( byte ) IO.INPUT_BUFFER.Count );
342         break;
343     case 0xE1:
344         Memory.SetRegisterValue( 0x04, IO.GetFromInputBuffer() );
345         break;
346     case 0xB0:
347         int unix_time = ( int ) ( DateTime.UtcNow - new DateTime(1970,1,1,0,0,0)).TotalSeconds;
348         tmp = BitConverter.GetBytes( unix_time );
349         OO = tmp[ 3 ];
350         PP = tmp[ 2 ];
351         QQ = tmp[ 1 ];
352         RR = tmp[ 0 ];
353         tmp = new byte[] { PP, OO };
354         A = ( ushort ) BitConverter.ToUInt16( tmp, 0 );
355         tmp = new byte[] { RR, QQ };
356         B = ( ushort ) BitConverter.ToUInt16( tmp, 0 );
357         Memory.SetRegisterValue( 0x05, A );
358         Memory.SetRegisterValue( 0x06, B );
359         break;
360     case 0xB1:
361         DateTime dt = DateTime.Now;
362         tmp = BitConverter.GetBytes( MilantexDateTime() );
363
364         OO = tmp[ 3 ];
365         PP = tmp[ 2 ];
366         QQ = tmp[ 1 ];
367         RR = tmp[ 0 ];
368
369         tmp = new byte[] { PP, OO };
370         A = ( ushort ) BitConverter.ToUInt16( tmp, 0 );
371         tmp = new byte[] { RR, QQ };
372         B = ( ushort ) BitConverter.ToUInt16( tmp, 0 );
373
374         Memory.SetRegisterValue( 0x05, A );
375         Memory.SetRegisterValue( 0x06, B );
376         break;
377     }
378 }
379
380 static string u_binarni_string( int n, int size ) {
381     char[] b = new char[ size ];
382     int pos = size - 1;
383     int i = 0;

```

```

384     while ( i < size ) {
385         if ( ( n & ( 1 << i ) ) != 0 ) b[ pos ] = '1';
386         else b[ pos ] = '0';
387         pos--; i++;
388     }
389     return new string( b );
390 }
391
392 public static UInt32 MilantexDateTime() {
393     string s = "";
394     DateTime dt = DateTime.Now;
395     UInt32 time = 0;
396
397     s += u_binarni_string( dt.Year - 1970, 6 );
398     s += u_binarni_string( dt.Month, 4 );
399     s += u_binarni_string( dt.Day, 5 );
400     s += u_binarni_string( dt.Hour, 5 );
401     s += u_binarni_string( dt.Minute, 6 );
402     s += u_binarni_string( dt.Second, 6 );
403
404     time = Convert.ToUInt32( s, 2 );
405
406     return time;
407 }
408 }
409 }

```

3.4 SET INSTRUKCIJA PROCESORA MILANTEX TX16 RAČUNARA

Za potrebe izrade emulatora, za procesor Milantex Tx16 računara je konstruisan kompletan set instrukcija po ugledu na RISC (Reduced instruction set computing) modelu procesora, uz elemente nekih drugih starijih procesora Intel arhitekture.

Procesor broji 36 instrukcija koje su prikazane u sledećoj tabeli:

Bajt kôd	Naziv	Parametri instrukcije		
		Destin	Izvor 1	Izvor 2
00	NOP			
01	MOV	R		V
02	INC	R		
03	DEC	R		
04	CPY	R1	R2	
05	SAVE	R1	R2	
06	LOAD	R1	R2	
07	INT	V		
08	SHL	R		V
09	SHR	R		V
0A	RTL	R		V
0B	RTR	R		V
0C	JMP	A		
0D	JFS	A		
0E	JFC	A		
0F	LDF	V		
10	CAF			
11	CLF			

Bajt kôd	Naziv	Parametri instrukcije		
		Destin	Izvor 1	Izvor 2
12	CMP	R1	R2	
13	AND	A		
14	OR	A		
15	XOR	A		
16	NOT	R		
17	ADD	R1	R2	
18	SUB	R1	R2	
19	CALL	A		
1A	RET			
1B	PUSH	R		
1C	PULL	R		
1D	PUT	R		
1E	GET	R		
1F	MUL	R1	R2	
20	DIV	R1	R2	
21	MOD	R1	R2	
22	CFS	A		
23	CFC	A		

Tabela 1 - Spisak instrukcija procesora Milantex Tx16 računara sa spisom parametara

Parametri su predstavljeni skraćenicama i predstavljaju sledeće tipove podataka:

- R, R1 i R2 predstavljaju registre procesora (postoje registri A, B, C, D, E i F)
- A predstavlja adresu u programskoj memoriji
- V predstavlja dvobajtnu vrednost (literal) ručno upisanu u kodu programa

3.5 INTERAPTI PROCESORA MILANTEX Tx16 RAČUNARA

Procesor Milantex Tx16 računara podržava 12 ugrađenih interapta, tj. hardverskih funkcija posebne namene. Posebna instrukcija INT se koristi za aktiviranje interapta i pokreće se interapt čiji je broj prosleđen kao jedini osmobitni argument naredbe uz INT instrukciju.

Spisak interapta procesora Milantex Tx16 računara

Brojevi Interapta su osmobitne vrednosti koje se kao jedini argument instrukcije pišu u naredbi u izvornom kodu programa u assembleru. Spisak interapta je dat u sledećoj tabeli:

Broj	Opis funkcionalnosti interapta	Struktura ulaznih parametara i izlaza
0xD0	Smešta broj priključenih diskova u registar A.	Broj diskova LL je upisan u reg. A[KKLL]
0xD1	Upisuje vrednost KK iz registra A na disk pod rednim brojem LL, gde je registar A = KKLL, na adresu na disku koja je smeštena u registar B.	A[KK LL] = KK - vrednost, LL - disk B[WW XX] = WWXX - adresa na disku
0xD2	Iščitava vrednost sa diska por rednim brojem LL, gde LL bajt od 8 najnižih bitova registra A = KKLL, sa adrese na disku koja je smeštena u registar B. Iščitano vrednost smešta u registar D, na 8 najnižih bitova registra D.	A[KK LL] = LL - disk B[WW XX] = WWXX - adresa na disku Vrednost ZZ je upisana u registar D[YYZZ]
0xA0	Smešta broj priključenih displeja u registar A.	Broj displeja LL je upisan u reg. A[KKLL]
0xA1	Šalje karakter LL iz registra A na displej pod rednim brojem KK, gde je A = KKLL na prvo sledeće mesto na displeju.	A[KK LL] = KK - displej, LL - karakter
0xA2	Šalje signal displeju pod rednim brojem KK da očisti ceo sadržaj i zameni svaku poziciju karakterom LL iz registra A, gde je A = KKLL.	A[KK LL] = KK - displej, LL - karakter
0xA3	Smešta trenutnu slobodnu poziciju YYZZ na displeju pod rednim brojem LL u registar D.	A[KKLL] = LL - displej Vrednost YYZZ je upisana u reg. D[YYZZ]
0xA4	Postavlja vrednost WWXX za poziciju odakle displej LL ispisuje karaktere.	A[KK LL] = LL - displej B[WW XX] = WWXX - pozicija
0xE0	Smešta broj zapisa iz INPUT BUFFER-a u registar A.	Broj zapisa LL INPUT BUFFER-a je upisan u registar A[KKLL]
0xE1	Iščitava vrednost iz INPUT BUFFER-a u registar D.	Vrednost ZZ iz INPUT BUFFER-a je upisana u registar D D[YYZZ] i izbrisana iz INPUT BUFFER-a
0xB0	Prenosi 32bit TIMESTAMP [OOPPQRRR] trenutnog vremena u registre E[OOPP] i F[QRRR], podeljeno na pola.	TIMESTAMP [OOPPQRRR] = E[OOPP] i F[QRRR] TIMESTAMP je u Linux Timestamp formatu!
0xB1	Prenosi trenutni datum i vreme u registre E i F po specijalnom Milantex Date Time formatu.	E[OOPP] i F[QRRR] = Milantex Date Time

Tabela 2 - Spisak interapta procesora Milantex Tx16 računara

Milantex Date Time format

Interapt 0XB1 nakon izvršavanja u registre E i F smešta vrednosti koje, kada se razlože, predstavljaju informacije o trenutnom datumu i vremenu, a raspored po bitovima je upisan u sledećoj tabeli:

E																F															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Godine						Meseci					Dani					Sati					Minuti					Sekunde					
0-63 od 63 + 1970						0-12 od 15					0-31 od 31					0-23 od 31					0-59 od 63					0-59 od 63					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Tabela 3 - Raspored informacija o datumu i vremenu smeštenom u registre E i F nakon izvršavanja INT 0XB1

3.6 KLASA ZA PRIKAZ REGISTRA (REGISTERDISPLAY KLASA)

Klasa RegisterDisplay je klasa koja je nasleđena iz klase Control koja predstavlja grafički objekat koji može da se prikaže u okviru Windows Formi, tj. prozora. Osnovna nemena ove klase je da grafički predstavi stanje jednog registra i da prati njegove promene i grafički ih prezentuje korisniku koji posmatra glavni prozor emulatora.

Klasa RegisterDisplay, naravno, ima i svoju grafički komponentu koja je dizajnirana uz pomoć ugrađenih alata za dizajniranje grafičkih komponenata koji su sastavni deo Visual Studio 2012 Express Edition.

Klasa CPU sadrži šest javnih metoda i konstruktor, koji su ovde nabrojani:

- public RegisterDisplay()
- void ClearBit(ushort index)
- void SetBit(ushort index)
- void SetValue(ushort Value)
- void Increment()
- void Decrement()

Pored javnih, klasa sadrži i dva zaštićena metoda:

- bool BitState(int value, int position)
- void OnPaint(PaintEventArgs pe) - Prepisani metod iz nasleđene klase

Osim metoda, klasa sadrži još dva elementa od kojih je jedan trenutna vrednost u registru i zove se Value, tipa *ushort*, dok je drugi element lista instanci klase PictureBox koja predstavlja grafičku komponentu za prikaz slika u prozorima i u ovoj klasi služi da prikaže sivi ili crveni kvadratić u zavisnosti od toga da li je bit unutar bajtova, koji predstavljaju vrednost u registru, nula ili jedan, respektivno.



Slika 5 - Izgled grafičkog prikaza registra F u prozoru emulatora

Izvorni kod klase RegisterDisplay

```

1 using System;
2 using System.Collections.Generic;
3 using System.Collections;
4 using System.ComponentModel;
5 using System.Data;
6 using System.Drawing;
7 using System.Text;
8 using System.Windows.Forms;
9
10 namespace Milantex_Tx16_Emulator {
11     public partial class RegisterDisplay : Control {
12         private List<PictureBox> bits = new List<PictureBox>();
13         private ushort Value;
14
15         public RegisterDisplay() {
16             InitializeComponent();
17             Value = 0;
18             this.CreateControl();
19             this.MaximumSize = new Size( 400 , 25 );
20             this.MinimumSize = new Size( 400 , 25 );
21             this.Size = new Size( 400 , 25 );
22             for ( int i = 0 ; i < 16 ; i++ ) {
23                 PictureBox pb = new PictureBox();
24                 pb.CreateControl();
25                 pb.Parent = this;
26                 pb.Location = new Point( i * 25 , 0 );

```

```
27         pb.Size = new Size( 25 , 25 );
28         pb.Image = Ikonice.Images[ 0 ];
29         bits.Add( pb );
30     }
31 }
32
33 public void ClearBit( ushort index ) {
34     try { bits[ index ].Image = Ikonice.Images[ 0 ]; } catch ( Exception ) { }
35 }
36
37 public void SetBit( ushort index ) {
38     try { bits[ index ].Image = Ikonice.Images[ 1 ]; } catch ( Exception ) { }
39 }
40
41 protected bool BitState( int value, int position ) {
42     int bit = value & ( 1 << position );
43     return bit > 0;
44 }
45
46 public void SetValue( ushort Value ) {
47     if ( Value < 0 ) Value = 0;
48     if ( Value > ushort.MaxValue ) Value = ushort.MaxValue;
49
50     BitArray larrBits = new BitArray( System.BitConverter.GetBytes( Value ) );
51     if ( larrBits[ 15 ] ) SetBit( 0 ); else ClearBit( 0 );
52     if ( larrBits[ 14 ] ) SetBit( 1 ); else ClearBit( 1 );
53     if ( larrBits[ 13 ] ) SetBit( 2 ); else ClearBit( 2 );
54     if ( larrBits[ 12 ] ) SetBit( 3 ); else ClearBit( 3 );
55     if ( larrBits[ 11 ] ) SetBit( 4 ); else ClearBit( 4 );
56     if ( larrBits[ 10 ] ) SetBit( 5 ); else ClearBit( 5 );
57     if ( larrBits[ 9 ] ) SetBit( 6 ); else ClearBit( 6 );
58     if ( larrBits[ 8 ] ) SetBit( 7 ); else ClearBit( 7 );
59     if ( larrBits[ 7 ] ) SetBit( 8 ); else ClearBit( 8 );
60     if ( larrBits[ 6 ] ) SetBit( 9 ); else ClearBit( 9 );
61     if ( larrBits[ 5 ] ) SetBit( 10 ); else ClearBit( 10 );
62     if ( larrBits[ 4 ] ) SetBit( 11 ); else ClearBit( 11 );
63     if ( larrBits[ 3 ] ) SetBit( 12 ); else ClearBit( 12 );
64     if ( larrBits[ 2 ] ) SetBit( 13 ); else ClearBit( 13 );
65     if ( larrBits[ 1 ] ) SetBit( 14 ); else ClearBit( 14 );
66     if ( larrBits[ 0 ] ) SetBit( 15 ); else ClearBit( 15 );
67
68     this.Value = Value;
69 }
70
71 public void Increment() {
72     Value++;
73     if ( Value > ushort.MaxValue ) Value = 0;
74     SetValue( Value );
75 }
76
77 public void Decrement() {
78     Value--;
79     if ( Value < 0 ) Value = ushort.MaxValue;
80     SetValue( Value );
81 }
82
83 protected override void OnPaint( PaintEventArgs pe ) {
84     base.OnPaint( pe );
85 }
86 }
87 }
```

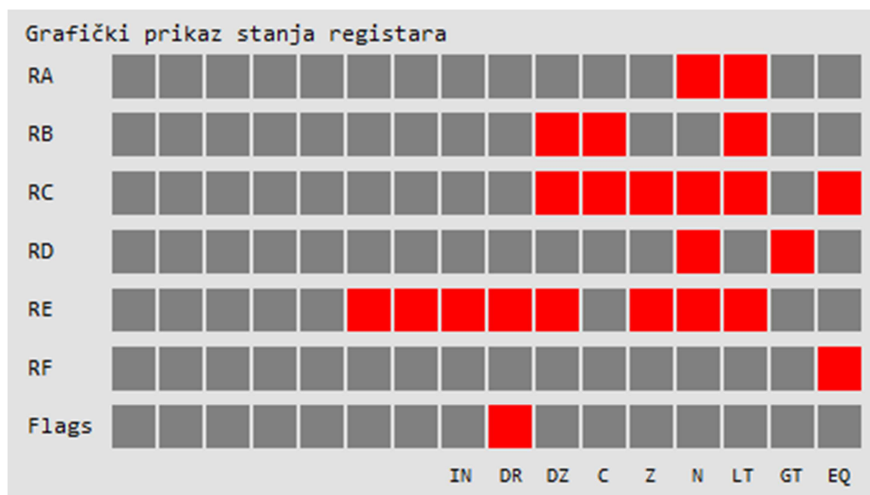
Izvorni kod grafičke komponente klase RegisterDisplay

Klasa RegisterDisplay ima grafičku komponentu koja u svom prostoru prikazuje niz slika crvenih ili sivih kvadratića čiji redosled zavisi od stanja bitova i koji predstavljaju trenutnu vrednost u registru. Grafička komponenta klase je dizajnirana korišćenjem dizajner alata Visual Studio 2012 Express.

```

1 namespace Milantex_Tx16_Emulator {
2     partial class RegisterDisplay {
3         private System.ComponentModel.IContainer components = null;
4
5         protected override void Dispose( bool disposing ) {
6             if ( disposing && ( components != null ) ) {
7                 components.Dispose();
8             }
9             base.Dispose( disposing );
10        }
11
12        private void InitializeComponent() {
13            this.components = new System.ComponentModel.Container();
14            System.ComponentModel.ComponentResourceManager resources =
15                new System.ComponentModel.ComponentResourceManager( typeof( RegisterDisplay ) );
16            this.Ikonice = new System.Windows.Forms.ImageList( this.components );
17            this.SuspendLayout();
18            this.Ikonice.ImageStream = ( ( System.Windows.Forms.ImageListStreamer )
19                ( resources.GetObject( "Ikonice.ImageStream" ) ) );
20            this.Ikonice.TransparentColor = System.Drawing.Color.Transparent;
21            this.Ikonice.Images.SetKeyName( 0 , "off.png" );
22            this.Ikonice.Images.SetKeyName( 1 , "on.png" );
23            this.ResumeLayout( false );
24        }
25
26        private System.Windows.Forms.ImageList Ikonice;
27    }
28 }
29

```



Slika 6 - Prikaz registara u prozoru emulatora Milantex Tx16 računara

3.7 KLASA MEMORIJE (KLASA MEMORY)

Klasa Memory je apstraktna i statična klasa koja obavlja nekoliko bitnih funkcija u emulatoru, od kojih pored privremenog skladištenja vrednosti koje u nju program upisuje, služi i za konverziju tabele znakova i simbola, tj. karaktera koje koristi Milantex Tx16 arhitektura računara iz i u UTF-8 enkodirane karaktere koji se koriste na računaru na kojem se emulacija izvršava.

Ipak, osnovna funkcija klase je da oponaša rad operativne memorije mikroračunara, ali i ostalih brzih memorija unutar procesora, kao što su registri i stekovi, ali i da se stara o grafičkim predstavljanjima vrednosti iz memorije u okviru grafičkog interfejsa emulatora. Prema tome, ova klasa, pored klase CPU, čini jedan od najbitnijih i centralnih modula emulatora.

Klasa Memory broji devetnaest javnih statičnih metoda koji su ovde nabrojani:

- void SetIBP(ushort value)
- ushort GetIBP()
- void PutAddress(ushort Address)
- ushort GetAddress()
- void Push(ushort Value)
- ushort Pull()
- void Put(ushort Value)
- ushort Get()
- void SetFlag(int flag)
- void ClearFlag(int flag)
- void ClearLoadedFlag()
- void ClearFlags()
- void LoadFlagForTesting(int flag)
- byte ReadNextByteFromProgramCode()
- void InitMemory()
- void SetRegisterValue(byte Register, ushort Value)
- ushort GetRegisterValue(byte Register)
- void WriteToMemory(ushort Address, ushort Value)
- ushort ReadFromMemory(ushort Address)

Pored nabrojanih javnih metoda, klasa sadrži i dva metoda koji nisu javni i to:

- void PrintMemoryChange(ushort Address, ushort Value)
- void PrintMemoryChangeDo(ushort Address, ushort Value)

Klasa sadrži i pomoćne elemente, kao i statusne indikatore za potrebe ostalih modula emulatora. Neki od elemenata su složeni tipovi podataka, a jedan broj elemenata su nizovi elemenata, kao i objekti koji predstavljaju liste elemenata.

Veličina radne memorije emuliranog sistema je fiksna i iznosi 65536 bajtova, odnosno 64KB. Grafički prikaz memorije je u interfejsu rešen u vidu matrice od 256 sa 256 piksela. Klasa Memory upravlja i registrima emuliranog sistema, a dostupni registri za konkretnu arhitekturu su: A, B, C, D, E i F registar. Osim ovih namenskih registara, postoji i specijalni registar čiji bitovi predstavljaju flagove koji označavaju stanja određenih delova sistema.

Flagovi pružaju i informacije o tome na koji način i sa kakvim rezultatima je završena prethodno obavljena aritmetička operacija ili kontrolna instrukcija u procesoru. U okviru ove klase se nalazi i niz bajtova koji predstavljaju program koji je učitao (flešovan) u mikroračunar i maksimalna veličina memorije programa je 65536 bajtova, odnosno 64KB.

Emulator je jedini koji može da manipuliše vrednostima u programskoj memoriji, tako da ne postoji način da se instrukcijama u programu utiče na stanje ili sam kôd programa nakon što je emulator pokrenut. Klasa sadrži string koji predstavlja mapu karaktera sa kojom radi emulirana arhitektura. Kako emulirani arhitektura ne koristi standardnu ASCII mapu, postoje posebni metodi u drugim modulima emulatora koji vrše konverziju za potrebe prikaza na grafičkim prikazom stanja izlaznog uređaja, tj. displeja, kao i za potrebe konverzije karaktera unetih sa tastature u emulator.

Kompletna mapa karaktera sa vrednostima svakog karaktera u heksadecimalnom zapisu je prikazana u tabeli (korišćen je font adekvatan za prikaz svih specijalnih karaktera):

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
P	Q	R	S	T	U	V	W	X	Y	Z	Č	Ć	Ž	Š	Đ
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
q	r	s	t	u	v	w	x	y	z	č	ć	ž	š	đ	À
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	а	б	в
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
г	д	ђ	е	ж	з	и	ј	к	л	љ	м	н	њ	о	п
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
р	с	т	ћ	у	ф	х	ц	ч	џ	ш	ø	1	2	3	4
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
5	6	7	8	9	⓪	①	②	③	④	⑤	⑥	⑦	⑧	⑨	~
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
!	?	#	\$	%	^	&	(	)	[	]	{	}	<	>	≤
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
≥	=	@	*	/	+	-	.	,	:	;	'	"	_	!	¢
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
£	¤	¥	¦	§	¨	©	ª	«	»	±	°	¹			
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿	Δ	
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
Π	Σ	√	∞	∩	♪	♫	♦	♥	♣	♠	♂	♀	⅓	⅔	⅕
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
⅔	⅔	⅔	⅔	⅔	⅔	⅔	⅔	⅔	⅔	⅔	⅔	⅔	⅔	⅔	⅔
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
\	ˆ	%	θ	ˆ	ˆ	ˆ	ˆ	ˆ	ˆ	ˆ	ˆ	ˆ	ˆ	ˆ	ˆ
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

Tabela 4 - Mapa karaktera podržanih na Milantex Tx16 računaru

Izvorni kod klase Memory

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Drawing;
6 using System.Windows.Forms;
7
8 namespace Milantex_Tx16_Emulator {
9     public abstract class Memory {
10         public static string CharacterMap = "
ABCDEFHIJKLMNOPQRSTUVWXYZČČŠšabcdefghijklmnopqrstuvwxyzččššdABVГДЂЕЖЗИЈКЉМЊНЊОПРСТУФХЦЦШШабвгдђежзијкљмњнџопр
сѣуфхццшш0123456789①②③④⑤⑥⑦⑧⑨~! ?#$%^&()[]{}<>≤≥=@*/+-
.,;:'\"_!¢£¤¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñ
,.,;:'\"_!¢£¤¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñ";
11
12         protected static Bitmap MemMap;
13         public static PictureBox MEMORY_DISPLAY;
14         public static NewRegisterDisplay _RA, _RB, _RC, _RD, _RE, _RF, _FLAGS;
15         public static UInt64 BrojKrugova = 0;
16
17         public static int LoadedFlagIndex = 15;
18         public static bool LoadedFlag = false;
19
20         public static bool DisplayMemoryChanges = true;
21         public static bool DisplayRegistarChanges = true;
22         public static bool MemoryInitDone = false;
23
24         public static ushort[] MEMORY = new ushort[ 65536 ];
25         public static byte[] PROGRAM;
26         public static ushort[] byte_locations;
27         public static ushort RA, RB, RC, RD, RE, RF, FLAGS;
28
29         protected static List<ushort> AddressStack = new List<ushort>();
30         protected static List<ushort> LIFO = new List<ushort>();
31         protected static List<ushort> FIFO = new List<ushort>();
32
33         protected static int IBP = 0;
34         public static void SetIBP( ushort value ) { IBP = value; }
35         public static ushort GetIBP() { return ( ushort ) IBP; }
36
37
38         public static void PutAddress( ushort Address ) {
39             AddressStack.Add( Address );
40         }
41
42
43         public static ushort GetAddress() {
44             ushort ret = 0;
45             if ( AddressStack.Count > 0 ) {
46                 ret = AddressStack[ AddressStack.Count - 1 ];
47                 AddressStack.RemoveAt( AddressStack.Count - 1 );
48             }
49             return ret;
50         }
51
52
53         public static void Push( ushort Value ) {
54             FIFO.Add( Value );
55         }
56
57
58         public static ushort Pull() {
59             ushort ret = 0;
60             if ( FIFO.Count > 0 ) {
61                 ret = FIFO[ 0 ];
62                 FIFO.RemoveAt( 0 );
63             }
64             return ret;
65         }
66
67
68         public static void Put( ushort Value ) {
69             LIFO.Add( Value );
70         }

```

```

71
72
73     public static ushort Get() {
74         ushort ret = 0;
75         if ( LIFO.Count > 0 ) {
76             ret = LIFO[ LIFO.Count - 1 ];
77             LIFO.RemoveAt( LIFO.Count - 1 );
78         }
79         return ret;
80     }
81
82     public static void SetFlag( int flag ) {
83         SetRegisterValue( 0xFF, ( ushort ) ( FLAGS | ( ushort ) Math.Pow( 2, flag ) ) );
84     }
85
86     public static void ClearFlag( int flag ) {
87         int number = GetRegisterValue( 0xFF );
88         number &= ~( 1 << flag );
89         SetRegisterValue(0xFF, (ushort) number );
90     }
91
92     public static void ClearLoadedFlag() {
93         ClearFlag( LoadedFlagIndex );
94         LoadedFlag = false;
95         LoadedFlagIndex = 15;
96     }
97
98     public static void ClearFlags() {
99         SetRegisterValue( 0xFF, ( ushort ) ( 0xF0 & GetRegisterValue( 0xFF ) ) );
100        LoadedFlag = false;
101    }
102
103    public static void LoadFlagForTesting( int flag ) {
104        LoadedFlagIndex = flag;
105        LoadedFlag = ( FLAGS & ( int ) ( Math.Pow( 2, ( double ) flag ) ) ) > 0;
106    }
107
108    public static byte ReadNextByteFromProgramCode() {
109        if ( PROGRAM.Length > 0 ) {
110            byte to_return = PROGRAM[ IBP ];
111            IBP++;
112            if ( IBP == PROGRAM.Length ) {
113                IBP = 0;
114                BrojKrugova++;
115            }
116            return to_return;
117        }
118        return 0x00;
119    }
120
121    public static void InitMemory() {
122        if ( MEMORY_DISPLAY == null ) return;
123        PROGRAM = new byte[ 0 ];
124        IBP = 0;
125
126        if ( MEMORY_DISPLAY.Image != null ) MEMORY_DISPLAY.Image.Dispose();
127        MemMap = new Bitmap( 256, 256 );
128        MEMORY_DISPLAY.Image = ( Image ) MemMap;
129
130        for ( ushort i = 0 ; i < ushort.MaxValue ; i++ ) {
131            MEMORY[ i ] = new ushort();
132            MEMORY[ i ] = 0;
133            PrintMemoryChange( i, MEMORY[ i ] );
134        }
135
136        SetRegisterValue( 0x01, 0 );
137        SetRegisterValue( 0x02, 0 );
138        SetRegisterValue( 0x03, 0 );
139        SetRegisterValue( 0x04, 0 );
140        SetRegisterValue( 0x05, 0 );
141        SetRegisterValue( 0x06, 0 );
142        SetRegisterValue( 0xFF, 0 );
143
144        MemoryInitDone = true;
145    }
146
147    delegate void PrintMemoryChangeCallback( ushort Address, ushort Value );

```

```

148     protected static void PrintMemoryChange( ushort Address, ushort Value ) {
149         if ( MEMORY_DISPLAY.InvokeRequired ) {
150             PrintMemoryChangeCallback d=new PrintMemoryChangeCallback(PrintMemoryChangeDo);
151             MEMORY_DISPLAY.Invoke( d, new object[] { Address, Value } );
152         } else PrintMemoryChangeDo( Address, Value );
153     }
154
155     protected static void PrintMemoryChangeDo( ushort Address, ushort Value ) {
156         int y = Address / 256;
157         int x = Address % 256;
158         byte c_part = ( byte ) ( Value / ( ushort ) 256 );
159         MemMap.SetPixel( x, y, Color.FromArgb( c_part, c_part, c_part ) );
160         MEMORY_DISPLAY.Image = ( Image ) MemMap;
161     }
162
163     public static void SetRegisterValue( byte Register, ushort Value ) {
164         switch ( Register ) {
165             case 0x01:
166                 RA = Value;
167                 if ( DisplayRegistarChanges ) _RA.SetValue( Value );
168                 break;
169             case 0x02:
170                 RB = Value;
171                 if ( DisplayRegistarChanges ) _RB.SetValue( Value );
172                 break;
173             case 0x03:
174                 RC = Value;
175                 if ( DisplayRegistarChanges ) _RC.SetValue( Value );
176                 break;
177             case 0x04:
178                 RD = Value;
179                 if ( DisplayRegistarChanges ) _RD.SetValue( Value );
180                 break;
181             case 0x05:
182                 RE = Value;
183                 if ( DisplayRegistarChanges ) _RE.SetValue( Value );
184                 break;
185             case 0x06:
186                 RF = Value;
187                 if ( DisplayRegistarChanges ) _RF.SetValue( Value );
188                 break;
189             case 0xFF:
190                 FLAGS = Value;
191                 if ( DisplayRegistarChanges ) _FLAGS.SetValue( Value );
192                 break;
193         }
194     }
195
196     public static ushort GetRegisterValue( byte Register ) {
197         ushort value = 0;
198         switch ( Register ) {
199             case 0x01: value = RA; break;
200             case 0x02: value = RB; break;
201             case 0x03: value = RC; break;
202             case 0x04: value = RD; break;
203             case 0x05: value = RE; break;
204             case 0x06: value = RF; break;
205             case 0xFF: value = FLAGS; break;
206         }
207         return value;
208     }
209
210     public static void WriteToMemory( ushort Address, ushort Value ) {
211         if ( MemoryInitDone ) {
212             MEMORY[ Address ] = Value;
213             if ( DisplayMemoryChanges ) PrintMemoryChange( Address, Value );
214         }
215     }
216
217     public static ushort ReadFromMemory( ushort Address ) {
218         if ( MemoryInitDone ) return MEMORY[ Address ];
219         return 0;
220     }
221 }
222 }

```

3.8 KLASA ULAZA I IZLAZA (I/O KLASA)

Klasa IO je apstraktna i statična klasa koja obuhvata rad sa virtualnim ulaznim i izlaznim uređajima, kao i sa uređajima za trajno skladištenje podataka, tj. sa virtualnih diskovima koji mogu da se priključe emulatoru.

Elementi koje klasa sadrži su:

- Lista virtualnih displeja kojih može da bude do 256;
- Lista virtualnih hard diskova od po 64KB kojih može da bude 256;
- *Buffer* od 256 karaktera ukucanih preko virtualne tastature koje program upisan u programsku memoriju emulatora preuzima tokom svog izvršavanja;

Klasa IO broji šesnaest javnih statičnih metoda koji su ovde nabrojani:

- `bool AttachDisplay( )`
- `bool RemoveDisplay( Display Ovaj )`
- `bool RemoveDisplay( int index )`
- `void SendToDisplay( byte DisplayNumber, byte CharacterCode )`
- `void ClearDisplayWith( byte DisplayNumber, byte CharacterCode )`
- `ushort GetDisplayCurrentPosition( byte DisplayNumber )`
- `void SetDisplayCurrentPosition( byte DisplayNumber, ushort NewPosition )`
- `bool AddachDisk( )`
- `bool RemoveDisk( Disk Ovaj )`
- `bool RemoveDisk( int index )`
- `void ProveriPersistantFlagove( )`
- `void PutIntoInputBuffer( byte value )`
- `byte GetFromInputBuffer( )`
- `byte GetInputBufferSize( )`
- `void WriteToDisk( byte DiskNumber, UInt16 Address, byte Value )`
- `byte ReadFromDisk( byte DiskNumber, UInt16 Address )`

Pored nabrojanih javnih metoda, klasa sadrži i dva metoda koji nisu javni i to:

- `void Proveri_flag_IN( )`
- `void SetTextBoxValue( )`

Klasa sadrži i pomoćne elemente, kao i javne brojače priključenih diskova i displeja koje klasa procesora koristi za utvrđivanje ispravnosti i mogućnosti izvršavanja određenih instrukcija.

Liste displeja i virtualnih diskova su liste koje nabrajaju instance displeja i virtualnih diskova, respektivno, odnosno objekte klasa `Display` i `Disk`.

Izvorni kod klase IO

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5
6 namespace Milantex_Tx16_Emulator {
7     public abstract class IO {
8         public static byte NumberOfDisks = 0;
9         public static byte NumberOfDisplays = 0;
10        public static List<Display> DISPLAYS = new List<Display>(256);
11        public static List<Disk> DISKS = new List<Disk>( 256 );
12        public static List<byte> INPUT_BUFFER = new List<byte>( 256 );
13        public static System.Windows.Forms.TextBox INPUT_BUFFER_TEXTBOX;
14
15        public static bool AttachDisplay() {
16            if ( NumberOfDisplays < 255 ) {
17                Display add = new Display();
18                add.CreateControl();
19                DISPLAYS.Add( add );
20                add.SetName( "Display " + DISPLAYS.IndexOf( add ) );
21                add.Show();
22                NumberOfDisplays = ( byte ) DISPLAYS.Count;
23                return true;
24            } else return false;
25        }
26
27        public static bool RemoveDisplay( Display Ovaj ) {
28            int c = DISPLAYS.Count;
29            DISPLAYS.Remove( Ovaj );
30            NumberOfDisplays = ( byte ) DISPLAYS.Count;
31            if ( c > DISPLAYS.Count ) return true;
32            else return false;
33        }
34
35        public static bool RemoveDisplay( int index ) {
36            int c = DISPLAYS.Count;
37            DISPLAYS.RemoveAt( index );
38            NumberOfDisplays = ( byte ) DISPLAYS.Count;
39            if ( c > DISPLAYS.Count ) return true;
40            else return false;
41        }
42
43        public static void SendToDisplay( byte DisplayNumber, byte CharacterCode ) {
44            if ( DisplayNumber < NumberOfDisplays ) {
45                DISPLAYS[ DisplayNumber ].PutCharacter( ( ushort ) CharacterCode );
46            }
47        }
48
49        public static void ClearDisplayWith( byte DisplayNumber, byte CharacterCode ) {
50            if ( DisplayNumber < NumberOfDisplays ) {
51                DISPLAYS[ DisplayNumber ].ClearDisplayWith( ( ushort ) CharacterCode );
52            }
53        }
54
55        public static ushort GetDisplayCurrentPosition( byte DisplayNumber ) {
56            if ( DisplayNumber < NumberOfDisplays ) {
57                return DISPLAYS[ DisplayNumber ].GetCurrentPosition();
58            } else return 0x0000;
59        }
60
61        public static void SetDisplayCurrentPosition( byte DisplayNumber, ushort NewPosition ) {
62            if ( DisplayNumber < NumberOfDisplays ) {
63                DISPLAYS[ DisplayNumber ].SetCurrentPosition( NewPosition );
64            }
65        }
66
67        public static bool AddachDisk() {
68            if ( NumberOfDisks < 255 ) {
69                Disk add = new Disk( "Disk" + DISKS.Count + ".mdsk" );
70                DISKS.Add( add );
71                NumberOfDisks = ( byte ) DISKS.Count;
72                return true;
73            } else return false;
74        }
75    }

```

```

76     public static bool RemoveDisk( Disk Ovaj ) {
77         int c = DISKS.Count;
78         DISKS.Remove( Ovaj );
79         NumberOfDisks = ( byte ) DISKS.Count;
80         if ( c > DISKS.Count ) return true;
81         else return false;
82     }
83
84     public static bool RemoveDisk( int index ) {
85         int c = DISKS.Count;
86         DISKS.RemoveAt( index );
87         NumberOfDisks = ( byte ) DISKS.Count;
88         if ( c > DISKS.Count ) return true;
89         else return false;
90     }
91
92     protected static void Proveri_flag_IN() {
93         if ( INPUT_BUFFER.Count > 0 ) Memory.SetFlag( 8 );
94         else Memory.ClearFlag( 8 );
95     }
96
97     public static void ProveriPersistantFlagove() {
98         Proveri_flag_IN();
99     }
100
101     public static void PutIntoInputBuffer( byte value ) {
102         INPUT_BUFFER.Add( value );
103         Proveri_flag_IN();
104     }
105
106     public static byte GetFromInputBuffer() {
107         if ( INPUT_BUFFER.Count > 0 ) {
108             byte ret = INPUT_BUFFER[ 0 ];
109             INPUT_BUFFER.RemoveAt( 0 );
110             SetTextBoxValue();
111             return ret;
112         }
113         else return 0x00;
114     }
115
116     delegate void RemoveFirstInputBoxCallback( );
117     protected static void SetTextBoxValue( ) {
118         if ( INPUT_BUFFER_TEXTBOX.InvokeRequired ) {
119             RemoveFirstInputBoxCallback d = new RemoveFirstInputBoxCallback( SetTextBoxValue );
120             INPUT_BUFFER_TEXTBOX.Invoke( d );
121         }
122         else {
123             if (INPUT_BUFFER_TEXTBOX.Text.Length>3)INPUT_BUFFER_TEXTBOX.Text=
124                 INPUT_BUFFER_TEXTBOX.Text.Substring(3);
125             else if ( INPUT_BUFFER_TEXTBOX.Text.Length == 3 ) INPUT_BUFFER_TEXTBOX.Text = "";
126         }
127     }
128
129     public static byte GetInputBufferSize() {
130         return ( byte ) INPUT_BUFFER.Count;
131     }
132
133     public static void WriteToDisk( byte DiskNumber, UInt16 Address, byte Value ) {
134         if ( DiskNumber < NumberOfDisks ) {
135             DISKS[ DiskNumber ].Write( Address, Value );
136         }
137     }
138
139     public static byte ReadFromDisk( byte DiskNumber, UInt16 Address ) {
140         if ( DiskNumber < NumberOfDisks ) {
141             return DISKS[ DiskNumber ].Read( Address );
142         } else return 0;
143     }
144 }

```

3.9 KLASA DISPLEJA (DISPLAY KLASA)

Namena klase Display je da prikazuje tekstualne podatke poslate na izlaz emulatora u prostoru od 40 karaktera širine sa 15 karaktera visine.

Klasa Display ima grafičku komponentu u obliku forme sa tekstualnim poljem koje može da menja boje u nekoliko predefinisanih kombinacija boje pozadine i teksta koje su odabrane tako da ostavljaju utisak korišćenja starih monohromatskih monitora niske rezolucije bez kursora.

Klasa Display broji šest javnih metoda i konstruktor koji su ovde nabrojani:

- Display()
- void SetCharacter(ushort Address, ushort value)
- ushort GetCurrentPosition()
- void SetCurrentPosition(ushort Position)
- void PutCharacter(ushort value)
- void ClearDisplayWith(ushort initial_character_code_value)
- void SetName(string name)

Pored javnih metoda, klasa sadrži i zaštićene metode i *event handler*-e, kao što su:

- void Display_FormClosed(object sender, FormClosedEventArgs e)
- Font ChangeFontSize(Font font, float fontSize)
- void ToClearDisplay(string value)
- void ToDisplay(ushort Address, ushort value)

Klasa sadrži element:

- int CurrentAddress - Čija je namena da vodi računa o poziciji kursora na displeju

Iz osobina i naziva metoda može da se vidi da klasa Display zapravo u sebi uključuje i funkcionalnosti tipične grafičke kartice, a ne samo displeja koji je u suštini uređaj koji samo prikazuje sliku korisniku. Obzirom da je u pitanju emulacija, a ne simulacija kod koje bi namena modula displeja bila samo da prikazuje sliku iz signala koji dobija od grafičke kartice, u slučaju kada je ovaj modul sastavni deo emulatora, odlučio sam da neke nadležnosti grafičke kartice premestim u klasu display isključivo iz razloga što je to praktičnija realizacija od one u kojoj bih morao da osmišljam protokol kojim bi modul grafičke kartice, kada bi postojao, morao da komunicira sa modulom displeja kako bi mu preneo instrukciju šta, kako i gde da prikaže u svom prostoru za prikaz karaktera.

Izvorni kod klase Display

```

1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9
10 namespace Milantex_Tx16_Emulator {
11     public partial class Display : Form {
12         public int CurrentAddress = 0;
13         protected bool start = true;
14
15         public void SetCharacter( ushort Address, ushort value ) {
16             if ( this.IsDisposed ) return;
17             if ( value > 255 ) value = 255;
18             int index = ( int ) value;
19
20             if ( index >= Memory.CharacterMap.Length ) index = Memory.CharacterMap.Length - 1;
21
22             if ( Address >= 0 && Address < 560 ) {
23                 Address = ( ushort ) ( ( Math.Ceiling( ( double ) ( Address / 40 ) ) ) * 2 + Address );
24                 ToDisplay( Address, value );
25             }
26         }
27
28         public ushort GetCurrentPosition() {
29             return (ushort) CurrentAddress;
30         }
31
32         public void SetCurrentPosition( ushort Position ) {
33             CurrentAddress = Position;
34         }
35
36         public void PutCharacter( ushort value ) {
37             if ( Memory.CharacterMap.Length > value ) {
38                 SetCharacter( (ushort) CurrentAddress++, value );
39                 if ( CurrentAddress >= 560 ) {
40                     CurrentAddress = 0;
41                 }
42             }
43         }
44
45         public void ClearDisplayWith( ushort initial_character_code_value ) {
46             if ( Memory.CharacterMap.Length > initial_character_code_value ) {
47                 string init = "";
48                 for ( int y = 0 ; y < 14 ; y++ ) {
49                     for ( int x = 0 ; x < 40 ; x++ )
50                         init += Memory.CharacterMap[ initial_character_code_value ];
51                 init += "\r\n";
52                 ToClearDisplay( init );
53             }
54             if ( start ) CurrentAddress = 0;
55             else {
56                 start = false;
57                 CurrentAddress = -1;
58             }
59         }
60     }
61
62     public Display( ) {
63         InitializeComponent();
64         ClearDisplayWith( 0 );
65         CurrentAddress = 0;
66     }
67
68     public void SetName( string name ) {
69         this.Text = name;
70     }
71
72     private void Display_FormClosed( object sender, FormClosedEventArgs e ) {
73         IO.RemoveDisplay( this );
74         this.Dispose();

```

```

75     }
76
77     protected delegate void DisplayDelegate( ushort Address, ushort value );
78     protected void ToDisplay( ushort Address, ushort value ) {
79         try {
80             if ( DISP.InvokeRequired ) {
81                 DisplayDelegate d = new DisplayDelegate( ToDisplay );
82                 DISP.Invoke( d, new object[] { Address, value } );
83             }
84             else {
85                 DISP.Text = DISP.Text = DISP.Text.Substring( 0, Address ) +
86                     Memory.CharacterMap[ value ] +
87                     DISP.Text.Substring( Address + 1 );
88             }
89         }
90         catch ( Exception ) { }
91     }
92
93
94     protected delegate void ClearDisplayDelegate( string value );
95     protected void ToClearDisplay( string value ) {
96         try {
97             if ( DISP.InvokeRequired ) {
98                 ClearDisplayDelegate d = new ClearDisplayDelegate( ToClearDisplay );
99                 DISP.Invoke( d, new object[] { value } );
100             }
101             else {
102                 DISP.Text = value;
103             }
104         } catch ( Exception ) { }
105     }
106
107     protected Font ChangeFontSize( Font font, float fontSize ) {
108         if ( font != null ) {
109             float currentSize = font.Size;
110             if ( currentSize != fontSize ) font =
111                 new Font(font.Name,fontSize,font.Style,font.Unit,font.GdiCharSet,font.GdiVerticalFont);
112         }
113         return font;
114     }
115
116     private void Display_KeyPress( object sender, KeyPressEventArgs e ) {
117         if ( ( int ) e.KeyChar == 43 )
118             if ( this.Font.Size + 1 <= 32 )
119                 this.Font = ChangeFontSize( this.Font, this.Font.Size + 1 );
120         if ( ( int ) e.KeyChar == 45 )
121             if ( this.Font.Size - 1 >= 8 )
122                 this.Font = ChangeFontSize( this.Font, this.Font.Size - 1 );
123         if ( e.KeyChar == 'c' || e.KeyChar == 'C' ) {
124             if ( this.DISP.BackColor == Color.Black ) this.DISP.BackColor = Color.DarkBlue;
125             else if ( this.DISP.BackColor == Color.DarkBlue ) this.DISP.BackColor = Color.Teal;
126             else if ( this.DISP.BackColor == Color.Teal ) this.DISP.BackColor = Color.DarkGray;
127             else if ( this.DISP.BackColor == Color.DarkGray ) this.DISP.BackColor = Color.Black;
128         }
129         if ( e.KeyChar == 't' || e.KeyChar == 'T' ) {
130             if ( this.DISP.ForeColor == Color.Black ) this.DISP.ForeColor = Color.White;
131             else if ( this.DISP.ForeColor == Color.White ) this.DISP.ForeColor = Color.Black;
132         }
133         this.DISP.Font = this.Font;
134     }
135 }
136 }

```

Izvorni kod grafičke komponente klase Display

Klasa Display ima i grafičku komponentu pri emulatoru. Ova grafička komponenta je klasa koja nasleđuje klasu Windows Form, tj. prozor u Windows operativnom sistemu koja je dizajnirana korišćenjem dizajner alata Visual Studio 2012 Express.

Ovde je prikazan izvorni kod formiranje izgleda prozora koji predstavlja virtualni displej emulatora sa svim svojim elementima:

```

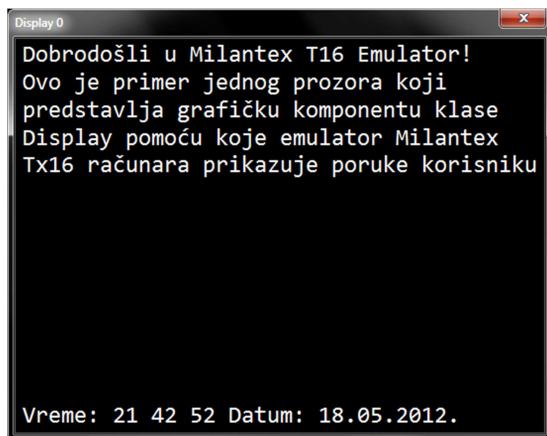
1 namespace Milantex_Tx16_Emulator {
2     partial class Display {
3         private System.ComponentModel.IContainer components = null;
4
5         protected override void Dispose( bool disposing ) {
6             if ( disposing && ( components != null ) ) {
7                 components.Dispose();
8             }
9             base.Dispose( disposing );
10        }
11
12        private void InitializeComponent() {
13            this.components = new System.ComponentModel.Container();
14            System.ComponentModel.ComponentResourceManager resources =
15                new System.ComponentModel.ComponentResourceManager( typeof( Display ) );
16            this.imageList1 = new System.Windows.Forms.ImageList( this.components );
17            this.DISP = new System.Windows.Forms.TextBox();
18            this.SuspendLayout();
19
20            this.imageList1.ColorDepth = System.Windows.Forms.ColorDepth.Depth4Bit;
21            this.imageList1.ImageSize = new System.Drawing.Size( 16, 16 );
22            this.imageList1.TransparentColor = System.Drawing.Color.Transparent;
23
24            this.DISP.BackColor = System.Drawing.Color.Black;
25            this.DISP.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
26            this.DISP.Cursor = System.Windows.Forms.Cursors.Arrow;
27            this.DISP.Dock = System.Windows.Forms.DockStyle.Fill;
28            this.DISP.Font = new System.Drawing.Font( "Consolas", 18F,
29                System.Drawing.FontStyle.Regular,
30                System.Drawing.GraphicsUnit.Point, ( ( byte ) ( 204 ) ) );
31            this.DISP.ForeColor = System.Drawing.Color.White;
32            this.DISP.Location = new System.Drawing.Point( 0, 0 );
33            this.DISP.Margin = new System.Windows.Forms.Padding( 3, 4, 3, 4 );
34            this.DISP.Multiline = true;
35            this.DISP.Name = "DISP";
36            this.DISP.ReadOnly = true;
37            this.DISP.Size = new System.Drawing.Size( 536, 399 );
38            this.DISP.TabIndex = 0;
39            this.DISP.Text = resources.GetString( "DISP.Text" );
40            this.DISP.KeyPress += new System.Windows.Forms.KeyPressEventHandler( this.Display_KeyPress );
41
42            this.AutoScaleDimensions = new System.Drawing.SizeF( 13F, 28F );
43            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
44            this.BackColor = System.Drawing.Color.DimGray;
45            this.ClientSize = new System.Drawing.Size( 536, 399 );
46            this.Controls.Add( this.DISP );
47            this.DoubleBuffered = true;
48            this.Font = new System.Drawing.Font( "Consolas", 18F, System.Drawing.FontStyle.Regular,
49                System.Drawing.GraphicsUnit.Point, ( ( byte ) ( 204 ) ) );
50            this.ForeColor = System.Drawing.Color.White;
51            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
52            this.KeyPreview = true;
53            this.Margin = new System.Windows.Forms.Padding( 3, 4, 3, 4 );
54            this.MaximizeBox = false;
55            this.MinimizeBox = false;
56            this.Name = "Display";
57            this.ShowIcon = false;
58            this.Text = "Display";
59            this.TopMost = true;
60            this.FormClosed += new System.Windows.Forms.FormClosedEventHandler( this.Display_FormClosed );
61            this.ResumeLayout( false );
62            this.PerformLayout();
63        }
64
65        private System.Windows.Forms.ImageList imageList1;
66        private System.Windows.Forms.TextBox DISP;
67    }
68 }

```

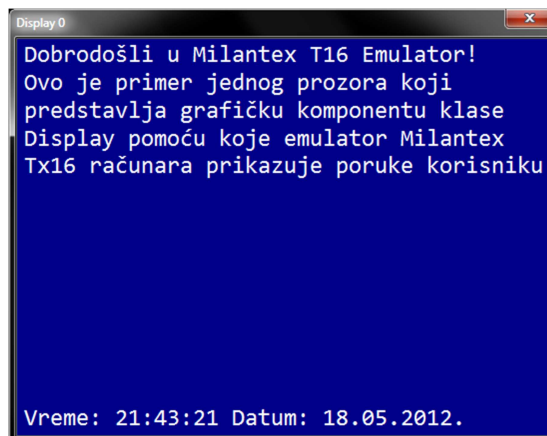
Boje pozadine grafičkog prikaza displeja

Prozor koji predstavlja displej može da menja pozadinsku boju pritiskom na taskter [C] dok je fokus miša u prozoru displeja. Pritiskom na taster [T] menja se boja teksta između bele i crne na smenu.

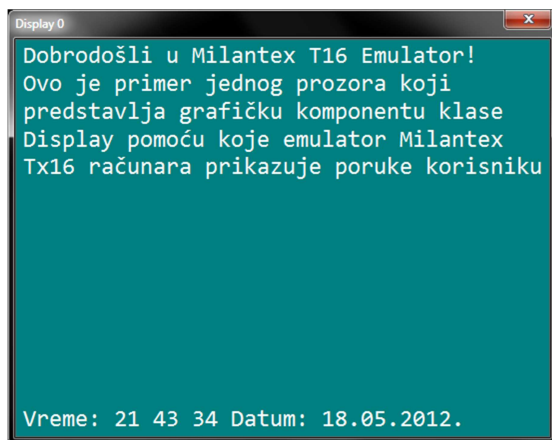
Odabrane su kombinacije boja koje su slične bojama starih monohromatskih monitora niskih rezolucija na kojima je nekada prikazivan ispis poruka ranijih generacija računarskih sklopova.



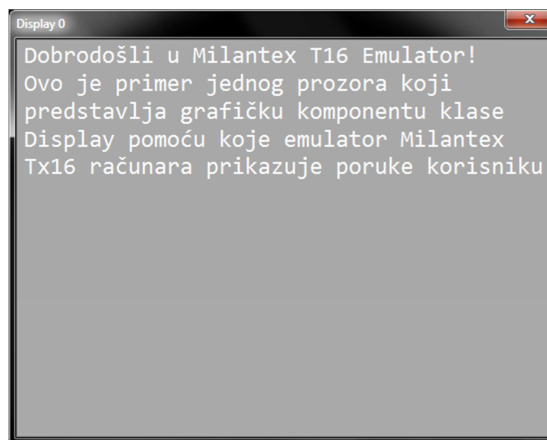
Slika 7 - Crna pozadina



Slika 8 - Plava pozadina



Slika 9 - Cijan zelena pozadina



Slika 10 - Srebrno siva pozadina

3.10 KLASA DISKA (DISK KLASA)

Namena klase Disk je da upravlja upisivanjem sadržaja u datoteku i čitanjem sadržaja iz datoteke koja predstavlja virtualni hard disk priključen na emulator. Svaki virtualni disk ima kapacitet od 64KB i skladišti informacije potrebne programu koji se izvršava u emulatoru.

Klasa Display broji tri javna metoda, konstruktor i destruktor koji su ovde nabrojani:

- Disk(string FileName)
- ~Disk()
- void Write(UInt16 Address, byte Value)
- byte Read(UInt16 Address)
- UInt16 Size()

Pored ovim metoda, klasa ima i jedan zaštićeni element i to:

- string file_name - Predstavlja ime datoteke u kojoj je sadržaj virtualnog diska koji predstavlja instancirani objekat ove klase.

Izvorni kod klase Disk

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.IO;
6
7 namespace Milantex_Tx16_Emulator {
8     public class Disk {
9         protected string file_name;
10
11         public Disk( string FileName ) {
12             file_name = FileName;
13             if ( !File.Exists( file_name ) ) {
14                 byte[] m = new byte[ UInt16.MaxValue ];
15                 for ( int i = 0 ; i < UInt16.MaxValue ; i++ ) m[ i ] = 0xFF;
16                 File.WriteAllBytes( file_name, m );
17             }
18         }
19
20         ~Disk() { IO.RemoveDisk( this ); }
21
22         public void Write( UInt16 Address, byte Value ) {
23             if ( File.Exists( file_name ) ) {
24                 FileStream fs = File.OpenWrite( file_name );
25                 fs.Seek( Address, SeekOrigin.Begin );
26                 fs.WriteByte( Value );
27                 fs.Close();
28             }
29         }
30
31         public byte Read( UInt16 Address ) {
32             byte r = 0;
33             if ( File.Exists( file_name ) ) {
34                 FileStream fs = File.OpenRead( file_name );
35                 fs.Seek( Address, SeekOrigin.Begin );
36                 r = ( byte ) fs.ReadByte();
37                 fs.Close();
38             }
39             return r; ;
40         }
41
42         public UInt16 Size() {
43             return (UInt16) File.Open( file_name, FileMode.Open ).Length;
44         }
45     }
46 }

```

3.11 KLASA ZA EDITOR TEKSTA SA PAMETNIM OBELEŽAVANJEM

Klasa za editor teksta sa pametnim obeležavanjem ili CodeEditor klasa je realizovana uz pomoć *OpenSource* biblioteke i klase koja se zove UrielGuy.SyntaxHighlightingTextBox. U pitanju je besplatna komponenta koja može da se implementira u svaku aplikaciju.

Svrha ove klase je da omogući prikaz koda programa na taj način da programer lakše može da piše programe, jer se određene ključne reči kao što su simbolički nazivi instrukcija, nazivi registara, numeričke vrednosti u raznim formatima, kao i karakteri lakše uočavaju kada su obojeni drugom bojom ili su na neki način naglašeni u editoru koda.

Klasa CodeEditro je nasleđena iz pomenute klase SyntaxHighlightingTextBox, a sadrži sledeće javne *event handler* metode i konstruktore:

- CodeEditor()
- CodeEditor(ref TextBox source_code_editor)
- void sačuvajToolStripMenuItem_Click(object sender, EventArgs e)
- void zatvoriToolStripMenuItem_Click(object sender, EventArgs e)
- void CodeEditor_FormClosing(object sender, FormClosingEventArgs e)
- void EDITOR_TextChanged(object sender, EventArgs e)

Osim ovih metoda, klasa sadrži i dva elementa:

- TextBox SOURCE - Referenca ka editoru teksta u prozoru emulatora koji nema pametno obeležavanje koda i koji nakon dvoklika u svoje polje prikazuje instancu ove klase za pametno obeležavanje koda;
- bool can_close - Status koji naznačava klasi da može da zatvori svoju instancu kada je izmenjeni kod uspešno prosleđen emulatoru za dalju obradu.

Izvorni kod klase CodeEditor

```

1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9 using UrielGuy.SyntaxHighlightingTextBox;
10
11 namespace Milantex_Tx16_Emulator {
12     public partial class CodeEditor : Form {
13         protected TextBox SOURCE;
14         protected bool can_close = false;
15
16         public CodeEditor( ref TextBox source_code_editor ) {
17             InitializeComponent();
18             SOURCE = source_code_editor;
19             InitHighlighter();
20             EDITOR.Text = SOURCE.Text;
21         }
22
23         public CodeEditor() {
24             InitializeComponent();
25             InitHighlighter();
26         }
27
28         protected void InitHighlighter() {
29             int[] tabs = new int[ 32 ];
30             for ( int ix = 0 ; ix < tabs.Length ; ++ix ) tabs[ ix ] = 4 * ( ix + 1 );
31             EDITOR.SelectionHangingIndent = 4;
32             EDITOR.SelectionLength = 4;
33             EDITOR.SelectionStart = 0;
34             EDITOR.SelectionTabs = tabs;

```

```

35
36     MAIN.SendMessage( EDITOR.Handle, MAIN.EM_SETTABSTOPS, 1, new int[] { 16 } );
37
38     EDITOR.Seperators.Add( ' ' );
39     EDITOR.Seperators.Add( '\r' );
40     EDITOR.Seperators.Add( '\n' );
41     EDITOR.Seperators.Add( '\t' );
42     EDITOR.Seperators.Add( ',' );
43     EDITOR.Seperators.Add( '.' );
44     EDITOR.Seperators.Add( '-' );
45     EDITOR.Seperators.Add( '+' );
46     EDITOR.Seperators.Add( '*' );
47     EDITOR.Seperators.Add( '/' );
48     EDITOR.WordWrap = false;
49     EDITOR.ScrollBars = RichTextBoxScrollBars.Both;
50     EDITOR.FilterAutoComplete = true;
51
52
53     string[] blues = "mov,cpy,save,load,ldf,cmp,put,get,push,pull,shl,shr".ToUpper().Split( ',' );
54     string[] Maroons = "nop,caf,inc,dec,ret,jmp,jfc,jfs,call,cfc,cfs,int".ToUpper().Split( ',' );
55     string[] Navys = "add,sub,mul,div,mod,and,or,xor,not".ToUpper().Split( ',' );
56     foreach ( string val in blues )
57         EDITOR.HighlightDescriptors.
58             Add( new HighlightDescriptor( val, Color.Blue, null, DescriptorType.Word,
59                 DescriptorRecognition.WholeWord, true ) );
60     foreach ( string val in Maroons )
61         EDITOR.HighlightDescriptors.
62             Add( new HighlightDescriptor( val, Color.Maroon, null, DescriptorType.Word,
63                 DescriptorRecognition.WholeWord, true ) );
64     foreach ( string val in Navys )
65         EDITOR.HighlightDescriptors.
66             Add( new HighlightDescriptor( val, Color.Navy, null, DescriptorType.Word,
67                 DescriptorRecognition.WholeWord, true ) );
68
69
70     string[] Registers = "a,b,c,d,e,f".ToUpper().Split( ',' );
71     foreach ( string val in Registers )
72         EDITOR.HighlightDescriptors.
73             Add( new HighlightDescriptor( val, Color.RoyalBlue, new Font( EDITOR.Font,
74                 FontStyle.Bold&FontStyle.Underline),DescriptorType.Word,DescriptorRecognition.WholeWord,false));
75
76
77     EDITOR.HighlightDescriptors.Add( new HighlightDescriptor( ";", Color.Green, null,
78         DescriptorType.ToEOL, DescriptorRecognition.WholeWord, false ) );
79
80
81     EDITOR.HighlightDescriptors.Add( new HighlightDescriptor( "0x", Color.DarkCyan,
82         null, DescriptorType.Word, DescriptorRecognition.StartsWith, false ) );
83     EDITOR.HighlightDescriptors.Add( new HighlightDescriptor( "0b", Color.DarkCyan,
84         null, DescriptorType.Word, DescriptorRecognition.StartsWith, false ) );
85     for ( int i = 0 ; i < 10 ; i++ )
86         EDITOR.HighlightDescriptors.
87             Add( new HighlightDescriptor( i.ToString(), Color.DarkCyan, null,
88                 DescriptorType.Word, DescriptorRecognition.StartsWith, false ) );
89
90
91     EDITOR.HighlightDescriptors.Add( new HighlightDescriptor( "@", Color.DarkOrange, null,
92         DescriptorType.Word, DescriptorRecognition.StartsWith, false ) );
93
94
95     EDITOR.HighlightDescriptors.Add( new HighlightDescriptor( "'", "'", Color.DarkGreen, null,
96         DescriptorType.ToCloseToken, DescriptorRecognition.StartsWith, false ) );
97 }
98
99 private void sačuvajToolStripMenuItem_Click( object sender, EventArgs e ) {
100     if ( SOURCE != null ) {
101         SOURCE.Text = EDITOR.Text.Replace("\n", "\r\n").Replace("\r\r", "\r");
102     }
103 }
104
105 private void zatvoriToolStripMenuItem_Click( object sender, EventArgs e ) {
106     this.Close();
107 }
108
109 private void CodeEditor_FormClosing( object sender, FormClosingEventArgs e ) {
110     if ( SOURCE != null ) {
111         if ( SOURCE.Text != EDITOR.Text.Replace( "\n", "\r\n" ).Replace( "\r\r", "\r" ) ) {

```

```

112         System.Windows.Forms.DialogResult r = MessageBox.Show( "Da li želite da sačuvate izmene?",
113 "Izmene", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question, MessageBoxDefaultButton.Button3 );
114         if ( r == System.Windows.Forms.DialogResult.Yes ) {
115             sačuvajToolStripMenuItem_Click( sender, e );
116             can_close = true;
117         }
118         else if ( r == System.Windows.Forms.DialogResult.No ) {
119             can_close = true;
120         }
121         else {
122             can_close = false;
123         }
124     } else can_close = true;
125 } else can_close = true;
126 e.Cancel = !can_close;
127 }
128
129 private void EDITOR_TextChanged( object sender, EventArgs e ) {
130
131 }
132 }
133 }

```

Kod grafičke komponente klase CodeEditor

Klasa CodeEditor ima i grafičku komponentu koja je dizajnirana u Visual Studio 2012 Express Edition Form Designer-u. Ovo je izvorni kod komponente:

```

1 namespace Milantex_Tx16_Emulator {
2     partial class CodeEditor {
3         private System.ComponentModel.IContainer components = null;
4         protected override void Dispose( bool disposing ) {
5             if ( disposing && ( components != null ) ) { components.Dispose(); }
6             base.Dispose( disposing );
7         }
8
9         private void InitializeComponent() {
10             this.MENU = new System.Windows.Forms.MenuStrip();
11             this.kodToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
12             this.sačuvajToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
13             this.toolStripSeparator1 = new System.Windows.Forms.ToolStripSeparator();
14             this.zatvoriToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
15             this.EDITOR = new UrielGuy.SyntaxHighlightingTextBox.SyntaxHighlightingTextBox();
16             this.MENU.SuspendLayout();
17             this.SuspendLayout();
18
19             this.MENU.Items.AddRange( new System.Windows.Forms.ToolStripItem[] {
20                 this.kodToolStripMenuItem } );
21             this.MENU.Location = new System.Drawing.Point( 0, 0 );
22             this.MENU.Name = "MENU";
23             this.MENU.Size = new System.Drawing.Size( 1008, 24 );
24             this.MENU.TabIndex = 0;
25             this.MENU.Text = "menuStrip1";
26
27             this.kodToolStripMenuItem.DropDownItems.AddRange( new System.Windows.Forms.ToolStripItem[] {
28                 this.sačuvajToolStripMenuItem,
29                 this.toolStripSeparator1,
30                 this.zatvoriToolStripMenuItem } );
31             this.kodToolStripMenuItem.Name = "kodToolStripMenuItem";
32             this.kodToolStripMenuItem.Size = new System.Drawing.Size( 65, 20 );
33             this.kodToolStripMenuItem.Text = "&Program";
34
35             this.sačuvajToolStripMenuItem.Name = "sačuvajToolStripMenuItem";
36             this.sačuvajToolStripMenuItem.Size = new System.Drawing.Size( 114, 22 );
37             this.sačuvajToolStripMenuItem.Text = "&Sačuvaj";
38             this.sačuvajToolStripMenuItem.Click +=
39                 new System.EventHandler( this.sačuvajToolStripMenuItem_Click );
40             this.toolStripSeparator1.Name = "toolStripSeparator1";
41             this.toolStripSeparator1.Size = new System.Drawing.Size( 111, 6 );
42
43             this.zatvoriToolStripMenuItem.Name = "zatvoriToolStripMenuItem";
44             this.zatvoriToolStripMenuItem.Size = new System.Drawing.Size( 114, 22 );
45             this.zatvoriToolStripMenuItem.Text = "&Zatvori";
46             this.zatvoriToolStripMenuItem.Click +=
47                 new System.EventHandler( this.zatvoriToolStripMenuItem_Click );

```

```
48     this.EDITOR.AcceptsTab = true;
49     this.EDITOR.CaseSensitive = false;
50     this.EDITOR.Dock = System.Windows.Forms.DockStyle.Fill;
51     this.EDITOR.FilterAutoComplete = false;
52     this.EDITOR.Font = new System.Drawing.Font( "Consolas", 8.25F,
53     System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ( ( byte ) ( 204 ) ) );
54     this.EDITOR.Location = new System.Drawing.Point( 0, 24 );
55     this.EDITOR.MaxUndoRedoSteps = 50;
56     this.EDITOR.Name = "EDITOR";
57     this.EDITOR.ScrollBars = System.Windows.Forms.RichTextBoxScrollBars.ForcedVertical;
58     this.EDITOR.ShowSelectionMargin = true;
59     this.EDITOR.Size = new System.Drawing.Size( 1008, 724 );
60     this.EDITOR.TabIndex = 2;
61     this.EDITOR.Text = "";
62     this.EDITOR.WordWrap = false;
63     this.EDITOR.TextChanged += new System.EventHandler( this.EDITOR_TextChanged );
64
65     this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.None;
66     this.ClientSize = new System.Drawing.Size( 1008, 748 );
67     this.Controls.Add( this.EDITOR );
68     this.Controls.Add( this.MENU );
69     this.DoubleBuffered = true;
70     this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
71     this.MainMenuStrip = this.MENU;
72     this.MaximizeBox = false;
73     this.Name = "CodeEditor";
74     this.ShowIcon = false;
75     this.Text = "Editor";
76     this.FormClosing += new
77     System.Windows.Forms.FormClosingEventHandler( this.CodeEditor_FormClosing );
78     this.MENU.ResumeLayout( false ); this.MENU.PerformLayout();
79     this.ResumeLayout( false );
80     this.PerformLayout();
81
82 }
83
84 private System.Windows.Forms.MenuStrip MENU;
85 private System.Windows.Forms.ToolStripMenuItem kodToolStripMenuItem;
86 private System.Windows.Forms.ToolStripMenuItem sačuvajToolStripMenuItem;
87 private System.Windows.Forms.ToolStripSeparator toolStripSeparator1;
88 private System.Windows.Forms.ToolStripMenuItem zatvoriToolStripMenuItem;
89 private UrielGuy.SyntaxHighlightingTextBox.SyntaxHighlightingTextBox EDITOR;
90 }
91 }
```

```

Editor
Program
@init:  MOV A 0           ;; Podesi displej
        MOV C 0         ;; Podesava stanje registra C na 0
        CALL @cls;      ;; Ocisti ekran
        MOV B 0         ;; Podesava stanje registra B na 0 - koristi se kao pokazivac pozicije za pocetak citanja sa diska
        CALL @read_disk; ;; Pozivi ciranje stringa sa diska
        JMP @s;         ;; Skace bezuslovno na labelu @s

@s:      LDF 8           ;; Ucitava flag IN ( da li u INPUT BUFFER-u ima necega )
        CFS @gc;        ;; Ako ima, pozovi funkciju na labeli @gc
        CFC @time;      ;; Ako nema, pozovi funkciju za prikaz vremena i datuma u dnu ekrana
        JMP @s;         ;; Kada uradi jedno ili drugo, vrati se na labelu @s i tako u krug ( GLAVNA petlja programa )

@gc:     PUT F           ;; Smeta trenutne vrednosti registara u stek
        PUT B
        PUT D
        INT 0xA3        ;; Izvršava hardverski interapt koji uzima trenutnu slobodnu poziciju displeja
        MOV F 41        ;; Smeta vrednost 41 u D
        CAF             ;; Cisti flagove
        CMP F D         ;; Poredi 41 iz F sa izvucenom pozicijom u D
        LDF 1           ;; Ucitava stanje flaga VECE OD za proveru
        CFS @go_2_strt; ;; Ako je flag setovan, zove funkciju koja vraća poziciju (cursor) displeja na pocetak drugog reda!
        GET D           ;; Vraca sacuvane vrednosti registara iz steka nazad u njih
        GET B
        GET F

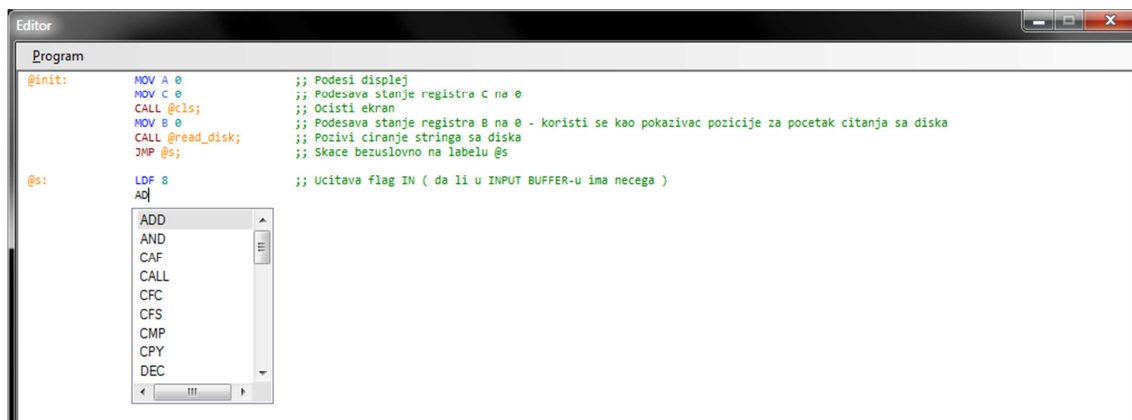
        INT 0xE1        ;; Iscitava sledecu vrednost iz INPUT BUFFER-a
        PUT A           ;; Privremeno cuva vrednost koja je bila u A u stek
        MOV A 239       ;; Smeta vrednost 239 ( backspace karakter ) u A
        CAF             ;; Cisti flagove
        CMP D A         ;; Poredi iscitani karakter iz INPUT BUFFER-a sa backspace karakterom
        LDF 0           ;; Ucitava za testiranje flag EQUAL (siti su)
        JFC @p_ch;      ;; Ako nisu isti karakteri, tj. nije pritisnut backspace, skoci na ispis ucitanog karaktera
        PUT D           ;; U suprotnom sacuvaj jos nekoliko vrednosti registara u stek
        PUT A
        MOV A 0         ;; Postavi vrednost A na 0
        INT 0xA3        ;; Pozovi hardverski interapt koji smesta trenutnu poziciju na displeju u registar D
        DEC D           ;; Pomeri D unazad
        PUT B           ;; Sacuvaj stanje registra B
        CPY B D         ;; Kopiraj stanje registra D u B
        INT 0xA4        ;; Pozovi hardverski interapt koji pomera poziciju kursora na displeju na vrednost u B
        PUT C           ;; Sacuvaj stanje registra C
        MOV C ' '       ;; Smeti space vrednost u C
        CALL @pchar;    ;; Pozovi funkciju koja ispisuje karakter iz C
        CAF             ;; Ocisti flagove
        MOV C 0xFFFF    ;; Smeti vrednost FFFF u C
        CMP D C         ;; Poredi vrednost D i C
        LDF 0           ;; Vidi da li su isti
        JFS @s_1;       ;; Ako jesu, skoci na stavu @s_1 odmah da bi se vratila stanja iz registra koja su promenjena
        INT 0xA4        ;; A ako nisu, postavi poziciju kursora na vrednost iz B

@s_1:    GET C           ;; Smeta trenutne vrednosti registara u stek
        GET B
        GET A
        GET D
        GET A

```

Slika 11 - CodeEditor klasa sa primerom označenog izvornog koda programa u Assembly jeziku

Klasa CodeEditor uključuje u sebi funkcionalnost još jedne klase koju u ovom radu neću detaljno opisivati, jer je ona jedan od sastavnih delova klase iz koje je nasleđena ova iz koje se instancira editor koda sa pametnim označavanjem, a u pitanju je prozor sa spiskom simboličkih naziva instrukcija koje su podržane u setu instrukcija procesora arhitekture Milantex Tx16 računara i iz koje korisnik može tokom kucanja da izabere instrukciju koju želi da mu editor upiše u kod na ispravan način.



Slika 12 - CodeEditor klasa sa delom izvornog koga i sa prozorom sa listom instrukcija koje su dostupne

3.12 KLASA ZA PREVOĐENJE IZVORNOG KODA (INSTRUCTIONCOMPILER KLASA)

Klasa InstructionCompiler je apstraktna klasa čija je namena da izvorni kod prevodi red po red u bajt kôd u skladu sa definicijom seta instrukcija. Klasa ima jeda osnovni metod koji koriste instancirani objekti drugih klasa tokom izvršavanja programa i to u postupku prevođenja koda, pre izvršavanja u emulatoru i to:

- `byte[] CompileProgramLine( string line )`

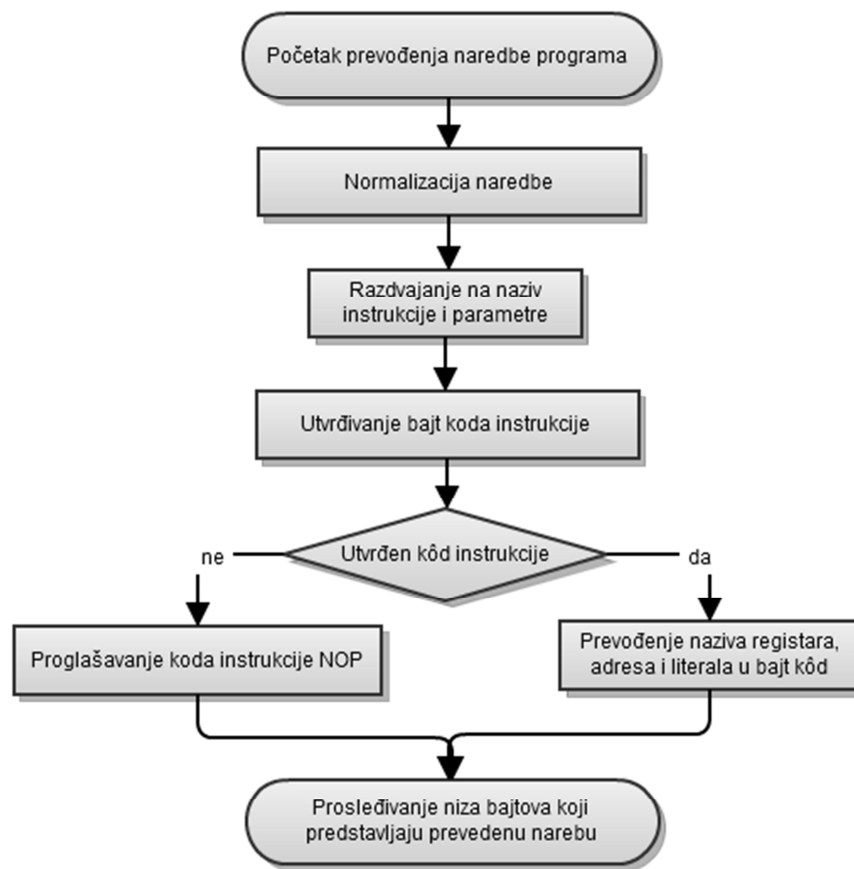
Metod za obradu parametara naredbe uzima liniju izvornog koda programa iz editora i nakon normalizacije, uklanjanja viška belina, deli instrukciju na elemente i to na:

- Naziv instrukcije (simbolički zapis instrukcije za procesor)
- Parametre instrukcije koji prolaze posebnu obradu

Ostali metodi ove klase su:

- `byte GetInstructionByteCode( string Instruction )`
- `string ByteArrayToHex( byte[] ba )`
- `byte[] BinaryRepresentationStringToByteArray( string input )`
- `byte[] GetLiteralValue( string v )`
- `byte RegisterNumber( string RegName )`
- `byte[] CompileTheRest( byte instruction_bytecode, string[] arguments )`

Proces prevođenja pojedinačne naredbe programa je predstavljen algoritmom:



Slika 13 - Algoritam procesa prevođenja jedne naredbe programa

Izvorni kod klase InstructionCompiler

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5
6 namespace Milantex_Tx16_Emulator {
7     public abstract class InstructionCompiler {
8         public static byte[] CompileProgramLine( string line ) {
9             string[] parts;
10            string instruction;
11
12            line = line.Trim().Replace( " ", " ").Replace( " ", " ");
13            parts = line.Split( ' ' );
14
15            if ( parts.Length < 1 ) return new byte[] { 0x00 };
16            else instruction = parts[ 0 ];
17            byte[] program_line_bytecode =
18            CompileTheRest( GetInstructionByteCode( instruction.Trim().ToUpper() ), parts );
19            return program_line_bytecode;
20        }
21
22        protected static byte GetInstructionByteCode( string Instruction ) {
23            byte _return = 0x00;
24            switch ( Instruction ) {
25                case "NOP" : _return = 0x00; break;
26                case "MOV" : _return = 0x01; break;
27                case "INC" : _return = 0x02; break;
28                case "DEC" : _return = 0x03; break;
29                case "CPY" : _return = 0x04; break;
30                case "SAVE" : _return = 0x05; break;
31                case "LOAD" : _return = 0x06; break;
32                case "INT" : _return = 0x07; break;
33                case "SHL" : _return = 0x08; break;
34                case "SHR" : _return = 0x09; break;
35                case "RTL" : _return = 0x0A; break;
36                case "RTR" : _return = 0x0B; break;
37                case "JMP" : _return = 0x0C; break;
38                case "JFS" : _return = 0x0D; break;
39                case "JFC" : _return = 0x0E; break;
40                case "LDF" : _return = 0x0F; break;
41                case "CAF" : _return = 0x10; break;
42                case "CLF" : _return = 0x11; break;
43                case "CMP" : _return = 0x12; break;
44                case "AND" : _return = 0x13; break;
45                case "OR" : _return = 0x14; break;
46                case "XOR" : _return = 0x15; break;
47                case "NOT" : _return = 0x16; break;
48                case "ADD" : _return = 0x17; break;
49                case "SUB" : _return = 0x18; break;
50                case "CALL" : _return = 0x19; break;
51                case "RET" : _return = 0x1A; break;
52                case "PUSH" : _return = 0x1B; break;
53                case "PULL" : _return = 0x1C; break;
54                case "PUT" : _return = 0x1D; break;
55                case "GET" : _return = 0x1E; break;
56                case "MUL" : _return = 0x1F; break;
57                case "DIV" : _return = 0x20; break;
58                case "MOD" : _return = 0x21; break;
59                case "CFS" : _return = 0x22; break;
60                case "CFC" : _return = 0x23; break;
61            }
62            return _return;
63        }
64
65        public static string ByteArrayToHex( byte[] ba ) {
66            StringBuilder sb = new StringBuilder(ba.Length * 2);
67            foreach (byte b in ba) sb.AppendFormat("{0:x2}", b);
68            return sb.ToString().ToUpper() + " ";
69        }
70
71        protected static byte[] BinaryRepresentationStringToByteArray( string input ) {
72            int numOfBytes = input.Length / 8;
73            byte[] bytes = new byte[ numOfBytes ];
74            for(int i=0;i<numOfBytes;++i) bytes[ i ] = Convert.ToByte( input.Substring( 8 * i, 8 ), 2 );

```

```

75     return bytes;
76 }
77
78 protected static byte[] GetLiteralValue( string v ) {
79     byte[] value = new byte[1];
80     ushort V = 0;
81
82     v = v.Trim();
83
84     if ( v.ToUpper().StartsWith( "0X" ) ) {
85         V = ushort.Parse( v.Substring( 2 ), System.Globalization.NumberStyles.HexNumber );
86         value = BitConverter.GetBytes( V );
87     }
88     else if ( v.ToUpper().StartsWith( "0B" ) ) {
89         value = BinaryRepresentationStringToByteArray( v.Substring( 2 ) );
90     } else if ( v[0] == '\\' && v[2] == '\\' ) {
91         char c = v[ 1 ];
92         V = ( byte ) Memory.CharacterMap.IndexOf( c );
93         value = BitConverter.GetBytes( V );
94     }
95     else if ( v.ToUpper() == "SPACE_CHARACTER" ) {
96         V = ( byte ) Memory.CharacterMap.IndexOf( ' ' );
97         value = BitConverter.GetBytes( V );
98     }
99     else {
100         value = BitConverter.GetBytes( ushort.Parse( v ) );
101     }
102
103     return value;
104 }
105
106 protected static byte RegisterNumber( string RegName ) {
107     byte reg = 0x00;
108     switch ( RegName.Trim().ToUpper() ) {
109         case "A": reg = 0x01; break;
110         case "B": reg = 0x02; break;
111         case "C": reg = 0x03; break;
112         case "D": reg = 0x04; break;
113         case "E": reg = 0x05; break;
114         case "F": reg = 0x06; break;
115         default : reg = 0xFF; break;
116     }
117     return reg;
118 }
119
120 protected static byte[] CompileTheRest( byte instruction_bytecode, string[] arguments ) {
121     byte[] for_return = new byte[ 1 ];
122     for_return[ 0 ] = instruction_bytecode;
123     byte[] value;
124
125     switch ( instruction_bytecode ) {
126         case 0x00:
127         case 0x10:
128         case 0x11:
129         case 0x1A:
130             for_return = new byte[ 1 ];
131             for_return[ 0 ] = instruction_bytecode;
132             break;
133         case 0x01:
134             for_return = new byte[ 4 ];
135             for_return[ 0 ] = instruction_bytecode;
136             for_return[ 1 ] = RegisterNumber( arguments[ 1 ] );
137             value = GetLiteralValue( arguments[ 2 ] );
138             for_return[ 2 ] = value[ 1 ];
139             for_return[ 3 ] = value[ 0 ];
140             break;
141         case 0x02:
142         case 0x03:
143         case 0x16:
144         case 0x1B:
145         case 0x1C:
146         case 0x1D:
147         case 0x1E:
148             for_return = new byte[ 2 ];
149             for_return[ 0 ] = instruction_bytecode;
150             for_return[ 1 ] = RegisterNumber( arguments[ 1 ] );
151             break;

```

```

152         case 0x04:
153         case 0x05:
154         case 0x06:
155         case 0x12:
156         case 0x13:
157         case 0x14:
158         case 0x15:
159         case 0x17:
160         case 0x18:
161         case 0x1F:
162         case 0x20:
163         case 0x21:
164             for_return = new byte[ 3 ];
165             for_return[ 0 ] = instruction_bytecode;
166             for_return[ 1 ] = RegisterNumber( arguments[ 1 ].Trim() );
167             for_return[ 2 ] = RegisterNumber( arguments[ 2 ].Trim() );
168             break;
169         case 0x07:
170             for_return = new byte[ 2 ];
171             for_return[ 0 ] = instruction_bytecode;
172             for_return[ 1 ] = ( byte ) GetLiteralValue( arguments[ 1 ] )[ 0 ];
173             break;
174         case 0x08:
175         case 0x09:
176         case 0x0A:
177         case 0x0B:
178             for_return = new byte[ 3 ];
179             for_return[ 0 ] = instruction_bytecode;
180             for_return[ 1 ] = RegisterNumber( arguments[ 1 ] );
181             for_return[ 2 ] = (byte)(Math.Min(GetLiteralValue(arguments[2])[0],(byte)16));
182             break;
183         case 0x0C:
184         case 0x0D:
185         case 0x0E:
186         case 0x19:
187         case 0x22:
188         case 0x23:
189             for_return = new byte[ 3 ];
190             for_return[ 0 ] = instruction_bytecode;
191             value = GetLiteralValue( arguments[ 1 ] );
192             for_return[ 1 ] = value[ 1 ];
193             for_return[ 2 ] = value[ 0 ];
194             break;
195         case 0x0F:
196             for_return = new byte[ 2 ];
197             for_return[ 0 ] = instruction_bytecode;
198             value = GetLiteralValue( arguments[ 1 ] );
199             for_return[ 1 ] = value[ 0 ];
200             break;
201     }
202     return for_return;
203 }
204
205 }
206 }

```

3.13 GLAVNA KLASA PROGRAMA (KLASA MAIN)

Klasa MAIN je glavna klasa programa koja inicijalizuje sve grafičke komponente glavnog prozora, instancira objekte klase za registre i memoriju i vezuje *event handler*-e za svako kontrolno dugme na glavnom prozoru emulatora i za pritiskanje tastera na tastaturi u polju emuliranja virtualne tastature.

Klasa MAIN sadrži samo jedan javni metod i to je konstruktor klase. Pored konstruktora, sadrži 24 privatna metoda i to:

- void DisplayMemoryChangesCheckbox_CheckedChanged(object s, EventArgs e)
- void DisplayRegisterChangesCheckbox_CheckedChanged(object s, EventArgs e)
- void Compile_Button_Click(object sender, EventArgs e)
- void StepButton_Click(object sender, EventArgs e)
- void PlayStopButton_Click(object sender, EventArgs e)
- void Tajmer_Tick(object sender, EventArgs e)
- void MAIN_FormClosing(object sender, FormClosingEventArgs e)
- void SPEED_ValueChanged(object sender, EventArgs e)
- void MAIN_Load(object sender, EventArgs e)
- void OpenDisplayButton_Click(object sender, EventArgs e)
- void AttachDisk_Click(object sender, EventArgs e)
- void DiskDisplay_ItemActivate(object sender, EventArgs e)
- void DiskDisplayContextMenu_ItemClicked(object s, ToolStripItemClickedEventArgs e)
- void DiskDisplayContextMenu_Opening(object sender, CancelEventArgs e)
- void JedanSekundTajmer_Tick(object sender, EventArgs e)
- void INPUT_KeyPress(object sender, KeyPressEventArgs e)
- void INPUT_KeyUp(object sender, KeyEventArgs e)
- void INPUT_CHAR_KeyPress(object sender, KeyPressEventArgs e)
- void INPUT_CHAR_KeyUp(object sender, KeyEventArgs e)
- void Flag_Label_Click(object sender, EventArgs e)
- void INPUT_BUFFER_KeyPress(object sender, KeyPressEventArgs e)
- void MAIN_KeyUp(object sender, KeyEventArgs e)
- void CLEAR_SOURCE_Click(object sender, EventArgs e)
- void SOURCE_DoubleClick(object sender, EventArgs e)
- static void RunnerTask()

Ostali elementi glavne klase emulatora su:

- const int EM_SETTABSTOPS = 0x00CB;
- static bool running = false;
- static bool exiting_program = false;
- CodeEditor ce;

Kroz klasu MAIN je omogućeno da se u više niti paralelno izvršava proces emulacije i proces monitoringa, tako da se instrukcije izvršavaju nezavisno od toga da li je grafički korisnički interfejs uspešno osvežen u cilju prikazivanja novonastalog stanja. Ukoliko se instrukcija izvrši, a do trenutka kada treba da počne nova instrukcija da se obrađuje, korisnički interfejs nije osvežen, instrukcija će se ipak izvršiti, a osvežavanje interfejsa će biti prekinuto sve do kraja izvršavanja nove instrukcije. Na taj način je sprečeno kašnjenje.

Izvorni kod klase MAIN

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.InteropServices;
4 using System.ComponentModel;
5 using System.Diagnostics;
6 using System.Data;
7 using System.Drawing;
8 using System.Linq;
9 using System.Text;
10 using System.Windows.Forms;
11 using System.Threading;
12
13 namespace Milantex_Tx16_Emulator {
14     public partial class MAIN : Form {
15         private void DisplayMemoryChangesCheckbox_CheckedChanged( object sender, EventArgs e ) {
16             bool state = running;
17             running = false;
18             Memory.DisplayMemoryChanges = DisplayMemoryChangesCheckbox.Checked;
19             running = state;
20         }
21
22         private void DisplayRegisterChangesCheckbox_CheckedChanged( object sender, EventArgs e ) {
23             bool state = running;
24             running = false;
25             if ( state ) {
26                 Memory._RA.SetValue( Memory.RA );
27                 Memory._RB.SetValue( Memory.RB );
28                 Memory._RC.SetValue( Memory.RC );
29                 Memory._RD.SetValue( Memory.RD );
30                 Memory._RE.SetValue( Memory.RE );
31                 Memory._RF.SetValue( Memory.RF );
32                 Memory._FLAGS.SetValue( Memory.FLAGS );
33             }
34             Memory.DisplayRegistarChanges = DisplayRegisterChangesCheckbox.Checked;
35             running = state;
36         }
37
38         public const int EM_SETTABSTOPS = 0x00CB;
39         [DllImport( "User32.dll", CharSet = CharSet.Auto )]
40         public static extern IntPtr SendMessage( IntPtr h, int msg, int wParam, int[] lParam );
41
42         public MAIN() {
43             InitializeComponent();
44
45             SendMessage( SOURCE.Handle, EM_SETTABSTOPS, 1, new int[] { 16 } );
46
47             IO.INPUT_BUFFER_TEXTBOX = INPUT_BUFFER;
48             Memory.MEMORY_DISPLAY = MEMORY_DISPLAY;
49             Memory._RA = _RA;
50             Memory._RB = _RB;
51             Memory._RC = _RC;
52             Memory._RD = _RD;
53             Memory._RE = _RE;
54             Memory._RF = _RF;
55             Memory._FLAGS = _FLAGS;
56             Memory.DisplayMemoryChanges = DisplayMemoryChangesCheckbox.Checked;
57             Memory.DisplayRegistarChanges = DisplayRegisterChangesCheckbox.Checked;
58             Memory.InitMemory();
59
60             SPEED_ValueChanged( this, null );
61         }
62
63         private void Compile_Button_Click( object sender, EventArgs e ) {
64             running = false;
65             PlayStopButton.Text = ( !running ? "▶" : "■" ) + " [F10]";
66
67             BYTECODE.Text = "";
68
69             string _program = SOURCE.Text;
70             _program = SOURCE.Text.Replace( " " , "SPACE_CHARACTER" );
71             string[] lines=SOURCE.Text.Replace("\r","").Replace(" ", " ").Replace(" ", " ").Split('\n');
72
73             int location = 0;
74             Memory.byte_locations = new ushort[ lines.Length ];

```

```

75     foreach ( String line in lines ) {
76         string lineA = line.Trim();
77         if ( lineA.Trim() == "" ) lineA = "NOP";
78         if ( lineA.Contains( "@" ) && lineA.Contains( ":" ) ) {
79             string label=lineA.Substring(lineA.IndexOf('@'),lineA.IndexOf(':')-lineA.IndexOf('@'));
80             _program = _program.Replace( label + ":", "" );
81             _program = _program.Replace( label + ";;", "0x" + location.ToString( "X4" ) );
82         }
83         location++;
84     }
85
86     lines = _program.Split( '\n' );
87
88     Memory.InitMemory();
89
90     ushort instruction_count = 0;
91     foreach ( string Line in lines ) {
92         List<byte> list = new List<byte>();
93         string line = Line;
94         if ( line.Contains( ";;" ) ) line = line.Substring( 0, line.IndexOf( ";;" ) ).Trim();
95         byte[] ibc = InstructionCompiler.CompileProgramLine( line );
96         list.AddRange( Memory.PROGRAM );
97         list.AddRange( ibc );
98         Memory.byte_locations[ instruction_count ] = ( ushort ) ( Memory.PROGRAM.Length - 1 );
99         BYTECODE.Text += InstructionCompiler.ByteArrayToHex( ibc );
100        Memory.PROGRAM = list.ToArray();
101        instruction_count++;
102    }
103
104    int size = BYTECODE.Text.Replace( " ", "" ).Length / 2;
105    PreostaloProgramskogProstora.Text="Iskorišćeno "+size+
106    + " B od 65536 B programskog prostora. "+Math.Round(((double)size/65536,2)+"%"; }
107
108    private void StepButton_Click( object sender, EventArgs e ) {
109        running = false;
110        CPU.ExecuteNextInstruction();
111    }
112
113    Thread runner;
114    protected static bool running = false;
115    protected static bool exiting_program = false;
116
117    protected static void RunnerTask() {
118        while ( ! exiting_program ) {
119            while ( running ) {
120                CPU.ExecuteNextInstruction();
121                if ( CPU.sleep_time > 0 ) Thread.Sleep( CPU.sleep_time );
122            }
123        }
124    }
125
126    private void PlayStopButton_Click( object sender, EventArgs e ) {
127        if ( runner == null ) {
128            runner = new Thread( new ThreadStart( RunnerTask ) );
129            runner.Start();
130        }
131
132        PlayStopButton.Text = ( running ? "▶" : "■" ) + " [F10]";
133        running = !running;
134    }
135
136    private void Tajmer_Tick( object sender, EventArgs e ) {
137    }
138
139    private void MAIN_FormClosing( object sender, FormClosingEventArgs e ) {
140        if ( runner != null ) {
141            exiting_program = true;
142            runner.Abort();
143            runner = null;
144        }
145    }
146
147    private void SPEED_ValueChanged( object sender, EventArgs e ) {
148        int val = ( SPEED.Maximum - 10 ) - SPEED.Value;
149        if ( val < 0 ) val = 0;
150        SPEED_DISPLAY.Text = "čekaj " + val.ToString() + "ms između instrukcija.";
151        CPU.sleep_time = val;

```

```

152     }
153
154     private void MAIN_Load( object sender, EventArgs e ) {
155
156     }
157
158     private void OpenDisplayButton_Click( object sender, EventArgs e ) {
159         IO.AttachDisplay();
160     }
161
162     private void AttachDisk_Click( object sender, EventArgs e ) {
163         if ( IO.AddachDisk() ) {
164             DiskDisplay.Items.Add("Disk"+(IO.NumberOfDisks-1)+".mdsk","Disk "+
165                 (IO.NumberOfDisks-1).ToString()+" (64KB)",0); }
166     }
167
168     private void DiskDisplay_ItemActivate( object sender, EventArgs e ) {
169         ListView v = ( ListView ) sender;
170         string file_name = v.SelectedItems[ 0 ].Name;
171         Process.Start( file_name );
172     }
173
174     private void DiskDisplayContextMenu_ItemClicked(object s,ToolStripItemClickedEventArgs e) {
175         if ( IO.NumberOfDisks > 0 ) {
176             if ( e.ClickedItem == DeleteLastAttachedDisk ) {
177                 if ( IO.RemoveDisk( IO.NumberOfDisks - 1 ) ) {
178                     DiskDisplay.Items.RemoveAt( DiskDisplay.Items.Count - 1 );
179                 }
180             }
181         }
182     }
183
184     private void DiskDisplayContextMenu_Opening( object sender, CancelEventArgs e ) {
185         if ( IO.NumberOfDisks > 0 ) {
186             DeleteLastAttachedDisk.Enabled = true;
187         }
188         else {
189             DeleteLastAttachedDisk.Enabled = false;
190         }
191     }
192
193     private void JedanSekundTajmer_Tick( object sender, EventArgs e ) {
194         SYSTEM_TIME.Text = DateTime.Now.ToLongTimeString();
195     }
196
197     private void INPUT_KeyPress( object sender, KeyPressEventArgs e ) {
198         if ( !(
199             Char.IsDigit( e.KeyChar ) ||
200             e.KeyChar=='A' || e.KeyChar=='B' || e.KeyChar=='C' || e.KeyChar=='D' || e.KeyChar=='E' || e.KeyChar=='F' ||
201             e.KeyChar=='a' || e.KeyChar=='b' || e.KeyChar=='c' || e.KeyChar=='d' || e.KeyChar=='e' || e.KeyChar=='f'
202         ) ) {
203             if ( !(e.KeyChar==13 || e.KeyChar==8) )
204                 MessageBox.Show( "Broj mora da bude u heksadecimalnom zapisu!\r\n\r\nKey code: "+(int)e.KeyChar);
205             e.Handled = true;
206         }
207         else {
208             if ( INPUT.Text.Length / 3 > 255 ) {
209                 e.Handled = true;
210             }
211         }
212     }
213
214     private void INPUT_KeyUp( object sender, KeyEventArgs e ) {
215         if ( e.KeyCode == Keys.Enter && INPUT.Text.Length > 0 ) {
216             string t = INPUT.Text.ToUpper().Trim();
217             if ( t.StartsWith( "0x" ) ) t = t.Substring( 2 );
218             byte c = byte.Parse( t, System.Globalization.NumberStyles.HexNumber );
219             IO.PutIntoInputBuffer( c );
220             INPUT_BUFFER.AppendText( c.ToString( "X2" ) + " " );
221             INPUT.Text = "";
222             e.Handled = true;
223         }
224         else if ( e.KeyCode == Keys.Back && INPUT.Text.Length > 0 ) {
225             INPUT.Text = INPUT.Text.Substring( 0, INPUT.Text.Length - 1 );
226             INPUT.SelectionStart = INPUT.Text.Length;
227         }
228     }

```

```

229
230     private void INPUT_CHAR_KeyPress( object sender, KeyPressEventArgs e ) {
231         if ( Memory.CharacterMap.IndexOf( e.KeyChar ) >= 0 ) e.Handled = false;
232         else e.Handled = true;
233     }
234
235     private void INPUT_CHAR_KeyUp( object sender, KeyEventArgs e ) {
236         if ( e.KeyCode == Keys.Enter && INPUT_CHAR.Text.Length > 0 ) {
237             string t = INPUT_CHAR.Text.Trim();
238             byte c = ( byte ) Memory.CharacterMap.IndexOf( INPUT_CHAR.Text[0] );
239             IO.PutIntoInputBuffer( c );
240             INPUT_BUFFER.AppendText( c.ToString( "X2" ) + " " );
241             INPUT_CHAR.Text = "";
242             e.Handled = true;
243         }
244         else if ( e.KeyCode == Keys.Enter && INPUT_CHAR.Text.Length == 0 ) {
245             byte c = ( byte ) Memory.CharacterMap.IndexOf( "\n" );
246             IO.PutIntoInputBuffer( c );
247         }
248         else if ( e.KeyCode == Keys.Back && INPUT_CHAR.Text.Length > 0 ) {
249             INPUT_CHAR.Text = INPUT_CHAR.Text.Substring( 0, INPUT_CHAR.Text.Length - 1 );
250             INPUT_CHAR.SelectionStart = INPUT_CHAR.Text.Length;
251         }
252     }
253
254     private void Flag_Label_Click( object sender, EventArgs e ) {
255         bool state = running;
256         running = false;
257         Label l = ( Label ) sender;
258         MessageBox.Show( l.Tag.ToString(), "Objašnjenje flag-a", MessageBoxButtons.OK,
259             MessageBoxIcon.Information ); running = state;
260     }
261
262     private void INPUT_BUFFER_KeyPress( object sender, KeyPressEventArgs e ) {
263         if ( Memory.CharacterMap.IndexOf( e.KeyChar ) >= 0 ) {
264             byte c = ( byte ) Memory.CharacterMap.IndexOf( e.KeyChar );
265             IO.PutIntoInputBuffer( c );
266         }
267         e.Handled = true;
268     }
269
270     private void MAIN_KeyUp( object sender, KeyEventArgs e ) {
271         if ( e.KeyCode == Keys.F5 ) {
272             Compile_Button_Click( sender, e );
273         }
274         else if ( e.KeyCode == Keys.F9 ) {
275             StepButton_Click( sender, e );
276         }
277         else if ( e.KeyCode == Keys.F10 ) {
278             PlayStopButton_Click( sender, e );
279         }
280     }
281
282     private void CLEAR_SOURCE_Click( object sender, EventArgs e ) {
283         running = false;
284         SOURCE.Text = "";
285         PlayStopButton.Text = ( !running ? "▶" : "■" ) + " [F10]";
286     }
287
288     protected CodeEditor ce;
289     private void SOURCE_DoubleClick( object sender, EventArgs e ) {
290         if ( ce == null || ce.IsDisposed ) {
291             ce = new CodeEditor( ref SOURCE );
292             ce.CreateControl();
293             ce.Show();
294         } else if ( !ce.IsDisposed ) {
295             ce.Show();
296         }
297     }
298 }
299 }

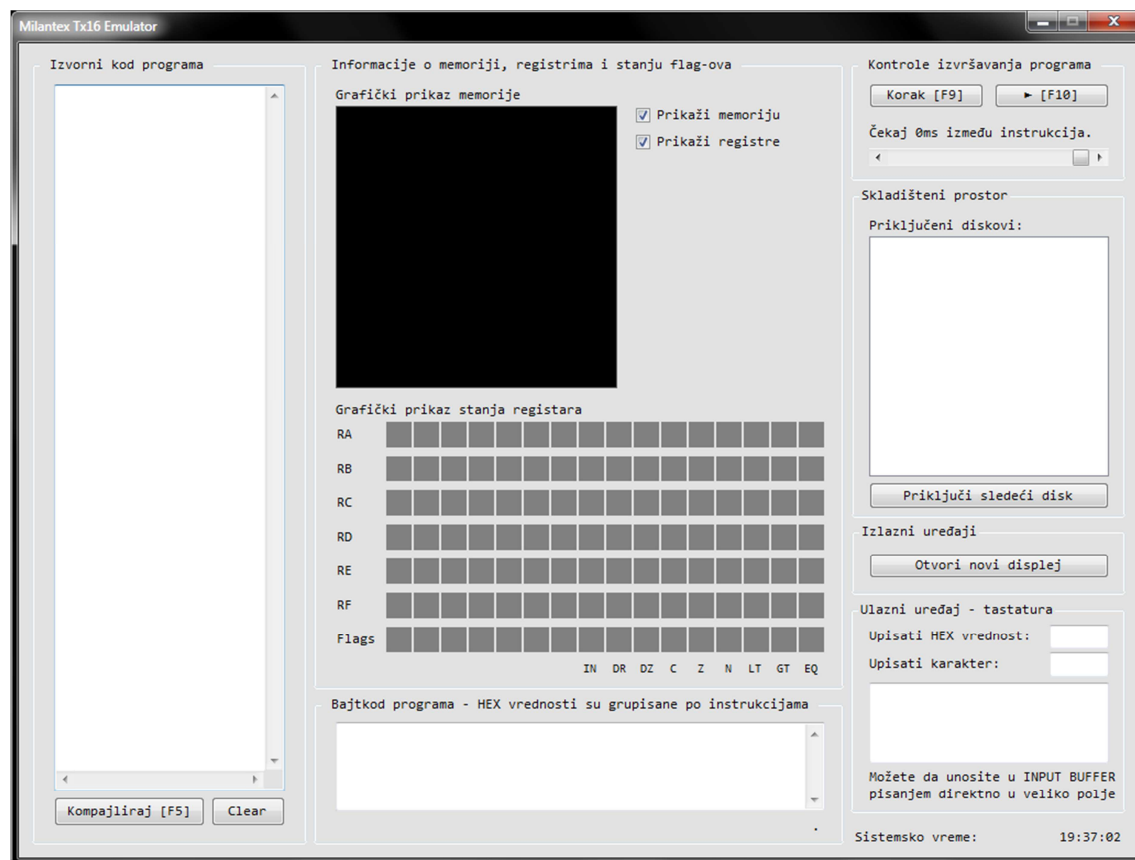
```

4 OPIS RADA EMULATORA MILANTEX Tx16 RAČUNARA

Emulator arhitekture Milantex Tx16 računara je Windows aplikacija pisana u programskom jeziku C# uz .NET *framework* biblioteke. Korišćeno je razvojno okruženje Visual Studio 2010 Express Edition. Program je razvijen uz verziju 3.5 .NET *framework*-a.

4.1 KORISNIČKI INTERFEJS EMULATORA

Korisnički interfejs emulatora je tako osmišljen da kompletan centralni prozor sadrži najbitnije elemente za pisanje programa, kontrole za njegovo izvršavanje, kao i za praćenje stanja registara, memorije i kontrolu priključivanja diskova i displeja.



Slika 14 - Prikaz osnovnog prozora emulatora arhitekture Milantex Tx16 računara

Korisnički interfejs emulatora je tako koncipiran da se svi potrebni elementi, za pisanje programa, njegovo prevođenje u bajt kôd, pokretanje i praćenje rada računara tokom izvršavanja tog programa, nalaze na jednom mestu, bez potrebe da se ulazi u menije, podmenije i da se otvaraju dodatni prozori kako bi se pratio rad emuliranog računara.

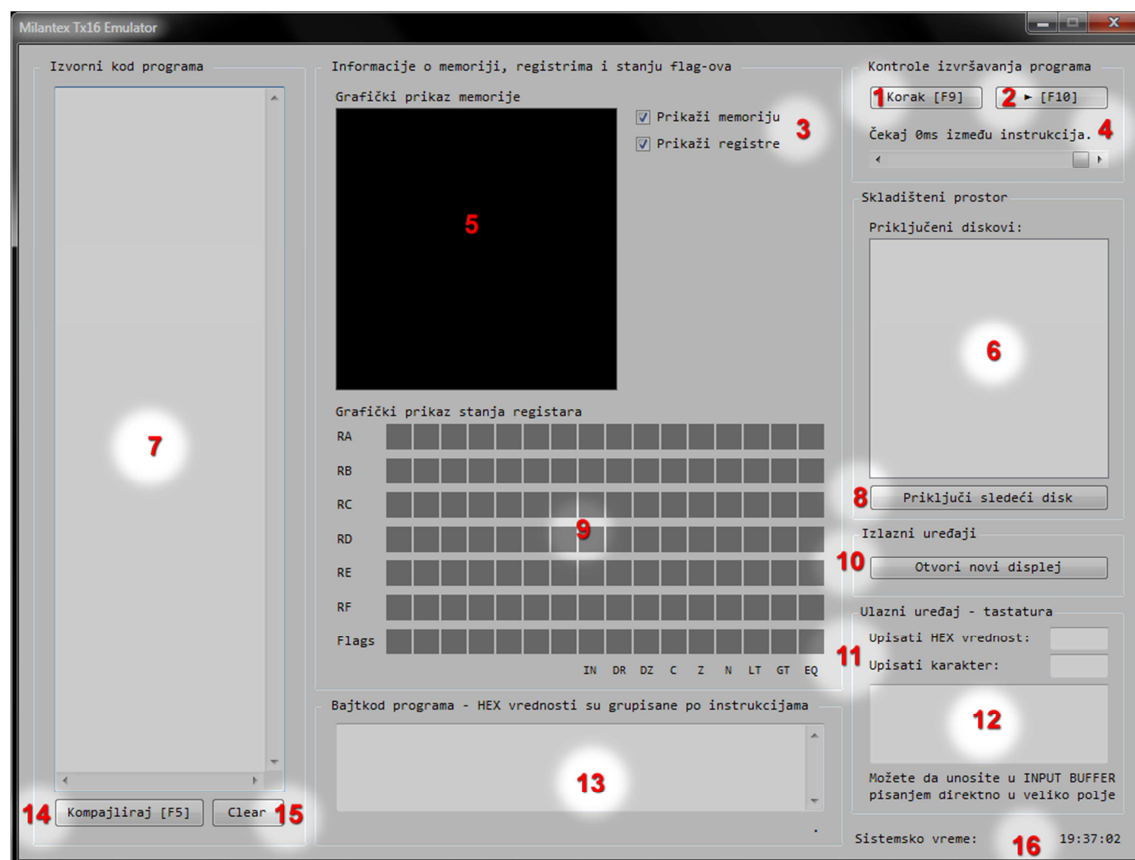
Raspored komponenata korisničkog interfejsa

Glavni prozor emulatora nema menije niti statusnu liniju, već je podeljen u sedam grupa elemenata:

- Dve grupe elemenata obuhvataju polje za unos koda programa i prikaz bajt koda.
- Centralna i najveća grupa prikazuje stanja memorije i registara tokom izvršavanja.
- Jedna grupa obuhvata kontrole izvršavanja i brzine emulacije.
- Poslednje tri grupe, sa leve strane prozora obuhvataju komponente ulaza i izlaza.

Komponente korisničkog interfejsa i njihove lokacije u glavnom prozoru

Kao što je napisano, komponente za kontrolu, unos i monitoring izvršavanja emulatora su raspoređene po grupama elemenata korisničkog interfejsa. Ovo je pregled najvažnijih komponenata za funkcionalnost emulatora.



Slika 15 - Prikaz pozicija najvažnijih komponenata glavnog prozora emulatora

Opis najvažnijih komponenata nabrojanih u prikazu:

1. Dugme za izvršavanje jedne po jedne instrukcije iz koda programa;
2. Dugme za pokretanje i prekidanje procesa emuliranja;
3. Polja za uključivanje i isključivanje animiranja monitoringa memorije i registara;
4. Jezičak za podešavanje pauze između izvršavanja instrukcija programa;
5. Komponenta za prikaz 64KB radne memorije procesora u matrici 256x256 piksela;
6. Polje sa prikazanim priključenim virtualnim diskovima;
7. Polje za unos izvornog koda programa u jeziku assembler;
8. Dugme za dodavanje prvog sledećeg slobodnog virtualnog diska;
9. Monitoring komponente za praćenje i animiranje stanja bitova u registrima;
10. Dugme za otvaranje prvog sledećeg slobodnog displeja;
11. Polja za unos pojedinačnih karaktera u heksadecimalnom ili unikatnom formatu;
12. Polje za unos zastopnih karaktera kucanih direktno preko tastature;
13. Polje za prikaz bajt koda prevedenog programa u heksadecimalnoj notaciji;
14. Dugme za kompajliranja, tj. prevođenja izvornog koda iz assembler jezika u bajt kôd;
15. Dugme za čišćenje editora izvornog koda programa;
16. Prikaz sistemskog vremena koje interapti Milantex Tx16 procesora prosleđuju programu.

4.2 PISANJE IZVORNOG KODA PROGRAMA

Izvorni kôd programa se piše editoru koda u assembler programskom jeziku posebno konstruisanom za potrebe ovog emulatora, a prema setu instrukcija procesora Milantex Tx16 računara. Izvorni kôd programa se prevodi u bajt kâd zajedno sa svim pratećim argumentima, tj. parametrima prema striktno definisanom formatu za svaku instrukciju.

Svaka naredba u Milantex Tx16 asemblerskom programskom jeziku prati tačno određeni šablon. Postoje naredbe koje imaju samo naziv instrukcije bez dodatnih argumenata, zatim naredbe koje pored instrukcije imaju jedan argument i naredbe koje imaju dva argumenta posle naziva instrukcije.

Spisak svih mogućih naredbi je prikazan u Tabeli 1 - „Spisak instrukcija procesora Milantex Tx16 računara sa spisikom parametara“

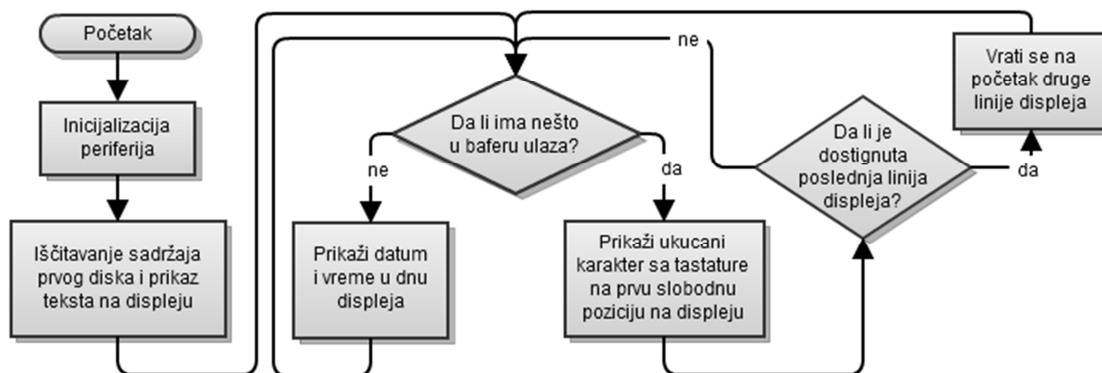
Nazivi instrukcija su prikazani u prvoj koloni tabele, dok su argumenti prikazani u tri polja posle naziva, gde svako polje predstavlja jedan bajt.

Naredbe u Milantex Tx16 programskom jeziku se prevode u instrukcije za procesor koje imaju veličinu od jednog bajta do četiri bajta, u zavisnosti od konkretne instrukcije.

4.3 PROGRAM KORIŠĆEN U DEMONSTRACIJI

Za potrebe demonstracije rada emulatora osmišljen je program koji koristi većinu instrukcija procesora Milantex Tx16 računara i koristi ulazni i izlazni uređaj (virtualnu tastaturu i displej), kao i jedan virtualni disk na kojem je smeštena poruka koju program iščitava sa diska i ispisuje korisniku na ekran.

Funkcionalnost programa je opisana sledećim uopštenim algoritmom:



Slika 16 - Uopšteni algoritam programa za demonstraciju rada emulatora

Program prilikom inicijalizacije za glavni izlazni uređaj podešava displej 0, tj. prvi displej priključen na izlaz emulatora.

Za ulaz emulatora, koristi se virtualna tastatura i ukucani karakteri se preuzimaju iz bafera tastature pomoću interapta procesora namenjenog za tu potrebu.

Poruka koja se ispisuje u vrhu displeja prilikom pokretanja programa se iščitava sa samog početka virtualnog diska pod rednim brojem 0, tj. prvog priključenog virtualnog diska.

4.4 IZVORNI KÔD PROGRAMA ZA DEMONSTRACIJU

Ovde je prikazan kompletan izvorni kod programa za demonstraciju rada Milantex Tx16 emulatora koji je opisan algoritmom sa Slike 16.

```
;; Autor:                Tair Milan
;; Datum:               18.02.2012.
;; Platforma:          Milantex Tx16
;; Jezik:              Milantex Tx16 assembler
;; Opis:               Program za demonstraciju rada emulator Milantex Tx16 procesora

@init:    MOV A 0          ;; Podesi displej
          MOV C 0          ;; Podesava stanje registra C na 0
          CALL @cls;       ;; Ocisti ekran
          MOV B 0          ;; Podesava stanje registra B na 0 - koristi se kao pokazivac pozicije za pocetak citanja sa diska
          CALL @read_disk; ;; Pozivi ciranje stringa sa diska
          JMP @s;          ;; Skace bezuslovno na labelu @s

@s:       LDF 8            ;; Ucitava flag IN ( da li u INPUT BUFFER-u ima necega )
          CFS @gc;         ;; Ako ima, pozovi funkciju na labeli @gc
          CFC @time;       ;; Ako nema, pozovi funkciju za prikaz vremena i datuma u dnu ekrana
          JMP @s;          ;; Kada uradi jedno ili drugo, vrati se na labelu @s i tako u krug ( GLAVNA petlja programa )

@gc:      PUT F            ;; Smesta trenutne vrednosti registara u stek
          PUT B
          PUT D
          INT 0xA3         ;; Izvrsava hardverski interapt koji uzima trenutnu slobodnu poziciju displeja
          MOV F 41         ;; Smesta vrednost 41 u D
          CAF              ;; Cisti flagove
          CMP F D          ;; Poredi 41 iz F sa izvucenom pozicijom u D
          LDF 1            ;; Ucitava stanje flaga VECE OD za proveru
          CFS @go_2_strt;  ;; Ako je flag setovan, zove funkciju koja vraca poziciju (cursor) displeja na pocetak drugog reda!
          GET D            ;; Vraca sacuvane vrednosti registara iz steka nazad u njih
          GET B
          GET F

          INT 0xE1         ;; Iscitava sledecu vrednost iz INPUT BUFFER-a
          PUT A            ;; Privremeno cuva vrednost koja je bila u A u stek
          MOV A 239        ;; Smesta vrednost 239 ( backspace karakter ) u A
          CAF              ;; Cisti flagove
          CMP D A          ;; Poredi iscitani karakter iz INPUT BUFFER-a sa backspace karakterom
          LDF 0            ;; Ucitava za testiranje flag EQUAL (siti su)
          JFC @p_ch;       ;; Ako nisu isti karakteri, tj. nije pritisnut backspace, skoci na ispis ucitanog karaktera
          PUT D            ;; U suprotnom sacuvaj jos nekoliko vrednosti registara u stek
          PUT A
          MOV A 0          ;; Postavi vrednost A na 0
          INT 0xA3         ;; Pozovi hardverski interapt koji smesta trenutnu poziciju na displeju u registar D
          DEC D            ;; Pomeri D unazad
          PUT B            ;; Sacuvaj stanje registra B
          CPY B D          ;; Kopiraj stanje registra D u B
          INT 0xA4         ;; Pozovi hardverski interapt koji pomera poziciju kursora na displeju na vrednost u B
          PUT C            ;; Sacuvaj stanje registra C
          MOV C ' '        ;; Smesti space vrednost u C
          CALL @pchar;     ;; Pozovi funkciju koja ispisuje karakter iz C
          CAF              ;; Ocisti flagove
          MOV C 0xFFFF    ;; Smesti vrednost FFFF u C
          CMP D C          ;; Poredi vrednost D i C
          LDF 0            ;; Vidi da li su isti
          JFS @s_1;        ;; Ako jesu, skoci na stavu @s_1 odmah da bi se vratila stanja iz registra koja su promenjena
          INT 0xA4         ;; A ako nisu, postavi poziciju kursora na vrednost iz B

@s_1:     GET C            ;; Smesta trenutne vrednosti registara u stek
          GET B
          GET A
          GET D
          GET A
          JMP @s;          ;; Kada su sve vrednosti vracene u registre iz steka, vrati se nazad na @s
                          ;; Vraca sacuvanu vrednost registara A iz steka nazad u A
@p_ch:    GET A            ;; Kopira vrednost iz D u C
          CPY C D          ;; Kopira vrednost iz D u C
          CALL @pchar;     ;; Stampa na ekran karakter pod tom vrednoscu
          PUT F            ;; Smesta trenutne vrednosti registara u stek
          PUT B
          PUT D
          INT 0xA3         ;; Izvrsava hardverski interapt koji smesta poziciju kursora na displeju u D
          MOV F 519        ;; Smesta vrednost 519 (poslednja pozicija u predposlednjem redu na ekranu)
          CAF              ;; Cisti flagove
          CMP D F          ;; Poredi vrednosti D i F
          LDF 1            ;; Ucitava za testiranje flag VECE od
          CFS @go_2_strt;  ;; Ako je kursor na poziciji vecoj od 519, pozovi funkciju @go_2_start da ga vrati na pocetak 2. reda
          GET D            ;; Vraca sacuvane vrednosti registara iz steka nazad u njih
          GET B
          GET F
          RET              ;; Završava funkciju i vraća se gde je pozvana
```

```

@go_2_strt: PUT B          ;; Smesta trenutne vrednosti registara u stek
MOV B 40      ;; Smesta u B vrednost 40 (pocetak drugog reda na displeju)
INT 0xA4      ;; Poziva hardverski interapt koji postavlja kursor displeja na vrednost iz B
GET B         ;; Vraca iz registra stanje koje je bilo u B pre promene
RET          ;; Završava funkciju i vraća se nazad gde je pozvana

@pchar:      PUT B          ;; Smesta trenutne vrednosti registara u stek
PUT A
MOV B 0xFF00  ;; Smesta u B FF00
AND B A       ;; Sabira sa vrednoscu u A
CPY A C       ;; Kopira C u A
OR A C        ;; Izvršava logicku operaciju OR nad bitovima registra A i C
INT 0xA1      ;; Poziva interapt koji ispisuje karakter na ekran
GET A         ;; Vraca sacuvane vrednosti registara iz steka nazad u njih
GET B
RET          ;; Završava se funkcija i skace nazad gde je pozvana

@cls:        PUT B          ;; Smesta trenutne vrednosti registara u stek
PUT A
MOV B 0xFF    ;; Smesta FF u registar B
AND B A       ;; Sabira B i A
CPY A C       ;; Kopira vrednost karaktera C u registar A
OR A C        ;; Izvršava logicku operaciju OR nad bitovima A i C
INT 0xA2      ;; Izvršava hardverski interapt koji cisti ekran odredjenim karakterom
GET A         ;; Vraca sacuvane vrednosti registara iz steka nazad u njih
GET B
RET          ;; Završava se funkcija i skace nazad gde je pozvana

@time:       PUT A          ;; Smesta trenutne vrednosti registara u stek
PUT B
PUT C
PUT D
PUT E
PUT F

INT 0xA3      ;; Smesta trenutnu poziciju kursora na ekranu u D
PUSH D        ;; Gura tu vrednost u stek

MOV A 0       ;; Stavlja 0 u A (prvi ekran)
MOV B 520     ;; Smesta u B 520 (prva pozicija u poslednjem redu na ekranu)
INT 0xA4      ;; Poziva hardverski interapt koji setuje poziciju kursora na ekranu 0 na 520 (prema registrima A i B)
MOV C 'V'     ;; Smesta vrednosot karaktera V u registar C
CALL @pchar;  ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C 'r'     ;; Smesta vrednosot karaktera r u registar C
CALL @pchar;  ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C 'e'     ;; Smesta vrednosot karaktera e u registar C
CALL @pchar;  ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C 'm'     ;; Smesta vrednosot karaktera m u registar C
CALL @pchar;  ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C 'e'     ;; Smesta vrednosot karaktera e u registar C
CALL @pchar;  ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C ':'     ;; Smesta vrednosot karaktera : u registar C
CALL @pchar;  ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C ' '     ;; Smesta vrednosot karaktera [space] u registar C
CALL @pchar;  ;; Poziva funkciju koja ispisuje uneti karakter na ekran

INT 0xB1      ;; Hardverski interapt koji prenosi trenutni datum i vreme u registre E i F po Milantex D/T formatu
PUT F         ;; Smesta trenutne vrednosti registara u stek
PUT E
SHR F 12      ;; Pomera bitove registra F za 12 mesta u desno
SHL E 15      ;; Pomera bitove registra E za 15 mesta u levo
SHR E 11      ;; Pomera bitove registra E za 11 mesta u desno
OR F E        ;; Izvršava logicko OR nad bitovima registara F i E
CPY A F       ;; Kopira vrednost registra F u A
MOV B 10      ;; Smesta 10 u B
CALL @itoap;  ;; Poziva funkciju koja vrednost iz registra A ispisuje na ekran
GET E         ;; Vraca sacuvane vrednosti registara iz steka nazad u njih
GET F

PUT F         ;; Smesta trenutne vrednosti registara u stek
PUT A
PUT B
MOV A 2       ;; Smesta vrednost 2 u registar A
MOV B 1       ;; Smesta vrednost 1 u registar B
MOD F A       ;; Racuna ostatak pri deljenju F sa A
CAF          ;; Cisti flagove
CMP F B       ;; Poredi F i B
LDF 0         ;; Ucitava flag EQUAL (jednoako) za testiranje
CFS @2_tacke; ;; Ako je flag setovan, poziva funkciju @2_tacke
CFC @razmak;  ;; Ako nije setovan, poziva funkciju @razmak
CALL @pchar;  ;; Poziva funkciju za ispis karaktera na ekran
GET B         ;; Vraca sacuvane vrednosti registara iz steka nazad u njih
GET A
GET F

```

```

PUT F          ;; Smesta trenutne vrednosti registara u stek
SHL F 4        ;; Pomera bitove registra F za 4 mesta u levo
SHR F 10       ;; Pomera bitove registra F za 10 mesta u desno
CPY A F        ;; Kopira F u A
MOV B 10       ;; Smesta vrednsot 10 u B
CALL @itoap;   ;; Poziva funkciju koja ce vrednost iz registra A ispisati na ekran
GET F          ;; Vraca sacuvane vrednosti registara iz steka nazad u njih
PUT F          ;; Smesta trenutne vrednosti registara u stek
PUT A
PUT B
MOV A 2        ;; Smesta vrednost 2 u registar A
MOV B 1        ;; Smesta vrednost 1 u registar B
MOD F A        ;; Racuna ostatak pri deljenju F sa A
CAF           ;; Cisti flagove

CMP F B        ;; Poredi F i B
LDF 0          ;; Ucitava flag EQUAL (jednoako) za testiranje

CFS @2_tacke;  ;; Ako je flag setovan, poziva funkciju @2_tacke
CFC @razmak;   ;; Ako nije setovan, poziva funkciju @razmak
CALL @pchar;   ;; Poziva funkciju za ispis karaktera na ekran

GET B          ;; Vraca sacuvane vrednosti registara iz steka nazad u njih
GET A
GET F

PUT F          ;; Smesta trenutne vrednosti registara u stek
SHL F 10       ;; Pomera bitove registra F za 10 mesta u levo
SHR F 10       ;; Pomera bitove registra F za 10 mesta u desno
CPY A F        ;; Kopira vrednost iz F u A
MOV B 10       ;; Smesta vrednost 10 u registar B
CALL @itoap;   ;; Poziva funkciju koja ce vrednost iz registra A ispisati na ekran
GET F          ;; Vraca sacuvane vrednosti registara iz steka nazad u njih

MOV C ' '      ;; Smesta vrednsot karaktera [space] u registar C
CALL @pchar;   ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C 'D'      ;; Smesta vrednsot karaktera D u registar C
CALL @pchar;   ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C 'a'      ;; Smesta vrednsot karaktera a u registar C
CALL @pchar;   ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C 't'      ;; Smesta vrednsot karaktera t u registar C
CALL @pchar;   ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C 'u'      ;; Smesta vrednsot karaktera u u registar C
CALL @pchar;   ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C 'm'      ;; Smesta vrednsot karaktera m u registar C
CALL @pchar;   ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C ':'      ;; Smesta vrednsot karaktera : u registar C
CALL @pchar;   ;; Poziva funkciju koja ispisuje uneti karakter na ekran
MOV C '.'      ;; Smesta vrednsot karaktera . u registar C
CALL @pchar;   ;; Poziva funkciju koja ispisuje uneti karakter na ekran

PUT E          ;; Smesta trenutne vrednosti registara u stek
SHL E 10       ;; Pomera bitove registra E za 10 mesta u levo
SHR E 11       ;; Pomera bitove registra E za 11 mesta u desno
CPY A E        ;; Kopira vrednost iz E u A
MOV B 10       ;; Smesta vrednost 10 u registar B
CALL @itoap;   ;; Poziva funkciju koja ce vrednost iz registra A ispisati na ekran
GET E          ;; Vraca sacuvane vrednosti registara iz steka nazad u njih

MOV C '.'      ;; Smesta vrednost karaktera . (tacka) u registar C
CALL @pchar;   ;; Poziva funkciju koja ispisuje uneti karakter na ekran

PUT E          ;; Smesta trenutne vrednosti registara u stek
SHL E 6        ;; Pomera bitove registra E za 6 mesta u levo
SHR E 12       ;; Pomera bitove registra E za 10 mesta u desno
CPY A E        ;; Kopira vrednost iz E u A
MOV B 10       ;; Smesta vrednost 10 u registar B
CALL @itoap;   ;; Poziva funkciju koja ce vrednost iz registra A ispisati na ekran
GET E          ;; Vraca sacuvane vrednosti registara iz steka nazad u njih

MOV C '.'      ;; Smesta vrednost karaktera . (tacka) u registar C
CALL @pchar;   ;; Poziva funkciju koja ispisuje uneti karakter na ekran

PUT E          ;; Smesta trenutne vrednosti registara u stek
SHR E 10       ;; Pomera bitove registra E za 10 mesta u desno
PUT A          ;; Cuva vrednost A u steku
MOV A 1970     ;; Smesta 1970 u registar A
ADD E A        ;; Sabira E i A - rezultat smesta u E
CPY A E        ;; Kopira vrednost iz E u A
MOV B 1000     ;; Smesta vrednost 1000 u registar B
CALL @itoap;   ;; Poziva funkciju koja ce vrednost iz registra A ispisati na ekran

GET A          ;; Vraca sacuvane vrednosti registara iz steka nazad u njih
GET E

```

```

MOV C '.'          ;; Smesta vrednost karaktera . (tacka) u registar C
CALL @pchar;       ;; Poziva funkciju koja ispisuje uneti karakter na ekran

PUT A              ;; Smesta trenutnu vrednost registara A u stek
MOV A 0            ;; Smesta 0 u A
PULL B             ;; Izvuci B iz steka
INT 0xA4           ;; Poziva hardverski interapt koji postavlja kursor na poziciju iz registra B na ekran iz registra A
GET A              ;; Vraca sacuvane vrednost registara A iz steka nazad u A

GET F              ;; Vraca sacuvane vrednosti registara iz steka nazad u njih
GET E
GET D
GET C
GET B
GET A
RET               ;; Zavrшава funkciju i vraća se na mesto gde je pozvana

@itoap: PUT A       ;; Funkcija koja broj (vrednost), razlaze na cifre i ispisuje na ekran
        PUT B       ;; Smesta stanja registara u stek
        PUT C
        PUT D
        PUT E
        PUT F
        MOV F 1      ;; Za proveru da li je doslo do cifre jedinice da je samo ispise
        CAF          ;; Ciscenje flagova
        CMP B F      ;; Poredjenje delitelja sa jedinicom
        LDF 1        ;; Proverava da li je delitelj veci od jedan
        JFS @t_4;    ;; Ako jeste, ide na korak 4
        JMP @t_5;    ;; Ako nije, ide na korak 5
        @t_4: CAF    ;; Na koraku 4 cisti flagove
        CMP A B      ;; Poredi vrednost A (broj) sa deliteljem
        LDF 1        ;; Pita da li je veci A
        JFS @t_6;    ;; Ako je A veci ide na korak 6
        LDF 0        ;; Pita da li je veci A
        JFS @t_6;    ;; Ako je A veci ide na korak 6
        JMP @t_7;    ;; Ako nije ide na korak 7
        @t_5: CPY C A ;; Smesta izvucenu cifru u visini delitenja u C
        MOV D '0'    ;; Smesta u D vrednost koja opisuje karakter 0
        ADD C D      ;; Dodaje na karakter 0 vrednost cifre i tako ide na karakter za tu cifru u mapi
        CALL @pchar; ;; Ispisuje karakter
        JMP @end;    ;; Skace na kraj
        @t_6: CPY C A ;; Upisuje broj koj ije ostao posle prethodnih koraka u C
        DIV C B      ;; Deli taj broj sa sledecim deliteljem ( 10x manji nego prosli put )
        MOV D '0'    ;; Dodaje vrenost cifre 0 - karaktera u D
        ADD C D      ;; Sabira izvucenu cifru kao vrednost sa vrednostu karaktera 0 i dobija karakter za cifru
        CALL @pchar; ;; Ispisuje dobijeni karakter na ekran
        JMP @t_B;    ;; Skace na korak B
        @t_7: MOV C '0' ;; Upisuje karakter za cifru 0 za ispis
        CALL @pchar; ;; Ispisuje karakter iz C
        JMP @t_B;    ;; Ide na korak B (koji je odmah ispod)
        @t_B: MOD A B ;; Racuna ostatak pri deljenju broja u A sa deliteljem u B
        MOV D 10     ;; Upisuje 10 u D
        DIV B D      ;; Deli delitelj sa 10 i tako dobija sledeci delitelj
        JMP @t_A;    ;; Vraca se ponovo na korak A i tako dokle god ne dodje do kraja i ode na @end
        @end: GET F   ;; Vraca iz steka u registar sacuvana stanja
        GET E
        GET D
        GET C
        GET B
        GET A
        RET          ;; Zavrшава funkciju i vraća se gde je pozvana

@itos: PUT B
        PUT C
        PUT D
        PUT E
        PUT F
        MOV E 1
        MOV F 0
        MOV B 10000
        MOV C 0
        @itos1: CPY D A
        DIV D B
        CAF
        CMP B E
        LDF 0
        JFS @itos2;
        JMP @itos3;
        @itos2: PUT B
        MOV B '0'
        ADD D B
        GET B
        PUT A
        MOV A 0

```

```

        CPY C D
        CALL @pchar;
        GET A
        JMP @itosE;
@itos3:  CAF
        CMP D F
        LDF 0
        JFS @itos5;
@itos4:  PUT B
        MOV B '0'
        ADD D B
        GET B
        PUT A
        MOV A 0
        CPY C D
        CALL @pchar;
        GET A
        MOV C 1
        JMP @itos6;
@itos5:  CAF
        CMP C F
        LDF 0
        JFS @itos6;
        PUT A
        MOV A 0
        MOV C '0'
        CALL @pchar;
        GET A
@itos6:  MOD A B
        PUT D
        MOV D 10
        DIV B D
        GET D
        JMP @itos1;
@itosE:  GET F
        GET E
        GET D
        GET C
        GET B
        RET

@read_disk: PUT A      ;; Sacuvaj vrednosti registara u stek
           PUT B
           PUT C
           PUT D
           PUT E
           PUT F
@r_ch:    MOV E 0xFF    ;; Koji karakter oznacava kraj stringa
           INT 0xD2     ;; Citaj karakter
           CAF          ;; Cisti flagove
           CMP D E      ;; Poredi iscitan karakter sa reg. E
           LDF 0        ;; Proverices da li su iste vrednosti
           JFS @r_end;  ;; Ako jesu, skoci na kraj citanja
           CPY C D      ;; Ako nisu, ispisi karakter
           CALL @pchar; ;; Ispisuje karakter
           INC B        ;; Pomeri poziciju za citanje za jedan
           JMP @r_ch;   ;; Kreni ponovo sa citanjem
@r_end:   NOP          ;; Na kraju ne radi nista
           GET F        ;; Vрати sacuvane vrednosti registara iz steka
           GET E
           GET D
           GET C
           GET B
           GET A
           RET          ;; Vрати se gde si pozvan

@razmak:  MOV C ' '     ;; Smesti karakter [space] u reg. C za ispis
           RET          ;; Vрати se gde si pozvan

@2_tacke: MOV C ':'     ;; Smesti karakter [:] u reg. C za ispis
           RET          ;; Vрати se gde si pozvan

```

Izvorni kôd programa je obeležen pomoću pametnog editora za obeležavanje koda koji se koristi u samom emulatoru i boje i oznake su identične prikazu koda u editoru teksta koji prikazuje objekat instanciran iz klase CodeEditor opisanoj ranije u radu.

4.5 PREVOĐENJE IZVORNOG KODA PROGRAMA

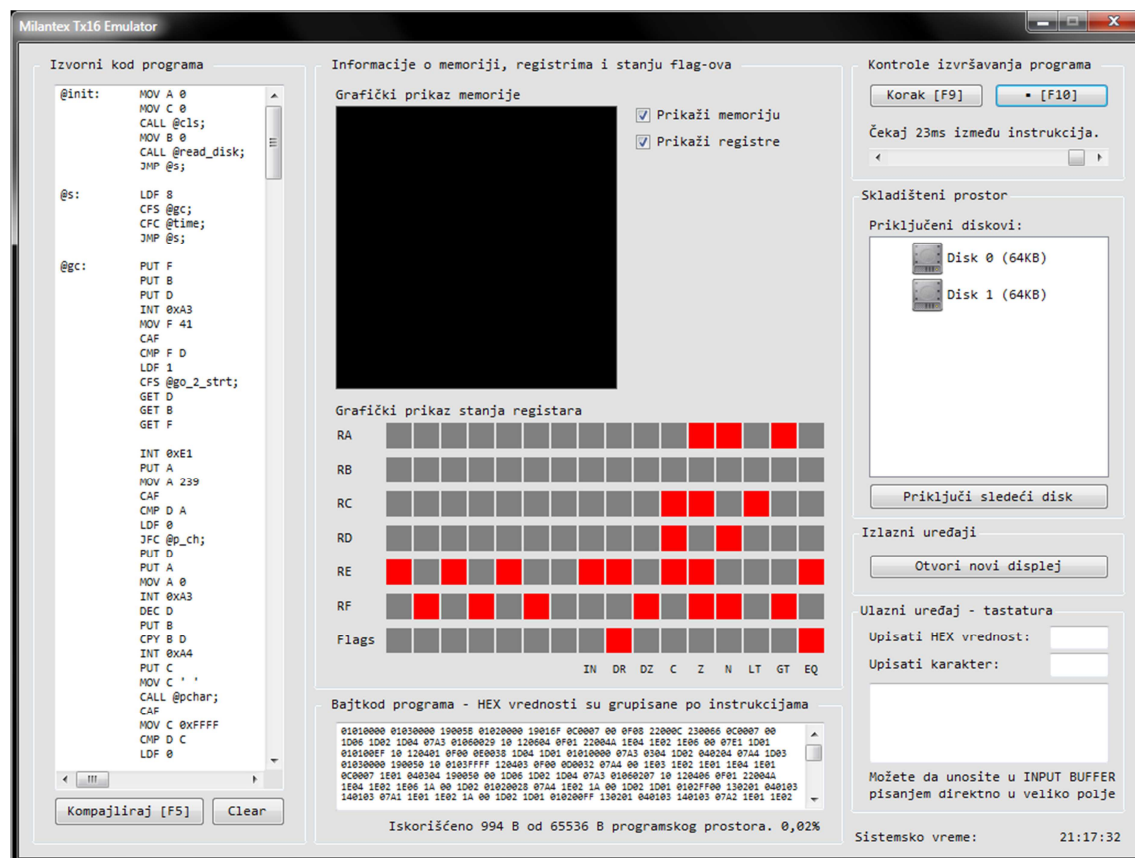
Izvorni kôd programa se iz assemblera prevodi u bajt kod pomoću prevodioca ugrađenog u sam emulator. Klikom na dugme za prevođenje, koje je na Slici 15 - „Prikaz pozicija najvažnijih komponenata glavnog prozora emulatora“, obeleženo pod rednim brojem 14, izvorni kôd se prevodi red po red i bajt kod se upisuje u programsku memoriju modula procesora emulatora, gde čeka izvršavanje.

Prikaz bajt koda u heksadecimalnoj notaciji sa razmakom između pojedinačnih instrukcija se nalazi u polju za prikaz bajt koda u donjem centralnom delu glavnog prozora emulatora.

4.6 POKRETANJE EMULATORA

Nakon što je program preveden i upisan u programsku memoriju, emulator može da se pokrene klikom na dugme za pokretanje procesa emulacije koje je na Slici 15, obeleženo pod rednim brojem 2. Emulator će početi da izvršava program od prve instrukcije u programskoj memoriji.

Brzinu emulacije je moguće podešavati jezičkom ispod dugmeta za pokretanje emulacije, dok je takođe moguće prekinuti automatsko izvršavanje emulacije i klikom na dugme koje je na Slici 15 obeleženo rednim brojem 1, moguće je izvršavati jednu po jednu instrukciju iz programske memorije.



Slika 17 - Prikaz glavnog prozora emulatora u toku izvršavanja programa za demonstraciju rada

4.7 PRAĆENJE STATUSA EMULACIJE

Dok je emulacija u toku i dok se program izvršava, na glavnom prozoru emulatora je moguće pratiti status memorije i registara, kao i stanja flagova u registru flagova u centralnom delu prozora emulatora.

Registri su prikazani kao šesnaest kvadratića u nizu obeleženi slovom koje predstavlja naziv registra čije stanje prikazuju. Svaki od šesnaest kvadratića predstavlja jedan bit registra koji je dvo bajtni, tj. čine ga dva bajta, niži i viši bajt. Flagovi su posebno označeni na dnu, a pomeranjem miša preko naziva flaga ispod kvadratića koji predstavlja prikazuje naziv flaga.

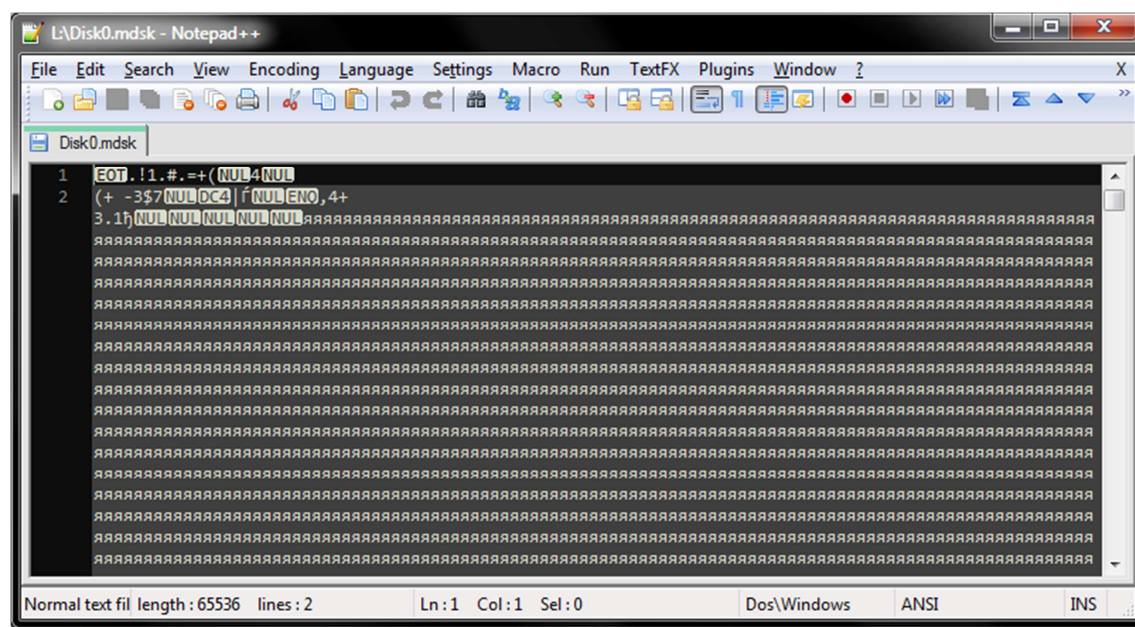
Crveni kvadratić predstavlja stanje 1, dok sivi kvadratić predstavlja stanje 0.

Prikaz radne memorije je smešten u gornji centralni deo prozora u matricu od 256 sa 256 piksela koji u zavisnosti od vrednosti bajta u memoriji menjaju osvetljenost od crne kada je vrednost bajta 0, do bele boje kada je stanje bajta 255. Brojanje bajtova u matrici odgovara rednom broju memorijske lokacije, tako što je početna, nulta memorijska lokacija na poziciji 0 sa 0 u matrici i kreće se horizontalno do desne ivice matrice, gde prelazi u novi red od leve ivice i tako se kreće sve do poslednje memorijske lokacije u 64 KB koliko je kapacitet radne memorije, koja odgovara loakciji 255 sa 255 u matrici.

4.8 DODAVANJE VIRTUALNOG DISKA I PRIKAZ NJEGOVOG SADRŽAJA

Virtualni disk se priključuje, tj. dodaje u emulator klikom na dugme za dodavanja virtualnih diskova prikazano na Slici 15 pod rednim brojem 8. Kada se dodaje novi disk, on zauzima poziciju koja je prva slobodna.

Prikaz sadržaja diska je omogućen tako što se dvoklikom na ikonicu koja predstavlja disk u listi priključenih diskova obeleženom rednim brojem 6 na Slici 15 otvara podrazumevani HexEditor ili teksualni editor, kao što je Notepad++ koji omogućava pregledanje i menjanje sadržaja diska van emulatora.



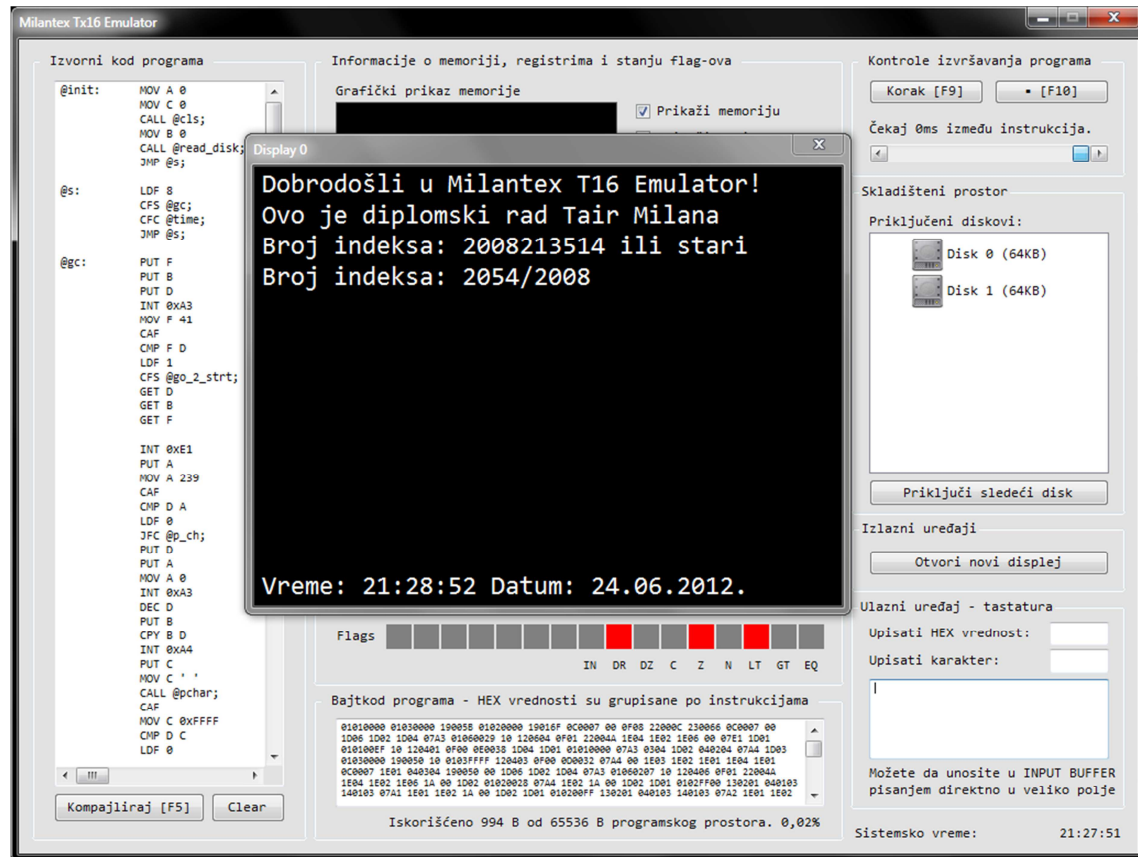
Slika 18 - Prikaz sadržaja prvog priključenog diska

4.9 DODAVANJE DISPLEJA I UPRAVLJANJE NJEGOVIM IZGLEDOM

Displej se priključuje, tj. dodaje u emulator klikom na dugme za dodavanje displeja koje je na Slici 15 obeleženo pod rednim brojem 10.

Displej se prikazuje kao nezavisan prozor sa crnim poljem za prikaz matrice karaktera dimenzije 40 karaktera širine sa 15 karaktera visine. U nazivu prozora koji drži displej stoji redni broj displeja što pomaže programeru prilikom kucanja programa da zna na koji displej želi da pošalje ispis teksta.

Moguće je menjanje pozadinske boje displeja kako je naznačeno u opisu klase Display, kao i menjanje boje teksta na displeju.



Slika 19 - Prikaz priključenog displeja sa ispisom programa za demonstraciju i tekstem unetim sa virtualne tastature

5 ZAKLJUČAK

U radu je predstavljen metod realizacije emulatora arhitekture računara osmišljenog za potrebe izrade ovog rada. Predstavljani su osnovni metodi izrade procesora za emulator, dok je za konkretnu realizaciju programa za emuliranje pomenute arhitekture korišćen pristup interpretiranja instrukcija iz programske memorije procesora.

Iz potrebe za bržom i jednostavnijom izradom emulatora korišćeno je integrisano razvojno okruženje i kôd emulatora je pisan u programskom jeziku C#. Grafički korisnički interfejs je pojednostavljen kako bi omogućio pregled svih komandi, komponenti za monitoring stanja i prikaz svih periferija.

Cilj izrade ovog rada je bio da se kombinuju znanja stečena tokom studija iz oblasti programiranja, osnova računarske tehnike, arhitekture računara uopšte, kao i programiranja grafičkih aplikacija.

Arhitektura Milantex Tx16 računara je osmišljena u cilju obezbeđivanja potpuno nove, do sada neobrađene arhitekture, obzirom da je na Internetu moguće pronaći izvorne kodove emulatora za većinu arhitektura i na taj način je onemogućeno dovođenje u pitanje originalnost rada i samog emulatora, tj. dokazano da nije u pitanju plagijat.

Jedna od osnovnih ideja za izradu ovog rada je bila da se napravi program koji bi, između ostalog, predstavljao mogućnost korišćenja jednostavnijih emulatora za manje složene arhitekture računara i sa manjim i jasnijim setom instrukcija, u edukativne svrhe i za uvođenje studenata informatike i računarstva, a primarno sa programerskih smerova, u oblast programiranja sistemskih aplikacija na najnižem nivou, kako bi se ujedno upoznali i sa arhitekturom računara i neigovim funkcionisanjem na najnižem programskom nivou.

Budući koraci bi trebalo da budu definisanje finalnog seta instrukcija, razvoj standardizovanih ulazno-izlaznih virtualnih uređaja za priključivanje na emulator i izrada inteligentnijih editora izvornog koda sa mogućnošću provere sintakse i pronalaženje najčešćih semantičkih grešaka, a u cilju izrade emulatora koji bi se koristio u edukativne svrhe.

6 LITERATURA

- [1] Tošić D., *Mikroprocesori sa elementima programiranja*, Zavod za udžbenike i nastavna sredstva, Beograd, 2003.
- [2] Jeff Rothenberg, *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation*, Council on Library and Information Resources, Washington DC, 1999.
- [3] Tomašević V., *Osnovi računarske tehnike*, Univerzitet Singidunum, 2012.
- [4] Jeffrey van der Hoeven „*Modular emulation as a long-term preservation strategy for digital objects*“, International Web Archiving Workshop, 2005. godine, Beč, Austrija.
- [5] Victor Moya del Barrio, *Study of the techniques for emulation programming*, 2001
- [6] Emulator, Vikipedija - besplatna enciklopedija, en.wikipedia.org/wiki/Emulator