МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД

из предмета

Програмирање и програмски језици

Emulator x86

Ученик

Филип Хаџић, IVb

Ментор

Нина Алимпић

Београд, јун 2011.

УВОД

Интернет апликација "*Emulator x86*" је едукативни програм, који омогућава тестирање кода написаног у асемблеру за x86 процесоре.

Пројекат је написан у програмском језику C#, а графички подржан у Silverlight-у. Ради лакшег и једноставнијег приступа, програм ради на интернету. Ова интернет апликација има предност што може да се покрене са било које платформе, која има подршку Silverlight-а, при чему није потребана инсталација. На овај начин сам приближио програм свим корисницима рачунара, па чак и неких SmartPhone уређаја.

Интернет адреса *Emulatora x86* је: www.emulatorx86.somee.com

АСЕМБЛЕР И ХИПОТЕТИЧКИ ПРОЦЕСОР

Асемблер је најнижи ниво програмског језика за програмирање рачунара, микроконтролора, микропроцесора. "Emulator x86" је апликација која симулира однос асемблера и фамилије хипотетичких процесора x86 (886, 8286, 8486 и 8686). Ови процесори на себи имају CPU регистре, што представљају меморијске локације конструисане од flip-flop кола. Процесори x86 имају тачно 4 регистра величине 16-бита. То су:

AX – Akumulator
BX – Bazni registar
CX – Count registar
DX – Data registar

Осим ових регистара на овим процесорима се налазе и flag регистри као и IP регистар.

flag регистри – регистри за чување резултата поређења два податка (мање, једнако)
IP регистар – регистар који чува вредност адресе за следећу инструкцију

Emulator x86 је "повезан" са меморијским простором, који има могућност памћења и читања података.

Процесори x86 имају 20 основних инструкција, који су подељени на:
-инструкције без аргумента (5 инструкција)
-инструкције са једним аргументом (8 инструкција)
-инструкције са два аргумента (7 инструкција)

Инструкције без аргумента су:
BRK – привремено зауставља рад програма, при чему је могуће прочитати тренутне вредности регистара
IRET – повратак из прекида (Intrrupt Return)
HALT – прекида рад програма
GET – учитава вредност који корисник унесе (вредност се уписује у AX регистар)
PUT – исписује вредност која се налази у AX регистру

ОПКОД ових инструкција изгледа овако: “0 0 0 0 0 м м м”

Инструкције са једним аргумента су:

NOT – инструкција логичке негације
JE – скок кода у случају да је поређење пре ове инструкције било једнако
JB – скок кода у случају да је поређење пре ове инструкције било мање
JBE – скок кода у случају да је поређење пре ове инструкције било мање или једнако
JA – скок кода у случају да је поређење пре ове инструкције било веће
JAE – скок кода у случају да је поређење пре ове инструкције било веће или једнако
JMP – скок кода

ОПКОД ових инструкција изгледа овако: “0 0 0 и и м м м”

Инструкције са једним аргумента су:

OR – инструкција логичког “или” и резултат се уписује у први аргумент
AND – инструкција логичког “и” и резултат се уписује у први аргумент
CMP – поређење аргумената
SUB – одузимање аргумената и резултат се уписује у први аргумент
ADD – сабирање аргумената и резултат се уписује у први аргумент
MOV – копирање другог аргумената у први аргумент (формат аргумената: рег, рег/мем/конст)
MOV – копирање другог аргумената у први аргумент (формат аргумената: мем, рег)

ОПКОД ових инструкција изгледа овако: “и и и р р м м м”

Аргументи се могу поделити у три групе:

-регистарски (формати: AX,BX,CX,DX)
-меморијски (формати: [BX],[xxxx+BX],[xxxx])
-константни (формати: xxxx)

ПРОГРАМИРАЊЕ

Прва идеја за “Emulator x86” ми је била да направим веб апликацију, због својих предности. За реализацију морао сам да програмирам у више програмских језика:

C#, Javascript, HTML

Коришћен framework је Silverlight.

Међусобан рад ових прогрмаских језика је заснован на објектно-оријентисаном и веб прогрмирању. Овај склоп програмских језика омогућава лако решавање скоро сваког програмског проблема.

Веб програмирање

У данашње време у области програмирања све се више учи и користи интернет програмирање. Веб програмирање има неких предности и неких мана. Предности веб апликација је то што се може приступити свакој апликацији на интернету са било ког рачунара у свету, није потребна инсталација, сваки стандардни оперативни систем може да покрене интенрнет апликацију. Највећи проблем ових апликација је то што је не могуће покренути апликацију ако наш уређај није повезан на интернет, мада у савременој техници скоро сваки уређај има могућност лаког приступа интернету. У данашње време веб програмирање доминира у области презентација, реклама, једноставних прогрма, у односу на класичне windows form апликације. Велика предност је и у лаком коришћењу база података, па обично свака веб апликација се повезује на неку датотеку.

За приступ веб апликацији потребно ју је поставити на сервер који има могућност приступа интернету и адекватане програме (сервисе) за одржавање протокола и за читање/слање разних података. Обично су ти сервери на Linux или Microsoft-овој платформи. Emulator x86 користи предности Silverlight-а који је производ Microsoft-а, па је Emulator x86 постављен на јавни и бесплатан hosting сајт www.somee.com , који ради на Microsoft-овој платформи.

Класични веб програмски језици су:
ASP, ASP.NET, ColdFusion, JSP/Java, PHP, Perl, Python…

Silverlight

Silverlight је алат за писање апликација. Прва верзија је настала 2007. године, што је представљала реплику Adobe Flex-a i Adobe Flash-a. Microsoft је успешно пласирао овај производ, па су сви претраживачи имплементирали подршку за овај алат. Silverlight је одлично решење за прављење веб апликација, јер подржава све дотадашње ASP контроле, са тим што им је омогућио и анимације. Silverlight има много више простора за сређивање изгледа скоро сваког елемента. При оваквом окружењу се добијају много бољи резултати, са иситим уложеним временом. Једина мана (као и код Flash player-a) је то што ако неки уређај нема browser који подржава Silverlight, не може приступити страници.

Silverlight команде се пишу у XML формату. Ово су неки примери кода из пројекта:

- Дефинисање стране:

```
<UserControl xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
xmlns:ei="http://schemas.microsoft.com/expression/2010/interactions"
xmlns:local="clr-namespace:Emulatorx86"
xmlns:toolkit="http://schemas.microsoft.com/winfx/2006/xaml/presentation/toolkit"
xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
mc:Ignorable="d"
      x:Class="Emulatorx86.MainPage"
      Width="1000" Height="700">
</UserControl>
```

- Дефинисање анимације на дугмету File:

```
<Storyboard x:Name="File">
      <ColorAnimation Duration="0" To="#FF454545"
      Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)"
      Storyboard.TargetName="canvas1" d:IsOptimized="True"/>
</Storyboard>
```

- Дефинисање Grid-a (подлоге):

```
      <Grid x:Name="LayoutRoot" Background="#660074B1">
            <Rectangle x:Name="rectangle" Fill="#FF565656" Height="45"
Stroke="Black" VerticalAlignment="Top" StrokeThickness="0"/>
            <Button Content="Debug" Margin="0,94,17,0" HorizontalAlignment="Right"
Width="120" Height="33" VerticalAlignment="Top" FontSize="13.333" Click="Debug" />
```

```
            <TextBox x:Name="KomandBox" Margin="8,94,0,8" TextWrapping="Wrap"
Text="" HorizontalAlignment="Left" Width="250" AcceptsReturn="True"
TextChanged="KomandBox_TextChanged" FontSize="16" FontWeight="Bold"/>
        <Button Content="Pokreni" Margin="0,131,17,0" HorizontalAlignment="Right"
Width="120" Height="33" VerticalAlignment="Top" FontSize="13.333"
Click="Button_Click_2" />
            <Button Content="Sledeci korak" Margin="0,168,17,0"
HorizontalAlignment="Right" Width="120" Height="33" VerticalAlignment="Top"
FontSize="13.333" Click="Button_Click_1" />
            <Button Content="Resetuj" Margin="0,205,17,0"
HorizontalAlignment="Right" Width="120" Height="33" VerticalAlignment="Top"
FontSize="13.333" Click="Button_Click_3" />
            <Button Content="Obriši memoriju" Margin="0,242,17,0"
HorizontalAlignment="Right" Width="120" Height="33" VerticalAlignment="Top"
FontSize="13.333" Click="Button_Click_4" />
            <TextBlock HorizontalAlignment="Left" Height="18" Margin="8,60,0,0"
TextWrapping="Wrap" VerticalAlignment="Top" Width="125" FontSize="16"
FontWeight="Bold"><Run Text="x86 Kod:"/></TextBlock>
            <TextBlock HorizontalAlignment="Left" Height="18" Margin="276,60,0,0"
TextWrapping="Wrap" VerticalAlignment="Top" Width="190" FontSize="16"
FontWeight="Bold"><Run Text="Emulovan x86 kod:"/></TextBlock>
            <TextBlock HorizontalAlignment="Right" Height="45" Margin="0,46,147,0"
TextWrapping="Wrap" VerticalAlignment="Top" Width="167" FontSize="16"
FontWeight="Bold" RenderTransformOrigin="0.497,-0.049"><Run Text="Ulazno"
Foreground="Green"/><Run Text=" / "/><Run Text="Izlazni" Foreground="Red"/><Run
Text=" panel"/></TextBlock>
        <Canvas HorizontalAlignment="Left" Height="100" VerticalAlignment="Top"
Width="100"/>

    </Grid>
```

На овај начин се много брже испрограмирала графика за цео сајт, због тога Silverlight штеди време, а при томе се не губи на функционалности, а добија се на лепшем и прегледнијем изгледу контрола.

C#

Emulator x86 је у C# добио и саму функционалност. Пошто је C# објектно-оријентисан програмски језик у њему сам правио различите класе. Укупно сам користио 24 класе и 7 UserControl-а. Свака инструкција у асемблеру има своју класу, а све су наслеђене од класе Инструкција.

-Класа Инструкција

```
public abstract class Instrukcija
{
        protected Argument a;
        protected Argument b;
        public abstract List<string> EmulovanKod();
        public abstract void IzvrsiKomandu();
}
```

Свака инструкција мора да има два аргумента (ако јој није потребан неки од аргумената, аргумент је једнак `null`), емулован код служи за приказ опкода.

Аргумент сам реализовао као посебну класу, јер аргумент може да буде различите врсте (регистарски, меморијски и константа).

Класа Аргумент има више функција међу којима је најбитинија “ArgumentPraser(`string` s)”, чији је задатак сличан као и конструкторов. Ова функција треба да из `string`-а постави врсту аргумента и индекс меморије.

```
private void ArgumentPraser(string s)
{
    if (s.Length == 2 && s[1] == 'X')
    {
        if (s[0] == 'A' || s[0] == 'B' || s[0] == 'C' || s[0] == 'D')
        {
            va = Pomocna.VrstaArgumenta.registarski;
            indexmemorije = s;
        }
        else throw new Exception();
    }
    else
        if (s[0] == '[' && s[s.Length - 1] == ']')
        {
            va = Pomocna.VrstaArgumenta.memorijski;
            if (s.EndsWith("+BX]"))
            {
                if (Pomocna.IspravanBroj(s.Substring(1, s.Length - 5)))
                {
                    indexmemorije = s.Substring(1, s.Length - 2);
                }
                else { throw new Exception(); }
```

```
                }
                else
                    if (s.Length == 4 && s[2] == 'X')
                        if (s[1] == 'A' || s[1] == 'B' || s[1] == 'C' || s[1] == 'D')
                        {
                            indexmemorije = s.Substring(1, 2);
                        }
                        else throw new Exception();
                    else
                    {
                        if (Pomocna.IspravanBroj(s.Substring(1, s.Length - 2)))
                        { indexmemorije = s.Substring(1, s.Length - 2); }
                        else
                        { throw new Exception(); }
                    }
            }
            else
            {
                if (Pomocna.IspravanBroj(s))
                {
                    va = Pomocna.VrstaArgumenta.konstanta;
                    indexmemorije = s;
                }
                else { throw new Exception(); }
            }
        }
```

Осим ове функције постоје и особине: Vrednost,Vrsta,IndexMemorije.
Аргумент је једна од најкоришћенијих класа, јер са њом осим што користи као веза између инструкција и меморија, користи за проверу исправности уписаних команди.

Класе са два аргумента

Класе са два аргументом су међусобно сличне.

-Класа Add / IzvrsiKomandu

```
public override void IzvrsiKomandu()
        {
            a.Vrednost = (a.Vrednost + b.Vrednost) % 65536;
        }
```

Ово је једна од сличности са осталим класама. Рецимо у класи Sub је само разлика у минусу или у класи Mov је разлика што се не користи a.Vrednost при рачунању нове вредности аргумента. Због оваквих предности сам се одлучио баш да правим класу за сваку инструкцију.

Емулован код сваке инструкције се разликује, по правилима пројектанта процесора, али постоје правила за аргументе.

-Класа Add / EmulovanKod

```
public override List<string> EmulovanKod()
      {
          string s="";
          string s1 = "";
          string s2 = "";
          List<string> l = new List<string>();
          s += "101";
          s+=Pomocna.RegistarOPCode(Pomocna.BrojArgumenta.prvi, a);
          if (b.Vrsta == Pomocna.VrstaArgumenta.konstanta)
          {
              s += "111";
              string ps = b.IndexMemorije;
              ps = ps.PadLeft(4, '0');
              s1 = ps.Substring(0, 2);
              s2 = ps.Substring(2);
          } if (b.Vrsta == Pomocna.VrstaArgumenta.registarski)
          {
              s += Pomocna.RegistarOPCode(Pomocna.BrojArgumenta.drugi, b);
          }
          if (b.Vrsta == Pomocna.VrstaArgumenta.memorijski)
          {
              if (b.IndexMemorije.Contains("+BX"))
              {
                string ps = b.IndexMemorije.Substring(0, b.IndexMemorije.IndexOf('+'));
                  ps = ps.PadLeft(4, '0');
                  s1 = ps.Substring(0, 2);
                  s2 = ps.Substring(2);
                  s += "101";
              }
              else
        if (b.IndexMemorije.Contains("BX") && b.IndexMemorije.Length == 2) s += "100";
                  else
                  {
                      string ps = b.IndexMemorije;
                      ps = ps.PadLeft(4, '0');
                      s1 = ps.Substring(0, 2);
                      s2 = ps.Substring(2);
                      s += "110";
                  }
          }
          l.Add(s);
          l.Add(s2);
          l.Add(s1);
          return l;
      }
```

Да би емулатор препознао инструкцију, програм мора да сваки унети ред упореди са исправним правилном синтаксом, на тај начин програм може да препозна корисникову команду. У случају да не постоји команда коју је корисник уписао, програм јавља грешку и број реда у којој је грешка.

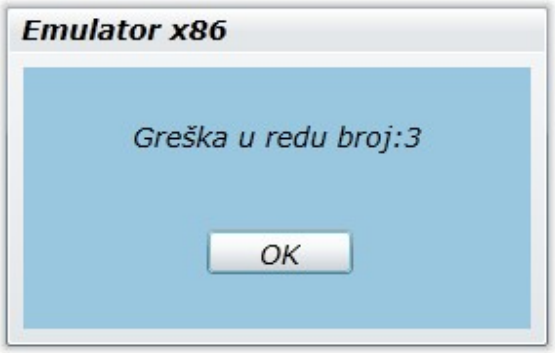
-Класа Add / ProveriIspravnostKoda

```
public static bool ProveriIspravnostKoda(string s)
        {
            Argument pomar1, pomar2;
            if (s.StartsWith("ADD "))
            {
                try
                {
                    s = s.Substring(4);
                    string[] ar = s.Split(',');

                    pomar1 = new Argument(ar[0]);
                    if (pomar1.Vrsta != Pomocna.VrstaArgumenta.registarski)
                        return false;
                    pomar2 = new Argument(ar[1]);
                }
                catch { return false; }
            }
            else { return false; }
            return true;
        }
```

У класе са два аргумента спадају команде:
Add,Sub,Cmp,And,Or,Mov

У случају да је корисник погреши при упису команди. Emulator x86 помаже корисницима, јер им тачно напише у ком реду се налази грешка. На овај начин је унапређена комуникација програма и корисника.



Класе са једним аргументом

Класе са једним аргументом се могу да се и зову jump класе, са тим што им се и придодаје Not класа. Jump класе служе за скок кода на лабелу која јој је аргумент. Зато што има 7 jump инструкција и међусобно су сличне, направио сам посебну класу, која је родитељ свих jump интрукција. Jump класа је изведена од класе инструкције.

```
public abstract class Jump:Instrukcija
    {
        protected Argument IPLabel;
        protected Registri reg;
    }
```

Ово је пример функције која извршава Ја команда.

```
        public override void IzvrsiKomandu()
        {
            if(!reg.Manje && !reg.Jednako)
            reg.IPSledeci = Pomocna.DecBrUHexString(IPLabel.Vrednost);
        }
```

На почетку испитује да ли су чекирани flag регистри Мање и Једнако. Ако јесте онда се поставља селдећи IP на вредност лабеле аргумента Jump објекта и при томе долази до скока на други део кода. Ако није код наставља по реду да ради. Остали део кода ових класа је сличан класама са два аргумента.

Класе са једним аргументом су:

Not,Ja,Je,Jae,Jb,Jbe,Jmp,Je

Класе без аргумената

Инструкције без аргумента су занимљиве јер приказују поруке корисницима. Проблем сам имао јер није могла да се прикаже порука са MessageBox.Show("…"), јер оваква команда ради на другој нити, па би код наставио са извршавањем, иако корисник није угасио приказану поруку. На тај начин би у случају да имамо више инструкција са једним аргументом, поруке би се приказале по принципу ЛИФО. Због оваквих проблема направио сам посебну класу Message и додао kraj и stop променљиве. Свака инструкција без аргумента која приказује поруку као променљиву има Message.

-Класа Halt / IzvrsiKomandu

```
public override void IzvrsiKomandu()
      {
          m.kraj = true;
          m.stop = true;
          mess = new Message("HALT instrukcija");
          mess.SubmitClicked += new EventHandler(mess_SubmitClicked);
          mess.Show();
      }
```

Меморије-UserControl

На главној форми се налазе две UserControle. Оне представљају меморије, па тако имамо праву меморију и регистарску меморију.

Задатак праве меморије је да исправно памти и враћа све вредности које процесор затражи. Она мора да се освежи при промени неке вредности, да се помера са корисниковим жељама. Са леве стране се налази TextBox који служи за промену меморијских простора које корсник жели да види.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000_	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001_	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002_	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003_	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004_	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

При оваквом распореду (0-F) много је лакше наћи жељену вредност.
Меморија се аутоматски шифтује на основу промене вредности TextBox-а.

-Класа MemorijaBoxControl / Refresh

```
private void Refresh()
 {
     if (MemorijaBox.Text.IndexOf("_") == -1)
         MemorijaBox.Text += "_";
     if (MemorijaBox.Text.Length == 4)
     {
         string hex = MemorijaBox.Text.Substring(0, 3) + "0";
         int dec = Pomocna.HexStringUDecBr(hex);
         string pomhex = "";
         for (int i = 1; i <= 4; i++){
             pomhex = Pomocna.DecBrUHexString(dec + 16 * i);
             pomhex = pomhex.Substring(0, pomhex.Length - 1);
             TextBlock x = (TextBlock)LayoutRoot.FindName("Lb" + i.ToString());
             if (pomhex.Length == 4)
                 pomhex = pomhex.Substring(1, 3);
             x.Text = pomhex.PadLeft(3, '0') + "_";
         }
         TextBox t;
         for (int i = 0; i <= 4; i++){
             for (int j = 0; j <= 16; j++){
              t = (TextBox)this.LayoutRoot.FindName(i.ToString() + j.ToString());
                 string hex1 = MemorijaBox.Text.Substring(0, 3) + "0";
            int dec1 = Pomocna.HexStringUDecBr(hex1);
            string s = Pomocna.DecBrUHexString(this[(i * 16 + j + dec1) % 65536]);
                 t.TextChanged -= new TextChangedEventHandler(t_TextChanged);
                 t.Text = s.PadLeft(2, '0');
                 t.TextChanged += new TextChangedEventHandler(t_TextChanged);
              }}}}
```

Контрола Registri је задужена за регулисање стања у регистрима у процесору. Она служи за чување и читање вредности података у регистрима. Једину функцију коју има је Reset.

-Класа Registri / Resetuj

```
internal void Resetuj()
 {
     IPSledeci = "0";
     IP = "0";
     AX = 0;
     BX = 0;
     CX = 0;
     DX = 0;
     Manje = false;
     Jednako = false;
 }
```

Ова класа има доста особина. Свака особина је задужена за сваки регистар (AX,BX,CX,DX,IP,flag registri).

-Класа Registri / AXHEX

```
public string AXHEX
      {
          get { return TBAX.Text; }
          set { TBAX.Text = value.ToString(); } }
```

Главна форма

Главна форма заправо даје и живот Emulatoru x86. На главној форми се налази мени на горњем делу екрана, са ког је могуће направити нов, снимити или отворити пројекат.

-Класа MainPage / SnimiKao_Click

```
private void SnimiKao_Click(object sender, System.Windows.RoutedEventArgs e)
        {
            SaveFileDialog saveDialog = new SaveFileDialog();
            saveDialog.Filter = "Script Emulator X86|*.sx86";
            bool? open = saveDialog.ShowDialog();
            if (open.HasValue && open.Value)
            {
                fs = saveDialog.OpenFile();
                byte[] fileByte = (new UTF8Encoding(true)).GetBytes(KomandBox.Text);
                fs.Write(fileByte, 0, fileByte.Length);
                fs.Flush();
            }
        }
```

У главној форми се налази дугме Debug где се кликом на њега, читају инструкције из простора за упис команди. Ако програм препозна инструкцију он је уписује у меморију и касније при покретању своје скрипте, програм чита из меморије инструкцију па је онда и извршава.

-Препознавање и провера исправности инструкција из KomandBox-а

```
while (komande.Length > 0)
            {

              ....

                if (inst.Contains(":"))
                {
                    if (nl.DodajUNiz(new Labela(Reg.IPSledeci, inst.Substring(0,
inst.IndexOf(':')))))
                    { inst = inst.Substring(inst.IndexOf(':') + 1); }
                    else { Pomocna.Messege("Greška u redu broj:" +
Convert.ToString(instrukcije.Count + 1)); break; }
                }
                if (Add.ProveriIspravnostKoda(inst) || Mov.ProveriIspravnostKoda(inst) ||
Not.ProveriIspravnostKoda(inst) || And.ProveriIspravnostKoda(inst) ||
Or.ProveriIspravnostKoda(inst) || Sub.ProveriIspravnostKoda(inst) || (inst == "GET ") ||
Cmp.ProveriIspravnostKoda(inst) || Jb.ProveriIspravnostKoda(inst) ||
Jbe.ProveriIspravnostKoda(inst) || Je.ProveriIspravnostKoda(inst) ||
Jne.ProveriIspravnostKoda(inst) || Ja.ProveriIspravnostKoda(inst) ||
Jae.ProveriIspravnostKoda(inst) || Jmp.ProveriIspravnostKoda(inst) ||
Put.ProveriIspravnostKoda(inst) || Halt.ProveriIspravnostKoda(inst) ||
Brk.ProveriIspravnostKoda(inst) || Iret.ProveriIspravnostKoda(inst))
                {
                    DodajInstrukciju(inst);
                }
```

```
                else
                {
        Pomocna.Messege("Greška u redu broj:" + Convert.ToString(instrukcije.Count + 1));
                    break;
                }           }
```

-Додавање инструкција из KomandBox-а у меморију

```
            switch (s)
            {
                case "ADD": i = new Add(a, b);
                    l = i.EmulovanKod();
                    il = new InputLb(listBox2, Reg.IPSledeci.PadLeft(4, '0'),
Pomocna.BinUHex(l[0]), l[1], l[2], "ADD", operatori[0], operatori[1]);
                    PovecajIP(Pomocna.BinUHex(l[0]), l[1], l[2]);
                    break;
                case "MOV": i = new Mov(a, b);
                    l = i.EmulovanKod();
                    il = new InputLb(listBox2, Reg.IPSledeci.PadLeft(4, '0'),
Pomocna.BinUHex(l[0]), l[1], l[2], "MOV", operatori[0], operatori[1]);
                    PovecajIP(Pomocna.BinUHex(l[0]), l[1], l[2]);
                    break;
                case "AND": i = new And(a, b);
                    l = i.EmulovanKod();
                    il = new InputLb(listBox2, Reg.IPSledeci.PadLeft(4, '0'),
Pomocna.BinUHex(l[0]), l[1], l[2], "AND", operatori[0], operatori[1]);
                    PovecajIP(Pomocna.BinUHex(l[0]), l[1], l[2]);
                    break;

....
}
}
....

Reg.Resetuj();
            for (int i = 0; i < inlniz.Count; i++)
            {
                if (inlniz[i].ImeInstrukcije.StartsWith("J"))
                {
                    inlniz[i].PrviArgument = nl.VratiIP(inlniz[i].PrviArgument);
                    inlniz[i].s3 = inlniz[i].PrviArgument;
                }
                DodajUMemoriju(inlniz[i].s2, inlniz[i].s3, inlniz[i].s4);
                inlniz[i].DodajUListBox();
            }
```

Пошто су убачене инструкције у меморију, кликом на Pokreni или Sledeći korak, инструкције морају сада да се преведу из меморије у код и онда да се изврше.То је разлог коришћења већликог броја case петљи. При извршавању инструкције IP се намешта на следећу адресу команде.

-Додавање инструкција из меморије у класау Instrukcija

```
Instrukcija i = null;
                switch (sp1)
                {
                    case "111": i = new Mov(new Argument(sp3, Reg, Memorija), new
Argument(sp2, Reg, Memorija)); break;
                    case "110": i = new Mov(new Argument(sp2, Reg, Memorija), new
Argument(sp3, Reg, Memorija)); break;
                    case "100": i = new Sub(new Argument(sp2, Reg, Memorija), new
Argument(sp3, Reg, Memorija)); break;
                    case "101": i = new Add(new Argument(sp2, Reg, Memorija), new
Argument(sp3, Reg, Memorija)); break;
                    case "010": i = new And(new Argument(sp2, Reg, Memorija), new
Argument(sp3, Reg, Memorija)); break;
                    case "001": i = new Or(new Argument(sp2, Reg, Memorija), new
Argument(sp3, Reg, Memorija)); break;
                    case "011": i = new Cmp(new Argument(sp2, Reg, Memorija), new
Argument(sp3, Reg, Memorija), Reg); break;
                    case "000":
                        {
                            switch (s2)
                            {
                                case "00":
                                    {
                                        switch (s3)
                                        {
                                            case "111": i = new Put(Panel, Reg); break;
                                            case "101": i = new Halt(this); break;
                                            case "011": i = new Brk(this); break;

                                            ...

                            }}}
```

Касније се и извршава инструкција:

```
i.IzvrsiKomandu();
```

Затим се испитује да ли је то послења инструкција, ако јесте исписује обавештење о крају.

Ово је један круг превођења из KomandBox>Меморија>Превођење>Извршавање.

На овај начин се и позива Pokreni само што програм ради све док не дође до краја.

ЗАКЉУЧАК

Сматрам да је ово један добар програмчић на коме се може снаћи сваки корисник. Има своје предности у односу на конкурентне програме овог типа. Надам се да ће се користити у едукативне сврхе, при чему ће и корисник бити задовољан перформансама ове апликације.
Намерно сам узео за дизајн комбинацију беле и плаве боје ради опуштања корисника, мислим да би било која друга комбинација боја била нападна по корисника. У овој документацији је могло још кода да стане, али сматрам да би било врло непрегледно и тешко за читање.

За прављење ове апликације много сам научио о Web програмирању. Мислим да су Silverlight и C# победничка комбинација за сваки пројекат, што и доказује Emulator x86. Препоручујем учење Silverlight-а.

ЛИТЕРАТУРА

Silverlight 4 Unleashed, Аутор: **Laurent Bugnion**

Professional Silverlight 4, Аутори: **Jason Beres,Bill Evjen,Devin Rader**

Microsoft Silverlight 4 Step by Step, Аутор: **Laurence Moroney**

Silverlight 4 How-To, Аутор: **Rajesh Lal**

Professional C# 4.0 and .NET 4, Аутор: **Kindle Edition**

Pro C# 2010 and the .NET 4 Platform, Аутор: **Andrew Troelsen**

Pro ASP.NET 4 in C# 2010, Аутор: **Matthew MacDonald**

HTML5: Up and Running, Аутор: **Mark Pilgrim**

JavaScript(TM) Step by Step, Аутор: **Steve Suehring**

САДРЖАЈ