

# PicoBlaze disassembler

---

**Kunsabo, Ivan**

**Undergraduate thesis / Završni rad**

**2016**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:200:728226>

*Rights / Prava:* [In copyright](#) / [Zaštićeno autorskim pravom.](#)

*Download date / Datum preuzimanja:* **2025-02-05**

*Repository / Repozitorij:*

[Faculty of Electrical Engineering, Computer Science and Information Technology Osijek](#)



**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**

**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I  
INFORMACIJSKIH TEHNOLOGIJA**

**Sveučilišni studij**

**PICOBLAZE DISASSEMBLER**

**Završni rad**

**Ivan Kunsabo**

**Osijek, 2016**

## Sadržaj

1. UVOD .....	1
1.1. Zadatak završnog rada .....	1
2. PICOBLAZE – KCPSM6 .....	2
2.1. Svojstva KCPSM6.....	2
2.2. PicoBlaze KCPSM6 skup naredbi .....	3
2.3. Postupak assembliranja.....	4
3. DISASSEMBLER .....	5
3.1. Primjeri disassemblera .....	5
4. DESKTOP APLIKACIJA.....	6
4.1. Prevođenje naredbi .....	6
4.2. Grafički izgled aplikacije .....	8
4.3. Povezivanje C++ koda i C#.....	8
4.4. „Browse“ i „drag and drop“ prosljeđivanje datoteke .....	9
4.4.1. „Browse“ način prosljeđivanja.....	9
4.4.2. „Drag and drop“ način prosljeđivanja .....	10
4.5. Primjer rada aplikacije .....	10
5. ZAKLJUČAK .....	13
LITERATURA.....	14
SAŽETAK.....	15
ABSTRACT .....	15
ŽIVOTOPIS .....	16
PRILOG A: C++ program PicoBlaze disassemblera .....	17
PRILOG B: C# program PicoBlaze disassemblera .....	23
PRILOG C: Elektronička verzija završnog rada na CD-u .....	26



## **1. UVOD**

Cilj ovog završnog rada je napraviti aplikaciju koja prevodi strojni jezik, odnosno jezik računala, u jezik razumljiv čovjeku. U radu su opisana svojstva PicoBlaze-a te što je disassembler. Nadalje, objašnjeno je u kojem programskom jeziku je pisana aplikacija, objašnjeno je kako aplikacija prepoznaje naredbe te kako ih prevodi red po red. Sljedeće je prikazan izgled grafičkog sučelja, povezanost C++ i C# kodova i kako je omogućen „drag and drop“ i „browse“ pristup aplikaciji, odnosno na taj način proslijediti datoteku. Na kraju je prikazano na primjeru kako aplikacija radi.

### **1.1. Zadatak završnog rada**

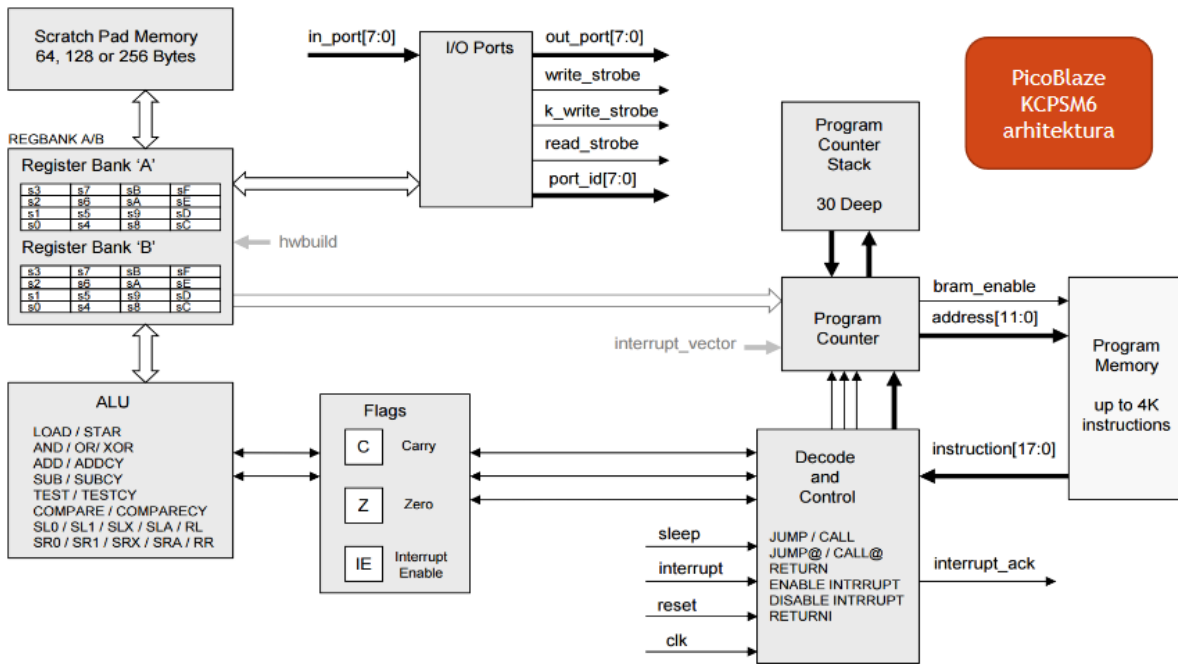
U ovom završnom radu potrebno je bilo napraviti desktop aplikaciju za disassembler mikroprocesora PicoBlaze. Aplikacija ima funkciju pretvorbe strojnog koda iz heksadekadske datoteke u datoteku koja sadržava riječima pisane naredbe mikroprocesora PicoBlaze.

## **2. PICOBLAZE – KCPSM6**

PicoBlaze je softverski definiran 8-bitni procesor za FPGA tehnologiju. FPGA(skr. Field Programmable Gate Arrays) tehnologija su čipovi koji su programirani od strane krajnjeg korisnika. KCPSM6 (skr. Ken Chapman's Programable State Machine) je PicoBlaze za Xilinx Spartan-6, Virtex-6 i 7-seriju FPGA. On zauzima manje od 5% Spartan-6 FPGA čipa. Na jednom Spartan-6 FPGA čipu može se definirati jedan ili više KCPSM6 procesora, jedini uvjet je osnovno poznavanje HDL(VHDL) i assembler jezika. Nexys<sup>TM</sup>3 je FPGA razvojni sustav tvrtke Digilent te je Spartan-6 FPGA čip na maketi. Maketa je podržana u programskom alatu XilinxISE od verzije 14.3 na dalje. Od ulazno/izlaznih uređaja koristi se 8 LED-ica, 8 sklopki, 5 tipkala, 4 znamenke 7 segmentnog pokaznika, PS/2 tipkovnica i UART serijska komunikacija.

### **2.1. Svojstva KCPSM6**

Svojstva KCPSM6 je 8-bitni mikroprocesor koji ima 8-bitne registre definirane u 2 banke po 16 registara. Radni takt je do 105MHz za Spartan-6, on može biti i viši za napredniji FPGA. Nadalje, ima 18-bitne naredbe(21 aritmetičko logičke, 8 upravljačkih te 2 podatkovne) za koje ima 4K(4096) bita memorije. Blok RAM memorija je proizvoljne veličine te stog za programsko brojilo ima 30mjesta. Svaka naredba se izvršava u dva radna takta, može izvršiti od 52 do 119 milijuna naredbi u sekundi. Postoje zastavice Z (zero) i C (carry) za informaciju o rezultatu ALU naredbi i zastavica IE (interrupt enable) koja omogućuje interrupt ulazni bit. Na slici 2.1. možemo vidjeti arhitekturu PicoBlaze KCPSM6.



Slika 2.1. Arhitektura PicoBlaze KCPSM6[1]

## 2.2. PicoBlaze KCPSM6 skup naredbi

Postoje tri velike skupine naredbi i jedna dodatna, a to su: podatkovne, aritmetičko-logičke i upravljačke naredbe te dodatna skupina pomoćnih naredbi. Na slici 2.2. možemo vidjeti sve naredbe.

PODATKOVNE	ARITMETIČKO-LOGIČKE	UPRAVLJAČKE
▶ LOAD, STAR	▶ ADD, ADDCY, SUB, SUBCY	▶ JUMP, CALL, RETURN
▶ STORE, FETCH	▶ AND, OR, XOR	▶ DISABLE INTERRUPT
▶ INPUT, OUTPUT	▶ TEST, TESTCY	▶ ENABLE INTERRUPT
▶ OUTPUTK	▶ COMPARE, COMPARECY	▶ RETURNI DISABLE
▶ REGBANK A/B	▶ SL0, SL1, SLX, SLA, RL	▶ RETURNI ENABLE
▶ HWBUILD	▶ SR0, SR1, SRX, SRA, RR	

POMOĆNE NAREDBE
▶ Ove naredbe se ne izvode na procesoru, samo olakšavaju pisanje programa
▶ CONSTANT, NAMEREG, ADDRESS, STRING, TABLE, INST, INCLUDE

Slika 2.2. Naredbe PicoBlaze KCPSM6[1]

### 2.3. Postupak asembliranja

Asemblerski jezik je programski jezik nižeg stupnja kojeg koriste računala ili bilo koji drugi uređaji s mogućnošću programiranja. Svaki asemblerski jezik specifičan je za određenu računalnu arhitekturu, u usporedbi s nekim jezicima koji su na višoj razini i koji ne ovise o istoj. Program koji se naziva assembler pretvara asemblerski jezik u izvršni strojni jezik. Proces pretvorbe naziva se assembly, dok assembly time predstavlja korak u kojem je assembler pokrenut. Mnogi assembleri često nude dodatne mehanizme kako bi olakšali razvoj programa, kontrolirali asemblerski proces ili kako bi olakšali otklanjanje grešaka.

ADDRESS	000		
LOAD	S0,06		01106
LOAD	S1,60		01160
LOAD	S1,S0		00100
SUB	S1,01		19101
SUB	S0,01		19001
JUMP	NZ,003		36003
JUMP	007		22007
LOAD	S0,S0		00000
RETURN			25000



Slika 2.3. Primjer asembliranja

Na slici 2.3. možemo vidjeti primjer asembliranja. Postupak izvođenja programa je red po red, svaka naredba, osim pretprocesorskih, zauzima točno jednu adresu. Pomoću toga svaka naredba ima svoju adresu i može biti zapisana kao podatak u računalu. Taj podatak je heksadecimalnog tipa.

Ako svaku prevedenu naredbu spremimo, dobit će se datoteka u kojoj se nalaze samo heksadecimalni brojevi te će se ta datoteka nazivati heksadecimalna s ekstenzijom .hex. Ta datoteka je vrlo važna za aplikaciju jer jer ova aplikacija disassembler, odnosno radi obrnuti postupak od asembliranja, te će početna datoteka aplikacije biti heksadecimalna.



### 3. DISASSEMBLER

Disassembler predstavlja računalni program koji prevodi strojni jezik u asemblerski, odnosno vrši inverzne radnje u odnosu na assembler. Također, disassembler se razlikuje od decompiler-a koji strojni jezik pretvara u neki viši jezik prije nego u asemblerski. „Disassembly“, koji predstavlja izlaz(engl. Output) disassemblera često je napravljen s ciljem čitljivosti za ljude, a ne da bi bio prikladan kao ulaz(engl. Input) za assembler.

U asemblerskom jeziku dopušteno je korištenje konstanti i programerskih komentara. Iako, oni su uobičajeno uklonjeni iz asemliranog strojnog koda od strane assemblera.

Pisanje disassemblera koji proizvodi kod, koji kad je asemliran, proizvodi upravo originalni binarni je moguće, međutim, često se pojavljuju razlike. Ako je dobiven potpuno točni diassembly, javljaju se problemi u slučaju da program zahtjeva modifikaciju. Npr., naredba *jump* strojnog jezika može biti generirana asemblerskim kodom kako bi se skočilo na određenu lokaciju u kodu, ili na određeni broj bitova, a Disassembler ne može znati kakva je namjera. Međutim, ako programer želi dodati naredbe između *jump*-a i njegovog odredišta, neophodno je razumijevanje rada programa kako bi se odredilo hoće li *jump* biti apsolutan ili relativan, odnosno, hoće li njegovo odredište ostati fiksno, ili će se premjestiti kako bi se preskočila i izvorna i dodana naredba.

#### 3.1. Primjeri disassemblera

Disassembleri mogu biti samostalni ili interaktivni. Kada je samostalni disassembler pokrenut, on generira datoteku u asemblerskom jeziku koja može biti ispitana, a dok interaktivni odmah pokazuje efekt bilo koje promjene koju korisnik napravi. Tako da, ako disassembler ne zna da je dio programa zapravo kod on ga tretira kao podatak, a ako korisnik navede da je kod, rezultat disasembliranja koda je prikazan odmah tako da omogući korisniku da ga ispita i poduzme daljnje korake u njegovom izvođenju.

## 4. DESKTOP APLIKACIJA

Aplikacija PicoBlaze disassembler napisana je u C++ programskom okruženju. Ostvarena je na principu da aplikacija prima parametar, odnosno putanju do naziva heksadekadske datoteke u kojoj se nalaze naredbe te onda aplikacija prevodi red po red svaku naredbu i sprema prevedenu naredbu u novu datoteku. Na kraju, rezultat aplikacije je nova prevedena datoteka koja je razumljiva čovjeku, odnosno napisana je naredbama za PicoBlaze.

### 4.1. Prevođenje naredbi

Prevođenje naredbi ostvareno je pomoću `strncmp` i `strcmp` naredbi. Prva naredba je oblika `strncmp(string1, string2, n)`; i radi na način da uspoređuje `string1` sa `string2`, ali samo prvih „n“ znakova i u slučaju jednakosti vraća 0. Druga naredba je oblika `strcmp(string1, string2)`; i radi na sličan način kao prva naredba, odnosno uspoređuje cijeli `string1` sa `string2` i u slučaju jednakosti vraća 0. Bilo je potrebno obuhvatiti sve naredbe PicoBlaze-a i svaku naredbu posebno prevesti. Na slici 4.1. možemo vidjeti sve naredbe koje su prevedene, gdje je `aaa`: 12-bitna adresa raspona od 000 do FFF, `kk`: 8-bitna konstanta raspona od 00 do FF, `pp`: 8-bitni port ID raspona od 00 do FF, `p`: 4-bitni port ID raspona od 0 do F, `ss`: 8-bitna memorijska adresa raspona od 00 do FF, `x`: 8-bitni registar raspona banke od s0 do sF, `y`: 8-bitni registar raspona banke od s0 do sF.

Page	Opcode	Instruction	Page	Opcode	Instruction	Page	Opcode	Instruction
<b>Register loading</b>			<b>Shift and Rotate</b>			<b>Interrupt Handling</b>		
55	00xy0	LOAD sX, sY	67	14x06	SL0 sX	83	28000	DISABLE INTERRUPT
55	01xkk	LOAD sX, kk	67	14x07	SL1 sX	83	28001	ENABLE INTERRUPT
71	16xy0	STAR sX, sY	67	14x04	SLX sX	84	29000	RETURNI DISABLE
71	17xkk	STAR sX, kk	67	14x00	SLA sX	84	29001	RETURNI ENABLE
<b>Logical</b>			<b>Register Bank Selection</b>			<b>Jump</b>		
56	02xy0	AND sX, sY	70	37000	REGBANK A	87	22aaa	JUMP aaa
56	03xkk	AND sX, kk	70	37001	REGBANK B	88	32aaa	JUMP Z, aaa
57	04xy0	OR sX, sY	<b>Input and Output</b>			88	36aaa	JUMP NZ, aaa
57	05xkk	OR sX, kk	73	08xy0	INPUT sX, (sY)	88	3Aaaa	JUMP C, aaa
58	06xy0	XOR sX, sY	73	09xpp	INPUT sX, pp	88	3Eaaa	JUMP NC, aaa
58	07xkk	XOR sX, kk	74	2Cxy0	OUTPUT sX, (sY)	89	26xy0	JUMP@ (sX, sY)
<b>Arithmetic</b>			<b>Scratch Pad Memory</b>			<b>Subroutines</b>		
59	10xy0	ADD sX, sY	(64, 128 or 256 bytes)			92	20aaa	CALL aaa
59	11xkk	ADD sX, kk	81	2Exy0	STORE sX, (sY)	93	30aaa	CALL Z, aaa
60	12xy0	ADDCY sX, sY	81	2Fxss	STORE sX, ss	93	34aaa	CALL NZ, aaa
60	13xkk	ADDCY sX, kk	82	0Axy0	FETCH sX, (sY)	93	38aaa	CALL C, aaa
61	18xy0	SUB sX, sY	82	0Bxss	FETCH sX, ss	93	3Caaa	CALL NC, aaa
61	19xkk	SUB sX, kk	<b>Version Control</b>			94	24xy0	CALL@ (sX, sY)
62	1Axy0	SUBCY sX, sY	101	14x80	HWBUILD sX	96	25000	RETURN
62	1Bxkk	SUBCY sX, kk				97	31000	RETURN Z
<b>Test and Compare</b>						97	35000	RETURN NZ
63	0Cxy0	TEST sX, sY				97	39000	RETURN C
63	0Dxkk	TEST sX, kk				97	3D000	RETURN NC
64	0Exy0	TESTCY sX, sY				98	21xkk	LOAD&RETURN sX, kk
64	0Fxkk	TESTCY sX, kk						
65	1Cxy0	COMPARE sX, sY						
65	1Dxkk	COMPARE sX, kk						
66	1Exy0	COMPARECY sX, sY						
66	1Fxkk	COMPARECY sX, kk						

**Slika 4.1. Naredbe strojnog jezika prevedene u naredbu za PicoBlaze[1]**

Naredba 00xy0 prevedena je kao LOAD sX, sY, što bi značilo na primjeru naredba 00210 prevedena je kao LOAD s2, s1 te radi funkciju da kopira iz registra sY u registrar sX, u našem slučaju to je iz registra s1 u registar s2.

To je u C programskom jeziku ostvareno ovim kodom:

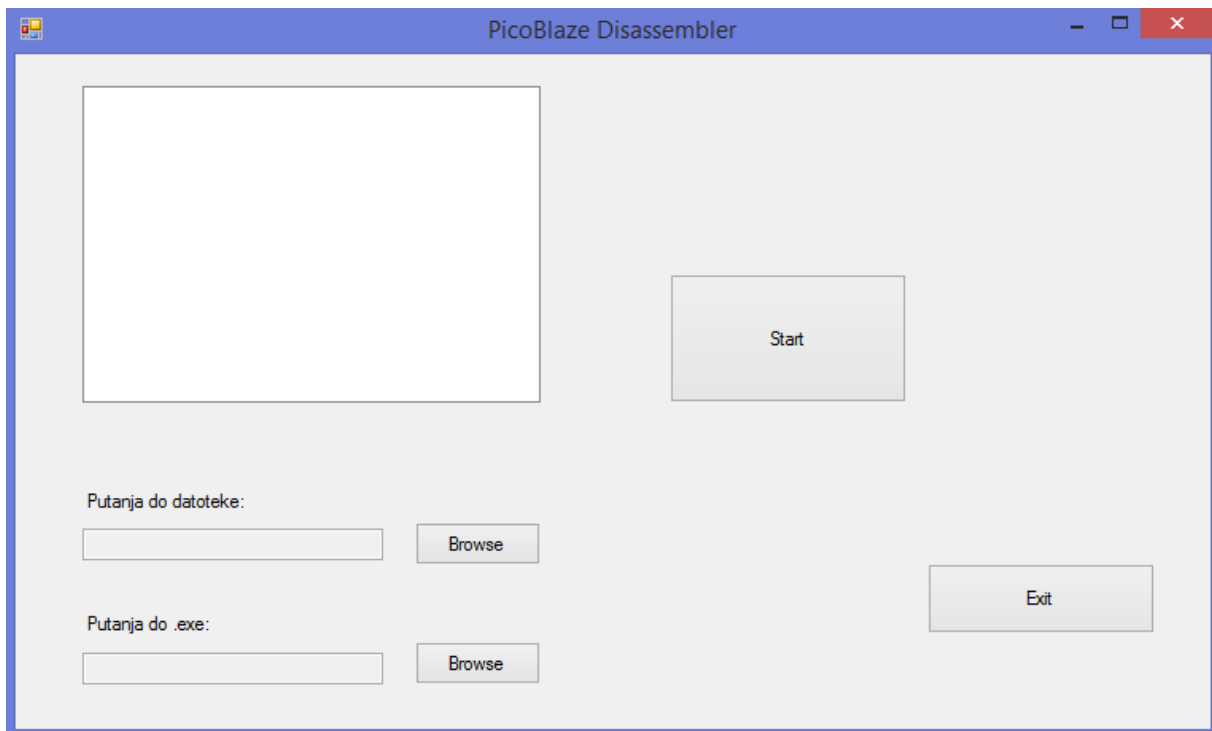
```
while(!feof(fp1)){
#define DULJINA_HEX_KODA 6
#define DULJINA_NAREDBE 100
char hex[6], naredba[DULJINA_NAREDBE];
memset(hex, 0, DULJINA_HEX_KODA * sizeof(char));
memset(naredba, 0, DULJINA_NAREDBE * sizeof(char));
fgets(hex,6,fp1);
if( strcmp(hex, "00", 2) == 0 ){
    sprintf(naredba, "LOAD s%c, s%c\n", hex[2], hex[3]);
}
fputs(naredba,fp2);
}
```

Na početku smo definirali dvije char veličine, „hex“ koja predstavlja naredbu koju će aplikacija pročitati iz prve datoteke i „naredba“ koja predstavlja prevedenu naredbu koja će se spremiti u drugu datoteku.

Na vrlo sličan način obrađene su sve naredbe.

## 4.2. Grafički izgled aplikacije

Grafički izgled aplikacije napravljen je u C# zbog jednostavnosti pravljenja formi.



**Slika 4.2. Grafički prikaz aplikacije**

Na slici 4.2. možemo vidjeti od čega se grafički dio aplikacije sastoji. Vidimo da je izgledom jednostavna te se sastoji od 4 gumba(eng. Button) koji služe redom za izlaz aplikacije, pokretanje deasembliranja, jedan za traženje heksadecimalne datoteke i jedan za traženje exe datoteke. Dalje se sastoji još od jedne liste(eng. Listbox), gdje će se spremati putanje heksadecimalnih datoteka.

## 4.3. Povezivanje C++ koda i C#

Pošto je grafičko sučelje aplikacije napravljeno u C# programskom jeziku, a samo prevođenje naredbi u C++ potrebno je povezati dva programska jezika. To je odrađeno pomoću klase process u C# programskom jeziku. C# programski jezik omogućuje pokretanje izvršnih

datoteka sa računala. Potrebno je bilo stvoriti novi proces, predati joj putanju izvršne datoteke i predati datoteci argumente, što je u ovoj aplikaciji, prvi argument je gdje će se spremiti nova datoteka i drugi argument putanja do heksadecimalne datoteke.

U C# programskom jeziku to je ostvareno ovim kodom:

```
if (path.Text == "") {
    MessageBox.Show("Molimo unesite putanju hex dokumenta!");
    return;
}
if (exePath.Text == "") {
    MessageBox.Show("Molimo unesite putanju exe dokumenta!");
    return;
}
Process myProcess = new Process();
myProcess.StartInfo.UseShellExecute = false;
myProcess.StartInfo.FileName = exePath.Text;
myProcess.StartInfo.Arguments = outputPath + " " + path.Text;
myProcess.StartInfo.CreateNoWindow = true;
myProcess.Start();
myProcess.WaitForExit(1000);
myProcess.Kill();
MessageBox.Show("Završeno!
Nova datoteka spremljena je ovdje: " + outputPath);
```

#### 4.4. „Browse“ i „drag and drop“ prosljeđivanje datoteke

Za uspješan rad aplikacije potrebno je omogućiti korisniku da odabere sam koju heksadecimalnu datoteku želi disasembliirati. Omogućena su dva načina, prvi je pretraži(eng. Browse) način, gdje korisnik samostalno pronade putanju do datoteke koja je spremljena na računalu te drugi način da korisnik željenu datoteku spremljenu bilo gdje na računalu prevuče klikom miša na listu gdje se spremaju sve datoteke tog tipa.

##### 4.4.1. „Browse“ način prosljeđivanja

Za omogućavanje browse načina potrebno je napraviti varijablu tipa klase OpenFileDialog te onda pomoću nje se napravi filter i otvaranje prozora za biranje. Ako je sve prošlo u redu, na listu se dodaje putanja do odabrane datoteke.

U C# programskom jeziku to je ostvareno ovim kodom:

```

OpenFileDialog ofd = new OpenFileDialog();
ofd.Filter = ".hex|*.hex";
    if (ofd.ShowDialog() == DialogResult.OK)
    {
        fileList.Items.Add(ofd.FileName);
        fileList.SelectedIndex = fileList.Items.Count - 1;
        getOutputPath(ofd.FileName);
    }

```

Na način sličnom ovome se napravilo prosljeđivanje putanje .exe datoteke.

#### 4.4.2. „Drag and drop“ način prosljeđivanja

Za drag and drop funkciju potrebno je bilo omogućiti u svojstvima liste AllowDrop. Te se naprave funkcije za DragEnter i DragDrop. U DragDrop funkciji se dodaje na listu nova datoteka.

U C# programskom jeziku to je ostvareno ovim kodom:

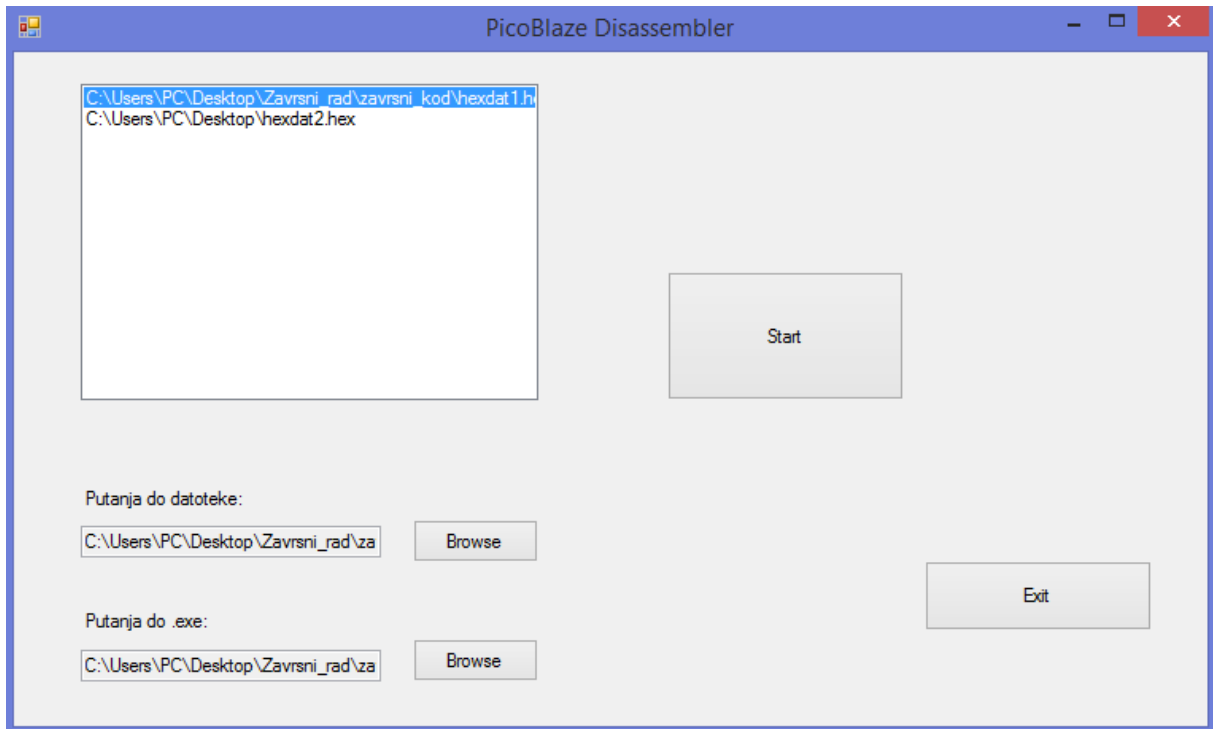
```

private void fileList_DragEnter(object sender, DragEventArgs e) {
    if (e.Data.GetDataPresent(DataFormats.FileDrop)) e.Effect = DragDropEffects.Copy;
}
private void fileList_DragDrop(object sender, DragEventArgs e) {
    string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);
    foreach (string file in files) {
        fileList.Items.Add(file);
        fileList.SelectedIndex = fileList.Items.Count - 1;
        getOutputPath(file);
    }
}

```

#### 4.5. Primjer rada aplikacije

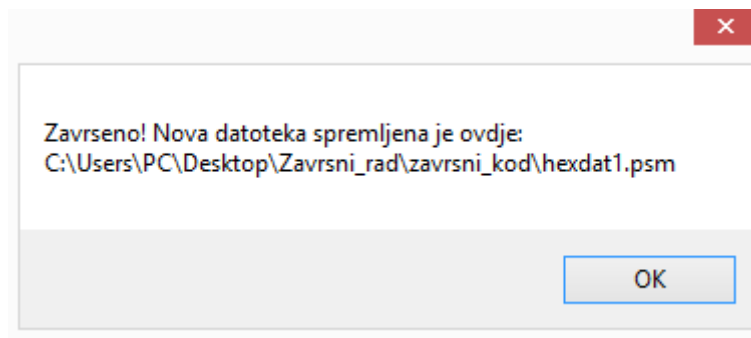
Aplikacija radi na način da je potrebno učitati putanju gdje je spremljena izvršna datoteka. Izvršna datoteka je kompajlirana C++ datoteka. Potrebno je odabrati heksadecimalnu datoteku koja će se prevesti. Moguće je učitati više heksadecimalnih datoteka ili prevući funkcijom „drag and drop“ te će se one sve dodati na listu. Ona datoteka na listi koja je označena će se prevesti kada se stisne gumb start.



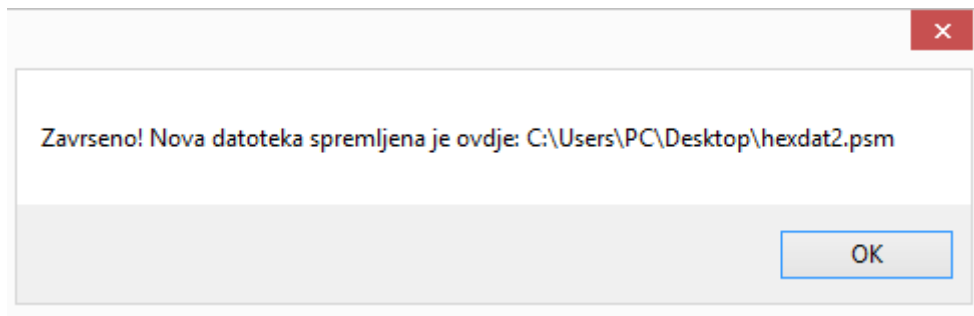
**Slika 4.3. Učitavanje izvršne i heksadecimalne datoteke**

Na slici 4.3. možemo vidjeti učitane izvršnu datoteku i dvije heksadecimalne datoteke.

Kada je prevođenje završeno dobiva se poruka da je prevođenje gotovo i ispiše se gdje je spremljena nova datoteka.



**Slika 4.4. Poruka sa putanjom gdje je spremljena nova hexdat1 datoteka**



**Slika 4.5. Poruka sa putanjom gdje je spremljena nova hexdat2 datoteka**

Nova prevedena datoteka sa ekstenzijom .psm je spremljena u direktorij gdje je bila heksadecimalna datoteka koja se prevodila. Na slikama 4.4. i 4.5. se to vidi, prevedena datoteka od datoteke koja je bila na radnoj površini računala spremljena je opet na radnu površinu, dok prevedena datoteka prve datoteke spremljene u direktorijima spremljena je točno u taj direktorij gdje je bila heksadecimalna datoteka.



## 5. ZAKLJUČAK

U ovom završnom radu napravljena je aplikacija pod nazivom PicoBlaze disassembler koja ima zadatak prevođenja heksadecimalne datoteke sa naredbama razumljivim računalu u datoteku s naredbama razumljivim čovjeku, odnosno naredbama za PicoBlaze. Opisano je što je i svojstva PicoBlaze KCPSM6 te postupak assembliranja na jednostavnom primjeru te zašto je heksadecimalna datoteka jedan od najvažnijih aspekata ovog rada. Nadalje, objašnjeno je što je disassembler i njegova primjena. Sljedeće je pojašnjen rad aplikacije, kako aplikacija prepoznaje svaku naredbu i daljnji postupak prevođenja red po red. Nakon toga, prikazano je izgled grafičkog sučelja, kako je povezan C++ programski jezik sa C# programskim jezikom i objašnjeno je dva načina prosljeđivanja datoteke, prvi način da se traži datoteka na računalu(engl. Browse) te drugi način „drag and drop“ prosljeđivanje datoteke. Na kraju, prikazano je kako aplikacija radi na primjeru kada joj se predaju dvije heksadecimalne datoteke i gdje sprema nove psm datoteke.

## LITERATURA

[1] Priručnik: PicoBlaze - KCPSM6 user guide 2014, lipanj 2016.

[http://www.xilinx.com/ipcenter/processor\\_central/picoblaze/member/KCPSM6\\_Release9\\_30\\_Sept14.zip](http://www.xilinx.com/ipcenter/processor_central/picoblaze/member/KCPSM6_Release9_30_Sept14.zip)

[2] Internet: Disassembler, lipanj 2016.

<https://en.wikipedia.org/wiki/Disassembler>

[3] Internet: Assembliranje, rujan 2016

[https://en.wikipedia.org/wiki/Assembly\\_language#Assembler](https://en.wikipedia.org/wiki/Assembly_language#Assembler)

[4] Internet: C# programerski vodič, rujan 2016.

<https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx>

[5] Internet: C++ programerski vodič, rujan 2016.

<http://www.cplusplus.com/doc/tutorial/>

[6] Predavanja, auditorne i laboratorijske vježbe iz kolegija Arhitektura računala

## **SAŽETAK**

**Naslov:** PicoBlaze Disassembler

U ovom radu riječ je o računalnoj aplikaciji PicoBlaze disassembler. Prije svega, dan je uvid u karakteristike PicoBlazea za lakše razumijevanje cilja aplikacije. Također, opisan je postupak assembliranja i pokazan na jednostavnom primjeru prevodenje red po red. Opisano je što je disassembler i njegova primjena. U radu je naglašena heksadecimalna datoteka jer je ona ključ rada programa, ona je ono što će aplikacija primiti i prevoditi. Sljedeće, objašnjeno je kako je aplikacija napravljena u C++ programskom jeziku, a grafičko sučelje u C#. Pojašnjeno je kako se prevode naredbe red po red u C++ programskom jeziku, kako su se povezali C++ i C# programski jezici. Naposljetku, objašnjeno je omogućavanje „drag and drop“ i „browse“ funkcije u C# programskom jeziku te je ukratko pokazan rad aplikacije na dvije datoteke spremljene na različitim lokacijama računala.

**Ključne riječi:** PicoBlaze disassembler, assembliranje, heksadecimalna datoteka, C++, C#

## **ABSTRACT**

**Title:** PicoBlaze Disassembler

This paper is about computer application called PicoBlaze disassembler. First of all, there is a review of the characteristics of the PicoBlaze so it is easier to understand the aim of the application. Also, the procedure of assembling was described and shown on a simple example. Disassembler and its application were described. In this paper, hex file was highlighted because it is the essence of the program, and the application will receive it and translate it. Furthermore, it is explained how the application was made in C++, and the graphical interface in C#. Also, it is explained how the commands are translated row by row in C++ and how were C++ and C# connected. Finally, „drag and drop“ and „browse“ functions were explained and it was briefly shown how the application works.

**Key words:** PicoBlaze disassembler, assembling, hex file, C++, C#

## **ŽIVOTOPIS**

Ivan Kunsabo rođen je 4. lipnja 1994. godine u Osijeku, Hrvatska. Nakon završene Osnovne škole Frana Krste Frankopana u Osijeku, upisuje Prirodoslovno-Matematičku gimnaziju u Osijeku. Godine 2013. upisao je sveučilišni preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

## PRILOG A: C++ program PicoBlaze disassemblera

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
using namespace std;

int main(int argc, char *argv[]){
    FILE *fp1, *fp2;

    // ulazna hex datoteka
    if ((fp1 = fopen(argv[2], "r"))==NULL)
    {
        printf("Datoteka %s ne postoji!\n",argv[2]);
        // cout<<argv[0]<<endl<<argv[1]<<endl<<argv[2]<<endl;
        system("pause");
        return 1;
    }

    // izlazna disassembler datoteka
    if ((fp2 = fopen(argv[1], "w+")) == NULL)
    {
        printf("Datoteka disassembler.psm ne postoji!\n");
        system("pause");
        return 1;
    }

    char constant[100];
    sprintf(constant, "CONSTANT LEDICE, 00 \n");
    fputs(constant,fp2);
    sprintf(constant, "CONSTANT HEX1, 01 \n");
    fputs(constant,fp2);
    sprintf(constant, "CONSTANT HEX2, 02 \n");
    fputs(constant,fp2);
    sprintf(constant, "CONSTANT PB_STAT, 00 \n");
    fputs(constant,fp2);
    sprintf(constant, "CONSTANT TIPKALA, 01 \n");
    fputs(constant,fp2);
    sprintf(constant, "CONSTANT SKLOPKE, 02 \n");
    fputs(constant,fp2);

    // čitati red po red iz hex datoteke i ispisivati u disassembler datoteku
    while(!feof(fp1)){

        #define DULJINA_HEX_KODA 6
        #define DULJINA_NAREDBE 100

        char hex[6], naredba[DULJINA_NAREDBE];
        char trenutna_adresa[10];
```

```

// postavlja vrijednosti polja znakova na 0
memset(hex, 0, DULJINA_HEX_KODA * sizeof(char));
memset(naredba, 0, DULJINA_NAREDBE * sizeof(char));

//čita jedan red iz hex datoteke
fgets(hex,6,fp1);

// strcmp(str1, str2, len); uspoređuje str1 sa str2 ali samo prvih
"len" znakova (len je int vrijednost),
// strcmp(str1, str2); uspoređuje str1 sa str2 i u slučaju jednakosti
vraca 0

if( strcmp(hex, "ADDRESS", 7) == 0) {
    printf(trenutna_adresa, "%c %c %c\n", hex[9], hex[10],
hex[11]);
}

//Register loading
if( strcmp(hex, "00", 2) == 0 ){
    printf(naredba, "LOAD %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "01", 2) == 0 ){
    printf(naredba, "LOAD %c, %%c\n", hex[2], hex[3], hex[4]);
}
else if( strcmp(hex, "16", 2) == 0 ){
    printf(naredba, "STAR %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "17", 2) == 0 ){
    printf(naredba, "STAR %c, %%c\n", hex[2], hex[3], hex[4]);
}

//Logical
else if( strcmp(hex, "02", 2) == 0 ){
    printf(naredba, "AND %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "03", 2) == 0 ){
    printf(naredba, "AND %c, %%c\n", hex[2], hex[3], hex[4]);
}
else if( strcmp(hex, "04", 2) == 0 ){
    printf(naredba, "OR %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "05", 2) == 0 ){
    printf(naredba, "OR %c, %%c\n", hex[2], hex[3], hex[4]);
}
else if( strcmp(hex, "06", 2) == 0 ){
    printf(naredba, "XOR %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "07", 2) == 0 ){
    printf(naredba, "XOR %c, %%c\n", hex[2], hex[3], hex[4]);
}
}

```

```

//Arithmetic
else if( strcmp(hex, "10", 2) == 0 ){
    sprintf(naredba, "ADD %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "11", 2) == 0 ){
    sprintf(naredba, "ADD %c, %c%c\n", hex[2], hex[3], hex[4]);
}
else if( strcmp(hex, "12", 2) == 0 ){
    sprintf(naredba, "ADDCY %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "13", 2) == 0 ){
    sprintf(naredba, "ADDCY %c, %c%c\n", hex[2], hex[3], hex[4]);
}
else if( strcmp(hex, "18", 2) == 0 ){
    sprintf(naredba, "SUB %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "19", 2) == 0 ){
    sprintf(naredba, "SUB %c, %c%c\n", hex[2], hex[3], hex[4]);
}
else if( strcmp(hex, "1A", 2) == 0 ){
    sprintf(naredba, "SUBCY %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "1B", 2) == 0 ){
    sprintf(naredba, "SUBCY %c, %c%c\n", hex[2], hex[3], hex[4]);
}

//Test and Compare
else if( strcmp(hex, "0C", 2) == 0 ){
    sprintf(naredba, "TEST %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "0D", 2) == 0 ){
    sprintf(naredba, "TEST %c, %c%c\n", hex[2], hex[3], hex[4]);
}
else if( strcmp(hex, "0E", 2) == 0 ){
    sprintf(naredba, "TESTCY %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "0F", 2) == 0 ){
    sprintf(naredba, "TESTCY %c, %c%c\n", hex[2], hex[3], hex[4]);
}
else if( strcmp(hex, "1C", 2) == 0 ){
    sprintf(naredba, "COMPARE %c, %c\n", hex[2], hex[3]);
}
else if( strcmp(hex, "1D", 2) == 0 ){
    sprintf(naredba, "COMPARE %c, %c%c\n", hex[2], hex[3],
hex[4]);
}
else if( strcmp(hex, "1E", 2) == 0 ){

```

```

        sprintf(naredba, "COMPARECY s%c, s%c\n", hex[2], hex[3]);
    }
    else if( strncmp(hex, "1F", 2) == 0 ){
        sprintf(naredba, "COMPARECY s%c, %c%c\n", hex[2], hex[3],
hex[4]);
    }

    //Shift and Rotate
    else if( strncmp(hex, "14", 2) == 0 ){
        if( hex[3] == '0' && hex[4] == '6' ){
            sprintf(naredba, "SL0 s%c\n", hex[2]);
        }
        else if( hex[3] == '0' && hex[4] == '7' ){
            sprintf(naredba, "SL1 s%c\n", hex[2]);
        }
        else if( hex[3] == '0' && hex[4] == '4' ){
            sprintf(naredba, "SLX s%c\n", hex[2]);
        }
        else if( hex[3] == '0' && hex[4] == '0' ){
            sprintf(naredba, "SLA s%c\n", hex[2]);
        }
        else if( hex[3] == '0' && hex[4] == '2' ){
            sprintf(naredba, "RL s%c\n", hex[2]);
        }
        else if( hex[3] == '0' && hex[4] == 'E' ){
            sprintf(naredba, "SR0 s%c\n", hex[2]);
        }
        else if( hex[3] == '0' && hex[4] == 'F' ){
            sprintf(naredba, "SR1 s%c\n", hex[2]);
        }
        else if( hex[3] == '0' && hex[4] == 'A' ){
            sprintf(naredba, "SRX s%c\n", hex[2]);
        }
        else if( hex[3] == '0' && hex[4] == '8' ){
            sprintf(naredba, "SRA s%c\n", hex[2]);
        }
        else if( hex[3] == '0' && hex[4] == 'C' ){
            sprintf(naredba, "RR s%c\n", hex[2]);
        }
    }

    //Version Control
    else if( hex[3] == '8' && hex[4] == '0' ){
        sprintf(naredba, "HWBUILD s%c\n", hex[2]);
    }
}

//Register Bank Selection
else if( strcmp(hex, "37000") == 0 ){
    sprintf(naredba, "REGBANK A\n");
}
else if( strcmp(hex, "37001" ) == 0 ){
    sprintf(naredba, "REGBANK B\n");
}

//Input and Output
else if( strncmp(hex, "08", 2) == 0 ){
    sprintf(naredba, "INPUT s%c, s%c\n", hex[2], hex[3]);
}
}

```



```

else if( strncmp(hex, "09", 2) == 0 ){
    sprintf(naredba, "INPUT s%c, %c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "2C", 2) == 0 ){
    sprintf(naredba, "OUTPUT s%c, s%c\n", hex[2], hex[3]);
}
else if( strncmp(hex, "2D", 2) == 0 ){
    sprintf(naredba, "OUTPUT s%c, %c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "2B", 2) == 0 ){
    sprintf(naredba, "OUTPUTK %c%c, %c\n", hex[2], hex[3], hex[4]);
}

//Scratch Pad Memory (64, 128 or 256 bytes)
else if( strncmp(hex, "2E", 2) == 0 ){
    sprintf(naredba, "STORE s%c, s%c\n", hex[2], hex[3]);
}
else if( strncmp(hex, "2F", 2) == 0 ){
    sprintf(naredba, "STORE s%c, %c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "0A", 2) == 0 ){
    sprintf(naredba, "FETCH s%c, s%c\n", hex[2], hex[3]);
}
else if( strncmp(hex, "0B", 2) == 0 ){
    sprintf(naredba, "FETCH s%c, %c%c\n", hex[2], hex[3], hex[4]);
}
}

//Interrupt Handling
else if( strcmp(hex, "2800") == 0 ){
    sprintf(naredba, "DISABLE INTERRUPT\n");
}
else if( strcmp(hex, "2801" ) == 0 ){
    sprintf(naredba, "ENABLE INTERRUPT\n");
}
else if( strcmp(hex, "2900") == 0 ){
    sprintf(naredba, "RETURNI DISABLE\n");
}
else if( strcmp(hex, "2901" ) == 0 ){
    sprintf(naredba, "RETURNI ENABLE\n");
}

//Jump
else if( strncmp(hex, "22", 2) == 0 ){
    sprintf(naredba, "JUMP %c%c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "32", 2) == 0 ){
    sprintf(naredba, "JUMP Z, %c%c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "36", 2) == 0 ){
    sprintf(naredba, "JUMP NZ, %c%c%c\n", hex[2], hex[3], hex[4]);
}

```

```

}
else if( strncmp(hex, "3A", 2) == 0 ){
    sprintf(naredba, "JUMP C, %c%c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "3E", 2) == 0 ){
    sprintf(naredba, "JUMP NC, %c%c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "26", 2) == 0 ){
    sprintf(naredba, "JUMP@ (s%c, s%c)\n", hex[2], hex[3]);
}

//Subroutines
else if( strncmp(hex, "20", 2) == 0 ){
    sprintf(naredba, "CALL %c%c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "30", 2) == 0 ){
    sprintf(naredba, "CALL Z, %c%c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "34", 2) == 0 ){
    sprintf(naredba, "CALL NZ, %c%c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "38", 2) == 0 ){
    sprintf(naredba, "CALL C, %c%c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "3C", 2) == 0 ){
    sprintf(naredba, "CALL NC, %c%c%c\n", hex[2], hex[3], hex[4]);
}
else if( strncmp(hex, "24", 2) == 0 ){
    sprintf(naredba, "CALL@ (s%c, s%c)\n", hex[2], hex[3]);
}
else if( strcmp(hex, "25000") == 0 ){
    sprintf(naredba, "RETURN\n");
}
else if( strcmp(hex, "31000" ) == 0 ){
    sprintf(naredba, "RETURN Z\n");
}
else if( strcmp(hex, "35000") == 0 ){
    sprintf(naredba, "RETURN NZ\n");
}
else if( strcmp(hex, "39000" ) == 0 ){
    sprintf(naredba, "RETURN C\n");
}
else if( strcmp(hex, "3D000" ) == 0 ){
    sprintf(naredba, "RETURN NC\n");
}
else if( strncmp(hex, "21", 2) == 0 ){
    sprintf(naredba, "LOAD@RETURN s%c, %c%c\n", hex[2], hex[3],
hex[4]);
}
}

```

```

// piše jedan red u datoteku disassembler.psm
fputs(naredba,fp2);
}
cout<<argv[0]<<endl<<argv[1]<<endl<<argv[2]<<endl;
fclose(fp1);
fclose(fp2);
system("pause");
return 0;
}

```

## PRILOG B: C# program PicoBlaze disassemblera

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace zavrzni_gotov
{
    public partial class Form1 : Form
    {
        OpenFileDialog ofd = new OpenFileDialog();
        string outputPath;

        public Form1()
        {
            InitializeComponent();
        }

        private void browse_Click(object sender, EventArgs e)
        {
            ofd.Filter = ".hex|*.hex";
            if (ofd.ShowDialog() == DialogResult.OK)
            {
                filesList.Items.Add(ofd.FileName);
                filesList.SelectedIndex = filesList.Items.Count - 1;
                getOutputPath(ofd.FileName);
            }
        }

        private void filesList_DragEnter(object sender, DragEventArgs e)
        {
            if (e.Data.GetDataPresent(DataFormats.FileDrop)) e.Effect =
DragDropEffects.Copy;
        }

        private void filesList_DragDrop(object sender, DragEventArgs e)
        {

```

```

        string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);
        foreach (string file in files) {
            fileList.Items.Add(file);
            fileList.SelectedIndex = fileList.Items.Count - 1;
            getOutputPath(file);
        }
    }

    private void getOutputPath(string file)
    {
        int n = 0;
        for (int i = file.Length-1; i > 0; i--)
        {
            if (file[i] == '.')
            {
                n = i;
                break;
            }
        }

        outputPath = file.Substring(0, n) + ".psm";
    }

    private void start_Click(object sender, EventArgs e)
    {
        if (path.Text == "") {
            MessageBox.Show("Molimo unesite putanju hex dokumenta!");
            return;
        }
        if (exePath.Text == "") {
            MessageBox.Show("Molimo unesite putanju exe dokumenta!");
            return;
        }
        Process myProcess = new Process();
        myProcess.StartInfo.UseShellExecute = false;
        myProcess.StartInfo.FileName = exePath.Text;
        myProcess.StartInfo.Arguments = outputPath + " " + path.Text;
        myProcess.StartInfo.CreateNoWindow = true;
        myProcess.Start();
        myProcess.WaitForExit(1000);
        myProcess.Kill();
        MessageBox.Show("Završeno! Nova datoteka spremljena je ovdje: " +
outputPath);
    }

    private void fileList_SelectedValueChanged(object sender, EventArgs e)
    {
        path.Text = fileList.GetItemText(fileList.SelectedItem);
        getOutputPath(path.Text);
    }

    private void exit_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }
}

```

```
private void browseExe_Click(object sender, EventArgs e)
{
    ofd.Filter = ".exe|*.exe";
    if (ofd.ShowDialog() == DialogResult.OK)
    {
        exePath.Text = ofd.FileName;
    }
}
}
```

## **PRILOG C: Elektronička verzija završnog rada na CD-u**