# Simulator mikroprocesora PicoBlaze

**Samaržija, Teo**



**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**
**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA**
**I INFORMACIJSKIH TEHNOLOGIJA**

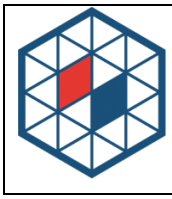
**Preddiplomski sveučilišni studij**

# SIMULATOR MIKROPROCESORA PICOBLAZE

**Završni rad**

**Teo Samaržija**

**Osijek, 2023.**



FERIT

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

| Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju | |
|---|---|
| **Osijek, 29.09.2023.** | |
| **Odboru za završne i diplomske ispite** | |
| **Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju** | |
| **Ime i prezime Pristupnika:** | Teo Samaržija |
| **Studij, smjer:** | Sveučilišni prijediplomski studij Računarstvo |
| **Mat. br. Pristupnika, godina upisa:** | R4273, 26.07.2018. |
| **OIB Pristupnika:** | 50107259317 |
| **Mentor:** | izv. prof. dr. sc. Ivan Aleksi |
| **Sumentor:** | , |
| **Sumentor iz tvrtke:** | |
| **Naslov završnog rada:** | Simulator mikroprocesora PicoBlaze |
| **Znanstvena grana rada:** | **Arhitektura računalnih sustava (zn. polje računarstvo)** |
| **Zadatak završnog rad:** | Temu rezervirao Teo Samaržija - Zadatak ovog završnog rada je izraditi Internet aplikaciju za simulaciju rada mikroprocesora PicoBlaze. |
| **Prijedlog ocjene završnog rada:** | Dobar (3) |
| **Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:** | Primjena znanja stečenih na fakultetu: 2 bod/bodaPostignuti rezultati u odnosu na složenost zadatka: 2 bod/bodaJasnoća pismenog izražavanja: 1 bod/bodaRazina samostalnosti: 2 razina |
| **Datum prijedloga ocjene od strane mentora:** | 29.09.2023. |
| **Datum potvrde ocjene od strane Odbora:** | 02.10.2023. |

| | |
|---|---|
| Potvrda mentora o predaji konačne verzije rada: | Mentor elektronički potpisao predaju konačne verzije. |
| | Datum: |

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

# IZJAVA O ORIGINALNOSTI RADA

**Osijek, 03.10.2023.**

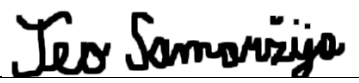
| | |
|---|---|
| **Ime i prezime studenta:** | Teo Samaržija |
| **Studij**: | Sveučilišni prijediplomski studij Računarstvo |
| **Mat. br. studenta, godina upisa:** | R4273, 26.07.2018. |
| **Turnitin podudaranje [%]:** | 1 |

Ovom izjavom izjavljujem da je rad pod nazivom**: Simulator mikroprocesora PicoBlaze**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivana Aleksija.

Moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:



# SADRŽAJ:

# 1. UVOD

PicoBlaze je mikroprocesor koji je opisan VHDL jezikom od strane tvrtke Xilinx [17]. Koristi se u ugrađenim računalnim sustavima. PicoBlaze je procesor dizajniran tako da se u cijelosti može implementirati programibilnim elektroničkim sklopovima (engl. FPGA – *Field-Programmable Gate Array*) te je pisan u programskom jeziku VHDL (VHDL je kratica od ***V****ery High Speed Integrated Circuits* ***H****ardware* ***D****escription* ***L****anguage*, što znači *jezik za opis strojne opreme (hardware) od integriranih krugova jako velike brzine* [17]).

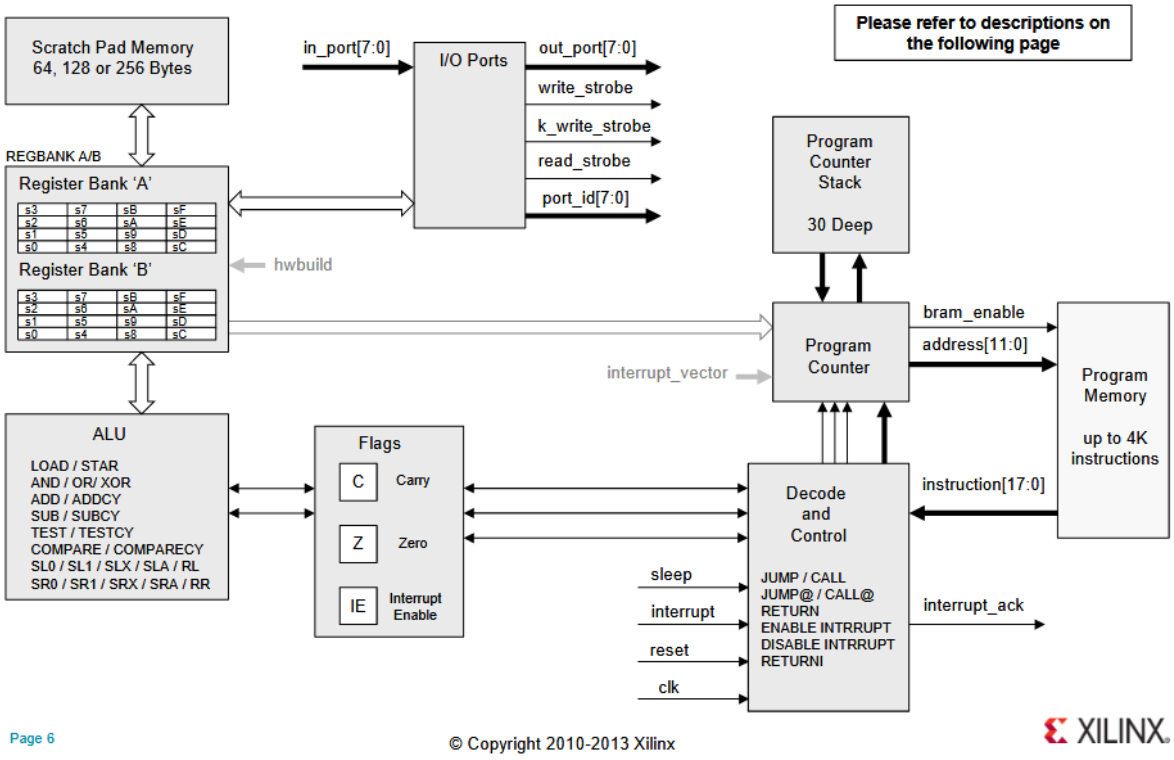
## 1.1. Zadatak završnog rada

Zadatak ovog završnog rada je izraditi Internet aplikaciju za simulaciju rada mikroprocesora PicoBlaze.

# 2. KORIŠTENE TEHNOLOGIJE

## 2.1. PicoBlaze procesor

PicoBlaze je soft procesor implementiran u VHDL-u i intelektualno je vlasništvo tvrtke Xilinx. On je osambitan, tj. sadrži tridesetdva registra od po osam bitova. Zato što je osambitan, bez dodatnih pomagala može adresirati najviše 256 bajtova memorije. Harvardske je arhitekture, što znači da mu je memorija s podatcima i s programskim kodom odvojena. Nema instrukcija namijenjenih čitanju ili pisanju iz memorije s programskim kodom ili u nju. Instrukcije su mu osamnaestbitne i najviše ih može imati 4096. Registri su mu podijeljeni u dvije regbanke i odjednom se može koristiti samo 16 registara. Ima tri zastavice: zero, carry i interrupt enabled. Može komunicirati s do 256 ulaznih i izlaznih uređaja.




**Slika 2.1.** *Unutarnja arhitektura procesora PicoBlaze*
[18]

## 2.2. Firefox

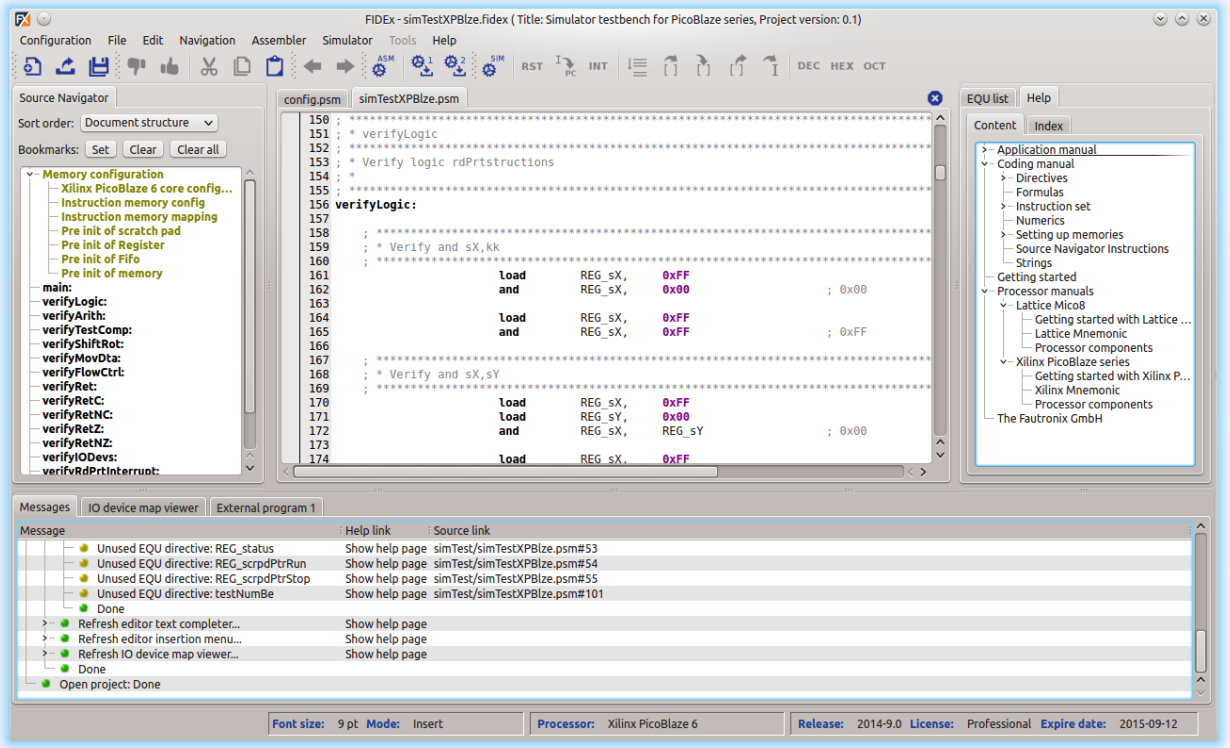
Firefox je internetski preglednik koji se dobije uz Oracle Linux i jedini je internetski preglednik koji radi prihvatljivo na Oracle Linuxu. Uz Oracle Linux još se dobije i Konqueror 4.14.8 (prastara verzija iz 2008. godine, doba Internet Explorera 7, i nema očitog načina da se instalira nova), koji je praktički beskoristan za današnji internet (rijetko koja se današnja internetska stranica prikazuje

ispravno u njemu), i uz njega se ne dobivaju nikakvi alati za programiranje. Danas i najmanje informatički pismeni ljudi znaju što je to Firefox jer su Firefox i Chrome danas najpopularniji internetski preglednici. Uz Firefox dobivaju se alati za programiranje: debugger za JavaScript, inspektor CSS-a te alati kojima se može aproksimirati kako će web-stranica izgledati na raznim mobitelima.

# 3. PREGLED POSTOJEĆIH SIMULATORA

Glavna mana današnjih simulatora PicoBlazea je to što ih se na računalo mora instalirati da bi funkcionirali. Nije ih moguće pokrenuti u internetskom pregledniku ili skinuti ZIP komprimiranu mapu i otpakirati je gdje želimo. Također, mnogi današnji simulatori pokušavaju simulirati PicoBlaze u VHDL-ovske detalje, što nužno povlači za sobom elemente zatvorenog koda (jer se PicoBlaze ne može compilirati GHDL-om, compilerom za VHDL otvorenog koda). Simuliranje PicoBlazea u VHDL-ovske detalje nekada je korisno, ali za potrebe laboratorijskih vježbi iz Arhitekture računala nije.

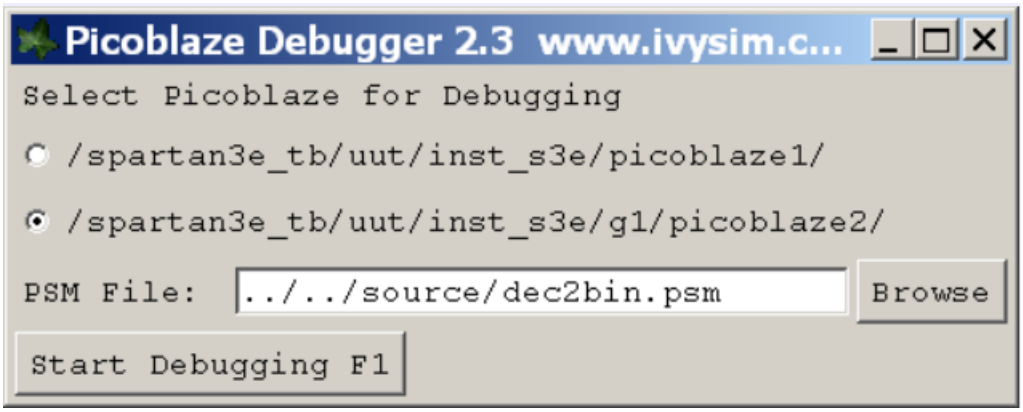
## 3.1. FIDEx simulator




**Slika 3.1.** *FIDEx simulator*
[19]

FIDEx simulator za PicoBlaze je zatvorenog koda, što mu je nedostatak. Značajna prednost pred simulatorom PicoBlazea napravljenog za ovaj rad je što može prikazivati imenovane registre, a ne samo nazive registara s0-sf, te također može prikazivati imenovane ulaze i izlaze imenom koje se koristi u asemblerskom kodu. Može simulirati različite verzije PicoBlazea: PicoBlaze 3 i PicoBlaze 6 te Lattice Mico 8.

## 3.2. PicoBlaze Debugger




**Slika 3.2.** *PicoBlaze Debugger*
[6]

PicoBlaze Debugger je dodatak Siemensovom programu ModelSim. Zatvorenog je koda. Može se skinuti u ZIP datoteci (ali moramo, da bismo ga koristili, već imati instaliran ModelSim). Kao i u PicoBlaze simulatoru napravljenom za ovaj rad, nedavno promijenjeni registri i flagovi obojeni su različitom bojom (da nam ne bi promaknula neočekivana promjena vrijednosti registra). Prednost mu je nad PicoBlaze simulatorom napravljenim za ovaj rad da se prijelazom miša na konstantu ili ime varijable (named registers) u asemblerskom kodu otvori pop-up prozor koji pokazuje relevantne informacije. On ima dvije vrste breakpointova: permanent i temporary. Simulator izrađen za ovaj rad podržava samo jednu vrstu breakpointova – permanent.

### 3.3. PicoBlaze IDE

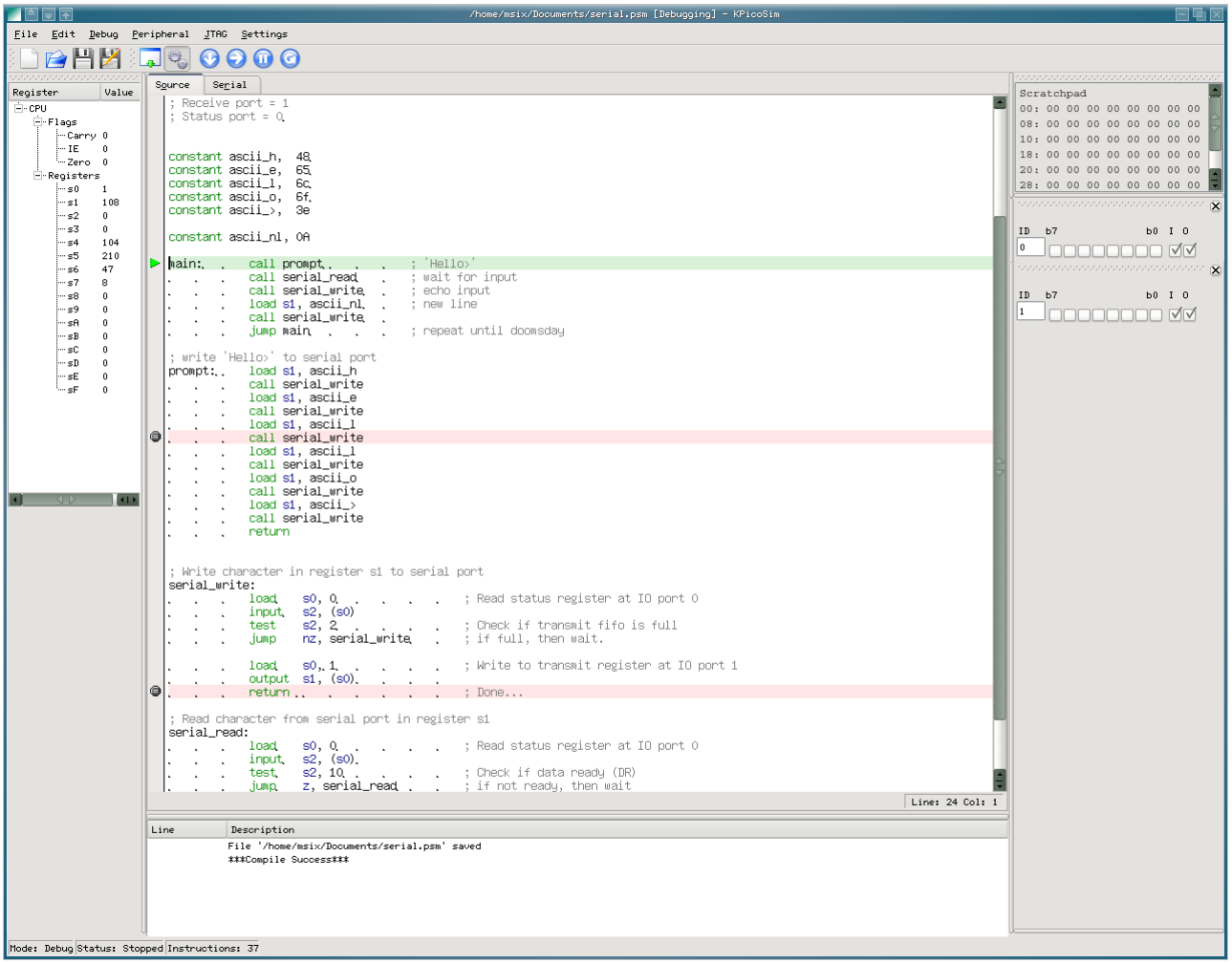
```
waveform_port       DSOUT     2                    ; bit0 will be data
counter_port        DSOUT     4
pattern_10101010    EQU       $AA
interrupt_counter   EQU       s3
```

**Slika 3.3:** *PBlaze IDE koristi asemblersku sintaksu nekompatibilnu s tradicionalnom*
[20]

Tvrtka Mediatronix nekada je proizvodila simulator PicoBlazea zvan PBlaze IDE. On je bio pisan u Delphiju (dijalektu Pascala) i koristio je SynEdit framework. Simulirao je PicoBlaze 1, PicoBlaze 2, PicoBlaze 3 i CoolBlaze. Jedna od njegovih prednosti je da on highlightira linije asemblerskog koda u kojima se nalaze greške, dok PicoBlaze Simulator napravljen za ovaj rad greške u asemblerskom kodu javlja pomoću alerta tipa „*Line #2: The AST node 'load'should have exactly three child nodes (a comma is also a child node).*”. PBlaze IDE također podržava automatsko formatiranje asemblerskog koda, što PicoBlaze Simulator napravljen za ovaj rad ne podržava. Kao i simulator

PicoBlazea napravljen za ovaj rad, podržava aritmetičke izraze u konstantama, uključujući i izraze sa zagradama. Nedostatak mu je što nije kompatibilan sa Xilinxovim asemblerom za PicoBlaze, recimo, u njemu su konstante po defaultu dekadske i mora ih se prefiksirati sa znakom $ da bi bile interpretirane kao heksadekadske. PicoBlaze Simulator napravljen za ovaj rad cilja da može asemblirati sve programe koje može i Xilinxov asembler (uz to što dodaje nove značajke, kao što su aritmetički izrazi u konstantama te if-grananja i while-petlje u pretprocesoru). Danas se PBlaze IDE nigdje na internetu ne može downloadati.

### 3.4. KPicoSim

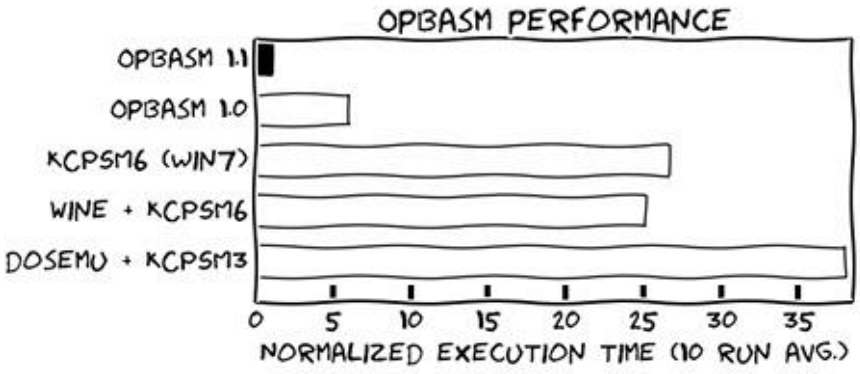



**Slika 3.3:** *KPicoSim*
[21]

KPicoSim je otvorenog koda, što mu je prednost. Napravio ga je Mark Six. Može eksportirati i VHDL, i HEX i MEM datoteke, a PicoBlaze Simulator rađen za ovaj rad može eksportirati samo HEX (odnosno, ne može ni HEX ako se pokrene u WebPositiveu, a vjerojatno ne može ni u Safariju). Nedostatak KPicoSima je komplicirana instalacija, naime mora se compilirati iz izvornog

koda da bi se koristio. Podjednako je funkcionalan kao PBlaze IDE. Posljednja verzija mu je izdana 2009. godine i ne spominje se podrška za PicoBlaze 6. To znači da npr. ne simulira regbanke niti jump@ instrukciju i njoj srodne instrukcije.

### 3.5. Open PicoBlaze Assembler (Opbasm)




**Slika 3.4.** *Open PicoBlaze Assembler*
*(Opbasm je napravljen da bude izuzetno brz asembler)*
[22]

Opbasm je asembler za PicoBlaze pisan u Pythonu. Napravljen je zato što je korištenje Xilinxovih asemblera na Linuxu izuzetno sporo. Koristi m4 pretprocesorske makronaredbe da prevodi npr. aritmetičke izraze na asemblerski jezik. Sadrži optimizator koji identificira mrtav kod. Ali ne sadrži nikakav simulator. Zbog tog što ne sadrži simulator, znatno je manje koristan nego ostali ovdje navedeni programi.

# 4. STRUKTURA SIMULATORA ZA PICOBLAZE U JAVASCRIPTU

Simulator PicoBlazea u JavaScriptu nema back-end (kôd koji se vrti na serveru), već se u cijelosti vrti u internetskom pregledniku. Sveukopno ima oko 4000 redaka koda.

1. PicoBlaze.html – sadrži HTML kôd i CSS kôd te JavaScript vezan za postavljanje izgleda web-aplikacije, sintaksno bojanje asemblerskog koda, postavljanje simulatorske niti, komunikaciju između tokenizera, parsera, pretprocesora i asemblera (u svijetu compilera to se zove *driver*) te za dohvaćanje primjera asemblerskog koda s autorova GitHub profila. Naknadno je dodana mogućnost dodavanja točaka prekida (*breakpoints*), mogućnost skidanja heksadekadske datoteke koju proizvodi assembler (koja se uz pomoć alata za programiranje PicoBlazea veoma lako pretvori u binarni format koji treba PicoBlazeu) te zoran prikaz sedam-segmentnih pokaznika (*seven-segment display*) i prekidača (*switches*, ovdje reagiraju na pritisak miša) pomoću SVG-a generiranog u JavaScriptu. Skidanje heksadekadske datoteke radi se preko standardnih JavaScript klasa *HTMLAnchorElement* (svojstvo *download* i metoda za stvaranje eventa iz JavaScripta *click*), *Uint8Array*, *URL* i *Blob*, bilo je relativno komplicirano za napraviti (Jedan od nedostataka pravljenja web-aplikacija je to što su takve stvari, koje se lagano naprave u drugim okruženjima, na webu radi sigurnosti teške za napraviti.). Točke prekida rade se tako da se brojevi linija koda nalaze u zasebnim HTML elementima  koje generira JavaScript i ti elementi osim brojeva linija koda sadržavaju i ikone koje predstavljaju točke prekida koje su po defaultu nevidljive. Globalni objekt *machineCode*, osim heksadekadskih stringova koje predstavljaju naredbe u strojnom kodu, čuva i podatke o linijama asemblerskog koda iz kojih dolaze naredbe. Velik dio HTML-a (sve tablice) također se generira u JavaScriptu, uglavnom pomoću JavaScriptovih višelinijskih stringova i JavaScriptiove naredbe *innerHTML*. Razni internetski preglednici rade probleme ako se tako pokušaju generirati SVG elementi (u JavaScriptu se generiraju SVG elementi koji predstavljaju sedam-segmentne pokaznike, prekidače i LED-ice), pa je svaki slučaj to rađeno JavaScriptovim naredbama *createElementNS*, *setAttribute* i *appendChild*. U datoteci PicoBlaze.html još se nalazi i kod za simulaciju UART-a, protokol pomoću kojega PicoBlaze može komunicirati s terminalima i simulatorima terminala, ali on je po defaultu isključen. Terminali su, naime, uređaji s tipkovnicom i ekranom pomoću kojih se može komunicirati s programima na udaljenim računalima koji imaju CLI sučelje, to jest, sučelje slično tipičnim DOS-ovim programima. Naknadno je datoteka PicoBlaze.html razdijeljena u datoteke PicoBlaze.html (koja sada sadrži samo HTML te samo onaj JavaScript koji je potreban da se sakrije autorova e-mail adresa od spambotova), styles.css (koja sadrži CSS), headerScript.js

(koja sadrži deklaracije potrebne drugim datotekama te skriptu za ponovno postavljanje layouta ako prozor promijeni veličinu) i footerScript.js (koja sadrži funkcije za počinjanje i završavanje simulacije te JavaScript koji stvara SVG-e). A 22. kolovoza 2023. godine izdvojeni su nizovi s mnemonikama i pretprocesorskim direktivama iz datoteke headerScript.js u zasebnu datoteku koju nazvanu list_of_directives.js, da bi se olakšalo pravljenje automatskih testova parsera.

Pri pisanju datoteke PicoBlaze.html pojavila su se tri ozbiljnija problema. Prvi je bio vezan za pokušaj da se u JavaScriptu napravi da program PicoBlaze sintaksno oboji (*syntax highlight*, u biti, da različite vrste riječi u programskom jeziku budu obojane različitom bojom) asemblerski program kako ga korisnik ukucava. To se, čak i uz mnogo pokušaja, nije moglo napraviti. Na kraju se moralo zaključiti da se ne pokušava bojati program dok ga korisnik zapisuje, nego da se doda gumb „*Highlight Assembly*" koji korisnik stisne kad želi da mu se sintaksno oboji program. Otvoreno je pitanje o tom problemu na forumu o izradi programskih jezika [2] (pretpostavka je bila da su mnogi koji su izradili programski jezik napravili i editor za njega u JavaScriptu, pa su možda naletjeli na isti problem i uspjeli ga riješiti), pa su se moderatori žalili da je pitanje *off-topic*, i na forumu o programiranju općenito [3], na kojem skoro tjedan dana nije dobiven nikakav odgovor. Odgovor koji je ponuđen na forumu o programiranju općenito predlaže da se iskoristi to što je asemblerski kod u monospace fontu, pa da se doda još jedan transparentan <pre> element ispred tog <pre> elementa u koji se ukucava asemblerski kod, te da se u tom transparentnom <pre> elementu prikazuje highlightirani asemblerski kod. Ta ideja nije tako dobra je će se, ako se to napravi, sigurno slomiti kompatibilnost s WebPositiveom (jer on ignorira CSS naredbe da prikazuje asemblerski kod u monospace fontu), ako ne i s Firefoxom 52 (u kojem monospace fontovi, kako se čini, nisu zapravo monospace).

Druga dva problema na koja se naiđe kada se piše PicoBlaze.html bila su da su različiti internetski preglednici različito interpretirali CSS koji im je napisan. Prvi od njih je da su se *tooltipsi* koji se pojave kada korisnik prijeđe mišem preko gumba sa slikom (*play*, *pause*, *stop*, *single step* i *fast forward*) dobro prikazivali u Firefoxu, ali su u Chromeu bili pomaknuti znatno u lijevo. Netko je na internetskom forumu predložio da se to jednostavno može riješiti tako da u CSS pravila za gumbe doda pravilo „`position: relative`" [3]. Takvo rješenje, koliko god se na prvi pogled činilo apsurdnim, bilo je točno. Drugi problem bio je da WebPositive, internetski preglednik koji se dobije uz operativni sustav Haiku, blisko srodan Safariju (i WebPositive i Safari baziraju se na WebKitu), nije stavljao svijetlosivu pozadinu na gumbe, pa su slova na gumbima (jer su onda to bila crna slova na ljubičastoj

pozadini) u njemu bila nečitka. No, kad se u CSS doda „`background: #cccccc;`”, radi kako treba. WebPositive još uvijek renderira neke stvari u PicoBlaze Simulatoru pogrešno, recimo, on asemblerske programe ne renderira u monospace fontu, i nema očitog načina da mu se zada da to napravi. Isto tako, WebPositive krivo renderira SVG gradijente u ikonama *play, pause, fast forward* i *single step*. I to nije samo u simulatoru PicoBlazea, nego i drugdje na autorovoj web-stranici, recimo, u WebPositiveu su strelice na dnu SVG PacMana nevidljive dok se na njih ne prijeđe mišem. No, te stvari ne sprječavaju da se PicoBlaze Simulator koristi u WebPositiveu. Drugi internetski preglednik za operativni sustav Haiku, Otter (bliski srodnik Operi), bolje renderira web-stranice nego što to radi WebPositive, ali Otter ne podržava dovoljno JavaScripta da se u njemu pokrene PicoBlaze Simulator (iako u Otteru, na primjer, SVG PacMan radi bez problema). Zbog tehničkih problema (Safari se, recimo, može pokrenuti samo na Macu), autor nije uspio isprobati svoj PicoBlaze Simulator u Safariju i Operi, no ne očekuje tamo velike probleme. U svakom slučaju, autor smatra da svatko tko želi koristiti njegov PicoBlaze Simulator, lako može na svoje računalo instalirati neki internetski preglednik u kojem se on može pokrenuti. U svim preglednicima na mobilnom operacijskom sustavu Androidu u kojima je isproban ovaj PicoBlaze Simulator, osim u Dolphinu (Dolphin se bazira na WebKitu, kao i Safari i WebPositive), *layout* se pokida kad se prijeđe u *landscape* (pejzažni) način rada (kad mobitel okrenemo horizontalno umjesto vertikalno). Autor misli da razumije zašto, ali ne zna to ispraviti. Otvoreno je pitanje o tom problemu na internetskom forumu [7], ali još nije došao neki koristan odgovor. S obzirom na to da se taj problem ne događa u Dolphinu, a Dolphin je bliski srodnik Safariju (oboje se baziraju na WebKitu), pretpostavka je da se taj problem ne događa ni u Safariju na iPhoneu, iako to za sada nije isprobano. U Firefoxu 52, ako se pokrene na Windowsima XP (ali, začudo, ne i ako se ta ista verzija Firefoxa pokrene na Solarisu),  element s brojevima linija i <pre> element u koji se ukucava asemblerski kôd imaju različite prorede, tako da linije koda i oznake brojeva linija nisu *alignirane*. To je najvjerojatnije bug u operativnom sustavu Windows XP. Slično tako, u nekim starijim verzijama Microsoft Edgea, baziranima na EdgeHTML-u (moderne verzije Microsoft Edgea baziraju se na Blinku, kao i Chrome),  element s brojevima linija ne *scrollira* se zajedno s onim <pre> elementom u koji se ukucava asemblerski program. Taj problem za sada nije riješen. Tako da se u Firefoxu 52 pokrenutom na Windows XP-u i u nekim starijim verzijama Microsoft Edgea, u kojima simulator PicoBlazea inače radi, ne mogu koristiti točke prekida (*breakpoints*). U Firefoxu 52 na Windows XP još se i može tako da, recimo, asemblerski program koji se želi debuggirati kopira u Notepad i u Notepadu u statusnoj traci vidi se koji je redni broj linije u kojoj mis-

limo da je problem, pa onda taj broj kliknemo u Firefoxu 52, ali takav *work-around* ne postoji u Microsoft Edgeu. Pojavio se i problem pri izboru boje za 7-segmentne pokaznike; cilj je bio da izgleda slično kao na pravom PicoBlazeu. I na ekranu na laptopu Acer Nitro 5 uistinu je tako izgledalo. Međutim, kad se to isprobalo na mobitelu Samsung Galaxy J3 (na kojem je, izgleda, zasićenje boja znatno slabije), tamo su brojevi na 7-segmentnim pokaznicima bili nečitki, nijansa crvene koje su bile odabrane za upaljene segmentne jedva da se razlikovala od nijanse tamnosive koja je prikazivala ugašene segmente. To je riješeno tako da je nijansa crvene boje sada znatno posvijetljena. Često se pojavljuju problemi u igrama bojama: što izgleda lijepo na jednom ekranu, često je na drugom ekranu nečitko. Pri pisanju primjera „*Regbanks-Flags Test*" primijećen je problem sa sintaksnim bojanjem asemblerskog koda. Naime, PicoBlaze Simulator asemblerski kod:

```
address 0
load s0, 0
load s0, a
regbank a
```

oboji ovako:

```
1. address 0
2. load s0, 0
3. load s0, a
4. regbank a
```

***Slika 4.1**: Neispravno sintaksno bojanje koda*

Što je netočno. Token *a* u naredbi „`load s0, a`" heksadekadska je konstanta (a je heksadekadski 10), i treba biti obojena isto kao token *0* u naredbi „`load s0, 0`", a ne kao u „`regbank a`", gdje je *a* naziv regbanke. S obzirom na to kako je sintaksni bojač koda u PicoBlaze Simulatoru strukturiran, nema jednostavnog rješenja za taj problem. U IDE-u za x86 asembler FlatAssembler takve se diskrepancije ne mogu dogoditi, jer on za sintaksno bojanje koda koristi isti algoritam koji koristi interno za parsiranje koda. Ali tako se ne može raditi ni za programski jezik AEC ni za PicoBlaze asembler, jer za oba ta jezika tokenizer briše komentare, a ne može se dopustiti da se pri sintaksnom bojanju koda obrišu komentari. Za sada postoji jednostavan *work-around*: kada su tokeni *a*, *b* i *c* heksadekadske konstante, mogu se umjesto njih pisati *0a*, *0b* i *0c*, i bit će obojane točno. Korisnik Discorda *ahnfelt*, autor programskog jezika Firefly, kaže da je on naletio na sličan problem pri sintaksnom bojanju programskog jezika Firefly. *Ahnfelt* predlaže da se u PicoBlaze Simulatoru u sintaksnom highlighteru deklarira globalna varijabla *lastHighlightedMnemonic*, da se u nju spre-

me mnemonike prije nego što ih se highlightira, te da se tokene *a* i *b* highlightira CSS klasom *flag* samo ako je *lastHighlightedMnemonic* jednak *regbank*, a *c* samo ako je *lastHighlightedMnemonic* jednak *jump*, *return* ili *call*. No, primijetite da sintaksno bojanje koda ni tada ne bi bilo točno. U naredbi „`jump c`", token *c* je heksadekadska konstanta (naredba znači „*Skoči na 13. naredbu u EPROM-u.*", jer je *c* heksadekadski broj 12, a naredbe se broje od nule), a u naredbi „`jump c, abort`" token *c* je zastavica *carry* (naredba znači: „*Ako je zastavica carry postavljena u jedinicu, skoči na label koji se zove 'abort'.*"). Da bismo znali je li *c* u PicoBlazeovom asemblerskom kodu heksadekadska konstanta ili zastavica, ne moramo samo znati prethodni token, nego i idući. Takve stvari ne smetaju parseru, ali su vrlo zahtjevne pri sintaksnom bojanju koda. Kad je autor, nekoliko tjedana nakon što je napisao „*Regbanks-Flags Test*", pisao primjer „*Preprocessor Test*", pojavio se i problem da sintaksni highlighter nakon znakova veće (>) i manje (<) umeće točka-zarez (;), čineći programe koji koriste te operatore sintaksno netočnima. O tome postoji upozorenje na početak programa „*Preprocessor Test*". Naknadno je dodano da *syntax highlighter* odbija *highlightirati* programe koji sadrže znakove <, > i &, s porukom o pogrešci: „*Sorry about that, but syntax highlighting of the programs containing `<`, `&`, and `>` is not supported yet.*". Za sada ovo pitanje nije riješeno. Nedavno je autorov suradnik sa sveučilišta u Argentini *agustiza* otkrio još jedan problem u PicoBlaze.html-u, naime, da način na koji je napisan CSS uzrokuje layout trashing i usporava cijeli program oko 12 puta (više o tome može se pročitati na kraju ovog teksta). Nakon toga, ispostavilo se da „*Download Hexadecimal*" ne funkcionira u WebPositiveu. O tome su alertom upozoreni korisnici WebPositivea koji pokušavaju skinuti heksadekadsku datoteku. Vjerojatno onda ne funkcionira ni u Safariju.

Dok je pisan potprogram PicoBlaze.html, postojale su mogućnosti dodatnih problema, no do njih nije došlo. Recimo, postojala je velika vjerojatnost da će TOR Browser odbijati *fetchati* PicoBlazeove asemblerske programe s GitHub profila (s domene raw.githubusercontent.com) iz JavaScripta koji se nalazi na web-stranici (na domeni flatassembler.github.io), da je to jedan od načina na koji TOR Browser štiti od *cross-site scripting* napada. Međutim, kada je isprobano *fetchanje* primjera asemblerskih programa u TOR Browseru, radilo je. Otvoren je *thread* na forumu o tome. U svakom slučaju, TOR Browser ne pruža očekivanu zaštitu.

2. TreeNode.js – sadrži JavaScript klasu pod nazivom *TreeNode*, koja sadrži metode vezane za evaluaciju parsiranih aritmetičkih izraza, metodu za ispis LISP-ovih izraza radi debugiranja parsera, te metode za pretragu struktura koje radi pretprocesor. Ta datoteka ima 100 redaka koda.

3. assembler.js – radi semantičku provjeru asemblerskog koda (recimo, je li prvi argument naredbe *load* uistinu registar) pomoću strukture koje radi parser te spaja strukturu koju radi parser i strukture koje radi pretprocesor u strojni kod u heksadekadskom obliku. Također radi neke sintaksne provjere koje parser ne radi, recimo, nalazi li se između dva argumenta naredbe *load* zarez. Ima 1050 redaka koda. Pretvoriti strojni kod u heksadekadskom obliku u binarni oblik (kakav razumije PicoBlaze) nije lagano u JavaScriptu, jer najmanja jedinica memorije koja se u JavaScriptu može adresirati jest byte, 8 bitova, a svaka naredba u strojnom kodu PicoBlazea je 18 bitova, što nije cijeli broj byteova. Ali je zato strojni kôd u heksadekadskom obliku iznimno lagano pretvoriti u binarni kôd kakav razumije PicoBlaze pomoću Xilinxovih alata. Xilinxov asembler isto ispisuje heksadekadske datoteke, a ne binarne. Za razliku od ostalih potprograma, assembler.js i simulator.js svoje rezultate ne vraćaju kao povratnu vrijednost funkcije, nego ih pišu u globalne varijable. Potprogram assembler.js piše svoje rezultate u globalni objekt *machineCode*, odakle ih potprogram simulator.js čita.
4. parser.js – Parser je dio compilera (u ovom slučaju, compilera za asemblerski jezik) koji drugim dijelovima compilera kaže koja je riječ u programskom jeziku gramatički povezana s kojom drugoj riječi. To radi tako što pravi strukturu zvanu AST, *abstract syntax tree*, apstraktno sintaksno stablo. Parser za programski jezik AEC dugačak je 950 redaka, dok datoteka parser.js sadrži 125 redaka. Zapravo, jedino što je nužno parsirati u asembleru za PicoBlaze jesu aritmetički izrazi. Algoritam napisan u datoteci parser.js ide ovako:
    1. Računalo treba pronaći parove otvorenih i zatvorenih zagrada u nizu koji je dao tokenizer. Parovi otvorenih i zatvorenih zagrada nalaze se, naime, u aritmetičkim izrazima te kao oznaka da je ono što se nalazi u registru pokazivač. Kada računalo nađe neki par zagrada, ono treba prebaciti ono što je između zagrada u novi niz, obrisati to iz originalnog niza, i pokrenuti rekurziju s novim nizom kao argumentom. Ako zagrade nisu dobro zatvorene, računalo treba javiti poruku o pogrešci. U parseru za AEC zadano je da se i zagrade obrišu iz sintaksnog stabla. U asemblerskom jeziku za PicoBlaze to ne bi imalo smisla, budući da zagrade imaju značenje da je u registru pokazivač, pa bi se time samo zakomplicirao asembler. Potprogram u parser.js zato za svaki par zagrada umeće čvor s tekstom „ ( ) “, i njegova su djeca (polje *children* iz klase *TreeNode*) niz koji vrati rekurzija.
    2. Računalo treba proći kroz niz koji je dao tokenizer i za svaku riječ provjeriti nalazi li se na popisu mnemonika (tako se tradicionalno zovu glagoli u asemblerskom jeziku) ili pretprocesorskih direktiva. Ti se popisi nalaze u datoteci PicoBlaze.html. Ako se riječ na koju je računalo upravo naišlo nalazi jednom od tih popisa, a ta riječ nije jednaka *enable*

ili *disable* i da je dužina niza jednaka jedinici (jer *enable* i *disable*, osim što mogu biti glagoli, mogu biti i, recimo to tako, prilozi glagola *returni*), računalo treba premjestiti sve između tog glagola i znaka za novi red (isključivo) u novi niz. Zatim računalo treba pokrenuti rekurziju i proglasiti ono što rekurzija vrati djecom čvora u kojem je taj glagol. To funkcionira zato što svaka rečenica u asemblerskom jeziku počinje glagolom te, osim u aritmetičkim izrazima, ne postoji lingvistička rekurzija, to jest, u asemblerskom jeziku ne postoje složene rečenice.

3. Računalo treba parsirati aritmetičke izraze. Prvo se treba riješiti unarnih operatora, njih se detektira kao tokeni + (plus) i – (minus) ispred kojih se ne nalazi ništa (prvi token u nizu), ili se ispred njih nalaze tokeni , (zarez), ( (otvorena zagrada) ili token koji sadrži znak za novi red. Zatim treba konstruirati lambda-funkciju *parseBinaryOperators* koja prima niz operatora te prolazi originalni niz u potrazi za njima i kada ih nađe, njihove susjedne tokene proglašava njihovom djecom i briše iz niza. Ta se lambda-funkcija prvo poziva za niz samo s operatorom potenciranja, ^, jer on ima najveći prioritet, zatim se poziva za operatore množenja i dijeljenja, * i /, te konačno za zbrajanje i oduzimanje, + i -. Parser za programski jezik AEC mora paziti na razliku između lijevo-asocijativnih operatora i desno-asocijativnih operatora (kao što su ternarni uvjetni operator `?:`, operator pridruživanja `:=` i skraćena pridruživanja +=, −=, *= i /=). No, budući da su svi aritmetički operatori lijevo-asocijativni, to u parseru za asemblerski jezik nije potrebno.

Na kraju se dobije asemblerski kod u obliku LISP-ovog S-izraza, koje JavaScript u datoteci PicoBlaze.html za potrebe traženja grešaka u asembleru ispisuje na preglednikovu konzolu JavaScriptovom naredbom *console.log*.

Kasnije je na početak potprograma parser.js dodan dio za parsiranje if-grananja, if-else-grananja i while-petlji, algoritmom u biti identičnim onim što se koristi u compileru za AEC, uz jedinu bitnu razliku da potprogram u parser.js ne podržava *elseif* naredbu. Zapravo, isprva mu je bilo zadano da izvrti dvije for-petlje: jednu koja parsira if-grananja, a drugu, poslije nje, koja parsira while-petlje. Također mu je bilo zadano da prije no što parsira if-grananje ili while-petlju, provjeri da ga nije roditeljska rekurzija već isparsirala, tako da provjeri da li je polje children u čvoru s tekstom „*while*” ili „*if*” prazno, te da parsira samo ako je prazno. Kasnije je to izmijenjeno da se parsiranje i *if* i *while* naredbi vrši u jednoj *for*-petlji te su izbačene provjere jesu li te naredbe već parsirane (to se ne može dogoditi ako se sve to radi u jednoj *for*-petlji). Time je malo pojednostavljen kod, a funkcionalnost bi trebala ostati ista. Tako da parser.js sada ima 247 redaka.

Za C++ (možda i za JavaScript) postoje mnogi frameworksi koji su namijenjeni tome da olakšaju pisanje parsera, među njima su najpoznatiji YACC i BISON. U ovom radu nisu korišteni za parsiranje AEC-a. Korisnik Reddita *Fofeu* tvrdi da je doktor informatike koji specijalizira teoriju compilera. On kaže da, osim što frameworksi za pisanje parsera skraćuju kôd parsera 5 ili 10 puta, također odbijaju parsirati nekonzistentne gramatike, i da je to izuzetno važno [12]. Po njemu, velika je vjerojatnost da je gramatika AEC-a zapravo nekonzistentna, ali da to još nije uočeno.

Da se PicoBlaze asemblerski kôd parsirao pomoću alata kao što je BISON (recimo da postoji BISON kompatibilan s JavaScriptom), kako bi mu se moglo objasniti onaj problem s *enable* i *disable*, da su te riječi nekada glagoli, a nekada prilozi, i da ih treba različito parsirati ovisno o tome? Otvoreno je pitanje o tome na forumu o izradi programskih jezika[2].

5. preprocessor.js – Prima strukturu koju pravi parser i određuje adrese labelsa (labelsi su oznake na koje je moguće skočiti naredbom *goto*, ili, kako se u asemblerskom jeziku zove ta naredba, *jump*). To je znatno lakše napraviti za PicoBlaze nego za Intelove i AMD-ove (x86) procesore, jer su za PicoBlaze sve naredbe u strojnom jeziku jednake dužine (18 bitova), pa možemo svaki put kada naiđemo na mnemoniku u AST-u povećati trenutnu adresu za jedan. Za x86, taj algoritam ne bi bio točan, jer, recimo, asemblerska naredba „`int 0x3`“ (u doba DOS-a služila za pozivanje debuggera) ima 8 bitova (u heksadekadskom formatu je cc), dok „`int 0x20`“ (u doba prvih verzija DOS-a služila za zatvaranje programa) ima 16 bitova (u heksadekadskom formatu je 20cd), a „`mov rax,[3*rbx+7]`“ ima 40 bitova (heksadekadski 8b485b440007). Potprogram preprocessor.js također izvršava i sprema rezultate direktiva *constant* i *namereg* (za preimenovanje registara u smislene nazive) u *Map* (klasa čiji objekti sadržavaju parove ključ-vrijednost, dostupna u standardnoj biblioteci JavaScripta od vremena Internet Explorera 11). Vidi se da se pretprocesor u kontekstu asemblera jako razlikuje od pretprocesora u kontekstu compilera. U compilerima pretprocesor se pokreće još prije tokenizera. U asemblerima pretprocesor se pokreće nakon parsera, ali prije jezgre asemblera. U mnogim je asemblerima pretprocesor Turing-potpun (Turing-complete), što je rijedak slučaj u višim programskim jezicima (to još vjerojatno vrijedi jedino za PERL). Zapravo, to se u višim programskim jezicima smatra lošim jer programske jezike ne treba moći parsirati samo compiler za taj programski jezik, nego i drugi alati za programiranje. Jedna od čestih kritika PERL-a je upravo *Only PERL can parse PERL.*. Potprogram preprocessor.js ima 140 redaka.

   Kasnije je dodana i FlatAssemblerova naredba *display* u pretprocesor asemblera za PicoBlaze u PicoBlaze Simulatoru. Ta naredba za vrijeme asembliranja ispisuje stringove ili kon-

stante na terminal, da se može provjeriti jesu li konstante dobro izračunate. Također su dodana if-grananja, if-else-grananja i while-petlje, slične kao što postoje u FlatAssembleru. Glavna razlika između tih stvari u FlatAssembleru i u asembleru za PicoBlaze je u tome što se u if-grananjima i while-petljama u PicoBlaze asembleru ne mogu nalaziti mnemonike ili naredba *address*, jer bi se onda pojavio problem s izračunavanjem adresa labela. U FlatAssembleru, recimo, mogu postojati programi kao što je ovaj:

```
address 0
firstLabel:
if firstLabel=secondLabel
    load s0,s0
endif
secondLabel:
```

Na tako nešto, FlatAssembler upada u beskonačnu petlju, jer čim se izračuna adresu labele *secondLabel*, ona se promijeni. Taj je problem riješen tako da se zada pretprocesoru da se u if-grananjima i while-petljama mogu nalaziti samo pretprocesorske naredbe, a ne i mnemonike. Neke funkcije if-grananja i while-petlja time se gube, ali neke ostaju, a lagano je za implementirati. FlatAssemblerove makro-naredbe bile bi još korisnije nego if-grananja i while-petlje.

Nakon implementacije *display* i *if* i *while*, uočen je jedan veoma neobičan bug u pretprocesoru. Naime, pretprocesorska naredba „`display a`" (za prelazak u novi red, jer je ASCII kod znaka za novi red 10, a 10 je u heksadekadskom a) ne funkcionira ako je u vrijeme asembliranja UART isključen. Problem nije točnije dijagnosticiran (kako pretprocesor uopće može znati je li UART uključen ili isključen).

Pretprocesor, prije nego što krene procesirati program, definira dvije konstante: *PicoBlaze_Simulator_in_JS* i *PicoBlaze_Simulator_for_Android*. Ukoliko pretprocesor misli da je pokrenut u internetskom pregledniku, *PicoBlaze_Simulator_in_JS* postavlja u 1, a *PicoBlaze_Simulator_for_Android* u 0. Ukoliko pretprocesor detektira da je pokrenut u programu *PicoBlaze_Simulator_for_Android* (više o njemu na kraju ovog teksta), tako što je deklariran globalni objekt *PicoBlaze*, koji u *PicoBlaze_Simulator_for_Android* služi kao sučelje pomoću kojeg komuniciraju potprogrami pisani u JavaScriptu s potprogramima pisanima u Javi (potprogrami pisani u Javi vide ga pod nazivom *WebAppInterface*), to je *vice versa*. Gotovo svi asembleri danas imaju pretprocesorske naredbe koje omogućuju asembliranom programu da detektira kojom se verzijom asemblera asemblira, uglavnom i naprednije nego ove što su ugrađene u simulator PicoBlazea. Ta mogućnost *assembler sniffinga* objavljena je u

verziji v2.8.3. Nedugo nakon što je inplementirano, radu na PicoBlaze Simulatoru pridružio se *agustiza* sa sveučilišta u Argentini.

Compiler za AEC nema pretprocesor. Jedino u njemu što bi donekle podsjećalo na pretprocesor je to što driver, ako su prve dvije riječi u nizu koji vrati tokenizer „`#target WASI`" ili „`#target browser`", postavlja da globalna varijabla *compilation_target* bude jednaka toj drugoj riječi, i onda izbriše te prve dvije riječi iz niza (prije nego što ga preda parseru). Budući da tokenizer briše komentare, prije rečenice „`#target WASI`" ili „`#target browser`" mogu postojati komentari, ali ne mogu postojati nikakve deklaracije. Naime, postoje dvije veoma različite okoline u kojima se WebAssembly (koji ispisuje compiler za AEC) može pokretati, to su internetski preglednik (*browser*) i WASI (*WebAssembly System Interface*, što je pokušaj da se napravi standard pomoću kojeg je moguće izrađivati command-line programe koji će se vrtjeti na više operacijskih sustava koristeći WebAssembly). Te okolike, između ostalog, zahtijevaju da se globalne varijable deklariraju na različit način. A svi programi koje ispisuje compiler za AEC koriste globalnu varijablu zvanu *$stack_pointer*. To je, dakle, bio *quick-and-dirty fix* da se compiler za AEC može koristiti ako se cilja WASI.

6. simulator.js – Taj se potprogram pokreće u zasebnoj dretvi kad pritisnemo tipku *play* ili *fast forward*, a u istoj dretvi ako pritisnemo tipku *single step*. On čita strojni kod u heksadekadskom obliku koji je u globalni objekt *machineCode* (deklariran u PicoBlaze.html) upisao potprogram assembler.js, te simulira PicoBlaze pišući i čitajući iz memorije (globalni objekt memory tipa *Uint8Array* deklariran u PicoBlaze.html), registara (niz, zvan *registers*, od dva globalna objekta tipa *Uint8Array* deklarirana u PicoBlaze.html, te globalna varijabla *PC*, *program counter*, koja pokazuje na naredbu u memoriji koja se trenutno izvršava) i zastavica (globalnih nizova *flagC*, *flagZ* te globalnih varijabli *flagIE* i *regbank*), pišući u izlaze (globalni objekt *output*), čitajući, koristeći DOM, ulaze iz one tablice s 256 HTML-ovih *input*-a, te upravljajući stogom *callStack*. Naknadno je dodano da za svaku naredbu provjerava je li na njoj breakpoint (to je moguće tako što globalni objekt *machineCode*, osim heksadekadskih stringova koji predstavljaju naredbe u strojnom kodu, sadrži i redni broj linije asemblerskog koda odakle naredbe dolaze, pa potprogram simulator.js može provjeriti nalazi li se redni broj linije asemblerskog koda odakle dolazi trenutna naredba strojnog koda u globalnom nizu *breakpoints*). Potprogram simulator.js ima 690 redaka. Isprva je bilo u planu da se jezgra simulatora napiše u AEC-u, a ne u JavaScriptu (preko compilera za AEC koji cilja WebAssembly, standardizirani JavaScript bytecode), no odlučeno je da se to ipak ne radi

tako. Naime, compiler za programski jezik AEC kompatibilan je samo s najnovijim internetskim preglednicima, ne može ciljati niti Microsoft Edge, a, *rebus sic stantibus*, danas tako nešto nije opcija ako se želi da web-aplikacija bude popularna. Uostalom, sigurno bi se naletjelo na neke probleme vezane za komunikaciju potprograma pisanih u JavaScriptu i potprograma pisanih u AEC-u, tako da je upitno da li bi bilo jednostavnije pisati taj simulator u AEC-u.

Pri pisanju simulator.js ispostavilo se da je on daleko sporiji od drugih simulatora PicoBlazea, o čemu je pisano pred kraj ovog teksta. Također je nastao problem sa zastavicama - kako se zastavice (flags) na PicoBlazeu ponašaju kada se promijeni *regbank*: da li zadržavaju svoj sadržaj ili postoje zasebne zastavice za svaki *regbank* (kao na računalu Z80, koji je po mnogo čemu sličan PicoBlazeu, ali o njemu ima daleko više informacija na internetu). U simulator.js na kraju je uprogramirano da za svaki *regbank* postoje zasebne zastavice (kao na Z80), a ispod tablice sa zastavicima dodana je napomena „*I have good reasons to think the emulation of flags is unrealistic, especially when it comes to REGBANKs.*". Također je postojao problem da je i u assembler.js i u simulator.js bio uprogramiran krivi *opcode* (Ono što se u asemblerskom jeziku zove mnemonika, u strojnom jeziku zove se *opcode*. Riječ *opcode* dolazi od ***op****eration* ***code***.) za mnemoniku *star* (***st****ore* ***ar****gument*, stavi podatak u registar u drugoj *regbanki*, naime, zamišljeno je da glavni program koristi jednu regbanku, a funkcije koriste drugu regbanku, i da se argumenti koje se šalju funkcijama spremaju u drugu regbanku), ali to je brzo dijagnosticirano i ispravljeno kad je primjer *Binary to Decimal* pokrenut na pravom PicoBlazeu (uspoređena je heksadekadska datoteka koju ispisuje asembler PicoBlazea s heksadekadskom datotekom koju ispisuje Xilinxov asembler).

Korisnik GitHuba *agustiza* koristio je PicoBlaze Simulator na sveučilištu u Argentini te je uočio da je pogrešno implementirana *bit-shifting* operacija *sra*, te je napravljen *pull request* s ispravkom tog buga.

Trenutno *agustiza* radi na tome da simulator.js poštuje načela koja se uče na kolegiju Razvoj programske podrške po objektivno orijentiranim načelima, recimo, da se ne koriste globalne varijable i da se koriste čiste funkcije.

7. tokenizer.js – Tokenizer je dio compilera koji kaže drugim dijelovima compilera gdje završava koja riječ, a gdje počinje druga, u programskom jeziku. Riječi u programskom jeziku zovu se tokeni. U programskim jezicima riječi ne moraju nužno biti odvojene razmacima, nego većina programskih jezika koristi malo kompleksnije mehanizme odvajanja riječi. Potprogram tokenizer.js ima 95 linija. Tokenizer za AEC, za usporedbu, ima 260 linija. Jedna od glavnih razlika između algoritma koji je korišten za programski jezik AEC i algoritma

koji je korišten za PicoBlaze asemblerski kod je ta što u PicoBlazeovom asemblerskom kodu nije pokušavano zapisivati broj stupca (*column number*), jer su svi redovi u asemblerskom jeziku kratki, pa podatci o stupcima gdje počinje koji token neće previše pomoći u traženju pogreške. Također, tokenizer za AEC ne smatra znak za novi red tokenom, dok tokenizer za PicoBlazeov asemblerski jezik smatra. Tokenizer za AEC mora se brinuti o escape-sequence znakovima u stringovima (\" i sl.), dok tokenizer za PicoBlazeov asembler ne mora. U compileru za AEC tokenizer zamjenjuje tokene poput `'a'` odgovarajućim ASCII vrijednostima (što možda i nije bila dobra ideja, jer poruke o pogreškama tipa *unexpected token* zbog toga mogu biti nejasne), u asembleru za PicoBlaze to radi potprogram TreeNode.js.

Za vrijeme pisanja PicoBlazeovog asemblerskog programa nazvanog „*Regbanks-Flags Test*", pojavio se idući problem. Programi kao što je ovaj:

```
address 0
load s9, " "
```

Prva naredba znači „*Počni pisati strojni kod od adrese 0.*" (i svaki PicoBlaze asemblerski program mora počinjati takvom naredbom), a druga naredba znači „*U registar s9 učitaj ASCII vrijednost razmaka.*". Kad se PicoBlaze simulatoru zadalo da to asemblira, dobila se ovakva poruku o pogrešci: „*Line #2: The AST node 'load's should have exactly three child nodes (a comma is also a child node).*". To je na brzinu riješeno tako što je umjesto druge naredbe napisano:

```
load s9, 20 ;Space
```

Naime, 20 je heksadekadski ASCII kod od razmaka (ASCII kod od razmaka je 32, a 32 je u heksadekadskom sustavu 20). Ovakvi programi nisu se dali asemblirati zbog nekog buga u tokenizeru, ali to je ispravljeno.

Osim tih datoteka, u GitHub repozitoriju nalaze se slike napravljene u GIMP-u (Linuxov Paint) i Inkscapeu (Linuxov Publisher) te primjeri programa za PicoBlaze koje je moguće dohvatiti klikajući na *examplese*. Slike koje predstavljaju *play*, *pause*, *stop*, *fast forward* i *single step* su u SVG formatu i uređivane u Inkscapeu. Ikona za *Assembler Test* u GIF je formatu i uređivana u GIMP-u. Ikone za primjer *Hexadecimal Counter* te ikona koja predstavlja točku prekida (*breakpoint*) isto su uređivane u GIMP-u, ali spremljene su u PNG formatu, jer ih PNG format bolje sažima nego GIF format. Pozadina je fotografija mikroprocesora PicoBlaze, posvijetljena u GIMP-u i spremljena kao GIF.

22. kolovoza 2023. godine korisnik GitHuba *agustiza* napravio je *pull request* u kojem se nalaze upute JEST-u (JavaScriptski framework koji služi za automatska testiranja) kako da automatski

testira tokenizer, koji je prihvaćen od autora. *Agustiza* je napisao da misli da bi automatske testove bilo mnogo lakše raditi da su potprogrami JavaScriptski moduli, a ne da se uključuju sa `<script src="...">`, no time bi se slomila kompatibilnost s Firefoxom 52 (jer je prva verzija Firefoxa koja podržava module Firefox 60), a važno je da simulator PicoBlazea funkcionira u njemu. *Agustiza* je zadao BabelJS-u da prepiše (*rewrite*) prethodno napisane JavaScriptske potprograme u module u kojima je svaka funkcija *export*ana prije nego što ih uključi naredbom *require* u NodeJS programe za testiranje. Kasnije su dodani automatsko testiranje parsera, evaluacije aritmetičkih izraza od strane potprograma TreeNode.js te malo automatskog testiranja simulator.js-a. Međutim, *test coverage* još je uvijek jako nizak, i, ako netko želi dalje raditi na ovom projektu, nužno je ne oslanjati se isključivo na automatske testove, već se mora raditi i ručno testiranje, po mogućnosti i u Firefoxu 52 i u modernom browseru.

U compileru za AEC još postoji, kao i u compilerima za većinu programskih jezika, semantički analizer. To je dio compilera koji „lovi" gramatički netočne rečenice koje prolaze kroz parser, ali koje bi srušile jezgru compilera da dođu do nje. Takvih izraza, koji se na prvi pogled čine gramatički ispravnima, a zapravo nisu, ima i u ljudskim jezicima, i zovu se gramatičke iluzije. Najpoznatiji primjer takvog izraza jesu komparativne iluzije, rečenice tipa „*More people have been to Russia than I have.*", „*Više je ljudi bilo u Rusiji nego što sam ja bio.*". Na prvi se pogled ta rečenica doima ispravna, no, ako se pokuša odrediti preciznije značenje, ispostavit će se da nešto s tom rečenicom sintaksno ne valja. Da bi poredbena rečenica imala smisla, ako je subjekt glavne surečenice u množini, a predikat joj je glagol s nultom valencijom (kao glagol biti u egzistencijalnom značenju), poredbena surečenica ne može imati isti taj predikat, ali u jednini. Parser u našem mozgu, očito, kao ni parseri u većini compilera, nije napravljen da lovi sve sintaksne greške. Ipak, semantički analizer nije potreban za asemblerski jezik, jer je na asemblerskom jeziku vjerojatno nemoguće konstruirati rečenicu kao što je „*More people have been to Russia than I have.*", iako su takve rečenice moguće u višim programskim jezicima.

Ako se radi neki viši programski jezik (a ne asembler), nikad se ne može biti siguran da će se compiler ponašati smisleno u svim mogućim situacijama. Primjerice, na internetskom forumu može se informirati na koje se sve probleme može naletjeti kad se implementira ternarni uvjetni operator `?:` [2]. Očito je da mnogi programski jezici s tim operatorom imaju problema. Recimo, PHP ga neispravno parsira, on ga parsira kao lijevo-asocijativni operator, a treba ga se parsirati kao desno-asocijativni operator (što onemogućuje da se pomoću operatora ?: u PHP-u skraćeno pišu *else-if* izrazi)[16]. Compiler AEC koji prevodi programski jezik AEC na x86 asemblerski jezik krivo prevodi taj operator na asemblerski jezik. Naime, on prevodi drugi i treći operand prije nego što prevede prvi operand, a to, kako upozorava korisnik StackOverflowa *rici*[3], može dovesti do greške kao

što je dijeljenje s nulom. Naime, netko bi se mogao pokušati zaštititi od dijeljenja s nulom tako što napiše „`d = 0 ? 0 : n / d`” (ako je varijabla d jednaka nuli; = u AEC-u znači jednakost, a ne pridruživanje kao što znači u C-u), a to neće funkcionirati u dijalektu mog programskog jezika koji se prevodi na x86 jer se „`n / d`” (treći operand) izračunava prije nego što se izračunava „`d = 0`” (prvi operand). Pa je *kaya3* odgovorio da je jedna od stvari na koju moramo paziti kad implementiramo `?:` da se naš compiler ponaša smisleno ako netko zabunom pokuša kao drugi i treći operand staviti strukture različitog tipa (Za AEC, smisleno bi bilo da semantički analizer to ne propusti do jezgre compilera.). Pri provjeri kako će se AEC compiler ponašati na takav programski kod, dobila se ovakva poruka o pogrešci:

*Line 10, Column 29, Internal compiler error: Some part of the compiler attempted to compile an array with size less than 1, which doesn't make sense. Throwing an exception!*

Iako programski kod koji mu je zadan na kompajliranje nije imao nizove. Ova poruka o pogrešci spadala bi u kategoriju *not even wrong*, ne može se smisliti kako bi neka greška u compileru dovela do te poruke. U biti, taj kôd kojim je testirano kako će compiler AEC reagirati na to da netko stavi strukture različitog tipa kao drugi i treći operand operatora `?:`, *triggerao* je dva buga u compileru AEC: jedan u semantičkom analizeru, a jedan u jezgri compilera. Onaj u jezgri compilera *triggerao* se samo ako su strukture prazne. Sigurno postoji još bezbroj takvih situacija u kojima se AEC compiler neće ponašati smisleno. Pa, ipak, viši programski jezici razlog su zašto računala mogu sve ono što mogu danas. Da postoje samo strojni jezici i asemblerski jezici, internet gotovo sigurno ne bi postojao.

U nekim programskim jezicima još mogu postojati i rečenice kao što je „*Time flies like an arrow.*” (Je li *flies* ovdje glagol ili imenica? Je li *like* ovdje glagol ili veznik? O tome ovisi kako će se parsirati ta rečenica.). Za takve programske jezike compileri između tokenizera i parsera imaju leksički analizer, koji parseru naznačuje koja je riječ koje vrste. Takve su rečenice moguće u C-u i C++-u, i to je poznato kao *typedef problem*. On je pojava da, izvan konteksta, `prvi*drugi;` (točka-zarez je dio izraza) može značiti i „*Deklariraj pokazivač koji se zove 'drugi' koji će pokazivati na instancu klase ili strukture koja se zove 'prvi'.*”, ali može značiti i „*Uzmi vrijednosti varijabli koje se zovu 'prvi' i 'drugi', pomnoži ih, stavi rezultat na sistemski stog i onda ga zanemari.*” [13]. Drugo značenje je, naravno, besmisleno (zanemari li se da se operator množenja `*` u C++-u može preopteretiti u nešto sa side-effectsima), ali je gramatički moguće. Rečenice tipa „*Time flies like an arrow.*” nisu moguće niti na PicoBlazeovom asemblerskom jeziku (osim što one riječi enable i disable mogu biti i glagoli i prilozi, ali to se trivijalno riješi u parseru) niti u programskom jeziku AEC (pretpostavlja

se da je tako), zato ni u PicoBlaze Simulator ni u compiler za taj programski jezik nije ugrađen leksički analizer.

## 4.1. Primjeri programa za PicoBlaze

Web-aplikacija nudi osam primjera programa za PicoBlaze. Oni se nalaze, kao i njihov popis, u JSON formatu, na GitHub profilu, te ih potprogram PicoBlaze.html dohvaća koristeći JavaScriptovu naredbu *fetch*. Prvi se primjer zove *Fibonacci Sequence*. On ispisuje Fibonaccijeve brojeve koji stanu u jedan byte (koliko su veliki PicoBlazeovi registri, svih 32), dakle, manje od 256. One koje se mogu prikazati u BCD (*binary coded decimal*, tako se u 8 bitova mogu prikazati brojevi manji od 100) formatu tako i ispisuje, a one koji se ne mogu, ispisuje u heksadekadskom formatu. Također koristi bitovne operacije da bi izbrojao koliko ima jedinica u binarnom zapisu svakog od tih brojeva. Kad završi, javlja da je završio naredbom *return*, što ne bi prošlo na pravom PicoBlazeu. Da bi ispisao broj u novi red, on ispisuje na port sa sljedećom adresom, što na pravom PicoBlazeu isto ne bi prošlo. Ima 116 redaka, uključujući brojne komentare.

| | | | |
|---|---|---|---|
| **15** | 00 | 07 | 00 |
| **16** | 00 | 13 | 00 |
| **17** | 00 | 03 | 00 |
| **18** | 00 | 08 | 00 |
| **19** | 00 | 21 | 00 |
| **1a** | 00 | 03 | 00 |
| **1b** | 00 | 09 | 00 |
| **1c** | 00 | 34 | 00 |
| **1d** | 00 | 02 | 00 |
| **1e** | 00 | 10 | 00 |
| **1f** | 00 | 55 | 00 |
| **20** | 00 | 05 | 00 |
| **21** | 00 | 11 | 00 |
| **22** | 00 | 89 | 00 |
| **23** | 00 | 04 | 00 |

***Slika 4.2**: Dio izlaza programa "Fibonacci Sequence". 7. broj u Fibonaccijevom nizu jest broj 13, i on u svom binarnom zapisu (1101) ima 3 jedinice. 8. broj u Fibonaccijevom nizu jest 21, i on u svom binarnom zapisu (10101) isto ima 3 jedinice. Deveti broj u Fibonaccijevom nizu je 34, i on u svom binarnom zapisu (100010) ima dvije jedinice. 10. broj u Fibonaccijevom nizu jest broj 55, koji u svom binarnom zapisu (110111) ima 5 jedinica. A 11. broj u Fibonaccijevom nizu jest broj 89, koji u svom binarnom zapisu (1011001) ima 4 jedinice.*

Program *Gray Code* koristi bitovne operacije da bi binarne brojeve pretvarao u Grayev kod i iz njega. To je način kodiranja brojeva koji se često koristi u digitalnoj elektronici, ima svojstvo da se

susjedni brojevi razlikuju samo u jednoj binarnoj znamenki. Recimo, brojanje od 0 do 10 u binarnom sustavu ide (promijenjene znamenke u susjednim brojevima su boldirane, a lijeve nule, koje nije potrebno pisati, u zagradama su): (000)0, (000)**1**, (00)**10**, (00)1**1**, (0)**100**, (0)10**1**, (0)1**10**, (0)11**1**, **1000**, 100**1**, 10**10**. U Grayevu kodu ti brojevi su: (000)0, (000)**1**, (00)**1**1, (00)1**0**, (0)**1**10, (0)11**1**, (0)1**0**1, (0)10**0**, **1**100, 110**1**, 11**1**1. To svojstvo je korisno jer se ne mora računati na to da su računala koja izmjenjuju poruke potpuno sinkronizirana, da jedno računalo počne čitati broj sa žice tek kad ga je drugo računalo već dovršilo mijenjati. To je osobito bilo važno za računala iz 1940-ih, na kojima implementiranje protokola sinkronizacije (Manchestersko kodiranje...) nije bilo jednostavno ili čak ni moguće. Program *Gray Code* koristi algoritam opisan na engleskoj Wikipediji[17], te se vrti u beskonačnoj petlji čitajući s ulaza. Ima 25 redaka.

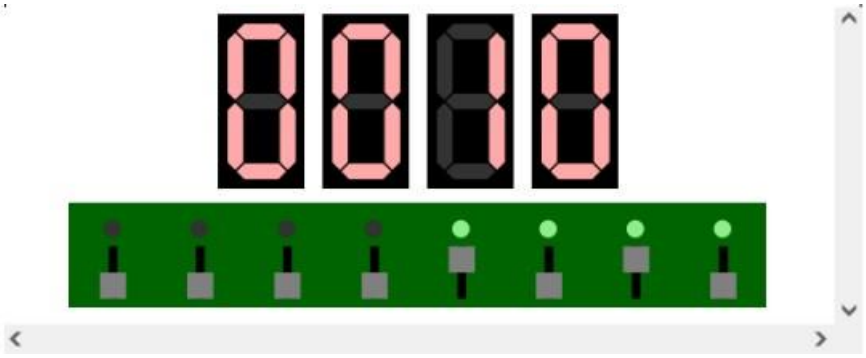
Program *Assembler Test* je besmislen program koji koristi sve naredbe koje podržava assembler za PicoBlaze, da bude lakše testirati assembler. Ako se pokrene u simulatoru, vrti se u beskonačnoj petlji. Ima 83 reda.

```
1.  ;Examples of the assembly syntax:
2.  address 0
3.  ;Data directives...
4.  beginningOfDataDirectives: ;Label
5.  load s0, s1
6.  load s1, 14'd / 2
7.  star s2, s3
8.  star s3, 1 + 2 * 3
9.  namereg sf, stack_pointer ;Preprocessor
10. store s4, (stack_pointer)
11. store s5, FF
12. fetch s6, (sa)
13. fetch s7, A
14. input s8, (sb)
15. input s9, (1 + 2) * 3
16. output sa, (sc)
17. output sa, -1 + 15'd / 2
18. constant eight, 15'd/2 ;Rounding
19. outputk eight, a
20. jump endOfDataDirectives
21. regbank a
22.
```

***Slika 4.3**: Program 'Assembler Test' besmislen je program koji se dobije uz simulator PicoBlazea, njemu je jedini cilj ispoljiti greške u asembleru ukoliko one postoje*

Naknadno su dodani primjeri *Binary to Decimal* i *Hexadecimal Counter*. *Binary to Decimal* postavljen je kao prvi primjer s lijeva. On čita 8-bitni binarni broj unesen pomoću onih SVG-ovskih prekidača, pretvara ga u decimalni broj i rezultat prikazuje na sedam-segmentnim pokaznicima, te se vrti u beskonačnoj petlji. Pretvaranje binarnog broja u decimalni radi se algoritmom za pretvaranje binarnog broja u BCD, uz izmjenu da pazi nalazi se broj između 0 i 99, između 100 i 199 ili je više

od 200 (8-bitni binarni brojevi mogu biti najviše 255). Uz to on zapošljava i LED-ice tako što na njima prikazuje broj pretvoren u Grayev kod. *Binary to Decimal* ima 54 retka. Isproban je na pravom PicoBlazeu i radio je. On koristi regbankse i zastavice, ali ne oslanja se na neku pretpostavku o tome kako se zastavice ponašaju kad se promijeni regbank, tako da to nije dokaz da je emulacija zastavica u simulatoru PicoBlazea napravljenim za ovaj rad realistična što se tiče regbanksa.



***Slika 4.4**: Primjer izvršavanja programa "Binary to Decimal." Preko prekidača mu je unesen binarni broj (0000)1010 (prekidač gore, naravno, označava jedinicu, a prekidač dolje nulu), a on ga je pretvorio u dekadski broj 10 te je to ispisao na 7-segmentnom displeju. Također ga je pretvorio u Grayev kod (0000)1111, te je to ispisao pomoću LED-ica (primijetite da su zadnje 4 LED-ice upaljene, dok su prve četiri ugašene).*

*Hexadecimal Counter* je program koji pali i gasi LED-ice te prikazuje heksadekadske brojeve na 7-segmentnim pokaznicima. Nakon svih tih primjera dodan je i primjer *Decimal to Binary*, koji pretvara dekadske brojeve u binarne, a dekadski brojevi se u njega upisuju koristeći UART, a isto tako se iz UART-a čitaju binarni brojevi koji su rezultati. *Decimal to Binary* najkompliciraniji je od tih primjera programa za PicoBlaze mikroprocesor: ima 283 retka. *Decimal to Binary* je sada drugi po redu s lijeva u popisu primjera u simulatoru.

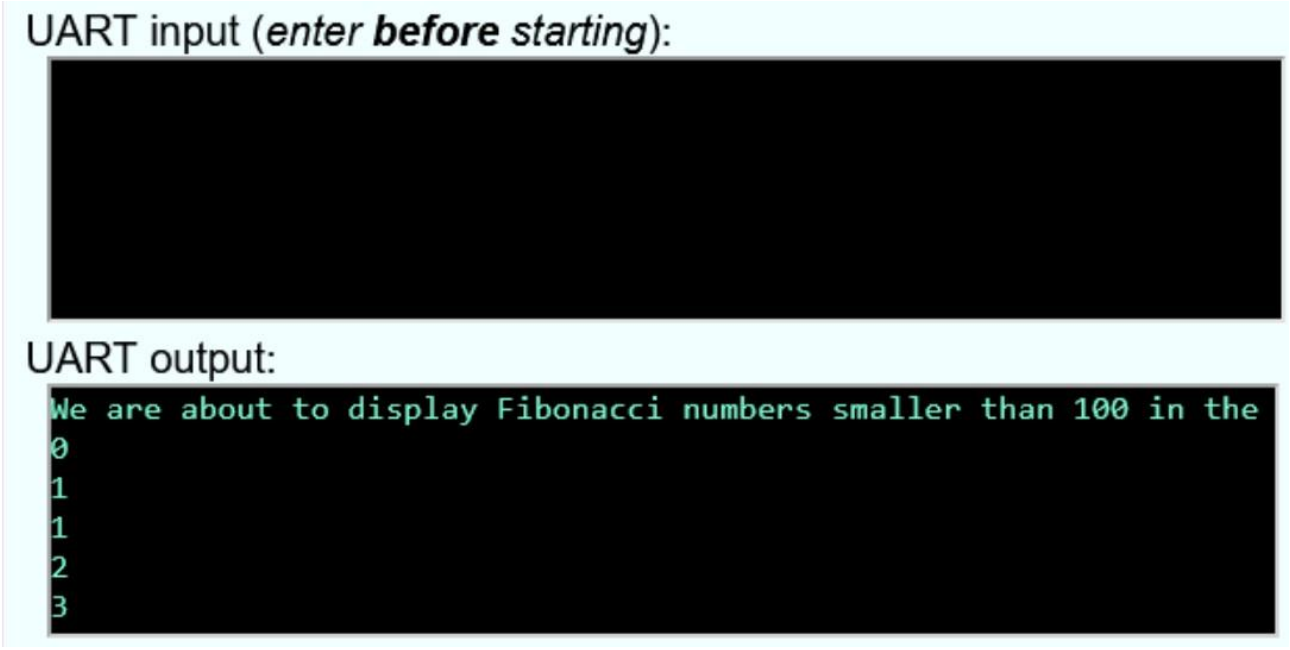
U program je upisan i primjer *Regbanks-Flags Test*. To je program koji provjerava kako se zastavice na PicoBlazeu uistinu ponašaju kad se promijeni *regbank*. Ima 211 redaka, a jezgra mu je ovo:

```
regbank a
load s0, 0
sub  s0, 0
regbank b
load s0, 1
sub  s0, 0
regbank a
jump z , success
jump nz, failure
```

Naredba „`regbank a`” znači „*Postavi regbanku A kao trenutnu regbanku.*” (*regbank* je ovdje mnemonika: glagol, a ne imenica). Naredba „`load s0, 0`” znači „*Učitaj u registar s0, u trenutnoj regbanki, broj 0.*”. U dijalektima asemblera koji ciljaju na ugrađene sustave (*embedded systems*) za učitavanje vrijednosti u registar obično se koristi mnemonika *load* (engleska riječ za *učitati*), a u asemblerskim dijalektima koji ciljaju PC-e i mobitele obično se koristi mnemonika *mov* (od latinskog *movere*, *premjestiti*). Naredba „`sub s0, 0`” znači „*Oduzmi vrijednost 0 od registra s0.*”, mnemonika *sub* dolazi od latinske riječi za *oduzeti*, *subtrahere*, od *sub* (ispod) i *trahere* (vući). Ako se nakon izvršenja te naredbe u registru nalazi vrijednost nula (kao u ovom slučaju), zastavica z u trenutnoj regbanki postavlja se u jedinicu. Naredba „`regbank b`”, naravno, znači „*Postavi regbanku B kao trenutnu regbanku.*”. Zatim se u registar *s0* učitava vrijednost 1, pa, kad se ponovno izvrti naredba „`sub s0, 0`”, zastavica *z* postavlja se u nulu (a ne u jedinicu, kao zadnji put). Zatim se PicoBlaze vraća u regbanku A i ispituje je li zastavica *z* postavljena u jedinicu ili u nulu.

Na koji će label skočiti (*jump*) nakon što se izvrti taj dio programa? Postoji stav da PicoBlaze u oba regbansa koristi iste zastavice i tada skače na label *failure*. U simulatoru PicoBlazea napravljenim za ovaj rad skače na *success*. Program *Regbanks-Flags Test* ispisuje svoje rezultate na UART, a, u slučaju da to ne upali, označava ih i LED-icama. Tako da se pokretanjem tog programa na pravom PicoBlazeu može provjeriti kako se zastavice na PicoBlazeu uistinu ponašaju kad se promijeni regbanka.

Nekoliko tjedana nakon napisanog *Regbanks-Flags Testa* znatno je unaprijeđen pretprocesor asemblera za PicoBlaze, i te izmjene u pretprocesoru testira program *Preprocessor Test*. Program *Preprocessor Test* za vrijeme asembliranja, koristeći if-else-grananja i while-petlji, izračunava brojeve u Fibonaccijevu nizu manje od 100 i ispisuje ih na terminal koristeći novododanu naredbu *display* (preuzetu iz FlatAssemblera). Program *Preprocessor Test* ne sadrži mnemonike i ne prevodi se ni u kakav strojni kod. On je dodan na kraj, kao 8. primjer.




***Slika 4.5**: Stanje terminala nakon asembliranja primjera "Preprocessor Test"*

Nakon tih 8 primjera slijedi link na ZIP komprimiranu mapu s programima pisanim na dijalektu programskog jezika AEC koji cilja x86, većina s mnogo umetnutog asemblerskog koda, i link na upute kako na Linuxu isprobati analogni sat pisan tim dijalektom AEC-ovog programskog jezika. To je dodano jer bi danas svaki programer trebao znati osnove x86 asemblerskog koda (asemblerski kod računala na kojem radi i za kojeg pravi programe), jer znati PicoBlaze asemblerski kôd nije zamjena za to. Analogni sat pisan onim dijalektom programskog jezika AEC koji cilja WebAssembly može se pokrenuti u modernom internetskom pregledniku, ali pokrenuti onaj drugi je znatno kompliciranije, i zato su linkani na upute kako to napraviti na Linuxu (a upute kako to napraviti na Windowsima nalaze se u ZIP komprimiranoj mapi). Compiler za AEC koji cilja na x86 ispisuje asemblerski kôd kompatibilan s i486, ali se onaj analogni sat zbog naredbi koje se koriste u umetnutom asemblerskom kodu (*movzx*...) ne može pokrenuti na i486, nego samo na i586 i novijima (dakle, recimo, može u DosBoxu, danas najčešće korištenom simulatoru DOS-a). Pri pisanju programa roseForDOS.aec, bilo je važno da se u umetnuti asembler ne ubaci ni jedna instrukcija nekompatibilna s i486, tako da bi se roseForDOS.aec iz te ZIP komprimirane mape trebao moći pokrenuti na i486 (iako to nije isprobano jer je danas teško naći i 486-icu koja radi). A, u načelu, svaki se program za stare x86 procesore može pokrenuti na današnjim računalima. DOS se može pokrenuti na današnjim računalima, i neki programi funkcioniraju bez problema, a neki samo s manjim problemima. Recimo, sve što je isprobano u QBASIC-u u DOS-u na današnjem računalu, radilo je. Igrica *Prince of Persia*, ako se pokrene u DOS-u na današnjem računalu, može se pokrenuti i

odigrati prvi level, ali se zaglavi pri prijelazu iz prvog levala na drugi (u DosBoxu, koji simulira i stari hardware, može se cijela odigrati). Međutim, Windows 3.11 ne može se pokrenuti na današnjem računalu. Pred kraj prekida instalaciju s porukom da se procesor ne može prebaciti u 32-bitni način rada. Bilo bi zanimljivo istražiti zašto se to događa. Na internetu na više mjesta piše da se Windows 95 ne može pokrenuti na današnjem računalu jer u njemu postoji bug koji mu onemogućava da se pokrene na računalu s više od 480 MB RAM-a, ali da se Windows 98 može. Windows XP znatno se manje oslanja na BIOS nego Windows 98 i radi daleko više pretpostavki o tome kako hardware funkcionira, i zato se Windows XP ne može pokrenuti na današnjem računalu.


***Slika 4.6**: Analogni sat pisan u programskom jeziku AEC pokrenut u DosBoxu*

Na Redditu je otvoren subreddit o PicoBlazeu gdje se mogu dodati novi primjeri.

Tako da u flexboxu s primjerima ima ukupno 10 *divova*: osam primjera PicoBlaze programa i dva *diva* s linkovima. U slučaju da netko ovaj PicoBlaze simulator pokuša pokrenuti u internetskom pregledniku u kojem se JavaScript ne uspije izvrtjeti, u tom flexboxu nalazi se poruka o tome i link na web-stranicu TOR Browsera. TOR Browser, naime, koristi SpiderMonkey compiler za JavaScript, koji je poznat po tome da najbolje optimizira kod, znatno bolje nego V8 (koji koristi Chrome),

a važno je da se JavaScript u PicoBlaze simulatoru dobro optimizira, jer se pri simulaciji mnogo puta izvršava isti JavaScript.

## 4.2. Mane simuliranja PicoBlazea u JavaScriptu

Naravno, simulator PicoBlazea koji je napravljen u JavaScriptu ima i svoje nedostatke naspram *fully-featured* simulatora. Prvo, on ne pokušava interpretirati VHDL, tako da ne može simulirati PicoBlazeove s modificiranim VHDL kodom (osim ako se ne promijeni JavaScript, ali opet to neće pomoći da se nađu greške u VHDL kodu). Drugo, grafičko sučelje mu je mnogo manje efektivno nego sučelje koje pružaju najčešće korišteni simulatori PicoBlazea. Dobro je poznato da je u web-aplikacijama teško napraviti dobro korisničko sučelje. Korisnik foruma *atheistforums.org* zvan *bennyboy* predložio je da se pregledaju primjeri dobrog web-dizajna na web-stranici CSS frameworka *Bootstrap*, da bi se napravio ljepši dizajn za ovaj PicoBlaze Simulator koji je lagan za implementirati na webu [7]. Treća je mana što je nemoguće napraviti realistični tajming. Jedna od prednosti PicoBlazea za korištenje u ugrađenim sustavima, gdje trebaju mali procesori, jest upravo to što je lagano odrediti koliko će se dugo neki komad koda izvršavati ako znamo na koliko MHz-a radi (PicoBlaze može raditi na frekvenciji do oko 130 MHz), svaka instrukcija traje točno dva takta. U JavaScriptu je to nemoguće simulirati, jer JavaScript, na primjer, ima sakupljanje smeća koje se pokreće (što se JavaScriptskog programa tiče) nedeterministički. Uz to, na danas prosječnom računalu ovaj simulator može izvršiti oko 20 instrukcija po sekundi (u fast-forwardu u Firefoxu, pregledniku koji najbrže izvršava JavaScript), daleko manje od 75.000.000 operacija u sekundi koji bi bili potrebni za realistični tajming. Jedan čovjek na internetskom forumu tvrdi da ga vjerojatno najviše usporava to što se simulacijska dretva prekine kad se izvrši jedna instrukcija, a ponovno postavlja kad se treba izvršiti nova, a na postavljanje i uništavanje JavaScriptove dretve troše se mnogi računalni resursi [12]. Dakle, da bi se postigao realističan tajming, moralo bi se posve preurediti simulator, a vjerojatno i napisati ga u drugom programskom jeziku (no teško bi se pokrenuo u internetskom pregledniku). Nastojalo ga se ubrzati tako što se manipulacija stringovima zamijenila bitovnim operacijama gdje je to jednostavno za napraviti, ali to nije urodilo plodom. Također, simulacija UART-a poprilično je nerealistična, a nije očito kako bi se u programskom jeziku JavaScript napravio dobar simulator terminala. Iako ovaj simulator nije dostatan za neke potrebe, zasigurno je dostatan za potrebe studenata i olakšat će im *hassle* instalacije naprednijih simulatora.

Korisnik foruma atheistforums.org zvan *bennyboy* ima dvije ideje kako ubrzati simulator PicoBlazea. On misli da potprogram assembler.js znatno usporava to što se za svaki čvor u apstraktnom sintaksnom stablu mnogo puta provjerava je li riječ o registru, te da bi se to trebalo provjeriti samo jednom i rezultat te provjere pridružiti *boolean* varijabli [7]. On također misli da bi se u simulator.js trebao više koristiti *switch-case*, a manje *if-else*, jer se *switch-case* compilira u optimiziraniji asemblerski kod [7]. Bilo bi zanimljivo provjeriti teze da se *switch-case* compilira u brži asemblerski kod

nego *if-else*. *HappySkeptic* je pokušao analizirati performanse PicoBlaze simulatora dok se vrti program *Decimal to Binary* pomoću Chromeovih alata za programiranje i došao je do zaključka da PicoBlaze Simulator provodi više od 99 % svog vremena crtajući SVG dijagrame [7] (koji su, naravno, za program *Decimal to Binary* beskorisni, jer program *Decimal to Binary* koristi UART za komunikaciju s korisnikom, a ne prekidače, LED-ice ili 7-segmentne pokaznike, koji se prikazuju SVG-ovima). U tom bi se slučaju simulator PicoBlazea, barem za programe kao što je *Decimal to Binary*, mogao znatno ubrzati tako da se omogući korisniku da *disable*a SVG-ove, kao što se sada može *disable*ati UART. No ovo objašnjenje je sporno. Naime da je tako, PicoBlaze simulator vrtio bi se neprihvatljivo sporo u WebPositiveu, koji veoma sporo crta SVG-ove (zato je SVG PacMan neigriv u WebPositiveu). A u WebPositiveu simulator PicoBlazea vrti se otprilike jednako brzo kao što se vrti u Chromeu.

30. kolovoza 2023. godine korisnik GitHuba *agustiza* dijagnosticirao je problem kao *layout trashing*. Kako on objašnjava, kad PicoBlaze upiše novu vrijednost registra u tablicu, tablica neprimjetno promijeni veličinu za nekoliko piksela, ali dovoljno da se poremeti layout i da browser mora cijelu web-stranicu renderirati iznova. Koristeći napredni CSS, kaže da je uspio ubrzati svoju verziju PicoBlaze simulatora 12 puta. Naime, na *agustiz*inom računalu u Chromeu u verziji programa pravljenoj za ovaj rad primjer *Fibonacci Sequence* izvrti se za oko 60 sekundi, a u *agustiz*inoj verziji za 5 sekundi. U planu je spojiti te dvije verzije PicoBlaze simulatora. *For the time being*, dodan je checkbox koji omogućuje korisniku da onemogući osvježavanje tablica s registrima i zastavicama u svakom koraku simulacije. Onemogućavanje da se tablice s registrima i zastavicama osvježavaju pri svakom koraku simulacije ubrzava *Fibonacci Sequence* približno dva puta (ne baš 12 puta, ali nešto ubrzanja bolje je nego ništa).

Budući da je simulator PicoBlazea bio neprihvatljivo spor ako se pokrene na mobilnom telefonu, razmišljalo se o pokušaju da se napravi Android aplikaciju gdje će simulator biti pisan u Javi (dakle, da autor ponovno napiše simulator.js, ali ne u JavaScriptu, nego u Javi). Java isto nije idealan jezik za pisanje simulatora (sakupljanje smeća...), ali neki uspješni simulatori ipak jesu pisani u Javi (jDosbox...), jer je Java ipak brža nego JavaScript (statičko tipiranje...). Napravljena je stoga native Android aplikacija koja će omogućiti mobitelu da se pretvara da je PicoBlaze. Ta aplikacija je omogućila samo to da asembler pisan u JavaScriptu (assembler.js, parser.js, preprocessor.js, tokenizer.js i TreeNode.js) i program pisan u Javi komuniciraju (budući da je JavaScript koji koristi asembler za PicoBlaze dosta napredan, Rhino ili Duktape ne bi bili od koristi da ga se pokrene, nego se baš morao koristiti JavaScript engine ugrađen u Android, koji ima komplicirano sučelje) te da program pisan u Javi dohvaća i parsira onaj JSON popis primjera i primjere koristeći frameworke Retrofit i GSON, pa se od te ideje odustalo. Ako netko želi nastaviti ovaj rad, dostupan je na autorovu GitHub

profilu. Mnogi na internetskim forumima kažu da, kad se s razvoja desktop ili mobilnih aplikacija prebace na razvoj web-aplikacija, kao da su se vratili dva desetljeća u prošlost. No nije tako. Napraviti korisničko sučelje od gumbova i labela uistinu je lakše kad se cilja mobitel nego kad se cilja web: kad se cilja mobitel, to se može napraviti bez kodiranja. Popuniti tablicu podatcima (kao što PicoBlaze Simulator puni tablicu podatcima iz globalnog objekta *machineCode*, strojnim kodom u heksadekadskom obliku i linijama asemblerskog koda odakle dolaze naredbe) već je podjednako teško na mobitelu i na webu. A dohvatiti nešto s interneta ili parsirati JSON daleko je lakše na webu nego na mobitelu.

# 5. ZAKLJUČAK

U ovom završnom radu napravljen je simulator za mikroprocesor PicoBlaze od tvrtke Xilinx. S pomoću napravljenog simulatora omogućeno je izvođenje i provjera rada asembler programa. Nakon što se program napiše, slijedi provjera sintakse. Zatim se riječima pisani program prevodi u strojni zapis, odnosno stvara se hex datoteka. S jedne strane, iz generirane hex datoteke je moguće provesti simulaciju. Simulator omogućuje prikaz rezultata na izlaznim uređajima kao što su LED i HEX pokaznik te omogućuje učitavanje podataka sa sklopki. S druge strane, hex datoteka se može preuzeti i poslati u memoriju stvarnog PicoBlaze mikroprocesora implementiranog na Xilinx FPGA čipu. Funkcionalnost rada simulatora dokazana je usporedbom rezultata simulatora i rezultata na stvarnoj maketi Nexys 3 tvrtke Digilent. Simulator se pokazao kao koristan alat za pisanje asembler programa mikroprocesora PicoBlaze.

# LITERATURA


[1.] Y. Fain, Programiranje Java, Dobar plan, Zagreb, 2013.

[2.] Internetski forum LangDev StackExchange, dostupan na: https://langdev.stackexchange.com/ [zadnji put posjećen 26. rujna 2023.]

[3.] Internetski forum StackOverflow , dostupan na: https://stackoverflow.com/ [zadnji put posjećen 26. rujna 2023.]

[4.] Internetski forum CodeReview StackExchange, dostupan na: https://codereview.stackexchange.com/ [zadnji put posjećen 26. rujna 2023.]

[5.] Internetski forum forum.hr, dostupan na: https://www.forum.hr/index.php [zadnji put posjećen 26. rujna 2023.]

[6.] Web-stranica simulatora PicoBlaze Debugger, dostupno na: http://www.ivysim.com/picoblaze/debug/guide/ [zadnji put posjećen 7. listopada 2023.]

[7.] Internetski forum AtheistForums, dostupan na: https://atheistforums.org/ [zadnji put posjećen 26. rujna 2023.]

[8.] K. R. Irvine, *Assembly Language for x86 Processors*, Pearson Education, New York, 2011.

[9.] D. Milićev, Objektno orijentisano programiranje na jeziku C++, Mikro knjiga, Beograd, 2001.

[10.] Predavanja iz kolegija Arhitektura računala, 2020.

[11.] PDF datoteka, Merlin, 2020.

[12.] Reddit, dostupan na: https://www.reddit.com/ [zadnji put posjećen 26. rujna 2023.]

[13.] J. Šribar, B. Motik, Demistificirani C++, Element d. o. o., Zagreb, 2010.

[14.] *w3schools* (o JaviScriptu), tutorijali, dostupno na: https://www.w3schools.com/ [zadnji put posjećen 26. rujna 2023.]

[15.] *Mozilla Developer Network*, dostupan na: https://developer.mozilla.org/en-US/ [zadnji put posjećen 26. rujna 2023.]

[16.] wiki.php.net, dostupan na: https://wiki.php.net/ [zadnji put posjećen 26. rujna 2023.]

[17.] Wikipedia, dostupan na: https://en.wikipedia.org/ [zadnji put posjećen 26. rujna 2023.]

[18.] Dokumentacija o PicoBlazeu, dostupno na web-stranici sveučilišta u Alabami: https://www.eng.auburn.edu/~nelson/courses/elec4200/PicoBlaze/kcpsm6.pdf [zadnji put posjećen 7. listopada 2023.]

[19.] Web-stranica Fidex simulatora: https://www.fautronix.com/en/en-fidex-screenshots [zadnji put posjećen 7. listopada 2023.]

[20.] Arhivirana web-stranica tvrtke Mediatronix: https://web.archive.org/web/20051027184859/http://www.mediatronix.com/pBlazeIDE.htm [zadnji put posjećen 7. listopada 2023.]
[21.] Web-stranica KPicoSima: https://marksix.home.xs4all.nl/kpicosim.html [zadnji put posjećen 7. listopada 2023.]
[22.] Web-stranica OpbAsma: https://kevinpt.github.io/opbasm/ [zadnji put posjećen 7. listopada 2023.]

## SAŽETAK:

U ovom tekstu opisan je simulator PicoBlazea u programskom jeziku JavaScript. Taj se simulator može pokrenuti u modernim internetskim preglednicima[1]. U tekstu slijede detalji o tome kako je kako je izrađivan ovaj simulator te koje su njegove prednosti i mane u usporedbi s već postojećim simulatorima. Također se uspoređuju dijelovi simulatora s odgovarajućim dijelovima nekih drugih programa koje su ranije napravljeni (najviše s compilerom koji prevodi programski jezik AEC na WebAssembly). Nisu korišteni nikakvi radni okviri (frameworksi), kôd je pisan uglavnom u VIM-u (za manje izmjene) i Eclipseu (za veće izmjene), za uređivanje slika korišteni su GIMP i Inkscape, za traženje pogrešaka u programu korišteni su alati za programiranje koji se dobiju uz Firefox i (danas ugasli) internetski servis LGTM. Za formatiranje koda korišteni su Prettier (za HTML i CSS) i ClangFormat (za JavaScript). Posljednja verzija koja je napravljena samostalno je v2.8.3; nedugo nakon toga radu na simulatoru pridružio se *agustiza*, koji tvrdi da koristi ovaj simulator PicoBlazea na sveučilištu u Argentini.

Ključne riječi: assembler, emulator, HTML, JavaScript, PicoBlaze.

[1] `https://flatassembler.github.io/PicoBlaze/PicoBlaze.html`

## ABSTRACT:

Simulator for PicoBlaze

This text describes a simulator of the PicoBlaze soft-processor, written in the JavaScript programming language. This simulator can be run in modern Internet browsers[2]. The text explains the details of how the simulator was made and what the advantages and disadvantages are of that simulator compared to the existing simulators. Parts of the simulator are compared with similar parts of other programs the author has made earlier (mostly a compiler that compiles a programming language called AEC to WebAssembly). No frameworks were used, the code was written mainly in VIM (for minor changes) and Eclipse (for major changes), GIMP and Inkscape were used for image editing, programming tools that come with Firefox were used for debugging the program, and the LGTM internet service (which is no longer available) was also used to search for errors. Prettier (for HTML and CSS) and ClangFormat (for JavaScript) were used to format the code. The last version that was made without anyone's help is v2.8.3. Not long after the version v2.8.3 was released, a GitHub user called *agustiza*, who claims to be using this project at a university in Argentina, started contributing to the project.

Keyword: assembler, emulator, HTML, JavaScript, PicoBlaze

[2] `https://flatassembler.github.io/PicoBlaze/PicoBlaze.html`

# ŽIVOTOPIS

Teo Samaržija rođen je 1999. u Osijeku. Osnovnu školu završio je u Donjem Miholjcu (Osnovna škola August Harambašić), a u srednju školu išao je u Opću gimnaziju u Donjem Miholjcu i u Opću gimnaziju u Našicama (kraj 2. razreda i početak 3. razreda).

Često je sudjelovao na informatičkim natjecanjima. U 6. razredu osnovne škole osvojio je 3. mjesto na županijskom natjecanju, u 7. razredu 4. mjesto na državnom natjecanju, u 8. razredu 6. mjesto na državnom natjecanju (sve je to organizirao Infokup). U srednjoj školi se, kad je bio 3. razred, natjecao na HONI-ju, te je osvojio 15. mjesto. 2019. godine je, kao student prve godine, sudjelovao na STEM Games natjecanju iz programiranja, na kojem je njegova ekipa osvojila 7. mjesto. Osim na natjecanjima iz informatike, sudjelovao je, kad je bio 2. razred srednje škole, i na AZOO-vom državnom natjecanju iz latinskog jezika, gdje je osvojio 7. mjesto.

Upisao je 2018. godine FERIT u Osijeku, smjer Računarstvo, i trenutno je student treće godine.