Verifikacijsko okruženje za oblikovanje računalnog sustava temeljenog na procesoru arhitekture RISC-V

Grubišin, Luka

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1644

VERIFIKACIJSKO OKRUŽENJE ZA OBLIKOVANJE RAČUNALNOG SUSTAVA TEMELJENOG NA PROCESORU ARHITEKTURE RISC-V

Luka Grubišin

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1644

VERIFIKACIJSKO OKRUŽENJE ZA OBLIKOVANJE RAČUNALNOG SUSTAVA TEMELJENOG NA PROCESORU ARHITEKTURE RISC-V

Luka Grubišin

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1644

Pristupnik:	**Luka Grubišin (0036537921)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Josip Knezović

Zadatak: **Verifikacijsko okruženje za oblikovanje računalnog sustava temeljenog na procesoru arhitekture RISC-V**

Opis zadatka:

Proučiti postojeća rješenja za provjeru ispravnosti procesora arhitekture RISC-V te razviti vlastito verifikacijsko okruženje za izvođenje asemblerskih programa i usporedbu s referentnim modelom sustava iz programskog paketa SSPARCSS. Programski paket SSPARCSS je razvijen na FER-u a služi za poučavanje osnovnih koncepata arhitekture procesora i računalnih sustava. SSPARCSS omogućuje simulaciju i izvođenje programa za različite arhitekture procesora, uključujući i arhitekturu RISC-V na koju se fokusira ovaj zadatak. Nakon provjere ispravnosti razvijenog verifikacijskog okruženja na osnovu usporedbe izvođenja programa u odnosu na SSPARCSS, potrebno je definirati i razviti niz ispitnih programa i osnovni ponašajni model procesora arhitekture RISC-V koji će biti podvrgnut ispitnim programima u razvijenom verifikacijskom okruženu. Po potrebi napraviti i prilagodbe referentnog modela u SSPARCSS-u s ciljem učinkovite usporedbe s razvijenim verifikacijskim okruženjem.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

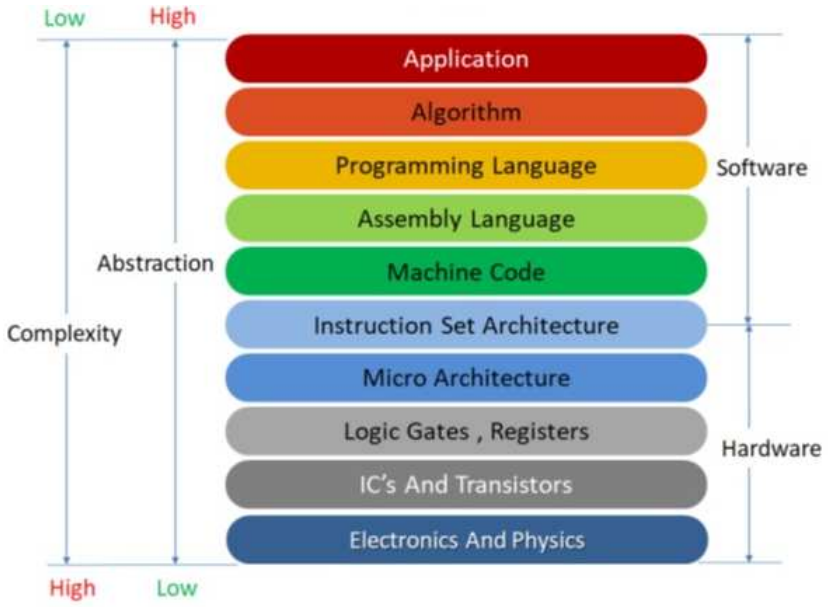
1. Uvod

Procesor je središnja komponenta svakog računalnog sustava. Zbog široke uporabe računala u za različite svrhe razvijene su brojne različite arhitekture procesora kako bi bili što bolje prilagođeni za specifičnu primjenu.

Računalni sustav, odn. izvođenje programa može se opisati na različite načine, s više ili manje detalja. Njihovi odnosi prikazani su slikom 1.1. U slučaju opisa procesora kao gradivne komponente sustava prikladna razina opisa je razina skupa instrukcija ISA (engl. *instruction set architecture*). Ovakav opis sastoji se od dva glavna dijela - skupa registara procesora i skupa instrukcija koje procesor može izvršiti. To je iz perspektive razvoja programske podrške odn. softvera i najkorisnije, jer većini programera nije potrebno detaljnije znanje o samoj implementaciji procesora, odn. poznavanje sustava na razinama ispod skupa instrukcija koje daju više detalja o sklopovskoj izvedbi sustava. Poznavanje njegovog skupa instrukcija i skupa registara dovoljno je za pisanje programa (minimalno na razini zbirnog jezika ili asemblera) koji će se izvršavati na njemu.

Na nižoj razini apstrakcije koristi se opis mikroarhitekture procesora koja opisuje osnovne gradivne elemente računalnog sustava i samog procesora kao što su registri, aritmetičko-logička jedinica, sklopovi za posmak, sklopovi za pristup memoriji i drugi elementi protočne strukture procesora, ali i složenije strukture poput sklopa za predviđanje grananja, priručnih memorija i sličnog. Ispod razine mikroarhitekture opisuje se njezina izvedba logičkim vratima, dok se na najnižoj razini arhitektura procesora opisuje na razini individualnih tranzistora i drugih elemenata.

Većina današnjih korisničkih računala koristi procesore arhitekture x86 ili njezinu proširenu verziju x86_64. Ova arhitektura je u vlasništvu tvrtki Intel i AMD te samo one razvijaju procesore s tom arhitekturom. Ona pripada porodici CISC (engl. *complex*



Slika 1.1. Različiti opisi arhitekture procesora i njihovi odnosi [1]

instruction set computer) arhitektura. Kao što sam naziv govori, arhitekture tipa CISC sastoje se od skupa instrukcija koje omogućuju izvođenje više povezanih funkcija u jednoj instrukciji, ali pod cijenu veće složenosti i duljine izvođenja.

Alternativu ovakvom pristupu predstavljaju računala s procesorima arhitekture tipa RISC (engl. *reduced instruction set computer*). Ovakvi procesori sadrže skup manjih i jednostavnijih instrukcija koje se lakše dekodiraju i brže izvode, a složene funkcionalnosti CISC instrukcija ostvaruju se programski. Zbog ove jednostavnosti, RISC arhitekture lakše su za razvoj i prilagodbu pa se ovakvi procesori često koriste u ugradbenim računalima, mobilnim uređajima i specijaliziranim računalima visokih performansi, a isto se tako često koriste i u edukacijske svrhe. Primjeri porodica RISC arhitektura su ARM, MIPS, SPARC, AVR i RISC-V. Posebno je zanimljiva arhitektura RISC-V koja je detaljnije opisana u nastavku.

2. Arhitektura skupa instrukcija RISC-V

2.1. Povijest i ciljevi arhitekture RISC-V

Razvoj arhitekture RISC-V započeli su profesori Krste Asanović i David Patterson i studenti Yunsup Lee i Andrew Waterman na američkom sveučilištu Berkeley u sklopu Laboratorija za paralelno računarstvo (engl. *Parallel Computing Laboratory, Par Lab*). Zbog velikog interesa za arhitekturu, 2015. osnovana je neprofitna organizacija RISC-V Foundation koja je preuzela razvoj specifikacije skupa instrukcija RISC-V ISA. 2020. godine u Švicarskoj je osnovana RISC-V International Association kako bi se omogućila međunarodna suradnja u razvoj [2]. RISC-V International danas broji oko 4000 članova u 70 zemalja svijeta [3].

Arhitektura je u početku bila razvijena za istraživačke i obrazovne svrhe, ali i zbog brojnih problema koje donosi korištenje komercijalnih ISA, uglavnom vezanih uz složenost samih arhitektura, ali i povezivanja sa softverom, popularnost samo u specifičnim primjenama, problema vezanih uz intelektualno vlasništvo i mogućnosti zabrane korištenja neke ISA iz političkih razloga (na primjer uvođenjem embarga) [4].

Iz tih razloga, RISC-V ISA je u potpunosti otvorena (engl. *open source*) i slobodna za korištenje. Bilo tko može pridonijeti njenom razvoju i bilo tko može napraviti implementaciju ISA što je čini idealnom za idealnom za hobiste i manje firme koje si ne mogu priuštiti licence za korištenje komercijalnih arhitektura. U opisu ISA stavljen je i velik naglasak na izbjegavanje detalja mikroarhitekture ili prilagodbu instrukcija za specifičnu implementaciju što olakšava jednostavnu simulaciju ili ostvarenje RISC-V ISA u FPGA, ASIC ili full-custom tehnologijama. Činjenica da je ISA razvijena od početka znači da je jednostavnija od brojnih postojećih arhitektura koje se razvijaju već desetljećima i postaju sve složenije i složenije kako bi održale kompatibilnost sa starim sustavima i uvele

nove funkcionalnosti. Modularan pristup dizajnu i korištenje ekstenzija omogućava laku prilagodbu specifičnim potrebama i primjenu RISC-V ISA u gotovo svakoj domeni.

2.2. Osnovni skup instrukcija RISC-V ISA i najčešća proširenja

Osnovni dio RISC-V ISA čini minimalni skup instrukcija i registara za rad s cijelim brojevima na koju se mogu dodavati različita proširenja za ostvarenje specifičnih funkcionalnosti ili poboljšanje performansi.

Postoje četiri ovakve arhitekture:

- RV32I - osnovna 32-bitna arhitektura
- RV64I - osnovna 64-bitna arhitektura
- RV32E - 32-bitna arhitektura sa smanjenim brojem registara za primjenu u malim mikrokontrolerskim sustavima
- RV64E - 64-bitna arhitektura sa smanjenim brojem registara za primjenu u malim mikrokontrolerskim sustavima

Osim ove četiri arhitekture, specifikacija predviđa i 128-bitnu RV128I ISA ako u budućnosti bude potrebe za proširenjem adresnog prostora na 128 bitova. U ostatku ovog rada, pretpostavlja se korištenje osnovne arhitekture RV32I.

Osnovna ISA dovoljna je ostvarenje iznimno jednostavnog, ali potpuno funkcionalnog računalnog sustava, ali joj nedostaju brojne druge često korištene i potrebne funkcionalnosti za koje su definirana proširenja.

Neke od najčešće korištenih proširenja su:

- Zifencei - Instrukcije za osiguravanje ispravnog poretka pristupa memoriji
- Zicsr - Instrukcije za rad s kontrolnim i statusnim registrima
- M - Instrukcije za cjelobrojno množenje i dijeljenje
- A - Atomarne instrukcije

- F - Instrukcije za rad s brojevima u zapisu s pomičnim zarezom jednostruke preciznosti
- D - Instrukcije za rad s brojevima u zapisu s pomičnim zarezom dvostruke preciznosti
- C - Komprimirane instrukcije za uštedu prostora u instrukcijskoj memoriji

U ostatku ovog rada koristit će se osnovna ISA RV32I sa Zicsr ekstenzijom.

2.3. Registri

2.3.1. Osnovni skup registara

RV32I ISA definira skup od 32 registra opće namjene od `x0` do `x31`. Osim njih, u procesoru je prisutno i programsko brojilo, no njemu nije moguće izravno pristupiti. Iako se radi o registrima opće namjene, RV32I ISA predviđa i alternativne uloge registara zbog lakše standardizacije i izrade programskih prevoditelja, operacijskih sustava i sličnih programa. Tako se primjerice, registar `x1`, alternativnog naziva `ra`, često koristi za spremanje povratne adrese potprograma, a registar `x2`, alternativnog naziva `sp`, kao pokazivač na vrh stoga. Također je važan i registar `x0`, odnosno `zero`. Čitanjem ovog registra uvijek se dobiva vrijednost nula, a pisanje u njega se ignorira što je iznenađujuće često korisna funkcionalnost. Ostali registri, njihovi alternativni nazivi i kratki opisi prikazani su u tablici 2.1.

Tablica 2.1. Osnovni registri arhitekture RISC-V

Ime registra	Alternativno ime	Svrha
`x0`	`zero`	Hardwired zero
`x1`	`ra`	Return address
`x2`	`sp`	Stack pointer
`x3`	`gp`	Global pointer
`x4`	`tp`	Thread pointer
`x5`	`t0`	Temporary / alternate link register
`x6`	`t1`	Temporary
`x7`	`t2`	Temporary

x8	s0/fp	Saved register / frame pointer
x9	s1	Saved register
x10	a0	Function argument / return value
x11	a1	Function argument / return value
x12	a2	Function argument
x13	a3	Function argument
x14	a4	Function argument
x15	a5	Function argument
x16	a6	Function argument
x17	a7	Function argument
x18	s2	Saved register
x19	s3	Saved register
x20	s4	Saved register
x21	s5	Saved register
x22	s6	Saved register
x23	s7	Saved register
x24	s8	Saved register
x25	s9	Saved register
x26	s10	Saved register
x27	s11	Saved register
x28	t3	Temporary
x29	t4	Temporary
x30	t5	Temporary
x31	t6	Temporary

2.3.2. Statusni registri

Proširenje Zicsr definira niz dodatnih statusnih registara kao i naredbe za rad s njima. Specifikacija definira velik broj registara kojima se može pristupiti na različitim razinama privilegije. RISC-V specifikacija definira tri glavne razine privilegija:

- Machine (M) - Najprivilegiranija, jedina koja mora biti implementirana
- Supervisor (S)
- User/Application (U) - Najmanje privilegirana

Jednostavni mikrokontrolerski sustavi obično koriste samo razinu privilegije M . Napredniji mikrokontroleri koriste razine M i S, dok je za pokretanje operacijskih sustava potrebno implementirati sve tri razine privilegija [5]. Model procesora korišten u ovom radu implementira samo M razinu privilegija i sljedeće statusne registre definirane u [5]:

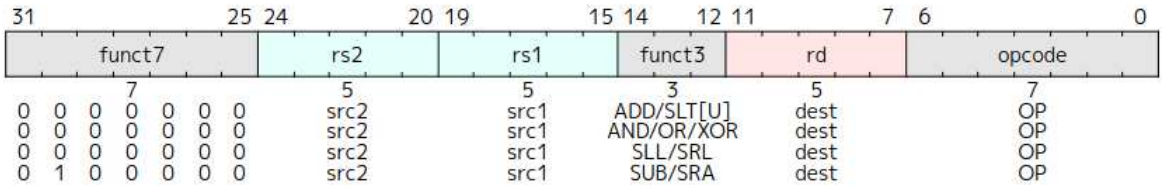
- `mvendorid` - ID prodavača
- `marchid` - ID arhitekture
- `mimpid` - ID implementacije
- `mhartid` - ID hardverske dretve
- `mstatus` - Statusni registar
- `misa` - ISA i proširenja
- `mie` - Registar za omogućavanje prekida
- `mtvec` - Bazna adresa programa za obradu iznimaka
- `mscratch` - Pomoćni registar za korištenje pri obradi iznimki. Najčešće se koristi za pohranu pokazivača na vrh stoga.
- `mepc` - Povratna adresa iznimke
- `mcause` - Uzrok iznimke
- `mtval` - Neporavnata adresa koja je uzrokovala iznimku, adresa instrukcije koja je uzrokovala iznimku ili slična informacija korisna pri obradi iznimke
- `mip` - Prekid na čekanju
- `mcycle` - Brojač ciklusa (64 bita)
- `minstret` - Brojač obavljenih instrukcija (64 bita)

2.4. Instrukcije

RV32I ISA definira četiri osnovna formata instrukcija - R, I, S i U. Osim njih, prisutni su i formati B i J koji su varijante S i U formata s drugačijim načinom kodiranja neposrednih brojčanih vrijednosti. Najnižih 7 bitova svake instrukcije sadrži operacijski kod što olakšava dekodiranje. Ostali bitovi mogu sadržavati indekse izvorišnih ili odredišnih registara, neposredne konstante ili proširenja operacijskog koda.

2.4.1. R-tip

Format tipa R koriste se kod aritmetičkih i logičkih instrukcija nad operandima zapisanima u registrima. Sadrže dva izvorišna registra `rs1` i `rs2` te odredišni registar `rd`. Osim registara, instrukcije sadrže polja `funct3` i `funct7` koja točno određuju instrukciju koja će se izvesti.



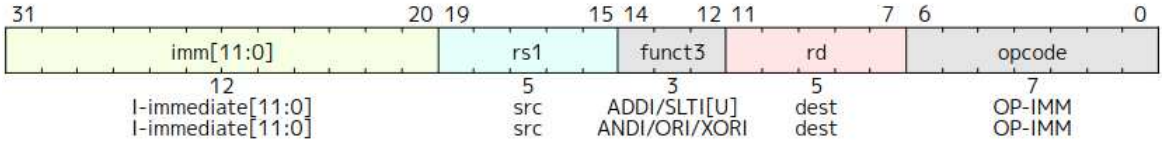
Slika 2.1. Instrukcije R-tipa [4]

Instrukcije koje koriste ovaj format su:

- Aritmetičke instrukcije ADD i SUB
- Instrukcije uspoređivanja SLT i SLTU
- Logičke instrukcije AND, OR i XOR
- Instrukcije posmaka SLL, SRL i SRA

2.4.2. I-tip

Format tipa I koristi se za aritmetičke i logičke instrukcije pri čemu je jedan operand zapisan u registru a drugi kao predznačno proširena 12-bitna neposredna vrijednost (eng. *immediate*) u instrukciji. Instrukcije I-tipa stoga osim polja `opcode` sadrže polja `rs1`, `imm[11:0]`, `rd` i `funct3`.

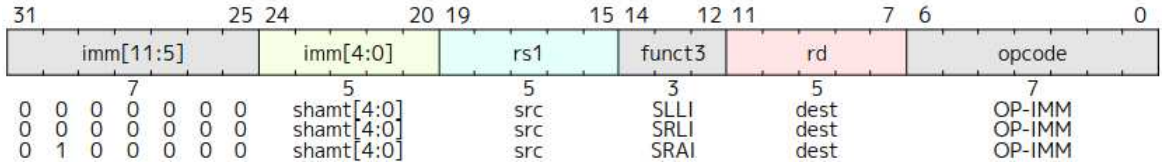


Slika 2.2. Instrukcije I-tipa [4]

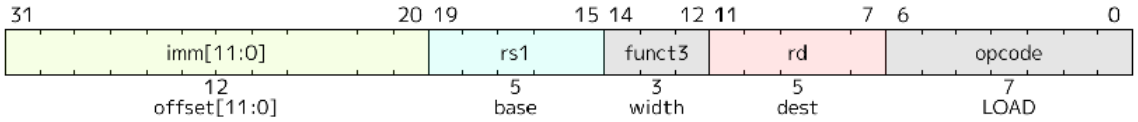
Instrukcije koje koriste ovaj format su:

- Instrukcija zbrajanja sadržaja registra s konstantom ADDI
- Instrukcije usporedbe sadržaja registra s konstantom SLTI i SLTIU
- Logičke instrukcije s konstantom ANDI, ORI i XORI
- Instrukcije posmaka SLLI, SRLI i SRAI
- Instrukcija za učitavanje podatka iz memorije u registar LOAD
- Instrukcija neizravnog bezuvjetnog skoka JALR

Instrukcije posmaka za konstantnu vrijednost SLLI, SRLI i SRAI koriste varijantu formata tipa I u kojoj se za zapisivanje konstante koristi samo 5 od predviđenih 12 bitova. Instrukcija LOAD u polje `funct3` zapisuje širinu podatka koji se učitava, a u polje `imm[11:0]` pomak u odnosu na baznu adresu zapisanu u `rs1`.



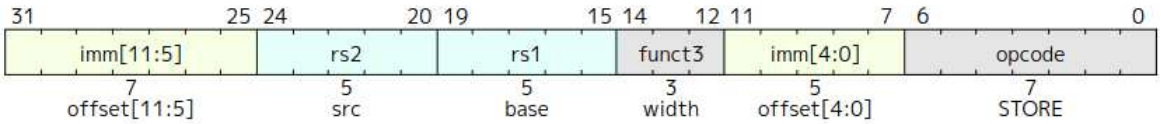
Slika 2.3. Instrukcije posmaka [4]



Slika 2.4. Instrukcija LOAD [4]

2.4.3. S-tip

Instrukcija STORE sprema sadržaj iz registra u memoriju i jedina je instrukcija tipa S u RV32I ISA. Instrukcije S-tipa stoga osim polja `opcode` sadrže polja `imm[11:5]`, `rs1`, `rs2`,

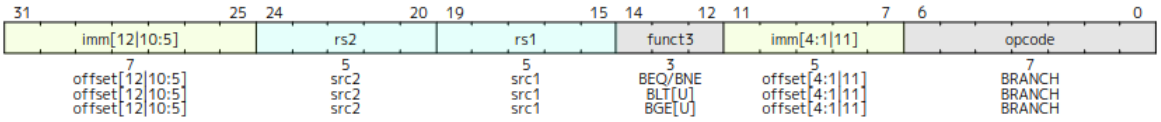


Slika 2.5. Instrukcija STORE [4]

`funct3` i `imm[4:0]`. Konstanta je zapisana u dva dijela u poljima `imm[11:5]` i `imm[4:0]`, dok polje `funct3` sadrži širinu podatka, kao i kod instrukcije LOAD.

2.4.4. B-tip

Format tipa B varijanta je tipa S s drugačijim poretkom bitova zapisane konstante prikazanim na slici2.6.

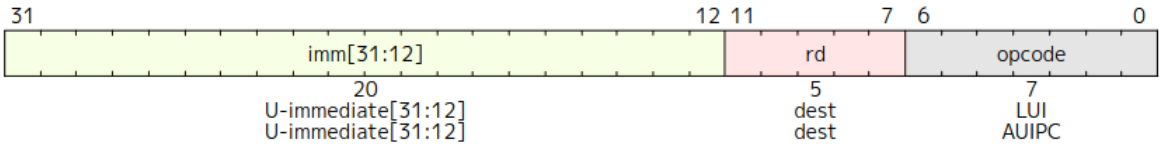


Slika 2.6. Instrukcije B-tipa [4]

Ovakav način zapisivanja konstante koristi se zato što B-tip koriste instrukcije uvjetnog skoka BEQ, BNE, BLT, BLTU, BGE i BGEU. Budući da instrukcije u memoriji moraju biti poravnate na 2 bajta, najniži bit adrese instrukcije uvijek će biti 0 pa ga nije potrebno uključiti pri zapisu konstante što omogućava skok na veću udaljenost.

2.4.5. U-tip

Instrukcije U-tipa sadrže samo tri polja: `imm[31:12]`, `rd` i `opcode`. Koriste se kako bi se omogućilo jednostavno upisivanje 32-bitnih vrijednosti u registre.



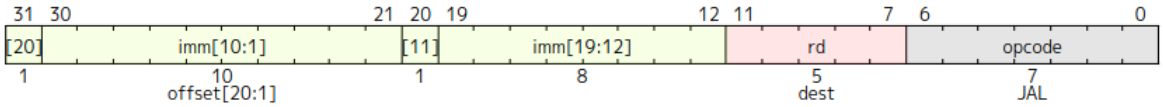
Slika 2.7. Instrukcije U-tipa [4]

Instrukcije koje koriste ovaj format su:

- LUI - load upper immediate
- AUIPC - add upper immediate to PC

2.4.6. J-tip

J-tip formata varijanta je U-tipa formata koji koristi instrukcija bezuvjetnog skoka JAL.

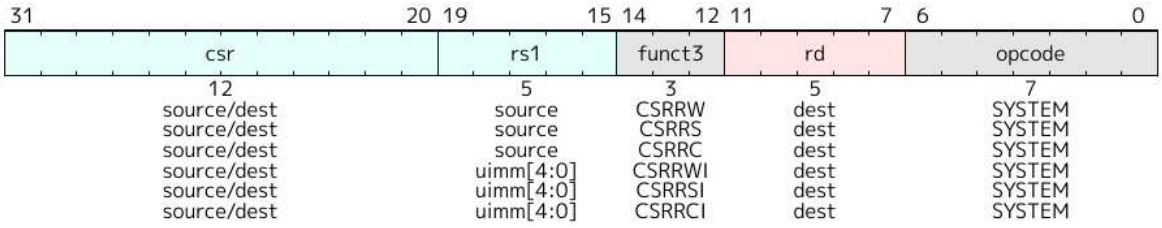


Slika 2.8. Instrukcije J-tipa [4]

Razlika između tipa U i tipa J analogna je razlici između tipa S i tipa B.

2.4.7. Instrukcije za rad s kontrolnim i statusnim registrima

Instrukcije za rad s kontrolnim i statusnim registrima koriste format nalik I-tipu, s time da je polje za 12-bitnu neposrednu vrijednost zamijenjeno 12-bitnim poljem kojim se specificira statusni registar kojem se pristupa.



Slika 2.9. Instrukcije za rad sa statusnim registrima [4]

Za instrukcije koje koriste neposrednu vrijednost, koristi se 5-bitno polje inače namijenjeno za izvorišni registar `rs1`. Proširenje Zicsr definira šest instrukcija za rad sa statusnim registrima.

- CSRRW atomički čita sadržaj danog statusnog registra, po potrebi ga proširuje nulama i zapisuje ga u registar `rd`, a nakon toga kopira sadržaj registra `rs1` u statusni registar.
- CSRRS atomički čita sadržaj statusnog registra, proširuje ga i upisuje u `rd`. Nakon toga se sadržaj registra `rs1` koristi kao maska za postavljanje bitova u statusnom registru.
- CSRRC radi jednako kao CSRRS, ali umjesto postavljanja briše bitove s pomoću maske u `rs1`.

- CSRRWI radi kao CSRRW, ali se umjesto sadržaja `rs1` koristi 5-bitna neposredno zadana vrijednost proširena nulom.
- CSRRSI radi isto što i CSRRSI, ali koristi neposredno zadanu vrijednost.
- CSRRCI radi kao i CSRRS, ali s neposrednom zadanom vrijednosti umjesto `rs1`

3. Programski paket SSPARCSS

SSPARCSS (engl. *Software Suite for Processor ARChitecture Simulation and Study*) je programski paket namijenjen za simulaciju jednostavnih računalnih sustava u obrazovne svrhe. Razvijen je na Fakultetu elektrotehnike i računarstva 2020. godine kako bi zamijenio stari programski paket ATLAS koji se u slične svrhe koristio od 1991. godine. Dvije glavne komponente SSPARCSS-a su asembler [6] i simulator [7].

Asembler služi za prevođenje programa pisanih u asemblerskom jeziku u izvršnu datoteku pogodnu za učitavanje u memoriju i izvođenje na određenom procesoru unutar simulatora. Asembler u SSPARCSS-u poprilično je fleksibilan kako bi mogao raditi s bilo kojim procesorom koji je opisan za simulaciju i za koji postoji konfiguracijska datoteka pisana u jeziku ADEL (engl. *Assembler DEscription Language*).

Simulator služi za simuliranje rada računalnih sustava opisanih u posebnom jeziku COMDEL [8] (engl. *COMponent DEscription Language*) koji je detaljnije opisan u nastavku. Za simulaciju su trenutno definirani računalni sustavi s procesorom arhitekture ARM i RISC-V.

3.1. Jezik COMDEL

Jezik COMDEL koristi se za opis računalnog sustava simuliranog u SSPARCSS-ovom simulatoru. Sličan je programskom jeziku C i jeziku za opis sklopovlja VHDL s dodatnim mogućnostima vezanim uz animaciju rada komponenti koje se opisuju. Struktura simuliranog računalnog sustava opisana je u datoteci s nastavkom *.system* u kojoj je opisan vršni prikaz sustava. U njoj se opisuje način povezivanja glavnih komponenti opisanih u zasebnim datotekama. Komponente se opisuju u datotekama s nastavkom *.comdel*. Dva glavna dijela opisa komponenti su ponašajni model opisan u procesnom bloku (nalik na

VHDL) i animacijski dio koji služi za prikaz rada komponente u simulatoru. Svaka komponenta u sebi također može sadržavati potkomponente čime se omogućava hijerarhijski razvoj sustava.

3.2. Izvođenje programa u simulatoru

3.2.1. Pisanje programa

Kako bismo mogli izvesti program u simulatoru, potrebno ga je napisati u mnemoničkom jeziku. Mnemonički programi pohranjuju se kao tekstualne datoteke s nastavkom *.a*. Svaki red mnemoničkog programa sastoji se od labele, instrukcije i komentara. Labela se nalazi na početku reda i služi za organizaciju programa i jednostavnije pisanje naredbi skoka u koje se umjesto odredište adrese može navesti labela koju će asembler pri prevođenju zamijeniti odgovarajućom adresom. Instrukcija slijedi nakon labele i piše se u skladu s pravilima opisanim u konfiguracijskoj datoteci asemblera za korišteni procesor. Komentari se pišu na kraju reda te započinju znakom ';'. Osim samih instrukcija programa, u instrukcijski dio linije mogu se smjestiti i asemblerske direktive. One se ne koriste pri izvođenju programa, već ih koristi asembler pri asembliranju. Asemblerske direktive također su specifične za svaku konfiguraciju asemblera, ali su obično vrlo slične za sve sustave. Najčešće korištene asemblerske direktive su:

- Direktivom ORG zadaje se adresa od koje će se u memoriji smjestiti dio programa ili podatci definirani koji su zapisani nakon nje.
- Direktivom EQU labeli se pridružuje konstanta kako bi se olakšalo pisanje programa. Asembler će pri prevođenju zamijeniti labelu stvarnom brojčanom vrijednošću labele.
- Direktive DB, DH, DW[1] koriste se za definiranje sadržaja memorije prije pokretanja programa. DB definira sadržaj jednog bajta memorije, DH jedne poluriječi (2 bajta), a DW jedne riječi (4 bajta). Ovisno o arhitekturi procesora koji se simulira, ove veličine mogu biti drugačije.
- Direktiva DSTR[1] u memoriju umeće nul-terminirani niz znakova (engl. *string*).
- Direktiva DS[1] definira prazan memorijski prostor popunjen nulama. Služi kako bi se korisniku olakšao pregled sadržaja memorije za vrijeme simulacije.

RISC-V specifikacija također definira svoje asemblerske direktive [9]. Trenutno su za SSPARCSS-ov asembler definirane:

- .equ radi isto što i EQU
- .align poravnava podatke ili program na adresu koja je višekratnik zadane potencije broja dva
- .byte, .half i .word odgovaraju direktivama DB, DH i DW
- .string radi isto što i DSTR
- .zero radi isto što i DS

3.2.2. Prevođenje i pokretanje

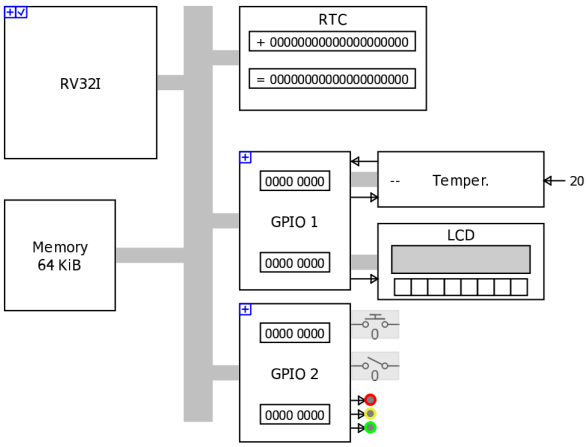
Nakon pisanja programa, potrebno ga je asemblirati s pomoću asemblerskog prevoditelja čime nastaje nova datoteka s nastavkom *.e* koja je pogodna za učitavanje u simulator. Zatim je u simulatoru potrebno odabrati model računalnog sustava i izvršnu datoteku koja sadrži program koji će se izvoditi. Simulator omogućava izvršavanje programa kontinuirano ili instrukciju po instrukciju te omogućava detaljan pregled stanja pojedinih komponenti sustava u danom trenutku kao i pregled sadržaja memorije. Prikaz komponenti definiran je na razini pojedine komponente i ovisi o konkretnom sustavu koji se simulira. U nastavku je detaljnije opisan računalni sustav temeljen na procesoru arhitekture RISC-V.

3.3. Polazišni sustav s procesorom RISC-V

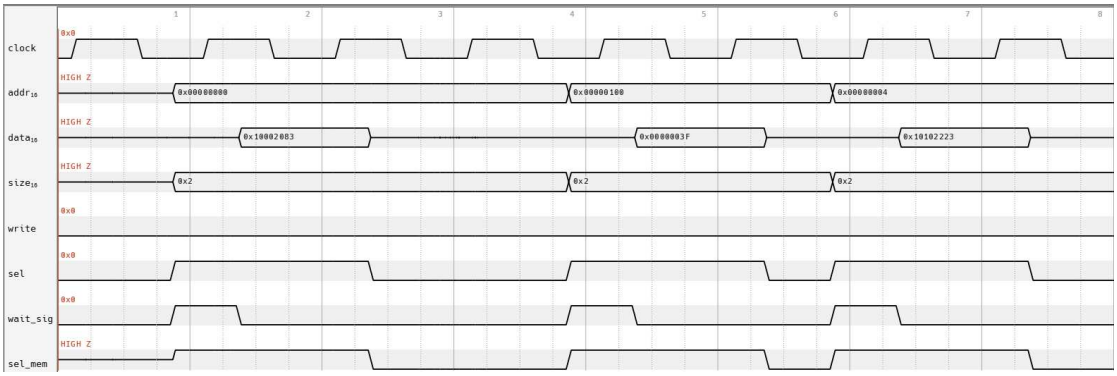
Polazišni sustav s procesorom RISC-V u simulatoru SSPARCSS prikazan je na slici 3.1.

Sustav se sastoji od procesora osnovne ISA RV32I s proširenjem Zicsr, adresnog dekodera, memorije od 64 KiB, dvije GPIO jedinice i RTC sklopa. Komponente su povezane adresnom i podatkovnom sabirnicom, zajedničkim signalom takta i različitim dodatnim upravljačkim signalima. Procesor memoriji pristupa preko dekodera koji čitajući adresu određuje kojoj se komponenti želi pristupiti. Memorija se nalazi na adresama od 0x00000000 do 0x0000FFFF. Pristup memoriji prikazan je vremenskim dijagramom 3.2.

[1] Direktivama DS, DH, DW, DSTR i DS najčešće prethodi direktiva ORG.



Slika 3.1. Polazišni sustav s procesorom RISC-V u simulatoru SSPARCSS



Slika 3.2. Vremenski dijagram pristupa memoriji

Na polovici niske poluperiode signala takta `clock`, procesor na adresnu sabirnicu postavlja adresu kojoj pristupa i postavlja signale `size`, `write` i `sel` ovisno o veličini podatka i radi li se o čitanju ili pisanju. Dekoder čita adresu i odmah prema njoj određuje kojoj se jedinici pristupa i podiže odgovarajući signal (u ovom slučaju to je `sel\_mem`). Memorija drži aktivnim signal `wait\_sig` sve dok ne postavi podatak na podatkovnu sabirnicu. Procesor na svaki rastući brid takta provjerava stanje signala `wait\_sig` i kada detektira da nije aktivan, čita podatke i spušta signal `sel`. Dekoder detektira da signal `sel` više nije aktivan i gasi vlastite kontrolne signale. Kada memorija, ili neka druga jedinica kojoj se pristupa detektira da njezina linija za odabir nije više aktivna, odspaja se s podatkovne sabirnice čime prijenos završava.

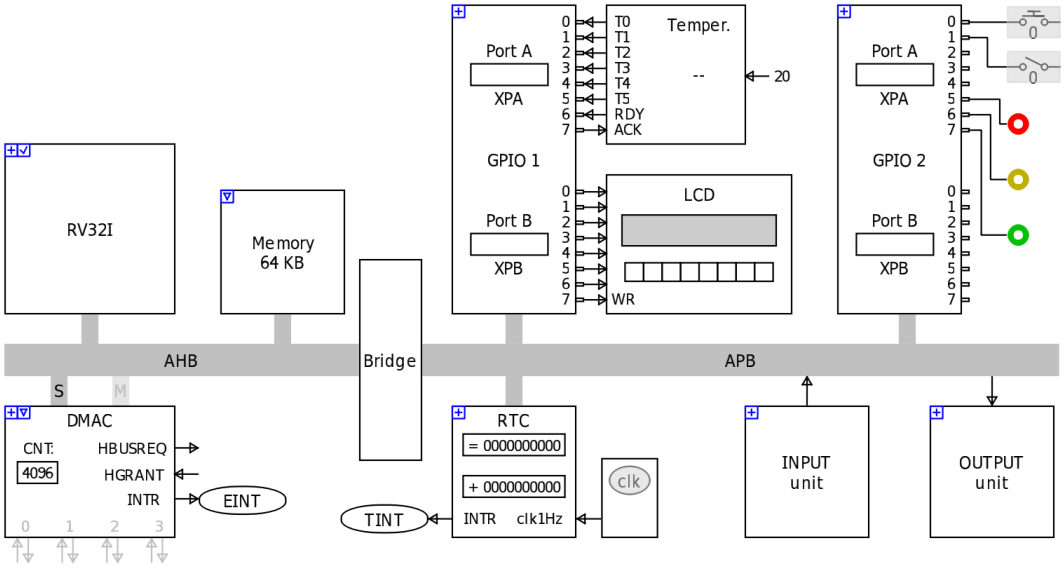
Vanjske jedinice su memorijski mapirane i iz perspektive procesora im se pristupa jednako kao i memoriji, dekoder po potrebi pali dodatne upravljačke signale koji omogućuju pristup registrima vanjskih jedinica. GPIO 1, na kojeg su spojeni temperaturni senzor i LCD prikaznik nalazi se na adresi 0xFFFF0F00. GPIO 2, na kojeg su spojeni

sklopka, prekidač i 3 svjetleće diode nalazi se na adresi 0xFFFF0B00. RTC sklop nalazi se na adresi 0xFFFF0E00.

3.4. Nadograđeni sustav s procesorom RISC-V

Sustav s procesorom RISC-V nadograđen je po uzoru na postojeći sustav s procesorom ARM kako bi se proširile njegove mogućnosti, ali i kako bi se mogle koristiti zajedničke komponente što olakšava održavanje i moguće kasnije nadogradnje. Sustav je podijeljen na dva dijela kao što je vidljivo na slici 3.3.

Unutarnji dio sustava čine procesor, memorija i novododana DMA jedinica spojeni na pojednostavljenu verziju AMBA (engl. *Advanced Microcontroller Bus Architecture*) AHB (engl. *Advanced High-performance Bus*) sabirnice.



Slika 3.3. Nadograđeni sustav s procesorom RISC-V u simulatoru SSPARCSS

Dva GPIO sklopa, RTC i dvije ulazno-izlazne jedinice koje se koriste kod DMA prijenosa (komponente *INPUT* i *OUTPUT*) spojene su na pojednostavljenu APB (engl. *Advanced Peripheral Bus*) sabirnicu. Dva dijela sustava komuniciraju preko AHB-APB mosta, odnosno komponente *Bridge*. Sve vanjske jedinice i dalje su memorijski mapirane, a periferije spojene na GPIO jedinice nisu značajno mijenjane.

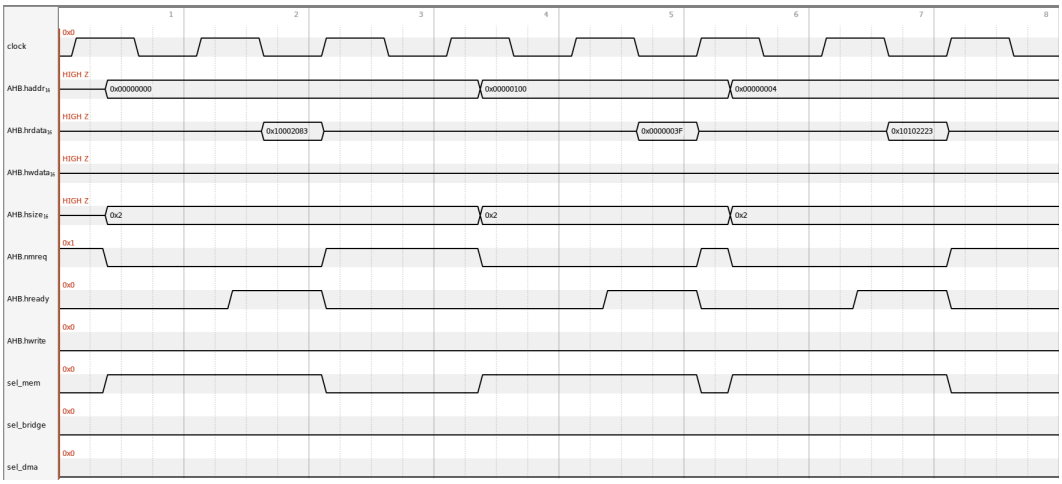
Osim ovih promjena, u sustavu su napravljene i nadogradnje vezane uz prekidni sustav i prikaz elemenata procesora u simulatoru SSPARCSS kao i nadogradnja asemblera

kojom su uvedene neke od RISC-V pseudoinstrukcija [10].

3.4.1. Procesor i memorija

Način pristupa memoriji (a tako i vanjskim jedinicama) je izmijenjen kako bi se omogućila kompatibilnost s novim komponentama i prema protokolu AMBA (APB i AHB) Slika 3.4. prikazuje nekoliko pristupa memoriji.

Signal `sel` zamijenjen je signalom `nmreq` koji ima istu funkcionalnost, ali radi u negativnoj logici, a umjesto signala `wait`, koristi se suprotni signal `ready`. Podatkovna sabirnica zamijenjena je s dvije sabirnice, jednom za čitanje i jednom za pisanje. Ovakva arhitektura sabirnice napravljena je kako bi se eliminirala potreba za čekanjem ako se u istom taktu s neke lokacije čita podatak i piše na neku drugu, no budući da trenutna verzija procesora nema protočnu arhitekturu, ove dvije sabirnice nikad se ne koriste istovremeno, pa su unutar procesora spojene na isti podatkovni registar koji se koristi i pri čitanju i pri pisanju.



Slika 3.4. Vremenski dijagram pristupa memoriji u novom sustavu

Dodane su i dvije nove linije za sinkronizaciju s DMA sklopom, `BUSREQ` i `BUSGRANT` spojene na priključke `HBUSREQ` i `HGRANT` jedinice DMA. Njihova funkcionalnost detaljnije je opisana u poglavlju o DMA.

3.4.2. GPIO

Funkcionalnost sklopova GPIO i pripadnih periferija nije mijenjana u odnosu na postojeći sustav. Svaka GPIO-jedinica ima dvoje 8-bitnih vrata koja mogu biti ulazna ili

izlazna. Podatci se s vrata mogu čitati ili na njih upisivati korištenjem podatkovnih registara DR (engl. *Data Register*) GPIO-jedinice. Svaki od 8 priključaka koji pripadaju jednim vratima može se konfigurirati kao ulazni ili izlazni upisivanjem odgovarajuće vrijednosti u registar smjera za vrata DDR (engl. *Data Direction Register*) [11].

Na GPIO-jedinice spojeni su temperaturni sklop, sklop LCD, prekidač, sklopka i 3 svjetleće diode različitih boja. Diode svijetle kada je na odgovarajući izlazni priključak sklopa GPIO dovedena logička jedinica. Prekidač i sklopka postavljaju odgovarajući ulazni priključak sklopa GPIO u logičku nulu ili jedinicu ovisno o tome jesu li pritisnuti.

Komponenta LCD ima 8 priključaka. Sedam priključaka (DATA) koristi se za slanje ASCII-znaka na ekran, dok osmi priključak (WR) služi kako bi se potvrdilo upisivanje. Na rastući brid tog priključka, znak na ostalih sedam priključaka upisuje se u unutarnji registar LCD-ekrana. LCD-ekran sastoji se od dvije grupe po osam registara. U prikaznim registrima nalaze se znakovi koji su trenutno prikazani na ekranu, dok se u ulazne registre upisuju znakovi pročitani s vanjskih priključaka. Ulazni registri funkcioniraju kao FIFO (engl. *First In First Out*). Upisivanjem novog znaka, sadržaj jednog registra upisuje se u sljedeći, a ako su svi puni, sadržaj posljednjeg registra se gubi. Ako se na LCD pošalje znak LF (engl. *Line Feed*) (kod 0x0A), ne upisuje se u ulazni registar, nego se sadržaj ulaznih registara kopira u prikazne registre. Slanjem znaka CR (engl. *Carriage return*) (kod 0x0D), u sve ulazne registre upisuje se znak razmaka čime ih se efektivno briše [11].

Temperaturni sklop simulira rad temperaturnog senzora spojenog na AD pretvornik. Ima sedam izlaznih priključaka, šest za prijenos podatka (TEMP) koji odgovara temperaturi od 0 do 63 stupnja i jedan za sinkronizaciju (RDY) kojom temperaturni uređaj javlja da se na ostalih šest nalazi valjani podatak. Osmi priključak (ACK) je ulazni. Na rastući brid signala na tom priključku pokreće se mjerenje temperature, a na padajući brid daje mu se do znanja da je podatak pročitan, nakon čega će uređaj spustiti signal RDY [11].

3.4.3. RTC

Sklop RTC služi za brojanje impulsa koji dolaze na njegov vanjski priključak. Ako je na taj priključak spojen pravilni izvor takta poznate frekvencije, RTC može služiti za mjerenje vremena, a inače služi kao obično brojilo [12].

Prema specifikaciji RISC-V [5], postoje brojni načini za izvedbu prekidnog sustava, no najjednostavniji definira tri vrste prekida za svaku razinu privilegija: vanjski prekid (engl. *external interrupt*), prekid od sata (engl. *timer interrupt*) i programski prekid (engl. *software interrupt*). U skladu s time, procesor u simulatoru ima dva priključka, `TINT` i `EINT`. Prekidna linija od RTC-a spojena je na `TINT`, dok je na `EINT` spojena prekidna linija DMA sklopa.

Prekidi se mogu omogućiti ili onemogućiti unutar procesora pisanjem u odgovarajući statusni registar, ovisno o razini privilegije u kojoj procesor radi. Za razinu privilegije M to je registar `mie` (engl. *Machine Interrupt Enable*). Ako je neka prekidna linija aktivna, to je vidljivo u drugom statusnom registru, čak i ako su prekidi onemogućeni. Kod razine M to je registar `mip` (engl. *Machine Interrupt Pending*).

3.4.4. DMA

Jedinica DMA (engl. *Direct Memory Access*) služi za sklopovsko ubrzavanje prijenosa veće količine podataka, najčešće između memorije i ulazno-izlazne jedinice, ali i s jedne UI jedinice na drugu, ili s jedne adrese u memoriji u drugu.

Kada bi procesor obavljao ovakve prijenose, svaki podatak prvo bi morao biti pročitan s izvorišne lokacije, zapisan u registar procesora i zatim upisan na odredišnu lokaciju. Ovakav prijenos je poprilično spor, pogotovo ako je vanjska jedinica spora i procesor ju mora čekati. U tom slučaju prijenos bi se mogao obavljati za vrijeme obrade prekida kojeg bi izazvala vanjska jedinica kada je spremna, no ni to nije idealno rješenje jer je tada potrebno potrošiti dodatno procesorsko vrijeme na druge poslove vezane uz obradu prekida, poput spremanja konteksta, identifikacije vanjske jedinice koja je izazvala prekid i sličnog. DMA-jedinica aktivna je samo kada su uvjeti za prijenos ostvareni, što može dojaviti procesor, ili vanjska jedinica koja je posebnim linijama izravno spojena na DMA-jedinicu. Za vrijeme trajanja DMA prijenosa, procesor je neaktivan jer DMA-jedinica upravlja sabirnicom pa procesor ne može dohvaćati instrukcije iz memorije. Unatoč tome, ovakva organizacija sustava ne samo da rezultira nekoliko puta bržim prijenosom podataka, već i omogućava procesoru obavljanje ostalih korisnih poslova dok nema podataka za prijenos.

Jedinica DMA u sustavu u simulatoru SSPARCSS temeljena je na ARM-ovom sklopu

SMDMAC PL 081[13]. Podržava prijenose iz memorije u memoriju, s vanjske jedinice u memoriju i iz memorije na vanjsku jedinicu. Ima četiri ulaza za spajanje na vanjske jedinice preko kojih se daje zahtjev za DMA prijenosom, dvije linije za sinkronizaciju s procesorom i mogućnost spajanja na prekidnu liniju procesora. Podržava blokovski prijenos podataka širine bajta, poluriječi (2 B) ili riječi (4 B). Registri DMA sklopa prikazani su u tablici 3.1. Adrese registara jedinice DMA definirane su relativno u odnosu na baznu adresu 0x00FF0000 [13].

Tablica 3.1. Registri DMA jedinice [13]

Adresa	Ime	Opis
+0x00	`Control`	1-bitni registar za omogućavanje kanala
+0x04	`Status/Clear`	1-bitni registar stanja spremnosti (ako se čita) 0-bitni registar za brisanje stanja spremnosti (ako se piše)
+0x10	`SrcAddr`	32-bitni registar za adresiranje izvora podataka
+0x14	`DestAddr`	32-bitni registar za adresiranje odredišta podataka
+0x18	`Sizes`	15-bitni registar za zadavanje veličine bloka B i veličine prijenosa T
+0x1C	`Config`	8-bitni registar konfiguracijske riječi kanala

Za rad s DMA-jedinicom, u sustavu su također prisutne jedna ulazna i jedna izlazna jedinica, na adresama 0xFFFF0D00 i 0xFFFF0C00, komponente s oznakama *INPUT* i *OUTPUT* na slici 3.3. Vanjske jedinice imaju FIFO veličine 8 riječi i spojene su vanjskim linijama na DMA-jedinicu kako bi mogle davati zahtjeve za DMA prijenosom. Ulazna jedinica daje zahtjev za prijenosom kada u FIFO-u ima 4 ili više podatka. Izlazna jedinica daje zahtjev za prijenosom kada u FIFO-u ima 4 ili više prazna mjesta. Vanjske jedinice imaju dva registra: registar `Data` iz kojeg se čitaju ili u koji se upisuju podatci i registar `TransferSize` u kojem se zadaje broj podataka za prijenos. Kada su prenese podatak, vrijednost u registru `TransferSize` se smanjuje.

Kada želi obaviti prijenos, DMA-jedinica će procesoru preko linije `HBUSREQ` dati zahtjev za upravljanjem sabirnicom. Procesor nakon izvršavanja svake instrukcije provje-

rava stanje te linije te će, ako je aktivna, predati kontrolu nad sabirnicom DMA-jedinici aktivacijom signala `HGRANT`. Nakon što obavi prijenos, DMA-jedinica spušta signal `HBUSREQ`. Nakon toga, procesor spušta signal `HGRANT` i nastavlja s normalnim radom.

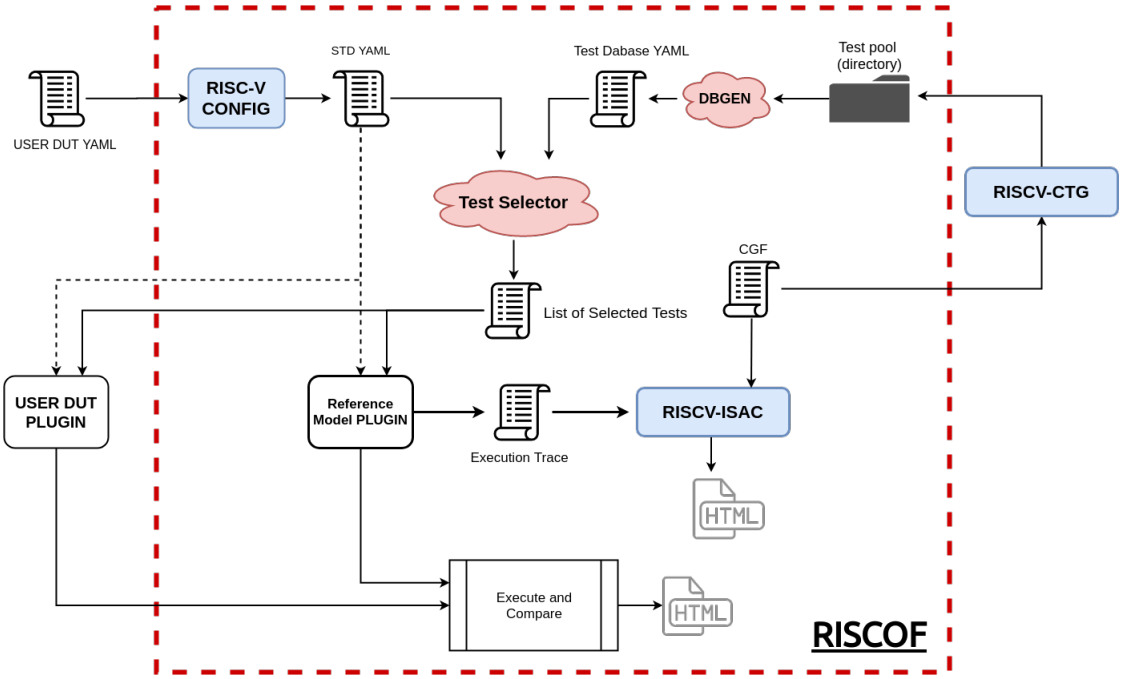
4. Sustav za verifikaciju

Ispitivanje funkcionalnosti ključan je korak u razvoju kako programske podrške, tako i sklopovskih sustava. Postoje brojne različite metode ispitivanja, no uglavnom se svode na dovođenje skupa ulaznih podataka ili okolnosti u sustav koji se testira i uspoređivanje njegovog ponašanja s preddefiniranim očekivanjima. Očekivanja mogu biti definirana specifikacijom, izračunata matematički ili dobivena podvrgavanjem sustava koji se smatra ispravnim istim ulaznim varijablama koje se koriste za testiranje. Takav sustav često se naziva zlatni model ili referentni model. U ovom slučaju, referentni model je sustav s procesorom RISC-V u simulatoru SSPARCSS.

4.1. Postojeća rješenja

Kako je svaki sustav koji se razvija jedinstven, tako je i testiranje svakog sustava drugačije. Ovisno o složenosti sustava, zahtijevanoj pouzdanosti, brzini rada, jednostavnosti korištenja i brojnim drugim faktorima, ispitivanje se može svesti na jednostavan postupak koji se može lako automatizirati, ili na mukotrpan posao koji često traje nekoliko puta dulje od samog razvoja.

Što se tiče procesora arhitekture RISC-V, definirana je službena okolina RISCOF [14] (engl. *RISC-V Compatibility Framework*) koja se koristi za ispitivanje usklađenosti sa specifikacijom (engl. *conformance testing*), kao i skup osnovnih ispitnih primjera koji se koriste [15]. Arhitektura okoline RISCOF prikazana je na slici 4.1.



Slika 4.1. RISCOF [14]

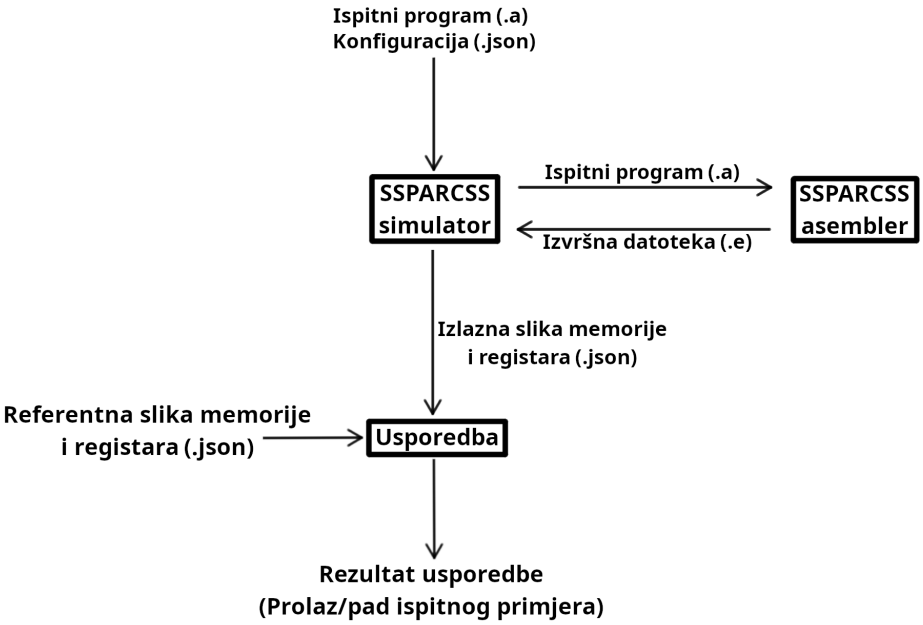
Za testiranje se od korisnika očekuje datoteka u formatu YAML (engl. *Yet Another Markup Language*) koja definira ISA sustava koji se ispituje i dodatak pisan u programskom jeziku Python koji okolina koristi za prevođenje ispitnog programa, njegovo izvršavanje i dobivanje rezultata izvršavanja. Kao referentni modeli za testiranje najčešće se koriste modeli Sail [16] i Spike [17].

4.2. Arhitektura razvijenog sustava za verifikaciju

Središnji dio sustava za ispitivanje koji je razvijen u sklopu ovog rada je simulator SSPARCSS koji je modificiran kako bi ga se moglo pokretati bez grafičkog sučelja [18]. Simulatoru se na ulazu osim programa kojeg treba asemblirati i izvršiti predaje i konfiguracijska datoteka u formatu JSON (engl. *JavaScript Object Notation*) kojom mu se daje do znanja kojim registrima i memorijskim lokacijama želimo znati sadržaj po završetku izvršavanja programa. Simulator automatski pokreće asembler koji prevodi program te nakon toga pokreće simulaciju.

Kad simulacija završi, simulator će stvoriti izlaznu .json datoteku koja sadrži stanje na kraju simulacije svih registara i memorijskih lokacija navedenih u ulaznoj konfiguracijskoj datoteci. Izlazna se datoteka onda može koristiti za usporedbu s datotekom koja sadrži poznate ispravne rezultate (referentna .json datoteka) kako bi se ispitala isprav-

nost modela koji se ispituje. Pokretanje simulatora i usporedba izlaznih i referentnih datoteka obavlja se unutar skripte napisane u programskom jeziku Python. Rad sustava prikazan je na slici 4.2.



Slika 4.2. Prikaz rada verifikacijskog okruženja

4.3. Ispitni primjeri

Gotovo bilo koji program s odgovarajućom konfiguracijom može se koristiti kao ispitni primjer. Jedino je bitno da ima konačno vrijeme izvođenja kako bi simulacija mogla završiti i kako bi simulator mogao generirati izlaznu datoteku. Dio ispitnih primjera napisan je posebno za ovaj sustav, uglavnom kako bi se ispitala funkcionalnost različitih vanjskih jedinica. Osim njih, uvelike se koriste i postojeći programi iz zbirke zadataka koja je još u izradi, a nastala je po uzoru na zbirci zadataka za stari procesor FRISC [19].

5. Zaključak

Arhitektura RISC-V predstavlja zanimljivi novi pristup razvoju procesora. Njezine najveće prednosti su otvorenost standarda i velika fleksibilnost koja se postiže korištenjem različitih proširenja. Iako nije do kraja razvijena, već je i sad zastupljena u mikrokontrolerskim sustavima i jednostavnim računalima, gdje postiže impresivne performanse s obzirom na to da je RISC-V i dalje relativno nova arhitektura. Daljnjim razvojem sigurno će se početi koristiti i u drugim područjima primjene. Upravo zbog svoje otvorenosti i jednostavnosti često se koristi i u obrazovne svrhe.

U sklopu ovog rada jedan takav mikrokontrolerski sustav s procesorom arhitekture RISC-V proširen je novim vanjskim jedinicama te je razvijeno verifikacijsko okruženje kako bi se olakšao njegov daljnji razvoj.

Korišteni procesor trenutno implementira osnovni skup instrukcija RV32I i proširenje Zicsr te omogućuje izvođenje programa u privilegiranom strojnom (engl. machine) načinu rada M . Prisutne su i brojne mogućnosti za nastavak razvoja i unaprjeđenje sustava, primjerice uvođenje protočne strukture u arhitekturu procesora, implementacija dodatnih proširenja, novih vanjskih jedinica ili uvođenje dodatnih razina privilegije, osim trenutno jedine prisutne M razine.

Literatura

[1] "What is instruction set architecture?". [Mrežno]. Adresa: https://www.learncomputerscienceonline.com/instruction-set-architecture/

[2] "History of risc-v". [Mrežno]. Adresa: https://riscv.org/about/history/

[3] "About risc-v". [Mrežno]. Adresa: https://riscv.org/about/

[4] A. Waterman i K. Asanović, *The RISC-V Instruction Set Manual, Volume I: Unprivileged Architecture, Version 20240411*, 2024. [Mrežno]. Adresa: https://github.com/riscv/riscv-isa-manual/releases/download/20240411/priv-isa-asciidoc.pdf

[5] ——, *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 20240411*, 2024. [Mrežno]. Adresa: https://github.com/riscv/riscv-isa-manual/releases/download/20240411/priv-isa-asciidoc.pdf

[6] D. Basch, *Korisnička dokumentacija SSPARCSS asemblera*, 2020.

[7] ——, *Korisnička dokumentacija SSPARCSS simulatora*, 2022.

[8] D. Basch i G. Hacek, *Modelacijski jezik COMDEL2, v 4.14.*, 2024.

[9] A. Bradburry, P. Dabblet, i M. Clark, *RISC-V Assembly Programmer's Manual*, 2017. [Mrežno]. Adresa: https://github.com/riscv-non-isa/riscv-asm-manual/blob/main/riscv-asm.md

[10] V. Đurić, *Verifikacijsko okruženje za oblikovanje računalnog sustava temeljenog na procesoru arhitekture RISC-V*, Završni rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, 2024.

[11] M. Kovač i D. Hofman, “Arhitektura računala 1r - vanjska jedinica gpio”, 2022.

[12] ——, “Arhitektura računala 1r - prekidni ui prijenos”, 2022.

[13] ——, “Arhitektura računala 1r - dma prijenos”, 2022.

[14] *RISC-V Compatibility Framework*. [Mrežno]. Adresa: https://riscof.readthedocs.io/en/stable/index.html

[15] “Risc-v architecture test sig”. [Mrežno]. Adresa: https://github.com/riscv-non-isa/riscv-arch-test

[16] “Riscv sail model”. [Mrežno]. Adresa: https://github.com/riscv/sail-riscv

[17] “Spike risc-v isa simulator”. [Mrežno]. Adresa: https://github.com/riscv-software-src/riscv-isa-sim

[18] S. Šarić, *Unaprjeđenje edukacijskog simulacijskog paketa SSPARCSS za automatizirano izvođenje simulacija bez korištenja grafičkog korisničkog sučelja*, Diplomski rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, 2024.

[19] D. Basch, M. Žagar, B. Mihaljević, M. Orlić, J. Knezović, I. Bosnić, D. Hofman, i M. Kovač, *Zbirka programskih zadataka za procesor FRISC, 2. izdanje*, 2017.

Sažetak

Verifikacijsko okruženje za oblikovanje računalnog sustava temeljenog na procesoru arhitekture RISC-V

Luka Grubišin

U ovom radu ukratko je opisana instrukcijska arhitektura RISC-V, osnovni skup registara, osnovne instrukcije i proširenje Zicsr. Opisan je programski paket SSPARCSS, koji se sastoji od simulatora i asemblerskog prevoditelja, kao i postojeći sustav s procesorom arhitekture RISC-V u SSPARCSS-u. Sustav u simulatoru nadograđen je i spojen na nove vanjske jedinice te je opisana komunikacija između procesora, memorije i vanjskih jedinica. Razvijeno je i opisano verifikacijsko okruženje koje se može koristiti u daljnjem razvoju sustava s procesorom RISC-V, ali i drugih računalnih sustava za simulator SSPARCSS.

Ključne riječi: RISC-V, SSPARCSS, simulacija, ispitivanje, arhitektura računala

Abstract

Development of verification environment for the development of computing system based on RISC-V ISA

Luka Grubišin

This thesis studies the basic RISC-V instruction set architecture, it's registers and instructions, as well as the Zicsr extension. It also describes the SSPARCSS software package, which consists of an assembler and a simulator, and the existing RISC-V system in it. The simulated RISC-V system has been upgraded and connected to new peripheral units. The communication protocols between the processor, memory and peripherals have been explained. A testing environment has been made to be used in the further development of the RISC-V system, as well as other systems in the SSPARCSS simulator.

Keywords: RISC-V, SSPARCSS, simulation, testing, computer architecture

Privitak A: Postavljanje i pokretanje verifikacijskog okruženja

Programski paket SSPARCSS napravljen je u razvojnom okviru Qt. Kao razvojno okruženje koristi se Qt Creator s 32-bitnim prevodiocem MinGW 7.3.0. Qt Creator i prevoditelj instaliraju se zajedno i moguće ih je preuzeti s poveznice `https://www.qt.io/download-open-source`. Koristi se verzija okvira Qt 5.12.2 koja je nešto starija pa ju je potrebno preuzeti s arhive na poveznici `https://download.qt.io/archive/qt`.

Nakon što su ove komponente instalirane, potrebno je klonirati repozitorije koji sadrže izvorne kodove SSPARCSS asemblera, simulatora i simuliranih sustava. SSPARCSS asembler nalazi se na `https://bitbucket.org/basch/conas-configurable-assembler/src/master`, a simulator na `https://bitbucket.org/basch/atlas/src/master`. Model RISC-V sustava, ispitni programi i konfiguracije nalazi se na `https://gitlab.com/jknezovic.hr/ferve-ssparcss`.

SSPARCSS asembler i simulator prevode se u Qt Creatoru. Jedina promjena koju je potrebno napraviti je postaviti putanju do aplikacije asemblera u metodi startAssemblerApplication() u datoteci *atlas/simulator_gui/src/simulator/gui/WithoutGUI.cpp*.

Nakon prevođenja, izvršne datoteke nalazit će se u direktoriju *build*. Kako bi se simulator mogao pokretati iz naredbenog retka, potrebno je u PATH varijablu okruženja dodati Qt biblioteke ili ih je potrebno kopirati u direktorij gdje se nalaze izvršne datoteke simulatora ili asemblera. Ovime je gotovo postavljanje okruženja.

Verifikacijsko okruženje izvedeno je kao skripta u programskom jeziku Python. Skripta kao ulazne argumente prima:

- Putanju do izvršne datoteke simulatora (--sim)
- Putanju do .system datoteke koja određuje model sustava (--system)
- Putanju do direktorija koji sadrži ispitne programe (--programs)
- Ime procesora (--processor, trenutno su u simulatoru podržane samo opcije RISCV i ARM)
- Putanju do direktorija koji sadrži ulazne .json konfiguracije (--configs)
- Putanju do direktorija koji sadrži referentne .json datoteke za usporedbu (--references)

Nakon pokretanja, skripta će prevesti i pokrenuti svaki program za kojeg je definirana ulazna konfiguracija i nakon toga, ako je zadan direktorij s referentnim datotekama, usporediti izlazne vrijednosti svakog pokrenutog programa s referentnim vrijednostima i ispisati rezultate.