Tampereen yliopisto

Onni Hytönen

RISC-V-LIUKUHIHNASIMULAATTORI OPETUSKÄYTÖSSÄ

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Tammikuu 2021

TIIVISTELMÄ

Onni Hytönen: RISC-V-liukuhihnasimulaattori opetuskäytössä
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Tammikuu 2021

Tietokoneen käskykanta-arkkitehtuuri on rajapinta tietokoneen laitteiston ja sovellusohjelmien välissä. Käskykanta-arkkitehtuuri on oleellinen osa opetusta, kun opetetaan tietokoneen sisäistä toimintaa esimerkiksi prosessorin ja muistin välillä. Prosessorin liukuhihnan toiminnan ymmärtäminen on tässä keskeisessä osassa. Tämän vuoksi opetuksen apuna on käytetty liukuhihnasimulaattoria, jolla voidaan simuloida käytettävän käskykannan mukaisten assembly-käskyjen suoritusta prosessorin liukuhihnalla. Tampereen yliopiston kurssilla *Tietokoneen arkkitehtuuri* on aiemmin käytetty opetuksessa esimerkkiarkkitehtuurina MIPS-käskykantaa. Laitteiston ja sovellusten kehittyessä on kuitenkin tullut tarvetta uudistaa myös käskykanta-arkkitehtuuria, ja opetuskäytön olisi hyvä seurata teollisuuden tarpeita. MIPS-käskykanta ja kurssilla käytetty MIPS-simulaattori MipsIT olisi tarkoituksena vaihtaa uudempiin, ajan mukaisiin teknologioihin.

RISC-V on moderni, rajoitetun käskykannan ideologiaan pohjautuva tietokoneen käskykanta-arkkitehtuuri. Sen suosio on ollut kasvussa, ja lähivuosina sen käyttäjämäärä tulee moninkertaistumaan. RISC-V on myös alunperin kehitetty opetuskäyttöön, ja myös tästä syystä se on potentiaalinen MIPS:n korvaaja. Tässä tutkimuksessa selvitetään markkinoilta löytyviä RISC-V-simulaattoreita ja niiden ominaisuuksia. Työn tavoitteena on löytää yksi tai useampi simulaattori, joita voitaisiin hyödyntää opetuskäytössä. Simulaattoria voitaisiin myöhemmin käyttää Tampereen yliopiston kurssin *Tietokoneen arkkitehtuuri* harjoitustyössä.

Tutkimusta varten määritettiin opetusympäristön kautta muutamia kriteerejä mahdollisille simulaattoreille. Simulaattoreita valittiin lopulta 3 kappaletta tutkimukseen useaan valintakriteeriin perustuen. Simulaattoreita testattiin muutamalla vanhan harjoitustyön ohjelmakoodilla. Tämän jälkeen niitä pisteytettiin niiden ominaisuuksiin ja käytettävyyteen perustuvin parametrein. Tutkimuksessa huomattiin, että yhtä ainoaa simulaattoria ei välttämättä kannata valita, vaan voidaan valita useampi simulaattori, joita käytettäisiin kurssin harjoitustyön eri vaiheissa. Myös muutamissa muissa yliopistoissa on toimittu näin, joten lopputuloksena päädyttiinkin kahteen eri simulaattoriin. Näistä toista voitaisiin käyttää esimerkiksi kurssin viikkoharjoituksissa, ja toisella voitaisiin tehdä varsinainen harjoitustyö.

Avainsanat: RISC-V, liukuhihna, simulointi, käskykanta, opetus, tietokoneen arkkitehtuuri

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

ABI	Application binary interface, Sovelluksen binäärirajapinta
ARM	Advanced RISC Machines, mobiililaitteissa ja sulautetuissa järjestelmissä suosittu suoritinarkkitehtuuriperhe
ASIC	Sovelluskohtainen integroitu piiri (engl. Application Specific Integrated Circuit). Tiettyä käyttötarkoitusta varten suunniteltu mikropiiri.
CAGR	Keskimääräinen [esimerkiksi markkinan] vuosittainen kasvu (engl. Compound Annual Growth Rate)
CISC	Kompleksin käskykannan tietokone (engl. Complex Instruction Set Computer)
Eclipse	Eclipse Foundationin kehittämä ohjelmointiympäristö
FPGA	Ohjelmoitava porttimatriisi (engl. Field Programmable Gate Array). Uudelleenohjelmoitava logiikkapiiri, jolle voidaan suunnitella laitteistokuvauskielellä piirejä.
IP-lohko	Intellectual property core. Valmis, uudelleenkäytettävä, usein lisensoitu jonkin ominaisuuden toteuttava lohko, jota voidaan käyttää mikropiirin rakennuspalikkana.
ISA	Käskykanta-arkkitehtuuri (engl. Instruction Set Architecture). Rajapinta tietokoneen laitteiston ja ohjelmiston välissä.
MIPS	Yhdentyyppinen RISC-suoritinarkkitehtuuri (engl. Microprocessor without Interlocked Pipeline Stages)
RISC	Rajoitetun käskykannan tietokone (engl. Reduced Instruction Set Computer)
RISC-V	Avoin RISC-suoritinarkkitehtuuri (luetaan RISC-Five)
SoC	Järjestelmäpiiri (engl. System on a Chip)
x86	Intelin kehittämä CISC-tyyppinen suoritinarkkitehtuuri

1 JOHDANTO

Digitaalitekniikan opetukseen yliopistoissa sisältyy usein yhtenä osana tietokoneen arkkitehtuuri, jota opetettaessa on sopivaa tutustua tarkemmin prosessorin sisäisen liukuhihnan toimintaan. Erilaiset prosessorin liukuhihnaa tai datapolkua simuloivat ohjelmat tukevat aiheen piirissä myös käytännönläheisempää oppimista. Opetuskäyttötarkoituksessa on etua, jos itse simulaattori ja sen simuloima arkkitehtuuri ovat molemmat yksinkertaisia ja helppoja ymmärtää. Myös yhtenäisyyksistä teollisuudessa käytettyihin arkkitehtuureihin on hyötyä oppimisen ja helpon työelämään siirtymisen kannalta. Tampereen yliopistossa on *Tietokoneen arkkitehtuuri* -nimisellä kurssilla käytetty tähän tarkoitukseen aiemmin MIPS-simulaattoria, joka on kuitenkin vuosien saatossa ikääntynyt, eikä se vastaa enää nykypäivän vaatimuksia. Myös simuloitavilla prosessoreilla on tapana vanhentua ajan myötä, ja näin ollen opetustarkoituksessa käytettäviä simulaattoreitakin on syytä päivittää tai vaihtaa aika ajoin.

RISC-V on Berkeleyn yliopistossa kehitetty avoimen standardin käskykanta-arkkitehtuuri, joka on saanut viime vuosina suurta suosiota. Sille on ehditty kehittää myös useita simulaattoreita. Koska RISC-V on modernimpi vastine MIPS:lle, on sen tutkiminen nykyaikana kiinnostavampaa kuin MIPS:n tutkiminen. Jotta opetus säilyisi ajantasaisena, voitaisiin RISC-V-simulaattori nähdä potentiaalisena korvaajana MIPS-simulaattorille. Tässä työssä selvitetään markkinoilta löytyviä RISC-V-liukuhihnasimulaattoreita ja vertaillaan niiden ominaisuuksia. Erityisenä painotuksena pidetään simulaattorien sopivuutta opetuskäyttötarkoituksiin. Simulaattorivalintojen puitteissa työ keskittyy graafisella käyttöliittymällä varustettuihin simulaattoreihin.

Luvussa 2 nostetaan esiin RISC-V-arkkitehtuurin opetuskäytön kannalta kiinnostavimmat pirteet. Näiden pohjalta luvussa 3 esitellään sekä valintakriteerit, joiden pohjalta tutkimukseen valitaan simulaattoreita, että vertailuparametrit, joita myöhemmin sovelletaan valittuihin simulaattoreihin. Valitut simulaattorit on esitelty luvussa 4. Simulaattoreita pisteytetään parametrien avulla, ja tulokset esitellään luvussa 5. Viimeisessä luvussa 6 pohditaan tulevaisuuden näkymiä, ja sitä, miten kurssin harjoitustyö tulisi rakentaa, jotta valitun simulaattorin ominaisuuksia voitaisiin hyödyntää maksimaalisesti opetustarkoituksessa.

2 MOTIVAATIO RISC-V:N KÄYTTÖÖN

2.1 Rajoitetun käskykannan suunnittelufilosofia

Nykyaikaiset tietokoneiden käskykanta-arkkitehtuurit voidaan jakaa kahteen eri suunnittelufilosofiaa noudattavaan tyyppiin: *CISC* ja *RISC*. Jako perustuu käskykannan kokoon ja operaatioiden monimutkaisuuteen. CISC-mallisella käskykannalla pystytään suorittamaan useita monimutkaisia operaatioita jopa vain yhdellä konekäskyllä. Esimerkiksi työpöytäkäytössä jo 1980-luvulta alkaen hallinnut *x86*-ISA edustaa CISC-suunnittelufilosofiaa.

CISC-mallinen monimutkainen käskykanta vaatii kuitenkin paljon alla olevalta laittestolta. Pelkästään käskykannan laajuus kasvattaa prosessorin kokoa. Myös monimutkaisten operaatioiden suoritus ja mahdollinen jälkeenpäin tehty osiin pilkkominen vaatii paljon ylimääräistä tukea laitteistolta. Vaikka esimerkiksi x86:a voidaan käyttää myös mobiililaitteissa, ei se kuitenkaan välttämättä ole sen optimaalinen ympäristö [1, 2]. Esimerkiksi laitteiden pieni koko ja vaatimus alhaisesta virrankulutuksesta saattavat estää CISC-tyyppisten prosessorien käytön mobiililaitteissa [1].

Kun turvaudutaan RISC-suunnittelufilosofian mukaiseen rajoitettuun käskykantaan, laitteistolta tarvitaan tukea pienemmälle määrälle käskyjä. Näin ollen voidaan suunnitella yksinkertaisempi laitteisto, joka mahtuu pieneen tilaan ja kuluttaa vähän virtaa. Myös suunnitteluvirheiden määrä todennäköisesti vähenee, sillä yksinkertaisempi laitteisto on usein myös helpompi suunnitella. Tuloksena on siis potentiaalisesti stabiilimpi laitteisto.

Käskykannan monimutkaisuus ei myöskään ole suora tae käskykannan nopeudesta. RISC-mallisessa käskykannassa jokainen konekäsky on yksinkertainen, ja käskykanta on suunniteltu siten, että jokainen käsky voidaan suorittaa vain yhden prosessorin kellosyklin aikana. Lyhyet käskyt mahdollistavat tehokkaan ja nopean käskyjen liukuhihnoituksen. Vaikka yksi CISC-käsky vaatisikin usean RISC-käskyn, voidaan perättäisiä RISC-käskyjä suorittaa tehokkaammin samassa ajassa.

RISC-tyyppisellä käskykannalla varustettuja prosessoreja (pääosin ARM) on viime vuosina käytetty yhä useammissa eri käyttökohteissa, kuten näytönohjaimissa ja mobiililaitteissa [1]. Sen käyttö vaatii kuitenkin lisenssin [3]. Siksi sen rinnalle onkin vuoden 2010 jälkeen tullut avoimeen standardiin perustuva RISC-V, joka ei vaadi käyttäjältään minkäänlaista lisenssiä [4].

2.2 RISC-V-käskykanta-arkkitehtuuri

2.2.1 Versiot ja laajennettavuus

RISC-V on Berkeleyn yliopistossa vuodesta 2010 alkaen kehitetty käskykanta-arkkitehtuuri, joka nimensä mukaisesti perustuu rajoitettuun käskykantaan. Käskykannan todellista kokoa on kuitenkin vaikea havainnollistaa suoraan, sillä RISC-V-käskykannasta löytyy monta erilaista versiota. Tällä hetkellä RISC-V:sta löytyy 32-bittinen yleisversio (engl. base integer instruction set) RV32I, sekä 64-bittinen yleisversio RV64I. Näiden lisäksi on kehitteillä versiot RV32E ja RV128I, joita ei vielä ole jäädytetty. RV32E on muuten identtinen RV32I:n kanssa, mutta se sisältää vähemmän rekistereitä kuin RV32I (vain 16 kappaletta 32:n sijaan). RV32E mahtuu tästä syystä arviolta 25 % pienempään tilaan kuin RV32I ja onkin siten tarkoitettu käytettäväksi pienimmissä sulautetuissa järjestelmissä. [5] RV128I taas on valmistautumista tulevaisuuteen, jolloin saattaa nousta tarve 128-bittisille järjestelmille [6, s. 4, 41]. Kaikki RISC-V-yleisversiot on lueteltuna taulukossa 2.1 [6].

Taulukko 2.1. *RISC-V-yleisversiot (base instruction sets) [6].*

Nimi	Versio	Tila
RV32I	2.1	Ratifioitu
RV32E	1.9	Avoin
RV64I	2.1	Ratifioitu
RV128I	1.7	Avoin

RISC-V-yleisversiot ovat hyvin rajattuja käskykantoja, ja ne kykenevät lähinnä yksinkertaisiin kokonaislukuoperaatiohin ja muistioperaatioihin. RISC-V on kuitenkin laajennettavissa oleva kanta, ja sen mukana tarjotaan useita eri standardilaajennuksia. Nämä laajennukset tarjoavat esimerkiksi tukea kerto- ja jakolaskuille sekä liukulukulaskennalle. Käytössä olevat laajennukset merkataan usein yleisversion perään kirjainsarjana. Esimerkiksi RV32IMAFD pitää sisällään perusoperaatiot (I), kerto- ja jakolaskulaajennuksen (M), atomisten operaatioiden laajennuksen (A), sekä laajennukset perustarkkuuden (F) ja kaksoistarkkuuden (D) liukuluvuille. Kaikki RV32IMAFD-, sekä RV64IMAFD-käskykantojen käskyt löytyvät liittestä A. Kaikki RISC-V-käskykannat ja laajennukset on lueteltu taulukossa 2.2.

Suositeltuina pidetyt RISC-V-konfiguraatiot sisältävät edellä mainitut laajennokset I, M, A, F ja D, sekä Zicsr- ja Zifencei-laajennokset [5]. Laajennos G toimii lyhenteenä tämän kaltaisille konfiguraatioille: pitkän kirjainsarjan sijaan voidaan sanoa lyhyesti RV32G tai RV64G.

Taulukko 2.2. *RISC-V-standardilaajennukset. [6]*

Subset	Name	Implies
Base ISA		
Integer	I	
Reduced Integer	E	
Standard Unprivileged Extensions		
Integer Multiplication and Division	M	
Atomics	A	
Single-Precision Floating-Point	F	Zicsr
Double-Precision Floating-Point	D	F
General	G	IMADZifencei
Quad-Precision Floating-Point	Q	D
Decimal Floating-Point	L	
16-bit Compressed Instructions	C	
Bit Manipulation	B	
Dynamic Languages	J	
Transactional Memory	T	
Packed-SIMD Extensions	P	
Vector Extensions	V	
User-Level Interrupts	N	
Control and Status Register Access	Zicsr	
Instruction-Fetch Fence	Zifencei	
Misaligned Atomics	Zam	A
Total Store Ordering	Ztso	
Standard Supervisor-Level Extensions		
Supervisor-level extension “def”	Sdef	
Standard Hypervisor-Level Extensions		
Hypervisor-level extension “ghi”	Hghi	
Standard Machine-Level Extensions		
Machine-level extension “jkl”	Zxmjkl	
Non-Standard Extensions		
Non-standard extension “mno”	Xmno	

2.2.2 Rekisterit

Taulukossa 2.3 on listattuna kaikki 32 RISC-V-arkkitehtuurin yleiskäyttöistä rekisteriä, sekä liukulukurekisterit. Taulukon ensimmäisessä sarakkeessa näkyy rekisterin nimi ja toisessa sarakkeessa sovelluksen binäärirajapinnan (ABI) määrittelemä rekisterin alias. Kolmas sarake kertoo rekisterin tarkoituksen. Neljännessä sarakkeessa kerrotaan, kenen vastuulla on tallettaa rekisteri pinoon, kun rekisterin arvoa muokataan esimerkiksi aliohjelmassa tai keskeytyskäsittelijässä. [6, s. 137] Taulukon rekisterien lisäksi on olemassa vielä ohjelmalaskuri `pc` [6, s. 13] sekä Zicsr-laajennuksen mukana tulevia kontrolli- ja statusrekistereitä [6, s. 55].

Taulukko 2.3. *RISC-V-rekisterien (pl. `pc`) assembler-nimet, tehtävät ja arvojen tallennusvastuu aliohjelmakutsuissa. [6, s. 137]*

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

Kaikki RISC-V-yleisversiot sisältävät 32 rekisteriä, pois lukien RV32E, joka sisältää ainoastaan rekisterit `x0`–`x15` [6, s. 34]. Näistä `x0` on kovakoodattu arvoon 0. Loput rekisterit `x1`–`x31` ovat yleiskäyttöisiä rekistereitä, joskin osalla näistä on ennalta määrätty tarkoitus. Näiden lisäksi on vielä olemassa ohjemalaskuri `pc`, joka pitää kirjaa muistiosoitteesta, josta löytyy ohjelman seuraava käsky. Mikäli käytössä on versio, joka tukee liukulukuja, on käytössä myös 32 lisärekisteriä `f0`–`f32`. Rekisterien leveys on aina sama kuin version bittisyys. RV32I:n rekisterit ovat siis 32 bittiä leveitä, ja RV64I:n rekisterit 64 bittiä leveitä.

2.2.3 Käskyt

Jokainen RISC-V-käsky ottaa syötteenään 3 parametria. RISC-V-käskykanta on kehitetty yksinkertaisuus edellä, ja vaihteleva parametrien määrä vaatisi kiinteää määrää monimutkaisemman laitteiston. Luonnollinen parametrien lukumäärä on 3, sillä esimerkiksi aritmeettiset operaatiot tarvitsevat usein 2 operandia ja kohteen tulokselle. [7, s. 63–65]

Käskyjen parametreina on aina joko rekistereitä tai vakioarvoja. Mikäli käskyn nimi päättyy *i*-kirjaimeen, käsitellään vakioarvoja. Muussa tapauksessa arvojen on oltava rekistereissä. Esimerkiksi *Add*-käsky

```
add x20, x21, x22
```

laskee yhteen rekisterien `x21` ja `x22` sisällöt ja tallettaa summan rekisteriin `x20`, kun taas *Add immediate* -käsky

```
addi x20, x21, 22
```

summaa rekisterin `x21` kanssa vakioarvon 22 ja tallettaa tuloksen rekisteriin `x20`.

Käskykanta sisältää vain oleellisimmat käskyt, joita tietokoneen käskykanta-arkkitehtuuri tarvitsee. Osa RISC-V-käskyistä onkin monikäyttöisiä, kuten esimerkiksi jo mainittu *add immediate*. Vakioarvon lisäämisen ohella sillä voidaan toteuttaa esimerkiksi *no operation* -käsky, jolloin prosessori ei tee mitään kyseisen käskysyklin aikana. Apuna käytetään rekisteriä `x0`, joka (kuten luvussa 2.2.2 on kerrottu) sisältää aina lukuarvon 0. Käskyllä

```
addi x0, x0, 0,
```

lisätään rekisteriin `x0` vakioarvo 0, ja saatu summa $(0 + 0 = 0)$ talletetaan takaisin rekisteriin `x0` [6, s. 20]. Samaa ajatusta voidaan käyttää myös vakioarvon asettamisessa rekisteriin. Käsky

```
addi x20, x0, 5
```

asettaa rekisterin `x20` arvoksi 5. Käsky

```
addi x20, x21, 0
```

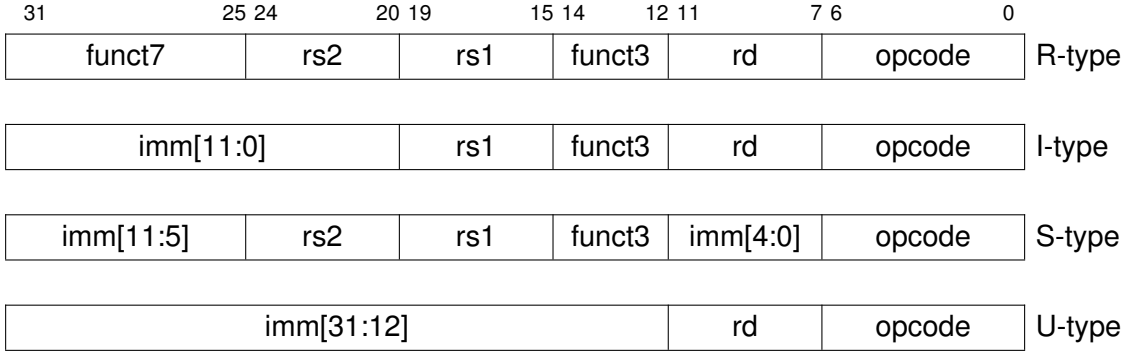
puolestaan kopioi `x21`:n arvon rekisteriin `20` lisäämällä `x21`:n arvoon 0 ja tallettamalla tuloksen rekisteriin `x20`. Ohjelmaan 2.1, on koottu yhteen kaikki toistaiseksi esitellyt käskyt. Kaikki RV32/64IMAFD-standardikäskyt löytyvät liitteestä A [6, s. 90–94].

```
1 add  x20, x21, x22  // x20 = x21 + x22
2 addi x20, x21, 22   // x20 = x21 + 22
3 addi x0,  x0,  0    // no operation
4 addi x20, x0,  5    // x20 = 5
5 addi x20, x21, 0    // x20 = x21
```

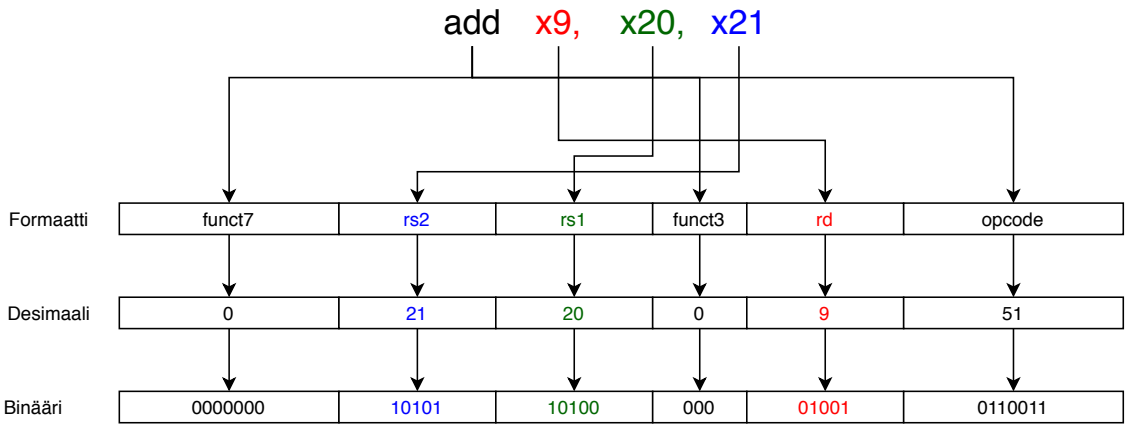
Ohjelma 2.1. *Esimerkkejä RISC-V-käskykannan `add`- ja `addi` käskyistä.*

Konekäskytasolla kaikki RISC-V-konekäskyt ovat versiosta riippumatta 32 bittiä pitkiä. Käskyt on suunniteltu siten, että keskenään samankaltaiset käskyt noudattavat samankaltaista rakennetta. Esimerkiksi `add`- ja `sub`-käskyt noudattavat molemmat *R*-tyypin käskyrakennetta. Peruskäskytyypit *R*, *I*, *S* ja *U* sekä niiden rakenteet on kuvattu taulukossa 2.4. Kentät `opcode`, `funct3` ja `funct7` ovat osa operaatiokoodia, `rd` kohderekisteri, `rs1` ja `rs2` operandirekisterejä, ja `imm` vakioarvoja (engl. `immediate value`) [7, s. 83–84]. Mainittujen käskytyyppien lisäksi on olemassa myös muun tyyppisiä käskyjä (muun muassa *B* ja *J*) [6, s. 16–17].

Taulukko 2.4. *RV32I-spesifikaation mukaisten käskytyyppien konekäskytason formaatit [6, s. 16]*



Myös eri käskytyyppien välillä esiintyy samankaltaisuutta konekäskytason bittikuvioissa. Näin käskyjen dekoodaus on yksinkertaisempaa, mikä johtaa alla olevan laitteiston yksinkertaisuuteen. Kuten Taulukosta 2.4 nähdään, ovat rekistereitä vastaanottavien käskyjen rekistereitä kuvaavat kentät usein samoilla paikoilla [6, s. 15]. 7 vähiten merkitsevää bittiä puolestaan kuvaavat aina operaatiokoodia käskytyypistä riippumatta. Myös `funct3`, joka kuvaa operaatiokoodin lisämäärettä [7, s. 83] löytyy kaikista sitä tarvitsevista käskyistä samalta paikalta. Kuvassa 2.1 on esimerkki `add`-käskyn kuvautumisesta *R*-tyypin bittikuvioon [7, s. 81–82]. Jos käsky olisi `sub`-käsky, ainoa mikä muuttuisi, olisi `funct7`-kenttä arvosta 0 arvoon 0100000_2 [7, s. 85].



Kuva 2.1. Add*-käskyn kuvautuminen konekielitason formaattiin.*

Standardikäskyjen lisäksi käskykannasta löytyy myös pseudokäskyjä, joita ohjelmoija voi käyttää apunaan ohjelmakoodia kirjoittaessaan, mutta jotka kuitenkin assembler kääntää oikeiksi standardikäskyiksi. Pseudokäskyjen avulla on helpompi kirjoittaa ymmärrettävää ohjelmakoodia. Esimerkiksi aiemmin mainittuja käskyjä varten on olemassa pseudokäskyt `nop` (*no operation*), `li` (*load immediate*) ja `mv` (*move*), joilla voidaan toteuttaa kuvatut tilanteet ohjelmoijalle helpommalla tavalla. Myös `add`-käskyä voidaan pseudokäskynä käyttää vakioarvojen summaamiseen. Assembler muuntaa käskyn `addi` -käskyksi. [7, s. 125] Vaihtoehtoiset tavat toteuttaa ohjelman 2.1 käskyt löytyvät taulukosta 2.5.

Taulukko 2.5. *Esimerkkikäskyt RV32I-käskyinä ja pseudokäskyinä*

RV32I-käsky	Pseudokäsky
`add x20, x21, x22`	N/A
`addi x20, x21, 22`	`add x20, x21, 22`
`addi x0, x0, 0`	`nop`
`addi x20, x0, 5`	`li x20, 5`
`addi x20, x21, 0`	`mv x20, x21`

Kuten taulukosta 2.5 nähdään, eivät pseudokäskyt aina ota kolmea parametria, mutta niitä vastaavat standardikäskyt kuitenkin toimivat aina tällä tavalla. Lista kaikista RISC-V-pseudokäskyistä löytyy liitteestä B [6, s. 139–140].

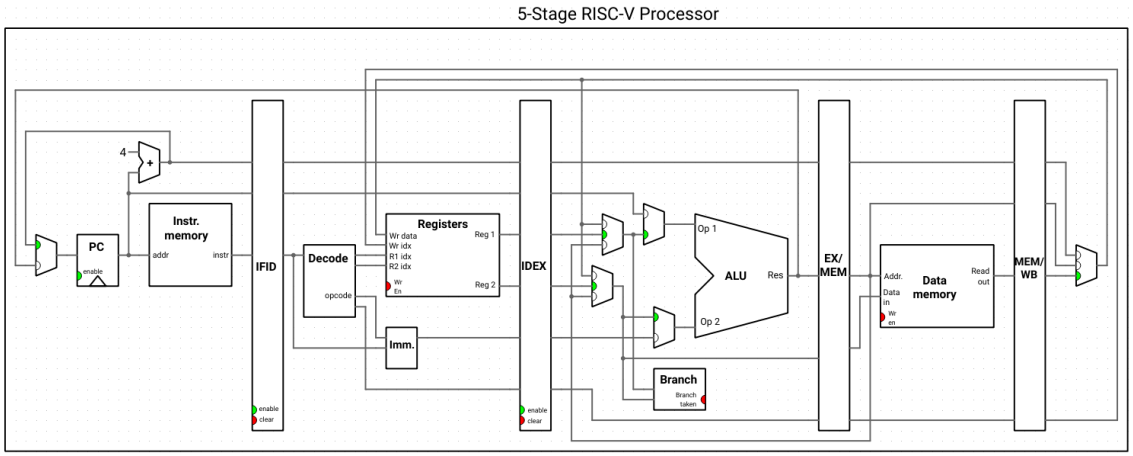
2.2.4 Käskyjen liukuhihnoitus

Vaikka niin kutsuttu *Single cycle* -toteutus on toimiva ratkaisu käskyjen suorittamisessa, on se kuitenkin hidas ratkaisu, eikä sitä nykyään enää usein käytetä oikeissa toteutuksissa [7, s. 261–262]. Käskyjä liukuhihnoittamalla voidaan saavuttaa suuria nopeushyötyjä. Liukuhihnoituksella tarkoitetaan prosessorin sisällä tapahtuvaa käskyjen eri vaiheiden rinnakkaista suoritusta. Käskyt valmistuvat sekventiaalisesti, mutta ne voidaan jakaa pienempiin osiin. Esimerkiksi ensimmäisen käskyn siirtyessä dekoodausvaiheeseen, voidaan seuraavaa käskyä alkaa jo hakemaan samanaikaisesti. Kuvassa 2.2 on esitetty Ripes-simulaattorin, josta kerrotaan lisää luvussa 4.2, 5-portainen RISC-V-liukuhihna.

Esimerkkikuvan 2.2 mukaisella 5-portaisella liukuhihnalla jokainen käsky jaetaan viiteen vaiheeseen, jotka ovat:

1. Käskyn haku (engl. Instruction Fetch, lyhenne IF)
2. Käskyn dekoodaus (engl. Instruction Decode, lyhenne ID)
3. Käskyn suoritus (engl. Execute, lyhenne EX)
4. Muistioperandin haku (engl. Memory access, lyhenne MEM)
5. Takaisinkirjoitus rekisteriin (engl. Write Back, lyhenne WB). [7, s. 276]

5-portaisella liukuhihnalla voidaan liukuhihnan täynnä ollessa suorittaa siis viiden eri käskyn eri vaiheita rinnakkain. Portaiden lukumäärä voi kuitenkin vaihdella toteutuksesta riip-



Kuva 2.2. *RISC-V-prosessorin 5-tasoinen liukuhihna Ripes-simulaattorissa.*

puen. Mikäli käskyt aiheuttavat data- tai kontrollihasardeja, voi laitteisto välttyä näiltä eri keinoin. Laitteisto voi esimerkiksi tukea käskyjen forwardointia. Tarvittaessa liukuhihna voidaan myös tyhjentää, mutta tämä aiheuttaa ylimääräistä latenssia. [7, s. 262–272]

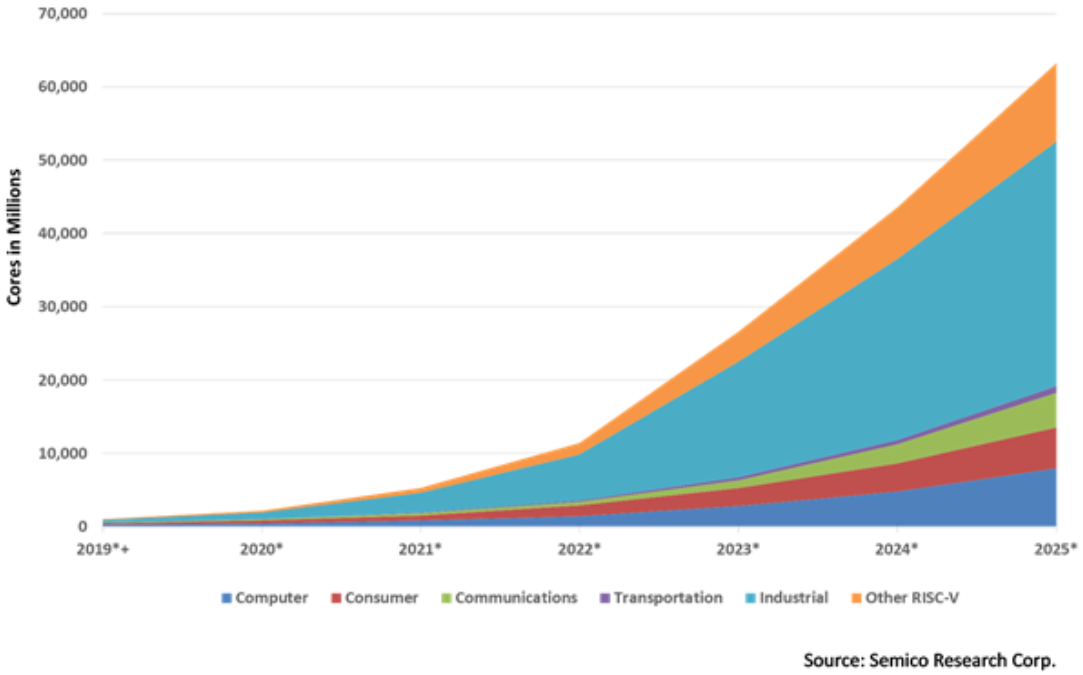
2.3 RISC-V:n suosio

2.3.1 Lähivuosien ennuste

RISC-V-markkinoiden kasvua on viime vuosina tutkittu useaan otteeseen. Markkinatutkimuslaitos Semico Research ennustaa vuoden 2019 raportissaan RISC-V-markkinoille 146,2 %:n keskimääräistä vuosittaista kasvua vuosien 2018 ja 2025 välille [8, katso 9]. Keskimääräinen vuosittainen kasvu eli CAGR (engl. compound annual growth rate) kuvaa jonkun yksikön (esimerkiksi markkina tai yritys) menojen, liikevaihdon, tai muun kiinnostavan suureen trendiä. Kasvussa hyödynnetään prosentuaalista muutosta, joka kumuloituu vuosittain. CAGR:ää voidaan käyttää myös kuvaamaan tuottoastetta, eli tietyllä aikavälillä tuotetun voiton suhdetta sijoitettuun pääomaan. CAGR voidaan laskea kaavalla 2.1:

$$CAGR = \sqrt[n]{\frac{B}{A}} - 1, \tag{2.1}$$

jossa n on tarkasteltavan aikavälin pituus vuosissa, B tarkasteltavan suureen arvo tarkasteltavan aikavälin loppupisteessä, ja A suureen arvo välin alkupisteessä. [10, s. 157–158] CAGR:ää voidaan soveltaa RISC-V-prosessoriydinten lukumäärään markkinoilla olevissa laitteissa. Tällöin 146,2 %:n CAGR tarkoittaisi sitä, että RISC-V-ydinten lukumäärä noin 1,5-kertaistuu joka vuosi. Ennusteen mukaan vuonna 2025 RISC-V-ytimiä olisi markkinoilla yhteensä arviolta 62,4 miljardia kappaletta. [8, katso 9] Ydinten lukumäärän kasvu on esitetty markkina-alueittain eriteltynä kuvaajassa 2.3. Eritellyt markkina-alueet ovat Tietokone-, kuluttaja-, tietoliikenne-, liikenne- ja teollisuusmarkkinat sekä muut RISC-V-markkinat. Ylivoimaisesti suurin markkina-alue on teollisuus.



Kuva 2.3. *RISC-V-prosessoriydinten lukumäärän ennuste lähivuosille [8, katso 9]*

Vertailumielessä olisi kiinnostavaa tietää myös RISC-V-ydinten lukumäärä vuonna 2018. Tutkimuksessa käytetyssä lähteessä ei kuitenkaan kerrota Semicon arviota määrästä, joten se on jouduttu laskemaan erikseen. Hyödyntämällä annettua CAGR:n arvoa 146,2 % ja Semicon arvioimaa loppusummaa $B = 1{,}1\,\mathrm{mrd}$, voidaan alkusumma selvittää ratkaisemalla yhtälö 2.1 tuntemattoman A:n suhteen (kaava 2.2):

$$A = \frac{B}{(CAGR + 1)^n}. \tag{2.2}$$

Sijoittamalla tunnetut B:n ja CAGR:n arvot voidaan yhtälöstä ratkaista RISC-V-ydinten määrä vuonna 2018. Tulokseksi saadaan $117\,089\,698{,}55\cdots \approx 117$ miljoonaa ydintä. Kasvun suuruutta voidaan kuvata myös muutosprosentilla eli ROC:lla (engl. rate of change), joka voidaan laskea kaavalla 2.3:

$$ROC = \frac{B - A}{A} \times 100, \tag{2.3}$$

jossa B on tarkasteltavan suureen arvo tarkasteltavan aikavälin loppupisteessä, ja A suureen arvo välin alkupisteessä [11]. Käyttämällä aiemmin laskettua vuoden 2018 prosessoriydinten lukumäärää, ja Semicon ilmoittamaa vuoden 2025 lukumäärää saadaan kaavan 2.3 avulla tuloksena suhdeluku näiden vuosien arvojen välille. Tuloseksi saadaan $54\,729{,}759\ldots\,\% \approx 54\,700\,\%$, mikä tarkoittaa sitä, että markkinoilla olevien RISC-V-prosessoriydinten lukumäärä yli 500-kertaistuu vuoteen 2025 mennessä.

Semico on myös tutkinut jokaista markkina-aluetta erikseen neljässä eri tuotekatego-

riassa: Advanced performance multicore SoC, value SoC, basic SoC ja FPGA. Eritellyt kasvuennusteet on lueteltu taulukossa 2.6. Kasvu on merkittävää jokaisella markkina-alueella ja jokaisessa tuotekategoriassa. Vaikka kuvaajan 2.3 mukaan absoluuttisesti suurin markkina-alue olisikin teollisuus, on suurinta kasvua ennustettu kuitenkin tietoliikennemarkkinoille. Yksittäisistä tuotekategorioista Basic SoC:n on ennustettu kasvavan eniten kaikilla markkina-alueilla. [8, katso 9]

Taulukko 2.6. *RISC-V:n käytön kasvun ennuste eri tuotekategorioissa [8, katso 9]*

	Computer	Consumer	Communications	Transportation	Industrial	Other RISC-V	Total
Adv. Perf Multicore SoC	60%	73%	224%	153%	106%	171%	144%
Value Multicore SoC	60%	92%	222%	159%	122%	201%	177%
Basic SoC	63%	115%	217%	166%	127%	215%	190%
FPGA	62%	72%	190%	163%	102%	176%	149%
Total	61%	81%	209%	160%	110%	185%	158%

Myös Tractica, joka on vuodesta 2020 alkaen ollut osa Omdiaa [12], on tutkinut RISC-V-markkinoita. Tractica julkaisi raportissaan *RISC-V-Processors* vuonna 2019 ennusteen RISC-V-pohjaisten IP-lohkojen ja näiden ohjelmistojen ja työkalujen liikevaihdon kasvusta. Raportin mukaan liikevaihto kasvaisi vuonna 2018 lasketusta 52 miljoonasta dollarista 1,1 miljardiin dollariin vuoteen 2025 mennessä. [13, katso 14]

Kaavaa 2.3 hyödyntämällä voidaan laskea muutosprosentti vuoden 2018 tilanteeseen verrattuna. Tulokseksi saatu $2015,4\% \approx 2000\%$ tarkoittaa sitä että vuonna 2025 liikevaihto olisi noin 20-kertainen vuoteen 2018 verrattuna. Kaavan 2.1 avulla saadaan keskimääräiseksi vuosittaiseksi kasvuksi vuosien 2018–2025 väliselle 7 vuodelle noin 55 %. Yhteenveto Semicon ja Tractican markkinatutkimuksista ja niistä analysoiduista tuloksista on esitetty taulukossa 2.7.

Tutkimustulokset tukevat väitettä, että RISC-V:n suosio on nopeassa kasvussa, ja piirien kysyntä tulee tulevaisuudessa kasvamaan. Koska ennusteen mukaan markkinoilla olevien piirien määrä tulee lähivuosina moninkertaistumaan, on uusille alan osaajille varmasti tarvetta. Koulutuksen pitää kyetä vastaamaan teollisuuden tarpeisiin.

2.3.2 Suosion syitä

RISC-V erottautuu muista käytössä olevista RISC-tyyppisistä prosessori- ja käskykanta-arkkitehtuureista sen avoimuudellaan. Esimerkiksi ARM:n käyttö vaatii kalliita lisenssejä. Sen sijaan RISC-V:n käyttö ei vaadi minkäänlaisia lisenssejä, vaan sen käyttö on ilmaista. Avoimen standardin myötä käyttäjän on myös mahdollista räätälöidä arkkitehtuuri omiin tarpeisiinsa sopivaksi. Olemassa olevien RISC-V-laajennuksien lisäksi käyttäjä voi

Taulukko 2.7. *Yhteenveto Semicon ja Tractican markkinatutkimuksista analysoiduista tuloksista*

(a) *Semicon tutkimus (tarkasteltava aikaväli 2018-2025)*

Tunnus kaavassa 2.1 / 2.3	Selite	Arvo
n	Vuosien lukumäärä	7
A	RISC-V-prosessoriytimiä vuonna 2018	n. $117 \cdot 10^6$ kpl
B	RISC-V-prosessoriytimiä vuonna 2025	n. $64{,}2 \cdot 10^9$ kpl
ROC	Muutosprosentti	n. 54 700 %
CAGR	Keskimääräinen vuosittainen kasvu	146,2 %

(b) *Tractican tutkimus (tarkasteltava aikaväli 2018-2025)*

Tunnus kaavassa 2.1 / 2.3	Selite	Arvo
n	Vuosien lukumäärä	7
A	Liikevaihto vuonna 2018	n. $52 \cdot 10^6$ $
B	Liikevaihto vuonna 2025	n. $1{,}1 \cdot 10^9$ $
ROC	Muutosprosentti	n. 2000 %
CAGR	Keskimääräinen vuosittainen kasvu	n. 55 %

tehdä omia muokkauksiaan käskykantaan ja laajentaa sitä myös omilla laajennuksillaan. Tämä on yksi syy siihen, miksi RISC-V:lle on olemassa niin paljon erilaisia käyttömahdollisuuksia aina kevyimmistä sulautetuista järjestelmistä jopa Linux-työpisteisiin ja palvelimiin asti. Toisaalta mitä enemmän potentiaalisia käyttökohteita on olemassa, sitä enemmän myös potentiaalisia käyttäjiä tulee olemaan, ja sitä enemmän arkkitehtuuria otetaan käyttöön.

Avoin standardi on myös tuonut markkinoille uudenlaisia tapoja yrityksille valmistaa omia piirejään. RISC-V:n kehittäjien vuonna 2015 perustama SiFive toi vuonna 2016 ensimmäisenä yrityksenä maailmassa markkinoille RISC-V-käskykanta-arkkitehtuurin toteuttavan mikropiirin [15]. Sama yritys ylläpitää nykyään DesignShare-alustaa, jonka avulla järjestelmäpiirejä suunnittelevien yritysten on helppo luoda prototyyppejä. SiFive tarjoaa alustallaan useita RISC-V-piirejä erilaisiin käyttötarkoituksiin, ja yritykset voivat valikoida haluamansa piirit ilman suuria etukäteen tehtyjä lisenssimaksuja. [16]

Myös viime vuosina suosioon nousseita FPGA-piirejä voidaan käyttää RISC-V-prototyypitykseen, sillä RISC-V voidaan toteuttaa FPGA:lla helposti ja alhaisin kustannuksin. Kaikki tähän vaadittavat työkalut sekä prosessoriarkkitehtuurin lähdekoodit ja IP-lohkot löytyvät internetistä ilmaiseksi. Ledin esittelee kirjassaan tavan toteuttaa RISC-V-prosessori Xilinx Artix-7 -FPGA:lla. [5] Esimerkkitoteutus Arty A7-35T -kehityslaudalla maksaa vain 100-200 yhdysvaltain dollaria [17].

RISC-V:n suosiota on nostanut sen monipuolisuus eri käyttöympäristöissä. Myös sen

helppokäyttöisyys, modulaarisuus ja helppo laajennettavuus antavat yrityksille mahdollisuuksia räätälöidä sitä omiin tarpeisiinsa. SiFiven kaltaisten tekijöiden mukanaan tuomat alhaiset prototyypityskustannukset puolestaan luovat yrityksille uudenlaisia mahdollisuuksia tehdä uusia innovaatioita aiempaa pienemmillä riskeillä. Pienille yrityksille alhaiset kustannukset voivat olla jopa kriittinen elinehto. Koska ASIC-piirien valmistus on kallista ja vie kauan aikaa, on prototyyppivaihe erityisen tärkeässä asemassa. Mahdollisuus hyödyntää valmiita IP-lohkoja ilman ylimääräisiä kustannuksia alentaa kynnystä kokeilla uusia teknologioita. Myös helppo ja halpa mahdollisuus käyttää FPGA:ta on iso etu RISC-V:lle.

2.4 Liukuhihnasimulaattori opetuskäytössä

2.4.1 Aiemmat arkkitehtuurit ja simulaattorit

Valitulla käskykanta-arkkitehtuurilla on merkitystä opetuskäytössä. Yksinkertainen käskykanta mahdollistaa yksinkertaisen laitteiston ja liukuhihnan. Opetuskäytössä yksinkertaisuus korostuu, jotta arkkitehtuurin opiskelu säilyy suoraviivaisena, eikä se vaadi liikaa perusymmärryksen kannalta ylimääräisten ominaisuuksien opettelua. Pattersonin ja Hennessyn kirjan aiempia painoksia [18] on käytetty vielä vuoteen 2020 asti myös Tampereen yliopistossa kurssilla *Tietokoneen arkkitehtuuri*. Kirjan esimerkkiarkkitehtuuri on opetuskäytössä pitkään suosittu RISC-mallinen MIPS, joka on 1980-luvulla kehitetty yksinkertainen arkkitehtuuri.

MIPS on hyvin läheistä sukua RISC-V:n kanssa ja on samaan tapaan kehitetty alunperin yliopistomaailmassa [7, s. 62]. Vuodesta 2018 lähtien sen on kuitenkin omistanut Wave Computing [19]. Arkkitehtuureilla on paljon yhteisiä käskyjä. Myös niiden rekisterit ovat hyvin samankaltaisia ja niitä on molemmilla 32 kappaletta. Edellisen seurauksena myös konekielitason käskyformaatit ovat arkkitehtuureilla hyvin samankaltaiset. [7, s. 145–146]

Eroja syntyy muun muassa haarautumiskäskyissä. RISC-V kykenee vertaamaan kahden rekisterin arvoa suoraan ja haarautumaan vertailun tuloksena, kun taas MIPS joutuu asettamaan erillisen rekisterin arvoksi vertailun tuloksen (tosi/epätosi tai 1/0) ja tämän jälkeen haarautumaan rekisterissä olevan arvon perusteella. MIPS myös kykenee ainoastaan *pienempi kuin* -vertailuoperaatoihin, ja täten ohjelmoija joutuu itse määrittämään operandien järjestyksen. Näistä yksinkertaistuksista huolimatta MIPS on kuitenkin huomattavasti laajempi käskykanta kuin RISC-V. [7, s. 145–146]

MIPS:n simuloiminen on helppoa, ja simulaattoreita on paljon. Tampereen yliopistossa on käytetty Lundin yliopistossa kehitettyä MipsIT-simulaattoria [20]. Vanhat harjoitustyöt, joissa MipsIT:iä on käytetty, löytyvät liitteistä C ja D.

2.4.2 RISC-V:n mahdollisuudet opetuskäytössä

Vaikka MIPS on todettu toimivaksi käskykanta-arkkitehtuuriksi jo vuosia sitten, on se kuitenkin menettänyt suosiotaan kaupallisissa järjestelmissä. Tämän seurauksena Patterson ja Hennessy vaihtoivatkin kirjansa viidennessä painoksessa MIPS:n RISC-V:een [7, s. xvi–xvii]. RISC-V:n suosio kasvaa koko ajan, ja sitä käytetään kaupallisesti enenevissä määrin. RISC-V:n käyttäminen opetustarkoituksessa on siis paljon mielekkäämpää, kuin nyt jo vanhentuneen MIPS:n. RISC-V-käskykanta on myös hyvin lähellä MIPS:iä, mutta se on hieman pienempi, ja sitä on helpompi ohjelmoida [7, s. 145–146]. Yksinkertainen ja helppokäyttöinen käskykanta on eduksi varsinkin opiskelijoille, jotka saavat kurssilla ollessaan ensi kosketuksensa assembly-ohjelmointiin tai vähintään RISC:iin.

Kurssihenkilökunnan näkökulmasta yhtäläisyydet aiemmin opetetun käskykannan kanssa helpottavat myös materiaalien ja harjoitustöiden uusimista, kun mahdollisten muutosten tekeminen helpottuu ja ylipäätään muutosten tarve vähenee. Myös RISC-V:n modulaarisuus tuo mukanaan uusia mahdollisuuksia. Sitä voidaan hyödyntää esimerkiksi haastavuudeltaan eri tyyppisiin harjoitustöihin: heĺpompiin tehtäviin riittää rajatumpi käskykanta, kun taas haastavampiin tehtäviin voidaan hyödyntää standardilaajennuksia. Myös kattava valikoima erilaisia simulaattoreita mahdollistaa eri tyyppisten harjoitustöiden luomisen, eikä kaikkiin harjoitustöihin tarvitse käyttää edes samaa simulaattoria [21]. Lisäksi yhtenä suurimmista RISC-V:n eduista voidaan vielä mainita käskykanta-arkkitehtuurin avoimuus. Avoimen standardin myötä tarve mahdollisesti kalliille lisensseille poistuu, ja yliopistojen on mahdollista ottaa teknologia käyttöön matalalla kynnyksellä.

2.4.3 RISC-V:n aiempi käyttö opetuksessa

RISC-V on kehitetty Berkeleyn yliopistossa Yhdysvalloissa ja se on alusta asti suunniteltu opetuskäyttötarkoitusta varten. Berkeleyssä on sen opetuksessa käytetty simulointitarkoituksiin ainakin Venus- ja Spike-simulaattoreita, joista kerrotaan lisää tässä järjestyksessä luvuissa 4.3 ja 4.4 [22, 23].

Tanskan teknillisessä yliopistossa on käytetty RISC-V:tä kurssilla *Computer Architecture and Engineering* [24]. Kurssin oppimateriaalina on käytetty Pattersonin kirjaa [7]. Kurssilla on käytetty kahta eri simulaattoria, jotka molemmat valittiin myös tähän tutkimukseen. Kurssin alkupuolella on käytetty Venus-simulaattoria, ja myöhemmin Spike-simulaattoria. [21]

Myös Tampereen yliopistossa on herännyt kiinnostusta RISC-V:n adoptoinnista *Tietokoneen arkkitehtuuri* -kurssin käyttöön. Arkkitehtuurin vaihto MIPS:stä RISC-V:hen luonnollisesti tarkoittaa myös opetuksessa käytettävän simulaattorin vaihtamista MipsIT:stä uuteen. Luvussa 4 selvitetään eri vaihtoehdot, ja tutkimuksen tarkoituksena olisi löytää kurssille sopiva simulaattori.

3 TUTKIMUSMENETELMÄT JA VERTAILUPARAMETRIT

3.1 Simulaattorien valintakriteerit

Koska markinoilta löytyy suuri määrä erilaisia RISC-V-simulaattoreita, ei tässä tutkimuksessa voida käsitellä niitä kaikkia. Tutkimukseen on valittu 3 kappaletta simulaattoreita erilaisiin kriteereihin perustuen.

Tutkimuksessa on suosittu **graafisen käyttöliittymän** sisältäviä simulaattoreita. Graafisen käyttöliittymän avulla opiskelijat voivat käyttää simulaattoria pekästään hiirellä klikkaillen, sen sijaan että heidän tarvitsi opetella pitkä litania erilaisia komentorivikomentoja. Myös prosessorin toiminnan visualisointi helpottuu graafisen käyttöliittymän myötä huomattavasti verrattuna komentorivikäyttöliittymään.

Simulaattorilla pitää myös kyetä **suorittamaan ja simuloimaan RISC-V-käskykannan mukaista assembly-koodia**. Tämän vaatimuksen täyttävät simulaattorit voidaan pääosin jakaa kahteen kategoriaan: simulaattoreihin, jotka suorittavat ohjelmakoodia ja myös näyttävät tapahtumat graafisessa liukuhihnanäkymässä, sekä puhtaisiin ohjelmakoodisimulaattoreihin. Tutkimukseen valittiin simulaattoreita molemmista kategorioista. Ohjelmakoodin kielenä on usein RISC-V-käskykannan mukainen assembly, tai vaihtoehtoisesti C. Tutkimuksen kannalta kiinnostavampaa on kuitenkin assemblyn simuloiminen.

Opiskelijoilla tulee olla helppo ja **ilmainen** mahdollisuus käyttää simulaattoria omilla koneillaan. Näin ollen simulaattorien tulee olla lisensoitu siten, että niitä saa käyttää vapaasti vähintään opetustarkoituksiin. Avoin lähdekoodi katsotaan myös eduksi. Kaikki valitut simulaattorit toteuttavat edellä mainitun kriteerin, ja useimpien lähdekoodit löytyvät jostain avoimesta lokaatiosta, kuten GitHubista. Simulaattorien tulee myös toimia moderneilla käyttöjärjestelmillä. Vähimmäisvaatimukseksi on asetettu Windows 10 -yhteensopivuus, mutta tuki muillekin alustoille (kuten Linuxille) katsotaan eduksi.

3.2 Vertailuparametrit

Kurssin aiemmat harjoitustyöt (liitteet C ja C) määrittävät simulaattorien vaatimukset. Vertailu tapahtuu testaamalla jokaista simulaattoria luvussa 3 esitellyillä testiparametreilla. Testaajana toimii tutkimuksen tekijä, joka on aiemmin kurssilla ollessaan suorittanut edellä mainitut harjoitustyöt. Tutkittavat parametrit sekä niiden pisteytys ja painotus on lueteltu

taulukossa 3.1. Parametreista 3 ensimmäistä on suoraan simulaattorin teknisiin ominaisuuksiin liittyviä, ja loput enemmänkin sen käyttöön liittyviä parametreja.

Taulukko 3.1. *Tutkimuksessa käytetyt simulaattorien vertailuparametrit.*

Parametri	Pisteskaala (min-max)	Painotus/tärkeys (1-3, 1=pieni hyöty, 3=tärkeä)
Tekniset		
Kyky assemblyn simulointiin	1-3	3
Tuki välimuistin simuloinnille	1-3	1
Graafinen liukuhihnanäkymä	1-3	2
Käyttöön liittyvät		
Helppokäyttöisyys	1-3	3
Visuaalisen ilmeen yhteneväisyys	1-3	1
Saatavuus	1-3	3
Avoimuus	1-3	3

Simulaattorin kyky assemblyn simulointiin on olennainen parametri. Esimerkiksi tuettujen käskyjen määrässä saattaa olla eroja eri simulaattorien välillä. Myös muut simuloinnin tarkkuuteen liittyvät ominaisuudet on huomioitava. Osa tarjolla olevista simulaattoreista kykenee ainoastaan *Single cycle* -prosessointiin, kun taas osa kykenee käskyjen ajamiseen ja tarkastelemiseen 5-portaisen liukuhihnan eri vaiheissa. Myös liukuhihnan ominaisuuksille, kuten hasardien tunnistuksen tai forwardoinnin tuelle, annetaan huomiota.

Simulaattorin eduksi katsotaan myös **tuki välimuistin simuloinnille**. Tampereen yliopiston kurssilla Tietokoneen arkkitehtuuri on viime vuosina ollut kaksi harjoitustyötä, joista ensimmäinen on liittynyt liukuhihnaan, ja toinen välimuistiin. Vaikka tässä työssä keskitytäänkin enimmäkseen liukuhihnan simulointiin, on välimuistin simuloinnin tuesta kuitenkin jatkon kannalta hyötyä.

Tutkimukseen on valittu sekä puhtaita assembly-simulaattoreita, että liukuhihnan toimintaa graafisesti esittäviä simulaattoreita. **Graafinen liukuhihnanäkymä** katsotaan ehdottomasti eduksi, mutta tämä ei kuitenkaan välttämättä ole pakollista, ja mahdollinen kurssiharjoitustyö voidaan tarvittaessa suunnitella myös siten, että tällaiselle ominaisuudelle ei tule tarvetta. Harjoitustyön suunnittelusta kerrotaan lisää kappaleessa 6.

Simulaattorin käyttöön liittyvistä parametreista ensimmäinen on **helppokäyttöisyys**. Mitä vähemmän simulaattorin käyttölittyymä vaatii opettelua, sitä enemmän opiskelijoille jää aikaa itse simulointien ajamiseen ja harjoitustyön tekemiseen. Myös mahdollisuus muokata käyttöliittymän ulkoasua mieleisekseen tekee käyttämisestä mukavampaa ja siten helpompaa. Osaksi helppokäyttöisyyttä voidaan katsoa myös saavutettavuus, eli onko esimerkiksi värisokeita huomioitu käyttöliittymän suunnittelussa.

Erityisesti opetuskäytössä korostuva parametri on simulaattorin **visuaalisen ilmeen yhteneväisyys** kurssimateriaalin kanssa. Referenssimateriaalina käytetään Pattersonin ja Hennessyn kirjan RISC-V-painosta [7]. Esimerkiksi liukuhihnan komponenttien samankaltaisuus kirjan kuvien kanssa helpottaa opiskelijoita yhdistämään visuaalisesti simulaattorissa näkyvät komponentit oppimateriaalin komponentteihin. Rekisteri- tai muistikartan samankaltaisuus kirjan taulukoiden kanssa katsotaan eduksi. Lisäksi käskyjen syntaksin on suotavaa olla yhtäläinen kirjan esimerkkien kanssa. Mikäli käskyjä on mahdollista tarkastella konekielitasolla, olisi kätevää pystyä selkeästi erottamaan käskyn eri kentät omiksi kokonaisuuksikseen, aivan kuten kirjan taulukoissakin on tehty (kts. taulukko 2.4 ja kuva 2.1).

Opiskelijoiden kannalta tärkeä parametri on simulaattorin **saatavuus**. Tähän liittyy esimerkiksi käyttöjärjestelmien tuki. Jo tutkittavien simulaattorien valintavaiheessa simulaattoreilta edellytettiin ainakin jonkinlaista tukea nykyaikaisille käyttöjärjestelmille. Mitä useampaa alustaa simulaattori tukee, sitä parempi saatavuus.

Myös simulaattorin **avoimuus** on tärkeä parametri, ja se liittyykin osittain saatavuuteen. Kaikki maksulliset ja erillistä lisenssiä vaativat simulaattorit on jo valintavaiheessa suljettu tutkimuksen ulkopuolelle. Simulaattori voi kuitenkin olla ilmainen, mutta silti suljettua lähdekoodia. Avoin lähdekoodi katsotaan eduksi, sillä se alentaa opiskelijoiden kynnystä ladata simulaattori tarvittaessa omalle tietokoneelleen. Mahdollinen latauksen aikainen tarve tunnuksen luonnille laskee simulaattorin saatavuutta.

4 RISC-V:N SIMULOINTI

4.1 Simulaattorivalinnat

Markkinoilta löytyy useita RISC-V:tä simuloivia simulaattoreita. Tutkimusta varten simulaattorien määrä on rajattu kolmeen simulaattoriin, jotka on listattu taulukossa 4.1. Rajaukseen ovat vaikuttaneet valintakriteerien täyttyminen sekä vertailuparametrien osuvuus. Arviot näistä on tehty simulaattorien ensivaikutelmien pohjalta. Myös simulaattorien mahdollinen aiempi käyttö opetuksessa nosti kyseisten simulaattorien houkuttelevuutta tutkimuksen kannalta.

Taulukko 4.1. *Tutkimukseen valitut simulaattorit*

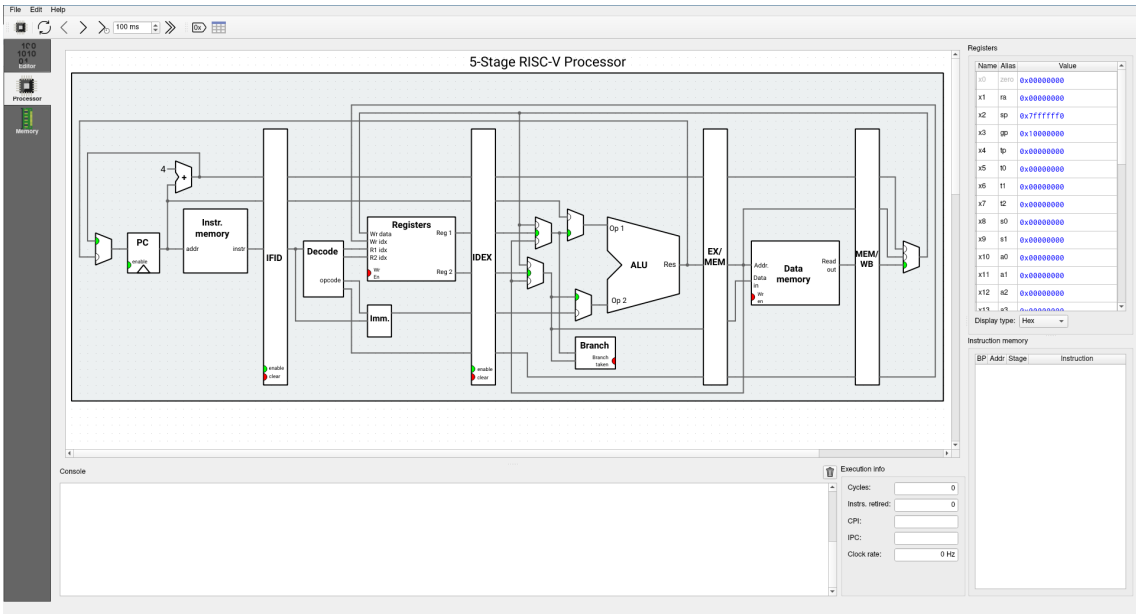
Simulaattori
Ripes [25]
Venus [26, 27, 28]
Spike [29]

Kaikki kolme simulaattoria ovat luonteeltaan hyvin erilaisia. Tutkimus ei ota kantaa siihen, onko näistä vain yksi ylitse muiden. Lopputulemana on mahdollista valita myös useampi simulaattori näistä kolmesta vaihtoehdosta. Vaihtoehtoja on esitelty tarkemmin kappaleissa 4.2, 4.3 ja 4.4.

4.2 Simulaattori 1: Ripes

Morten Petersenin kirjoittama *Ripes* [25] muistuttaa monilta osin aiemmin käytettyä *MipsIT*-simulaattoria [20]. MipsIT:n tapaan ohjelmasta löytyy graafinen liukuhihnanäkymä, joka on esitetty kuvassa 4.1. Liukuhihnanäkymässä näkyy prosessorin lohkokaavio. Lohkojen välillä kulkevia johtoja voidaan valita hiirellä klikkaamalla, jolloin johto korostetaan värillä. Näin on helpompaa erottaa mihin kaikkiin sisääntuloihin esimerkiksi multiplekserin ulostulo kytkeytyy. Lohkojen ja komponenttien ulostulot voidaan näyttää lukuarvona suoraan lohkokaaviossa. Lohokovaavio näyttää myös hyvin samankaltaiselta kurssimateriaalin [7] kanssa.

Ohjemasta löytyy kolme välilehteä: Koodieditori, prosessorinäkymä ja muistieditori. Koodieditoriin kirjoitetaan ajettava ohjelma joko RISC-V-assemblynä tai C-koodina. Ohjelmaa



Kuva 4.1. *Ripes-simulaattori*

voidaan suorittaa joko valitulla kellotaajuudella, tai kellosykli kerrallaan. Ajettavassa ohjelmassa voidaan myös asettaa keskeytyskohtia tai palata taaksepäin. Ripes tukee kaikkia RV32IM-käskykannan käskyjä ja pseudokäskyjä.

Prosessorinäkymässä nähdään valitun prosessorin liukuhihna. Ripes tukee erilaisia prosessoreja. *Single Cycle processor* suorittaa jokaisen käskyn yksitellen koko liukuhihnan läpi yhdellä kellosyklillä. *5-Stage Processor* toimii 5-portaisena liukuhihnana, ja se kykenee suorittamaan käskyjä rinnakkain eri liukuhihnavaiheissa. Jälkimmäinen prosessorityyppi sisältää myös tuen hasardien tunnistamiselle ja forwardoinnille, mutta ohjelmaa voidaan ajaa prosessorilla myös ilman näitä ominaisuuksia. Jokaista prosessorityyppiä voidaan tarkastella joko *Standard*- tai *Extended*-tilassa. Extended-tila lisää standardinäkymän datapolun rinnalle lisäinformaatiota liukuhihnan kontrollipolusta.

Muistinäkymässä voidaan tarkastella muistia, jota voidaan käsitellä ohjelmakoodin puolelta käsin *load*- ja *store*-käskyillä. Myös välimuisteja (datamuisti ja käskymuisti) tuetaan, ja näiden parametreja voidaan muuttaa (esimerkiksi write-back, write-allocate).

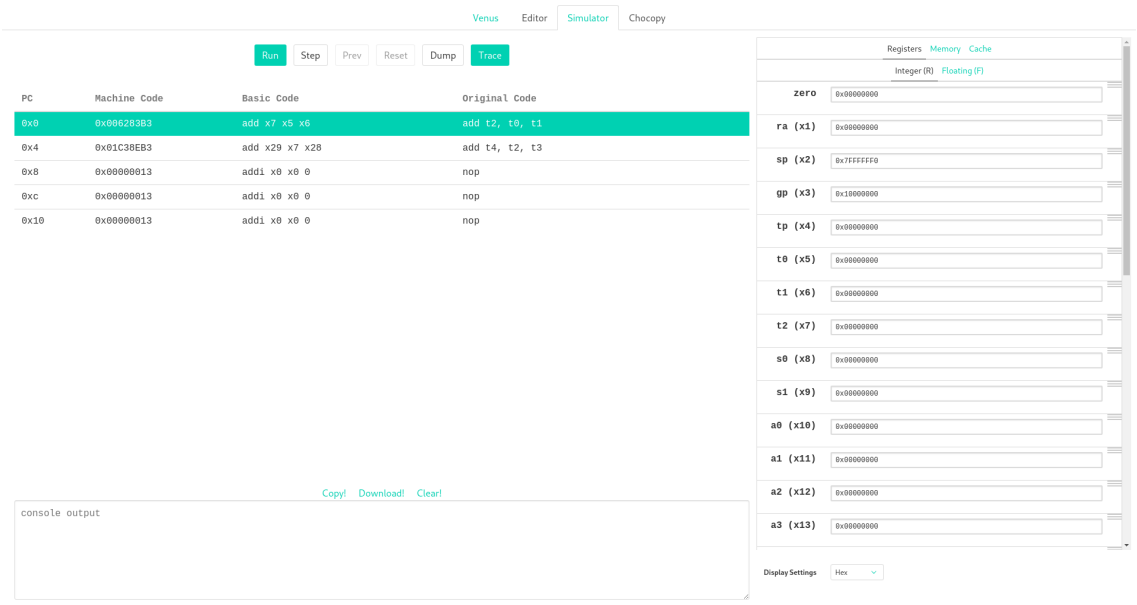
Ripes on testattu toimivaksi sekä Windowsilla että Linuxilla. Ohjelmointikielenä on käytetty enimmäkseen C++:aa ja Qt-kirjastoja. Lähdekoodi on avointa ja lisensoitu MIT-lisenssin alle. [25]

4.3 Simulaattori 2: Venus

Venus on alunperin Berkeleyn yliopistossa opiskelleen Keyhan Vakilin opetuskäyttöä varten kehittämä yksinkertainen RISC-V-simulaattori [26]. Venuksen lähdekoodi on MIT-lisenssin mukaisesti avointa. GitHubin perusteella Vakil ei itse enää ylläpidä Venusta, vaan sen kehitystä on jatkanut Stephan Kaminsky. [27]

Venus tukee suurinta osaa RV32IM-käskykannan käskyistä ja tuetuiksi käskyiksi muuntuvia pseudokäskyjä. Simulaattorilla on mahdollista muuttaa rekistereissä, muistissa ja välimuistissa olevia arvoja käsin myös kesken ohjelman suorituksen. Ohjelmaa voidaan suorittaa askel kerrallaan eteen tai taaksepäin.

Venus poikkeaa Ripes:sta siten, että siitä puuttuu graafinen liukuhihnanäkymä. Se on siis pelkkä Assembly-simulaattori. Venus myös kykenee ainoastaan *Single cycle* -tyyppiseen suoritukseen, eikä se siis tue käskyjen 5-portaista liukuhihnoittamista kuten Ripes. Kuva 4.2 on kuvakaappaus Venus-simulaattorin käyttöliittymästä.



Kuva 4.2. *Venus-simulaattori*

Venusta ajetaan selaimessa, eli se ei vaadi käyttäjältään minkäänlaista asennusta, ja on siksi todella hyvin alustariippumaton [27]. Se on kuitenkin kirjoitettu Javan kanssa yhteensopivalla Kotlin-ohjelmointikielellä, joten käyttäjä voi halutessaan asentaa koneelleen simulaattorin JVM-version.

4.4 Simulaattori 3: Spike

Spike on ehkä tunnetuin ja merkittävin RISC-V-simulaattori. Se tukee lähes kaikkia olemassa olevia RISC-V-laajennuksia. Spike on avointa lähdekoodia ja sitä saa jakaa eteenpäin. [29]

Spike tukee GDB:n käyttöä, joten ohjelmaa voidaan debugata C-ohjelmoijalle tutuilla työkaluilla [29]. Spiken asentaminen vaatii kuitenkin Linuxin osaamista, eikä asennusprosessikaan ole helppo. Tutkimuksen puitteissa ei saatu riittävän toimivaa asennusta aikaiseksi, jotta Spiken testaaminen ohjelmakoodilla olisi mielekästä. Simulaattorista voidaan kuitenkin mainita, että se ei sisällä graafista liukuhihnanäkymää, saati edes kovin graafista käyttöliittymää. Edistyksellinen käyttö vaatii siis komentoriviosaamista, mikä vaikuttaa

negatiivisesti helppokäyttöisyyteen.

Spikea on käytetty muissa yliopistoissa opetuskäytössä [21, 22, 23]. Tämä on toki perusteltua, sillä teollisuudessa sille on tarvetta sen laajan käskykantatuen vuoksi. Sitä voidaan myös käyttää yhdessä muiden emulointiympäristöjen kanssa, ja se tulee muun muassa *riscv-tools*-paketin mukana [30]. Tutkimukseen Spike on valittu mukaan referenssimielessä. Tutkimuksen kohdekurssin puitteissa Spike ei kuitenkaan anna mitään lisää muihin vaihtoehtoihin nähden, ja se on liian vaikeakäyttöinen.

4.5 Muut tarkastellut simulaattorit

Taulukossa 4.2 on listattuna muutamia muita markkinoilta löytyviä vaihtoehtoja. Osaa listatuista simulaattoreista on testattu lyhyehkösti, ja tämän jälkeen rajattu pois tutkimuksesta jonkun esiin nousseen piirteensä vuoksi. Osa taas on rajattu muiden tekijöiden vuoksi pois. Osaa ei ehditty testata lainkaan.

Taulukko 4.2. *Markkinoilta löytyviä simulaattoreita*

Simulaattori
Freedom Studio [31]
Jupiter [32]
riscOVPsim [33]
risc-v-simulator [34]
RISC-V Web Simulator [35]

Taulukon 4.2 simulaattoreista eniten huomiota sai SiFiven kehittämä *Freedom Studio*. Se on Eclipse-pohjainen RISC-V-kehitysympäristö, jolla voidaan emuloida SiFiven tarjoamia kehitysalustoja, ja simuloida RISC-V-käskykannalle käännettyä ohjelmakoodia (ainakin C tai assembly). Simulaattorin pitäisi toimia Linuxilla, Windowsilla ja macOS:lla. Vaikka ominaisuuksia simulaattorista löytyisi varmasti paljon, koitui sen käyttö kuitenkin turhan hankalaksi. Pelkästään *Hello world* -ohjelman suorittaminen koitui riittävän hankalaksi ohjelman Linux-versiolla. Tästä syystä sitä ei valittu mukaan tutkimukseen, vaikka työkalua oletettavasti käytetäänkin jopa teollisuudessa.

Jupiter on ominaisuuksiltaan hyvin Venuksen kaltainen, ja se on myös kehitetty opetuskäyttötarkoitusta varten. Toisin kuin suoraan selaimessa toimiva Venus, Jupiter vaatii myös asennuksen Tästäkään huolimatta se ei kuitenkaan tee mitään merkittävästi paremmin kuin Venus, minkä vuoksi se jätettiin pois tutkimuksesta.

5 PARAMETRIEN SOVITUS JA TUTKIMUSTULOKSET

Tutkimukseen valittuja simulaattoreita testattiin kokeilemalla niiden käyttöliittymää, ja ajamalla muutamia lyhyitä ohjelmakoodin pätkiä simulaattoreissa. Käytetyt ohjelmakoodit löytyvät liittestä C. Tutkimuksessa määriteltyjä vertailuparametreja sovitettiin valittuihin simulaattoreihin. Pisteytyksessä voidaan nähdä siltä osin epätarkkuutta, että pisteytysskaala on rajattu vain välille 1–3. Tästä johtuen osa pisteistä on korkeintaan suuntaa antavia, mutta tutkimuksen puitteissa tämä on kuitenkin riittävä tarkkuus. Yhteenveto tutkimustuloksista on esitetty taulukossa 5.1.

Taulukko 5.1. *Yhteenveto eri simulaattoreista ja niiden suoriutumisesta tutkimuksessa.*

	Simulaattori		
Parametri (max pts.)	Ripes	Venus	Spike
Kyky assemblyn simulointiin (3)	2	1	3
Tuki välimuistin simuloinnille (3)	3	3	3
Graafinen liukuhihnanäkymä (3)	3	1	1
Helppokäyttöisyys (3)	2	3	1
Visuaalisen ilmeen yhteneväisyys (3)	3	1	1
Saatavuus (3)	2	3	1
Avoimuus (3)	3	3	3
Yhteensä (21)	18	15	13

Ripes saa eniten pisteitä. Huomionarvoista on kuitenkin, että myös Spike pärjää yllättävän hyvin Ripesiin ja Venukseen verrattuna, eivätkä muutenkaan simulaattorien erot vielä näy kovin merkittävästi. Spike tukee suurinta määrää RISC-V-käskyjä, joten se loistaa assemblyn simulointikyvyssään. Ripesin suurin etu on puolestaan graafinen liukuhihnanäkymä. Venus taas loistaa web-käyttöliittymänsä vuoksi helppokäyttöisyydessä ja saatavuudessa, vaikka onkin näissä vain hieman Ripesia parempi. Tutkimuksen puitteissa kuitenkin osa parametreistä on tärkeämpiä kuin toiset. Lopulliset tulokset, eli taulukon 5.1 tulokset kerrottuna taulukossa 3.1 kuvatuilla parametrien painotuksilla on esitettynä taulukossa 5.2.

Taulukko 5.2. *Simulaattorien pisteytykset kerrottuna parametrien painotuksella.*

	Simulaattori		
Parametri (max pts.)	Ripes	Venus	Spike
Kyky assemblyn simulointiin (9)	6	3	9
Tuki välimuistin simuloinnille (3)	3	3	3
Graafinen liukuhihnanäkymä (6)	6	2	2
Helppokäyttöisyys (9)	6	9	3
Visuaalisen ilmeen yhteneväisyys (3)	3	1	1
Saatavuus (9)	6	9	3
Avoimuus (9)	9	9	9
Yhteensä (48)	39	36	30

Spike kärsii muun muassa vaikeakäyttöisyydestään. Myöskään tutkimuksen kohteena olevan kurssin puitteissa paremmasta simulointikyvystä ei ole merkittävää etua. Näin ollen erot muihin simulaattorien välillä kasvavat kun otetaan huomioon parametrien painotukset, eikä Spikea luultavasti kannata valita kurssin käyttöön.

Painotustenkin jälkeen Ripes saa edelleen eniten pisteitä. Myös Venus kuitenkin suoriutuu sen verran hyvin, että sitäkään ei kannata suorilta unohtaa, ja harjoitustyön suunnittelussa on suotavaa pohtia molempien käyttöä. Ripesia ja Venusta voidaan myös hyödyntää hyvin erityyppisissä tehtävissä, sillä toinen pärjää hyvin joissain käyttötarkoituksissa, kun taas toinen hieman erilaisessa käytössä. Tutkimustuloksina voidaan siis todeta sekä Ripesin että Venuksen tulleen valituksi.

6 YHTEENVETO JA TUTKIMUKSEN JÄLKEINEN TYÖ

Tutkimuksen tarkoituksena oli löytää sopiva RISC-V-simulaattori, jota voitaisiin käyttää jatkossa Tampereen yliopistossa kurssilla *Tietokoneen arkkitehtuuri*. Tässä onnistuttiin, ja sopivia simulaattoreita löydettiinkin jopa 2 kappaletta. Simulaattorit ovat keskenään hyvin erilaisia, mutta suoriutuvat tehtävästään riittävän hyvin, jotta niille voitaisiin löytää käyttöä kurssin puitteissa.

Tutkimuksen ulkopuolelle jää varsinaisten harjoitustöiden suunnittelu ja laatiminen, sekä myöhempi testaaminen kurssin opiskelijoilla. Harjoitustöissä (tai mahdollisesti viikkoharjoituksissa) voitaisiin käyttää tutkimuksessa valittuja simulaattoreita. Simulaattorien avulla, tehtävästä riippuen, luotaisiin joko oma ohjelma tai vaihtoehtoisesti tutkittaisiin tehtävänannossa annettua ohjelmaa.

RISC-V-assembly tulee olemaan merkittävänä osana harjoitustöitä. Kurssilla aiemmin käytetyissä MIPS-harjoitustöissä ei ole assemblyä juurikaan itse kirjoitettu, mutta sitä on pitänyt osata simuloida, ja tulkita sen toimintaa. Assemblyn itse kirjoittaminen voisi kuitenkin olla yksi harjoitustyö, tai mahdollisesti osa viikkoharjoituksia, mikäli sellaiset kurssille tehdään tulevaisuudessa. Assemblyä on nimittäin helpompi ymmärtää, jos sitä on edes muutaman rivin joskus kirjoittanut. Kurssin harjoituksissa voisikin esimerkiksi kirjoittaa RISC-V-assemblyllä yhden tai useamman lyhyen ohjelman, joita voitaisiin simuloida esimerkiksi Venus-simulaattorilla. Tämän kaltaiset harjoitukset voisivat sopia esimerkiksi kurssin alkupuolelle.

Kurssin loppupuolella voitaisiin vaihtaa monipuolisempaan Ripes-simulaattoriin, jolla tehtäisiin varsinainen harjoitustyö tai -työt. Vanhoissa harjoitustöissä on pitänyt muun muassa tutkia käskyjen toimintaa liukuhihnalla, ja paikantaa hasardeja ja niiden aiheuttajia. Tämänkaltainen tehtävä voisi jatkossakin olla yksi osa kurssin harjoitustöitä. Myös koodin debuggaamista voidaan helposti sisällyttää tähän tehtävään.

Lopullinen harjoitustöiden muoto jää varmasti vielä auki, ja se tulee riippumaan myös paljon kurssin tulevasta sisällöstä. Jotta harjoitustyö ylipäätään voidaan toteuttaa, tarvitsee kurssin opetusmateriaalit ja arkkitehtuuriesimerkit ensin muuntaa MIPS-arkkitehtuurista RISC-V:een. Harjoitustyön vaativuutta ajatellen jää myös nähtäväksi, kuinka paljon opetettavia asioita halutaan jättää pelkästään harjoitustyön puolelle, vai onko kaikkia harjoitustöissä eteen tulevia käsitteitä tarkoitus edes pintaraapaista ennen harjoitustyötä.

LÄHTEET

[1] Gayde, W. *How Arm Came to Dominate the Mobile Market*. Maaliskuu 2020. URL: https://www.techspot.com/article/1989-arm-inside/ (viitattu 01. 01. 2021).

[2] Choi, M. ja Lim, S.-H. x86-Android performance improvement for x86 smart mobile devices. eng. *Concurrency and computation* 28.10 (2016), 2770–2780. ISSN: 1532-0626.

[3] Ltd, A. *How Arm Licensing Works*. 2015. URL: https://www.arm.com/why-arm/how-licensing-works (viitattu 01. 01. 2021).

[4] RISC-V Foundation. *RISC-V Foundation Website*. URL: https://riscv.org/ (viitattu 19. 10. 2020).

[5] Ledin, J. *Modern Computer Architecture and Organization: Learn X86, ARM, and RISC-V Architectures and the Design of Smartphones, PCs, and Cloud Servers*. eng. Birmingham: Packt Publishing, Limited, 2020. ISBN: 9781838984397.

[6] Waterman, A. ja Asanović, K. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA*. Document Version 20191213. RISC-V Foundation. Joulukuu 2019.

[7] Patterson, D. A. ja Hennessy, J. L. *Computer organization and design RISC-V edition : the hardware/software interface*. eng. 1. ed. Amsterdam ; Elsevier/ Morgan Kaufmann, 2017. ISBN: 9780128122754.

[8] Research, S. *RISC-V Market Analysis: The New Kid on the Block*. Kesäkuu 2019.

[9] Osier-Mixon, J. *Semico Forecasts Strong Growth for RISC-V - RISC-V International*. Marraskuu 2019. URL: https://riscv.org/announcements/2019/11/9679/ (viitattu 03. 12. 2020).

[10] *Compound Annual Growth Rate, Encyclopedia of Education Economics and Finance*. 2014.

[11] *Rate of Change (ROC)*. 2020. URL: https://www.investopedia.com/terms/r/rateofchange.asp (viitattu 03. 12. 2020).

[12] Omdia. 2020. URL: https://omdia.tech.informa.com/who-we-are/about-us (viitattu 03. 12. 2020).

[13] Tractica. *RISC-V Processors*. 2019.

[14] Businesswire.com. *RISC-V Is Experiencing a Period of Optimism and Growth with Global Revenue Expected to Reach $1.1 Billion by 2025, According to Tractica*. Elokuu 2019. URL: https://www.businesswire.com/news/home/20190827005048/en/RISC-V-Is-Experiencing-a-Period-of-Optimism-and-Growth-with-Global-Revenue-Expected-to-Reach-1.1-Billion-by-2025-According-to-Tractica (viitattu 03. 12. 2020).

[15] Takahashi, D. *SiFive launches open source RISC-V custom chip*. Marraskuu 2016. URL: https://venturebeat.com/2016/11/29/sifive-launches-open-source-risc-v-custom-chip/ (viitattu 26. 10. 2020).

[16] *DesignShare product description*. SiFive. URL: https://www.sifive.com/designshare (viitattu 26. 10. 2020).

[17] *Xilinx products: Arty A7-35T: Artix-7 FPGA Development Board for Makers and Hobbyists*. Xilinx, Inc. URL: https://www.xilinx.com/products/boards-and-kits/1-elhaap.html (viitattu 26. 10. 2020).

[18] Patterson, D. A. ja Hennessy, J. L. *Computer Organization and Design, Fourth Edition: The Hardware/Software Interface*. eng. Elsevier Science & Technology, 2008. ISBN: 9780123744937.

[19] Smith, R. *MIPS Acquired by AI Hardware Vendor Wave Computing*. URL: https://www.anandtech.com/show/12989/mips-acquired-by-wave-computing (viitattu 19. 10. 2020).

[20] Brorsson, M. MipsIt: a simulation and development environment using animation for computer architecture education (toukokuu 2002). DOI: 10.1145/1275462.1275479.

[21] Schoeberl, M. 2017. URL: https://github.com/schoeberl/cae-lab (viitattu 19. 10. 2020).

[22] URL: https://inst.eecs.berkeley.edu/~cs61c/sp19/labs/lab05/ (viitattu 19. 10. 2020).

[23] URL: https://inst.eecs.berkeley.edu/~cs250/fa13/handouts/ (viitattu 23. 11. 2020).

[24] Schoeberl, M. 2017. URL: http://www2.imm.dtu.dk/courses/02155/ (viitattu 19. 10. 2020).

[25] Petersen, M. B. *Ripes*. URL: https://github.com/mortbopet/Ripes (viitattu 19. 10. 2020).

[26] Vakil, K. *Venus*. URL: https://github.com/kvakil/venus (viitattu 19. 10. 2020).

[27] Kaminsky, S. *Venus*. URL: https://github.com/ThaumicMekanism/venus (viitattu 19. 11. 2020).

[28] Kaminsky, S. *Venus*. URL: https://venus.cs61c.org/ (viitattu 19. 11. 2020).

[29] RISC-V Foundation. *Spike*. URL: https://github.com/riscv/riscv-isa-sim (viitattu 19. 10. 2020).

[30] riscv. *riscv/riscv-tools*. Kesäkuu 2018. URL: https://github.com/riscv/riscv-tools (viitattu 01. 01. 2021).

[31] *Software - SiFive*. 2020. URL: https://www.sifive.com/software (viitattu 07. 12. 2020).

[32] Castellanos, A. *Jupiter*. URL: https://github.com/andrescv/Jupiter (viitattu 19. 10. 2020).

[33] Imperas. *riscOVPsim*. URL: https://www.imperas.com/riscvovpsim-free-imperas-riscv-instruction-set-simulator (viitattu 19. 10. 2020).

[34] Shaban, M. *risc-v-simulator*. URL: https://github.com/ProfessorShaban/risc-v-simulator (viitattu 19. 10. 2020).

[35] Borin, E. *RISC-V Web Simulator*. URL: https://www.ic.unicamp.br/~edson/disciplinas/mc404/2019-2s/ab/sim6/web/static/sim.html (viitattu 19. 10. 2020).

A RV32/64IMAFD-KÄSKYKANNAN STANDARDIKÄSKYT

Tässä liitteessä on koostettuna kaikki RV32IMAFD- ja RV64IMAFD-käskykantojen käskyt. Taulukot ovat peräisin RISC-V-spesifikaatiosta. [6, s. 90–94]

RV32I Base Integer Instruction Set

	Source Registers	Destination Registers	Accumulating CSRs
LUI		*rd*	
AUIPC		*rd*	
JAL		*rd*	
JALR†	*rs1*	*rd*	
BEQ	*rs1*, *rs2*		
BNE	*rs1*, *rs2*		
BLT	*rs1*, *rs2*		
BGE	*rs1*, *rs2*		
BLTU	*rs1*, *rs2*		
BGEU	*rs1*, *rs2*		
LB†	$rs1^A$	*rd*	
LH†	$rs1^A$	*rd*	
LW†	$rs1^A$	*rd*	
LBU†	$rs1^A$	*rd*	
LHU†	$rs1^A$	*rd*	
SB	$rs1^A$, $rs2^D$		
SH	$rs1^A$, $rs2^D$		
SW	$rs1^A$, $rs2^D$		
ADDI	*rs1*	*rd*	
SLTI	*rs1*	*rd*	
SLTIU	*rs1*	*rd*	
XORI	*rs1*	*rd*	
ORI	*rs1*	*rd*	
ANDI	*rs1*	*rd*	
SLLI	*rs1*	*rd*	
SRLI	*rs1*	*rd*	
SRAI	*rs1*	*rd*	
ADD	*rs1*, *rs2*	*rd*	
SUB	*rs1*, *rs2*	*rd*	
SLL	*rs1*, *rs2*	*rd*	
SLT	*rs1*, *rs2*	*rd*	
SLTU	*rs1*, *rs2*	*rd*	
XOR	*rs1*, *rs2*	*rd*	
SRL	*rs1*, *rs2*	*rd*	
SRA	*rs1*, *rs2*	*rd*	
OR	*rs1*, *rs2*	*rd*	
AND	*rs1*, *rs2*	*rd*	
FENCE			
FENCE.I			
ECALL			
EBREAK			

RV32I Base Integer Instruction Set (continued)

	Source Registers	Destination Registers	Accumulating CSRs	
CSRRW‡	*rs1*, *csr**	*rd*, *csr*		*unless *rd*=x0
CSRRS‡	*rs1*, *csr*	*rd**, *csr*		*unless *rs1*=x0
CSRRC‡	*rs1*, *csr*	*rd**, *csr*		*unless *rs1*=x0

‡carries a dependency from *rs1* to *csr* and from *csr* to *rd*

RV32I Base Integer Instruction Set (continued)

	Source Registers	Destination Registers	Accumulating CSRs	
CSRRWI‡	*csr**	*rd*, *csr*		*unless *rd*=x0
CSRRSI‡	*csr*	*rd*, *csr**		*unless uimm[4:0]=0
CSRRCI‡	*csr*	*rd*, *csr**		*unless uimm[4:0]=0

‡carries a dependency from *csr* to *rd*

RV64I Base Integer Instruction Set

	Source Registers	Destination Registers	Accumulating CSRs
LWU†	$rs1^A$	*rd*	
LD†	$rs1^A$	*rd*	
SD	$rs1^A$, $rs2^D$		
SLLI	*rs1*	*rd*	
SRLI	*rs1*	*rd*	
SRAI	*rs1*	*rd*	
ADDIW	*rs1*	*rd*	
SLLIW	*rs1*	*rd*	
SRLIW	*rs1*	*rd*	
SRAIW	*rs1*	*rd*	
ADDW	*rs1*, *rs2*	*rd*	
SUBW	*rs1*, *rs2*	*rd*	
SLLW	*rs1*, *rs2*	*rd*	
SRLW	*rs1*, *rs2*	*rd*	
SRAW	*rs1*, *rs2*	*rd*	

RV32M Standard Extension

	Source Registers	Destination Registers	Accumulating CSRs
MUL	*rs1*, *rs2*	*rd*	
MULH	*rs1*, *rs2*	*rd*	
MULHSU	*rs1*, *rs2*	*rd*	
MULHU	*rs1*, *rs2*	*rd*	
DIV	*rs1*, *rs2*	*rd*	
DIVU	*rs1*, *rs2*	*rd*	
REM	*rs1*, *rs2*	*rd*	
REMU	*rs1*, *rs2*	*rd*	

RV64M Standard Extension

	Source Registers	Destination Registers	Accumulating CSRs
MULW	*rs1*, *rs2*	*rd*	
DIVW	*rs1*, *rs2*	*rd*	
DIVUW	*rs1*, *rs2*	*rd*	
REMW	*rs1*, *rs2*	*rd*	
REMUW	*rs1*, *rs2*	*rd*	

RV32A Standard Extension

	Source Registers	Destination Registers	Accumulating CSRs	
LR.W†	$rs1^A$	*rd*		
SC.W†	$rs1^A$, $rs2^D$	*rd**		*if successful
AMOSWAP.W†	$rs1^A$, $rs2^D$	*rd*		
AMOADD.W†	$rs1^A$, $rs2^D$	*rd*		
AMOXOR.W†	$rs1^A$, $rs2^D$	*rd*		
AMOAND.W†	$rs1^A$, $rs2^D$	*rd*		
AMOOR.W†	$rs1^A$, $rs2^D$	*rd*		
AMOMIN.W†	$rs1^A$, $rs2^D$	*rd*		
AMOMAX.W†	$rs1^A$, $rs2^D$	*rd*		
AMOMINU.W†	$rs1^A$, $rs2^D$	*rd*		
AMOMAXU.W†	$rs1^A$, $rs2^D$	*rd*		

RV64A Standard Extension

	Source Registers	Destination Registers	Accumulating CSRs	
LR.D†	$rs1^A$	*rd*		
SC.D†	$rs1^A$, $rs2^D$	*rd**		*if successful
AMOSWAP.D†	$rs1^A$, $rs2^D$	*rd*		
AMOADD.D†	$rs1^A$, $rs2^D$	*rd*		
AMOXOR.D†	$rs1^A$, $rs2^D$	*rd*		
AMOAND.D†	$rs1^A$, $rs2^D$	*rd*		
AMOOR.D†	$rs1^A$, $rs2^D$	*rd*		
AMOMIN.D†	$rs1^A$, $rs2^D$	*rd*		
AMOMAX.D†	$rs1^A$, $rs2^D$	*rd*		
AMOMINU.D†	$rs1^A$, $rs2^D$	*rd*		
AMOMAXU.D†	$rs1^A$, $rs2^D$	*rd*		

RV32F Standard Extension

	Source Registers	Destination Registers	Accumulating CSRs	
FLW†	*rs1*^{A}	*rd*		
FSW	*rs1*^{A}, *rs2*^{D}			
FMADD.S	*rs1*, *rs2*, *rs3*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FMSUB.S	*rs1*, *rs2*, *rs3*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FNMSUB.S	*rs1*, *rs2*, *rs3*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FNMADD.S	*rs1*, *rs2*, *rs3*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FADD.S	*rs1*, *rs2*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FSUB.S	*rs1*, *rs2*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FMUL.S	*rs1*, *rs2*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FDIV.S	*rs1*, *rs2*, frm*	*rd*	NV, DZ, OF, UF, NX	*if rm=111
FSQRT.S	*rs1*, frm*	*rd*	NV, NX	*if rm=111
FSGNJ.S	*rs1*, *rs2*	*rd*		
FSGNJN.S	*rs1*, *rs2*	*rd*		
FSGNJX.S	*rs1*, *rs2*	*rd*		
FMIN.S	*rs1*, *rs2*	*rd*	NV	
FMAX.S	*rs1*, *rs2*	*rd*	NV	
FCVT.W.S	*rs1*, frm*	*rd*	NV, NX	*if rm=111
FCVT.WU.S	*rs1*, frm*	*rd*	NV, NX	*if rm=111
FMV.X.W	*rs1*	*rd*		
FEQ.S	*rs1*, *rs2*	*rd*	NV	
FLT.S	*rs1*, *rs2*	*rd*	NV	
FLE.S	*rs1*, *rs2*	*rd*	NV	
FCLASS.S	*rs1*	*rd*		
FCVT.S.W	*rs1*, frm*	*rd*	NX	*if rm=111
FCVT.S.WU	*rs1*, frm*	*rd*	NX	*if rm=111
FMV.W.X	*rs1*	*rd*		

RV64F Standard Extension

	Source Registers	Destination Registers	Accumulating CSRs	
FCVT.L.S	*rs1*, frm*	*rd*	NV, NX	*if rm=111
FCVT.LU.S	*rs1*, frm*	*rd*	NV, NX	*if rm=111
FCVT.S.L	*rs1*, frm*	*rd*	NX	*if rm=111
FCVT.S.LU	*rs1*, frm*	*rd*	NX	*if rm=111

RV32D Standard Extension

	Source Registers	Destination Registers	Accumulating CSRs	
FLD†	*rs1*^{A}	*rd*		
FSD	*rs1*^{A}, *rs2*^{D}			
FMADD.D	*rs1*, *rs2*, *rs3*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FMSUB.D	*rs1*, *rs2*, *rs3*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FNMSUB.D	*rs1*, *rs2*, *rs3*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FNMADD.D	*rs1*, *rs2*, *rs3*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FADD.D	*rs1*, *rs2*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FSUB.D	*rs1*, *rs2*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FMUL.D	*rs1*, *rs2*, frm*	*rd*	NV, OF, UF, NX	*if rm=111
FDIV.D	*rs1*, *rs2*, frm*	*rd*	NV, DZ, OF, UF, NX	*if rm=111
FSQRT.D	*rs1*, frm*	*rd*	NV, NX	*if rm=111
FSGNJ.D	*rs1*, *rs2*	*rd*		
FSGNJN.D	*rs1*, *rs2*	*rd*		
FSGNJX.D	*rs1*, *rs2*	*rd*		
FMIN.D	*rs1*, *rs2*	*rd*	NV	
FMAX.D	*rs1*, *rs2*	*rd*	NV	
FCVT.S.D	*rs1*, frm*	*rd*	NX	*if rm=111
FCVT.D.S	*rs1*, frm*	*rd*	NX	*if rm=111
FEQ.D	*rs1*, *rs2*	*rd*	NV	
FLT.D	*rs1*, *rs2*	*rd*	NV	
FLE.D	*rs1*, *rs2*	*rd*	NV	
FCLASS.D	*rs1*	*rd*		
FCVT.W.D	*rs1*, frm*	*rd*	NV, NX	*if rm=111
FCVT.WU.D	*rs1*, frm*	*rd*	NV, NX	*if rm=111
FCVT.D.W	*rs1*	*rd*		
FCVT.D.WU	*rs1*	*rd*		

RV64D Standard Extension

	Source Registers	Destination Registers	Accumulating CSRs	
FCVT.L.D	*rs1*, frm*	*rd*	NV, NX	*if rm=111
FCVT.LU.D	*rs1*, frm*	*rd*	NV, NX	*if rm=111
FMV.X.D	*rs1*	*rd*		
FCVT.D.L	*rs1*, frm*	*rd*	NX	*if rm=111
FCVT.D.LU	*rs1*, frm*	*rd*	NX	*if rm=111
FMV.D.X	*rs1*	*rd*		

B RISC-V-PSEUDOKÄSKYT

RISC-V sisältää paljon pseudokäskyjä. Taulukoissa B.1 ja B.2 on listattuna kaikki pseudokäskyt. Taulukot ovat peräisin RISC-V-spesifikaatiosta. [6, s. 139–140]

Taulukko B.1. *RISC-V pseudoinstructions. [6, s. 139]*

pseudoinstruction	Base Instruction(s)	Meaning
`la rd, symbol` (*non-PIC*)	`auipc rd, delta`$[31:12]+$`delta`$[11]$ `addi rd, rd, delta[11:0]`	Load absolute address, where `delta` $=$ `symbol` $-$ `pc`
`la rd, symbol` (*PIC*)	`auipc rd, delta`$[31:12]+$`delta`$[11]$ `l{w|d} rd, rd, delta[11:0]`	Load absolute address, where `delta` $=$ `GOT`$[$`symbol`$]-$ `pc`
`lla rd, symbol`	`auipc rd, delta`$[31:12]+$`delta`$[11]$ `addi rd, rd, delta[11:0]`	Load local address, where `delta` $=$ `symbol` $-$ `pc`
`l{b|h|w|d} rd, symbol`	`auipc rd, delta`$[31:12]+$`delta`$[11]$ `l{b|h|w|d} rd, delta[11:0](rd)`	Load global
`s{b|h|w|d} rd, symbol, rt`	`auipc rt, delta`$[31:12]+$`delta`$[11]$ `s{b|h|w|d} rd, delta[11:0](rt)`	Store global
`fl{w|d} rd, symbol, rt`	`auipc rt, delta`$[31:12]+$`delta`$[11]$ `fl{w|d} rd, delta[11:0](rt)`	Floating-point load global
`fs{w|d} rd, symbol, rt`	`auipc rt, delta`$[31:12]+$`delta`$[11]$ `fs{w|d} rd, delta[11:0](rt)`	Floating-point store global

The base instructions use `pc`*-relative addressing, so the linker subtracts* `pc` *from* `symbol` *to get* `delta`*. The linker adds* `delta[11]` *to the 20-bit high part, counteracting sign extension of the 12-bit low part.*

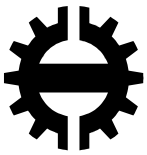
`nop`	`addi x0, x0, 0`	No operation
`li rd, immediate`	*Myriad sequences*	Load immediate
`mv rd, rs`	`addi rd, rs, 0`	Copy register
`not rd, rs`	`xori rd, rs, -1`	One's complement
`neg rd, rs`	`sub rd, x0, rs`	Two's complement
`negw rd, rs`	`subw rd, x0, rs`	Two's complement word
`sext.w rd, rs`	`addiw rd, rs, 0`	Sign extend word
`seqz rd, rs`	`sltiu rd, rs, 1`	Set if $=$ zero
`snez rd, rs`	`sltu rd, x0, rs`	Set if $\neq$ zero
`sltz rd, rs`	`slt rd, rs, x0`	Set if $<$ zero
`sgtz rd, rs`	`slt rd, x0, rs`	Set if $>$ zero
`fmv.s rd, rs`	`fsgnj.s rd, rs, rs`	Copy single-precision register
`fabs.s rd, rs`	`fsgnjx.s rd, rs, rs`	Single-precision absolute value
`fneg.s rd, rs`	`fsgnjn.s rd, rs, rs`	Single-precision negate
`fmv.d rd, rs`	`fsgnj.d rd, rs, rs`	Copy double-precision register
`fabs.d rd, rs`	`fsgnjx.d rd, rs, rs`	Double-precision absolute value
`fneg.d rd, rs`	`fsgnjn.d rd, rs, rs`	Double-precision negate
`beqz rs, offset`	`beq rs, x0, offset`	Branch if $=$ zero
`bnez rs, offset`	`bne rs, x0, offset`	Branch if $\neq$ zero
`blez rs, offset`	`bge x0, rs, offset`	Branch if $\leq$ zero
`bgez rs, offset`	`bge rs, x0, offset`	Branch if $\geq$ zero
`bltz rs, offset`	`blt rs, x0, offset`	Branch if $<$ zero
`bgtz rs, offset`	`blt x0, rs, offset`	Branch if $>$ zero
`bgt rs, rt, offset`	`blt rt, rs, offset`	Branch if $>$
`ble rs, rt, offset`	`bge rt, rs, offset`	Branch if $\leq$
`bgtu rs, rt, offset`	`bltu rt, rs, offset`	Branch if $>$, unsigned
`bleu rs, rt, offset`	`bgeu rt, rs, offset`	Branch if $\leq$, unsigned

Taulukko B.2. *RISC-V pseudoinstructions. [6, s. 140]*

pseudoinstruction	Base Instruction	Meaning
`j offset`	`jal x0, offset`	Jump
`jal offset`	`jal x1, offset`	Jump and link
`jr rs`	`jalr x0, 0(rs)`	Jump register
`jalr rs`	`jalr x1, 0(rs)`	Jump and link register
`ret`	`jalr x0, 0(x1)`	Return from subroutine
`call offset`	`auipc x1, offset`$[31:12]$ + `offset`$[11]$	Call far-away subroutine
	`jalr x1, offset[11:0](x1)`	
`tail offset`	`auipc x6, offset`$[31:12]$ + `offset`$[11]$	Tail call far-away subroutine
	`jalr x0, offset[11:0](x6)`	
`fence`	`fence iorw, iorw`	Fence on all memory and I/O
`rdinstret[h] rd`	`csrrs rd, instret[h], x0`	Read instructions-retired counter
`rdcycle[h] rd`	`csrrs rd, cycle[h], x0`	Read cycle counter
`rdtime[h] rd`	`csrrs rd, time[h], x0`	Read real-time clock
`csrr rd, csr`	`csrrs rd, csr, x0`	Read CSR
`csrw csr, rs`	`csrrw x0, csr, rs`	Write CSR
`csrs csr, rs`	`csrrs x0, csr, rs`	Set bits in CSR
`csrc csr, rs`	`csrrc x0, csr, rs`	Clear bits in CSR
`csrwi csr, imm`	`csrrwi x0, csr, imm`	Write CSR, immediate
`csrsi csr, imm`	`csrrsi x0, csr, imm`	Set bits in CSR, immediate
`csrci csr, imm`	`csrrci x0, csr, imm`	Clear bits in CSR, immediate
`frcsr rd`	`csrrs rd, fcsr, x0`	Read FP control/status register
`fscsr rd, rs`	`csrrw rd, fcsr, rs`	Swap FP control/status register
`fscsr rs`	`csrrw x0, fcsr, rs`	Write FP control/status register
`frrm rd`	`csrrs rd, frm, x0`	Read FP rounding mode
`fsrm rd, rs`	`csrrw rd, frm, rs`	Swap FP rounding mode
`fsrm rs`	`csrrw x0, frm, rs`	Write FP rounding mode
`frflags rd`	`csrrs rd, fflags, x0`	Read FP exception flags
`fsflags rd, rs`	`csrrw rd, fflags, rs`	Swap FP exception flags
`fsflags rs`	`csrrw x0, fflags, rs`	Write FP exception flags

C TIETOKONEEN ARKKITEHTUURI -KURSSIN VANHA HARJOITUSTYÖ 1

Tämä liite sisältää Tampereen yliopiston kurssin *Tietokoneen arkkitehtuuri* vanhan harjoitustyö 1:n. Harjoitustyön koodeja on käytetty simulaattorien vertailuun. Koodit ja kaikki muukin perustuu MIPS-arkkitehtuuriin, mutta ne ovat sovellettavissa RISC-V-ympäristöön lähes sellaisenaan.



TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan laitos
TIE-51200 Tietokoneen arkkitehtuuri

Harjoitustyö 1: Liukuhihna

Nimi	**Email**	**op. num.**

Liukuhihna-työ

Harjoitustyön tarkoitus

Tämän työn tarkoitus on auttaa ymmärtämään, kuinka prosessorin liukuhihna toimii.

Työssä käytetään mipsIT-simulaattoria. Simulaattori on tehty Ruotsissa Lundin yliopistossa.

Harjoitustyöstä palautetaan tämä dokumentti täydennettynä, paperilla. Deadline ja palautuspaikka mainitaan kurssin moodle2-sivulla tjsp.

Vastauksina kysymyksiin riittävät lyhyet mutta kuitenkin tarkat vastaukset. Vastaukset kirjoitetaan siististi lyijykynällä, jotta virheitä voi korjata. Suttuiset, hutaistut tai muulla tavoin epämääräiset palautukset johtavat bumerangiin tai harjoituksen kokonaan uudelleen tekemiseen.

Harjoituksessa ajettavat ohjelmanpätkät ovat assembly-ohjelmia, jotka pitää kääntää mipsit-IDE:llä , ja sen jälkeen ajaa mipspipeS- tai mipspipeXL-simulaattorissa.

Projektin tyypiksi pitää valita “assembler” eikä C/assembler.Työssä käytetään mipsIT-simulaattoria. Simulaattori on tehty Ruotsissa Lundin yliopistossa.

Ensimmäinen ohjelma, ajetaan mipspipeS-simulaattorissa. Aja ohjelmaa kellojakso kerrallaan(simulaattorin step-toiminto), näkymässä jossa prosessorin datapolku/liukuhihna on näkyvissä. Syötä rekistereihin t1 ja t2 jotain tuntemiasi arvoja ennen ohjelman käynnistämistä(ohjelman lataamisen jälkeen)

```
#include <iregdef.h>
        .set noreorder
        .text
        .globl start
        .ent start
start:
        add     t0, t1, t2
        nop
        nop
        nop
        nop
        .end start
```

Seuraavassa viidessä kysymyksissä selitä myös kyseisessä vaiheessa olevien muxien asennot ja syyt niiden asentoihin.

Mitä tapahtuu ensimmäisessä liukuhihnavaiheessa (IF) ?

_____

_____

Mitä tapahtuu toisessa liukuhihnavaiheessa (ID) ?

_____

_____

Mitä tapahtuu kolmannessa liukuhihnavaiheessa(EX)

_____

_____

Mitä tapahtuu neljännessä liukuhihnavaheessa(MEM)

_____

_____

Mitä tapahtuu viidennessä liukuhihnavaiheessa(WB)

_____

_____

Mistä englanninkielisistä sanoista liukuhihnavaiheiden nimet tulevat?

IF:_____ ID:_____

EX:_____ MEM:_____

WB:_____

Kuinka monen kellojakson kuluttua käskyn hakemisen alkamisesta käskyn tulos saadaan talletettua kohderekisteriin?

_____

Kuinka monen kellojakson kuluttua lähderekisterien lukemisesta käskyn tulos saadaan talletettua kohde rekisteriin?

_____

Missä liukuhihnavaiheessa eri aritmetiikkaoperaatiot eroavat toisistaan?

_____

Yhtä liukuhihnavaihetta ei käytetä aritmeettisissa operaatiossa? Mitä vaihetta? Miksi?

_____

Vaihda käskyksi ”lw t0, 0(t1), käännä ohjelma, uudelleenkäynnistä simulaattori ja lataa ohjelma simulaattoriiin.

Mikä aritmeettinen operaatio ALU-vaiheessa suoritetaan? Miksi?

_____

_____

_____

Käytetäänkö kaikkia liukuhihnavaiheita? Jos ei, miksei?

_____

_____

Vaihda käskyksi ”sw t0, 4(t1)” ja tutki sitä kuin yllä

Mikä aritmeettinen operaatio ALU-vaiheessa suoritetaan? Miksi?

_____

_____

Käytetäänkö kaikkia liukuhihnavaiheita? Jos ei, miksei?

_____

_____

Vaihda käskyksi ”beq t0, t1, Dest sekä lisää koodin loppuun label Dest.

Ohjelma siis seuraavanlainen

```
#include <iregdef.h>
        .set noreorder
        .text
        .globl start
        .ent start
start:
        beq t0, t1, Dest
        nop
        nop
        nop
        nop
Dest:
        nop
        .end start
```

Mitä missäkin liukuhihnavaiheessa tehdään?

Mikä aritmeettinen operaatio ALU-vaiheessa suoritetaan? Miksi? Jos et keksi vastausta, kokeile syöttää rekistereihin t0 ja t1 eri arvoja, jolloin voit laskuoperaation tuloksesta päätellä, mikä operaatio suoritettiin.

Käytetäänkö kaikkia liukuhihnavaiheita? Jos ei, miksei?

Liukuhihnaesimerkki

Aja seuraava ohjelma simulaattorilla

```
#include <iregdef.h>
        .set noreorder  # Avoid reordering instructions
        .text           # Start generating instructions
        .globl start    # The label should be globally known
        .ent start

start:                  # The label marks an entry point
        lui     $9, 0xbf90 # Load upper half of port address
                        # Lower half is filled with zeros
repeat:
        lbu $8, 0x0($9) # Read from the input port
        nop             # Needed after load
        sb $8, 0x0($9)  # Write to the output port
        b       repeat  # Repeat the read and write cycle
        nop             # Needed after branch
        .end start
```

Seuraa käskyjen suoritusta liukuhihnalla, kunnes kyllästyt.

Missä vaiheessa kyllästyit? Miksi?

_____

_____

_____

Tarkastellaan seuraavaa ohjelmaa

```
#include <iregdef.h>
        .set noreorder
        .text
        .globl start
        .ent start
start:
        add     t2, t0, t1
        add     t4, t2, t3
        nop
        nop
        nop
        .end start
```

Kuinka monen kellojakson jälkeen ensimmäisen käskyn kohderekisteri, t2, saavuttaa oikean arvonsa?

_____

Kuinka monen kellojakson jälkeen toinen käsky tarvii rekisterin t2 arvoa? _____

Huomaatko tässä jotain ongelmaa? Miksi tällaista hasardia kutsutaan? _____

Tällainen hasardi voidaan ratkaista koodin uudelleenjärjestelyllä, lisäämällä NOP-käskyjä, liukuhihnan pysäyttämisellä(hasardin tunnistus), tai forwarding(= bypassing)-tekniikalla.

Selitä miten NOPien lisääminen toimii? Kuka sen tekee?

_____

_____

_____

Selitä mitä koodin uudelleenjärjestämisellä tarkoitetaan? Kuka sen tekee?

_____

_____

_____

Selitä, miten liukuhihnan pysäyttäminen toimii? Kuka sen tekee?

_____

_____

_____

Oikea Mips-prosessori käyttää forwardingia näiden hasardien ratkaisemiseen. S-simulaattori ei tue forwardingia, XL-simulaattori tukee. Vaihda käyttöön XL-versio simulaattorista, ja aja ohjelman simulaatio tällä simulaattorilla.

Miten tunnistetaan, että forwardingia tulee käyttää?
(vastaukseksi ei kelpaa “hazard detection unit tunnistaa”, vaan sen hazard detection unitin toimintalogiikka, mistä se sen päättelee?)

_____

_____

_____

Mistä minne, ja milloin tässä esimerkissä forwardingia käytetään?
(Paikka liukuhihnalla)

_____

_____

Aja seuraava ohjelma S-simulaattorilla. Aseta rekistereihin t0 ja t1 sama arvo.

```
#include <iregdef.h>
        .set noreorder
        .text
        .globl start
        .ent start
start:
        nop
        nop
        beq t0, t1, start
        addi t0, t0, 1
        nop
        nop
        nop
        .end start
```

Kuinka monta kellojaksoa on kulunut, kun beq-käsky on valmis hyppäämään? _____

Mitä on tapahtunut sitä seuravalle addi-käskylle, kun haarautumista on laskettu?

_____

Mikä ongelma tässä tilanteessa on?

_____

_____

Miksikä tällaista hasardia kutsutaan?

_____

Mitkä ovat mahdollisia ratkaisuja tähän?

_____

_____

_____

Vaihda simulaattorin XL-versioon. Kuinka tämä suorittaa beq-käskyn?

_____

_____

Aja seuraava ohjelma S-simulaattorilla. Aseta rekistereihin t0 ja t1 eri arvot, ja t2een osoite, jonka päässä olevan muistin arvon tiedät. Aja koodia yksi kellojakso kerrallaan.

```
#include <iregdef.h>
        .set noreorder
        .text
        .globl start
        .ent start
start:  lw      t0, 0(t2)
        add     t1, t1, t0
        nop
        nop
        nop
        .end start
```

Kuinka monen kellojakson päästä latauskäskyn aloittamisesta kohderekisteri(t0) saavuttaa oikean arvon?

Kuinka monen kellojakson päästä latauskäskyn aloittamisesta t0-rekisterin arvo luetaan yhteenlaskukäskyssä?

Vaihda simulaattoriin XL. Tässä tarvittava arvo voidaan forwardoida liukuhihnalle seuraaville käskyille heti kun arvo on valmis.

Milloin latauskäskyssä ladattu arvo saapuu muistilta liukuhihnalle ja voidaan forwardoida eteenpäin?

Milloin yhteenlaskukäsky viimeistään tarvii tämän arvon(olettaen että tämä arvo forwardoidaan suoraan ALUlle eikä sitä lueta rekistereistä)

Mikä ongelma tästä seuraa?

Mainitse kaksi mahdollista ratkaisuehdotusta tähän ongelmaan. Kumpaa käytetään MIPSissä? Kumpaa arvelet x86ssa (PC) käytettävän ja miksi?

Palaute. Pakko vastata.

1) Oliko tässä harjoitustyössä mitään hyvää*?*

Kyllä ☐

Ei ☐

2) Kuinka kauan harjoitustyön tekemiseen meni aikaa? tunteja/oppilas _____

3) Mikä tässä harjoitustyössä oli opettavaisinta/mitä opin tästä?

4) Parantaisin tätä harjoitustyötä seuraavalla tavalla:

D TIETOKONEEN ARKKITEHTUURI -KURSSIN VANHA HARJOITUSTYÖ 2

Tämä liite sisältää Tampereen yliopiston kurssin *Tietokoneen arkkitehtuuri* vanhan harjoitustyö 2:n. Harjoitustyö on välimuistiin liittyvä harjoitustyö. Harjoitustyön luonteen vuoksi sitä ei ole tässä tutkimuksessa sivuttu kovin moneen otteeseen, mutta se on tässä kuitenkin mukana, sillä välimuistin simuloiminen oli yksi parametri simulaattorien vertailussa.

TIE-51200 Tietokoneen arkkitehtuuri

Harjoitustyö 2: Välimuistityö

Ryhmän numero _____

Jäsen #1
- nimi _____
- opiskelijanumero _____
- sähköposti _____

Jäsen #2
- nimi _____
- opiskelijanumero _____
- sähköposti _____

Jäsen #3
- nimi _____
- opiskelijanumero _____
- sähköposti _____

Palautettu _____

Bumerangi _____

Hyväksytty _____

Virheellisiä vastauksia _____

Arvosana _____

Vastaa lyhyesti, mutta muista ilmoittaa vastauksissa mittayksikkö. Jos ilmoitat osoitteita tai muita lukuarvoja 16-kantaisena eli heksadesimaalimuodossa, käytä etuliitettä 0x, esim. 0x100A. Jos jotain asetusta ei ole määritelty, käytä simulaattorin käynnistyksessä tarjoamaa oletusarvoa. Käynnistä tarvittaessa simulaattori uudestaan, jos olet välillä kokeillut erikoisia asetuksia. Pyri käyttämään oikeita termejä itse keksittyjen sijaan.

Obs! Huom! OMG! Työssä käytetään simulaattorin perusversiota (Mips.exe). Älä siis käytä simulaattorin liukuhihnoitettuja versioita.

Projektin tyyppi tulee olla aina C(minimal)/Assembler, jos ei toisin sanota!

1 Esivalmistelut

Lue Patterson & Hennessyn luku 5 “Large and Fast: Exploiting Memory Hierarchy” (erityisesti kohdat 5.1-5.3) sekä MipsIT-simulaattorin käyttöohje. Myös seuraavaan MipsIT-ohjeeseen kannattaa tutustua:

- http://www.eit.lth.se/fileadmin/eit/courses/eit090/MipsIt/MipsITEnvRef.html

Selaa työohje läpi.

Kauanko arvioit työn suoritukseen kuluvan aikaa?_____

Merkitse ylös käyttämäsi tunnit.

2 Välimuistin rakentamisperiaatteet

Välimuisti on muisti, joka on pienempi, mutta nopeampi kuin päämuisti. Muistiviittausten paikallisuudesta johtuen välimuistin käyttö voi tarjota järjestelmälle näennäisesti yhtä nopean muistin kuin välimuisti samalla, kun koko on yhtä suuri kuin päämuistin. Käytännössä välimuisteilla saavutettu suorituskyky vaihtelee riippuen välimuistin koosta, lohkon koosta ja muista parametreista. Tämä riippuu hyvin paljon myös ohjelmasta ja sen datasta. Lyhyesti sanottuna kaikki riippuu oikeiden parametrien valinnasta. Tehtyäsi tämän harjoitustyön sinun pitäisi ymmärtää välimuistien perusperiaatteet ja kuinka

välimuistin eri parametrit vaikuttavat järjestelmän toiminnan tehokkuuteen.

2.1 Välimuistin toiminta

Tutki tiedostossa `matrixsum.c` olevaa koodia. Ohjelmassa on kaksi funktiota, jotka palauttavat kahden matriisin summan. Funktioiden ainoa ero on matriisien indeksointijärjestyksessä. Tämä voi tuntua epäolennaiselta, mutta välimuistin kanssa tällä voi olla valtava ero.

Indeksoidaan matriisia A[3][4] ohjelman funktioiden indeksoimisjärjestyksillä. Merkitse seuraavaan taulukkoon ensimmäiseen sarakkeeseen matriisin alkioiden järjestys muistissa. Merkitse tämän jälkeen kahteen muuhun sarakkeeseen alkioiden

indeksointijärjestys molemmilla tavoilla.

Järjestys muistissa	SumByColRow	SumByRowCol
A[1][1]	1.	1.
A[3][4]	12.	12.

Tee uusi projekti MipsIt IDE:en matrixsum-ohjelmalle, käännä se ja lataa se simulaattoriin. Aja ohjelmaa simulaattorissa oletusasetuksin ja tutki välimuistin toimintaa. Data- ja käskyvälimuistia pystyt tarkastelemaan klikkaamalla kyseisen välimuistin lohkoa

simulaattorissa.

Mitä tapahtuu, kun välimuistissa tulee huti?

Mitä tapahtuu, kun välimuistissa tulee osuma?

Mikä on tagin tarkoitus välimuistissa?

2.2 Välimuistin merkitys

Jotta saadaan datan indeksoinnista johtuvat erot paremmin esiin, poista käskyvälimuisti käytöstä (**Disable**). Poista myös käskyvälimuistin hudista aiheutuva viive käytöstä (**Disable penalty**). Aseta datavälimuistin parametreiksi seuraavat

- koko 64
- lohkon koko 8
- assosiatiivisuus 4

Mitkä ovat ohjelman ilmoittamat suoritusajat?

Paljonko ero on prosenteissa?

Mistä ero johtuu?

---

3 Välimuistin rakenteet ja algoritmit

Tutkitaan seuraavaksi välimuistin eri parametrien vaikutusta suorituskykyyn. Sekä käsky- että datavälimuistilla voidaan säätää välimuistin kokoa (**Size**), lohkon kokoa (**Block size**), assosiatiivisuutta (**Blocks in sets**) ja korvauspolitiikkaa (**Replacement policy**). Datavälimuistilla on lisäksi parametri, jolla asetetaan päämuistin päivitysstrategia (**Write policy**).

Miksi tämä asetus löytyy datavälimuistilta, mutta ei käskyvälimuistilta?

3.1 Välimuistin koko

Tee uusi projekti käyttäen tiedostoa `example0.s`. **Huom! Tässä tehtävässä projektin tyyppi tulee olla muista tehtävistä poiketen Assembler.** Vaihda projektin asetuksista ohjelman entry pointiksi example0. Käännä se ja lataa simulaattoriin. Ota ensin molemmat välimuistit pois käytöstä ja aja ohjelma.
Montako kellojaksoa ohjelman suoritus vie?

Koskematta muihin välimuistin asetuksiin, minkä kokoiset välimuistit valitsisit ohjelmalle example0?
Käskyvälimuisti: sanaa Datavälimuisti: sanaa
Miksi?

Aseta välimuisteille sopivat koot (edellä valitut) ja aja ohjelma uudelleen. Montako kellojaksoa ohjelman suoritus kestää välimuistin kanssa?

3.2 Välimuistin rakenne

Välimuistin toiminnan ja rakenteen ymmärtämiseksi on oleellista, että keskeisten termien merkitys on selvillä. TÄMÄ ON ERITYISEN TÄRKEEÄÄ! SELVITÄ TERMIEN MERKITYS SIIS ITSELLESI KUNNOLLA!

Selitä seuraavien termien merkitys:

- suorasijoitettava (direct mapped)

- joukkoassosiatiivinen (set associative)

- täysassosiatiivinen (fully associative)

- joukko (set) ja joukkojen määrä eri välimuistityypeillä

- lohkon koko (block size) (joskus näkee myös termiä cache line size)

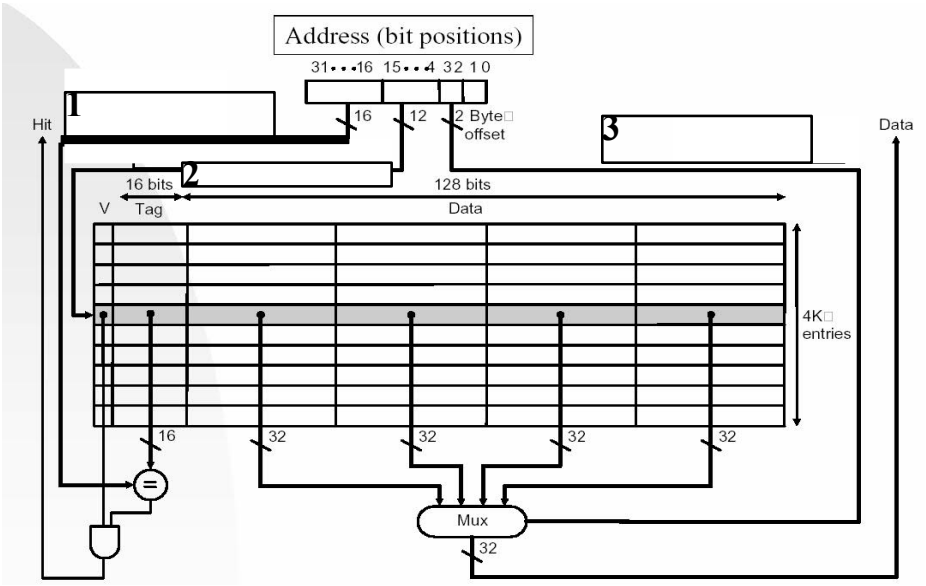
- tag-kenttä

- index-kenttä

- block offset

- byte offset

Sijoita “block offset”, “tag” ja “index” alla olevan kuvan laatikoihin 1, 2 ja 3.



Kuva1. Välimuisti

Välimuistin assosiatiivisuusaste on _____

3.2.1 Suorasijoitettava välimuisti

Koko S=16 sanaa ja lohkon koko B=1 sana. Tavuosoitettavan päämuistin koko M=1024 sanaa. Sananleveys W=32 bittiä. Ilmoita seuraavien kenttien leveydet laskukaavoineen, käyttäen muuttujia S, B, M ja W. Osoitteen leveys on pienin mahdollinen. **Merkitse lisäksi "X" niihin kenttiin, jotka talletetaan välimuistiin.**

☐päämuistille menevän osoitteen minimileveys =_____

☐index =_____

☐block offset =_____

☐byte offset =_____

☐tag =_____

3.2.2 Täysassosiatiivinen välimuisti

Koko S=64 sanaa ja lohkon koko B=2 sanaa. Tavuosoitettavan päämuistin koko M=1024 sanaa. Sananleveys W=32 bittiä. Ilmoita seuraavien kenttien leveydet laskukaavoineen, käyttäen muuttujia S, B, M ja W. Osoitteen leveys on pienin mahdollinen. **Merkitse lisäksi**

"X" niihin kenttiin, jotka talletetaan välimuistiin.

☐päämuistille menevän osoitteen minimileveys =_____

☐index =_____

☐block offset =_____

☐byte offset =_____

☐tag =_____

3.3 Hudit ja osumat

Lataa ohjelma example0 simulaattoriin. Poista datavälimuisti ja sen hudeista aiheutuva sakko käytöstä (**Disable**, **Disable penalty**). Aseta käskyvälimuistien asetuksiksi seuraavat:

- Suorasijoitettava välimuisti
- Välimuistin koko 8 sanaa
- Lohkon koko 1 sana

Ota esille käskyvälimuistin sekä päämuistin sisällöt ja suorita ohjelmaa komennolla Step, kunnes välimuisti on täynnä. Seuraa samalla ohjelman suoritusta käskylistauksen avulla. Kun välimuisti on täynnä, anna komento Step. Mikä lohko (käsky) poistetaan ja miksi?

_____

_____

Mikä käsky noudettiin välimuistiin (ks. ohjelmalistaus)? Mistä osoitteesta?

_____

Jatka ohjelman suoritusta, kunnes koko silmukka on välimuistissa. Tutki tämän jälkeen välimuistin käyttäytymistä ajamalla ohjelmaa askel kerrallaan Step-painikkeella. Ohjelma päivittää Cycle count -arvon välimuistien ikkunoissa, kun kyseiseen välimuistiin viitataan. Cycle countin kasvua tarkkailemalla voidaan havaita viittaukset päämuistiin. Päämuistin saantiaika (**Access time**) voidaan määrittää kellojaksoina muistin asetuksista. Hudin voi havaita myös siitä, että viitatun muistiosoitteen sisältö haetaan välimuistiin toisen muistipaikan tilalle. Mikäli päämuistiin ei tarvitse viitata, yhden käskyn suoritus vie yhden kellojakson. Montako viittausta tuli päämuistiin yhden silmukan kierroksen aikana? Mihin osoitteisiin?

Mitkä käskyt aiheuttivat viittauksia päämuistiin?

Miksi vain näiden osoitteiden suhteen esiintyy huteja?

Mistä johtuvat konfliktihudit (conflict misses)?

Mistä johtuvat kapasiteettihudit (capacity misses)?

Kumpaa tyyppiä esimerkkitapauksen hudit ovat?

3.4 Poistoalgoritmit

Millaisessa välimuistityypissä täytyy käyttää poistoalgoritmeja ja millaisessa ei? Miksi?

Selitä seuraavat poistoalgoritmit lyhyesti

Random

FIFO

LRU

Resetoi CPU, lataa ohjelma uudelleen simulaattoriin, ota datavälimuisti käyttöön (poista myös ruksi kohdasta **Disable penalty**) ja muuta käskyvälimuistin asetukset seuraaviksi

- Täysassosiatiivinen
- Koko 8 sanaa, lohkon koko 1 sana
- Poistoalgoritmi LRU

Montako joukkoa on tämän tehtävän täysassosiatiivisessa muistissa?

Ota molempien välimuistien ja päämuistin sisällöt näkyviin. Suorita ohjelmaa example0, kunnes välimuisti täyttyy. Mikä lohko tullaan poistamaan seuraavan viittauksen aikana? Miksi?

Anna komento Step ja varmista vastauksesi. Mikä käsky välimuistiin luettiin poistetun tilalle?

Jatka kunnes koko silmukka on välimuistissa. Tutki tämän jälkeen välimuistin toimintaa ajamalla ohjelmaa askel kerrallaan. Mikä muistioperaatio aiheuttaa viittauksen päämuistiin?

Kuinka monta lukuhutia esiintyy jokaisella silmukan kierroksella ja miksi? Entä montako kirjoitushutia esiintyy ja miksi?

Suorita ohjelma loppuun komennolla Run. Tutki tuloksia. Mitkä ovat välimuistien osumaprosenttien arvot (Hit Rate)?

Montako kellojaksoa ohjelman suoritus kestää?
