Tampereen yliopisto

Toivo Snåre

RISC-V-SUORITINARKKITEHTUURIN VIRTUALISOINTI

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Toukokuu 2022

TIIVISTELMÄ

Toivo Snåre: RISC-V-suoritinarkkitehtuurin virtualisointi
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Toukokuu 2022

Virtualisointi on yksi modernien tietojärjestelmien kulmakivistä. Sen merkittävimpiä käyttökohteita ovat palvelinympäristöt sekä sulautetut järjestelmät. Esimerkiksi pilvipalveluiden toiminta perustuu fyysisten resurssien jakamiseen useiden käyttäjien kesken, mikä tehdään käytännössä virtualisointia hyödyntäen. Sulautetuissa järjestelmissä virtualisoinnin merkittävin etu on sen tarjoama eristys. Virtualisoinnin avulla useat osajärjestelmät voivat toimia samalla laitteistolla pitäen ne silti eristettyinä toisistaan.

RISC-V on uusi Berkeleyn yliopistossa kehitetty suoritinarkkitehtuuri. Se erottuu muista suoritinarkkitehtuureista erityisesti sen vapaudella ja avoimuudella. Kuka tahansa voi vapaasti suunnitella, valmistaa ja myydä RISC-V-prosessoreita ilman lisenssimaksuja. RISC-V on huomattavasti uudempi suoritinarkkitehtuuri, kuin sen suurimmat kilpailijat, mutta sen suosiolle on ennustettu suurta kasvua tulevaisuudessa.

Vapauden ja avoimuuden lisäksi RISC-V:n valtteja ovat sen tekniset ominaisuudet, kuten laaja virtualisoitavuus, joka on ollut yksi arkkitehtuurin suunnittelutavoitteista alusta alkaen. Tässä tutkielmassa selvitetään, miten virtualisointi on otettu huomioon RISC-V-suoritinarkkitehtuurin suunnittelussa. Ensimmäisenä tutkielman kirjallisuustutkimusosassa selvitetään, miten virtualisointi toimii käytännössä. Kirjallisuuslähteistä löydettiin yhtenäinen joukko tunnettuja virtualisointitekniikoita, joiden käyttökelpoisuus tutkitaan seuraavaksi RISC-V-suoritinarkkitehtuurilla.

Tutkielman tulokset osoittavat, että virtualisointimahdollisuudet on otettu RISC-V-suoritinarkkitehtuurin suunnittelussa perusteellisesti huomioon. Arkkitehtuurista on tunnistettavissa monia suunnittelupäätöksiä, jotka on tehty virtualisointia silmällä pitäen. Seurauksena RISC-V soveltuu kaikkien tunnetuimpien virtualisointitekniikoiden käyttämiseen. Johtopäätöksinä todetaan, että RISC-V:n ennustetun suosion kasvun syyt ovat selkeästi nähtävissä. Virtualisoitavuus yhdistettynä vapauden, avoimuuden ja muiden teknisten ominaisuuksien kanssa tekevät RISC-V:stä hyvin soveltuvan suoritinarkkitehtuurin moniin käyttökohteisiin.

Avainsanat: RISC-V, virtualisointi, suoritinarkkitehtuuri, käskykanta, sulautetut järjestelmät

SISÄLLYSLUETTELO

1. Johdanto 1
2. RISC-V 3
2.1 Menestyksen avaimet 4
2.2 Etuoikeutettu arkkitehtuuri 6
2.3 Control- ja status-rekisterit 7
2.4 Keskeytykset, poikkeukset ja ansat 7
2.5 Muistinsuojaus sivutuksella 9
2.6 Muistinsuojaus PMP:llä 10
3. Virtualisointi 11
3.1 Järjestelmävirtualisointi 11
3.2 Klassinen virtualisointi 14
3.3 Dynaaminen binääritranslaatio 15
3.4 Laitteistoavusteinen virtualisointi 16
3.5 Paravirtualisointi 18
3.6 Käyttöjärjestelmätason virtualisointi 19
4. Tulokset 21
4.1 Klassinen virtualisointi 21
4.2 Dynaaminen binääritranslaatio 23
4.3 Laitteistoavusteinen virtualisointi 23
4.4 Paravirtualisointi 24
4.5 Käyttöjärjestelmätason virtualisointi 25
5. Yhteenveto 26
Lähteet 27

1. JOHDANTO

Virtualisointi on yksi modernien tietojärjestelmien kulmakivistä. Yhä useampi tietojärjestelmä toteutetaan nykyään hyödyntämällä pilvipalveluita, joiden toiminta perustuu virtualisointiin [17, 18]. Palvelinympäristöjen lisäksi virtualisoinnin toiseksi merkittäväksi käyttökohteeksi on viime vuosikymmenen aikana muodostunut sulautetut järjestelmät [11, 15].

Yksi pilvipalvelujen käytön merkittävimmistä eduista omien konesalien ylläpitoon verrattuna on niiden ylläpitokustannukset. Perinteisessä palvelinympäristössä laitteistoresurssit joudutaan yleensä mitoittamaan teoreettisen maksimikuorman perusteella. Pilvipalveluissa sen sijaan laskentatehoa ja tallennuskapasiteettia voidaan jakaa virtualisoinnin avulla dynaamisesti isoista datakeskuksista muodostuvista resurssipooleista. Resurssien jakaminen usean käyttäjän kesken parantaa laitteiston hyötysuhdetta, millä säästetään sähköä ja muita kustannuksia. [18]

Virtualisoitujen palvelimien etuja ovat myös niiden suoraviivainen käyttöönotto ja skaalautuvuus. Palvelimen tarvitsemat virtuaaliresurssit voidaan usein määritellä ja pystyttää muutamassa minuutissa. Resurssien määrää voidaan skaalata myöhemmin uudelleen esimerkiksi automaattisesti palvelimen kuorman perusteella [17]. Lisäksi koko virtuaalinen ympäristö voidaan siirtää reaaliajassa kokonaan toiseen fyysiseen sijaintiin, jos esimerkiksi palvelimen fyysistä laitteistoa tarvitsee huoltaa [17]. Migraatio pystytään tekemään ilman palvelun käyttökatkoa [7, 17].

Sulautetuissa järjestelmissä virtualisoinnin merkittävin etu on sen tarjoama eristys. Pinta-alan, virran ja kustannusten minimoimiseksi samalle sulautetulla laitteistolle pakataan nykyään useita osajärjestelmiä, joilla voi olla eri kriittisyystasoja. Osajärjestelmien eristäminen toisistaan on erityisen tärkeää tilanteissa, jossa turvallisuuskriittisten osajärjestelmien pitää toimia rinnan ei-kriittisen osajärjestelmien kanssa. [15]

RISC-V-suoritinarkkitehtuuri on toinen viime vuosikymmenen aikana syntynyt kehitys sulautettujen järjestelmien tieteenalalla. RISC-V on uusi Berkeleyn yliopiston tutkimusprojektissa syntynyt suoritinarkkitehtuuri, joka erottuu muista suoritinarkkitehtuureista avoimuudellaan ja vapaudellaan. Kuka tahansa voi vapaasti suunnitella, valmistaa ja myydä RISC-V-prosessoreita ilman lisenssimaksuja [21]. RISC-V:n on ennustettu nousevan merkittävään asemaan tulevaisuudessa [8].

Virtualisointimahdollisuudet on otettu alusta alkaen huomioon RISC-V:n suunnittelussa [20]. RISC-V ja sen virtualisointi on kuitenkin suhteellisen uusi aihe, joten siitä on saatavilla suhteellisen vähän tietoa. Tässä tutkielmassa on tarkoitus selvittää, mitkä ovat ne suunnittelupäätökset, jotka tekevät RISC-V-prosessoreista helposti virtualisoitavia. Tarkastelu tehdään käytännössä tutkimalla usean eri virtualisointitekniikan käyttökelpoisuus RISC-V:llä. Tutkimalla konkreettisia virtualisointitekniikoita saadaan hyvä käsitys, miten RISC-V sopii virtualisointiin käytännössä. Johtopäätöksenä todetaan, että RISC-V:n laajat virtualisointiominaisuudet voivat tehostaa sen käyttöönottoa massamarkkinoilla.

Tutkielma toteutettiin käytännössä kirjallisuuskatsauksena. Lähteiden valinnassa on suosittu tunnettuja vertaisarvioituja lehtiä ja konferenssijulkaisuja. Huomioitaviin kriteereihin on kuulunut myös lähteiden julkaisuajankohdat. Lähteinä on pyritty käyttämään vertaisarvioituja artikkeleita, jotka kuvaavat alan tuoreinta tietoa. Toisaalta lähteinä on käytetty myös tutkielman aihepiirien historiallisesti merkittävämpiä artikkeleita, joihin myöhempi tutkimus perustuu. Perusluontoiseen tietoon parhaita lähteitä ovat olleet tutkielman aihepiirejä käsittelevät oppikirjat. RISC-V-suoritinarkkitehtuurin perustava lähde on sen tekninen määrittely.

Seuraavaksi luvussa 2 käydään läpi RISC-V-suoritinarkkitehtuurin pääpiirteet ja tarkastellaan tarkemmin virtualisoinnin kannalta merkittävät yksityiskohdat. Luvussa 3 käydään läpi virtualisointiin liittyvä teoria ja tunnetut virtualisointitekniikat. Luvussa 4 selvitetään edellisen luvun virtualisointitekniikoiden käyttökelpoisuus RISC-V:llä. Luvussa 5 tiivistetään työn merkittävimmät tulokset ja pohditaan niiden laajempaa merkitystä.

2. RISC-V

RISC-V:n historia on perusteltua aloittaa vuodesta 1980, jolloin Berkeleyn yliopiston professori David A. Patterson ja hänen oppilaansa David R. Ditzel julkaisivat artikkelin "The case for the reduced instruction set computer"[10]. Ennen artikkelin julkaisua RISC-tyylisiä suoritinarkkitehtuureja oli tutkittu hetki Berkeleyn yliopiston lisäksi myös Standfordin yliopistossa sekä IBM:llä, mutta artikkelia pidetään RISC- ja CISC-käsitteiden (engl. Reduced Instruction Set Computer vs. Complex Instruction Set Computer) syntyhetkenä [4, 9, 10, 20]. Suurin osa artikkelin julkaisun jälkeen yleistyneistä suoritinarkkitehtuureista on RISC-tyylisiä [9].

Kaikki RISC-arkkitehtuurit eivät ole identtisiä, mutta ne täyttävät yleensä tietyt tekniset tuntomerkit. Keskeisimpänä piirteenä RISC-käskykanta koostuu yleensä pienistä atomisista käskyistä [4]. Monimutkaiset operaatiot tehdään siis usealla pienellä käskyllä. Vastakohtana esimerkiksi CISC-tyylisessä x86-käskykannassa on oma käsky (`repne scasb`) merkkijonon pituuden laskemiseen [5]. RISC-tyyliset käskyt voidaan yleensä suorittaa yhdessä kellojaksossa, mikä tekee prosessorin liukuhihnoituksesta helpompaa monimutkaisiin CISC-tyylisin käskyihin verrattuna [4, 9]. Lisäksi käskyt on koodattu yleensä vakiomittaiseen kehykseen, mikä helpottaa käskynpurkua [4].

Muistinkäsittelyyn on RISC-prosessorissa tyypillisesti vain kaksi käskyä, joista toinen on rekisteristä muistiin kirjoittamiseen (store-käsky) ja toinen muistista rekisteriin lukemiseen (load-käsky) [4]. Muut käskyt eivät siis lue parametrejä suoraan muistista, kuten CISC-tyylinen käsky voisi tehdä, vaan operaatiot tehdään rekistereihin load-käskyillä ladatuilla arvoilla. Viimeisenä tuntomerkkinä RISC-arkkitehtuureissa on yleensä suhteellisen suuri määrä yleiskäyttöisiä rekistereitä. Rekistereitä ei ole siis tyypillisesti varattu yhteen käyttötarkoitukseen, kuten pinon käsittelyyn, vaan käskyjen lähde- ja kohderekistereinä voi toimia mikä tahansa yleiskäyttöisistä rekistereistä.

Artikkelissaan Patterson ja Ditzel argumentoivat RISC-suoritinarkkitehtuurien etujen puolesta, jotka perustuvat pitkälti niiden yksinkertaisuuteen. Yksinkertainen RISC-arkkitehtuuri on halvempi, nopeampi ja helpompi toteuttaa rajallisen kokoiselle sirulle kuin monipuolinen mutta myös monimutkainen CISC-arkkitehtuuri. Yksinkertaisesta rakenteesta jää myös enemmän tilaa esimerkiksi liukuhihnoitukselle tai välimuisteille, jotka parantavat prosessorin suorituskykyä. [10]

He myös argumentoivat, että syyt artikkelin julkaisun aikaisten suoritinarkkitehtuurien kompleksisuuden kasvulle eivät ole perusteltuja. Esimerkiksi tiettyjen korkean tason ohjelmointikielten rakenteiden tukemiseen erikoistuneiden käskyjen lisääminen voi ideana olla hyvä, mutta artikkelissa viitatut tutkimustulokset osoittavat, että prosessorin suorituskyky on yleensä yhtä hyvä tai jopa parempi, jos monimutkaiset käskyt jaetaan useampaan atomiseen käskyyn. Lisäksi suuri määrä monimutkaisia käskyjä tekee korkean tason ohjelmointikielen kääntäjän toteuttamisesta vaikeaa. CISC-arkkitehtuurien kääntäjien tuottamassa koodissa käytetään usein vain pientä murto-osaa koko käskykannasta [4, 9]. [10]

Artikkelin julkaisun jälkeisten vuosikymmenten aikana RISC-tutkimustyö jatkui sekä akateemisessa maailmassa että kaupallisella alalla. Esimerkiksi erityisesti mobiilimarkkinoita johtavan [11] ARM-suoritinarkkitehtuurin ensimmäinen versio kehitettiin 80-luvulla. Pattersonin ja Berkeleyn yliopiston vaikutus ARM:iin on nähtävissä alkaen heti arkkitehtuurin nimestä, sillä ARM-lyhenteen R-kirjain viittaa nimenomaan RISC:iin. ARM-suorittimia on nykyään laajasti käytössä muun muassa älypuhelimissa ja sulautetuissa järjestelmissä.

Berkeleyn yliopistossa jatkokehitettiin myöhemmin SPARC-suoritinarkkitehtuuri, joka tutkimuskäytön lisäksi menestyi hyvin kaupallisella alalla. Yliopiston tutkimusprojekteissa käytettiin myös Standfordin yliopistossa kehitettyä MIPS-arkkitehtuuria. Vuonna 2010 Pattersonin johtama tutkijaryhmä halusi valita yhden suoritinarkkitehtuurin seuraavien tutkimusprojektien alustaksi. He löysivät kuitenkin kaikista olemassa olevista vaihtoehdoista puutteita. Esimerkiksi ARM-arkkitehtuuria ei voitu valita tutkimuskäyttöön sen suljetun lisenssimallin vuoksi, joka tekee esimerkiksi omien käskyjen lisäämisestä erittäin vaikeaa. Käskykannan modifiointiin liittyy merkittäviä lisenssimaksuja. [20]

Järkevimmäksi ratkaisuksi jäi kokonaan uuden suoritinarkkitehtuurin kehittäminen. Uusi suoritinarkkitehtuuri oli järjestyksessä viides Berkeleyn yliopiston RISC-toteutus, joten sen nimeksi muodostui RISC-V [10, 20, 21]. Projektin oli tarkoitus valmistua nopeasti yliopiston omia tutkimusprojekteja varten, mutta todellisuudessa se herätti alusta alkaen paljon mielenkiintoa myös yliopiston ulkopuolella [20]. RISC-V:n menestys perustuu pitkälti arkkitehtuurin järkeviin suunnittelupäätöksiin, jotka on voitu tehdä 30 vuoden ajalta kerätyn tiedon pohjalta [9, 20].

2.1 Menestyksen avaimet

RISC-V erottuu muiden suoritinarkkitehtuurien joukosta erityisesti sen avoimuudella. RISC-V on avoin standardi, joka käytännössä koostuu kaksiosaisesta käskykantamäärittelystä [21, 22], joka on saatavilla RISC-V:n kehitystä ohjaavan voittoa tavoittelemattoman RISC-V International -yhdistyksen www-sivulta [14]. Kuka tahansa voi vapaasti suunnitella, valmistaa ja myydä käskykantamäärittelyn mukaisia prosessoreita ilman lisenssimaksuja. Avoimuuden seurauksena RISC-V-prosessoreita on runsaasti, joista suuri osa

on vapaasti käytettävissä olevia avoimen lähdekoodin toteutuksia, mikä vähentää uusien järjestelmien kehityskuluja [20].

Käskykantamäärittely sanelee ainoastaan ohjelmiston ja laitteiston välisen rajapinnan, mutta jättää sen toteutustavan avoimeksi. Yksi arkkitehtuurin suunnittelutavoitteista on tukea monenlaisia käyttökohteita ja toteutuksia pienimmistä mikrokontrollereista laajasti rinnakkaisiin moniydinprosessoreihin [21], minkä RISC-V mahdollistaa modulaarisella rakenteellaan. RISC-V käskykanta muodostuu minimaalisesta perusosasta (engl. base integer ISA) ja valinnaisista laajennoksista (engl. extensions) [21].

Käskykannan perusosa määrittelee minimaalisen joukon käskyjä, jotka ovat välttämättömiä kaikille toteutuksille, kuten muistinkäsittelykäskyt, aritmeettis-loogiset käskyt sekä haarautumiskäskyt [21]. Virallisia perusosia on tutkielman kirjoitushetkellä määritelty neljä kappaletta [21]. Ne ovat käytettävien käskyjen osalta lähes identtisiä, mutta eroavat sananpituuksiltaan. Sananpituus määrittää muun muassa osoiteavaruuden koon ja rekisterien leveyden. Esimerkiksi RV32I-perusosassa sananpituus 32 bittiä, kun taas RV64I- ja RV128I-perusosissa sananpituudet ovat 64 ja 128 bittiä. RV32E-perusosa eroaa kolmesta muusta enemmän, sillä se on karsittu versio RV32I-perusosasta pienille mikrokontrollereille. Siinä yleiskäyttöisiä rekistereitä on vain 16 kappaletta, kun muissa niitä on 32 kappaletta.

Virallisia RISC-V laajennoksia on muun muassa kerto- ja jakokäskyille (M-laajennos), IEEE-754-liukuluvuille (F- ja D-laajennokset) sekä atomisille operaatioille (A-laajennos). Laajennoksilla voi olla riippuvuussuhteita. Esimerkiksi laitteistoavusteiseen virtualisointiin suunniteltu H-laajennos edellyttää etuoikeutetun tilan ja käyttäjätilan toteuttavat S- ja U-laajennokset. [21]

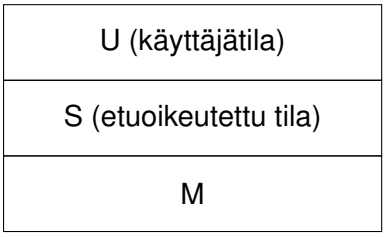
Laajennosten avulla käskykanta voidaan muokata erityisiin ja vaativiin käyttökohteisiin, mutta jo pelkästä perusosasta muodostuva käskykanta sopii hyvin muun muassa yksinkertaisiin sulautettuihin järjestelmiin. Peruosan mutkattomuus ja avoimuus tekevät siitä myös hyvin opetuskäyttöön soveltuvan. Lisäksi akateemisen maailman ja yritysmaailman välinen välimatka lyhenee, jos molemmissa käytetään samoja standardeja, mikä voi tehostaa RISC-V:n käyttöönottoa massamarkkinoilla. [20]

Teknisestä näkökulmasta katsottuna RISC-V perii kaikki alkuperäisen RISC:in edut. Esimerkiksi käskyndekoodauslogiikan toteuttaminen on mutkatonta, koska kaikki käskyt koodataan vakiomittaiseen neljän tavun kehykseen [21]. Käskynpurkua on helpotettu arkkitehtuurin suunnittelussa entisestään pitämällä tietyt kentät samassa kohdassa eri käskyjen välillä [21]. Esimerkiksi osasta käskyistä löytyvä kohderekisteri on kaikissa käskykehyksissä koodattu samaan kohtaan. Toisena esimerkkinä kaikkien käskykehysten välittömien lukujen merkkibitti on aina eniten merkitsevän bitin paikalla, mikä tehostaa prosessorin sign extension -logiikkaa [21]. Seuraavaksi käsitellään RISC-V:n muut tekniset yksityiskohdat, jotka ovat virtualisoinnin kannalta merkittäviä.

2.2 Etuoikeutettu arkkitehtuuri

RISC-V:n yksi alkuperäisistä käyttökohteista Berkeleyn yliopistossa oli erilaisten etuoikeutettujen arkkitehtuurien tutkiminen, joten käskykannan etuoikeutettu osa on suunniteltu perusosasta täysin erillisenä laajennoksena [20]. Virallinen etuoikeutettu arkkitehtuuri on suunniteltu olemassa olevien käyttöjärjestelmien, kuten Unixin kaltaisten käyttöjärjestelmien ajamiseen [20]. Se on kuitenkin helposti korvattavissa toisella toteutuksella koskematta käskykannan perusosaan [20]. Monista kilpailijoistaan poiketen RISC-V-prosessori voidaan suunnitella myös ilman minkäänlaista etuoikeutettua osaa [20], millä voidaan säästää tilaa esimerkiksi yksinkertaisissa sulautetuissa järjestelmissä.

Käynnistyessään RISC-V-prosessorin jokainen ydin (engl. "hart") toimii M-suoritustilassa (engl. machine mode), jossa kaikkien käskyjen suorittaminen on mahdollista [22]. M-tilaa käytetään usein esimerkiksi käynnistyksenlataajan tai BIOS-tyylisten apurutiinien suorittamiseen [22]. M-tila voi olla esimerkiksi yksinkertaisten sulautettujen järjestelmien tapauksessa ainoa suoritustila [22], mutta monissa RISC-V-prosessoreissa on myös U-laajennoksen mukainen U-suoritustila, eli käyttäjätila (engl. user mode). Käyttäjätila on tarkoitettu käyttäjäsovellusten ajamiseen rajatuilla oikeuksilla, mikä voidaan käytännössä tehdä esimerkiksi PMP-mekanismilla, joka esitellään lyhyesti luvussa 2.6. Oikeustason suhteen käyttäjätilan ja M-tilan välissä sijaitsee vielä S-laajennoksen toteuttama S-tila, eli etuoikeutettu tila (engl. supervisor mode). Etuoikeutetussa tilassa toimivat yleensä esimerkiksi käyttöjärjestelmien ytimet [22]. Sen kanssa voidaan hyödyntää sivutusta, mikä on usein vaatimus modernien Unixin kaltaisten käyttöjärjestelmien ajamiseen. Sivutusmekanismi esitellään tarkemmin luvussa 2.5. Suoritustilojen järjestys on esitetty kuvassa 2.1.



Kuva 2.1. *RISC-V-prosessorin suoritustilat. [22]*

S-laajennos lisää etuoikeutetun tilan lisäksi muutamia etuoikeutettuja käskyjä sekä control- ja status-rekistereitä. Korkeamman oikeuden suoritustilat voivat käyttää matalamman oikeuden suoritustilan käskyjä ja rekistereitä, mutta ei toisinpäin. Esimerkiksi sivutusmekanismin välimuistin tyhjennykseen käytetyn `sfence.vma`-käskyn voi suorittaa vain M- ja S-tiloissa. U-tilassa käskyn suoritus tuottaa poikkeuksen. Myös M-tilalla on omia käskyjä ja rekistereitä, joita ei voi käyttää muissa suoritustiloissa. [22]

2.3 Control- ja status-rekisterit

Yleiskäyttöisten rekisterien lisäksi RISC-V-arkkitehtuurissa on prosessorin tilaa ohjaavia ja kuvaavia control- ja status-rekistereitä (engl. CSRs). Esimerkiksi mstatus-rekisterillä voidaan muun muassa kytkeä keskeytykset globaalisti pois päältä tai vaihtaa prosessorin tavujärjestystä [22]. Control ja status-rekisterien lukemiseen ja kirjoittamiseen on omat käskyt, jotka ovat osa standardia Zicsr-laajennosta [21].

Kaikkien control- ja status-rekisterien käyttäminen ei ole mahdollista kaikilta suojaustasoilta. Rekisterien nimien ensimmäinen kirjain kertoo yleensä, mikä on alin suojaustaso, mistä niitä voidaan käyttää. Esimerkiksi sstatus-rekisterissä on vain osa mstatus-rekisterin kentistä, mutta sen käyttäminen onnistuu M-tilan lisäksi myös S-tilassa [22]. Alemmilta suojaustasoilta rekisterien käyttöyritys tuottaa poikkeuksen [21].

2.4 Keskeytykset, poikkeukset ja ansat

Erityisesti keskeytys-sanan täsmällinen merkitys voi vaihdella kontekstista riippuen [11]. RISC-V-spesifikaatiossa annetaan kuitenkin tarkka määritelmä keskeytyksille (engl. interrupts) sekä poikkeuksille (engl. exceptions) ja ansoille (engl. traps). Tämän tutkielman tekstissä käytetään RISC-V-spesifikaation terminologiaa.

Poikkeuksella tarkoitetaan prosessorin mekanismia reagoida ristiriitaisen käskyn suoritukseen. Poikkeus seuraa esimerkiksi laittoman käskyn suorituksesta, virtuaalimuistiavaruuden ulkopuolella olevan muistiosoitteen lukemisesta tai käskyn suorituksesta ilman tarvittavia oikeuksia [21]. Poikkeukset ovat siis käskyjen suorituksen suhteen synkronisia, mikä erottaa ne keskeytyksistä [21]. Keskeytykset ovat käskyjen suhteen asynkronisia, eli keskeytys voi tapahtua myös käskyjen suorituksen välissä [21]. Keskeytyksen lähde on usein prosessin ulkopuolinen oheislaite. Esimerkiksi mikrokontrollerissa voi olla sisääntulona jalka, jossa jännitteen positiivinen reuna tuottaa keskeytyksen.

Sekä poikkeukset että keskeytykset ovat ansojen osajoukkoja [21]. Ansan seurauksena RISC-V-laitteisto kirjoittaa automaattisesti ansan käsittelyn kannalta merkittäviä tietoja tiettyihin status-rekistereihin. Ansan syy kirjoitetaan mcause-rekisteriin taulukon 2.1 mukaisella koodauksella [22]. Rekisteriin kirjoitettavan arvon eniten merkitsevä bitti merkitsee, onko kyseessä poikkeus vai keskeytys. Monista ansoista kirjoitetaan myös lisätietoa mtval-rekisteriin [22]. Esimerkiksi sivutuspoikkeuksen yhteydessä sinne kirjoitetaan poikkeuksen tuottanut virtuaalinen muistiosoite. Ennen ansan käsittelyrutiiniin hyppäämistä ohjelmalaskurin arvo kirjoitetaan mepc-rekisteriin ja prosessorin suoritustila kirjoitetaan mstatus-rekisterin PPM-kenttään, jotta prosessorin tila voidaan palauttaa ansan käsittelyn jälkeen [22].

Kun ansan tiedot on kirjoitettu status-rekistereihin, prosessori hyppää M-tilassa mtvec-

Keskeytys	Ansakoodi	Ansan syy
1	1	S-tilan ohjelmistokeskeytys
1	3	M-tilan ohjelmistokeskeytys
1	5	S-tilan ajastinkeskeytys
1	7	M-tilan ajastinkeskeytys
1	9	S-tilan ulkoinen keskeytys
1	11	M-tilan ulkoinen keskeytys
0	0	Laiton käskyosoite
0	2	Laiton käsky
0	3	Pysäytyspiste (engl. breakpoint)
0	8	Järjestelmäkutsu U-tilasta
0	9	Järjestelmäkutsu S-tilasta
0	11	Järjestelmäkutsu M-tilasta
0	12	Käskysivutuspoikkeus (engl. instruction page fault)
0	13	Lukusivutuspoikkeus (engl. load page fault)

Taulukko 2.1. *Esimerkkejä mcause- ja scause-rekisterien arvojen merkityksistä. Rekisterin eniten merkitsevä bitti on esitetty ensimmäisessä sarakkeessa ja muista biteistä muodostuva arvo toisessa sarakkeessa. [22]*

rekisterin osoittamaan ansankäsittelyrutiiniin [22]. Tyypillisesti ensimmäiseksi ansankäsittelyrutiini puskee käyttämiensä rekisterien arvot pinoon, jotta se voi palauttaa niiden arvot takaisin ennen ansasta palaamista. Seuraavaksi suoritus ohjautuu ansan syyn mukaiseen koodiin. Esimerkiksi käyttäjätilassa ajettava ohjelma tekee tyypillisesti käyttöjärjestelmälle järjestelmäkutsun kirjoittamalla kutsun numeron ja parametrit ennalta sovittuihin rekistereihin ja suorittamalla `ecall`-käskyn, joka tuottaa järjestelmäkutsupoikkeuksen. Ansankäsittelyrutiini lukee mcause-rekisterin sekä järjestelmäkutsun numeron sisältävän rekisterin arvon ja suorittaa sen perusteella oikean järjestelmäkutsun. Ansasta palataan `mret`-käskyllä, joka palauttaa automaattisesti edellisen ohjelmalaskurin arvon ja suoritustilan mepc- ja mstatus-rekistereistä [22].

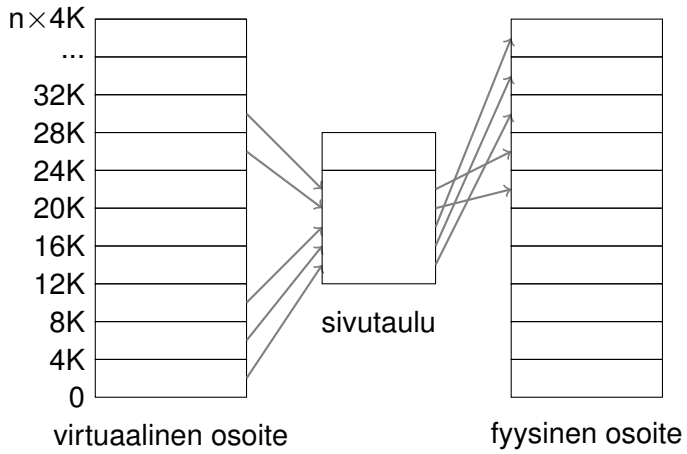
Normaalisti prosessorin suoritustila vaihtuu M-tilaan ansan seurauksena, mutta RISC-V:stä löytyy myös mahdollisuus käsitellä valittuja ansoja S-tilassa. S-tilaan delegoitavat poikkeukset valitaan kirjoittamalla medeleg-rekisteriin [22]. Keskeytyksille on vastaava mideleg-rekisteri [22]. S-tilassa käsitettävien ansojen tiedoille on omat rekisterit, joita voidaan käyttää myös S-tilassa [22]. Esimerkiksi ansan syyn kirjoitetaan mcause-rekisterin sijaan scause-rekisteriin ja ansavektori löytyy stvec-rekisteristä. S-tilassa käsiteltävästä ansasta palataan `sret`-käskyllä [22].

2.5 Muistinsuojaus sivutuksella

Sivutus on RISC-V:n ensisijainen mekanismi hienojakoiseen muistinsuojaukseen. S-laajennokseen kuuluva sivutusmekanismi on samankaltainen muun muassa ARM- ja x86-suoritinarkkitehtuurien kanssa, mikä helpottaa olemassa olevien käyttöjärjestelmien siirtämistä RISC-V:lle.

Sivutuksen ollessa päällä käyttäjätilassa ajettavat ohjelmat käyttävät virtuaalimuistiosoitteita, jotka laitteisto muuttaa automaattisesti fyysisiksi muistiosoitteeksi määritetyn kuvauksen perusteella. Kuvauksen määrittää etuoikeutetussa tilassa ajettava ohjelmisto, kuten käyttöjärjestelmän ydin. Kuvaus määritetään sivutauluilla, jotka ydin kirjoittaa keskusmuistiin. Virtuaalisten osoitteiden translaatioprosessi fyysisiksi osoitteiksi alkaa aina juurisivutaulusta, jonka ydin määrittää kirjoittamalla sen fyysisen osoitteen satp-rekisteriin [22].

Kuvaus voidaan tehdä neljän kilotavun kokoisien sivujen (engl. pages) tarkkuudella. Virtuaalisen neljän kilotavun osoiteavaruuden sisällä on siis aina lineaarinen kuvaus tiettyyn loogiseen neljän tavun osoiteavaruuteen. Kokonaiset virtuaaliset sivut voivat kuitenkin määritetyn kuvauksen mukaan sijaita fyysisessä keskusmuistissa missä tahansa järjestyksessä. Jokaiselle sivulle voidaan määritellä erikseen luku-, kirjoitus- ja suoritusoikeuksista muodostuvat suojaukset [22]. Jos esimerkiksi käyttäjätilassa ajettava prosessi yrittää kirjoittaa sivulle, jolla ei ole kirjoitusoikeutta, niin laitteisto tuottaa poikkeuksen [22].



Kuva 2.2. *Sivutaululla määritetty kuvaus virtuaalisista osoitteista fyysisiin osoitteisiin.*

Prosessit saadaan eristettyä ja suojattua toisiltaan käyttämällä eri kuvausta niiden suorituksen aikana. Kontekstin vaihdon aikana käyttöjärjestelmän ydin muuttaa kuvauksen kirjoittamalla satp-rekisteriin toisen prosessin sivutaulun osoitteen. Sivutuksen avulla prosessit voivat käyttää samoja virtuaalisia muistiosoitteita, jotka kuitenkin kuvautuvat eri kohtiin fyysisessä keskusmuistissa. Prosesseille luodaan siis illuusio, että niillä olisi koko keskusmuisti käytettävissä itselleen. Sivutuksen toimintaperiaate on esitetty kuvassa 2.2.

2.6 Muistinsuojaus PMP:llä

RISC-V:n Physical Memory Protection (PMP) on sivutusta yksinkertaisempi ja rajallisempi mekanismi muistinsuojaukseen. Sillä voidaan estää tai sallia valittujen fyysisen muistin alueiden käyttö tietyiltä suojaustasoilta. Käytännössä PMP:llä voidaan antaa oikeuksia S- ja U-tiloille, joilla ei ole oletuksella oikeutta mihinkään osaan fyysisestä muistista, tai poistaa M-tilalta oikeuksia, jolla on oletuksella oikeus käyttää koko fyysistä muistia [22].

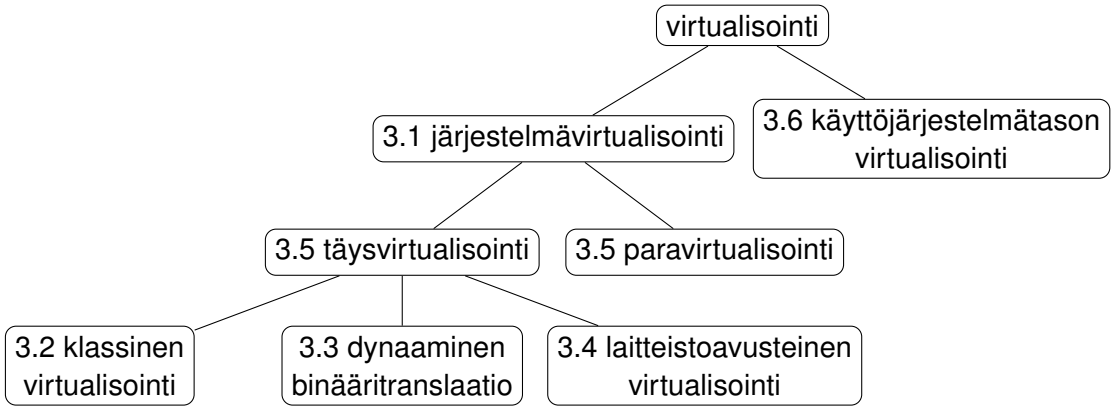
Oikeuksia voidaan valita yksittäisille PMP-alueille, joita voi olla prosessorin toteutuksesta riippuen 0-64 kappaletta [22]. Jokaiselle alueelle on kaksi control-rekisteriä [22]. Ensimmäiseen kirjoitetaan muistiosoite ja toiseen konfiguraatio, joka merkitsee miten muistiosoite tulisi tulkita. Konfiguraatio-rekisterin avulla voidaan esimerkiksi valita, että muistiosoiterekisteriin kirjoitetusta osoitteesta alkaa PMP-alue, jonka pituus on 4 kilotavua.

Konfiguraatio-rekisteristä löytyy lisäksi alueen suojaukset, jotka koodataan sivutusta vastaavalla tavalla [22]. Jokaiselle alueelle on mahdollista sallia erikseen lukeminen (R-bitti), kirjoittaminen (W-bitti) tai suorittaminen (X-bitti). Jos jokin biteistä on nolla, sitä vastaava operaatio on estetty. Esimerkiksi käskynhaku PMP-alueelta, jolla ei ole X-bittiä, tuottaa poikkeuksen.

PMP-alue voidaan lukita konfiguraatio-rekisterin L-kentällä siten, että sen konfiguraatiota ei pysty enää muuttamaan edes M-tilasta. Lukitun PMP-alueen suojaukset ovat voimassa S- ja U-tilojen lisäksi myös M-tilassa. Lukitut alueet pysyvät lukittuina prosessorin seuraavaan uudelleenkäynnistykseen asti. [22]

3. VIRTUALISOINTI

Virtualisointi on laaja käsite, joka voi nykyään tarkoittaa montaa eri asiaa [11]. Yhteistä virtualisoinnin kaikille osa-alueille on kuitenkin se, että jotain oikeaa tai fyysistä korvataan virtuaalisella versiolla. Tässä tutkielmassa käsiteltävät virtualisoinnin osa-alueet ja niiden väliset suhteet on esitetty kuvassa 3.1.



Kuva 3.1. *Virtualisointitekniikat ja niitä käsittelevien lukujen numerot.*

Käytännössä virtualisointitekniikoita voi olla vaikeaa luokitella, koska niitä yhdistellään usein keskenään. Esimerkiksi hyvin yleisesti muuten täysvirtualisoidussa ympäristöissä käytetään paravirtualisoituja laiteajureita suorituskyvyn kasvattamiseksi [6, 18]. Tekniikoiden käytännön toteutukset voivat myös vaihdella huomattavasti. Seuraavissa luvuissa käydään läpi parhaiten tunnettujen virtualisointitekniikoiden pääpiirteet.

3.1 Järjestelmävirtualisointi

Kun virtualisoinnista puhutaan yleisesti, tarkoitetaan yleensä järjestelmävirtualisointia. Järjestelmävirtualisointi tarkoittaa usean käyttöjärjestelmän ajamista samalla laitteistolla tarjoamalla niille virtuaalinen illuusio laitteistosta, joka näyttää vastaavalta kuin oikea laitteisto [11, 17, 18]. Virtuaalista laitteistoa kutsutaan usein virtuaalikoneeksi (engl. virtual machine, VM) [11, 15, 17, 18].

Tyypillisesti käyttöjärjestelmän ydin toimii prosessorin etuoikeutetussa tilassa, jossa se pystyy hallita koko laitteistoa. Virtualisoidulle käyttöjärjestelmälle ei kuitenkaan voida antaa oikeutta hallita koko laitteistoa, jotta se voi toimia rinnakkain muiden virtuaalikoneiden

kanssa, joten järjestelmävirtualisoitua käyttöjärjestelmää ei ajeta etuoikeutetussa tilassa. Matalamman etuoikeuden tilassa sille voidaan rajoittaa pääsy vain tiettyihin virtuaaliresursseihin, jotka on sille varta vasten osoitettu. Ohjelmistoa, joka vastaa virtuaaliresurssien jakamisesta ja yleisesti virtuaalikoneiden ohjaamisesta, kutsutaan virtualisointiytimeksi (engl. hypervisor, virtual machine monitor, VMM) [11, 15, 17, 18].

Laitteiston merkittävimmät osat ovat prosessori ja keskusmuisti. Virtualisointiytimen pitää jakaa prosessori virtuaalikoneiden kesken siten, että kaikki niistä saavat osan saatavilla olevasta suoritusajasta. Prosessorin sisäinen tila, kuten lippujen ja muiden rekisterien arvot, pitää myös virtualisoida siten, että kaikilla virtuaalikoneilla on illuusio omasta prosessorista. Kun virtualisointiydin tekee kontekstinvaihdon virtuaalikoneesta toiseen, se tallentaa edellisen virtuaalikoneen prosessorin tilan talteen ja hakee uuden virtuaalikoneen prosessorin tilan sen paikalle. Prosessorin virtualisointi on verrattavissa käyttöjärjestelmän ytimen prosessiabstraktioon. Virtualisointiydin tekee kontekstinvaihtoja virtuaalikoneiden välillä vastaavalla tavalla, kuin käyttöjärjestelmän ydin tekee prosessien välillä.

Virtuaalikoneille pitää luoda myös illuusio siitä, että niillä olisi koko fyysinen keskusmuisti käytettävissä itselleen. Käyttöjärjestelmien yhteiselo vaatii kuitenkin, että virtuaalikoneet eivät esimerkiksi sotke toisten virtuaalikoneiden toimintaa kirjoittamalla niiden varaamille muistialueille. Virtuaalikoneiden muistin virtualisointiin ja eristämiseen tarvitaan muistinsuojausta. Luvussa 3.4 käsitellään tarkemmin, miten muistin virtualisointi voidaan tehdä käytännössä. Muistin virtualisointi ja virtuaalimuisti ovat käsitteinä hyvin lähellä toisiaan, mutta niiden merkityksillä on huomattava ero. Käyttöjärjestelmät käyttävät usein normaaleissa virtualisoimattomissakin ympäristöissä virtuaalimuistia prosessien eristämiseen. Muistin virtualisoinnissa on puolestaan tarkoituksena virtualisoida käyttöjärjestelmän muistinkäyttö, joka voi perustua virtuaalimuistiin.

Virtuaalikoneiden viimeinen merkittävä osa on IO-laitteet, kuten tallennus- ja verkkolaitteet. Fyysinen laite voidaan esimerkiksi antaa suoraan yhden virtuaalikoneen käytettäväksi [17]. Toinen vaihtoehto on jakaa laite useamman virtuaalikoneeen kesken luomalla jokaiselle oma virtuaalilaite, jotka multipleksaavat todellista laitetta [17]. Käytännössä IO-laitteiden virtualisointiin kuuluu muun muassa keskeytyspyyntöjen uudelleenohjausta virtuaalikoneille [17]. IO-virtualisointia ei käsitellä tässä tutkielmassa tarkemmin.

Virtualisointiytimet on perinteisesti luokiteltu kahteen ryhmään [3, 15, 17, 18]. Tyypin 1 virtualisointiytimen keskeinen piirre on se, että se on ainoa ohjelma, jota ajetaan prosessorin etuoikeutetussa tilassa. Se ei siis vaadi mitään alla olevaa käyttöjärjestelmää, koska se hoitaa itse monet käyttöjärjestelmän tehtävät, kuten fyysisten resurssien varaamisen ja ohjaamisen. Tyypillisesti kuitenkin yksi tyypin 1 virtualisointiytimen virtuaalikoneista on erityisessä asemassa [18]. Sen kautta tehdään esimerkiksi virtualisointiytimen konfigurointi. Se voi myös tarjota laitteistoajureita virtualisointiytimelle [16].

Tyypin 2 virtualisointiydin on normaali käyttöjärjestelmän käyttäjätilassa ajettava proses-

si. Virtualisointiytimien pitää kuitenkin yleensä toimia käyttöjärjestelmän etuoikeutetussa tilassa vähintäänkin vastaanottaakseen virtualikoneen tuottamia poikkeuksia, joten moniin tyypin 2 virtualisointiytimiin kuuluu pieni ydinmoduuli, joka palvelee käyttäjätilassa ajettavaa osaa [17].

Nykyään virtualisointiytimien kahtiajako ei ole niin selkeää [18]. Yksi yleisimmistä järjestelmävirtualisointiratkaisuista, johon muun muassa maailman suurin pilvipalvelutarjoaja Amazon vaihtoi muutama vuosi sitten, on Kernel Virtual Machine (KVM) [2]. KVM käytännössä muuttaa Linux-ytimen tyypin 1 virtualisointiytimeksi. Käyttöjärjestelmän käyttäjätilassa voidaan kuitenkin edelleen ajaa muita ohjelmia normaalisti, joten sitä voidaan pitää myös tyypin 2 virtualisointiytimenä [16, 18]. Lisäksi monissa suoritinarkkitehtuureissa virtualisointimahdollisuudet on otettu huomioon jo suunnitteluvaiheessa tai virtualisointitukea on lisätty myöhemmin arkkitehtuurilaajennoksena. Esimerkiksi x86-arkkitehtuurilla KVM-virtualisointiydin ajaa virtualisoituja käyttöjärjestelmiä kokonaan omalla prosessorin suojaustasolla, mikä vesittää edelleen perinteistä virtualisointiydinlajittelua. Laitteistoavusteista järjestelmävirtualisointia käsitellään tarkemmin luvussa 3.4.

Järjestelmävirtualisointia voidaan siis käytännössä tehdä monilla eri tyyppisillä virtualisointiytimillä ja tekniikoilla. Tekniikoiden käyttökelpoisuus riippuu usein suoritinarkkitehtuurin ominaisuuksista, jolla niitä halutaan käyttää. Popek ja Goldberg esittelivät artikkelissaan [12] kuitenkin yleisen virtualisointiytimen määritelmän, joka pätee suurimalle osalle virtualisointitekniikoista. Määritelmä muodostuu kolmesta vaatimuksesta.

Ensimmäisenä vaatimuksena sanotaan, että virtualisointiytimellä pitää olla täysi hallinta kaikista laitteiston resursseista. Toisin sanottuna virtuaalikoneelle pitää olla mahdotonta käsitellä resursseja, joita ei ole sille varta vasten varattu. Esimerkiksi, jos virtuaalikone pystyisi poistamaan globaalisti kaikki keskeytykset käytöstä, sillä olisi vaikutusta koko virtualisointiytimen ja muiden virtuaalikoneiden toimintaan, joten se pitää pystyä estämään.

Toiseksi virtualisointiytimen pitää tarjota virtuaalikoneille oikeaa vastaava ympäristö. Vastaavalla tarkoitetaan, että kaikkien ohjelmien pitää toimia identtisesti sekä virtuaalikoneella että oikealla laitteistolla ajettuna. Hyväksyttäviä poikkeuksia ovat laitteistoresurssien saatavuudesta ja ajoituksesta aiheutuvat eroavaisuudet. Virtualisointiydin voisi esimerkiksi varata neljänneksen fyysisestä keskusmuistista kullekin kolmesta virtuaalikoneesta ja jättää viimeisen neljänneksen omaan käyttöönsä. Toinen poikkeus hyväksytään, koska virtuaalikoneet ja virtualisointiydin joutuvat jakamaan saatavilla olevan suoritusajan.

Viimeisenä vaatimuksena suurin osa virtuaalikoneen käskyistä pitää suorittaa suoraan oikealla prosessorilla ilman virtualisointiytimen väliintuloa. Vaatimus rajaa emulaattorit pois virtualisointiytimien joukosta.

3.2 Klassinen virtualisointi

Yksi tunnetuimmista ja käytetyimmistä järjestelmävirtualisointitekniikoista tunnetaan parhaiten englannin kielisellä termillä "trap and emulate", mutta sitä kutsutaan usein myös klassiseksi virtualisoinniksi. Popek ja Goldberg esittelivät artikkelissaan [12] formaalin todistuksen teoreemalle, joka määrittelee klassisen virtualisoinnin mahdollistavat vaatimukset suoritinarkkitehtuurille. Vaatimukset ovat riittävät, muttei välttämättömät suoritinarkkitehtuurin virtualisointiin [11]. Popekin ja Goldbergin työtä pidetään kuitenkin edelleen hyvin merkittävänä, koska suuri osa artikkelin julkaisua seuraavasta virtualisointitutkimuksesta perustuu siihen [11, 17].

Alkuperäisen teoreeman muodostuksessa tehtyjen oletusten vuoksi se soveltuu aukottomasti ainoastaan artikkelin julkaisun aikaisten ”kolmannen sukupolven” suoritinarkkitehtuurien virtualisoitavuuden tarkasteluun. Teoreeman mallia on kuitenkin päivitetty nykyaikaisille suoritinarkkitehtuureille Pennemanin ja muiden [11] toimesta. Uudistetussa mallissa otetaan huomioon muun muassa sivuttava virtuaalimuisti, IO sekä ansat. Malli on kehitetty erityisesti ARM-suoritinarkkitehtuuria varten, mutta se sopii arkkitehtuurien samankaltaisuuden vuoksi myös RISC-V:n tarkasteluun.

Teoreeman muodostuksessa Popek ja Goldberg esittelivät uusia käsitteitä. Ensimmäisenä etuoikeutetulla käskyllä (engl. privileged instruction) tarkoitetaan suoritinarkkitehtuurin käskyä, joka prosessorin käyttäjätilassa suoritettuna tuottaa poikkeuksen [12]. Etuoikeutetun käskyn määritelmään ei tarvitse tehdä muutoksia päivitetyssä mallissa [11].

Ehdollisilla käskyillä on puolestaan useampi alatyyppi, joiden määritelmät vaativat tarkennuksia alkuperäisestä mallista. Ohjaus-ehdollisella käskyllä (engl. control sensitive instruction) tarkoitetaan käskyä, joka muuttaa prosessorin suoritustilaa, control- tai statusrekistereitä tai aktiivista sivutaulua [11]. Tila-ehdollisella käskyllä (engl. mode sensitive instruction) tarkoitetaan puolestaan käskyä, jonka suorituskäyttäytyminen riippuu prosessorin suoritustilasta [11]. Viimeiseksi Penneman ja muut määrittelivät uuden ehdollisen käskyn alatyypin, jolle ei ole vastinetta alkuperäisessä mallissa. Käsky on konfiguraatioehdollinen (engl. configuration sensitive), jos sen suorituskäyttäytyminen riippuu control- tai status-rekisterien tilasta [11].

Käsitteiden avulla voidaan lausua Popekin ja Goldbergin teoreema. Teoreeman mukaan suoritinarkkitehtuuri on (klassisesti) virtualisoitavissa, jos suoritinarkkitehtuurin käskykannan ehdollisten käskyjen joukko on etuoikeutettujen käskyjen osajoukko [12]. Toisin sanottuna, jos käyttäjätilassa yritetään suorittaa käskyä, jota siellä ei pitäisi suorittaa, laitteiston tulee tuottaa poikkeus [17]. Teoreema pitää paikkansa myös moderneilla suoritinarkkitehtuureilla päivitetyn mallin kanssa [11].

Klassinen virtualisoitavuus mahdollistaa virtualisoidun käyttöjärjestelmän ajamisen muuttamattomana prosessorin käyttäjätilassa. Kaikki klassisesti virtualisoitavan suoritinarkki-

tehtuurin käskyt, jotka toimisivat eri tavalla oikealla laitteistolla etuoikeutetussa tilassa ajettuna, tuottavat varmasti poikkeuksen. Etuoikeuteussa tilassa ajetaan joko tyypin 1 virtualisointiydintä tai tyypin 2 virtualisointiytimen ydinmoduulia, joka vastaanottaa virtualisoitavan käyttöjärjestelmän tuottamat poikkeukset ja emuloi niiden vaikutuksen. Esimerkiksi, jos virtualisoitu käyttöjärjestelmä yrittäisi ajaa käskyä, joka poistaa keskeytykset käytöstä, laitteisto tuottaisi poikkeuksen, jonka seurauksena suoritus siirtyisi virtualisointiytimeen, joka näkisi poikkeuksen tuottaneen käskyn ja emuloisi sen vaikutuksen päivittämällä virtuaalisen keskeytyslipun tilaa.

3.3 Dynaaminen binääritranslaatio

Popek ja Goldberg esittelivät artikkelissaan [12] myös virtualisointitekniikoita suoritinarkkitehtuureille, jotka eivät täyttäneet heidän asettamiaan vaatimuksia. Esiteltyjä tekniikoita voidaan pitää alkukantaisina versioina nykyään yleisistä dynaamisista binääritranslaatiotekniikoista (engl. dynamic binary translation, DBT) [11]. DBT on virtualisointitekniikkana monikäyttöinen, koska se ei tarvitse erityistä tukea suoritinarkkitehtuurilta tai käyttöjärjestelmältä, mikä erottaa sen muista tutkielmassa mainituista virtualisointitekniikoista [19]. DBT-virtualisointiytimet ovat kuitenkin tunnetusti huomattavasti monimutkaisempia kuin klassista virtualisointia hyödyntävät vastineet [11].

Dynaamisen binääritranslaation ideana on korvata ongelmalliset käskyt muilla käskyillä, joilla on sama vaikutus [17, 19]. Ongelmallisia käskyjä ovat ne, jotka eivät ole virtualisoitavissa Popekin ja Goldbergin vaatimusten mukaisesti. Ongelmallinen käsky siis käyttäytyy eri tavalla virtualisointiytimen alla käyttäjätilassa suoritettuna kuin normaalissa tapauksessa etuoikeutetussa tilassa suoritettuna. Ongelmallinen käsky ei myöskään tuota poikkeusta käyttäjätilassa suoritettuna, joten klassista trap and emulate -toimintatapaa ei ole mahdollista käyttää suoraan.

Tekniikan dynaamisuus viittaa siihen, että ongelmallisten käskyjen korvaus tapahtuu ajoaikana [19]. Ennen virtualisoitavan käyttöjärjestelmän käynnistystä virtualisointiydin skannaa käyttöjärjestelmän ensimmäisestä käskystä alkaen käskyjä eteenpäin korvaten ongelmalliset käskyt vastaavilla käskyillä. Käskyjen skannaus jatkuu, kunnes löydetään hyppy, funktiokutsu tai muu haarautumiskäsky. Viimeinen käsky korvataan hypyllä virtualisointiytimeen, jotta sama ongelmallisten käskyjen translaatioprosessi voidaan toistaa seuraavalle joukolle käskyjä. Prosessin viimeisenä vaiheena suoritetaan edellisen kierroksen viimeisenä korvattu haarautumiskäsky, eli jatketaan siitä mihin virtualisoitavan käyttöjärjestelmän suorituksessa jäätiin. [1, 17]

Tekniikan kanssa voidaan hyödyntää trap and emulate -tyylisiä emulointirutiineja, jos ongelmalliset käskyt korvataan ohjelmistopoikkeuksen tuottavilla käskyillä. Poikkeuksen seurauksena suoritus tulee siirtymään hetkellisesti virtualisointiytimelle, missä alkuperäisen käskyn vaikutus voidaan jäljitellä virtuaaliselle laitteistolle. Toisaalta translaation teke-

minen suoraan virtualisoitavan käyttöjärjestelmän käskyjen joukkoon johtaa yleensä parempaan tehokkuuteen. Ansan seurauksena joudutaan yleensä käyttämään kymmeniä, ellei satoja kellojaksoja prosessorin hetkellisen tilan tallentamiseen, jotta siihen voidaan palata translaatioprosessin jälkeen. Ansat ovat ongelmallisia myös prosessorin välimuistien kannalta. Heti kontekstin vaihdon jälkeen välimuistista löytyy vain vanhan kontekstin muistipaikkoja, joten prosessorin pitää käyttää paljon välimuistia hitaampaa keskusmuistia. [17]

Virtualisoitavan käyttöjärjestelmän käyttäjätilan ohjelmat voidaan suorittaa DBT-virtualisointiytimen alaisuudessa normaalisti, mikä takaa niille lähes natiivin suorituskyvyn [1, 11]. Käyttäjätilan koodi toimii virtualisointiytimenkin kanssa käyttäjätilassa, joten sille ei tarvitse tehdä muutoksia. Vaikka virtualisoitavan käyttöjärjestelmän ytimen käskyille pitää tehdä raskasta translaatiota, voidaan sen suorituskykyä parantaa välimuistipuskureiden avulla. Ennen translaation tekemistä voidaan tarkistaa, löytyykö translaation tulos jo välimuistista, jos sama translaatio on tehty jo aikaisemmin. Uudet translaatiotulokset talletetaan välimuistiin, jolloin seuraavalla kerralla ne voidaan suorittaa välittömästi. [17, 19]

3.4 Laitteistoavusteinen virtualisointi

Virtualisointimahdollisuudet on usein otettu huomioon modernien suoritinarkkitehtuurien suunnittelussa. Laitteistoavusteisella virtualisoinnilla tarkoitetaan virtualisointitekniikoita, joissa hyödynnetään laitteiston virtualisointiin suunniteltuja ominaisuuksia. Sen merkittävin etu muihin virtualisointitekniikoihin verrattuna on parempi suorituskyky [15].

Laitteistoavusteisessa virtualisoinnissa virtuaalikoneita ajetaan tyypillisesti omassa suoritustilassa, joka näyttää virtuaalikoneelle samalta, kuin normaali etuoikeutettu tila [1, 17]. Virtuaalikoneilla on käytettävissä muun muassa kaikki normaalit control- ja status-rekisterit, jotka löytyvät normaalisti etuoikeutetusta tilasta. Etuoikeutettujen rekisterien käyttämiseen ei tarvitse käyttää trap and emulate -tekniikkaa, koska virtuaalikoneet voivat käyttää niitä suoraan.

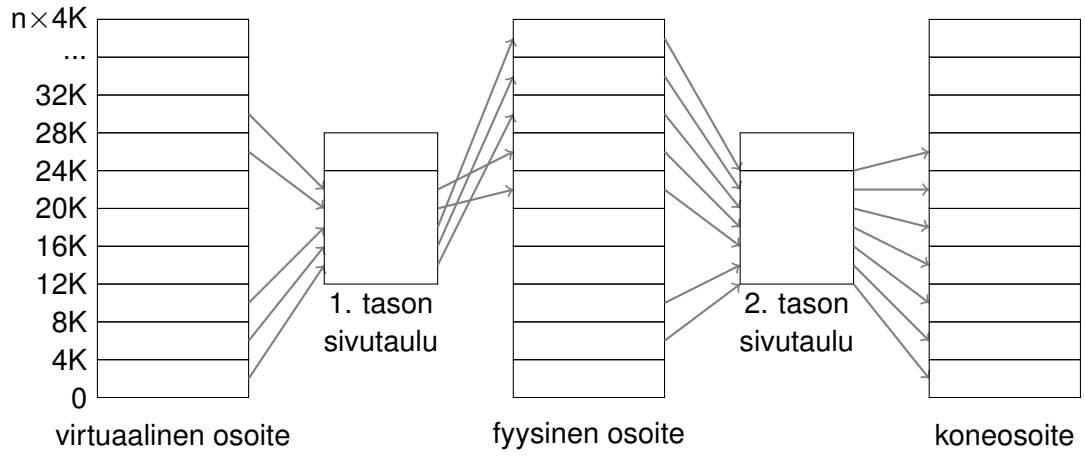
Normaalisti etuoikeutettujen rekisterien käyttäminen suoraan vaikuttaisi koko laitteiston sen kautta muiden virtuaalikoneiden toimintaan. Laitteistoavusteisessa virtualisoinnissa virtuaalikoneiden näennäisen etuoikeutetun tilan yläpuolella on kuitenkin vielä enemmän etuoikeutettu suoritustila, jolla on todellinen hallinta koko laitteistosta [1, 17]. Virtuaalikone voi esimerkiksi kytkeä keskeytykset pois päältä omassa suoritustilassaan, mutta se ei vaikuta korkeamman oikeustason tilan toimintaan, jossa virtualisointiydin toimii.

Ennen virtuaalikoneen käynnistämistä virtualisointiydin initialisoi näennäisen etuoikeutetun tilan rekisterit samanlaiseen tilaan, kuin ne olisivat oikealla laitteistolla. Virtuaalikoneiden kontekstinvaihdossa virtualisointiydin tallentaa edellisen virtuaalikoneen rekisterien arvot talteen ja vaihtaa tilalle seuraavan virtuaalikoneen arvot. Virtuaalikoneiden konteks-

tinvaihto on verrattavissa käyttöjärjestelmän prosessien kontekstinvaihtoon. Vaihdettavia rekisterejä on vain enemmän, koska käyttäjätilan rekisterien lisäksi myös etuoikeutetun tilan rekisterit pitää vaihtaa.

Laitteistoavusteista virtualisointia voidaan hyödyntää myös klassisesti virtualisoimattomilla suoritinarkkitehtuureilla, joihin kuuluu muun muassa x86 [1] ja ARMv7 [11]. Epäetuoikeutettujen käskyjen ehdollisuus ei ole ongelma, koska kaikki käskyt ajetaan virtualisointiytimen alaisuudessa samanlaisessa suoritustilassa, kuin normaalissa virtualisoimattomassakin ympäristössä. Etuoikeutettujen käskyjen suorittamisesta ei seuraa poikkeuksia, eikä niitä tarvitse emuloida, kuten muissa trap and emulate -tyylisissä virtualisointitekniikoissa. Laitteistoavusteisen virtualisoinnin suorituskyky edut johtuvat juuri ansojen välttämisestä [1]. Mitä vähemmän virtualisointiytimen tarvitsee tulla väliin, sitä enemmän virtuaalikoneille jää vapaata suoritusaikaa.

Laitteiston avulla voidaan tehostaa myös muistin virtualisointia. Jos käytettävällä laitteistolla ei ole tukea muistin virtualisointiin, käytetään samaan tarkoitukseen useimmiten sivutukseen ja varjosivutauluihin (engl. shadow page tables) perustuvaa tekniikkaa [1, 17]. Siinä virtuaalikoneille luodaan illuusio jatkuvasta fyysisestä muistiavaruudesta tekemällä virtuaalikoneen käsittelemistä fyysisistä muistiosoitteista kuvaus todellisiin koneosoitteisiin (engl. machine addresses), joilla muistipaikkojen luku- ja kirjoitusoperaatiot lopulta tehdään. Virtualisoitava käyttöjärjestelmä voi itse käyttää sivutusta prosessien eristämiseen, jolloin se luo kuvauksen virtuaalisista osoitteista fyysisiin osoitteisiin itse ylläpitämällään ensimmäisen tason sivutaululla. Lisäksi virtualisointiydin ylläpitää omaa toisen tason sivutaulua, joka kuvaa fyysiset osoitteet edelleen todellisiksi koneosoitteiksi.



Kuva 3.2. *Kaksitasoinen osoitetranslaatio.*

Kaksitasoisen muistitranslaatioprosessin tekeminen pelkällä ohjelmistolla olisi erittäin hidasta, koska virtualisointiytimen pitäisi jokaisen muistiviittauksen yhteydessä käydä manuaalisesti läpi molemmat sivutaulut [11]. Ongelma voidaan välttää käyttämällä translaation tekemiseen laitteiston sivutusmekanismia. Normaalisti laitteistolla voidaan tehdä ainoastaan yksi translaatio yhden sivutaulun perusteella, joten ensimmäisen ja toisen tason

sivutaulut pitää sulattaa yhteen. Yhdistettyä sivutaulua kutsutaan varjosivutauluksi, koska sen pitää varjostaa ensimmäisen ja toisen tason sivutaulujen muutoksia [1, 11, 17]. Se kuvaa virtuaaliset osoitteet suoraan koneosoitteiksi ilman välivaiheita. Virtualisointiytimen tehtäväksi jää varjosivutaulun ylläpitäminen ensimmäisen ja toisen tason sivutauluihin tehtävien muutosten perusteella. Virtualisointiydin voi esimerkiksi virtuaalikoneen oman ensimmäisen tason sivutaulun kuvauksessa estää kirjoitusoperaatioiden tekemisen, jolloin se voi aina sivutuspoikkeuskäsittelijässä tarkastaa, mitä muutosta virtuaalikone oli tekemässä omaan sivutauluunsa, ja sen perusteella päivittää varjosivutaulua [1, 17].

Menemättä enempää varjosivutaulujen yksityiskohtiin voidaan sanoa lyhyesti, että niiden ylläpitäminen on hidasta ja monimutkaista [1]. Monissa suoritinarkkitehtuureissa on laitteistotuki kaksitasoiselle osoitetranslaatiolle vaihtoehtona varjosivutauluille [1, 17]. Kaksitasoisen osoitetranslaation toimintaperiaate on esitetty kuvassa 3.2. Siinä virtuaalikone voi käyttää vapaasti ensimmäisen tason sivutaulua omiin tarkoituksiinsa. Ensimmäisen tason sivutaulu ladataan siis suoraan laitteistolle ilman varjosivutaulua. Lisäksi virtualisointiydin määrittelee laitteistolle omalta suojaustasoltaan toisen tason sivutaulun. Virtuaalikoneita ajettaessa laitteisto tekee kaksitasoisen osoitetranslaation automaattisesti sivutaulujen perusteella, jolloin ohjelmiston ei tarvitse huolehtia itse niiden yhdistämisestä.

3.5 Paravirtualisointi

Paravirtualisointi on binääritranslaation ohella toinen järjestelmävirtualisointitekniikka, jota voidaan hyödyntää klassisesti virtualisoimattomissa ympäristöissä. Paravirtualisointiytimet eroavat muista virtualisointiytimistä, sillä että ne eivät täytä toisena mainittua yleisen virtualisointiytimen määritelmän ehtoa. Paravirtualisointiydin ei siis tarjoa virtuaalikoneille täysin oikeaa vastaava ympäristöä. Paravirtualisoidussa käyttöjärjestelmässä ongelmalliset virtualisoimattomat käskyt korvataan rajapintakutsuilla virtualisointiytimelle (engl. hypercalls). Paravirtualisointiydin jäljittelee rajapintakutsulla korvatun käskyn vaikutuksen virtuaaliselle laitteistolle. [11, 17, 19]

Käytännössä rajapintakutsu voidaan tehdä samanlaisella mekanismilla, millä käyttäjätilassa ajettava ohjelma tekee järjestelmäkutsun käyttöjärjestelmän ytimelle. Jos kuitenkin paravirtualisointiydin ja virtualisoitu käyttöjärjestelmä toimivat samalla suojaustasolla, niiden väliseen kommunikaatioon ei tarvita suorituskyvyn kannalta kalliita ansoja. Rajapintakutsut voidaan tehdä yksinkertaisesti samalla mekanismilla, millä mikä tahansa muukin funktiokutsu tehtäisiin. Funktiokutsujen etuja ansoihin verrattuna ovat edellä mainitut prologien ja epilogien koot sekä välimuistien eheys.

Paravirtualisoitu käyttöjärjestelmä pitää etukäteen muuttaa hyödyntämään virtualisointiytimen tarjoamaa rajapintaa [17, 18, 19]. Rajapinta voi poiketa oikean laitteiston tarjoamasta rajapinnasta huomattavasti. Standardoitujen rajapintojen käyttö ei ole levinnyt kuin korkeintaan muutamaan käyttöjärjestelmään. Seurauksena vain harvat käyttöjärjestelmät tu-

kevat paravirtualisointia suoraan. Käyttöjärjestelmien ja paravirtualisointiytimien kehittäjät tarjoavat useammin käyttöjärjestelmälaajennoksia, joilla tietty käyttöjärjestelmä saadaan paravirtualisoitua tietyn paravirtualisointiytimen kanssa. [11]

Käyttöjärjestelmälaajennosten kehittäminen ja ylläpitäminen kaikille käyttöjärjestelmien ja virtualisointiytimien yhdistelmille on ongelmallista. Laajennosten kehittäminen on kallista ja vie paljon aikaa varsinkin, jos kustomoidun käyttöjärjestelmän turvallisuus ja muu toiminta halutaan varmistaa formaalisti. Käyttöjärjestelmien lisensointi voi myös estää niiden lähdekoodin modifioinnin, mikä tekee paravirtualisoinnin yhteensopivuudesta huonon. [11]

Paravirtualisoinnin vastakohta on täysvirtualisointi, jossa virtualisoitavaan käyttöjärjestelmään ei tarvitse tehdä mitään muutoksia etukäteen [11, 19]. Kaikki edellä mainitut virtualisointitekniikat kuuluvat täysvirtualisointitekniikoiden joukkoon. Dynaamisessa binääritranslaatiossa virtualisoitavaan käyttöjärjestelmään tehdään muutoksia automaattisesti ajoaikana. Sitä ei kuitenkaan pidetä paravirtualisointina, koska virtualisoitavaan käyttöjärjestelmään ei tarvitse tehdä manuaalisia muutoksia etukäteen.

3.6 Käyttöjärjestelmätason virtualisointi

Käyttöjärjestelmätason virtualisointi on suhteellisen uusi lähestymistapa virtualisointiin verrattuna edellä mainittuihin järjestelmävirtualisointitekniikoihin, jotka ovat laajasti tutkittuja ja käytettyjä [13]. Käyttöjärjestelmätason virtualisoinnilla eli säiliöinnillä tarkoitetaan virtualisointitekniikaa, jossa käyttäjätilassa ajettaville prosesseille tarjotaan oma virtualisoitu ja muista prosesseista eristetty ympäristö [13, 18]. Säiliöinti ei siis kuulu järjestelmävirtualisointitekniikoiden joukkoon, koska siinä ei virtualisoida koko käyttöjärjestelmän ympäristöä, eli laitteistoa, vaan ainoastaan prosessien ympäristöjä, joita kutsutaan usein säiliöiksi tai konteiksi (engl. containers) [13, 18].

Säiliöinnin etuna järjestelmävirtualisointiin verrattuna on se, että virtualisoidut prosessit voivat käyttää yhteistä käyttöjärjestelmän ydintä ja sen palveluita [13, 18]. Esimerkiksi oheislaitteiden käyttö onnistuu normaaleilla järjestelmäkutsuilla. Järjestelmävirtualisoinnissa järjestelmäkutsu tehdään ensin virtualisoidulle ytimelle, jonka ajurin kautta kutsu viimein päätyy virtualisointiytimelle ja laitteistolle. Käyttöjärjestelmätason virtualisoinnissa on vähemmän välivaiheita, mikä johtaa parempaan suorituskykyyn [13]. Virtualisoitujen käyttöjärjestelmien duplikaatit ajurit ja muut ytimen palvelut vievät myös enemmän tilaa, kuin käyttöjärjestelmätasolla virtualisoitu järjestelmä.

Käytännössä säiliöinnin todellinen toimintaperiaate riippuu käyttöjärjestelmästä. Esimerkiksi Linuxissa säiliöinti voidaan toteuttaa hyödyntämällä nimiavaruuksia (engl. namespaces) [18]. Jokainen prosessi Linuxissa kuuluu yhteen tai useampaan nimiavaruuteen. Nimiavaruus on eristetty joukko käyttöjärjestelmän resursseja, joita ainoastaan siihen

kuuluvat prosessit pystyvät käsittelemään [18]. Prosessit voidaan siten eristää toisistaan sijoittamalla ne eri nimiavaruuksiin. Eristys parantaa palveluiden tietoturvaa, sillä tietomurretusta palvelusta on vaikeampi päästä käsiksi toiseen palveluun, jos ne on eristetty hyvin toisistaan.

Säiliöinnillä on eristyksen lisäksi myös muut järjestelmävirtualisointitekniikoiden kanssa yhteiset käyttöönotto- ja skaalautuvuusedut [18]. Järjestelmävirtualisoitujen käyttöjärjestelmien ympäristöön voi kuulua esimerkiksi virtuaalisia prosessoreita, levyjä ja verkkokortteja. Vastaavasti säiliöidyn prosessin ympäristö muodostuu muun muassa käyttäjistä ja käyttöoikeuksista, tiedostojärjestelmistä sekä verkkokonfiguraatioista. Linuxilla säiliöinti tehdään yleensä työkaluilla, jotka helpottavat säiliöympäristöjen luontia. Yksi käytetyimmistä säiliöintiratkaisuista on Docker-ohjelmisto, jolla Linux-ytimen nimiavaruusrajapintaa voidaan käyttää helppokäyttöisen käyttöliittymän kautta [18]. Dockerin avulla haluttu säiliöympäristö määritellään tiedostoon, jonka perusteella se voidaan pystyttää nopeasti minne tahansa identtisessä muodossa riippumatta käyttöjärjestelmän muusta konfiguraatiosta.

4. TULOKSET

Seuraavaksi kerrataan edellisessä luvussa mainitut virtualisointitekniikat käymällä läpi, mitä vaatimuksia niiden käyttäminen asettaa suoritinarkkitehtuureille. Virtualisointitekniikoiden käyttökelpoisuus selvitetään erityisesti RISC-V:llä.

4.1 Klassinen virtualisointi

Klassisessa virtualisoinnissa käyttöjärjestelmiä voidaan ajaa muuttamattomina prosessorin käyttäjätilassa. Se ei siis vaadi erityistä ohjelmiston räätälöintiä tai muuta ohjelmistotukea. Laitteiston sen sijaan pitää täyttää tietyt vaatimukset. Ensinnäkin klassisen virtualisoinnin yhteinen piirre muiden täysvirtualisointitekniikoiden kanssa on etuoikeutetun arkkitehtuurin tarve. Suojaustasoja tarvitaan, jotta virtualisointiydin voi täyttää Popekin ja Goldbergin määritelmän (katso luku 3.1) ensimmäisen kohdan. Virtualisointiytimen pitää olla etuoikeutetussa asemassa virtuaalikoneihin verrattuna, jotta se pystyy hallitsemaan laitteiston resursseja.

Suojaustasojen lisäksi suoritinarkkitehtuurin pitää täyttää Popekin ja Goldbergin asettamat vaatimukset. RISC-V-suoritinarkkitehtuurin alkuperäisiin suunnittelutavoitteisiin kuului klassinen virtualisoitavuus [20]. Se ei kuitenkaan takaa, että RISC-V on klassisesti virtualisoitavissa. Asia voidaan selvittää formaalisti käymällä läpi käskykannan kaikki käskyt Goldbergin ja Popekin esittämien ehtojen mukaisesti. Tutkimuskohteena käytetään standardia RV32G-käskykantaa, johon kuuluu RV32I-perusosan ja standardin etuoikeutetun arkkitehtuurin lisäksi M-, A-, F-, D-, Zicsr- ja Zifencei-laajennokset. Tutkimus tehdään käytännössä varmistamalla, että tutkittavan käskykannan kaikki ehdolliset käskyt ovat etuoikeutettuja käskyjä.

Ensimmäisenä käskykannan perusosasta voidaan sanoa, että se on suurimmaksi osaksi ehdoton. Aritmeettis-loogisten käskyjen, haarautumiskäskyjen, load-store-käskyjen ja muiden perusluontoisten käskyjen suorituskäyttäytyminen ei riipu millään tavalla prosessorin suoritustilasta tai control- tai status-rekisterien tilasta. Ensimmäinen poikkeus on `ecall`-käsky, jota käytetään järjestelmäkutsujen tekemiseen. Käsky tuottaa poikkeuksen, jonka seurauksena prosessorin suoritustila voi muuttua esimerkiksi U-tilasta S-tilaan, joten se voidaan luokitella ohjaus-ehdolliseksi-käskyksi. Se täyttää myös ohjaus-ehdollisenkäskyn toisen tuntomerkin, sillä poikkeuksen tuottaneen käskyn osoite tallennetaan auto-

maattisesti status-rekisteriin. Käskyn tuottama poikkeus riippuu myös prosessorin suoritustilasta, jotta ansan käsittelijä voi tietää, mistä palvelupyyntö on tehty, joten se on lisäksi tila-ehdollinen.

Toinen poikkeus on `ebreak`-käsky, jota voidaan käyttää ohjelman keskeytykseen debuggertyökaluilla. Käsky tuottaa breakpoint-poikkeuksen, joten se voidaan luokitella `ecall`-käskyn kanssa samasta syystä ohjaus-ehdolliseksi. Molemmat käskyt tuottavat kuitenkin poikkeukset myös käyttäjätilassa, joten ne voidaan luokitella etuoikeutetuiksi käskyiksi ja siten virtualisoitaviksi. Virtualisointiytimen pitää vain huomioida `ecall`-käskyn tila-ehdollisuus siten, että se tarkistaa palvelupyynnön lähdetilan ansakoodin sijaan omista virtuaalikoneiden tilaa kuvaavista tietorakenteista.

Etuoikeutettuun arkkitehtuuriin kuuluu itsessään neljä käskyä: `mret`, `sret`, `wfi` ja `sfence.vma`, joiden suoritus ei onnistu kaikissa suoritustiloissa. Käskyt ovat etuoikeutettuja, koska RISC-V-spesifikaatio määrittelee, että laitteiston tulee tuottaa poikkeus, jos prosessorin sen hetkisellä suoritustilalla ei ole vaadittavia oikeuksia niiden suorittamiseen. Seurauksena niiden ehdollisuus ei ole virtualisoinnin kannalta ongelma. Esimerkiksi `sfence.vma`-käsky on selkeästi ohjaus-ehdollinen, koska se invalidoi käytettävän sivutaulun TLB-välimuistissa. Käyttäjätilassa suoritettuna se kuitenkin tuottaa poikkeuksen, joten virtualisointiydin voi emuloida sen vaikutuksen virtuaaliselle laitteistolle.

Sama pätee Zicsr-laajennoksen käskyille, joita käytetään control- ja status-rekisterien lukemiseen ja kirjoittamiseen. Zicsr-käskyt ovat ohjaus-ehdollisia, koska ne muuttavat control- ja status-rekisterien arvoja, sekä tila-ehdollisia, koska niiden suorituskäyttäytyminen riippuu prosessorin suoritustilasta. Käskyt ovat kuitenkin myös etuoikeutettuja ja siten virtualisoitavia, koska RISC-V spesifikaatio määrittelee, että laitteiston tulee tuottaa poikkeus, jos käskyn rekisteri ei ole käytettävissä sen hetkisessä suoritustilassa.

Liukulukulaajennoksiin (F ja D) kuuluvat käskyt näyttävät ensisilmäyksellä ongelmallisilta. Käskyjen suoritukseen vaikuttaa liukulukuyksikön fcsr-status-rekisterin pyöristystilaa ohjaava frm-kenttä, joten niitä voidaan pitää konfiguraatio-ehdollisina. Käskyt asettavat ylivuotoja ja muita virhetilanteita kuvaavia lippuja fcsr-rekisterin fflags-kenttään, joten niitä voidaan pitää myös ohjaus-ehdollisina. Käskyt eivät ole etuoikeutettuja, koska ne eivät tuota poikkeuksia missään tilanteessa.

Todellisuudessa fcsr-rekisterin käyttöä ei kuitenkaan tarvitse emuloida. Rekisteriä voi käyttää suoraan U-tilasta, joten se toimii käytännössä yleiskäyttöisen rekisterin tavoin. Virtualisointiydin voi antaa virtuaalikoneiden käyttää fscr-rekisteriä vapaasti ja tallentaa sen kontekstinvaihdossa muiden yleiskäyttöisten rekisterien tavoin. Sama pätee muille U-tilasta käytettäville control- ja status-rekistereille.

Kaikki käskyt läpikäytyä voidaan sanoa, että RV32G-mukainen RISC-V-suoritinarkkitehtuuri on klassisesti virtualisoitavissa, koska ehdolliset käskyt ovat etuoikeutettujen käskyjen

osajoukko. Käytännössä klassinen virtualisointi voidaan tehdä ajamalla virtualisointiydintä etuoikeutetussa S-tilassa ja virtuaalikoneita U-tilassa. Standardissa käskykannassa etuoikeutettuja käskyjä on vain muutama, joten niiden käsittely trap and emulate -periaatteella ei tuota suuria määriä ansoja. Arkkitehtuuriin on kuitenkin suunniteltu ominaisuus, jolla klassisen virtualisoinnin tehokkuutta voidaan parantaa vähentämällä ansojen määrää entisestään.

Klassisesti virtualisoitavia käyttöjärjestelmiä voidaan ajaa suoraan S-tilassa hyödyntämällä mstatus-rekisterin T-lippuja [22]. Esimerkiksi TVM-lipulla (Trap Virtual Memory) voidaan käytännössä ottaa muistinsuojauksen hallintaoikeudet pois S-tilalta. Sen ollessa päällä sivutaulun muuttamiseen ja invalidointiin käytetyt käskyt tuottavat poikkeuksen, jonka seurauksena M-tilassa toimiva virtualisointiydin voi esimerkiksi päivittää käytetyt varjosivutaulut. S-tilassa suoritettuna virtualisoitu käyttöjärjestelmä voi käsitellä satp-rekisteriä lukuun ottamatta muita control- ja status-rekistereitä suoraan ilman ansoja ja emulointia, mikä parantaa järjestelmän suorituskykyä.

4.2 Dynaaminen binääritranslaatio

Laitteistolta ei dynaamisessa binääritranslaatiossa tarvita muuta, kuin muistinsuojaus ja suojaustasot, jotta virtuaalikoneet voidaan eristää toisistaan ja virtualisointiytimestä. Binääritranslaatiota käytetään usein myös klassisesti virtualisoitavissa ympäristöissä ansojen välttämiseksi [17]. RISC-V:llä klassinen virtualisointi voidaan kuitenkin tehdä minimaalisella määrällä ansoja, joten binääritranslaation käyttäminen ei vaikuta järkevältä, vaikka se olisi mahdollista. DBT-virtualisointiytimet ovat tyypillisesti monimutkaisia ja isokokoisia [11], joten ne eivät välttämättä sovellu esimerkiksi pieniin sulautettuihin järjestelmiin.

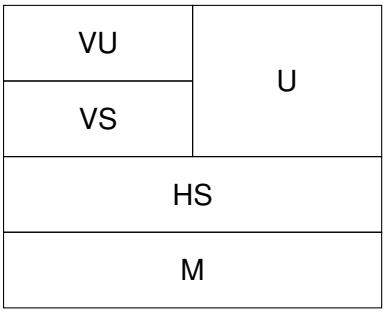
4.3 Laitteistoavusteinen virtualisointi

RISC-V on edellisen perusteella hyvin virtualisoitavissa ilman mitään arkkitehtuurilaajennoksia. RISC-V:n standardi H-laajennos lisää kuitenkin arkkitehtuuriin laitteistotukea virtualisoinnin tehostamiseen. Laajennoksen toiminnallisuus on analoginen x86- ja ARM-arkkitehtuurien laitteistoavusteiseen virtualisointiin suunniteltujen laajennosten kanssa [1, 11, 22].

H-laajennos muuttaa etuoikeutetun S-tilan HS-tilaksi, joka on tarkoitettu virtualisointiytimelle. Tyypin 2 virtualisointiydin voi toimia myös U-tilassa, jolloin HS-tilassa toimii käyttöjärjestelmän ydin. HS-tila lisää ominaisuuksia S-tilaan verrattuna, mutta ei poista mitään, joten S-tilaan suunniteltuja käyttöjärjestelmiä voidaan käyttää muuttamattomina HS-tilassa. [22]

Lisätyillä ominaisuuksilla virtualisointiydin voi käynnistää virtualisoidun käyttöjärjestelmän rajatussa VS-tilassa. VS-tila näyttää virtuaalikoneelle vastaavalta, kuin oikea S-tila, jo-

ten S-tilaan suunniteltuja käyttöjärjestelmiä voidaan käyttää muuttamattomina myös VS-tilassa [22]. Todellisuudessa kuitenkin HS-tilassa toimivalla virtualisointiytimellä on oikea hallinta laitteistosta. Se pystyy esimerkiksi valitsemaan, mitkä keskeytykset ovat käytössä, ennen kuin virtuaalikoneen valinnoilla on siihen mitään vaikutusta. Suoritustilojen järjestys on esitetty kuvassa 4.1.



Kuva 4.1. *H-laajennoksen sisältävän RISC-V-prosessorin suoritustilat. [22]*

Virtualisointiytimellä on myös hallinta keskusmuistista. Muistin virtualisointiin RISC-V käyttää tavanomaista kaksitasoista osoitetranslaatiota. HS-tilasta virtualisointiydin voi määritellä hgatp-rekisterissä toisen tason sivutaulun osoitteen [22]. Laitteisto tekee kaksitasoisen translaation automaattisesti, kun prosessori on virtualisoidussa suoritustilassa (VS tai VU) [22].

H-laajennos lisää virtualisoinnin tehostamiseen innovatiiviset [15] load- ja store-käskyt virtuaalikoneiden muistinkäsittelyyn. Käskyjen avulla virtualisointiydin voi käyttää virtuaalikoneiden muistiavaruuksia lisäämättä niiden kuvauksia omiin sivutauluihin [22]. Laitteisto tekee siis niille automattisesti kaksitasoisen osoitetranslaation. Käskyjä voidaan käyttää jopa U-tilasta, mikä helpottaa tyypin 2 virtualisointiytimien implementaatioita [22].

H-laajennos on suunniteltu helposti emuloitavaksi alustoilla, jotka eivät implementoi sitä. Virtualisointiydin voi toimia normaalisti S-tilassa. HS-tilan lisäämien control-rekisterien käyttäminen tuottaa ansan, jolloin ne voidaan emuloida M-tilassa. Suurin osa virtualisointiytimen käyttämistä control-rekistereistä on laillisia S-tilan rekistereitä, joten niiden käyttöä ei tarvitse emuloida. [22]

4.4 Paravirtualisointi

Paravirtualisoinnissa virtuaalikoneet ja virtualisointiydin ovat tietoisia toistensa olemassaolosta. Virtualisoitavat käyttöjärjestelmät pitää etukäteen muuttaa hyödyntämään virtualisointiytimen rajapintaa. Paravirtualisoinnille edes muistinsuojaus ei ole välttämätön vaatimus, koska muun muassa virtualisointiytimen ja virtuaalikoneiden fyysisen keskusmuistin jakaminen voidaan sopia etukäteen. Tietoturvan parantamiseksi virtuaalikoneet kannattaa kuitenkin eristää toisistaan laitteiston mekanismeilla.

RISC-V:llä paravirtualisoidut käyttöjärjestelmät voidaan eristää toisistaan PMP-mekanismilla

[15]. M-tilassa toimiva paravirtualisointiydin jakaa jokaiselle virtuaalikoneelle osan fyysisestä muistiavaruudesta. Virtualisoitavien käyttöjärjestelmien pitää olla tietoisia omasta alueestaan. Käytettävä muistiavaruus voidaan määritellä käyttöjärjestelmän käännösvaiheessa tai virtualisointiydin voi välittää sen dynaamisesti käyttöjärjestelmän käynnistysvaiheessa. Virtualisoitava käyttöjärjestelmä voi toimia pelkästään U-tilassa, mutta se on vapaa käyttämään myös S-tilaa, jos esimerkiksi tarvitsee sivutusta prosessien eristämiseen.

4.5 Käyttöjärjestelmätason virtualisointi

Käyttöjärjestelmätason virtualisoinnissa säiliöiden luonnista ja ylläpidosta vastaa käyttöjärjestelmä itse. Virtualisoitavat yksiköt ovat yleensä normaaleja prosesseja, joiden järjestelmäkutsuille ydin tekee vain ylimääräisiä tarkistuksia. Käyttöjärjestelmätason virtualisointi edellyttää siten laitteistolta samoja fundamentteja, kuin modernit käyttöjärjestelmät. RISC-V:stä löytyy sivutusmekanismi ja muut modernien käyttöjärjestelmien edellytykset, joten RISC-V on myös käyttöjärjestelmätason virtualisointiin soveltuva suoritinarkkitehtuuri.

5. YHTEENVETO

Tutkielman tuloksista voidaan vahvistaa, että virtualisointimahdollisuudet on otettu RISC-V-suoritinarkkitehtuurin suunnittelussa perusteellisesti huomioon. Virtualisointitukea ei ole paikattu jälkikäteen, vaan se on ollut RISC-V:n merkittävä suunnittelutavoite alusta alkaen. Arkkitehtuurista on tunnistettavissa monia suunnittelupäätöksiä, jotka on tehty virtualisointia silmällä pitäen.

Merkittävimpänä päätöksenä RISC-V on suunniteltu klassisesti virtualisoitavaksi, mikä on vahvistettavissa arkkitehtuurin spesifikaatiosta. Virtualisoinnin suorituskyvyn kasvattamiseen voidaan käyttää RISC-V:n standardia H-laajennosta, joka lisää laitteistotuen muun muassa muistin virtualisointiin. Yksinkertaisiin sulautettuihin järjestelmiin RISC-V:n PMP-mekanismi on hyvin soveltuva ja yksinkertainen menetelmä osajärjestelmien paravirtualisointiin.

Kaikki suunnittelupäätökset on tehty harkitusti yli 30 vuoden ajalta kerätyn tiedon perusteella. Monet teknisistä päätöksistä ovat tavanomaisia seuraten yksinkertaisesti hyväksi todettuja periaatteita, joten arkkitehtuuri on helppo oppia ja olemassa olevien ohjelmistojen siirtäminen RISC-V-alustalle on mutkatonta. Seurauksena RISC-V soveltuu myös hyvin kaikkien tunnetuimpien ja yleisimpien virtualisointitekniikoiden käyttämiseen.

Toisaalta RISC-V innovoi sen liiketoimintamallilla tai tarkemmin sen puutteella. Arkkitehtuurin vapaus ja avoimuus yhdistettynä modulaarisuuden, virtualisoitavuuden ja muiden teknisten ominaisuuksien kanssa tekevät siitä erinomaisen alustan uusien ideoiden syntymiselle sekä akateemisessa maailmassa että yritysmaailmassa. RISC-V:n ennustetun kasvun syyt ovat selkeästi nähtävissä, joten RISC-V-prosessoreita tullaan varmasti tulevaisuudessa näkemään entistä enemmän kaikissa käyttökohteissa. Laajat virtualisointiominaisuudet tekevät RISC-V:stä erityisen hyvän valinnan palvelinympäristöihin ja sulautettuihin järjestelmiin.

LÄHTEET

[1] K. Adams ja O. Agesen. ”A Comparison of Software and Hardware Techniques for X86 Virtualization”. *SIGARCH Comput. Archit. News* 34.5 (lokakuu 2006), s. 2–13. ISSN: 0163-5964. DOI: 10.1145/1168919.1168860. URL: https://doi-org.libproxy.tuni.fi/10.1145/1168919.1168860.

[2] Amazon Web Services. *Now Available – Compute-Intensive C5 Instances for Amazon EC2*. 2017. URL: https://aws.amazon.com/blogs/aws/now-available-compute-intensive-c5-instances-for-amazon-ec2/ (viitattu 20. 02. 2022).

[3] R. Goldberg. ”Architectural Principles for Virtual Computer Systems”. Tohtorinväitöskirja. Harvard University, 1973.

[4] S. Heath. *Microprocessor Architectures: RISC, CISC and DSP*. eng. Newnes, 2014. ISBN: 0750623039.

[5] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4*. 325462-076US. Joulukuu 2021. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html (viitattu 24. 04. 2022).

[6] J.-H. Kim ja H.-W. Jin. ”Virtio Front-End Network Driver for RTEMS Operating System”. *IEEE Embedded Systems Letters* 12.3 (2020), s. 91–94. DOI: 10.1109/LES.2019.2957570.

[7] P. Kokkinos, D. Kalogeras, A. Levin ja E. Varvarigos. ”Survey: Live Migration and Disaster Recovery over Long-Distance Networks”. *ACM Comput. Surv.* 49.2 (heinäkuu 2016). ISSN: 0360-0300. DOI: 10.1145/2940295. URL: https://doi.org/10.1145/2940295.

[8] J. Osier-Mixon. ”Semico Forecasts Strong Growth for RISC-V” (marraskuu 2019). URL: https://riscv.org/announcements/2019/11/9679/ (viitattu 04. 05. 2022).

[9] D. Patterson. ”Reduced Instruction Set Computers Then and Now”. *Computer* 50.12 (2017), s. 10–12. DOI: 10.1109/MC.2017.4451206.

[10] D. Patterson ja D. Ditzel. ”The Case for the Reduced Instruction Set Computer”. *SIGARCH Comput. Archit. News* 8.6 (lokakuu 1980), s. 25–33. ISSN: 0163-5964. DOI: 10.1145/641914.641917. URL: https://doi-org.libproxy.tuni.fi/10.1145/641914.641917.

[11] N. Penneman, D. Kudinskas, A. Rawsthorne, B. De Sutter ja K. De Bosschere. ”Formal Virtualization Requirements for the ARM Architecture”. *J. Syst. Archit.* 59.3 (maaliskuu 2013), s. 144–154. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2013.02.003. URL: https://doi.org/10.1016/j.sysarc.2013.02.003.

[12] G. Popek ja R. Goldberg. ”Formal Requirements for Virtualizable Third Generation Architectures”. *Commun. ACM* 17.7 (heinäkuu 1974), s. 412–421. ISSN: 0001-0782. DOI: 10.1145/361011.361073. URL: https://doi.org/10.1145/361011.361073.

[13] E. Reshetova, J. Karhunen, T. Nyman ja N. Asokan. ”Security of OS-Level Virtualization Technologies”. Teoksessa: *Secure IT Systems*. Toim. K. Bernsmed ja S. Fischer-Hübner. Cham: Springer International Publishing, 2014, s. 77–93. ISBN: 978-3-319-11599-3.

[14] RISC-V International. *RISC-V Website*. URL: https://riscv.org/ (viitattu 17. 03. 2022).

[15] B. Sá, J. Martins ja S. Pinto. *A First Look at RISC-V Virtualization from an Embedded Systems Perspective*. 2021. arXiv: 2103.14951 [cs.AR].

[16] S. G. Soriga ja M. Barbulescu. ”A comparison of the performance and scalability of Xen and KVM hypervisors”. Teoksessa: *2013 RoEduNet International Conference 12th Edition: Networking in Education and Research*. 2013, s. 1–6. DOI: 10.1109/RoEduNet.2013.6714189.

[17] A. Tanenbaum ja H. Bos. *Modern Operating Systems*. 4. painos. Boston, MA: Pearson, 2014. ISBN: 978-0-13-359162-0.

[18] S. K. Tesfatsion, C. Klein ja J. Tordsson. ”Virtualization Techniques Compared: Performance, Resource, and Power Usage Overheads in Clouds”. eng. Teoksessa: *ICPE 2018 - Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. Vol. 2018-. ICPE ’18. ACM, 2018, s. 145–156. ISBN: 9781450350952.

[19] VMware. *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*. Tekninen raportti. 2007. URL: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf (viitattu 07. 04. 2022).

[20] A. Waterman. *Design of the RISC-V Instruction Set Architecture*. Tekninen raportti. EECS Department, University of California, Berkeley, tammikuu 2016. URL: https://people.eecs.berkeley.edu/~krste/papers/EECS-2016-1.pdf (viitattu 06. 02. 2022).

[21] A. Waterman ja K. Asanović. *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA*. Tekninen raportti. EECS Department, University of California, Berkeley, joulukuu 2019. URL: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf (viitattu 22. 01. 2022).

[22] A. Waterman, K. Asanović ja J. Hauser. *The RISC-V Instruction Set Manual Volume II: Privileged Architecture*. Tekninen raportti. EECS Department, University of California, Berkeley, joulukuu 2021. URL: https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf (viitattu 22. 01. 2022).