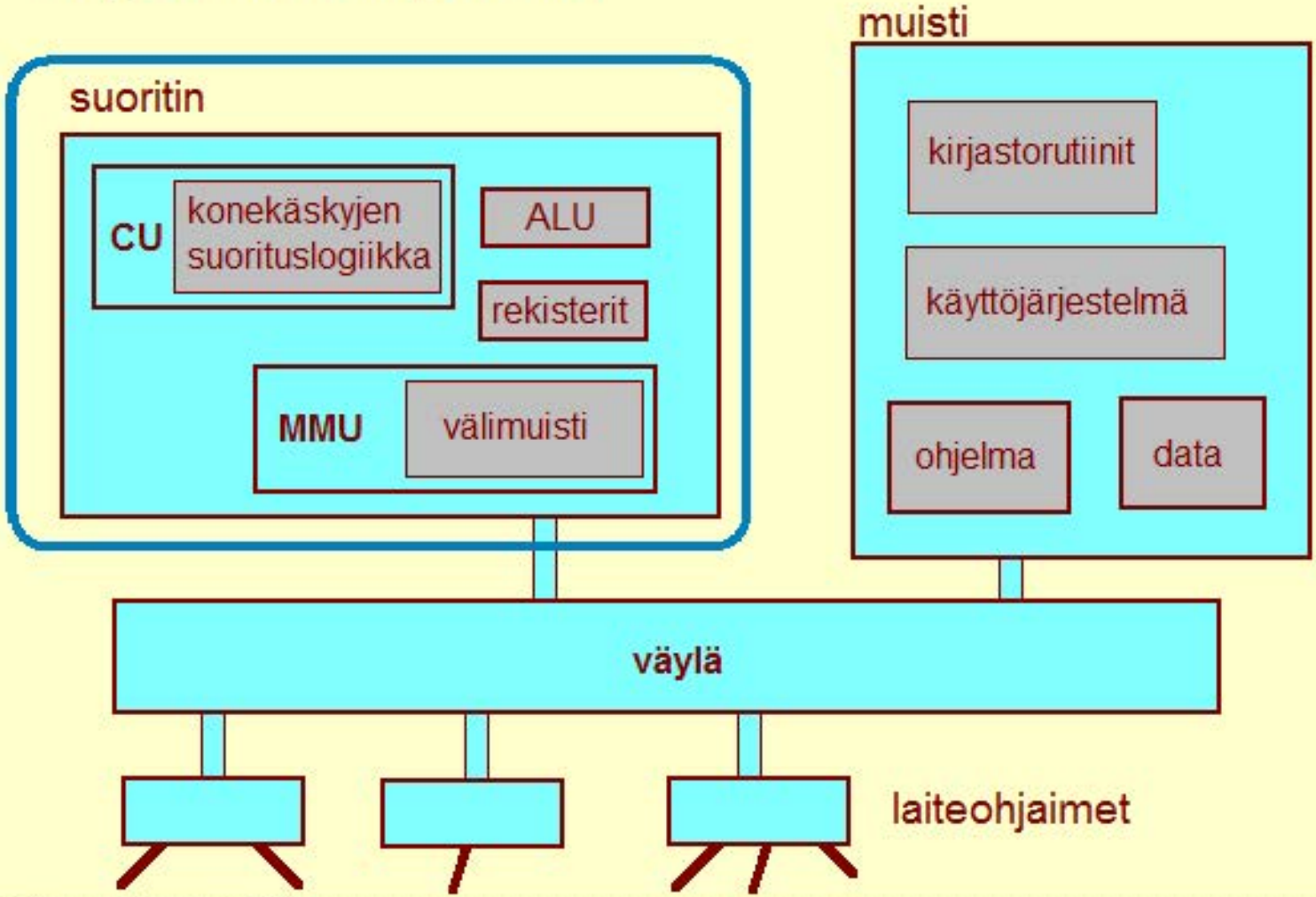
Suoritin ja väylä

- Suorittimen rakenne
- Väylän rakenne
- Käskyjen suoritussykli
- MMU'n toiminta ja osoitteenmuunnos
- Poikkeukset ja keskeytykset
- Suorittimen tilat
- Ttk-91 simulaaattorin rakenne

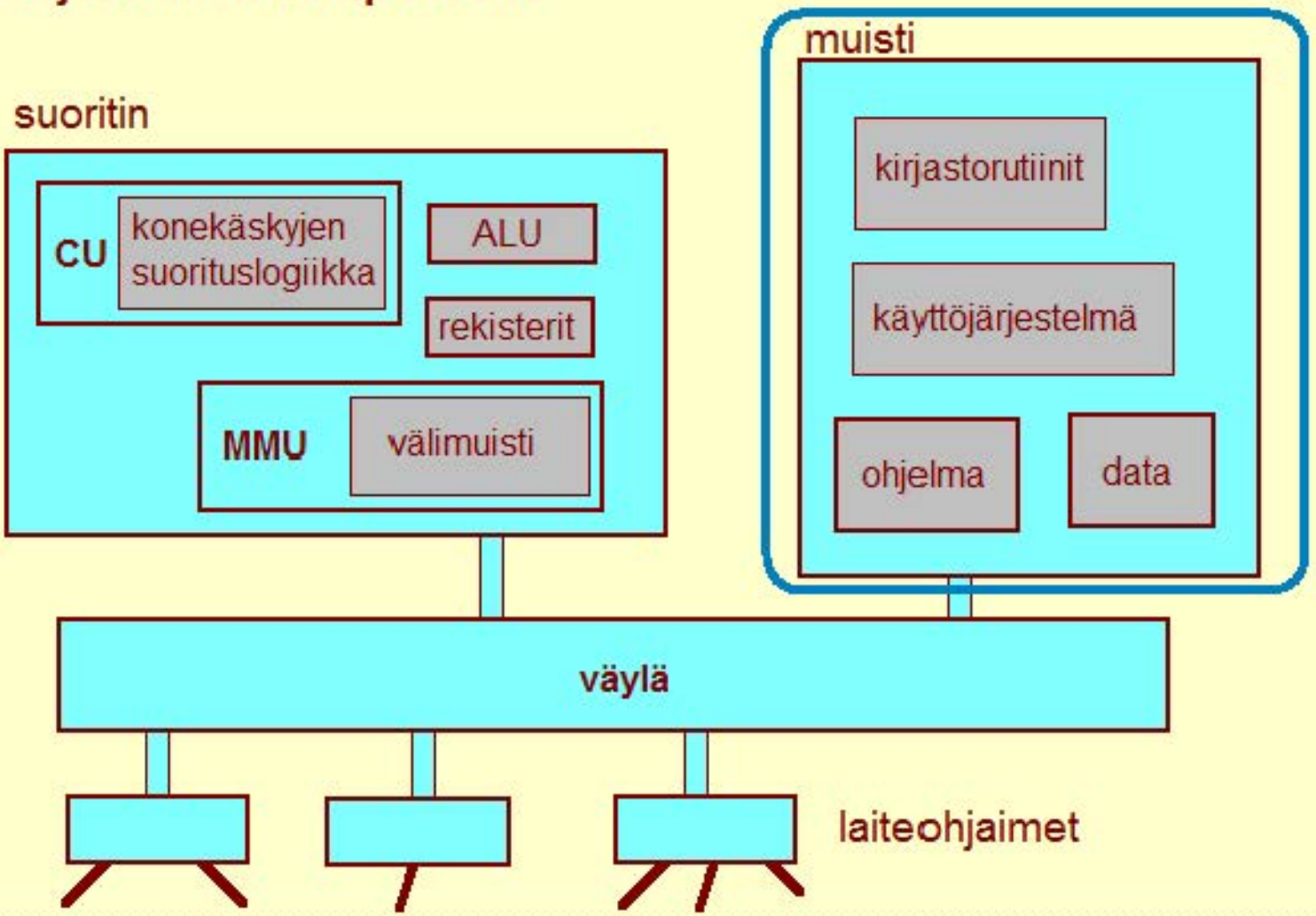
Tässä luennossa tutustutaan suorittimen rakenteeseen käskyjen suoritussyklin tasolla ja väylän rakenteeseen yleisellä tasolla. Esittelemme, kuinka ohjelman käyttämät osoitteet eroavat muistipiirien käyttämistä osoitteista ja kuinka MMU suorittaa tähän liittyvää osoitteenmuunnosta. Käymme myös läpi keskeytysten ja poikkeusten käsittelyn käskyjen suoritussyklin ja keskeytyskäsittelijöiden avulla. Suorittimen erilaiset suoritustilojen toteutus ja tilasiirtymät niiden välillä käsitellään seuraavaksi. Lopuksi esittelemme ttk-91 simulaattorin yleisen toimintaperiaatteen ja käymme läpi, kuinka ttk-91 simulaattorissa toteutetaan käskyjen suoritusta simuloiva käskyjen suoritussykli.

Järjestelmän komponentit



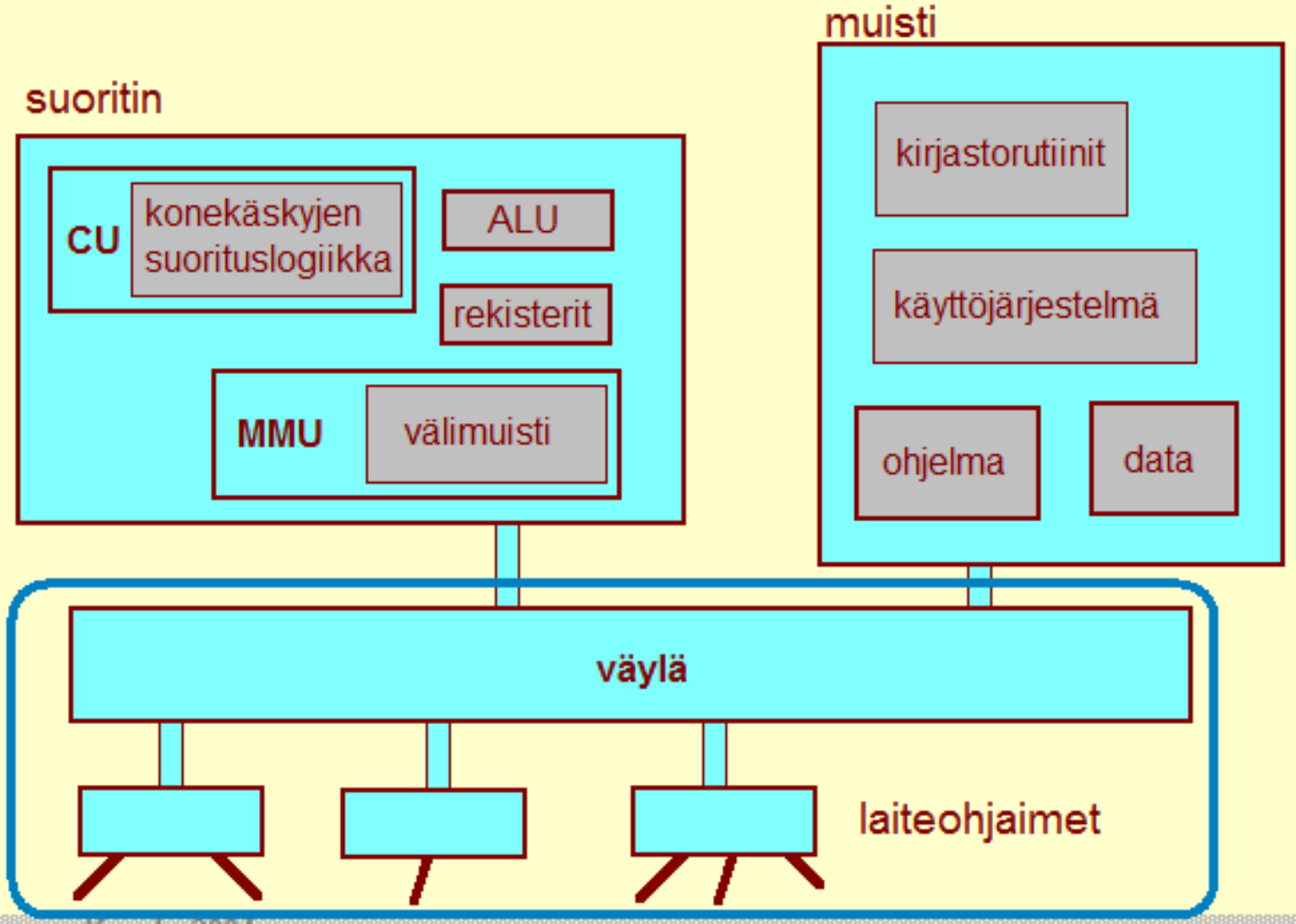
Keskitymme tässä luvussa pääasiassa suorittimen toimintaan, vaikka käymmekin pääpiirteissään läpi myös muistin ja väylän rakennetta. Suorittimessa on muutama suurempi rakenneyksikkö. Muistinhallintayksikkö (MMU) käsittelee kaikki muistiviitteet ja pitää yllä välimuistia. ALU sisältää kaikki matemaattisten operaatioiden toteutuspiirit - kaikki varsinainen työ tehdään siellä. Rekisterit ovat pieni joukko nopeata muistia, mikä on tarpeeksi nopeata ALU'n kanssa operoimiseen. Kontrolliyksikkö on suorittimen 'aivot'. Se kertoo joka kellopulssilla, mitä sillä hetkellä suorittimella sen kellopulssin aikana tapahtuu.

Järjestelmän komponentit



Keskusmuisti on homogeeninen, suht'koht hidas muistialue, jossa sijaitsee kaikki ohjelman suorituksensa aikana tarvitsema ohjelmakoodi ja data. Ohjelmalla ei ole käytössään koko muistia, vaan ainoastaan pieni osa siitä. Muistissa sijaitsee myös käyttöjärjestelmä ja useitakin erilaisia yleiskäyttöisiä ohjelma- ja data-kirjastoja. Siellä sijaitsee myös muiden 'samaan aikaan' suorituksessa olevien ohjelmien koodi- ja data-alueet, mutta unohdamme niiden olemassaolon toistaiseksi.

Järjestelmän komponentit

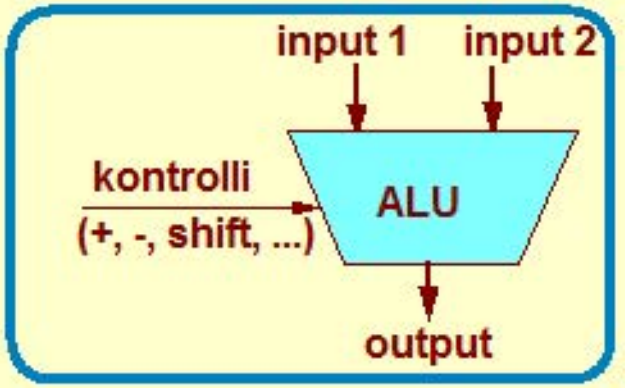


Väylä on suuri joukko johtoja ja niiden käyttöä ohjaavaa kontrollilogiikkaa. Väylä yhdistää suorittimen muistiin ja kaikki tieto suorittimen ja muistin välillä kulkee väylän kautta. Väylä yhdistää myös suorittimen laiteohjaimiin, jotta suoritin voi antaa laiteohjaimille komentoja niihin kytkettyjen laitteiden lukemista ja kirjoittamista varten. Laiteohjaimien toimintaa käsitellään myöhemmällä luennolla.

Ttk-91 suorittimen rakenne

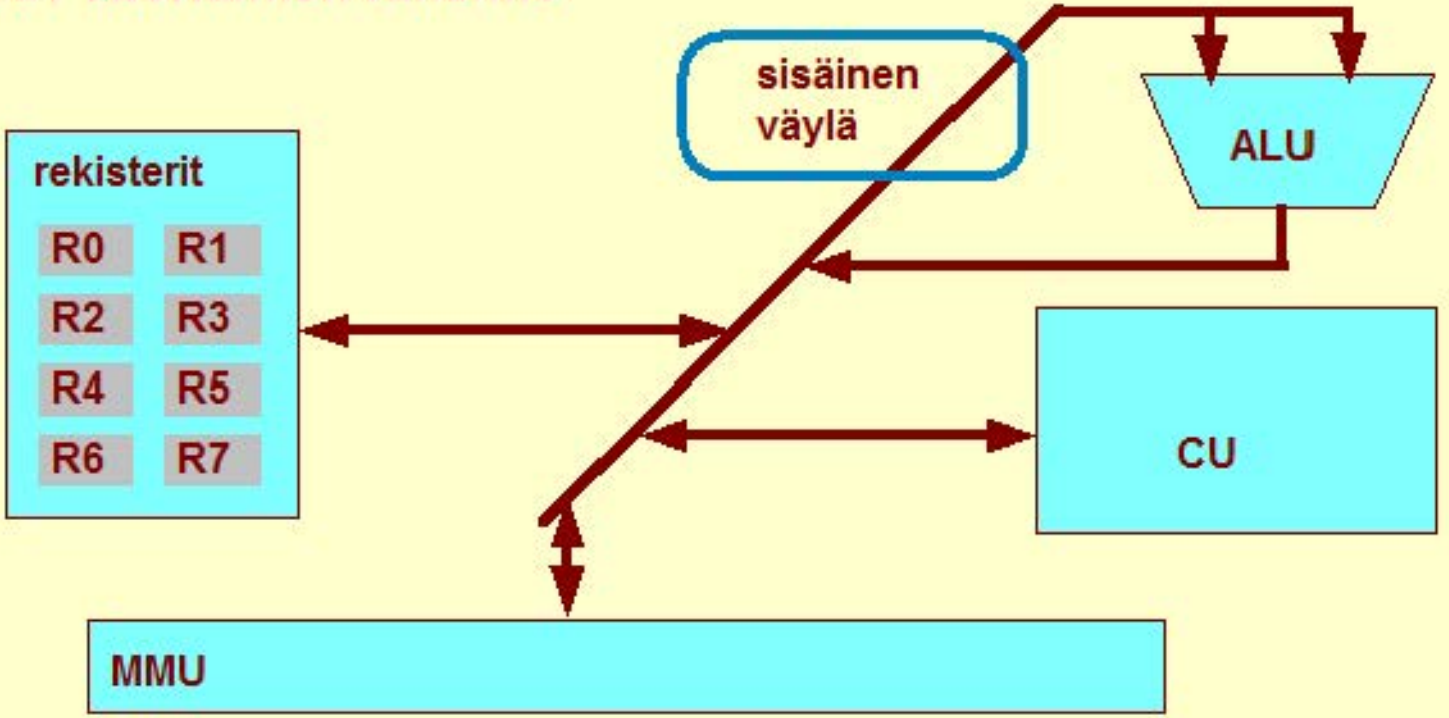
Tarkastellaan nyt suorittimen rakennetta vähän tarkemmin. Suorittimen sisällä on siis useita erillisiä komponentteja, jotka pelaavat hienosti yhteen. Suorittimen sisäinen konekäskyissä viitattavissa oleva muisti koostuu rekistereistä, joita ttk-91'ssä on 8 kappaletta. Todellisessa koneessa niitä on vähän enemmän, mutta vain kymmeniä, ei satoja. Jos rekistereitä on kovin paljon, niiden osoittamiseen konekäskyissä tarvitaan paljon bittejä ja ohjelman koosta tulee iso. Toisaalta on myös teknologisia rajoituksia: pieni rekisterijoukko on nopeampi kuin suuri. Suuri mutta hidas rekisterijoukko voi vaatia koko muun laitteiston hidastamista samalle tasolle.

Ttk-91 suorittimen rakenne



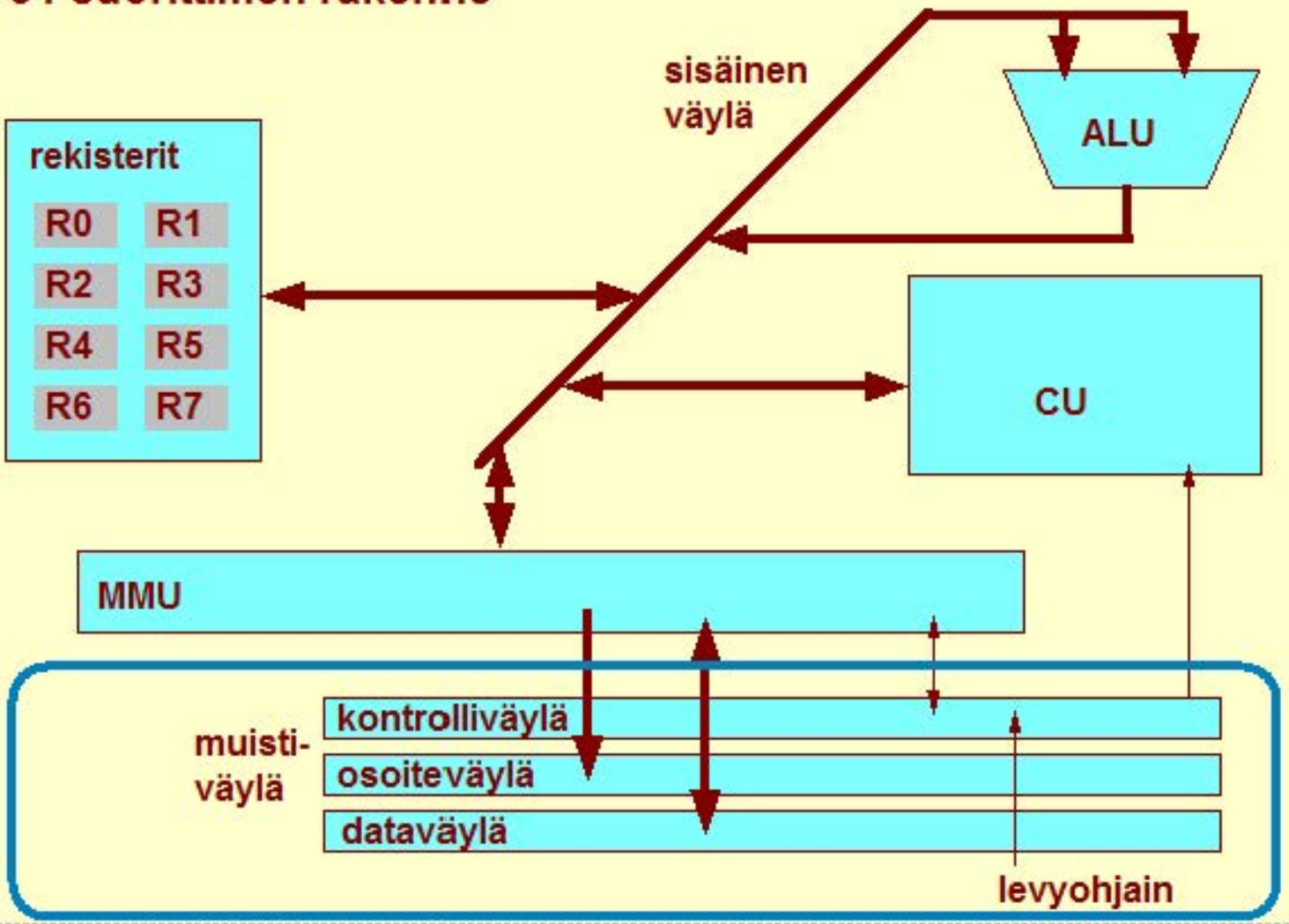
ALU tekee kaiken työn laitteistossa. Kaikki dataa muovaava työ tehdään ALUssa. ALUssa on omat piirinsä jokaiselle erilaiselle operaatiolle, kuten esimerkiksi kokonaislukujen yhteenlaskulle, kertolaskulle, bittien siirrolle vasemmalle, jne. ALUlla on kaksi sisäänmenoa, yksi ulostulo ja kontrollivalinta, jonka avulla tällä kertaa suoritettava laskentapiiri valitaan. ALU toimii siten, että sisäänmenoihin annetaan halutut arvot ja kontrollilla valitaan operaatio. Sitten odotellaan piirien toiminta-aika ja operaation tulos on valmiina johonkin talletettavaksi.

Ttk-91 suorittimen rakenne



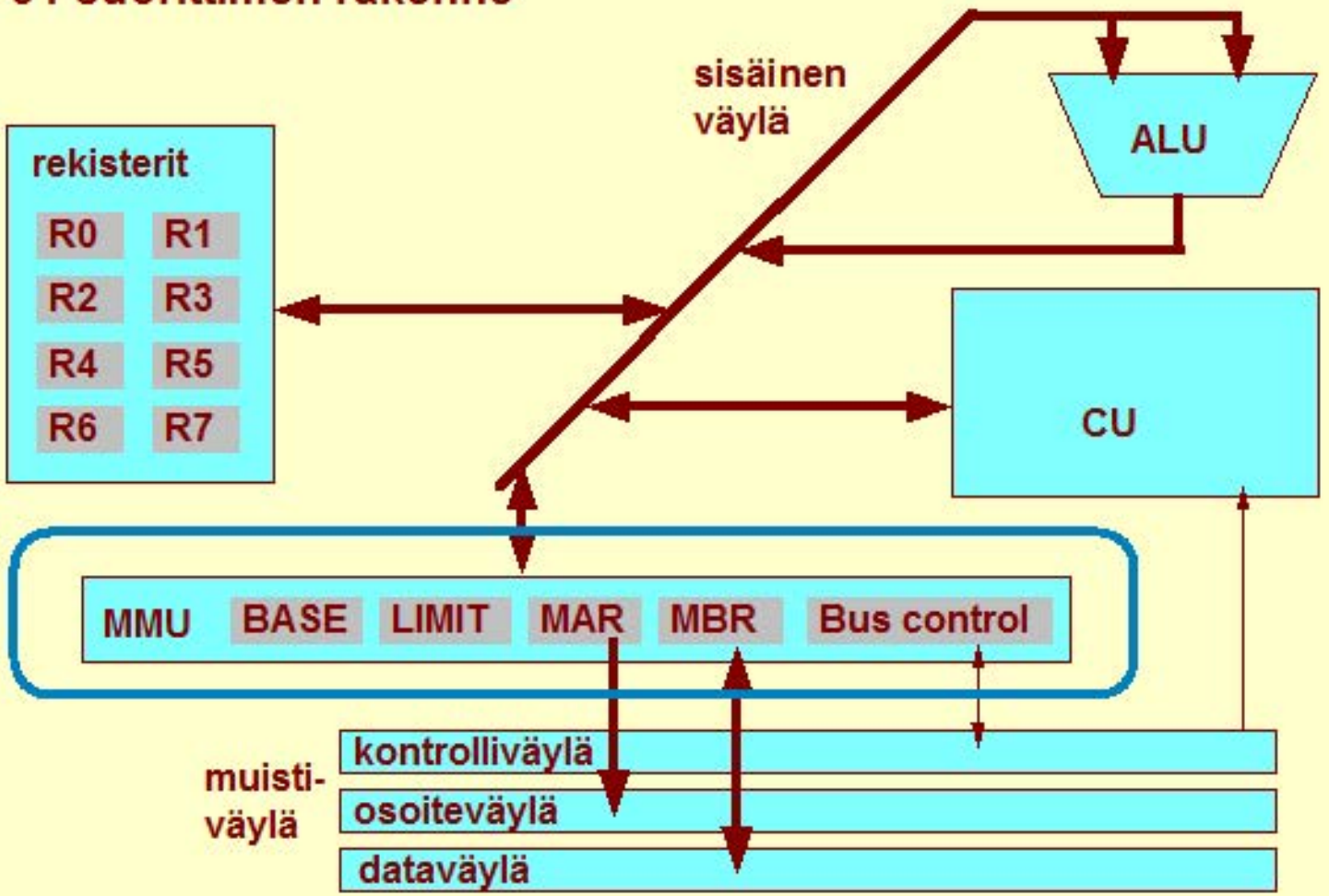
Suorittimen sisällä on sen sisäinen väylä datan siirtämiseksi komponentilta toiselle. Väylään kuuluu monta rinnakkaista johdinta, joten se vie usein merkittävän määrän lastun pinta-alasta. Nykyaikaisissa koneissa väylä ei ole näin yksinkertainen, vaan siihen kuuluu oikeasti useita erilaisia osaväyliä, jotka hyvin heterogeenisella tavalla yhdistävät suorittimen eri komponentit. Toisaalta, on olemassa oikeitakin suorittimia, joiden sisäinen väylä on vähän tämän kaltainen. Sisäisessä väylässä on usein ylimääräisiä piuhoja, joiden avulla voidaan valvoa tiedon muuttumattomuutta suorittimen sisäisissä tiedonsiirroissa. Tästä myöhemmin lisää.

Ttk-91 suorittimen rakenne



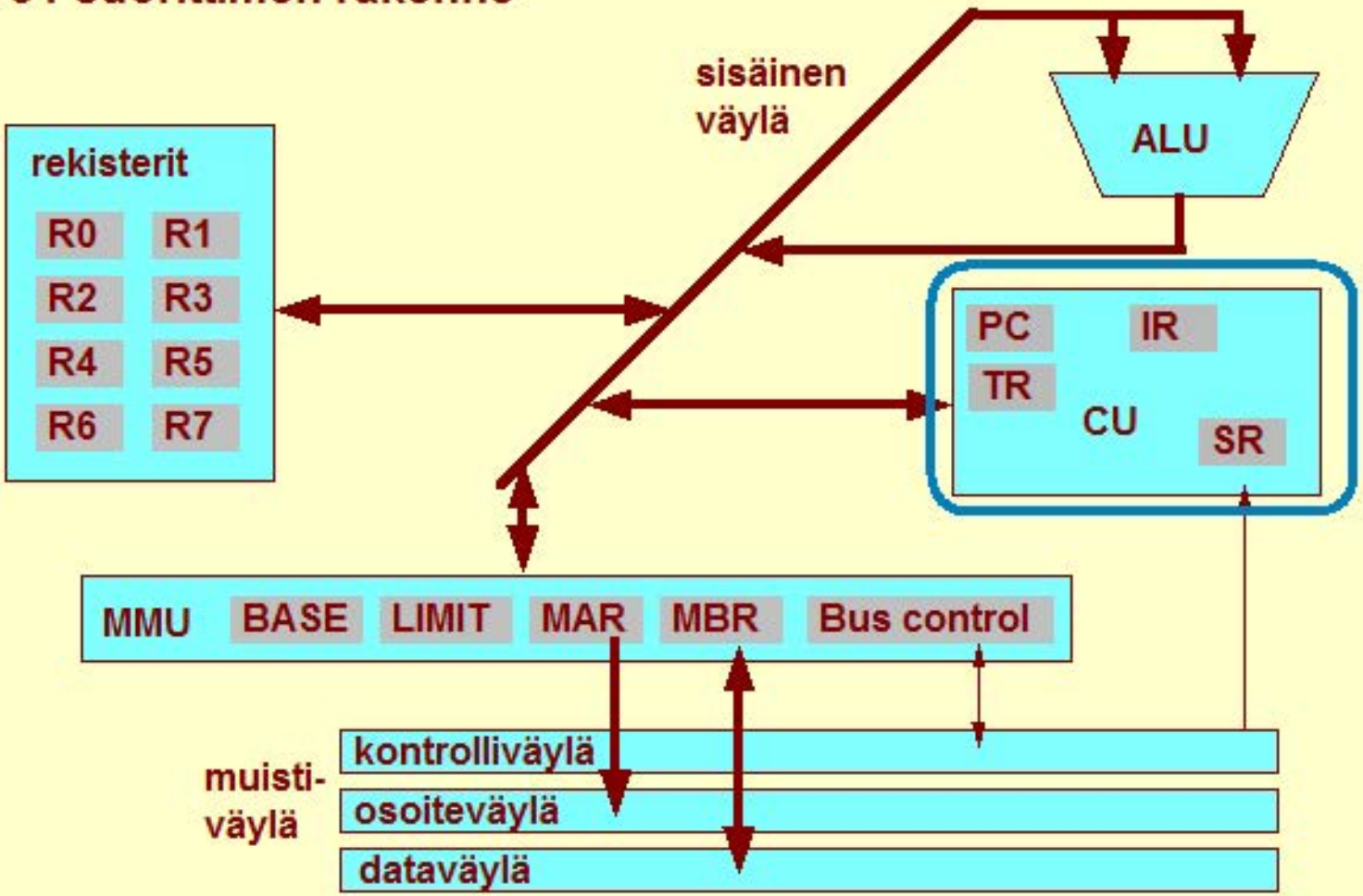
Muistiväylässä on oikeastaan kolme komponenttia. Varsinainen tiedonsiirto tapahtuu dataväylän kautta ja siinä on esimerkiksi 32 tai 64 rinnakkaista piuhaa. Osoitteet välitetään ns. osoiteväylän kautta, jolla on omat 32 tai 64 johdintaan. Joissakin laitteistoissa osoite- ja dataväylät on yhdistetty tilan säästämiseksi suorituskyvyn kustannuksella. Kontrolliväylän avulla huolehditaan sekä itse väylän varaamisesta että kontrollitiedon välittämisestä muille laitteille (esim. muistille) ja/tai muilta laitteilta (esim. levyohjaimelta). Kontrolliväylän koko voi olla esimerkiksi 10 tai 20 johdinta.

Ttk-91 suorittimen rakenne



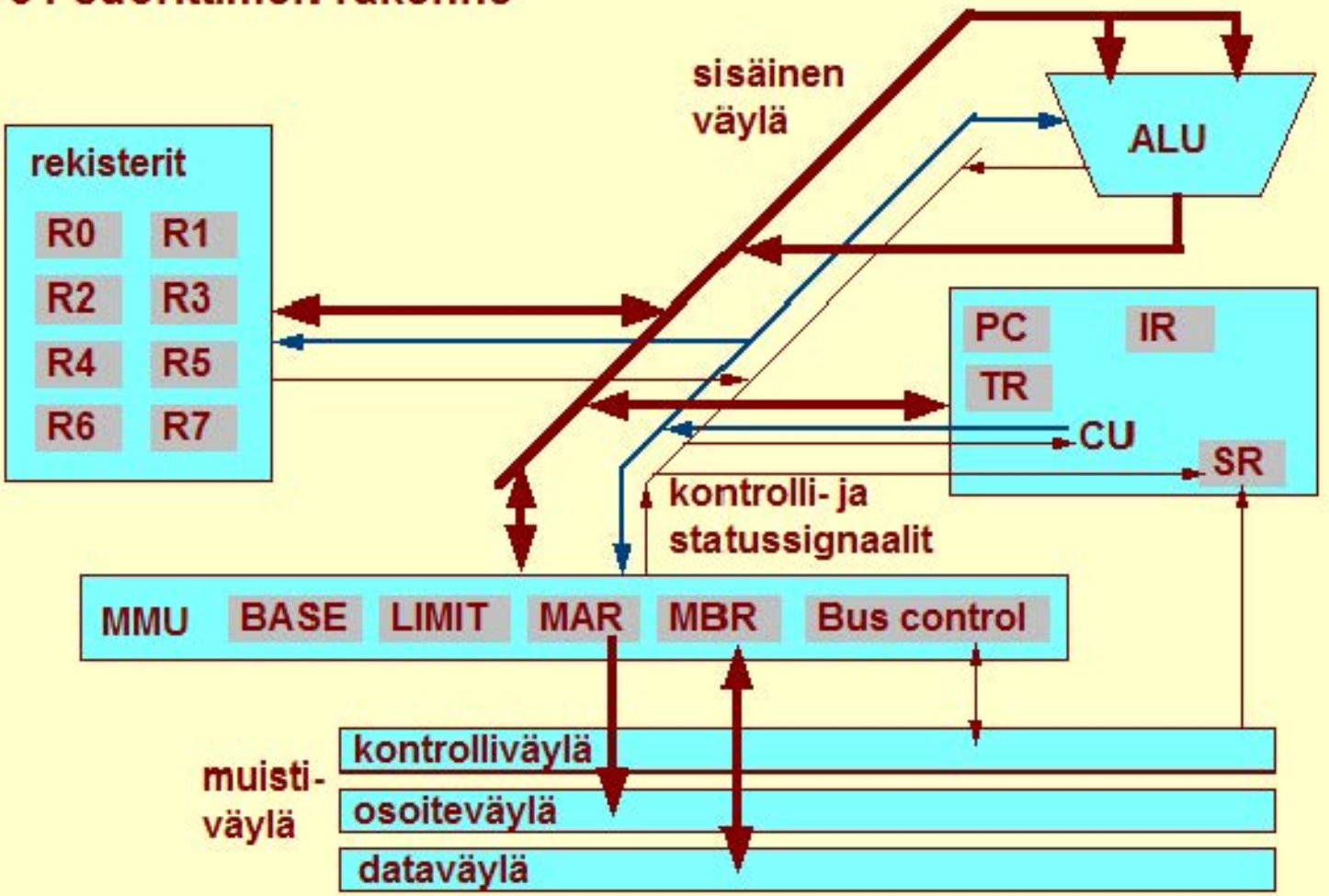
Muistinhallintayksikkö pitää kirjaa ohjelman käytössä olevasta muistialueesta BASE ja LIMIT rekistereiden avulla. Muistista luku tapahtuu siten, että luettavan muistipaikan osoite laitetaan MAR rekisteriin, josta se 'automaattisesti' kirjoittuu osoiteväylälle. Kontrolliväylälle annetaan lukusignaali ja sitten odotellaan. Vähän ajan kuluttua muistipiiri havaitsee lukupyynnön ja tutkii osoiteväylää. Siellä on muistiosoite, jonka osoittamasta muistipaikasta haetaan arvo ja kirjoitetaan se dataväylälle. Dataväylälle kirjoitettu arvo tulee sitten näkyviin MBR rekisteriin, josta se voidaan ottaa käyttöön.

Ttk-91 suorittimen rakenne



Kontrolliyksikkö ohjaa tätä kaikkea. Se antaa yksityiskohtaiset ohjeet jokaiselle komponentille jokaisella kellopulssilla. Nykyinen konekäsky on käskyrekisterissä (IR). IR'ssä on konekäskyn eri kentät valmiiksi eroteltuina, joten ne on helppo sieltä lukea. Tilarekisterissä (SR) näkyy kaikki käsittelemättömät virhetilanteet ja muilta laitteilta tulleet signaalit. Kontrolliyksikössä on ylimääräinen rekisteri TR kätevänä työtilana. Todellisissa suorittimissa on paljon erilaisia työtilarekistereitä nopeuttamassa laskentaa. Seuraavan konekäskyn osoite on rekisterissä PC.

Ttk-91 suorittimen rakenne



Suorittimen sisällä on huomattava joukko erilaisia kontrolli- ja statusjohtimia. Niiden avulla kontrolliyksikkö käskyttää kutakin eri komponenttia. Kontrollisignaalilla esimerkiksi välitetään tieto rekisterijoukolle, että sijoitapas sisäiselle väylälle rekisterin R1 arvo. Tai esimerkiksi kerrotaan ALU'lle, että teepäs nyt yhteenlasku. ALU voi myös tilajohtimien avulla kertoa kontrolliyksikölle, että ALUssa tapahtui nollalla jako. MMU voi signaloida epäkelvosta muistiosoitteesta ja kontrolliyksikkö voi käskyttää MMU'ta tekemään muistista luvun tai sinne kirjoittamisen.

Todellisia suoritinesimerkkejä

Ensimmäinen nykyaikainen tietokone, jossa ohjelma- ja datamuisti ovat samassa muistissa

EDSAC

(http://www.dcs.warwick.ac.uk/~edsac/Gallery/Gallery005.html, 29.11.2004)

Mikropiirillä toteutetut suorittimet: ensimmäinen Intel i4004 ja modernimpi i960 Cobra

i4004

i960 Cobra

(http://micro.magnet.fsu.edu/chipshots/intel/i4004large.html, 29.11.2004)

(http://micro.magnet.fsu.edu/chipshots/intel/i960large.html, 29.11.2004)

Esimerkki suorittimesta, jonka suunnittelijoilla oli liikaa resursseja käytössään

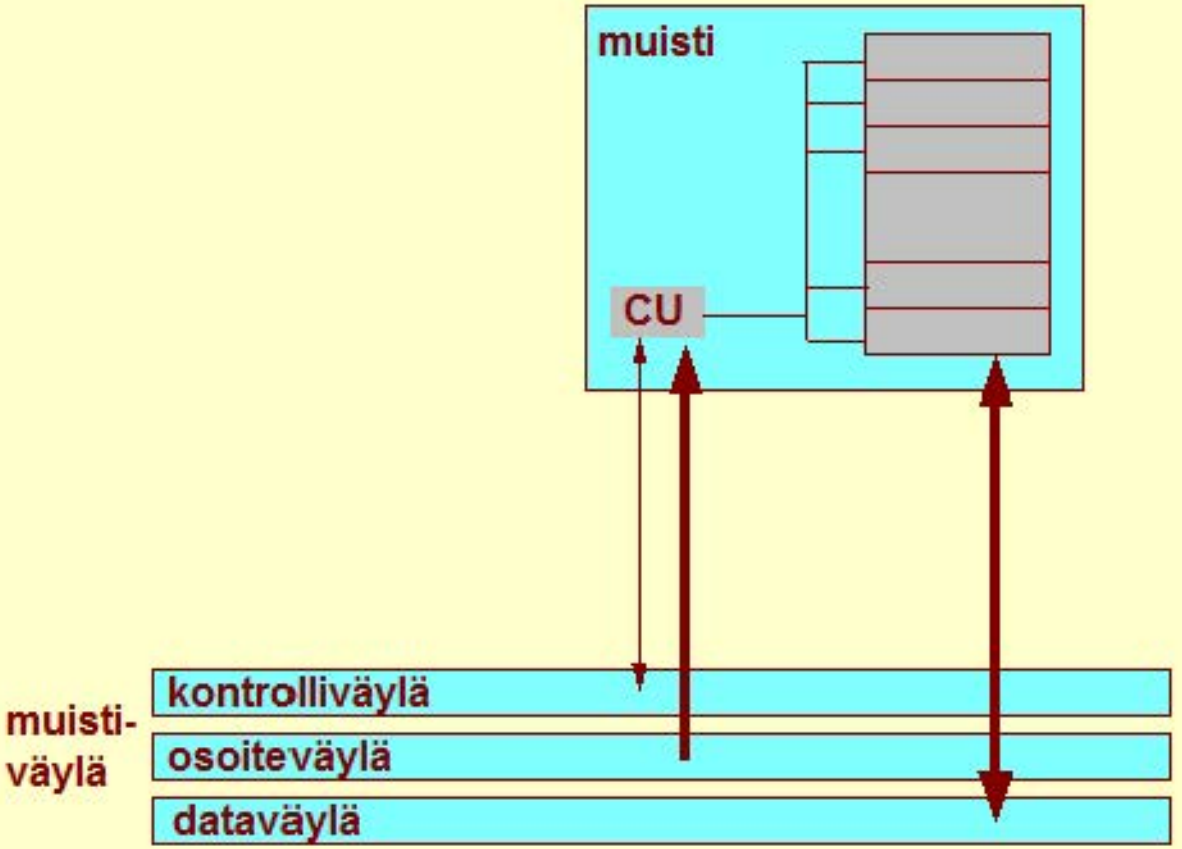
Excalibur

(http://micro.magnet.fsu.edu/creatures/pages/g3sword.html, 29.11.2004)

(Huomaa, että muualla verkossa olevien materiaalien tekijänoikeudet ovat kyseisten sivustojen haltijoilla)

Todelliset suorittimet toteutetaan samanlaisina kuin ttk-91'kin. Varhaisissa koneissa (esim. EDSAC) rekisteri saattoi olla nykyisen jääkaapin kokoinen, ja johtimet olivat kunnon sähköjohtimia. Lastuille toteutetuissa koneissa sekä laskentapiirit että niitä yhdistävät johtimet on toteutettu mikropiireillä. Esimerkkinä tästä on ensimmäinen yhdelle lastulle suunniteltu suoritin, Intelin i4004 tai modernimpi i960 Cobra. Joskus lastuihin jää jopa hukkatilaa, jota huumorintajuiset suunnttelijat sitten voivat hyödyntää. Excalibur-suorittimessa on lastulle piirretty Excalibur-miekka, jolla ei varmastikaan ole mitään funktionaalista merkitystä.

Ttk-91 muistin rakenne



Muistin rakenne on huomattavsti yksioikoisempi. Muistipiirissä ei ole kuin yksinkertainen kontrolliyksikkö, joka tulkitsee väylän kautta tulleita muistin luku- ja kirjoituspyyntöjä. Jos kyseessä on lukupyyntö, niin otetaan muistipaikan osoite osoiteväylältä ja haetaan kyseisen muistipaikan sisältö ja kirjoitetaan se dataväylälle. Kirjoituspyynnön kohdalla luetaan osoitteen lisäksi myös dataväylältä arvo ja talletetaan se kyseiseen muistipaikkaan. Tarkempi esitys väylän ja muistin toiminnasta annetaan sitten seuraavalla luentokurssilla Tietokoneen rakenne.

Käskyjen nouto ja suoritussykli

Hae PC'n osoittama konekäsky muistista

- lisää samalla PC'n arvoa yhdellä

Suorita konekäsky

- muuta jonkun rekisterin ja/tai muistipaikan arvoa
- jos (ehdollinen) hyppykäsky, niin PC'n arvo voi muuttua

Konekäskyn suorituksen ensimmäinen vaihe on itse käskyn hakeminen muistista. Suorittimellahan on rekisteri PC juuri tätä tarkoitusta varten, joten tässä vaiheessa pyydetään siis MMU'ta tekemään muistista luku PC'n osoittamasta muistipaikasta. Samassa yhteydessä PC'n arvoa kasvatetaan yhdellä, koska oletusarvoisesti tämän käskyn jälkeen seuraava konekäsky löytyy tämän konekäskyn jälkeisestä muistipaikasta. Tämä vaihe koostuu siis yksinkertaisesta muistista luvusta.

Käskyjen nouto ja suoritussykli

Hae PC'n osoittama konekäsky muistista

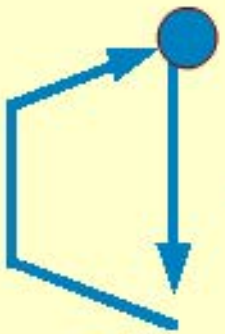
- lisää samalla PC'n arvoa yhdellä

Suorita konekäsky

- muuta jonkun rekisterin ja/tai muistipaikan arvoa
- jos (ehdollinen) hyppykäsky, niin PC'n arvo voi muuttua

Toisessa vaiheessa juuri äsken muistista haettu konekäsky suoritetaan. Tutkailemme siis konekäskyn operaatiokoodia ja toteutamme sen mukaisen operaation käskyssä mainituille operandeille. Tämäkin on aika yksinkertainen tehtävä, koska kaikki mahdolliset operaatiot on jo valmiiksi toteutettuina kontrolliyksikön piireissä. Useimmat konekäskyt muuttavat suoritettaessa jonkin rekisterin tai muistipaikan arvoa, mutta erilaiset hyppykäskyt voivat myös vaihtaa seuraavaksi suoritettavan konekäskyn osoitetta PC'ssä.

Käskyjen nouto ja suoritussykli



Hae PC'n osoittama konekäsky muistista

- lisää samalla PC'n arvoa yhdellä

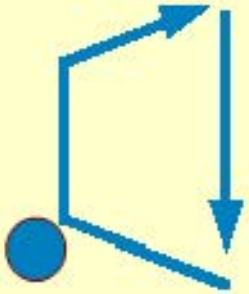
Suorita konekäsky

- muuta jonkun rekisterin ja/tai muistipaikan arvoa
- jos (ehdollinen) hyppykäsky, niin PC'n arvo voi muuttua

Tässä on kaikki, muuta ei ole!

Ehkä kaikkein vaikein asia suorittimen toiminnan ymmärtämisessä on hoksata, että tässä on kaikki. Seuraavaksi mennään siis jälleen hakemaan seuraavaksi suoritettavaa konekäskyä jne. Tällä tavoin konekäskyjä suoritetaan yksi kerrallaan, halutussa järjestyksessä. Käskyjen suorittaminen alkaa, kun pöytäkone laitetaan päälle, ja loppuu virran katkettua. Sylimikroissa ja muissa kannettavissa laitteissa on erilaisia virransäästötiloja, mutta näitä ei käsitellä tällä kurssilla.

Käskyjen nouto ja suoritussykli



Hae PC'n osoittama konekäsky muistista

- lisää samalla PC'n arvoa yhdellä

Suorita konekäsky

- muuta jonkun rekisterin ja/tai muistipaikan arvoa
- jos (ehdollinen) hyppykäsky, niin PC'n arvo voi muuttua

Suoritin ei tiedä mitään ohjelmista!

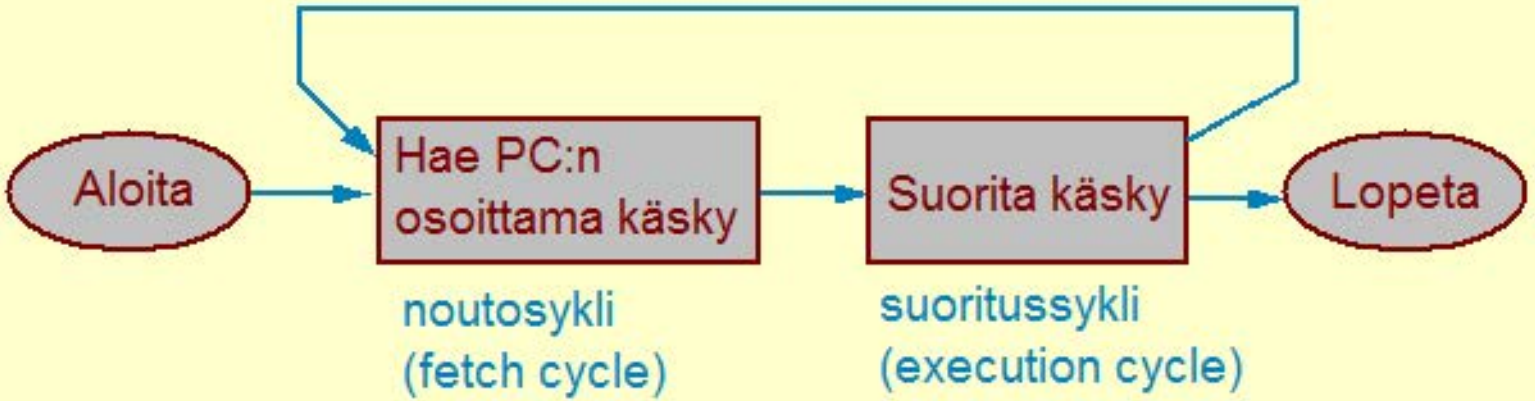
Merkittävää käskyjen suorittamisessa on myös se, että laitteisto siis vain suorittaa konekäskyjä yksi kerrallaan. Se ei näe mitään konekäskyä suurempia kokonaisuuksia. Esimerkiksi, suoritin ei tiedä mitään ohjelmista. Tietokoneohjelmat ovat ihmisen muovaamia ajatusrakennelmia, jotka on toteutettu suoritettavien konekäskyjen joukkona. Ohjelmalla on merkitystä ihmiselle, mutta kone ei siitä välitä. Samalla tavalla voisi ajatella auton moottoria. Moottori vain pyörii eri nopeuksilla ihmisen antamien ohjeiden perusteella, eikä sille ole mitään väliä, miten moottoria käytetään auton liikuttamiseen.

Systeemin tila

Käskyn suoritus muuttaa systeemin tilaa

- ulkoinen, konekäskyssä viitattavissa oleva rekisteri (esim. R3 tai SR)
- sisäinen rekisteri (esim. TR, PC tai SR'n Z- tai O-bitti)
- muisti (esim. Mem[36])
- ulkoinen laite (esim. levyohjaimen datarekisteri)

Käskyjen nouto- ja suoritussykli



Yhden konekäskyn suoritus näkyy systeemissä siinä, että jotain muutetaan! Jopa NOP-käsky tekee siis jotain: sen nouto ja suoritus -syklin aikana PC:n arvo on kasvanut yhdellä. Useimmat käskyt tietenkin tekevät jotain enemmän hyödyllistä. Aikaisemmin mainittujen rekistereiden tai muistin arvon muuttamisen lisäksi ne voivat muuttaa tilarekisterin bittejä tai jonkin laiteohjaimen kontrollitietoja kyseisen laitteen ohjaamiseksi. Tärkeätä on huomata, että konekäsky ei koskaan tee mitään yllättävää, vaan kaikki toiminnat ovat täysin deterministisiä. Se, että laite ihmisen mielestä ei joskus toimi oikein, ei koskaan ole suorittimen vika!

Ttk-91 konekäskyn rakenne

Käskyn esitys bittitasolla on aina samantyyppinen

(bitit on numeroitu oikealta vasemmalle, nollasta alkaen)

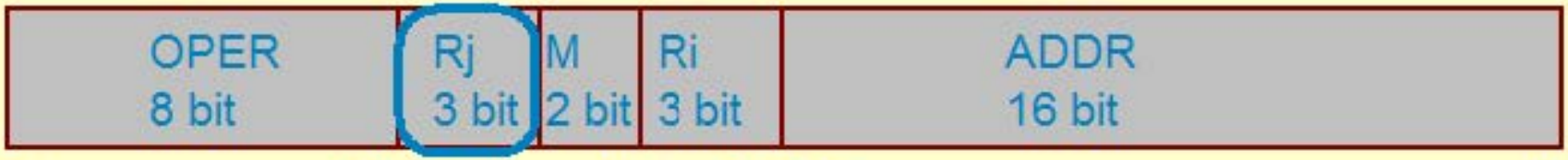
OPER operaatiokoodi, opcode, 0-112 (kaikki arvot eivät ole laillisia)
Rj 1. operandi, tulosrekisteri, R0-R7 (yleisrekisteri)
Ri indeksirekisteri, R1-R7 (arvo 0 indikoi että indeksirekisteri ei ole käytössä)
ADDR osoiteosa, vakio-osa, arvoalue [-32767, +32767]
M moodi, muistinoutojen määrä toisen operandin laskemiseksi (ennen mahdollista muistiin talletusta STORE-käskyssä)

- 0 välitön osoitus (STORE-käsky: suora osoitus)
- 1 suora osoitus (STORE-käsky: epäsuora osoitus)
- 2 epäsuora osoitus (STORE-käsky: epäkelpo arvo, virhetilanne)
- 3 epäkelpo arvo, virhetilanne

Ttk-91 koneen konekäskyt ovat aina samanlaisia rakenteeltaan. Joissakin todellisissa koneissa voi olla useita konekäskyn rakennetyyppejä, mutta tämä monimutkaistaa käskyjen noutoa muistista ja niiden tulkintaa käskyjen suoritusaikana. Ttk-91 koneen konekäskyssä ensimmäisenä on 8 bitin operaatiokoodi, jossa jokaiselle eri konekäskylle on varattu oma numeronsa. Esimerkiksi ADD on 17 ja CALL on 49. Loput operaatiokoodit löytyvät kurssin verkkomateriaalista. Kaikki 256 arvoa eivät ole käytössä. Käyttämättömiä operaatiokoodeja voi myöhemmin ottaa käyttöön uusien konekäskyjen toteutuksen yhteydessä. Esimerkiksi, konekäsky SHRA (opcode 27) lisättiin arkkitehtuuriin jälkeenpäin.

Ttk-91 konekäskyn rakenne

Käskyn esitys bittitasolla on aina samantyyppinen



(bitit on numeroitu oikealta vasemmalle, nollasta alkaen)

OPER **operaatiokoodi, opcode, 0-112 (kaikki arvot eivät ole laillisia)**
Rj **1. operandi, tulosrekisteri, R0-R7 (yleisrekisteri)**
Ri **indeksirekisteri, R1-R7 (arvo 0 indikoi että indeksirekisteri ei ole käytössä)**
ADDR **osoiteosa, vakio-osa, arvoalue [-32767, +32767]**
M **moodi, muistinoutojen määrä toisen operandin laskemiseksi (ennen mahdollista muistiin talletusta STORE-käskyssä)**

- 0 välitön osoitus (STORE-käsky: suora osoitus)
- 1 suora osoitus (STORE-käsky: epäsuora osoitus)
- 2 epäsuora osoitus (STORE-käsky: epäkelpo arvo, virhetilanne)
- 3 epäkelpo arvo, virhetilanne

Ensimmäinen operandi ja tulosrekisteri Rj on seuraavassa 3 bitin kentässä. Kolmella bitillä voi ilmaista positiiviset luvut 0-7, jotka kaikki ovat käytössä, vastaten rekistereitä R0-R7.

Ttk-91 konekäskyn rakenne

Käskyn esitys bittitasolla on aina samantyyppinen

(bitit on numeroitu oikealta vasemmalle, nollasta alkaen)

OPER operaatiokoodi, opcode, 0-112 (kaikki arvot eivät ole laillisia)
Rj 1. operandi, tulosrekisteri, R0-R7 (yleisrekisteri)
Ri indeksirekisteri, R1-R7 (arvo 0 indikoi että indeksirekisteri ei ole käytössä)
ADDR osoiteosa, vakio-osa, arvoalue [-32767, +32767]
M moodi, muistinoutojen määrä toisen operandin laskemiseksi (ennen mahdollista muistiin talletusta STORE-käskyssä)

- 0 välitön osoitus (STORE-käsky: suora osoitus)
- 1 suora osoitus (STORE-käsky: epäsuora osoitus)
- 2 epäsuora osoitus (STORE-käsky: epäkelpo arvo, virhetilanne)
- 3 epäkelpo arvo, virhetilanne

Indeksirekisteri on myös 3 bitin kentässä. Indeksirekisteriä käytetään 2. operandin määrittelemiseen, mutta sitä ei tarvita aina. Tämän asian ilmaisemiseen voisi käyttää ylimääräistä bittiä, mutta ttk-91'ssä on päädytty tilaa säästävään ratkaisuun, jossa Ri-kentän arvo 0 ilmaisee, että indeksirekisteri ei ole käytössä tällä kertaa. Tästä tietenkin seuraa, että indeksirekisterinä voi käyttää vain rekistereitä R1-R7. Ttk-91'ssä indeksirekistereinä käytetään yleisrekistereiden kanssa yhteisiä fyysisiä rekistereitä, mutta todellisissa suorittimissa ne voisivat olla myös fyysisesti erillisiä rekisterijoukkoja.

Ttk-91 konekäskyn rakenne

Käskyn esitys bittitasolla on aina samantyyppinen

(bitit on numeroitu oikealta vasemmalle, nollasta alkaen)

OPER operaatiokoodi, opcode, 0-112 (kaikki arvot eivät ole laillisia)
Rj 1. operandi, tulosrekisteri, R0-R7 (yleisrekisteri)
Ri indeksirekisteri, R1-R7 (arvo 0 indikoi että indeksirekisteri ei ole käytössä)
ADDR osoiteosa, vakio-osa, arvoalue [-32767, +32767]
M moodi, muistinoutojen määrä toisen operandin laskemiseksi (ennen mahdollista muistiin talletusta STORE-käskyssä)

- 0 välitön osoitus (STORE-käsky: suora osoitus)
- 1 suora osoitus (STORE-käsky: epäsuora osoitus)
- 2 epäsuora osoitus (STORE-käsky: epäkelpo arvo, virhetilanne)
- 3 epäkelpo arvo, virhetilanne

ADDR-kenttä on konekäskyn 16 viimeistä bittiä. Tilanteesta riippuen sinne voi tallettaa muistiosoitteen tai vakion. Tämän vuoksi sitä myös sanotaan joskus osoitekentäksi ja joskus vakiokentäksi. Vakiokentän ensimmäinen bitti ilmaisee sen etumerkin, joten sen 15-bittinen itseisarvo on korkeintaan 32767. Etumerkkibitti on 0 positiivisille ja 1 negatiivisille vakioille. On huomattava, että tarvittaessa suurempia osoitteita tai vakioita, ne on talletettava muistiin data-segmenttiin eikä käskyn vakio-osaan konekäskyn sisälle koodisegmenttiin.

Ttk-91 konekäskyn rakenne

Käskyn esitys bittitasolla on aina samantyyppinen

(bitit on numeroitu oikealta vasemmalle, nollasta alkaen)

OPER operaatiokoodi, opcode, 0-112 (kaikki arvot eivät ole laillisia)
Rj 1. operandi, tulosrekisteri, R0-R7 (yleisrekisteri)
Ri indeksirekisteri, R1-R7 (arvo 0 indikoi että indeksirekisteri ei ole käytössä)
ADDR osoiteosa, vakio-osa, arvoalue [-32767, +32767]
M moodi, muistinoutojen määrä toisen operandin laskemiseksi (ennen mahdollista muistiin talletusta STORE-käskyssä)

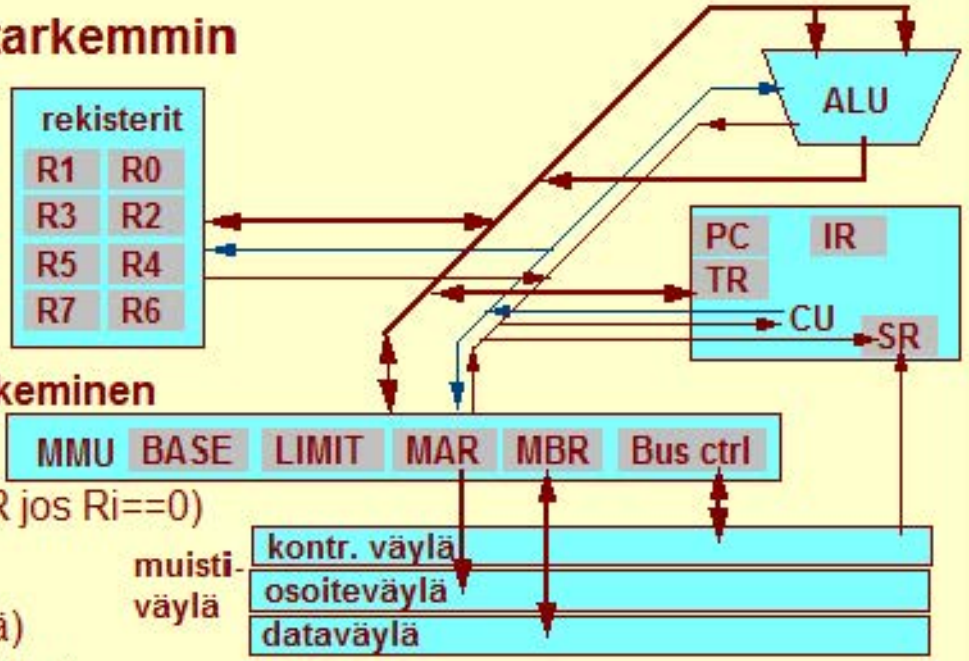
LOAD R1, A
STORE R1, A

- 0 välitön osoitus (STORE-käsky: suora osoitus)
- 1 suora osoitus (STORE-käsky: epäsuora osoitus)
- 2 epäsuora osoitus (STORE-käsky: epäkelpo arvo, virhetilanne)
- 3 epäkelpo arvo, virhetilanne

Moodi-kenttä ilmaisee jäkimmäiser operandin arvon laskentaa varten tarvittavien muistinoutojen määrän. STORE-käskyssä jälkimmäinen operandi on muistiosoite, jonka laskemiseen tarvitaan muistinoutoja yksi vähemmän kuin vastaavassa tilanteessa LOAD-käskyn yhteydessä tapahtuvan operandin arvon hakemiseksi. Tämän vuoksi STORE-käskyissä on aina M-kentän arvo yksi vähemmän kuin vastaavassa LOAD-käskyssä. Jos M-kentän arvo on liian suuri, niin siitä aiheutuu suoritusaikainen virhetilanne. Tämähän ei siis ole mikään varsinainen virhe, vaan vääränmuotoinen käsky, jonka suoritin havaitsee ja aivan oikein toimien kieltäytyy suorittamasta.

Nouto- ja suoritussykli vähän tarkemmin

Noutovaihe
- muistista MBR'n kautta IR'ään
- lisää 1 PC'hen



Käskyn purku ja muistiosoitteen laskeminen
- OPER, Rj, M, Ri, ADDR
- TR ← (Ri) + ADDR (pelkkä ADDR jos Ri==0)

Operandin nouto (ei kaikilla käskyillä)
- muistista MBR'n kautta TR'ään (0-2 krt)

ALU-operaatio
- tulos rekisteriin R0-R7, TR ja/tai SR

Muistiin talletus (ei kaikilla käskyillä)
- muistiin MBR'n kautta

Käskyjen nouto- ja suoritussykli toteutetaan nyt suorittimelle yksi vaihe kerrallaan. Vaiheiden toteutuksen tekee kontrolliyksikkö, joka sisäisen logiikkansa avulla tietää koko ajan, mitä kunkin komponenentin pitää kullakin hetkellä tehdä. Käskyn noutovaihe tapahtuu karkeasti ottaen siten, että MMU'ta ja muistiväylää ohjataan sopivasti hakemaan PC'n osoittama sana muistista MBR'n kautta IR'ään. Samassa yhteydessä PC'n arvoa kasvatetaan yhdellä. Tähän tarkoitukseen voi käyttää suorittimen yleiskäyttöistä ALU'a tai sitten kontrolliyksikön sisällä voi olla tätä tarkoitusta varten oma 'inkrementoi'-piirinsä.

Nouto- ja suoritussykli vähän tarkemmin

Noutovaihe

- muistista MBR'n kautta IR'ään
- lisää 1 PC'hen

Käskyn purku ja muistiosoitteen laskeminen

- OPER, Rj, M, Ri, ADDR
- TR ← (Ri) + ADDR (pelkkä ADDR jos Ri==0)

Operandin nouto (ei kaikilla käskyillä)

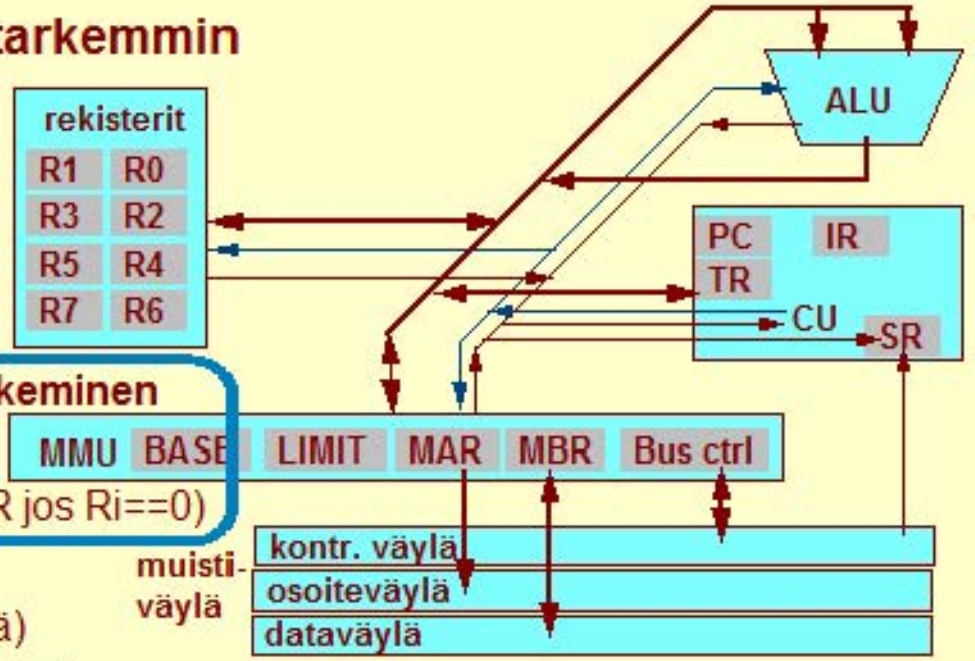
- muistista MBR'n kautta TR'ään (0-2 krt)

ALU-operaatio

- tulos rekisteriin R0-R7, TR ja/tai SR

Muistiin talletus (ei kaikilla käskyillä)

- muistiin MBR'n kautta

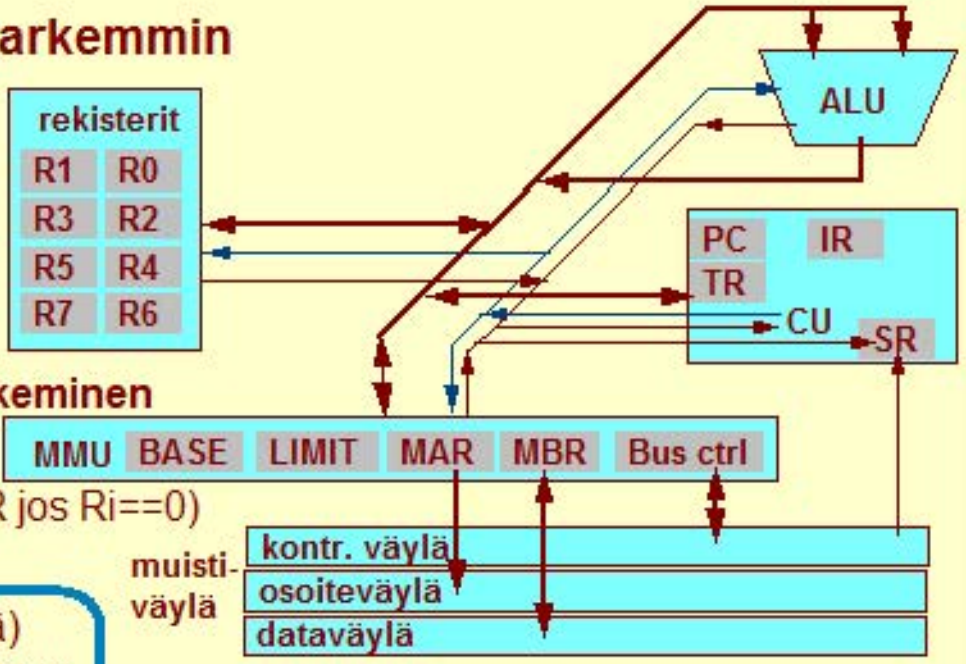


Käskyn purku tapahtuu jakamalla IR'ssä oleva käsky kenttiensä mukaisiin osiin. Rekisteriin TR lasketaan toisen operandin laskennassa tarvittava arvo, (Ri) + ADDR. Jos indeksirekisteriä ei käytetä, niin tätä indikoi kentän Ri'n arvo nolla konekäskyssä. Jos taas vakiokenttää ADDR ei tarvita, niin sinne laitetaan arvo 0.

Nouto- ja suoritussykli vähän tarkemmin

Noutovaihe

- muistista MBR'n kautta IR'ään
- lisää 1 PC'hen



Käskyn purku ja muistiosoitteen laskeminen

- OPER, Rj, M, Ri, ADDR
- TR ← (Ri) + ADDR (pelkkä ADDR jos Ri==0)

Operandin nouto (ei kaikilla käskyillä)

- muistista MBR'n kautta TR'ään (0-2 krt)

ALU-operaatio

- tulos rekisteriin R0-R7, TR ja/tai SR

Muistiin talletus (ei kaikilla käskyillä)

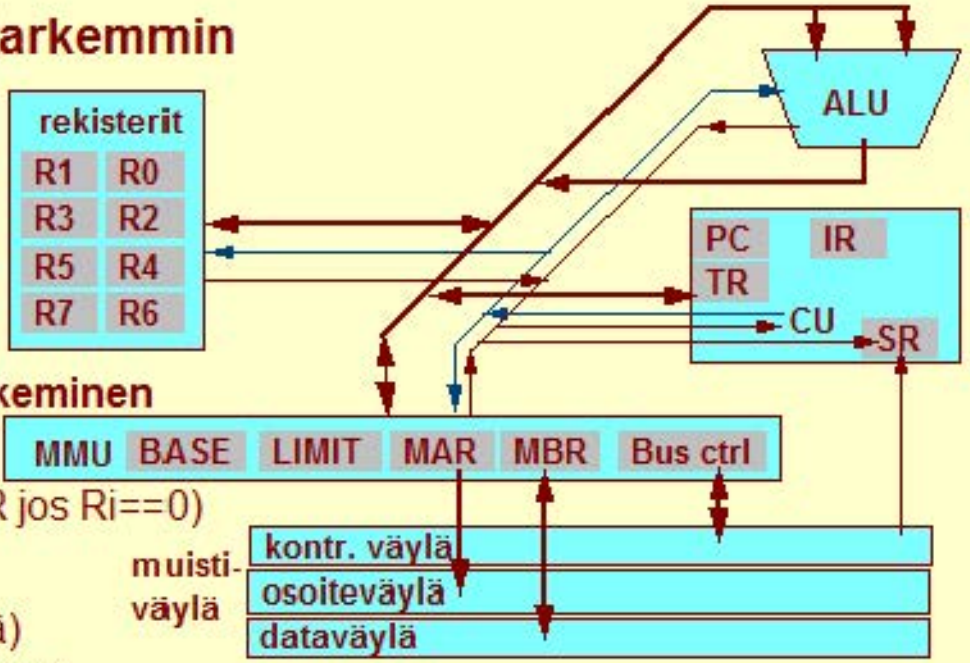
- muistiin MBR'n kautta

Jälkimmäisen operandin viittausmoodista riippuen TR'ssä olevaa arvoa joko käytetään sellaisenaan tai sitten sitä käytetään muistiosoitteena ja varsinainen operandi haetaan TR'n osoittamasta paikasta muistista. Arvo sijoitetaan TR'ään osoitteen paikalle, joten joka tapauksessa tämän vaiheen jälkeen TR'stä löytyy jälkimmäisen operandin arvo. Epäsuoran muistiviitteen kohdalla TR'ää käytetään muistisoitteena vielä toisenkin kerran. Ensimmäisellä kerrallahan muistista haettiin vasta varsinaisen datan osoite, ja toisella kerralla sitten varsinainen data. Kaikilla käskyillä ei tätä vaihetta ole.

Nouto- ja suoritussykli vähän tarkemmin

Noutovaihe

- muistista MBR'n kautta IR'ään
- lisää 1 PC'hen



Käskyn purku ja muistiosoitteen laskeminen

- OPER, Rj, M, Ri, ADDR
- TR ← (Ri) + ADDR (pelkkä ADDR jos Ri==0)

Operandin nouto (ei kaikilla käskyillä)

- muistista MBR'n kautta TR'ään (0-2 krt)

ALU-operaatio

- tulos rekisteriin R0-R7, TR ja/tai SR

Muistiin talletus (ei kaikilla käskyillä)

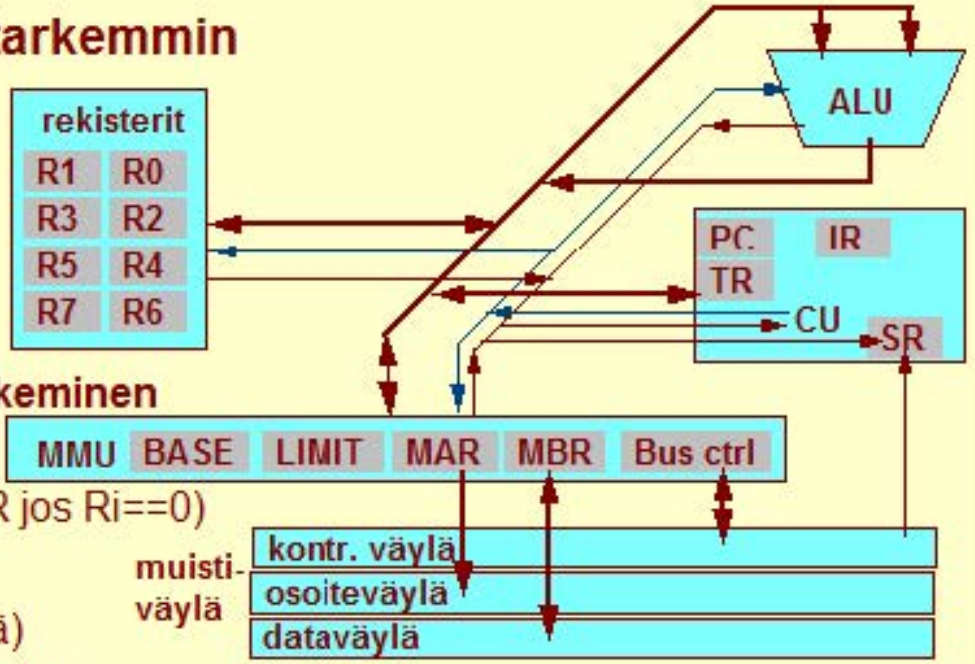
- muistiin MBR'n kautta

ALU-vaiheessa ensimmäinen operandi on siis työrekisterissä R1-R7 ja toinen operandi on edellisessä vaiheessa laskettu tai haettu muistista TR'ään. Nämä annetaan nyt ALU'lle, ja operaation tulos talletetaan joko suoraan työrekisteriin R0-R7, tilapäisrekisteriin TR ja/tai tilarekisteriin SR. Tulos laitetaan TR'ään vaikkapa STORE-käskyn yhteydessä, koska se sisältää muistisoitteen. Tilarekisteriin SR tulos tallettuu vaikkapa COMP-käskyn yhteydessä, koska vertailun tulos menee aina SR'ään.Myös yhteenlaskun yhteydessä voi tilarekisteri päivittyä esimerkiksi ylivuodon sattuessa.

Nouto- ja suoritussykli vähän tarkemmin

Noutovaihe

- muistista MBR'n kautta IR'ään
- lisää 1 PC'hen



Käskyn purku ja muistiosoitteen laskeminen

- OPER, Rj, M, Ri, ADDR
- TR ← (Ri) + ADDR (pelkkä ADDR jos Ri==0)

Operandin nouto (ei kaikilla käskyillä)

- muistista MBR'n kautta TR'ään (0-2 krt)

ALU-operaatio

- tulos rekisteriin R0-R7, TR ja/tai SR

Muistiin talletus (ei kaikilla käskyillä)

- muistiin MBR'n kautta

Muistiintalletusvaiheessa kontrolliyksikkö tietää ennalta, että talletettava arvo on jossain työrekisterissä ja käytettävä muistiosoite on valmiina TR:ssä. Nämä tiedot pitää nyt sopivasti antaa MMU'lle, jolloin työrekisterin arvo kopioituu MBR'n kautta muistiin. Tämä suoritusvaihe on ttk-91 -koneessa vain STORE- ja PUSH-käskyillä. PUHSR-käsky on käytännössä hyvin monimutkainen, koska se oikeastaan koostuu seitsemästä tavallisesta PUSH-käskystä. Voitte kuvitella, miten se toteutettaisiin, mutta emme käy sitä tässä sen tarkemmin läpi.

Käskyn noutovaihe

Vie PC'n arvo MAR'iin
(ja siten osoiteväylälle)

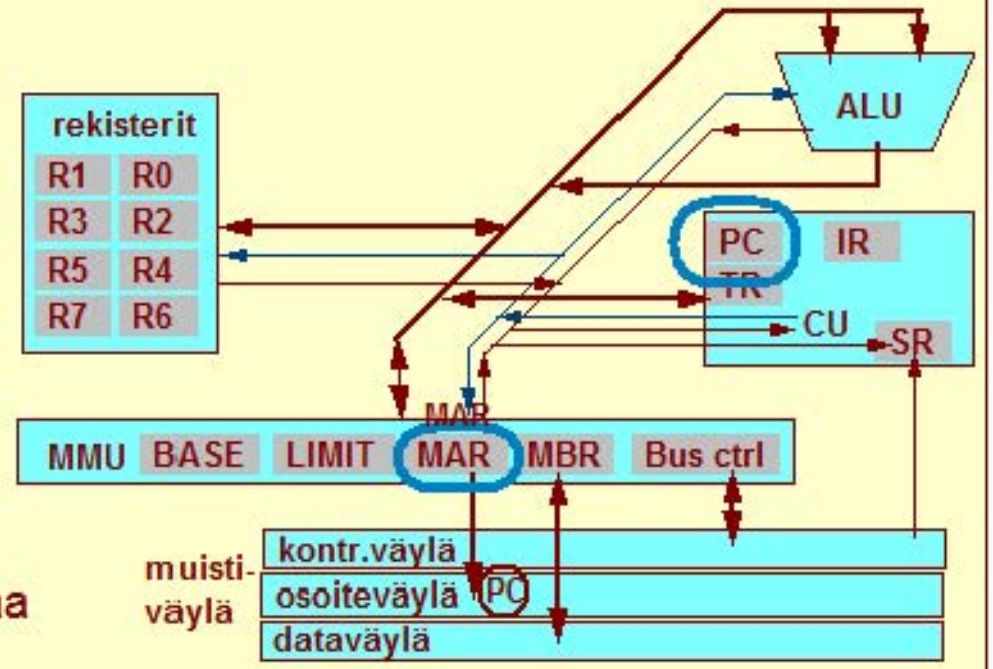
Aseta muistin lukusignaali kontrolliväylälle asentoon "lue"

Odota, kunnes muistipiiri toimittaa sanan dataväylälle, josta se automaattisesti kopioituu MBR'ään

Siirrä konekäsky MBR'stä IR'ään

ADD R1, =4

Kasvata PC'n arvoa yhdellä



Tarkastellaaan nyt suorittimen toimintaa käskyn nouto- ja suoritussyklin vaiheissa vielä tarkemmin. Käskyn noutovaihe voidaan toteuttaa seuraavasti. Ensinnäkin kopioimme PC'ssä olevan seuraavan käskyn osoitteen MAR'iin. CU tekee tämän syöttämällä PC'n arvon sisäiselle dataväylälle ja antamalla sitten MMU'lle kontrollipiuhojen välityksellä ohjeen, että 'kopioi sisäisellä väylällä oleva arvo MAR rekisteriin'. Toiminnot ovat hyvin yksinkertaisia ja ne voidaan toteuttaa yksinkertaisten elektronisten kytkimien avulla.

Käskyn noutovaihe

Vie PC'n arvo MAR'iin
(ja siten osoiteväylälle)

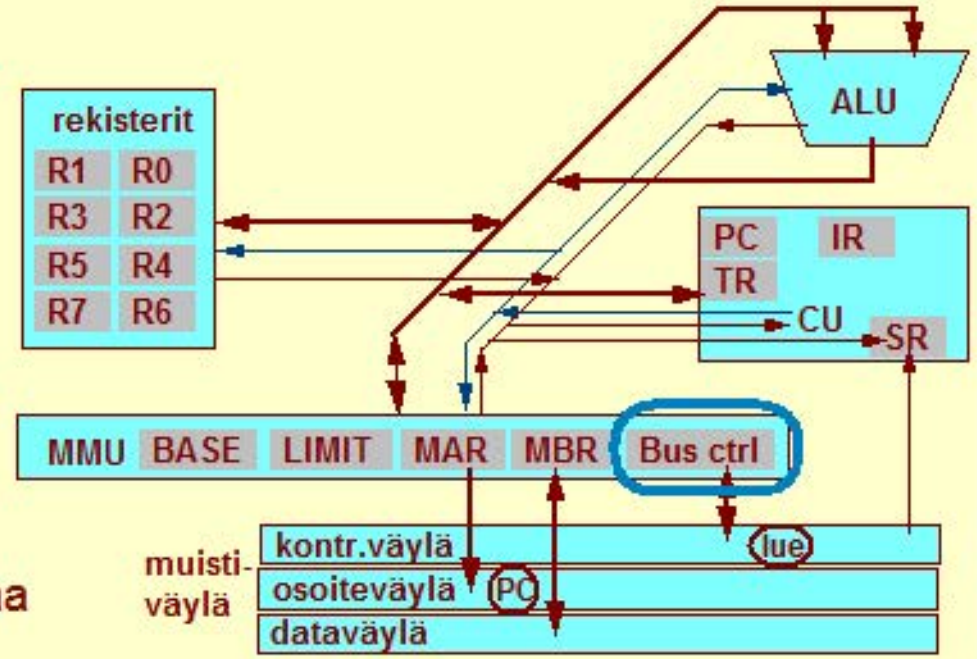
Aseta muistin lukusignaali kontrolliväylälle asentoon "lue"

Odota, kunnes muistipiiri toimittaa sanan dataväylälle, josta se automaattisesti kopioituu MBR'ään

Siirrä konekäsky MBR'stä IR'ään

ADD R1, =4

Kasvata PC'n arvoa yhdellä



Muistista luku on vielä yksinkertaisempaa. MMU'ta pyydetään asettamaan lukusignaali muistiväylän kontrollipiuhoille. Muistipiiri huomaa tämän hyvin pian ja ryhtyy toimimaan, lukien muistiosoitteen osoiteväylältä.

Käskyn noutovaihe

Vie PC'n arvo MAR'iin
(ja siten osoiteväylälle)

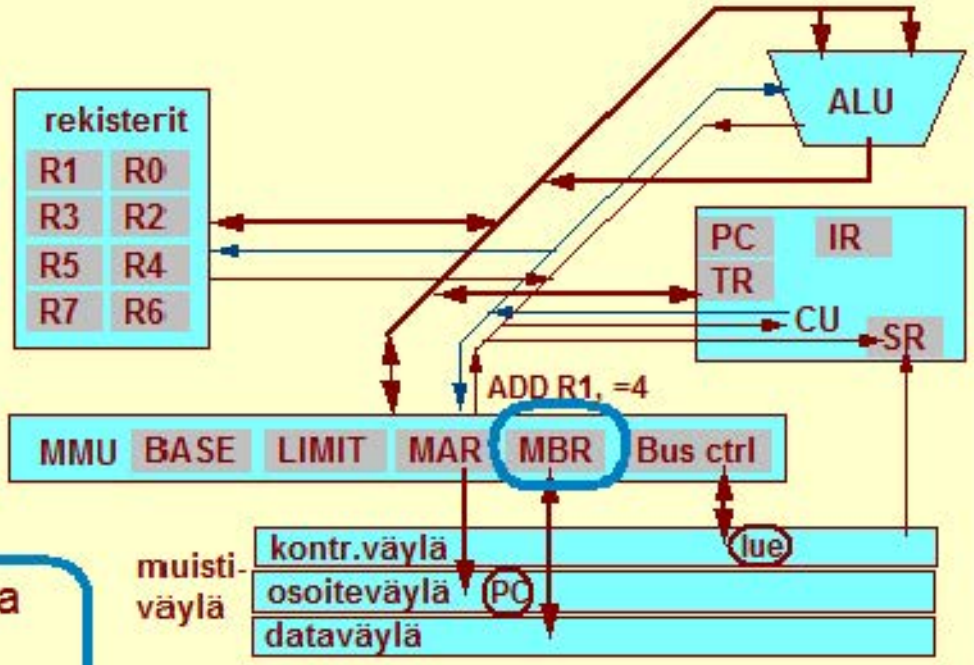
Aseta muistin lukusignaali
kontrolliväylälle asentoon "lue"

Odota, kunnes muistipiiri toimittaa sanan dataväylälle, josta se automaattisesti kopioituu MBR'ään

Siirrä konekäsky MBR'stä IR'ään

ADD R1, =4

Kasvata PC'n arvoa yhdellä



Muistipiirin toiminta väylän takana kestää jonkin verran. Toteutuksesta riippuen me joko odotamme deterministisen ajan tai sitten muistipiiri ilmaisee kontrollipiuhojen avulla, milloin haluttu sana on valmiina luettavissa dataväylältä ja sitten myös MBR'stä.

Käskyn noutovaihe

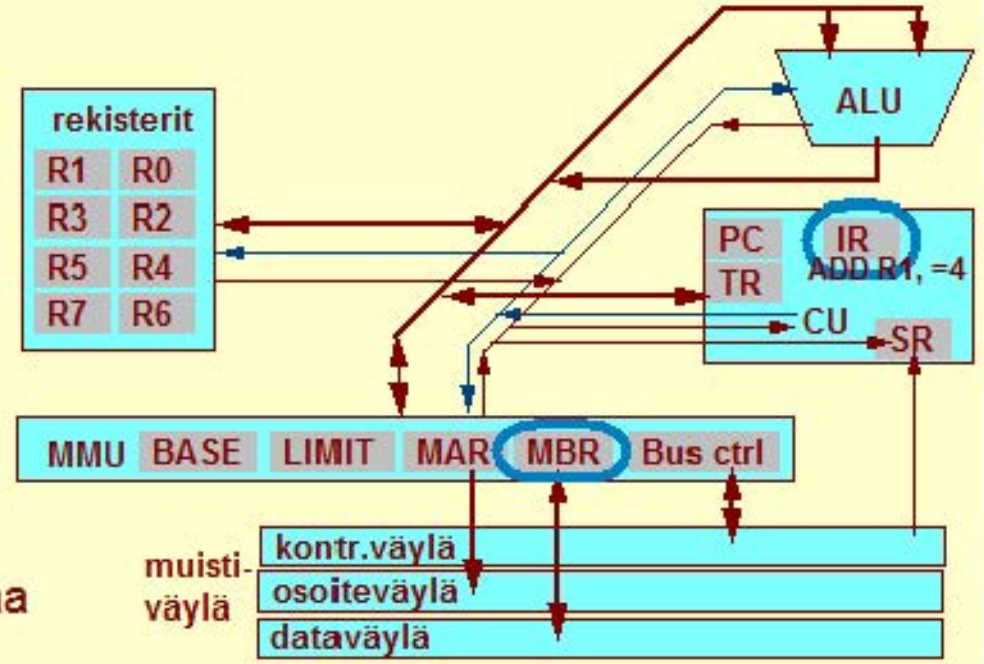
Vie PC'n arvo MAR'iin
(ja siten osoiteväylälle)

Aseta muistin lukusignaali
kontrolliväylälle asentoon "lue"

Odota, kunnes muistipiiri toimittaa
sanan dataväylälle, josta se
automaattisesti kopioituu MBR'ään

Siirrä konekäsky MBR'stä IR'ään

Kasvata PC'n arvoa yhdellä



ADD R1, =4

Seuraavaksi konekäsky pitää kopioida MBR'stä käskyrekisteriin. Tämäkin on hyvin suoraviivaista. Annetaan ensin MMU'lle komento kopioida MBR'n arvo sisäiselle väylälle ja sitten komennetaan IR'ää lukemaan sisäisellä väylällä näkyvä arvo. Käskyrekisterissä on nyt valmiina seuraavaksi suoritettava konekäsky.

Käskyn noutovaihe

Vie PC'n arvo MAR'iin
(ja siten osoiteväylälle)

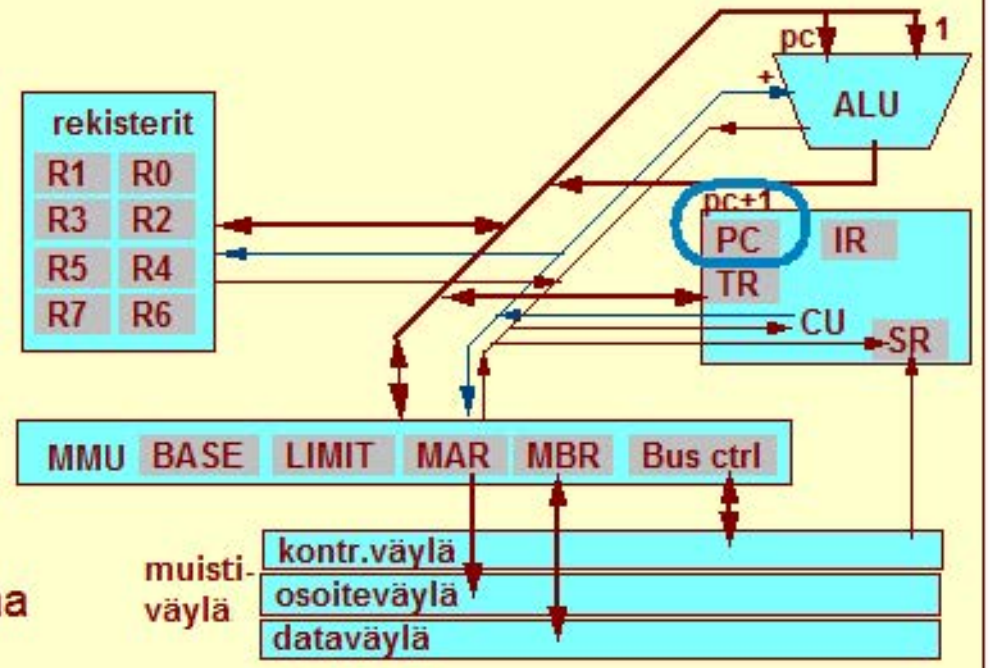
Aseta muistin lukusignaali
kontrolliväylälle asentoon "lue"

Odota, kunnes muistipiiri toimittaa
sanan dataväylälle, josta se
automaattisesti kopioituu MBR'ään

Siirrä konekäsky MBR'stä IR'ään

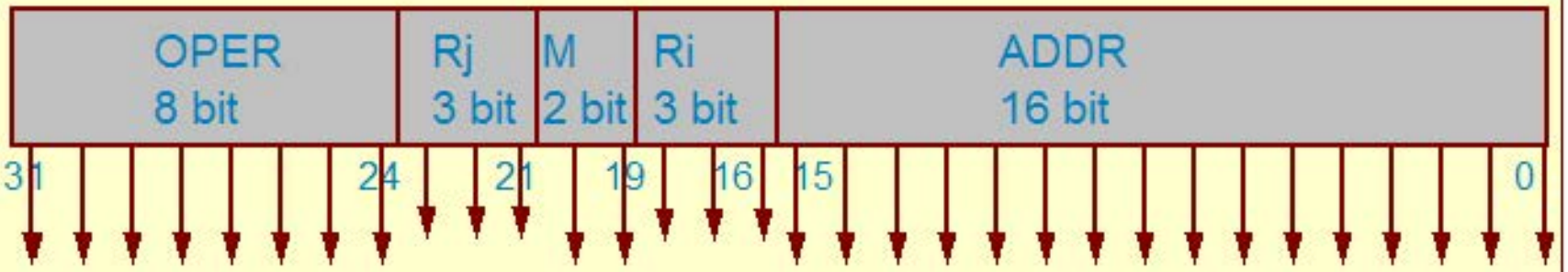
ADD R1, =4

Kasvata PC'n arvoa yhdellä



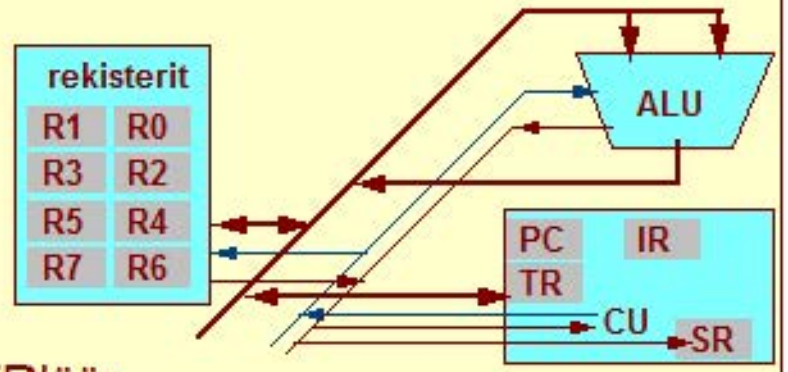
Lopuksi PC'n arvoa pitää vielä kasvattaa yhdellä. Käytämme tässä ALU'a yhteenlaskun tekemiseen. Siirrämme ALU'n sisäänmenoihin PC'n nykyarvon ja sitten vakion 1, annamme yhteenlaskusignaalin ja lopulta siirrämme ALU'n ulostulon PC'n uudeksi arvoksi. Käskyn noutovaihe on nyt valmis. Jokainen välivaihe on todella yksinkertainen ja CU ohjaa toimintaa yksinkertaisten kytkimien avulla, joita se asettelee kontrollipiuhojen välityksellä. CU'n toteutusta tutkitaan tarkemmin seuraavalla Tietokoneen rakenne -kurssilla, mutta tälle kurssille tämä tarkastelutaso saa riittää.

Käskyn purku



Purku tapahtuu automaattisesti IR'ssä

- kukin kenttä suoraan luettavissa sitä varten rakennettujen johtimien kautta

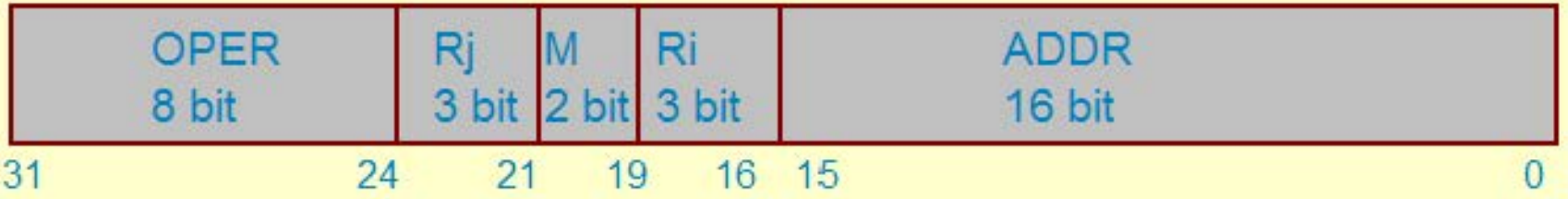


Jälkimmäisen operandin laskentaa varten TR'ään lasketaan tehollinen muistisoite (EA, Effective Address)

- jos, Ri = 0, niin TR ← ADDR, muutoin TR ← (Ri) + ADDR
- yhteenlasku tehdään ALU'ssa

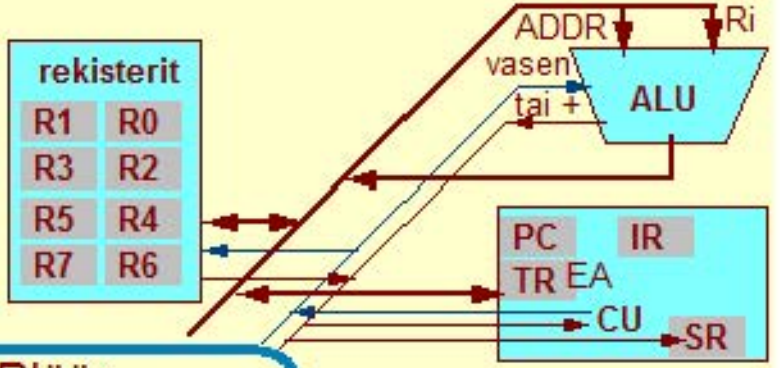
Käskyn purku IR'ssä on automaattista. Käskyrekisteri on suunniteltu nimenomaan sillä tavalla, että käskyn kentät on sieltä helppo lukea. Jokaista kenttää varten on valmiit johtimet, joiden avulla juuri sen kentän sisältö on luettavissa.

Käskyn purku



Purku tapahtuu automaattisesti IR'ssä

- kukin kenttä suoraan luettavissa sitä varten rakennettujen johtimien kautta



Jälkimmäisen operandin laskentaa varten TR'ään lasketaan tehollinen muistisoite (EA, Effective Address)

- jos, Ri = 0, niin TR ← ADDR, muutoin TR ← (Ri) + ADDR
- yhteenlasku tehdään ALU'ssa

Käskyn purun yhteydessä valmistaudumme toisen operandin arvon laskemiseen. Laskemme niin sanotun tehollisen muistiosoitteen rekisteriin TR. TR'n arvoksi tulee käskyn indeksirekisterin ja vakio-osan summa. Jos indeksirekisterin kentässä oli nolla, niin TR'ksi tulee vain osoite-osan arvo. TR'n arvon laskemiseen käytetään tässä suorittimen yleiskäyttöistä ALU'a. Kirjallisuudessa tätä TR'n arvoa sanotaan joskus teholliseksi muistiosoitteeksi ainoastaan silloin, kun sitä tosiaan käytetään muistiosoitteena. Käytämme termiä tässä yleisemmässä merkityksessä.

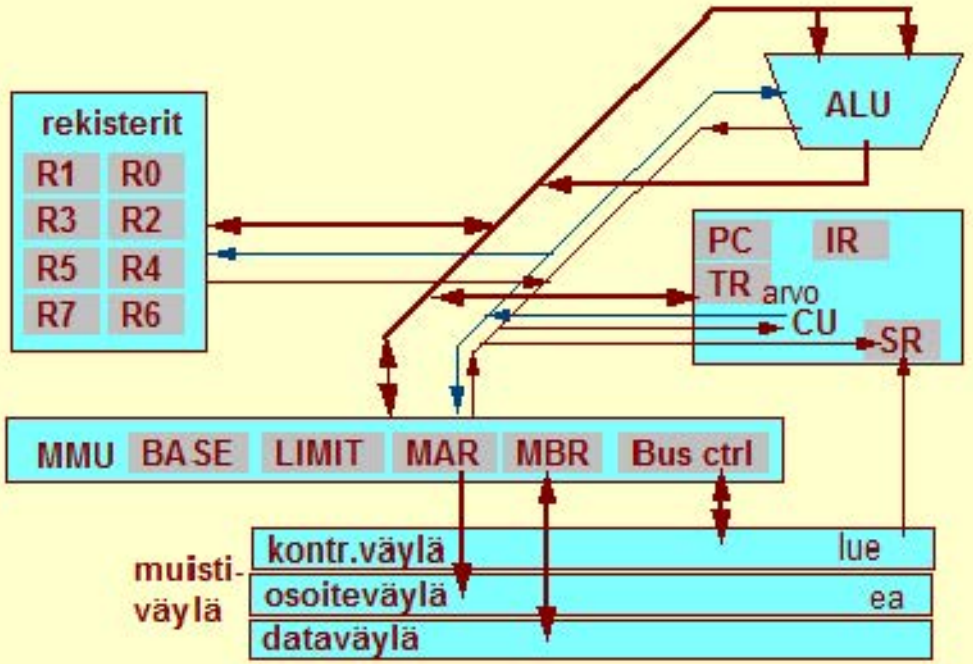
Operandin luku (0-2 krt)

Vie muistisoite MAR'iin

Aseta signaali "lue" muistiväylän kontrollipiuhoille

Odota, kunnes muistipiiri toimittaa väylän kautta uuden arvon MBR'ään

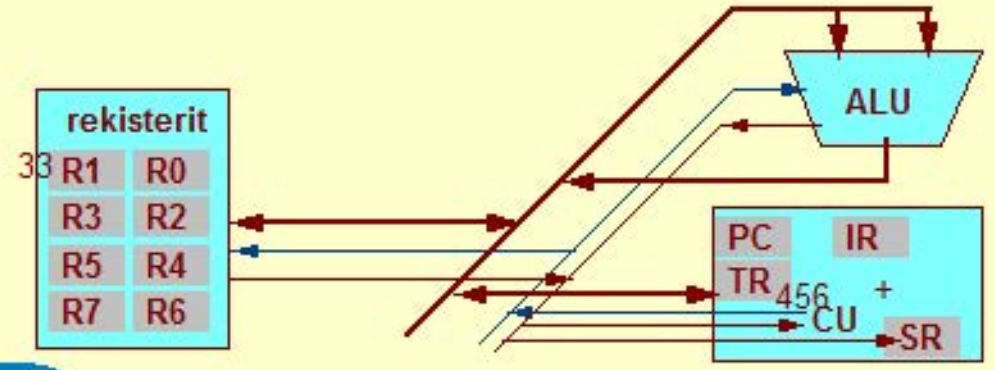
Siirrä sana MBR'stä TR'ään




```
ADD  R1, =4     ; välitön operandi: 0 kpl muistinoutoja
ADD  R1, 24     ; normaali viite:   1 kpl muistinoutoja
ADD  R1, @24    ; epäsuora viite:   2 kpl muistinoutoja
```


Operandin luku tapahtuu 0-2 kertaa. Jos kyseessä on välitön operandi, niin TR'ssä olevaa arvoa käytetään sellaisenaan 2. operandina. Normaalissa muistiosoituksessa TR'n arvoa käytetään muistiosoitteena, ja sen osoittamasta muistipaikasta haetaan varsinainen operandi, joka sijoitetaan TR'ään muistiosoitteen paikalle. Epäsuorassa muistiosoituksessa ensimmäisellä kerralla haettu arvo olikin vasta varsinaisen tiedon osoite, joten muistista lukusykli toistetaan. Joka tapauksessa, tämän vaiheen jälkeen 2. operandi löytyy sisäisestä rekisteristä TR.

ALU-operaatio



Lähtötilanne

- käsky haettu ja purettu osiin IR'ssä
- 1. operandi työrekisterissä
- 2. operandi TR'ssä

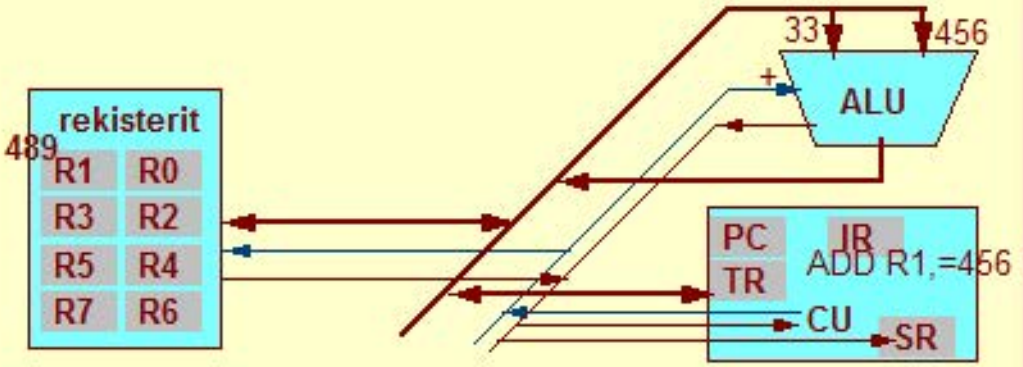
ADD R1, =456

Käskyn suoritus ALU:ssa

- vie operandit yksi kerrallaan sisäistä väylää pitkin ALU'un
- anna ALU'lle sopiva ohjaussignaali
- odota, että ALU'n tulos valmis
- talleta tulos työrekisteriin, MBR'ään, PC'hen ja/tai SR'ään

ALU-vaiheen lähtökohtana on aikaisemmissa vaiheissa tehty työ. Käsky on siis nyt haettu muistista ja sen eri kentät on helposti luettavissa IR'ssä. Ensimmäinen operandi on käskyn mainitsemassa työrekisterissä Rj, ja toisen operandin arvo on TR'ssä. Konekäskyn operaatiokoodista voidaan suoraan päätellä ALU-piirille annettava kontrollisignaali tai sitten operaatiokoodia voi sellaisenaan käyttää ALU'n kontrollisignaalina.

ALU-operaatio



Lähtötilanne

- käsky haettu ja purettu osiin IR'ssä
- 1. operandi työrekisterissä
- 2. operandi TR'ssä

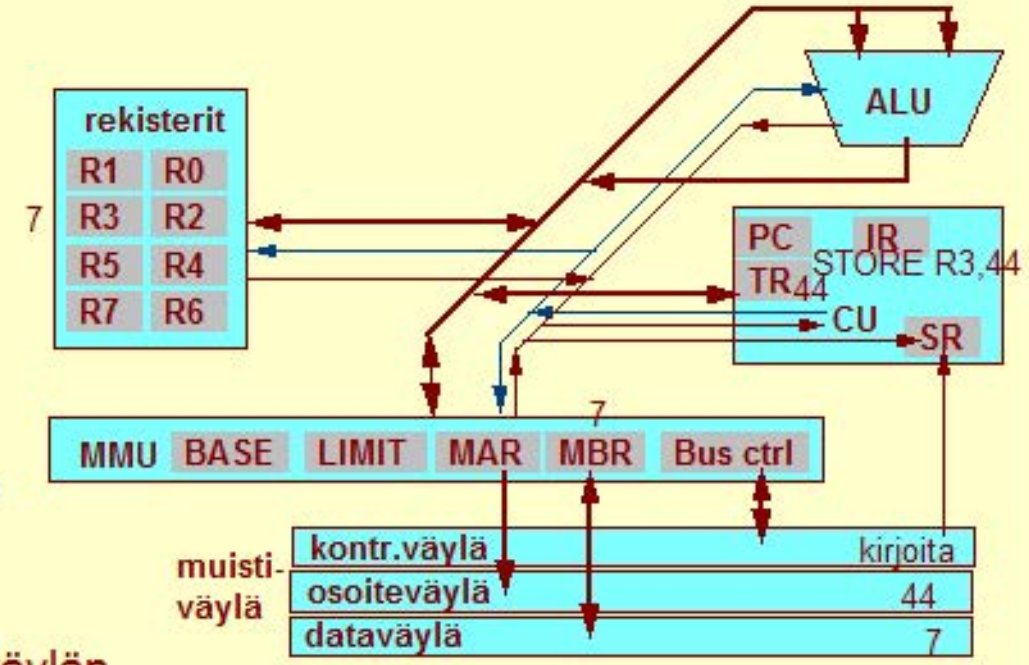
ADD R1, =456

Käskyn suoritus ALU:ssa

- vie operandit yksi kerrallaan sisäistä väylää pitkin ALU'un
- anna ALU'lle sopiva ohjaussignaali
- odota, että ALU'n tulos valmis
- talleta tulos työrekisteriin, MBR'ään, PC'hen ja/tai SR'ään

Varsinainen ALU-operaatio voi nyt alkaa. Operandit syötetään ALU'lle yksi kerrallaan sisäistä väylää käyttäen ja ALU'lle annetaan kontrollisignaalina haluttu toiminto. Vähän ajan kuluttua laskutoimituksen tulos on ALU'n ulostulossa ja se voidaan sieltä kopioida työrekisteriin, MBR'ään, PC'hen ja/tai SR'ään, konekäskystä riippuen. Esimerkiksi STORE-käskyn yhteydessä ALU'lle annetaan komento 'vasen' ja tulos laitetaan MBR'ään muistiin kopiointia varten. Huomaa, että tässä vaiheessa tehdään kaikki tietokoneen tekemä laskentatyö. Kaikki muu on tämän työn tekemiseen kohdistuvaa hallintoa.

Tuloksen muistiinkirjoitus



Vie muistisoite MAR'iin

Vie kirjoitettava sana MBR'ään

- ellei se jo ole siellä

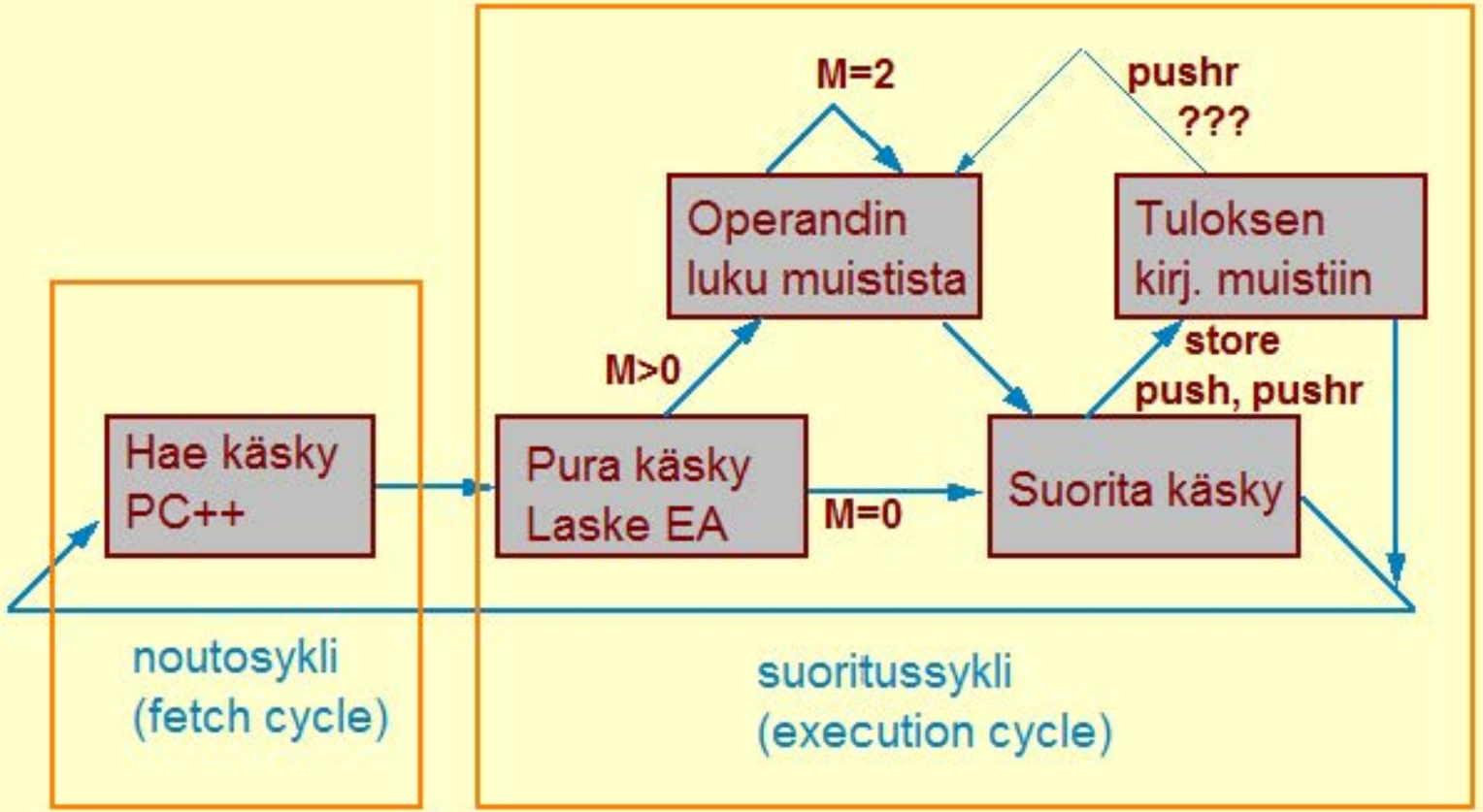
Aseta signaali "kirjoita" muistiväylän kontrollipiuhoille

Odota, kunnes MBR'ssä oleva arvo on kirjoitettu muistiin

STORE R3, 44

Muistiintalletus on syklin viimeinen vaihe. Se tapahtuu vain STORE- ja PUSH-käskyillä. Aikaisemmissa vaiheissa työrekisterin arvo on kopioitu MBR'ään, josta arvo pitää nyt tallettaa muistiin. TR'ssä on valmiina muistisoite, joten se kopiodaan ensin MMU'n MAR'iin ja sieltä se 'valuu' automaattisesti muistiväylän osoitepiuhoille. Muistipiirille pitää antaa vielä kontrollisignaali 'kirjoita', jonka seurauksena pienen odottelun jälkeen haluttu arvo on kirjoitettu muistiin. Käskyn suoritus on nyt valmis ja voimme siirtyä seuraavaan käskyyn, jonka osoite löytyy PC'stä.

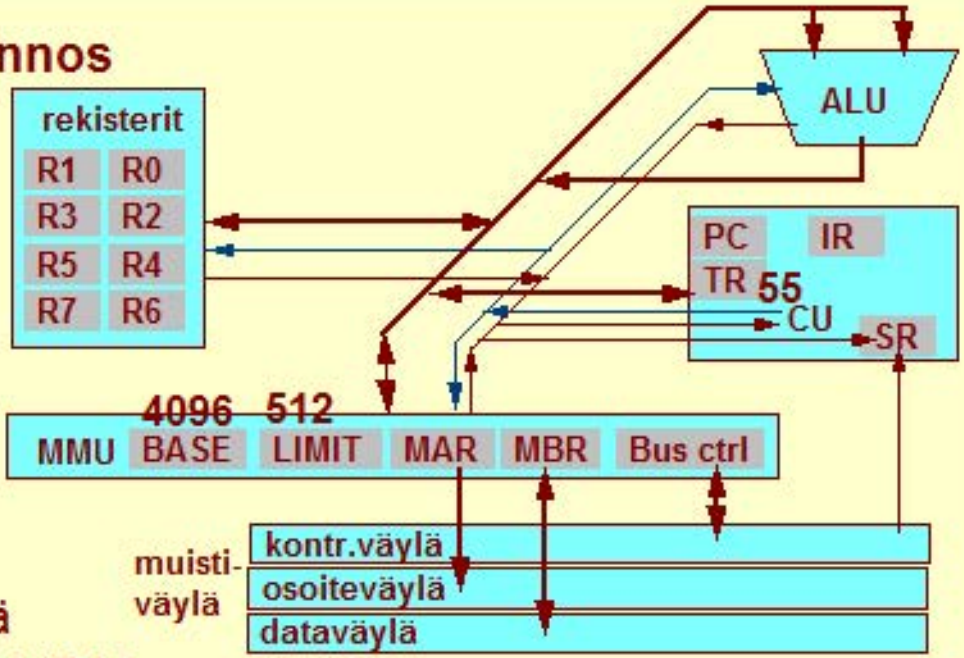
Ttk-91 nouto- ja suoritussykli



Olemme nyt tarkentaneet käskyjen nouto- ja suoritussykliä jonkin verran. Noutosykli on ennallaan, mutta suoritussyklissä on nyt useita erillisiä vaiheita. Operandin lukuvaihe tehdään vain M-kentän arvon ollessa suurempi kuin nolla, ja se voidaan myös suorittaa kaksi kertaa epäsuoran muistinosoitusviitteen yhteydessä. Käskyn tulos kirjoitetaan muistiin ainoastaan STORE-, PUSH- ja PUSHR-käskyillä. PUSHR-käsky on itse asiassa niin monimutkainen, että se sotkee koko tämän hienon kuvan. Oikeassa suorittimessa monimutkaiset käskyt jätetään yleensä toteuttamatta ja niiden toteutus on sen sijaan tehty aliohjelmilla, joihin päästään käsiksi myöhemmin selitettävän keskeytysmekanismin avulla.

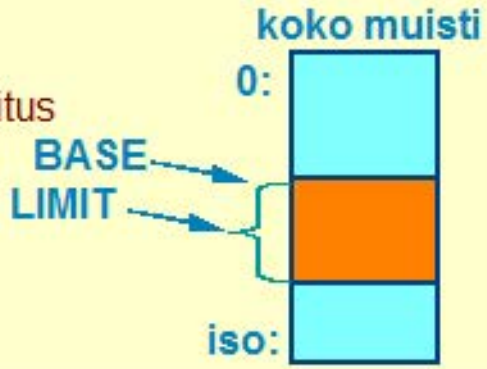
MMU'n tekemä osoitteenmuunnos

Ohjelman käyttämät muistiosoitteet (VA) ovat sen omia, välillä [0, LIMIT-1]



MMU'lle annettua ohjelman käyttämää muistiosoitetta (VA) ei käytetä suoraan, vaan se pitää muuttaa keskusmuistiosoittteeksi (PA)

- tarkista, onko VA laillinen osoite, eli onko VA 0 [0, LIMIT-1]
- jos ei ole, niin aseta SR'n bitti M päälle ja lopeta käskyn suoritus
- muutoin, laske keskusmuistiosoite kaavalla PA ← BASE + VA ja aseta tämä arvo MAR'iin keskusmuistiosoitteeksi

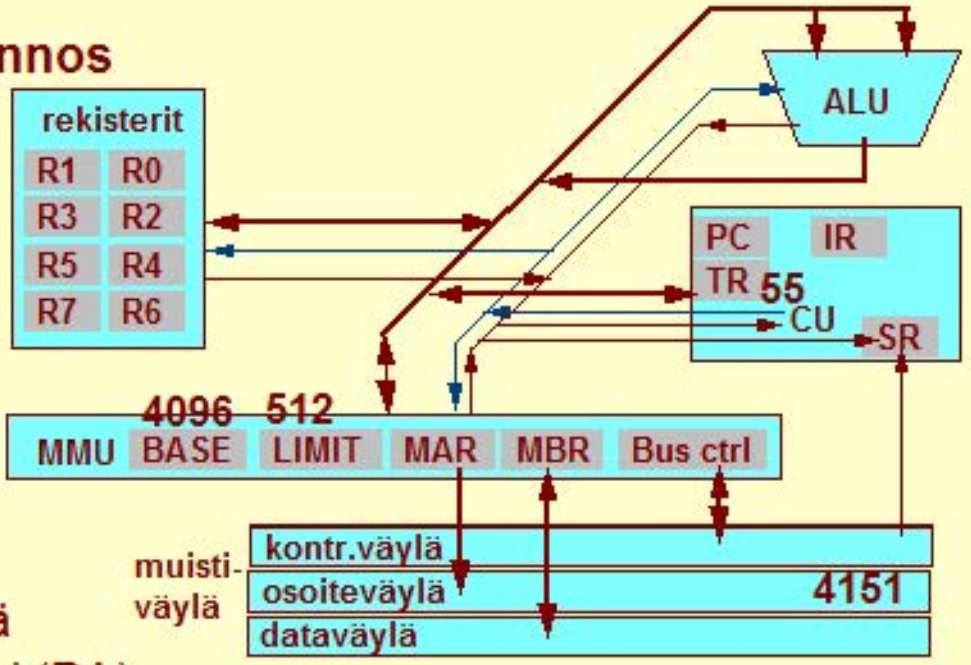


VA = Virtual Address PA = Physical Address

Edellä esitetty MMU'n toiminta oli tahallaan yksinkertaistettu. Todellisuudessa se on vähän monimutkaisempi. Ohjelmallahan on käytössään ihan oma muistiavaruutensa eli käytettävien muistiosoitteiden joukko, joka on aika pieni ja välillä [0, LIMIT-1]. Ohjelmaa käynnistettäessä käyttöjärjestelmä on sijoittanut ohjelman jonnekin päin suurta keskusmuistia, ja asettanut MMU'n rajarekisterit (BASE ja LIMIT) osoittamaan tämän ohjelman käyttämän muistialueen. Sen alkuosoite on siis MMU'n sisäisessä rekisterissä BASE ja sen koko rekisterissä LIMIT.

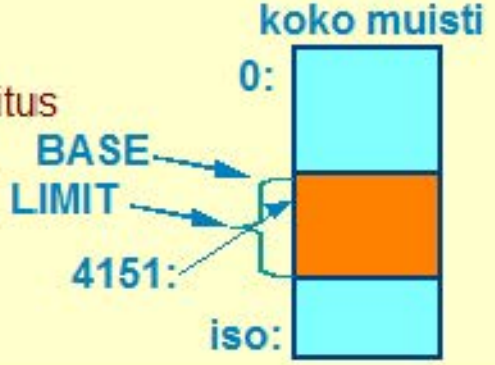
MMU'n tekemä osoitteenmuunnos

Ohjelman käyttämät muistiosoitteet (VA) ovat sen omia, välillä [0, LIMIT-1]



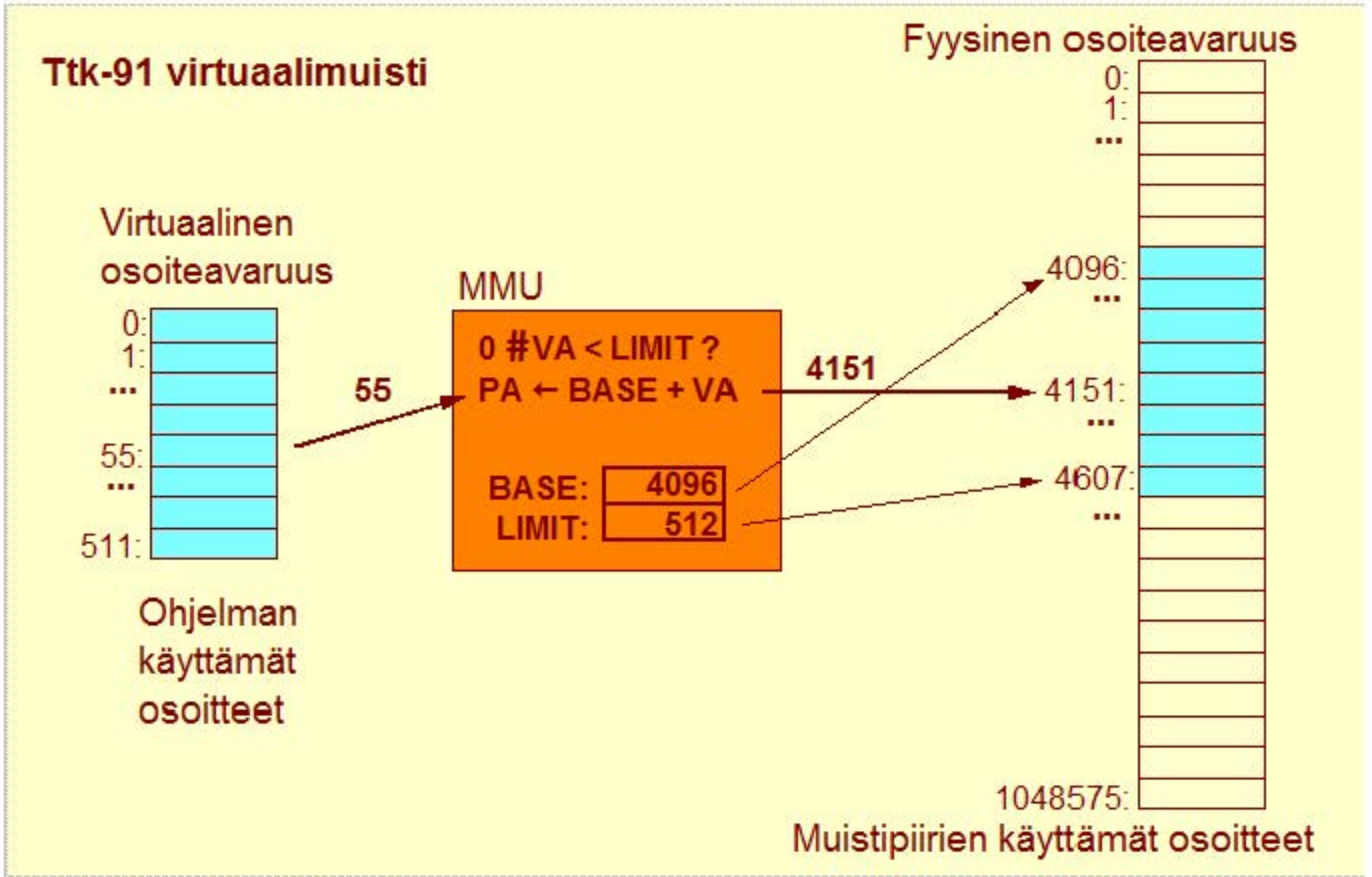
MMU'lle annettua ohjelman käyttämää muistiosoitetta (VA) ei käytetä suoraan, vaan se pitää muuttaa keskusmuistiosoittteeksi (PA)

- tarkista, onko VA laillinen osoite, eli onko VA 0 [0, LIMIT-1]
- jos ei ole, niin aseta SR'n bitti M päälle ja lopeta käskyn suoritus
- muutoin, laske keskusmuistiosoite kaavalla PA ← BASE + VA ja aseta tämä arvo MAR'iin keskusmuistiosoitteeksi



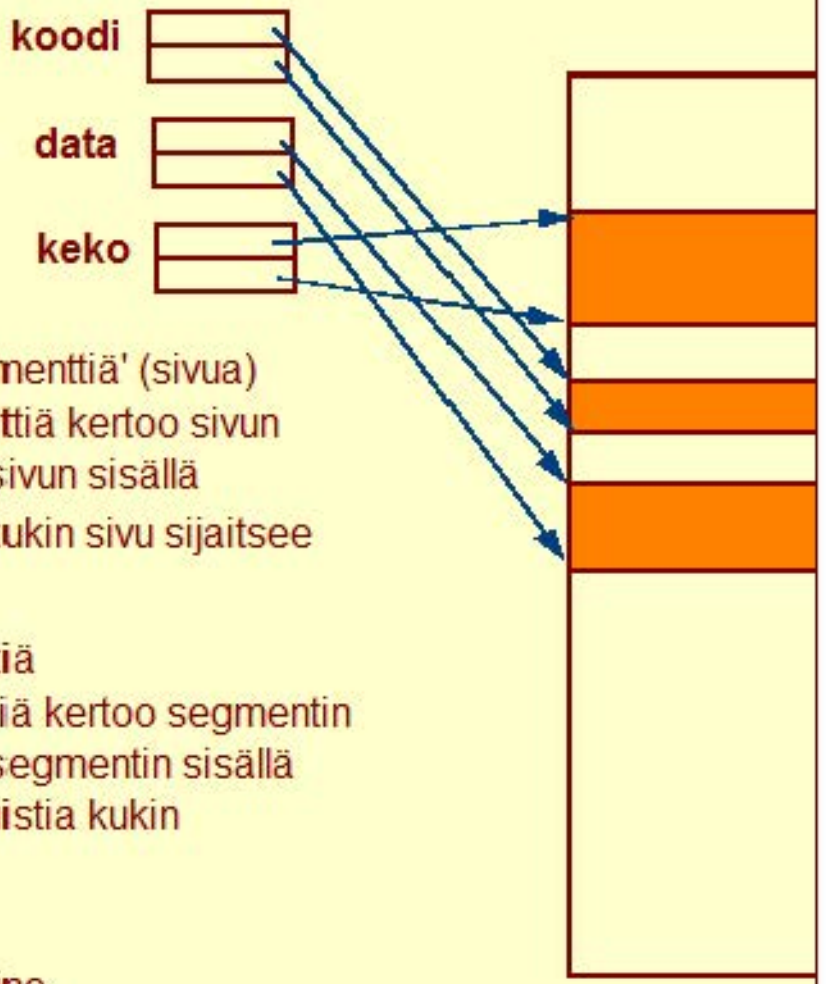
VA = Virtual Address PA = Physical Address

Jokainen MMU'lle annettu muistiosoite pitää nyt (a) tarkistaa ja (b) muuntaa lailliseksi keskusmuistiosoitteeksi. MMU'n laitteisto tekee tämän automaattisesti joka muistiviitteen yhteydessä.



Ttk-91 tukee siis rajarekistereitä käyttävää virtuaalimuistia. Rajarekistereitä on yksi pari, jotka kuvaavat ohjelman käytössä olevan keskusmuistialueen. Todellisissa arkkitehtuureissa käytössä olevia muistisegmenttejä voisi olla useampia, esimerkiksi erikseen koodisegmentille, datasegmentille, pinolle ja keolle. Jokaiselle muistisegmentille tarvittaisiin tällöin oma rajarekisteriparinsa ja joka muistiviitteessä pitäisi myös jollain tavoin ilmaista, mikä muistisegmentti on kyseessä. Ttk-91'ssä tätä ei tarvita, koska kaikki muistiviitteet kohdistuvat yhteen ja samaan muistisegmenttiin.

Virtuaalimuistin osoitteenmuunnosmenetelmät



Kanta- ja rajarekisterit

- base- ja limit-rekisteriparit
- voi olla useita

Sivuttava

- hyvin monta samankokoista pienehköä 'segmenttiä' (sivua)
- muistiosoitteessa ensimmäiset (esim.) 20 bittiä kertoo sivun nron, loput (esim.) 12 bittiä tiedon osoitteen sivun sisällä
- sivutaulu kertoo, missä päin keskusmuistia kukin sivu sijaitsee

Segmentoiva

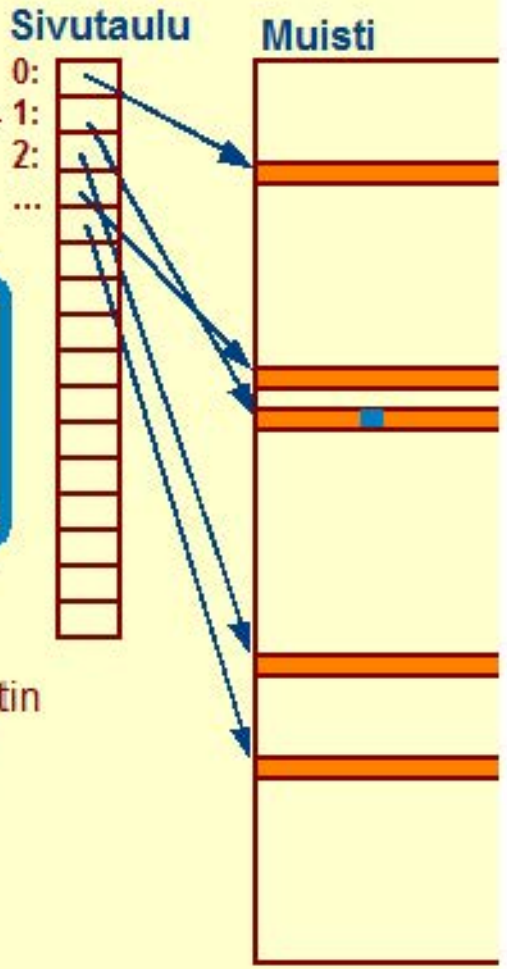
- aika monta eri kokoista suurehkoa segmenttiä
- muistiosoitteessa ensimmäiset (esim.) 8 bittiä kertoo segmentin nron, loput (esim.) 24 bittiä tiedon osoitteen segmentin sisällä
- segmenttitaulu kertoo, missä päin keskusmuistia kukin segmentti sijaitsee

Yhdistelmä

- Segmentoiva sivutus, monitasoinen sivutus, jne.

Virtuaalimuistin osoitteenmuunnosmenetelmiä on itse asiassa useita. Tietyssä suorittimessa käytössä on kuitenkin vain yksi: se, mikä tämän suorittimen MMU'ssa on toteutettuna. Kanta- ja rajarekisteri parien joukko on edelleenkin käytössä oleva menetelmä. On myös mahdollista, että eri muistisegmenttien käyttö on implisiittisesti toteutettu siten, että automaattisesti kaikki koodiviitteet ovat aina koodisegmentissä, kaikki dataviitteet datasegmentissä jne. Etuna usean muistisegmentin käyttämisessä on esimerkiksi keskusmuistin helpompi hallinto, kun uudelle ohjelmalle ei tarvitse löytää suurta yhtenäistä muistilohkoa vaan usea pienempi kelpaa.

Virtuaalimuistin osoitteenmuunnosmenetelmät



Kanta- ja rajarekisterit

- base- ja limit-rekisteriparit
- voi olla useita

Sivuttava

- hyvin monta samankokoista pienehköä 'segmenttiä' (sivua)
- muistiosoitteessa ensimmäiset (esim.) 20 bittiä kertoo sivun nron, loput (esim.) 12 bittiä tiedon osoitteen sivun sisällä
- sivutaulu kertoo, missä päin keskusmuistia kukin sivu sijaitsee

Segmentoiva

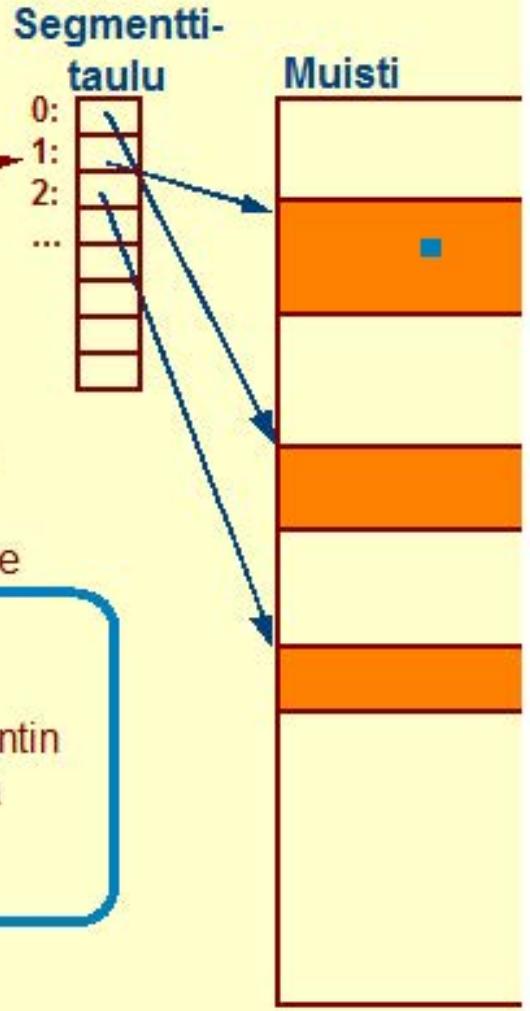
- aika monta eri kokoista suurehkoa segmenttiä
- muistiosoitteessa ensimmäiset (esim.) 8 bittiä kertoo segmentin nron, loput (esim.) 24 bittiä tiedon osoitteen segmentin sisällä
- segmenttitaulu kertoo, missä päin keskusmuistia kukin segmentti sijaitsee

Yhdistelmä

- Segmentoiva sivutus, monitasoinen sivutus, jne.

Sivuttavassa virtuaalimuistissa keskusmuisti on jaettu samankokoisiin pienehköihin (esim. 1-4 KB) muistilohkoihin, sivukehyksiin. Vastaavasti, ohjelman oma muistiosoitteiden joukko on jaettu samankokoisiin sivuihin. Nyt ohjelman mikä tahansa osoitealueen sivu voidaan sijoittaa fyysisesti minne päin tahansa keskusmuistiin vapaaseen sivukehykseen. Muistinhallinta tulee täten vielä helpommaksi. Haittana tässä menetelmässä on jokaisen muistiviitteen yhteydessä tapahtuva osoitteenmuutos sivutaulun kautta.

Virtuaalimuistin osoitteenmuunnosmenetelmät



Kanta- ja rajarekisterit

- base- ja limit-rekisteriparit
- voi olla useita

Sivuttava

- hyvin monta samankokoista pienehköä 'segmenttiä' (sivua)
- muistiosoitteessa ensimmäiset (esim.) 20 bittiä kertoo sivun nron, loput (esim.) 12 bittiä tiedon osoitteen sivun sisällä
- sivutaulu kertoo, missä päin keskusmuistia kukin sivu sijaitsee

Segmentoiva

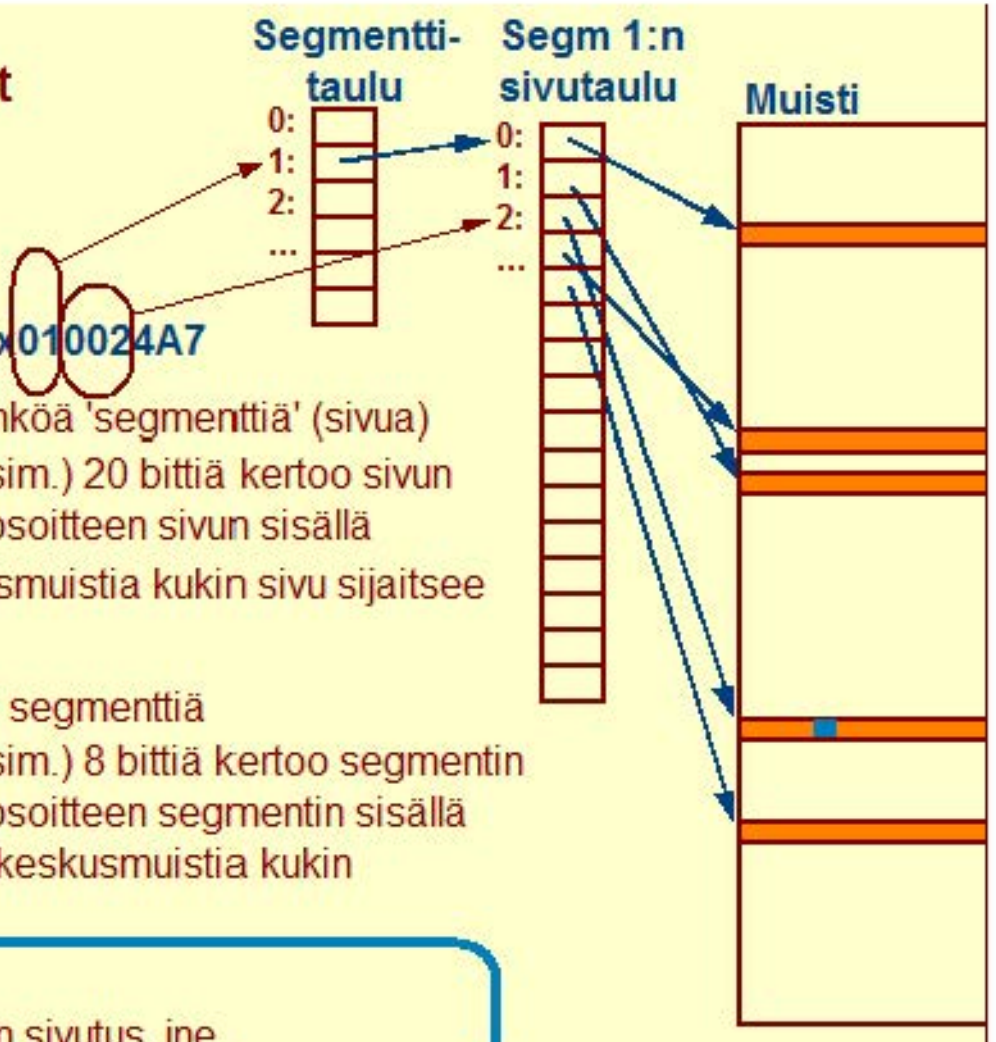
- aika monta eri kokoista suurehkoa segmenttiä
- muistiosoitteessa ensimmäiset (esim.) 8 bittiä kertoo segmentin nron, loput (esim.) 24 bittiä tiedon osoitteen segmentin sisällä
- segmenttitaulu kertoo, missä päin keskusmuistia kukin segmentti sijaitsee

Yhdistelmä

- Segmentoiva sivutus, monitasoinen sivutus, jne.

Kolmas vaihtoehto on segmentoiva virtuaalimuisti. Siinä ohjelman käyttämä muistialue jaetaan loogisiin kokonaisuuksiin, segmentteihin, jotka kukin allokoidaaan keskusmuistista yhtenäisenä muistilohkona. Erona kanta- ja rajarekisteri toteutukseen on, että segmenttejä on tyypillisesti paljon enemmän ja että niihin päästään käsiksi samantapaisella osoitteenmuunnoksella kuin sivuttavassa virtuaalimuistissakin. Etuna segmentoivalla virtuaalimuistilla on, että loogiset kokonaisuudet sijoitetaan aina yhtenäiselle keskusmuistialueelle, jolloin muistin suojausongelmat on helpompi ratkaista.

Virtuaalimuistin osoitteenmuunnosmenetelmät



Kanta- ja rajarekisterit

- base- ja limit-rekisteriparit
- voi olla useita

Sivuttava

- hyvin monta samankokoista pienehköä 'segmenttiä' (sivua)
- muistiosoitteessa ensimmäiset (esim.) 20 bittiä kertoo sivun nron, loput (esim.) 12 bittiä tiedon osoitteen sivun sisällä
- sivutaulu kertoo, missä päin keskusmuistia kukin sivu sijaitsee

Segmentoiva

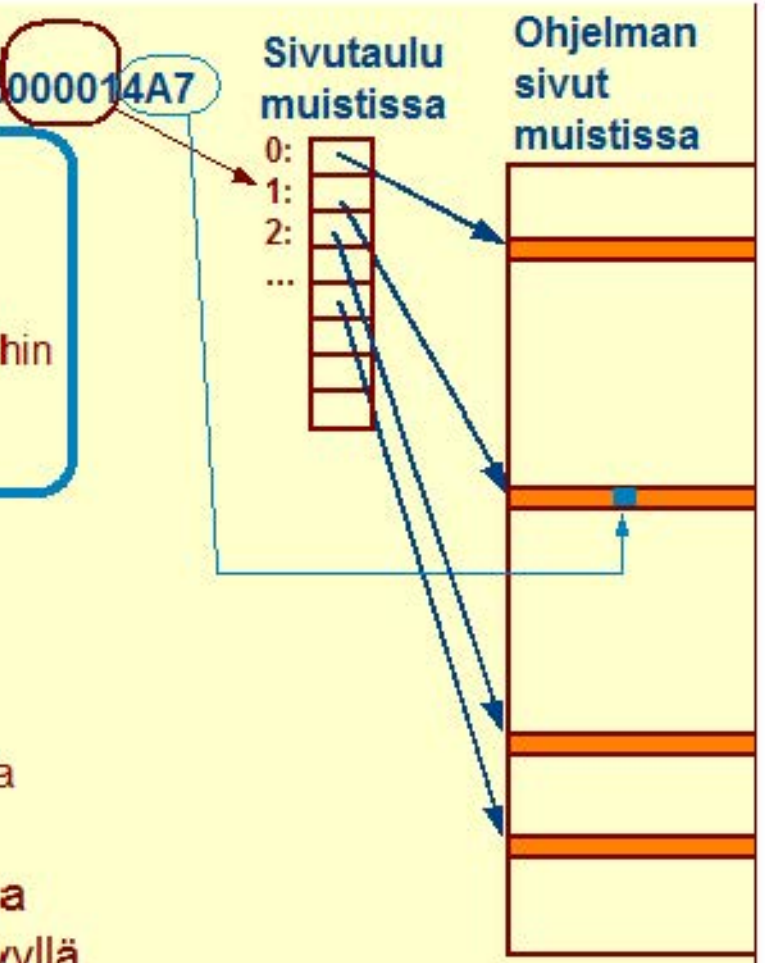
- aika monta eri kokoista suurehkoa segmenttiä
- muistiosoitteessa ensimmäiset (esim.) 8 bittiä kertoo segmentin nron, loput (esim.) 24 bittiä tiedon osoitteen segmentin sisällä
- segmenttitaulu kertoo, missä päin keskusmuistia kukin segmentti sijaitsee

Yhdistelmä

- Segmentoiva sivutus, monitasoinen sivutus, jne.

Käytännössä tilanne on vielä monimutkaisempi. Näillä kaikilla eri lähestymistavoilla on omat hyvät ja huonot puolensa, joten käytännön toteutukset ovat yleensä jonkin sortin monitasoisia yhdistelmiä näistä menetelmistä. Esimerkiksi, voi olla, että muistin suojaus on toteutettu segmentoinnilla ja segmenttien sisällä muistinhallinta on toteutettu tehokkaammalla sivutuksella. Emme käsittele näitä yhdistelmätoteutuksia tämän enempää tällä kurssilla.

Sivuttava virtuaalimuisti



Keskusmuisti on ajettu tasakokoisiin sivukehyksiin ja ohjelman osoiteavaruus saman kokoisiin sivuihin

- mikä tahansa ohjelman sivu voidaan tallettaa mihin tahansa sivukehykseen
- kirjanpito sivutaulussa (iso taulukko muistissa)

Osotteenmuutosta nopeuttaa välimuistin kaltainen osoitteenmuunnospuskuri (TLB, Translation Lookaside Buffer)

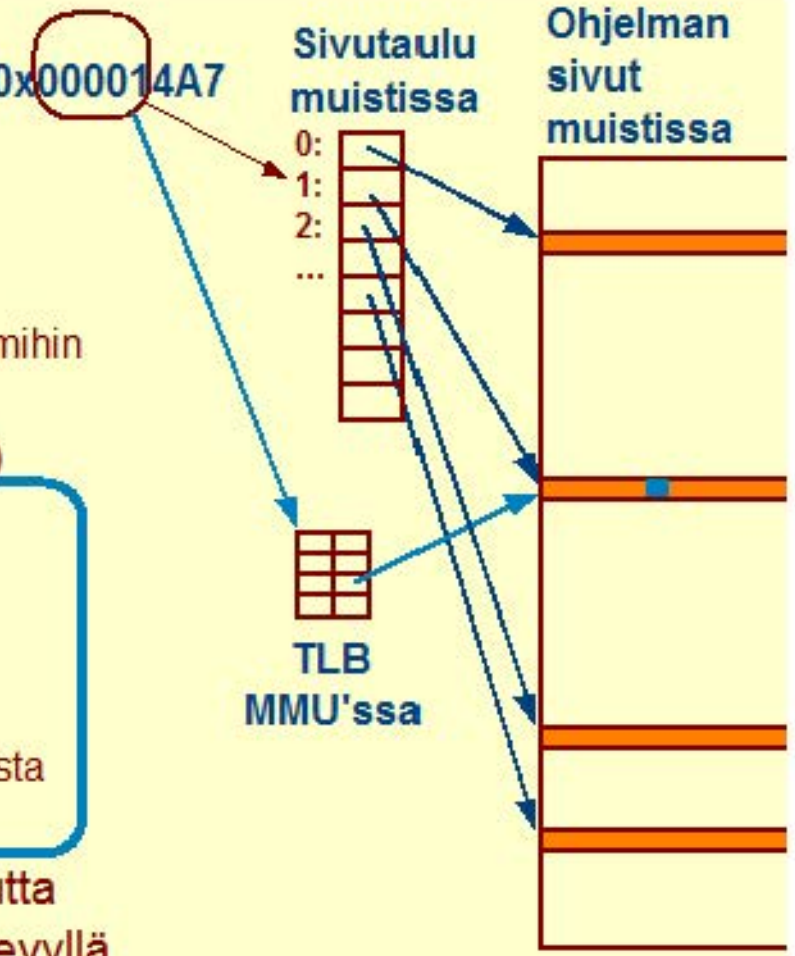
- TLB on osa muistinhallintayksikköä (MMU)
- lähes kaikki (esim. 99.9%) osoitteenmuutoksista löytyy TLB'stä

Kaikki tieto ei sijaitse keskusmuistissa, mutta kaikesta on lähes ajantasalla oleva kopio levyllä

- oma levy tai oma partitio tiedostolevyllä

Kuvaamme tässä sivuttavaa virtuaalimuistia vähän tarkemmin. Perusidea on siis, että muistin allokoinnin optimoimiseksi keskusmuisti on jaettu samankokoisiin sivukehyksiin. Ohjelman käyttämien muistiavaruus eli käytettävien osoitteiden joukko on jaettu samankokoisiin sivuihin ja mikä tahansa sivu voidaan sitten sijoittaa mihin tahansa sivukehykseen. Kirjanpito hoidetaan erillisen sivutaulun avulla. Huonona puolena on, että sivutaulusta voi tulla aika iso ja se on pidettävä koko ajan muistissa suorituskykyvaatimusten vuoksi.

Sivuttava virtuaalimuisti



Keskusmuisti on ajettu tasakokoisiin sivukehyksiin ja ohjelman osoiteavaruus saman kokoisiin sivuihin

- mikä tahansa ohjelman sivu voidaan tallettaa mihin tahansa sivukehykseen
- kirjanpito sivutaulussa (iso taulukko muistissa)

Osotteenmuutosta nopeuttaa välimuistin kaltainen osoitteenmuunnospuskuri (TLB, Translation Lookaside Buffer)

- TLB on osa muistinhallintayksikköä (MMU)
- lähes kaikki (esim. 99.9%) osoitteenmuutoksista löytyy TLB'stä

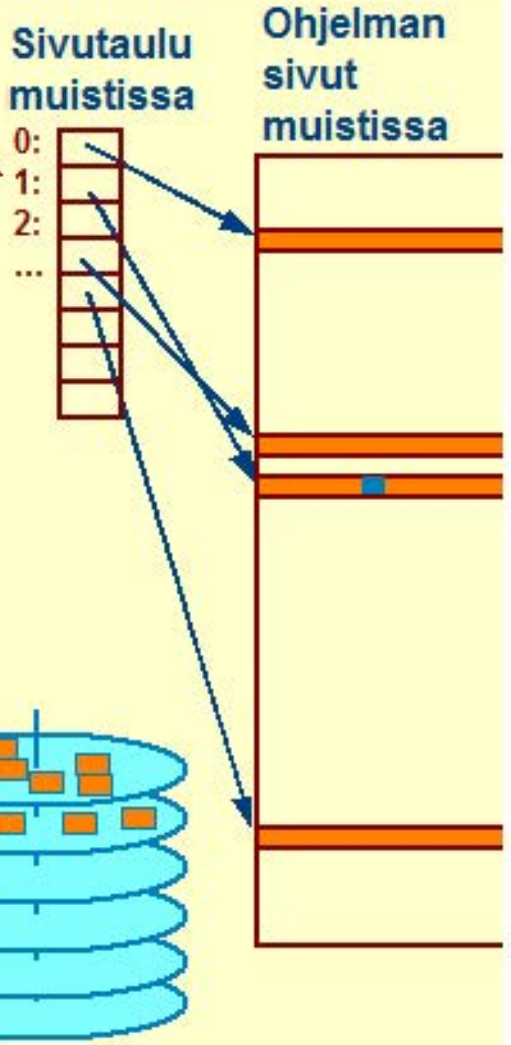
Kaikki tieto ei sijaitse keskusmuistissa, mutta kaikesta on lähes ajantasalla oleva kopio levyllä

- oma levy tai oma partitio tiedostolevyllä

Osoitteenmuutoksen tekeminen muistissa olevan sivutaulun kautta jokaisella muistiviitteellä olisi kuitenkin aivan liian hidasta. Jokaista jo sinällään hidasta muistiviitettä varten kun pitäisi tehdä vielä toinen muistiviite sivutauluun. Tämä ongelman välttämiseksi MMU'ssa on erillinen välimuistin tapainen laitteisto (TLB), jossa on tallessa kaikki viime aikoina tehdyt osoitteenmuutokset. Joka muistiviitteen kohdalla tarkistetaan nyt ensin, jos osoitteenmuunnos löytyy TLB'stä ja onneksi se lähes aina löytyy sieltä. Ilman TLB'tä virtuaalimuistin käyttö olisi aivan liian hidasta sen etuihin verrattuna.

Sivuttava virtuaalimuisti

Osoite: 0x000014A7



Keskusmuisti on ajettu tasakokoisiin sivukehyksiin ja ohjelman osoiteavaruus saman kokoisiin sivuihin

- mikä tahansa ohjelman sivu voidaan tallettaa mihin tahansa sivukehykseen
- kirjanpito sivutaulussa (iso taulukko muistissa)

Osotteenmuutosta nopeuttaa välimuistin kaltainen osoitteenmuunnospuskuri (TLB, Translation Lookaside Buffer)

- TLB on osa muistinhallintayksikköä (MMU)
- lähes kaikki (esim. 99.9%) osoitteenmuutoksista löytyy TLB'stä

Kaikki tieto ei sijaitse keskusmuistissa, mutta kaikesta on lähes ajantasalla oleva kopio levyllä

- oma levy tai oma partitio tiedostolevyllä

Sivuttavassa virtuaalimuistissa on myös se merkittävä piirre, että ohjelman kaikkien tietojen ei tarvitse sijaita keskusmuistissa. Kaikki tiedot pidetään levyllä ja vain kullakin hetkellä tarvittavat sivut pidetään keskusmuistissa. Ei tietenkään ole lainkaan triviaalia määritellä ja toteuttaa käsitettä 'tällä hetkellä tarvittavat sivut'. Kun ohjelma viittaa keskusmuistista puuttuvaan sivuun, MMU generoi sivunpuutoskeskeytyksen ja ohjelman suoritus pysähtyy kunnes käyttöjärjestelmä on saanut kopioitua puuttuvan sivun jonnekin päin keskusmuistia. Muistanette, että tähän levyn lukemiseen kuluva aika on todella pitkä suorittavan ohjelman aikaskaalassa.

Virtuaalimuistin piirteitä

Yleinen muistinhallintaongelma

- miten paljon (keskus)muistitilaa kullekin ohjelmalle?
- onko ohjelmalle annettu muistitilan määrä vakio vai vaihteleeko se suorituksen aikana?
- missä päin muistia kunkin ohjelman muistitila on?
- onko ohjelmalle käytössä oleva muistitila yhtenäinen vai paloittainen alue?
- onko ohjelman käytössä oleva muistitila samassa kohtaa koko suorituksen ajan?

Samalla ratkotaan 'helposti' muita ongelmia

- kirjanpito eri ohjelmien muistitilan käytöstä
- yhden ohjelman muistitilan suojaus muilta samaan aikaan järjestelmässä olevilta ohjelmilta
- entä jos ohjelma tarvitsee enemmän muistitilaa kun on vapaana?
- entä jos ohjelma tarvitsee enemmän muistitilaa kuin koneessa on?

Virtuaalimuistin päätarkoituksena on avittaa yleisen muistinhallintaongelman ratkaisussa. Ratkaisumenetelmiä on hyvin erilaisia ja mitään ainoata oikeata tapaa toteuttaa muistinhallinta ei ole. Esimerkiksi, joissakin järjestelmissä kullekin ohjelmalle annetaan ohjelman käynnistyessä tietty määrä keskusmuistia käyttöönsä ja sillä tulee pärjätä. Joissakin muissa järjestelmissä taas ohjelmalle allokoidun keskusmuistin määrää voidaan lisätä (tai pienentää) ohjelman suoritusaikana. Sivuttavassa virtuaalimuistissa yhden ohjelman käytössä oleva keskusmuistialue vaihtelee sijainniltaan dynaamisesti koko ajan suorituksen edetessä.

Virtuaalimuistin piirteitä

Yleinen muistinhallintaongelma

- miten paljon (keskus)muistitilaa kullekin ohjelmalle?
- onko ohjelmalle annettu muistitilan määrä vakio vai vaihteleeko se suorituksen aikana?
- missä päin muistia kunkin ohjelman muistitila on?
- onko ohjelmalle käytössä oleva muistitila yhtenäinen vai paloittainen alue?
- onko ohjelman käytössä oleva muistitila samassa kohtaa koko suorituksen ajan?

Samalla ratkotaan 'helposti' muita ongelmia

- kirjanpito eri ohjelmien muistitilan käytöstä
- yhden ohjelman muistitilan suojaus muilta samaan aikaan järjestelmässä olevilta ohjelmilta
- entä jos ohjelma tarvitsee enemmän muistitilaa kun on vapaana?
- entä jos ohjelma tarvitsee enemmän muistitilaa kuin koneessa on?

Virtuaalimuistin käytännön ratkaisuja monimutkaistaa vielä se, että samassa yhteydessä on usein ratkottu myös muita ongelmia. Esimerkiksi, on kätevää yhdistää muistin suojausongelman ratkaisu sivuttavan virtuaalimuistin sivutauluihin. Kunkin sivun kohdalle voidaan sivutauluun tämänhetkisen sivukehyksen lisäksi laittaa myös tieto siitä, mitkä kaikki ohjelmat saavat kyseiseen sivuun viitata ja millä tavalla. Toisaalta, sivuttavassa virtuaalimuistissa voidaan luontevasti suorittaa ohjelmia, joiden osoiteavaruus on isompi kuin keskusmuistin koko, kunhan vain 'kullakin hetkellä tarvittavien sivujen joukko' mahtuu muistiin.

Virtuaalimuistin ongelmia

Joka muistiviitteen yhteydessä täytyy tehdä aika monimutkainen kuvaus virtuaaliosoitteesta (VA) fyysiseen keskusmuistiosoitteeseen (PA)

- TLB auttaa, mutta aina silloin tällöin tieto pitää hakea muistissa olevasta sivutaulusta

Sivutaulut ovat suuria ja vievät paljon muistitilaa

- nyt juuri käyttämättömät sivutaulun osat voi pistää levylle

Aina joskus tulee viite sivuun, joka ei sijaitse keskusmuistissa

- hyvin suuri kustannus (aika)
- lääke: isompi keskusmuisti,
- lääke: lataa heti alkuun tarvittavat sivut muistiin ohjelman käynnistyessä

Lisää tietoa? ➔ Tietokoneen rakenne ja käyttöjärjestelmäkurssit

Virtuaalimuistin toteutukseen liittyy muutamia ongelma-alueita. Olennainen osa mitä tahansa virtuaalimuistia on osoitteenmuunnos, jota varten MMU'hun on rakennettu erikoislaitteisto. Kanta- ja rajarekisteriin perustuva osoitteenmuunnos on hyvin deterministinen, mutta sivuttavan virtuaalimuistin TLB'stä ei aina löydy tarvittavaa osoitteenmuunnosta. Tällaisen 'TLB-hudin' hintana on yksi tai useampi muistiviite, mikä ihan oikeasti hidastaa laskentaa.

Virtuaalimuistin ongelmia

Joka muistiviitteen yhteydessä täytyy tehdä aika monimutkainen kuvaus virtuaaliosoitteesta (VA) fyysiseen keskusmuistiosoitteeseen (PA)

- TLB auttaa, mutta aina silloin tällöin tieto pitää hakea muistissa olevasta sivutaulusta

Sivutaulut ovat suuria ja vievät paljon muistitilaa

- nyt juuri käyttämättömät sivutaulun osat voi pistää levylle

Aina joskus tulee viite sivuun, joka ei sijaitse keskusmuistissa

- hyvin suuri kustannus (aika)
- lääke: isompi keskusmuisti,
- lääke: lataa heti alkuun tarvittavat sivut muistiin ohjelman käynnistyessä

Lisää tietoa? ➔ Tietokoneen rakenne ja käyttöjärjestelmäkurssit

Joka ohjelmalle tarvitaan oma sivutaulu ja ne vaativat huomattavan määrän muistitilaa, jota voisi käyttää muuhunkin. Eräs mahdollisuus on siirtää käyttämättömät osat sivutaulusta levylle, mutta tällöin taas ongelmaksi muodostuu tuon epämääräisen käsitteen 'käyttämättömät osat' määrittely ja toteutus. Ongelmana on nyt myös TLB-hudin vielä korkeampi hinta, jos osoitteenmuunnoksen yhteydessä pitää tehdä jopa levyltä luku, jotta puuttuva sivutaulun osa ensin saadaan paikalleen.

Virtuaalimuistin ongelmia

Joka muistiviitteen yhteydessä täytyy tehdä aika monimutkainen kuvaus virtuaaliosoitteesta (VA) fyysiseen keskusmuistiosoitteeseen (PA)

- TLB auttaa, mutta aina silloin tällöin tieto pitää hakea muistissa olevasta sivutaulusta

Sivutaulut ovat suuria ja vievät paljon muistitilaa

- nyt juuri käyttämättömät sivutaulun osat voi pistää levylle

Aina joskus tulee viite sivuun, joka ei sijaitse keskusmuistissa

- hyvin suuri kustannus (aika)
- lääke: isompi keskusmuisti,
- lääke: lataa heti alkuun tarvittavat sivut muistiin ohjelman käynnistyessä

Lisää tietoa? ➔ Tietokoneen rakenne ja käyttöjärjestelmäkurssit

Yleisin virtuaalimuistin ongelma liittyy kuitenkin suorituskykyyn. Jos vain osa ohjelman käyttämästä muistiavaruudesta todellisuudessa sijaitsee keskusmuistissa, niin aika ajoin tietoa pitää lukea levyltä. Tämä hidastaa ohjelman suoritusta niin merkittävästi, että joillakin sovellusalueilla (esim. realiaikasovellukset) pitää erikoisjärjestelyin taata, että sovellus toimisi nopeasti virtuaalimuistista huolimatta. Joissakin järjestelmissä voidaan tällaiset ohjelmat 'kiinnittää' muistiin siten, että niiden koko muistiavaruus on ohjelman suoritusaikana koko ajan keskusmuistissa.

Virtuaalimuistin ongelmia

Joka muistiviitteen yhteydessä täytyy tehdä aika monimutkainen kuvaus virtuaaliosoitteesta (VA) fyysiseen keskusmuistiosoitteeseen (PA)

- TLB auttaa, mutta aina silloin tällöin tieto pitää hakea muistissa olevasta sivutaulusta

Sivutaulut ovat suuria ja vievät paljon muistitilaa

- nyt juuri käyttämättömät sivutaulun osat voi pistää levylle

Aina joskus tulee viite sivuun, joka ei sijaitse keskusmuistissa

- hyvin suuri kustannus (aika)
- lääke: isompi keskusmuisti,
- lääke: lataa heti alkuun tarvittavat sivut muistiin ohjelman käynnistyessä

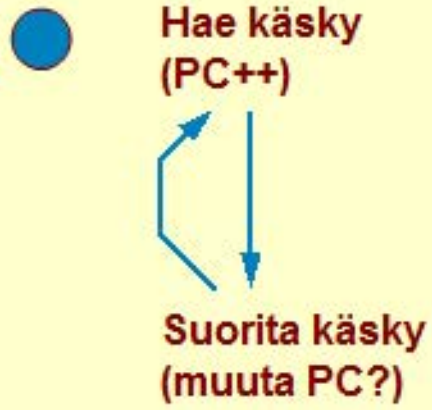
Lisää tietoa? ➔ Tietokoneen rakenne ja käyttöjärjestelmäkurssit

Emme käsittele virtuaalimuistia tämän enempää tällä kurssilla.Jos haluatte lisää tietoja sen laitteistopuolen toteutuksesta (esim. TLB'stä), niin menkää Tietokoneen rakenne -kurssille. Ohjelmistopuolen toteutusasiat taas löytyvät laitoksen käyttöjärjestelmäkursseilta. Pääosa virtuaalimuistin toteutusta on nimenomaan ohjemistoa ja se vaihtelee käyttöjärjestelmien (ja niiden versioiden) mukaan.

Keskeytystilanteet

Mikä tahansa tilanne, jonka käsittely vaatii poikkeuksen käskyjen normaaliin suoritusjärjestykseen

- tilanne itsessään ei ole yllättävä, ainoastaan sen tapahtuma-aika!
- kaikki keskeytystilanteet on ennalta tunnettuja ja niihin on varauduttu



Rakkaalla lapsella on monta nimeä

- poikkeus, keskeytys, virhetilanne, trappi, KJ-kutsu, ...
- exception, interrupt, fault, trap, failure, SVC, ...

Jatkossa yleisnimi keskeytys tai poikkeus tarkoittaa kaikkia näitä tapauksia tai tyyppejä

- ne kaikki ratkaistaan samalla tavalla

Aikaisemmin esitetty käskyjen nouto- ja suoritussykli oli hieman yksinkertaistettu esitys. Todellisuudessa on myös tilanteita, joissa tämä normaali sykli pitää keskeyttää jonkin akuutin tapahtuman vuoksi. On huomattavaa, että nämä akuutit tapahtumat eivät sinällään ole yllättäviä tai tuntemattomia, vaan ainoastaan niiden tapahtumishetki on. Kaikki mahdolliset keskeytystilanteet on etukäteen tiedossa, mutta niiden täsmällinen tapahtuma-aika ei (välttämättä) ole.

Keskeytystilanteet

Mikä tahansa tilanne, jonka käsittely vaatii poikkeuksen käskyjen normaaliin suoritusjärjestykseen

- tilanne itsessään ei ole yllättävä, ainoastaan sen tapahtuma-aika!
- kaikki keskeytystilanteet on ennalta tunnettuja ja niihin on varauduttu

Rakkaalla lapsella on monta nimeä

- poikkeus, keskeytys, virhetilanne, trappi, KJ-kutsu, ...
- exception, interrupt, fault, trap, failure, SVC, ...

Jatkossa yleisnimi keskeytys tai poikkeus tarkoittaa kaikkia näitä tapauksia tai tyyppejä

- ne kaikki ratkaistaan samalla tavalla

Erilaisilla keskeytystilanteilla on hyvin monta nimeä. Suomenkielisiä ovat esimerkiksi poikkeus, keskeytys, virhetilanne, trappi, KJ-kutsu, jne. Englanninkielisiä ovat exception, interrupt, fault, trap, failure, SVC, etc. Eri järjestelmät käyttävät eri nimiä, eikä nimien käytössä ole juurikaan yhtenäistä käytäntöä. Se mitä yhdessä sanotaan exception'iksi, voi toisessa olla fault. Ja käsite interrupt voi yhdessä systeemissä tarkoittaa hieman eri asiaa kuin toisessa. Ne ovat kuitenkin perusidealtaan kaikki hyvin samankaltaisia ja niiden toteutusmekanismi on yhtenäinen.

Keskeytystilanteet

Mikä tahansa tilanne, jonka käsittely vaatii poikkeuksen käskyjen normaaliin suoritusjärjestykseen

- tilanne itsessään ei ole yllättävä, ainoastaan sen tapahtuma-aika!
- kaikki keskeytystilanteet on ennalta tunnettuja ja niihin on varauduttu

Rakkaalla lapsella on monta nimeä

- poikkeus, keskeytys, virhetilanne, trappi, KJ-kutsu, ...
- exception, interrupt, fault, trap, failure, SVC, ...

Jatkossa yleisnimi keskeytys tai poikkeus tarkoittaa kaikkia näitä tapauksia tai tyyppejä

- ne kaikki ratkaistaan samalla tavalla

Tällä kurssilla käsittelemme näitä kaikkia yhtenäisesti ja käytämme niistä yleisnimityksiä poikkeus tai keskeytys. Muistakaa kuitenkin, että jatkossa esitetyt asiat pätevät kaikkiin edellämainittuihin tapauksiin, oli niiden nimi sitten mikä tahansa.

Keskeytysten käsittely

Jokainen mahdollinen keskeytystyyppi on ennalta tunnettu

- mitään yllättävää ei siis tapahdu
- ainoastaan mahdollisen keskeytyksen tapahtuma-aika ei ole ennalta tiedossa

Jokaiselle keskeytystyypille on oma käyttöjärjestelmän tuntema keskeytyskäsittelyrutiini

interrupt handler

- kutsutaan aina kyseisen keskeytyksen tapahtuessa, "yllättävä aliohjelmakutsu"

Keskeytykset suoritetaan suorittimen etuoikeutetussa suoritustilassa

- tilarekisterin bitti P (Priviledged)

Jokaisen käskyn suorituksen jälkeen tarkistetaan mahdollisten keskeytysten olemassaolo SR'stä ja haaraudutaan keskytyskäsittelijään tarvittaessa

- paluu keskeytyskäsittelijästä 'return from interrupt' -konekäskyllä

IRET

Keskeytykset voi olla estetty

DISABLE

ENABLE

- tilarekisterin bitti D (interrupts Disabled)

Keskeytystyyppejä on siis useita erilaisia. Ne on kuitenkin kaikki ennalta tunnettuja, eikä niissä ole mitään yllättävää. Edes laskennassa tapahtuneet virhetilanteet eivät tässä mielessä laitteiston kannalta ole varsinaisia virheitä. Esimerkiksi nollalla jakaminen ei sinällään ole väärin, vaan ainoastaan sen tulos on hyvin erilainen kuin mitä jakolaskussa yleensä tapahtuu. Vaikka olemmekin siis varautuneet kaikkiin mahdollisiin keskeytystilanteisiin, niin emme silti tiedä milloin, jos koskaan, ne tapahtuvat. Juuri tämä tapahtuma-ajan epädeterministisyys tekee keskeytyksistä hankalia. Meidän täytyy varautua siihen, että keskeytys voi tulla milloin vain, minkä tahansa konekäskyn aikana.

Keskeytysten käsittely

Jokainen mahdollinen keskeytystyyppi on ennalta tunnettu

- mitään yllättävää ei siis tapahdu
- ainoastaan mahdollisen keskeytyksen tapahtuma-aika ei ole ennalta tiedossa

Jokaiselle keskeytystyypille on oma käyttöjärjestelmän tuntema keskeytyskäsittelyrutiini — interrupt handler

- kutsutaan aina kyseisen keskeytyksen tapahtuessa, "yllättävä aliohjelmakutsu"

Keskeytykset suoritetaan suorittimen etuoikeutetussa suoritustilassa

- tilarekisterin bitti P (Priviledged)

Jokaisen käskyn suorituksen jälkeen tarkistetaan mahdollisten keskeytysten olemassaolo SR'stä ja haaraudutaan keskytyskäsittelijään tarvittaessa

- paluu keskeytyskäsittelijästä 'return from interrupt' -konekäskyllä — IRET

Keskeytykset voi olla estetty — DISABLE

- tilarekisterin bitti D (interrupts Disabled) — ENABLE

Keskeytysten käsittely on järjestelmässä toteutettu siten, että jokaiselle keskeytystyypille on oma aliohjelmansa, jota jollain tavalla kutsutaan kyseisen keskeytyksen sattuessa. Näitä aliohjelmia kutsutaan keskeytyskäsittelijöiksi ja niihin päästään käsiksi milloin vain koko järjestelmässä tunnettujen osoitteiden perusteella. Koska keskeytys voi periaatteessa tapahtua milloin vain, niin voimme ajatella keskeytysten käsittelyn olevan yllättävä aliohjelmakutsu.

Keskeytysten käsittely

Jokainen mahdollinen keskeytystyyppi on ennalta tunnettu

- mitään yllättävää ei siis tapahdu
- ainoastaan mahdollisen keskeytyksen tapahtuma-aika ei ole ennalta tiedossa

Jokaiselle keskeytystyypille on oma käyttöjärjestelmän tuntema keskeytyskäsittelyrutiini

interrupt handler

- kutsutaan aina kyseisen keskeytyksen tapahtuessa, "yllättävä aliohjelmakutsu"

Keskeytykset suoritetaan suorittimen etuoikeutetussa suoritustilassa

- tilarekisterin bitti P (Priviledged)

Jokaisen käskyn suorituksen jälkeen tarkistetaan mahdollisten keskeytysten olemassaolo SR'stä ja haaraudutaan keskytyskäsittelijään tarvittaessa

- paluu keskeytyskäsittelijästä 'return from interrupt' -konekäskyllä

IRET

Keskeytykset voi olla estetty

DISABLE

ENABLE

- tilarekisterin bitti D (interrupts Disabled)

Useat keskeytyskäsittelyrutiinit joutuvat käsittelemään käyttöjärjestelmän sisäisiä tietorakenteita ja sen vuoksi ne kaikki ttk-91'ssä toimivat suorittimen etuoikeutetussa suoritustilassa. Todellisessa koneessa voi olla, että vain jotkut keskeytyskäsittelijät toimivat etuoikeutetussa tilassa, mutta ainakin jotkut toimivat näin. Etuoikeutettu suoritustila tietenkin edellyttää korkeita laatuvaatimuksia keskeytyskäsittelijöiltä, koska ne voivat käsitellä minkä tahansa ohjelman mitä tahansa tietorakenteita. Keskeytyskäsittelijät myös aina huolehtivat, että mitään luottamuksellisia tietoja ei vuoda takaisin keskeytyneelle ohjelmalle.

Keskeytysten käsittely

Jokainen mahdollinen keskeytystyyppi on ennalta tunnettu

- mitään yllättävää ei siis tapahdu
- ainoastaan mahdollisen keskeytyksen tapahtuma-aika ei ole ennalta tiedossa

Jokaiselle keskeytystyypille on oma käyttöjärjestelmän tuntema keskeytyskäsittelyrutiini

interrupt handler

- kutsutaan aina kyseisen keskeytyksen tapahtuessa, "yllättävä aliohjelmakutsu"

Keskeytykset suoritetaan suorittimen etuoikeutetussa suoritustilassa

- tilarekisterin bitti P (Priviledged)

Jokaisen käskyn suorituksen jälkeen tarkistetaan mahdollisten keskeytysten olemassaolo SR'stä ja haaraudutaan keskytyskäsittelijään tarvittaessa

- paluu keskeytyskäsittelijästä 'return from interrupt' -konekäskyllä

IRET

Keskeytykset voi olla estetty

DISABLE
ENABLE

- tilarekisterin bitti D (interrupts Disabled)

Keskeytysten käsittely tapahtuu käskyjen suoritussykliä muokkaamalla. Jokaisen käskyn lopussa tarkistetaan nyt mahdollisten keskeytysten olemassaolo tilarekisteristä (SR), ja, jos jokin keskeytysbitti on päällä, niin haaraudutaan kyseisen keskeytystyypin keskeytyskäsittelyrutiiniin. Haarautuminen toteutetaan yksinkertaisesti sijoittamalla PC'n arvoksi keskeytyskäsittelijän alkuosoite. Toteutus on siis hyvin paljon hyppy- tai CALL-käskyn kaltainen, vaikka kohdeosoitetta ei annetakaan ohjelmakoodissa. Paluu keskeytyskäsittelyrutiinista muistuttaa hyvin paljon aliohjelmasta paluuta, ja se tapahtuu etuoikeutetulla IRET-käskyllä. Paluun yhteydessä suorittimen suoritustila palautetaan ennalleen.

Keskeytysten käsittely

Jokainen mahdollinen keskeytystyyppi on ennalta tunnettu

- mitään yllättävää ei siis tapahdu
- ainoastaan mahdollisen keskeytyksen tapahtuma-aika ei ole ennalta tiedossa

Jokaiselle keskeytystyypille on oma käyttöjärjestelmän tuntema keskeytyskäsittelyrutiini **interrupt handler**

- kutsutaan aina kyseisen keskeytyksen tapahtuessa, "yllättävä aliohjelmakutsu"

Keskeytykset suoritetaan suorittimen etuoikeutetussa suoritustilassa

- tilarekisterin bitti P (Priviledged)

Jokaisen käskyn suorituksen jälkeen tarkistetaan mahdollisten keskeytysten olemassaolo SR'stä ja haaraudutaan keskytyskäsittelijään tarvittaessa

- paluu keskeytyskäsittelijästä 'return from interrupt' -konekäskyllä **IRET**

Keskeytykset voi olla estetty **DISABLE** **ENABLE**

- tilarekisterin bitti D (interrupts Disabled)

Sovelluksissa ja erityisesti käyttöjärjestelmärutiineissa voi helposti olla tilanteita, joissa jokin tietty koodinpätkä pitää suorittaa yhteen menoon loppuun, ilman mitään keskeytyksiä. Kyseessä voi olla vaikkapa jonkin tietorakenteen sisäinen eheysvaatimus tai jokin muu samanaikaisuuden hallintaan liittyvä vaatimus. Yksi lähetymistapa tällaisten ongelmien ratkaisuun on keskeytysten estäminen, jolloin yksinkertaisesti käskyn lopussa ei tarkisteta keskeytysten olemassaoloa, vaan käskyjä suoritetaan puhtaasti ohjelman määräämässä järjestyksessä. Keskeytysten esto/salliminen tapahtuu omilla konekäskyillään. Yleensä keskeytykset estetään vain lyhyeksi, esim. 10-20 konekäskyn suorituksen ajaksi kerrallaan.

Keskeytyslajit

Käskyn aiheuttamat virhetilanteet

Käskyn aiheuttamat muut poikkeustilanteet

- ei siis mikään virhetilanne, vaan haluttu deterministinen käyttäytyminen
- johtaa aina keskeytyksen tapahtumiseen
- jokin erikoistoimenpide, jonka on toteutettu keskeytyskäsittelyn avulla

Ulkoapäin suorittimelle tulleet signaalit

- suorittimen ulkopuolelta, kontrolliväylän kautta

Keskeytyksiä on kolmea eri lajia, jotka poikkeavat merkittävästi toisistaan. Ensimmäinen laji on ns. virhetilanteet. Virhetilanne on mikä tahansa nyt suoritettavan konekäskyn aikana tapahtuva virhe- tai poikkeustilanne, jonka vuoksi tätä konekäskyä ei tällä kertaa voida suorittaa tavalliseen tapaan loppuun. Tilanne on siis yllättävä ja sen on aiheuttanut juuri suorituksessa oleva konekäsky. Esimerkkinä voi olla vaikkapa konekäskyssä oleva virheellinen operaatiokoodi. Keskeytyskäsittelyn jälkeen tämän ohjelman suoritus voi joko jatkua tai sitten ei, poikkeustilanteesta riippuen.

Keskeytyslajit

Käskyn aiheuttamat virhetilanteet

Käskyn aiheuttamat muut poikkeustilanteet

- ei siis mikään virhetilanne, vaan haluttu deterministinen käyttäytyminen
- johtaa aina keskeytyksen tapahtumiseen
- jokin erikoistoimenpide, jonka on toteutettu keskeytyskäsittelyn avulla

Ulkoapäin suorittimelle tulleet signaalit

- suorittimen ulkopuolelta, kontrolliväylän kautta

Toinen keskeytyslaji on myös suorituksessa olevan konekäskyn aiheuttama, mutta tällä kertaa kyseessä on jokin tietty toiminto, jonka toteutus on vain todettu käteväksi tehdä keskeytyskäsittelyn kautta. Esimerkiksi käyttöjärjestelmän palvelukutsu (SVC) on tällainen toiminto. Kyseessä ei siis ole mikään virhetilanne ja tämän ohjelman suoritus jatkuu normaalisti keskeytyskäsittelijästä paluun jälkeen.

Keskeytyslajit

Käskyn aiheuttamat virhetilanteet

Käskyn aiheuttamat muut poikkeustilanteet

- ei siis mikään virhetilanne, vaan haluttu deterministinen käyttäytyminen
- johtaa aina keskeytyksen tapahtumiseen
- jokin erikoistoimenpide, jonka on toteutettu keskeytyskäsittelyn avulla

Ulkoapäin suorittimelle tulleet signaalit

- suorittimen ulkopuolelta, kontrolliväylän kautta

Keskeytys voi myös aiheutua suorittimen ulkopuolelta tulleen signaalin kautta. Esimerkiksi jokin levyohjain voi tällaisen I/O-laitekeskeytyksen avulla kertoa käyttöjärjestelmälle, että 'antamasi homma on tehty, mitä nyt?' Tämä keskeytyslaji eroaa edellisistä siitä, että näillä keskeytyksillä ei yleensä ole mitään tekemistä suoritettavan olleen konekäskyn tai edes tämän ohjelman kanssa, vaan että sillä halutaan antaa suorittimen suoritusvuoro vähäksi aikaa jollekin käyttöjärjestelmän palvelurutiinille. Keskeytysmekanismilla on siis ratkaistu hyvin monenlaisia järjestelmän ongelmia.

Konekäskyn aiheuttamat virhetilanteet

Virheellinen käskyosoite	Invalid code address
Virheellinen dataosoite	Invalid data address
Tuntematon käsky	Invalid opcode
Nollalla jako	Divide by Zero
Alustamaton data	Invalid operand
Kokonaisluvun ylivuoto	Integer overflow
Liukuluvun ylivuoto/alivuoto	Floating point overflow Floating point underflow
Virtuaalimuistiosoitteen kuvaama data ei ole keskusmuistissa	Page fault Segment fault

Yhdellä konekäskyllä on vain muutama mahdollinen virhetilanne. Jotkut virhetilanteet voivat tapahtua millä tahansa konekäskyllä, kun taas jotkut toiset ovat konekäskykohtaisia. Käskyn tai datan virheellinen osoite voi sattua mille tahansa käskylle. Virheellinen käskyn tai datan osoite tapahtuu esimerkiksi osoitinmuuttujien tai taulukkoindeksien suoritusaikaisen manipuloinnin vuoksi, varsinkin jos kääntäjä tai ohjelmoija ei ole generoinut viittausalueen validisuutta tarkistavaa koodia.

Konekäskyn aiheuttamat virhetilanteet

Virheellinen käskyosoite		Invalid code address
Virheellinen dataosoite		Invalid data address
Tuntematon käsky	5 = 0x00000005	Invalid opcode
Nollalla jako		Divide by Zero
Alustamaton data		Invalid operand
Kokonaisluvun ylivuoto		Integer overflow
Liukuluvun ylivuoto/alivuoto		Floating point overflow Floating point underflow
Virtuaalimuistiosoitteen kuvaama data ei ole keskusmuistissa		Page fault Segment fault

Virheellinen operaatiokoodi voi tapahtua järjestelmissä, missä on mahdollista suorittaa datasegmenttiä koodina, esimerkiksi ttk-91'ssä. Todellisissa järjestelmissä koodi- ja data-alueet on eristetty, jolloin virhetilanne havaitaan jo käskyä luettaessa edellämainitun 'virheellinen käskyosoite' -keskeytyksen muodossa. Ttk-91 koneessa esimerkiksi kokonaisluvun 5 32-bittisessä esitystavassa 8 ensimmäistä bittiä ovat kaikki nollia, ja jos kyseistä kokonaislukua yritettäisiin tulkita konekäskynä, niin operaatiokoodi 0 havaittaisiin virheelliseksi. Tämän piirteen vuoksi useimmissa todellisissa konekielissäkin opcode 0 aiheuttaa virhetilanteen.

Konekäskyn aiheuttamat virhetilanteet

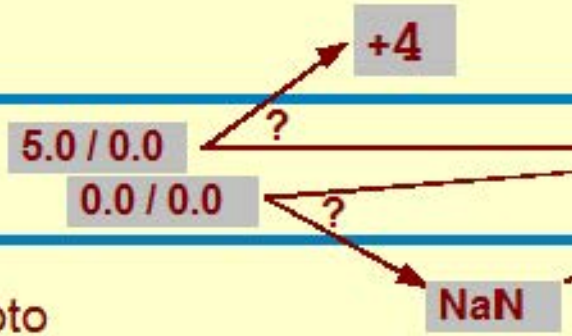
Virheellinen käskyosoite — Invalid code address

Virheellinen dataosoite — Invalid data address

Tuntematon käsky — Invalid opcode

Nollalla jako — Divide by Zero

Alustamaton data — Invalid operand



Kokonaisluvun ylivuoto — Integer overflow

Liukuluvun ylivuoto/alivuoto — Floating point overflow, Floating point underflow

Virtuaalimuistiosoitteen kuvaama data ei ole keskusmuistissa — Page fault, Segment fault

Tyypillinen virhetilanne on aritmeettinen nollalla jako, minkä operaation tulos ei ole hyvin määritelty. Liukuluvuilla tosin on mahdollista, että nollalla jaosta ei tulekaan keskeytystä, vaan tulos voi olla +/- ääretön. Yleisesti käytetyssä IEEE'n liukulukustandardissa kun on esitystapa myös käsitteille +/- ääretön. Toisaalta, jos myös jaettava on nolla, niin tuloksena on virhetilanne tai sitten myös tähän IEEN'n standardiin kuuluva määrittelemätön luku NaN (Not a Number). Lukujen +/- ääretön ja NaN käyttäminen jatkossa muiden matemaattisten operaatioiden operandina johtaa helposti 'alustamaton data' tms -keskeytykseen.

Konekäskyn aiheuttamat virhetilanteet

Virheellinen käskyosoite		Invalid code address
Virheellinen dataosoite		Invalid data address
Tuntematon käsky		Invalid opcode
Nollalla jako		Divide by Zero
Alustamaton data		Invalid operand
Kokonaisluvun ylivuoto	1000 0001 + 1000 0001 = 0000 0002 ?	Integer overflow
Liukuluvun ylivuoto/alivuoto		Floating point overflow Floating point underflow
Virtuaalimuistiosoitteen kuvaama data ei ole keskusmuistissa		Page fault Segment fault

Aritmeettisista operaatioista voi myös olla tuloksena luku, joka on yksinkertaisesti liian suuri tai pieni käytettyyn esitystapaan. Kokonaislukujen yhteenlaskun tuloksena voi syntyä liian suuri luku, josta asetuksista riippuen joko aiheutuu ylivuotokeskeytys tai sitten Java-ympäristöjen tapaan ylivuoto vain 'unohdetaan' ja liian suuri tulos hyväksytään modulo-aritmetiikan sääntöjen mukaisesti ottamalla mukaan vain ne vähiten merkitsevät bitit, jotka mahtuvat esitystapaan. Esimerkiksi byte-arvoilla laskettuna Javassa pätee, että 65+65=2.

Konekäskyn aiheuttamat virhetilanteet

Virheellinen käskyosoite	Invalid code address
Virheellinen dataosoite	Invalid data address
Tuntematon käsky	Invalid opcode
Nollalla jako	Divide by Zero
Alustamaton data	Invalid operand
Kokonaisluvun ylivuoto	Integer overflow
Liukuluvun ylivuoto/alivuoto	Floating point overflow Floating point underflow
Virtuaalimuistiosoitteen kuvaama data ei ole keskusmuistissa	Page fault Segment fault

LOAD R1, Apu

Muistanette, että virtuaalimuistia käytettäessä kaiken ohjelman muistiavaruuden ei tarvitse sijaita keskusmuistissa. Osa voi olla levyllä. Jos nyt ohjelmassa tapahtuu (koodi- tai data)muistiviite muistiosoitteeseen, jota ei ole kopioitu levyltä keskusmuistiin, konekäskyn suorituksen aikana tapahtuu ns. sivunpuutoskeskeytys. Keskeytyskäsittelijän jälkeen suoritusvuoro siirtyy muistinhallinnan rutiineille, jotka kopioivat puuttuvan virtuaalimuistin sivun keskusmuistiin, jonka jälkeen tämän käskyn suoritusta yritetään uudelleen, toivottavasti tällä kertaa ilman keskeytystä.

Konekäskyn aiheuttamat muut poikkeustilanteet

SVC käsky
- käyttöjärjestelmän aliohjelma

	ttk-91:	Pentium II:
	`SVC SP, =HALT`	`INT 4`
	`SVC SP, =READ`	`INT 7`

I/O konekäsky
- voi olla puhdas laitteistototeutus tai keskeytyskäsittelijä

Trace keskeytys
- laitteistotuki ohjelmien kehitysympäristöjen toteutukseen

Käyttäjän määrittelemä keskeytys
- laitteistotuki esim. Javan `throw/catch` tai `try/catch` -operaatioiden toteutukseen

Mikä tahansa suorittimen arkkitehtuurissa määritelty konekäsky, jota ei tässä suorittimen versiossa ole toteutettu laitteistotasolla
- keskeytyskäsittelijä toteuttaa harvoin käytetyn konekäskyn

Konekäsky voi siis aiheuttaa myös tahallisia keskeytyksiä, jotka tapahtuvat nyt deterministisesti joka kerta tämän konekäskyn suorituksen yhteydessä. Ne ovat itse asiassa erilaisia aliohjelmien kutsuja, jotka on vain kätevä toteuttaa keskeytysmekanismin kautta. Yksinkertaisin sellainen on SVC-käsky, jonka avulla kutsutaan käyttöjärjestelmän palveluja. Tällaisia palveluja voisi olla esimerkiksi tiedostojen käsittely tai tietoliikennepalvelujen käyttö. Kutsuttu palvelu tarkistaa aina aluksi, onko kutsujalla oikeus käyttää tätä palvelua.

Konekäskyn aiheuttamat muut poikkeustilanteet

SVC käsky

- käyttöjärjestelmän aliohjelma

I/O konekäsky

```
IN  R1, =KBD
```

- voi olla puhdas laitteistototeutus tai keskeytyskäsittelijä

Trace keskeytys

- laitteistotuki ohjelmien kehitysympäristöjen toteutukseen

Käyttäjän määrittelemä keskeytys

- laitteistotuki esim. Javan `throw/catch` tai `try/catch` -operaatioiden toteutukseen

Mikä tahansa suorittimen arkkitehtuurissa määritelty konekäsky, jota ei tässä suorittimen versiossa ole toteutettu laitteistotasolla

- keskeytyskäsittelijä toteuttaa harvoin käytetyn konekäskyn

I/O konekäskyt voidaan toteuttaa hyvin yksinkertaisille laitteille suoraan laitteistototeutuksena, mutta monimutkaisemmille laitteille tarvitaan ohjelmistoa, johon päästään käsiksi keskeytysmekanismin kautta. I/O-konekäskystä ei päältä näe, kummalla tavalla se on toteutettu.

Konekäskyn aiheuttamat muut poikkeustilanteet

SVC käsky

- käyttöjärjestelmän aliohjelma

I/O konekäsky

- voi olla puhdas laitteistototeutus tai keskeytyskäsittelijä

Trace keskeytys

- laitteistotuki ohjelmien kehitysympäristöjen toteutukseen

stop with CALL instr

stop with every instr

stop when X written

stop when R4 written

Käyttäjän määrittelemä keskeytys

- laitteistotuki esim. Javan `throw/catch` tai `try/catch` -operaatioiden toteutukseen

Mikä tahansa suorittimen arkkitehtuurissa määritelty konekäsky, jota ei tässä suorittimen versiossa ole toteutettu laitteistotasolla

- keskeytyskäsittelijä toteuttaa harvoin käytetyn konekäskyn

Sovellusten kehitys tapahtuu usein erilaisten kehitysympäristöjen avustuksella, joissa ohjelmien suoritusta voidaan säännellä yksityiskohtaisen säännöstön avulla. Ttk-91'n simulaattorissakin on tällainen kehitysympäristö valmiina integroituna, koska laitteiston tilaa voidaan tarkastella joka konekäskyn suorituksen jälkeen. Tällaisen kehitysympäristön toteuttaminen vaatii laitteiston suunnittelijalta tukea sitä varten suunniteltujen keskeytysten muodossa.

Konekäskyn aiheuttamat muut poikkeustilanteet

SVC käsky

- käyttöjärjestelmän aliohjelma

I/O konekäsky

- voi olla puhdas laitteistototeutus tai keskeytyskäsittelijä

Trace keskeytys

- laitteistotuki ohjelmien kehitysympäristöjen toteutukseen

Käyttäjän määrittelemä keskeytys

- laitteistotuki esim. Javan `throw/catch` tai `try/catch` -operaatioiden toteutukseen

Mikä tahansa suorittimen arkkitehtuurissa määritelty konekäsky, jota ei tässä suorittimen versiossa ole toteutettu laitteistotasolla

- keskeytyskäsittelijä toteuttaa harvoin käytetyn konekäskyn

Joissakin ohjelmointikielissä on määritelty ohjelmointikäsitteitä, joiden avulla ohjelman kontrolli voidaan siirtää keskeytyksiä muistuttavan mekanismin avulla yllättäviinkin paikkoihin koodia. Tällaisten rakenteiden toteuttamiseen tarvitaan joko dedikoituja keskeytyksiä tai käyttöjärjestelmäpalveluja (jotka sitten toteuttavat samat asiat yleiskäyttöisten keskeytysten avulla).

Konekäskyn aiheuttamat muut poikkeustilanteet

SVC käsky

- käyttöjärjestelmän aliohjelma

I/O konekäsky

- voi olla puhdas laitteistototeutus tai keskeytyskäsittelijä

Trace keskeytys

- laitteistotuki ohjelmien kehitysympäristöjen toteutukseen

Käyttäjän määrittelemä keskeytys

- laitteistotuki esim. Javan `throw/catch` tai `try/catch` -operaatioiden toteutukseen

Mikä tahansa suorittimen arkkitehtuurissa määritelty konekäsky, jota ei tässä suorittimen versiossa ole toteutettu laitteistotasolla

- keskeytyskäsittelijä toteuttaa harvoin käytetyn konekäskyn

Suorittimen käskykanta-arkkitehtuuriin voi sisältyä enemmän käskyjä, kuin piiriin kätevästi mahtuu. Piirin toteuttaja voi tällöin säästää tilaa piirillä toteuttamalla harvoin käytettyjä konekäskyjä aliohjelmina muiden konekäskyjen avulla. Tällaiset konekäskyt näyttävät konekielisessä esitystavassa aivan tavallisilta konekäskyiltä, mutta käytännössä ne toimivat sitten SVC-käskyjen tapaan. Esimerkiksi Intelin 8086-suorittimissa ei ollut toteutettuna liukulukukäskyjä, mutta käskykanta-arkkitehtuuriin sisältyvä fadd-käsky voitiin suorittaa tällä menetelmällä kokonaisluku- ja bittisiirtokäskyjen avulla.

Suorittimen ulkopuolelta tulleet keskeytykset

Kellolaitekeskeytys

- tuki käyttöjärjestelmälle
- tuki sovelluksille
- esim. joka 1, 10 ja/tai 50 ms

Linux: KJ saa suoritusvuoron joka 10 ms

```
wait (5.0);      /* odota 5 sekuntia */
```

Laitekeskeytys, I/O-keskeytys

- tuki käyttöjärjestelmälle epäsuoran I/O'n toteuttamiseksi

Laitteistovirhe

- tuki käyttöjärjestelmälle muiden laitteiden antamien virheilmoitusten havaitsemiseksi

Kolmas keskeytyslaji oli suorittimen ulkopuolelta tulleet keskeytykset. Näillä ei siis yleensä ole mitään tekemistä suoritusvuorossa olevan konekäskyn tai ohjelman kanssa. Useimmissa laitteistoissa on suorittimen ulkopuolinen laite, joka herätyskellon tapaan aiheuttaa kellolaitekesketyksen annetuin aikavälein. Tällä tavalla voidaan taata, että käyttöjärjestelmä saa aina aika ajoin suoritusvuoron järjestelmään ja pystyy tekemään hallintotyöt ajoissa.

Suorittimen ulkopuolelta tulleet keskeytykset

Kellolaitekeskeytys

- tuki käyttöjärjestelmälle
- tuki sovelluksille
- esim. joka 1, 10 ja/tai 50 ms

Laitekeskeytys, I/O-keskeytys

- tuki käyttöjärjestelmälle epäsuoran I/O'n toteuttamiseksi

Laitteistovirhe

- tuki käyttöjärjestelmälle muiden laitteiden antamien virheilmoitusten havaitsemiseksi

Toinen tyypillinen ulkoinen keskeytys on laitekeskeytys, jonka avulla esimerkiksi kovalevyä kontrolloiva laiteohjain ilmoittaa käyttöjärjestelmälle kuuluvalle laiteajurille, että sen antama levylukuoperaatio on saatu loppuun. Kyseisen laiteohjainkortin täytyy tietenkin nyt olla sellainen, että se osaa asettaa kontrolliväylään kuuluvalle I/O-keskeytyspiuhalle arvon yksi. Kaikki I/O-laitteet eivät ole näin viisaita, että osaisivat manipuloida keskeytyssignaaleja.

Suorittimen ulkopuolelta tulleet keskeytykset

Kellolaitekeskeytys

- tuki käyttöjärjestelmälle
- tuki sovelluksille
- esim. joka 1, 10 ja/tai 50 ms

Laitekeskeytys, I/O-keskeytys

- tuki käyttöjärjestelmälle epäsuoran I/O'n toteuttamiseksi

Laitteistovirhe

- tuki käyttöjärjestelmälle muiden laitteiden antamien virheilmoitusten havaitsemiseksi

Kolmas ulkoinen keskeytyssignaali on jonkin sortin virheilmoitus. Esimerkiksi muistipiiri tai jokin muu laite voi tällä tavoin ilmoittaa sellaisesta virhetilanteesta, johon käyttöjärjestelmä osaisi reagoida.

Keskeytyskäsittelijä

Osa käyttöjärjestelmää
- huolella tehtyä koodia
- käyttäjä ei voi muuttaa

Keskeytyskäsittelijään siirtymisen yhteydessä asetetaan suoritin ja MMU etuoikeutettuun tilaan (supervisor state)
- tilarekisterin (SR) bitti P päälle
- saa käyttää kaikkia konekäskyjä
- saa viitata mihin tahansa kohtaan muistia (BASE = 0, LIMIT = 'hyvin iso luku')

Keskeytyskäsittelijästä paluun yhteydessä (IRET käsky) suorittimen ja MMU'n tila asetetaan ennalleen
- saa käyttää vain 'tavallisia' konekäskyjä
- saa viitata vain omaan muistialueeseen

Keskeytyskäsittelijät ovat osa käyttöjärjestelmää. Kaikki keskeytystyypit on ennalta tunnettuja ja kaikille on oma keskeytyskäsittelijänsä käyttöjärjestelmässä. Käyttäjä ei siis voi muuttaa keskeytyskäsittelijää. Toisaalta, järjestelmä voi sallia, että keskeytyskäsittelijästä kutsutaan käyttäjän määrittelemää aliohjelmaa. Esimerkiksi Javan throw-catch ohjelmointikäsite voidaan toteuttaa tällä menetelmällä, kun ohjelmoija voi itse määritellä poikkeustyypin throw-ilmauksella ja antaa kyseiseen tapahtumaan liittyvä poikkeuskäsittelijä catch-lauseella.

Keskeytyskäsittelijä

Osa käyttöjärjestelmää

- huolella tehtyä koodia
- käyttäjä ei voi muuttaa

Keskeytyskäsittelijään siirtymisen yhteydessä asetetaan suoritin ja MMU etuoikeutettuun tilaan (supervisor state)

- tilarekisterin (SR) bitti P päälle
- saa käyttää kaikkia konekäskyjä
- saa viitata mihin tahansa kohtaan muistia (BASE = 0, LIMIT = 'hyvin iso luku')

IRET ClearCache LoadBase Enable LoadLimit Disable

Keskeytyskäsittelijästä paluun yhteydessä (IRET käsky) suorittimen ja MMU'n tila asetetaan ennalleen

- saa käyttää vain 'tavallisia' konekäskyjä
- saa viitata vain omaan muistialueeseen

Keskeytyskäsittelijään siirryttäessä joko aina tai ainakin joillekin keskeytystyypeille suoritin (ja MMU) asetetaan etuoikeutettuun suoritustilaan. Tässä tilassa voi nyt tehdä mitä vain: käyttää mitä tahansa konekäskyä ja viitata mihin tahansa kohtaa muistia. Tavallisesti ohjelman annetaan käyttää vain ns. turvallisia konekäskyjä ja viitata vain omiin muistialueisiinsa. Etuoikeutettu suoritustila on laitteiston valmistajan antama tuki luotettavien käyttöjärjestelmien tekemiseksi.

Keskeytyskäsittelijä

Osa käyttöjärjestelmää

- huolella tehtyä koodia
- käyttäjä ei voi muuttaa

Keskeytyskäsittelijään siirtymisen yhteydessä asetetaan suoritin ja MMU etuoikeutettuun tilaan (supervisor state)

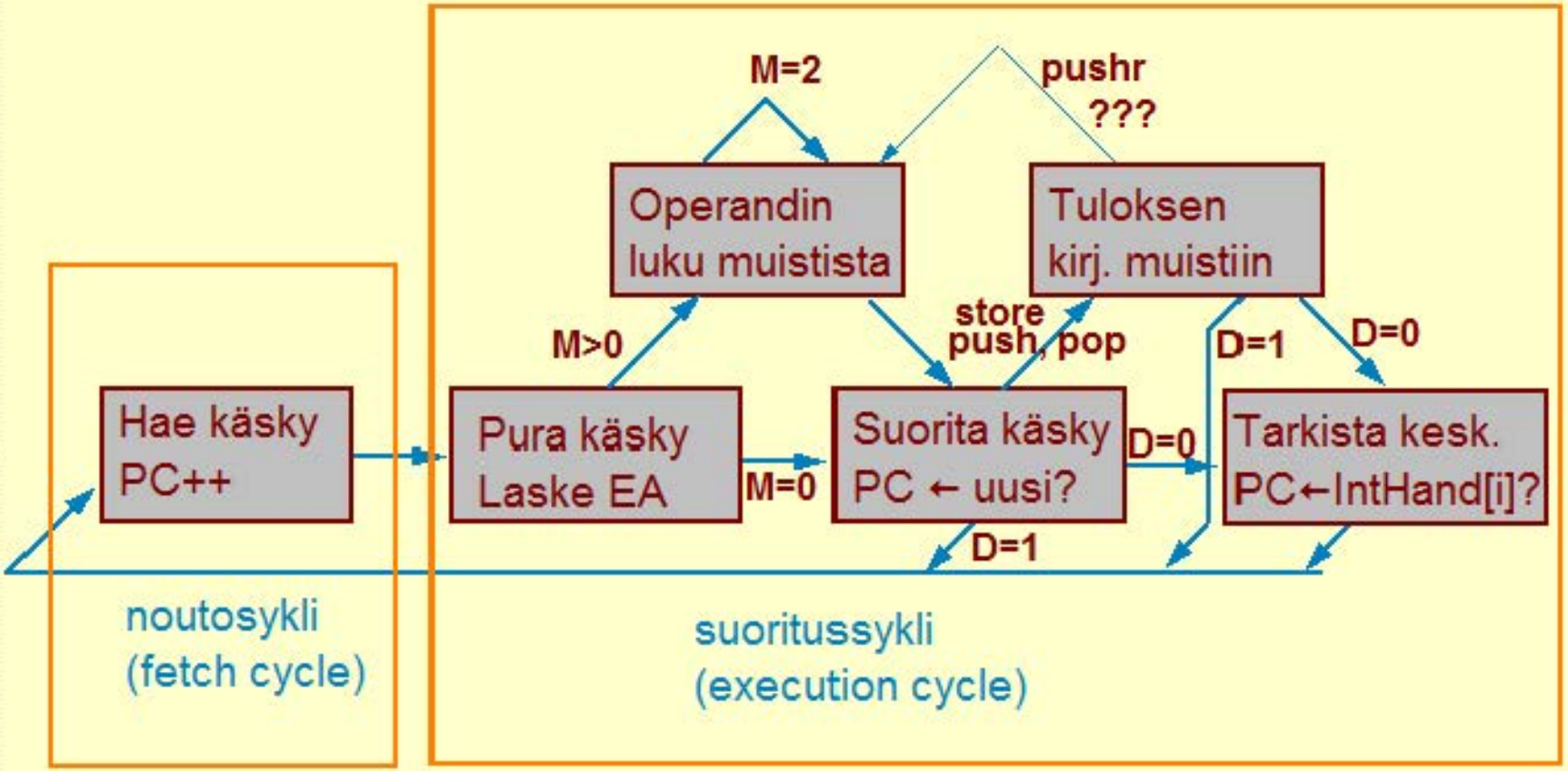
- tilarekisterin (SR) bitti P päälle
- saa käyttää kaikkia konekäskyjä
- saa viitata mihin tahansa kohtaan muistia (BASE = 0, LIMIT = 'hyvin iso luku')

Keskeytyskäsittelijästä paluun yhteydessä (IRET käsky) suorittimen ja MMU'n tila asetetaan ennalleen

- saa käyttää vain 'tavallisia' konekäskyjä
- saa viitata vain omaan muistialueeseen

Keskeytyskäsittelijästä palataan erityisellä IRET-käskyllä. IRET toimii muuten EXIT-käskyn tavoin, mutta se palauttaa myös suorittimen suoritustilan ennalleen.

Keskeytykset käskyjen nouto- ja suoritussyklissä

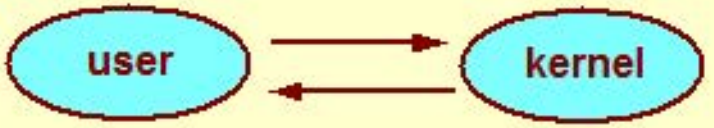


Käskyjen nouto- ja suoritussykliä on nyt siis laajennettu vielä vähän lisää. Ellei keskeytyksiä ole estetty, niin jokaisen käskyn suorituksen lopussa tarkistetaan keskeytysten mahdollinen olemassaolo. Jos käsittelemättömiä keskeytyksiä havaitaan, niin (tärkein niistä) otetaan käsittelyyn muuttamalla PC'n arvoksi keskeytyskäsittelijän osoite.

Suorittimen suoritustilat (suojaustilat)

Käyttäjätila

- user mode, normal mode
- vain 'tavalliset, 'turvalliset' konekäskyt sallittuja
- vain ohjelman oma muistialue viitatavissa

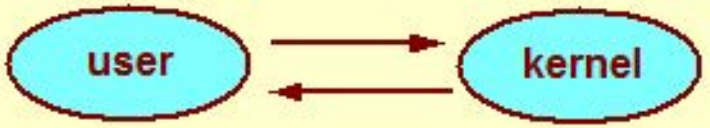


Etuoikeutettu tila

- käyttöjärjestelmän ytimen tila
- supervisor state, kernel mode, priviledged mode
- voi käyttää kaikkia konekäskyjä, erityisesti myös suojatun käyttöjärjestelmän etuoikeutettuja konekäskyjä
- voi viitata kaikkialle muistiin, erityisesti myös käyttöjärjestelmän omiin muistialueisiin
- on mahdollista vaihtaa virtuaalimuistin osoitteenmuunnoksessa käytettyjä taulukkoja tai rajarekistereitä (eli siis virtuaalimuistiavaruutta)
- on mahdollista käyttää myös suoria muistiosoitteita (PA) ilman virtuaalimuistin osoitteenmuunnosta
- toteutusapu suojatun käyttöjärjestelmän tekemiseen
- voi olla myös useampi kuin kaksi tilaa todellisuudessa

Suoritin voi siis olla kahdessa eri suoritustilassa. Normaalin sovelluksen suorituksen aikana se on ns. 'käyttäjätilassa' tai 'user-tilassa', missä ainoastaan kaikki tavalliset konekäskyt ja ohjelman oma muistialue on käytettävissä. Tässä tilassa ohjelma ei kerta kaikkiaan pysty aiheuttamaan vahinkoa - tahallaan tai vahingossa - millekään muulle ohjelmalle tai käyttöjärjestelmälle. Ohjelmalla ei ole mitään keinoa edes viitata muiden ohjelmien tai käyttöjärjestelmän rakenteisiin.

Suorittimen suoritustilat (suojaustilat)



Käyttäjätila

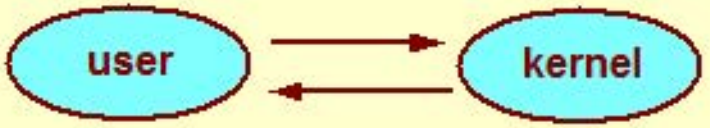
- user mode, normal mode
- vain 'tavalliset, 'turvalliset' konekäskyt sallittuja
- vain ohjelman oma muistialue viitatavissa

Etuoikeutettu tila

- käyttöjärjestelmän ytimen tila
- supervisor state, kernel mode, priviledged mode
- voi käyttää kaikkia konekäskyjä, erityisesti myös suojatun käyttöjärjestelmän etuoikeutettuja konekäskyjä
- voi viitata kaikkialle muistiin, erityisesti myös käyttöjärjestelmän omiin muistialueisiin
- on mahdollista vaihtaa virtuaalimuistin osoitteenmuunnoksessa käytettyjä taulukkoja tai rajarekistereitä (eli siis virtuaalimuistiavaruutta)
- on mahdollista käyttää myös suoria muistiosoitteita (PA) ilman virtuaalimuistin osoitteenmuunnosta
- toteutusapu suojatun käyttöjärjestelmän tekemiseen
- voi olla myös useampi kuin kaksi tilaa todellisuudessa

Etuoikeutetussa suoritustilassa suoritin sen sijaan pystyy tekemään mitä vain. Sillä on käytössään kaikki konekäskyt ja koko muisti. Ainoastaan käyttäjärjestelmän jotkut osat suorittavat koodia tässä tilassa ja käyttöjärjestelmässä on selkeä valvontasysteemi, joka sallii etuoikeutettuun suoritustilaan pääsyn ainoastaan etukäteen määritellyissä tilanteissa. Etuoikeutetussa tilassa suorittavaa koodia ei tavallinen ohjelma tai käyttäjä pääse muuttamaan, vaan sitä varten ohjelmalle tai käyttäjälle tulee olla annettuna ns. systeemimanagerin eli superuser oikeudet.

Suorittimen suoritustilat (suojaustilat)



Käyttäjätila

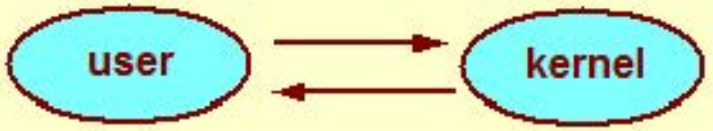
- user mode, normal mode
- vain 'tavalliset, 'turvalliset' konekäskyt sallittuja
- vain ohjelman oma muistialue viitatavissa

Etuoikeutettu tila

- käyttöjärjestelmän ytimen tila
- supervisor state, kernel mode, priviledged mode
- voi käyttää kaikkia konekäskyjä, erityisesti myös suojatun käyttöjärjestelmän etuoikeutettuja konekäskyjä
- voi viitata kaikkialle muistiin, erityisesti myös käyttöjärjestelmän omiin muistialueisiin
- on mahdollista vaihtaa virtuaalimuistin osoitteenmuunnoksessa käytettyjä taulukkoja tai rajarekistereitä (eli siis virtuaalimuistiavaruutta)
- on mahdollista käyttää myös suoria muistiosoitteita (PA) ilman virtuaalimuistin osoitteenmuunnosta
- toteutusapu suojatun käyttöjärjestelmän tekemiseen
- voi olla myös useampi kuin kaksi tilaa todellisuudessa

Etuoikeutetussa suoritustilassa ohjelma voi myös muuttaa ja vaihtaa virtuaalimuistin osoitteenmuunnosrekistereitä tai -taulukoita ja sillä tavoin myös käytännössä vaihtaa käytettävissä olevaa osoiteavaruutta. Kernel-tilassa on myös mahdollista jättää virtuaalimuistin osoitteenmuunnos kokonaan tekemättä ja käyttää koodissa puhtaita fyysisiä muistiosoitteita. Tällä tavoin jotkut käyttöjärjestelmän kriittiset osat voidaan toteuttaa koko ajan keskusmuistissa olevina, joiden käyttö on nopeata eikä haittaa muiden ohjelmien virtuaalimuistin osoitteenmuutosmekanismeja.

Suorittimen suoritustilat (suojaustilat)



Käyttäjätila

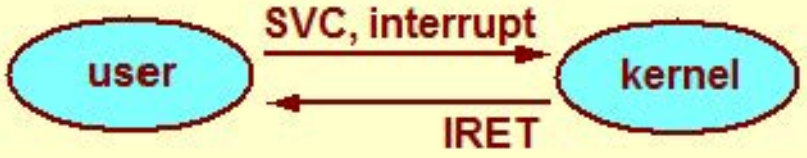
- user mode, normal mode
- vain 'tavalliset, 'turvalliset' konekäskyt sallittuja
- vain ohjelman oma muistialue viitatavissa

Etuoikeutettu tila

- käyttöjärjestelmän ytimen tila
- supervisor state, kernel mode, priviledged mode
- voi käyttää kaikkia konekäskyjä, erityisesti myös suojatun käyttöjärjestelmän etuoikeutettuja konekäskyjä
- voi viitata kaikkialle muistiin, erityisesti myös käyttöjärjestelmän omiin muistialueisiin
- on mahdollista vaihtaa virtuaalimuistin osoitteenmuunnoksessa käytettyjä taulukkoja tai rajarekistereitä (eli siis virtuaalimuistiavaruutta)
- on mahdollista käyttää myös suoria muistiosoitteita (PA) ilman virtuaalimuistin osoitteenmuunnosta
- toteutusapu suojatun käyttöjärjestelmän tekemiseen
- voi olla myös useampi kuin kaksi tilaa todellisuudessa

Etuoikeutettu suoritustila on siis laitteiston toteuttajien antama tuki suojattujen käyttöjärjestelmien tekijöille. Käyttöjärjestelmän tekeminen ilman etuoikeutettua tilaa on vaikeata, mutta tällaisiakin käyttöjärjestelmiä on tehty. Mikä tahansa virheellinen ohjelma voi tällaisessa systeemissa sotkea koko järjestelmän, joten sovellusohjelmoinnin täytyy tietenkin olla sitten aika hyvin tehty. Todellisissa suorittimissa voi myös olla useampia suoritustiloja, mutta käsittelemme nyt jatkossa ainoastaan kahden suoritustilan järjestelmiä.

Suorittimen suoritustilan muuttaminen



Käyttäjätila ⇨ etuoikeutettu tila

- keskeytys tai suora palvelupyyntö (SVC-käsky)
- keskeytyskäsittelijä tarkistaa, onko tällä ohjelmalla oikeutta tilan vaihtoon

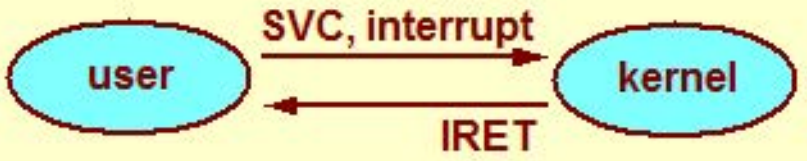
```
PC_old    ← PC
FP_old    ← FP
SR_old    ← SR

FP        ← SP
SR[p]     ← 1
PC        ← IntHandler[i]
```

Etuoikeutettu tila ⇨ käyttäjätila

- etuoikeutettu konekäsky IRET (Interrupt handler RETurn)
- palauttaa kontrollin keskeytyneeseen ohjelmaan
- palauttaa suoritustilan ennen keskeytystä vallinneeseen

Suorittimen suoritustila muuttuu siis vain hyvin valvotuissa olosuhteissa. Tila muuttuu etuoikeutetuksi joko kaikkien tai vain joidenkin keskeytysten yhteydessä ja keskeytyskäsittelijä tarkistaa aina, että tilanmuutos on sallittu. Esimerkiksi, käyttäjätason ohjelma ei voi kutsua käyttöjärjestelmän sisäistä tiedostojenhallintaan liittyvää rutiinia, mutta käyttöjärjestelmän jotkut ohjelmat taas voivat. Keskeytyskäsittelijä voi sitten käyttötarkistusten jälkeen tehdä ihan mitä tahansa muutoksia mihin tahansa rekistereihin tai muistialueisiin.

Suorittimen suoritustilan muuttaminen



Käyttäjätila ⇨ etuoikeutettu tila

- keskeytys tai suora palvelupyyntö (SVC-käsky)
- keskeytyskäsittelijä tarkistaa, onko tällä ohjelmalla oikeutta tilan vaihtoon

Etuoikeutettu tila ⇨ käyttäjätila

- etuoikeutettu konekäsky IRET (Interrupt handler RETurn)
- palauttaa kontrollin keskeytyneeseen ohjelmaan
- palauttaa suoritustilan ennen keskeytystä vallinneeseen

```
PC  ← PC_old
FP  ← FP_old
SR  ← SR_old
```

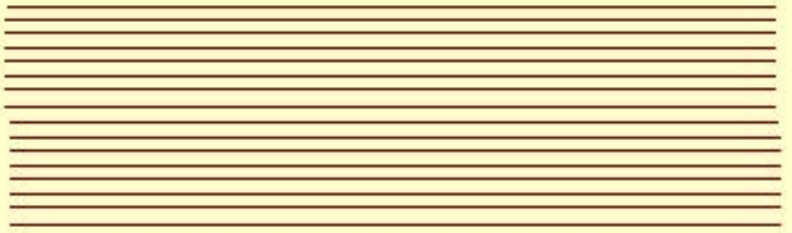

Etuoikeutetusta tilasta siirrytään käyttäjätilaan keskeytyskäsittelijästä palatessa IRET käskyn suorituksen yhteydessä. Tämä tapahtuu palauttamalla tilarekisterin arvo entiselleen. Jos suoritustila oli ennen keskeytystä user-tila, niin tämä palauttaa sen ennalleen. Jos taas tila oli jo ennen keskeytystä etuoikeutettu tila, niin suoritus jatkuu etuoikeutetussa tilassa. Aikaisempi tila voidaan tallettaa aktivointitietueen tapaan keskeytyneen ohjelman pinoon tai sitten joihinkin keskeytyskäsittelyn erityisrekistereihin tai sen omaan muistissa olevaan pinoon.

Väylät

Tiedon siirtoa varten laitteistossa

Johdinkimppu

- mikropiirillä, lastulla
- piirilevyllä
- kaapeli



Yksi kirjoittaa, kaikki näkee, oikea lukee

- kullakin laitteella oma osoite

Eri tasoja

- suorittimen sisäinen väylä
- muistiväylä suorittimen ja muistin välillä
- I/O-väylä muistiväylän ja I/O-laitteiden välillä

Useita eri tapoja yhdistellä ja toteuttaa

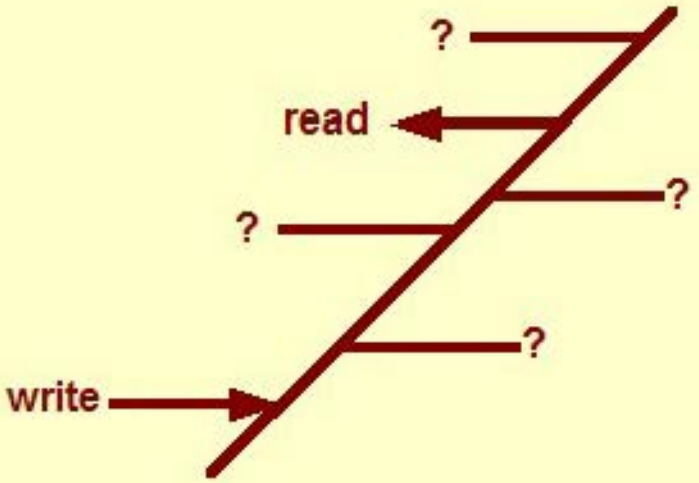
Väylät hoitavat kaiken tiedonsiirron tietokonelaitteistossa. Väylät ovat yksinkertaisesti sähköjohtimia ja jokaisessa johtimessa on koodattuna 1 tai 0 erilaisilla jännitetasoilla, joita väylän päässä olevat laitteet voivat lukea ja kirjoittaa. Johdinkimput voidaan toteuttaa piilastun johtimilla, emolevyn piirikortilla näkyvillä metallisilla johtimilla tai kaapelissa olevilla esimerkiksi kuparisilla sähköjohdoilla. Sekä lastulla että emolevyllä usean rinnakkaisen sähköjohtimen toteuttaminen on usein hankalaa ja niiden toteutus vie paljon arvokasta pinta-alaa. Sama pinta-ala voitaisiin lastulla käyttää vaikkapa rekistereihin.

Väylät

Tiedon siirtoa varten laitteistossa

Johdinkimppu

- mikropiirillä, lastulla
- piirilevyllä
- kaapeli



Yksi kirjoittaa, kaikki näkee, oikea lukee

- kullakin laitteella oma osoite

Eri tasoja

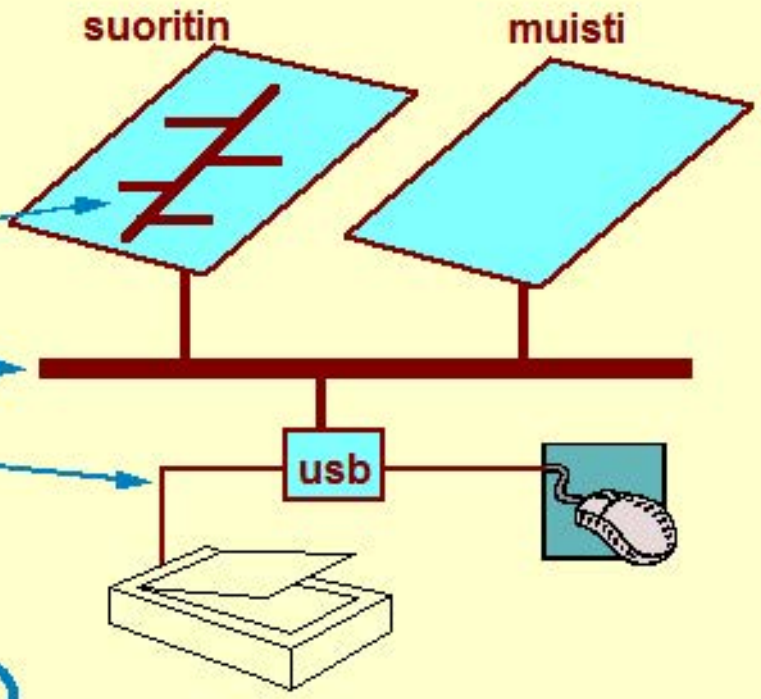
- suorittimen sisäinen väylä
- muistiväylä suorittimen ja muistin välillä
- I/O-väylä muistiväylän ja I/O-laitteiden välillä

Useita eri tapoja yhdistellä ja toteuttaa

Väylä yhdistää siis sähköjohtokimpulla usean eri laitteen toisiinsa. Väylälle voi kirjoittaa vain yksi laite kerrallaan, jotta siellä oleva tieto ei menisi sekaisin. Kaikki väylällä olevat laitteet 'näkevät' tai 'kuulevat' väylällä olevan tiedon, mutta ainoastaan se laite (tai ne laitteet), joille tieto on tarkoitettu, lukee sen. Kunkin väylän käyttöön liittyy tietty protokolla, jota kaikki sen väylän käyttäjät huolellisesti noudattavat. Esimerkkinä voisi olla vaikkapa luentosali, jossa opiskelijat viittaamalla varaavat puhevuoron, jotta vain yksi kerrallaan olisi äänessä. Jos kaikki puhuisivat yhtä aikaa, kenenkään puheesta ei saisi selvää.

Väylät

Tiedon siirtoa varten laitteistossa



Johdinkimppu

- mikropiirillä, lastulla
- piirilevyllä
- kaapeli

Yksi kirjoittaa, kaikki näkee, oikea lukee

- kullakin laitteella oma osoite

Eri tasoja

- suorittimen sisäinen väylä — **hyvin nopea, hyvin luotettava**
- muistiväylä suorittimen ja muistin välillä — **aika nopea, luotettava**
- I/O-väylä muistiväylän ja I/O-laitteiden välillä — **hidas, sopiva laitteelle, virhealtis?**

Useita eri tapoja yhdistellä ja toteuttaa

Väyliä on laitteistossa siis useita eri tasoisia. Suorittimen sisäiseen kommunikointiin tarkoitettu väylä on kaikkein nopein. Se sisältää myös erinäisiä ylimääräisiä tarkistussignaaleja, joiden avulla voidaan varmistua tiedon muuttumattomuudesta tiedonsiirron aikana. I/O- ja verkkolaitteiden väylät ovat yleisesti ottaen huomattavasti hitaampia, eikä niissä (aina) ole mukana ylimääräisiä virheenkorjausbittejä. Laitteiden ja verkon tiedonsiirrossa tapahtuvat virheet korjataan eri tavalla, yleensä tekemällä tiedonsiirto uudelleen. Suorittimen sisäisessä tiedonsiirrossa ei ole aikaa uudelleenlähetyksiin, koska jo siihen varautuminenkin hidastaisi laskentaa merkittävästi.

Väylät

Tiedon siirtoa varten laitteistossa

Johdinkimppu

- mikropiirillä, lastulla
- piirilevyllä
- kaapeli

Lisää tietoa? ➔ Tietokoneen rakenne -kurssi

Yksi kirjoittaa, kaikki näkee, oikea lukee

- kullakin laitteella oma osoite

Eri tasoja

- suorittimen sisäinen väylä
- muistiväylä suorittimen ja muistin välillä
- I/O-väylä muistiväylän ja I/O-laitteiden välillä

Useita eri tapoja yhdistellä ja toteuttaa

ei 'oikeata' tapaa yhdistellä

väylille on standardeja, yhdistelytavoille ei ole

jokaisella emolevyllä oma tapansa yhdistellä eri väyliä

laitteistoon voi jälkeenpäin liittää uusia väyliä vanhoihin väyliin sovittimien avulla

Tietokonelaitteistoissa on hyvin moninaisia tapoja yhdistellä komponentteja väylähierarkiassa. Mitään yleisesti hyväksyttyä standardia ei ole. Yleensä kuitenkin sisäinen väylä ja muistiväylä ovat staattisia, mutta I/O- ja verkkolaitteiden väyliä voi dynaamisesti muutella järjestelmän elinaikana liittämällä uusia I/O-väyliä tai verkkosovittimia muistiväylään tai siihen ennestään liitettyihin muihin väyliin. Tämä antaa hyvin joustavan tavan toteuttaa laitteiston väylähierarkia. Esimerkiksi, kun luennoitsija osti aikoinaan uuden SCSI-liitäntäisen kuvanlukijan, niin voidakseen käyttää sitä hän ensin liitti laitteistoonsa SCSI-väylän sovittimen. Laitteiston väylähierarkia muokattiin siis uuteen laitteeseen sopivaksi.

Väylähierarkia

Tyypillinen Pentium II järjestelmän emolevy

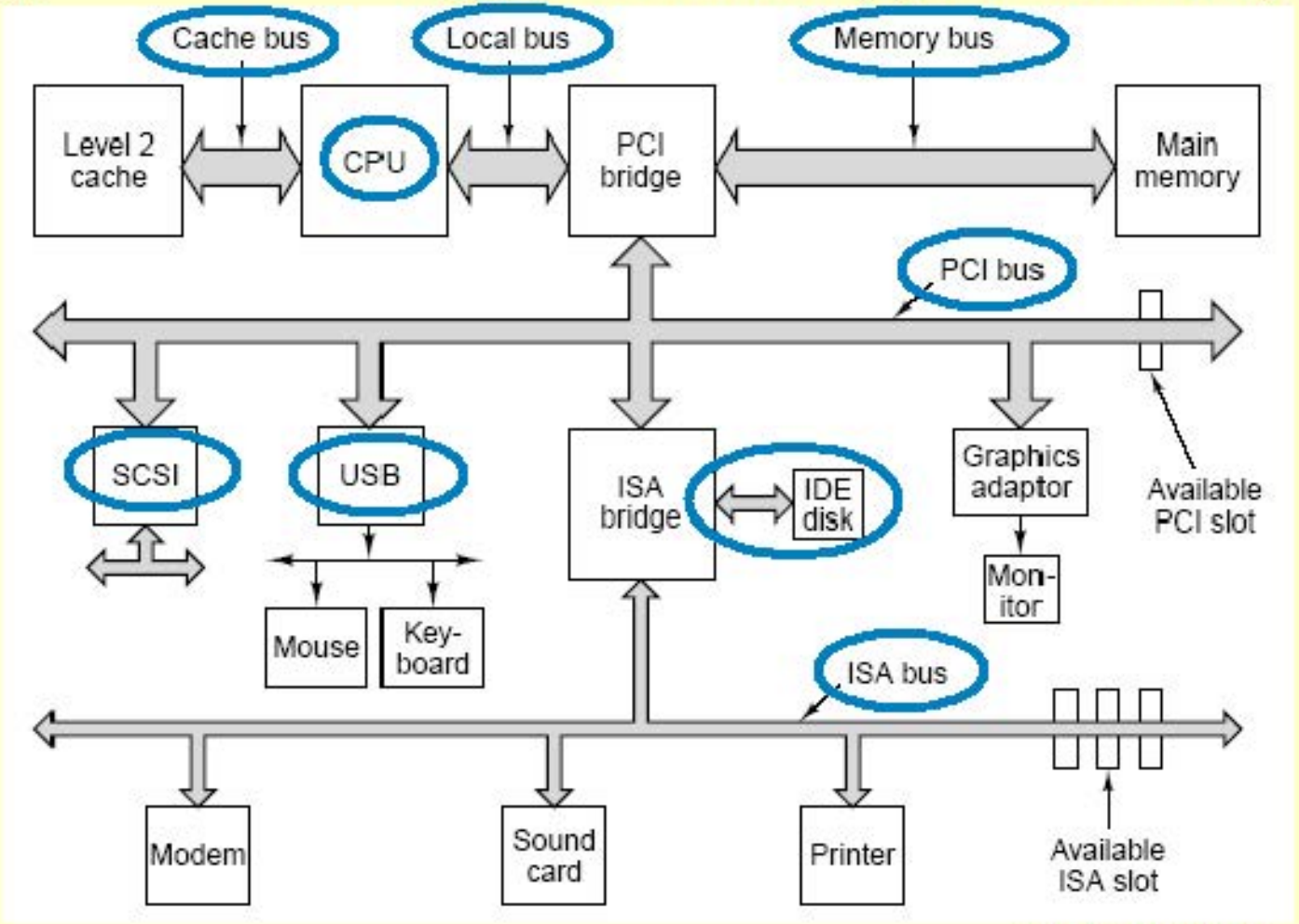


Fig. 3-50 [Tane99]

Väylähierarkiasta esimerkiksi sopii hyvin Tanenbaum'in oppikirjassa mainittu Pentium-II suoritinta hyödyntävä emolevy. Siinä on kahdeksan erilaista väylää suorittimen sisäisen väylän lisäksi. PCI-sillan avulla suorittimeen yhdistetään sekä muistiväylä että nopeimpien laitteiden PCI-väylä. PCI-väylään on sitten monitorin lisäksi liitetty hitaampien väylien sovittimia, mukaanlukien hyvien hitaiden laitteiden aikaisemmin yleisesti käyttämä ISA-väylän sovitin. Tässä sovittimessä on myös tavallisia ISA-laitteita huomattavasti nopeampia laitteita varten IDE-väylä. IDE-väylään liitetään esimerkiksi kovalevy tai DVD-lukija. Kunkin väylän nopeutta rajaa hitain laite, joka siihen voidaan kytkeä.

Väylähierarkia

Tyypillinen Pentium II järjestelmän emolevy

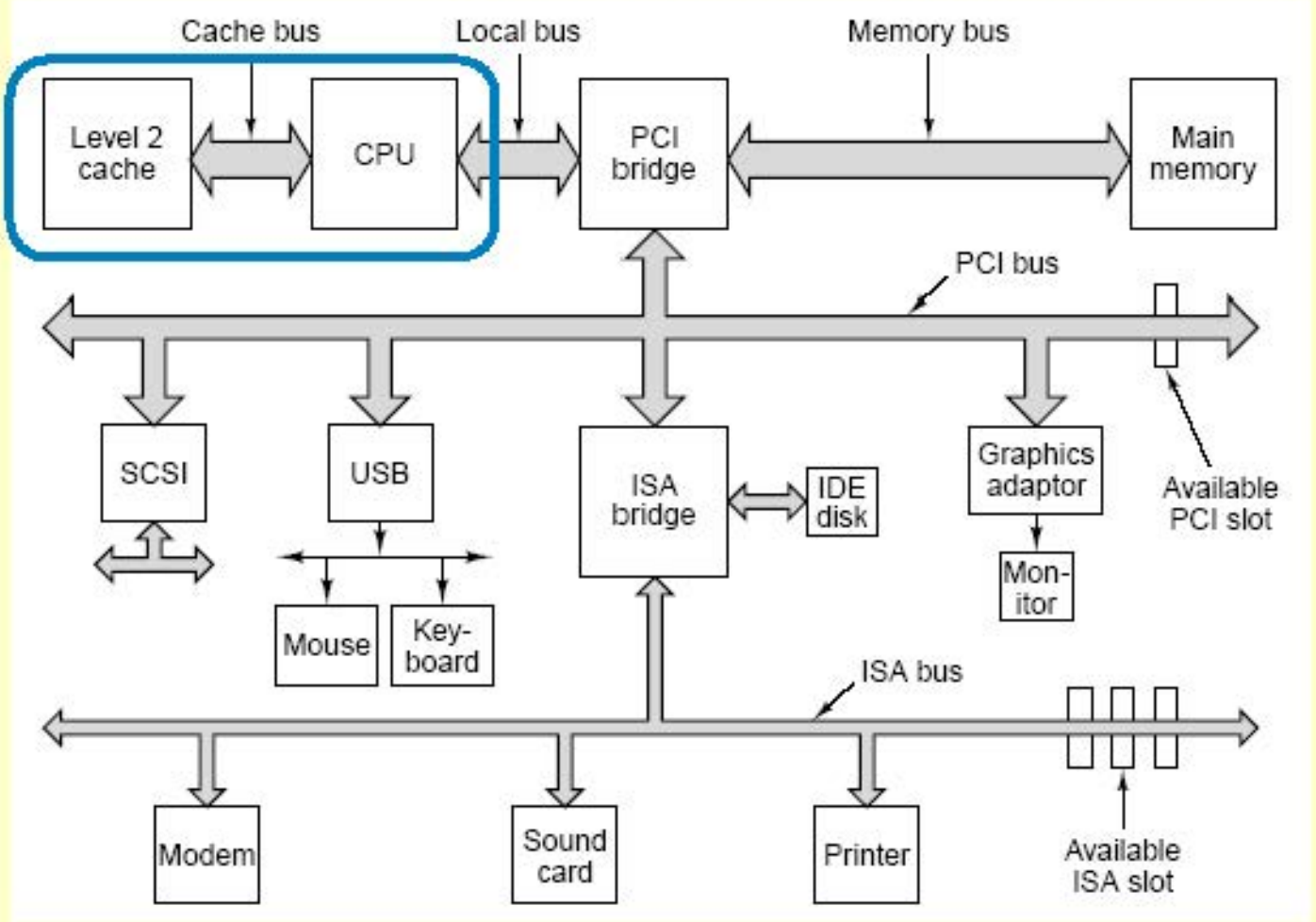


Fig. 3-50 [Tane99]

Tässä esimerkissä välimuisteja on kaksi tasoa. Nopein taso (Level 1) on toteutettu samalla lastulla suorittimen kanssa ja se on liitetty suorittimeen (jonkin) suorittimen sisäisen väylän kanssa. Tason 1 välimuisti on siis osa suoritinta. Tason kaksi (Level 2) välimuisti on toteutettu omilla lastuillaan. Se on paljon suurempi kuin tason 1 välimuisti ja sille on oma hyvin nopea väylä suorittimeen. Useissa nykyisissä suorittimissa on 3 tasoa välimuistia, joista sekä tason 1 että tason 2 välimuistit on integroitu suorittimeen.

Väylähierarkia

Tyypillinen Pentium II järjestelmän emolevy

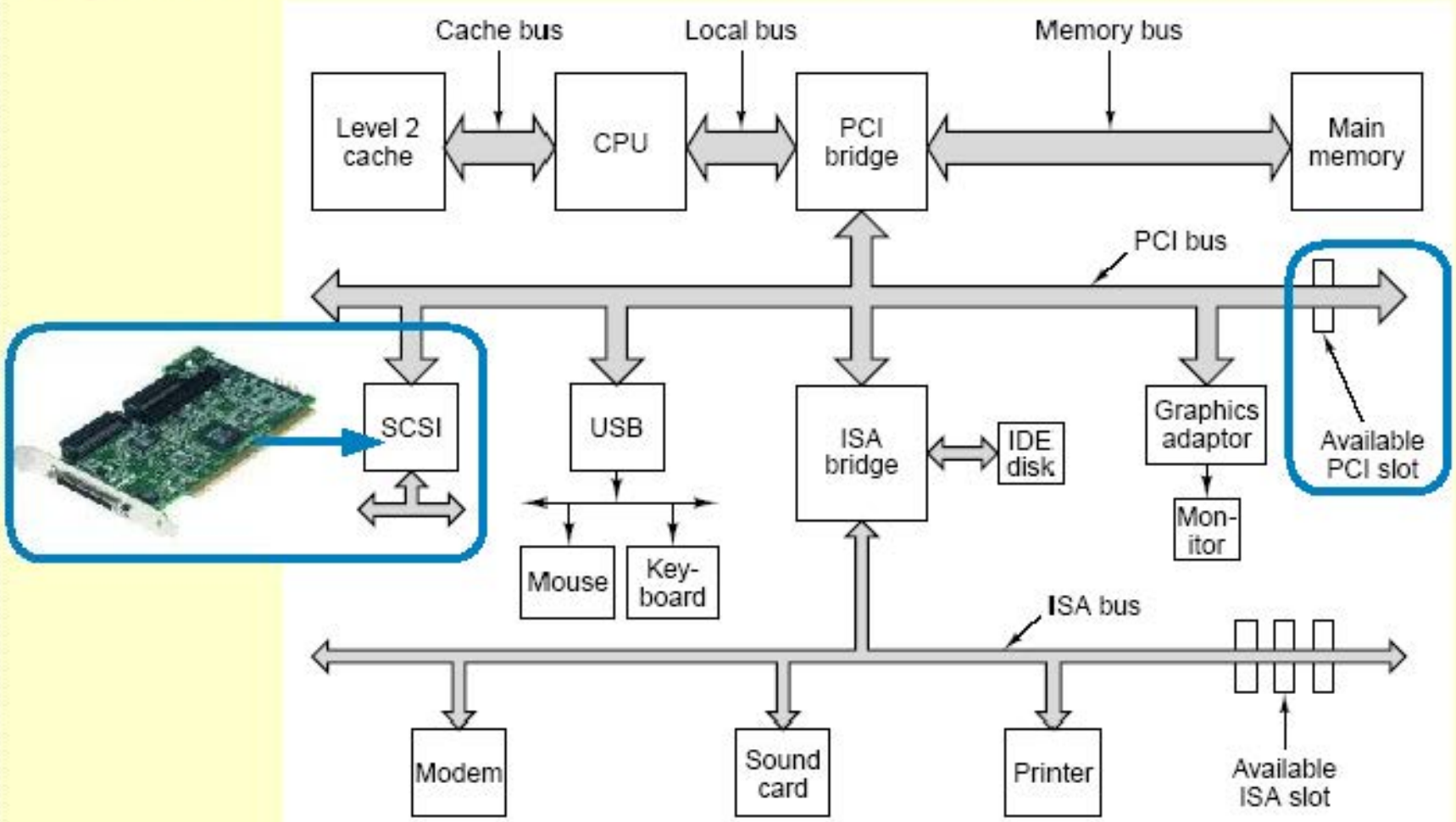


Fig. 3-50 [Tane99]

Emolevyllä olevaan PCI-väylään on tässä laitteistossa liitetty mm. SCSI-väylän sovitin, minkä avulla laitteistoon voidaan liittää 7 erilaista SCSI-laitetta. Sovitin on kiinnitetty laitteiston sisällä olevaan PCI-väylään vapaaseen paikkaan yksinkertaisesti painamalla se paikalleen. Sovittimen lisäksi käyttöjärjestelmään on täytynyt asentaa tämän sovittimen laiteajuri, jotta myös käyttöjärjestelmä ja sitä kautta kaikki sovellukset osaavat käyttää tätä uutta sovitinta. Tietenkin myös jokaista SCSI-väylään liitettävää laitetta varten käyttöjärjestelmä tarvitsee oman ajurinsa. Emolevyllä toteutettu PCI-väylä on kiinteä ja siinä on enää 1 paikka vapaana.

Ttk-91 järjestelmän simulaattori

Tavallinen (esim.) Pascalilla tai Javalla kirjoitettu ohjelma

```
main () {

/* määrittelyt */
...


/* koodi */
...

}
```

Ttk-91 suorittimen osat tietorakenteina

- rekisterit, MMU, CU, muisti

Simuloi käskyjen suoritussykliä yksi käsky kerrallaan

- debugger-ympäristöön kuuluva animointi on se vaikea osa

Toteuttaa myös ttk-91 järjestelmän käyttöjärjestelmän osia

- assemblerkääntäjä, lataaja, debugger, keskeytyskäsittelijät

Graafinen käyttöliittymä

Minkä tahansa suorittimen toimintaa voidaan simuloida halutulla tarkkuudella ohjelmallisesti. Esimerkiksi Intel rakentaa täydelliset simulaattorit uusista lastuista ja testaa lastujen toiminnan tällä tavoin ennen niiden valmistuksen aloittamista. Intelin simulaattorissa on mukana hyvin paljon yksityiskohtia, mukaanlukien myös suorittimien eri komponenttien nopeus. Meidän simulaattorimme on paljon yksinkertaisempi. Se simuloi ainoastaan konekielisten ohjelmien funktionaalista käyttäytymistä siten, että suoritetun laskennan eli ohjelman lopputulos on simulaattorissa sama kuin mitä se olisi, jos sama ohjelma olisi suoritettu todellisessa tietokoneessa. Simulaattorin voi kirjoittaa millä tahansa ohjelmointikielellä.

Ttk-91 järjestelmän simulaattori

Tavallinen (esim.) Pascalilla tai Javalla kirjoitettu ohjelma

Ttk-91 suorittimen osat tietorakenteina
- rekisterit, MMU, CU, muisti

Simuloi käskyjen suoritussykliä yksi käsky kerrallaan
- debugger-ympäristöön kuuluva animointi on se vaikea osa

Toteuttaa myös ttk-91 järjestelmän käyttöjärjestelmän osia
- assemblerkääntäjä, lataaja, debugger, keskeytyskäsittelijät

Graafinen käyttöliittymä

```
int  Reg[8]        /* rekisterit */

int  MBR, MAR;     /* MMU */
int  BASE, LIMIT;

int PC, IR, TR, SR;   /* CU */

int  OpCode   /* IR kentät */
int  Rj, M, Ri, Addr;

int   srG = 31;  /* SR bitit */
int   srE = 30;
int   srL = 29;
int   srO = 28;
...
int   srD = 21;

int Mem[maxLIMIT];/* muisti */
```


Simulaattorissa suorittimen ja muun laitteiston sisäiset tietoa tallettavat rakenteet, kuten esimerkiksi rekisterit ja muisti, toteutetaan normaaleina ohjelman tietorakenteina. Näihin tietorakenteisiin voidaan nyt tallettaa ja niistä voidaan lukea tietoa, eli ne tässä mielessä käyttäytyvät funktionaalisesti samalla tavalla kuin vastaavat laitetoteutukset. Esimerkiksi, siinä missä todellisella suorittimella on rekisteri TR tilapäisen tiedon tallettamista varten, niin simulaattorissa globaali muuttuja TR ajaa saman asian. Laitteistototeutuksen rekisterin TR lukua vastaa nyt simulaattorin muuttujan TR arvon lukeminen. Toiminta on hyvin erilaista, mutta lopputulos on hyvin samanlainen.

Ttk-91 järjestelmän simulaattori

Tavallinen (esim.) Pascalilla tai Javalla kirjoitettu ohjelma

Ttk-91 suorittimen osat tietorakenteina

- rekisterit, MMU, CU, muisti

Simuloi käskyjen suoritussykliä yksi käsky kerrallaan

- debugger-ympäristöön kuuluva animointi on se vaikea osa

Toteuttaa myös ttk-91 järjestelmän käyttöjärjestelmän osia

- assemblerkääntäjä, lataaja, debugger, keskeytyskäsittelijät

Graafinen käyttöliittymä

```
while(true) {
  GetInstruction(PC);
  Increment(PC);
  GetFields(IR);
  for (i=0; i<M; i++) {
     TR =ComputeEA();
     TR = GetOperand(TR);
     }
  ExecuteInstr(OpCode);
  if (fStore) {
     StoreResult (TR);
     }
}
```

```
/* ADD käsky */
Reg[Rj] += TR;
/* aseta O-bitti SR:ssä? */
/* aseta GEL -bitit? */
```


Käskyjen suoritus simulaattorissa tarkoittaa käskyjen suoritussyklin toteuttamista ohjelmallisesti. Vaikeinta tässä on ymmärtää ja uskoa, että mitään muuta ei todellakaan tarvita. Tietokoneen toiminta on näin yksinkertaista! Oikeassa koneessa toteutusta monimutkaistaa erilaiset ajoitukseen ja sisäisten väylien toteutukseen liittyvät yksityiskohdat. Simulaattorin toteutusta on tahallaan monimutkaistettu ottamalla mukaan hieno ohjelmankehitysympäristö, jonka avulla suoritusta on havainnollistettu. Tämä ylimääräinen tilpehööri muodostaa suurimman osan käyttämästänne ttk-91 simulaattorista.

Ttk-91 järjestelmän simulaattori

Tavallinen (esim.) Pascalilla tai Javalla kirjoitettu ohjelma

Ttk-91 suorittimen osat tietorakenteina

- rekisterit, MMU, CU, muisti

Simuloi käskyjen suoritussykliä yksi käsky kerrallaan

- debugger-ympäristöön kuuluva animointi on se vaikea osa

Toteuttaa myös ttk-91 järjestelmän käyttöjärjestelmän osia

- assemblerkääntäjä, lataaja, debugger, keskeytyskäsittelijät

Graafinen käyttöliittymä

Ttk-91 simulaattoriinne (esim. Titokone) sisältyy myös paljon muuta käyttöjärjestelmään kuuluvaa ohjelmistoa, mitä ilman ei mikään todellinenkaan laitteisto olisi kovin käyttäjäystävällinen. Nämä kaikki varusohjelmistot on toteutettu varsinaisen ttk-91 simulaattorin kanssa integroituna pakettina. Itse ttk-91 simulaattorin voisi kuka tahansa Tietokoneen toiminta -kurssin suorittanut toteuttaa muutamassa viikossa alle 1000 rivin ohjelmalla.

Ttk-91 järjestelmän simulaattori

Tavallinen (esim.) Pascalilla tai Javalla kirjoitettu ohjelma

Ttk-91 suorittimen osat tietorakenteina

- rekisterit, MMU, CU, muisti

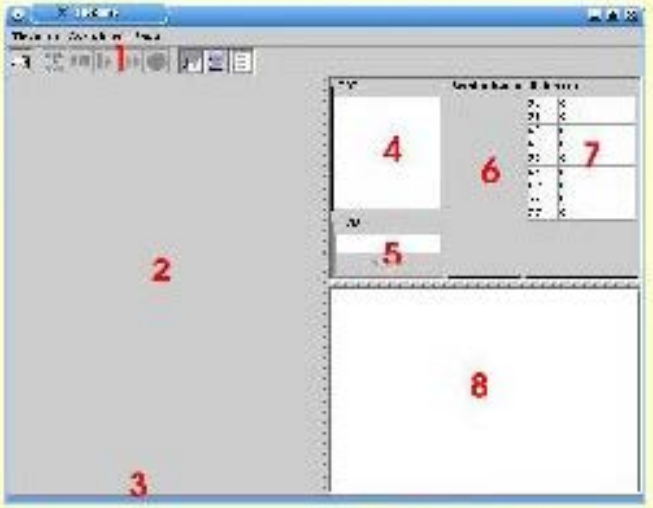
Simuloi käskyjen suoritussykliä yksi käsky kerrallaan

- debugger-ympäristöön kuuluva animointi on se vaikea osa

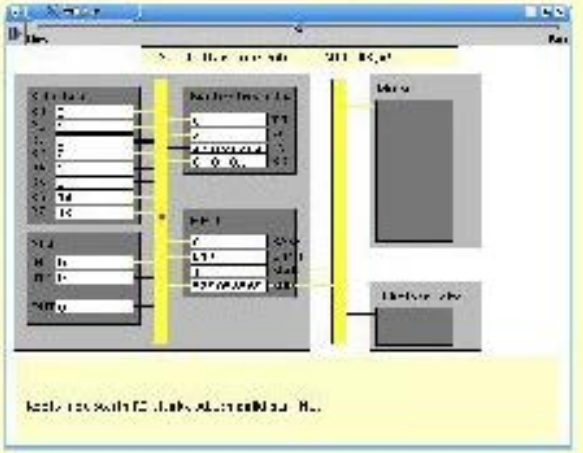
Toteuttaa myös ttk-91 järjestelmän käyttöjärjestelmän osia

- assemblerkääntäjä, lataaja, debugger, keskeytyskäsittelijät

Graafinen käyttöliittymä

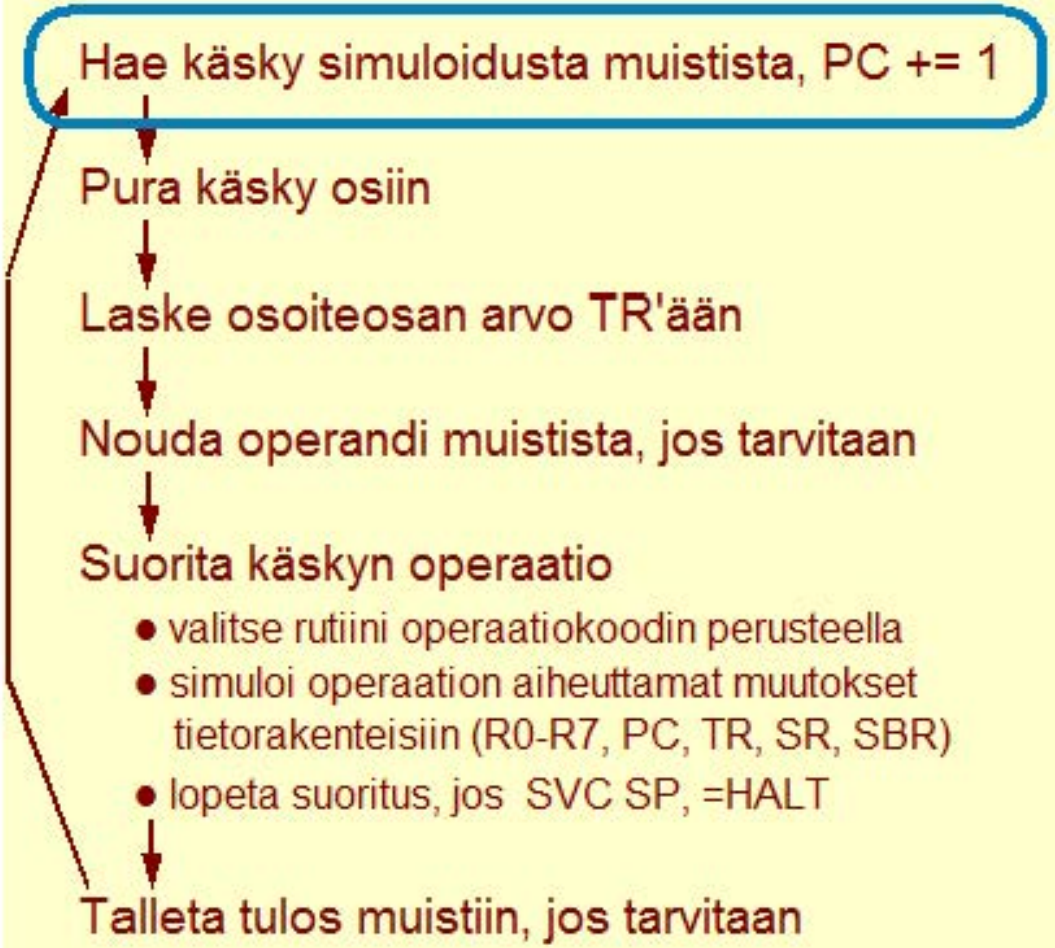


Titokoneen käyttöliittymä



Käyttämänne ttk-91 simulaattoriympäristön koodista suurimman osan muodostaa graafinen käyttöliittymä. Opetuskäyttöön suunnitellulle ohjelmistolle helppokäyttöisyys on ehkä tärkein vaatimus, joten on ollut myös järkevää panostaa siihen jonkin verran. Summa summarum, itse simulointi on helppoa, mutta kääntäjien, lataajien, ohjelmankehitysympäristöjen ja graafisten käyttöliittymien totcutusta varten teidän tulee opiskella vielä muutamalla muullakin tietojenkäsittelytieteen kurssilla. Ttk-91 koneen suorittimen toimintatason simulaattorin toteutukseen riittää tämän kurssin tiedot.

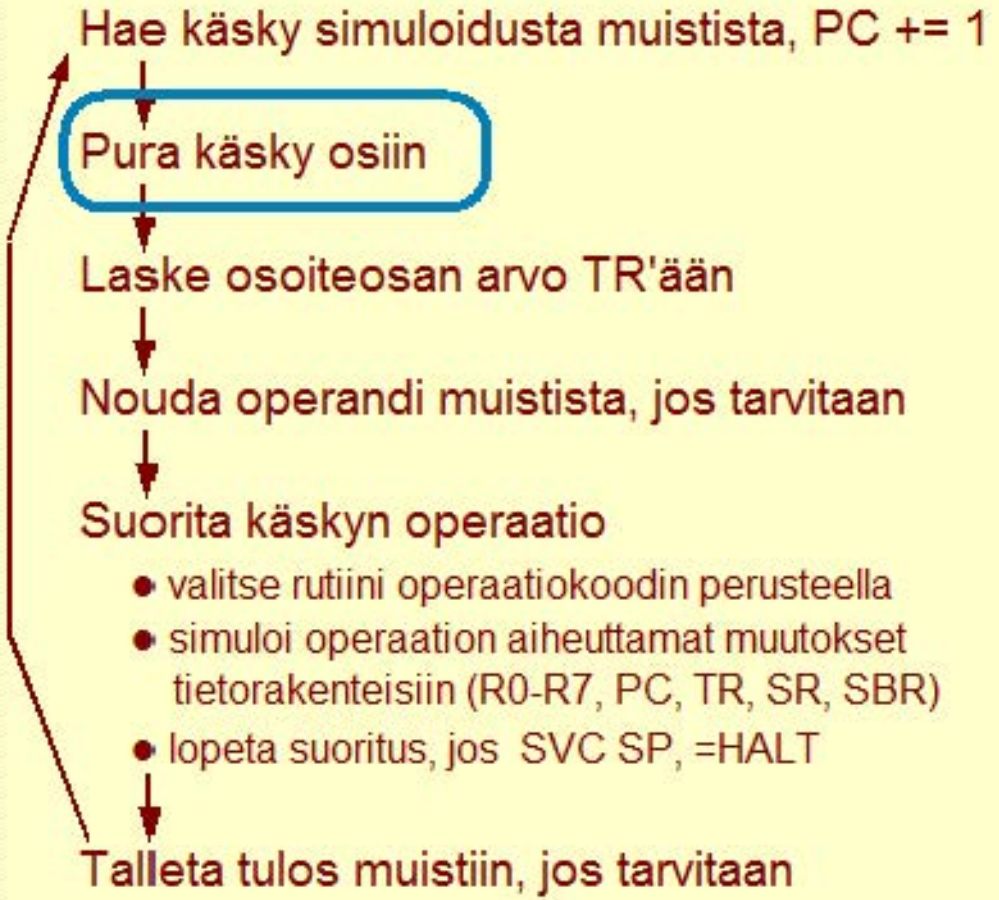
Ttk-91 simulaattorin suoritussykli




```
if (Check(PC) |= fOK)
    BadInstrAddr (PC);
IR = Mem[PC];
PC++;
```


Tarkastellaan nyt simulaattorin käskyjen suoritussyklin toteutusta hieman tarkemmin. Esitetyssä koodissa on jonkin verran tarkistuksia paikallaan, mutta osa on tahallaan jätetty pois, jotta itse asia ei peittyisi tarkistuksiin. Lopullisessa koodissa on tietenkin kaikki tarkistukset aina paikallaan. Käskyn nouto toteutetaan yksinkertaisesti hakemalla muistia simuloivasta taulukosta Mem paikanlaskurin osoittamasta muistipaikasta PC seuraava konekäsky käskyrekisteriin IR. Samassa yhteydessä PC'n arvoa kasvatetaan yhdellä, jotta se osoittaa seuraavaksi suoritettavaan oletusarvoiseen konekäskyyn.

Ttk-91 simulaattorin suoritussykli




```
OpCode = IR >> 24;
Rj = (IR >> 21) % 8;
M = (IR >> 19) % 4;
Ri = (IR >> 16) % 8;
Addr = IR % 32768;
if (Addr / 16384 == 1)
    Addr =  -(Addr % 16384);
```

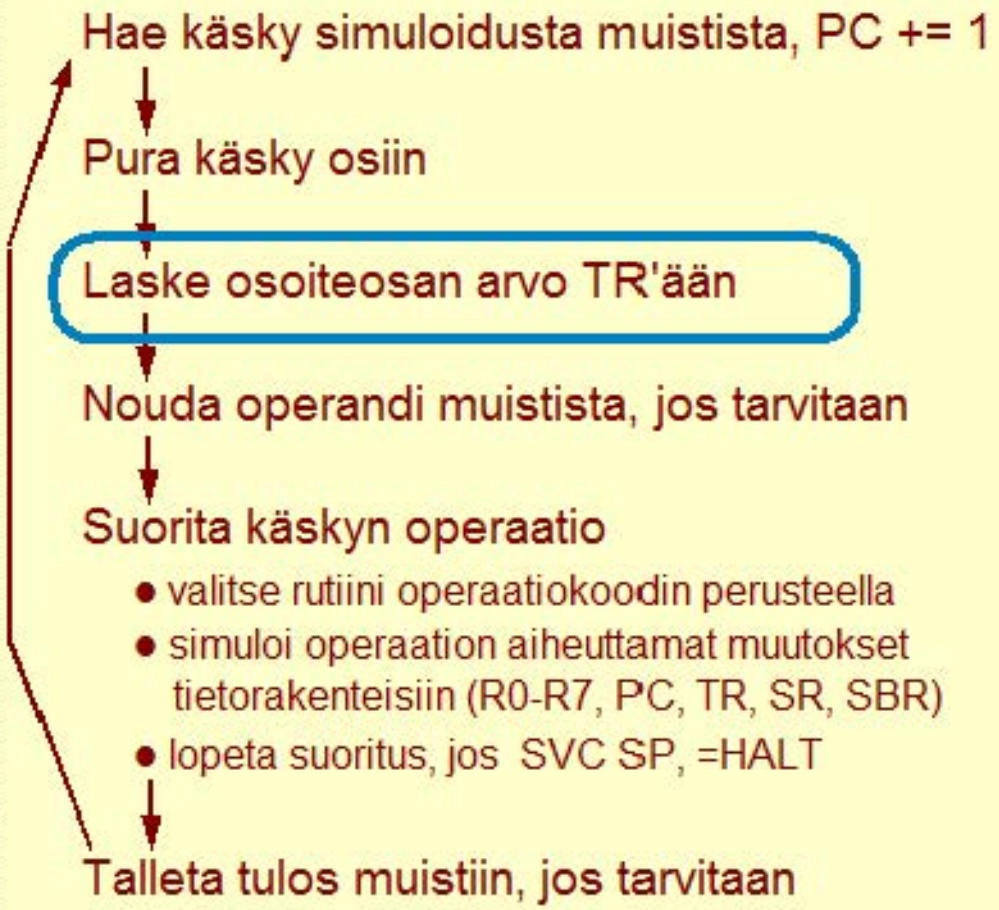
```
M = (IR<<11) >> 19;
```

```
M = (IR % 1048576 / 262144)
```

```
M = (IR / 262144) % 4;
```


Käskyn purku osiin on simulaattorissa itse asiassa monimutkaisempaa kuin todellisessa laitteistossa. Oikeassa käskyrekisterissähän sen kukin kenttä on suoraan luettavissa kenttäkohtaisten johdotusten avulla. Simulaattorissa kentät erotellaan aritmetiikka- ja/tai bittioperaatioilla ja kukin kenttä talletetaan omaan muuttujaansa. Esimerkiksi kenttä M saadaan esiin siirtämällä bittejä 19 bittiä oikealle ja ottamalla sitten 2 oikeanpuolesta bittiä talteen modulo-4 operaatiolla. Samaan tulokseen voisi päätyä myös monella muulla laskukaavalla. Huomaa, että osoiteosan etumerkkibitti pitää tässä ottaa erikseen huomioon tarkistamalla etumerkki ja muuttamalla Addr-kentän arvo tarvittaessa.

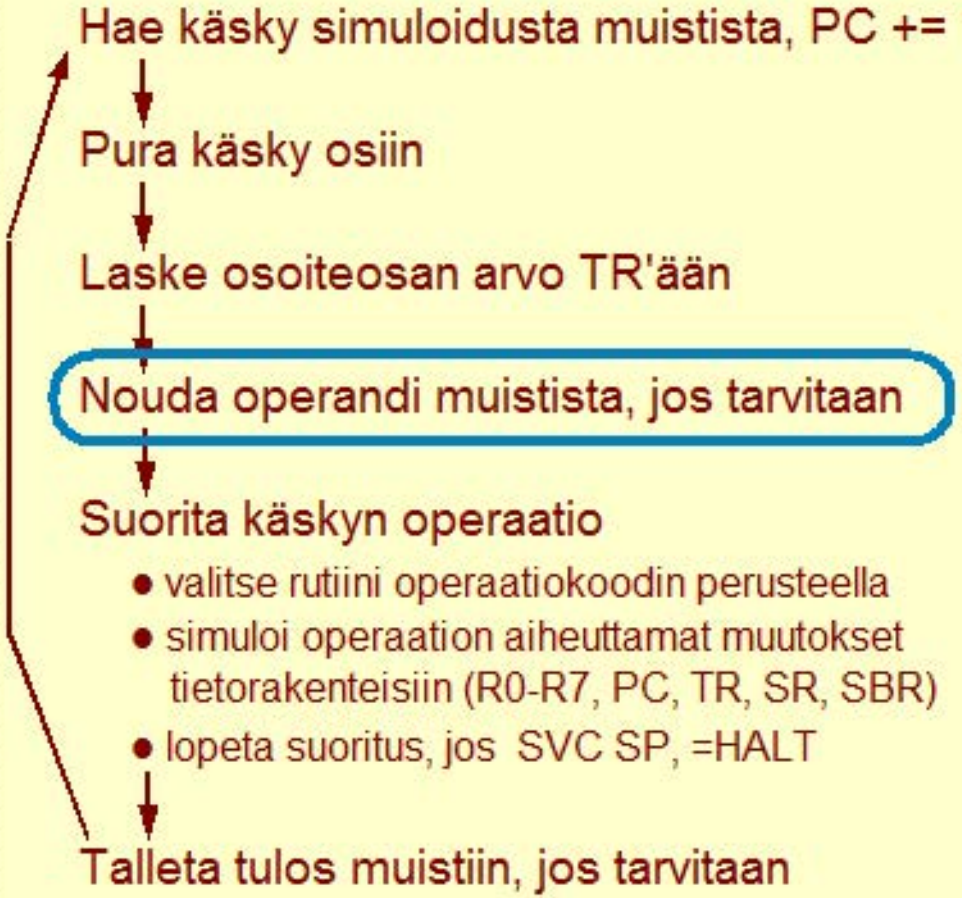
Ttk-91 simulaattorin suoritussykli




```
if (Ri == 0)
    TR = Addr;
else
    TR = Reg[Ri] + Addr;
```


Tehollinen muistisoite EA lasketaan seuraavaksi rekisteriä TR simuloivaan muuttujaan TR yksinkertaisen lausekkeen avulla. Jos indeksirekisterinä käytetään rekisteriä 0, niin se indikoi, että EA'ta laskettaessa indeksirekisterin arvoa ei oteta lainkaan huomioon. Huomaa, että osoiteosan Addr arvo on muutettu ttk-91'n käyttämästä etumerkin sisältämästä esitystavasta tämän ohjelmointikielen käyttämään esitystapaan jo edellisessä käskyn purkuvaiheessa.

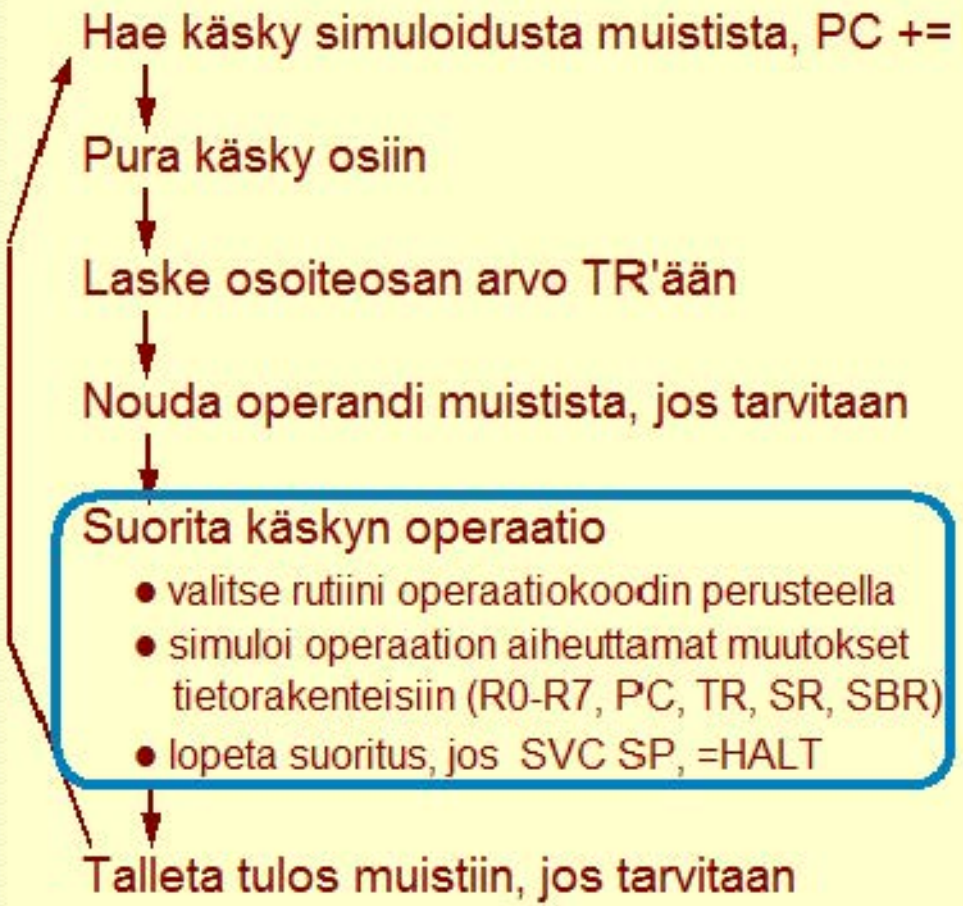
Ttk-91 simulaattorin suoritussykli




```
for (i = 0; i < M; i++) {
    if (Check(TR) |= fOK)
        BadDataAddr (TR);
    TR = Mem[TR];
    }
```


Operandin arvo pitää noutaa muistista 0, 1 tai 2 kertaa moodi-kentän mukaisesti. Osoite löytyy valmiiksi laiterekisteriä TR simuloivasta muuttujasta TR ja muistista haettu arvo sijoitetaan samaan paikkaan. Tämän vaiheen jälkeenhän 2. operandi on valmiina TR'ssä.

Ttk-91 simulaattorin suoritussykli




```
switch (OpCode) {
...
case ADD:
     Rj += TR;
     /* tarkista ylivuoto jotenkin */
     break;
case MUL:
     Rj *= TR;
     /* tarkista ylivuoto jotenkin */
     break;
...
case COMP:
     SR[srG] = SR[srE] = SR[srL] = 0;
     if (Reg[Rj] > TR) SR[srG] = 1;
     if (Reg[Rj] == TR) SR[srE] = 1;
     if (Reg[Rj] < TR) SR[srL] = 1;
    break;
case JEQU:
     if (SR[srE] == 1) PC = TR;
```


Nyt päästään itse asiaan eli konekäskyn toiminnan simulointiin. Haaraudumme operaatiokoodin perusteella sopivaan koodinpätkään (tai aliohjelmaan), jossa juuri tämän operaation toiminta on kuvattu. Toiminnan simulointi on usein hämmästyttävän yksinkertaista, mikä hyvin heijastaa tietokoneen toiminnan yksinkertaisuutta. Esimerkkejä löytyy tästä vielä lisää palkkia vierittämällä.

Ttk-91 simulaattorin suoritussykli


```
switch (OpCode) {
case STORE:
case PUSH:
    if (Check(TR) |= fOK)
        BadDataAddr (TR);
    Mem[TR] = SBR;
    break;
case PUSHR:
    ....
}
```


Käskyn lopuksi pitää vielä ehkä tehdä tuloksen talletus muistiin. Tässäkin vaiheessa erilaiset oikeellisuustarkistukset vaativat enemmän koodia kuin itse muistiin kirjoituksen simulointi. Tietenkin, jos olisimme halunneet simuloida väylän toimintaa tarkemmin, niin kaikkien muistiviitteiden simulointi olisi huomattavasti monimutkaisempaa. Tässä on kuitenkin tyydytty simuloimaan funktionaalista lopputulosta, eli sitä, että simuloituun muistiin oikeaan kohtaan tallettuu haluttu arvo.

Mitä opimme?

Käskyjen nouto- ja suoritussykli

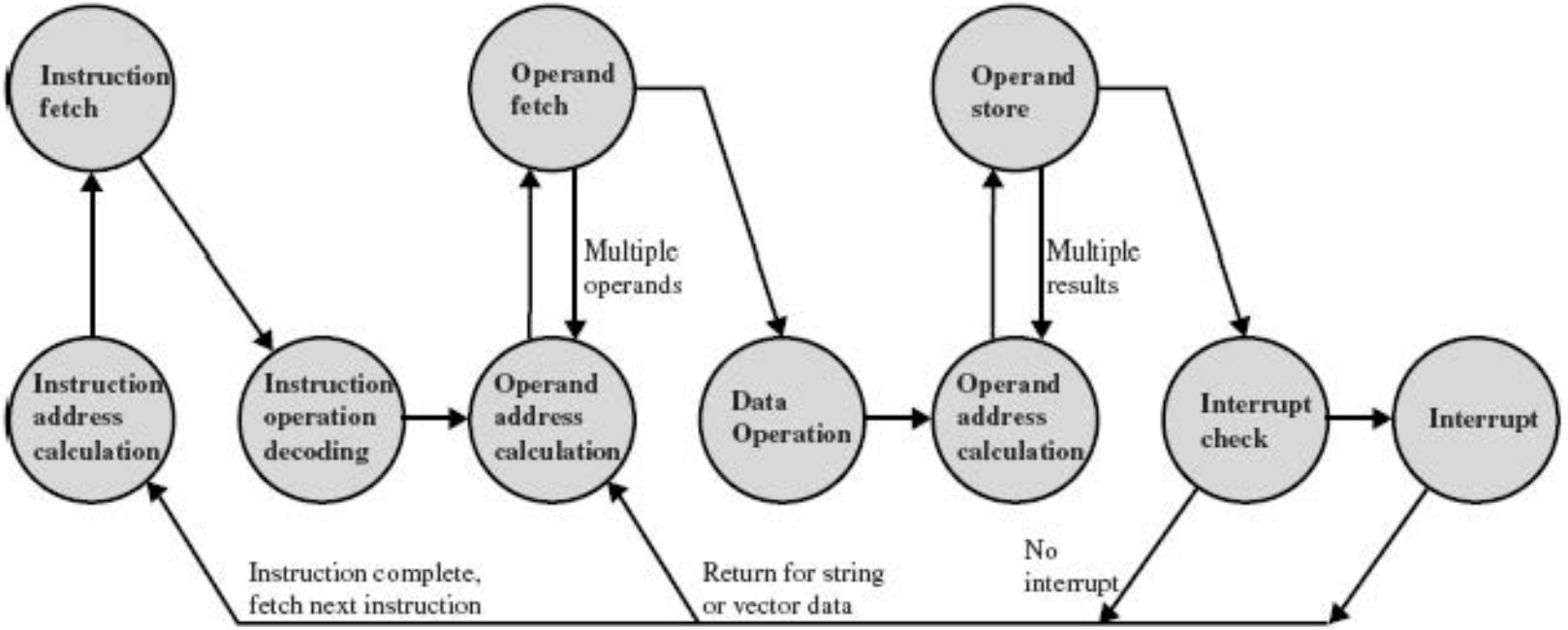


Figure 3.12 Instruction Cycle State Diagram, With Interrupts

[Stal03] Stallings, Computer Organization and Architecture, 6th Ed, 2003.

Olemme nyt käyneet läpi suorittimen ja väylän perustoiminnot. Väylän yksityiskohtaisempi toiminta jätettiin seuraavalle kurssille, mutta suorittimen toimintaan perehdyttiin yksityiskohtaisesti. Tarkastelimme erityisesti käskyjen suoritussyklin toimintaa sekä itse laitteistossa että järjestelmän toimintaa kuvaavassa simulaattorissa. Käskyjen nouto- ja suoritussyklin yksityiskohdat vaihtelevat vähän suorittimesta toiseen, mutta perusratkaisu on silti samanlainen.