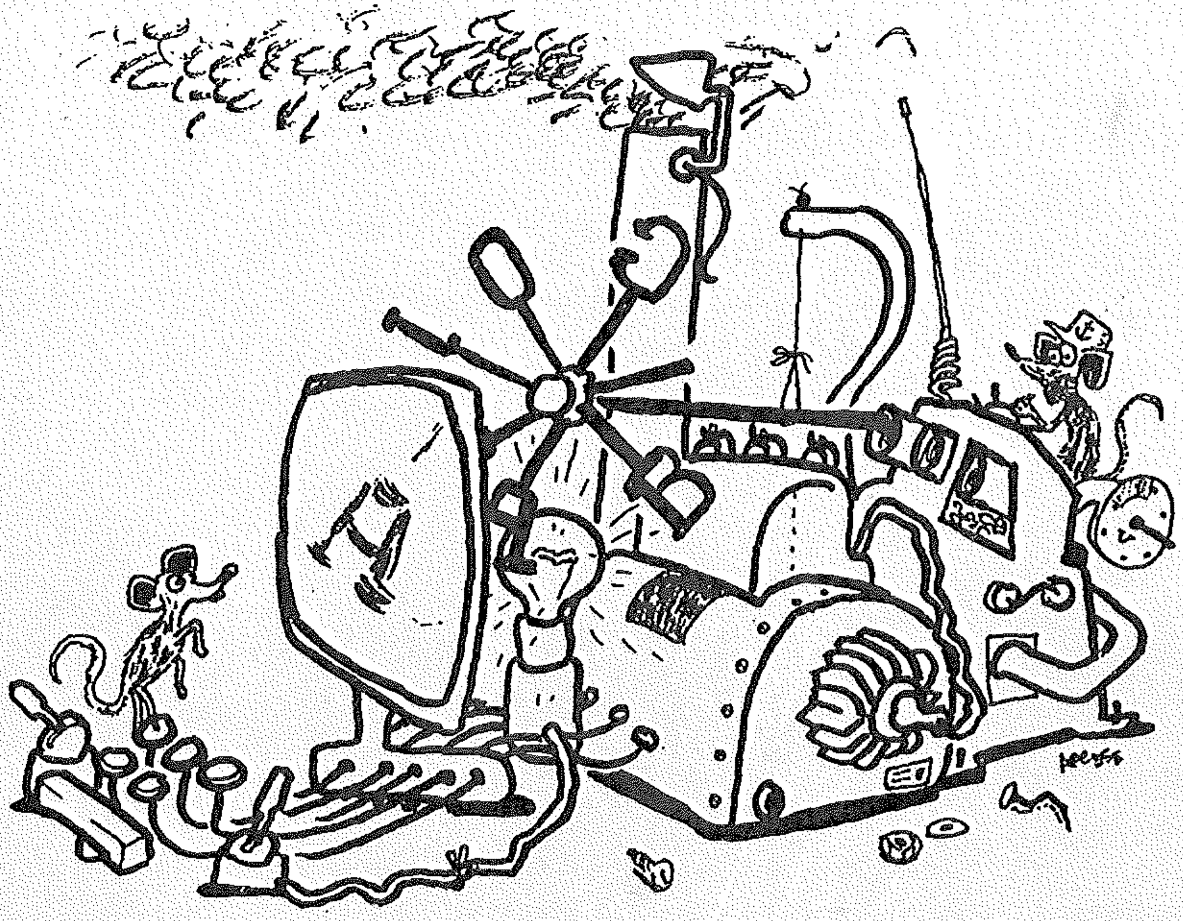


TIETOKONEEN TOIMINTA

Auvo Häkkinen

D390

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

TIETOKONEEN TOIMINTA

Auvo Häkkinen

Alussa

Tietojenkäsittelytieteen approbatur-oppimäärän kurssi Tietokoneen toiminta on "opastettu kiertokäynti tietokonejärjestelmän maailmaan". Tutustumme kurssilla tietokoneen, sen prosessorin ja oheislaitteiden rakenteeseen ja niiden toiminnan pääperiaatteisiin sekä käyttöjärjestelmän toiminnan perusteisiin.

Prosessorin toimintaa tarkastelemme kurssia varten räätälöidyn yksinkertaistetun symbolisen konekielen avulla. Sen avulla selvitämme kuinka tietokone suorittaa ohjelmia. Selvitämme myös, kuinka Pascalin rakenteet voidaan toteuttaa tällä laiteläheisemmällä tasolla.

Käyttöjärjestelmän osalta selvitämme sen perustehtäviä: prosessinhallintaa, muistinhallintaa, tiedostojärjestelmää ja siirräntää. Tärkeimpänä tavoitteenamme on ymmärtää moniajon perusteet. Otamme selvää missä tilanteissa ja kuinka prosessori saadaan suorittamaan käyttöjärjestelmän koodia, sekä kuinka käyttöjärjestelmä vaihtaa suoritettavia prosesseja.

Kurssin keskeisenä tarkoituksena on antaa myöhempiä tietojenkäsittelytieteen opintoja varten yleiskuva tietokonelaitteistosta ja sen toiminnasta. Niinpä kurssi sisältää runsaasti tietotekniikan sanastoa ja käsitteitä. Monia niistä käydään läpi, approbatur kurssien tyyliin, kuitenkin vain esittelynomaisesti pysäyttämättä hakuvartta lähempää tarkastelua varten. Samoja asioita käsitellään laajemmin ja syvällisemmin useilla cum-laude- ja laudatur-oppimäärän kursseilla. Tervemenoa.

Vaikka tutustummekin symbolisen konekielen alkeisiin, ei tarkoituksemme ole tulla assembler-ohjelmoijiksi. Tarkoituksena ei myöskään ole aina esitellä ainoaa oikeaa ratkaisua tai kaikkia mahdollisia ratkaisuvaihtoehtoja kaikkiin ongelmiin. Tavoitteemme on pikemminkin todeta, että "voisihan sen tehdä noinkin, mutta...". Käytä siis monistetta lukiessasi tervettä järkeä. Omaa tai lainattua.

Huomaa myös, että tämänkin monisteen tarkoituksena on vain auttaa kurssin seuraamista. Moniste ei siis välttämättä sovellu itseopiskeluun. Se näkyy osittain siten, että tekstin seassa on esimerkkejä ja käsitteitä, jotka selvästikin kaipaisivat lisätarkennuksia. Nämä lisätarkennukset annetaan luennoilla. Ja kysyäkin saa. Muista tutustua myös muuhun kurssilla kerrottuun materiaaliin.

Kun näet monisteessa numerotietoja, vilkaise kalenteriisi. Tietotekniikka kehittyy todella nopeaa tahtia, ja tässä esitetyt numerotiedot vanhenevat nopeasti. Olen nuo numerot kuitenkin laittanut mukaan, jotta saisimme niistä edes jonkinnäköisen haarukoidun käsityksen.

"Tärkeintä ei ole tieto vaan toiminta, sillä toiminta on ainoa tie tietoon."

Auvo Häkkinen
2.10.1995

Esipuhe vuoden 1998 painokseen

Uuden version laatiminen kävi ajankohtaiseksi, kun perusopetuskieli muuttui syksyllä 1997 Pascalista Javaan. Vuoden 1998 painokseen olen muuttanut ohjelmaesimerkit Javaksi tai lähellä Javaa olevaksi pseudokieleksi. Samalla olen tarkistanut joitakin tietokoneiden suorituskykyyn ja kapasiteettiin liittyviä lukuarvoja. Tietoliikenteen osuutta on myös lisätty ja DOS-käyttöjärjestelmään liittyvää asiaa on vähennetty.

Monisteen uusi painos on jouduttu tekemään melkoisella kiireellä. Tästä ja tekstinkäsittelyjärjestelmän vaihdosta johtuen olen joutunut hieman karsimaan alkuperäisiä kuvia, taulukoita ja muuta rekvisiittaa. Lisäksi mukaan otetut kuvat on liitetty sivuille tekniikalla, joka ei yllä samalle tasolle kuin aikaisemmissa painoksissa. Toivottavasti tämä ei häiritse lukijaa.

Timo Karvi
30.1.1998

Sisältö

Luku 1

Johdanto

1.1 Tietokonejärjestelmä

Tietokone (engl. computer) on muistilla varustettu laite, joka käsittelee tietoa automaattisesti sen muistiin tallennetussa ohjelmassa olevia toimintaohjeita noudattaen. Tietokoneen toiminta perustuu *laitteiston* (hardware) ja *ohjelmiston* (software) yhteispeliin. Laitteisto ja ohjelmisto muodostavat yhdessä *tietokonejärjestelmän.*

Yhdeksi tietokoneeksi katsotaan kaikki rakenteellisesti toisiinsa kytketyt, yhteisen ohjauksen alaiset tietotekniset laitteet. Tietokonejärjestelmässä tarvittavasta ohjauksesta vastaa *keskusyksikkö* (central processing unit, CPU), joka suorittaa sovellusohjelmien lisäksi aika-ajoin käyttöjärjestelmää (operating system). Keskusyksikköön kuuluvat prosessori, keskusmuisti, ohjaimet (controller) sekä näitä emokortilla yhdistävät väylät (bus). Keskusyksikön alaisia laitteita ovat mm. tiedon syöttö- ja tulostuslaitteet, tiedonsiirtolaitteet sekä tiedon tallennuslaitteet. Niitä kutsutaan yhteisesti *oheislaitteiksi* (peripherals).

Tiedon syöttö- ja tulostuslaitteet tarjoavat ihmiselle mahdollisuuden käyttää tietokonetta. Syöttö- ja tulostuslaitteet muuttavat tietokoneessa käytettävän digitaalisen tiedon esitysmuodon ihmisen ymmärtämään muotoon ja päinvastoin. Tiedon tallennuslaitteita tarvitaan tiedon pysyvää tallentamista varten ja tiedonsiirtolaitteiden avulla tietokone voidaan yhdistää esimerkiksi toisiin tietokoneisiin.

1.2 Tietokoneiden historiaa

Yrityksiä mekanisoida laskentaa on esiintynyt jo varhaisella ajalla. Helmitaulun edeltäjä abakus oli käytössä antiikin Kreikassa ja Rooman valtakunnassa. Blaise Pascal suunnitteli ensimmäisen mekaanisen laskukoneen 1642. Charles Babbage (1792-1871) kehitti myös tavallisia mekaanisia laskukoneita, mutta hänet tunnetaan erityisesti suunnitelmistaan rakentaa tietokonetta muistuttava, monimutkaisiin laskutoimituksiin kykenevä laite, jonka muistiin olisi voitu varastoida yli 1000 50-numeroista lukua kerrallaan. Babbagen ideat olivat kuitenkin vaikeasti toteutet-

tavissa sen aikaisella tekniikalla. Muita mekaanisia tietoa käsitteleviä koneita olivat Jacquardin ja Hollerithin 1800-luvun lopulla suunnittelemat reikäkorttikoneet. Näitä ruvettiin rakentamaan ja markkinoimaan 1900-luvun alussa. Esimerkiksi IBM perustettiin 1914 ja sen tuotevalikoimaan kuuluivat erityisesti reikäkorttikoneet. Suomeen reikäkorttikoneita alkoi tulla 1920- ja 1930-lukujen vaihteessa.

Teoreettisella puolella matematiikan perusteisiin liittyvä tutkimus oli 1900-luvun alkupuolelle tultaessa aiheuttanut laajaa mielenkiintoa algoritmisia menetelmiä kohtaan. Vuonna 1928 saksalainen matemaatikko David Hilbert esitteli matematiikan perusteisiin liittyvän tutkimusohjelman, jonka tavoitteena oli mm. kehittää yleinen algoritmi kaikkien matemaattisten probleemien ratkaisemiseksi. Vaikka Hilbertin ohjelma romuttui suurimmaksi osaksi jo 1930, kun Kurt Gödel julkaisi kuuluisat epätäydellisyystuloksensa, matemaattiset totuudet ratkaisevan algoritmin olemassaolo jäi edelleen avoimeksi, joskin totuus oli korvattava termillä todistuvuus.

Kysymys sai ratkaisunsa 1936, jolloin Alonzo Church jätti artikkelinsa julkaistavaksi ja Alan Turing käsikirjoituksensa tarkastettavaksi. Kumpikin osoitti, ettei algoritmia ollut olemassa. Churchin ratkaisu perustui λ-kalkyyliin. Turing konstruoi tietokoneen teoreettisen mallin, ns. universaalin Turingin koneen, jonka avulla hän osoitti monia muitakin ongelmia algoritmisesti ratkeamattomiksi. Turingin koneella saattoi olla myös merkitystä nykyaikaisen tietokoneen toimintaperiaatteiden suunnittelussa, sillä von Neumann tunsi hyvin Turingin työn.

1930-luvulla alkoi myös teknisellä puolella useita tietokoneen kehitysprojekteja. Kondrad Zuse suunnitteli binääriaritmetiikkaan perustuvia relekoneita 1930- ja 40-lukujen vaihteessa Saksassa. Zuse oli alunperin rakennusinsinööri ja hänen pyrkimyksenään oli aluksi sellaisen koneen rakentaminen, jolla automaattisesti voitaisiin suorittaa ikävät ja pitkät rutiininomaiset rakennusteknilliset laskut. Sodan aikana hänen koneitaan käytettiiin mm. V2-rakettien suunnittelussa, mutta niistä kehitettiin sodan jälkeen transistoroituja malleja, jotka soveltuivat moniin teknillisiin tehtäviin.

1937 H. Aiken alkoi toteuttaa Havardissa sähkömagneettista relekonetta, jollaisen IBM rakensi ja jota Yhdysvaltojen laivasto käytti 1944 (Mark I). Kone poikkesi nykyaikaisista tietokoneista varsinkin siinä, ettei ehdollinen haarautuminen käskyjä suoritettaessa ollut mahdollista.

Atanasoff käytti elektroniikkaa vuodesta 1939 Iowan valtion yliopistossa mekanisoimaan aritmeettisia operaatioita. Englannissa rakennettiin Turingin johdolla Colossi 1943 salakirjoitusten purkamista varten.

Mark I:n kokemusten pohjalta konstruoitiin Yhdysvalloissa ENIAC. Suunnittelun johdossa olivat matemaatikko H. Goldstine ja insinöörit J. P. Eckert ja J. W. Mauchly. Kone valmistui 1946 ja sen muistiin mahtui vain 20 lukua kerrallaan.

Vasta laajan muistin konstruointi ja John von Neumannin periaate, jonka mukaan laskuohjelma varastoidaan tietokoneen muistiin, teki tietokoneet todella käyttökelpoisiksi. Oivallukset toteutettiin 1947-1951 Princetonissa. Tuloksena oli tietokone EDVAC, jota käytettiin ensi sijassa vetypommilaskuissa.

Ensimmäisen polven koneet perustuivat elektroniputkien käyttöön. Tämä vaihe jatkui noin vuoteen 1958, jolloin transistorit tulivat käyttöön ja syntyivät toisen polven koneet. Putkikoneiden aikana rakennettiin monessa maassa omia koneita erityisesti korkeakoulujen suojissa. Suomessa rakennettiin ESKO, Ruotsissa BESK ja Tanskassa DASK. BESK valmistui jo 1954 ja ESKO vuonna 1960. ESKO oli jo valmistuessaan vanhentunut, mutta kahta muuta käytettiin vielä 1960-luvulla.

1.3 Tietokoneiden luokittelua

Aina 1960-luvun alkuun asti kaikki tietokoneet olivat suurikokoisia *yleistietokoneita*. Ne veivät tilaa satoja neliömetrejä, lämmittivät koko rakennuksen, maksoivat pienen omaisuuden ja niiden toiminnan ylläpitämiseksi tarvittiin kymmeniä ohjelmoijia, operaattoreita ja muuta teknistä henkilökuntaa. 60-luvulla rakennettiin ensimmäiset kooltaan, teholtaan ja hinnaltaan pienemmät koneet, *pientietokoneet* (minicomputer).

Tietokoneita käsittelivät vain alan ammattilaiset ja tietojenkäsittely oli keskitetty tietokonekeskuksiin *keskustietokoneen* (mainframe) yhteyteen.

70-luvulla Intel julkaisi yhdellä sirulla tietokoneen ohjaukseen ja käskyjen suoritukseen tarvittavan elektroniikan. Koska termiä pientietokone käytettiin huomattavasti suuremmasta oliosta, alettiin yhdellä sirulla olevaa prosessoria kutsua mikroprosessoriksi. Vastaavasti tietokonetta, jonka prosessorina oli mikroprosessori, alettiin kutsua *mikrotietokoneeksi*.

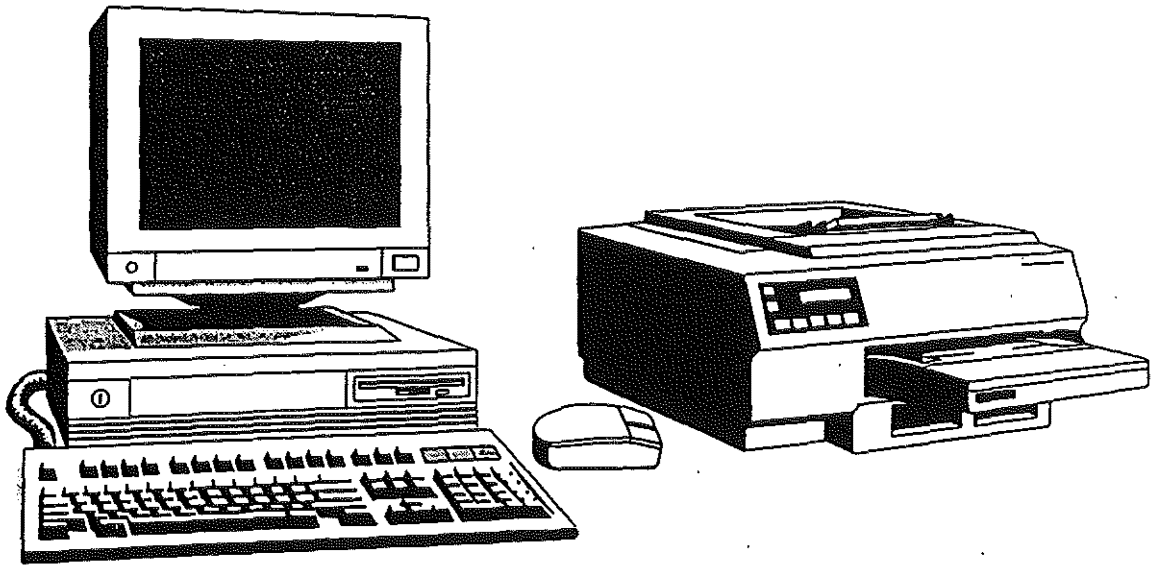
Aluksi yleis-, pien- ja mikrotietokoneet erosivat paitsi nopeudeltaan myös teknisesti huomattavasti toisistaan. Esimerkiksi yleistietokoneessa sananpituus oli vähintään 32 bittiä, pientietokoneissa 16 bittiä ja mikroissa 8 bittiä. Lisäksi yleistietokoneet pystyivät hyödyntämään suurta keskusmuistia, pientietokoneet keskikokoista ja mikrot vain pientä keskusmuistia.

Yleistietokone oli tehokas kone, jonka käytöstä huolehti atk-keskus. Sen käyttäjäkunta oli yleensä suuri. Yleistietokoneissa oli useita kymmeniä megatavuja keskusmuistia, tuhansia megatavuja levymuistia ja sen teho riitti palvelemaan satakunta yhtäaikaista käyttäjää.

Pientietokone oli kooltaan ja teholtaa hiukan pienempi. Tavallisesti sen käytöstä vastasi yksi osasto tai projekti. Pientietokoneessa saattoi olla samanaikaisia käyttäjiä kymmenkunta. Keskusmuistin ja oheislaitteiden osalta se ei eronnut yleiskoneista.

Mikrotietokonetta ei aluksi pidetty edes tietokoneena, vaan lähinnä esimerkiksi televisioon tai autoon sulautettuna laitteena. 80-luvun alussa tapahtui kuitenkin mikrotietokoneiden varsinainen läpimurto *henkilökohtaisena tietokoneena* (personal computer, PC). Konehan oli tarkoitettu vain yhden käyttäjän kerrallaan käytettävksi. Myös ohjelmistotarjonta parantui nopeasti ja mikrotietokoneita alettiin pitää vakavasti otettavina tietokonejärjestelminä.

Integroitujen piirien valmistusteknologian valtavan kehityksen ansiosta näitä tek-

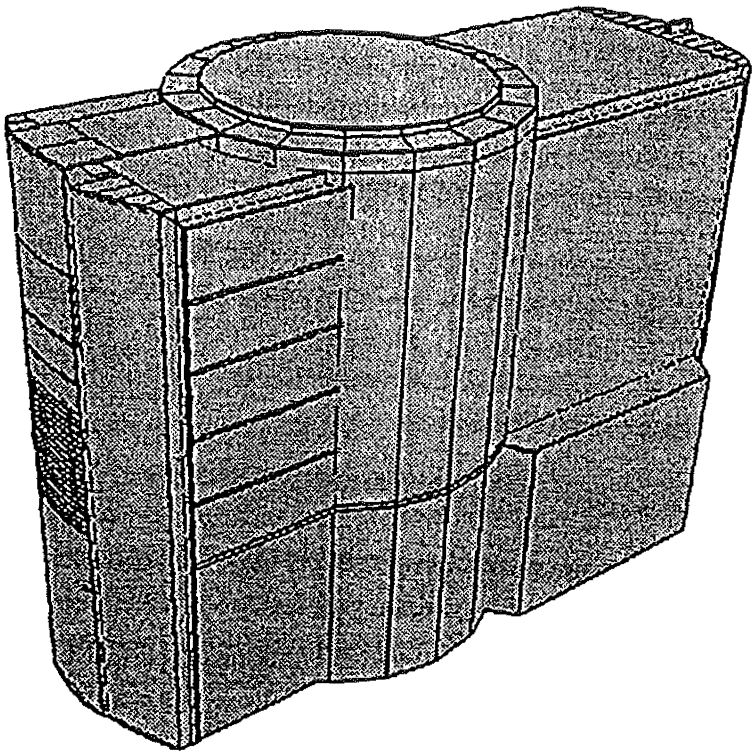


Kuva 1.1: Tyypillinen mikrotietokoneen kokoonpano. Keskusyksikkö (jossa prosessori ja keskusmuisti), näyttö, näppäimistö ja hiiri sekä kirjoitin. Keskusyksikön kanssa samaan koteloon on pakattu myös kovalevy ja levykeasema. Nykyisin kokoonpanoon kuuluu yhä useammin myös CD-ROM-asema sekä äänikortti kaiuttimineen. Tietoliikenneyhteyksiä varten laitteistoon kuuluu vielä modeemi tai verkkokortti.

nisiä eroja ei enää ole. Mikroissakin on jo 32-bitin sananpituus, ne pystyvät osoittamaan samankokoista muistia kuin isommatkin koneet ja niiden nopeus on kasvanut huimasti. Täten enää ei ole teknisiä perusteita erotella yleistietokoneita, pientietokoneita ja mikroja.

Tällä hetkellä mikrotietokoneidenkin kapasiteetti riittää jo useamman yhtäaikaisen käyttäjän tarpeisiin ja ne soveltuvat hyvin työasemakäyttöön. Mikrotietokoneen ja pientietokoneen välinen raja onkin kadonnut. Uusimmat mikrotietokoneet ovat huomattavasti tehokkaampia kuin muutaman vuoden takaiset pientietokoneet.

Selkein käsite luokittelussa on *superkone*. Sillä tarkoitetaan kaikkein tehokkaimpia, erityisesti tieteelliseen laskentaan kehitettyjä tietokoneita. Supertietokoneet sisältävät useita, jopa tuhansia erikoisrakenteisia ja rinnakkain toimivia vektoriprosessoreja. Supertietokoneissa päästään yli 1000 MFLOPS-yksikön tehoon (Millions of Floating Point Operations per Second). Vuonna 1995 tehokkaimmaksi mainitun supertietokoneen tehoksi ilmoitettiin yli 280 GFLOPS (Giga = 1000 Mega). Superkone soveltuu hyvin suurta laskentatarkkuuttta ja nopeutta vaativiin tehtäviin kuten esimerkiksi satelliittikuvien käsittelyyn ja differentiaaliyhtälöiden numeeriseen ratkaisuun.



Kuva 1.2: Tieteellisen laskennan palveluun asennettiin keväällä 1995 uusi supertietokone Cray C94. Koneessa on 4 vektoriprosessoria, jotka käyttävät jaettua muistia. Keskusmuistia koneessa on 1 GB ja sen välittömässä yhteydessä on 4 GB puolijohdemuistia. Kunkin prosessorin teoreettinen nopeus on noin 1 GFLOPS.

Työasema ja palvelimet

80-luvun alussa alettiin siirtyä keskitetystä tietojenkäsittelystä kohti hajautettua tietojenkäsittelyä. Tämä tarkoitti ensisijaisesti sitä, että tietoteknisiä laitteita vietiin myös atk-keskusten ulkopuolelle. Aluksi kyseessä olivat edelleen keskuskonepohjaiset pääteyhteydet.

Vähitellen *tietokoneverkkoon* (computer network) liitettiin entistä älykkäämpiä laitteita ja järjestettiin erillisten tietokoneiden välinen tiedonsiirto. Myös tiedonkäsittelykykyä hajautettiin siten, että tyhmien päätelaitteiden sijasta verkkoon kytkettiin koneita, joilla oli paikallista tiedonkäsittelykykyä. Toisin sanoen niillä oli oma prosessori ja omaa muistia. Alettiin puhua älykkäistä *työasemista* (workstation). Tyypillisessä työasemassa on kookas graafinen näyttö, näppäimistö ja hiiri, mutta siinä ei välttämättä ole paikallista talletuskapasiteettia.

Työasemakoneet liitetään kaapelointijärjestelyin ja liittimin *lähiverkkoon* (local area network, LAN), johon on kytketty myös *palvelimina* (server) toimivat tietokoneet. Palvelimiin on puolestaan kytketty muut tarpeelliset oheislaitteet kuten levyt,

kirjoittimet, piirturit jne. Työasema voi käyttää niitä verkon välityksellä.

Lähiverkoilla jaetaan tyypillisesti palvelimella olevaa levytilaa, jossa sijaitsee ohjelmistot, tietokannat, yhteiset kuva-arkistot jne. Kun tietoja halutaan käsitellä, lataa työasema ohjelmat ja niiden tarvitseman datan omaan muistiinsa verkon kautta. tämän jälkeen tiedon käsittely suoritetaan paikallisesti työasemassa.

Palvelimien ansiosta tiedot ovat keskitetysti yhdessä paikassa, mutta kuitenkin kaikkien saatavilla. Sen vuoksi esimerkiksi ohjelmistojen uusien versioiden asentaminen tai tiedostojen varmuuskopiointi on helppoa. Myös tietojen oikeellisuus on helpompi säilyttää, kun niistä ei ole käytössä useita kopioita.

Oheislaitteiden jaon lisäksi verkkoon kytkeytyminen sallii helpomman koneiden ja niiden käyttäjien välisen kommunikoinnin. Sähköpostin ja erilaisten tietopalvelujen kuten uutisryhmien ja Web-palvelun ansiosta viestien välitys sujuu mutkattomasti.

Lähiverkko voidaan kytkeä edelleen toisiin verkkoihin ja siten saadaan yhteydet myös muiden yritysten koneisiin, esimerkiksi pankkeihin ja julkisiin palveluverkkoihin.

1.4 Ohjelmasta prosessiksi

Käskyjonoa, joka kuvaa miten tietty tehtävä suoritetaan, kutsutaan *ohjelmaksi* (program). Ohjelma on tietojenkäsittelytehtävän esitys sarjana tietokoneen suoritettavaksi tarkoitettuja toimenpiteitä. Kaikkien tärkein tietokoneen ohjelmista on käyttöjärjestelmä: ilman käyttöjärjestelmää ei muitakaan ohjelmia voi käyttää.

Ohjelma muodostuu jollakin ohjelmointikielellä kirjoitetuista algoritmeista. Jotta prosessori voisi suorittaa ohjelman, on se ensin muutettava prosessorin ymmärtämään muotoon. Tietokoneen elektroniset piirit tunnistavat ja suorittavat suoraan vain hyvin rajallisen joukon yksinkertaisia käskyjä. Hyvin harvoin nämä käskyt ovat monimutkaisempia kuin laske yhteen kaksi lukua, tarkista onko luku nolla, siirrä tietoa keskusmuistista prosessoriin tai päinvastoin tai siirry ohjelmassa eteen- tai taaksepäin. Tietokoneen *käskykannalla* (instruction set) tarkoitetaan juuri niitä alkeellisia käskyjä, joita kone pystyy suorittamaan. *Konekielinen ohjelma* on ohjelman esitys kyseisessä käskykannassa.

Yleensä prosessorin tunnistama konekieli on pyritty tekemään niin yksinkertaiseksi kuin mahdollista (ottaen huomioon tietokoneen suunnitellun käyttötarkoituksen ja suorituskykytason). Syynä on tavallisesti tarvittavan elektroniikan monimutkaisuuden ja kustannusten pienentäminen. Konekielen yksinkertaisuuden ja koneenläheisen käsitemaailman vuoksi ihmisten on vaikeaa ja työlästä käyttää sitä.

Konekielen käytön hankaluutta voidaan vähentää luomalla käskykanta, jolla ohjelmoiminen on ihmiselle huomattavasti helpompaa kuin prosessorinläheisen konekielen käyttäminen. Myös nämä uudet käskyt muodostavat kielen, *lausekielen*. Lausekielinen ohjelma on muunnettava konekielen käskyiksi. Muunnos voidaan toteuttaa kahdella eri tavalla. Tapojen välinen periaatteellinen ero tulee näkyviin siinä, miten

tällä uudella kielellä kirjoitetut ohjelmat suoritetaan tietokoneessa.

Ensimmäinen vaihtoehto on korvata ensin kertaurakkana jokainen lausekielen lause yhdellä tai useammalla konekielen käskyllä. Näin saadaan tulokseksi uusi konekielinen ohjelmatiedosto, joka muistiin ladattuna tekee täsmälleen saman tehtävän kuin lausekielellä kirjoitettu ohjelma. Saatu konekielinen ohjelma tallennetaan esimerkiksi levylle ja se on suoritettavissa yhä uudelleen lataamalla se keskusmuistiin. Tätä tapaa kutsutaan *kääntämiseksi* (compilation).

Toinen tapa on ottaa uudella kielellä kirjoitetusta ohjelmasta käsiteltäväksi lause kerrallaan, tutkia mitä lauseen tulisi tehdä ja antaa prosessorin suoritettavaksi heti vastaavat konekieliset käskyt. Tätä tapaa kutsutaan *tulkitsemiseksi* (interpretation). Tässä menetelmässä ei muodosteta levylle erillistä konekielistä koodia, vaan tulkkaus ja suoritus vuorottelevat. Tämän vuoksi sekä tulkin että ohjelman on oltava yhtäaikaa keskusmuistissa.

Prosessi

Keskusmuistiin suoritettavaksi otettua ohjelmaa kutsutaan *prosessiksi*. Prosessin suorittaminen saa aikaan muutoksia keskusmuistin sisältöön eli prosessin käsittelemien muuttujien arvoihin, dataan. Ohjelmakoodi sisältää muutoksiin tarvittavat käskyt, mutta itse ohjelma on sinänsä passiivinen. Tarvitaan prosessori, joka suorittaa ohjelman käskyt ja tekee käskyissä määritellyt operaatiot muistissa olevalle datalle tai oheislaitteille.

Yhden käyttäjän järjestelmissä prosessorilla on suoritettavanaan vain yksi käyttäjän prosessi. *Moniajojärjestelmissä* (multiprogramming) voi muistissa sen sijaan olla yhtä aikaa useita sovellusprosesseja. Käyttöjärjestelmän tehtävänä on valita prosessoriin suoritettavaksi vuorotellen kukin sovellus sekä huolehtia sovellusprosessien tarvitsemista peruspalveluista. Näitä ovat mm. keskusmuistitilan varaaminen ja vapauttaminen, tiedostojen hallinta ja käyttö sekä keskusmuistin ja oheislaitteiden välinen siirräntä.

Moniajojärjestelmässä ohjelman ja prosessin ero korostuu. Jos nimittäin usea henkilö käyttää samaa ohjelmaa, syntyy monta prosessia. Sen sijaan ohjelman koodi on parhaimmassa tapauksessa vain yhteen kertaan keskusmuistissa, joskin toiset käyttöjärjestelmät monistavat myös ohjelman.

Prosessien tasapuolista *vuorottamista* (scheluding) varten laitteistoon kuuluu aikaviipalekello. Kun prosessori on suorittanut aikaviipaleen verran jotain tiettyä prosessia, se siirtyy suorittamaan käyttöjärjestelmän koodia. Tällöin käyttöjärjestelmä voi tehdä tarvittavat prosessorin asetusten muutokset, jotta prosessori saadaan suorittamaan taas seuraavaa prosessia. Käyttöjärjestelmän ja laitteiston tulee luonnollisesti huolehtia myös siitä, ettei mikään prosessi pääse häiritsemään jonkun toisen prosessin suoritusta. Tämän vuoksi käyttöjärjestelmää suorittaessaan prosessori voi suorittaa sellaisiakin ns. *etuoikeutettuja* (priviledged) käskyjä, jotka eivät ole sallittuja tavalliselle sovellukselle.

Jotta käynnistettyjen prosessien hallinta olisi mahdollista, ylläpitää käyttöjärjestelmä jokaisesta prosessista *prosessin kuvaajaa* (process control block, PCB). Sinne on kirjattu mm. prosessien tunnistetiedot, ohjelmakoodin ja datan sijainti, tietoa ohjelman käyttämistä tiedostoista sekä vuorottamisessa tarvittavaa tietoa. Prosessin kuvaajassa on myös tallealue rekistereiden arvoille, jotta prosessin vaihdon yhteydessä voidaan tallettaa siihen mennessä rekistereihin tehdyt muutokset.

Ohjelmakoodi itse on yleensä staattinen, muuttumaton, kun taas ohjelman data-alueen sisältö muuttuu käskyissä kuvatulla tavalla. Periaatteessa prosessi voisi muuttaa myös omaa koodiansakin. Mutta jos prosessi ei muuta koodia, voi useita saman ohjelman suorotuksia edetä rinnakkain, kunhan jokaista suorituksessa muuttuvaa osaa varten on kullekin prosessille omat varaukset (eli data-alueet). Ohjelman sanotaan tällöin olevan *vapaakäyntinen* (re-entrant).

1.5 Laitteisto

Tietokoneen elektroniset piirit pystyvät suorittamaan konekieliset ohjelmat ilman tulkkien ja kääntäjien apua. Nämä elektroniset piirit, muisti sekä syöttö- ja tulostuslaitteet muodostavat tietokoneen laitteiston. Laitteisto koostuu integroiduista piireistä, piirilevyistä, kaapeleista, virtalähteistä, muisteista, jne.

Monissa tietokoneissa on konekielisten käskyjen suorittamisessa tarvittavat ohjaus- ja ajoitussignaalit talletettu *mikrokoodina* pysyvästi lukumuistiin (read only memory, ROM). Mikrokoodin ja sitä tulkitsevan laitteiston tehtävänä on purkaa konekieliset käskyt vieläkin yksinkertaisempiin signaaleihin, jotka ohjaavat digitaalilogiikan veräjien ja kiikkujen yms. peruselektroniikan avulla tapahtuvaa käskyn suoritusta. Mikrokoodit ovat kiinteitä, ne on asennettu koneeseen jo valmistusvaiheessa, eikä käyttäjä voi niitä muuttaa.

Koska mikrokoodit ovat laitteen toiminnan kannalta yhtä tärkeitä kuin itse laitteisto (ilman niitä ei muukaan laitteisto toimi), kutsutaan mikrokoodeja ja tulkinnassa tarvittavaa elektroniikkaa joskus nimellä *laitelmisto* (laiteohjelmisto, firmware). Laitelmistoa käytetään yleensä paikoissa, missä ohjelmisto ei saa kadota missään olosuhteissa. Tällaisia ohjelmallisesti toteutettavia osia ovat myös laitteiston itsetestausrutiinit sekä käynnistämiseen liittyvät rutiinit.

Joainen ohjelmiston avulla suoritettava operaatio voidaan (ainakin teoriassa) korvata laitteistolla, joka suorittaa kyseisen operaation. Vastaavasti jokaista laitteiston suorittamaa operaatiota voidaan emuloida ohjelmistolla. Näinollen laitteisto ja ohjelmisto ovat tietokoneen toiminnan kannalta samanarvoisessa asemassa. Tärkeintä on toimintojen lopputulos, ei se, onko toiminnot toteutettu laitteistolla vai ohjelmistolla.

Mitään itsestäänselviä nyrkkisääntöjä ei ole siitä, että tietty operaatio pitäisi toteuttaa laitteistolla tai ohjelmistolla. Valintaa vaikuttavat mm. hinta, nopeus, luotettavuus sekä odotettavissa oleva muutostiheys. Suuntauksena on toteuttaa yhä enemmän toimintoja suoraan laitteistolla (syynä nopeus). Mikrokoodien ja yhteenso-

pivien tietokoneperheiden yleistyttyä on myös vastakkainen trendi yleistynyt (syynä helppo ylläpidettävyys).

Nykyisissä tietokoneissa on laite- ja/tai mikrokooditasolla monia operaatioita, jotka ensimmäisissä koneissa suoritettiin vielä konekielisillä ohjelmilla. Esimerkiksi liukulukuaritmetiikka, aliohjelmien kutsuminen ja paluu aliohjelmista, laskurikäskyt (vakion lisääminen ja vähentäminen), merkkijonojen ja taulukoiden käsittely, kellon ylläpito ja ohjelman suorituksen keskeyttäminen ja toisen suorituksen aloittaminen ovat tällaisia operaatioita.

1.6 Virtuaalikone ja hierarkkinen konemalli

Tietokoneen käyttäjän kannalta on samantekevää, miten kukin käsky on toteutettu, vain prosessin lopputulos ja kenties sen nopeus ovat tärkeitä. Konekielellä ohjelmoivan ei siis periaatteessa tarvitse tietää mikrokoodin olemassaolosta eikä toteutuksesta mitään, lausekielellä ohjelmoivan ei tarvitse tietää konekielestä mitään eikä sovelluksen käyttäjän tarvitse tietää lausekielistä tai ohjelmoinnista mitään.

Tämän vuoksi on usein selkeämpää ajatella, että kullakin käyttäjällä on olemassa hypoteettinen tietokone, jonka konekielenä on juuri ne käskyt tai komennot, joita koneen käyttäjä käyttää. Kukin käyttäjä voi tarkastella asiaa huomioimatta hierarkiassa alemmalla tasolla olevia, hänen 'konekieltään' toteuttavia tasoja.

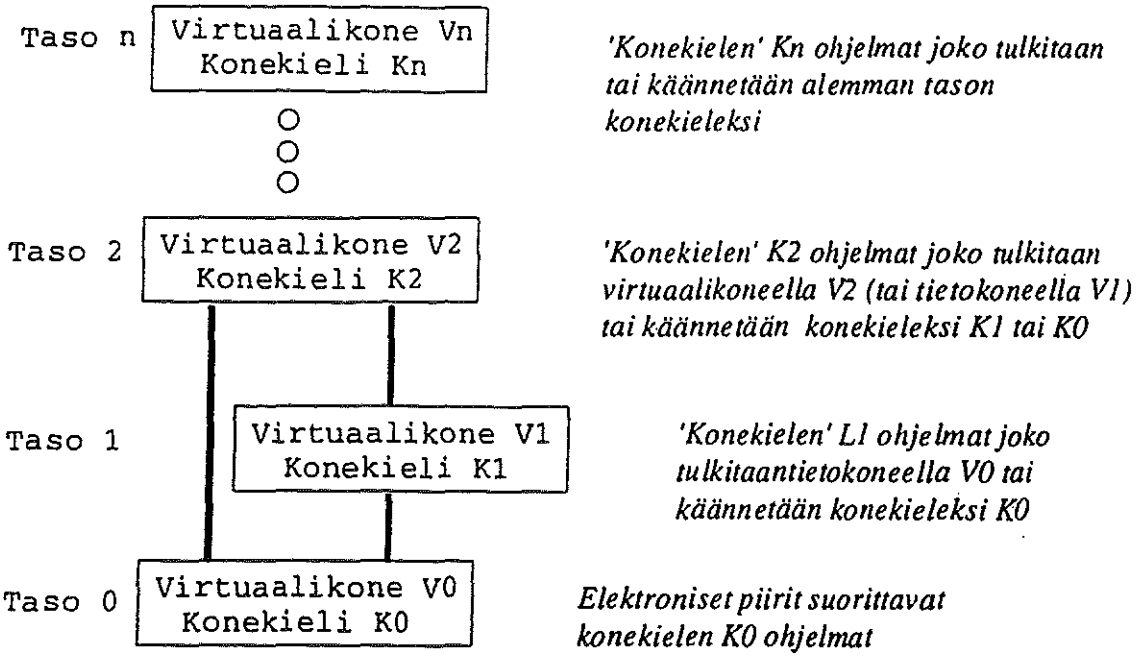
Kutakin tasoa voidaan kutsua *virtuaalikoneeksi*, sillä kukin käyttäjä voi aivan rauhallisesti pitää työskentelytasoaan todellisena fyysisenä tietokoneena, vaikka vastaavaa konetta ei tosiasiassa olisikaan olemassa.

Jokaisella virtuaalikoneella on tavallaan oma konekieli, joka käsittää kaikki ne käskyt, jotka ko. virtuaalikone osaa suorittaa. Tason virtuaalikone toteuttaa nämä toiminnot alempien tasojen avulla. Eri tasojen virtuaalikoneiden konekielien ei kuitenkaan tarvitse välttämättä olla erillisiä (kuva 1.3).

Kun hierarkian ylemmiltä tasoilta siirrytään alaspäin, lähestytään todellista fyysistä konetta ja virtuaalikoneiden tarjoamat palvelut ovat yksinkertaisempia ja lähempänä koneen kuin käyttäjän käsitteistöä.

Vastaavanlaiseen hierarkkiseen ajattelutapaan törmää tietojenkäsittelyssä tuon tuostakin. Sekä laitteistojen että ohjelmistojen suunnittelu ja toteutus on helpompaa tehdä pienempinä kokonaisuuksina, tasoittain. Kullakin tasolla on omat selkeät tehtävät. Taso toteuttaa tehtävänsä sopivaksi katsomallaan tavalla ja tarjoaa ne palveluina ylemmälle tasolle. Tason sisällä voi toteutuksessa käyttää alemman tason tarjoamia palveluja. Tason toteutusta voi muuttaa vapaasti, kunhan vain liittymä ylempään tasoon säilyy samanlaisena.

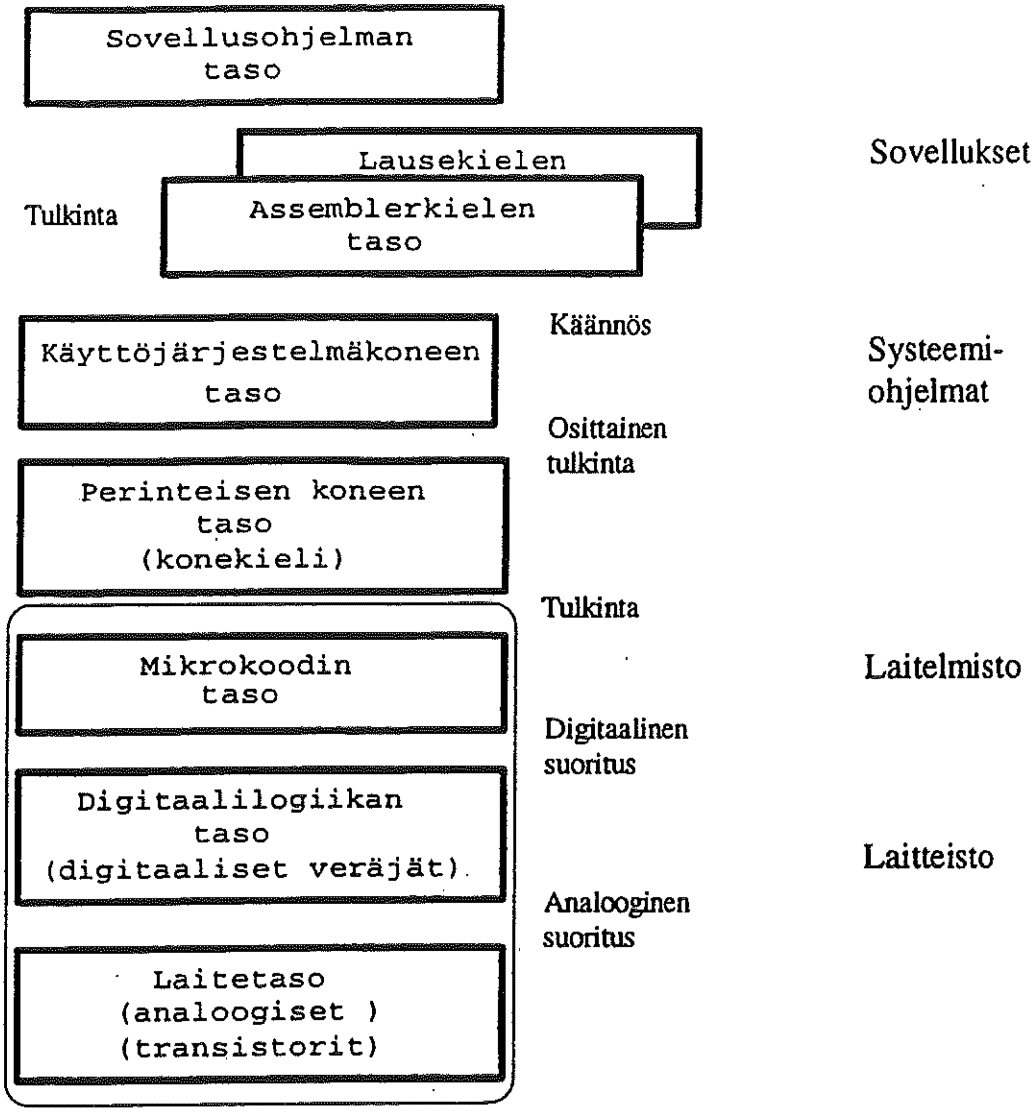
Ensimmäisissä tietokoneissa oli aluksi vain yksi taso, joka vastasi kuvan 1.4 tasoa *konekielen taso*. Ohjelma ja sen syötteet annettiin koneelle suoraan kytkimistä binäärilukuina, jota prosessori pystyi sellaisenaan tulkitsemaan, tai ohjelmointi oli johdotusten muuttamista.



Kuva 1.3: Virtuaalikonemalli.

50-luvulla kehitettiin ensimmäiset kaksitasoiset koneet, joihin oli lisätty *kääntäjä* reikäkorteilla olevien, aluksi symbolisella konekielellä (assembler) ja myöhemmin alkeellisilla lausekielillä esitettyjen ohjelmien suorittamiseksi. Myös kääntäjä oli reikäkorteilla ja se oli ladattava ensin koneen muistiin. Symbolisen konekielen ja alkeellisten lausekielien käyttönotto helpotti ohjelmointityötä, mutta koneen käyttö vaati kuitenkin edelleen kytkimien painelemista ja koneenkäyttäjää huolehtimaan mm. siitä, että oikeat tiedot olivat oikeaan aikaan reikäkortinlukijalla.

Seuraava askel oli kehittää koneen operointiin liittyviä rutiiniluonteisia tehtäviä helpottamaan *yksinkertainen komentotulkki* (command interpreter, shell), joka ladattiin nuistiin perusohjelmaksi. Sovellusohjelman sekaan laitettiin ohjauskortteja, jolloin tarvittaessa kontrolli voitiin siirtää komentotulkille esimerkiksi reikäkortinlukijan käynnistämistä varten. Komentotulkin voidaan katsoa olevan ensimmäinen askel kohti käyttöjärjestelmää.



Kuva 1.4: Hierarkkinen konemalli

Käyttöjärjestelmän kehittäminen lisäsi koneeseen kolmannen tason. Aluksi koneessa suoritettiin vain yhtä ohjelmaa kerrallaan. Vähitellen laitteistojen kehittyessä ja nopeutuessa kehitettiin myös käyttöjärjestelmää siten, että useiden ohjelmien yhtäaikainen suoritus oli mahdollista. Käyttöjärjestelmä huolehti laitteistoläheisistä toiminnoista ja siitä, että jokainen suoritettavaksi otettu ohjelma sai vuorollaan prosessoriaikaa.

Vanhimmissa tietokoneissa elektroniset piirit suorittivat suoraan konekielistä koodia. Vasta 70-luvulla alettiin koneita rakentaa siten, että konekäskyjen suorituksen vaatimat ajoitus- ja ohjaussignaalit talletettiin *mikrokoodina* muistiin. Laitteisto tulkitsi sekä suoritti mikrokoodin ohjaamana konekieliset käskyt. Mikrokoodien käyttöönotto lisäsi siis laitteistoon uuden tason konekielisen ohjelman ja laitteiston välille. Prosessorin käskynsuorituksen nopeuttamiseksi on jälleen luovuttu mikro-

koodeihin perustuvista toteutuksista. Nykyiset prosessorit perustuvat jälleen langoitukseen.

1.7 Tietokonearkkitehtuuri

Tietokoneen arkkitehtuurikuvaukseen kuuluu pääasiassa niiden komponenttien määrittely, joihin ohjelmoija voi viitata symbolisen konekielen tasolla. Näinollen arkkitehtuuri kuvaa osittain *ohjelmiston ja laitteiston välisen rajapinnan.* Arkkitehtuurin käsitteitä ovat rekisterit ja niiden koko, muistipaikat ja niiden osoittaminen. Arkkitehtuuri kiinnittää myös sen, millaisia käskyjä kuuluu prosessorin käskykantaan. Arkkitehtuurikuvauksessa ei kuitenkaan vielä kiinnitetä, millaisilla toimintoyksiköillä tai tekniikoilla toiminnat tullaan suorittamaan. Arkkitehtuurikuvauksesta käy ilmi, mitä toimintoja voidaan tehdä konekielellä, mutta ei se, kuinka ne tullaan tekemään.

Toimintoyksikköjen suunnittelupäätöksiä ovat mm. yksikköjen lukumäärä ja niiden tehtäväjako, toimintojen rinnakkaisuus, eri osien välisten väylien leveydet sekä osien väliset puskuroinnit. Nämä laitteistopiirteet eivät näy mitenkään ohjelmoijalle. Ohjelmoijan ei esimerkiksi tarvitse tietää, onko prosessorissa välimuistia (cache) tai käytetäänkö muistinhallinnassa virtuaalimuistia.

Kun laitteisto lopulta toteutetaan, kiinnitetään vielä laitteiston fyysiset yksityiskohdat. Näitä yksityiskohtia ovat esimerkiksi toteutusteknologia, tila, pakkaustiheys ja virrankulutus.

Esimerkiksi se, että prosessorissa on kertolaskukäsky, päätetään arkkitehtuurin tasolla. Organisaatiotasolla taas päätetään, toteutetaanko kertolasku laitteistotason algoritmilla vai mahdollisesti erillisellä kertolaskuyksiköllä. Kertolaskuoperaation nopeus puolestaan riippuu organisoinnin lisäksi myös käytetystä toteutustekniikasta.

Tässä monisteessa liikutaan prosessoriarkkitehtuurin, symbolisen konekielen ja käyttöjärjestelmän tasoilla. Laitteiston toteutukseen liittyvät yksityiskohdat oletetaan ratkaistuiksi.

Luku 2

Tiedon esittäminen ja sen oikeellisuuden tarkistus

Tiedon perusyksikkö tietokoneessa on *bitti* (engl. bit, lyh. b). Bitti on kaksijärjestelmän luku 0 tai 1. Bitit on järjestetty tavuiksi ja sanoiksi. *Tavu* (engl. byte, lyh. B) on pienin mahdollinen kerralla osoitettavissa oleva tietomäärä ja se sisältää 8 bittiä. Sananpituus vaihtelee koneesta riippuen: tietokonesana (engl. word, lyh. W) muodostuu yleensä kahdesta, neljästä tai kahdeksasta tavusta. Joskus suuremmista tavukokonaisuuksista käytetään nimitystä kaksoissana. Tietokonesanan bitit numeroidaan tavallisesti siten, että vähiten merkitsevä bitti (engl. least significant bit), eli oikeanpuoleisin bitti, on järjestysnumeroltaan 0.

Kaikki tietokoneessa esitettävä tieto, sekä käskyt että käsiteltävä data, on oltava binäärimuodossa. Näinollen on kullekin esitettävälle asialle annettava koodi binäärijärjestelmässä. On siis luotava säännöstö siitä, miten tietoa esitetään bittien avulla. Esitykseen varattavien bittien lukumäärä rajoittaa erilaisten mahdollisten koodien lukumäärää. Jos koodaukseen käytetään esimerkiksi 2 bittiä saadaan $2^2 = 4$ erilaista koodia ja jos koodaukseen käytetään 8 bittiä saadaan $2^8 = 256$ erilaista koodia.

2.1 Binäärilukujärjestelmä

2- eli binäärilukujärjestelmä on yksinkertaisin numeerisen tiedon esitysmuoto. Sen kantalukuna on 2 ja käytössä on vain kaksi numeroa 0 ja 1, joita kutsutaan biteiksi (engl. binary digit). Etuina vain kahden merkin käytöstä on se, että binäärijärjestelmään perustuvat laitteistot on helppo toteuttaa ja lisäksi merkit on aina luotettavasti erotettavissa toisistaan.

Jos lukuja esitettäessä halutaan korostaa sitä, mikä lukujärjestelmä on kyseessä, merkitään kirjallisessa esityksessä lukujärjestelmän kantaluku alaindeksiksi numerosarjan perään. Esimerkiksi 10112_2 tai 101110_{10}.

Luvun arvo riippuu binäärijärjestelmässä 10-järjestelmän tavoin sekä numeroiden

arvoista että niiden paikasta luvun esityksessä. Arvo saadaan muodostettua siten, että kukin numero kerrotaan sen paikan perusteella määräytyvällä painokertoimella ja lopuksi tulot lasketaan yhteen. Yksittäisen numeron painokerroin saadaan korottamalla kantaluku paikan osoittamaan potenssiin.

Esimerkki.

$$\begin{aligned} 1101000.11 &= 1*2^6+1*2^5+0*2^4+1*2^3+0*2^2+0*2^1+ \\ &\quad 0*2^0+1*2^{-1}+1*2^{-2} \\ &= 64+32+0+8+0+0+0+0.5+0.25 \\ &= 104.7510_{10} \end{aligned}$$

Kymmenjärjestelmässä annetun luvun kokonaisosa muunnetaan 2-järjestelmään jakamalla luku ja saatu osamäärä toistuvasti luvulla 2, kunnes osamäärä on 0. Tulos saadaan merkitsemällä jakojäännökset ylös käännetyssä järjestyksessä, ts. viimeksi saatu bitti on eniten merkitsevä (engl. most significant bit)

Esimerkki. $59_{10} = 111011_2$

59 : 2	=	29	jakoj.	1
29 : 2	=	14		1
14 : 2	=	7		0
7 : 2	=	3		1
3 : 2	=	1		1
1 : 2	=	0		1

Kymmenjärjestelmässä annetun luvun desimaaliosa muunnetaan 2-järjestelmään kertomalla desimaaliosaa toistuvasti luvulla 2. Tulos saadaan merkitsemällä kokonaisosat ylös saadussa järjestyksessä (ts. ensiksi saatu bitti on eniten merkitsevä).

Esimerkki. $0.73_{10} \approx 0.1011101_2$

$$\begin{aligned} 0.73*2 &= 1.46 \\ 0.46*2 &= 0.92 \\ 0.92*2 &= 1.84 \\ 0.84*2 &= 1.68 \\ 0.68*2 &= 1.36 \\ 0.36*2 &= 0.72 \\ 0.72*2 &= 1.44 \\ &\ldots \end{aligned}$$

Kirjallisessa esityksessä käytetään myös 8- eli *oktaalijärjestelmää* ja 16- eli *heksadesimaalijärjestelmää*. Esimerkiksi tavun sisältö voidaan esittää mukavasti kahden heksadesimaalinumeron avulla. Oktaalijärjestelmän kantaluku on 8 ja numerot 0, 1, 2, 3, 4, 5, 6 ja 7. Heksadesimaalijärjestelmässä kantalukuna on 16 ja sen numerot 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, ja F.

Muunnokset 8- ja kymmenjärjestelmän sekä 16- ja kymmenjärjestelmän välillä ovat harvinaisempia. Ne tehdään samalla tavalla kuin ylläesitetyt binääri- ja kymmenjärjestelmän väliset muunnokset, vain jakajana ja kertojana oleva kantaluku vaihtuu. 2-, 8- ja 16-järjestelmien väliset muunnokset kannattaa yleensä tehdä 2-järjestelmän kautta ryhmittelemällä bittejä 3:n ja 4:n bitin ryhmiin binääripisteestä alkaen.

Esimerkki.

7	A	5.	C	16-järjestelmä
0111	1010	0101.	1100	2-järjestelmä

3	6	4	5.	6	8-järjestelmä
011	110	100	101.	110	2-järjestelmä

2.2 Datan esitys

Lausekielisissä ohjelmissa tietoa käsitellään aina muuttujien avulla. Muuttujien esittelyn yhteydessä määritellään muuttujille myös tyyppi. Sen perusteella kielen kääntäjä varaa muuttujalle tilaa sekä päättää, kuinka muuttujan arvo esitetään binäärimuodossa. Binäärimuodossa olevan tiedon kulloinenkin tulkinta suoritetaan käskykoodin perusteella. Tietokone ei siis aseta mitään tulkintaa tietokonesanan tai -tavun sisällölle. Bittijonon tulkinta määräytyy yksinomaan sitä käsittelevän käskyn perusteella. Niinpä symbolisella konekielellä ohjelmoija voi halutessaan laskea vaikkapa alunperin kirjaimiksi tarkoitettuja bittijonoja yhteen. Sen sijaan useimmat lausekielen kääntäjät tarkistavat käännöksessä, että muuttujia käytetään vain niiden tyypin kertomalla tavalla.

Esitettävän datan päätyypit ovat kokonaisluvut, reaaliluvut ja merkit. Näiden esitys on otettu osittain huomioon jo koneen suunnitteluvaiheessa. Muiden ohjelmointikieliin kuuluvien tyyppien ja tietorakenteiden (engl. data structure) esitystapa määräytyy kielen ja laitteiston perusteella. Tarkastelemme seuraavaksi eräitä tapoja mm. Javan muuttujien esittämiseksi.

Kokonaisluvut

Kokonaisluku (Javan tyyppi int, monissa muissa kielissä integer) esitetään tallettamalla sen arvo binäärilukuna sanaan (16 tai 32 bittiä). Esimerkiksi luku 15 esitetään seuraavasti:

0	0000000	00000000	00000000	00001111
31		23	15	7

Negatiivisille luvulle on useampiakin esitysmuotoja:

- Ensimmäinen bitti tulkitaan etumerkiksi (1 = -, 0 = +) ja seuraavat bitit luvun itseisarvoksi.
- 2:n komplementtimuoto, jossa positiiviset luvut esitetään sellaisenaan, mutta negatiiviset komplementtimuodossa, eli $x = x$ ja $-x = 2^{\text{sananpituus}} - x$. Komplementtiesityksessäkin on huomioitava etumerkki.

2:n komplementtiesitys voidaan muodostaa helposti laitteistossa invertoimalla (kääntämällä) luvun ykköset nolliksi ja nollat ykkösiksi ja lisäämällä lopuksi 1. Käytettäessä 2:n komplementtimuotoa muuttuu myös vähennyslasku yhteenlaskuksi, ja sitä varten ei tarvita erillistä laitteistoa. Lisäksi vältytään kahdelta esitysmuodoltaan erilaiselta nollalta (+0 ja -0).

Esimerkki. $A - B = 89 - 41 = 48$

B:n itseisarvo:	00101001	esitykseen varataan tässä 8 bittiä
invertoidaan:	11010110	
lisätään 1:	1	
	11010111	B on talletettu tässä muodossa
lisätään A:	01011001	
	100110000	tulokseksi 48

Kun esitykseen varataan 8 bittiä, hylätään ylivuotanut yhdeksäs bitti. Ylivuotobitillä on merkitystä vain, jos molemmat yhteenlaskettavat ovat samanmerkkiset, mutta tuloksen merkki ei ole sama kuin yhteenlaskettavien merkki. Tällöin on laskennassa tapahtunut joko yli- tai alivuoto, eli saatu tulos ei mahdu esitykselle varattuun tilaan. Jos lasketaan $B - A$, on tulos negatiivinen ja etumerkkibitti (bitti 7) on 1. Luvun itseisarvon määräämiseksi on bittijono komplementoitava uudelleen.

Esimerkki. $B - A = 41 - 89 = -48$

A:n itseisarvo:	01011001	
invertoidaan:	10100110	
lisätään 1:	1	
	10100111	$167 = 2^8 - A = 256 - 89$
lisätään B:	00101001	41
	11010000	208, etumerkkibitti 1 = -
invertoidaan:	00101111	
lisätään 1:	1	
	00110000	48, joten tulokseksi tulee -48

Allaolevassa taulukossa on vertailtu 8-bittisiä kokonaislukuesityksiä. 2:n komplementtimuotoa käytettäessä suurin esitettävissä oleva luku on 127 ja pienin -128.

desimaaliesitys	etumerkki ja itseisarvoesitys	2:n komplementtiesitys
+0	$00000000 = 00_H$	$0000000 = 00_H$
−0	$10000000 = 80_H$	$0000000 = 00_H$
127	$01111111 = 7F_H$	$0111111 = 7F_H$
−127	$11111111 = FF_H$	$1000001 = 81_H$
−128	ei esitettävissä	$1000000 = 80_H$

Jos kokonaisluvun esitykseen varataan 16 bittiä, on suurin esitettävissä oleva kokonaisluku 32767 ja pienin -32768. Kun esitykseen varataan 32 bittiä, on lukualue $-2147483648 \cdots 2147483647$.

Reaaliluvut

Reaaliluvut (Javan tyyppi float, monissa muissa kielissä real) (esim. $0.23 * 10^{-4}$) esitetään liukulukumuodossa (engl. floating-point), jolloin kirjataan erikseen luvun *etumerkki* (engl. sign, S), *merkitsevät numerot* (mantissa, M) ja luvun *suuruusluokka* (eksponentti, E). Kunkin osan esittämiseen varattavien bittien lukumäärä määrää laskentatarkkuuden sekä suurimman ja pienimmän esitettävissä olevan luvun.

Liukuluku esitetään tavallisesti normeeratussa muodossa, jossa 1/kantaluku $\leq$ mantissa < 1 eli mantissan merkitsevin numero on nollasta poikkeava. *Kantalukuna* käytetään jotain 2:n potenssia (esimerkiksi 2 tai 16). Eksponentin merkin esittämiseen ei käytetä tulkintaa etumerkki ja luvun suuruus, vaan negatiivisten arvojen esittämiseen varaudutaan valitsemalla jokin sopiva positiivinen arvo *nollatasoksi* (engl. bias). Tämä arvo lisätään todelliseen eksponenttiin ja näin saatu arvo

talletetaan varattuun tilaan. Kantalukua ja nollatasoa ei tarvitse erikseen esittää, sillä ne pysyvät muuttumattomina.

Liukulukujen esitykselle on IEEE:n (engl. Institute of Electrical and Electronic Engineers) laatimat standardit erikseen 32 bitin yksinkertaisen tarkkuuden (engl. single precision) liukuluvulle ja 64 bitin kaksinkertaisen tarkkuuden (engl. double precision) esitysmuodoille. Allaolevassa taulukossa on esitetty eri kenttien pituudet molemmissa IEEE:n standardeissa.

	Yksinkertainen tarkkuus	Kaksinkertainen tarkkuus
Etumerkki (S)	1b	1b
Eksponenttiosa (E)	8b	11b
Mantissaosa (M)	23b	52b
Nollataso	127	1023
Lukualue	n. 10^{74}	n. 10^{614}
Merkitseviä numeroita	7	16

IEEE:n standardissa kantaluku on 2, joten normeeratussa muodossa mantissan ensimmäinen bitti on aina 1. Niinpä sitä ei tarvitse tallettaa ja saadaan yksi merkitsevä bitti lisää mantissaan. Luvun esityksessä näkymätöntä 1-bittiä kutsutaan *piilobitiksi* (engl. hidden bit). Liukulukujen laskutoimituksissa piilobitti otetaan luonnollisesti esiin. Luvun arvo määräytyy nyt, piilobitti huomioiden, kaavalla

$$-1^S * 1.M * 2^{E-127}.$$

Luku 0 esitetään siten, että sekä mantissaosa, että eksponenttiosa ovat molemmat nollia. Standardissa on varauduttu myös luvun ∞, normeeraamattoman luvun ja virheellisen laskutoimituksen (esim. $7/0$ tai $\sqrt{-5}$) esittämiseen. Alla olevassa taulukossa on kuvattu näiden esitystä yksinkertaisen tarkkuuden liukulukuna.

	$E = 0$	$E = 1 \cdots 254$	$E = 255$
$M = 0$	$-1^S * 0$	$-1^S * 1.M * 2^{E-127}$	$-1^S * \infty$
$M \neq 0$	$-1^S * 0.M * 2^{-126}$	$-1^S * 1.M * 2^{E-127}$	Virhe: ei numero

Esimerkki. Luvun 43.75 esitys IEEE:n yksinkertaisen tarkkuuden liukulukuna:

$$\begin{aligned} 43.75 &= 101011.11_2 \\ &= 1.0101111 * 2^5 \\ &= 1.0101111 * 2^{132-127} \\ &= 1.0101111 * 2^{10000100-127} \end{aligned}$$

S	E	M		
0	10000100	0101111	00000000	00000000
31	23			

Reaalilukuja suljetulla välillä on ylinumeroituva määrä, liukulukuja äärellinen määrä. Siten liukulukuaritmetiikka ei tuota aina aivan tarkkoja tuloksia. Suuria ongelmia laskentatarkkuuden kanssa syntyy etenkin silloin, kun on tarvetta suorittaa laskutoimituksia sekaisin sekä pienillä että suurilla luvuilla.

Liukuluvuilla laskenta tehdään liukulukujen käsittelyyn erikoistuneella prosessorilla tai ohjelmallisesti emuloimalla, jolloin kääntäjä liittää tulosmoduuliin tarvittavat rutiinit. Tarvittaessa laitteisto / ohjelmisto suorittaa kokonaislukuesityksen muunnoksen liukulukumuotoon. Liukulukujen käsittelyyn erikoistuneella laitteistolla laskenta voi sujua jopa 10 kertaa nopeammin kuin ohjelmallisesti tehtynä.

Merkit

Yhden merkin (Javan tyyppi char) tallettamiseksi varataan tilaa yksi tavu. Kaikille sallituille merkeille on määritelty binäärinen merkkikoodi. Aikojen kuluessa tietokonevalmistajat ovat määritelleet useitakin tällaisia koodeja. Tavallisimmat merkkikoodistot ovat alkuaan 7-bittinen *ASCII* (engl. American Standard Code for Information Interchange) ja 8-bittinen *EBCDIC* (engl. Extended Binary Code Decimal Interchange Code). Alkuperäisessä ASCII-koodistossa on 128 erilaista merkkiä ja EBCDIC-koodistossa 256.

Koska kansainvälisissä merkistöissä on runsaasti eroja, on käyttöön yleistymässä myös 16-bittinen *Unicode*-koodisto. Tällöin merkistössä voi olla yli 65 000 koodia.

Nykyään on esitysstandardiksi vakiintunut ISO Latin -merkistö. Se perustuu 8-bittiseen ASCII-koodiin.

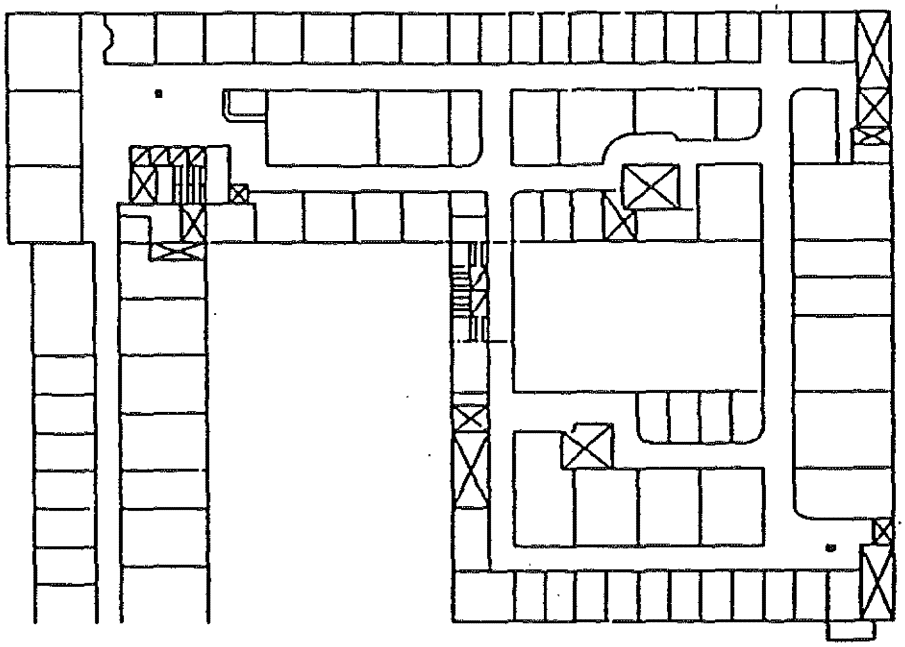
Kuvat

Tietokoneessa käsitellään tavallisesti kahden tyyppisiä kuvia, viiva- eli vektorikuvia ja rasterikuvia. Kummatkin voivat olla joko mustavalkoisia tai värillisiä.

Vektorikuvat muodostuvat joukosta vektoriviivoja. Vektori on viivan osa tai käyrä, joka perustuu matemaattiseen yhtälöön. Muistiin kuva talletetaan kirjaamalla kuvan kaikkien vektoreiden koordinaatit, piirtokaavat sekä vektoreiden rajaamien alueiden värit.Tehokkuussyistä käytetään tavallisesti koordinaatistoa, jossa kaikki koordinaatit ovat kokonaislukuja.

Rasterikuvat talletetaan muistiin bittikarttoina (engl. bitmap). Kustakin kuvan pisteestä on talletettu sen väri. Mustavalkokuvissa jokaista kuvan mustaa pistettä vastaa bitti 1 ja jokaista valkoista pistettä vastaa bitti 0. Harmaasävy- ja värikuvissa kunkin kuvapisteen väri esitetään 8:lla tai 24:lla bitillä. Tällöin saadaan esitettyä 256 tai yli 16 miljoona värisävyä (ns. täysvärit).

Kuvatiedoston alussa on ennen varsinaista kuvatietoa otsake (engl. header). Siihen on talletettu mm. kuvan dimensio, talletusformaatin tunniste ja 8:lla bitillä koodattaessa vielä tieto siitä miten värinumerot on valittu (ns. paletti). Täysvärikuvissa ei tarvita erillistä palettia. Tavallisimmin värikoodauksessa käytetään RGB-järjes-

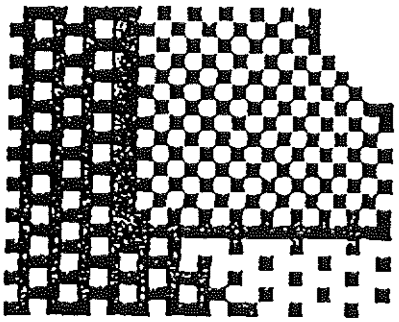


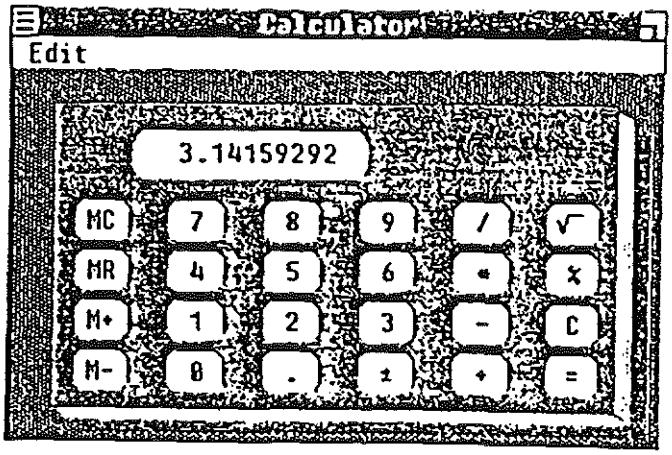
Kuva 2.1: Viivapiirros tietojenkäsittelytieteen laitoksesta.

telmää, jossa kukin väri esitetään punaisen, vihreän ja sinisen värin intensiteetin arvona. Kunkin arvon esittämiseen käytetään 8 bittiä. Esimerkiksi kolmikko (0,0,0) tuottaa mustan värin, kolmikko (255,255,255) valkean värin ja kolmikko (255,0,255) violetin värin.

Kaikki kuvankäsittelyohjelmat osaavat käsitellä pakkaamattomia bittikarttoja. Niiden lisäksi on lukuisia muita talletusmuotoja, jotka pyrkivät useita eri *pakkausmenetelmiä* (engl. compress) käyttäen tallettamaan bittikartan mahdollisimman vähän tilaa vievään muotoon. Osa noista talletusmuodoista hukkaa samalla alkuperäisten kuvien väritietoa. Siitä ei kuitenkaan ole välttämättä haittaa, sillä tuota hukkaamista ei kovin helposti huomaa ihmissilmin. Kuvan käytön yhteydessä pakkaaminen puretaan jälleen tavalliseksi bittikartaksi. Tällä hetkellä yleisimpiä talletusmuotoja ovat mm. GIF (engl. Graphics Interchange Format), JPEG (engl. Joint Photographics Expert Group) ja TIFF (engl. Tagged Image File Format).

Mikrojen tehon kasvun ja CD-ROM-laitteistojen yleistymisen myötä on myös multimedia ja liikkuvan kuvan käyttö yleistynyt. Pakkaamattomana hyvälaatuinen liikkuva kuva (25 kuvaa / s) vie helposti tilaa yli 20 megatavua per sekunti. Pakkaamalla tietomäärä voidaan kuitenkin kutistaa murto-osaan alkuperäisestä. Tavanomaisten tiivistystapojen lisäksi tilasäästöä saadaan, kun kokonaisten peräkkäisten kuvien sijasta kuvatiedostoon talletetaan vain muutokset edelliseen kuvaan. Yleisimmät liikkuvan kuvan pakkausmuodot ovat MPEG (engl. Motion Picture Experts Group), AVI, MOV ja INDEO.






```
0111 0111 1010 1010 1010 1010 0000 ...
1101 1101 1101 0101 0101 0110 0000 ...
0111 0111 1010 1010 1010 1010 0000 ...
1101 1101 1101 0101 0101 0101 0000 ...
```

Kuva 2.2: Rasterikuva ja sen vasemman alanurkan esitys muistissa.

Ääni

Digitaalinen ääni esitetään käyttäen PCM-koodausta (engl. Pulse Code Modulation). Koodausta varten äänestä otetaan näyte esimerkiksi taajuudella 44,1 kHz, eli noin 44000 kertaa sekunnissa, ja se kvantisoidaan. Kukin näyte koodataan esimerkiksi 16:lla bitillä. Äänikorteissa käytetään yleisesti General-MIDI-standardia (engl. Musical Instruments Digital Interface), joka standardoi näytteiden numerokoodit. Ääntä talletettaessa esitys vielä pakataan tilan säästämiseksi.

2.3 Käskyjen esitys

Käskyjen koodauksesta ei ole olemassa yleispäteviä sääntöjä, vaan kukin laitevalmistaja on räätälöinyt käskykantaansa omien tarpeitten ja laitteen käyttötarkoituksen mukaan. *RISC*-koneiden (engl. Reduced Instruction Set Computer) valmistajat pyrkivät selviämään mahdollisimman pienellä käskymäärällä (esim. RISC 1:ssä 31 käskyä) ja yksinkertaisilla tietotyypeillä. *CISC*-koneissa (engl Complicated Instruction Set Computer) on sen sijaan miltei jokaista ajateltavissa olevaa käyttötarkoitusta varten oma konekäskynsä. Esimerkiksi VAX-järjestelmissä on noin 340 käskyä mm. erikseen tavuilla, sanoilla ja kaksoissanoilla operoivat käskyt. Intel-prosessoreiden käskykantaan kuuluu lähes 200 käskyä. RISC-koneissa ohjelmat ovat noin 70% pitempiä, mutta pieni käskykanta mahdollistaa koneen helpon ja tehokkaan toteu-

tuksen, jolloin yksittäisen käskyn suoritusaika on vastaavasti lyhyempi.

Yhteistä kaikille käskykannoille on se, että käskyjä ohjaussignaaleiksi dekoodattaessa on osattava erottaa *operaatiokoodi*, käskyn *operandit* sekä niiden *tyyppi* ja *sijainti*. Operaatiokoodi määrää suoritettavan toiminnon lisäksi myös sen, kuinka käskyn loppuosa tulkitaan eli kuinka määritetään käskyn operandien sijainti. Operaatiokoodi antaa myös ohjeen tulkita operandit. Esimerkiksi operaatiokoodi 'ADD' määrää, että yhteenlaskettavien operandien sisältö on tulkittava kokonaisluvuiksi ja operaatiokoodi 'ADDF' , että operandit on tulkittava liukuluvuiksi. Sen sijaan muistista prosessoriin tietoa lataava käsky 'LOAD' ei tarvitse operandiensa sisällölle tulkintaa: se vain siirtää bitit paikasta toiseen.

Saman koneen käskykantaan kuuluvat käskyt saattavat olla eripituisia: Joissakin käskyissä riittää pelkkä operaatiokoodi, eräissä käskyissä tarvitaan vain yksi operandi, kun taas joissakin käskyissä operandeja voi olla useampiakin. Esimerkiksi VAX-järjestelmissä käskyn operaatiokoodi voi olla 1 tai 2 tavua pitkä, operandeja voi olla 0 - 6 kappaletta ja kukin operandi vie tilaa 1 tai 2 tavua.

Vaihtelevanmittaisten käskyjen etuna on se, että lyhyet käskyt eivät vie ylimääräistä muistitilaa ja lyhyiden käskyjen nouto ja käsittely nopeutuu. Vastaavasti käskylaskurin kasvatus on hieman ongelmallisempaa: ennen käskylaskurin kasvatusta on noudettava ainakin osa käskystä prosessoriin, jotta voidaan päätellä käskyn koko pituus.

2.4 Tiedon oikeellisuus

Silloin tällöin voi myös tietokoneessa syntyä virheitä. Tavallisin virhe on jonkun bitin muuttuminen muistissa tai tietoa siirrettäessä. Tiedonsiirto oheislaitteiden ja muistin välillä on virhealttiimpaa kuin prosessorin ja muistin väliset siirrot, sillä oheislaiteliitännät ovat fyysisesti suojattomammat kuin keskusyksikön sisällä olevat osat ja oheislaitteissa on myös liikkuvia osia. Virheiden havaitsemista varten on tiedon yhteyteen talletettu muutama lisäbitti, joiden avulla laitteisto (esim. keskusmuisti) voi suorittaa tarkistuksia.

Pariteetti

Helpoin tapa havaita yhden bitin muuttuminen on lisätä yksi ylimääräinen *pariteettibitti*. Pariteettibitille lasketaan arvo tavussa tai sanassa olevien 1-bittien lukumäärän perusteella. Jos pariteettibitti määrätään siten, että 1-bittien lukumääräksi pariteettibitti mukaanlukien tulee parillinen luku, saadaan aikaan *parillinen* pariteetti. Jos taas 1-bittien lukumääräksi tulee pariton luku, saadaan *pariton* pariteetti. Paritonta pariteettia käytetään yleisemmin kuin parillista pariteettia, sillä tällöin jokaisessa tarkistettavassa osassa on aina vähintään yksi 1-bitti.

Muistivirheiden tarkistamisen yhteydessä laitteisto laskee 1-biteistä varmistussummaa ja vertaa sitä muistissa olevaan pariteettibittiin. Jos pariteetti on virheel-

linen, yritetään esimerkiksi siirtoa uudelleen tai annetaan ylemmälle tasolle virheilmoitus.

Yhden pariteettibitin käyttö ei pysty huomaamaan tilanteita, joissa esim. kaksi bittiä invertoituu. Se ei myöskään pysty ilmaisemaan, mikä bitti on virheellinen. Virheen havaitseminen onkin huomattavasti helpompaa kuin sen korjaaminen.

Hammingin koodi

Monimutkaisempi varmistuskeino on käyttää Hammingin koodia varmistuksiin. Sen avulla pystytään paikallistamaan ja korjaamaan yhden bitin virheet ja havaitsemaan kahden tai useamman bitin virheet. Menetelmä ei ole riippuvainen tarkistettavan kokonaisuuden pituudesta. Koodin käyttö ei näy käyttäjällä: Pariteettibitit muodostetaan ja lisätään automattisesti laitteistolla ja poistetaan, kun tietoa käytetään.

Hammingin koodissa on pariteettibittejä useampia. Tarkistusbittien lukumäärä määräytyy tarkistettavan kokonaisuuden pituuden mukaan. Mukaan lisätään pariteettibittejä siten, että kaikki bitit, joiden järjestysnumero alusta lukien on jokin 2:n potenssi, ovat tarkistusbittejä. Siten pariteettibittejä ovat bitit 1, 2, 4, 8, 16, 32, 64, 128 jne. Pariteettibiteille määrätään arvo käyttäen parillista pariteettia.

Kullakin pariteettibitillä tarkistetaan vain tiettyjen bittien oikeellisuutta. Alustalukien n:nettä bittiä tarkistavat ne pariteettibitit, joiden järjestysnumerojen summa on n. Luvun n bittiesityksessä on ko. painoa vastaavissa kohdissa bitti 1. Esimerkiksi bittiä numero 7 tarkistavat bitit 4, 2 ja 1 (7 = 111), ja bittiä numero 27 tarkistavat bitit 16, 8, 2 ja 1 (27 = 11011).

Esimerkki. Olkoon talletettuna teksti 'TT'. Sen ASCII-koodi on 01010100 01010100.

	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
			0		1	0	1		0	1	0	0	0	1	0		1	0	1	0	0
bitti 1:	?	.	0	.	1	.	1	.	0	.	0	.	0	.	0	.	1	.	1	.	0
bitti 2:		?	0	.	.	0	1	.	.	1	0	.	.	1	0	.	.	0	1	.	.
bitti 4:				?	1	0	1	.	.	.	.	0	0	1	0	.	.	.	.	0	0
bitti 8:								?	0	1	0	0	0	1	0	.	.	.	.	.	.
bitti 16:																?	1	0	1	0	0

Biteissä 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 ja 21 on parillinen määrä 1-bittejä (4), joten bitiksi 1 tulee 0.

Biteissä 2, 3, 6, 7, 10, 11, 14, 15, 18 ja 19 on parillinen määrä 1-bittejä (4), joten bitiksi 2 tulee 0.

Biteissä 4, 5, 6, 7, 12, 13, 14, 15, 20 ja 21 on pariton määrä 1-bittejä (3), joten bitiksi 4 tulee 1.

Biteissä 8, 9, 10, 11, 12, 13, 14 ja 15 on parillinen määrä 1-bittejä (2), joten bitiksi 8 tulee 0.

Biteissä 16, 17, 18, 19, 20 ja 21 on parillinen määrä 1-bittejä (2), joten bitiksi 16 tulee 0.

(Huomasit varmaan, että esim. bitti 2 tarkistaa järjestysnumerosta 2 alkaen ensin kahta bittiä, jättää kaksi seuraavaa väliin, tarkistaa taas kahta jne. Vastaavasti bitti 4 tarkistaa järjestysnumerosta 4 alkaen ensin neljää bittiä, jättää seuraavat neljä väliin, tarkistaa neljä seuraavaa jne.)

Virheen löytämiseksi muodostetaan kukin pariteettibitti uudelleen (tällä kertaa myös tarkistusbitit on tarkistettavana) ja verrataan niitä edellisiin tarkistusbitteihin. Jos kaikki tarkistusbitit täsmäävät, on tieto säilynyt oikeana. Virheellinen bitti voidaan paikantaa ja korjata laskemalla virheellisten pariteettibittien järjestysnumerot yhteen.

Jos esimerkiksi edellä esitettyä merkkiparia vastaanotettaessa saadaan bittijono

0 0 0 1 1 0 1 0 0 1 0 0 1 1 0 0 1 0 1 0 0

huomataan, että tarkistusbitit 1, 4 ja 8 ovat virheellisiä. Virheellinen bitti on bitti numero 13, se on muuttunut 0:sta 1:ksi. Yhden virheellisen bitin korjaaminen on ainakin teoreettisesti mahdollista, mutta käytännössä virheen havaitseminen tiedonsiirrossa aiheuttaa aina uudelleenlähetyksen.

(Välikysymys: Miten suuri osuus kokonaisbittimäärästä kuluu tarkistusbitteihin?)

Luku 3

Keskusyksikkö

Kuvassa 3.1 on esitetty esimerkkikoneemme keskusyksikkö. Keskusyksikköön kuuluvat *prosessori* (engl. central processing unit, CPU), *keskusmuisti* (engl. memory, MEM) sekä *laiteohjaimet* (engl. controller). Keskusyksikkö kootaan emolevyllä (engl. mother board) olevien *väylien* (engl. bus) yhteyteen. Väylillä on liittimet laajennuskortteja varten lisälaitteiden liittämiseksi.

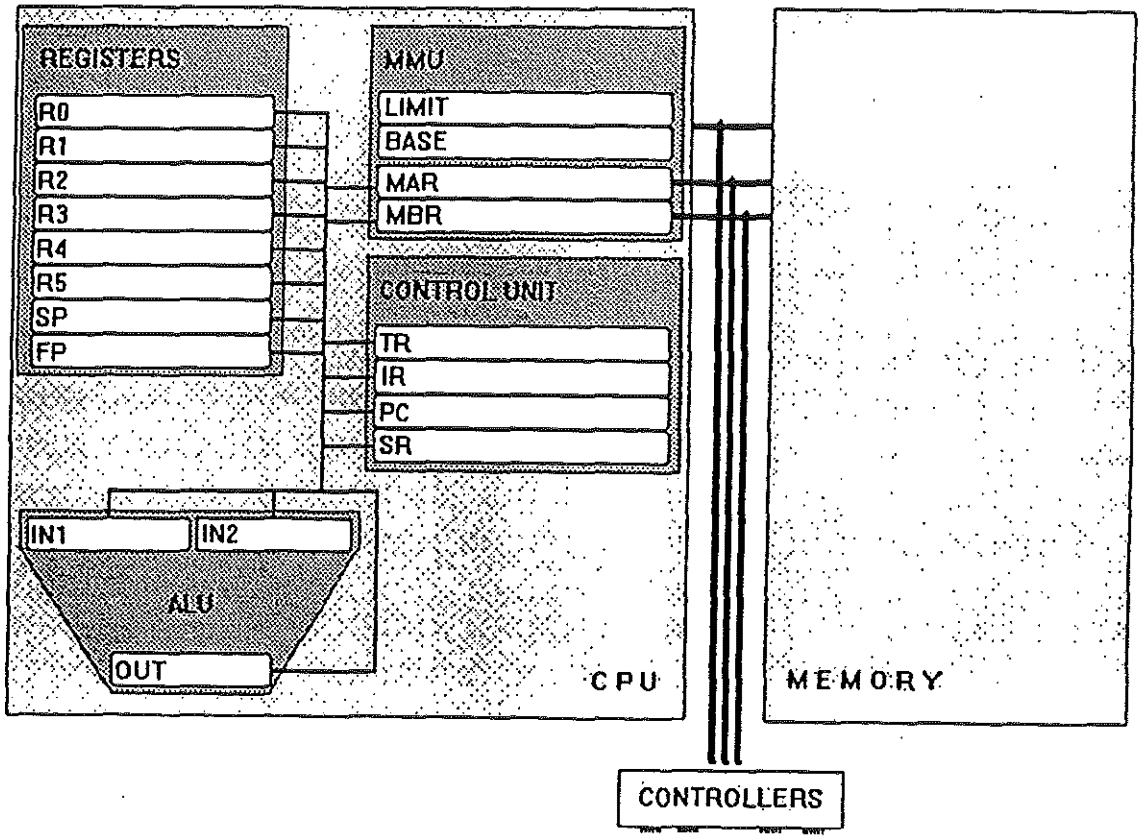
Tärkein keskusyksikön osista on ehdottomasti prosessori. Prosessori suorittaa keskusmuistissa olevia ohjelmia ja ohjaa niiden perusteella muuta laitteistoa. Prosessori noutaa muistista suoritettavat käskyt sekä niiden tarvitseman datan. Tulokset prosessori tallettaa takaisin muistiin. Kun ohjelma tarvitsee tietoa oheislaitteelta tai haluaa tulostaa jollekin laitteelle, se pyytää palvelua käyttöjärjestelmältä ja prosessori käynnistää tarvittavan siirron. Siirto keskusmuistin ja oheislaitteen välillä tapahtuu laiteohjaimen ohjauksen alaisena. Kun ohjain on tehnyt siirron, aiheuttaa laiteohjain prosessorille *keskeytyksen* (engl. interrupt). Tällöin prosessori voi käydä tarkistamassa ohjaimelta kuinka siirto onnistui.

3.1 Väylät

Väylä on siirtotie, johon voidaan liittää prosessori-, muisti- ja oheislaiteohjainkortteja. Kullakin väylään liitetyllä laitteella on oma osoitteensa, jonka se tunnistaa väylältä.

Väylän johtimet voidaan jakaa *data-*, *osoite-* ja *ohjausväylään* (engl. data bus, address bus, control bus). Esimerkiksi kirjoittaessaan muistiin prosessori varaa ensin väylän käyttöönsä ohjausväylän avulla. Tämän jälkeen se siirtää osoiteväylälle kirjoitettavan tiedon osoitteen ja dataväylälle siirrettävän tiedon. Siirto käynnistyy, kun prosessori ilmoittaa ohjausväylää pitkin, että kyseessä on kirjoitusoperaatio. Kaikki väylään kytketyt ohjaimet 'kuuntelevat' väylällä liikkuvia osoitteita. Kun esimerkiksi muistikortti havaitsee itseensä liittyvän osoitteen ja kirjoitusohjauksen, se siirtää tiedon väylältä omiin muistipiireihinsä.

Eri laitteistoissa data- ja osoiteväylien leveydet ja nopeudet vaihtelevat jonkin



Kuva 3.1: Esimerkkikoneemme keskusyksikkö

verran. Dataväylän leveys ei ole aina sama kuin esimerkiksi sananpituus ja lisäksi data- ja osoiteväylät on voitu yhdistää siten, että samoja johtimia pitkin lähetetään peräkkäin sekä osoite että data.

Suurissa laitteistoissa väylärakenne on usein laitteistovalmistajan itse suunnittelema, jolloin tämä rajoittaa väylään liitettävien korttien valikoimaa. Pienissä koneissa käytetään sen sijaan yleiskäyttöisiä väyläratkaisuja, joka takaa myös lisäkorttien runsaan saatavuuden. PC-mikron tavallisimmat emolevyn väylät ovat ISA (engl. Industri Standard Architecture, dataväylä 16b), VLB (engl. Vesa Local Bbus, dataväylä 32b) ja PCI (engl. Peripheral Component Interconnect, dataväylä 32 tai 64 b).

Prosessori	Prosessorin sisällä	Prosessorin ulkopuolella	Osoiteväylä
8088	16 b	8 b.	20 b
8086	16 b	16 b	20 b
80286	32 b	16 b	24 b
80386	32 b	32 b	32 b
80486	64 b	32 b	32 b
Pentium	64 b	32/64 b	32 b

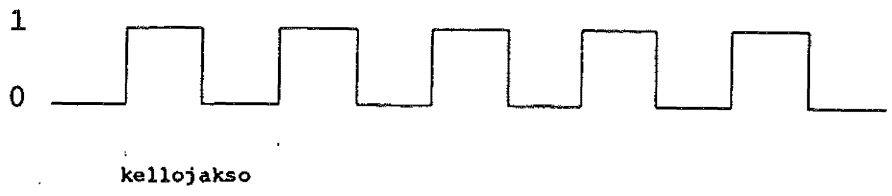
3.2 Prosessori CPU

Prosessori on tietokoneen keskeisin komponentti. Se tulkitsee ja suorittaa käskyt eli käsittelee dataa käskyjen määräämällä tavalla sekä ohjaa kaikkea muuta toimintaa generoimiensa signaalien avulla. Prosessorin tärkeimmät rakenneosat ovat *ohjausyksikkö* (engl. control unit, CU), *aritmeettis-looginen yksikkö* (engl. arithmetic-logical unit, ALU), *rekisterit* (engl. registers) sekä *muistinhallintayksikkö* (engl. memory management unit, MMU). Liukulukuaritmetiikkaa varten prosessoriin kuuluu usein vielä liukulukuyksikkö (engl. floating-point unit, FPU).

Ohjausyksikkö CU

Konekielisiä käskyjä vastaavat langoitukset ja/tai mikrokoodit sekä niiden tulkitsemiseen tarvittava laitteisto muodostavat ohjausyksikön. Se tulkitsee konekielisiä käskyjä ja ohjaa tiedon käsittelyä käskyjen perusteella generoitavilla ohjaussignaalikombinaatioilla.

Signaalien oikea-aikainen lähetys ja vastaanotto tapahtuvat kellon avulla, joka tuottaa värähtelyjä säännöllisin väliajoin. Kellojakso on pienin aika, jonka kuluessa Prosessori:n sisäisessä tilassa voi tapahtua muutoksia. Kellotaajuus puolestaan kertoo sen kuinka monta kellojaksoa kello tuottaa sekunnissa. Kellojakso on 5 ns - 1 mikros ja kellotaajuus vastaavasti 200MHz - 1 MHz. Laboratorio-oloissa on päästy piisiruihin, jotka vaihtavat tilaansa jopa 30 miljardia kertaa sekunnissa.



Kuva 3.2: Digitaalinen aika.

Aritmeettis-looginen yksikkö ALU

Useimmat käskyt suoritetaan aritmeettis-loogisessa yksikössä, ALUssa. ALU:ssa on kaksi sisäänmenoa operandeille ja yksi ulostulo tulokselle. Operaation tulos viedään ulostulosta edelleen johonkin rekisteriin. ALU:ssa suoritetaan kokonaislukuaritmetiikan lisäksi operandien vertailut sekä sivuttaissiirrot ja bittioperaatiot.

Rekisterit

Rekisterit ovat prosessorin sisäisiä muistipaikkoja. Niitä käytetään prosessia suoritettaessa tiedon lyhytaikaiseen varastointiin. Rekisterit erotetaan toisistaan, kuten tavalliset muistipaikatkin, osoitteen avulla. Rekisteriosoitteet ovat kuitenkin lyhyitä ja niitä on nopea käsitellä. Rekisterissä oleva tieto on lisäksi nopeasti käytettävissä, sillä rekisterit on toteutettu nopealla teknologialla eikä hidasta muistinoutoa tarvita. Operandien siirto rekistereistä vie vain muutamia nanosekunteja, kun niiden noutaminen muistista vie jopa kymmenen kertaa enemmän (esim. 70 - 120 ns).

Rekistereiden nimet ja niiden käyttötarkoitukset vaihtelevat eri arkkitehtuureissa. Tyypillisiä prosessorin rekistereitä ovat esimerkiksi ohjausyksikön CU rekisterit:

- käskyosoitin PC (engl. program counter tai instruction pointer, IP),
- käskyrekisteri IR (engl. instruction register),
- lippu / tilarekisteri SR (engl. status register),
- sisäiset työrekisterit TR (engl. temporary registers),

muistinhallintayksikön MMU rekisterit:

- muistiosoiterekisteri MAR (engl. memory address register),
- muistin puskurirekisteri MBR (engl. memory buffer register),

sekä ohjelmoijan käyttöön tarkoitetut rekisterit:

- yleisrekisterit,
- akkurekisterit,
- indeksirekisterit,
- pino-osoitin SP (engl. stack pointer) ja
- ympäristöosoitin FP (engl. frame pointer).

Ohjausyksikön ja muistinhallintayksikön rekisterit on varattu vain prosessorin sisäiseen käyttöön. Niihin ei voi viitata suoraan ohjelmista käsin.

Intelin 8086-80286-prosessoriperheiden ohjelmoitavat rekisterit

Kaikkien rekistereiden leveys on 16 bittiä.

Yleisrekistereitä AX, BX, CX ja DX voidaan käyttää myös 8:n bitin levyisinä nimillä AH , AL jne. (High ja Low). Tyypillinen käyttö: AX = akku, BX = kantarekisteri, CX = laskuri ja DX = data.

Osoitinrekisterit: pino-osoitin SP (stack pointer) osoittaa pinon huipulle, ympäristöosoitin BP (base pointer) osoittaa pinossa olevaan ja juuri suoritettavaan aliohjelmaan liittyvään tilavaraukseen.

Indeksirekisterit lähdeindeksi SI (source index) ja kohdeindeksi DI (destination index) liittyvät merkkijonojen ja taulukoiden käsittelyyn.

Ohjelman viitattavissa voi olla kerrallaan neljä 64 KB:n kokoista muistialuetta, joiden alkuun osoittaa segmenttirekisterit CS (code segment), DS (data segment), SS (stack segment) ja ES (extra segment).

Tila- ja ohjausrekisterit: Käskyosoitin IP (instruction pointer) osoittaa seuraavaksi suoritettavan käskyn, lippurekisteri FLAGS sisältää tietoa käskyn suorituksen tuloksesta.

Intelin 80386-Pentium-prosessoriperheiden ohjelmoitavat rekisterit

Uudemmissa prosessoreissä rekisterit ovat 32-bittisiä. Yleisrekisterit, osoitinrekisterit, indeksirekisterit sekä tila- ja ohjausrekisterit ovat muuten samat, mutta nimet ovat vastaavasti EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EIP ja EFLAGS. Yleisrekistereitä voi käsitellä myös 16-bittisinä kokonaisuuksina (EAH, EAL jne..)

Segmenttirekisterit ovat edelleen 16-bittiset CS, DS, SS, ES sekä uudet segmenttirekisterit FS ja GS.

Cray X-MP rekistereitä

Supertietokone CRAY X-MP:ssä on **kolme ensisijaista rekisterijoukkoa**: osoiterekisterit (A Address Registers) 8 kpl a' 32 bittiä, skalaarirekisterit (S Scalar Registers) 8 kpl a' 64 bittiä, vektorirekisterit (V Vector Registers) 8 kpl kussakin 64 alkiota a' 64 bittiä.

Tämän lisäksi CRAY X-MP:ssä on **kaksi toissijaista rekisterijoukkoa**, jotka toimivat puskureina edellisille: väliosoiterekisterit (B Intermediate Address Registers) 64 kpl a' 32 bittiä ja talletusrekisterit (T Scalar Save Registers) 64 kpl a' 64 bittiä. Toimintoyksiköt voivat suoraan käyttää vain A-, S- ja V-rekistereitä

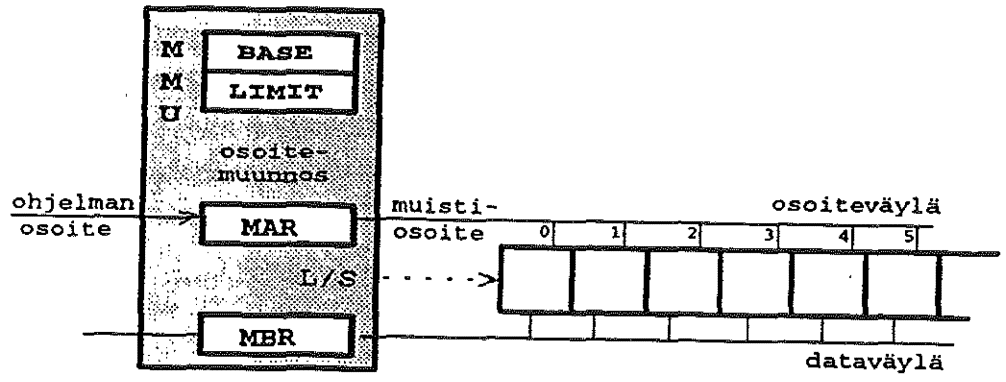
Käskyosoitin PC sisältää seuraavaksi suoritusvuorossa olevan käskyn osoitteen ohjelman alun suhteen. Kun ohjausyksikkö siirtää PC:n arvon muistiosoiterekisteriin MAR, muuttaa muistinhallintayksikkö MMU osoitteen fyysiseksi osoitteeksi ja laitteisto noutaa käskyn muistista käskyrekisteriin IR. Ohjausyksikkö tulkitsee käskyn ja tuottaa suorituksen vaatiman ohjauksen. Käskynoudon ja suorituksen yhteydessä prosessori tallettaa tietoja epäonnistumisista tilarekisteriin SR.

Suurten tietomäärien väliaikaistalletuksessa voi käyttää myös *pinoa* (engl. stack). Pinoa käytetään yleisesti aliohjelmarakenteen toteutuksessa paluuosoitteiden ja parametrien välittämiseen. Pinon toteutukseen käytetään tavallista keskusmuistia. Pinoon viittaaminen on nopeaa, koska huipun osoite pidetään tallessa rekisterissä SP ja operandin osoitteen siirtäminen muistiinviittausta varten MAR-rekisteriin on helppoa.

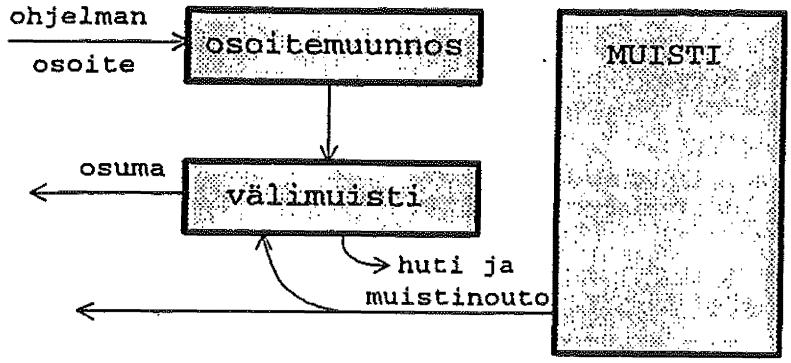
Muistinhallintayksikkö MMU

Muistinhallintayksikön kautta on liittymä prosessorin ulkopuoliseen väylään ja sitä kautta edelleen muistiin. Muistinhallintayksikkö tarkistaa jokaisen muistinoudon yhteydessä osoitteen kelvollisuuden ja tekee osoitemuunnoksen. Kukin prosessi käyttää omia sisäisiä osoitteita ja saa viitata vain omalle muistialueelleen. Prosessin sisäinen osoite on muutettava ennen muistille toimittamista todelliseksi fyysiseksi osoitteeksi.

Oletamme, että esimerkkikoneemme muistinhallintayksikössä on osoite- ja puskurirekistereiden MAR ja MBR lisäksi myös kaksi rajarekisteriä: BASE ja LIMIT. Kun suoritettava prosessi vaihtuu, asettaa käyttöjärjestelmä rekisterin BASE arvoksi ohjelman ensimmäisen käskyn muistiosoitteen ja rekisterin LIMIT arvoksi ohjelman pituuden. Osoitemuunnoksessa MMU tarkistaa, että MAR-rekisteriin tuotu ohjelman sisäinen osoite on pienempi kuin LIMIT. Sen jälkeen MMU lisää MAR-rekisteriin arvoon BASE-rekisterin arvon. Näin on saatu todellinen fyysinen muistiosoite, josta noudetaan tai jonne talletetaan.



Kuva 3.3: Muistinhallintayksikkö ja sen liittymät keskusmuistiin.



Kuva 3.4: Kun viitataan muistissa olevaan tietoon, tarkistetaan ensin, löytyykö viitatun muistipaikan sisältö välimuistista.

Nouto muistista: Sen muistipaikan osoite, jonka sisältö halutaan lukea, viedään MAR-rekisteriin. Tämän jälkeen MMU tekee suojauksen vaatiman tarkistuksen sekä suorittaa osoitemuunnoksen. Sen jälkeen MMU varaa väylän ohjausväylää käyttäen, siirtää muistiosoitteen osoiteväylälle ja antaa ohjausväylää käyttäen signaalin 'lataa' (engl. Load, L). Jonkin ajan kuluttua ko. muistipaikan sisältö on rekisterissä MBR, ja väylä voidaan vapauttaa.

Talletus muistiin: Sen muistipaikan osoite, jonne halutaan tallettaa, viedään MAR-rekisteriin. Kirjoitettava tieto viedään MBR-rekisteriin. MMU varaa väylän suojaustarkistuksen ja osoitemuunnoksen jälkeen. Saatu muistiosoite siirretään MAR-rekisteristä osoiteväylälle ja kirjoitettava tieto MBR-rekisteristä dataväylälle. Tämän jälkeen lähetetään ohjausväylää käyttäen signaali 'talleta' (engl. store, S), jonka johdosta tieto tallettuu haluttuun muistipaikkaan. Lopuksi vapautetaan väylä.

Nykyaikaisissa prosessoreissa on toteutettu virtuaalimuisti ja osoitemuunnos tapahtuu sivutaulujen ja / tai segmenttitaulujen avulla. Luvussa '8.3 Muistinhallinta' on esitetty eräs tapa toteuttaa virtuaalimuisti.

Välimuisti

Prosessorin on noudettava muistista kaikki käskyt ja myös käsiteltävä data, jollei se ole jo valmiiksi rekistereissä. Koska keskusmuisti on eri piirillä kuin prosessori ja ulkoisen väylän siirtonopeus on huomattavasti pienempi kuin prosessorin sisällä, on keskusmuistista nouto hidasta verrattuna prosessorin sisäisiin siirtoihin. Tämän vuoksi prosessorin yhteydessä on tavallisesti keskusmuistia nopeammalla teknologialla toteutettu välimuisti (engl. cache). Siellä pidetään viimeisimpien keskusmuistista noudettujen muistipaikkojen sisältöjä, ja usein myös niiden lähellä olleiden muistipaikkojen sisältöjä. Jos viitattavan muistipaikan sisältö on tässä välimuistissa, säästyy prosessori hitaammalta muistinoudolta (kuva 3.4).

Prosessorin sisäisen välimuistin lisäksi voi laitteistossa olla välimuistia myös prosessorin ulkopuolella. Esimerkiksi Pentium-prosessorissa on 8 kB välimuistia käskyjä varten ja 8 kB välimuistia dataa varten, ja emolevyllä vielä erikseen tavallisesti 256 kB ulkoista välimuistia.

3.3 Keskusmuisti

Muistiintallennetun ohjelman periaatteen mukaan keskusmuisti on ohjelman suoritusaikana sekä käskyjen että datan talletuspaikkana. Muistiin tieto on talletettu tavuina ja jokaisella tavulla (tai sanalla) on oma osoitteensa. Muistissa olevaan tietoon viitataan aina tällä muistiosoitteella.

Nykytekniikalla yhdelle muistilastulle saadaan mahtumaan 1 - 256 Mbittiä. Muistimoduulin (engl. Single In-line Memory Module, SIMM) koosta ja käytetyistä piireistä riippuen yhdelle muistimodulille on pakattu muistia 1 - 64 Mtavua.

Käyttömuisti (engl. Random Access Memory, RAM) tarkoittaa tavanomaista keskusmuistia, jonka sisältöä prosessi voi muuttaa. Käyttöjärjestelmä, sovellusprosessit ja niiden käsittelemät data-alueet sijaitsevat RAM-muistissa. Dynaaminen ja staattinen RAM ovat käyttäjän kannalta samanlaisia. Nimitykset liittyvät muistin tekniseen toteutukseen: dynaamista muistia täytyy aika-ajoin virkistää (laitteistotoiminto), kun taas staattisessa muistissa tieto säilyy muutenkin kunhan muistipiirit vain saavat virtaa. SRAM-muisti on nopeampaa kuin DRAM-muisti, mutta samalla myös kalliimpaa. Sitä käytetään mm. prosessorin ulkopuolisen välimuistin toteutukseen. RAM-muistin sisältö tyhjenee, kun laitteistosta katkaistaan virta.

Lukumuisti (engl. read only memory, ROM) tarkoittaa muistia, josta vain luetaan tietoa eikä sen sisältöä voi normaalisti muuttaa. Lisäksi lukumuistiin talletettu tieto säilyy myös virran katkaisun jälkeen. Ohjelmoitavien ROM-muistien (engl. programmable ROM, PROM) sisältö voidaan ladata kertaalleen erityisellä laitteella. Joidenkin ROM-muistien sisältöä voidaan myös myöhemmin muuttaa (engl. eraseable programmable ROM, EPROM). Lukumuistiin on talletettu mm. laitteiston käynnistykseen liittyviä rutiineja, ohjauksessa tarvittavat mikrokoodit sekä eräitä käyttöjärjestelmään kuuluvia osia.

Laitteistossa on usein myös paristovarmennettuja CMOS-muistipiirejä (engl. Complementary Metal Oxide Semiconductor), joita käytetään normaalin lukumuistin tavoin, mutta joiden sisältöä voi muuttaa ilman erityislaitteistoa. Esimerkiksi mikrotietokoneissa tällaiseen muistiin on talletettu tietoa laitteistokokoonpanosta mm. levyn tyyppi sekä päiväys ja kello.

3.4 Ohjaimet

Oheislaitteet kytketään prosessorin ja keskusmuistin yhteyteen *ohjaimien* välityksellä. Oheislaitteita tarvitaan tiedon syöttöä, tulostusta, pysyvää talletusta ja tiedonsiir-

toa varten. Kutakin laitetyyppiä kohden on oma ohjaimensa ja yksi ohjain voi ohjata yhtä tai useampaa laitetta. Ohjain voi olla integroitu emolevylle, kuten esimerkiksi näppäimistön ohjain, tai ohjain voi olla erillisellä lisäkortilla, joka liitetään väyläliitäntään. Esimerkkejä ohjaimista ovat näytönohjain, levyohjain (tavallisesti muutamia levyjä per ohjain), tietoliikenneohjaimet ja laiteväyläohjaimet. Väyläohjaimen avulla prosessoriin voidaan liittää lisäkorttiväylä, jonka liittimiin voidaan edelleen asentaa muita ohjainkortteja.

Prosessi ei itse keskustele ohjaimien kanssa, vaan se pyytää apua käyttöjärjestelmältä. Käskykantaan kuuluu tätä varten erityinen *palvelupyyntökäsky*. Yksinkertaisin tapa järjestää siirto oheislaitteelta keskusmuistiin on tehdä se täysin käyttöjärjestelmän ohjauksen alaisena. Tämä sitoo kuitenkin prosessorin hitaaseen siirräntään. Onneksi monet ohjaimet pystyvät siirtämään tietoa itsenäisesti *DMA-siirtona* (engl. Direct Memory Access) laitteen ja keskusmuistin välillä. Kun käyttöjärjestelmä on käynnistänyt siirron, se voi laittaa prosessorin suorittamaan siirron ajaksi jotain toista prosessia. Prosessin suoritus ja siirto tapahtuvat siten aidosti rinnakkain. Tämä aiheuttaa kilpailua keskusmuistiin menevän väylän käytöstä. Kun siirto on valmis, ilmoittaa ohjain siitä prosessorille *keskeytyksellä*.

Koska kaikki väylään liitetyt ohjaimet *kilpailevat* väylän käytöstä, rajoittaa väylän siirtonopeus väylään liitettävien laitteiden määrää. Tästä syystä suurissa järjestelmissä käytetään järjestelyä, jossa laiteohjaimet ja oheislaitteet on liitetty keskusyksikköön erillisen I/O-prosessorin välityksellä. I/O-prosessorilla on omaa muistia ja oma ohjelma suoritettavanaan. Sen ainoana tehtävänä on ohjata oheislaitteiden toimintaa. I/O-prosessorilla voi olla myös erillinen väylä keskusmuistiin.

Myös PC-mikrotietokoneen väylään voidaan liittää erillinen SCSI-laiteväylä (engl. Small Computer System Interface). Sen liittimiin sitten kytketään ohjaimien kautta oheislaitteita: kiintolevyjä, nauha-asemia, CD-ROM-asemia, kuvanlukijoita jne.

Kaikista mikro- ja pientietokoneista löytyy I/O-ohjainkortilta sekä sarja- (RS-232-liitäntä) että rinnakkaisliitäntä (Centronics-liitäntä). Näihin voidaan kytkeä liittimellä ulkoiset laitteet, esim. modeemi sarjaliitäntään ja kirjoitin rinnakkaisliitäntään. PCMCIA (engl. PC Memory Card International Assosiation) on kehittänyt oheislaitteiden liittämistä helpottavan yleiskäyttöisen PC-Card standardin ja liitännän, jonka kautta myös suurta nopeutta vaativat laitteet on helppo liittää suoraan laiteväylälle. PC-Card on suunnilleen luottokortin kokoinen liitäntäkortti, esimerkiksi modeemi / faksi, verkkosovitin, kiintolevy tms. Tämä liitäntä on yleistynyt erityisesti kannettavien mikrotietokoneiden yhteydessä.

3.5 Käskyjen suoritus

Prosessori suorittaa tietojen käsittelyn sille annettujen käskysarjojen eli ohjelmien mukaisesti. Koneen tunnistamat perusoperaatiot on määritelty tietokoneen käskykannassa. Käskyjä laitteiston ohjaussignaaleiksi dekoodattaessa on osattava erottaa operaatiokoodi, operandit sekä niiden tyyppi ja sijainti.

Käskyt ja niiden käsittelemä data ovat keskusmuistissa. Prosessori suorittaa käskyt pääsääntöisesti peräkkäin niiden esiintymisjärjestyksessä. Suoritusjärjestys voi muuttua, kun suoritettavaksi tulee hyppykäsky. Käskyjen normaali suoritus saattaa katketa myös laitteiston tai ohjelmiston aiheuttaman keskeytyksen vuoksi. Tällöin prosessori siirtyy suorittamaan käyttöjärjestelmää, ja voi tehdä tilanteen vaatimat toimenpiteet. Kun keskeytys on käsitelty, voi prosessori jatkaa suoritusta tarvittaessa keskeytyneestä kohdasta, tai käyttöjärjestelmä vaihtaa suoritettavaksi uuden prosessin.

Yhden käskyn suoritus muodostuu useista vaiheista. Vanhemmissa prosessoreissa tarvittiin jopa 10 kellojaksoa kutakin käskyä kohden. Uudemmat prosessorit pystyvät lisääntyneen rinnakkaisuuden ansiosta suorittamaan 1 - 4 käskyä per kellojakso. Yhden konekielisen käskyn aikana prosessori suorittaa seuraavia toimintoja:

1. Ohjausyksikkö CU siirtää käskyn osoitteen muistinhallintayksikön MMU muistiosoiterekisteriin (MAR := PC) ja valmistautuu uuden käskyn noutamiseen kasvattamalla käskyosoittimen arvoa (PC := PC + käskynpituus).

2. MMU tarkistaa osoitteen oikeellisuuden ja tekee osoitemuunnoksen sekä käynnistää muistinoudon.

3. MMU ottaa käskyn väylältä vastaan (MBR := MEM[MAR]) ja välittää sen edelleen ohjausyksikölle (IR := MEM[MAR]), joka tulkitsee sen.

4. Ohjausyksikkö CU selvittää käskyn operaatio-osan perusteella käskyssä tarvittavien operandien osoitteet ja noutaa datan muistista tai rekistereistä ALU:un.

5. Ohjausyksikkö CU suorituttaa käskykoodin määräämät operaatiot tavallisimmin ALU:ssa ja tallettaa tulokset esimerkiksi rekistereihin.

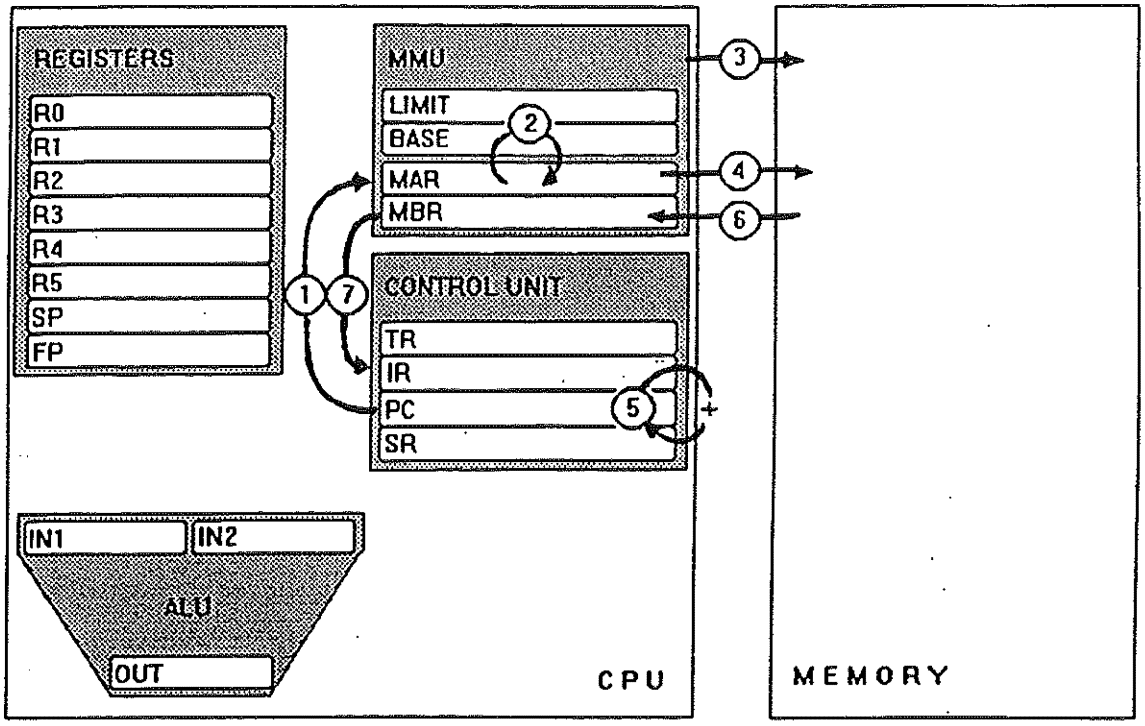
6. Ohjausyksikkö CU tarkistaa tilarekisteristä SR oliko käskyn suorituksessa poikkeuksia tai aiheuttivatko prosessorin ulkopuoliset ohjaimet keskeytyksiä. Jos käskyn suoritusaikana sattui poikkeus tai keskeytys, siirtyy prosessori suorittamaan käyttöjärjestelmään kuuluvaa koodia.

Kaikkiin edellä esitettyihin vaiheisiin liittyy laitteiston eri osien välistä tilakoodi-, ohjaus- ja ajoitussignaalien välitystä. Käskyn suoritusaikana tallettuu tietoa epäonnistumisista tilarekisteriin SR.

Käskyn nouto

Kuvassa 3.5 on esitetty käskynouto. Käskyn noutovaihe on aina kaikille käskyille samanlainen: Ohjausyksikkö tietää, että seuraavaksi suoritettavan käskyn osoite on rekisterissä PC.

Ohjausyksikön lähettämät signaalit siirtävät käskyosoittimen PC arvon muistiosoiterekisteriin MAR (1), ja MMU tekee osoitemuunnoksen (2). Ohjaus 'Lataa' (3)



Kuva 3.5: Käskyn nouto

käynnistää muistinoudon ja fyysinen osoite siirretään osoiteväylälle (4). Hetken kuluttua osoitetun muistipaikan sisältö on muistin puskurirekisterissä MBR (6). Sieltä se siirretään edelleen käskyrekisteriin IR (7).

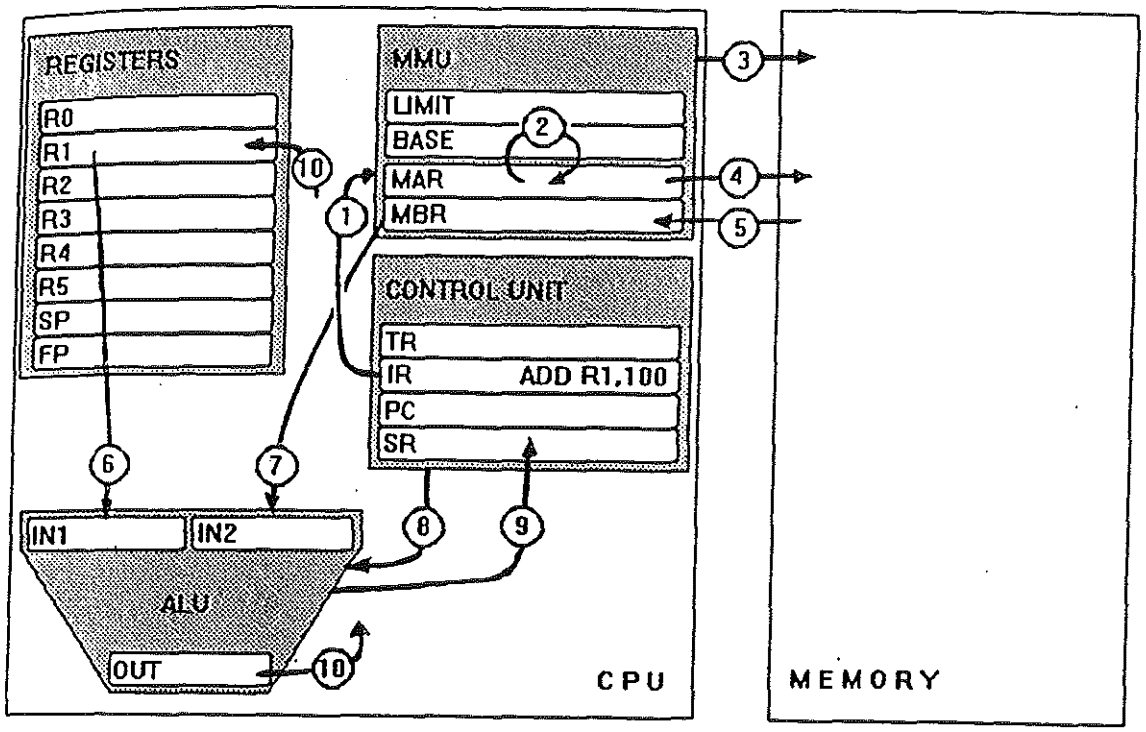
Seuraavan käskyn noutoon prosessori valmistautuu kasvattamalla käskyosoittimen PC arvoa (5). Tavallisesti tätä varten on omaa laitteistoa.

Käskyn suoritus

Kuvassa 3.6 on esitetty esimerkkinä käskyn ADD R1,100 suoritusta. Käskynoudon jälkeen käsky on käskyrekisterissä IR. Käskykoodin perusteella ohjausyksikkö löytää mikrokoodista käskyn suorittamisessa tarvittavat ohjaustiedot. Tässä tapauksessa käsky on yhteenlaskukäsky ja sen operandit ovat rekisterissä R1 ja osoitteessa 100. Tulos on talletettava rekisteriin R1.

Laskutoimituksen tekemiseksi on operandit siirrettävä ALU:un. Koska jälkimmäinen operandi on muistissa, on tehtävä muistinouto. Ohjausyksikön CU lähettämien signaalien ansiosta osoite 100 siirtyy käskyrekisteristä IR (esim. käskyn kaksi viimeistä tavua) muistiosoiterekisteriin MAR (1). Muistinhallintayksikkö MMU tarkistaa osoitteen oikeellisuuden ja tekee osoitemuunnoksen (2).

Ohjaus 'Lataa' (3) käynnistää muistinoudon ja fyysinen osoite siirretään osoiteväylälle (4). Hetken kuluttua osoitetun muistipaikan sisältö on muistin puskurire-



Kuva 3.6: Käskyn ADD R1, 100 suoritus.

kisterissä MBR (5).

Nyt käskyn molemmat operandit ovat prosessorin sisällä ja käsky voidaan suorittaa. Käskyn ensimmäinen operandi siirretään R1:stä ALU:n ensimmäiseen sisäänmenoon (6), ja käskyn jälkimmäinen operandi siirretään MBR-rekisteristä ALU:n toiseen sisäänmenoon (7). Ohjauksen 'yhteenlaske' vaikutuksesta ALU laskee saamansa operandit yhteen (8), ja ohjaa tuloksen ALU:n ulostulosta edelleen rekisteriin R1 (10). Jos käskyn laskutoimituksessa sattuu yli- tai alivuoto, merkitään siitä tieto tilarekisteriin SR (9).

Jos tilarekisteriin ei nostettu poikkeus- tai keskeytysbittejä pystyyn, jatkaa prosessori noutamalla muistista prosessin seuraavan käskyn. Käskynoudon yhteydessähän kasvatettiin käskylaskurin PC arvoksi seuraavan käskyn osoite. Jos käsky olisi ollut hyppykäsky, olisi sen suoritusvaihe asettanut PC:lle kokonaan uuden arvon, mikä aiheuttaisi siirtymiseen eteen- tai taaksepäin koodissa.

Prosessin vaihto

Kun prosessori käskysyklin lopussa huomaa tilarekisteristä SR poikkeustilanteen tai keskeytyksen, siirtää laitteisto prosessorin suorittamaan käyttöjärjestelmän koodia. Tällöin käyttöjärjestelmällä on mahdollisuus lopettaa virheellisesti toimivan prosessin suoritus, tehdä prosessin pyytämiä palveluja tai vaihtaa suoritettavaa prosessia

Kun käyttöjärjestelmä vaihtaa suoritettavaa prosessia, on suorituksessa olleen prosessin rekistereihin tekemät muutokset talletettava. Seuraavaksi suoritukseen tuleva prosessihan tallettaa rekistereihin taas uusia arvoja. Tätä varten käyttöjärjestelmällä on jokaisen prosessin ikiomassa prosessinkuvaajassa, PCB.ssä, tallealue rekistereiden vanhoille arvoille. Prosessia vaihtaessaan käyttöjärjestelmä kopioi väistyvän prosessin rekistereiden arvot prosessorista ko. prosessin tallealueelle.

Käyttöjärjestelmä saa prosessorin suorittamaan seuraavaa prosessia kopioimalla rekistereille uudet arvot suoritukseen valittavan prosessin prosessinkuvaajasta. Suojausta ja muistinhallintaa varten käyttöjärjestelmä asettaa vielä muistinhallinnan tarvitsemat arvot MMU:hun (meillä BASE-rekisteriin arvoksi ohjelman ensimmäisen käskyn osoitte ja LIMIT-rekisteriin arvoksi ohjelman pituus). Koska rekistereiden arvojen vaihdon yhteydessä kopioituu myös käskyosoittimen PC vanha arvo, jatkaa suoritettavaksi valittu prosessi siitä mihin se edellisellä kerralla jäi. Ja koska muistinhallintayksikön MMU rekistereillä on nyt uudet arvot, tapahtuvat muistiviittaukset nyt tämän prosessin alueelle.

Luku 4

TTK-91-symbolinen konekieli

Hierarkkisen koneen mallissa esiintyvän tason 'perinteinen konetaso' nimitys juontaa juurensa ensimmäisistä tietokoneista, sillä tämä taso kehitettiin ensimmäisenä laitteiston päälle. Ohjelmointi tapahtui esittämällä ohjelmat suoraan tietokoneen konekielellä bittijonoina. Ohjelmien teko helpottui, kun konekielitason "päälle" kehitettiin symbolinen konekieli. Symbolisessa konekielessä bittijonon eri osat, käskykoodi ja operandit, kuvataan luettavammassa muodossa kirjainsymbolein. Jotta kirjoitettu ohjelma voitaisiin suorittaa, täytyy se kääntää todelliselle konekielelle aivan kuten lausekieletkin.

Tutustumme nyt symboliseen konekieleen realistisen TTK-91 esimerkkikoneemme avulla. Kone on tyypillinen yleisrekisteriarkkitehtuuri, joskin olemme tehneet joitakin yksinkertaistuksia. Käytämme mm. vain kokonaislukuaritmetiikkaa ja sanaosoitteita. Emme puutu toistaiseksi siirräntään, vaan tarkastelemme koneemme toimintaa vain prosessorin ja keskusmuistin osalta.

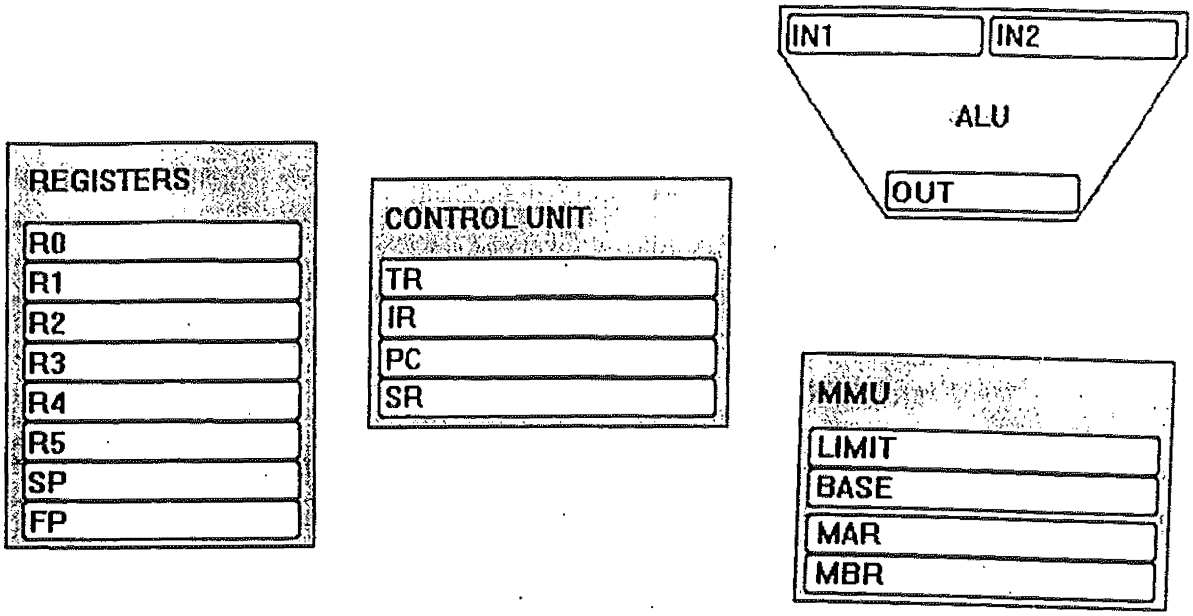
4.1 Tietokoneen TTK-91 rakenne

TTK-91:n keskusyksikön osat ovat väylällä toisiinsa yhdistetyt prosessori CPU, keskusmuisti MEM ja laiteohjaimet. Prosessoriin kuuluu ohjausyksikkö CU, aritmeettis-looginen yksikkö ALU, muistinhallintayksikkö MMU sekä rekisterit.

Ohjelmoitavat rekisterit

Koneessamme on kahdeksan ohjelmoitavaa yleisrekisteriä:

- rekisteri R0 on työrekisteri,
- rekisterit R1-R5 ovat työ- ja indeksirekistereitä,
- rekisteri SP (eli R6) on pino-osoitin (engl. stack pointer) ja
- rekisteri FP (eli R7) on ympäristöosoitin (engl. frame pointer).



Kuva 4.1: Rekisterit, ohjausyksikkö, aritmeettis-looginen yksikkö ja muistinhallintayksikkö.

Rekisteriä R0 ei voi käyttää normaalisti indeksointiin. Vaikka se olisi mainittu indeksirekisterinä (ts. käskyn jälkimmäisenä operandina), sitä ei käytetä eikä sen sisältö vaikuta laskettavaan osoitteeseen. Myös rekistereitä SP ja FP voi käyttää harkiten työ- ja indeksirekistereinä, vaikka niille onkin varattu oma erityistarkoituksensa. Palaamme rekistereiden SP ja FP käyttöön tarkemmin aliohjelmatoteutuksen yhteydessä.

Ohjausyksikkö

Ohjausyksikköön CU kuuluu 4 ei-ohjelmoitavaa laiterekisteriä:

- *Käskyosoittimessa* PC on seuraavaksi suoritettavan käskyn osoite ohjelman alun suhteen.
- Prosessori noutaa käskyn muistista *käskyrekisteriin* IR, jossa se tulkitaan.
- *Tilarekisteriin* SR tallettuu mm. vertailujen tulokset, tietoja käskyn suorituksessa sattuneista poikkeuksista sekä tieto ulkopuolisen laitteen keskeytyspyynnöstä. SR:ssa on ainakin bitit:

vertailu:

G (engl. greater) asettuu, kun vertailussa ensimmäinen operandi on suurempi kuin jälkimmäinen operandi

E (engl. equal) asettuu, kun vertailussa operandit ovat yhtäsuuria

L (engl. less) asettuu, kun vertailussa ensimmäinen operandi on pienempi kuin jälkimmäinen operandi

poikkeus:

O (engl. overflow) asettuu, kun ALU:ssa tapahtuu luvun yli- tai alivuoto

Z (engl. division by zero) asettuu, kun ALU:ssa jaetaan nollalla

U (engl. unknown instruction) asettuu, kun CU ei tunnista käskyä

M (engl. forbidden memory address) asettuu, kun MMU toteaa, että yritetään viitata kielletylle muistialueelle

keskeytys:

I (engl. interrupt) asettuu, kun prosessorin ulkopuolinen laiteohjain aiheuttaa keskeytyksen

S (engl. supervisor call) asettuu, kun ohjelma tekee palvelupyynnön käyttöjärjestelmälle SVC-käskyllä

P (engl. privileged mode, etuoikeutettu tila) asettuu, kun prosessori siirtyy suorittamaan käyttöjärjestelmään kuuluvaa koodia

D (engl. interrupts disabled, keskeytykset estetty) asettuu, kun käyttöjärjestelmä suorittaa ns. kriittistä toimintoa

- *Apurekisteriä* TR käytetään käskyn suorituksen aikana tilapäiseen säilytykseen. Sille on käyttöä mm. osoitelaskennassa.

Aritmeettis-looginen yksikkö

Aritmeettis-looginen yksikkö ALU suorittaa laskutoimitukset, vertailut, bittioperaatiot ja sivuttaissiirrot. Ohjausyksikkö vie operandit ALU:n sisäänmenoihin (ALUin1, ALUin2) ja ALU:un annetaan tieto suoritettavasta operaatiosta. Tulos viedään ALU:n ulostulosta (ALUout) käskyssä mainittuun rekisteriin. Vertailun tulos ja merkintä laskutoimituksessa tapahtuneesta virheestä (esim. nollallajako) viedään kuitenkin aina tilarekisteriin SR.

Muistinhallintayksikkö

Muistinhallintayksiköstä MMU on liittymä väylälle ja sitä kautta myös keskusmuistiin. Muistiin viitattaessa toimitetaan ohjelman sisäinen osoite muistiosoiterekisteriin MAR ja muistiin talletettava tieto muistin puskurirekisteriin MBR. Muistinhallintayksikkö muuttaa ohjelman sisäisen osoitteen fyysiseksi muistiosoitteeksi kantarekisteriä BASE ja rajarekisteriä LIMIT apuna käyttäen.

Keskusmuisti

Koneen keskusmuisti koostuu 32 bittiä leveistä muistipaikoista, sanoista ja muistiosoitteet kulkevat siis sanan välein. Myös kaikki koneen rekisterit ovat 32-bittisiä. Käskyjen toimintaa kuvattaessa viittaamme muistipaikan sisältöön käyttäen merkintää MEM[x], missä x on muistiosoite ($0 \leq x \leq 2^{32} - 1$).

4.2 Käskyrakenne

TTK-91 on *rekisteriarkkitehtuuri*, mikä tarkoittaa sitä, että käskyillä voidaan osoittaa eksplisiittisesti joukko rekistereitä. Lisäksi kaksioperandisissa käskyissä ensimmäisenä operandina on aina rekisteri ja tulokset sijoitetaan aina rekisteriin, joko käskyn ensimmäisenä operandina olevaan rekisteriin tai tilarekisteriin.

Yksinkertaisuuden vuoksi otamme käskykantaan vain vakiomittaisia käskyjä: käskyn pituus on aina neljä tavua eli 32 bittiä.

Esitys bittitasolla

Esitämme käskyt kokonaisuudessaan bittitasolla seuraavassa muodossa:

Varaamme käskykoodille ensimmäisen tavun 8 bittiä (bitit 31..24). Käskykoodin neljä ensimmäistä bittiä (bitit 31..28) määräävät käskyn pääluokan ja neljä seuraavaa bittiä (bitit 27..24) ilmaisevat varsinaisen operaation.

Käskyn ensimmäinen operandi on aina rekisterissä. Koska rekistereitä on 8, riittää sen ilmaisemiseen 3 bittiä (kenttä Rj, bitit 21..23). Useimpien käskyjen tulos viedään tässä kentässä kerrottuun rekisteriin.

Kenttä Ri (bitit 16..18) ilmaisee käskyssä mahdollisesti käytettävän toisen rekisterin, indeksirekisterin.

Käskyn kaksi viimeistä tavua (kenttä ADDR) varaamme käskyn 2. operandin osoittamiselle. Kentän sisältö voidaan tulkita joko suoraan operandin arvoksi tai operandin osoitteeksi. Tulkinta määräytyy kentän M perusteella. Suurin kentässä ADDR esitettävissä oleva luku on $2^{15} - 1 = 32767$ ja pienin $-2^{15} = -32768$.

Kenttä M (bitit 19..20) ilmaisee, kuinka monta muistinoutoa tarvitaan 2. operandin (arvo tai kohdeosoite) määräämiseksi:

- M = 00 ei muistinoutoa
  - operandi on käskyn osana kentässä ADDR tai
  - operandi on rekisterissä Ri

- M = 01 yksi muistinouto
  - operandi on ADDR-kentässä kerrotussa osoitteessa tai
  - operandi on rekisterissä Ri kerrotussa osoitteessa
- M = 10 kaksi muistinoutoa
  - operandin osoite on ADDR-kentässä kerrotussa osoitteessa, nouda ensin osoite ja sitten ko. osoitteessa oleva arvo

Esitys symbolisessa konekielessä

Esitämme käskyt symbolisessa konekielessä muodossa

```
viite  OPER Rj,M ADDR(Ri)
```

missä

- OPER käskyn symbolinen nimi;
- Rj ensimmäinen operandi (rekisteri R0..R7,SP,FP) ;
- M 2. operandin osoitusmuoto (eli muistinoutojen lkm); se voi olla jokin seuraavista:

  = välitön operandi,

  suora osoitus (tyhjä, ei siis merkitä),

  @ epäsuora osoitus;
- ADDR osoiteosa (muistiosoite tai vakio),
- Ri mahdollinen indeksirekisteri (rekisteri R0..R7,SP,FP).

Käskyn eteen voi laittaa viitteeksi symbolisen osoitteen, joka ei saa alkaa numerolla. Symboliselle viitteelle on käyttöä erityisesti hyppykäskyjen kohteen ilmaisemisessa.

Jos käskyn jollain osalla ei ole erityistä merkitystä, sen voi jättää merkitsemättä. Siten esimerkiksi käsky ADD R2,R3 tarkoittaa samaa kuin käsky ADD R2,=0(R3). Vastaavasti käsky LOAD R5,100(R0) merkitään lyhyemmin muodossa LOAD R5,100. Rekisteri R0:n sisältöhän tulkitaan 0:ksi, jos se esiintyy indeksirekisterin paikalla.

Lähes kaikille käskyille on käytettävissä seuraavat muodot:

```
OPER Rj,ADDR          suora muistiosoitus
OPER Rj,Ri            suora rekisteriosoitus
OPER Rj,=ADDR         välitön operandi
OPER Rj,@ADDR         epäsuora muistiosoitus
OPER Rj,@Ri           epäsuora rekisteriosoitus
OPER Rj,ADDR(Ri)      indeksoitu osoitus
OPER Rj,=ADDR(Ri)     indeksoitu välitön osoitus
OPER Rj,@ADDR(Ri)     indeksoitu epäsuora muistiosoitus
```

Poikkeuksia tästä ovat:

- CALL ja hyppykäskyt, joissa 2. operandina ei voi olla välitön operandi. Käskyt muuttavat PC:n arvoa, joten jälkimmäisen operandin on aina oltava kohdeosoite.
- STORE-käsky, jossa jälkimmäisenä operandina ei voi olla välitön operandi tai rekisteri. Käsky tallettaa tuloksensa muistiin, joten 2. operandin on aina oltava kohdeosoite.
- POP-käskyn jälkimmäisenä operandina tulee aina olla kohderekisteri.
- NOP-käskyssä ei jälkimmäisellä operandilla ole merkitystä.

4.3 Muistiosoitukset

Muistiosoitusten ymmärtämisen helpottamiseksi seuraavassa käydään läpi eri osoitukset esimerkkien avulla. Tarkastellaan käskyä LOAD, jonka perusmuoto siirtää muistipaikan sisällön rekisteriin. Käskyä voidaan käyttää seuraavin tavoin:

1. Suora muistiosoitus:

   ```
   LOAD R1, 100
   ```

   Muistipaikan 100 sisältö viedään rekisteriin R1.

2. Suora rekisteriosoitus:

   ```
   LOAD R1, R2
   ```

   Rekisterin R2 sisältö viedään rekisteriin R1.

3. Välitön operandi:

   ```
   LOAD R1, =100
   ```

   Luku 100 viedään rekisteriin R1.

4. Epäsuora muistiosoitus:

```
LOAD R1, @100
```

Rekisteriin R1 viedään muistipaikassa 100 ilmaistun muistipaikan sisältö. Esimerkiksi

```
R1: ?        M[100]: 200          M[200]: 10

            LOAD R1, @100

R1: 10
```

5. Epäsuora rekisteriosoitus:

```
LOAD R1, @R2
```

Samoin kuin edellisessä kohdassa, mutta muistipaikan tilalla on rekisteri.

6. Indeksoitu osoitus:

```
LOAD R1, 100(R2)
```

R1:een viedään sen muistipaikan sisältö, jonka osoite saadaan laskemalla yhteen 100 ja R2:n sisältö. Esimerkiksi

```
R1: ?      R2: 10      M[100]: 200      M[110]: 150

          LOAD R1, 100(R2)

R1: 150
```

7. Indeksoitu välitön osoitus:

```
LOAD R1, =100(R2)
```

R1:een viedään luku, joka saadaan laskemalla yhteen 100 ja R2:n sisältö. Edellisen esimerkin tilanteessa R1:een tulee 110.

8. Indeksoitu epäsuora muistiosoitus:

```
LOAD R1, @100(R2)
```

R1:een viedään sen muistipaikan sisältö, jonka osoite on muistipaikassa 100+R:n sisältö. Esimerkiksi

```
R1: ?     R2: 10    M[100]: 200     M[110]: 150      M[150]: 300

          LOAD R1, @100(R2)

R1: 300
```

4.4 Yhteenveto TTK-91 käskyistä

Symboliseen konekieleemme kuuluu tiedonsiirtokäskyjä, aritmeettis-loogisia käskyjä, hyppykäskyt (ehdoton ja ehdollisia), palvelupyyntökäsky, aliohjelmakäskyt sekä pinon käsittelykäskyt. Seuraavassa luettelossa on selitetty lyhyesti kunkin käskyn toiminta.

Tiedonsiirtokäskyt

LOAD vie 2. operandin arvon rekisterin Rj arvoksi.

STORE tallettaa rekisterissä Rj olevan kokonaisluvun 2. operandin arvoksi.

IN lukee 2. operandina kerrotulta laitteelta kokonaisluvun rekisteriin Rj. (esim. näppäimistöltä lukeminen IN R1,=KBD)

OUT tulostaa rekisterissä Rj olevan kokonaisluvun 2. operandina kerrotulle laitteelle. (esim. näytölle tulostus OUT R1,=CRT)

Aritmeettiset ja loogiset käskyt

ADD lisää rekisterissä Rj olevaan lukuun 2. operandin arvon.

SUB (engl. subtsract) vähentää rekisterissä Rj olevasta luvusta 2. operandin arvon.

MUL (engl. multiply) kertoo rekisterissä Rj olevan luvun 2. operandin arvolla.

DIV (engl. divide) jakaa rekisterissä Rj olevan luvun 2. operandin arvolla. Jakolaskussa vain kokonaisosalla on merkitystä.

MOD (engl. modulo) jakaa rekisterissä Rj olevan luvun 2. operandin arvolla. Jakolaskussa vain jakojäännösosalla on merkitystä.

NOT (engl. bitwise NOT) bittitason EI-operaatio. Jos sana on yksinkertaisuuden vuoksi 8 bittiä, niin käsky tekee seuraavaa:

```
NOT 10101010  =  01010101
```

AND (engl. bitwise AND) bittitason JA-operaatio. Esimerkiksi:

```
        11001100
AND     10101010
        --------
        10001000
```

OR (engl. bitwise OR) bittitason TAI-operaatio. Esimerkiksi

```
        11001100
OR      10101010
        --------
        11101110
```

XOR (engl. bitwise XOR) bittitason poissulkeva TAI-operaatio. Esimerkiksi

```
        11001100
XOR     10101010
        --------
        01100110
```

SHL (engl. shift left) siirtää rekisterin Rj sisältämiä bittejä vasemmalle 2. operandin ilmoittaman määrän. Täyttää oikeaa päätä 0-biteillä.

SHR (engl. shift right) kuten SHL, mutta siirtää oikealle. Täyttää vasenta päätä 0-biteillä.

COMP (engl. compare) vertaa ensimmäisen operandin arvoa 2. operandin arvoon ja asettaa vertailun tuloksen tilarekisterin bitteihin L=pienempi, E=yhtäsuuri, G=suurempi

Haarautumiskäskyt

JUMP (engl. unconditional jump) ehdoton hyppy 2. operandina olevaan osoitteeseen. ts. PC := 2. operandin arvo. Ensimmäisellä operandilla ei ole merkitystä.

Ehdollinen haarautuminen (engl. conditional jump) rekisterin Rj sisällön perusteella 2. operandina kerrottuun osoitteeseen, ts. PC := 2. operandina oleva osoite. Käsky sisältää vertailun nollaan ja asettaa tilarekisterin SR. Ensimmäisellä operandilla ei ole merkitystä. Merkitään c(Ri):llä rekisterin Ri sisältöä.

JNEG (engl. jump if negative) jos c(Rj) < 0, niin hyppää

JZER (engl. jump if zero) jos c(Rj) $= 0$, niin hyppää

JPOS (engl. jump if positive) jos c(Rj) > 0, niin hyppää

JNNEG (engl. jump if not negative) jos c(Rj) ≥ 0, niin hyppää

JNZER (engl. jump if not zero) jos c(Rj) $\neq 0$, niin hyppää

JNPOS (engl. jump if not positive) jos c(Rj) ≤ 0, niin hyppää

Ehdollinen haarautuminen tilarekisterin SR sisällön perusteella 2. operandina kerrottuun osoitteeseen (käyttö COMP-käskyn yhteydessä). Ensimmäisellä operandilla ei ole merkitystä.

JLES (engl. jump if less) jos bitti L asetettu, niin hyppää

JEQU (engl. jump if equal) jos bitti E asetettu, niin hyppää

JGRE (engl. jump if greater) jos bitti G asetettu, niin hyppää

JNLES (engl. jump if not less) jos bitti E tai G asetettu, niin hyppää

JNEQU (engl. jump if not equal) jos bitti L tai G asetettu, niin hyppää

JNGRE (engl jump if not greater) jos bitti L tai E asetettu, niin hyppää

Pinokäskyt

Rekisterin Rj (pino-osoitin) arvona on pinon päällimmäisen alkion osoite. Pino-osoittimena käytetään tavallisesti rekisteriä SP (eli R6), mutta mikä tahansa muukin rekisteri kelpaa. Tällöin ohjelmoijan on itse huolehdittava 'sopivan' alkuarvon asettamisesta.

PUSH Kasvattaa pino-osoittimen SP arvoa yhdellä. Vie 2. operandin muistiin pinon päällimmäiseksi alkioksi.

POP Poistaa pinon päällimmäisen alkion ja asettaa sen 2. operandina annetun rekisterin Ri arvoksi. Vähentää pino-osoittimen SP arvoa yhdellä. 2. operandin tulee aina olla rekisteri.

Aliohjelmakäskyt

CALL (engl. call procedure) aliohjelmakutsu. Ensimmäisenä operandina pino-osoitin SP ja 2. operandina aliohjelman alkuosoite. Tallettaa rekistereiden PC ja FP (paluuosoite, ympäristöosoitin) arvot pinoon. Asettaa FP:n arvoksi pino-osoittimen uuden arvon ja PC:n arvoksi 2. operandin arvon.

EXIT (engl. exit procedure) paluu aliohjelmasta kutsua seuraavaan käskyyn. Ensimmäisenä operandina pino-osoitin SP ja 2. operandina kutsua ennen pinoon vietyjen parametrien lukumäärä. Palauttaa ympäristöosoittimen vanhan arvon (FP) ja paluuosoitteen (PC) pinosta. Vähentää pino-osoittimen arvoa 2. operandin arvon verran.

Palvelupyyntö

SVC (engl. supervisor call) käyttöjärjestelmän palvelurutiinin kutsu (systeemikutsu, joka aiheuttaa keskeytyksen). Ensimmäisenä operandina pinon huippu SP ja 2. operandina palvelun numero. Palvelun tarvitsemat parametritiedot vietävä pinoon ennen palvelurutiinin kutsua.

Muut käskyt

NOP (engl. no operation) ei toimintaa. Operandeilla ei merkitystä.

4.5 Muuttujien tilanvaraus

Kääntäjän ohjauskäskyillä, eli ns. valekäskyillä, annetaan ohjeita kääntäjälle globaaleja muuttujia varten tarvittavista tilanvarauksista, sekä tunnusten ja arvojen samaistuksista. Näiden ohjauskäskyjen ansiosta voimme jättää symbolisessa konekielessä muuttujien 'numeeriset' talletusosoitteet kääntäjän murheeksi. Käytämme siis käskyissä numeeristen osoitteiden paikalla aina symbolisia tunnuksia. Kääntäjän ohjauskäskyt eivät ole konekielen käskyjä, joten niistä ei synny konekielistä koodia.

TTK-91:ssä varataan globaaleille muuttujille tilat välittömästi käännetyn koodin perästä. Tätä varausta kutsutaan staattiseksi tilanvaraukseksi, koska se on olemassa koko ohjelman suorituksen ajan.

Staattisen alueen jälkeinen ohjelman käytössä oleva muistitila jaetaan vielä kahteen osaan, *pinoksi*, ja *keoksi*. Pinoa käytetään ainakin aliohjelmakutsussa mm. parametrien välityksessä ja väliaikaisen tiedon tallettamisessa. Dynaamisille, esimerkiksi Javan metodin New avulla luoduille rakenteille varataan tilaa puolestaan keosta. Pinon käyttöön tutustumme yksityiskohtaisesti aliohjelmien yhteydessä. Pinosta ja keosta tehtäviä varauksia kutsutaan *dynaamiseksi tilanvaraukseksi*, koska varauksen koko vaihtelee ohjelman suoritusvaiheiden mukaisesti.

Ohjelman työtilana käyttämä pino alkaa toteutuksessamme välittömästi staattisen tilanvarauksen jälkeen. Keko alkaa data-alueen lopusta ja se kasvaa kohti pinoa. Tutustumme tällä kurssilla vain pinon käyttöön.

Kääntäjän ohjauskäskymme ovat EQU (engl. equals), DC (engl. data constant) ja DS (engl. data segment). EQU-käskyä käytetään samaistukseen ja DC- ja DS-käskyillä varataan tilaa globaaleille muuttujille.

tunnus EQU arvo

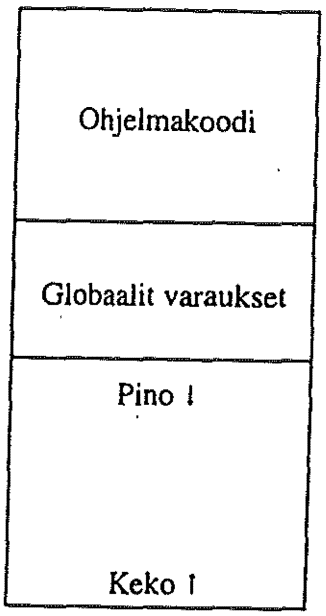
Samaistuskäskyllä EQU määrittelemme symboliselle tunnukselle kokonaislukuarvon "arvo". Tunnusta voi käyttää käskyn ADDR -kentässä, jolloin kääntäjä käsittelee sen kuten vastaavaan paikkaan olisi kirjoitettu ko. kokonaisluku "arvo".

tunnus DC arvo

Muistinvarauskäskyllä DC varaamme yhden muistisanan vakiota varten. Käsky samaistaa varatun osoitteen ja symbolin "tunnus" sekä asettaa varatun muistipaikan sisällöksi luvun "arvo". Tunnusta voi käyttää käskyn ADDR-kentässä kuten muita osoitetta.

tunnus DS koko

Muistinvarauskäskyllä DS varaamme muistialueen, jonka koko on "koko" (sanoina). Käsky samastaa varatun muistialueen alkuosoitteen ja symbolin "tunnus". Käy-



Kuva 4.2: Muistin käyttö.

tämme käskyä globaalien muuttujien tilanvaraukseen. Tunnusta voi käyttää käskyn ADDR -kentässä kuten muita osoitetta.

Luku 5

Konekielisiä ohjelmia

Seuraavassa on kokoelma konekielisiä ohjelmia. Osassa esimerkeistä on tyydytty esittelemään vain ohjelman pätkä kokonaisen ohjelman sijasta, osassa on täydellinen ohjelma. Koneemme rajoituksista johtuen kykenemme käsittelemään vain kokonaislukuja.

Symbolisella konekielellä ohjelmoitaessa periaatteena on, että oletetaan ohjelman alkavan muistipaikasta 0. Todellisuudessahan ohjelma voidaan ladata mihin kohtaan muistia tahansa. Tästä ei kuitenkaan aiheudu mitään ongelmaa, sillä muistinhallintayksikkö huolehtii tarvittavista osoitemuunnoksista kuten olemme nähneet.

5.1 Yksinkertaisen aritmeettisen lausekkeen laskeminen

Lasketaan muistipaikkojen 100 ja 101 sisältö yhteen, jaetaan summa muistipaikan 102 sisällöllä ja viedään tulos muistipaikkaan 100:

```
        LOAD  R1, 100
        ADD   R1, 101
        DIV   R1, 102
        STORE R1, 100
```

5.2 Peräkkäisten muistipaikkojen nollaus

Nollataan peräkkäiset muistipaikat. Ensimmäisen muistipaikan osoite on R2:ssa, viimeisen R3:ssa:

```
        LOAD   R1, =0
Loop    STORE  R1, 0(R2)
        ADD    R2, =1
        COMP   R2, R3
        JNGRE  Loop
```

5.3 Summan laskeminen

Lasketaan muistipaikoissa 100-200 olevien lukujen summa rekisteriin R1, jonka jälkeen lopetetaan:

```
        LOAD  R2, =100        -- indeksirekisteri
        LOAD  R1, =0
Alku    JNEG  R2, Loppu
        ADD   R1, 100(R2)
        SUB   R2, =1
        JUMP  Alku
Loppu   SVC   SP, =HALT
```

5.4 Monimutkaisemman summan laskeminen

Lasketaan summa $\sum_{i=1}^{20} i^2$. Javan tyyliin lasku sujuisi seuraavasti:

```
s = 0;
for (i = 1; i < 21; i++) s = s+i*i;
```

Matkitaan ylläolevaa koodia assemblerilla. Varataan muuttujille s ja i muistipaikat, annetaan niille alkuarvot kääntäjän ohjauskäskyillä ja käytetään näitä:

```
s     DC    0
i     DC    1

Taas  LOAD  R1, i
      MUL   R1, R1
      ADD   R1, s
      STORE R1, s
      LOAD  R1, i
      ADD   R1, =1
      STORE R1, i
      COMP  R1, =21
      JLES  Taas
      SVC   SP, =HALT
```

Sama ohjelma olisi voitu tehdä käyttäen pelkästään rekistereitä. Tällöin olisi vältytty muutamalta LOAD- ja STORE-käskyltä. Esimerkistä käy ilmi optimoimattoman kääntäjän tyyli generoida koodia.

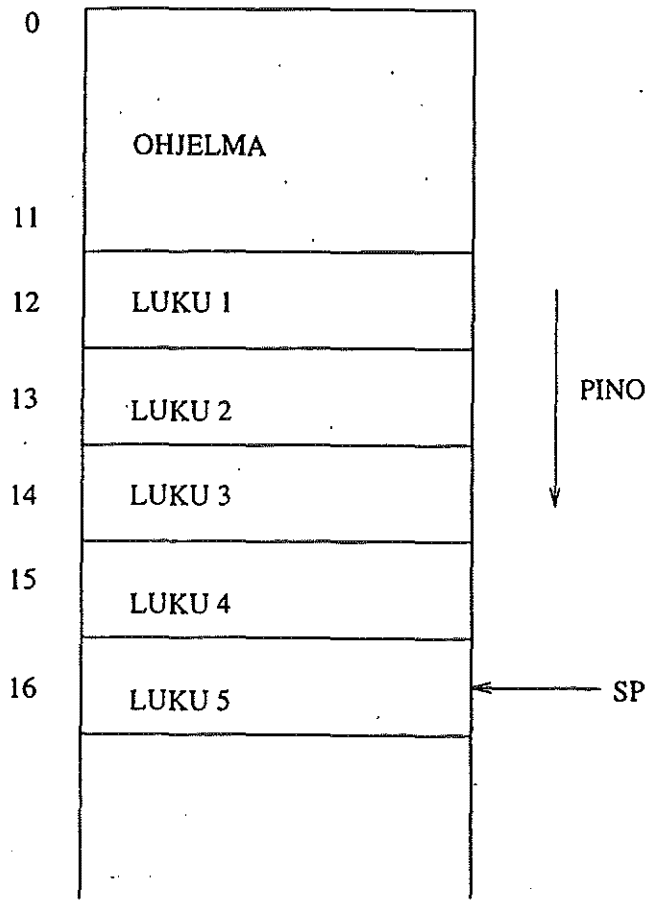
5.5 Pinon käyttö

Pino alkaa ohjelman ja staattisten muistivarausten jälkeen kuten nähtiin edellisessä luvussa. Konekielessä pino tarkoittaa, että staattisen alueen jälkeisiä muistipaikkoja voidaan ottaa käyttöön PUSH-operaatiolla muistipaikka kerrallaan. POP-operaatiolla puolestaan vapautetaan muistipaikkoja pinosta alkaen viimeisimmästä.

Seuraavassa on assembler-ohjelma, joka lukee lukuja näppäimistöltä kunnes kirjoitetaan 0 ja tulostaa luvut näytölle käänteisessä järjestyksessä.

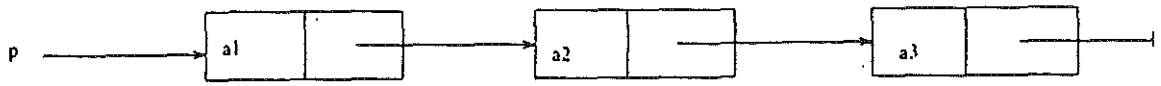
```
00:        LOAD   R1, =0
01: Lue    IN     R2, =KBD
02:        JZER   R2, Kirj
03:        PUSH   SP, R2
04:        ADD    R1, =1
05:        JUMP   Lue
06: Kirj   JZER   R1, Loppu
07:        POP    SP, R2
08:        OUT    R2, =CRT
09:        SUB    R1, =1
10:        JUMP   Kirj
11: Loppu  SVC    SP, =HALT
```

Tilanne muistissa, kun esimerkiksi viisi lukua on luettu:



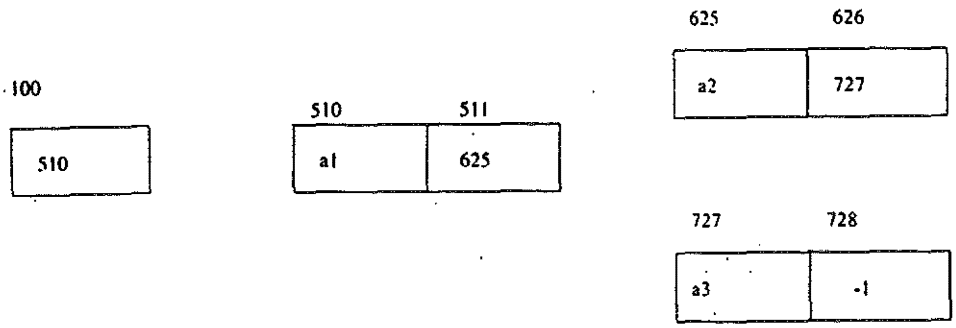
5.6 Linkitetty lista

Havainnollisesti linkitetty lista koostuu solmuista, joissa on tietoa ja viite toiseen (seuraavaan) solmuun. Lista on tietorakenne, johon voidaan tallettaa ajoaikana myöhemmin tarvittavaa tietoa. Edellytyksenä yleensä on, että listaa käydään läpi järjestyksessä. Listasta sieltä täältä tiedon hakeminen on hidasta.



Kuva 5.1: Lista

Oletetaan, että muistiin on talletettu lista, jonka solmuissa on kokonaislukuja. Jokaista solmua kohti on varattu kaksi peräkkäistä muistipaikkaa. Ensimmäisessä on kyseinen kokonaisluku ja jälkimmäisessä viite seuraavaan solmuun eli seuraavan kaksikon ensimmäiseen muistipaikkaan. Viite on yksinkertaisesti muistipaikan osoite. Oletetaan lisäksi, että muistipaikassa 100 on viite listan alkuun. Esimerkiksi tilanne voisi olla seuraava:



Kuva 5.2: Lista muistissa

Listan alkioiden summan laskeminen sujuu nyt epäsuoraa osoitusta käyttäen:

```
        LOAD   R1, 100
        JNEG   R1, Loppu     -- negatiivinen luku ilmaisee tyhjää
                                listaa
        LOAD   R2, =0        -- lasketaan summa R2:een
Loop    ADD    R2, @R1
        ADD    R1, =1
        LOAD   R1, @R1
```

```
        JNEG   R1, Loppu
        JUMP   Loop
Loppu   SVC    SP, =HALT
```

5.7 Aliohjelmat

Aivan kuten lausekielinen ohjelma myös konekielinen ohjelma voi koostua *pääohjelmasta* ja *aliohjelmista*. Aliohjelma suorittaa tehtävän, jota pääohjelma tarvitsee. Tyypillisesti pääohjelma *kutsuu* aliohjelmaa kerran tai useammin.

Aliohjelmalla on argumentit eli *parametrit*. Kutsuessaan aliohjelmaa pääohjelma käyttää parametreina haluamiaan sallittuja arvoja tai muuttujia. Tällöin puhutaan *todellisista parametreista*. Aliohjelma kirjoitetaan käyttäen *muodollisia parametreja*.

Aliohjelmat jaetaan *proseduureihin* ja *funktioihin*. Proseduuri tekee jonkin operaation käyttäen parametriensa arvoja tai muuttaa yhtä tai useampaa parametria. Funktio voi tehdä samoin, mutta lisäksi se *palauttaa* pääohjelmalle jonkin arvon.

Aliohjelmarakenne voi olla hierarkkinen useimmissa lausekielissä. Eli aliohjelmissa voi olla aliohjelmia ja aliohjelma voi kutsua toista aliohjelmaa. Konekielessä aliohjelmat eivät yleensä ole sisäkkäisiä, vaan ne on kirjoitettu peräkkäin pääohjelman jälkeen. Konekielessäkin aliohjelma voi kutsua aliohjelmaa.

Erityisesti viime vuosina on ruvettu puhumaan paljon olioista ja metodeista. Oliokielet ovat ns. moduulikielten yleistyksiä. Metodit ovat useimmiten tavallisia aliohjelmia, mutta toisinaan niihin liittyy dynaaminen sidonta. Ei-oliokielissä jo käännösvaiheessa tiedetään, milloin ja mitä aliohjelmaa kutsutaan. Sen sijaan dynaaminen sidonta tarkoittaa sitä, että asia ratkaistaan täysin vasta ajoaikana. Tällä kurssilla tarkastelemme vain staattisia aliohjelmia.

Aliohjelman parametrit voivat olla *viiteparametreja* (call-by-reference) tai *arvoparametreja* (call-by-value). Java-kielessä samoinkuin C:ssäkin kaikki parametrit ovat aina arvoparametreja. Sen sijaan Pascalissa, Adassa ja C++:ssa on myös viiteparametrit. Konekielellä ohjelmoitaessa voidaan käyttää kumpaa tahansa parametrityyppiä. Tyypin valinta riippuu tilanteesta.

Viite- ja arvoparametrin lisäksi on olemassa myös ns. *nimiparametri*, mutta se on melko vaikeasti toteutettavissa ja sitä on toisinaan hankala käyttää. Se on esiintynyt vain Algol-kielessä.

Konekielitasolla viite- ja arvoparametrin ero on helposti ymmärrettävissä. Viiteparametrin tapauksessa kutsuva ohjelma välittää aliohjelmalle viitteen todelliseen parametriin eli todellisen parametrin muistipaikan osoitteen. Jos esimerkiksi pääohjelmassa on käytössä muuttuja A, jolle on varattu muistipaikka 100, ja aliohjelmalle annetaan A viiteparametrina, niin aliohjelmalle välitetäänkin luku 100 eikä A:n arvoa. Aliohjelma voi nyt päästä käsiksi A:n arvoon epäsuoran viittauksen avulla. Samoin se voi muuttaa A:n arvoa ja muutokset jäävät voimaan aliohjelmasta poistuttaessa.

Arvoparametrin tapauksessa tilanne on oikeastaan yksinkertaisempi. Aliohjelmalle välitetään A:n arvo sellaisenaan. Aliohjelma voi nyt käyttää tätä arvoa, mutta se ei voi tehdä muutoksia itse A:han, jonka arvo siis säilyy aliohjelmasta poistuttaessa.

Konekielessä normaali tapa välittää parametreja aliohjelmalle tapahtuu pinon avulla. Eli parametrit viedään kutsuvaiheessa pinoon, josta aliohjelma saa ne käyttöönsä. Arvoparametrin tapauksessa pinoon viedään todellisen parametrin arvo, viiteparametrin tapauksessa pinoon tulee todellisen parametrin osoite.

Aktivointitietue

Aliohjelman suoritukseen liittyvää tietojoukkoa kutsutaan *aktivointitietueeksi* (engl. activation record) tai usein myös *aliohjelman ympäristöksi* (engl. environment). Aktivointitietue on kutsukohtaisesti muodostettava 'postilaatikko', jonne kutsuja tallettaa mm. paluuosoitteen ja aliohjelman kaipaamat parametrit ja josta kutsuttu varaa tilat paikallisille muuttujille.

Jos ohjelmointikieli sallii dynaamiset rakenteet (esim. taulukot, joiden koko määräytyy vasta suoritusaikana) ja rekursion, varataan aktivointitietueelle tilaa kutsun yhteydessä pinosta. Sinne varataan tilaa tarpeen mukaan funktion palauttamalle arvolle, paluuosoitteelle, parametreille ja aliohjelman paikallisille muuttujille. Lisäksi aktivointitietueessa on viitteitä aiemmin pinoon muodostettuihin aktivointitietueisiin. Varattuja tiloja vapautetaan aliohjelmasta paluun yhteydessä.

Toteutus TTK-91:ssä

Käytämme toteutuksessamme *dynaamista muistinhallintaa.* Varaamme siis tilaa aliohjelmien aktivointitietueille kutsun yhteydessä pinosta ja vapautamme sen aliohjelmasta paluun yhteydessä.

Jo aiemmin sovimme, että pinon huippuosoittimena käytämme rekisteriä SP (eli R6) ja ympäristö-osoittimena rekisteriä FP (eli R7). Pino-osoittimen SP arvona on aina pinon päällimmäisen alkion osoite. Ympäristöosoittimen FP arvona on puolestaan aina se osoite, jonka yläpuolella on ensimmäinen kutsutussa aliohjelmassa tehtävä tilavaraus. Aliohjelma viittaa suoritusaikana pinosta tehtyihin varauksiin ympäristöosoittimen avulla. Tilanvarausten osoitteethan eivät ole tiedossa vielä ohjelmoitaessa, joten niiden selvittely on pitänyt jättää tehtäväksi suoritusaikana.

Kun prosessi pääsee ensimmäisen kerran suoritukseen asettaa käyttöjärjestelmä ympäristöosoittimen FP alkuarvoksi käynnistyvän ohjelman viimeisen käskyn osoitteen. Jos ohjelmassa on tehty tilanvarauksia globaaleille muuttujille valekäskyin DC ja DS, on ensimmäinen varaus siten paikassa FP+1. Pino-osoittimen arvoksi käyttöjärjestelmä asettaa viimeisen valekäskyllä varatun paikan osoitteen.

Aliohjelmaan siirryttäessä on

- varattava tilaa pinosta funktion palauttamaa arvoa varten,
- vietävä pinoon parametrien arvot tai osoitteet,
- talletettava paluuosoite pinoon, jotta osataan palata takaisin oikeaan käskyyn,
- talletettava FP:n vanha arvo, jotta aliohjelmasta paluun jälkeen päästäisiin käsiksi niihin ympäristötietoihin, jotka olivat voimassa ennen kutsua,
- asetettava FP osoittamaan uuden paikallisen tilanvarauksen alkuun sekä
- asetettava PC:lle arvoksi aliohjelman ensimmäisen käskyn osoite.

Kun prosessin suoritus siirtyy aliohjelmaan, on aliohjelmassa vielä tehtävä tilanvaraukset aliohjelman paikallisille muuttujille.

Myös prosessorin rekistereiden R0..R5 arvot voi olla syytä tallettaa aliohjelman suoritusajaksi. Tällöin kutsuttu aliohjelma voi käyttää vapaasti rekistereitä omiin tarkoituksiinsa tai kutsua muita aliohjelmia. Talletuksen voi tehdä joko ennen aliohjelmakutsua tai vasta kutsutussa aliohjelmassa. Arvojen palautus takaisin rekistereihin tapahtuu luonnollisesti samassa ohjelmanosassa missä ne on talletettu: joko paluun jälkeen kutsujassa tai ennen paluuta aliohjelmassa. Rekisterit talletetaan pinoon PUSH-käskyillä ja palautetaan POP-käskyillä.

Sekä kutsuun että paluuseen näyttää siis liittyvän runsaasti jokaisella kerralla tehtäviä rutiineja. Vain parametrien ja paikallisten muuttujien käsittely vaihtelee eri kutsuissa, eikä rekistereiden talletus ole pakollista.

Paluuosoitteen talletus ja ympäristöosoittimen vanhan arvon talletus sekä uuden arvon asetus sisältyvät meillä aliohjelmakutsukäskyyn CALL ja niiden palauttaminen paluukäskyyn EXIT. Muista edellä mainituista tehtävistä on ohjelmoijan itse huolehdittava.

5.8 Esimerkki: Yksinkertainen aliohjelma

Kirjoitetaan symbolisella konekielellä pääohjelma ja aliohjelma. Aliohjelman nimi on KulutaAikaa ja sillä on yksi arvoparametri, joka on kokonaisluku. Aliohjelman pseudokoodi Javan kaltaisella kielellä on:

```
void KulutaAikaa(int Lkm) {
   int i;
   for (i = 0; i <= Lkm; i++)
       Tulosta(i);
}
```

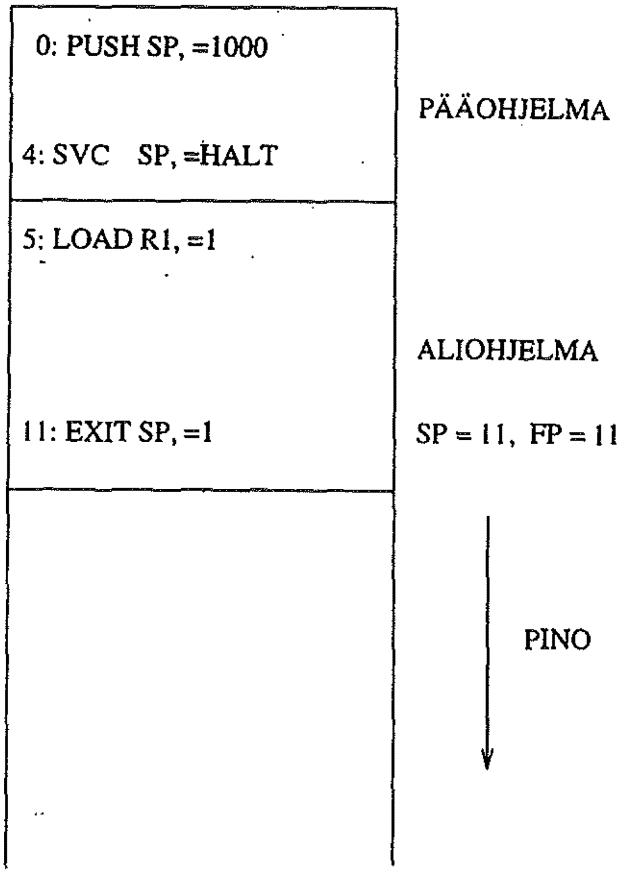
Pääohjelma vain kutsuu aliohjelmaa kahdesti arvoilla 1000 ja 5.

Staattista tilanvarausta ei tarvita, koska pääohjelmassa ei käytetä mitään tietoja. Todellisen parametrin arvo välitetään pinossa. Aliohjelma ei tarvitse paikallisia tilanvarauksia, koska kaikki oleellinen voidaan säilyttää rekistereissä ja parametrissa. Koska pääohjelma ei käytä myöskään rekistereitä, ei rekisterien arvoja ennen aliohjelmakutsua tarvitse viedä säilöön (pinoon).

```
 0:          PUSH    SP, =1000   -- arvoparametri 1000 pinoon
 1:          CALL    SP, KulAi   -- suoritetaan aliohjelma
 2:          PUSH    SP, =5      -- toinen parametri pinoon
 3:          CALL    SP, KulAi   -- toinen kutsukerta
 4:          SVC     SP, =HALT   -- pääohjelma päättyy

 5: KulAi    LOAD    R1, =1      -- aliohjelma alkaa
 6: Viela    COMP    R1, -2(FP)
 7:          JGRE    Pois
 8:          OUT     R1, =CRT
 9:          ADD     R1, =1
10:          JUMP    Viela
11:          EXIT    SP, =1      -- aliohjelma päättyy
```

Pääohjelma koostuu käskyistä 0-4 ja aliohjelma käskyistä 5-11. Konekielisessä ohjelmassa aliohjelmalle on annettu lyhennetty nimi KulAi, sillä esimerkiksi KOKSI-simulaattori asettaa rajoituksia symbolien pituudelle.

Tilanne muistissa ennen suorituksen alkamista on:



Tässä on siis oletettu, että ohjelma alkaa muistipaikasta 0. Hypyt ovat tämän mukaisia. Ohjelma kuitenkin ladataan muistiin mielivaltaiseen kohtaan ja muistinhallintayksikkö prosessorissa huolehtii osoitemuunnoksista.

Ensimmäiseksi välitetään todellinen arvoparametri, luku 1000, aliohjelmalle käskyllä

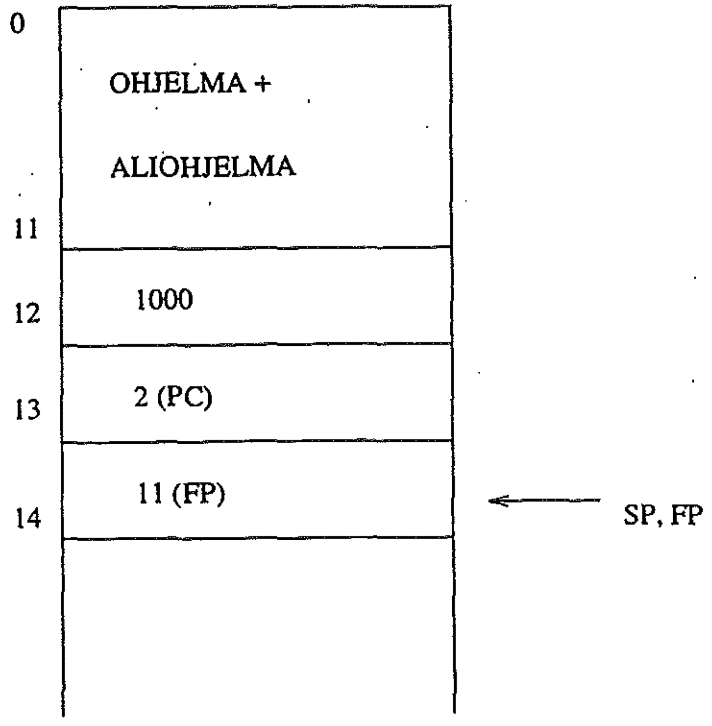
```
0: PUSH SP, =1000
```

Parametri sijoitetaan siis pinoon, josta aliohjelma voi käydä hakemassa sen. Pinon huipulle osoittaa rekisteri SP. Tässä vaiheessa siis SP osoittaa muistipaikkaan 12 eli sisältää luvun 12.

Tämän jälkeen kutsutaan aliohjelmaa. CALL-käsky saa aikaan seuraavaa:

- PC:n vanha arvo laitetaan pinoon. PC:n vanha arvo on 2 eli aliohjelmakutsun jälkeinen käsky 2 olisi normaali seuraava käsky. CALL aiheuttaa hyppykäskyn tapaan PC:n normaalista poikkeavan muutoksen, tässä tapauksessa arvoon 5. Vanha arvo on kuitenkin vietävä säilöön, jotta aliohjelmasta osattaisiin palata oikeaan kohtaan.
- FP:n vanha arvo viedään pinoon. Vanha arvo on 11 ja uusi arvo sen muistipaikan osoite, johon vanha arvo talletetaan.
- PC:n, FP:n ja SP:n arvot päivitetään uutta tilannetta vastaaviksi.
- Hypätään aliohjelman ensimmäiseen käskyyn.

Pinon tilanne ensimmäisen aliohjelmakutsun jälkeen on:



Tässä tilanteessa alkaa aliohjelma toimia. Parametreihin päästään käsiksi rekisterin FP avulla kätevästi. Parametrit ovat paikoissa FP-2, FP-3, ..., FP-k-1, jos parametreja on k kpl. Nyt parametreja on vain yksi ja se on paikassa FP-2. Parametrin osoite saadaan siis indeksoimalla käyttäen indeksirekisterinä FP:tä.

Aliohjelman suorituksen aikana pinon sisältö ei muutu, koska aliohjelma ei käytä paikallisia muuttujia vastaavia muistipaikkoja, se ei kutsu muita aliohjelmia eikä se palauta mitään arvoa kutsuvalle ohjelmalle.

Lopuksi aliohjelma suorittaa EXIT-käskyn. Se saa aikaan, että

- Kaikki mitä pinossa on FP:n jälkeen, vapautetaan.
- FP:n vanha arvo kopioidaan FP:hen.
- PC:n vanha arvo kopioidaan PC:hen.
- Pinosta poistetaan FP, PC ja EXIT-käskyn parametrin osoittama määrä muistipaikkoja. Tässä tapauksessa poistetaan siis muistipaikat 14, 13 ja 12 pinosta, jonka jälkeen muistin ja pinon tilanne on sama kuin alussa.

5.9 Arvo- ja viiteparametri aliohjelmassa

Laaditaan aliohjelma Laske(int n, int &x). Koska Javassa ei ole viiteparametreja, sovitaan siitä, että pseudokielessä viiteparametri ilmaistaan symbolin '&' avulla (kuten C++:ssa). Aliohjelma Laske suorittaa yksinkertaisen aritmeettisen laskutoimituksen, jonka tulos palautetaan x:ssä:

$$x = n + 10;$$

Pääohjelma vain kutsuu aliohjelmaa kerran n:n arvolla 15 ja viiteparametrilla M, missä M on pääohjelman muuttuja. Laskettu tulos myös tulostetaan näytölle.

Pää- ja aliohjelman koodi on:

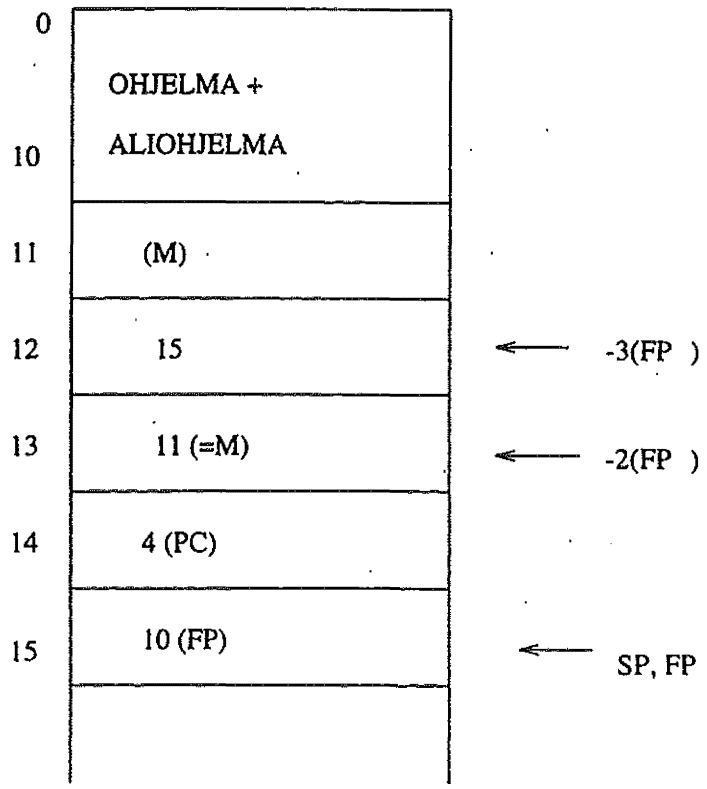
```
        M     DS   1

00: Main  LOAD  R1, =15
01:       PUSH  SP, R1
02:       PUSH  SP, =M   -- viiteparametrin osoite pinoon
03:       CALL  SP, Laske
04:       LOAD  R2, M
05:       OUT   R2, =CRT
06:       SVC   SP, =HALT
```

```
        n     EQU  -3
        x     EQU  -2
        K     EQU  10

07: Laske LOAD   R3, n(FP)
08:       ADD    R3, =K
09:       STORE  R3, @x(FP)
10:       EXIT   SP, =2
```

Pääohjelmassa viedään ensin todelliset parametrit pinoon (käskyt 01 ja 02). Arvoparametri, luku 15, viedään sellaisenaan pinoon. Todellinen viiteparametri on muistipaikka M. Koska M on viiteparametri, sen arvoa ei viedä pinoon (eihän sitä ole edes vielä olemassa), vaan pinoon viedään M:n osoite käyttäen välitöntä muistiosoitusta =M.

Aliohjelmassa noudetaan ensin arvoparametri, lasketaan sitten lausekkeen arvo ja viedään tulos todellisen viiteparametrin arvoksi. Todelliseen viiteparametriin eli muistipaikkaan M päästään käsiksi epäsuoran osoituksen avulla: pinosta löytyy M:n osoite paikasta x(FP) ja tätä hyväksi käyttäen päästään sijoittamaan M:ään.

Muistin sisältö aliohjelmakutsun jälkeen on seuraava:



5.10 Monipuolinen esimerkki

Lopuksi monipuolinen esimerkki, jossa ohjelmisto koostuu pääohjelmasta ja kahdesta aliohjelmasta, joista toinen on proseduuri ja toinen funktio.

Funktio Sqr(int n):

```
int Sqr(int n) {

    return n * n;
}
```

Proseduuri Summa(int &X, int Y):

```
void Summa(int &X, int Y) {

    int Z;

    Z = A + B;
    X = Sqr(X) + Y + Z +5 ;
}
```

missä A ja B ovat pääohjelman (globaaleja) muuttujia.

Pääohjelma:

```
int A, B;

lue luku A;
B = Sqr(2);
tulosta B;
Summa(B,A);
tulosta B;
```

Siis A ja B ovat pääohjelman muuttujia, joille on varattava tilaa muistista staattisesti eli jo ennen ohjelman suoritusta. Sen sijaan Z on aliohjelman paikallinen muuttuja, jolle varataan tila pinosta vasta ajoaikana ja jonka tila vapautetaan , kun sitä ei enää tarvita. Funktion paluuarvo välitetään pinossa, joten sille on varattava tila ennen kuin parametreja viedään pinoon.

Pääohjelma:

```
        A   DS   1
        B   DS   1

00: Main   IN    R3, =KBD
01:        STORE R3, A
02:        PUSH  SP, =0      -- paikan varaus funktion paluuarvolle
```

```
03:         PUSH  SP, =2
04:         CALL  SP, Sqr
05:         POP   SP, R2        -- Sqr:n laskema arvo otetaan pinosta
06:         STORE R2, B
07:         OUT   R2, =CRT
08:         PUSH  SP, =B        -- Viiteparametri pinon
09:         PUSH  SP, A         -- Arvoparametri pinoon
10:         CALL  SP, Summa
11:         LOAD  R2, B
12:         OUT   R2, B
13:         SVC   SP, =HALT
```

Aliohjelmat:

```
       n     EQU    -2
  Sqr_Ret    EQU    -3

14: Sqr     LOAD  R3, n(FP)
15:         MUL   R3, R3
16:         STORE R3, Sqr_Ret(FP)
17:         EXIT  SP, =1



       X     EQU   -3
       Y     EQU   -2
       Z     EQU    1

18: Summa PUSH   SP, =0       -- Paikallisen muuttujan Z tilanvaraus
19:       LOAD   R1, A
20:       ADD    R1, B
21:       STORE  R1, Z(FP)
22:       PUSH   SP, =0       -- Paikka funktion paluuarvolle
23:       PUSH   SP, @X(FP)
24:       CALL   SP, Sqr
25:       POP    SP, R3
26:       ADD    R3, Y(FP)
27:       ADD    R3, =5
28:       ADD    R3, Z(FP)
29:       STORE  R3, @X(FP)
30:       EXIT   SP, =2
```

Aliohjelma Sqr on suoraviivainen. Uutena piirteenä on vain paluuarvon talletus pinoon. Oikea paikka on ennen parametreja, joita nyt on vain yksi. Siis paluuarvon paikka on FP-3.

Aliohjelmassa Summa lasketaan ensin arvo A+B. Aliohjelmassa voidaan tietenkin viitata suoraan pääohjelman muuttujiin (muistipaikkoihin), joita siis A ja B ovat.

A:n ja B:n summa talletetaan aliohjelman lokaaliin muuttujaan Z, jolle on varattu paikka pinosta aivan aliohjelman alussa. Paikan osoite on FP+1.

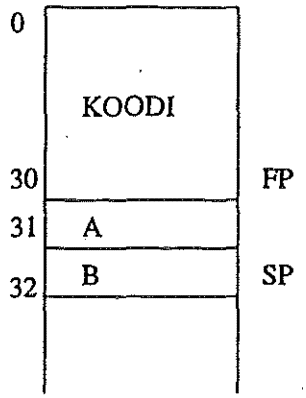
Seuraavaksi lasketaan lausekkeen Sqr(X)+Y+Z+5 arvo. Tätä varten aliohjelmassa Summa kutsutaan funktiota Sqr arvoparametrilla X. Huomattakoon tässä pieni monimutkaisuus:

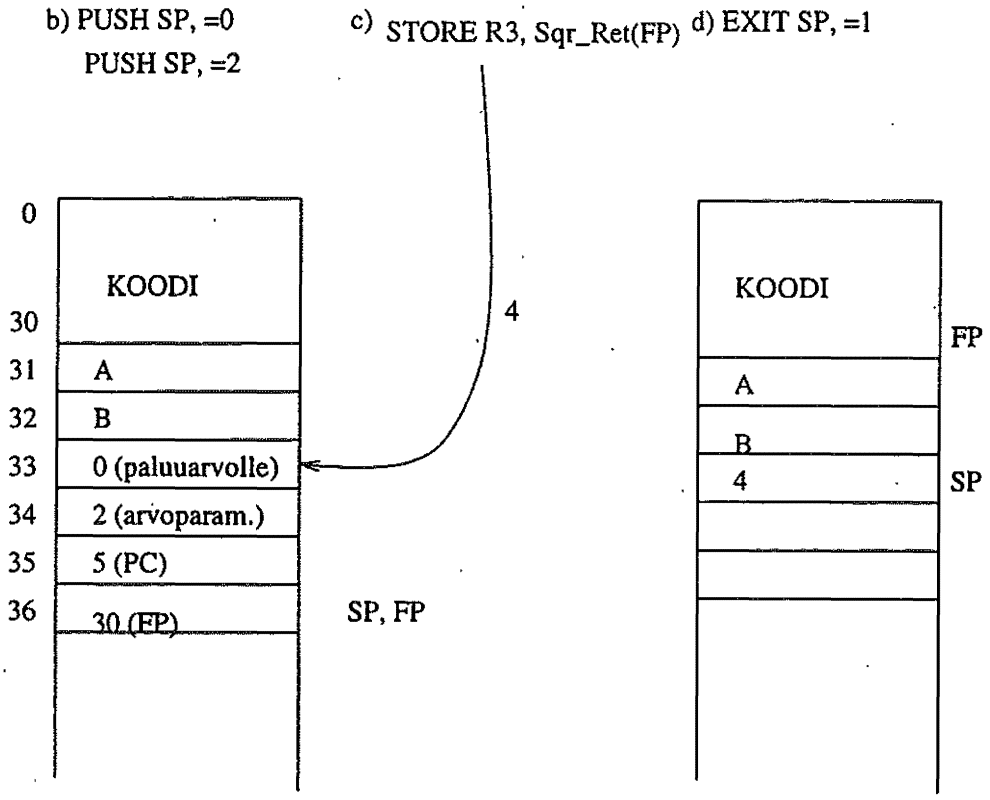
- Summa kutsuu Sqr:ää todellisella arvoparametrilla X.
- X on kuitenkin Summan muodollinen viiteparametri.
- Sqr:n kutsuhetkellä X:n arvona on viite B:hen.
- Sqr:n tulee käsitellä B:n arvoa eikä B:n osoitetta.
- Siksi pinoon Sqr:lle viedään B:n arvo, joka saadaan käskyllä `PUSH SP, @X(FP)`.

Sqr:n laskema arvo tulee pinoon, josta se saadaan POP-operaatiolla rekisteriin. Lopuksi laskettu arvo talletetaan parametrin X arvoksi eli B:n arvoksi.

Seuraavassa on kuvattu pinon sisältöä ohjelman suorituksen eri vaiheissa:

a) ALUSSA



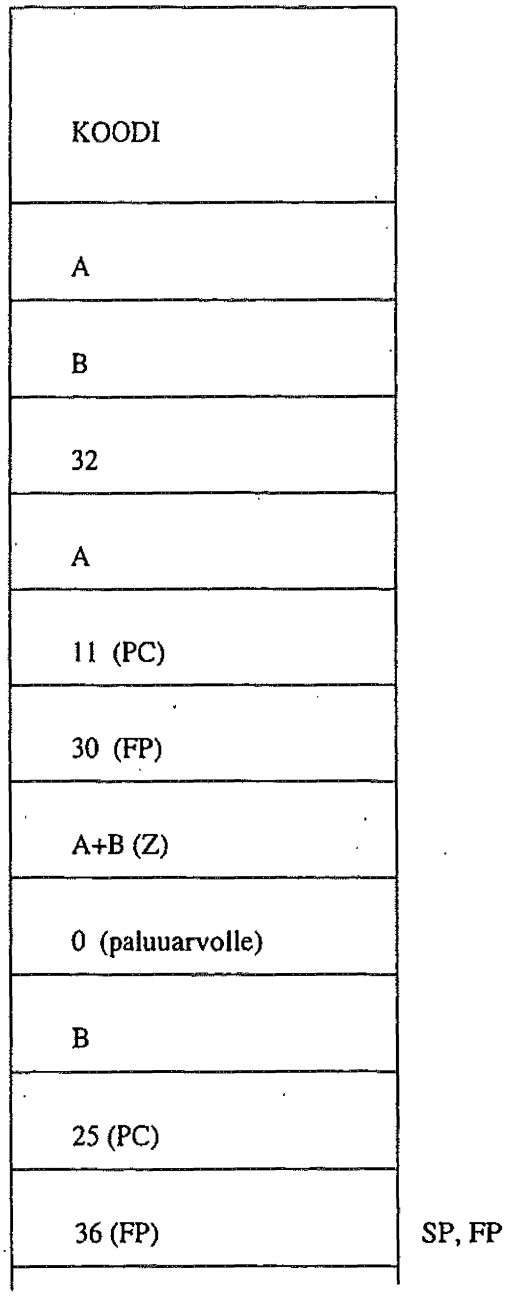

b) PUSH SP, =0
PUSH SP, =2
c) STORE R3, Sqr_Ret(FP)
d) EXIT SP, =1
0
KOODI
30
31 A
32 B
33 0 (paluuarvolle)
34 2 (arvoparam.)
35 5 (PC)
36 30 (FP)
4
SP, FP
KOODI
FP
A
B
4
SP

e) 5: POP SP, R2

8: PUSH SP, =B
9: PUSH SP, A
10: CALL SP, Summa

18: PUSH SP, 0

21: STORE R1, Z(FP)
22: PUSH SP, =0
23: PUSH SP, @X(FP)
24: CALL SP, Sqr



5.11 Käskyn suoritus TTK-91-rekisteritasolla

Prosessori toteuttaa käskyn määrittelemät toimenpiteet yhdistelemällä useita pieniä ja yksinkertaisia toimintoja, jotka pääasiassa ovat tiedon (bittijono) siirtämistä esim. rekisteristä toiseen rekisteriin, rekisteristä muistiin tai muistista rekisteriin. Siirtoihin voidaan liittää myös siirrettävän tiedon muuntaminen tai yhdistäminen toiseen siirrettävään bittijonoon erilaisten loogisten operaatioiden avulla (käytännössä suorittamalla siirto jonkin loogisen operaation suorittavan toimintayksikön 'läpi' esim. ALU:n).

Käskyn suoritus voidaan jakaa karkeasti kuuteen vaiheeseen:

1. Käskyn nouto ja PC:n kasvatus
2. Käskyn tulkinta (dekoodaus)
3. Osoitelaskenta ja 2. operandin nouto
4. Käskyn varsinainen suoritus
5. Tulosten talletus
6. Tilarekisterin tutkiminen sekä poikkeusten ja keskeytysten käsittely

Nämä vaiheet voivat vielä jakautua useisiin eri osavaiheisiin. Kaikkien käskyjen suoritukseen kuuluu aina vaiheet 1, 2 ja 6, mutta muista vaiheista joku/jotkut voivat puuttua. Lisäksi joidenkin käskyjen suoritukseen voi liittyä useitakin lisävaiheita. Suorituksen vaatima ohjaus on langoitettu prosessorin toteutukseen tai käskyjen suorituksesta vastaa ohjausyksikköön sijoitettu mikrokoodi. Mikrokoodi sisältää konekielisten käskyjen suoritusta ohjaavat ohjaussignaalit, joita ohjausyksikkö lähettää eri toimintayksiköille kellolaitteen tuottaman kellopulssin tahdistamana.

Emme perehdy mikrokoodin tai langoituksen periaatteisiin tai tarkempaan toimintaan, vaan tarkastelemme käskyn suoritusta vain prosessorin rekistereiden tasolla ja kuvaamme sanallisesti toiminnot, joiden toteutuksesta ohjausyksikkö huolehtii.

Käskyn nouto

Käskyn nouto on aina samanlainen käskystä riippumatta:

```
MAR             <-- MMU <- MAR <-- PC
PC              <-- PC + 1
ohjausväylä     <-- 'varaa väylä'
osoiteväylä     <-- MAR
ohjausväylä     <-- 'lataa muistista'
MBR             <-- MEM[MAR]          (siirto dataväylää pitkin)
ohjausväylä     <-- 'vapauta väylä'
IR              <-- MBR
```

Suoritettava käsky noudetaan siis käskyrekisteriin IR. Käskyn noudon jälkeistä toimintaa ohjaa IR:ssä oleva käskyn bittikombinaatio. Jos MMU toteaa saadun muistiosoitteen virheelliseksi, se asettaa SR:n bitin M (engl. forbidden memory address)

Käskyn tulkinta

Käskyn tulkinta tarkoittaa suurinpiirtein IR:ssä sijaitsevan käskyn 'jakamista' eri kenttiin ja nimenomaan käskykoodin tulkitsemista. Käskykoodi tarkistetaan ja käskyn osien avulla ohjausyksikkö päättää jatkotoimista. Käskykoodin avulla voidaan esim. "etsiä" oikea mikrokoodipätkä, joka sisältää ko. käskyn suoritusta ohjaavat ohjaussignaalikombinaatiot. Käskyn tulkinta voi myös osittain nivoitua seuraavien vaiheiden yhteyteen. Jos käskykoodi osoittautuu tuntemattomaksi, asetetaan SR:n bitti U (engl. unknown instruction).

Osoitteenlaskenta ja 2. operandin nouto

2. operandin nouto ja sitä mahdollisesti edeltävä osoitteenlaskenta on samanlainen lähes kaikille käskyille. Käskyn jälkimmäinen operandi määräytyy käskykoodin OPER ja kenttien M, Ri ja ADDR perusteella. Käskyn tyypistä riippuen toinen operandi voi olla arvo tai osoite, tai se voi puuttua kokonaan.

Ensiksi suoritetaan indeksointi, ellei indeksirekisterin paikalla ole R0:

```
jos (Ri<>0) niin TR <-- ADDR + Ri muuten TR <-- ADDR
```

Mahdollinen indeksointi tapahtuu siis yhteenlaskuna: yhteenlaskettavat siirretään ALU:n sisäänvienteihin ja ohjausyksikkö antaa ALU:lle ohjaussignaalin "+". Tulos ohjataan prosessorin sisäiseen työrekisteriin TR. Tämän jälkeen suoritetaan nolla, yksi tai kaksi muistinoutoa M-kentän perusteella. Saatu tulos (arvo tai osoite) ohjataan taas työrekisteriin TR.

```
jos (M=01) niin TR <-- MEM[TR]
jos (M=10) niin TR <-- MEM[MEM[TR]]
```

Noudettaessa operandeja muistista tehdään normaali muistinouto, johon liittyy fyysisen muistiosoitteen laskenta ja tarkistus sekä väylän varaus ja vapautus:

```
MAR             <-- MMU <-- MAR <-- TR
ohjausväylä     <-- 'varaa väylä'
osoiteväylä     <-- MAR
ohjausväylä     <-- 'lataa muistista'
MBR             <-- dataväylä
ohjausväylä     <-- 'vapauta väylä'
```

Jos käsky on STORE, CALL tai hyppykäsky, tulkitaan työrekisterin TR sisältö kohdeosoitteeksi, muuten työrekisteriin TR on saatu käskyn jälkimmäisen operandin arvo. Jos käsky on POP tai NOP, ei saatua tulosta käytetä käskykohtaisessa suorituksessa.

Käskyn suoritus

Käskyn suoritus tapahtuu vasta käskysyklin loppupuolella, kun tarvittava 2. operandi on noudettu tai 2. operandina oleva kohdeosoite on selvitetty.

Käskykoodin ilmoittaman operaation vaatimat toimenpiteet suoritetaan pääsääntöisesti siten, että ohjausyksikkö siirtää operandit ALU:n sisäänvienteihin ja antaa ALU:lle käskykoodin ilmoittaman ohjauksen (esim "+", AND, SHL, COMP). Esimerkiksi ADD-käskyn suoritus

```
ALU_in1          <-- Rj
ALU_in2          <-- TR
ohjaus 'add'
Rj               <-- ALU_out
```

Tieto suorituksen epäonnistumisesta (aiheutetaan poikkeus) ja vertailukäskyn COMP tulos talletetaan tilarekisteriin SR tähän tarkoitukseen varattuihin bitteihin.

Ehdollisissa hyppykäskyissä, joita ei edellä eksplisiittinen vertailukäsky COMP, suorittaa ALU ensin vertailun nollaan ja vasta tämän jälkeen ohjausyksikkö päättää bittien G, E ja L arvoista sekä käskykoodista vaihdetaanko PC:n arvo vai ei.

Tuloksen talletus

Tuloksen talletus tarkoittaa siirtoa ALUn ulostulosta johonkin rekisteriin tai talletusta muistiin. Esimerkkikoneessamme vain käskyt STORE ja PUSH voivat tallettaa "tuloksensa" muistiin. Tällöin käytetään normaalia muistitalletusta.

Alla vielä lyhyesti kaikkien tiedonsiirtokäskyjen ja pinoa käsittelevien käskyjen suoritus- ja tuloksen talletusvaiheiden keskeiset osat, kun edellä selitetty osoitelaskenta on tehty. Käskyn 2. operandin arvo tai käskyn vaatima kohdeosoite on siis tässä vaiheessa prosessorin työrekisterissä TR.

```
LOAD Rj <-- TR
```

```
STORE          MAR              <-- MMU <-- MAR <-- TR
               MBR              <-- Rj
               ohjausväylä      <-- 'varaa väylä'
               osoiteväylä      <-- MAR
               dataväylä        <-- MBR
               ohjausväylä      <-- 'talleta muistiin'
               ohjausväylä      <-- 'vapauta väylä'
```

```
IN      MAR             <-- TR
        ohjausväylä     <-- 'varaa väylä'
        osoiteväylä     <-- MAR
        ohjausväylä     <-- 'lue laitteelta'
        MBR             <-- dataväylä
        Rj              <-- MBR
        ohjausväylä     <-- 'vapauta väylä'

OUT     MAR             <-- TR
        MBR             <-- Rj
        ohjausväylä     <-- 'varaa väylä'
        osoiteväylä     <-- MAR
        dataväylä       <-- MBR
        ohjausväylä     <-- 'kirjoita laitteelle'
        ohjausväylä     <-- 'vapauta väylä'

PUSH    SP              <-- SP + 1          kasvata pino-osoitinta
        MAR             <-- MMU <-- MAR <-- SP
        MBR             <-- TR
        ohjausväylä     <-- 'varaa väylä'
        osoiteväylä     <-- MAR
        dataväylä       <-- MBR
        ohjausväylä     <-- 'talleta muistiin'
        ohjausväylä     <-- 'vapauta väylä'

POP     Ri              <-- MEM[SP]
        SP              <-- SP - 1      vähennä pino-osoitinta

CALL    SP              <-- SP + 1      paluuosoite pinoon
        MEM[SP]         <-- PC
        SP              <-- SP + 1      kutsujan ympäristöosoitin
                                        pinoon
        MEM[SP]         <-- FP
        FP              <-- SP          kutsutun ympäristö FP:n
                                        arvoksi
        PC              <-- TR          kontrolli 2. operandin
                                        osoitteeseen
```

```
EXIT        SP          <-- FP          katkaise paikalliset
                                        varaukset pinosta
            FP          <-- MEM[SP]     kutsujan ympäristö pinosta
            SP          <-- SP - 1
            PC          <-- MEM[SP]     paluuosoite pinosta
            SP          <-- SP - 1
            SP          <-- SP-TR       poista pinosta 2. operandin
                                        ilmaisema määrä parametreja

SVC         aseta SR:n bitti S eli aiheuta poikkeus
            palvelun numero on TR:ssä
            palvelun parametrit viety kutsua ennen pinoon
```

Poikkeusten ja keskeytysten käsittely

Tilarekisterin SR sisältö tarkastetaan aina ennen seuraavan käskyn noutoa suoritetun käskyn aiheuttamien poikkeusten tai palvelupyynnön havaitsemiseksi. Samalla tarkastetaan myös, onko jokin keskeytys "tulossa" prosessorin ulkopuolelta joltakin laiteohjaimelta.

Kun prosessori havaitsee poikkeuksen tai keskeytyksen, se siirtyy suorittamaan käyttöjärjestelmän koodia (keskeytyskäsittely). Mikäli aihetta "erikoistoimiin" ei ole, niin aloitetaan seuraavan käskyn nouto kohdan 1) mukaisesti PC:n osoittamasta muistipaikasta.

Poikkeuksen tai keskeytyksen sattuessa ladataan sitä vastaava käsittelijännumero prosessorin työrekisteriin TR. Palvelupyyntökeskeytyksen numero on osoitelaskennan jäljiltä valmiiksi rekisterissä TR, mutta laitteistokeskeytyksen käsittelijännumero on pyydettävä keskeytyksen aiheuttaneelta laitteelta. Poikkeuksia vastaavat käsittelijänumerot on helppo selvittää tilarekisteristä. Käsittelijöiden numerot ovat:

```
Poikkeukset                0:    luvun yli/alivuoto (O-bitti)
                           1:    nollalla jako (Z-bitti)
                           2:    tuntematon käsky (U-bitti)
                           3:    kielletty muistiviittaus (M-bitti)
Laitteistokeskeytykset:
                           5:    muistin pariteettivirhe
                           6:    kello
                           7:    näppäimistö
                           8:    hiiri
                           9:    levyasema
                          10:    kirjoitin
Palvelupyyntökeskeytykset: (S-bitti asetettu, palvelun numero TR:ssä)
                          11:    HALT, lopettaa ohjelman suorituksen
                          12:    READ, lukee I/O-laitteelta
```

```
13:     WRITE, kirjoittaa I/O-laitteelle
14:     TIME, palauttaa tunnit, minuutit ja
            sekunnit
15:     DATE, palauttaa vuoden, kuukauden ja
            päivän
```

Käsittelijän osoite saadaan muistin alusta numeroa vastaavasta ja TR:n osoittamasta muistipaikasta, eli käsittelijään voidaan siirtyä 'suorittamalla' mikrokoodilla käsky `LOAD PC,@TR`. Ennen keskeytyskäsittelyn aloittamista laitteisto tallettaa vielä tilarekisterin SR, käskyosoittimen PC (paluuosoite) ja ympäristöosoittimen FP arvot pinoon. Keskeytyskäsittelyn jälkeen voidaan haluttaessa palata nopeasti takaisin suorittamaan keskeytyneen ohjelman seuraavaa käskyä. Jos käsittelijä tarvitsee muita rekistereitä, se tallettaa itse niiden vanhat arvot pinoon, josta ne voidaan käsittelyn jälkeen palauttaa takaisin.

Luku 6

Käännös, linkitys ja lataus

Isojen sovellusten laatiminen ja ylläpito käy helposti hankalaksi, ellei käytettävä ohjelmointikieli anna mahdollisuutta jakaa kokonaisuutta itsenäisiin *käännösyksiköihin* eli *käännösmoduuleihin*. Käännösyksikkö on tavallisimmin aliohjelma tai luokka, mutta muunkinlaisia yksiköitä on olemassa kielestä ja ympäristöstä riippuen.

Jako moduuleihin on usein välttämätöntä, jos projektissa on useita ohjelmoijia. Kullakin moduulilla on tavallisesti oma erillinen tehtävänsä, ja moduuli muodostuu tuossa tehtävässä tarvittavista aliohjelmista ja niiden käsittelemistä muuttujista ja tietorakenteista.

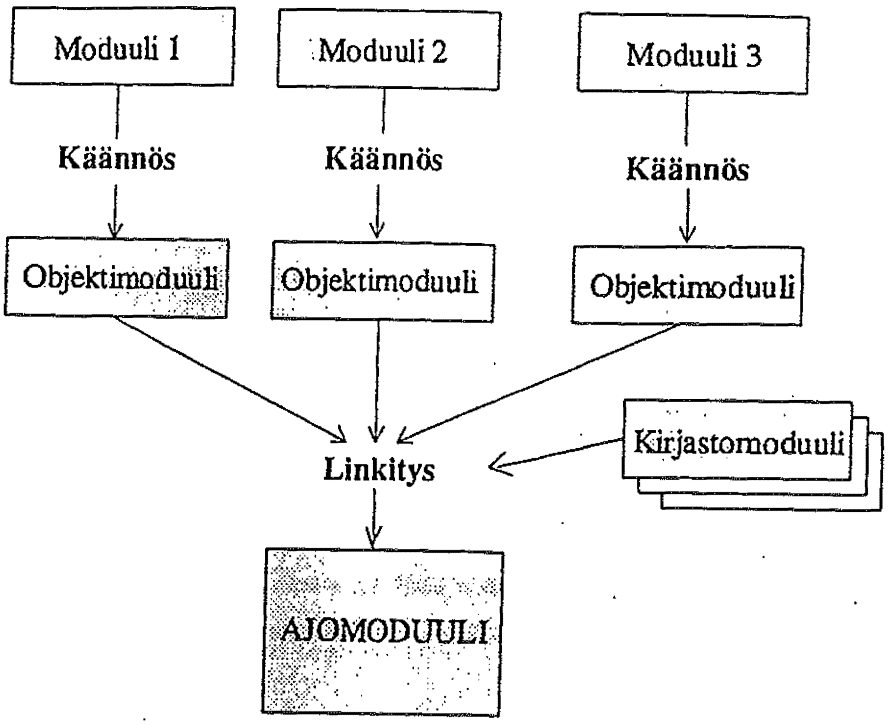
Hyvin suunnitellussa moduulimekanismissa tulee varautua ohjelmanosien erillään kääntämiseen sekä eri kielillä ohjelmoitujen ja konekieleksi käännettyjen osien yhteenliittämiseen linkitysvaiheessa.

Ohjelmanosien erillään kääntämisellä säästetään käännökseen kuluvaa aikaa, sillä vain muuttuneet moduulit pitää kääntää uudelleen. Myös uusien osien liittäminen ohjelmaan on nopeampaa. Moduulien erillääänkääntämisen etu tulee näkyviin erityisesti suurten ohjelmien kehitystyössä: lausekielisen ohjelman moduulaarinen rakenne helpottaa ohjelman suunnittelua, ohjelmointia ja ylläpitoa. Tavoitteena on jakaa isot ohjelmat moduuleihin siten, että ohjelmistolle saadaan selkeä hierarkinen rakenne ja kullakin moduulilla on selkeä osatehtävä.

Eri ohjelmointikielillä ohjelmoitujen moduulien yhteenlinkityksellä voidaan hyödyntää olemassaolevia rutiineja. Esimerkiksi Fortranilla on aikanaan tehty runsaasti matemaattisia rutiineja, joiden uudelleenohjelmointi vaikkapa Javalla olisi hankalaa ja selvää ajanhaaskausta.

Konekielisten moduulien käyttömahdollisuus on osoittautunut erittäin hyödylliseksi, sillä useimmissa ohjelmissa alue, joka on vain prosentti koko koodista, vastaa usein lähes 50 prosenttia suoritusajasta. Nämä suoritusajan kannalta kriittisimmät kohdat voidaan virittää tehokkaiksi konekielellä ja muut osat voidaan tehdä ohjelmoinnin kannalta mukavammin menetelmin, korkean tason lausekielellä.

Eri kielillä ohjelmoitujen moduulien linkitys edellyttää, että käytettävissä ohjelmointikielissä on sama kutsumekanismi ja parametrinvälitystapa. Tällöin kutsut-



Kuva 6.1: Käännös ja linkitys muodostavat parivaljakon, jonka tehtävänä on koota erikseen ohjelmoiduista moduuleista yksi yhtenäinen ajomoduuli, joka voidaan ladata keskusmuistiin suoritettavaksi.

tavan ei tarvitse tietää millä kielellä kutsuja on ohjelmoitu, eikä kutsujan tarvitse tietää aliohjelman toteutusta.

Moduulien väliset viittaukset

Moduulien välisiä viittauksia varten tarvitaan täsmälliset spesifikaatiot yhteisten tietorakenteiden käsittelylle, aliohjelmien kutsuille ja parametrien välitykselle. Tämä tarkoittaa sitä, että moduulin esittelyosassa kuvataan

- ne tunnukset, jotka esitellään tai määritellään tässä moduulissa ja joihin muut moduulit saavat viitata (ns. EXPORT-tunnukset, "viedä ulos", luokan tapauksessa julkiset (public) osat).

- ne tunnukset, jotka on esitelty / määritelty muissa moduuleissa ja joita käytetään tässä moduulissa (ns. IMPORT-tunnukset, "tuoda sisään").

- moduulin omat paikalliset tunnukset, joihin ei voi viitata muista moduuleista (luokan tapauksessa yksityiset (private) osat).

6.1 Käännös

Kääntäjän tehtävänä on muuttaa konekieltä korkeamman tason ohjelmointikielellä esitetty moduuli konekielelle. Kääntäjän tehtäviin kuuluu lauseiden ja lausekkeiden kuvaaminen konekielisin operaatioin, joissa kutakin lähdekielessä käytettyä tunnusta vastaa suhteellinen osoite. Koska kääntäjällä ei ole tietoa muista ohjelmakokonaisuuteen kuuluvista moduuleista, jäävät viittaukset muiden moduuleiden tunnuksiin vielä selvittämättä. Kääntäjä kokoaa tietoa näistä tunnuksista linkittäjää varten.

Käännös tehdään kahdessa vaiheessa. Syynä tähän on ensinnäkin se, että eteenpäin viittavien hyppykäskyjen osoiteosa voidaan rakentaa vasta, kun tiedetään kohteen osoite ja toiseksi se, että data-alueen alkuosoite selviää vasta, kun tiedetään koko ohjelman pituus.

Käännöksen ensimmäisessä vaiheessa kääntäjä

- tarkistaa lähdekielisen koodin syntaksin ja antaa tarvittaessa virheilmoituksia,
- muodostaa välikoodia, jossa eteenpäin viittaukset ovat vielä selvittämättä,
- laatii symbolitaulun, jossa kuhunkin ohjelman tunnukseen liittyy sen osoite ohjelman alun suhteen ja
- kokoaa linkitystä varten uudelleensijoitustietoa hypyistä ja viittauksista moduulin paikalliseen dataan

Käännöksen toisessa vaiheessa kääntäjä

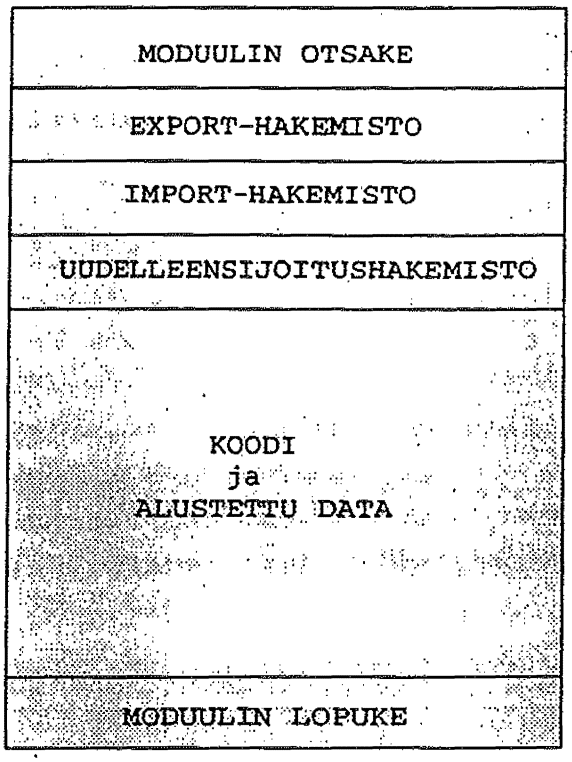
- muodostaa symbolitaulun perusteella koodin, jossa moduulin sisäiset osoitteet on paikattu kohdalleen, mutta ulkoiset osoitteet ovat vielä selvittämättä,
- kokoaa tulosmoduuliin tietoa linkittäjälle
  - paikallisiin osoitteisiin viittaavista käskyistä,
  - ulkoisista viittauksista eli IMPORT-tunnuksista sekä
  - ulosnäkyvistä tunnuksista eli EXPORT-tunnuksista,
- optimoi syntyvää koodia ja
- tuottaa mahdollisesti assembler-listauksen.

Symbolitaulussa on yksi alkio kutakin symbolista tunnusta kohden (kuva 6.2). Kun tunnus esiintyy ensimmäisen kerran symbolisessa konekielessä joko käskyn osoiteosassa tai viitteenä käskyn edessä, se viedään symbolitauluun. Kun tunnus esiintyy viitteenä jonkun käskyn tai kääntäjän ohjauskäskyn edessä sen arvo (ts. tunnusta vastaava osoite tai vakioarvo) voidaan merkitä tauluun. Symbolitauluun kerätään myös linkityksessä ja uudelleensijoituksessa tarvittavaa tietoa: tunnuksen tyyppi (esim. data, vakio, viite, export data, import data jne.), hyppykäskyjen osoitteet sekä dataan viittaavien käskyjen osoitteet.

Symboli	Tyyppi	Arvo	Uudelleen sijoitustietoa
A	data	? 32	Käskyissä 1, 9, 19
B	data	? 33	Käskyissä 6, 8, 11, 20
CRT	vakio	0	
Esm2	viite	0	
F	vakio	-3	
HALT	vakio	11	
...			

Kuva 6.2: Symbolitaulu.

Objektimoduuliin tuotetun koodin osoiteavaruus alkaa 0:sta ja siinä olevien muuttujien sekä hyppyosoitteiden arvot on kiinnitetty moduulin alun suhteen. Viittaukset muissa moduuleissa oleviin tunnuksiin (IMPORT-tunnukset) jäävät käännösvaiheessa vielä selvittämättä, ja niiden sitomista (engl. binding) varten kääntäjä kokoaa tietoa tuottamaansa objektimoduuliin. Kääntäjä kokoaa objektimoduuliin myös tietoa moduulin ulospäin näkyvistä tunnuksista (EXPORT-tunnukset) sekä niiden käskyjen osoitteet, joiden osoiteosaa on korjattava, jos objektimoduuli linkitetään yhteen jonkun toisen moduulin kanssa.



Kuva 6.3: Kääntäjän tuottaman objektimoduulin osat

Moduulin otsake-osaan (engl. header) on talletettu moduulin nimi ja linkittäjän

tarvitsemia tietoja: objektimoduulin osien pituudet, käännöspvm, kääntäjän nimi ja versio, sekä ensimmäisen suoritettavan käskyn osoite (ellei aina 0).

EXPORT-hakemistoon on talletettu tietoa niistä tunnuksista, joihin voidaan viitata muista moduuleista. Talletettavia tietoja ovat osoite, josta tunnukselle on varattu tilaa (eli symbolin arvo) ja mahdollinen käyttöoikeus (R/W/E/RW)

IMPORT-hakemistoon on talletettu tietoa niistä tunnuksista, jotka on esitelty tai määritelty muissa moduuleissa. Talletettavia tietoja ovat tunnus ja niiden käskyjen osoite, joissa ko. tunnusta on käytetty.

Varsinainen ohjelmakoodi ja sen käyttämä data (alustetut muuttujat) muodostavat objektimoduulin seuraavan osan. Muuttujille, joille ei ole annettu alkuarvoja, ei tarvitse varata tilaa objektimoduulissa. Ne on kuitenkin huomioitava data-alueen koossa, jotta ohjelmaa muistiin ladattaessa osataan varata tilaa myös alustamattomille muuttujille.

Uudelleensijoitushakemistossa (engl. relocation table) on niiden käskyjen osoitteet, joiden osoiteosaa on korjattava moduulien yhteiseen osoiteavaruuteen siirtymisen yhteydessä. Suoraviivainen tietyn siirtymän lisääminen kaikkiin osoiteosiin ei ole mahdollista, sillä käskyn osoiteosassa voi olla vakio. Kirjanpitoa joudutaan pitämään erikseen käskyistä, joissa on viitattu moduulin paikallisiin muuttujiin (data) ja erikseen hyppykäskyistä, sillä linkitettäessä yhdistetään erikseen moduulien koodialueet ja data-alueet.

Moduulin lopuke (engl. trailer) sisältää tarkistustietoja.

6.2 Koodin generoinnista

Seuraavassa luodaan katsaus siihen, minkälaista koodia generoituu tyypillisistä lausekielen rakenteista. Koodin generoinnissa ei ole käytettävissä vastaavanlaisia teorioita kuin jäsennyksessä, vaan laitteiston arkkitehtuuri ja käskykanta ratkaisevat osittain, minkälaista koodia tuotetaan. Kuitenkin vakiintuneiden tyyppien ja kontrollirakenteiden kääntäminen noudattaa seuraavassa esitettyjä periaatteita.

Totuusarvot

Totuusarvojen TRUE ja FALSE esittämiseen riittää yksi bitti. Usein tuhlataan tilaa tehokkuussyistä kuitenkin koko muistipaikka. Yleensä FALSE on 0 ja TRUE on 1.

Taulukot

Taulukolle varataan yhtenäinen muistialue, jonka alkuosoite tunnetaan. Muistialueen koko määräytyy taulukon alkioiden koon ja lukumäärän mukaan. Moniulotteisen taulukon alkiot talletetaan rivi- tai sarakejärjestyksessä peräkkäisiin muistipaikkoihin. Rivijärjestyksessä tallettaminen on yleisempää.

Esimerkiksi Javan taulukon

```
int [][] Taulu = new int[3][2];
```

kokonaisluvut tallettuisivat peräkkäisiin muistipaikkoihin esimerkiksi siten, että Taulu[0][0] on paikassa 1000, Taulu[0][1] paikassa 1001, Taulu[1][0] paikassa 1002 jne. Alkion osoite saadaan laskettua taulukon alkuosoitteen, indeksien ja alkion koon avulla. Kääntäjä tuottaa indeksien oikeellisuuden tarkistuksessa tarvittavan koodin ja selvittää indeksiviittauksesta oikean osoitteen. Symbolisella konekielellä ohjelmoitaessa ohjelmoijan on sen sijaan itse huolehdittava viittausten oikeellisuudesta.

Vaihtelevanmittaisen merkkijon eli merkkitaulukon esityksessä on kaksi perustapaa:

a) merkkijonon ensimmäisen tavun arvona on merkkijonon todellinen pituus ja sitä seuraavat tavut ovat merkkijonon merkkejä tai

b) merkkijonon lopussa on erillinen merkkijonon lopetusmerkki (ns. NULL-merkki).

Molemmissa tapauksissa merkkijonolle varataan tilaa ilmoitetun maksimipituuden verran.

Tietueet

Tietue (monissa kielissä record, C:ssä struct) koostuu kentistä, jotka voivat olla mitä tyyppiä tahansa. Tietue toteutetaan konekielitasolla vastaavalla tavalla kuin taulukko, eli sille varataan yhtenäinen muistialue, jonne kentät talletetaan peräkkäin. Kunkin kentän koko määräytyy kentän tyypin perusteella.

Joukot

Monissa kielissä on valmiiksi toteutettu joukkotyyppi. Toisaalta joukko voidaan toteuttaa itsekin.

Pienien joukkojen toteuttamiseen sopivat *bittivektorit*. Konekielitasolla yhdelle bitille voidaan varata koko sana tai todella vain yksi bitti. Perusidea on seuraava. Jos mahdollisia joukon alkioita on n kpl, esimerkiksi $a_1, \cdots, a_n$, niin varataan muistista $n+1$ peräkkäistä sanaa, $m_0, m_1, \cdots, m_n$. Joukko, joka sisältää osan alkioista $a_1, \cdots, a_n$, vaikkapa alkiot $a_{i_1}, \cdots, a_{i_k}$, koodataan seuraavasti:

		a_{i_1}			a_{i_j}		a_{i_k}			
m_0	m_1									m_n
n	0	1	0	$\cdots$	1	$\cdots$	1	0	$\cdots$	0

Eli jos a_i kuuluu joukkoon, on alkiota a_i vastaavassa muistipaikassa m_r arvo 1, muuten 0. Tilaa säästetään, jos alkiota kohti varataankin vain yksi bitti eikä koko

sanaa. Tällöin tosin ei ole niin helppoa käsitellä yhtä alkiolle varattua paikkaa, koska pienin osoitettava yksikkö on yleensä tavu.

Bittivektorit eivät yleensä sovellu tilanteisiin, joissa mahdollisia joukon alkioita on suuri määrä, mutta joukossa on vain osa näistä alkioista. Tällaisia tilanteita varten on kehitetty erilaisia tietorakenteita joukkojen hallintaan. Tietorakenteen valinnan ratkaisee ensisijaisesti se, minkälaisia joukko-operaatioita tarvitaan.

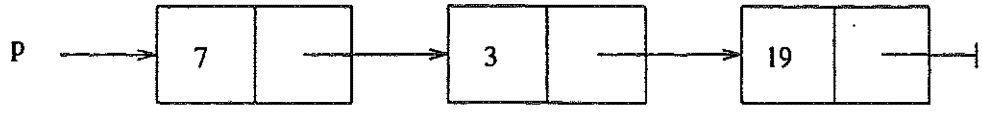
Viittaukset eli osoittimet

Viittaus olioon on keskeinen käsite monissa kielissä. Viittaus on hyvin yksinkertainen konekielitasolla. Esimerkiksi jos muistipaikka 100 viittaa muistipaikkaan 200, tarkoittaa tämä sitä, että muistipaikassa 100 on luku 200. Samalla tavalla voidaan viitata useita muistipaikkoja käsittäviin olioihin kuten taulukoihin. Jos taulukolle varataan vaikkapa muistipaikat 150 - 290, niin viittaus taulukkoon tarkoittaa samaa kuin viittaus taulukon ensimmäiseen alkioon eli ensimmäiseen muistipaikkaan.

Tarkastellaan esimerkkinä linkitetyn listan käsittelyä Javassa:

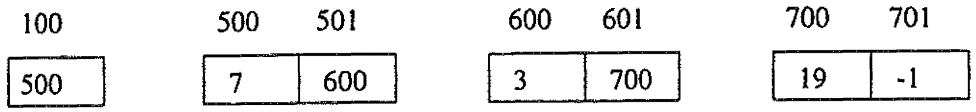
```
class Lista {
   int   tieto;
   Lista linkki;
}

...

Lista p = new Lista();
p.tieto = 7;
p.linkki = new Lista();
p.linkki.tieto = 3;
p.linkki.linkki = new Lista();
p.linkki.linkki.tieto = 19;
```

Yllä olevat lauseet luovat seuraavan rakenteen:



Kuva 6.4: Linkitetty lista.

Tällainen rakenne voidaan toteuttaa konekielitasolla varaamalla kaksi peräkkäistä muistipaikkaa jokaista alkiota kohti ja lisäksi yksi muistipaikka osoittimelle p, joka viittaa listan alkuun:



Kuva 6.5: Listan esitys konekielitasolla.

Viitaukset hallitaan kätevästi epäsuoran osoituksen avulla. Seuraavassa yllä olevan listan generointi symbolisella konekielellä. Oletetaan, että käytössä on aliohjelma New, joka tekee tarvittavat muistipaikkojen varaukset, kun parametrina annetaan osoitin ja varauksen koko.

```
PUSH   SP,  =p      ; osoittimen osoite pinoon parametriksi
PUSH   SP,  =2      ; varataan kaksi muistipaikkaa listan alkiolle
CALL   SP,  NEW
LOAD   R1,  =7
STORE  R1,  @p
LOAD   R2,  p
ADD    R2,  =1      ; nyt R2 osoittaa listan 1. alkion 2. kenttään
PUSH   SP,  R2
PUSH   SP,  =2
CALL   SP,  NEW
LOAD   R1,  =3
LOAD   R2,  @R2
STORE  R1,  @R2
ADD    R2,  =1
PUSH   SP,  R2
PUSH   SP,  =2
CALL   SP,  NEW
LOAD   R1,  =19
LOAD   R2,  @R2
STORE  R1,  @R2
```

Ajoaikana on huolehdittava tarpeettomien rakenteiden vapauttamisesta. Esimerkiksi jos edellä olevan listan osoittimen p arvoksi asetetaan NULL, ei listaan enää päästä käsiksi, vaikka varaukset ovat edelleen muistissa. Vapauttaminen voi taphtua joissakin järjestelmissä automaattisesti, mutta tavallisesti ohjelmoijan on itse huolehdittava asiasta.

Kontrollilauseiden kääntäminen

Ehdoton haarautuminen. Useissa kielissä on goto-lause, jolla voidaan hypätä toiseen kohtaan ohjelmassa. Tämä voidaan toteuttaa konekielessä yksinkertaisesti JUMP-käskyllä. Javassa ei ole got-lausetta, mutta kylläkin break- ja continue-lause, jotka myös toteutetaan JUMP-käskyn avulla.

Ehdollinen haarautuminen. Tämä tulee vastaan tavallisesti if-lauseen muodossa:

```
if (a == 5)   b =1;

if (a == 5)  b = 1;  else  b = 2;
```

Molemmat kääntyvät helposti konekielelle. Esimerkiksi jälkimmäinen tuottaa seuraavaa koodia:

```
a       DS    1
b       DS    1
        ...
if      LOAD    R1, a
        COMP    R1, =5
        JNEQU   else
        LOAD    R2, =1
        STORE   R2, b
        JUMP    *+3          ; kasvatetaan käskylaskuria 3:lla
else    LOAD    R2, =2
        STORE   R2, b
        ...
```

Samalla periaatteella kääntyvät monimutkaisemmat ehtolauseet, kuten esimerkiksi valintalause switch.

Toisto. Toistolauseet, kuten while-, do- ja for-lauseet, toteutetaan käyttämällä sopivasti ehdotonta ja ehdollista haarautumista. Esimerkiksi toistorakenne

```
a = 2;
while (a > 0)
{ a = a-1;}
```

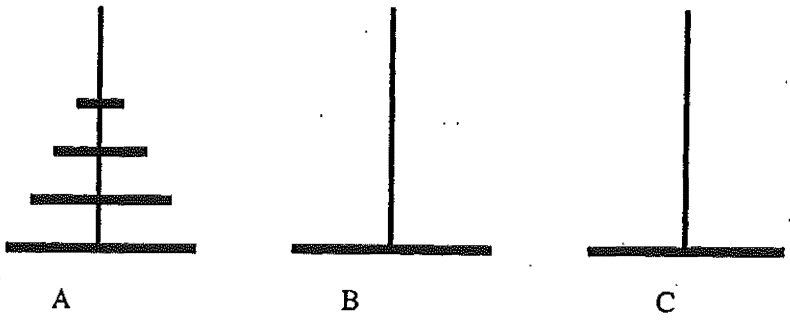
kääntyy konekielelle seuraavasti:

```
a       DS      1
        ...
        LOAD    R1, =2
        STORE   R1, a
while   JNPOS   R1, endwhile
```

```
            LOAD    R1, a
            SUB     R1, =1
            STORE   R1, a
            JUMP    while
endwhile ...
```

Aliohjelmat

Lausekielen aliohjelmat käännetään konekielen aliohjelmiksi. Parametrit välitetään yleensä pinossa, pinoon varataan tila myös aliohjelman paikallisille muuttujille. Rekursiiviset aliohjelmat toteutetaan samoin kuin tavallisetkin. Esimerkkinä rekursiivisesta aliohjelmasta tarkastellaan Hanoin tornien ongelmaa.



Kuva 6.6: Hanoin tornit.

Tangossa A on erikokoisia levyjä. Tehtävänä on siirtää levyt A:sta B:hen käyttämällän hyväksi tankoa C noudattaen seuraavia sääntöjä. Vain yhtä levyä kerrallaan saa siirtää. Siirrossa levy on otettava levypinon päältä ja vietävä toiseen tankoon. Levyä ei saa asettaa tangossa pienemmän päälle.

Laaditaan rekursiivinen ohjelma, joka ilmoittaa, millä siirroilla levypino saadaan siirretyksi tangosta A tankoon B:

```
void siirra(int n, int a, int b, int c)
{
  if (n == 1) tulosta("a-->b");
  else
  {
    siirra(n-1, a,c,b);
    tulosta("a-->b");
    siirra(n-1,c,b,a);
  }
}
```

Pseudokielisessä aliohjelmassa ensimmäisenä parametrina on siirrettävien levyjen lukumäärä. Kolme muuta parametria edustavat tankoja, jotka on koodattu kokonaisluvuiksi. Vastaava assembler-ohjelma on seuraava:

```
    n       DC

            IN      R1, =KBD
            STORE   R1, n
            PUSH    SP, R1
            PUSH    SP, =1
            PUSH    SP, =2
            PUSH    SP, =3
            CALL    SP, siirra
            SVC     SP, =HALT


        C   EQU     -2
        B   EQU     -3
        A   EQU     -4
        n   EQU     -5

siirra      LOAD    R1, n(FP)
            COMP    R1, =1
            JNEQU   viela
            LOAD    R2, A(FP)
            LOAD    R3, B(FP)
            OUT     R2, =CRT
            OUT     R3, =CRT
            EXIT    SP, =4
viela       LOAD    R1, n(FP)
            SUB     R1, =1
            PUSH    SP, R1
            PUSH    SP, A(FP)
            PUSH    SP, C(FP)
            PUSH    SP, B(FP)
            CALL    siirra
            LOAD    R2, A(FP)
            LOAD    R3, B(FP)
            OUT     R2, =CRT
            OUT     R3, =CRT
            LOAD    R1, n(FP)
            SUB     R1, =1
            PUSH    SP, R1
            PUSH    SP, C(FP)
            PUSH    SP, B(FP)
```

```
PUSH  SP, A(FP)
CALL  SP, siirra
EXIT  SP, =4
```

Luokka

Nykyisissä oliokielissä luokan käsite on keskeinen. Luokka koostuu erilaisista tyypeistä, muuttujista ja aliohjelmista eli metodeista. Luokka on korkean tason käsite, eikä sen toteutus tuo kovin paljon uutta konekielitasolla. Lähtökohtana on tietueen toteutus, mutta lisäpiirteitä tulee aliohjelmien liittämisestä luokkaan. Siten luokka toteutetaan varaamalla

- ensin tila muuttujille, yksi tai useampi muistipaikka per muuttuja;
- sitten osoittimet aliohjelmiin eli muistipaikat, jotka sisältävät aliohjelmien 1. käskyn osoitteen.
- Aliohjelmat voidaan sijoittaa muun koodin yhteyteen.

Lisäpiirteitä aiheutuu aliohjelmien dynaamisesta sidonnasta virtuaalisten aliohjelmien tapauksessa. Tällaisten aliohjelmien hallitsemiseksi ylläpidetään tietorakennetta, ns. virtuaalitaulua, jonka avulla sidonnat tehdään ajoaikana. Yksityiskohdat sivuutetaan.

Koodin optimointi

Optimoimaton kääntäjä tuottaa varsin tehotonta koodia. Esimerkiksi lauseista

```
A = A +1;
if (A > 0) A = 2 * A;
```

voisi generoitua seuraava käännös:

```
A       DS      1

        LOAD    R1, A
        ADD     R1, =1
        STORE   R1, A
        LOAD    R1, A
        JNPOS   R1, *+4
        LOAD    R1, A
        MUL     R1, =2
        STORE   R1, A
```

Tehokkaampaa olisi pitää arvoja rekistereissä mahdollisimman kauan:

```
A       DS    1

        LOAD  R1, A
        ADD   R1, =1
        JNPOS R1, *+3
        MUL   R1, =2
        STORE R1, A
```

Keskeinen osa optimointia on nimenomaan vähentää turhia muuttujien päivityksiä. Joissakin tilanteissa tämänlaatuisesta optimoinnista aiheutuu kuitenkin ongelmia. Jos esimerkiksi ohjelmointikieli mahdollistaa useiden prosessien luomisen ja jos kullakin prosessilla on oma kopionsa globaalista eli jaetusta muuttujasta, eivät eri prosessien versiot muuttujasta olekaan välttämättä samoja, josta aiheutuu virhetilanteita.

6.3 Linkitys

Linkittäjä (engl. linker) muodostaa käännetyistä moduuleista ajomoduulin, jossa erikseen käännettyjen moduulien väliset viittaukset on selvitetty ja kuvattu yhteiseen osoiteavaruuteen. Lisäksi linkittäjä liittää tulosmoduuliin tarvittavat kirjastorutiinit. Moduulit sijoitetaan annetussa järjestyksessä osoitteesta 0 alkaen ja moduulien sisäiset suhteelliset osoitteet ja eri moduulien väliset viittaukset muutetaan uusiksi yhdistetyn moduulin alun suhteen.

Myös monet linkittäjät toimivat tavallaan kaksivaiheisesti:

1. Lue objektimoduulit ja tee taulukko moduulien nimistä ja pituuksista sekä laske kullekin moduulille ja kunkin moduulin data-alueelle uudelleensijoitusvakiot (ts. arvo joka on lisättävä käskyn osoiteosan sisältöön)
2. a) Käsittele EXPORT- ja IMPORT-hakemistojen perusteella moduulien väliset viittaukset.
   b) Käsittele uudelleensijoitushakemistot.

Sekä 2a) että 2b)-kohdassa lisätään hyppykäskyjen (ja aliohjelmakutsujen) osoiteosaan sen moduulin uudelleensijoitusvakio, jonka alueelle hyppy tapahtuu. Dataan vittaavien käskyjen osoiteosaan pitää lisätä koodin uudelleensijoitusvakion lisäksi myös kyseessäolevan data-alueen uudelleensijoitusvakio.

Seuraavan sivun esimerkissä on hahmoteltu käännöstä ja linkitystä.

Tehtävä: Korvaa ensin kussakin objektimoduulissa esiintyvät paikalliset symbolit vastaavilla sisäisillä osoitteilla. Kokoa sitten yksi yhdistetty ajomoduuli, ja selvitä moduulien väliset viittaukset yhdistämällä IMPORT- ja EXPORT-osien tiedot. Muista huomioida uudelleensijoitusvakiot. Korjaa lopuksi kunkin moduulin uudelleensijoitushakemistossa mainittujen käskyjen osoiteosia.

```
Moduuli A, 301, 1, ...

Proc1 (h): 100
Proc2 (h): 200
   S (d): 201,204
   T (d): 203

000:     LOAD  R1,X
...
...
100:     CALL  Sp,Proc1
...
...
200:     CALL  SP,Proc2
201:L1   LOAD  R2,S
202:     JZER  R2,L2
203:     ADD   R2,T
204:     STORE R2,S
...
210:     JUMP  L1
211:L2   LOAD  R1,X
...
...
300:     SVC   SP,=HALT

301: X:11e

hypyt: 202, 210
data:  0, 211
```

```
Moduuli B, 100, 2, ...

Proc1 (h): 0
Proc2 (h): 50
   S (d): 100
   T (d): 101

Fun1 (h): 52

000:Proc1 ADD   SP,=2
...
049:      EXIT  SP,=0
050:Proc2 ADD   SP,=1
051:      PUSH  SP,S
052:      CALL  SP,Fun1
053:      POP   SP,R1
054:      STORE R1,S
055:      CALL  SP,Proc1
...
060:Loop  JNEG  R5,Ok
...
070:      JPOS  *+5
...
098:      JUMP  Loop
099:Ok    EXIT  SP,=0

100:S:11e
101:T:11e

hypyt: 55, 60, 70, 98
data:  51, 54
```

```
Moduuli C, 200, 0, ...

Fun1 (h): 0
Fun2 (h): 101

000:Fun1  ADD  SP,=2
...
...
100:      EXIT SP,=1
101:Fun2  LOAD R1,2(FP)
...
...
199:      EXIT SP,=0
```

Allaolevassa taulukossa koodien ja data-alueiden pituudet sekä uudellensijoitusvakiot

	Pituus	USV
A-koodi	301	0
B-koodi	100	301
C-koodi	200	401
A-data	1	300
B-data	2	201
C-data	0	3

6.4 Lataus

Kun käyttöjärjestelmä lataa ohjelman suoritusta varten keskusmuistiin, on sen tehtävä ajomoduulin ajonaikainen paikanmääritys. Tämä tarkoittaa sitä, että objektimoduulin sisäiset osoitteet eivät vastaa suoraan todellisia muistiosoitteita. Tarvitaan osoitemuunnos.

Yksiajojärjestelmässä ajomoduulin paikka muistissa voi olla kiinteä, eli suoritettava ohjelma voidaan viedä aina samaan kohtaan keskusmuistia. Tällaisessa tapauksessa ajomoduulin sisäiset osoitteet voidaan muuttaa todellisiksi keskusmuistiosoitteiksi jo ohjelman linkitysvaiheessa.

Moniajojärjestelmässä muistissa voi olla useita ajomoduuleja yhtä aikaa ja eri suorituskerroilla ohjelma voi sijaita eri osissa muistia. Käyttöjärjestelmän tehtävänä on etsiä riittävän suuri alue ohjelman koodille ja työtiloille sekä järjestää kuvaus ajomoduulin sisäisistä osoitteista todellisiksi fyysisiksi keskusmuistiosoitteiksi sekä käynnistää ohjelman suoritus.

Tilanahtauden vuoksi ohjelma tai sen osia saatetaan joutua poistamaan välillä keskusmuistista. Tätä kutsutaan *heittovaihdoksi* (engl. swapping) Uudelleenlatauksen yhteydessä ohjelma saatetaan sijoittaa täysin uuteen paikkaan, jolloin todelliset keskusmuistiosoitteet ovat erilaisia kuin ennen poistoa. Yksi mahdollisuus olisi tehdä uudelleensijoituksen vaatima osoitemuunnos aina ohjelmaa muistiin tuotaessa. Tämä edellyttää, että linkittäjä jättää ajomoduuliin lataajan käyttäväksi uudelleensijoitushakemiston. Tavallisin tapa on kuitenkin tehdä osoitteiden kuvaaminen todellisiksi muistiosoitteiksi vasta ajonaikana. Osoitemuunnos tehdään laitteistotoimenpiteenä muistinhallintayksikössä MMU, joten se ei siten hidasta ohjelman suoritusta.

Kuten aiemmin olemme todenneet, tämä muunnos sujuu yksinkertaisimmillaan siten, että muistinhallintayksikkö MMU lisää MAR-rekisteriin tuotuun ohjelman osoitteeseen kantarekisterin BASE arvon. Sen arvonahan on ohjelman alkuosoite. Virtuaalimuistijärjestelmässä osoitemuunnos tehdään sivutaulujen ja / tai segmenttitaulujen avulla.

Luku 7

Yleistä käyttöjärjestelmästä

Käyttöjärjestelmä on tietokoneen tärkein ohjelmisto. Sen avustuksella muut ohjelmat voivat käyttää prosessoria ja sen oheislaitteita. Käyttöjärjestelmän tarkoituksena on tehostaa ja helpottaa tietokoneen käyttöä. Varsinainen käyttöjärjestelmä koostuu varsin suuresta määrästä laitteistotoimintoja, mikrokoodia ja ohjelmistoa. Käyttöjärjestelmään kuuluviksi lasketaan usein myös monet tekniset varusohjelmat kuten ikkunointi, tietoliikenneohjelmistot, tiedostoapuohjelmat, editorit, kääntäjät, linkittäjät jne.

Käyttöjärjestelmä tarjoaa kullekin käyttäjälle illuusion omasta virtuaalikoneesta: käyttäjä voi kuvitella olevansa koneen ainoa käyttäjä (kuva 7.1). Tässä virtuaalikoneessa on helpot siirräntä- ja tiedostokäskyt, ja lisäksi runsas joukko valmiita apuohjelmia. Käyttöjärjestelmä jakaa koneen yhteiskäyttöisiä resursseja (prosessori, I/O, muisti, ohjelmat, tiedostot) soveluksille siten, että käyttäjien virtuaalikoneet eivät sotke toisiaan ja ne voivat toimia hallitusti yhdessä. Käyttöjärjestelmä tarjoaa liittymän laitteiston ja ohjelmiston välille.

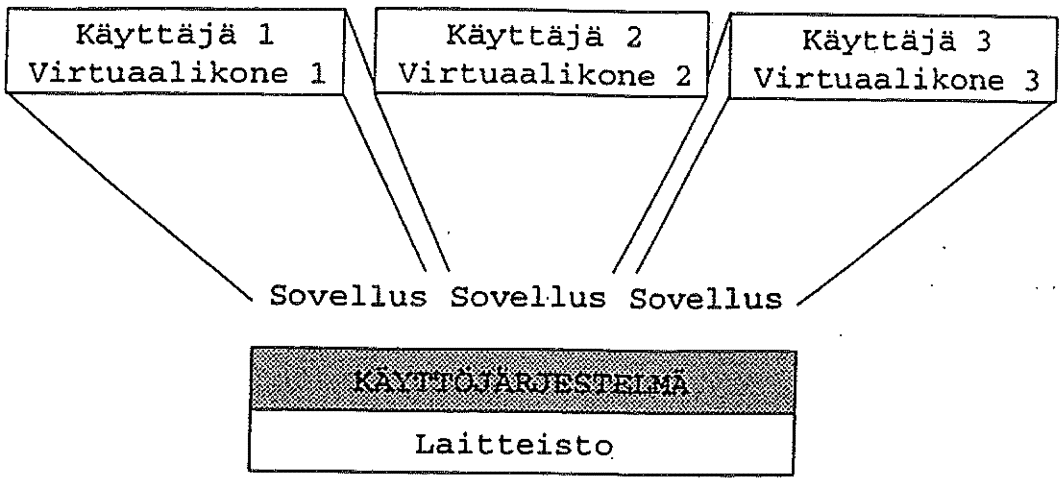
7.1 Käyttöjärjestelmän kehityksestä

Kytkimet, ei käyttöjärjestelmää

1940-luvun ensimmäisissä tietokoneissa ei ollut käyttöjärjestelmiä. Koneessa voitiin suorittaa vain yhtä ohjelmaa kerrallaan ja sen suorittaminen vaati jatkuvaa operointia. Ohjelma ja sen syötteet sekä ohjaustiedot annettiin koneelle suoraan kytkimistä.

Seuraavassa vaiheessa konekielisistä ohjelmista siirryttiin aluksi symbolisella konekielellä ja myöhemmin korkeamman tason kielillä esitettyihin ohjelmiin. Sekä ohjelma että sen data lävistettiin valmiiksi reikäkorteille, mutta ohjaustiedot annettiin edelleen kytkimistä. Esimerkki työn kulusta:

- lävistä ohjelma ja data reikäkorteille
- aseta ohjelma reikäkortin lukijaan ja käynnistä luku kytkimestä



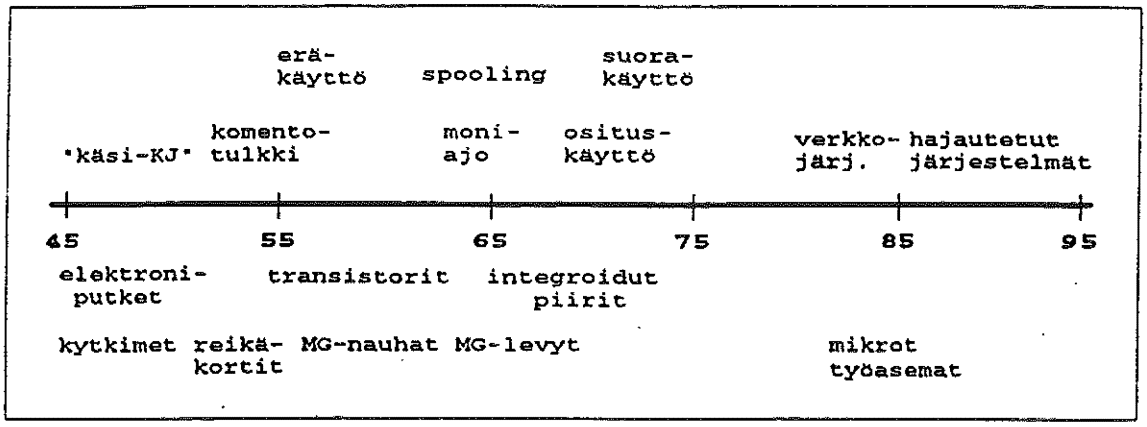
Kuva 7.1: Käyttöjärjestelmä antaa jokaiselle käyttäjälle illuusion omasta virtuaalikoneesta.

- aseta kääntäjä reikäkortin lukijaan ja käynnistä luku kytkimestä
- käynnistä kääntäjä antamalla operointipanelista PC:lle alkuarvo (vakioarvo)
- aseta syöttöaineisto kortinlukijaan
- käynnistä sovelluksen suoritus

Lataaja, komentotulkki

1950-luvulla saatiin ensimmäiset käyttöjärjestelmät, kun operaattorin rutiiniluonteiset tehtävät siirrettiin muistissa jatkuvasti olevalle 'perusohjelmalle'. Korttipakan sekaan laitettiin ohjauskortteja, joilla kerrottiin työvaiheet. Käyttöjärjestelmä oli siis pieni ohjelmien *lataaja* ja *komentotulkki* (engl. command interpreter, shell), joka ymmärsi yksinkertaista komentokieltä. Tieto työvaiheiden onnistumisesta palautui komentotulkille, jolloin se saattoi tarvittaessa keskeyttää työn tai jatkaa seuraavalla vaiheella.

Magneettinauhojen yleistyessä toimintaa nopeutettiin lukemalla erillisellä koneella ennakolta isompi joukko töitä reikäkorteilta nauhalle. Sitten nauha kiikutettiin keskuskoneelle, joka luki hitaan reikäkortinlukijan sijasta nyt 'kortinkuvia' selvästi nopeammalta nauhalta. Vastaavasti tulosteet ('rivinkuvat') ohjattiin nauhalle ja ne tulostettiin erillisellä koneella kirjoittimelle.



Kuva 7.2: Tietotekniikan ja käyttöjärjestelmän kehitys

Spooling, keskeytykset, eräkäyttö

Kun 60-luvun puolivälissä saatiin käyttöön magneettilevyt, luovuttiin erillisistä syöttö- ja tulostuskoneista ja alettiin käyttää *spooling-menetelmää* (engl. simultaneous peripheral operation on-line). Tämä tuli mahdolliseksi, sillä mg-levylle voitiin sekä kirjoittaa että lukea yhtäaikaa. Spoolingia käytettäessä luettiin kortit etukäteen nauhan sijasta levylle ja tulostukset ohjattiin ensin levylle, josta kirjoitin tulosti ne aikanaan paperille.

Seuraava innovaatio oli hitaan siirrännän ja laskennan limittäminen. Siihen tarvittiin kaksi uutta laitteistopiirrettä: I/O-prosessori ja *keskeytysmekanismi*. I/O-prosessorilla voitiin siirtää tietoa suoraan keskusmuistiin ja se vapautti prosessorin näistä tehtävistä. Työn seuraava vaihe voitiin lukea muistiin samalla kun prosessori suoritti työn edellistä vaihetta. Keskeytysmekanismin avulla voitiin suorituksessa oleva ohjelma keskeyttää tilapäisesti siirräntään liittyvää käsittelyä varten. Keskeytyksen käsittelyn jälkeen palattiin suorittamaan samaa ohjelmaa (engl. off-line).

Moniajo, suorakäyttö

1960-luvun puoliväliin saakka ongelmana oli edelleen se, että suoritettavana oli kerrallaan vain yksi työ ja siirrännän vuoksi prosessori joutui odottelemaan toimettomana.

Moniajojärjestelmään siirryttäessä otettiin keskusmuistiin samanaikaisesti useita töitä ja koneen resursseja (prosessori, I/O-laitteet, muisti, tiedostot) jaettiin vuorotellen niiden kesken niiden kulloisenkin tarpeen mukaisesti. Tämä lisäsi käyttöjärjestelmään uusina tehtävinä resurssien kontrolloinnin ja jakelun sekä vaati töiden suojaamista toisiltaan. Järjestelmän oli huolehdittava siitä, etteivät prosessit päässeet viittaamaan toisten muistialueille.

Alkuvaiheessa käytettiin ns. tapahtumaohjattua skedulointia eli suoritettavaa ohjelmaa vaihdettiin esiintyvien keskeytysten, esimerkiksi siirräntäpyyntöjen yhteydessä. Myöhemmin järjestelmät tehtiin aikaohjatuiksi: kullekin työlle annettiin aikaviipaleita vuorotellen ja aikaviipaleen päättyminen sekä siirrännän aloittaminen tai päättyminen aiheutti keskeytyksen. Jokaisen keskeytyksen jälkeen käyttöjärjestelmä päätti, mitä työtä suoritettiin seuraavaksi. Tällaista järjestelmää kutsutaan *osituskäyttöjärjestelmäksi* (engl. timesharing, multitasking).

Suurin puute käyttäjän kannalta oli siinä, että käyttöjärjestelmät perustuivat edelleen pelkästään *eräkäyttöön* (engl. batch processing). Eräkäytössä kuvataan erityisellä työnohjauskielellä kaikki työn vaiheet sekä kerrotaan tarvittavat ohjelmistot ja tiedostot ennenkuin työ annetaan kokonaisuudessaan koneen suoritettavaksi. Kun työ on käynnistetty, ei käyttäjä enää pysty vaikuttamaan työn kulkuun (korkeintaan voi seurata sitä tai katkaista sen).

Erätyöhön perustuva järjestelmä kävi pian vanhanaikaiseksi, sillä ihmis- ja konetyön hintasuhde oli alkanut ratkaisevasti muuttua. *Suorakäytön* (engl. on-line processing) lisääminen vaati luonnollisesti muutoksia myös käyttöjärjestelmään: syötteet ja komennot tuli lukea päätteeltä ja ainakin osa tulostuksista tuli ohjata päätteelle. Suorakäytössähän on olennaista käyttäjän ja tietokoneen välinen vuorovaikutus. Käyttäjä ohjaa työn etenemistä näppäimistöltä antamillaan syötteillä ja ohjelmat puolestaan kertovat ohjaustarpeestaan tulostamalla näytölle. Käyttöjärjestelmään tuli uusina vaatimuksina myös siedettävän vastausajan takaaminen.

1970-luvulla rakennettiin erä- ja suorakäyttö 'yhtenäiseen' käyttöjärjestelmään. Lisäksi käyttöön otettiin useissa tapauksissa *virtuaalimuisti*. Virtuaalimuistin ansiosta koko ohjelmakoodin ja kaiken datan ei tarvinnut olla kerralla muistissa. Näin saatiin muistiin tilaa useammalle prosessille. Käyttöjärjestelmä huolehti automaattisesti siitä, että muistissa oli aina tarvittavat osat.

Verkkojärjestelmät, hajautetut järjestelmät

1980-luvun puolivälin jälkeen, ja mikrotietokoneiden ja työasemien yleistyttyä, alettiin siirtyä keskitetystä tietojenkäsittelystä hajautettuun tietojenkäsittelyyn. Laitteistoja alettiin yhdistää toisiinsa tietokoneverkoilla. Yhteisten resurssien joustava käyttö toi käyttöjärjestelmään uusia käsitteitä ja vaati uusia ohjausohjelmia. Uusimmissa *hajautetuissa järjestelmissä* (engl. distributed operating system) ei enää vaadita, että ohjelmaa suoritettaisiin siinä koneessa, jossa käyttäjä on sen käynnistänyt, vaan käyttöjärjestelmä voi ohjata työn sille koneelle, joka on kulloinkin vapaana tai jonka kuorma on pienin.

Historiallinen kehitys näkyy useissa käyttöjärjestelmissä varsin voimakkaana. Vanhojen käyttöjärjestelmien piirteitä on raahattu mukana versiosta toiseen, jotta uusitusta käyttöjärjestelmästä huolimatta voitaisiin ajaa myös vanhoja ohjelmia.

7.2 Käyttöjärjestelmien perustyypit

Mikrotietokoneiden *yhden käyttäjän järjestelmät* tarjoavat yhdelle käyttäjälle kerrallaan liittymän koneeseen. Käyttöjärjestelmä tunnistaa tietyn komentokielen, ylläpitää tiedostojärjestelmää ja tarjoaa rutiinit oheislaitteiden hallintaan. (Esimerkkejä MS-DOS (engl Microsoft Disk Operating System), Amiga DOS)

Moniajojärjestelmässä on suoritettavana vuorotellen useita ohjelmia. Nyt käyttöjärjestelmän on edellä mainittujen tehtävien lisäksi huolehdittava resurssien tasapuolisesta jakelusta ja sen tulee taata kullekin ohjelmalle toisistaan riippumaton suoritus, eli ohjelmat on suojattava toisiltaan. Jaettavina resursseina on prosessori, keskusmuisti, eräät oheislaitteet sekä tiedostot. Moniajojärjestelmät ovat yleiskäyttöisiä ja ne antavat tietokoneen käyttäjälle paitsi mahdollisuuden tehdä omaa työtä muista riippumatta myös mahdollisuuden toimia yhteistyössä toisten kanssa. (Esimerkkejä Machintosh System 7, Windows, OS/2, Linux, Unix, VAX/VMS, VM jne.)

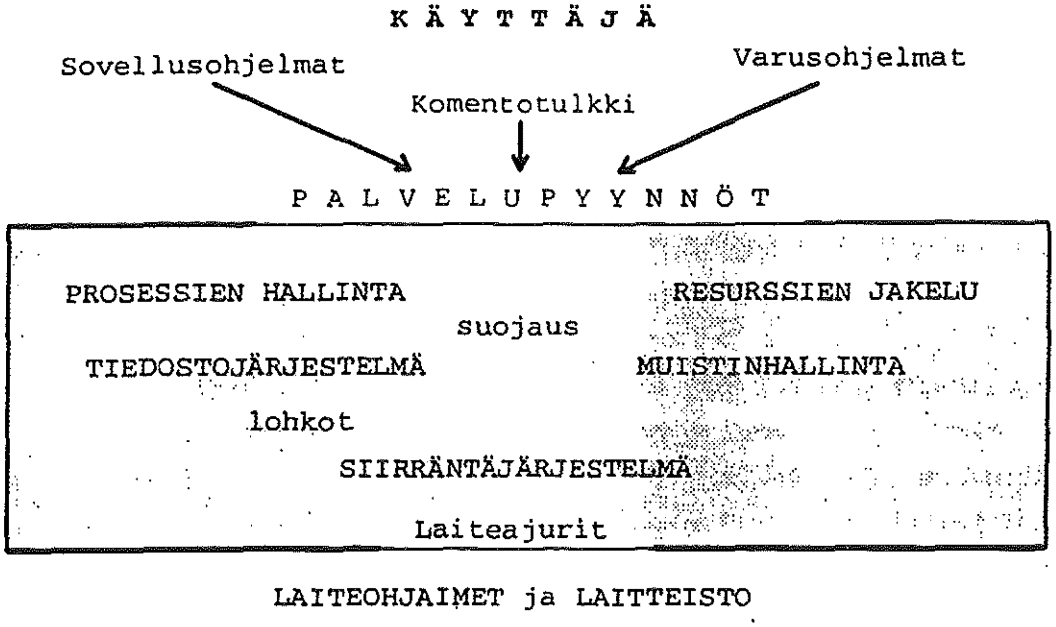
Moniprosessorijärjestelmässä (engl. multiprocessor) laitteistossa on toiminnan nopeuttamiseksi useita prosessoreita. Käyttöjärjestelmän tehtävänä on jakaa prosessit tarkoituksenmukaisesti näiden suoritettaviksi. Moniprosessorijärjestelmä on tavallisesti organisoitu siten, että prosessorit on kytketty toisiinsa ja ne jakavat yhteisen keskusmuistin, tiedostojärjestelmän ja oheislaitteet ('tiukasti kytketty laitteisto', engl. tightly coupled hardware). Kokonaisuuden ohjauksesta vastaa yksi käyttöjärjestelmä.

Prosessoreiden välinen yhteistyö voidaan järjestää myös siten, että se perustuu täysin itsenäisten koneiden väliseen sanomanvaihtoon ('löyhästi kytketty laitteisto', engl. loosely coupled hardware). Koneet on yhdistetty toisiinsa tietokoneverkolla. Eri koneissa olevat prosessorit ovat siis fyysisesti erillisiä ja kullakin on oma keskusmuisti.

Verkkojärjestelmässä (engl. network operating system) kullakin koneella on oma käyttöjärjestelmä ja omat prosessit suoritettavanaan, mutta kaikki koneet jakavat yhteisen tiedostojärjestelmän ja oheislaitteet ('löyhästi kytketty ohjelmisto'). Käyttäjille verkkojärjestelmä näkyy joukkona koneita, joihin voi kirjoittautua sisään tai joita voi käyttää erityisin etäkomennoin. (Esimerkkejä Solaris)

Hajautetussa järjestelmässä käyttäjien ei tarvitse tuntea järjestelmään kuuluvia koneita. Järjestelmä pystyy mm. käyttäjien tietämättä kuormantasaukseen (engl. load balancing) eli se pystyy itsenäisesti päättämään missä koneessa prosessi suoritetaan. Hajautetun järjestelmän koneiden käyttöjärjestelmät toimivat kiinteässä yhteistyössä, joten kaikissa koneissa on samanlainen käyttöjärjestelmä ('tiukasti kytketty ohjelmisto'). (Esimerkkejä Amoeba, Mach)

Suurten tietomäärien, tietokantojen (engl. database), hallintaan on kehitetty tietokannan hallintajärjestelmiä ja *tapahtumankäsittelyjärjestelmiä* (engl. database system). Koska tietokanta on keskitetty, usean sovelluksen yhteiskäytössä oleva resurssi, on sen keskitetty hallinta tärkeää jo esimerkiksi tiedon oikeellisuuden vuoksi. Myös tietosuojan järjestäminen on helpompaa keskitetysti. Tietokannoille, joissa



Kuva 7.3: Käyttöjärjestelmän tehtäviä

on usein ja paljon muutoksia (esim paikanvaraus, pankin tilitiedot), on tiedon ajantasalla pitäminen keskeisessä asemassa. Järjestelmän tehtävänä on lisäksi käsitellä yhteentörmäykset, joissa samaa tietoa haluttaisiin käsitellä samanaikaisesti useissa eri pisteissä.

7.3 Käyttöjärjestelmän tehtäviä

Käyttöjärjestelmän keskeisimmät osat ovat prosessien hallinta, muistinhallinta, siirräntäjärjestelmä sekä tiedostojärjestelmä (kuva 7.3).

Käyttöliittymä, komentotulkki

Käyttöjärjestelmä tarjoaa tietokoneen käyttäjälle joukon komentoja, joiden avulla käyttäjä voi käynnistää töitään suoritettavaksi ja ohjata niiden suoritusta. Käyttöliittymänä (engl. user interface) voi olla yksinkertainen rivipohjainen komentotulkki, jonka *kehotteeseen* (engl. prompt) nuo komennot syötetään näppäimistöltä. Työasema- ja mikrotietokoneympäristössä on käytettävissä myös monipuolisempia ja näyttävämpiä *graafisia käyttöliittymiä*.

Käyttöjärjestelmät tarjoavat yleensä miltei samat palvelut, mutta niiden toteutus ja käytettävät komennot vaihtelevat. Yleensä komennot ovat sitä yksinkertaisempia mitä lähemmäksi tavallista käyttäjää ne on tarkoitettu.

Moniajokoneessa käyttöjärjestelmä antaa järjestelmän *ylläpitäjille* (engl. superuser, system manager, root) suuremmat valtuudet kuin tavalliselle käyttäjälle. Ylläpitäjät voivat puuttua järjestelmän ohjaukseen ja valvontaan. He voivat vaikuttaa esimerkiksi työjonoihin ja resurssien jakeluun sekä voivat halutessaan katkaista jonkin prosessin suorituksen.

Prosessien ohjaus

Moniajojärjestelmissä on yhtä aikaa useita samanaikaisia ja / tai rinnakkaisia prosesseja. Muistissa on yhtäaikaa useita ohjelmia ja useiden eri käyttäjien ohjelmia ja siirräntää suoritetaan rinnan käskyjen kanssa. Käyttöjärjestelmä toteuttaa töiden ja prosessien vastaanoton sekä niihin liittyvien *työ- ja prosessijonojen* hallinnan. Käyttöjärjestelmä huolehtii töiden ja prosessien käynnistyksestä, niiden tarvitsemien resurssien varaamisista ja vapauttamisista sekä prosessien lopetuksesta. Prosessien hallinnan ylimmällä tasolla käyttöjärjestelmä päättää mitkä työt otetaan erätyöjonoista suoritukseen. Prosessien hallinnan hienojakoisemmalla tasolla käyttöjärjestelmä päättää mikä suoritukseen otetuista prosesseista saa prosessoriaikaa seuraavaksi ja kuinka paljon.

Rinnakkaisuus aiheuttaa ongelmia toimenpiteestä toiseen vaihdettaessa, toisistaan riippuvien toimenpiteitten synkronointiin ja toisistaan riippumattomien osien väliseen suojaukseen.

Resurssien hallinta ja jakelu

Jaettavia resursseja ovat keskusyksikön osat (prosessori ja keskusmuisti), järjestelmän käyttämät tukimuistit, *yksittäiskäyttöiset oheislaitteet*, esimerkiksi piirturit ja kirjoittimet sekä *yhteiskäyttöiset oheislaitteet*, esimerkiksi levyt ja niillä olevat tiedostot. Yksittäiskäyttöistä oheislaitetta voi käyttää vain yksi prosessi kerrallaan. Yhteiskäyttöisiä laitteita voi käyttää useat prosessit samanaikaisesti.

Käyttöjärjestelmä jakaa näitä resursseja pyydettäessä prosesseille sekä pitää kirjaa resursseista. Se on resurssien jakelupolitiikan toteuttaja ja se päättää kuka saa, koska saa ja paljonko saa. Käyttöjärjestelmän on pystyttävä ratkaisemaan kysymykset 'mitä annetaan?' (allokointi) ja 'milloin annetaan?' (skedulointi).

Resurssien hallintaan ja jakeluun liittyy runsaasti erilaisia kirjanpidollisia tietorakenteita: *kuvaajia*, jotka sisältävät tiedot resursseista ja niiden ominaisuuksista, sekä jonoja, joissa on resursseja odottavien prosessien tietoja. Kirjanpito on poikkeuksetta kaksitasoista: käyttöjärjestelmällä on paitsi globaali näkemys kaikista varatuista ja vapaista resursseista myös kirjanpito kunkin prosessin kuvaajassa siitä, mitä resursseja juuri sille prosessille on annettu.

Käyttöjärjestelmä sallii prosessien käyttää tietyin edellytyksin samoja resursseja samaan aikaan. Kyse on lähinnä tiedon *yhteiskäytöstä*. Useat prosessit käyttävät samoja ohjelmakirjastoja, samoja datatiedostoja ja käyttöjärjestelmän rutiineja. Käyttöjärjestelmä tarjoaa mahdollisuuden suorittaa samaa ohjelmakoodia yhtäaikaisesti

useissa eri prosesseissa. Perusteina resurssien yhteiskäytölle ovat kustannussäästöt, toisten työn hyväksikäyttö, tiedon jakaminen ja päällekkäisyyksien poistaminen (yksi koodi, monta käyttäjää).

Yhteiskäytössä ja rinnakkaisissa toimenpiteissä saatetaan ajautua tilanteeseen, jossa kaksi tai useampia töitä tarvitsee samoja resursseja. Käyttöjärjestelmä valvoo, että yhteiskäyttö sujuu kitkatta eikä siitä aiheudu virheitä, esimerkiksi ettei kaksi eri työn päivityssarjaa mene limittäin.

Palvelutoiminnot prosesseille

Käyttöjärjestelmä tarjoaa joukon palveluja, joiden käyttö on prosessille täysin näkyätöntä tai on selkeää ja liikkuu usein loogisella tasolla, ts. käyttäjän ei tarvitse tietää kuinka palvelu lopulta toteutetaan. Keskeisimmät näistä ovat *siirräntä- ja tiedostojärjestelmäpalvelut.*

Siirräntäjärjestelmä sisältää

- ohjausrutiinit, jotka muuntavat prosessin palvelupyynnöt käyttöjärjestelmän ja laitteiston toiminnoiksi (esim. lukeminen ja kirjoittaminen todellisia levyosoitteita käyttäen),
- saantimenetelmärutiinit, jotka huolehtivat tarvittavista osoitemuunnoksista ja puskuroinneista sekä
- alimman tason laiteajuriohjelmat (engl. driver), jotka ohjaavat yksittäisiä laitteita.

Tiedostojärjestelmän tarjoamia palveluja ovat mm. tiedostojen ja hakemistojen luonti, niiden päivitykset ja poistot, tiedostojen avaamiset, käyttö ja sulkemiset sekä tiedostojen suojausjärjestelmä.

Muita käyttöjärjestelmän prosesseille tarjoamia palveluja ovat mm. *prosessin luonti ja lopettaminen* sekä *keskusmuistitilan varaaminen* ja vapauttaminen.

Käyttöjärjestelmä pyrkii käsittelemään havaitsemiaan virheitä järkevästi, esimerkiksi levyvirheen sattuessa yritetään uudelleen, mutta ohjelmassa esiintyvä nollallajako aiheuttaa työn lopetuksen.

Suojaus

Kaikki koneessa toimivat ohjelmat on suojattava toisiltaan. Erityisen tärkeää on suojata käyttöjärjestelmä sovelluksilta. Käyttöjärjestelmän suojaus sovellukselta voidaan toteuttaa siten, että prosessori voi olla ohjelmaa suorittaessaan kahdessa eri suojaustilassa: *etuoikeutetussa tilassa* (engl. kernel mode, supervisor mode) tai *käyttäjätilassa* (engl. user mode). Kun prosessori on käyttäjätilassa, voi se suorittaa vain 'tavanomaisia' konekielen käskyjä.

Käyttöjärjestelmän keskeisimmät osat suoritetaan yleensä pysyvästi etuoikeutetussa tilassa, jolloin niillä on kaikki oikeudet. Siirtyminen etuoikeutettuun tilaan tapahtuu aina hallitusti palvelupyyntöjen kautta. Palvelupyynnössä prosessori laitetaan etuoikeutettuun tilaan. Tällöin prosessori voi suorittaa myös etuoikeutettuja käskyjä, joita ovat mm. muistin suojaukseen liittyvien rekistereiden asetus, laiteohjaimien rekistereihin viittaaminen (ns. suora I/O) sekä keskeytyksen esto ja salliminen. Myös kaikkiin muistipaikkoihin viittaaminen on sallittua prosessorin etuoikeutetussa tilassa.

Prosessien välinen keskusmuistitilan suojaus tapahtuu laitteistotasolla muistinhallintayksikössä MMU. Muistinhallintayksikkö tarkista osoitemuunnoksen yhteydessä jokaisen ajonaikaisen muistiviittauksen.

Myös yhteisillä oheismuistilaitteilla olevat käyttäjien tiedostot on suojattava. Käyttöjärjestelmän on taattava helppo pääsy niille, joilla on lupa käyttää tiedostoa, mutta sen on myös suojattava tiedostoja väärinkäyttöjä vastaan. Tämän vuoksi esim. suora levyosoitteiden käyttö ei ole sallittu sovelluksille, vaan ne joutuvat suorittamaan tiedostoihin liittyvän siirräntänsä käyttöjärjestelmän palvelupyynnöillä. Tällöin käyttöjärjestelmällä on tilaisuus tarkistaa tiedostojen *käyttöoikeuksiin* (suojaukseen) liittyvät asiat.

Muita vaatimuksia

Käyttöjärjestelmän eikä muiden prosessienkaan läsnäolo ei saa vaikuttaa epädeterministisesti minkään toisen prosessin tuloksiin. Prosessin on tuotettava aina samat tulokset samoilla syötteillä. Käyttöjärjestelmän on myös selviydyttävä prosessien erilaisista suoritusjärjestyksistä. Eri prosesseihin liittyvien tapahtumien on saatava sattua sekalaisessa järjestyksessä.

Lisäksi käyttöjärjestelmän olisi oltava tehokas, luotettava ja helposti ylläpidettävä. Sen tulisi olla kooltaan pieni, eikä se saisi tietenkään sisältää virheitä. Vaikka käyttöjärjestelmä kokoaakin tietoja käyttäjittäin resurssien käytöstä esimerkiksi laskutusta tai järjestelmän suorituskyvyn selvittämistä varten, se ei saa hidastaa tarpeettomasti sovellusten etenemistä.

Luku 8

Prosessien hallinta

Käyttöjärjestelmän keskeisimmät ja laiteläheisimmät osat muodostavat käyttöjärjestelmän *ytimen* (engl. kernel). Sen tehtävänä on luoda turvallinen ympäristö, jossa prosessit voivat syntyä, edetä sopusoinnussa ja kuolla tehtävänsä täytettyään. Tähän ympäristöön kuuluu prosessorin vaihtaminen prosessilta toiselle (vuorottaminen), keskeytysten käsittely sekä prosessien synkronointi (engl. syncronization) ja keskinäinen poissulkeminen (engl. mutual exclusion), eli semaforioperaatiot Up() ja Down().

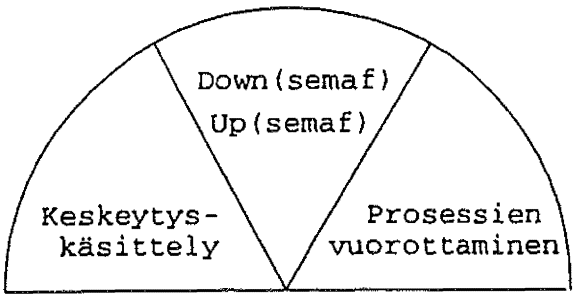
Käyttöjärjestelmän ydin toteutetaan osittain laitteistolla ja mikrokoodilla sekä osittain ohjelmistolla.

Laitteistotasolla on varattava osa käskyistä vain käyttöjärjestelmän käyttöön, jotta ohjelmat eivät pääsisi vaikuttamaan toisiinsa ja pystytään järjestämään suojaus. Näitä *etuoikeutettuja käskyjä* (engl. priviledged) ovat keskeytysten kieltäminen ja salliminen, prosessorin vaihto prosessilta toiselle, muistin suojaukseen vaikuttavien muistinhallintayksikön rekistereiden asetukset ja alimman tason siirräntä. Etuoikeutettuja käskyjä voi käyttää vain käyttöjärjestelmä silloin, kun prosessori on etuoikeutetussa tilassa (ts. on asetettu tilarekisteriin SR etuoikeutetun tilan osoittava bitti pystyyn). Käyttöjärjestelmän ydin ja siirrännän käynnistys suoritetaan aina etuoikeutetussa tilassa. Monet muut käyttöjärjestelmään kuuluvat rutiinit sekä sovellusohjelmat ovat käyttäjätilassa.

Laitteistotasolla on oltava keinot *muistin suojauksen* toteuttamiseksi. Myös osoitemuunnos on voitava tehdä laitteistotoimintona. Muistinhallintayksikön on huomattava viittausyritykset 'naapurin tontille'.

Virhetilanteissa ja keskeytysten yhteydessä on asetettava *keskeytyssyy* johonkin tilarekisterin SR lipukkeeseen. Tämän jälkeen on prosessorin osattava automaattisesti tallettaa vähintään PC:n arvo muistiin ja ladattava PC:lle uudeksi arvoksi käsittelyrutiinin alkuosoite.

Prosessien vuorottamista ja laitteiston käytön laskutusta varten tarvitaan aikaviipalekello ja reaaliaikakello. Aikaviipalekellon on tuotettava säännöllisin välein kellokeskeytyksiä, jolloin käyttöjärjestelmälle tarjoutuu tilaisuus vaihtaa suoritetta-



Kuva 8.1: Käyttöjärjestelmän ytimen osat

vaa prosessia.

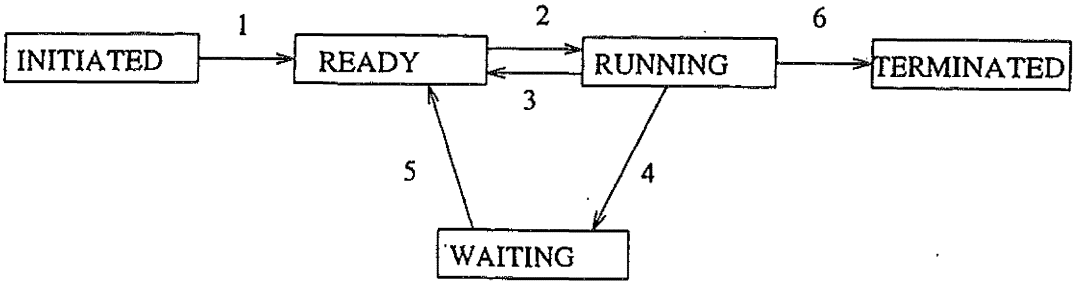
8.1 Prosessi

Keskusmuistiin suoritettavaksi otettua ohjelmaa kutsutaan prosessiksi. Kun käyttöjärjestelmä asettaa prosessorin suorittamaan prosessin koodia, se saa aikaan muutoksia datassa. Käyttöjärjestelmän kannalta prosessi on tietorakenne, johon liittyy keskusmuistissa olevat koodi ja data sekä joukko hallinnollista tietoa.

Käyttöjärjestelmä ylläpitää prosesseihin liittyviä tietoja *prosessitaulussa*, jossa on jokaiselle yksittäiselle prosessille oma alkio, *prosessin kuvaaja* (engl. process control block, PCB). Prosessin kuvaajaan talletetun tiedon perusteella käyttöjärjestelmä voi väliaikaisesti nukuttaa ohjelman suorituksen ja myöhemmin käynnistää sen jatkamaan siitä mihin se edellisellä kerralla jäi. Prosessin kuvaaja sisältää ainakin prosessien tunnistamisessa, vuorottamisessa, prosessin muistivarausten hallinnassa, tiedostojen käsittelyssä ja siirrännässä tarvittavia tietoja.

Prosessin *tunnisteena* on tavallisesti yksikäsitteinen numero ja prosessin omistajan käyttäjä- ja ryhmätunniste (engl. user id ja group id). Omistajan tunnistetiedot on saatu järjestelmään kirjoittauduttaessa ja prosessit numeroidaan juoksevasti. Omistajan tunnistetietoja tarvitaan ainakin tiedostojen käyttöoikeuksien tarkistamiseen. Prosessin tunnisteen avulla omistaja voi vaikkapa lopettaa prosessin suorituksen antamalla sopivan komennon näppäimistöltä.

Prosessien *vuorottamista* varten ajokelpoisista prosesseista (tai oikeammin niiden kuvaajista) on muodostettu READY-jono. Myös *prosessin tila* on merkitty prosessin kuvaajaan. Tilana on RUN, kun prosessi on suorituksessa prosessorissa. Tilana on READY, kun prosessi on jonossa odottamassa, että käyttöjärjestelmä valitsisi sen suoritettavaksi. Se voisi siis edetä, kunhan saisi vain prosessorin käyttöönsä. Prosessin tilana on WAIT, kun se ei voi edetä ennenkuin joku tietty tapahtuma tapahtuu. Kuviossa 8.2. näkyvät prosessin tilat ja mahdolliset siirtymät tilojen välillä.



Kuva 8.2: Prosessin tilasiirtymäkaavio.

Kaavioon on piirretty täydellisyyden vuoksi myös tilat INITIATED ja TERMINATED. Edellinen tarkoittaa, että prosessi on luotu, mutta se ei ole vielä valmis ajettavaksi. Jälkimmäisessä tapauksessa prosessi on päättänyt tehtävänsä normaalisti tai se on keskeytetty virheen johdosta.

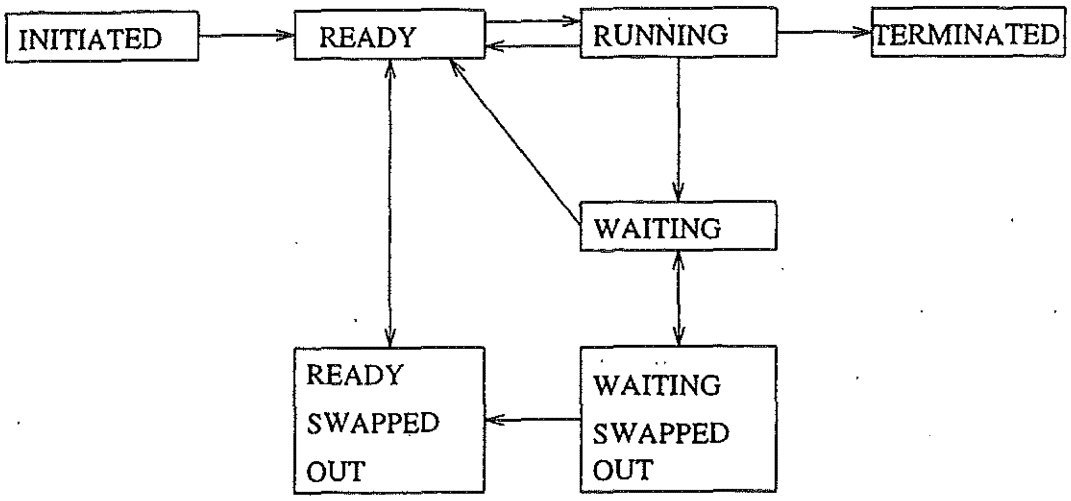
Siirtymien merkitys on seuraava:

1. Kaikki, mitä prosessi tarvitsee ajoa varten, on valmista.
2. Prosessi saa CPU:n.
3. Aikaviipale on täyttynyt.
4. Prosessi ei voi jatkaa, koska se odottaa I/O-operaation päättymistä.
5. I/O-operaatio on päättynyt.
6. Prosessi on suoritettu tai fataali virhe on tapahtunut.

Jos käyttöjärjestelmä on sellainen, että se voi viedä kokonaisia prosesseja levylle kesken suorituksen (swapping), kaavioon voidaan lisätä kaksi uutta tilaa, kuten kuviossa 8.3.

Siihen, mihin kohtaan jonoa prosessi joutuu, ja samalla myös siihen, kuinka pian prosessi saa prosessorin käyttöönsä, vaikuttaa prosessin kiireellisyysaste eli *prioriteetti*. Tavallisesti käyttäjien prosesseilla on pienempi prioriteetti kuin järjestelmän toteutukseen kuuluvilla prosesseilla. Prioriteetti voidaan esittää kokonaislukuarvona esimerkiksi siten, että mitä pienempi tuo arvo on sitä suurempi on prosessin prioriteetti.

Kun prosessi menettää suoritusvuoronsa prosessorissa ja sen tilalle tulee uusi, on väistyvän prosessin rekistereihin tekemät muutokset talletettava keskusmuistiin. Mikä olisikaan luontevampaa kuin varata tuo tallealue rekistereille prosessin kuvaajasta. Kun prosessi taas aikanaan saa seuraavan suoritusvuoron, palauttaa käyttöjärjestelmä arvot takaisin prosessoriin, ja prosessi jatkaa siitä mihin se edellisellä kerralla jäi.



Kuva 8.3: Prosessin tilat heittovaihdon tapauksessa

Kun prosessi tai sen osia tuodaan muistiin, kirjaa käyttöjärjestelmä prosessin kuvaajaan, missä prosessin koodi ja data sijaitsee. Prosessin kuvaajassa on joko alueen alkuosoite ja pituus tai virtuaalimuistia käytettäessä sivutaulun sijainti.

Myös prosessin avaamista tiedostoista pidetään kirjaa prosessin kuvaajassa. *Tiedostokuvaajataulusta* (engl. file descriptor) on viitteitä muihin tiedostonhallinnassa tarvittaviin käyttöjärjestelmän rakenteisiin. Myös prosessin *työhakemiston* polku on tapana kirjata prosessin kuvaajaan.

Edellä lueteltujen tietojen lisäksi prosessin kuvaajassa voi olla esimerkiksi semaforeihin ja siirräntään liittyviä tietoja ja vaikkapa laskutusta varten kulutettu prosessoriaika, siirräntään käytetty I/O-aika tai muita aikaleimoja.

Myös käyttöjärjestelmä on joukko järjestelmään kuuluvia prosesseja. Se on vastuussa järjestelmän kaikista muista prosesseista: Käyttöjärjestelmä luo ja tuhoaa sovellusprosesseja sekä käynnistää ja pysäyttää niiden suorituksen. Käyttöjärjestelmä jakaa prosessorin aikaa vuorotellen aikaviipaleina kaikille prosesseille sekä tarjoa palvelunaan prosesseille oheislaitteiden ja tiedostojen käytön.

Useat järjestelmät sallivat, että prosessi voi edelleen generoida uusia (lapsi)prosesseja. Tällöin uusien prosessien generointi on kuvattava ohjelmakoodissa ja on pyydettävä käyttöjärjestelmältä tätä luontipalvelua. Myös näihin uusiin prosesseihin liittyy omat koodinsa ja tietorakenteensa. Tehtävän jakaminen rinnakkaisiin prosesseihin on usein ongelman ratkaisun kannalta selkeää, sillä monet tehtävät ovat jo loogisesti rinnakkaisia.

8.2 Keskeytyskäsittely

Sekä laitteisto että käyttöjärjestelmä muodostuvat useista toisistaan riippumattomasti toimivista osista. Tällaisia laitteiston osia ovat esimerkiksi prosessorit, oheislaitteet ja kello. Oheislaitteiden käynnistys tapahtuu ohjelmallisesti käyttöjärjestelmän laiteajurilla, jonka jälkeen ohjain ja laite toimivat prosessorista riippumatta. Laiteohjain vaatii uudelleen prosessorilta ohjelmallista käsittelyä, kun esimerkiksi siirräntä päättyy tai syntyy jokin virhetilanne. Ohjain kertoo tarpeestaan prosessorille keskeytyksen avulla.

Esimerkkejä

- osituskäyttöjärjestelmän *aikaviipalekeskeytys*: Kullekin prosessille annetaan prosessori tietyksi aikaviipaleeksi. Annettu aikaviipale asetetaan kelloon. Aikaviipaleen päätyttyä kello aiheuttaa keskeytyksen ja seuraava prosessi saa prosessorin käyttöönsä.
- sunnuntai-ohjelmoijan ontuvien algoritmien aiheuttamat *poikkeustilanteet*: aritmetiikan ylivuoto, nollallajako, tuntematon tai luvaton käskykoodi, luvaton muistiosoite. Nämä poikkeustilanteet aiheuttavat ohjelman päättymisen.
- ohjelman tekemä käyttöjärjestelmärutiinin kutsu eli *palvelupyyntö* (tavallisimmin siirräntä). Kun rutiiniin siirrytään keskeytyksen kautta, päästään käyttöjärjestelmän etuoikeutettuun tilaan. Tavallisilla prosesseillahan ei oikeutta käyttää suoraan käyttöjärjestelmän alueelle kuuluvia muistiosoitteita.

Prosessori on saatava poikkeus- ja keskeytystapauksissa suorittamaan käyttöjärjestelmän koodia, jotta käyttöjärjestelmä voi tehdä tilanteen vaatimat toimenpiteet.

Keskeytyspyyntö

Ohjain (ulkoiset keskeytykset) tai ohjelma (sisäiset keskeytykset) aiheuttaa keskeytyksen merkitsemällä keskeytyspyynnön tilarekisterin SR sopivaan lipukkeeseen. Koska prosessori tutkii jokaisen suorittamansa konekäskyn jälkeen lipukkeita, se huomaa keskeytyksen.

Siirtyminen käyttöjärjestelmään

Keskeytyskäsittelyn aluksi laitteisto / mikrokoodi tallettaa prosessorista rekistereiden SR, PC ja FP arvot prosessin omaan pinoon, jotta keskeytyvän prosessin suoritusta voidaan jatkaa myöhemmin. Prosessin kuvaajaan tallettaminen ei ole mahdollista, sillä prosessori ei tiedä sen sijaintia.

Seuraavaksi asetetaan prosessori etuoikeutettuun tilaan ja uudet keskeytykset estetään ainakin kiireisimpien vaiheiden ajaksi. Tämä tapahtuu nostamalla tilarekisteristä pari bittiä pystyyn (TTK-91:ssä bitit P = priviledged mode ja D =

interrupts disabled). Kun prosessori on etuoikeutetussa tilassa, se suostuu suorittamaan myös pelkästään käyttöjärjestelmälle tarkoitettuja konekäskyjä. Keskeytysten esto puolestaan tarkoittaa yksinkertaisesti sitä, että prosessori ei silloin tutki kaikkia lipukkeita.

Prosessori selvittää keskeytyksen syyn tutkimalla *lipukkeita* ja tarvittaessa *kyselemällä* laiteohjaimilta (pollaus). Kun syy on selvillä ja siihen on liitetty joku numero, lataa prosessori ko. keskeytyksen käsittelijän ensimmäisen käskyn muistiosoitteen PC:n arvoksi keskusmuistissa olevasta *keskeytysvektoritaulusta* (engl. interrupt vector), eli PC := MEM[nro]. Kun prosessori nyt hakee seuraavan käskyn, tämä käsky kuuluukin jo käyttöjärjestelmän koodiin. Käsittelyrutiini tallettaa tarvittaessa muiden rekistereiden arvoja prosessin pinoon, ja suorittaa tarvittavat toiminnot (esimerkiksi siirtää jonkin prosessin READY-tilaan) tai käynnistää esimerkiksi ajuriprosessin tekemään tarvittavaa palvelua.

Takaisin sovellukseen

Kun keskeytys on käsitelty, käsittelyrutiini palauttaa pinoon talletettujen rekisterien arvot (myös PC) takaisin prosessoriin ja prosessori jatkaa keskeytynyttä ohjelmaa.

Keskeytyskäsittely voi päättyä myös siten, että sen jälkeen ei jatketakaan keskeytynyttä prosessia, vaan valitaan suoritukseen joku toinen prosessi. Keskeytyksen vuoksihan juuri suorituksessa olleen prosessin tilaksi on voinut tulla WAIT, sille annettu aikaviipale on täyttynyt tai toinen prioriteetiltaan suurempi prosessi tulee taas ajokelpoiseksi. Jos prosessia vaihdetaan, on edellisen prosessin tiedot talletettava prosessorista ja pinosta prosessin kuvaajaan. Ja sen jälkeen siirrytään vuorottajaan. Vuorottaja valitsee jonkin READY-jonossa olevan työn seuraavaksi prosessorin käyttäjäksi ja lataa prosessorin rekistereille arvot valitun prosessin kuvaajasta.

Keskeytysmekanismi on niin näppärä keino, että sitä käytetään myös käyttöjärjestelmän palvelujen käynnistämiseksi. Palvelupyyntö vastaa muuten normaalia lausekielen aliohjelmakutsua - sieltä palataan aikanaan takaisin kutsukohtaan, mutta käsittely tapahtuu käyttöjärjestelmässä. Jos palvelu vie kauan (esim. siirräntä), voi sovellus joutua odottamaan WAIT-tilassa ja prosessori voi suorittaa muita prosesseja välillä. Palvelun valmistuttua sitä pyytänyt prosessi pääsee taas READY-tilaan ja sieltä aikanaan taas suoritukseen. Keskeytykset on jaettu usein kiireellisyysluokkiin (esim. konevirhe - siirräntävirhe - kellolaite - palvelupyyntökeskeytys). Kiireisempi keskeytys voi aiheuttaa muutoksia palveluun keskeyttämällä vähemmän kiirellisen keskeytyksen käsittelyn. Keskeytyksen käsittely kieltää automaattisesti käsiteltävän luokan ja sitä vähemmän kiireellisten keskeytysten palvelun.

Koska keskeytyskäsittely suoritetaan etuoikeutetussa tilassa keskeytykset kokonaan tai osittain estettynä, tulee käsittelyrutiinin tehdä vain välttämätön ensiapu ja jättää muut toimenpiteet käyttäjätilassa pyörivälle, mutta käyttöjärjestelmään kuuluvalle palvelurutiinille.

8.3 Prosessien vuorottaminen

Vuorottaja (engl. scheduler) jakaa prosessoriaikaa järjestelmän eri prosesseille. Vuorottajan koodiin tullaan keskeytyskäsittelyn jälkeen aina, kun suorituksessa olleen prosessin kyky käyttää prosessoria on muuttunut. Syynä tähän voi olla aikaviipaleen täyttyminen tai prosessi jää odottamaan jotain tapahtumaa.

Prosessin tilakaaviossa ainostaan siirtymä 2 (ready $--> $ run) on mielenkiintoinen, sillä se voidaan toteuttaa monella eri tavalla. Yleensä useita prosesseja on ready-tilassa yhtäaikaa ja vuorottajan tehtävänä on valita seuraava prosessi suoritukseen. Tässä voidaan käyttää useita erilaisia kriteereitä:

- *Läpimenotiheys* (throughput) on niiden prosessien keskimääräinen lukumäärä, jotka päättyvät valitussa aikayksikössä. Suureella on merkitystä systeemin tuottavuuden tarkastelussa, mutta siihen tulee suhtautua varauksella. (Jos useimmat prosessit lopetetaan laitevian johdosta pian alkamisen jälkeen, läpimenotiheys on varsin suuri!)
- *Läpimenoaika* (turnaround time) on keskimääräinen aika, joka prosessilta kuluu suorituksen alusta loppuun asti. Tällä suureella on merkitystä eräajosysteemien käyttäjille.
- *Vasteaika* (response time) on keskimääräinen aika, joka kuluu syötteen kirjoittamisesta päätteelle ensimmäiseen prosessin reaktioon tähän syötteeseen. Tällä on merkitystä moniajosysteemin suorakäytössä.
- *Odotusaika* (wait time) tarkoittaa keskimääräistä ready- ja wait-tilassa vietettyä aikaa.

Nämä suureet eivät ole riippumattomia toisistaan. Toisaalta käytännössä on vaikeaa suunnitella systeemi niin, että kaikki edellä esitetyt muuttujat optimoitaisiin samanaikaisesti. Nykyään tavallisimmin minimoidaan odotusaikaa. Se ei riipu pelkästään laitteiston nopeudesta, kuten seuraava esimerkki valaisee.

Oletetaan, että 6 prosessia on ready-jonossa. P1, ... ,P5 tarvitsevat aikaa suoritukseen yhden minuutin, P6 55 minuuttia. Jos suoritusjärjestys on P1, P2, ... ,P6, keskimääräinen odotusaika on

$$\frac{1}{6}(0+1+2+3+4+5) = 2,5 \text{ minuuttia.}$$

Jos taas suoritusjärjestys on P6, P1, ... ,P5, keskimääräinen odotusaika on

$$\frac{1}{6}(0+55+56+57+58+59) = 47,5 \text{ minuuttia.}$$

Käyttäjärjestelmä voi vaikuttaa sekä ready- että wait-tilassa vietettyyn aikaan. I/O-aikaan voidaan vaikuttaa *tiedon puskuroinnilla* ja tehokkailla *laiteajureilla*. Ready-jonossa vietetyn ajan ratkaisee vuorottaja. Tarkastellaan seuraavassa jälkimmäistä tapausta.

Probleema on siis seuraava. Prosessori on juuri vapautunut joko I/O-operaation tai aikaviipaleen täyttymisen johdosta. Ready-jonossa on n prosessia. Kuinka valitaan prosessi, joka saa prosessorin seuraavaksi.

Ongelmaa voidaan yrittää ratkaista seuraavilla tavoilla:

- *Round Robin.* Ready-jono on todella jono konkreettisessa mielessä. Kun prosessi vaihtaa tilakseen ready, se asetetaan jonon loppuun. CPU:n saa jonon ensimmäinen prosessi. Periaate on helppo toteuttaa tehokkaasti, eikä mikään prosessi *nälkiinny* eli jää jonoon loputtomasti. Ratkaisevana haittana on oletus, että prosessit ovat samanarvoisia.
- *Prioriteettijono.* Jokaisella prosessilla on prioriteetti, joka on kokonaisluku. Prosessit ovat jonossa prioriteettin mukaisessa järjestyksessä. Eli jos Q on jonossa Q':n edellä, niin välttämättä Q:n prioriteetti p(Q) on suurempi tai yhtäsuuri kuin Q':n.

  Prioriteettijono voidaan toteuttaa listana tai taulukkona. Tällöin seuraavan suoritusvuoron saavan prosessin valinta on yhtä helppoa kuin round robinissa. Uuden prosessin lisääminen jonoon vaatii kuitenkin vähintään ajan $\log(n)$, missä n on jonon prosessien lukumäärä.

  Jos $\log(n)$ on liian paljon, voidaan jokaista prioriteettia kohti varata oma jono edellyttäen, ettei prioriteetteja ole liian paljon. Yksittäiset jonot toteuttavat round robin-periaatteen. Kun CPU vapautuu, seuraavaa suoritettavaa prosessia etsitään korkeimman prioriteetin omaavasta jonosta, sen jälkeen sitä seuraavsta jne kunnes löytyy epätyhjä jono.

Ongelmaksi jää, miten prioriteetit valitaan prosesseille. Onko olemassa objektiivisia kriteereitä, joiden perusteella joillekin prosesseille annetaan korkea ja joillekin alhainen prioriteetti? Ratkaisua auttaa kaksi avainhavaintoa:

1. Jos pienille prosesseille annetaan korkea ja suurille alhainen prioriteetti, minimoituu odotusaika, kuten nähtiin kuuden prosessin tapauksessa aikaisemmin.
2. Paljon I/O-toimintoa sisältävät prosessit kuluttavat suurimman osan ajasta wait- ja ready-tilassa. Antamalla tällaisille korkea prioriteetti, saadaan ne pois keskusmuistista nopeasti.

Ensimmäinen havainto johtaa periaatteeseen *Shortest Job First* eli SJF. Tällöin prosessin Q prioriteetti p(Q) lasketaan kaavasta

$$p(Q) = K/l(Q),$$

missä $l(Q)$ on Q:n seuraavan CPU-vaiheen pituus ja K on vakio. Todistetaan nyt, että SJF minimoi keskimääräisen odotusajan.

Valitaan ready-jonosta kaksi prosessia P ja Q. Oletetaan, että P on jonossa ennen Q:ta ja $l(P) > l(Q)$. Näytetään, että vaihtamalla P:n ja Q:n paikkaa keskimääräinen odotusaika pienenee. Tästähän seuraa, että SJF minimoi odostusajan.

Vaihtaminen ei vaikuta seuraavien prosessien odotusaikoihin:

- ennen P:tä olevat prosessit,
- Q:n jäljessä olevat prosessit.

Olkoot R1,...,Rk P:n ja Q:n välissä olevat prosessit. P:n, Q:n ja prosessien R1,...,Rk odotusajat muuttuvat seuraavasti:

- Q:n uusi aika on P:n vanha aika,
- prosessien R1,...Rk odotusaika vähenee määrällä $l(P) - l(Q)$,
- P:n uusi aika on Q:n vanha aika miinus $l(P) - l(Q)$.

Kaikkien prosessien yhteenlaskettu odotusaika vähenee siten määrällä

$$(k+1) * (l(P) - l(Q)),$$

joten väite on todistettu.

SFJ:n ongelmana on, ettei tunneta pituuksia $l(P)$. Niinpä $l(P)$:t voidaan yrittää arvata P:n aikaisemman käyttäytymisen perusteella. Yleensä käytetään aikaisempien CPU-vaiheiden painotettua summaa. Käyttöjärjestelmä kirjaa jokaiselle prosessille edellisen painotetun summan $ws(P)$. Kun uusi CPU-vaihe päättyy ajan l kuluttua, uusi arvo lasketaan kaavalla

$$ws(P) = \alpha * ws(P) + (1 - \alpha) * l,$$

missä α on painokerroin (vakio), $0 \leq \alpha \leq 1$. Jos α valitaan lähelle ykköstä, painotetaan mennyttä käyttäytymistä. Jos taas α on lähellä nollaa, painotetaan viimeaikaista käyttäytymistä. Tyypillisesti $\alpha = 0,5$. Prioriteetti lasketaan nyt kaavalla

$$p(P) = K/ws(P).$$

Prioriteettien ongelma on, että alhaisen prioriteetin prosessit saattavat nälkiintyä. Tämän ongelman ratkaisemiseksi voidaan soveltaa jompaa kumpaa seuraavista säännöistä:

1. Jos prosessilla on juuri ollut CPU käytössään, sen prioriteettia lasketaan niin, ettei se saa CPU:ta liian nopeasti uudestaan.
2. Kun prosessi vapauttaa CPU:n, odottavat prosessit saavat hyvityksen.

Menetelmä 1 on helppo toteuttaa, sillä muutos kohdistuu vain yhteen prosessiin. Menetelmässä 2 täytyy sen sijaan koko jono (kaikki jonot) käydä läpi. Valitettavasti menetelmä 1 ei takaa, etteikö prosessi voi nälkiintyä, jos järjestelmään tulee koko ajan korkean prioriteetin prosesseja. Menetelmä 2 takaa jokaiselle prosessille tilaisuuden ainakin joskus päästä run-tilaan.

Käytännössä täytyy soveltaa molempia menetelmiä, koska menetelmä 1 laskee prioriteetteja monotonisesti ja menetelmä 2 vastaavasti nostaa prioriteetteja. Molemmat ilmiöt ovat epätyydyttäviä. Soveltamalla kumpaakin periaatetta saadaan prioriteetista dynaaminen suure, joka pienenee jokaisen CPU-vaiheen jälkeen ja kasvaa odotusaikana.

8.4 Prosessien välinen synkronointi ja poissulkeminen

Prosessien keskinäinen eteneminen on epädeterminististä. Etukäteen ei voida tietää, missä järjestyksessä prosessorissa suoritettavat prosessit ohittavat tietyt ennalta valitut tarkkailupisteet.

Prosessien yhteistoiminta edellyttää kuitenkin, että tietyissä tapauksissa etenemisjärjestys on hallittua. Esimerkiksi käyttäjän laskentaprosessi ei voi edetä, ennen kuin lukijaprosessi on saanut laskennassa käytettävät tiedot luettua. Tai prosessi ei voi käyttää yhdelle käyttäjälle kerrallaan varattavaa resurssia ennenkuin edellinen käyttäjä on vapauttanut resurssin. Yhteiskäytössä olevien resurssien hallinta edellyttää prosessien välistä synkronointia. Vastaavasti yksittäiskäyttöisten resurssien käyttö edellyttää prosessien välistä poissulkemista.

Esimerkki Paikanvaraus. Jos suoritettavaa 'prosessia' vaihdetaan sopivasti, tulee paikka 76 varattua kahdesti.

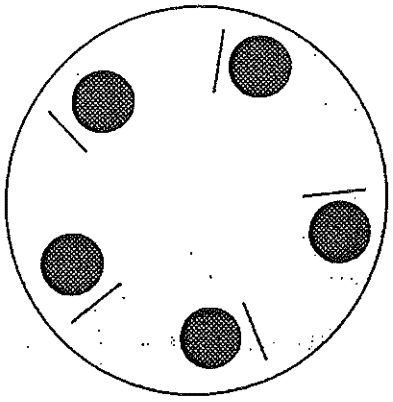
```
A näkee, että lennon XY
paikka 76 on vapaa
        .
        .                          B näkee, että lennon XY
        .                          paikka 76 on vapaa
A varaa paikan 76                       .
        .                               .
                                   B varaa paikan 76
```

Jos prosessi voi varata osan resursseista ja jää odottamaan, että resursseja vapautuisi lisää, voi syntyä *lukkiutumistilanne* (engl. deadlock). Lukkiutumistilanteessa kaksi tai useampia prosesseja varaa toistensa haluamia resursseja sopivasti ristiin ja jää odottamaan, että joku vapauttaisi etenemiseen tarvittavat lisäresurssit. Ääritilanteessa joku yksittäinen prosessi voi joutua odottamaan ikuisesti resurssin vapautumista ja nälkiintyy (engl. starvation). Kuvassa 8.4 on esitetty perinteinen esimerkki lukkiutumisesta ja nälkiintymisestä.

Prosessit eivät voi siis toimia täysin toisistaan riippumatta. Prosessien hallittuun yhteistoimintaan tarvitaan mekanismi, jolla joku prosessi voidaan tarvittaessa nukuttaa odottamaan toisen prosessin etenemistä tiettyyn pisteeseen ja herättää uudelleen.

Semaforit

Keskeisin käsite prosessien kommunikointimekanismin toteutuksessa on semafori (engl. semaphore). Se on laskuri, johon liittyy odotusjono ja alkuarvo. Laskurin alkuarvona on tavallisesti jaettavien resurssien lukumäärä ja jono on tyhjä. Resurssien lukumäärä on itseasiassa samalla etenemislupien lukumäärä. Kun laskurin arvo putoaa nollaan, ei resurssia janoava prosessi saa edetä ennenkuin joku



Kuva 8.4: Aterioivat filosofit. Pyöreän pöydän filosofeilla on kullakin spagettilautasen lisäksi vain yksi haarukka. Syömiseen he tarvitsevat kuitenkin kaksi haarukkaa. Lukkiutuminen: jokainen filosofi nappaa oman haarukkansa ja jää odottamaan, että vierestä vapautuisi toinen haarukka. Nälkiintyminen: filosofi joutuu katselemaan vieresta, kun vasemmalla tai oikealla puolella oleva vierusfilosofi syö jatkuvasti.

vapauttaa resurssin. Varattaessa resurssi vähennetään semaforin laskuria ja resurssia vapautettaessa laskuria kasvatetaan. Resurssin varaaminen ja vapauttaminen, ja samalla myös prosessin etenemisen pysäyttäminen ja jatkaminen, tehdään aina käyttöjärjestelmän palveluilla Down(semafori) ja Up(semafori).

Down(semafori)

```
jos     semaforin laskuri on nolla
niin    jää jonoon odottamaan eli
        - liitä prosessi semaforin odotusjonoon ja
        - merkitse prosessin tilaksi WAIT
muuten
        varaa resurssi vähentämällä semaforin laskuria yhdellä
        jatka etenemistä
```

Up(semafori)

```
jos     semaforin jono on tyhjä
niin    vapauta resurssi kasvattamalla semaforin laskuria yhdellä
muuten
        vapauta ensimmäinen semaforin odotusjonosta eli
        - merkitse prosessin tilaksi READY
        - liitä prosessi vuorottajan READY-jonoon
jatka etenemistä
```

Down()-operaatio siis joko vähentää vapaiden resurssien lukumäärää yhdellä, tai nukuttaa operaatiota tekevän prosessin. Up()-operaatiota suorittava prosessi puolestaan päästää yhden etenemislupaa odottavan prosessin READY-tilaan tai kasvattaa vapaana olevien resurssien lukumäärää. Prosessi ei voi koskaan nukahtaa Up()-operaatioon.

Esimerkki: Poissulkeminen. Kaksi prosessia käsittelee yhteisellä muistialueella olevaa tietorakennetta kuten pinoa. Prosessi VIE vie uuden alkion pinon päälle ja prosessi HAE ottaa sen pinon päältä.

```
Prosessi 'VIE':          Prosessi 'HAE':

LOAD  R1,X               LOAD  R1,top
STORE R1,@top            SUB   R1,=1
LOAD  R2,top             STORE R1,top
ADD   R2,=1              LOAD  R2,@top
STORE R2,top             STORE R2,Y
```

Jos aikaviipale täyttyy, kun VIE-prosessi on ehtinyt viedä uuden alkion pinoon, mutta ei ole vielä korjannut top-laskuria (ts. keskeytys tulee LOAD- ja ADD-käskyjen välissä), voi top-laskuri mennä sekaisin. Kun VIE-prosessi pääsee seuraavan kerran suoritukseen, se käyttää R2:ssa olevaa arvoa, vaikka HAE-prosessi olisikin käynyt muuttamassa top-laskurin arvoa. Tietorakenne on suojattava siten, että molemmat prosessit eivät voi olla yhtäaikaa käsittelemässä sitä.

Ratkaisu: Semafori mutex, jonka laskurin alkuarvo 1 ja siihen liittyvä odottavien prosessien jono tyhjä. Kun VIE-prosessi menee käsittelemään yhteistä aluetta, se varaa resurssin vähentämällä semaforin mutex laskurin arvoa yhdellä. Kun nyt tulee keskeytys käskyjen LOAD ja ADD välissä, ei prosessi HAE pääsee yhteiseen rakenteeseen käsiksi. Se joutuu down(mutex) -operaatiota tehdessään WAIT-tilaan ja semaforin jonoon odottamaan, että VIE-prosessi antaa sille jälleen etenemisluvan operaatiolla up(mutex).

```
Prosessi 'VIE':          Prosessi 'HAE':

down(mutex)              down(mutex)
LOAD  R1,X               LOAD  R1,top
STORE R1,@top            SUB   R1,=1
LOAD  R2,top             STORE R1,top
ADD   R2,=1              LOAD  R2,@top
STORE R2,top             STORE R2,X
up(mutex)                up(mutex)
```

Esimerkki: Synkronointi. Prosessi B ei saa edetä pitemmälle kuin L2 ennenkuin prosessi A on päässyt kohtaan L1. Ratkaisu: Semafori proceed, jolle alkuarvo 0 ja siihen liittyvä odottavien prosessien jono tyhjä.

```
    Prosessi A:                    Prosessi B:
    ...
    lue(puskuriin)                 ...
L1: up(proceed)      ------> L2:   down(proceed)
    ...                            ota(puskurista)
                       ...
```

Esimerkki: Synkroninen kohtaaminen. Kumpikaan prosessi ei saa edetä ennenkuin toinen on ehtinyt sopivaan kohtaan. Tässä esimerkkinä tavallinen puhelinyhteys. Ratkaisu: Semaforit proceed-A ja proceed-B, alkuarvot kuten edellä.

```
Prosessi A:                              Prosessi B:
...                                      ...
Valitse numero puhelimesta               Odota puhelua
up(proceed-B)      ------------->        down(proceed-B)
Odota, että kaveri vastaa                Vastaa puhelimeen
down(proceed-A)  <-------------          up(proceed-A)
Puhu, puhu, puhu                         Puhu, puhu, puhu
```

Up()- ja Down()-operaatiot on toteutettu käyttöjärjestelmän ytimessä, koska kaikkien prosessien (erityisesti käyttöjärjestelmän) on voitava käyttää niitä. Down()-operaatio voi aiheuttaa prosessin joutumisen odotustilaan (WAIT), jolloin on vaihdettava suoritettavaa prosessia. Lisäksi Up() on selkeä tapa herättää odottava prosessi. Myös keskeytysrutiinit käyttävät Up()-operaatiota.

Down()- ja Up()-operaatioiden toteutusten tulee olla jakamattomia: vain yksi prosessi saa kerrallaan suorittaa Down()- tai Up()-operaatiota. Yksiprosessorikoneissa tämä toteutetaan kieltämällä keskeytykset kriittisimpien vaiheiden ajaksi ja moniprosessorikoneissa pitämällä väylä varattuna.

Muita prosessien välisiä kommunikointitapoja ovat sanomanvälitys, monitorit sekä yksityiset semaforit. Niiden käyttöön ei tutustuta vielä tällä kurssilla.

Luku 9

Muistinhallinta

Moniajojärjestelmässä keskusmuisti on jaettu usean prosessin kesken. Koska koodausvaiheessa ei voida ennalta tietää, missä osassa muistia prosessia kulloinkin tullaan suorittamaan, ei ohjelmakoodissa voi käyttää fyysisiä muistiosoitteita, vaan käytetään siirtymiä koodin alun suhteen. Jos prosessille varattaisiin kiinteä muistialue koko sen suorituksen ajaksi, voisi ajomoduulien suhteelliset osoitteet muuttaa fyysisiksi osoitteiksi jo prosessin latausvaiheessa. Näin todella tehtiin ensimmäisissä moniajojärjestelmissä.

Nykyisissä järjestelmissä käyttöjärjestelmä voi siirrellä prosessia ajonaikana muistialueelta toiselle tai jopa levylle sen vaikuttamatta millään tavalla prosessin suoritukseen. Prosessin tai sen osien siirtelyyn on syynä muistinkäytön tehostaminen. Tämä ns. vapaasijoitteisuus edellyttää fyysisten muistiosoitteiden ajonaikaista laskemista.

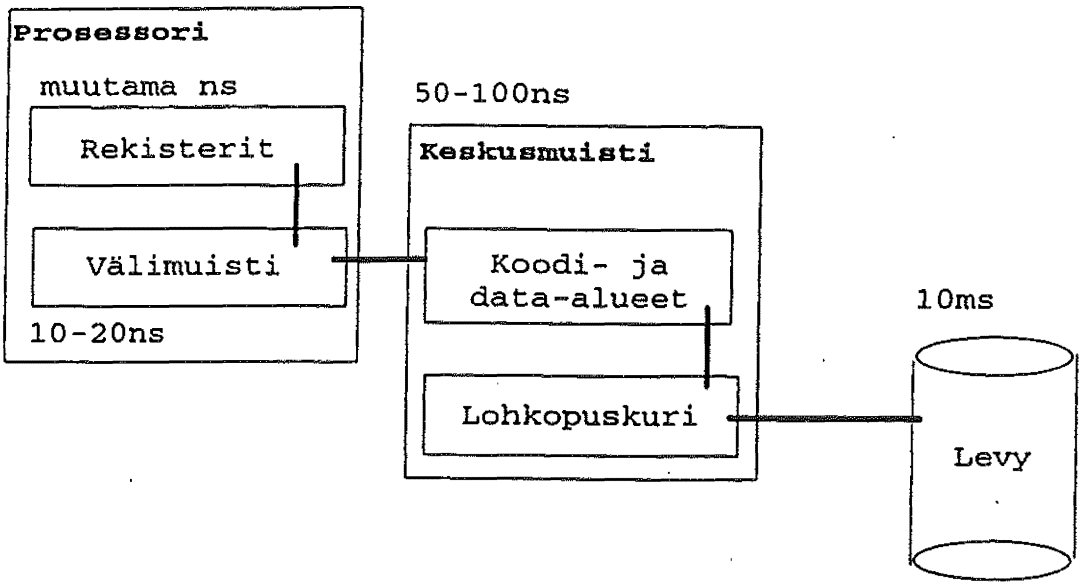
Osoitemuunnoksen lisäksi muistinhallinnan on huolehdittava siitä, että tarvittavat tiedot on tuotu levyltä keskusmuistiin. Muistin *kerrostuksessa* (engl. overlay) tehtävä jäi ohjelmoijan ongelmaksi: ohjelmassa kerrottiin mitä osia kulloinkin tarvittiin keskusmuistissa. Kerrostus on jo vanhentunut menetelmä - nykyaikaisissa järjestelmissä käyttöjärjestelmä huolehtii automaattisesti siitä, että tarvittavat osat ovat keskusmuistissa.

Koska muistissa on samanaikaisesti useiden eri käyttäjien prosesseja, on järjestelmän pidettävä huoli myös siitä, että ne eivät pääse vaikuttamaan toisiinsa. Osoitemuunnoksen lisäksi on kaikkien muistiviittausten oikeellisuus vielä tarkistettava.

Muistinhallinta on käyttäjälle täysin näkymätön toiminto. Käyttäjän ei tarvitse missään vaiheessa tietää missä osassa muistia hänen prosessinsa on tai miten osoitemuunnokset tapahtuvat.

9.1 Muistin organisointi

Perinteisillä tietokoneilla on lineaarinen osoiteavaruus, jossa muistiosoitteet on numeroitu peräkkäin nollasta alkaen. Pienin osoitettava osa on tavun tai sanan mittai-



Kuva 9.1: Muistihierarkia. Levyltä siirretään kokonainen levylohko (esim. 1024 tavua) keskusmuistiin, sieltä siirretään tietoa prosessin data-alueelle pienempinä yksiköinä. Kun prosessori tarvitsee muistissa olevaa tietoa, se tarkistaa ensin, onko tieto välimuistissa. Jos tarvitaan muistinouto, päivitetään samalla välimuistin sisältöä.

nen muistipaikka. Järjestelmän kirjanpidon (vapaat / varatut osat) helpottamiseksi jaetaan osoiteavaruus usein yksittäistä muistipaikkaa suurempiin segmentteihin tai vakiopituisiin sivuihin.

Koska keskusmuisti on kallista, käytetään sen laajennuksena eli tukimuistina nopeaa levyä, jonne prosesseja tai niiden osia voidaan tilanahtauden yllättäessä siirtää. Moniajojärjestelmien suuri muistin tarve ja nopean muistin korkea hinta synnytti lähes kaikkiin järjestelmiin aluksi tällaisen kaksitasoisen talletusjärjestelmän. Myöhemmin on siirrytty jo useampitasoiseen muistihierarkiaan (kuva 9.1).

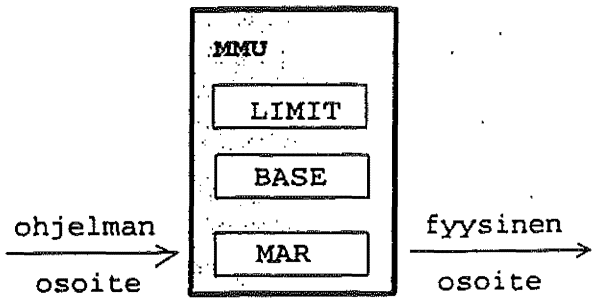
Keskusmuistin ja levyn välistä nopeuseroa (esim. 100 ns : 10 ms) pyritään lievittämään säilyttämällä *lohkopuskurissa* (engl. buffer memory, buffer cache) niitä tietoja, joita todennäköisesti tarvitaan lähitulevaisuudessa. Lohkopuskurista käytetään myös termiä *levyvälimuisti* (engl. disk cache).

Keskusmuistin ja rekistereiden välistä siirtoa nopeutetaan vastaavasti lisäämällä niiden väliin nopea *välimuisti* (engl. cache), jossa säilytetään viimeeksi viitattujen ja niiden lähellä olevien muistipaikkojen sisältöjä.

9.2 Kanta- ja rajarekisteriä käyttävä järjestelmä

Osoitemuunnos

Jos laitteistossa ei ole toteutettu virtuaalimuistia, käsitellään prosesseja yhtenäisinä kokonaisuuksina. Tarvittava osoitemuunnos sekä suojaus toteutetaan hyvin yksinkertaisesti kahden muistinhallintayksikön rekisterin, kanta- ja rajarekisterin avulla. Kun prosessi pääsee suoritettavaksi prosessoriin, sijoittaa käyttöjärjestelmä kantarekisteriin BASE ohjelmalle varatun yhtenäisen muistialueen alkuosoitteen. LIMIT-rekisteriin puolestaan asetetaan prosessille varatun muistialueen pituus.



Kuva 9.2: Muistinhallinnassa käytettävät rekisterit kanta- ja rajarekisteriin perustuvassa järjestelmässä.

Kun prosessi viittaa muistissa olevaan tietoon, tarkistaa muistinhallintayksikkö, että ohjelman osoite on pienempi kuin LIMIT-rekisterin arvo. Sen jälkeen se lisää ohjelman osoitteeseen BASE-rekisterin arvon.

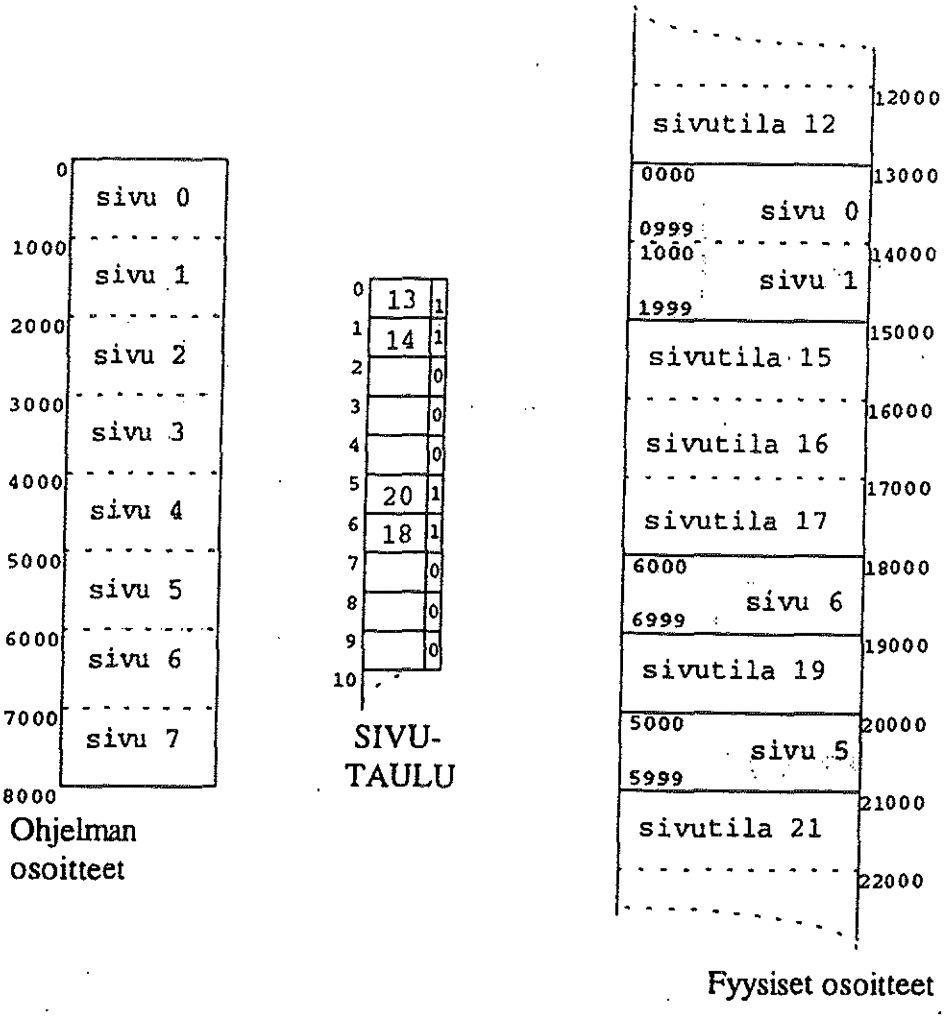
Algoritmi:

```
MAR <-- ohjelman sisäinen osoite
Jos     (MAR < 0) tai (MAR > LIMIT)
niin    aiheuta poikkeus 'virheellinen muistiviittaus'
muuten  MAR <-- MAR + BASE
```

Kun prosessin osoiteavaruus jaetaan segmentteihin siten, että koodille ja vakioille on oma segmenttinsä ja datalle on oma segmentti, käytetään kahta paria kanta- ja rajarekistereitä eli kummallekin osalle on omat rekisterinsä. Koska koodi on muuttumaton, voi sitä käyttää kaksi tai useampia prosesseja yhtäaikaa, kunhan ne varaavat omat tilat datallensa.

9.3 Virtuaalimuisti

Virtuaalimuistitoteutuksessa luovutaan siitä ajatuksesta, että prosessin kaikki tiedot olisivat koko ajan keskusmuistissa. Koko prosessia ei tuoda kerralla keskusmuis-



Kuva 9.3: Sivutaulun avulla kuvataan prosessin sivujen sijoittuminen fyysisiin sivutiloihin. (Tässä poikkeuksellisesti sivun koko on 1000.)

tiin, vaan keskusmuistissa pidetään vain kullakin hetkellä tarpeelliset prosessin osat. Muut osat ovat järjestelmän nopeassa tukimuistissa. Tämän lisäksi sallitaan myös, että prosessin loogisesti peräkkäiset osat voivat sijaita erillään keskusmuistissa (vapaastisijoittava ositus).

Virtuaalimuistijärjestelmissä keskusmuisti eli *fyysinen muistiavaruus* jakaantuu *sivutiloihin* (engl. page frame). Myös ohjelman osoiteavaruus eli *looginen osoiteavaruus* jakaantuu samankokoisiin *sivuihin* (engl. page). Sivun koko on esimerkiksi 512 B, 1024 B, 4096 B. (Käytämme monisteen esimerkeissä kuitenkin sivun kokoa 1000 päissälaskennan helpottamiseksi.) Prosessin sivujen sijoittuminen sivutiloihin on kirjattu prosessin *sivutauluun* (engl. page table). Jokaisella prosessilla on oma sivutaulunsa ja sen fyysinen muistiosoite on kirjattu kunkin prosessin omaan prosessin kuvaajaan. Kuvassa 9.3 on esitetty erään prosessin sivujen sijoittumista muistiin.

Sivutaulussa on yksi alkio kutakin ohjelman sivua kohden. Läsnäolobitti P (engl. presence bit) kertoo onko sivu keskusmuistissa vai levyllä. Jos sivu on muistissa, käy sen sijainti ilmi sivutauluun merkityn sivutilan numeron avulla. Tämän lisäksi sivutaulun alkiossa on muutettu-bitti M (engl. modified). Sen perusteella käyttöjärjestelmä osaa kirjoittaa sivun sisällön tarvittaessa levylle, jos sivutila otetaan muuhun käyttöön. Sivutaulun alkioon on talletettu hiukan myös muuta käyttöjärjestelmän tarvitsemaa tietoa (joita emme käsittele tällä kurssilla).

Sivutilan numero	P-bitti	M-bitti	Muuta

Osoitemuunnos

Ohjelman osoite voidaan helposti katkaista laitteistolla kahteen osaan siten, että siitä käy ilmi *sivunumero* ja siirtymä sivun sisällä. (Katkaisussa on itse asiassa kyse jakolaskusta: sivunumero = ohjelman osoite DIV sivun koko, ja siirtymä = ohjelman osoite MOD sivun koko). Sivunumeron perusteella löydetään oikea sivutaulun alkio. Ohjelman osoitetta vastaava fyysinen osoite saadaan katenoimalla peräkkäin sivutaulusta löytyvä sivutilan numero ja siirtymä sivun sisällä.

Esimerkki: Mikä on kuvan 9.3. prosessin osoitetta 0 vastaava fyysinen osoite, ts. missä sijaitsee prosessin ensimmäinen käsky? Mikä fyysinen osoite vastaa osoitetta 6345?

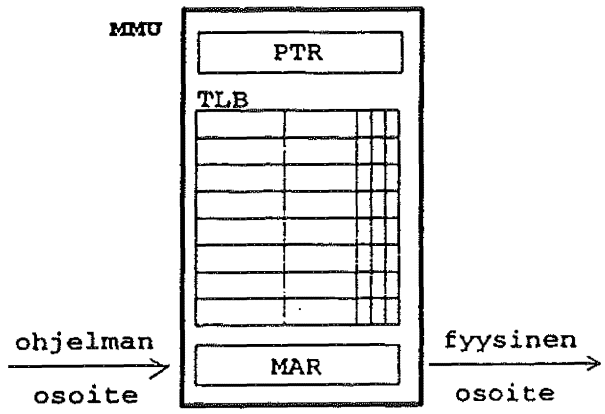
Kun osoite 0000 katkaistaan kahteen osaan, saadaan sivunumero 0 ja siirtymä sivun sisällä 000. Sivutaulun alkiosta 0 nähdään, että sivu 0 sijaitsee sivutilassa 13. Fyysinen osoite saadaan liittämällä sivutilan numero ja siirtymä peräkkäin. Prosessin ensimmäinen käsky on siten fyysisessä osoitteessa 13000.

Vastaavasti osoite 6345 sijaitsee sivulla 6 ja siirtymä sivun sisällä on 345. Sivutaulun alkion 6 perusteella saamme fyysiseksi osoitteeksi 18345.

Jotta jokaisella muistiviittauksella tehtävä osoitteenlaskenta olisi riittävän nopea, on koneissa, joiden osoiteavaruus on pieni, suoritettavan prosessin sivutaulun koko sisältö muistinhallintayksikön MMU rekistereissä (vrt. kanta- ja rajarekisterit!). Kun suoritettava prosessi vaihtuu, vaihdetaan myös muistinhallintayksikön rekistereiden sisältö.

Kun osoiteavaruus on suuri, on laitteiston muistinhallintayksikössä vain *rekisteri sivutaulun keskusmuistiosoitteelle* (engl. page table register, PTR). Osoitemuunnos vaatii nyt yhden muistinoudon, sillä sivutaulun alkio on tuotava tutkittavaksi muistinhallintayksikköön. Koska muunnos on tehtävä ajoaikana jokaisella muistiviitteellä, on muunnoksen oltava nopea. Tämän vuoksi säilytetään viimeeksi muunnoksissa tarvittuja tietoja MMU:ssa muutaman alkion (esim. 8 alkiota) *osoitemuunnospuskurissa* (engl. translation lookaside buffer, TLB).

TLB:hen on kopioitu tietoa viitatuista sivutaulun alkioista. Keskeisimmät TLB:n alkiossa olevat tiedot ovat sivunumero ja sivutilan numero sekä validibitti V. Vali-



Kuva 9.4: Muistinhallinnassa käytettävät rekisterit virtuaalimuistijärjestelmässä. Sivutaulurekisterin PTR sisältönä on suorituksessa olevan prosessin sivutaulun keskusmuistiosoite ja osoitemuunnospuskurissa TLB on viimeisimmissä osoitemuunnoksissa käytettyjä tietoja.

dibitit nollataan, kun suoritettavaksi tulee uusi prosessi, jotta seuraavan prosessin osoitemuunnoksessa ei käytettäisi edellisen prosessin tietoja.

Sivunumero	Sivutilanumero	P-bitti	V-bitti	Muuta

Jos prosessi viittaa äskettäin viitatulla sivulla oleviin muistipaikkoihin, on muunnoksessa tarvittava tieto nyt valmiiksi MMU:ssa eikä ylimääräistä muistinoutoa tarvita. Tämä on hyvin todennäköistä, sillä prosessit suorittavat tyypillisesti peräkkäiset käskyt peräkkäin ja esimerkiksi 1024 tavun sivulle sopii jo mukavia toistojakin.

TLB on nopeaa assosiatiivimuistia, joten etsintä siitä ei tapahdu peräkkäisetsintänä. Assosiatiivimuisti on toteutettu siten, että se pystyy tavallaan vertaamaan etsittävää kaikkien alkioiden tunnisteosaan (sivunumero) samanaikaisesti.

Koska prosessille kerrallakeskusmuistista allokoitujen sivutilojen lukumäärä on yleensä huomattavasti pienempi kuin prosessin maksimissaan tarvitsemien sivujen lukumäärä, on hyvinkin mahdollista, että viitattu sivu ei ole keskusmuistissa. Tällöin muistinhallintayksikkö aiheuttaa sivunpuutoskeskeytyksen (engl. page fault) ja käyttöjärjestelmä noutaa sivun levyltä muistiin.

Edellisillä mausteilla tarkennettu osoitemuunnosalgoritmimme on seuraava:

```
Jaa ohjelman osoite sivunumeroksi ja siirtymäksi
Etsi sivunumeroa TLB:stä
Jos  ei löydy tai validibitti V = 0 niin
     nouda sivutaulun alkio MEM[PTR+sivunumero]
     johonkin TLB:alkioon
     Jos      alkion läsnäolobitti P = 0
     niin     aiheuta sivunpuutoskeskeytys
     muuten   lisää TLB:n alkioon sivun numero
              aseta validibitti V = 1

fyysinen osoite <- katenoi(TLB.sivutilan numero, siirtymä)
```

9.4 Heittovaihto

Moniajojärjestelmässä syntyy usein tilanteita, joissa prosessille on varattu tilat keskusmuistista, mutta prosessi ei ole tarvinnut tai ei ole saanut pitkään aikaan käyttää prosessoria. Tällainen passiivinen prosessi olisi saatava pois keskusmuistista, jotta vapautunutta muistia voitaisiin allokoida aktiivisille prosesseille.

Käyttöjärjestelmässä toteutettu mekanismi, joka huolehtii prosessin tai sen osien väliaikaisesta muistista poistamisesta levylle, kulkee nimellä heittovaihto (engl. swapping). Muut prosessiin kuuluvat osat voidaan vapauttaa, mutta prosessin kuvaaja jätetään aina muistiin. Niitä prosessin osia, joita ei ole muutettu (esim. koodi) ei tarvitse kuitenkaan kirjoittaa tukimuistiin. Luonnollisesti tästä poistosta on tehtävä merkintä prosessin kuvaajaan.

Heittovaihdon on myös aika-ajoin tuotava aktiivisiksi READY-tilaan tulleiden prosessien rakenteita levyltä takaisin keskusmuistiin.

Jos vapaasta tilasta on jatkuvasti puutetta ja suoritettavana on useita prosesseja, voi prosessien heittovaihto aiheuttaa ylimääräistä rasitetta ja hidastaa järjestelmää. Tätä tilannetta kutsutaan *ruuhkaksi* (engl. trashing). Tässä tilanteessa on syytä vähentää suoritettavien prosessien lukumäärää.

Kanta ja rajarekisteriä käyttävä järjestelmä

Jos keskusmuistissa ei ole käynnistyvälle tai sinne uudelleen tuotavalle prosessille riittävän suurta yhtenäistä vapaata aluetta, poistetaan prosesseja tukimuistiin kunnes tilaa on riittävästi. Jos prosessi tarvitsee myöhemmin lisätilaa esimerkiksi siksi, että sen data tai pino kasvaa, sille allokoidaan uusi riittävän suuri tila. Jos keskusmuistissa ei kuitenkaan ole riittävän suurta yhtenäistä vapaata aluetta, siirretään koko prosessi tukimuistiin. Kun yhtenäistä tilaa myöhemmin on riittävästi voidaan kasvanut prosessi tuoda taas keskusmuistiin. Edellä oleva menettely estää lukkiutumisen.

Lisätilaa tarvittaessa heittovaihto voi poistaa prosesseja keskusmuistista, kunnes tilaa on riittävästi uudelle prosessille. Poistoja voidaan tehdä esimerkiksi seuraavassa järjestyksessä (poistoalgoritmi): hitaita tapahtumia esimerkiksi siirräntää odottavat passiiviset prosessit (WAIT-tilassa olevat), suuret ennen pieniä, ensiksi muistiin tuodut ja pienimmän prioriteetin prosessit.

Kun keskusmuistissa on tilaa uusille prosesseille tai uusia prosesseja tulee aktiivisiksi, tuo heittovaihto tukimuistista prosesseja keskusmuistiin esimerkiksi periaatteella pienet ennen suuria ja aiemmin poistetut ennen myöhemmin poistettuja (noutoalgoritmi).

Keskusmuistin vapaata tilaa hallitaan linkittämällä vapaat alueet toisiinsa. Listan solmuissa on pareja (alkuosoite,pituus) sekä viite listan seuraavan solmuun. Tilanvaraus tehdään tavallisesti First-Fit periaatteella (sijoitusalgoritmi) eli tila varataan ensimmäisestä mahdollisesta paikasta.

Muistinvarausten ja -vapautusten yhteydessä jää muistiin alueita, jotka yhdessä riittäisivät tilavaatimusten täyttämiseksi, mutta kukin erikseen on liian pieni. Tilannetta kutsutaan *pirstoutumiseksi* (engl. fragmentation).

Prosessia, jonka osoiteavaruus on suurempi kuin käytettävissä oleva muistiavaruus, ei voi ajaa kanta- ja rajarekisteriä käyttävässä järjestelmässä ollenkaan tai se vaatii erikoisjärjestelyjä (esim. kerrostus). Nykyisissä koneissa tämä ei ole enää ongelmana, sillä yleensä muistia on riittävästi isoillekin ohjelmille.

Virtuaalimuistijärjestelmä

Virtuualimuistin sivutilojen hallintaa varten käyttöjärjestelmä ylläpitää *sivutilataulua* (engl. page frame table). Siitä käy ilmi mitkä keskusmuistin sivutiloista on vapaita ja mitkä varattuja. Sivutilataulua tarvitaan aina, kun sivuja varataan prosesseille tai kun prosessien sivuja vapautetaan.

Jos vapaita sivutiloja ei ole, on jonkun prosessin joku sivu vietävä tukimuistiin ja puuttuva sivu tuodaan näin vapautuvaan sivutilaan. Kyseessä on täsmälleen sama tilanne kuin edellä esitetyssä kanta- ja rajarekisteriin perustuvassa järjestelmässä (nouto-, poisto- ja sijoitusalgoritmit). Tällä kertaa käsitellään kokonaisten prosessien sijasta vain prosessin osia, sivuja.

Prosessin sivujen muistiinnouto perustuu tavallisesti *tarvesivutukseen* (engl. demand paging): prosessin sivuja tuodaan muistiin vasta silloin, kun niitä tarvitaan. Käyttöjärjestelmä noutaa sivun muistiin, kun muistinhallintayksikkö huomaa puuttuvan sivun ja aiheuttaa sivunpuutoskeskeytyksen. Toinen vaihtoehto prosessin sivujen muistiintuomiselle on ennaltanouto (engl. prepaging). Ennaltanoudossa tuodaan kaikki prosessin sivut muistiin prosessin käynnistämisen yhteydessä, ja niitä heittovaihdetaan levylle tilanahtauden yllättäessä.

Muistiin tuotavan sivun sijoittaminen on helppoa. Koska kaikki sivut ja sivutilat ovat samankokoisia, otetaan käyttöön ensimmäinen eteensattuva vapaa sivutila.

Poistopäätöksiä varten laitteisto ja käyttöjärjestelmä ylläpitävät TLB:ssä ja sivu-

taulussa sivukohtaisia viitebittejä. Niiden perusteella käyttöjärjestelmä voi päätellä, mihin sivuun ei ole pitkiin aikoihin viitattu. Tällainen sivu on melko hyvä valinta poistettavaksi, sillä todennäköisesti prosessi on ohittanut jo sen vaiheen, jossa tuota sivua tarvittiin.

Luku 10

Tiedostojärjestelmä ja muistilaitteet

Prosessia suoritettaessa pidetään sekä käskyt että niiden käsittelemä tieto keskusmuistissa. Koska keskusmuisti on toteutettu nopealla tekniikalla, on se myös huomattavasti kalliimpaa kuin ulkoiset muistilaitteet. Kun lisäksi keskusmuistin koko on rajallinen eikä tietoa voida säilyttää siellä virtakatkojen yli, on laitteistoon kytketty tiedon pysyvää tallennusta varten ulkoisia tietovälineitä oheismuistiksi (engl. secondary storage). Pysyvän tallennuksen lisäksi käyttöjärjestelmä käyttää ulkoisia muisteja heittovaihdossa keskusmuistin laajennuksena, tukimuistina. Ulkoista muistia käytetään myös apu- ja välitallennuspaikkana esimerkiksi prosessin eri suoritusvaiheiden välisessä tiedonvälityksessä.

10.1 Tiedostojärjestelmän periaatteita

Tiedosto on joukko yhteenkuuluvaa dataa (tavuja), jolle tiedostojärjestelmä varaa levyltä tilaa lohkon kokoisina palasina (esim. 1024 tavua). Tiedostojärjestelmä pitää kirjanpitoa taltioilla olevien tiedostojen nimistä ja niihin kuuluvien lohkojen sijainnista sekä toimii välittäjänä prosessin tavuittain tapahtuvan käsittelyn ja tallennuksen lohkokäsittelyn välillä.

Tiedostojärjestelmän on sallittava tiedostojen luonti, tiedostoon kirjoittaminen ja sieltä luku sekä tiedostojen tuhoaminen. Tiedostojärjestelmän on varmistettava tiedostot laitevirheiltä ja estettävä tiedostojen luvaton käyttö. Toisaalta sen on myös sallittava hallittu yhteiskäyttö.

Levy on yleensä jaettu *uriin*, *sektoreihin* ja *lohkoihin*. Kuitenkin viimeisten kahden tai kolmen vuosikymmenen kokemukset ovat osoittaneet, ettei näillä alhaisen tason käsitteillä ole merkitystä käyttäjärjestelmän tiedostojärjestelmässä. Ne täytyy ottaa huomioon vasta laiteajurien laitteistosta riippuvassa osassa. On riittävää abstrahoida ulkoinen muisti peräkkäisinä lohkoina, jotka on numeroitu 0:sta N-1:een ja joiden koko on vakio, B tavua per lohko. Näitä lohkoja kutsutaan *lineaarisiksi lohkoiksi*. Tiedostot jaetaan puolestaan kokoa B oleviin *loogisiin lohkoihin* (vrt. sivut ja sivutilat virtuaalimuistin toteutuksessa).

Toinen hyväksi osoittautunut periaate on käsitellä tiedostoja *anonyyminä*. Käyttäjä antaa tosin tiedostolle nimen, mutta nimi ei ole varsinaisesti tiedoston ominaisuus. Miten tiedosto ja sen nimi erotetaan toisistaan, nähdään myöhemmin.

Kolmas periaate liittyy tiedostojen rakenteeseen: pitäisikö tiedostolla olla käyttöjärjestelmän tukema sisäinen rakenne vai pitäisikö tiedoston käyttöjärjestelmän kannalta olla täysin ilman rakennetta, vain jono tavuja. Jos käyttäjä haluaa jonkinlaisen rakenteen, hänen täytyy ohjelmoida se itse.

Struktuuria kannattavien perusteluna on, että tällöin tiedosto-operaatiot voidaan toteuttaa tehokkaammin käyttöjärjestelmän tasolla. Lisäksi ohjelmoijalta säästyy paljon vaivaa.

'Simplistien' perustelu puolestaan on, ettei ole koskaan mahdollista tehdä käyttöjärjestelmän tukemia tiedostorakenteita niin yleisiksi, että ne täyttäisivät useimpien käyttäjien tarpeet. Siten monien täytyy kuitenkin ohjelmoida omat struktuurinsa. Tällaisessa tilanteessa on parempi tehdä tiedostojärjestelmästä niin yksinkertainen kuin mahdollista ja saada aikaan tehokkuus tällä tavalla.

IBM:n OS/360 ja Digitalin VAX-järjestelmien DEC/VMS toteuttivat struktuuriin pohjautuvan tiedostojärjestelmän, kun taas UNIX ja sen johdannaiset valitsivat yksinkertaisuuden. Näyttää siltä, että historia on osoittanut UNIXin lähestymistavan paremmaksi.

10.2 Tiedostojen talletus levylle

Peruskysymys on myös, miten loogiset lohkot kuvataan lineaarisille lohkoille. On olemassa kaksi vaihtoehtoa: *peräkkäin* tai *ei-peräkkäin*. Edellisessä tiedosto sijoittuu lineaarisiin lohkoihin $n, n+1, \cdots, n+k-1$, jos tiedosto tarvitsee k lohkoa. Kuvaus

$$\text{looginen} \longrightarrow \text{lineaarinen}$$

on siten translaatio $k \mapsto k + n - 1$. Toisessa vaihtoehdossa tiedosto sijoitetaan lineaarisiin lohkoihin, jotka voivat olla erillään ja kaukana toisistaan.

Peräkkäisellä sijoittamisella on joitakin etuja. Ensinnäkin kirjanpito on mahdollisimman yksinkertaista. Riittää muistaa 1. lohkon alku ja lohkojen lukumäärä. Toiseksi jos lineaariset lohkot talletetaan myös fyysisesti peräkkäin levylle, tiedoston peräkkäinen lukeminen ja kirjoittaminen voi tapahtua minimaalisella vaivalla. Jos on juuri luettu lohko m, niin välittömästi tämän jälkeen levyn luku/kirjoituspää on lohkon $m + 1$ kohdalla. Moniajojärjestelmässä tosin on epätodennäköistä, että sama prosessi pääsisi keskeytyksen jälkeen käyttämään heti seuraavaa lohkoa. Se on todennäköisesti mennyt toisen prosessin käyttöön.

Peräkkäisellä talletuksella on myös haittoja. Jos halutaan luoda k:n lohkon tiedosto, on löydettävä k perättäistä vapaata lineaarista lohkoa. Jos olemassaoleva tiedosto kasvaa, niin voi sattua, että tiedostoon jo kuuluvat lineaariset lohkot $n, n+1, \cdots, n+k-1$ eivät riitä lisätiedoille ja lohko $n + k$ on jo toisen tiedoston käytössä. Nyt ei ole muuta mahdollisuutta, kuin kopioida koko tiedosto uuteen,

isompaan vapaaseen alueeseen. Lisäksi muisti voi ennenpitkää *osittua* (engl. external fragmentation). Tällöin vapaa tila on pienissä osissa siellä täällä. k:n lohkon tiedostoa ei voi tallettaa yhtenäiselle alueelle, vaikka muistissa olisikin yhteensä yli k lohkoa vapaana.

Viimeinen ongelma poistuu, jos tiedostot talletetaankin ei-peräkkäin. Tällöin tiedosto voi myös kasvaa. Ei ole myöskään tarvetta etsiä tarpeeksi suurta yhtenäistä aluetta. Haittana on tosin, että kirjanpito on monimutkaisempaa. Lisäksi tiedoston peräkkäinen lukeminen on tehottomampaa, sillä levyn luku/kirjoituspää joutuu siirtymään paikasta toiseen.

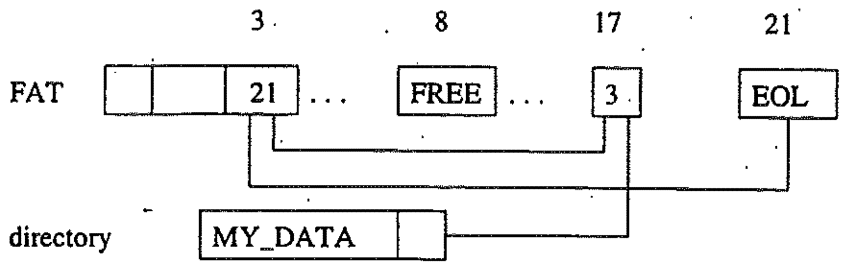
Kaiken kaikkiaan ei-peräkkäinen malli näyttää paremmalta edellyttäen, että löydetään tyydyttävä kuvaus loogisesta lineaariseen. Seuraavassa on muutamia ratkaisuvaihtoehtoja.

CP/M:n ratkaisu

CP/M tallettaa lineaaristen lohkojen numerot tiedoston *hakemistoalkioon* (engl. directory entry for a file). Jos kaikki numerot eivät mahdu yhteen tietueeseen, otetaan käyttöön useampia.

MS/DOS:in ratkaisu

MS/DOS käyttää taulukkoa, jossa on yhtä monta alkiota kuin levyllä on lohkoja (FAT = File Allocation Table). Tiedoston hakemistoalkiossa on 1. lineaarisen lohkon numero, esimerkiksi n_1. FAT-taulukossa paikassa n_1 on seuraavan lohkon numero jne. Jos n_k on viimeisen lohkon numero, niin FAT:in paikassa n_k on erikoismerkki 'listan loppu'.

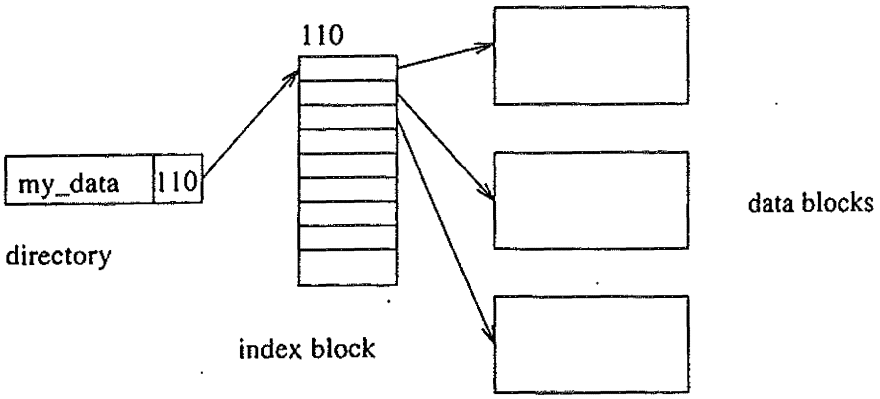


Kuva 10.1: FAT-taulukko.

UNIXin ratkaisu

Kolmas vaihtoehto on varustaa jokainen tiedosto *indeksillä* eli lohkolla, joka sisältää varsinaisten datalohkojen numerot.

Tällä on kuitenkin kolme epäkohtaa:



Kuva 10.2: Indeksin käyttö.

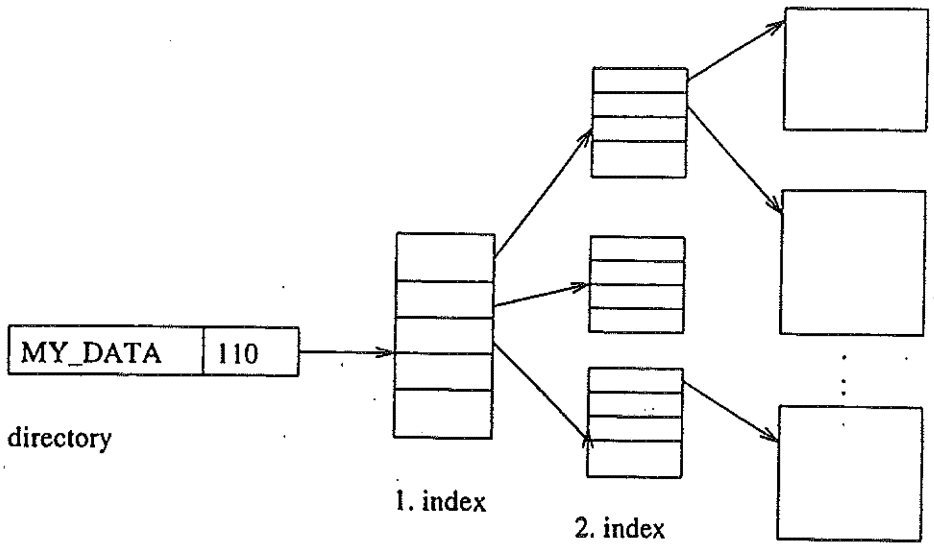
- Jokaista lohko-operaatiota kohti tarvitan kaksi lukua levyltä: indeksilohkon ja varsinaisen datalohkon lukeminen.
- Pieni, esimerkiksi 20 tavun tiedosto tarvitsee aina kaksi lohkoa.
- Jos lohkon numero vaatii 4 tavua ja lohkot ovat 1024 tavua kooltaan, niin indeksilohkoon mahtuu 256 numeroa. Siten tiedosto voi olla korkeintaan 256 KB, aivan liian vähän!

Kolmas ongelma voidaan ratkaista ottamalla käyttöön kaksitasoinen indeksi: Nyt tiedoston koko voi olla 64 MB. Jos se ei ole tarpeeksi, otetaan käyttöön kolmitasoinen indeksi. Kuitenkin kaksi ensimmäistä epäkohtaa tulevat pahemmiksi.

Tästä syystä UNIX soveltaa sekaratkaisua, jossa tiedostojen hallintaan käytetään tietorakennetta nimeltä *inode* (alunperin 'index node', sitten 'i-node', lopulta vain 'inode'). Inode sisältää kaiken tarvittavan tiedon tiedostosta (ei kuitenkaan tiedoston tietosisältöä). Inode on konkreettisen tiedoston anonyymi vastine, itse asiassa juuri tiedosto käyttöjärjestelmän kannalta.

Inode sisältää:

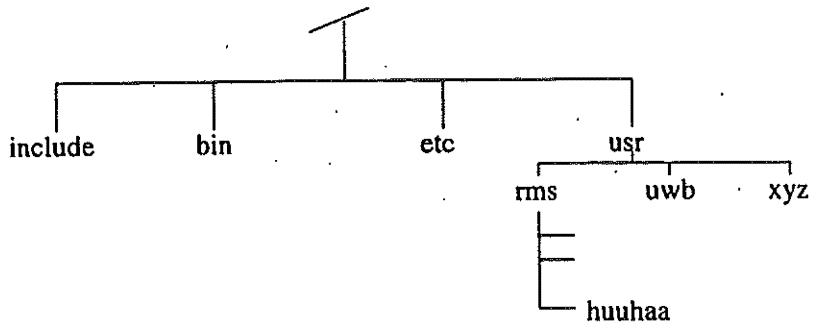
- 13 lohkonumeroa; ensimmäiset 10 ovat tiedoston 10 ensimmäisen datalohkon numerot. 11. on 1.tason indeksin numero, 12. on 2. tason indeksin ja 13. 3. tason indeksin numero;
- tiedoston koko tavuina;
- omistaja;
- käyttöoikeudet;
- yms.



Kuva 10.3: Kaksitasoinen indeksi.

Erityisissä tiedostoissa on luettelo, josta käy ilmi pari *(inode nro, nimi)*, eli miten nimi liittyy tiedostoon. Näitä erityistiedostoja kutsutaan *hakemistoiksi* (engl. directory). Ne ovat periaatteessa tavallisia tiedostoja, joskin niillä on keskeinen asema tiedostohallinnassa. Hakemistoilla on myös oma inode ja inode-numero. Lisäksi niillä voi olla nimi, joka löytyy hakemistosta.

Jotta ei jouduttaisi ikuiseen silmukkaan, jokaisessa tiedostojärjestelmässä on erityinen hakemisto, *juuri* (engl. root), jolla on erikoinen nimi '/'. Tämä nimi ei saa esiintyä missään hakemistossa. Siten hakemistot muodostavat puun:



Kuva 10.4: Hakemistopuu.

Jokainen tiedosto voidaan nyt identifioida hakupolun avulla. Esimerkiksi huuhaan hakupolku on /usr/rms/huuhaa. Nyt ei ole mitään estettä sille, että tiedostolla olisi useampia nimiä. Nimet voivat sijaita eri hakemistoissa tai samassa hakemistossa. Samaa nimeä ei kuitenkaan saa käyttää kahdelle eri tiedostolle samassa hakemis-

tossa.

10.3 Suojaus

Tiedostojen suojausongelma syntyy tarpeesta kieltää toisten käyttäjien tiedostojen luvaton käyttö ja tarpeesta sallia toisten käyttäjien tiedostojen käyttö. Tällaisia tarpeita ovat esimerkiksi samojen ohjelmien käyttö. Tavanomaisin suojaus on jakaa käyttäjät eri luokkiin ja jakaa myös tiedostojen käyttöoikeuksia näiden luokkien mukaan. Luokkajaoksi riittää usein omistaja (engl.user), omistajan kanssa samaan ryhmään kuuluvat (engl. group) ja muut (engl. others).

Tiedostomääreissä on kutakin luokkaa koskevat tiedot tiedoston käyttöoikeuksista. Käyttöoikeuksia voivat olla: ei mitään (-), oikeus suorittaa ohjelmatiedosto (engl. x = execute), oikeus lukea tiedoston sisältö (engl. r = read), oikeus kirjoittaa tiedostoon (engl. w = write) ja edellisten eri kombinaatiot.

Hakemistojen kohdalla nämä oikeudet tarkoittavat

- x: oikeus käyttää hakemistoa tiedoston hakupolussa,
- r: oikeus lukea hakemiston sisältö (komennolla dir, ls tms.) ja
- w: oikeus luoda tiedostoja hakemistoon tai poistaa tiedostoja hakemistosta.

Käyttöjärjestelmä tarkistaa tiedostojen (ja hakemistojen) käyttöoikeudet tiedoston avaamisen yhteydessä. Tarkistus tapahtuu tiedostoa käyttävän prosessin kuvaajassa olevien käyttäjä- ja ryhmätunnisteen ja tiedostomääreissä olevien omistaja- ja ryhmätunnisteen sekä käyttöoikeusbittien mukaan.

```
jos      tiedoston uid = prosessin uid
niin     tarkista oikeudet luokan omistaja biteistä
muuten   jos     tiedoston gid = prosessin gid
         niin    tarkista oikeudet luokan ryhmä biteistä
         muuten   tarkista oikeudet luokan muut biteistä
```

Prosessin käyttäjä- ja ryhmätunniste on kirjattu prosessin kuvaajaan PCB, kun moniajojärjestelmä kysyy käyttäjätunnuksen. Jokaiseen tunnukseen on järjestelmän käyttölupaa annettaessa liitetty nämä kaksi tunnistetta. Vastaavasti tiedostoa luotaessa tallettuu tiedostomääreisiin omistaja- ja ryhmätunniste. Uusille tiedostoille asetetaan käyttöoikeuksiin käyttöjärjestelmän oletusarvot. Niitä voi eräissä järjestelmissä myös muuttaa esim. tiedoston loogisen nimen määrittelyssä tai sidonnan ja tiedoston avaamisen yhteydessä. Suojaustietojen muuttamisoikeus on yleensä kiinteästi tiedoston omistajalla. Omistaja ei voi antaa sitä muille, eikä omistaja voi ottaa sitä itseltään pois.

search) ja siirtää lohkon (yksi tai useampia sektoreita) levyaseman ja keskusmuistin puskureiden välillä (engl. transfer).

Hakuvarren siirron ja pyörähdysviipeen kestoajat vaihtelevat hakuvarren ja haettavan sektorin sijainnista riippuen. Hakuvartta siirrettäessä joudutaan ylittämään keskimäärin 1/3 levyn urista. Pyörähdysviive on keskimäärin 1/2 kierroksen pyörähtämiseen kuluvasta ajasta. Siirto tapahtuu levyn pyörimisnopeudella ts. aina vakionopeudella.

Levymuisti on yleisimmin käytetty massatietoväline. Kiinteästi tietokoneeseen kytkettynä sen käyttö on nopeaa ja se sujuu ilman operointihenkilökunnan apua.

10.6 CD-ROM

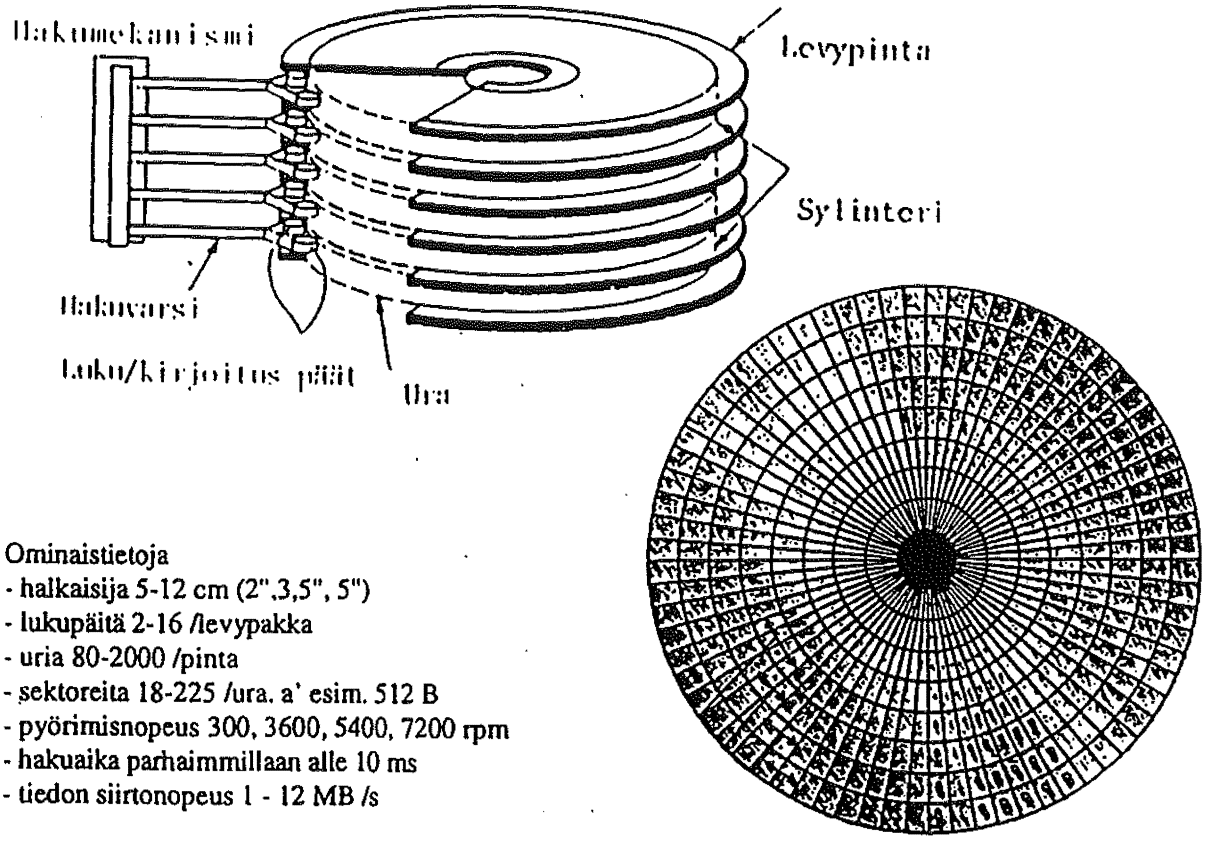
CD-ROM (engl. compact disk ROM, romppu) on ohuella metallifilmillä päällystetty levy, jonka suojana on ohut lasikalvo. Levyn filmille nostatetaan voimakkaalla laservalolla nystyjä, jotka tulkitaan 1-biteiksi. Luettaessa käytetään pienempitehoista punaista lasersädettä, joka heijastuu nystyistä eri tavoin kuin tasaisesta pinnasta.

CD-ROM-levyn halkaisija on noin 5 tuumaa. Tietoa talletetaan vain levyn toiselle puolelle. Levyllä on vain yksi ura, joka alkaa levyn keskeltä ja kiertää spiraalimaisesti levyn ulkoreunalle. Lähellä levyn keskustaa olevaa tietoa luettaessa levy pyörii nopeammin kuin ulkoreunalla olevaa tietoa luettaessa. Pyörimisnopeuden vaihtelulla lineaarinen kaarrenopeus pysyy vakiona, joten tieto tulee luettua vakionopeudella. Perusnopeudella levy pyörii 200 - 520 rpm. Tämän lisäksi tarjolla on myös tupla-, tripla- ja nelinopeuksisia levyjä. Niiden pyörimisnopeudet ovat vastaavasti 400 - 1040 rpm, 600 - 1560 rpm ja 800 - 2080 rpm. Kunnon videokuvan (25 kuvaa / sek) lukeminen levyltä vaatii nelikertaisen nopeuden. CD-ROM-asemien hakuajat vaihtelevat välillä 100-250 millisekuntia. Hakuajat ovat selvästi hitaampia kuin kiintolevyissä, joissa päästään alle 10 millisekunnin.

Levyn uran pituus on noin 5 kilometriä ja se koostuu yhtäpitkistä lohkoista, joita on 270 000 tai 333 000 kappaletta. Lohkon pituus on 2352 tavua. Jokaisen lohkon ensimmäiset 12 tavua käytetään synkronointiin ja seuraavat 4 tavua lohkon otsikkoon. Dataa sisältävillä levyillä tulee tämän jälkeen 2048 tavua dataa ja lopussa vielä virheen paikantamista ja korjausta varten 288 tavua. Ääni- ja videolevyjen lohkojen lopussa ei tuota virheen paikantamis- ja korjaustietoa ole, sillä pienet virheet ovat merkityksettömiä.

Toistaiseksi useimmat laitteistot pystyvät vain lukemaan olemassa olevaa tietoa tai kirjoittamaan vain kerran levylle; olemassaolevan tiedon poisto ei onnistu (engl. Write Once Read Many, WORM). Tällä hetkellä on saatavilla jo myös kirjoitus- ja vapautustoiminnoin varustettuja, mutta talletuskapasiteetiltaan pienempiä (20MB - 128MB), magneettis-optisia levyjä.

CD-ROM-levy on erittäin luotettava ja tiedon tallennuskapasiteetti on valtaisa, tavallisimmin yhdelle levylle sopii tietoa 550 MB, ja parhaimmillaan peräti 2 Gigatavua. CD-ROM-levy soveltuu erityisen hyvin pysyvän, kirjastointiluonteisen tiedon



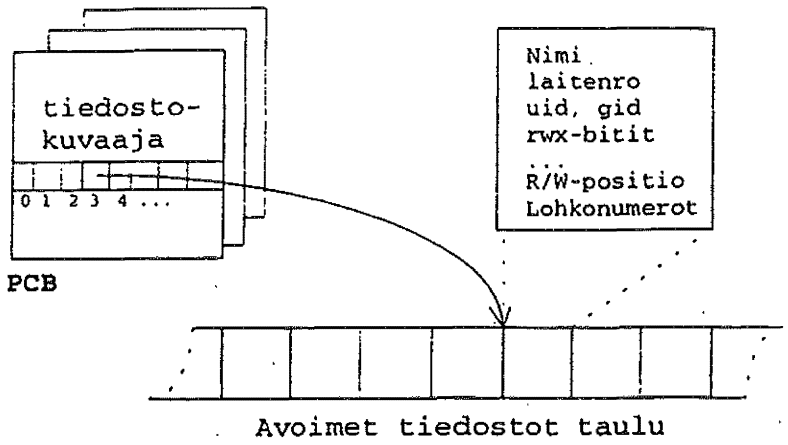
Ominaistietoja
- halkaisija 5-12 cm (2",3,5", 5")
- lukupäitä 2-16 /levypakka
- uria 80-2000 /pinta
- sektoreita 18-225 /ura. a' esim. 512 B
- pyörimisnopeus 300, 3600, 5400, 7200 rpm
- hakuaika parhaimmillaan alle 10 ms
- tiedon siirtonopeus 1 - 12 MB /s

Kuva 10.6: Levymuistin rakenne ja ominaistietoja.

taan *sylinteriksi*. Urat on jaettu edelleen *sektoreihin*, jotka ovat pienimpiä osoitettavissa olevia levymuistin osia. Levypinnat, sektorit ja urat on numeroitu ja etsittäessä tietoa levymuistista on tiedettävä millä levypinnalla, millä uralla (engl. track) ja missä sektorissa (engl. sector) haluttu tieto sijaitsee.

Levymuistiin kirjoittamista ja sieltä lukemista varten levyasemassa on *luku/kirjoituspäät*. Tavallisesti luku/kirjoituspäitä on yksi kutakin levypintaa kohden ja ne ovat erityisen hakuvarren päässä. Erittäin nopeissa levymuisteissa voi olla yksi tai kaksi kiinteää luku/kirjoituspäätä kutakin levypinnan uraa kohden. Levymuisti on poiminta- eli hajasaantimuisti, jonne voidaan tallettaa paitsi peräkkäistiedostoja myös suorasaantitiedostoja. Hajasaantisen tiedoston tietoja ei tarvitse käsitellä siinä järjestyksessä kuin ne sijaitsevat tiedostossa, vaan mikä tahansa tiedoston lohko voidaan lukea ja kirjoittaa muista riippumatta keskimäärin yhtä nopeasti.

Kun ajuri on laskenut lohkonumerosta levypinnan, uran ja sektorin numeron, se käynnistää ohjaimen tekemään siirtoa. Ohjain vie hakuvarren (eli luku/kirjoituspään) oikealle sylinterille (engl. seek), odottaa, että oikea lohko pyörähtää kohdalle (engl.



Kuva 10.5: Tiedostojen käsittelyssä tarvittavia käyttöjärjestelmän tietorakenteita

merkiksi tietueittain, ei jokaiseen prosessin I/O-lauseeseen liity fyysistä siirräntää.

Tiedostosta luettaessa käyttöjärjestelmä määrittää lukuposition perusteella lohkonumeron, jossa tavut sijaitsevat (positio DIV lohkon koko) ja etsii lohkoa lohkopuskurista. Jos lohkoa ei löydy, niin se varaa lohkolle tilaa puskurista ja käynnistää siirron taltiolta lohkopuskuriin. Kun siirto aikanaan valmistuu, kopioi käyttöjärjestelmä edelleen pyydetyn määrän tavuja prosessin työtilaan. Jos kaivatut tavut sensijaan löytyvät jo suoraan lohkopuskurista, ei noutoa taltiolta tarvita.

Kun tiedostoon kirjoitetaan, kopioi käyttöjärjestelmä ensin kirjoitettavat tavut prosessin työtilasta lohkopuskuriin. Kun puskurissa oleva lohko täyttyy, kirjoittaa käyttöjärjestelmä sen taltiolle. Jokaiseen tiedostoon kirjoitukseen ei siis välttämättä liity taltiolle tallettamista.

Prosessin lopuksi on kaikki käytetyt tiedostot suljettava, jotta käyttöjärjestelmä voi vapauttaa avauksen yhteydessä luomansa tietorakenteet. Useat järjestelmät sulkevat käytetyt tiedostot automaattisesti prosessin päättyessä.

Tiedostoa suljettaessa käyttöjärjestelmä kirjoittaa viimeisen vajaan kirjoituslohkon lohkopuskurista tiedostoon ja vapauttaa tiedostoon liittyvät omat hallinnolliset rakenteet. Jos tiedoston määreet ovat muuttuneet, on myös ne kirjoitettava takaisin taltiolle.

10.5 MG-levyt

Levymuistissa (engl. disk) on yhteisellä akselilla olevia, magnetoituvalla aineella päällystettyjä kevytmetallikiekkoja, jotka pyörivät jatkuvasti kiinteällä nopeudella. Levyköt ovat joko kiinteitä tai vaihdettavia. Levypinnat on jaettu samankeskisiin, ympyränmuotoisiin *uriin*. Päällekkäisten urien muodostamaa kokonaisuutta sano-

10.4 Tiedoston käyttö

Prosessin eri suorituskerroilla käytetään usein eri tiedostoja ja useat eri prosessit käyttävät yhteisiä tai toistensa tuottamia tiedostoja. Esimerkiksi kääntäjän syötteenä (käännettävänä) on eri suorituskerroilla eri tiedosto ja linkittäjä käyttää kääntäjän tuottamaa objektimoduulia. Tämän vuoksi prosessissa käytetään sisäistä tiedostontunnusta, loogista tiedostonimeä.

Prosessin sisäinen tiedostonimi sidotaan todelliseen eli fyysiseen tiedostonimeen ohjelmointikieleen kuuluvalla lauseella, tai tiedostoa avattaessa.

Prosessi voi käyttää vain tiedostoja, jotka on avattu käyttöä varten. Avauskäskyn tarkoituksena on saada käyttöjärjestelmä luomaan siirrossa ja lohkojen puskuroinnissa tarvittavat tietorakenteet. Ohjelmointikieleen kuuluvasta avauskäskystä käy ilmi mm. tiedoston looginen nimi sekä tiedoston käyttötapa (lukeminen, kirjoitus jne.). Kaikissa järjestelmissä ei tarvita eksplisiittistä tiedoston avaamista, vaan järjestelmä avaa tiedoston automaattisesti ensimmäisen luku- tai kirjoituspyynnön yhteydessä (kääntäjä siis generoi avauskäskyn).

Kun tiedosto avataan käyttöä varten, on käyttöjärjestelmän noudettava tiedostohakemisto keskusmuistiin ja etsittävä tiedoston nimen perusteella hakemistoalkio. Hakemistoalkiosta tutkitaan tiedostomääreet ja tarkistetaan onko käyttöoikeudet kunnossa. Lisäksi voi olla syytä tarkistaa, että tiedoston käyttö on turvallista: tiedostoa voi muuttaa vain yksi prosessi kerrallaan.

Tämän jälkeen käyttöjärjestelmä luo tiedoston käsittelyssä tarvittavat tietorakenteet. Tiedoston hakemistoalkio kopioidaan globaaliin *avoimet tiedostot tauluun* (engl. global open file table) ja siihen lisätään mukaan ainakin lukupositio / kirjoituspositio sekä laitenumero. Prosessin kuvaajassa olevaan *tiedostokuvaajatauluun* (engl. file descriptor) laitetaan viite avoimet tiedostot tauluun. Tiedoston avanneelle prosessille käyttöjärjestelmä palauttaa indeksin tiedostokuvaajatauluun. Jatkossa prosessi viittaa tähän tiedostoon vain ja ainostaan tämän indeksin kautta (kuva)

Jos tiedosto avataan lukemista varten, saattaa tiedostojärjestelmä vielä varata levylohkon kokoisen puskurin ja käynnistää ensimmäisen levylohkon lukemisen ennalta lohkopuskuriin.

Prosessi käyttää tiedostoa siihen koodattujen luku- ja kirjoituskäskyjen avulla. Kääntäjä muuttaa lausekieleen koodatut I/O-käskyt käyttöjärjestelmän palvelupyynnöiksi. Esimerkiksi yksinkertainen lause, joka lukee tietueen tiedostosta Sisafile, voisi kääntyä seuraavasti:

```
PUSH SP,SisaFile        ; mistä tiedostosta
PUSH SP,=Tietue         ; minne siirretään (muistios.)
PUSH SP,=40             ; paljonko siirretään
SVC SP,=READ            ; operaatio
```

Koska käyttöjärjestelmä puskuroi siirrettäviä tietoja lohkopuskureissa ja siirtää tietoja taltion ja keskusmuistin välillä lohkoittain, mutta prosessi käsittelee tietoa esi-

tallennukseen ja suurten ohjelmistojen jakeluun. Multimediasovellusten yleistymisen myötä myös CD-ROM-laitteet ovat yleistyneet nopeasti.

10.7 MG-nauhat

Magneettinauhalle (engl. magnetic tape) tieto talletetaan binäärikoodina siten, että kukin merkki esitetään nauhan leveyssuunnassa olevalla bittijonolla. Talletetun merkin lisäksi kussakin bittijonossa on ylimääräinen pariteettibitti.

Mg-nauha-asemassa on kaksi kelaa: syöttö- ja vastaanottokela. Laitteessa on yksi luku-kirjoituspää. Yleensä nauhaa voidaan lukea tai kirjoittaa vain yhteen suuntaan. Lukemisen tai kirjoittamisen jälkeen nauha on kelattava takaisin syöttökelalle. Takaisinkelaus voidaan suorittaa suuremmalla nopeudella kuin nauhan lukeminen tai kirjoittaminen. Tietyltä nauhalta joko luetaan tai sille kirjoitetaan, mutta molempia ei voida tehdä samanaikaisesti. QIC (engl. quarter inch cartridge), DAT (engl. digital audio tape) ja 8mm:n nauhakaseteilla sekä syöttö- että vastaanottokela ovat samassa kotelossa ja ne ovat kooltaan pieniä.

Kelanauhan pituus on tavallisesti 600, 1200 tai 2400 jalkaa ja sen talletustiheys on 1600 tai 6250 tavua/tuuma (engl. bpi). DAT-kasetilla on nauhaa 60, 90 tai 120 metriä ja talletuskapasiteetti esimerkiksi 60 m nauhalla on noin 1 GB pakkaamatonta, ja suunnilleen tuplasti pakattua tietoa. QIC-nauhan pituus on yleisimmin noin 600 jalkaa. Uusimmille 30 ja 38 raitaisille QIC-kaseteille sopii 2 ja 5 GB pakkaamatonta tietoa. Talletusnopeus on tiedon pakkautuvuudesta riippuen tavallisimmin 10 - 20 MB minuutissa. Saantiaika on noin 20 sekuntia.

Magneettinauhalle talletettu tiedosto on aina *peräkkäistiedosto*: tietyn 'lohkon' lukemiseksi on luettava myös kaikki sitä edeltävät lohkot. Tämän vuoksi talletettavat tiedot järjestetään nauhalle yleensä jonkin yksikäsitteisen avaimen mukaan nousevaan tai laskevaan järjestykseen.

Kunkin mg-nauhalla olevan tiedoston alkuun talletetaan *alkunimiö* (engl. label), jossa on mm. tiedoston nimi, versionumero, päiväys, omistaja, tiedoston tyyppi sekä pituus ja jaksotustietoa. Tiedoston lopussa on vastaavasti *loppunimiö*, joka sisältää tiedoston loppumerkin lisäksi tarkistustietoa.

Lohkokoko (tietuekoko) on vapaasti valittavissa ja tieto tallennetaan joko kiinteänmittaisina tai vaihtelevanmittaisina *jaksoina*. Jaksojen väliin jätetään *jaksoväli*, joka ei sisällä mitään tietoa. Sitä tarvitaan nauhan pysäyttämistä (jarrutus) ja uudelleen käynnistämistä varten (kiihdytys). Jaksonpituus vaikuttaa nauhan täyttösuhteeseen ja puskuritilan tarpeeseen keskusmuistissa. Tietojen käsittely on nopeampaa, jos jakson pituus on suuri, sillä tällöin tarvitaan vähemmän hitaita nauhan käynnistyksiä. Toisaalta suuri jaksonpituus vaatii myös suurempia puskurialueita käyttöjärjestelmän työtiloista.

Magneettinauhat ja -kasetit ovat edullisia ja luotettavia tietovälineitä, jotka on helposti siirrettävissä laitteesta toiseen. Koska magneettinauhalle voidaan tallentaa vain peräkkäistiedostoja, ja nauha ei ole kiinteästi tietokoneeseen kytketty (vaa-

tii operointiapua), soveltuu se parhaiten harvoin käytettävän tiedon tallennukseen, esimerkiksi varmuuskopiointiin. Suureen nauhavarmistustarpeeseen on tarjolla automaattisia nauhanvaihtajia, 'jukeboxeja', jotka voivat kätkeä sisäänsä yli sata nauhakasettia.

Massamuisti (engl. mass storage) on tarkoitettu todella suurten tietomäärien, satojen miljardien tavujen tallentamiseen. Suuretkaan levyasemat eivät yllä toistaiseksi samoihin tallennusmääriin. Massamuisteissa tiedon saantiaika ei ole ratkaisevassa asemassa, vaan tärkeintä on suuri talletuskapasiteetti. Massamuistin toteutus perustuu esimerkiksi mehiläiskennorakenteeseen asetettuihin magneettinauhoihin ja niitä käsitteleviin automaattisiin laitteistoihin.

10.8 DOS-levyn hallinta

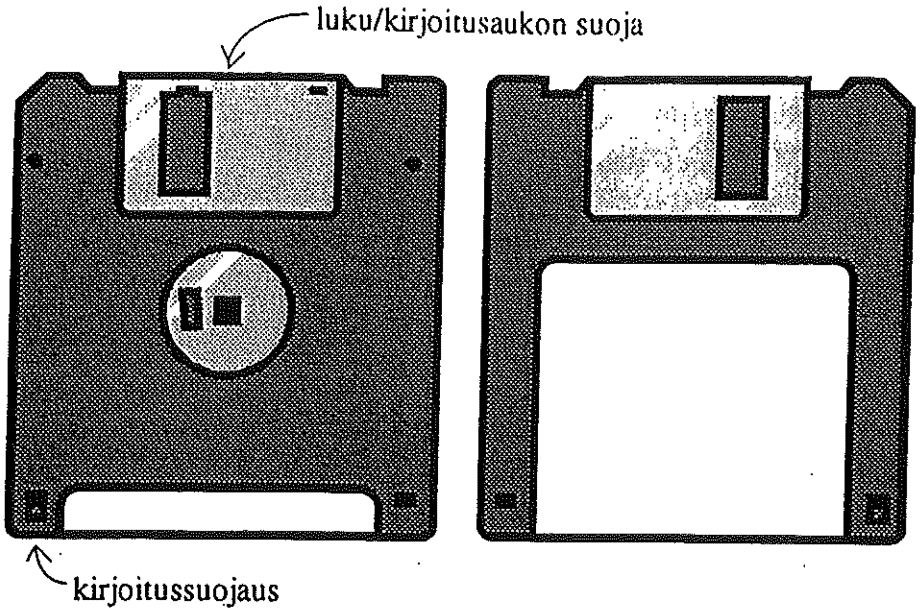
Mikrotietokoneissa käytetään yleisesti taipuisia levyjä eli levykkeitä (engl. floppy disk, diskette). Levykkeen tallennuskapasiteetti on tavallisimmin 1.44MB. Myös tätä suurempia magneto-optisia levykkeitä (20MB) on jo esitelty. Seuraavassa tarkastelemme levyjen hallintaa DOS-järjestelmän kannalta. Tämä on edelleen ajankohtaista, sillä esimerkiksi Windows95:n levynhallinta perustuu suurimmaksi osaksi DOS-ratkaisuihin.

Mikrotietokoneiden *kiinto-* eli *kovalevyissä* on levyistä ja luku/kirjoituspäistä muodostettu umpinainen, pieneen tilaan pakattu kiinteä kokonaisuus. Se ei ole yhtä altis likaantumiselle kuin levyke. Kovalevyn talletuskapasiteetti vaihtelee nykyisin 400 megatavusta yhdeksään gigatavuun. Kiintolevyn levyt pyörivät aina vakionopeudella, joka on määrätty kullekin kiintolevytyypille erikseen. Tyypillisimmät kierrosnopeudet ovat 3 600, 4 500, 5 400 ja 7 200 kierrosta minuutissa. Nopeimmat kiintolevyt toimivat peräti 10 000 kierroksen minuuttivauhdilla. Koska kovalevy pyörii jatkuvasti, ovat noudot siltä huomattavasti nopeampia kuin vain luettaessa tai kirjoitettaessa pyörivältä levykkeeltä (300 rpm).

Allaolevassa taulukossa on esitetty 1.44 MB:n ja muutaman kovalevyn teknisiä tietoja. Sektorin koko on näissä kaikissa 512 tavua. Gigaisilla levyillä sektorien lukumäärä / ura vaihtelee vyöhykkeittäin siten, että sisemmillä vyöhykkeillä on vähemmän sektoreita kuin ulommilla vyöhykkeillä.

Kapasiteetti	1.44MB	127 MB	512 MB	Gigaiset
Levypintoja	2	16	16	6 - 16
Uria/pinta	80	935	1024	n. 2000
Sektoreita/ura	18	17	63	53 - 96
Pyörimisnopeus	300 rpm	3600 rpm		7200 rpm

Tallennustiheys 1,44 MB vaatii HD-merkinnällä (engl. high density) varustetun levykkeen, kun taas 720 KB:n formaatille riittää DD-merkinnällä (engl. double density) varustettu levyke. Merkintä viittaa magnetoituvien pisteiden kokoon. 1,44 MB:n levykkeellä magnetoituvan pisteen koko on pienempi kuin 720 KB:n levykkeellä. Vaikka levykkeiden formaatit eroavatkin, takaa DOS ylöspäin yhteensopivuu-



Kuva 10.7: 1,44 MB:n levyke edestä ja takaa. Levyn keskellä pyöritysmekanismi, joka samalla määrää 1-sektorin paikan. Kun kirjoitussuoja asetetaan siten, että se ei peitä levyssä olevaa reikää, ei levylle voi tallettaa tietoa, eikä siltä voi poistaa tietoa.

den. Siten 1,44MB:n levykeasema pystyy käsittelemään myös 720 KB:n formaattia.

Alustus

Ennen kuin levyä voi käyttää on se *alustettava* FORMAT-ohjelmalla. Levyjen perusalustus (engl. low level format) on tehty jo tehtaalla. Perusalustus luo levylle ura- ja sektorijaon sekä lisää urien alkuun paikannustietoja, joiden perusteella luku/kirjoituspää voidaan ohjata tarkalleen uran kohdalle. Levyn alustus FORMAT-ohjelmalla luo DOS-tiedostojärjestelmän tarvitsemat kirjanpitorakenteet.

Uuden kovalevyn perustiedot (levypintojen, urien ja sektoreiden lukumäärät) on aluksi asetettava koneen CMOS-muistipiirille. Se tapahtuu SETUP-ohjelmalla, joka monissa koneissa käynnistyy, kun järjestelmän käynnistämisen yhteydessä painaa DEL-näppäintä. Seuraavaksi, ennen alustusta, on kovalevylle määriteltävä FDISK-ohjelmalla levyn jako *levyosioihin* (engl. partition). Samassa yhteydessä on kiinnitettävä se levyaseman tunnus, jolta käyttöjärjestelmä tullaan lataamaan. Erillisiä osioita voi olla korkeintaan 4 kappaletta. Kukin osio muodostaa oman loogisen levynsä ja ne on kukin alustettava erikseen. Osioihin viitataan levyasematunnuksilla C:, D:, E: jne.

```
C:>HELP FDISK
Configures a hard disk for use with MS-DOS
```

```
FDISK

C:>HELP FORMAT
Formats a disk for use with MS-DOS.

FORMAT drive: [/V[:label]] [/Q] [/U] [/F:size] [/B | /S]
FORMAT drive: [/V[:label]] [/Q] [/U] [/T:tracks /N:sectors] [/B|/S]
FORMAT drive: [/V[:label]] [/Q] [/U] [/1] [/4] [/B | /S]
FORMAT drive: [/Q] [/U] [/1] [/4] [/8] [/B | /S]

  /V[:label] Specifies the volume label.
  /Q         Performs a quick format.
  /U         Performs an unconditional format.
  /F:size    Specifies the size of the floppy disk to format (such
             as 160, 180, 320, 360, 720, 1.2, 1.44, 2.88).
  /B         Allocates space on the formatted disk for system files.
  /S         Copies system files to the formatted disk.
  /T:tracks  Specifies the number of tracks per disk side.
  /N:sectors Specifies the number of sectors per track.
  /1         Formats a single side of a floppy disk.
  /4         Formats a 5.25-inch 360K floppy disk in a high-density
             drive.
  /8         Formats eight sectors per track.
```

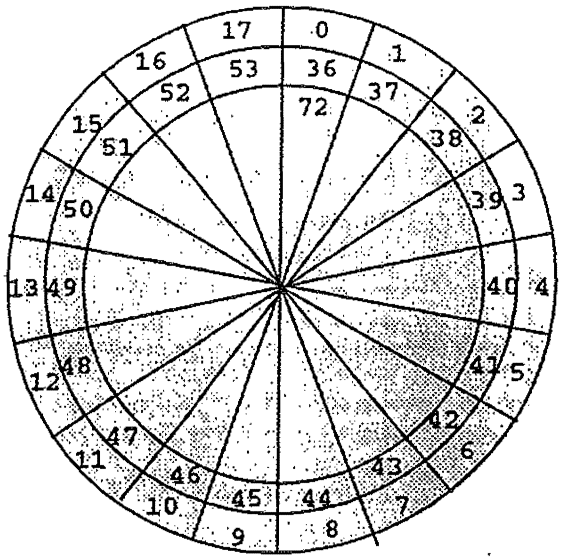
Lohkon koko

Pienin levyltä kerrallaan varattava kokonaisuus ei ole aina sama kuin pienin osoitettavissa oleva 512 tavun mittainen sektori. Tilaa varataan pitempinä, muutaman sektorin kokoisina lohkoina, *varausyksikköinä* (engl. cluster). Lohkon koko vaihtelee hieman käyttöjärjestelmäversion ja levyn koon mukaan: lohko voi olla 1, 2, 4, 8 tai jopa 32 sektoria pitkä. Jos lohkon koko on vaikkapa 4 sektoria, tarkoittaa se sitä, että esimerkiksi 10:n tavun tiedostolle varataan tilaa 2 KB (Huom: hakemistolistauksessa näkyy todellinen koko, ei varatun alueen koko). Pienten tiedostojen yhteydessä tapahtuvan tilan tuhlaamisen lisäksi varauslohkon koko vaikuttaa kirjanpitoon tarvittavien alueiden suuruuteen sekä tiedostojen käsittelynopeuteen.

Levyn tyyppi	Lohkon koko
720 KB	1024 B
1,44 MB	512 B
-16 MB	4096 B
16 - 128 MB	2048 B
128 - 256 MB	4096 B
256 - 512 MB	8192 B
512 -	16 384 B

Kirjanpidossa käytettävän varaustaulun alkio on 16 bittinen. Siten suurin lohkonumero, joka voidaan esittää on 65536. Jos lohko on esimerkiksi 2048 tavua, on suurin talletuskapasiteetti 128 Mtavua (65536 * 4 * 512). Tällöin jokaisen tiedoston viimeisessä lohkossa on keskimäärin 1024 tavua tyhjää tilaa. Lohkon koolla 8192 suurin talletuskapasiteetti on 512 MB ja tiedoston lopussa on keskimäärin 4096 tavua tyhjää tilaa.

Kun suuret kovalevyt jaetaan useammiksi osioiksi, voidaan käyttää pienempää lohkon kokoa ja säästää huomattavasti hukkatiloissa. Lohkon koko määräytyy osion koon perusteella oheisen taulukon mukaan.




```
Mikä levypinta?
(Lohko DIV 18) MOD 2

Mikä ura?
(Lohko DIV 36)

Mikä sektori?
(Lohko MOD 18) + 1
```

Kuva 10.8: Levylohkojen sijoittuminen 1,44 MB:n levykkeellä. DOS numeroi levypinnat ja urat alkaen numerosta 0, mutta sektorit alkaen numerosta 1.

Levykirjanpito

DOS käsittelee levyä, kuten se olisi jatkuva levyn ulkoreunasta alkava levylohkojen muodostama taulukko. Lohkot sijoittuvat levylle siten, että ensin numeroidaan ylimmän levypinnan uloimman uran sektorit, sitten jatketaan seuraavan levypinnan uloimmalle uralle jne. Tämän jälkeen numeroidaan ylimmän levypinnan toiseksi uloimman uran sektorit jne.

Kuvassa 10.8 on esitetty lohkojen sijaintia 1,44 MB:n levykkeellä. Kirjanpitoa varten on jokaisen levyn alusta varattu muutamia sektoreita järjestelmän käyttöön, loput voidaan käyttää datan tallennukseen. 1,44MB:n levykkeellä nämä varaukset sijoittuvat seuraavasti:

sektori	0:	alkulataustietue (engl. boot record)
sektorit	1-9:	varaustaulukko (engl. File Allocation Table, FAT)
sektorit	10-18:	varaustaulukon kopio
sektorit	19-32:	ylimmän tason tiedostohakemisto (juuri)
sektorit	33-2880:	datalohkot

Alkulataustietue sisältää käyttöjärjestelmän latauskäskyjen lisäksi myös tietoja levyn rakenteesta ja kovalevyn osiojaosta. Kovalevyn rakenteesta on kirjattu

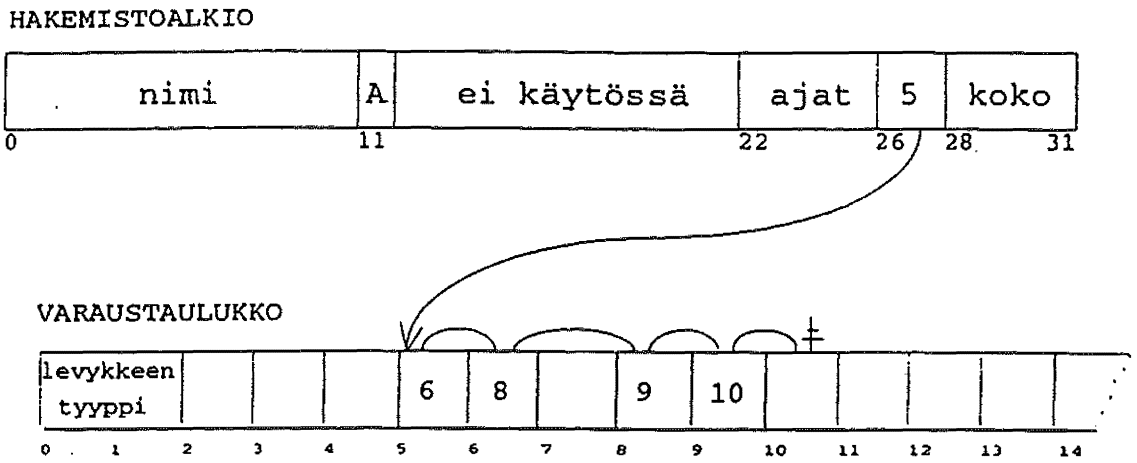
- sektorin koko tavuina, niiden lukumäärä ja sektorien lkm/ura,
- varauslohkon koko sektoreina,
- levypintojen lukumäärä (lukupäiden lkm),
- FAT-alueiden lukumäärä,
- juurihakemiston alkioiden lukumäärä,
- tiedot erillisistä osioista,
- piilosektorien lukumäärä.

Kustakin erillisestä osiosta on alkulataustietueeseen kirjattu mm. osion koko, sen alku- ja loppusylinteri, käyttöjärjestelmän tyyppi sekä onko ko. osio aktiivinen, ts. se osio, jolta KJ-ladataan.

Ylimmän tason levyhakemisto on kiinteän kokoinen ja se on aina vakiopaikassa heti varaustaulujen jälkeen. Muut hakemistopuun hakemistot on talletettu kuten tavalliset tiedostot. Levyhakemisto koostuu 32-tavuisista hakemistoalkioista (kuva 10.9). Hakemistoalkio sisältää tiedoston nimen (8+3 tavua), luontipäivän ja ajan (2+2 tavua), tiedoston koon tavuina (4 tavua), attribuutit (1 tavu) sekä tiedoston ensimmäisen lohkon numeron varaustaulukossa (2 tavua). Hakemistoalkion tavut 12-21 on varattu myöhempää mahdollista käyttöä varten.

Tiedostoon liittyviä määreitä sisältävässä attribuuttitavussa (tavu 11) on kutakin määrettä kohden oma bittinsä. Attribuuttitavun mahdollisia arvoja ovat paitsi allaluetellut arvot yksinään, myös niiden järkevät yhdistelmät.

1_H (Read Only)	tiedostoa voi lukea, mutta ei poistaa tai muokata
2_H (Hidden)	tiedosto on piilotettu, eikä sen nimi näy hakemistolistauksessa
4_H (System)	tiedosto kuuluu käyttöjärjestelmään, eikä sen nimi näy hakemistolistauksissa
8_H (Volume Label)	hakemistoalkion tavuissa 0-10 on taltion nimi
10_H (Directory)	hakemistoalkion tavuissa 0-10 on alihakemiston nimi
20_H (Archieve)	arkistointibitti, tämä bitti asetetaan ykköseksi, kun tiedostoa muutetaan. Tiedostoa varmuuskopioitaessa bitti nollataan.



Kuva 10.9: Hakemistoalkio ja varaustaulukko

Tiedoston attribuuttibittejä voi muuttaa komennolla ATTRIB ja taltiolle voi antaa nimen levyn alustuksen yhteydessä (FORMAT-komennon tarkenne /V) tai komennolla LABEL.

```
C:>HELP LABEL
Creates, changes, or deletes the volume label of a disk.

LABEL [drive:][label]

C:>HELP ATTRIB
Displays or changes file attributes.

ATTRIB [+R|-R] [+A|-A] [+S|-S] [+H|-H][[drive:][path]filename] [/S]

  +   Sets an attribute.
  -   Clears an attribute.
  R   Read-only file attribute.
  A   Archive file attribute.
  S   System file attribute.
  H   Hidden file attribute.
  /S  Processes files in all directories in the specified path.
```

Varaustaulukko sisältää tiedot levyn varatuista ja vapaista lohkoista. Kovalevyn varaustaulukko sisältää 16-bittisen alkion levyn jokaista lohkoa kohden - ei jokaista sektoria kohden. Koska varaustaulukon tuhoutumisen yhteydessä menetettäisiin koko levyn sisältö, on varaustaulukosta varmuuden vuoksi vielä kopio. Levykkeellä varaustaulukon alkio on 12-bittinen ja sen kaksi ensimmäistä alkiota varattu levyn tyypin ja samalla käytettävän formaatin kirjaamiseen (kuva 10.9).

Muissa varaustaulukon alkioissa on sisältönä seuraavan alkion indeksinumero, joka samalla on tieto siitä, onko vastaava lohko varattu vai ei. Vapaat lohkot on merkitty bittjonolla 000_H (kovalevyllä 0000_H). Samaan tiedostoon kuuluvat lohkot muodostavat ketjun, joka alkaa hakemistoalkiosta ja jonka perusteella tiedosto voidaan koota levyltä. Ketjun viimeisenä on bittijono $FF8_H$ (kovalevyllä $FFF8_H$). Varaustaulukon lohkonumeroista laiteajuri voi laskea, kuinka mones lohko on kyseessä levyn alusta lukien. Tämä on edelleen muutettavissa levypinnan, sektorin ja uran numeroksi (jotka toimitetaan levyohjaimelle). Myös levyn viottuneille lohkoille (engl. bad block) on oma merkintänsä (levykkeellä $FF7_H$, kovalevyllä $FFF7_H$).

Varaustaulukon kunnon voi tarkistaa komennolla CHKDSK (DOS 6:ssa myös ohjelmalla SCANDISK).

```
C:>HELP CHKDSK
Checks a disk and displays a status report.

CHKDSK [drive:][[path]filename] [/F] [/V]

  [drive:][path]  Specifies the drive and directory to check.
  filename        Specifies the file(s) to check for fragmentation.
  /F              Fixes errors on the disk.
  /V              Displays the full path and name of every file on
                  the disk.

Type CHKDSK without parameters to check the current disk.
```

Pirstoutuminen

Koska DOS varaa uudet lohkot aina siten, että se aloittaa vapaiden lohkojen etsimisen varaustaulukon alusta, syntyy aikaa myöten varausten ja vapautusten jälkeen tilanne, jossa uuden tiedoston lohkot eivät tule varatuksi peräkkäisistä paikoista. Tämä tiedoston pirstoutuminen hidastaa tiedostojen käsittelyä. Kovalevylle kannattaakin ajaa aika-ajoin levyn tiivistysohjelma, joka järjestelee tiedoston lohkoja uudelleen. DOS 6:n mukana tulee tätä tarkoitusta varten ohjelma nimeltä DEFRAG. Vanhemmissa DOS-ympäristöissä voi käyttää tätä tarkoitusta varten olevia kaupallisia ohjelmia.

Luku 11

Siirräntäjärjestelmä ja syöttö- ja tulostuslaitteet

11.1 Siirrännän hierarkia

Tiedon syöttöön ja tulostukseen, sen pysyvään tallennukseen sekä tiedonsiirtoon käytettävien oheislaitteiden ja keskusmuistin välinen siirräntä muodostaa hierarkkisen järjestelmän, jonka osina ovat sovellusohjelma, käyttöjärjestelmä ja laitteisto.

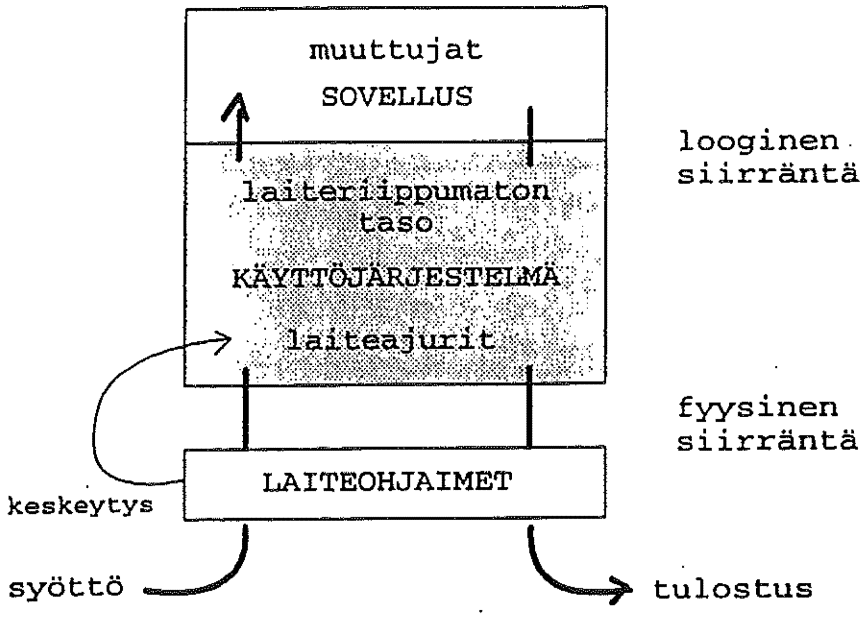
Sovellusohjelmataso

Ohjelman siirräntäkäskyt käsittelevät loogisia kokonaisuuksia: muuttujia, tietueita ja tiedostoja. Niistä käytetään ohjelman sisäisiä nimiä. Ohjelmissa ei tarvitse ottaa kantaa siihen minne esimerkiksi luettava tieto on talletettu tai kuinka todellinen siirräntä tullaan suorittamaan. Ohjelmissa siirräntä suoritetaan kutsumalla parametroitavia kirjastorutiineja esimerkiksi Readln(File1,X) tai Open(TdstoX,RW). Kirjastorutiinit huolehtivat parametrinvälityksestä ja käyttöjärjestelmään kuuluvien rutiinien palvelupyynnöstä.

Myös laitteisiin (näyttö, näppäimistö, sarjaliitäntä, rinnakkaisliitäntä) viitataan loogisin tunnuksin samalla tavoin kuin tavallisiin tiedostoihin. Käyttöjärjestelmän tehtävänä on huolehtia näiden loogisten tunnusten ja todellisten fyysisten laitteiden välisestä sidonnasta.

Käyttöjärjestelmätaso

Käyttöjärjestelmä sisältää rutiinit, jotka toteuttavat ja valvovat siirräntää. Rutiinit liittyvät fyysisen siirrännän ohjaukseen ja tiedostojärjestelmän sekä tiedon tallennuksen hallintaan. Käyttöjärjestelmän taso jakaantuu kahteen osaan: *laiteriippumattomaan siirräntään* ja *laiteriippuvaan siirräntään*. Laiteriippumaton osa tarjoaa sovelluksille yhtenäisen tavan käyttää kaikkia siirräntäpalveluja. Koska laitteistot eroavat toteutukseltaan ja vaadittavalta ohjaukselta, peittää laiteriippumaton taso



Kuva 11.1: Siirrännän hierarkia

nämä erikoisuudet sovellukselta. Laitteiden todelliseen käyttöön liittyvä ohjausohjelmisto (laiteriippuva siirräntä) on koodattu laiteajureihin (engl. device driver).

Järjestelmään kuuluvien laitteiden ominaisuudet on kuvattu *laitekuvaajissa*, joita on yksi kutakin järjestelmän laitetyyppiä kohden. Kuvaajiin kirjattavat tiedot vaihtelevat eri laitteilla. Ne sisältävät esimerkiksi laitteen yksilöivän tunnuksen (väyläosoite), ohjeet laitteen käytöstä (esim. montako uraa, sektoria, levypintaa, lohkon koko), viitteet mahdollisiin näppäimistön merkinmuunnostauluihin, laitteen tilatietoa: varattu, vapaa, rikki, laitteeseen liittyviä semaforeja (Pyyntö-Odottaa, Siirto-Valmis), viitteet jonottaviin palvelupyyntöihin ja viitteen laitetta käyttävän prosessin kuvaajaan.

Tiedostojärjestelmän ja tiedon tallennuksen hallinnan (laiteriippumaton siirräntä) tehtäviä ovat esimerkiksi selvittää loogisen tiedostonimen perusteella käytettävän laitteen tyyppi, pitää kirjaa levytilan vapaista ja varatuista alueista, siirrännän puskurointi (levylohkot) sekä mahdollinen luku/kirjoitusposition ylläpito. Jos laiteriippumaton taso ei löydä kaipaamaansa tietoa puskureista, pistää se laiteajurin siirräntätöihin.

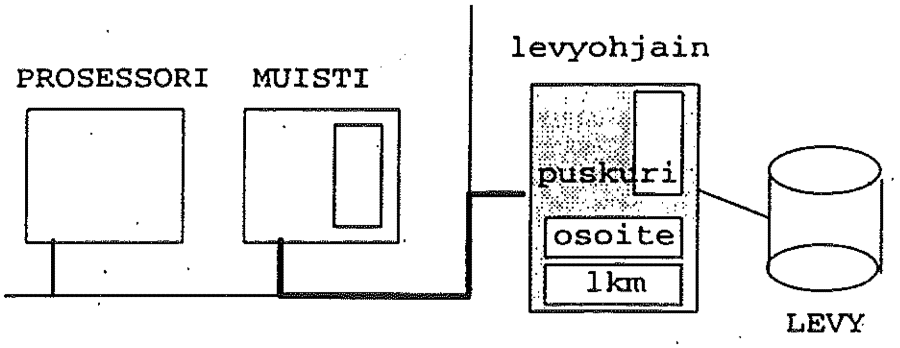
Fyysiseen siirräntään liittyviä eli laiteajurille kuuluvia tehtäviä ovat esimerkiksi (vaihtelevat eri laitteille) muodostaa parametrien ja laitekuvaajien perusteella laitetta ohjaavat käskyt (esim. levylohkonumerojen muuntaminen levypinnan, uran ja sektorin numeroksi), levypyyntöjen uudelleenjärjestely ja optimointi, varsinaisen ohjaimella tehtävän fyysisen siirrännän käynnistys, siirrännän kirjanpito sekä siirron oikeellisuuden tarkistaminen ja virheiden korjausyritykset.

Laitetaso

Laitteen ja keskusmuistin välisessä siirrännässä tarvittavasta ohjauksesta vastaa joko prosessori suorittamalla laiteajurin koodia tai tavallisemmin siirräntään erikoistunut ohjain tai prosessori.

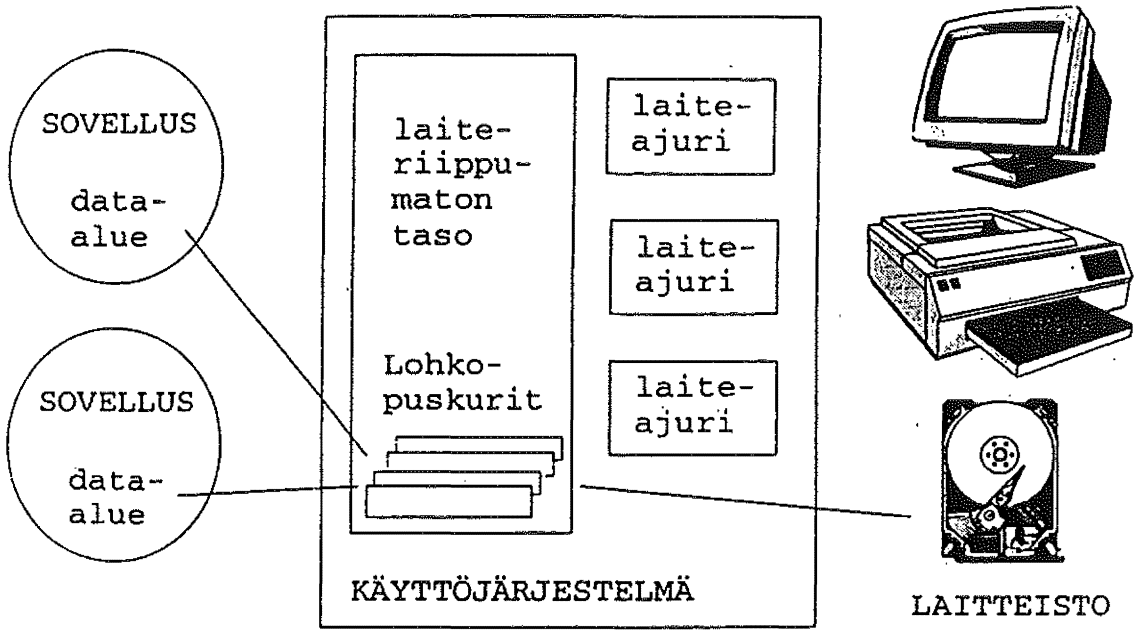
Jos prosessori vastaa täysin siirrännän ohjauksesta, ei voida hyödyntää rinnakkaisuutta ja muu käskyjen suoritus pysähtyy siirron ajaksi. Tällöin prosessorin käskykantaan kuuluu siirräntää varten implisiittiset käskyt (esim. IN ja OUT), joiden parametrina on laite, ja jotka siirtää merkin jonkin tietyn rekisterin ja laiteohjaimen välillä. Esimerkiksi näppäimistöltä lukeminen tapahtuu PC-mikrossa kahden rekisterin avulla: näppäimistön ohjaimen ohjausrekisteriin viedään tieto toimenpiteestä ja jäädään odottamaan näppäimen painalluksen aiheuttamaa keskeytystä. Sen jälkeen painettu merkki on siirrettävissä ohjaimen datarekisteristä edelleen muistiin.

Muistiinkuvatussa I/O:ssa (engl. memory-mapped I/O) on varattu tietyt muistin alueet ("laiterekisterit") prosessorin ja ohjaimien väliseen kommunikointiin. Näille alueille prosessori (eli ajuriprosessi) vie siirrännän ohjaustiedot ja kirjoitettavan datan. Laiteohjain tutkii jatkuvasti kommunikointialuetta. Sopivan käskyn huomattuaan ohjain aloittaa siirrännän. Suoritettuaan tehtävät, se laittaa tiedon onnistumisesta kommunikointialueelle ja aiheuttaa keskeytyksen. Prosessori huomaa keskeytyksen ja voi merkitä siirrännän vuoksi keskeytyneen prosessin taas etenemiskelpoiseksi.



Kuva 11.2: DMA-siirrossa ohjain huolehtii merkkien siirrosta ohjaimen ja keskusmuistin välillä.

DMA-siirtoa (engl. direct memory access) käytetään siirrettäessä tietoa keskusmuistin ja tukimuistilaitteen välillä. Siirron käynnistys ja lopputuloksen tarkastus tapahtuvat ohjelmallisesti laiterekisterien avulla, mutta siirron ohjauksesta vastaa ohjain itsenäisesti. Siirto keskusmuistiin / muistista voi siten tapahtua rinnan prosessorin käskysuorituksen kanssa. Ohjain siirtää tiedon ensin laitteelta puskuriinsa ja sieltä sitten sopivassa välissä keskusmuistiin. Tällä järjestelyllä ohjain voi ottaa tietoa vastaan samaa vauhtia kuin sitä siirretään esimerkiksi levyltä. Suoraan keskusmuistiin siirrettäessä joudutaan kilpailemaan prosessorin kanssa, ja hukattaisiin levyltä jatkuvasti tulevaa bittivirtaa. DMA-ohjain on esimerkki yksinkertaisesta tiettyä laitetta koskevaan siirräntään erikoistuneesta ohjaimesta.



Kuva 11.3: Siirrännän laiteriippumaton taso on välittäjänä sovelluksen ja laitteiston välillä. Laiteriippumaton taso herättää tarvittaessa laiteajurin huolehtimaan laitteen ja käyttöjärjestelmän työalueiden välisestä siirrosta. Kun siirto on valmis, toimittaa laiteriippumaton taso tiedon edelleen sovellukselle.

11.2 I/O-pyyntöjen käsittely

Prosessin kuvaajasta käsin päästään kiinni prosessin käyttämiin laitteisiin liittyviin prosessikohtaisiin ja laitesidonnaisiin tietoihin. Keskeisessä asemassa on tiedostokuvaajataulu. Siitä päästään edelleen avoimet tiedostot taulun alkioon, jonne on kirjattu luku/kirjoituspositito sekä laitenumero. Sen avulla löytyy laitekuvaaja. Kuvaajissa olevien semaforien avulla hoidetaan siirräntään osallistuvien osien välinen synkronointi ja poissulkeminen. Puskuroinnissa muodostavat laiteajuri ja laiteriippumaton taso alemman tason tuottaja-kuluttaja parin, sekä laiteriippumaton taso ja sovellusohjelma ylemmän tason tuottaja-kuluttaja parin.

Seuraavan sivun algortimeissa on hahmoteltu mitä tapahtuu, kun luetaan levyltä. Algoritmissa esiintyvien semaforien alkuarvona on 0.

Sovellusprosessi

Read(tiedostonumero, siirrettävä määrä, minne siirretään)

Käyttöjärjestelmä

Laiteriippumaton taso

Etene tiedostokuvaajataulu[tiedostonumero] --> avoimet tiedostot taulu
Katso laitenumero ja Etsi levyn laitekuvaaja
Määritä lukuposition perusteella monesko tiedoston lohko
Määritä tiedoston määreistä levylohkonnumero
Etsi numeron perusteella lohkopuskurista
Jos ei löydy
niin Varaa lohkolle puskuri
Muodosta pyyntöpaketti (lue, puskurin osoite, lohkonnumero)
Vie paketti ajurin jonoon
Up(Pyyntö_Odottaa) * herätä laiteajuri
Down(Pyyntö_Palveltu) * odota kunnes palveltu
Jos virhe niin Välitä tieto ylöspäin
Kopioi pyydetty tavumäärä lohkosta sovelluksen alueelle
Kasvata lukupositiota

Laiteajuri

toista
Down(Pyyntö_Odottaa) *odota pyyntöpakettia
Ota pyyntöpaketti jonosta
Laske laitekuvaajassa olevan tiedon perusteella levypinta, ura, sektori sekä siirrettävien tavujen lkm
Aseta laiteohjaimen rekistereihin mistä (levypinta, ura, sektori), minne (lohkopuskurin osoite) ja paljonko (lohkon koko) sekä operaatio (lue)
Käynnistä fyysinen I/O
Down(Siirto_Valmis) *odota siirron valmistumista
jos virhe niin välitä tieto ylemmälle tasolle
Up(Pyyntö_Palveltu) *herätä laiteriippumaton osa
Tuhoa palvelupyyntöpaketti
ikuisesti

Keskeytys ja sen käsittely (osittain laitteistolla)

Etsi levyn laitekuvaaja;
Up(Siirto_Valmis) * herätä ajuri

Laiteohjain

Tutki tehtävät operaatiot laiterekistereistä
Tee fyysinen DMA-siirto
Talleta tieto onnistumisesta laiterekistereihin
Aiheuta keskeytys

11.3 Syöttö

Tiedon syöttö- ja tulostuslaitteiden avulla tietokoneen käyttäjä voi kommunikoida koneen kanssa. Tietoa voidaan siirtää myös suoraan koneelle ilman ihmisen apua esimerkiksi joltain anturilta tai rekisteröintilaitteelta. Kun tietoa syötetään tietokoneelta toiselle puhutaan tavallisemmin tiedonsiirrosta.

Toistaiseksi suurin osa tiedoista syötetään koneelle *näppäimistöiltä* (engl. keyboard). PC-mikrotietokoneen näppäimistössä on 102 näppäintä. Kirjain- ja numeromerkkien lisäksi näppäimistöön kuuluu ohjausnäppäimet (Ctrl, Alt, Alt Gr, Esc, Insert, Delete, Home, End, Page Up, Page Down, Break, Scroll Lock, Print Scrn), toimintonäppäimet (F1- F12) sekä nuolinäppäimet. Syksyllä 1995 esiteltiin näppäimistö, jossa on kolme uutta näppäintä Windows-käyttöä varten. Näppäimen painallus aiheuttaa keskeytyksen ja näppäimistön ajuri saa näppäimen numeron, jonka se muuttaa merkkitaulunsa avulla esimerkiksi ASCII-koodiksi. Tällä tavalla ohjelmisto on riippumaton siitä, mitä kunkin näppäimen hatussa lukee ja erilaiset näppäimistöt pystytään hallitsemaan helposti.

Nykyisin myös *hiiri* (engl. mouse) tai sitä vastaava *ohjausrasia/pallo* (engl. track ball) kuuluu kaikkien mikrotietokoneiden ja työasemien vakiovarusteisiin. Niitä käytetään tiedon osoittamiseen, valikkovalintojen tekemiseen sekä piirtämiseen. Tavallisesti hiiri liitetään koneen sarjaporttiin tai laitteistossa voi olla sitä varten oma liitäntänsä, tai hiiri on toteutettu langattomana. Kannettavissa mikroissa ohjauspallo on usein kytketty suoraan näppäimistön yhteyteen.

Valmiita kuvia ja tekstejä voidaan syöttää koneelle *kuvanlukijoiden* eli *skannereiden* avulla. Kuvanlukija on tavallaan kuin puolikas tavallisesta kopiokoneesta. Toimintaidea on sama, mutta kuvaa ei kopioidakaan paperille, vaan se talletetaan tietokoneen muistiin. Lukijassa on lamppu, jolla kuva valaistaan ja heijastuva valo kootaan peileillä ja linsseillä pikselirivi kerrallaan valoherkille elementeille. Nämä valoherkät diodit tunnistavat niihin heijastuvan valon kirkkauden. Heijastuvan valon voimakkuus muutetaan sitä vastaavaksi numeroarvoksi. Yleensä harmaasävykuvissa arvo 0 kuvaa mustaa ja arvo 255 valkoista väriä. Värikuvia luettaessa kuvapisteiden kirkkausarvot tunnistetaan erikseen punaisesta, sinisestä ja vihreästä väristä ja kukin väri koodataan erikseen. Täysvärikuvan tallettamiseen käytetään 24 bittiä, eli 8 bittiä kustakin väristä. Yleisin kuvanlukijoiden tarkkuus on 600x300 kuvapistettä.

Halvimpia kuvanlukijoita ovat pienet käsiskannerit, mutta ne soveltuvat lähinnä vain pienten kuvien lukemiseen, sillä ihmisen lapsi ei pysty liikuttamaan kättä pitkään tasaisella liikkeellä. Yleisin skannerityyppi on tasoskanneri, joka on käytöltään tavallisen kopiokoneen kaltainen. Kuva asetetaan kuvapuoli alaspäin lasipinnalle ja ohjausohjelmalla käynnistetään kuvan selaus, jolloin lukupää kulkee kuvan alitse. Ammattityössä käytetään kalliita rumpumallisia skannereita, joissa kuva kiinnitetään lasirummulle ja rumpu pyörii paikallaan pysyvän lukupään ympäri.

Puhesyötössä tietokoneeseen liitetään puheentunnistin, joka suorittaa äänen aallonpituuksien analyysia. Ongelmana puheensyötössä on eri käyttäjien puhekielen vivahderikkaus, lauserakenteiden monimuotoisuus sekä sanojen monimerkitykselli-

syys, jolloin sanomien merkitys joudutaan selvittämään lauseyhteydestä. Puheen tuottaminen koneella on huomattavasti helpompaa kuin sen tunnistus. Puhesyöttö soveltuu hyvin lyhyiden ja rajattujen komentojen antamiseen. Tällöin käskyistä ja komennoista muodostetaan näytteiden perusteella muistiin räätälöidyt mallit, joihin saatua syötettä verrataan.

Hahmontunnistus perustuu näkyvän valon eri aallonpituuksien tunnistamiseen ('videonäkö'), tai siinä voidaan käyttää myös muita kuin näkyvään valoon perustuvia menetelmiä hyväksi, esimerkiksi ultraääniä (tutka) tai röntgensäteitä (tietokonetomografia). Muistiin hahmot talletetaan digitoituina kuvina. Koska kuvan yksittäisestä pisteestä joudutaan tallettamaan erittäin paljon tietoa, vaatii hahmontunnistus tietokoneelta suurta käsittelynopeutta. Hahmontunnistusta käytetään paljon robotiikassa ja vielä yleisemmin esimerkiksi viivakoodien ja tekstin lukemiseen valokynällä.

11.4 Tulostus

Tiedon tulostuksessa on tarkoitus esittää tieto ihmisen ymmärtämässä muodossa, teksteinä, kuvina tai äänenä. Tulostus voi olla kertakäyttöistä, jolloin tietoon perustuva päätös tehdään välittömästi. Tällöin tulostus voidaan tehdä näytölle tai tulostuksessa voidaan käyttää esimerkiksi puhesyntetisaattoria tai valmiiksi äänitettyjä sanomia. Pysyvämpää käyttöä varten tiedon esitys on talletettava koneesta riippumattomaan muotoon, esimerkiksi paperille, mikrofilmille tai -kortille.

Suurin osa pysyvistä tulostuksista tehdään paperille, sillä paperitulostus sopii sellaisenaan ihmisen luettavaksi. Mikrofilmi/kortti vaatii tiedon lukemista varten vielä erillisen laitteen. Koska mikrofilmi/kortti on edullinen ja teksti on talletettu pieneen tilaan, sopii se erityisen hyvin tiedon varastointiin, luettelointiin jne.

Tulostus paperille

Paperitulostus hoidetaan joko *kirjoittimilla* (engl. printer) tai *piirtureilla* (engl. plotter). Nykyisillä tulostimilla saadaan aikaiseksi myös väritulostuksia.

Kirjoittimien vertailukriteereinä käytetään usein tulostusnopeutta, -laatua sekä laitteen elinikää. Nämä vaihtelevat eri ympäristöissä ja eri laitteilla suuresti: esimerkiksi suuren keskuskoneen tulostusnopeudella on suuren käyttäjäkunnan vuoksi suurempi merkitys kuin mikron tulostusnopeudella, aputulostusten laatu saa olla heikompi kuin painoon menevän tekstin jne.

Myös kirjoittimien tulostustekniikat vaihtelevat suuresti: matriisikirjoitin muodostaa yhden merkin vaiheittain, merkkikirjoitin tulostaa merkin kerrallaan, rivikirjoitin rivin kerrallaan ja sivukirjoitin muodostaa sivun kerrallaan. Tulostuksessa käytetään joko kyniä, mustesuihkua, värinauhaa tai värijauhetta.

Massakirjoituksessa tulostuksen laadulle ei aseteta suuria vaatimuksia, vaan tärkeintä on tulostuksen suuri nopeus. Massakirjoitukseen tarkoitettujen kirjoittimien

nopeudet ovat noin 300 - 3000 riviä minuutissa. Tällaisissa kirjoittimissa kirjasintyypit on usein rajoitettu (tavallisesti 96 merkkiä). Massakirjoittimia käytetään tavallisesti atk-keskuksissa yleiseen numeeriseen ja tilastolliseen tulostukseen sekä ohjelmalistausten tulostukseen. Tekstintulostuksessa käytetään laatukirjoittimia, joiden tulostusjälki on hyvää kirjoituskonelaatua (engl. near letter quality, NLQ). Tulostus on usein huomattavasti hitaampaa kuin massakirjoituksessa.

Matriisikirjoittimessa (engl. dot matrix printer) muodostetaan merkit erillisistä pisteistä. Merkit muodostetaan $7 * 5 \cdots 30 * 50$ pisteestä. Tällöin kirjoittimen kirjoituspäässä on 7 - 30 värinauhaan nähden poikittaista ohutta neulamaista sauvaa, joita ohjataan värinauhan suuntaisesti. Merkit muodostetaan pisteistä vaiheittain siten, että laitteen ohjaamina vain niitä neuloja lyödään, joita tarvitaan tulostettavan merkin muodostukseen. Matriisikirjoittimessa ei ole erillistä kiinteä merkistöä: kirjasintyyppiä (engl. font) ja kokoa voidaan muuttaa jopa kesken tekstin. Useimmilla matriisikirjoittimilla voidaan tulostaa myös grafiikkaa. Jos vasarat ja neulat ovat tarpeeksi pieniä ja niitä on riittävän tiuhassa, voi matriisikirjoittimen tulostusjälki olla kirjelaatua. Matriisikirjoittimen nopeus massatulostuksessa voi olla jopa 900 merkkiä sekunnissa, kun taas laatutekstiä tulostettaessa (kaksi pyyhkäisyä) nopeus saattaa olla vain muutamia kymmeniä merkkejä sekunnissa. Matriisikirjoittimissa käytetään tavallisesti jatkolomakkeita. Myös tulostus useampikerroksiselle kopioivalle paperille onnistuu vaivattomasti.

Mustesuihkumenetelmässä (engl. ink jet printer) merkkimatriisin merkkiä ei muodosteta vasaran ja värinauhan avulla, vaan suuntaamalla suuttimista (50 - 100 kpl) tuleva pieni väripisara (tarkkuus esim. 360 dpi) suoraan paperiin. Muste tai värit sijaitsevat pienissä kapseleissa, joista pisara laukaistaan liikkeelle kuumentamalla synnytetyllä ilmakuplalla tai kiteen aiheuttamalla sähkökipinällä. Menetelmä on hiljainen ja se sisältää erittäin vähän liikkuvia osia. Paras tulostuslaatu saadaan sopivalla erikoispaperilla, joten yksittäisen sivun tulostus on kalliimpaa kuin laserkirjoittimella. Mustesuihkutulosteet eivät käytännössä ole arkistointikelpoisia, vaikka tulosteet eivät enää olekaan vesiliukoisia. Mustesuihkut ovat usein nopeampia (1-3 sivua / min) kuin vastaavat matriisikirjoittimet.

Laserkirjoittimissa (engl. laser printer) kokonainen tulostussivu muodostetaan ensin valoherkälle rummulle varaamalla se merkkien kohdalta sähköisesti lasersäteen avulla. Kirjoittimessa käytettävä värijauhe tarttuu rummun varautuneisiin kohtiin ja siirtyy siltä eräänlaisena kuivakopiointina paperille. Toteutuksena laserkirjoitin on huomattavasti kalliimpi kuin yllämainitut kirjoittimet. Sen etuina on kuitenkin suuri tulostusnopeus (10 - 1000 sivua /min) sekä painotuotelaatu (tulostustarkkuus tavallisesti 300 dpi tai 600 dpi ja parhaimmillaan 1200dpi).

Laserkirjoitin on oikeastaan erikoiskäyttöön rakennettu tietokone. Kirjoittimen ohjauksesta vastaa nopea prosessori ja kirjoittimella on myös muistia sivun talletamiseksi. Kun A4-paperin kuva-alue on noin 8,5" x 11", niin yhtä sivua varten tarvitaan 300 dpi:n tarkkuudella noin 1 MB muistia. Tavallisimmin muistia on 2 - 8 MB, mutta se on usein laajennettavissa jopa 64 MB:hen. Laserkirjoittimen perusmerkistöön kuuluu 5 - 15 ROM-muistiin tallennettua merkistöä, mutta mikrolta

voidaan tarvittaessa lähettää kirjoittimelle lisää merkistöjä.

Postscript-sivunkuvauskieltä tunnistavat laserkirjoittimet pystyvät myös kuvien tulostukseen. Postscript-sivunkuvauskieli on itseasiassa eräänlainen ohjelmointikieli. Tulostettava tieto, myös kuvat, lähetetään rinnakkaisliitännän kautta kirjoittimelle tavallisina tavuina. Kirjoittimelle tuleva tieto voi sisältää kirjoittimen ohjauskoodeja, merkistöjä, tulostettavaa tekstiä, viivagrafiikkakomentoja tai bittikarttoja. Kirjoitin tulkitsee saamaansa tekstiä ja muodostaa ensin kokonaan tulostettavan sivun bittikartan muistiin ja aloittaa vasta sen jälkeen varsinaisen tulostuksen. Tavallisin mikroon liitetyn Postscript-kirjoittimen tulostusnopeus on 8 - 10 sivua minuutissa, mutta monimutkaisen, kuvia sisältävän sivun tulostus voi viedä kymmeniä minuutteja.

Piirtureissa käytetään joko mustesuihkutekniikkaa tai vaihdettavia kyniä (4-9 kpl). Tasopiirturissa kynän liikettä ohjaa kaksi moottoria, joista toinen kuljettaa kynää paperin pituussuunnassa ja toinen sivuttaissuunnassa. Rumpupiirturissa paperia liikutellaan pituussuunnassa ja kynät liikkuvat siihen nähden poikkisuunnassa. Liike koostuu hyvin pienistä askelista, niin että piirtojälki haluttaessa poikkeaa hyvin vähän kaartuvasta käyrästä. Piirtureiden merkitys on vähentynyt laserkirjoittimien hintojen laskiessa ja niiden yleistyessä, ja myös siksi, että nyt myös matriisi-, mustesuihku- ja laserkirjoittimet pystyvät tuottamaan väritulosteita.

Tulostus näytölle

Valtaosa kaikista tietokoneen näytöistä perustuu televisiosta tuttuihin *katodisädeputkiin* (engl. cathody-ray tube, CRT). Katodisädeputki on lasista puhallettu, hyvin laaja- tai litteäpohjainen, kulmikas pullo, johon on vedetty tyhjiö. Sen kuvapinta on peitetty sisäpuolelta fosforoivalla aineella. Putken kaulaosassa on elektronitykki, joka putken toimiessa suuntaa elektronisuihkun näyttöpintaa kohti ja saa sen hehkumaan kohdalta, johon elektronisuihku osuu. Kun elektronisuihku pystyy pyyhkimään näyttöä noin 40000 kilometrin sekuntivauhdilla ja koska fosforihehku kestää joitakin sekunnin tuhannesosia sen jälkeen, kun suihku on lakannut vaikuttamasta, saadaan merkit tai kuvat näkymään pysyvästi virkistämällä kuvapisteitä uudelleen ja uudelleen esimerkiksi 50-75 kertaa sekunnissa. Välkkymätön näyttö vaatii virkistystä vähintään 70 kertaa sekunnissa.

Rasteripyyhkäisymemetelmässä elektronisuihku pyyhkii fosforoivaa pintaa toistuvasti juovia pitkin. Kun juovalla on kuvaan tai merkkiin kuuluva piste, sytyttää erityinen suihkunvalvontaelektroniikka fosforipisteen. Kuva muodostuu siis itseasiassa juovilla olevista yksittäispisteistä, jotka riittävän korkealuokkaisessa näytössä antavat vaikutelman jatkuvasta kuvasta. Näitä kuvan yksiköitä kutsutaan *pikseleiksi* (engl. pixel, picture element).

Kannettavissa tietokoneissa käytetään litteitä *puolijohdenäyttöjä*. Niissä käytetään merkkien ja kuvioiden muodostamiseen nestekiteitä, kaasuplasmaa tai elektroluminenssia.

Näyttö muodostuu matriisista, jonka alkioita sytyttämällä ja sammuttamalla voi-

daan muodostaa haluttu kuva. Nestekidenäytössä pisteet ovat ainetta, joka tulee läpinäkyväksi tai tummaksi sen mukaan onko se jännitekentässä vai ei. Kun kide päästää valon läpi näkee katsoja pisteen kohdalla näytön taustan. Kaasuplasmanäyttö puolestaan muistuttaa toimintaperiaatteeltaan loisteputkilamppua. Elektroluminenssinäytössä toteutusperiaate on sama kuin nestekide-näytössä: merkit muodostetaan aineesta joka virtakenttään joutuessaan muuttuu näkyväksi.

Perinteisen katodisädeputken haittana on sen suuri koko, nestekidenäytön ongelmana on ollut heikko kontrasti ja pieni katselukulma, kaasuplasma- ja elektroluminenssinäyttöjen murheena on edelleen suuri virrankulutus ja EL-näytöllä vielä valmistusprosessin hankaluus (erityisesti laadukkaan sinisen värin tuottaminen on ollut ongelma) sekä ainakin toistaiseksi kallis hinta.

Katodisädeputki tietokoneen näyttönä

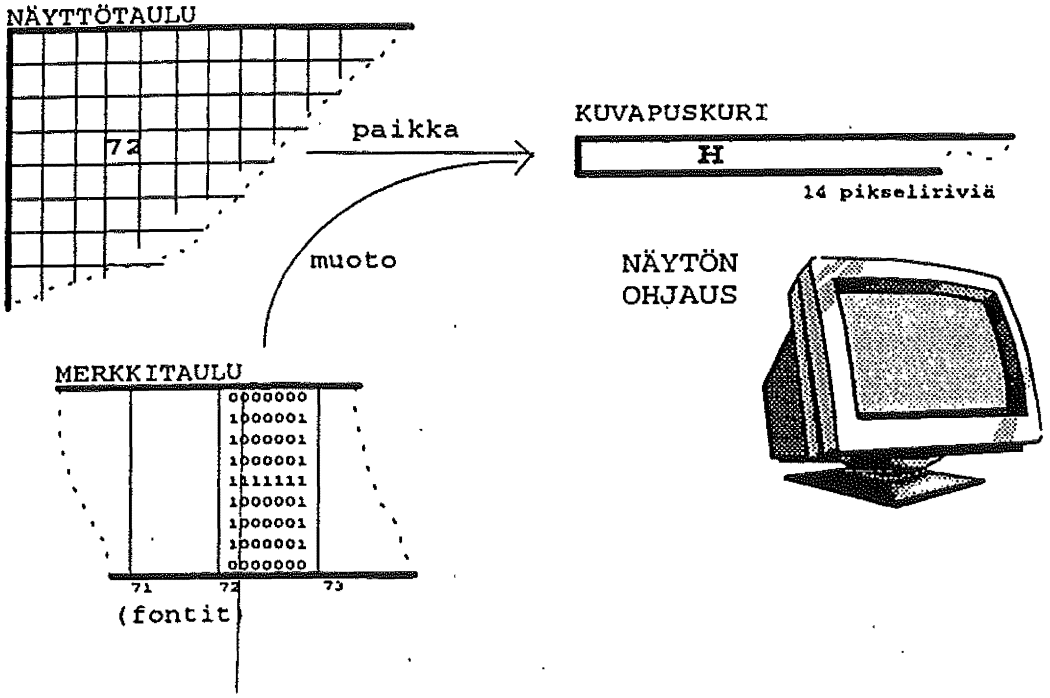
Pienissä ja keskisuurissa laitteistoissa käytetään tavallisimmin ns. *tyhmiä ASCII-päätteitä*. Tietokone voi tulostaa tällaisille päätteelle lähettämällä sinne tiedonsiirtolinjaa (RS-232) pitkin ASCII-merkkejä. Tavallisten näkyvien merkkien lisäksi ASCII-koodistossa on myös ohjausmerkkejä, joilla hoidetaan esimerkiksi kuvaruudun tyhjennys ja rivin vaihto sekä tiedonsiirron synkronointi. Painettaessa näppäintä ei merkki heti ilmesty näytölle, vaan pääte lähettää ensin näppäinkoodin tietokoneelle, joka puolestaan lähettää sitä vastaavan ASCII-koodin välittömästi takaisin näytölle (kaiutus). Näytön virkistämistä varten päätteellä on oltava vähän muistia.

ASCII-päätteiden käyttö suurissa pääteverkoissa on hankalaa, sillä jokainen näppäily vaatii keskusyksikön toimintaa. Suurissa verkoissa käytetäänkin usein päätteitä, joissa kommunikoidaan joko kokonaisia rivejä tai näytöllisiä lähettämällä. Menetelmässä on hankaluutena se, että tietoja päätteeltä lukeva ohjelma ei pysty tarkistamaan syötettyä tietoa sitä mukaa kuin sitä kirjoitetaan.

Mikrossa ja työasemassa näppäimistö ja näyttö on yleensä yhdistetty kiinteästi tietokoneeseen. Näyttöä vastaa muistissa tietty kiinteä muistialue, *muistiinkuvattu näyttö* (engl. memory-mapped). Prosessori voi tulostaa näytölle viemällä tulostettavan tiedon näyttömuistiin. Näyttöä ohjaavat piirit käyvät useita kymmeniä kertoja sekunnissa lukemassa alueen sisällön ja tulkitsevat sen näytölle.

Tekstin tulostusta varten näyttö jaetaan 25 vaakasuoraan riviin ja 80 pystysarakkeeseen. Kutakin ruutua ohjataan ASCII-koodilla. Koska kukin ASCII-koodi sopii yhteen tavuun, tarvitaan mustavalkoista tekstiä sisältävän näytön kuvaamiseksi vain noin 2 kilotavua muistia, eli näyttötaulu. Lisäksi tarvitaan 256:n alkion merkkitaulu, jonne talletetaan kirjainten, numeroiden ja muiden ASCII-merkkien kuvat pistematriiseina (fontteina). Merkkitaulun alkion kukin piste vastaa yhtä näytön pikseliä ja yksi kirjain muodostuu esimerkiksi 9*14:sta pisteestä. Näyttötaulun alkioon talletettu ASCII-koodi on indeksi merkkitaulussa olevaan merkin pikselikuvaan.

Näytönohjain muodostaa näyttötaulun ja merkkitaulun tiedoista kuvapuskuriin näytölle tulevat pikselit ja ohjaa elektronisuihkuja muodostamiensa pikselirivien perusteella. Merkkipohjaisissa näytöissä kuvapuskurin koko on esimerkiksi yksi teksti-



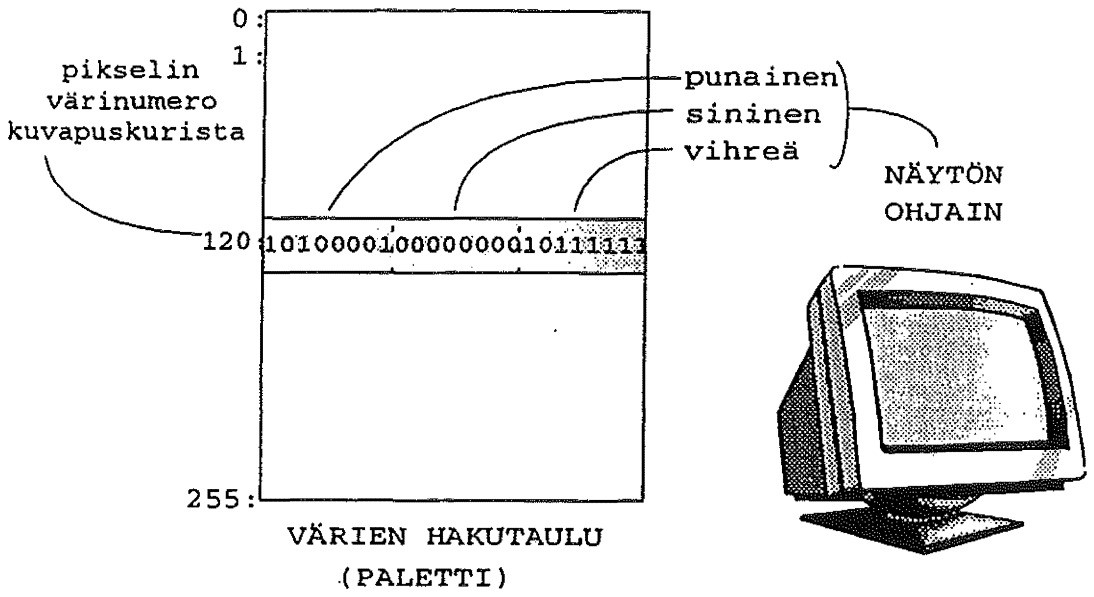
Kuva 11.4: Kirjaimen H (ASCII-koodi 72) muodostus kuvapuskuriin.

rivi (80 merkkiä eli 14 pikseliriviä). Kun näytönohjain on muodostanut yhtä merkkiriviä vastaavat pikselirivit, ne tulostetaan näytölle ja sen jälkeen käsitellään seuraava merkkirivi jne. Kun koko näytöllinen on tulostettu ruudulle, aloitetaan homma taas alusta (virkistys).

Tavallisesti ainakin yksi merkkitaulu sijaitsee ROM-muistissa ja RAM-muistissa voi olla lisää merkkitauluja eri merkkityypeille. Näytönohjaimen muisti on myös isäntäkoneelle tavallista RAM-muistia ja se voi viitata niihin kuten muualle keskusmuistiin.

Pikselipohjaisissa järjestelmissä, joissa myös kuvien tulostus on mahdollista, ei tarvita erikseen näyttötaulua, vaan kuvapuskurissa on yhtä monta alkiota kuin näytöllä on pikseleitä. Näyttö on täysin riippumaton kuvan monimutkaisuudesta, joten sitä on helppo muuttaa muuttamalla kuvapuskurin sisältöä. Graafiset ohjelmapakkaukset käsittelevät usein suoraan kuvapuskuria, jolloin näytön päivitys on nopeaa.

Jos mukaan halutaan myös värejä, ei pelkkä hehkuu/ei hehku- tieto riitä, joten kustakin pikselistä on talletettava enemmän kuin yhden bitin verran tietoa. Värinäytön kaikki värit voidaan generoida kolmesta perusväristä: sinisestä, punaisesta ja vihreästä. Kullekin värille on oma elektronitykkinsä, jotka ohjaavat kuhunkin näytön pikseliin tulevaa väriä. Jos kuhunkin pikseliin liittyvän perusvärin ja sen intensiteetin koodaamiseen käytetään esimerkiksi kahdeksaa bittiä, niin kutakin pikseliä kohden tarvitaan silloin 24 bittiä ja saadaan yli 16 miljoonaa väriä. Koska usein pienempikin värien lukumäärä riittää, voidaan muistin koossa säästää



Kuva 11.5: Pikselin kuvaus näytölle erillisen värihakutaulukon kautta. Tässä käytetään 256 väriä, joten kuvapuskurissa on väritietoa 8 bittiä per pikseli.

käyttämällä värikuvaukseen erillistä värien hakutaulua (palettia). Kuvapuskuriin talletetaan värin indeksi hakutaulussa ja värihakutauluun talletetaan käytettävien värin muodostuksessa tarvittavat punaisen, vihreän ja sinisen värin intensiteetit. Näytönohjain ohjaa elektronisuihkuja paletissa olevien voimakkuuksien mukaan.

Luku 12

Tietoliikenne

12.1 Sähköisen viestinnän historiaa

Vuonna 1810 saksalainen von Soemmering suunnitteli laitteen, joka koostui 26:sta johtimesta. Johtojen toinen pää asetettiin nestealtaaseen ja toiseen päähän johdettiin sähkövirtaa. Altaassa sähkövirta sai aikaan kuplia. Näin von Soemmering onnistui koodaamaan sanomia kuplien avulla. Hänen keksintönsä oli ensimmäinen sähköisen viestinnän sovellus ja sillä oli merkitystä sen tähden, että sotilaalliset piirit kiinnostuivat asiasta.

Vuonna 1839 englantilaiset W. F. Cooke ja Charles Wheatstone olivat perustaneet kolmentoista mailin lennätinlinjan rautateiden käyttöön. Heidän laitteensa koostui viidestä johtimesta, jotka toimivat voimanlähteenä pienille sähkömagneeteille. Näitä taas käytettiin liikuttamaan kahta neulaa kerrallaan. Neulojen avulla osoitettiin tarkoitettu kirjain.

Cooken ja Wheatstonen ratkaisu perustui trinäärikoodiin: neula liikkui oikealle, vasemmalle tai ei ollenkaan. Ratkaisu viisine johtimineen oli hankala ja kallis. Ensimmäisen käytännöllisen binääriseen koodaukseen perustuvan järjestelmän (kaksi johdinta, yksi neula) suunnitteli Samuel F. B. Morse, joskin Cooke ja Wheatstone myös päätyivät binäärikoodiin.

Pian kehittyivät laitteet, jotka tuottivat binäärikoodin pohjalta kirjaimia suoraan luettavaksi. Ongelmana näissä laitteissa oli kuitenkin synkronointi: mikäli joku osa koodista katosi tai muuttui matkalla, vastaanottajalla lopputulos oli sekasotku. Tästä syystä pidettiin aina muutaman kirjaimen lähettämisen jälkeen tavallista pitempi tauko, jotta osoittimet palautuisivat alkutilaan.

Lennättimen kohdalla ratkaisevaksi parannukseksi tuli v. 1900 paikkeilla yleisesti käyttöön otettu vaihtovirta. Tällöin luotiin myös ne tiedonsiirron perusteet, jotka ovat edelleen käytössä: asynkroninen ja synkroninen siirto, alku- ja loppubitti.

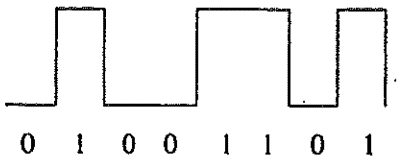
Tietokoneiden tiedonsiirto alkoi jo 50-luvulla. Tällöin samoin kuin vielä 60-luvullakin tiedonsiirto tapahtui pääasiassa keskuskoneen ja oheislaitteiden välillä. 60-luvun loppupuolella ARPA-organisaatio (*Advanced Research Projects Agency of*

the U.S. Department of Defence) alkoi rahoittaa tietokoneverkkotutkimusta, joka johti 1969 kokeiluverkon perustamiseen. Tämä ns. ARPA-verkko koostui neljästä Honeywell DDP-516 minikoneesta, joissa oli mm. 12 K 16 bitin sanoja muistia.

ARPA-verkko osoittautui toimivaksi ja 1971 siinä oli jo 23 konetta. Sotilaallinen verkko MILNET perustettiin saman teknologian varaan ja se liitettiin ARPA-verkkoon samoin kuin sen laajennus MINET Euroopassa. Lisäksi kaksi satellittiverkkoa, SATNET ja WIDEBAND, liitettiin ARPA-verkkoon. Monissa yliopistoissa ja muissa organisaatioissa oli 70-luvun puolivälin jälkeen lähiverkkoja ja myös näitä liitettiin ARPA-verkkoon. Näin syntyi ARPA internet, johon kuului 1980-luvulla tuhansia koneita ja yli 100 000 käyttäjää. Vähitellen tätä verkkojen verkkoa ruvettiin kutsumaan lyhyesti Internetiksi.

12.2 Asynkroninen ja synkroninen siirto

Asynkroninen sarjallinen siirto tapahtuu merkki kerrallaan. Merkkien välissä voi olla mielivaltaisia taukoja. Perustapauksessa lähettäjä koodaa kahdeksalla bitillä esitetyn merkin kahdeksaksi volttipulssiksi. Vastaanottaja ottaa pulssit vastaan ja



Kuva 12.1: Bittien koodaus volttipulsseiksi.

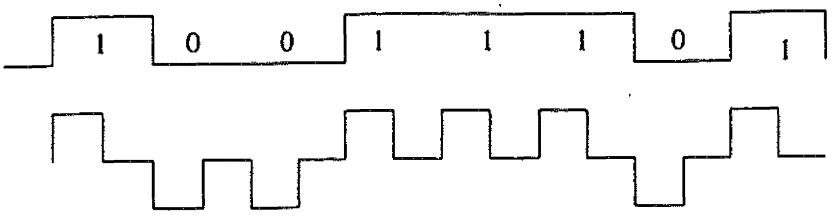
yrittää mitata niiden arvon keskeltä. Käytännössä pulssit vääristyvät siirron aikana, joten on tärkeää oikean tuloksen saamiseksi, että mittaus osuu lähelle pulssin keskikohtaa.

Siten vastaanottajan täytyy synkronoitua vastaanottamaan merkkiä. On havaittava merkin alku, on mitattava kahdeksan bitin arvot ja lopetettava tämän jälkeen. Bittien mittaus vaatii *bittisynkronointia*, joka saadaan aikaan *aloitus-* ja *lopetusbiteillä*. Tyypillisesti vastaanottajan kellotaajuus on N kertaa tulevan signaalin bittitaajuus. Kun vastaanottaja havaitsee virrassa muutoksen (aloitusbitti), se suorittaa mittauksen $N/2$ syklin jälkeen. Seuraavat mittaukset tehdään N syklin jälkeen.

Synkronisessa siirrossa bittisynkronointi saavutetaan toisella tavalla, mikä mahdollistaa harvemmat kontrollibitit ja siten tehokkaamman siirron. Synkronisessa siirrossa bittien koodaukseen liitetään ajastusinformaatiota.

Kun bitti koodataan volttipulsseiksi, tehdään se niin, että bitin keskellä tapahtuu muutos, joka auttaa vastaanottajaa tahdistamaan bittien mittaukset. Näin voidaan siirtää suuria siirtomääriä tehokkaasti ilman monia ylimääräisiä bittejä. Esimerkki synkronisesta koodauksesta on bipolaarinen koodaus kuvassa 12.2.

Asynkroninen siirto sopii tilanteisiin, joissa tietoja lähtee harvakseen. Tyyppiesimerkki on tiedonsiirto näppäimistöltä prosessorille. Synkroninen siirto sopii taas



Kuva 12.2: Bipolaarinen koodaus.

paremmin tilanteisiin, joissa tietoa siirretään paljon yhdellä kertaa, kuten esimerkiksi tiedoston siirrossa.

12.3 Modeemi

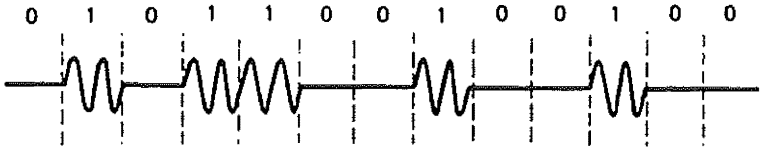
Digitaalisia signaaleja ei voi suoraan välittää puhelinlinjojen kautta. Puheen taajuus on välillä 200 - 8000 Hz ja puhelinlinja välittää taajuudet väliltä 400 - 3400 Hz. Tämä riittää puheen ymmärrettävään siirtoon, vaikkakaan kaikkia vivahteita ei ole mukana. Jos digitaalinen kanttiaalto lähetettäisiin sellaisenaan puhelinlinjalle, se vääristyisi paljon eikä vastaanottaja kykenisi tulkitsemaan sitä enää. Näin käy siksi, että puhdas kanttiaalto sisältää äärettömän monta eri taajuutta, joiden yhteisvaikutus vasta tuottaa 'kantin'. Tosin taajuudella on sitä vähemmän merkitystä kanttiaallon muotoon, mitä suurempi se on.

Modeemin avulla digitaalinen data voidaan lähettää *kantoaaltoa* muuttamalla. Muutosperiaatteita eli *modulaatiotekniikoita* on kolmenlaisia. Näitä voidaan myös yhdistellä.

Taajuusmodulaatiossa binäärinen data koodataan kantoaaltoon kantoaallon taajuutta vaihtelemalla.

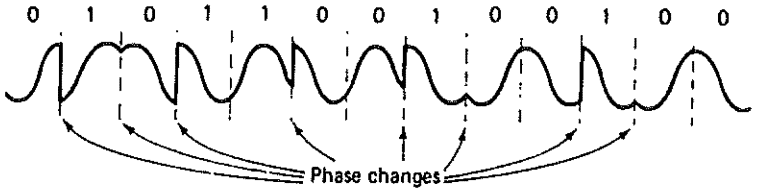
Kuva 12.3: Taajuusmodulaatio.

Amplitudimodulaatiossa sama saadaan aikaan kantoaallon korkeutta eli amplitudia vaihtelemalla.



Kuva 12.4: Amplitudimodulaatio

Vaihemodulaatiossa vaihdellaan kantoaallon vaihetta.



Kuva 12.5: Vaihemodulaatio

Edellä kuvattuihin modulointitekniikoihin perustuvia modeemeita kutsutaan *kantoaaltomodeemeiksi.* Näissä siis digitaalinen kantataajuinen signaali moduloidaan kantoaaltoon. Toisen lajin modeemeita ovat ns. *kantataajuusmodeemit.* Niissä digitaalinen signaali lähetetään siirtotielle alkuperäisellä taajuudellaan sopivasti koodattuna. Kantataajuusmodeemi sisältää signaalimuokkaimet ja sovittimet, joiden avulla kantataajuista signaalia voidaan siirtää fysikaalisilla yhteyksillä.

Kantataajuusmodeemeita käytetään dataliikenteessä kiinteän linjan yhteyksillä. Kantataajuusmodeemien suunnittelussa tavoitteena on kantoaaltomodeemeja yksinkertaisempien ja halvempien modeemien toteuttaminen. Tällaisten modeemien käytössä on eräitä rajoituksia. Ne vaativat tavallisesti laajemman siirtokaistan kuin puheensiirtokaista 300 - 3400 Hz. Tämän vaatimuksen täyttyminen riippuu datasiirtoyhteyden puhelinverkosta varaamien johto-osuuksien ominaisuuksista.

Yleensä modeemilla tarkoitetaan kantoaaltomodeemeita. Modeemeja on saatavilla sekä synkroniseen että asynkroniseen siirtoon. Eräät modeemityypit soveltuvat molemmille linjatyypeille.

Modeemiin liittyy oleellisesti sen siirtonopeus. Seuraavassa taulukossa ovat tärkeimmät siirtonopeudet:

Siirtonopeus	Standardi
300	V.21
1 200	V.22
2 400	V.22 bis
9 600	V.32
14 400	V.32 bis
28 800	V.34
33 600	V.34 bis

Uusin nopeusluokka on 56 kilobittiä sekunnissa. ITU-stardardia ei tästä vielä ollut 1997, mutta monet valmistajat ovat oman versionsa jo tuoneet markkinoille. 56 kb/s nopeus on ainoastaan käyttäjälle päin. Käyttäjältä pois päin toimii 28 kb/s nopeus. Tällainen ratkaisu on kehitetty, koska puhelinlinjat eivät muuten mahdollistaisi näin nopeita analogisia lähetyksiä. Esimerkiksi Internetin käytössä ratkaisu on sopiva.

Päätelaitenopeus

Päätelaitenopeus tarkoittaa sarjaliitännän ja modeemin välistä nopeutta eikä puhelinlinjalla käytettyä siirtonopeutta. Nykyaikaisilla nopeilla modeemeilla päätelaitenopeus voi olla moninkertainen linjanopeuteen verrattuna. Onkin suositeltavaa asettaa päätelaitenopeudeksi noin neljä kertaa siirtonopeutta suurempi arvo. Suurien päätelaitelaitenopeuksien käyttäminen edellyttää, että käytössä on tehokas sarjaliikennepiiri.

12.4 Sarjaliikennepiirit

Sarjaliikennepiiri eli UART (*Universal Asynchronous Receiver/Transmitter*) on mikrotietokoneiden piiri, joka huolehtii tietoliikenteestä. Se on kaikissa koneissa jo peruskokoonpanossa, mutta se sisältyy myös korttimodeemiin. Jos käytetään ulkoista modeemia, käytetään samalla koneen omaa UART-piiriä. Korttimodeemin tapauksessa käytetään kortin UART-piiriä.

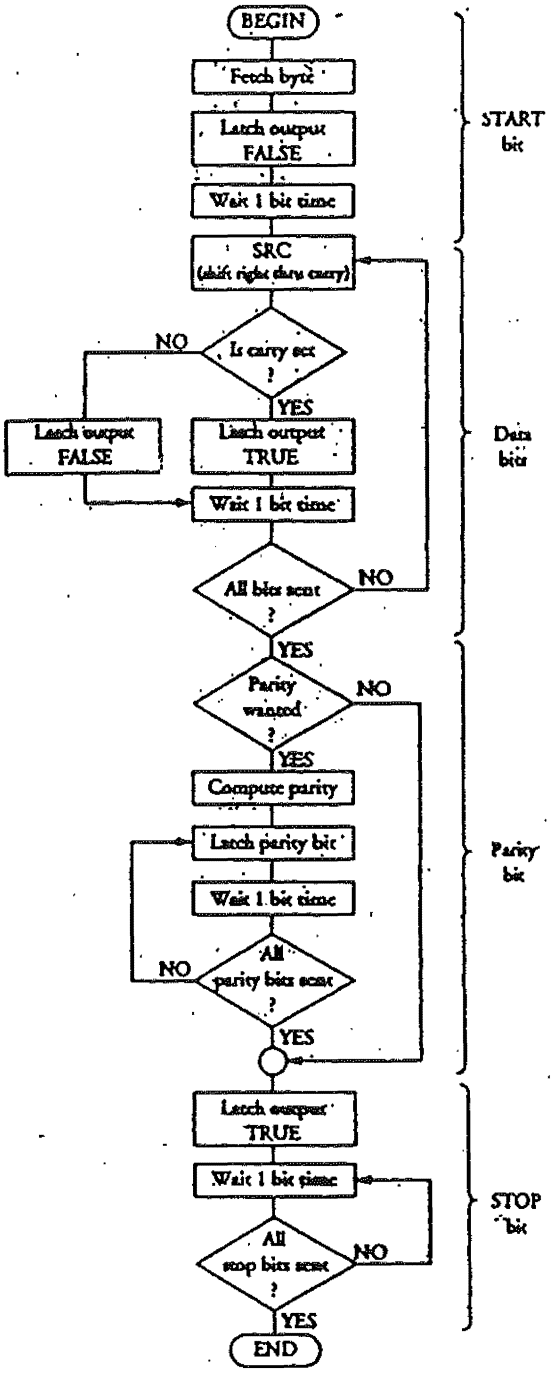
UART-piiri hoitaa kaikki tehtävät, jotka liittyvät tavun lähetykseen ja vastaanottoon. Kuvassa 12.6 on kulkukaavio, joka näyttää, mitä toimenpiteitä konekielitasolla yhden tavun lähettämiseen liittyy. Algoritmi ei vaikuta kovin monimutkaiselta, mutta analysoidaan tarkemmin esimerkiksi kohtaa 'Odota yhden bitin ajan'.

Jos kellotaajuus ja käskyjen kesto ovat tiedossa, odotuksen suorittavan aliohjelman kirjoittaminen ei ole vaikeaa. Otetaan vain käyttöön rekisteri ja yksinkertainen ohjelmasilmukka, jonka käskyjen kesto on tiedossa. Silmukkaa käydään läpi, kunnes rekisterissä oleva laskuri osoittaa, että aika on kulunut.

Algoritmissa odotus kuitenkin vaihtelee sen mukaan, mikä on linjan nopeus. Esimerkiksi 300 b/s linjan tapauksessa bitin lähetys kestää 3,333 millisekuntia, mutta 19,2 kb/s linjan tapauksessa vain 52 mikrosekuntia. On jo selvästi vaikempaa kirjoittaa aliohjelma, joka odottaa useita erilaisia aikoja tarkasti.

Periaatteessa voitaisiin käyttää samanlaista silmukkarakennetta kuin yhden nopeuden tapauksessa kuitenkin niin, että lisäksi on ulompi silmukka, joka kutsuu kullakin kerralla tarvittavaa yhden nopeuden silmukkaa. Valitettavasti tällaiset sisäkkäiset silmukat eivät ole riittävän tarkkoja pitemmällä aikavälillä. Siitä syystä käytetäänkin laitteiston omaa kelloa, joka on yleensä kristallioskillaatori. Sen taajuus on nopeimman lähetystaajuuden monikerta.

UART vapauttaa ohjelmoijan algoritmin ongelmista, koska se hoitaa tavun lähetyksen laitetasolla. Sarjaliikennepiirin ja prosessorin välinen kommunikointi perustuu keskeytyksiin, joita UART generoi toimintansa yhteydessä. *Lähetyskeskeytys* tapahtuu silloin, kun lähetyspuskuri vapautuu seuraavalle tavulle. Uuden tavun saapuminen ja vieminen FIFO-jonoon (First In, First Out) aiheuttaa *vastaanottokeskeytyksen.* Johtimen virran muutos saa aikaan *RS-232*-keskeytyksen ja virheet tai BREAH-sanoma saavat myös aikaan keskeytyksen. Kun UART on generoinut keskeytyksen, se ei generoi toista, ennenkuin edellinen on selvitetty.



Kuva 12.6: Yhden tavun lähettäminen konekielitasolla.

Käytännön UART

Sarjaliikennepiiri 8250 oli jo ensimmäisissä PC-mikroissa vuonna 1981. Se onkin jo vanhentunut ratkaisu, eikä sovellu uusien modeemien kanssa kuin alle 38 400 b/s päätelaitenopeudella. Piirissä on kohtuullisen paljon virheitä. Koska piiri on halpa, se on ollut pitkään markkinoilla ja vielä ainakin vuonna -94 sitä esiintyi uusissakin mikroissa.

Vuonna 1984 julkaistiin 16450-sarja, joka on yhteensopiva 8250:n kanssa. Tämä piiri pystyy suurempaan toimintanopeuteen, mutta siitä ei ole hyötyä nopeiden nykyaikaisten modeemien yhteydessä.

Vuonna 1987 julkaistiin 16550-sarja, joka on huomattavasti tehokkaampi kuin edelliset mallit. Ensimmäiset versiot olivat tosin hieman virheellisiä, joten uusi toimiva ratkaisu julkaistiin myöhemmin nimellä 16550A.

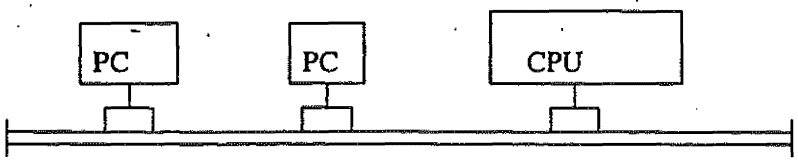
16550A:n tehokkuus perustuu enenkaikkea 16 merkin FIFO-puskuriin. Tämän johdosta piiri voi vastaanottaa merkkejä 1440 mikrosekuntia, ennen kuin tietoa häviää. Pitkää vastaanottoaikaa tarvitaan, koska PC ei aina ehdi palvelemaan ajoissa sarjapiiriä. 16550 ei pysty käyttämään FIFO-puskuria, joten se kannattaa ehdottomasti vaihtaa 16550A-piiriin.

16650 on uusi sarja, jossa on 32 tavun FIFO-puskuri ja sen suurin tiedonsiirtonopeus on 230 kb/s. Ohjelmistotuki on vielä piirille puutteellista.

16750-piirissä on 64 tavun puskuri ja sen suurin tiedonsiirtonopeus on 460 kb/s. Tämä on uusi piirisarja, jonka takia ohjelmistotuki on vielä puutteellista.

12.5 Lähiverkot

Jos organisaatio käyttää tietokoneita hyväkseen, sillä on usein lähi- eli paikallisverkko. Tyypillinen lähiverkkoratkaisu on *kilpavarausväylä*, jonka kaupallinen versio on nimeltään *Ethernet*. Siinä yhteen tiedonsiirtokaapeliin on kiinnitetty useita koneita:

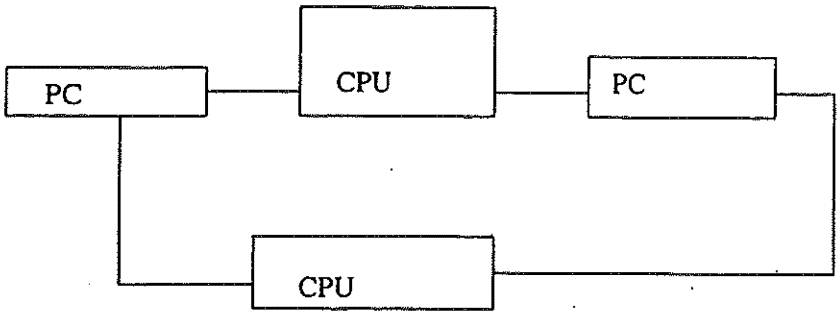


Kuva 12.7: Ethernet.

Kone voi lähettää väylälle sanoman milloin tahansa. Sanoma kulkee väylän kaikille koneille, mutta vain se, jolle sanoma on tarkoitettu, lukee sanoman. Jos kaksi konetta lähettää sanoman yhtäaikaa, tapahtuu *yhteentörmäys* ja sanomat tuhoutuvat. Tällaisessa tilanteessa lähettäneet koneet odottavat satunnaisen ajan ennenkuin

lähettävät sanoman uudelleen.

Toinen vaihtoehto on *vuororengas* eli *token ring*. Siinä *vuoromerkki* kiertää ren-



Kuva 12.8: Token ring.

kaassa, ja vain se kone, jolla on vuoromerkki, voi lähettää. Ratkaisu vaikuttaa yksinkertaisemmalta kuin kilpavarausväylä, mutta vuoromerkin ylläpito ja seuranta vaatii yllättävän monimutkaisen järjestelmän. Vuororenkaan tehokkuus on kuitenkin parempi kuin kilpavarausväylän.

Tavallisen väylän tai renkaan nopeus on 10 Mb/s. Väylässä kuormituksen tulee olla varsin alhainen, alle 30% maksimikapasiteetista, jotta yhteentörmäykset eivät rasittaisi kohtuuttomasti järjestelmän toimintaa. Vuororenkaassa kuormitus voi olla suurempi, alle 60%.

Väylän tai renkaan välityksellä koneet voivat lähettää sanomia toisilleen. On esimerkiksi mahdollista, että yhdessä koneessa on suuri levymuisti ja ilman omaa levymuistia olevat työasemakoneet käyttävät kaikki yhteistä levyä lähiverkon välityksellä.

Lähiverkkoja voidaan yhdistää *toistimien* (engl. repeater) tai *siltojen* (engl. bridge) avulla. Toistin lähettää kaikki väylän sanomat toiselle väylälle. Seurauksena koko systeemin kuormitus kasvaa, eikä ratkaisu sovellu paljon liikennettä sisältävien lähiverkkojen yhdistämiseen.

Silta on toistinta älykkäämpi. Se lähettää vain ne sanomat toiselle väylälle, joiden vastaanottaja ei ole alkuperäisellä väylällä. Siten kuormitus koko systeemissä ei kasva niin suureksi kuin toistimien tapauksessa. Siltojen ohjelmisto on varsin monimutkainen johtuen mm. siitä, että halutaan estää tuntemattoman osoitteen omaavien pakettien ikuinen kierto verkosta toiseen.

12.6 Liittyminen tietokoneverkkoon

Edellä on kuvattu modeemeita ja UART-piirejä, jotka ovat käytössä PC:n yhteydessä asynkronisessa tiedonsiirrossa puhelinverkon kautta. Liityttäessä esimerkiksi tietokoneverkkoon tehokas ratkaisu perustuu kuitenkin paikallisverkkoihin, kiinteisiin linjoihin ja reitittimiin.

Reititin ohjaa laajaverkkoon menevän liikenteen ulos lähiverkosta ja edelleen laajaverkon reitittimeen. Reitittimet ovat yleensä erillisiä laitteita, mutta alhaisilla nopeuksilla sellainen voidaan toteuttaa myös ohjelmallisesti. Reitittimien hinnat vaihtelevat niiden älykkyyden, muistin ja kapasiteetin mukaan alkaen noin 10 000 markasta.

Linjayhteys oman lähiverkon ja laajaverkon reitittimen välillä voidaan toteuttaa joko modeemilinjalla (valintainen tai kiinteä yhteys, 19,2 - 56 kb/s), suoralla liittymällä, ISDN-liitännällä, kiinteällä kaapelilla jne. Lisäksi vaaditaan, että lähiverkon koneissa on sopiva ohjelmisto. Esimerkiksi Internetiin liityttäessä koneissa täytyy olla TCP/IP-ohjelmisto. Se määrittelee yhteyskäytännön eli miten yhteys avataan, suljetaan, mitä sanomia lähetetään yms.

Tietokoneverkon kautta sanomia lähetettäessä yhteys on joko *piirikytkentäinen* tai *pakettikytkentäinen* riippuen verkosta ja ohjelmistosta. Edellisessä yhteyttä varten perustetaan kiinteä reitti, jota pitkin kaikki sanomat kulkevat. Jälkimmäisessä sanomat lähetetään yhtenä tai useampana pakettina ja paketit voivat kulkea eri reittiä määränpäähänsä.

12.7 Protokollapino

Tietokoneverkko-ohjelmistot ovat niin laajoja, että ne rakennetaan virtuaalikoneen tapaan kerros kerrokselta. Ylemmän kerroksen ohjelmat käyttävät alemman kerroksen palveluja. Jos kone A kommunikoi koneen B kanssa ja kummassakin on samaa standardia oleva kerroksellinen verkko-ohjelmisto, niin vastaavat kerrokset kummassakin koneessa kommunikoivat keskenään:

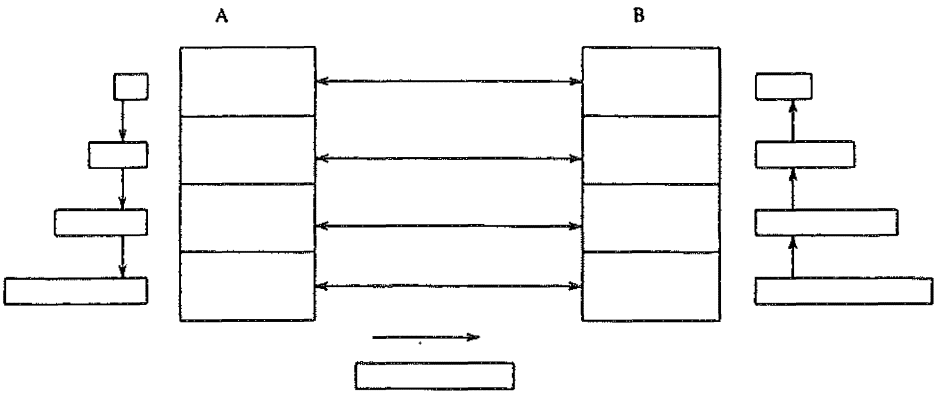
Kun A:n ylin kerros lähettää paketin ylimmälle kerrokselle B:ssä, käyttää se lähinnä alemman kerroksen palveluja. Jokainen kerros lisää varsinaiseen datapakettiin omia tietojaan. Ennen datapaketin lähettämistä kerrokset saattavat lähettää muita paketteja, joiden avulla varmistetaan lähetyksen onnistuminen. Jokainen kerros korjaa sen piiriin kuuluvat virheet.

Tietokoneverkko-ohjelmistoa kutsutaan usein *protokollapinoksi.* Termillä pino ei tässä yhteydessä ole mitään tekemistä tietorakenteen pino kanssa, vaan se tarkoittaa yksinomaan ohjelmiston kerroksellista rakennetta. Protokolla on puolestaan standardi tai ohjelmisto tietokoneverkkoympäristöön.

Internetin kerrosratkaisu on kuvan 12.10 mukainen.

Sovelluskerros (engl. application layer) sisältää perussovelluksia ja niitä tukevia toimintoja. Vanhimpia ja edelleen käyttökelpoisia sovelluksia ovat tiedostojen siirtoprotokolla FTP (engl. File Transfer Protocol), kaukaisen tietokoneen päätekäytön mahdollistava protokolla Telnet ja sähköpostiprotokollat.

Kuljetuskerros (engl. transport layer) huolehtii pakettien perillemenosta kaukaisten koneiden välisessä tiedonsiirrossa. Internetin tavallisin kuljetusprotokolla on TCP (Transmission Control Protocol). Se varmistaa, että kaikki lähetetyt paketit



Kuva 12.9: Kerroksellinen tietokoneverkko-ohjelmisto toiminnassa.

saapuvat oikein perille. Vastaanottaja kuittaa jokaisen saapuneen paketin ja tästä lähettäjä tietää, onko lähetys mennyt perille vai ei. Jos paketti on matkalla kadonnut tai vääristynyt, lähetetään paketti uudestaan. Vaihtoehto TCP:lle on UDP (User Datagram Protocol). UDP ei varmista pakettien perillemenoa, joten se ei sovi epäluotettaville linjoille tai sovelluksille, joissa kaikkien tietojen perillemeno on tärkeää.

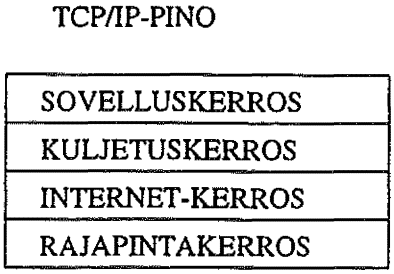
Internet-kerros vastaa tehtävistä, joita tarvitaan lähetettäessä paketteja usean aliverkon kautta. Internet-verkkohan koostuu useista aliverkoista, joiden arkkitehtuuri vaihtelee suuresti (laajaverkot, lähiverkot, satelliittiverkot). Tällaisessa tilanteessa tarvitaan ohjelmistoa huolehtimaan mm. reitityksen ja osoiteformaattien yksityiskohdista.

Rajapintakerros (engl. interface layer) kattaa kaikki toiminnot, joita tarvitaan lähetettäessä sanomia luotettavasti erilaisia tietoliikennekanavia pitkin koneelta toiselle.

12.8 PC:n liittäminen Internetiin

Tehokas yhteys Internetiin tapahtuu siis lähiverkon ja reitittimen kautta. Kuvassa 12.11 vasemmalla puolella on protokollapino tällaisessa tilanteessa. TCP/IP-pino käyttää ODI- tai NDIS-pakettiajuria, jotka puolestaan nojautuvat verkkoliittymän ajuriin.

ODI (Open Data-Link Interface) ja NDIS (Network Driver Interface Specification)



Kuva 12.10: TCP/IP-protokollapino.

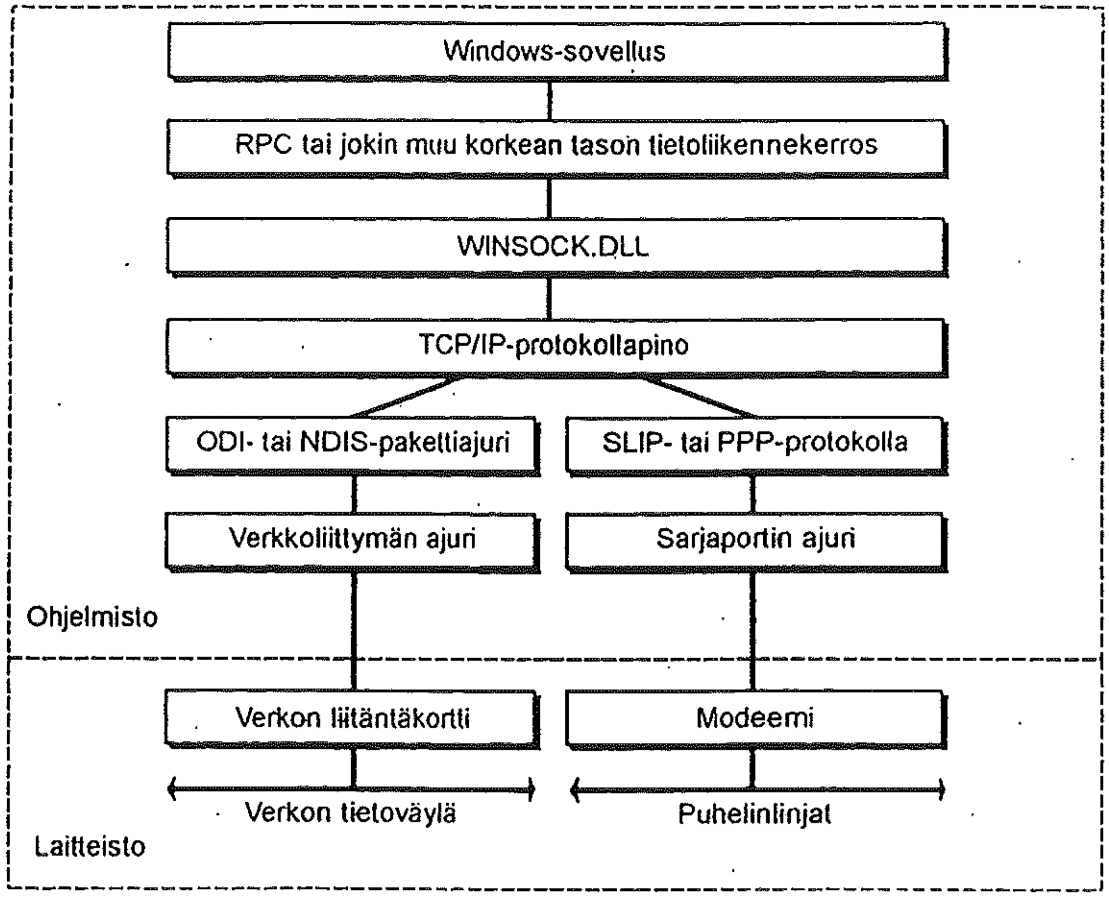
ovat pakettiajureita, joiden avulla TCP/IP ja muut kuljetusprotokollat voivat jakaa yhteisen verkkoliitäntäkortin.

Kotikäytössä ei yleensä ole lähiverkkoa eikä kiinteää yhteyttä, vaan PC:n täytyy kommunikoida Internetin palvelinkoneen kanssa modeemin ja asynkronisen tiedonvälityksen avulla. Tällöin ODI:n ja NDIS:n sijaan tarvitaan SLIP- tai PPP-protokolla.

SLIP kehystää IP-paketin ennen sen siirtämistä sarjaväylää pitkin niin, että vastaanottaja pystyy eristämään IP-paketin asynkronisesta tavuvirrasta. Protokolla ei tarjoa osoitteenmuodostusominaisuutta, paketin tyypin tunnistamista, virheen havaitsemista tai korjaamista eikä paketin pakkaamista. SLIP on yksinkertainen. Se ei ole virallinen standardi, vaikka se onkin laajassa käytössä. PPP (Point-to-Point) ratkaisee kaikki SLIPin puutteet ja se on Internetin virallinen standardi. PPP on myös valmiina Windows95:ssä.

Kuvioon on lisätty vielä sovellusten tekemistä helpottavia ohjelmistoja. TCP/IP-pinon päällä on Windows Socket -rajapinta. 1980-luvulla Berkeleyssä kehitettiin ns. *pistokkeisiin* (engl. socket) perustuva sovellusrajapinta Unix-ympäristöön. Windows Socket on Berkeleyn pistokkeiden pohjalta kehitetty sovellusrajapinta Windows-ympäristöön. Se sisältää useita aliohjelmia, joita voidaan käyttää hyväksi hajautettuja sovelluksia laadittaessa.

RCP eli Remote Procedure Call on lisärajapinta, jonka avulla sovellus voi kutsua etäisessä koneessä sijaitsevaa proseduuria. RCP:n käyttö tulee koko ajan lisääntymään, mutta se voi myös puuttua ohjelmistosta.



Kuva 12.11: Winsockin sijainti Windowsin ohjelmointiympäristössä.

Luku 13

Enemmän, nopeammin, tehokkaammin

Suorituskyvyn kannalta ratkaisevaa on kuinka paljon on suoritettavaa ja mikä on suoritusaika per käsky. Siihen kuinka paljon on suoritettavaa vaikuttaa konekielisten käskyjen monipuolisuus ja siihen mikä on suoritusaika / käsky vaikuttaa sekä organisaatio että laitteiston implementointi.

13.1 Parempi teknologia

Käskynsuoritusnopeuteen vaikuttavat laitteiston pakkaustiheys ja peruselektroniikan nopeus. Pakkaustiheys sekä eri komponenttien välinen etäisyys vaikuttaa siihen, kuinka nopeasti tietoa voidaan välittää eri toimintoyksikköjen välillä ja elektroniikan nopeus määrää sen, kuinka nopeasti kiikkujen sisäinen tila voidaan vaihtaa. Lähekkäin ja samalla piirillä olevat komponentit kommunikoivat nopeimmin, sillä signaalien siirtoviive on pienin.

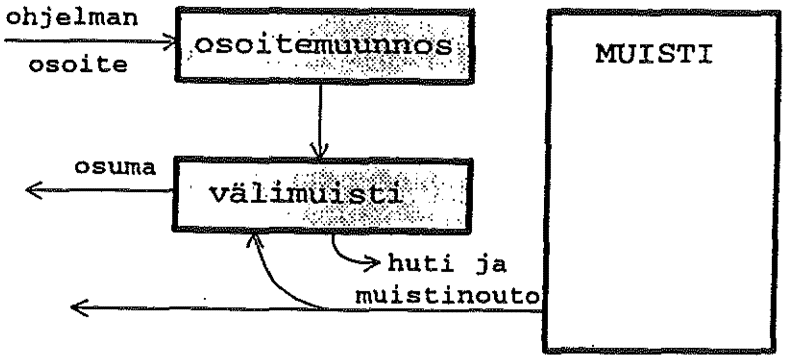
Suorin tapa lisätä prosessorin nopeutta on lyhentää kellojaksoa (engl. cycle). Jotta kellojakson lyhentäminen ylipäätänsä olisi mahdollista on käytettävä riittävän nopeata peruselektroniikkaa ja riittävän nopeata muistia. Muuten muistiviittauksen yhteyteen on lisättävä ylimääräisiä odotustiloja. Tavallista on, että prosessorin elektroniikka kulkee kehityksen kärjessä. Niinpä ratkaisut, jossa prosessorin sisäinen kellotaajuus on suurempi kuin sen ulkoinen kellotaajuus, ovat melko yleisiä.

Nopeus, jolla tietokone käsittelee tietojaan on suorassa suhteessa siihen tietomäärään, jonka se pystyy kerrallaan siirtämään. Kerralla käsiteltävän tiedon määrä riippuu olennaisesti koneen sananpituudesta ja tiedonsiirtoväylien leveydestä. Prosessorin sisällä dataväylä onkin usein leveämpi kuin sen ulkopuolella. Sananpituus ei ole välttämättä tiedonsiirron perusyksikkö prosessorin ja muistin välillä. Muistista voidaan siirtää kerralla useampikin sana prosessorin välimuistiin. Tietokone, jonka sananpituus on kaksi kertaa niin suuri kuin toisen koneen, voi siirtää lähes kaksi kertaa enemmän tietoa samassa ajassa.

13.2 Muistinoudon tehostaminen

Prosessori pystyy käsittelemään tietoja huomattavasti nopeammin kuin niitä voidaan toimittaa sille muistista tai muisti voi ottaa vastaan prosessorilta. Koska ohjelman kaikki käskyt on noudettava muistista ja niistä yli 60 prosenttia tarvitsee muistinoutoa tai -kirjoitusta, on muistiviittauksen tehokas toteuttaminen erityisen tärkeää. Hyvin suunnitellussa arkkitehtuurissa on muistiviitteiden lukumäärä ja siirtoihin käytetty aika minimoitu. Muistiviitteiden lukumäärää voidaan pienentää lisäämällä rekistereiden lukumäärää ja käyttöä, jolloin siirtojen lukumäärä vähenee. Viittaukseen kuluvaa aikaa voidaan vähentää jakamalla muisti esimerkiksi 2:een tai 4:ään *limittäiseen muistipankkiin* (engl. memory interleaving). Muistin toteutuksesta johtuen on jokaisen muistinoudon jälkeen odotettava tietty latenttiaika ennenkuin voidaan viitata taas samaan muistipankkiin. Kun peräkkäiset muistipaikat sijoitetaan eri muistipankkeihin ei tuota latenttiaikaa tarvitse odottaa.

Yleinen tapa tehostaa käskyjen muistinoutoa on tehdä siitä hierarkkinen laittamalla prosessoriin rekistereiden ja muistin väliin nopeammalla teknologialla toteutettu puskuri, välimuisti (kuva 13.1). Kun nouto keskusmuistista vie 70 - 100 ns, niin nouto välimuistista vie esimerkiksi vain noin 15 ns.



Kuva 13.1: Jos muistipaikan sisältö on välimuistissa, ei tarvita hitaampaa muistinoutoa. Muuten tehdään normaali muistinouto ja päivitetään noudetun muistipaikan sisältö myös välimuistiin.

Koska ohjelmat tavallisesti etenevät siististi käskyjen mukaisessa peräkkäisjärjestyksessä, voidaan muistinoutoja ennakoida tuomalla keskusmuistista käskynoudon ja datanoudon yhteydessä kerralla useita peräkkäisiä muistipaikkoja välimuistiin. Viitatessaan muistipaikkoihin prosessori kurkistaa ensin, löytyykö kaivatun muistipaikan sisältö välimuistista. Välimuisti on toteutettu nopeana assosiatiivimuistina, joka etsii tietoa sisällön perusteella. Etsintä sujuu vertaamalla laitteistolla etsittävää fyysistä muistiosoitetta samanaikaisesti kaikkien välimuistissa olevien rivien tunnisteosaan. Ellei muistipaikan sisältö ole välimuistissa, on tehtävä taas uusi muistinouto. Osumatodennäköisyys (engl. hit rate) on yli 90 % jo pienilläkin välimuisteilla (esim. Pentium 8 kB koodille ja 8 kB datalle).

Muistipaikkoihin tapahtuvat muutokset tehdään myös aina välimuistiin. Keskusmuistiin muutos voidaan kirjoittaa joko heti (läpikirjoittava välimuisti, engl. write-

through) tai vasta siinä vaiheessa, kun muistipaikan sisältö tulee ylikirjoitettavaksi välimuistissa (engl. write-back).

13.3 RISC

RISC-koneiden (engl. Reduced Instruction Set Computers) perusidea on tehdä koneen käskykannasta niin yksinkertainen, että prosessori voidaan toteuttaa yksinkertaisesti pienemmällä transistorimäärällä ja saada siten nopeaksi. Perinteisten CISC-koneiden (engl. Compilicated Instruction Set Computers) käskykantaan kuuluu satoja käskyjä, joista useista on vielä erikseen muodot tavu- ja sanaoperandien käsittelyyn. RISC-koneissa on vain muutamia kymmeniä erilaisia käskyjä (RISC-II:ssa 39) ja käytettävissä olevat datatyypit ja osoitusmuodot ovat yksinkertaisia ja niitä on vähän. Käytettävissä on tavallisesti vain 32-bittiset kokonaisluvut, ja muut datatyypit hoidetaan sopivasti emuloimalla. Monimutkaiset osoitteenlaskennat hoidetaan normaaleilla peräkkäisillä käskyillä sen sijaan, että kutakin varten olisi oma osoitusmuotonsa.

Kaikki käskyt ovat aina samanpituisia ja käskyn nouto ja tulkinta on yksinkertaista. Suorituksen ohjaus toteutetaan suoraan langoittamalla, joten yksi välivaihe eli mikrokoodi jää pois toteutuksesta. Tavoite on suorittaa yksi käsky yhden kellojakson aikana ja pitää tuo kellojakso silti erittäin lyhyenä (suuri kellotaajuus).

Lähes 90% suoritettavista käskyistä on yksinkertaisia operaatioita, jotka pystytään suorittamaan nopeasti ja tehokkaasti. Monimutkaisempia operaatioita varten ei ole omia käskyjä, vaan ne toteutetaan ohjelmallisesti tai laitteistotason algoritmeilla. Esimerkiksi kerto- ja jakolaskua varten ei ole omia käskyjä, vaan ne hoidetaan sivuttaissiirroilla ja yhteen- tai vähennyslaskulla. Koska käytössä on vain yksinkertaisia operaatioita, tulee RISC-ohjelmista tuplasti pitempiä kuin monipuolisemman käskykannan tarjoavista CISC-ohjelmista. Mutta, kun yksittäinen käsky suoritetaan nopeammin, kuluu RISC-koneelta vähemmän aikaa tuon kokonaisuuden suorittamiseen.

Koska käskysykli on tehty lyhyeksi, ei muisti pysty ruokkimaan prosessoria riittävän nopeasti. Prosessoriin kuuluu luonnollisesti välimuisti ja koodi sekä data sijoitetaan eri muistipankkeihin, joihin on molempia varten omat väylänsä. Tämän lisäksi prosessorissa on paljon rekistereitä ja niiden käyttöä on kasvatettu. RISC-II koneessa on peräti 138 rekisteriä, joista kerrallaan käytössä on 32. Aliohjelmien parametrinvälityksessä eli aktivointitietueena ei käytetä perinteiseen tapaan pinoa, vaan sekä sisäänmeno- että ulostuloparametrit välitetään rekistereissä (max 6 arvoa). Käsiteltävä tieto pyritään aina pitämään rekistereissä ja vain LOAD- ja STORE-käskyjen operandit voivat sijaita keskusmuistissa.

13.4 Rinnakkaisuus

Aeimmin esitetty käskyn nouto ja suoritus on huomattavasti yksinkertaistettu malli todellisten koneiden toiminnasta. Koneiden nopeutta voidaan parantaa lisäämällä laitteistoon uusia rinnakkain toimivia osia. Tavallisimmin näiden osien tehtävänä on noutaa tietoa käsiteltäväksi ennalta nopeampiin puskurimuisteihin tai suorittaa yhtäaikaisesti useita eri käskyjä tai käskyjen osia.

Tavallisin tapa käyttää rinnakkaisuutta käskyjen suorituksessa on tuoda suoritettavaksi uusia käskyjä jo edellisten käskyjen 'kuivausvaiheessa'. Tämä toteutetaan jakamalla käskyn suoritus *liukuhihnalla* (engl. pipeline) erillisiin limittäin eteneviin vaiheisiin, joissa kussakin voi olla yksi keskeneräinen käsky. Kaikissa nykyaikaisissa prosessoreissa käytetään tätä tapaa.

Tyypillisen viisivaiheisen liukuhihnan osat ovat käskynnouto (Fetch), käskyn tulkinta (Decode), operandien nouto (Load), käskyn suoritus (eXecute) ja tuloksen talletus (Store).

```
LOAD  R1,A   F  D  L  X  S
LOAD  R2,R3     F  D  L  X  S
ADD   R1,R2        F  D  L  X  S
jne.
```

Käsiteltävänä voi samanaikaisesti olla yhtämonta käskyä kuin liukuhihnassa on vaiheita. Periaatteessa liukuhihnalta valmistuu viisi käskyä samassa ajassa kuin ei-liukuhihnoitettu toteutus selviää yhdestä käskystä. Käytännössä tähän ei aina kuitenkaan päästä, sillä eräissä tapauksissa liukuhihna 'yskii' ja kaikki vaiheet eivät voi edetä tasaisesti. Tällaisia tilanteita syntyy, koska vain yksi vaihe kerrallaan voi viitata muistiin, eri vaiheiden suoritusajat vievät erilaisilla käskyillä eri ajan ja peräkkäiset käskyt voivat käyttää yhteisiä operandeja, jolloin jäljempänä tuleva käsky ei saa ladata operandin arvoa ennenkuin edellämenevä käsky on tallettanut tuloksensa operandille. Hyppykäskyjen yhteydessä saatetaan joutua liukuhihna tyhjentämään, sillä ennaltanouto on voinut tuoda liukuhihnalle vääriä käskyjä. Osa ongelmista voidaan ratkaista jo kääntäjän tasolla ja osa ongelmista ratkotaan laitteistolla.

Yhtäaikaisiin muistiviittauksiin tuo apua välimuistin käyttö. Tällöinhän varsinaista muistinoutoa ei tarvitsekaan tehdä, sillä käsky tai data on jo valmiiksi prosessorissa. Koska ohjelmat tyypillisesti käsittelevät suurella alueella ja pitkään samaa dataa, on järkevintä toteuttaa erikseen käskyvälimuisti ja datavälimuisti, jotta poistopäätöksiä tehtäessä ei korvattaisi pian tarvittavien datamuistipaikkojen sisältöä. Väylän varauksessa sattuvia yhteentörmäyksiä voidaan vähentää myös järjestämällä fyysinen muisti siten, että käskyille ja datalle on omat muistipankkinsa ja kummallekin alueelle on erikseen omat väylät.

Kääntäjä voi ryhmitellä käskyjä uudelleen tai tuottaa rekistereitä sopivasti käyttämällä sellaista koodia, jossa ei esiinny peräkkäisten käskyjen välisiä operandien riippuvuuksia. Tai kääntäjä voi lisätä käskyjen joukkoon ylimääräisiä NOP-käskyjä

(engl. no operation), jotka täyttävät liukuhihnaa, mutta eivät aiheuta toimintaa. Jos kääntäjä ei lisää NOP-käskyjä, on laitteiston pystyttävä tunnistamaan käskyjen väliset riippuvuudet ja viivytettävä perässä tulevia vaiheita.

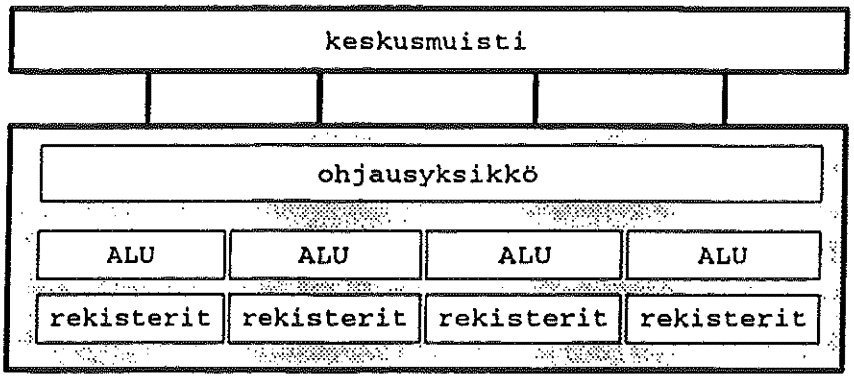
Myös hyppykäskyjen yhteydessä voidaan perässätulevat vaiheet täydentää NOP-käskyillä. Muuta kääntäjä ei pystykään tekemään, sillä se ei voi tietää hypätäänkö käskyä suoritettaessa vai ei. Konekielisiä ohjelmia tutkimalla on huomattu, että 15-30% suoritettavista käskyistä on hyppykäskyjä, ja noin 63% tapauksista hyppy todella suoritetaan (staattinen tilastotieto). Näinollen laitteisto kannattaa rakentaa siten, että hyppykäskyn perään liukuhihnalle haetaankin käskyjä hypyn kohteesta. Vielä parempaan tulokseen päästään, jos laitteistoon kuuluu *hyppyjen ennustuslogiikka* (engl. branch prediction). Se kokoaa tietoa suoritusaikana hyppykäskyistä (dynaaminen tilastotieto), ja päättää sen perusteella noudetaanko hihnalle hyppykäskyä seuraavia käskyjä vai hypyn kohteessa olevia käskyjä. Kun käsky suoritetaan ensimmäistä kertaa, kannattaa liukuhihnalle noutaa käskyjä hypyn kohteesta. Jos sama käsky tulee seuraavan kerran suoritukseen, katsotaan kuinka kävi edellisellä kerralla. Jos tehtiin hyppy, niin noudetaan käskyjä taas hypyn kohteesta. Jos ei hypätty, niin noudetaan hyppykäskyä seuraavia käskyjä. Jo pelkän yhden aiemman suorituskerran perusteella tämä ennustus osuu oikeaan 96% tapauksista. Kun otetaan vielä huomioon se, että useimpien silmukoiden kaikki käskyt sopivat kerralla välimuistiin, ei edes tarvita uusia muistinoutoja.

Erilaisten käskyjen erilaisen suoritusnopeuden aiheuttamien ongelmien yksinkertaisin ratkaisu olisi tehdä kustakin vaiheesta sen pituinen kuin se on hitaimmillaan. Tämä on kuitenkin tehotonta. Yleinen tapa on käyttää *eriytettyä laitteistoa* (engl. multiple functional units), joka koostuu erityisesti tiettyä tarkoitusta varten toteutetuista nopeista toiminto-yksiköistä. Tavallisimmin tätä menettelyä käytetään liukulukulaskennassa. Kun laitteisto havaitsee liukulukukäskyn, se antaa käskyn suorituksen liukulukuyksikölle. Samaan aikaan muu laitteisto voi noutaa ja suorittaa jo ohjelman seuraavia käskyjä (ellei ne tarvitse liukulukukäskyn operandeja).

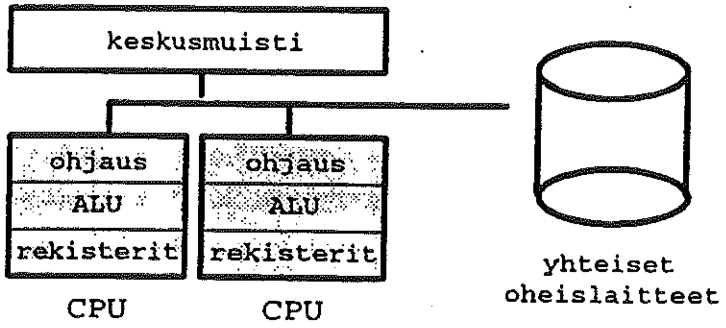
Vektoriprosessori (engl. array processor) on erikoistunut taulukkomuotoisen tiedon nopeaan käsittelyyn (kuva 13.2). Se pystyy suorittamaan yhtäaikaisesti saman operaation samanlaisten taulukoiden eri alkioille.

Moniprosessorijärjestelmissä (engl. multiprocessor) on useita erillisiä käskyjä suorittavia prosessoreita, joilla kullakin on suoritettavana omat käskynsä, mutta jotka jakavat yhteisen muistin (kuva 13.3). Yksinkertaisimmillaan kukin prosessori suorittaa omaa ohjelmaansa. Rinnakkaislaskennassa prosessorit suorittavat samaa tehtävää yhdessä.

Kun rinnakkaisten prosessoreiden lukumäärä kasvaa, tulee muistiinviittauksesta helposti pullonkaula. Avuksi tulee muistin limittäminen ja useampien väylien käyttö. Prosessorissa sijaitsevan välimuistin käyttö vähentää muistiviittausten lukumäärää, mutta ongelmaksi tulee välimuistin sisällön oikeellisuus. Saman muistipaikan sisältöhän voi olla yhtäaikaa usean prosessorin välimuistissa. Tämän vuoksi välimuisti on nuuskiva ja läpikirjoittava (engl. snoopy write-throuhg cache), eli muutos tehdään aina myös keskusmuistiin ja jokainen välimuisti tarkkailee väylällä liikkuvia osoit-



Kuva 13.2: Vektoriprosessori.



Kuva 13.3: Moniprosessorijärjestelmä.

teita ja mitätöi muuttuneiden välimuistissa olevien muistipaikkojen sisällöt. Toinen tapa vähentää muistiväylän käyttöä on liittää kunkin prosessorin yhteyteen vielä omaa paikallista muistia, jota käytetään usein tarvittavien käyttöjärjestelmän osien tallettamiseen (lähinnä keskeytyskäsittely).

Kun prosessoreita on kytketty toisiinsa tuhansittain (jopa 65 000) erityisellä kytkentäverkolla, puhutaan *massiivisesti rinnakkaisista järjestelmistä* (engl. massively parallel system). Tämän päivän nopeimmat tietokonejärjestelmät on poikkeuksetta toteutettu nopeista prosessoreista muodostetuilla kytkentäverkoilla.

Myös *hajautettua järjestelmää* (engl. distributed system) voidaan pitää moniprossorijärjestelmänä, vaikka se muodostuukin erillisten koneiden muodostamasta verkosta. Kullakin koneella on tällöin oma muisti ja ohjaus, mutta koneilla on yhteisiä ohielaitteita sekä yhteinen käyttöjärjestelmä ja ne kommunikoivat keskenään sanomanvälitystä käyttäen.

13.5 Miksi Pentium on nopeampi?

	"se alkuperäinen" 8088	Pentium
suoritusnopeus	0,33 MIPS	yli 100 MIPS
kellotaajuus	4,77 MHz	60 - 266 MHz
transistoreja	29 000	yli 3 100 000
väylät sisäiset /ulkoiset	16 b / 8 b	64 b / 64 b tai 32 b
liukuhihna	2-vaiheinen	5-vaiheinen

Pentiumin käskynsuoritusnopeus on yli 100 miljoona käskyä sekunnissa. Prosessorin toteutuksessa on käytetty uutta BiCMOS-tekniikkaa (engl. Bipolar Complementary Metal Oxide Semiconductor). Siten käyttöjännite on voitu alentaa 3.3 volttiin ja samalla lämmöntuotto on pienentynyt. Pentiumissa on pakattu yli 3.1 miljoonaa transistoria samalle piirille. Kellotaajuus oli ensimmäisissä versioissa 60 MHz ja nyt jo yli 200 MHz.

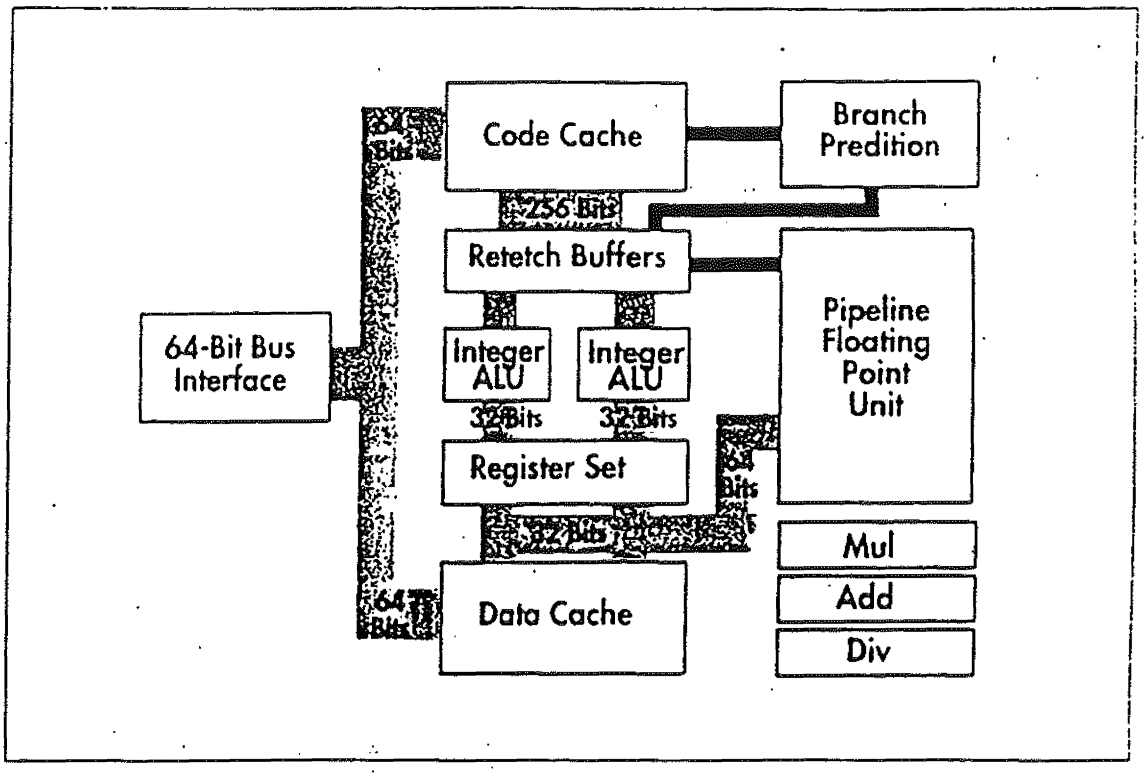
Vaikka Pentium on käskykannaltaan täysin yhteensopiva edellisen 486-version kanssa, on siihen otettu muita RISC-koneiden piirteitä. Pentium on toteutukseltaan eräänlainen CISC- ja RISC-prosessoreiden hybridi. Tehokkuussyistä mikrokoodin osuutta on vähennetty runsaasti ja tilalla on entistä enemmän langoitettua toteutusta. Toteutus perustuu kahden liukuhihnoitetun 32-bittisen kokonaislukuyksikön (engl. integer ALU) superskalaarirakenteeseen. Pentium voi käsitellä noin kaksi käskyä yhden käskysyklin aikana. Jotta näistä erillisistä linjoista saataisiin täysi hyöty on Pentium-koneille koodia tuottavien kääntäjien optimoitava koodia tarvittaessa käskyjä uudelleenjärjestämällä ja täyttämällä NOP-käskyillä.

Liukulukujen käsittelyä varten on prosessorin yhteydessä niinikään liukuhihnoitettu oma yksikkö (engl. floating point unit, FPU). Myös sen toteutusta on parannettu ja uusitun yksikön väitetään olevan 4-10 kertaa nopeampi kuin 486-prosessorissa.

Pentiumin liukuhihnassa on 5-vaihetta. Käskyjen liukuhihnalle noudossa käytetään laitteistossa toteutettua hyppykäskyjen ennustuslogiikkaa. Tämän on arveltu antavan hyppykäskyjen yhteydessä noin 25%:n nopeuden lisäyksen.

Välimuisti on jaettu erikseen käsky- ja datavälimuisteihin. Kummankin koko on 8 kB. Tämän lisäksi emolevyllä on tavallisesti 256 kB:n ulkoinen välimuisti. Välimuistit kirjoittavat tiedon keskusmuistiin vain, jos välimuistirivin sisältö tulee korvattavaksi uudella tiedolla.

Prosessorin sisällä dataväylä on 64 tavua leveä ja prosessorin ulkoinen dataväylä on 32 tai 64 bittinen. Eräiden prosessorin sisäisten osien välillä on jopa 256 bitin levyisiä väyliä. Vanhat emolevyt eivät käy Pentium-koneen toteutukseen. Emolevynä käytetään 64-bitin PCI-väylällä varustettua piirilevyä.



Kuva 13.4: Pentium-prosessorin rakennekaavio.

Helsinki 2001
Yliopistopaino