TIES325 Tietokonejärjestelmä

Jani Kurhinen
Jyväskylän yliopisto
Tietotekniikan laitos

Kevät 2008

Luku 1

Tietokone abstraktina yksikkönä

Tietokoneen asbtratiotasoa sen muotoisena kuin me sen tällä hetkellä tunnemme voidaan lähestyä kahdelta suunnalta: tietokoneen rakenne (computer organization) sekä tietokoneen arkkitehtuuri (computer architecture). Näiden termien määritelmiä eivät ole välttämättä täysin yksikäsitteisiä, mutta yleisesti hyväksytyn määritelmän mukaan tietokoneen arkkitehtuuri viittaa ominaisuuksiin, jotka koskettavat läheisesti sovelluskehittäjää. Tietokoneen rakenne puolestaan viittaa toiminnallisiin yksiköihin, näiden keskenäiseen suhteeseen sekä yksityiskohtiin, jotka toteuttavat arkkitehtuuriset määritykset. Esimerkkejä arkkitehtuuriin liittyvistä asioista ovat esimerkiksi käytössä olevan järjestelmän ymmärtämät käskyt, datan kuvaaminen (lukujen tarkkuus, esitystapa). Vastaavasti tietokoneen sisäinen toiminta, kuten laskentayksiköiden välinen signalointi tai kertolaskun suorittaminen, liittyvät tietokoneen rakenteeseen.

Arkkitehtuurin ja rakenteen välistä suhdetta voidaan kuvata yhden valmistajan laiteperheen avulla. Perheeseen kuuluvat laitteet noudattavat kaikki samaa arkkitehtuuria, joten ne ovat ohjelmallisesti yhteensopivia keskenään. Toisin sanoen perheen yhdelle jäsenelle tehty sovellus toimii (tietyin rajoituksin) perheen muissakin laitteissa. Laitteet ovat kuitenkin keskenään erilaisia rakenteensa suhteen, minkä seurauksena valmistaja voi tarjota tuotteitaan erilaisille asiakassegmenteille, joilla on erilainen maksukyky ja/tai erilaiset tarpeet. Esimerkiksi perheen halvassa mallissa ollaan voitu säästää kustannuksia jättämällä erillinen liukulukusuoritin pois, mikä hidastaa koneen toimintaa tietyn tyyppisessä käytössä. Jos käyttäjällä ei ole erityistä tarvetta liukuluvuille tai käytössä oleva budjetti ei anna myöden kalliimman mallin ostamiselle, tämä voi olla ainoa mahdollinen kompromissi. Toisaalta jatkossa on kuitenkin mahdollista vaihtaa uuteen (jopa eri valmistajan samaa samaa arkkitehtuuria noudattavaan) laitteeseen ilman, että menettää mahdollisuuden vanhojen sovellusten käyttämiseen. Itse asiassa tällainen laiteperheiden idea ei ollut mukana tietokoneiden varhaisessa kehityksessä, vaan se ole yksi tietojenkäsittelyä mullistanut idea 1960-luvulla.

1.1 Tietokoneen arkkitehtuuri

Suorittimen käskykanta muodostaa perustan tietokoneen arkkitehtuurille. Se määrittelee millaisista toiminnoista suoritin kykenee suoriutumaan. Suorittimien käskykannat jaetaan tällä hetkellä neljään luokkaan: RISC (Reduced instruction set computer), CISC (Complex instruction set computer) ja uudempina tulokkaina VLIW (Very long instruction word) sekä TTA (Transport Triggered Architecture). Toisaalta jako voidaan tehdä myös toimintatyypin perusteella: pinoarkkitehtuuri, akkuperustainen arkkitehtuuri sekä yleiskäyttöisten rekistereiden arkkitehtuuri. Lisäksi viimeksi mainittu voidaan jakaa vielä kahtia rekisteri-muisti-arkkitehtuureihin, jonka ominaispiirteenä on muistin osoittaminen useilla erilaisilla käskyillä, sekä lataus-tallennus-arkkitehtuuri, jossa puolestaan kaikki varsinainen datan käsittely tapahtuu prosessorin sisällä rekistereissä ja muistin kanssa operoidaan ainoastaan erillisillä muistista lataus- ja sinne kirjoitus -käskyillä.

Varsinaisen käskykannan lisäksi (tai oikeastaan sen osana) tietokoneen arkkitehtuuriin vaikuttavat myös muut yksityiskohdat. Näitä ovat muistin osoittamiseen ja osoittamistekniikka, käytössä olevat tietotyyppit ja niiden operaatiot, käskyjen suorituksen ohjelmallinen ohjaus sekä itse käskykannan kuvaus prosessorin konekielellä.

Muistin käyttöön liittyvät arkkitehtuurikohtaiset ratkaisut vaikuttavat siihen, miten tieto voidaan tallentaa tietokoneen käyttömuistiin: voiko se sijaita mielivaltaisessa paikassa vai onko tarvitseeko tiedon tallentamisessa noudattaa joitain tiettyjä sääntöjä. Muistinosoittamistekniikka määrää sen, miten tiettyyn tietoelementtiin voidaan osoittaa. Voidaanko siihen osoittaa suoraan muistiosoitteella, onko osoite ladattava etukäteen prosessorin rekisteriin tai voiko muistiin osoittaa epäsuorasti johonkin tallennuspaikkaan sijoitetun osoitearvon perusteella?

Tietotyypit ja niiden operaatiot kuvaavat sitä, millaisilla tietoyksiköillä prosessori kykenee toimimaan. Prosessoriarkkitehtuuri määrittää esimerkiksi kokonaisluvun 32-bittiseksi binääriluvuksi ja suuren tarkkuuden liukuluvun 64-bittiseksi IEEE 754 -standardin mukaiseksi. Operaatio pitävät sisällään aritmeettisten operaatioiden (kerto-, jako-, yhteen, vähennyslasku sekä bittioperaatiot) lisäksi tiedonsiirto (siirto/tallennus muistin ja rekistereiden välillä) sekä ohjauskäskyt (hypyt, vertailut).

Käskyjen suorituksen ohjelmallisen ohjauksen peruselementti on edellä mainitut ohjauskäskyt. Vaikka tämä on periaatteessa vain yksi yksi käskytyyppi, ansaitsee se erityismaininnan tärkeytensä vuoksi. Käytännössä kaikki "normaalit"[1] tietokoneet toimivat siten, että ne suorittavat käskyjä järjestyksessä omassa muistiavaruudessaan. Tällainen pelkästään lineaarisesti etenevä ohjelma ei ole kovin monipuolinen, sillä sen tekijän on pitänyt tuntea täysin tarkasti ohjelman suorituksen vaiheet. Näin olleen esimerkiksi mihinkään ulkoiseen syötteeseen ei voitaisi reagoida, koska tällainen ei voi olla ennalta tiedossa. Erilaisiin vertailuoperaatioihin perustuvat ehdolliset hypyt ovat tämän ominaisuuden perusta, mutta seuraavaan käskyyn osoittavan ohjelmalaskurin arvon ohjelmallinen muuttaminen on myös mahdollista joissain arkkitehtuureissa. Erityisesti kuitenkin aliohjelmakutsujen ja niistä palaamisen määrittäminen on kuvattuna

[1] Voidaan myös määritellä tietokonearkkitehtuuri, jossa seuraavaksi suoritettavan käskyn osoite välitetään aina osana edellistä käskyä, mutta ohjelmallisen ohjauksen tarve pätee erityisesti myös tämänlaiseen arkkitehtuurityyppiin.

prosessoriarkkitehtuurissa.

Viimeisenä listalla oli käskykannan kuvaus prosessorin konekielellä. Tähän on kaksi pääluokkaa: kiinteän sanapituuden arkkitehtuurit sekä vaihtuvan sanapituuden arkkitehtuurit. Kummassakin tavassa on omat hyvät ja huonot puolensa: esimerkiksi kiinteän sanapituuden arkkitehtuureissa käskyjen tulkintaan on yksinkertaisempaa, koska suorittimen ei tarvitse erikseen päätellä käskyn pituutta. Toisaalta tämä piirre tekee arkkitehtuurista myös jäykemmän (ja enemmän tilaa kuluttava), koska kaikkia käskysanoja varten pitää aina talleentaa tietty määrä bittejä, vaikka käskyn sisältämä informaatio voitaisiin koodata pajon tiiviimminkin.

Edellä oli kuvattuna hyvin tiiviisti tietokonearkkitehtuurin merkitys. Asiaan perehdytään vielä jatkossa lisää ja erityisesti käskykantoja käsitellään tarkemmin myöhemmin omassa luvussaan.

1.2 Tietokoneen rakenne

Kuten edellä mainittiin tietokoneen rakenne on käytännössä tapa toteuttaa jokin tietty arkkitehtuurimääritys. Kuvassa 1.1 on Tanenbaumin[2] esitys tietokoneen rakenteesta (hieman mukailtuna). Alimpana tasona tässä kuvauksessa on digitaalilogiikkakerros, joka luo perustan nykyaikaiselle tietokoneelle. Tämä kerros koostuu transistoreista rakennetuista logiikkaporteista, jotka suorittavat yksinkertaisia binäärisiä perustoimenpiteitä, kuten NOT, NAND, NOR, AND, OR. Näiden yksinkertaisen operaatioiden tarkasti suunnitelluilla yhdistelmillä voidaan toteuttaa kaikki tietokoneen tarvitsemat toiminpiteet tietyn bitin arvon säilyttämisestä kontrolloituun komponenttien väliseen signalointiin.

Kuva 1.1: Rakenteinen tietokoneen rakenne

Seuraava kerros ylöspäin mentäessä on mikroarkkitehtuurikerros. Sen tehtävänä on toteuttaa prosessorin käskykanta (eli tämän yläpuolella oleva kerros).

[2] Andrew S. Tanenbaum. Structured Computer Organization, 4th ed. Prentice-Hall. 1999.

Tämä on lopullisen tuotteen kannalta hyvin oleellinen kerros, sillä sen avulla voidaan tehdä hyvin erilaisia toteutuksia saman arkkitehtuuriinja samaan käskykantaan. Esimerkiksi mikroarkkitehtuurikerroksella voidaan valita toteutustapoja, jotka ovat hyvyn edullisia valmistaa, mutta joiden vuoksi tietokoneen kokonaistehokkuus ei pääse mahdollisimman korkealle tasolle.

Kolmas kerros on käskykanta. Kuten jo edellä luvussa 1.1 todettiin, se määrittelee tietokoneen hallitsemat operaatiot. Samalla on hyvä huomata, että se on samalla laitteiston ja ohjelmiston (johon siis myös käyttöjärjestelmä lasketaan) rajapinta. Teoriassa voisi olla mahdollista rakentaa laitteisto, joka kykenee suorittamaan korkean tason ohjelmointikieltä, kuten C, mutta tämä ei ole käytännön syistä järkevää. Sen sijaan on järkevämpää suunnitella laitteisto tiettyä, varsin suppeaa käskyjoukkoa varten, ja kehittää kääntäjiä, jotka kykenevät kääntämään näiden korkean tason kielten toiminnallisuuden prosessorin konekielelle. Tässä vaiheessa on syytä korostaa, että Assembler-kieli ei ole sama kuin käskykanta, vaan se on osa ylimmän abstraktiokerroksen ohjelmointikielien joukkoa. Lisäksi voidaan myös mainita, että näillä kolmelle kerroksella voidaan jo kuvata täysin toimiva tietokonejärjestelmä. Kaksi jäljellä olevaa korkeampaa kerrosta lisäävät ainoastaan käytettävyyttä erilaisten käyttömahdollisuuksien ja helpomman ohjelmoitavuuden kautta.

Seuraavana kerroksena ylöspäin mentäessä on käyttöjärjestelmä. Käyttöjärjestelmä tarjoaa mahdollisuuden hallita laitteistoa helpommalla tavalla verrattuna laitteiston suoraan ohjaukseen. Tämä tarkoittaa erityisten systeemikutsujen joukkoa, joilla sovellusohjelmoitsija voi laajentaa omaa sovellustaan prosessorin tarjoaman käskykannan lisäksi. Systeemikutsujen voidaan ajatella olevan rutiineja, jotka koostuvat (yleensä suuresta) joukosta käskykannan kutsuja. Näiden systeemikutsujen avulla käyttöjärjestelmä tarjoaa rajapinnan erilaisille I/O- ja muistilaitteille (sisältäen virtuaalimuistin) sekä mahdollistaa useiden suoritettavien ohjelmakoodien yhteistoiminnan[3].

Viimeinen kerros rakennemallissamme on sovellusohjelmointikerros. Tanenbaumin versiossa kerroksen nimi on Assembly-kieli-kerros, mutta ei ole mitään erityistä syytä erottaa tässä yhteydessä Assemblyä esimerkiksi C-kielestä. Toki jälkimmäinen on näistä kahdesta ihmisystävällisempää, mutta molemmat noudattavat tällä abstraktiotasolla täsmälleen samoja sääntöjä: Tietyllä kielellä kirjoitettu sovellusohjelma on ensin *käännettävä* käyttöjärjestelmän ymmärtämään muotoon[4]. Sovellusohjelmointikerroksen tehtävänä on ainoastaan helpottaa ohjelmoijan työtä. Vaikka assembler-ohjelmoitia pidetään monesti hankalana, on huomattavasti miellyttävämpää sanoa *decf 0x07,1* kuin *0000 1110 0111*[5].

[3] Tässä jätettiin tarkoituksella termi "samanaikainen suoritus"käyttämättä, sillä yhden prosessorin järjestelmä suorittaa vain yhtä ohjelmaa kerrallaan.

[4] Käyttöjärjestelmähän ei kuitenkaan ollut pakollinen kerros. Ilman käyttöjärjestelmää kääntäjän tulos on suoraan prosessorin ymmärtämää kieltä.

[5] Tokihan vielä mukavampaa olisi, jos voitaisiin sanoa *a--;* tai *vähennä laskurin arvoa yhdellä*, mutta perusidea näissä kaikissa on täsmälleen sama