# TIES325 Tietokonejärjestelmä

Jani Kurhinen
Jyväskylän yliopisto
Tietotekniikan laitos

Kevät 2008

# Luku 4

# Tietokoneen sisäinen toiminta

Edellisisää osioiss aon tarkasteltu tietokoneen kehittymistä ja sen yleistä toimintaa. Seuraavaksi sukellamme tietokoneen sisään tutkimaan, miten bittikuvioista syntyy varsinaista hyötydataa tai ohjausinformaatiota.

## 4.1 Lukujen esittäminen

Tämän materiaalin puitteissa ei lukujärjestelmiin tai erilaisiin tapoihin esittää lukuja uhrata juurikaan tilaa, ainoastaan sen verran, että lukijoille varmasti muistuu mieleen paruskursseilla opetetut asiat. Tietokoneen sisäisesti kaikki informaatio esitetään binäärisenä, eli kantaluvun 2 mukaisena. Binäärinen informaatio on monesti kuitenkin liian tilaa vievää, eikä sitä tueta kaikissa ohjelmoitikielissä. Tämän vuoksi heksadesimaalinen muoto, eli kantalukuun 16 perustuvat esitystapa, on yleensä hyödyllinen ja erityisesti binääri-heksa-muunnos on syytä osata. Miksi heksadesimaaliesitys sitten on niin sovelias? Siihen on kaksi syytä: ensimmäinen syy on se, että 16 on kahden potenssi, joten muunnokset näiden järjestelmien välillä ovat suoraviivaisia. Toisekseen se, että yksi haksaluku vastaa neljää bittiä, mahdollistaa tämän lukumuodon käytön nykyisin yleisesti käytössä olevien 8, 12[1], 16, 32, 64 ja 128 bittisten tallennusyksiköiden esittämiseen.

Edellä mainitut binääri- ja heksaluvut sovivat siis tietotekniikka-ammattilaisen työkaluiksi ja niiden käyttö positiivisten kokonaislukujen esittämiseen onkin suoraviivaista. Ensimmäinen hiemen asioita monimutkaistava tekijä on tarve saada kuvattua negatiivisiä lukuja. Tähän tarkoitukseen on historiassa esitetty erilaisia menetelmiä, mutta niin sanottu 2-komplementtiesitys on saavuttanut standardin aseman ominaisuuksiensa vuoksi. Näitä tärkeitä perusominaisuuksia on kolme: 1) Luvun eniten merkitsevä bitti esittää luvun etumerkkiä. 2) Menetelmässä ei ole kahta eri esitystapaa nollalle. 3) Merkkiä vaihtavat laskutoimitukset voidaan suorittaa samoin kuin merkin säilyttävät. Näistä erityisesti viimeinen ominaisuus on erityisen tärkeä.

2-komplementtiesityksessä siis luvun eniten merkitsevä bitti kuvaa luvun etumerkkiä. Tämä osoitetaan siten, että ei-negatiivista lukua kuvataan arvolla 0 ja negatiivista vastaavasti arvolla 1. Tästä on suorana seurauksena se, että loput luvun arvoa kuvaavat bitit kuvaavat luvun suuruutta. Haluttaessa vaihtaa

[1] Tietyissä järjestelmissä, kuten PIC mikro-ohjain, käytetään 12 bittisiä käskysanoja.

jonkin tietyn luvun etumerkki on menetelmässä aina sana riippumatta siitä suoritetaanko vaihto ei-negatiivisesta negatiiviseen vai päinvastoin. Tämä merkinvaihtoalgoritmi on kaksivaiheinen: 1) Muodosta luvusta binääritasolla komplementti, eli käännä kaikki bitit vastakkaiseen asentoon[2]. 2) Lisää saatuun välitulokseen yksi.

Testataan esimerkillä: Otetaan kymmenjärjestelmän luvun 3 nelibittinen binäärimuoto 0011 ja muunnetaan se ensin negatiiviseksi ja sitten takaisin.

$$0011 \overset{komp.}{\Rightarrow} 1100 \tag{4.1}$$

$$1100 \overset{+1}{\Rightarrow} 1101 = -3_{10} \tag{4.2}$$

$$1101 \overset{komp.}{\Rightarrow} 0010 \tag{4.3}$$

$$0010 \overset{+1}{\Rightarrow} 0011 = 3_{10} \tag{4.4}$$

Kahden etumerkin käännön seurauksena päädyttiin siis alkutilanteeseen kuten pitikin. Verrattaessa tulosta 4.2 alkuperäiseen lukuun, voidaan huomata, että negatiivisen luvun suuruus esitetään eri tavalla kuin positiivisen luvun. Tällä tavalla voidaan siis esittää kokonaislukualue $[-2^{n-1}, 2^{n-1}-1]$, missä n on binääriluvun bittien lukumäärä. On siis huomattava, että 2-komplementtiesityksellä voidaan esittää enemmän negatiivisia kuin positiivisia lukuja. Tämä johtuu siitä, että yhden bitin vaikuttaessa luvun etumerkkiin, negatiivisia sekä ei-negatiivisia lukuja on yhtä paljon ja nolla kuuluu ei-negatiivisten lukujen joukkoon. Edellisistä seuraa se, että luku, jossa on eniten merkitsevä bitti arvossa 1 ja muut bitit arvossa 0, on käytettävissä olevalla bittimäärällä kuvattuna mahdollisimman pieni luku. Nyt on siis kaksi ensimmäistä edellä mainituista 2-komplementtiesityksen eduista käyty läpi. Kolmas ja viimeinen on laskutoimitusten suoraviivaisuus, joka voidaan osoittaa esimerkiksi laskemalla vähennyslasku $2 - 5$. Käytetään samaa nelibittistä esitystapaa kuin edellä[3]:

$$0010 - 0101 = 1101 = -3_{10} \tag{4.5}$$

Edellä esitetty tulos 4.5 antaa siis vastaavan tuloksen kuin aiempi tulos 4.2, mikä on yhtäpitävä myös kymmenjärjestelmässä tehtävälle vähennyslaskulle. Toisaalta sama vähennyslasku voidaan myös muuntaa yhteenlaskuksi $2 + (-5)$ ja tulos on jälleen sama:

$$0010 + 1011 = 1101 = -3_{10} \tag{4.6}$$

Edelliset esimerkit eivät missään nimessä ole kattava joukko osoittamaan, että 2-komplementtiesitys toimii aina, mutta kuten jo edellä mainittiin, kuuluu se toisen kurssin aihepiiriin. Esimerkkien tarkoitus oli ainoastaan virkistää muistia.

[2] Tämä onnistuu esimerkiksi tekemällä XOR-operaatio sellaisen maskin kanssa, jossa kaikilla biteillä on arvo 1.

[3] Merkkiä vaihtavassa vähennyslaskussa joudutaan tilanteeseen, jossa ei ole enää enemmän merkitsevää bittiä lainattavaksi. Tällöin voidaan lainata ns. ylivuotobitti Carry, joka ei suoranaisesti vaikuta laskutulokseen, mutta saattaa näkyä jotenkin muuten riippuen prosessoriarkkitehtuurista.

Negatiivisten lukujen jälkeen seuraava laajennus lukukenttään on rationaalilukujen[4] lisääminen. Aivan kuten kymmenjärjestelmällä, kynällä ja paperilla voimme esittää lukuja, joissa alle ykkösen olevia osia, voimme tehdä samoin myös binääriluvuilla. Esimerkiksi luku $2,2_{10}$ voidaan ilmaista binäärimuodossa $10.10_b$. Desimaalipilkun sijaan käytännöksi on muodostunut tapa käyttää erilaista välimerkkiä, binääripistettä. Edellinen esimerkki oli erityisen yksinkertainen, koska sekä kokonaisosa että murto-osa olivat suoraan binäärisen kantaluvun potensseja. Vastaavasti voidaan kuitenkin ilmaista hieman hankalampi luku, kuten $12,3_{10} = 1100.11_b$.

Tapaa, jossa desimaalipilkku vain korvataan binääripisteellä ja muunnetaan kokonais- ja murto-osa erikseen lukujärjestelmästä toiseen, kutsutaan kiinteän pisteen esitystavaksi. Tämän kiinteän pisteen esitystavan suurimpana ongelmana onse, että se vaatii suuren tilan molemmille osilleen, jotta sillä voitaisiin päästä vastaavaan suuruusluokkaan kuin kokonaislukuesityksessä sekä kuvaamaan rationaalilukuja riittävällä tarkkuudella.

Ongelman kiertämiseksi on kehitetty kuvaus, josta käytetään nimitystä liukuluku. Nimen semantiikka voidaan tuoda esiin seuraavalla esimerkillä kymmenjärjestelmän esimerkillä:

$$100,5 = 100,5 \cdot 10^0 = 10,05 \cdot 10^1 = 1,005 \cdot 10^2 \tag{4.7}$$

Kyseessä on (erityisesti fyysikoiden suosima) tapa esittää luku sen suuruusluokan ja arvon yhdistelmänä. Eli ensinnäkin sen avulla voidaan korostaa luvun suuruusluokkaa, mutta sillä voidaan myös tietyissä tilanteissa tiivistää lukuarvon kuvausta, esimerkiksi $3000000 = 3 \cdot 10^6$. Vastaava pätee myös hyvin pieniin lukuihin ja kantaluvun negatiivisiin exponentteihin. Esimerkin 4.7 mukaisesti samoin voidaan tehdä myös binäärijärjestelmän luvuilla, eli menetelmä ei ole sidottu lukujärjestelmän kantaan. Yleisessä muodossa luvun esittämiseen tarvitaan etumerkki, merkitsevät numerot (M) ja suuruusluokka (S)[5] tietyn kantaluvun (K) mukaisesti esitettynä.

$$\pm M \cdot K^{(\pm)S} \tag{4.8}$$

Siinä missä kokonaisluvuilla operoitaessa meillä oli käytössämme kaikki arvot mahdollisimman suuren ja mahdollisimman pienen luvun välillä, ei tilanne olekaan sama enää liukulukujen tapauksessa. Kiinteän pisteen menetelmällä voitiin esittää käytössä olevasta tallennustilasta riippuen rationaalilukuja tasaisesti, koska murto-osa ei riippunut mitenkään kokonaisosasta. Liukuluvussa sen sijaan merkitsevät numerot käyttäytyvätkin eritavalla operoitaessa (itsesarvoltaan) pienillä luvuilla verrattuna (itseisarvoltaan) suuria lukuja. Liukuluvut muodostavatkin lukujonon, joka on tiheämpi nollan lähellä ja harvempi positiivisessa ja negatiivisessa reunassa. Lisäksi tulee myös huomata, että menetelmällä ei päästä mielivaltaisen lähelle nollaa, vaan sen ympärillä jää alue[6], jota ei voida kuvata.

[4] Tietokonella on aina käytässään ennalta määrätty määrä tilaa lukuarvojen tallennukseen. Tällöin esitettävät lukuarvot voivat olla tiettyjä diskreettejä pisteitä lukujanalla, joten reaalilukuavaruuteen emme tule pääsemäänkään.

[5] Periaatteessa suuruusluokka pitää jo terminä sisällään sen, että onko kyseessä positiivinen vai negatiivinen eksponentti.

[6] Yleisessä tapauksessa tämä alue siis sisältää myös nollan, ei pelkästään ympäristöä!

Historiassa on ollut toisiensa kanssa epäyhteensopivia liukulukujen esitystapoja, mutta nykyisin noudatataan *IEEE754* standardin mukaista tapaa. Standardissa on kaksi eri tarkkuutta, 32- ja 64-bittinen, joista jälkimmäisellä voidaan laajentaa lukualuetta sekä itsesarvoltaan suuressa päässä kuten myös nollan lähellä. Koska nolla on hyvin tärkeä lukuarvo monenlaisissa käytännön sovelluksissa, on stadardissa määritetty erityinen esitystapa myös sitä varten. Nollan lisäksi *IEEE754* sisältää myös muita erityistapauksia, jotka on lueteltu taulukossa 4.1.

Taulukko 4.1: IEEE754 liukulukustandardin erikoisarvot.

| | Etumerkki | Eksponentti | Murto-osa |
|---|---|---|---|
| 0 | 0/1 | Nollia | Nollia |
| $\infty$ | 0 | Ykkösiä | Nollia |
| $-\infty$ | 1 | Ykkösiä | Nollia |
| NaN | 0/1 | Ykkösiä | $\neq 0$ |

Muista kiireistä johtuen tämä osio on toistaiseksi kesken. Sopivan hetken koittaessa tähän lisätään ieee754-määritystä tarkentava sekä big/little-endian problematiikkaa käsittelevä osio.

## 4.2 Datan varastointi – endianit

Monitavuinen tietoalkio voidaan tallentaa tietokoneeseen kahdella eri tapaa: vähemmän merkitsevä tavu tai enemmän merkitsevä tavu ensin. Tietokoneen sisäisesti tällä ei ole suurta merkitystä[7], mutta tiedonsiirrossa eri endian-järjestelmää käyttävien koneiden välillä voi syntyä ongelmia.

## 4.3 Käskyjen esittäminen

Luvun alussa käsiteltiin tietokoneen sisäistä datan esittämistä. Samaan tapaan kuin data, myös varsinainen ohjelma, eli suorituskäskyt pitää pystyä kuvaamaan tietokoneen ymmärtämällä tavalla. Jos mietimme hieman prosessia, jota tietokone toistaa käskyjä suorittaessaan, voidaan siitä erittaa tietyt vaiheet tietokonearkkitehtuurista tai toteutustavasta riippumatta. Ensinnäkin suorittimen pitää selvittää seuraavan käskyn sijainti ja noutaa se. Tämän jälkeen noudettu käsky on tulkittava. Varsinaisen käskyn tulkitsemisen jälkeen on selvitettävä, mihin käsky liittyy, eli selvitettävä käskyn parametrien, eli operandien, sijainti ja noudettava ne. Vasta tämän jälkeen voidaan suorittaa itse käsky. Käskyn suorituksen seurauksena syntyy tulos, jonka sijoituspaikka on ensin selvitettävä ja sen jälkeen suoritettava tallennus. Tämän jälkeen prosessi alkaa alusta seuraavan käskyn osoitteen selvittämisellä.

Edellä kuvatun kaltainen prosessi tulee siis pystyä suorittamaan yhden käs-

[7] Valinta vaikuttaa tietynlaisten operaatioiden, kuten vertailujen tehokkuuteen. Se siis vaikuttaa prosessorin suorituskykyyn ja täten (mahdollisesti) näkyy epäsuorasti loppukäyttäjälle.

kyjakson[8] aikana. Tällöin yhden käskyn aikana suorittimen pitää siis tietää 1) Mikä käsky on kyseessä?, 2) Mitkä ovat operandit?, 3) Mihin tulos tallennetaan? sekä 4) Mistä suoritusta jatketaan? Yleisessä tapauksessa yhdellä käskyllä voisi olla mielivaltainen määrä parametreja[9], mutta käytännössä tällainen järjestelmä ei ole mitenkään järkevästi toteutettavissa. Mielivaltainen operandimäärä voidaan yleensä pilkkoa osiin, jossa osavaiheilla on maksimissaan kaksi operandia. Tarkastellaan tilannetta yksinkertaisehkon käskyn $C = A + B$ avulla. Käskyn lisäksi käskysanan tulee siis kyetä sisältämään maksimissaan neljä parametria. Tämän muotoinen käsky voidaan siis esittää seuraavasti:[10]

```
ADD C, A, B, i
```

Tämän mukaiset neljän osoiteparametrin järjestelmät ovat varsin harvinaisia, eikä niitä juurikaan ole käytössä kuin joissakin erikoissovelluksissa. Edellisen esimerkin neljäs parametri, seuraavan käskyn osoite, onkin yleensä toteutettu siten, ettei sitä tarvitse eksplisiittisesti ilmoittaa, vaan suoritin osaa itsenäisesti paikantaa sen[11]. Tätä varten prosessoreissa on sisäinen erityisrekisteri – ohjelmalaskuri (program counter), joka pitää kirjaa siitä, mistä ohjelman suoritus pitää jatkua. Implisiittisellä seuraavan käskyn osoitteella esimerkkimme muuntuu muotoon:

```
ADD C, A, B
```

Tällainen kolmen parametrin järjestelmän on suorittu erityisesti RISC-arkkitehtuuria noudattavissa suorittimissa. Vaikka tällainen käskyjen esitystapa onkin hyvin ohjelmoijaystävällinen, aiheuttaa se hankaluuksia suorittimen toiteutukseen, sillä yhteen käskyyn tulee varata tilaa operaation lisäksi kolmelle parametrille, minkä seurauksena käskysanat pidentyvät. Ylimääräisen tilan vähentämiseksi voidaan tuloksen tallennuspaikka olettaa implisiittisesti samaksi kuin toinen parametreistä. Esimerkkimme muuntuu siis seuraavanlaiseen muotoon:[12]

```
MOV C, &ZERO[13]
ADD C, A
ADD C, B
```

Edellisen esimerkin mukaisen suorittimen tulee edelleen kyetä operaatioihin useiden rekistereiden (tai periaatteessa muistipaikkojen) välillä samaan tapaan kuin kolmenkin parametrin järjestelmissä. Saavutettu etu rajoittuu siten vain suorittimen käskysanan lyhenemiseen.

Seuraava askel kohti lyhyempää käskyä on määritellä implisiittinen tuloksen tallennuspaikka vieläkin tiukemmin kuin edellä. Tällöin tulos tallennetaankin ennalta määrättyyn paikkaa sen sijaan, että se voitaisiin valita käskyn parametrina. Tähan tarkoitukseen (tätä suunnittelumallia käyttävässä) suorittimessa on

[8] Käskyjakso ei nykykoneissa yleisesti ole sama kuin kellojakso, vaan yhden käskyn suorittaminen vie yleensä useita kellojaksoja.

[9] Esimerkiksi $\sum_{i=0}^{n}$, missä n on jokin suurehko luku.

[10] Käskysanaa ADD käytetään tässä vastaamaan symbolisen konekielen yhteenlaskuoperaatiota, A ja B ovat operandien sijainnit, C on tuloksen tallennuspaikka ja i on seuraavan käskyn osoite.

[11] Yleisesti on havaittu käytännölliseksi tavaksi käyttää menetelmää, jossa käskyt esitetään lineaarisesti suoritusjärjestyksessä (poislukien hyppykäskyt, joissa operaation tulos tallenetaankin ohjelmalaskuriin tavallisen rekisterin tai muistipaikan sijaan).

[12] MOV-käskysanaa käytetään siirtämään dataa rekisteristä toiseen ja &ZERO kuvaa erikoisrekisteriä, jossa on aina arvo nolla.

[13] Tätä käskyä tarvitaan tyhjentämään muistissa mahdollisesti oleva tieto, koska tässä tapauksessa käskysana ADD ainoastaan lisää, ei ylikirjoita olemassaolevaa, kuten edellisessä.

erillinen erikoisrekisteri – akku (accumulator) [14]. Intelin x86-prosessoriperhettä pidetään perinteisesti akkujärjestelmänä, vaikka se ei sitä puhtaasti olekaan. Esimerkkimme muuntuu seuraavaan muotoon:[15]

```
MOV &ZERO
ADD A
ADD B
STORE C
```

Viimeinen keino lyhentää käskysanaa on poistaa ainut jäljellä oleva parametri. Miten tällainen systeemi voi enää toimia? Kyseessä on niin sanottu pinoarkkitehtuuri[16][17], jossa myös operandien paikat ovat implisiittisesti tiedossa: Ne sijaitsevat pinon huipulla. Esimerkkikäskymme voisi näyttää pinojärjestelmässä seuraavanlaiselta:[18]

```
PUSH
PUSH
ADD
STORE
POP
POP
POP
```

Käskysanojen ADD ja POP toiminta on helppo ymmärtää, sillä ne operoivat pinossa sisäisesti. Sen sijaan käskysanat PUSH ja STORE eivät voi toimia täysin itsenäisesti. Voitaisiin toki ajatella, että ohjeet myös näille käskyille löytyisi pinosta, mutta tämä johtaisi rekursiiviseen ongelmaan siitä, miten käskyn parametri on alunperin saatu pinoon. Käytännössä parametrittomia käskyjä käyttävä pinokone voikin toimia vain jonkin ulkoisen, erillisesti ohjatun järjestelmän ohessa.

Edellä esitetyn mukaisesti voidaankin todeta, että tietokoneen käskyn muodon määrittäminen ei ole aina yksiselitteistä. Lukuunottamatta parametritonta käskyä, on siis käskysanoihin aina liitettävä mukaan jotain muuttuvaa informaatiota, kuten muistiosoitteita tai viittauksia rekistereihin, ennalta tiedetyn käskyn konekieliesityksen lisäksi. Ylimääräisen informaatiotarpeen määrä riippuu toteutuksesta. Myös yhden toteutuksen sisällä voidaan joutua tilanteeseen, että kaikkia käskyjä ei voida yleistää samaan muotoon. Esimerkiksi ehdottomalle hyppykäskyllä on oleellista saada yksi parametri, kohdeosoite, kun taas erilaisille aritmeettisille operaatioille tulee (akkujärjestelmää monimutkaisemmissa systeemeissä) antaa useampia parametrejä. Tämä johtaakin kahteen erilaiseen käskykannan suunnittelufilosofiaan: kiinteämittaisiin käskyihin ja vaihtelevan mittaisiin käskyihin. Muita oleellisia käskykannan suunnitteluun liittyviä kysymyksiä ovat käytössä olevien käskyjen määrä, tuettujen datatyyppien laatu sekä rekisterien ja muistinosoitusmuotojen moninaisuus.

[14] Jälleen hyppykäskyt aiheuttavat hiemen selittelyä, mutta akun sijaan implisiittisesti määrätty tuloksen tallennuspaikka onkin ohjelmalaskuri.

[15] Käskysana STORE tallentaa akussa olevan arvon parametrinä annettuun paikkaa.

[16] Pino on tietorakenne, jossa tallennettavia yksiköitä pinotaan toisten päälle ja operaatio suoritetaan aina pinon päältä.

[17] Pinoarkkitehtuurista voidaan tehdä myös versio, jossa on akkujärjestelmän tapaan parametri, mikä itse asiassa tekee järjestelmästä huomattavasti käyttökelpoisemman.

[18] Käskysana PUSH lataa parametrin pinon päälle ja POP poistaa päällimmäisen pinon alkion.