Tiedon esitysmuodot

Tiedon tyypit ja tiedon esitys laitteistossa

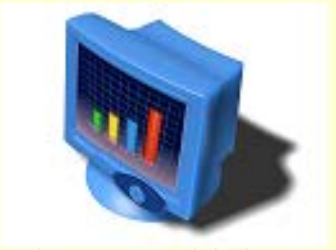
Lukujärjestelmät
Kokonaisluvut
Liukuluvut
Merkit, merkkijonot
Totuusarvot
Kuvat
Äänet
Mikä tahansa muu tieto, esimerkiksi hajut

Tässä luennossa käsitellään tietokonejärjestelmän ymmärtämän tiedon esitysmuodot. Luokittelemme ensin järjestelmän ymmärtämät tietotyypit ja sitten käymme läpi kaikki suorittimen ymmärtämät tietotyypit, painottuen kokonaislukujen ja liukulukujen esitystapoihin. Käsittelemme myös monitavuisen tiedon kaksi eri käytössä olevaa talletustapaa. Luennon alkupuolella kertaamme myös binääri, heksadesimaali ja oktaalijärjestelmien esitysmuodot ja niiden muunnokset eri lukujärjestelmien välillä.

Tietotyyppien luokittelu

Kommunikointi ihmisen kanssa

- kuva, ääni, merkit



kuva, merkki

ääni

Laitteiston sisäinen talletus

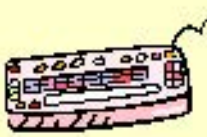
- kuvaformatit, ääniformatit, pakkausstandardit
- kokonaisluvut, liukuluvut, merkit, merkistöt
- kuvat, äänet, hajut
- ohjelmat

tunto, merkki

merkki

Suorittimen omana lajinaan ymmärtämät tietotyypit

- on olemassa konekäskyjä tälle tietotyypille
- kokonaisluvut (lähes kaikki suorittimet)
- liukuluvut (useimmat suorittimet)
- totuusarvot (jotkut suorittimet)
- merkit (jotkut suorittimet)
- äänet (erikoissuorittimet)
- kuvat (erikoissuorittimet)
- konekäskyt (kaikki suorittimet, vain omia käskyjä)



merkki

kuva, ääni, merkki

merkki

Mitä tahansa tietoa voidaan käsitellä laitteistossa, kunhan sen muodosta vain jotain sovitaan. Laitteiston kommunikointi ihmisen kanssa vaatii tietenkin, että tieto on sellaisessa muodossa, että ihminen voi sitä tuottaa tai ymmärtää. Ymmärtäminen voi perustua näköaistiin, tekstin tai (ehkä liikkuvien) kuvien muodossa. Tiedon voi myös kuulla tai aistia iholla tärähtelynä. Maku- tai hajuaisteja ei vielä juurikaan ole käytetty. Syötteen järjestelmälle voi antaa esimerkiksi ääninä, kuvina tai merkkeinä. Oleellista tämän tyyppisessä tiedossa on siis, että käyttäjän eli ihmisen täytyy pystyä sitä tuottamaan tai ymmärtämään.

Tietotyyppien luokittelu

Kommunikointi ihmisen kanssa

- kuva, ääni, merkit

Laitteiston sisäinen talletus

- kuvaformatit, ääniformatit, pakkausstandardit
- kokonaisluvut, liukuluvut, merkit, merkistöt
- kuvat, äänet, hajut
- ohjelmat

Suorittimen omana lajinaan ymmärtämät tietotyypit

- on olemassa konekäskyjä tälle tietotyypille
- kokonaisluvut (lähes kaikki suorittimet)
- liukuluvut (useimmat suorittimet)
- totuusarvot (jotkut suorittimet)
- merkit (jotkut suorittimet)
- äänet (erikoissuorittimet)
- kuvat (erikoissuorittimet)
- konekäskyt (kaikki suorittimet, vain omia käskyjä)

Laitteiston sisäinen tiedon talletus vaatii tiedolta täysin erilaista muotoa. Olennaista on, että tieto voidaan tallettaa bitteinä suht'koht pieneen tilaan käytettäville tallennuslaitteille. Tiedon tyypin tulee olla täsmällisesti speksattu, koska laitteisto ei sitä muuten voi täsmällisesti käsitellä. Toisaalta mikä tahansa tieto voidaan tallettaa laitteistoon, kunhan vain sen esitystapa on jollain tavoin speksattu. Tämä on itse asiassa osa ongelmaa, koska hyvin monelle tietotyypille on vaikka kuinka monta eri talletusformattia maailmassa. Olisi paljon yksinkertaisempaa, jos jokaiselle tietotyypille olisi yksi maailmanlaajuinen standardiesitystapa.

Tietotyyppien luokittelu

Kommunikointi ihmisen kanssa

- kuva, ääni, merkit

Laitteiston sisäinen talletus

- kuvaformatit, ääniformatit, pakkausstandardit
- kokonaisluvut, liukuluvut, merkit, merkistöt
- kuvat, äänet, hajut
- ohjelmat

Suorittimen omana lajinaan ymmärtämät tietotyypit

- on olemassa konekäskyjä tälle tietotyypille
- kokonaisluvut (lähes kaikki suorittimet)
- liukuluvut (useimmat suorittimet)
- totuusarvot (jotkut suorittimet)
- merkit (jotkut suorittimet)
- äänet (erikoissuorittimet)
- kuvat (erikoissuorittimet)
- konekaskyt (kaikki suorittimet, vain omia kaskyja)

ttk-91
```
LOAD  R1, R3
ADD   R1, R5
```

x86
```
mov  eax, p1
mov  r, ebx
```

i860
```
PFMULDD  F16, F12, F14
PFADDSS  F19, F17, F18
```

JVM
```
iLoad 1
iLoad 2
iAdd
iStore 0
```

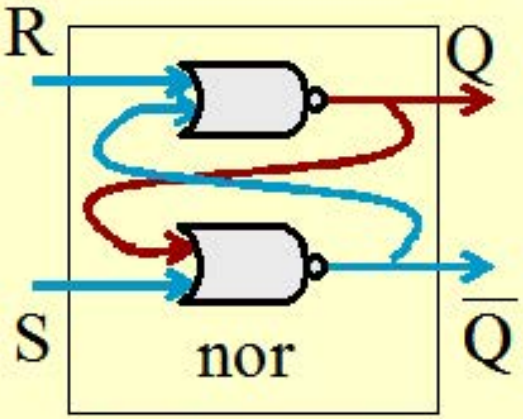
Mitä tietotyyppejä kännykkäsi tarvitsee?

Aivan oman lajinaan ovat sellaiset tietotyypit, joita varten suorittimessa on konekäskyjä. Useimmissa pöytäkoneissa suorittimella on konekäskyjä kokonaisluku- ja liukulukulaskentaan. Multimediakortin suoritin voi olla erikoistunut joidenkin kuva- tai äänifcrmattien käsittelyyn. Näytönohjainkortin suoritin voi sisältää joko erkoiskäskyjä kuvatiedon käsittelyyn, tai sitten kuvatiedon käsittelyä varten optimoituja kokonaisluku- tai liukulukukäskyjä. Jokainen suoritin osaa vain omaa konekieltään, mutta esimerkiksi Intelin suoritin ei ymmärrä Motorolan suorittimen konekieltä. Kaikki muu tieto pitää laskentaa varten ohjelmallisesti kuvata suorittimen ymmärtämään esitysmuotoon. Esimerkiksi kuvia käsitellään yleensä kokonaislukujen avulla.

Tiedon esitys laitteistossa

Kaikki tieto on bitteinä

- binäärijärjestelmän numerot 0 ja 1
- helppo toteuttaa biteillä
- helppo suunnitella logiikkaa Boolen algebran avulla



Yhden bitin rekisteri
(lisätietoja kurssilla
Tietokoneen rakenne)

Muisti on jaettu tasapituisiin sanoihin

- sana = word = 32 bittiä (ennen 16, kohta 64 tai 128?)

Sana on jaettu tasapituisiin tavuihin

- tavu = byte = 8 bittiä

Tieto siirretään muistiväylää pitkin sanoina

- joskus useampi (1-4) sana kerrallaan (lohko)

Suorittimen rekisterit ovat 1 tai 2 sanan pituisia

- sana: kokonaisluku, lyhyt (epätarkka) liukuluku, 1 tai 4 merkkiä
- 2 sanaa: pitkä kokonaisluku, tarkka liukuluku

Kaikki tieto laitteistossa on bitteinä. Miksi 2-järjestelmä eikä joku muu? Hyvä syy siihen on, että osaamme rakentaa binääritiedon talletukseen sopivia laitteita. Nor-portin arvo on 1, jos molemmat sen sisäänmenoista on 0. Tällaisia portteja on helppo rakentaa esimerkiksi transistoreilla ja niitä sopivasti yhdistelemällä voidaan rakentaa muistipiiri, jossa arvo säilyy. Toinen hyvä syy käyttää binäärijärjestelmää on George Boolen 1844 kehittämä Boolen algebra, minkä avulla juuri binäärijärjestelmäsysteemejä voidaan matemaattisesti analysoida ja optimoida. Boolen algebra on nykyäänkin tärkeä apuväline suorittimen logiikkaa suunniteltaessa.

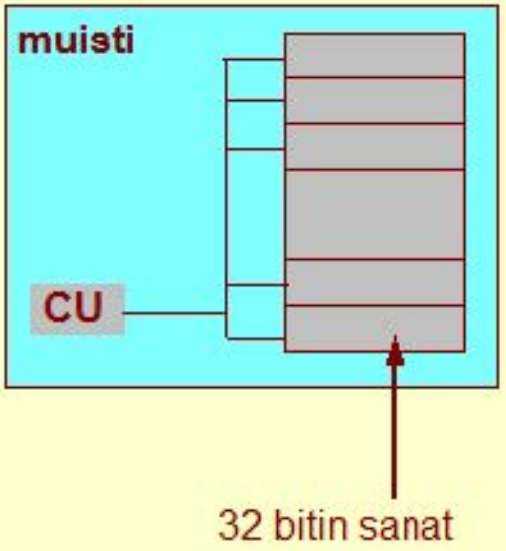
Tiedon esitys laitteistossa

Kaikki tieto on bitteinä

- binäärijärjestelmän numerot 0 ja 1
- helppo toteuttaa biteillä
- helppo suunnitella logiikkaa Boolen algebran avulla

Muisti on jaettu tasapituisiin sanoihin

- sana = word = 32 bittiä (ennen 16, kohta 64 tai 128?)



Sana on jaettu tasapituisiin tavuihin

- tavu = byte = 8 bittiä

Tieto siirretään muistiväylää pitkin sanoina

- joskus useampi (1-4) sana kerrallaan (lohko)

R1: 11**44**2233
R2: 00FF0000 (mask)

AND R1, R2
SHR R1, =16

R1: **44**

Suorittimen rekisterit ovat 1 tai 2 sanan pituisia

- sana: kokonaisluku, lyhyt (epätarkka) liukuluku, 1 tai 4 merkkiä
- 2 sanaa: pitkä kokonaisluku, tarkka liukuluku

Tietokoneen muistin perusyksikkö on sana. Ihan aikojen alussa sana oli 8 bittiä ja sitten 16 bittiä. Nyt se on 32 bittiä ja 64 bitin koneet ovat tuloillaan. Ehkä vielä tulevaisuudessa päästään 128 bitin sanoihin, mutta sen pitäisi jo riittää. Sana on muistinviittauksen perusyksikkö ja muistista haetaan ja sinne viedään tietoa aina kokonaisina sanoina. Jos tarvitsemme yhden tavun sanasta, niin ensin koko sana pitää hakea muistista, ja sitten suorittimessa bittimanipulaatiolla erottaa haluttu tavu sieltä esiin. Tämä on aikaa vievää ja juuri sen takia yhdelle 8-bitin merkille varataan usein kokonainen sana muistista.

Tiedon esitys laitteistossa

Kaikki tieto on bitteinä

- binäärijärjestelmän numerot 0 ja 1
- helppo toteuttaa biteillä
- helppo suunnitella logiikkaa Boolen algebran avulla

Muisti on jaettu tasapituisiin sanoihin

- sana = word = 32 bittiä (ennen 16, kohta 64 tai 128?)

Sana on jaettu tasapituisiin tavuihin

- tavu = byte = 8 bittiä

Tieto siirretään muistiväylää pitkin sanoina

- joskus useampi (1-4) sana kerrallaan (lohko)

Suorittimen rekisterit ovat 1 tai 2 sanan pituisia

- sana: kokonaisluku, lyhyt (epätarkka) liukuluku, 1 tai 4 merkkiä
- 2 sanaa: pitkä kokonaisluku, tarkka liukuluku

Sanaakin pienempi tiedon perusyksikkö on tavu, joka nykyään on aina 8 bittiä. Tavuun voi tallettaa hyvin pienen kokonaisluvun, mutta yleensä tavuihin talletetaan yksi merkki. Toisaalta, joissakin merkistöissä yhden merkin tallettamiseen tarvitaan kaksi tavua, mikä huomattavasti hankaloittaa esimerkiksi käyttöjärjestelmien tekemistä. Merkkimanipulaatiorutiinien toteutus kun menee vaikeaksi, jos edes merkin pituutta bitteinä ei voi etukäteen tietää. Bittien numerointi sekä sanoissa että tavuissa on oikealta vasemmalle, eli vähemmän merkitsevästä bitistä eniten merkitsevään bittiin ja numeroirti alkaa nollasta.

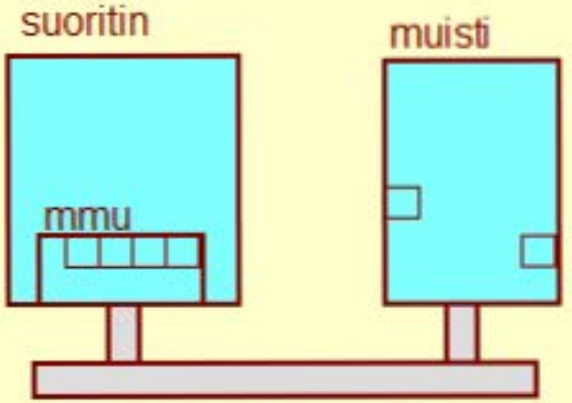
Tiedon esitys laitteistossa

Kaikki tieto on bitteinä

- binäärijärjestelmän numerot 0 ja 1
- helppo toteuttaa biteillä
- helppo suunnitella logiikkaa Boolen algebran avulla

Muisti on jaettu tasapituisiin sanoihin

- sana = word = 32 bittiä (ennen 16, kohta 64 tai 128?)



Sana on jaettu tasapituisiin tavuihin

- tavu = byte = 8 bittiä

Tieto siirretään muistiväylää pitkin sanoina

- joskus useampi (1-4) sana kerrallaan (lohko)

Suorittimen rekisterit ovat 1 tai 2 sanan pituisia

- sana: kokonaisluku, lyhyt (epätarkka) liukuluku, 1 tai 4 merkkiä
- 2 sanaa: pitkä kokonaisluku, tarkka liukuluku

Muistiväylää pitkin tieto siirtyy kokonaisina sanoina ja usein välimuistin tehokkuussyistä 1-4 sanan lohkoina. On tehokkaampaa väylän käyttöä siirtää kerralla aina vähän isompi määrä tietoa. Tästä yleislinjasta on tietenkin jonkin sortin variantteja, mutta ne kuuluvat jo jatkokurssin asioihin.

Tiedon esitys laitteistossa

Kaikki tieto on bitteinä

- binäärijärjestelmän numerot 0 ja 1
- helppo toteuttaa biteillä
- helppo suunnitella logiikkaa Boolen algebran avulla

-878666222

32 bitin sana

-878666222

3.1415927

'T'

'A'	'u'	't'	'o'

Muisti on jaettu tasapituisiin sanoihin

- sana = word = 32 bittiä (ennen 16, kohta 64 tai 128?)

Sana on jaettu tasapituisiin tavuihin

- tavu = byte = 8 bittiä

64 bitin kaksoissana

-8786662221999111 2

3.1415926535897932

Tieto siirretään muistiväylää pitkin sanoina

- joskus useampi (1-4) sana kerrallaan (lohko)

Suorittimen rekisterit ovat 1 tai 2 sanan pituisia

- sana: kokonaisluku, lyhyt (epätarkka) liukuluku, 1 tai 4 merkkiä
- 2 sanaa: pitkä kokonaisluku, tarkka liukuluku

Suorittimen rekistereiden pituus on 1 tai kaksi sanaa. Mikä tahansa muistissa oleva sana voidaan ladata rekisteriin ja sitten tulkita tilanteen mukaan esimerkiksi kokonaisluvuksi, liukuluvuksi tai konekäskyksi. Suuria kokonaislukuja ja tarkkoja liukulukuja käsiteltäessä suorittimessa voi olla kaksoistarkkuuden rekistereitä, joita vastaa muistissa kahden sanan lohko. Voi olla, että tällainen kaksoistarkkuden rekisteri vaatii kaksi konekäskyä tiedon hakemiseen muistista: ensimmäisellä haetaan 32 ensimmäistä bittiä, ja toisella LOAD-käskyllä loput 32 bittiä.

Tiedon esitys

Kysymys: miten esittää eri tyypisiä tietoja
Vastaus: koodataan tieto biteiksi

Kysymys: koodataan miten?
Vastaus: eri menetelmillä, jotka sopivat käyttötilanteeseen

Kaikille käsitellylle tiedolle on oma koodausmenetelmä

- kaikkia koodausmenetelmiä ei ole standardoitu
- useille tietotyypeille on monta koodausmenetelmää
  - kokonaisluvut, liukuluvut, merkit, merkkijonot, kuvat, äänet, ...

Ongelma: ymmärtävätkö koneet toisiaan?

- tiedon esitysmuotoa (koodausmenetelmää) voidaan joutua muuttamaan, kun tietoa siirretään koneesta toiseen
  - ihan yleinen tilanne
  - verkonhallintaohjelmat tekevät muunnokset automaattisesti

Vimpotin	
herkkä	0
särkyvä	1
normaali	2
kestävä	3
jäykkä	4
teräksinen	5

Tietokonelaitteistossa voidaan siis esittää mitä tahansa tietoa, kunhan vain etukäteen sovitaan sille tilanteen mukainen esitystapa. Kokonaisluvuille varmaankin kannattaa käyttää sellaista esitystapaa, mille suorittimen konekäskyt on toteutettu. Toisaalta taas esimerkiksi oman vimpottimen luokitteluun asteikolla herkästä teräksiseen sopii vaikkapa kokonaisluvut 0-5. Vimpottimen luokittelutyypin toteutus tapahtuu omassa ohjelmassa. Vimpottimen vimpotintyyppi voidaan siten valita ihan miksi vain ja ohjelma suunnitella sitten sen mukaisesti.

Tiedon esitys

Kysymys: miten esittää eri tyypisiä tietoja
Vastaus: koodataan tieto biteiksi

float: IEEE 32 bit
IEEE 63 bit extended

Kysymys: koodataan miten?
Vastaus: eri menetelmillä, jotka sopivat käyttötilanteeseen

int: 1-komplementti
2-komplementti

Kaikille käsitellylle tiedolle on oma koodausmenetelmä

- kaikkia koodausmenetelmiä ei ole standardoitu
- useille tietotyypeille on monta koodausmenetelmää
  - kokonaisluvut, liukuluvut, merkit, merkkijonot, kuvat, äänet, ...

char: EBCDIC
ASCII
UCS

fig: GIF, JPEG, TIFF

Ongelma: ymmärtävätkö koneet toisiaan?

- tiedon esitysmuotoa (koodausmenetelmää) voidaan joutua muuttamaan, kun tietoa siirretään koneesta toiseen
  - ihan yleinen tilanne
  - verkonhallintaohjelmat tekevät muunnokset automaattisesti

snd: MIDI
MS GS Wavetable
SW Synth

Kullekin tiedolle on siis tiedon käsittelyhetkellä tiedossa täsmällinen koodausmenetelmä. Esimerkiksi bittiyhdistelmä 1100 0111 voi tarkoittaa etumerkitöntä kokonaislukua 199 tai negatiivista luku -57 tai operaatiokoodia FADD tilanteesta riippuen. Kaikki tieto esitetään samannäköisillä biteillä, mutta bittien tulkinta riippuu käytettävästä tiedonkoodausmenetelmästä. Tilannetta vähän hankaloittaa yleismaailmallinen standardoinnin puute, mutta ei kovin paljoa. Tietokoneet ovat hyviä turhan tuntuisen nippelitiedon kanssa näpertelyyn.

Tiedon esitys

Kysymys: miten esittää eri tyypisiä tietoja
Vastaus: koodataan tieto biteiksi

Kysymys: koodataan miten?
Vastaus: eri menetelmillä, jotka sopivat käyttötilanteeseen

Kaikille käsitellylle tiedolle on oma koodausmenetelmä

- kaikkia koodausmenetelmiä ei ole standardoitu
- useille tietotyypeille on monta koodausmenetelmää
  - kokonaisluvut, liukuluvut, merkit, merkkijonot, kuvat, äänet, ...

Ongelma: ymmärtävätkö koneet toisiaan?

- tiedon esitysmuotoa (koodausmenetelmää) voidaan joutua muuttamaan, kun tietoa siirretään koneesta toiseen
  - ihan yleinen tilanne
  - verkonhallintaohjelmat tekevät muunnokset automaattisesti

-57 = 1011 1001
-57 = 1100 0110
-57 = 1100 0111
-57 = 0100 0110

Jopa kokonaislukujen esitysmuodot voivat olla erilaisia eri koneissa. Kun siirrämme tietoa koneesta toiseen, niin tietenkin esitystapa voi vaihtua, mutta itse tiedon tulee säilyä muuttumattona. Tieto on siis syvällisempi käsite kuin sen esitystapa. Luku -57 on aina luku -57, vaikka sen esitystapa yhdessä koneessa olisi 1011 1001 ja toisessa 0100 0110. Aina siirrettäessä tietoa verkon ylitse tiedonsiirto-ohjelmat ensin muokkaavat lähetettävän tiedon geneeriseen muotoon, josta vastaanottavassa solmussa se muunnetaan sen solmun käyttämään esitystapaan.

Suorittimen ymmärtämä tieto

Kaikki tieto on koodattuna biteiksi

Muistissa voidaan esittää mikä tahansa tieto sovitulla esitystavalla (koodauksella)

Suoritin osaa tehdä operatioita joillakin esitystavoilla koodatuille tiedoille

- kokonaisluvut ja liukuluvut (lähes aina)
- totuusarvot, merkit, merkkijonot (joskus)
- kuvat, äänet (vain erikoissuorittimilla)
- hajut, tunto (ei vielä)

Kaikkien muiden tietojenkäsittely tapahtuu ohjelmallisesti

- suorittimen ymmärtämien tietotyyppien avulla
- esim. merkit koodattu kokonaisluvuilla ja niitä voidaan käsitellä kokonaislukuina
  - kokonaislukukäsky tulkitsee merkkikoodin 'A' = 0100 0001 lukuna 65

Muistissa olevalle tiedolle voicaan siis käyttää mitä tahansa koodausta. Tottakai tietoa on vaikea käsitellä, ellei sitä manipuloivat ohjelmat ole tietoisia tuosta koodauksesta. Yleisesti käytössä olevasta tiedosta käytetään jotain yleisesti käytössä olevaa standardikoodausta, mutta käyttäjän ihan omille tietotyypeille hän voi itse valita sopivan koodaustavan ja sitten noudattaa sitä ohjelmassaan. Tiedon koodaustavan saaminen yleismaailmalliseksi standardiksi on aikaa vievää puuhaa ja vaatii paljon neuvotteluja erilaisten yhteistyötahojen välillä. Joillekin yrityksille tämä on tietenkin helpompaa kuin toisille...

Suorittimen ymmärtämä tieto

Kaikki tieto on koodattuna biteiksi

Muistissa voidaan esittää mikä tahansa
tieto sovitulla esitystavalla (koodauksella)

Suoritin osaa tehdä operatioita joillakin
esitystavoilla koodatuille tiedoille

- kokonaisluvut ja liukuluvut (lähes aina)
- totuusarvot, merkit, merkkijonot (joskus)
- kuvat, äänet (vain erikoissuorittimilla)
- hajut, tunto (ei vielä)

Kaikkien muiden tietojenkäsittely tapahtuu ohjelmallisesti

- suorittimen ymmärtämien tietotyyppien avulla
- esim. merkit koodattu kokonaisluvuilla ja niitä voidaan käsitellä kokonaislukuina
  - kokonaislukukäsky tulkitsee merkkikoodin 'A' = 0100 0001 lukuna 65

Suorittimella on siis konekäskyjä muutamalle perustietotyypille. Yleensä kaikki osaavat laskea kokonaisluvuilla ja melkein kaikki myös liukuluvuilla. Mutta esimerkiksi digitaalikelloni tai pesukoneeni tuskin tarvitsevat liukulukukäskyjä. Jotkut vanhemmat suorittimet sisältivät omia käskyjä totuusarvoille, mutta nykyisissä suorittimissa sama työ voidaan hyvin tehdä kokonaislukukäskyillä. Sama pätee merkkien käsittelyyn. Edelleenkin joissakin suorittimissa on omia käskyjä merkkijonojen käsittelyyn ja niiden avulla voidaan tehokkaasti siirtää muutakin tietoa muistin sisällä.

Suorittimen ymmärtämä tieto

Kaikki tieto on koodattuna biteiksi

Muistissa voidaan esittää mikä tahansa tieto sovitulla esitystavalla (koodauksella)

Suoritin osaa tehdä operatioita joillakin esitystavoilla koodatuille tiedoille

- kokonaisluvut ja liukuluvut (lähes aina)
- totuusarvot, merkit, merkkijonot (joskus)
- kuvat, äänet (vain erikoissuorittimilla)
- hajut, tunto (ei vielä)

```
ChComp  R1, 'A' ; tätä konekäskyä ei ole
```

0100 0001 → 'A' (merkki)

0100 0001 → 65 (kokonaisluku)

```
LOAD    R1, Ch   ; hae merkki
COMP    R1, =65  ; onko 'A'?
```

Kaikkien muiden tietojenkäsittely tapahtuu ohjelmallisesti

- suorittimen ymmärtämien tietotyyppien avulla
- esim. merkit koodattu kokonaisluvuilla ja niitä voidaan käsitellä kokonaislukuina
  - kokonaislukukäsky tulkitsee merkkikoodin 'A' = 0100 0001 lukuna 65

Muiden kuin suorittimen ymmärtämien tietojen käsittely tapahtuu ohjelmallisesti manipuloiden tiedon esitystapaa joko kokonaisluku- tai liukulukukäskyillä. Oletetaan esimerkiksi, että muuttujaan Ch on syötetty jokin merkki ja haluaisimme tarkistaa, onko se kirjain 'A'. Tehtävä olisi helppo, jos konekäskykantaan sisältyisi tietotyyppi 'merkki' ja vertailun voisi tehdä suoraan merkkien vertailukäskyllä ChComp. Nyt jos tällaista käskyä ei ole, niin voimme tehdä saman vertailun kokonaislukujen vertailukäskyllä, kun tiedämme, että merkin 'A' esitysmuoto koneessa on sama bittiyhdistelmä kuin kokonaisluvulla 65.

Binäärijärjestelmä

Desimaaliluvun numeroiden painoarvot oikealta vasemmalle

$1 = 10^0$, $10 = 10^1$, $100 = 10^2$, $1000 = 10^3$, ...

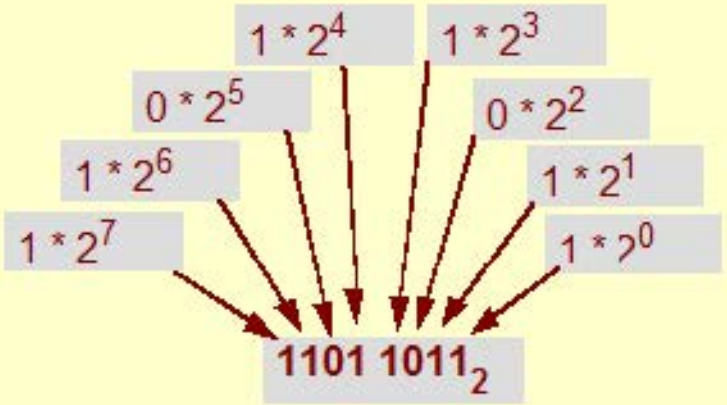
nrot: **6509**
järj. nrot: 3210

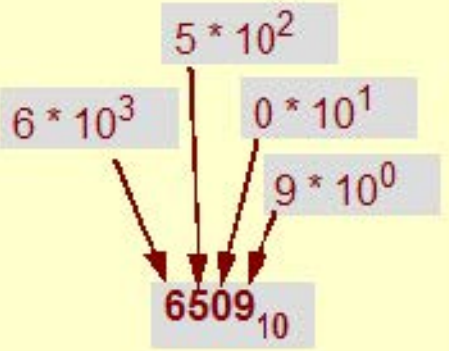
Binäärijärjestelmän kantaluku on 2, numerot 0 ja 1

bitit: **1101 1011**
järj. nrot: 7654 3210

Binääriluvun numeroiden painoarvot oikealta vasemmalle

$1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, ...





Kertaamme nyt tässä binääri-, heksadesimaali- ja oktaalijärjestelmien peruspiirteet siltä varalta, että tiedot ovat päässeet unohtumaan. Jos luulet, että sinulla on hyvin tiedossa näiden lukujärjestelmien peruspiirteet ja osaat sujuvasti muuttaa tiedon esitystapoja desimaalijärjestelmän ja näiden lukujärjestelmien välillä, niin voit hypätä suoraan seuraavaan alilukuun.

Binäärijärjestelmä

Desimaaliluvun numeroiden painoarvot oikealta vasemmalle

$1 = 10^0$, $10 = 10^1$, $100 = 10^2$, $1000 = 10^3$, ...

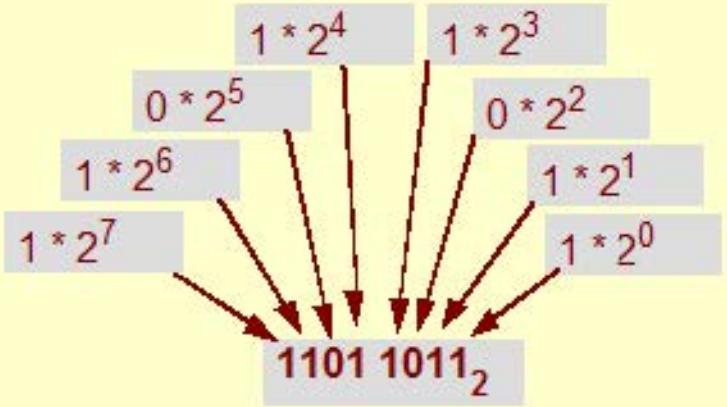
nrot: **6509**
järj. nrot: 3210

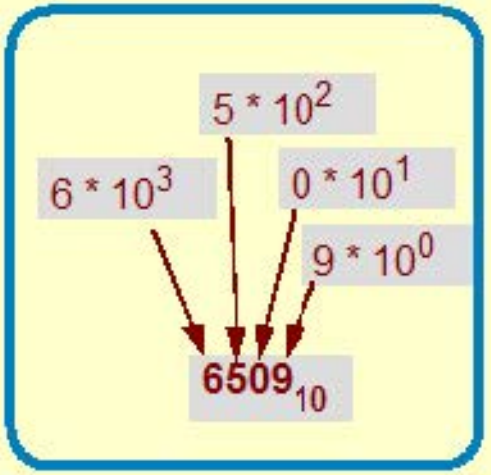
Binäärijärjestelmän kantaluku on 2, numerot 0 ja 1

bitit: **1101 1011**
järj. nrot: 7654 3210

Binääriluvun numeroiden painoarvot oikealta vasemmalle

$1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, ...





Desimaaliluvun esitysmuoto on peräkkäisten desimaalinumeroiden jono. Luvun numerot numeroidaan oikealta vasemmalle, alkaen nollasta. Numeroiden painoarvot ovat, jälleen oikealta vasemmalle, 10 potenssiin i, missä i on numeron järjestysluku. Painoarvot ovat siis 1, 10, 100, 1000, jne.

Binäärijärjestelmä

Desimaaliluvun numeroiden painoarvot oikealta vasemmalle

$1 = 10^0$, $10 = 10^1$, $100 = 10^2$, $1000 = 10^3$, ...

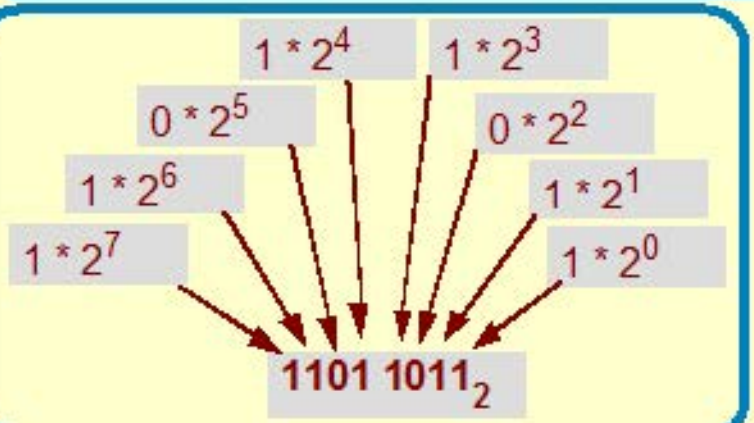
nrot: **6509**
järj. nrot: 3210

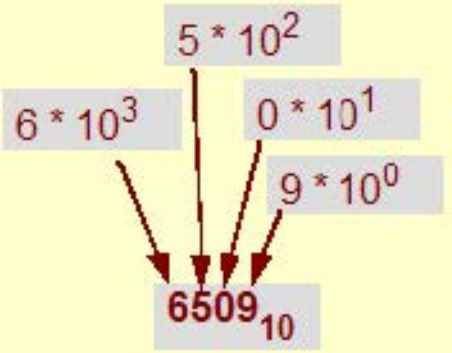
Binäärijärjestelmän kantaluku on 2, numerot 0 ja 1

bitit: **1101 1011**
järj. nrot: 7654 3210

Binääriluvun numeroiden painoarvot oikealta vasemmalle

$1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, ...





Binäärijärjestelmän lukujen esitysmuoto on ihan vastaava, kun ottaa huomioon, että kantaluku on nyt 2 ja numeroita (bittejä) on vain 0 ja 1. Bitit numeroidaan vastaavasti oikealta vasemmalle, alkaen nollasta, ja bitin painoarvo on kantaluku 2 korotettuna bitin järjestysnumeron ilmaisemaan potenssiin. Painoarvot ovat siis (desimaalijärjestelmän lukuina ilmaistuna) 1, 2, 4, 8, jne.

Binäärijärjestelmä

Desimaaliluvun numeroiden painoarvot oikealta vasemmalle

$1 = 10^0$, $10 = 10^1$, $100 = 10^2$, $1000 = 10^3$, ...

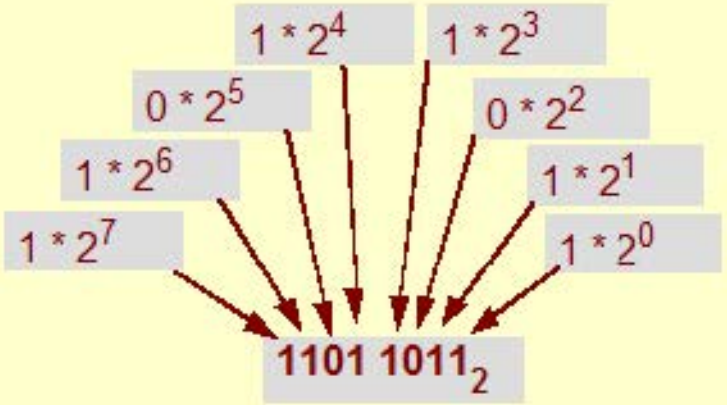
nrot: **6509**
jarj. nrot: 3210

Binäärijärjestelmän kantaluku on 2, numerot 0 ja 1

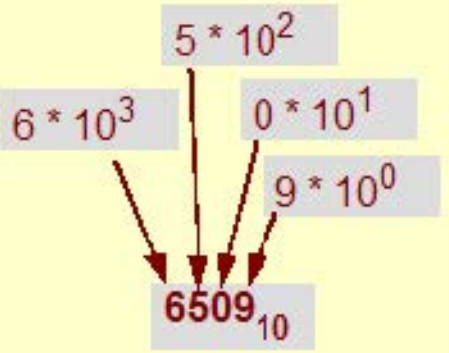
bitit: **1101 1011**
järj. nrot: 7654 3210

Binääriluvun numeroiden painoarvot oikealta vasemmalle

$1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, ...



0011 1001$_2$ = ?



Harjoitellaan pikkaisen. Mikä on tätä binaarilukua vastaava desimaaliluku? Mieti ensin ja katso vastaus seuraavasta näkymästä.

Binäärijärjestelmä

Desimaaliluvun numeroiden painoarvot oikealta vasemmalle

$1 = 10^0$, $10 = 10^1$, $100 = 10^2$, $1000 = 10^3$, ...

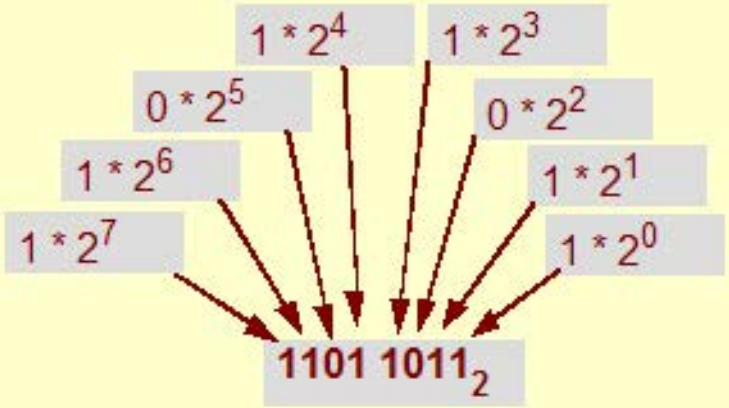
nrot: **6509**
järj. nrot: 3210

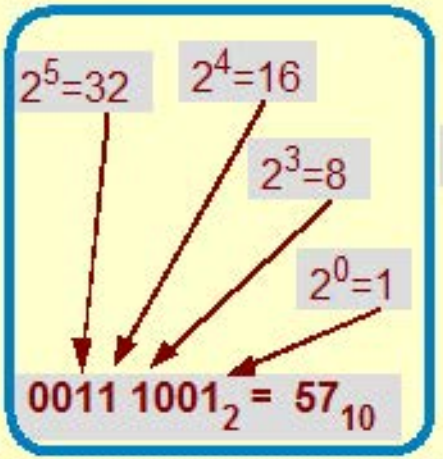
Binäärijärjestelmän kantaluku on 2, numerot 0 ja 1

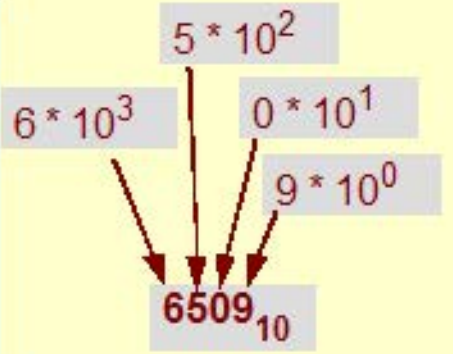
bitit: **1101 1011**
järj. nrot: 7654 3210

Binääriluvun numeroiden painoarvot oikealta vasemmalle

$1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, ...







Kutakin ykkösbittiä vastaavat binääriluvun painoarvot lasketaan yhteen ja tuloksena on vastaus 57.

Binäärijärjestelmä

Desimaaliluvun numeroiden painoarvot oikealta vasemmalle

$1 = 10^0$, $10 = 10^1$, $100 = 10^2$, $1000 = 10^3$, ...

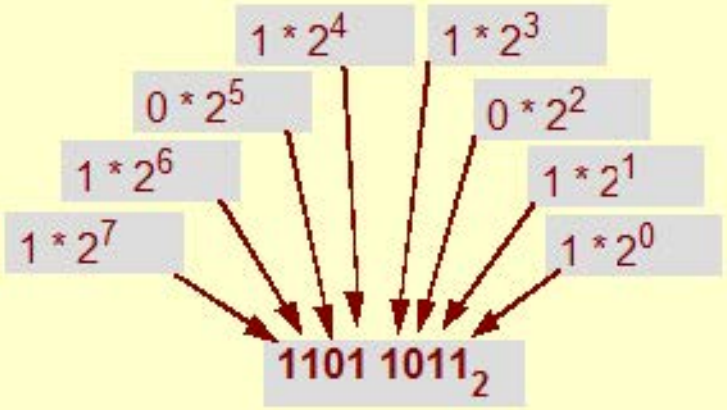
nrot: **6509**
järj. nrot: 3210

Binäärijärjestelmän kantaluku on 2, numerot 0 ja 1

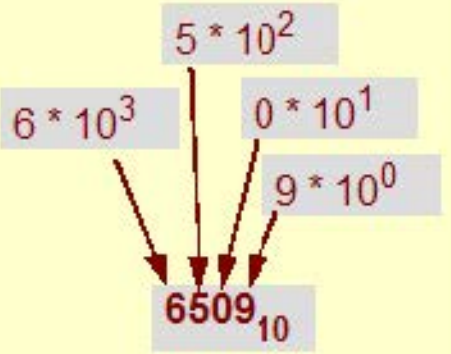
bitit: **1101 1011**
järj. nrot: 7654 3210

Binääriluvun numeroiden painoarvot oikealta vasemmalle

$1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, ...



0000 0011$_2$ = ?



Entä mitä desimaalilukua tarkoittaa binääriluku 11?

Binäärijärjestelmä

Desimaaliluvun numeroiden painoarvot oikealta vasemmalle

$1 = 10^0$, $10 = 10^1$, $100 = 10^2$, $1000 = 10^3$, ...

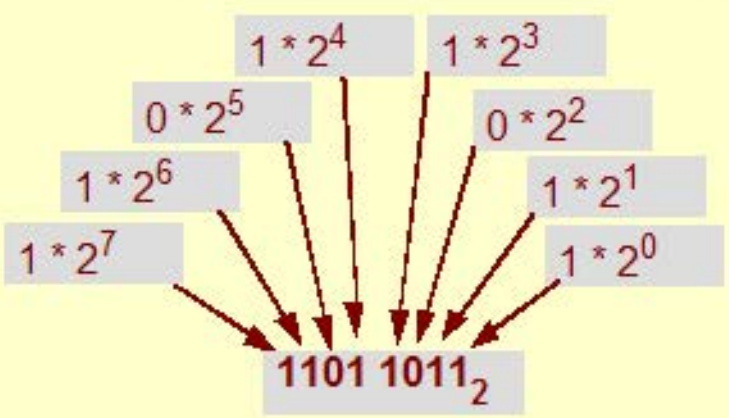
nrot: **6509**
järj. nrot: 3210

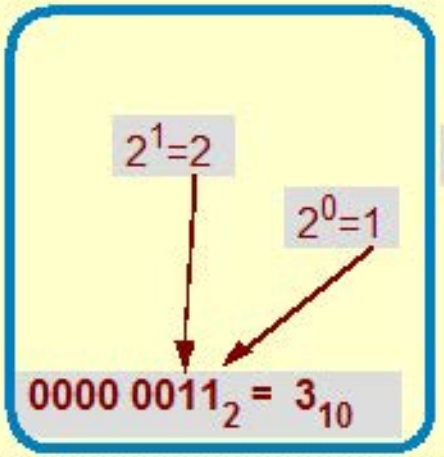
Binäärijärjestelmän kantaluku on 2, numerot 0 ja 1

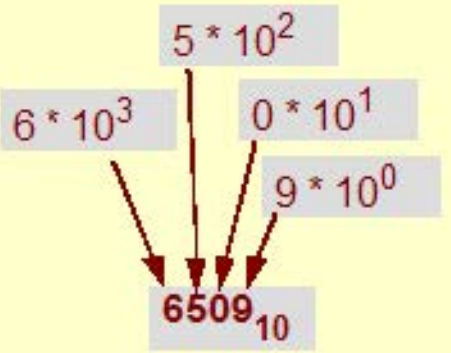
bitit: **1101 1011**
järj. nrot: 7654 3210

Binääriluvun numeroiden painoarvot oikealta vasemmalle

$1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, ...







Jos tämä meni väärin, niin sinun on syytä palata 'lähtöruutuun' eli tämän sivun alkuun, ja paneutua asiaan vähän enemmän keskittyen.

Binäärijärjestelmä

Desimaaliluvun numeroiden painoarvot oikealta vasemmalle

$1 = 10^0$, $10 = 10^1$, $100 = 10^2$, $1000 = 10^3$, ...

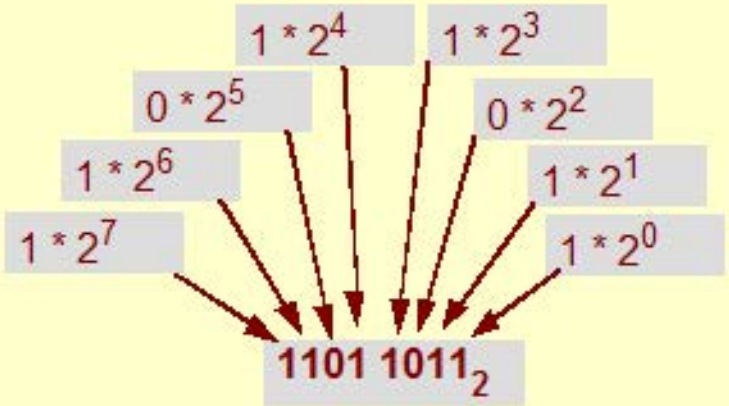
nrot: **6509**
järj. nrot: 3210

Binäärijärjestelmän kantaluku on 2, numerot 0 ja 1

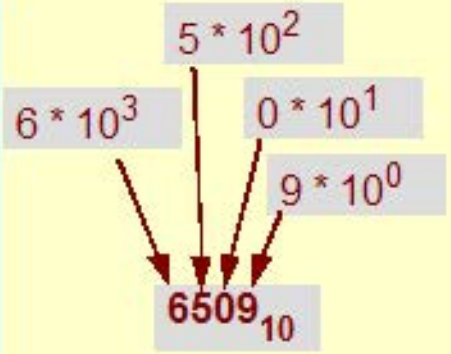
bitit: **1101 1011**
järj. nrot: 7654 3210

Binääriluvun numeroiden painoarvot oikealta vasemmalle

$1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, ...



0101 0101$_2$ = ?



Vielä viimeinen tämän tyyppinen harjoitus. Mikä desimaaliluku?

Binäärijärjestelmä

Desimaaliluvun numeroiden painoarvot oikealta vasemmalle

$1 = 10^0, 10 = 10^1, 100 = 10^2, 1000 = 10^3, ...$

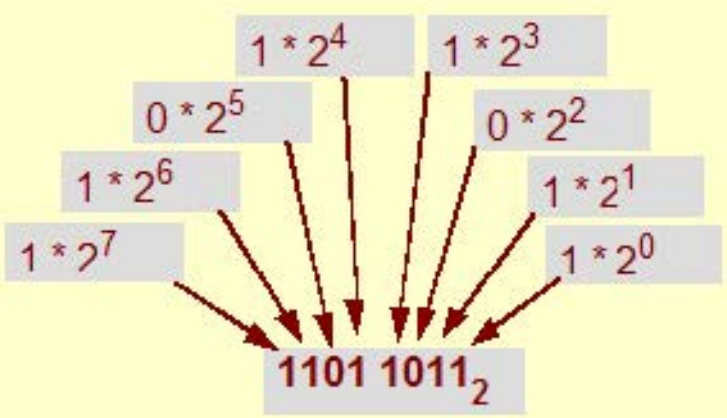
nrot: **6509**
järj. nrot: 3210

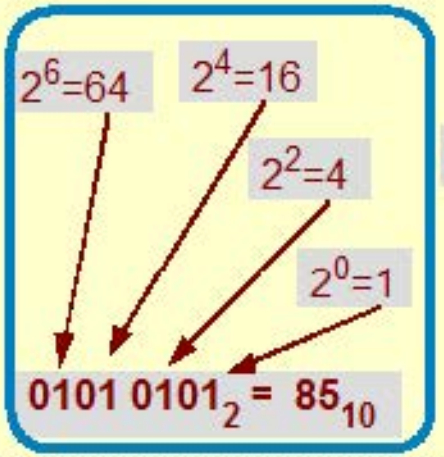
Binäärijärjestelmän kantaluku on 2, numerot 0 ja 1

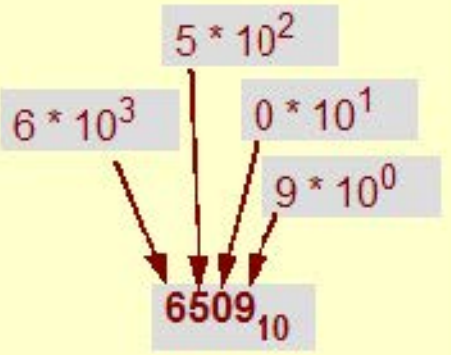
bitit: **1101 1011**
järj. nrot: 7654 3210

Binääriluvun numeroiden painoarvot oikealta vasemmalle

$1 = 2^0, 2 = 2^1, 4 = 2^2, 8 = 2^3, 16 = 2^4, 32 = 2^5, ...$







85.

Binäärijärjestelmä

Desimaaliluvun numeroiden painoarvot oikealta vasemmalle

$1 = 10^0$, $10 = 10^1$, $100 = 10^2$, $1000 = 10^3$, ...

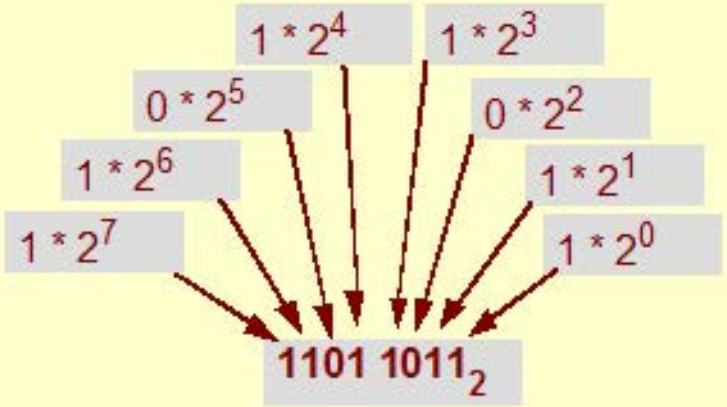
nrot: **6509**
järj. nrot: 3210

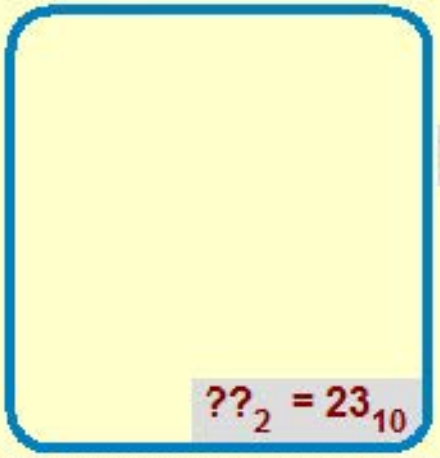
Binäärijärjestelmän kantaluku on 2, numerot 0 ja 1

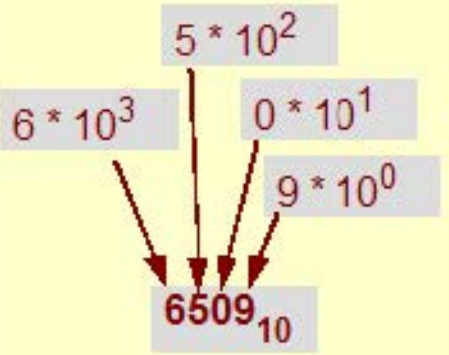
bitit: **1101 1011**
järj. nrot: 7654 3210

Binääriluvun numeroiden painoarvot oikealta vasemmalle

$1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, ...







Entäs toisinpäin. Mikä on desimaaliluvun 23 binääriesitys?

Binäärijärjestelmä

Desimaaliluvun numeroiden painoarvot oikealta vasemmalle

$1 = 10^0, 10 = 10^1, 100 = 10^2, 1000 = 10^3, ...$

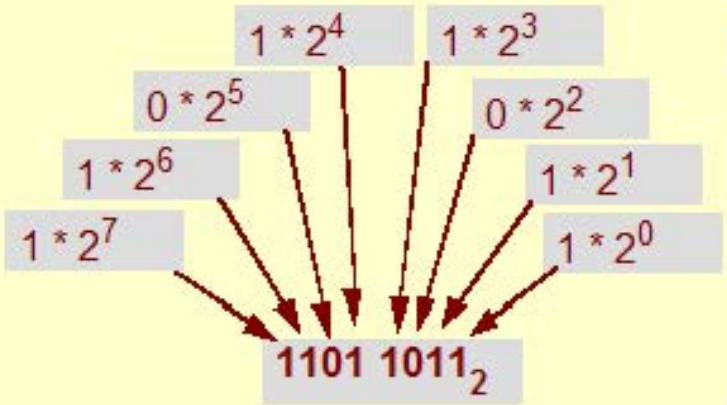
nrot: **6509**
järj. nrot: 3210

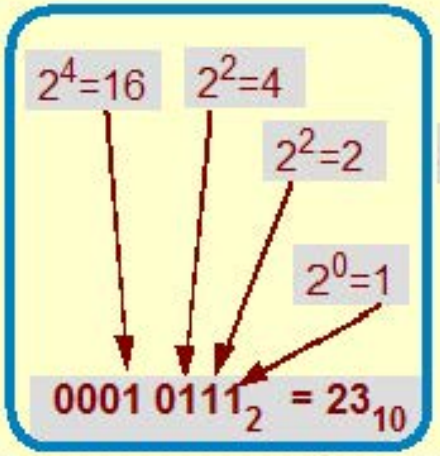
Binäärijärjestelmän kantaluku on 2, numerot 0 ja 1

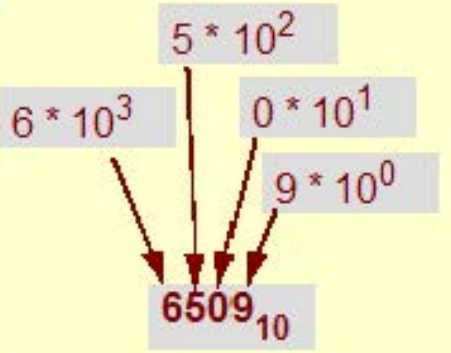
bitit: **1101 1011**
järj. nrot: 7654 3210

Binääriluvun numeroiden painoarvot oikealta vasemmalle

$1 = 2^0, 2 = 2^1, 4 = 2^2, 8 = 2^3, 16 = 2^4, 32 = 2^5, ...$







Tämä onkin jo vähän vaikeampaa. Näin pienillä luvuilla vastauksen saa liki kokeilemalla. Perusidea on kuitenkin ajatella kakkosen potensseja ja miettiä, minkä kakkosen potenssien summana kyseinen luku voidaan esittää. Lopuksi sitten kuvataan kukin kakkosen potenssi omaksi bitikseen, joista jokainen edustaa yhtä kakkosen potenssia.

Binäärilukujen laskutoimitukset

+	0	1
0	0	1
1	1	10

```
  11
 101111
+1100
------
 111011
```

```
 47
+12
---
 59
```

2-järjestelmä **10-järjestelmä**

*	0	1
0	0	0
1	0	1

```
    1101
   *1100
--  ----
  1101
+1101
------
10011100
```

```
 13
*12
---
 26
+13
---
156
```


Binääriluvuilla lasketaan ihan samalla tavalla kuin desimaaliluvuillakin. Laskutoimitukset ovat käytännössä tietenkin paljon helpompia 2-järjestelmässä kuin 10-järjestelmässä, koska perusoperaatioita 1-numercisten lukujen välillä on niin vähän. Binäärilukujen yhteenlaskutaulussa on vain kaksi riviä ja kaksi saraketta, ja ainoa epätriviaali laskutoimitus tulee kohdassa 1+1, jossa tuloksena on kaksi eli binäärinä '10'.

Binäärilukujen laskutoimitukset

+	0	1
0	0	1
1	1	10

```
   11
  101111
 +1100
 ------
  111011
```

2-järjestelmä

```
  47
 +12
 ---
  59
```

10-järjestelmä

*	0	1
0	0	0
1	0	1

```
    1101
   *1100
 ~~11~~
    1101
 +1101
 -------
 10011100
```

```
  13
 *12
 ---
  26
 +13
 ---
 156
```


Varsinainen monibittinen yhteenlasku tapahtuu nyt ihan samalla tavalla kuin opitte koulussa laskemaan 10-järjestelmän lukuja yhteen allekkain. Toimitus on nyt vain paljon helpompi, koska mahdollisia yhden bittiposition yhteenlaskuvaihtoehtoja on niin vähän. Muistinumero tai muistibitti tietenkin vähän hankaloittaa tilannetta, mutta ei paljoa.

Binäärilukujen laskutoimitukset

+	0	1
0	0	1
1	1	10

```
   11
  101111
 +1100
 ------
  111011
```

```
  47
 +12
 ---
  59
```

2-järjestelmä **10-järjestelmä**

*	0	1
0	0	0
1	0	1

```
    1101
   *1100
 ~~11~~
    1101
 +1101
 -------
 10011100
```

```
  13
 *12
 ---
  26
 +13
 ---
 156
```


Kertotaulu on vielä yksinkertaisempi kuin yhteenlaskutaulu. Minkä tahansa kahden bitin tulo kun on nolla, paitsi jos molemmat bitit ovat ykkösiä. Omat lapseni käyttivät kokonaisen vuoden ala-asteella 10-järjestelmän kertotaulun oppimiseen, joten hekin osaavat arvostaa binäärijärjestelmän helppoutta.

Binäärilukujen laskutoimitukset

+	0	1
0	0	1
1	1	10

*	0	1
0	0	0
1	0	1

```
   11
  101111
 +1100
 ------
  111011
```

```
  47
 +12
 ---
  59
```

2-järjestelmä

```
   1101
  *1100
 ------
   1101
  1101
 +1101
 -------
  10011100
```

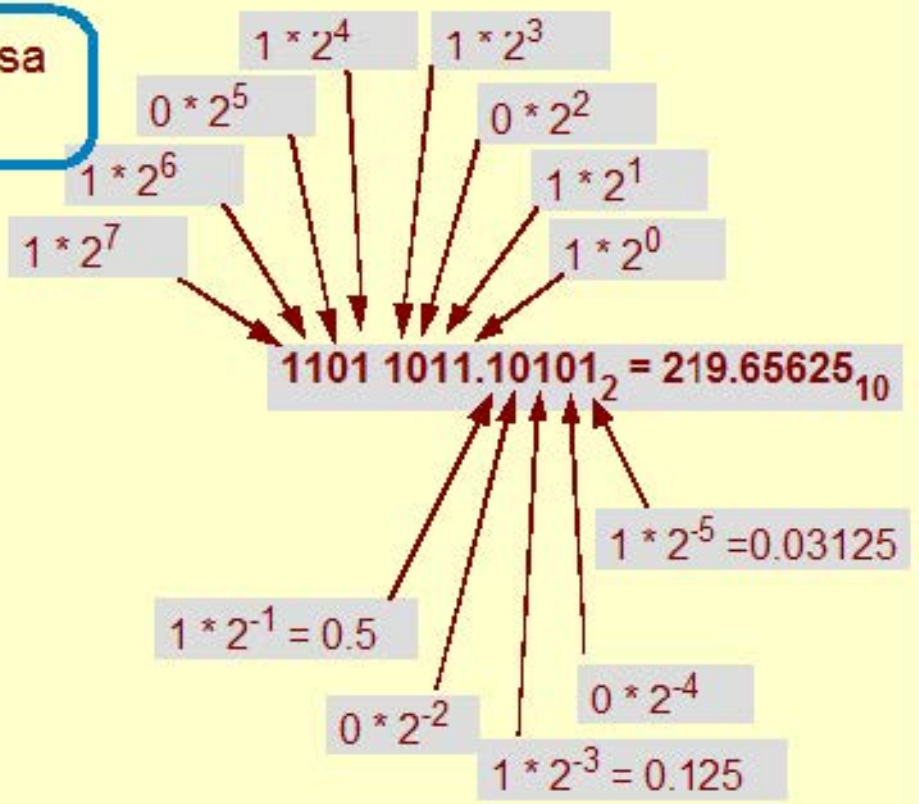
10-järjestelmä

```
  13
 *12
 ---
  26
 13
 ---
 156
```


Monibittisten lukujen kertominenkin tapahtuu samalla algoritmilla kuin mitä opitte koulussa 10-järjestelmälle. Binäärijärjestelmän kertotaulun yksinkertaisuuden vuoksi kerrottava tulee sellaisenaan mukaan jokaisen kertojan 1-bitin kohdalla. Kertolaskun toteutuksessa ei tarvitse osata kertoa lainkaan, riittää, kun kerrottava kopioidaan sellaisenaan aina tarvittaessa.

Binääripiste

Binääriluvuilla voi olla myös binääriosa
(vrt. desimaaliluvun desimaaliosa)

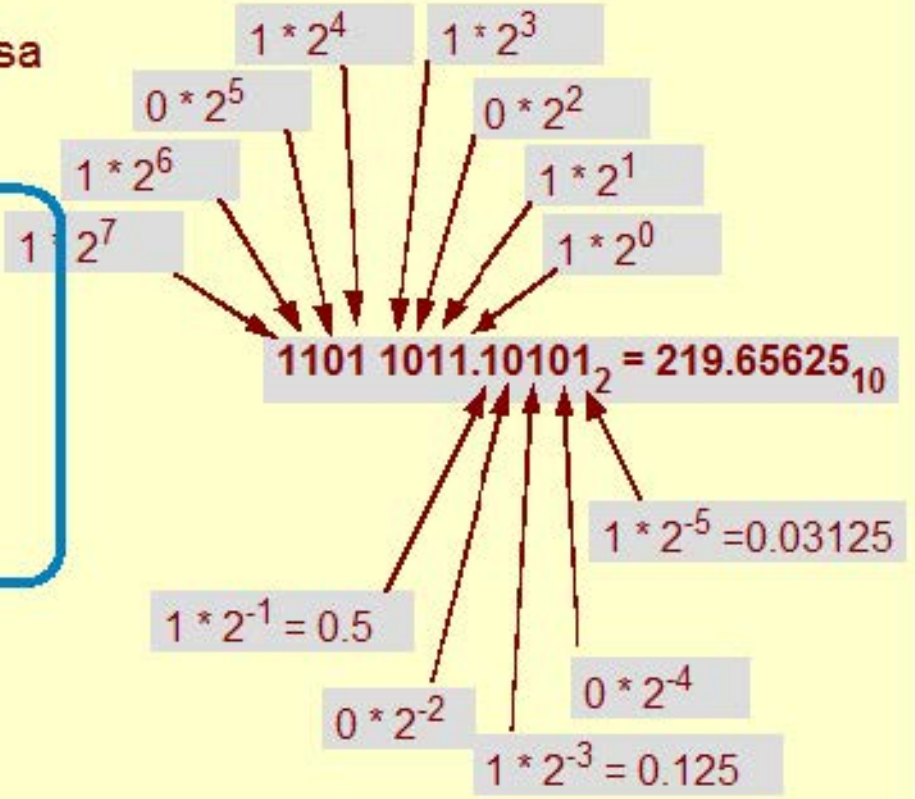


Binääriluvuilla on desimaalilukujen desimaaliosaa vastaava binääriosa. Kun desimaaliluvulla desimaalipisteen jälkeisillä luvuilla on painoarvot 1/10, 1/100, 1/1000, jne, niin binääriluvuilla binääripisteen jälkeen vastaavat bittien painoarvot ovat 1/2, 1/4, 1/8, jne. Luvun desimaaliarvon laskemiseksi sitten 1-bittejä vastaavat painokertoimet pitää laskea yhteen.

Binääripiste

Binääriluvuilla voi olla myös binääriosa
(vrt. desimaaliluvun desimaaliosa)

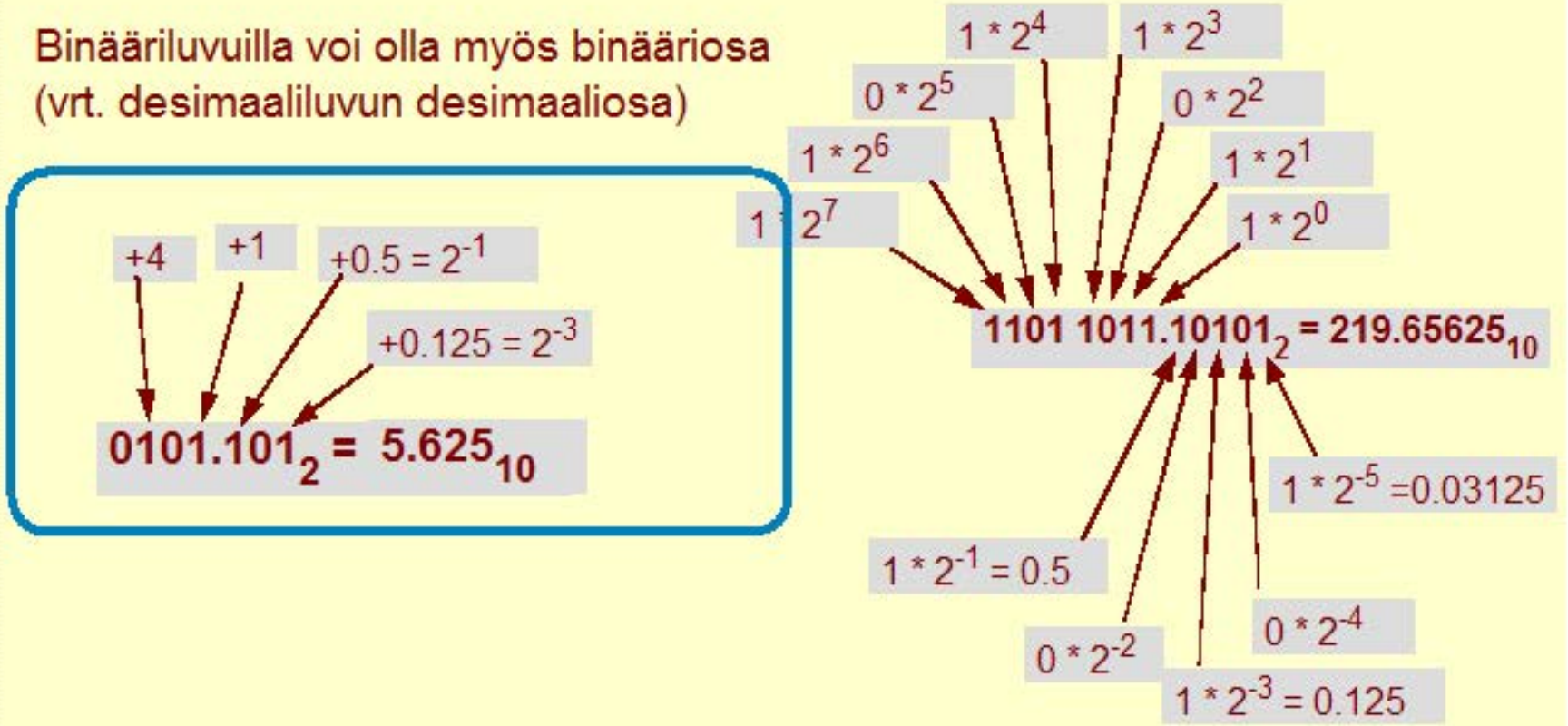
0101.101_2 = ??



Harjoitellaan vähän. Mitäs tämä on desimaalilukna?

Binääripiste

Binääriluvuilla voi olla myös binääriosa
(vrt. desimaaliluvun desimaaliosa)

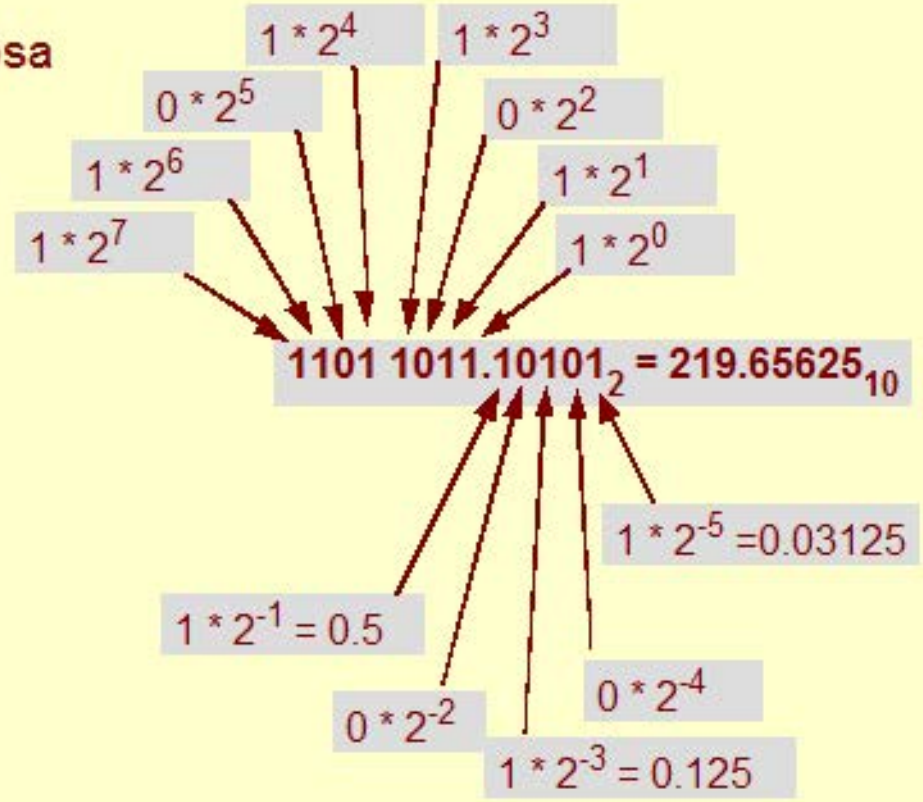


Kokonaisosa ja binääriosa lasketaan erikseen aikaisemmin mainittujen sääntöjen mukaisesti. 4+1=5 ja 1/2 + 1/8 = 0.625.

Binääripiste

Binääriluvuilla voi olla myös binääriosa
(vrt. desimaaliluvun desimaaliosa)

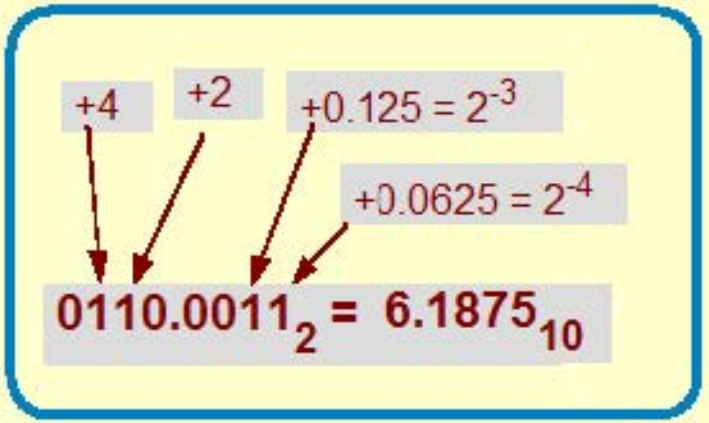
0110.0011_2 = ??

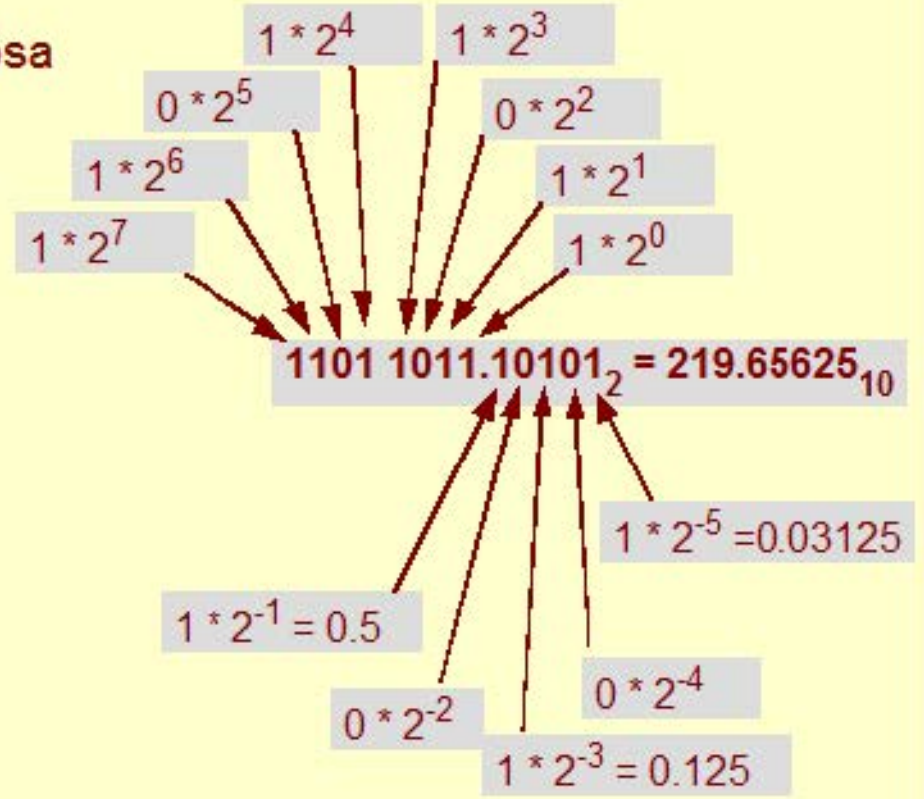


Entäs tämä?

Binääripiste

Binääriluvuilla voi olla myös binääriosa
(vrt. desimaaliluvun desimaaliosa)

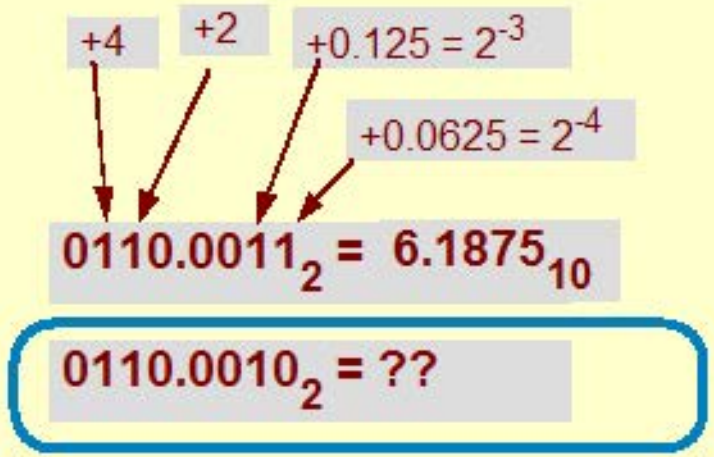


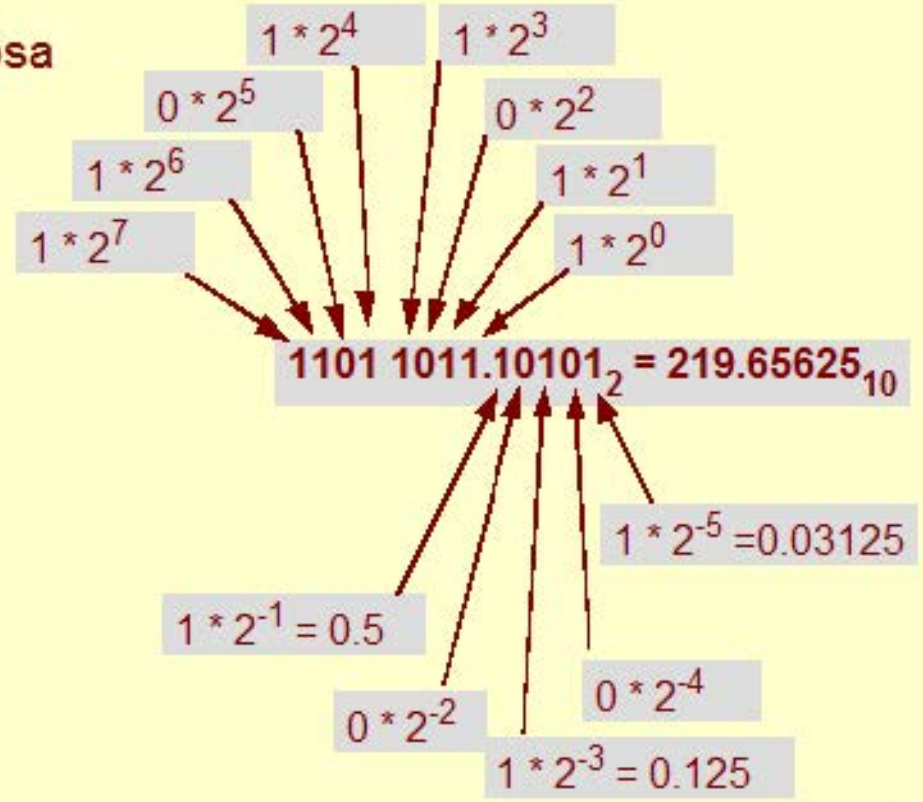


Meille tuottaa vähän hankaluuksia nuo ikävät murtoluvut 1/8 ja 1/16, jotka eivät ole mitään mukavia pyöreitä lukuja desimaalijärjestelmässä. Onneksi kaikille kakkosen negatiivisille potensseille on tarkka desimaaliesitys, toisin kuin esimerkiksi luvun kolme negatiivisille potensseille. Esimerkiksi luvulle 1/3 ei ole mitään tarkkaa desimaaliesitystä, vaan likiarvo 0.333333333.

Binääripiste

Binääriluvuilla voi olla myös binääriosa
(vrt. desimaaliluvun desimaaliosa)

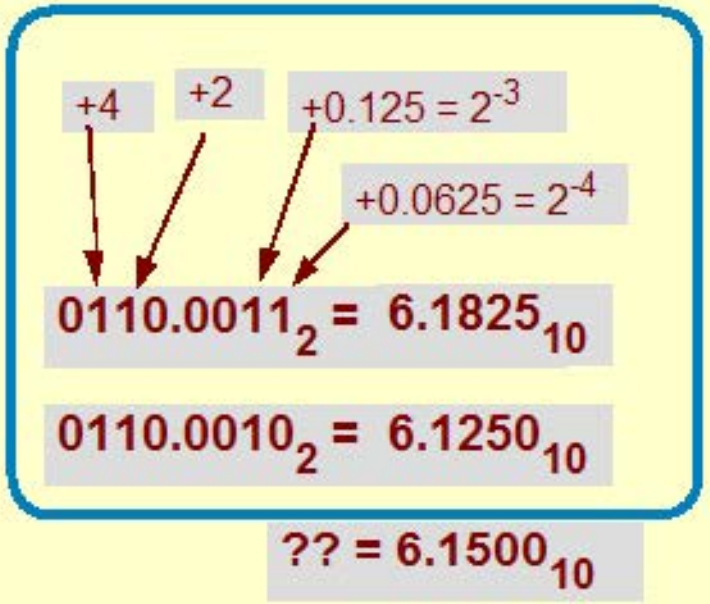


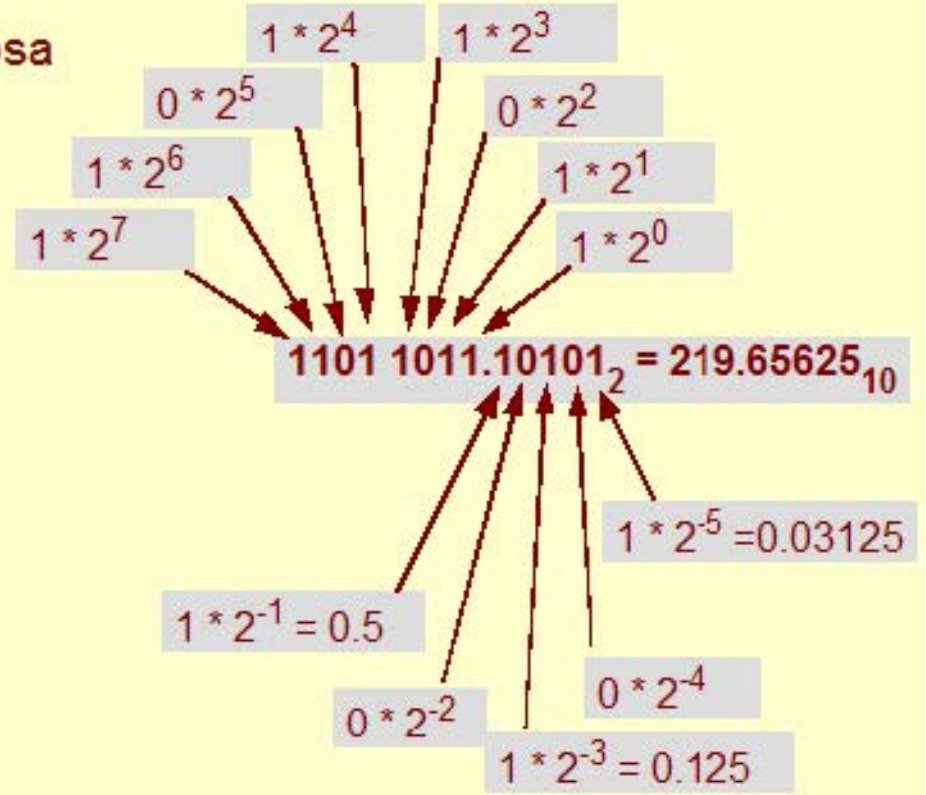


No entäpäs tämä tilanne, jossa äskeisen 8-bittisen esimerkin viimeinen bitti on vaihtunut nollaksi. Mitä desimaalilukua tämä tarkoittaa?

Binääripiste

Binääriluvuilla voi olla myös binääriosa
(vrt. desimaaliluvun desimaaliosa)

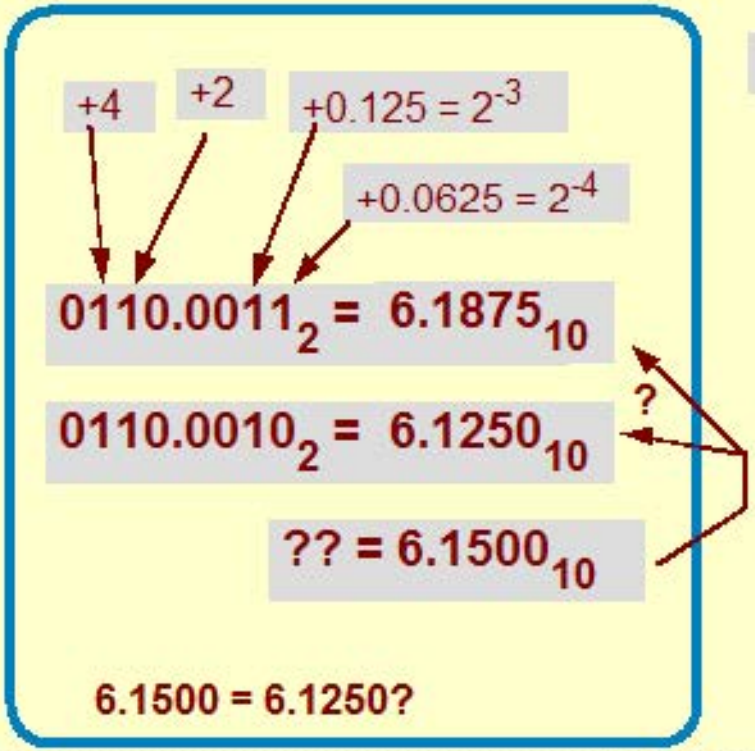


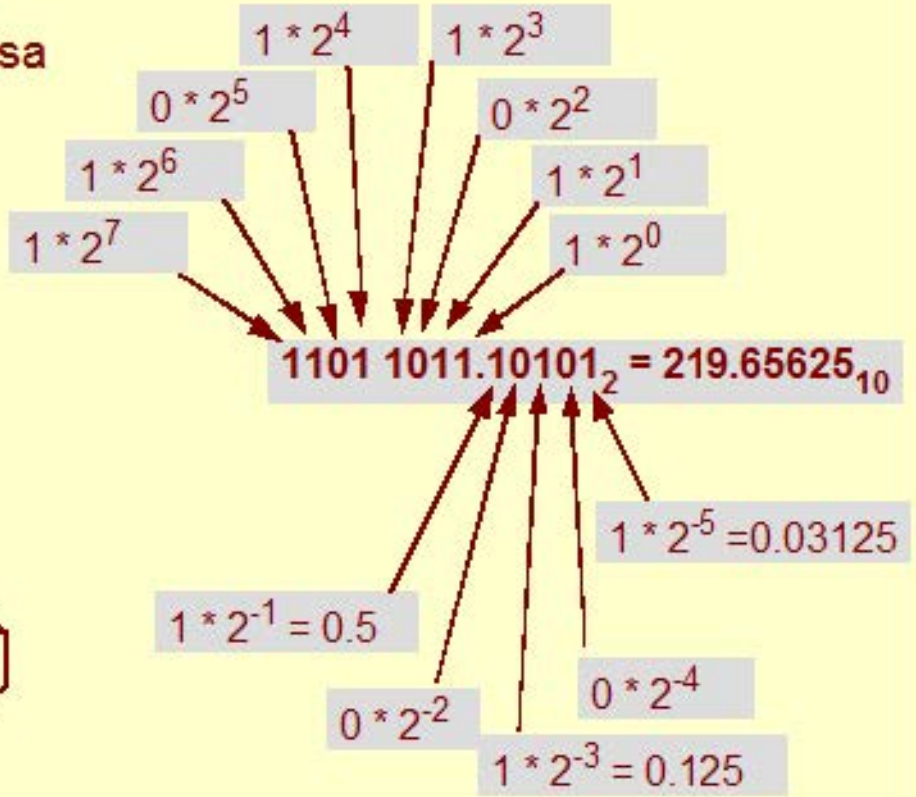


Vastaus on tietenkin 6 ja 1/8. Jos lukujen estystapa olisi tällainen 8-bitin esitys, jossa 4 viimeistä bittiä olisi binääriosaa, niin lukujen esitystarkkuus olisi tämän esimerkin mukainen. Jotkut luvut jäävät inhottavasti esitystapojen välimaastoon. Esimerkiksi, mikä on desimaaliluvun 6.15 esitysmuoto tässä esitystavassa?

Binääripiste

Binääriluvuilla voi olla myös binääriosa
(vrt. desimaaliluvun desimaaliosa)





Kysymys oli siis, että mikä on desimaaliluvun 6.15 esitysmuoto? Tarkkaa esitystä sille ei ole, mutta meidän tulisi valita jompi kumpi tätä lukua lähellä olevista luvuista. Luku 6.15 pitää siis pyöristää joko lukuun 6.1875 tai lukuun 6.1250. Tämä on ihan tyypillinen ongelma laskennassa, jossa usein joudutaan pyöristämään esimerkiksi laskennan lopputulos jollakin tavalla lukujen esitystavan mukaisesti. Jos 6.15 pyöristyy vaikkapa lähimpään esitystavan mukaiseen, eli lukuun 6.1250, niin se tarkoittaa, että tässä esitystavassa luvut 6.15 ja 6.1250 ovat yhtäsuuria!

Muunnokset lukujärjestelmien välillä

2-järjestelmästä 10-järjestelmään

- summataan bittien painoarvot 1-biteille
- käytiin läpi edellä

10-järjestelmästä 2-järjestelmään

- kokonaisosa ja desimaaliosa muunnetaan erikseen

10-järjestelmästä 2-järjestelmään, kokonaisosa

- jaa kokonaisosa toistuvasti 2'lla, kunnes jäljelle jää 0
- ota kustakin jakolaskusta jakojäännökset (0 tai 1) käänteisessä järjestyksessä kokonaisosan biteiksi

10-järjestelmästä 2-järjestelmään, desimaaliosa

- kerro toistuvasti desimaaliosa 2'lla, kunnes joko
  - desimaaliosaksi tulee 0, jolloin saadaan tarkka esitystapa
  - meillä on tarpeeksi bittejä halutun tarkkuuden mukaiseen esitystapaan
- ota kunkin kertolaskun tuloksesta sen kokonaisosat (0 tai 1) lasketussa järjestyksessä binääriosan biteiksi

Kuten edellä havaittiin, muunnokset binäärijärjestelmän ja desimaalijärjestelmän välillä eivät ole kauhian vaikeita, mutta vähän työläitä. Tietokoneet ovat loistavia juuri tällaisten tylsien ja vähän työläiden ongelmien ratkaisuun. Kävimme jo aikaisemmin läpi muunnosalgoritmin 2-järjestelmästä 10-järjestelmään, joten emme puutu tässä siihen sen enempää.

Muunnokset lukujärjestelmien välillä

2-järjestelmästä 10-järjestelmään

- summataan bittien painoarvot 1-biteille
- käytiin läpi edellä

10-järjestelmästä 2-järjestelmään

- kokonaisosa ja desimaaliosa muunnetaan erikseen

10-järjestelmästä 2-järjestelmään, kokonaisosa

- jaa kokonaisosa toistuvasti 2'lla, kunnes jäljelle jää 0
- ota kustakin jakolaskusta jakojäännökset (0 tai 1) käänteisessä järjestyksessä kokonaisosan biteiksi

10-järjestelmästä 2-järjestelmään, desimaaliosa

- kerro toistuvasti desimaaliosa 2'lla, kunnes joko
  - desimaaliosaksi tulee 0, jolloin saadaan tarkka esitystapa
  - meillä on tarpeeksi bittejä halutun tarkkuuden mukaiseen esitystapaan
- ota kunkin kertolaskun tuloksesta sen kokonaisosat (0 tai 1) lasketussa järjestyksessä binääriosan biteiksi

Muunnos desimaalijärjestelmästä binäärijärjestelmään oli selvästi vähän hankalampi tapaus. Kokonaisosa ja desimaaliosa muutetaan nyt erikseen binääriluvun kokonaisosaksi ja binääriosaksi. Esittelemme tässä pääpiirteet nopeasti, ja sitten asian paremmin kuvaavien esimerkkien avulla.

Muunnokset lukujärjestelmien välillä

2-järjestelmästä 10-järjestelmään

- summataan bittien painoarvot 1-biteille
- käytiin läpi edellä

57 / 2 = 28 jakojäännös 1

10-järjestelmästä 2-järjestelmään

- kokonaisosa ja desimaaliosa muunnetaan erikseen

10-järjestelmästä 2-järjestelmään, kokonaisosa

- jaa kokonaisosa toistuvasti 2'lla, kunnes jäljelle jää 0
- ota kustakin jakolaskusta jakojäännökset (0 tai 1) käänteisessä järjestyksessä kokonaisosan biteiksi

10-järjestelmästä 2-järjestelmään, desimaaliosa

- kerro toistuvasti desimaaliosa 2'lla, kunnes joko
  - desimaaliosaksi tulee 0, jolloin saadaan tarkka esitystapa
  - meillä on tarpeeksi bittejä halutun tarkkuuden mukaiseen esitystapaan
- ota kunkin kertolaskun tuloksesta sen kokonaisosat (0 tai 1) lasketussa järjestyksessä binääriosan biteiksi

Kokonaisosan muuntamisen algoritmi perustuu kakkosella jakamiseen ja jakojäännöksen ottamiseen talteen käänteisessä järjestyksessä. Esimerkiksi kokonaisosasta 57 nähdään, että se pariton eli että ensimmäisen 2'lla jakamisen jakojäännös on 1, mistä seuraa, että desimaaliluvun 57 binääriesityksen viimeinen bitti on 1.

Muunnokset lukujärjestelmien välillä

2-järjestelmästä 10-järjestelmään

- summataan bittien painoarvot 1-biteille
- käytiin läpi edellä

2 * 0.1875 = 0.375

10-järjestelmästä 2-järjestelmään

- kokonaisosa ja desimaaliosa muunnetaan erikseen

10-järjestelmästä 2-järjestelmään, kokonaisosa

- jaa kokonaisosa toistuvasti 2'lla, kunnes jäljelle jää 0
- ota kustakin jakolaskusta jakojäännökset (0 tai 1) käänteisessä järjestyksessä kokonaisosan biteiksi

10-järjestelmästä 2-järjestelmään, desimaaliosa

- kerro toistuvasti desimaaliosa 2'lla, kunnes joko
  - desimaaliosaksi tulee 0, jolloin saadaan tarkka esitystapa
  - meillä on tarpeeksi bittejä halutun tarkkuuden mukaiseen esitystapaan
- ota kunkin kertolaskun tuloksesta sen kokonaisosat (0 tai 1) lasketussa järjestyksessä binääriosan biteiksi

Desimaaliosa kerrotaan toistuvasti kakkosella, ja joka kierroksella tulon kokonaisosa otetaan talteen vastaukseen ja tulon desimaaliosa käytetään seuraavalla kierroksella uuden tulon laskentaan. Esimerkiksi, desimaaliosaa 0.1875 kerrottaessa kahdella, tulon 0.375 kokonaisosa 0 ilmaisee, että binääriesityksen binääriosan ensimmäinen bitti on 0. Näinhän pitää ollakin, koska luku 0.1875 on pienempi kuin 0.5. Ainostaan desimaaliosan ollessa vähintään 0.5 tulee binääriosan ensimmäisen bitin olla 1.

Esimerkki: Kokonaisosa 10-järjestelmästä 2-järjestelmään

$57_{10} = ?_2$

Tarkastellaan tässä desimaaliluvun kokonaisosan muuntamista binääriluvuksi esimerkin avulla. Muunnettavana on luku 57. Algoritmihan meni siis siten, että kokonaisosa jaettiin toistuvasti 2'lla kunnes päästiin osamäärään 0, ja sitten kaikki jakojäännökset kerättiin talteen käänteisessä järjestyksessä binääriluvuksi.

Esimerkki: Kokonaisosa 10-järjestelmästä 2-järjestelmään

$57_{10} = ?_2$

57 / 2 = 28, jää 1

Aloitamme algoritmin. 57 jaettuna 2'lla on 28 ja jakojäännös on 1. Unohdamme toistaiseksi jakojäännöksen ja jatkamme kokonaisosalla eli osamäärällä 28.

Esimerkki: Kokonaisosa 10-järjestelmästä 2-järjestelmään

$57_{10} = ?_2$

57 / 2 = 28, jää 1

28 / 2 = 14, jää 0

28 jaettuna 2'lla on 14, jakojäännös on 0.

Esimerkki: Kokonaisosa 10-järjestelmästä 2-järjestelmään

$57_{10} = ?_2$

57 / 2 = 28, jää 1

28 / 2 = 14, jää 0

14 / 2 = 7, jää 0

14 jaettuna 2'lla on 7, jakojäännös on 0.

Esimerkki: Kokonaisosa 10-järjestelmästä 2-järjestelmään

$57_{10} = ?_2$

57 / 2 = 28, jää 1

28 / 2 = 14, jää 0

14 / 2 = 7, jää 0

7 / 2 = 3, jää 1

7 jaettuna 2'lla on 3, jakojäännös on 1.

Esimerkki: Kokonaisosa 10-järjestelmästä 2-järjestelmään

$57_{10} = ?_2$

57 / 2 = 28, jää 1

28 / 2 = 14, jää 0

14 / 2 = 7, jää 0

7 / 2 = 3, jää 1

3 / 2 = 1, jää 1

3 jaettuna 2'lla on 1, jakojäännös on 1.

Esimerkki: Kokonaisosa 10-järjestelmästä 2-järjestelmään

$57_{10} = ?_2$

57 / 2 = 28, jää 1

28 / 2 = 14, jää 0

14 / 2 = 7, jää 0

7 / 2 = 3, jää 1

3 / 2 = 1, jää 1

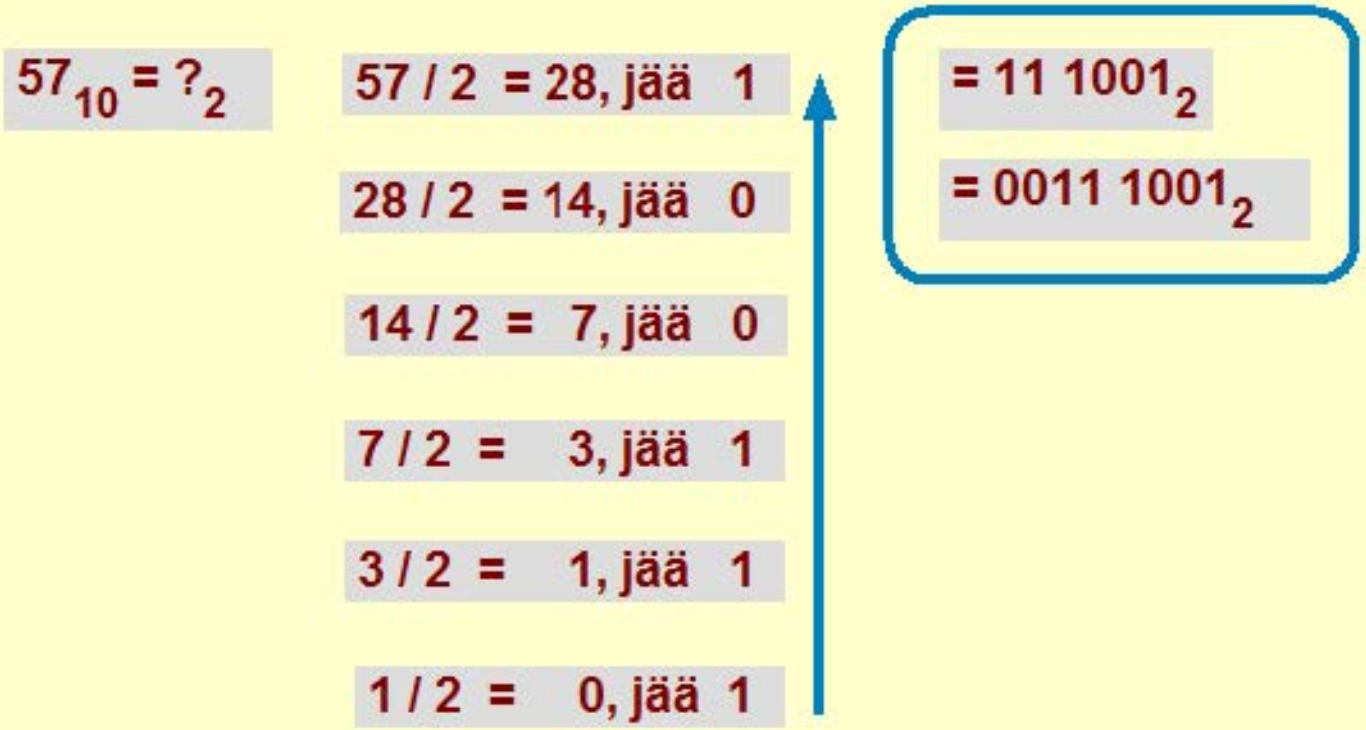
1 / 2 = 0, jää 1

loppu!

$= 11\ 1001_2$

1 jaettuna 2'lla on nolla, jakojäännös on 1. Algoritmi päättyy ja jakojäännösbitit kerätään käänteisessä järjestyksessä binääriluvuksi 111001.

Esimerkki: Kokonaisosa 10-järjestelmästä 2-järjestelmään



Aivan samoin kuin desimaalijärjestelmän kokonaisluvuillekin, myös binäärijärjestelmän kokonaislukujen alkuun voi laittaa 0-bittejä arvon siitä muuttumatta. Meille on aivan yleistä laittaa eteen tarpeeksi nolla-bittejä, jotta bittien kokonaismäärä olisi esimerkiksi 8, 16 tai 32, data-alkion pituudesta riippuen.

Esimerkki: Desimaaliosa 10-järjestelmästä 2-järjestelmään

$0.1875_{10} = ???_{2}$

Tarkastellaan nyt esimerkin avulla desimaaliosan muuntamista binääriosaksi. Muunnettavana on desimaaliosa 0.1875. Algoritmihan meni siten, että desimaaliosa kerrottiin aina kahdella ja tulon kokonaisosan bitit otetaan laskentajärjestyksessä binääriesityksen binääriosan biteiksi. Algoritmi päättyy, kun joko desimaaliosaksi tulee 0, josta se ei enää muuttuisi, tai sitten kun bittejä on laskettu haluttuun tarkkuuteen asti.

Esimerkki: Desimaaliosa 10-järjestelmästä 2-järjestelmään

$0.1875_{10} = ???_{2}$

$2 * 0.1875 = 0.375 = 0 + 0.375$

2 kertaa 0.1875 on 0.375. Kokonaisosa on 0 ja uusi desimaaliosa on 0.375.

Esimerkki: Desimaaliosa 10-järjestelmästä 2-järjestelmään

$0.1875_{10} = ???_{2}$

$2 * 0.1875 = 0.375 = 0 + 0.375$

$2 * 0.375 = 0.75 = 0 + 0.75$

2 kertaa 0.375 on 0.75. Kokonaisosa on 0 ja uusi desimaaliosa on 0.75.

Esimerkki: Desimaaliosa 10-järjestelmästä 2-järjestelmään

$0.1875_{10} = ???_{2}$

$2 * 0.1875 = 0.375 = 0 + 0.375$

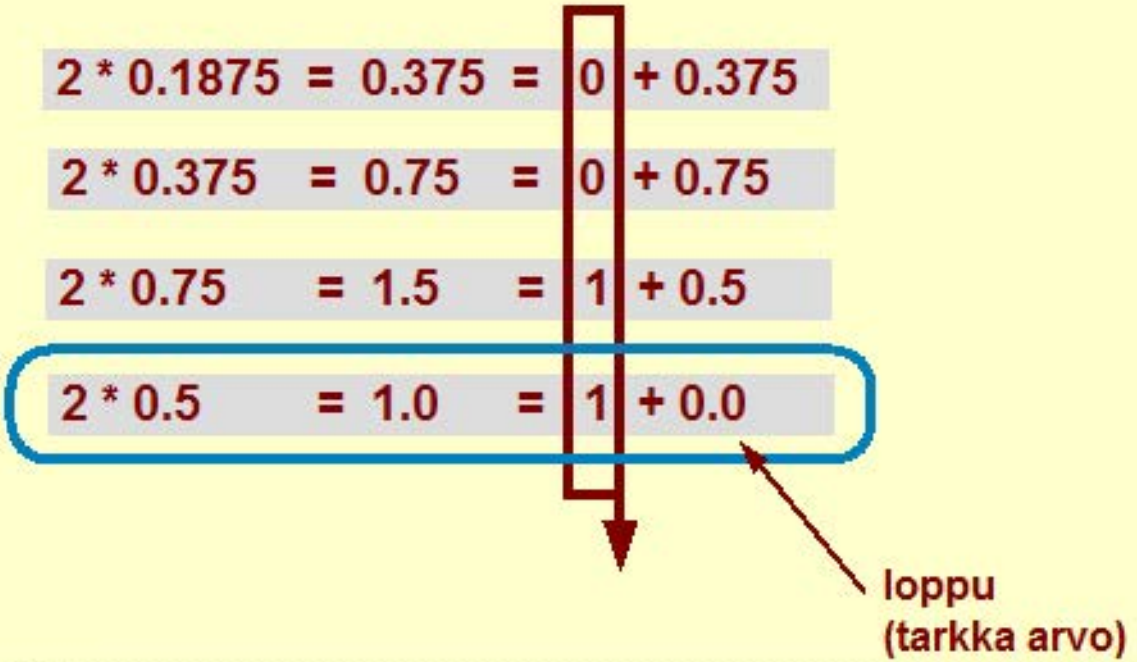
$2 * 0.375 = 0.75 = 0 + 0.75$

$2 * 0.75 = 1.5 = 1 + 0.5$

2 kertaa 0.75 on 1.5. Kokonaisosa on 1 ja uusi desimaaliosa on 0.5.

Esimerkki: Desimaaliosa 10-järjestelmästä 2-järjestelmään

$0.1875_{10} = ???_2$ $= 0.0011_2$



2 kertaa 0.5 on 1.0. Kokonaisosa on 1 ja uusi desimaaliosa on 0.0, joten algoritmi päättyy täsmälliseen arvoon. Vaikka desimaaliosaa kuinka tämän jälkeen kerrottaisiin kahdella, sen arvo pysyy nollana ja laskennassa saadun uuden bitin arvo eli tulon kokonaisosa on aina 0. Desimaaliosat (jotka ovat lukuja 0 tai 1) kerätään talteen binääriesityksen binääriosaksi laskentajärjestyksessä.

Esimerkki: Desimaaliosa 10-järjestelmästä 2-järjestelmään

$0.1875_{10} = ???_2$ $= 0.0011_2$ $= 0.00110000000000000000_2$

$2 * 0.1875 = 0.375 = 0 + 0.375$

$2 * 0.375 = 0.75 = 0 + 0.75$

$2 * 0.75 = 1.5 = 1 + 0.5$

$2 * 0.5 = 1.0 = 1 + 0.0$

Binääriosan loppuun voidaan laittaa 0-bittejä miten paljon tahansa lukuarvon siitä muuttumatta. Näin usein tehdäänkin, jos binääriosa halutaan sijoittaa johonkin vakiomittaiseen kenttään.

Heksadesimaaliesitys

Binäärilukujen käyttö on järkevää, mutta niitä on ikävä kirjoittaa

- liikaa numeroita $0010\ 1011\ 1100\ 1101\ 0001\ 0010\ 0011\ 0100_2$ vai 734859282_{10}

Kirjoitetaan luvut heksadesimaalijärjestelmässä (16-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  10 11 12 13 14 15

$43_{10} = 101011_2 = 2B_{16}$

Neljää bittiä vastaa aina yksi heksadesimaalijärjestelmän numero

- binääribisteen vasemmalla puolella täytä etunollilla tarvittaessa
- binääribisteen oikealla puolella laita nollia loppuun tarvittaessa

Yksi heksadesimaalijärjestelmän numero vastaa aina neljää bittiä

Laitteisto on helppo toteuttaa binäärijärjestelmäm avulla ja laskutoimituksetkin ovat siis yllättävän helppoja 2-järjestelmässä. Binäärijärjestelmässä on kuitenkin yksi heikkous: sillä ilmaistuista luvuista tulee helposti hyvin pitkiä. Pitkiä lukuja taas on vaikea lukea tai kirjoittaa, ja niiden kanssa tulee ihmisille helposti virheitä. Tietokoneellehan tästä pituudesta ei ole mitään haittaa, koska tietokone ei tee mitään virheitä, vahingossakaan.

Heksadesimaaliesitys

Binäärilukujen käyttö on järkevää, mutta niitä on ikävä kirjoittaa

- liikaa numeroita

Kirjoitetaan luvut heksadesimaalijärjestelmässä (16-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  10 11 12 13 14 15

$43_{10} = 101011_2 = 2B_{16}$

Neljää bittiä vastaa aina yksi heksadesimaalijärjestelmän numero

- binääribisteen vasemmalla puolella täytä etunollilla tarvittaessa
- binääribisteen oikealla puolella laita nollia loppuun tarvittaessa

Yksi heksadesimaalijärjestelmän numero vastaa aina neljää bittiä

Käytännössä kirjoitamme binääriluvut lähes aina 16- eli heksadesimaalijärjestelmän avulla. Heksadesimaalijärjestelmän alkuperäinen puhtaasti latinaan perustuva nimi oli 'sexadecimal', mutta IBM'n puristit muuttivat sen 1950-luvulla nykyiseen muotoonsa 'hexadecimal'. IBM'ltä termin käyttö levisi sitten myös muualle. 16-järjestelmässä on siis 16 numeroa, numeroarvoltaan 0-15. Ensimmäiset 10 numeroa ovat tuttuja ja viimeiseksi 6 numeroksi on valittu kirjaimet A, B, C, D, E ja F, jotka vastaavat numeroarvoja 10-15. Näin joka numerolla on yhden merkin esitystapa, käyttäen ennestään tuttuja merkkejä.

Heksadesimaaliesitys

Binäärilukujen käyttö on järkevää, mutta niitä on ikävä kirjoittaa

- liikaa numeroita

Kirjoitetaan luvut heksadesimaalijärjestelmässä (16-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  10 11 12 13 14 15

$43_{10} = 101011_2 = 2B_{16}$

Neljää bittiä vastaa aina yksi heksadesimaalijärjestelmän numero

- binääribisteen vasemmalla puolella täytä etunollilla tarvittaessa
- binääribisteen oikealla puolella laita nollia loppuun tarvittaessa

Yksi heksadesimaalijärjestelmän numero vastaa aina neljää bittiä

$0110_2 = 6_{16}$

$1010_2 = A_{16}$

$1010.10001011_2 = A.8B_{16} = 0xA.8$

$11.01001_2 = 0011.01001000_2 = 3.48_{16} = 003.48000_{16}$

Jokainen neljän bitin ryhmä vastaa nyt täsmälleen yhtä heksadesimaalijärjestelmän numeroa. Bitit ryhmitellään aina neljän bitin ryhmiin, binääribitistä lähtien vasemmalle ja oikealle. Jos josssakin ryhmässä on vähemmän kuin neljä bittiä, niin kuvitellaan ne sinne, joko alku- tai loppunolliksi.

Heksadesimaaliesitys

Binäärilukujen käyttö on järkevää, mutta niitä on ikävä kirjoittaa

- liikaa numeroita

Kirjoitetaan luvut heksadesimaalijärjestelmässä (16-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  10 11 12 13 14 15

$43_{10} = 101011_2 = 2B_{16}$

Neljää bittiä vastaa aina yksi heksadesimaalijärjestelmän numero

- binääribisteen vasemmalla puolella täytä etunollilla tarvittaessa
- binääribisteen oikealla puolella laita nollia loppuun tarvittaessa

Yksi heksadesimaalijärjestelmän numero vastaa aina neljää bittiä

$4_{16} = 0100_2$ $F_{16} = 1111_2$ $62.4_{16} = 0110\ 0010.0100_2 = 110\ 0010.01_2$

Jokainen heksadesimaalijärjestelmän numero vastaa täsmälleen neljää bittiä. Muunnos on suoraviivainen ja se tehdään merkki kerrallaan. Ylimääräiset nollat voi poistaa lopputuloksesta, joko alkunollat ennen binääribittiä tai loppunollat binääribitin oikealla puolella.

Heksadesimaaliesitys

Binäärilukujen käyttö on järkevää, mutta niitä on ikävä kirjoittaa

- liikaa numeroita

Kirjoitetaan luvut heksadesimaalijärjestelmässä (16-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  10 11 12 13 14 15

$43_{10} = 101011_2 = 2B_{16}$

Neljää bittiä vastaa aina yksi heksadesimaalijärjestelmän numero

- binääribisteen vasemmalla puolella täytä etunollilla tarvittaessa
- binääribisteen oikealla puolella laita nollia loppuun tarvittaessa

Yksi heksadesimaalijärjestelmän numero vastaa aina neljää bittiä

$100\ 0111\ 1001.1010\ 1111_2 = ?_{16}$

Harjoitellaan tätäkin. Mitäs tämä on heksana?

Heksadesimaaliesitys

Binäärilukujen käyttö on järkevää, mutta niitä on ikävä kirjoittaa

- liikaa numeroita

Kirjoitetaan luvut heksadesimaalijärjestelmässä (16-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  10 11 12 13 14 15

$43_{10} = 101011_2 = 2B_{16}$

Neljää bittiä vastaa aina yksi heksadesimaalijärjestelmän numero

- binääribisteen vasemmalla puolella täytä etunollilla tarvittaessa
- binääribisteen oikealla puolella laita nollia loppuun tarvittaessa

Yksi heksadesimaalijärjestelmän numero vastaa aina neljää bittiä

$100\ 0111\ 1001.1010\ 1111_2 = 479.AF_{16} = 00479.AF00_{16} = 0x479.AF$

4 7 9 A F

Bitit ryhmitellään neljän ryhmiin ja ensimmäiseen ryhmään laitetaan sieltä 'puuttuva' 0-bitti. Ryhmät muutetaan yksi kerrallaan heksadesimaalijärjestelmän numeroiksi. Esitystapaan voi tarvittaessa laittaa alku- tai loppunollia. Kirjoitetussa muodossa voi myös käyttää etuliitettä 0x alaindeksin 16 asemesta, mikä on peräisin kirjoituskoneajalta, jolloin alaindeksien käyttö oli usein hankalaa tai mahdotonta.

Heksadesimaaliesitys

Binäärilukujen käyttö on järkevää, mutta niitä on ikävä kirjoittaa

- liikaa numeroita

Kirjoitetaan luvut heksadesimaalijärjestelmässä (16-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  10 11 12 13 14 15

$43_{10} = 101011_2 = 2B_{16}$

Neljää bittiä vastaa aina yksi heksadesimaalijärjestelmän numero

- binääribisteen vasemmalla puolella täytä etunollilla tarvittaessa
- binääribisteen oikealla puolella laita nollia loppuun tarvittaessa

Yksi heksadesimaalijärjestelmän numero vastaa aina neljää bittiä

$12.0ADF_{16} = ?_2$

No entäs mitä tämä on binäärinä?

Heksadesimaaliesitys

Binäärilukujen käyttö on järkevää, mutta niitä on ikävä kirjoittaa

- liikaa numeroita

Kirjoitetaan luvut heksadesimaalijärjestelmässä (16-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  10 11 12 13 14 15

$43_{10} = 101011_2 = 2B_{16}$

Neljää bittiä vastaa aina yksi heksadesimaalijärjestelmän numero

- binääribisteen vasemmalla puolella täytä etunollilla tarvittaessa
- binääribisteen oikealla puolella laita nollia loppuun tarvittaessa

Yksi heksadesimaalijärjestelmän numero vastaa aina neljää bittiä

$12.0ADF_{16} = 1\ \ 2\ \ 0\ \ A\ \ D\ \ F_{16} = 0001\ 0010.0000\ 1010\ 1101\ 1111_2$

$0001\ 0010\ 0000\ 1010\ 1101\ 1111 = 1\ 0010.0000\ 1010\ 1101\ 1111_2$

Jokainen heksadesimaalinumero muutetaan neljän bitin rykelmäksi. Tulosta voi halutessa siistiä poistamalla turhat etu- tai loppunollat.

Oktaaliesitys

Aikaisemmin käytössä ollut esitysmuoto

$43_{10} = 101011_2 = 53_8 = 053$

- sopii hyvin 6-bittisiin tavuihin

Kirjoitetaan luvut oktaalijärjestelmässä (8-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7

Yksi oktaalijärjestelmän numero vastaa aina kolmea bittiä

Joissakin varhaisemmissa koneissa käytettiin 6-bittisiä tavuja, ja niiden sisällön kuvaamiseen on oktaalijärjestelmä sopivin. 6-bittinen tavu jaetaan 3 bitin ryhmiin, ja jokainen 3-bittinen ryhmä esitetään yhdellä oktaalijärjestelmän numerolla. Muutenhan ajatus on ihan vastaava kuin 16-järjestelmässäkin. Alaindeksin 8 asemesta oktaalijärjestelmää merkitään joskus kirjallisuudessa alkunollalla samaan tapaan kuin heksadesimaalijärjestelmää merkittiin alussa olevalla 0x'llä. Tämä esitystapa pitää vain tietää, jotta sen voisi tunnistaa vanhemmissa artikkeleissa.

Oktaaliesitys

Aikaisemmin käytössä ollut esitysmuoto

$43_{10} = 101011_2 = 53_8 = 053$

- sopii hyvin 6-bittisiin tavuihin

Kirjoitetaan luvut oktaalijärjestelmässä (8-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7

Yksi oktaalijärjestelmän numero vastaa aina kolmea bittiä

$100\ 0111\ 1001.1010\ 1111_2 = ?_8$

Harjoitellaan tätäkin vähän. Mitenkäs tämä muutetaan oktaaliesitykseen?

Oktaaliesitys

Aikaisemmin käytössä ollut esitysmuoto

$43_{10} = 101011_2 = 53_8 = 053$

- sopii hyvin 6-bittisiin tavuihin

Kirjoitetaan luvut oktaalijärjestelmässä (8-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7

Yksi oktaalijärjestelmän numero vastaa aina kolmea bittiä

$100\ 0111\ 1001.1010\ 1111_2 = ?_8$

$010\ 001\ 111\ 001.101\ 011\ 110_2$

Ensin tietenkin ryhmitellään bitit 3 bitin ryhmiin, binääripisteestä lähtien sekä oikealle että vasemmalle. Alkuun ja loppuun laitetaan nollia tarvittaessa.

Oktaaliesitys

Aikaisemmin käytössä ollut esitysmuoto

$43_{10} = 101011_2 = 53_8 = 053$

- sopii hyvin 6-bittisiin tavuihin

Kirjoitetaan luvut oktaalijärjestelmässä (8-järjestelmässä)

- numerot: 0, 1, 2, 3, 4, 5, 6, 7

Yksi oktaalijärjestelmän numero vastaa aina kolmea bittiä

$100\ 0111\ 1001.1010\ 1111_2 = 2171.536_8 = 02171.536$

$010\ 001\ 111\ 001.101\ 011\ 110_2$

2 1 7 1 5 3 6

Kolmen bitin ryhmät muutetaan sitten yksi kerrallaan oktaalinumeroiksi ja siinähän se vastaus sitten kaikessa yksinkertaisuudessaan on. Muunnokset oktaalijärjestelmästä binäärjärjestelmään tapahtuvat vastaavasti yksi oktaalijärjestelmän numero kerrallaan.

Big vs. Little Endian

Miten monitavuinen tieto talletetaan?

nelitavuisen sanan osoite → 0x1200:

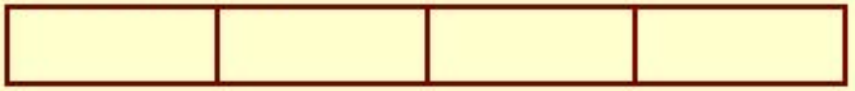
yksittäisten tavujen osoitteet → 0x1200 0x1201 0x1202 0x1203

nelitavuinen arvo → **0x11223344**

Big Endian

- eniten merkitsevä tavu pienimpään tavuosoitteeseen
- ttk-91 käyttää tätä

0x1200:



0x1200 0x1201 0x1202 0x1203

0x1200:

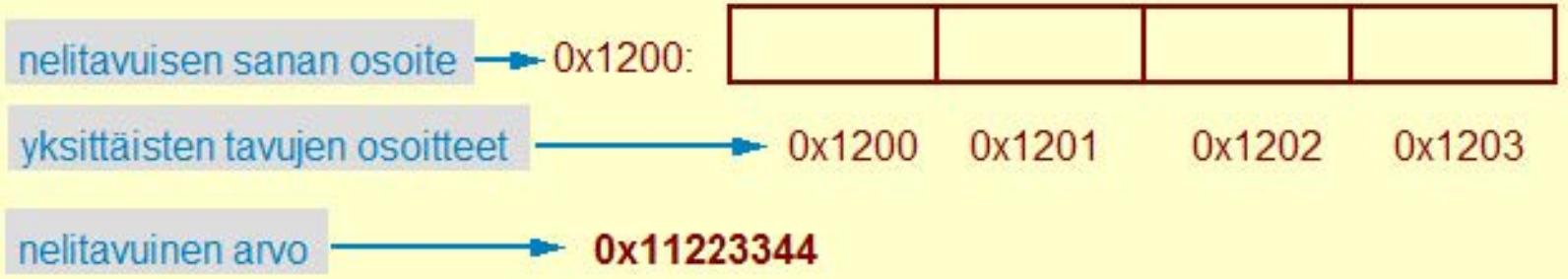
Little Endian

- vähiten merkitsevä tavu pienimpään tavuosoitteeseen

Monitavuinen tieto talletetaan yleensä peräkkäisiin tavuihin. Tässä on tietenkin kaksi vaihtoehtoista talletustapaa, ja molemmilla tavoilla on omat etunsa ja heikkoutensa. Esimerkkinä meillä on nelitavuinen sana 0x11223344, jossa siis eniten merkitsevän tavun arvo on 0x11 ja vähiten merkitsevän tavun arvo on 0x44.

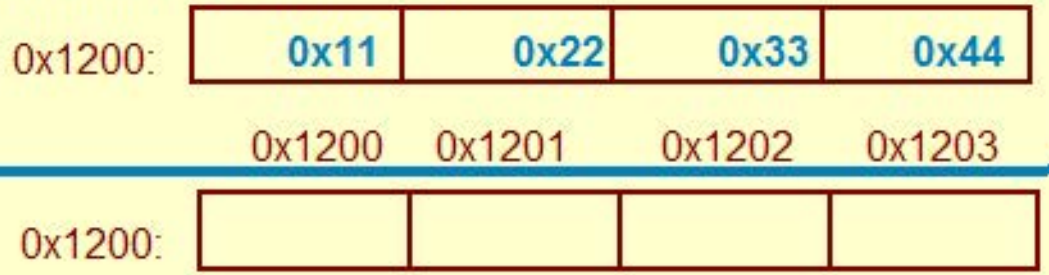
Big vs. Little Endian

Miten monitavuinen tieto talletetaan?



Big Endian

- eniten merkitsevä tavu pienimpään tavuosoitteeseen
- ttk-91 käyttää tätä



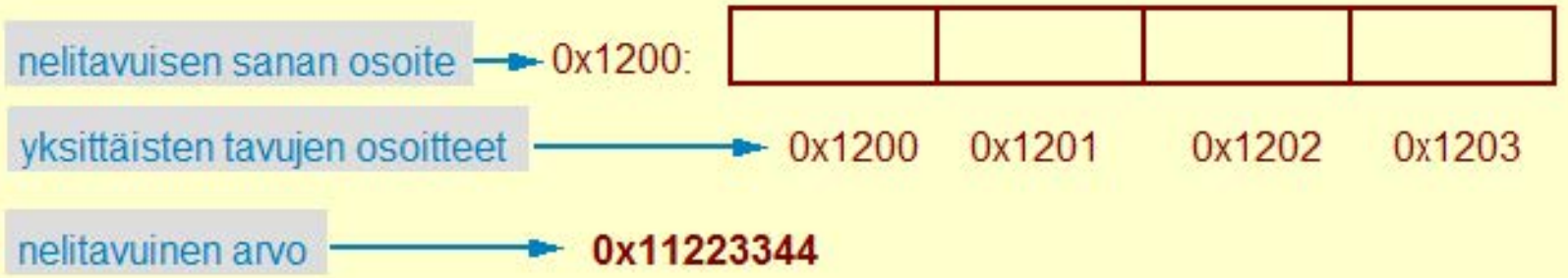
Little Endian

- vähiten merkitsevä tavu pienimpään tavuosoitteeseen

Big Endian (eli 'Big End first') talletus muodossa eniten merkitsevä tavu on ensin eli pienimmässä tavuosoitteessa. Tämä on luonnollinen talletusmuotc, koska lukujen esitystapa näyttää samalta kuin mitä olemme tottuneet kirjoittamaan jo koulussa. Kymmenjärjestelmän luvutkin kirjoitetaan siten, että eniten merkitsevä numero on ensimmäisenä.

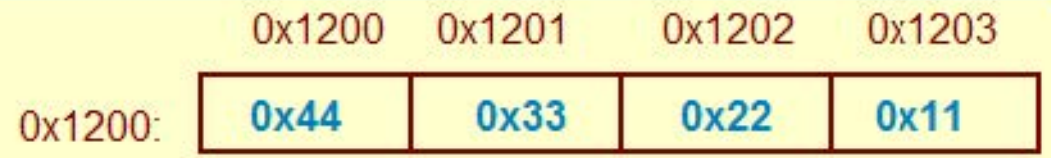
Big vs. Little Endian

Miten monitavuinen tieto talletetaan?



Big Endian

- eniten merkitsevä tavu pienimpään tavuosoitteeseen
- ttk-91 käyttää tätä

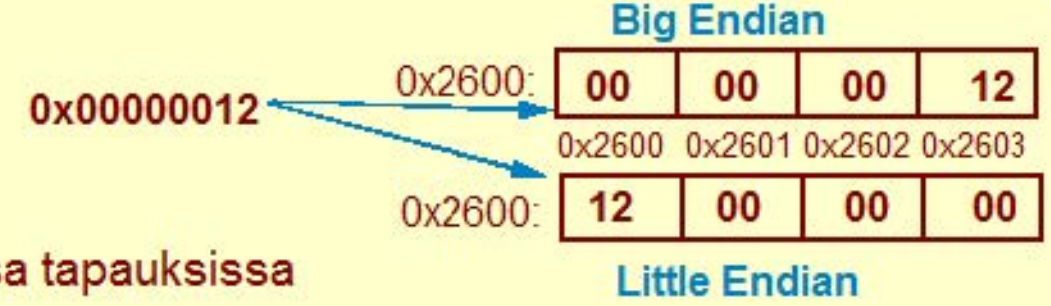


Little Endian

- vähiten merkitsevä tavu pienimpään tavuosoitteeseen

Little Endian talletuksessa tavut ovat päin vastaisessa järjestyksessä. Little Endian'ia käyttäen esitystavan pituuden muutokset on hieman helpompi toteuttaa, koska esimerkiksi 4-tavuisen sanan vähiten merktsevällä tavulla on sama osoite kuin alkuperäisellä 4-tavuisella sanalla.

Big vs. Little Endian



Monitavuisen tiedon
(esim. 4-tavuinen sana)
osoite on sama molemmissa tapauksissa

- tavujen järjestys on erilainen
- vähiten merkitsevän tavun osoite on Little Endian'issa sama kuin sanan osoite

Suorittimen suunnittelija päättää

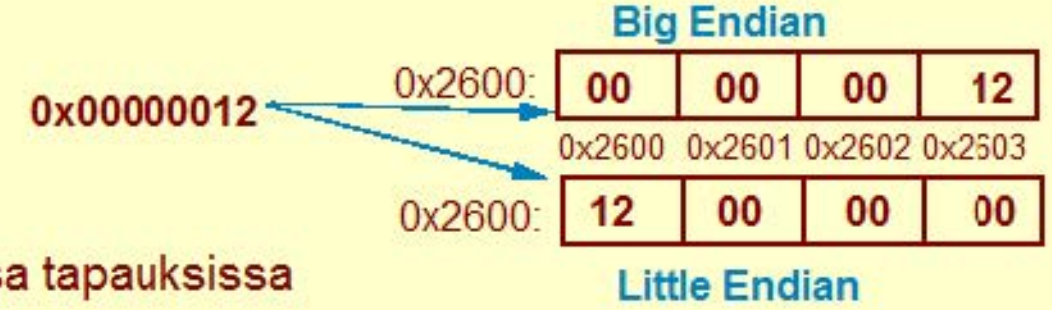
- matematiikkapiirien tulee tietää, miten luvut esitetty
- tulee ottaa huomioon, kun tietoa siirretään verkon yli

Power-PC on bi-endian - molemmat moodit käytössä

- voidaan valita ohjelmakohtaisesti ohjelman lataushetkellä
- etuoikeutetussa tilassa valinta vielä erikseen
- suoritin osaa laskea yhdellä tavalla, tavut käännetään 'lennossa' tarvittaessa.

Molemmissa esitystavoissa monitavuisen tiedon osoite on siis sama, mutta yksittäisten tavujen sijainti vaihtelee esitystavasta riippuen. Little Endian'ia käyttäen esimerkiksi pienten 4-tavuisten arvojen muuttaminen yhden tai kahden tavun mittaiseksi on helppoa, koska tieto pitää vain tallettaa nelitavuiseen sanaan ja sitten lukea samasta osoitteesta yhden tai kahden tavun tietona. Big Endian'ia käytettäessä esimerkin tavu 0x12 osoite on erilainen kuin sanan 0x12 osoite.

Big vs. Little Endian



Monitavuisen tiedon
(esim. 4-tavuinen sana)
osoite on sama molemmissa tapauksissa

- tavujen järjestys on erilainen
- vähiten merkitsevän tavun osoite on Little Endian'issa sama kuin sanan osoite

Suorittimen suunnittelija päättää

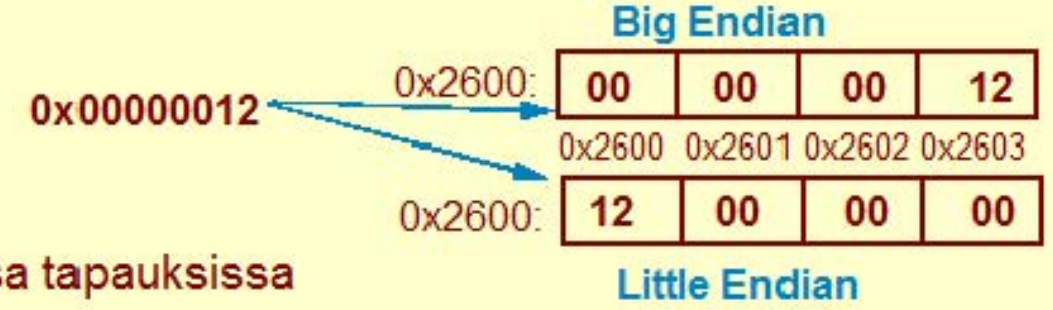
- matematiikkapiirien tulee tietää, miten luvut esitetty
- tulee ottaa huomioon, kun tietoa siirretään verkon yli

Power-PC on bi-endian - molemmat moodit käytössä

- voidaan valita ohjelmakohtaisesti ohjelman lataushetkellä
- etuoikeutetussa tilassa valinta vielä erikseen
- suoritin osaa laskea yhdellä tavalla, tavut käännetään 'lennossa' tarvittaessa.

Käytännössä asialla ei ole paljoakaan väliä ja suorittimen suunnittelija tekee tavujärjestyksestä jonkin sortin päätöksen, jonka mukaan sitten laskentapiirit toteutetaan. Molemmat tavat ovat kuitenkin käytössä, ja tämä on yksi niistä useasta seikasta, jotka pitää ottaa huomioon siirrettäessä tietoa verkon yli laitteistosta toiseen. Verkonhallintaohjelmistot kääntävät tavujärjestyksen 'lennossa' aina tarvittaessa.

Big vs. Little Endian



Monitavuisen tiedon
(esim. 4-tavuinen sana)
osoite on sama molemmissa tapauksissa

- tavujen järjestys on erilainen
- vähiten merkitsevän tavun osoite on Little Endian'issa sama kuin sanan osoite

Suorittimen suunnittelija päättää

- matematiikkapiirien tulee tietää, miten luvut esitetty
- tulee ottaa huomioon, kun tietoa siirretään verkon yli

Power-PC on bi-endian - molemmat moodit käytössä

- voidaan valita ohjelmakohtaisesti ohjelman lataushetkellä
- etuoikeutetussa tilassa valinta vielä erikseen
- suoritin osaa laskea yhdellä tavalla, tavut käännetään 'lennossa' tarvittaessa.

IBM'n Power-PC'ssä on molemmat moodit käytettävissä. Jokainen ohjelma voi itse valita kumpaa esitystapaa siinä käytetään ja suoritin osaa tietenkin sitten laskea molemmilla tavoilla. Lisäksi käyttöjärjestelmä voi vielä erikseen päättää, mitä esitystapaa etuoikeutetussa suoritustilassa käytetään. Itse laskentapiirit tehdään tietenkin vain yhdellä tavalla, mutta niitä edeltää sitten jonkin sortin tavujen kääntämissysteemi, joka aktivoidaan tarvittaessa.

Negatiiviset kokonaisluvut

+57 = **0**011 1001
luku talletusmuoto

Etumerkki bitti erikseen

-57 = **1**011 1001

Yhden komplementti

-57 = **1**100 0110

+1

Kahden komplementti

-57 = **1**100 0111

Vakiolisäys

- yleensä eniten merkitsevän bitin lukuarvo -1
- 8 bittiä: vakiolisäys on 127 = 2^7 - 1

$127 = 0x7F = 0111\ 1111_2$

-57
+127
=70

-57 = **0**100 0110

+57
+127
=184

+57= **1**011 1000

Kokonaislukujen esitystapa positiivisille luvuille on triviaali ja lähes aina vakio, mutta ei aina! Perusidea on kuitenkin, että yksi bitti pitää muodossa tai toisessa jättää etumerkkiä varten. Luvun itseisarvolle on käytettävissä yksi bitti vähemmän kuin mitä bittejä on yhteensä käytössä, ellei sitten käytetä etumerkittömiä, aina positiivisia kokonaislukuja. Yksinkertaisin tapa koodata etumerkki on varata ensimmäinen bitti ihan sitä varten ja koodata etumerkki sitten jollain tavoin siihen. Yleisesti käytetty menetelmä on koodata plus nollalla ja miinus ykkösellä, jolloin positiivisten lukujen esitystapa on normaali binääriesitys.

Negatiiviset kokonaisluvut

+57 = **0**011 1001
luku talletusmuoto

Etumerkki bitti erikseen

-57 = **1**011 1001

Yhden komplementti

-57 = **1**100 0110

+1

Kahden komplementti

-57 = **1**100 0111

Vakiolisäys

- yleensä eniten merkitsevän bitin lukuarvo -1
- 8 bittiä: vakiolisäys on 127 = 2^7 - 1

$127 = 0x7F = 0111\ 1111_2$

-57
+127
=70

-57 = **0**100 0110

+57
+127
=184

+57= **1**011 1000

Toinen yksinkertainen tapa koodata negatiiviset luvut on ottaa positiivinen luku ja komplementoida kaikki bitit. Tässäkin esitystavassa vasemmanpuoleinen bitti on nyt etumerkkibitti, mutta luvun itseisarvo ei ole enää helposti ihmisen luettavissa. Negaatio-operaatio on kuitenkin hyvin helppo tehdä, kun aina kaikki bitit vain komplementoidaan.

Negatiiviset kokonaisluvut

+57 = **0**011 1001
luku talletusmuoto

Etumerkki bitti erikseen

-57 = **1**011 1001

Yhden komplementti

-57 = **1**100 0110

+1

Kahden komplementti

-57 = **1**100 0111

Vakiolisäys

- yleensä eniten merkitsevän bitin lukuarvo -1
- 8 bittiä: vakiolisäys on 127 = 2^7 - 1

$127 = 0x7F = 0111\ 1111_2$

-57
+127
=70 → -57 = **0**100 0110

+57
+127
=184 → +57= **1**011 1000

Yleisin käytössä oleva negatiivisten kokonaislukujer esitysmuoto on kahden komplementti. Siinä otetaan ensin komplementti kaikista biteistä ja tulokseen lisätään yksi. Vasemmanpuoleinen bitti on jälleen etumerkkibitti. Tälle esitystavalle on yksinkertaisempi tehdä matematiikkapiirejä laskentaa varten ja sen vuoksi se on yleisin tavallisten negatiivisten kokonaislukujen talletusmuoto.

Negatiiviset kokonaisluvut

+57 = **0**011 1001
luku talletusmuoto

Etumerkki bitti erikseen

-57 = **1**011 1001

Yhden komplementti

-57 = **1**100 0110

+1

Kahden komplementti

-57 = **1**100 0111

Vakiolisäys

- yleensä eniten merkitsevän bitin lukuarvo -1
- 8 bittiä: vakiolisäys on 127 = 2^7 - 1

$127 = 0x7F = 0111\ 1111_2$

-57
+127
=70 → -57 = **0**100 0110

+57
+127
=184 → +57= **1**011 1000

Neljäs käytössä oleva negatiivisten lukujen esitysmuoto on vakiolisäys, jossa kaikki luvut muutetaan ensin positiivisiksi luvuiksi lisäämällä niihin jokin sovittu vakio. Talletusmuoto on siis aina etumerkitön positiivinen kokonaisluku, josta talletettu luku saadaan vähentämällä esitysmuodosta tunnettu vakio. Yleensä käytetty vakio se sellainen, että sen binääriesityksessä kaikki muut paitsi vasemmanpuoleinen bitti ovat ykkösiä. Tässäkin esityksessä vasemmanpuoleinen bitti on etumerkkibitti, mutta tällä kertaa bitti 1 indikoi plus-merkkiä. Myös positiivisten lukujen esitystapa poikkeaa siis tavallisesta binääriesityksestä.

Kahden komplementti

+57 = **0**011 1001
luku talletusmuoto

kahden komplementti

+127 = 0111 1111
...
1 = 0000 0001
0 = 0000 0000
-1 = 1111 1111
...
-128 = 1000 0000

Useimmiten käytössä tavallisille kokonaisluvuille

Vain yksi nolla

- yhden komplementissa +/- nolla
  (hankaloittaa piirejä ja vertailuja)

+0 = 0000 0000
-0 = 1111 1111
yhden komplementti

Helpot muunnokset arvon ja esitysmuodon välillä

- sama algoritmi (piiri) molempiin suuntiin!
  (komplementoi ja lisää yksi)

0011 1001	+57
1100 0110	komplementoi bitit
+1	lisää yksi
1100 0111	

- mikä on luvun -57 esitysmuoto?

1100 0111	esitysmuoto, neg luku
0011 1000	komplementoi bitit
+1	lisää yksi
0011 1001	itseisarvo 57
	arvo -57

- mikä on esitysmuodon 1100 0111 tarkoittama luku?

Kahden komplementti on useimmiten käytetty esitystapa tavallisille kokonaisluvuille. Merkittävänä etuna yhden komplementtiin verrattuna on se, että esitystavassa on vain yksi esitystapa nollalle. Yhden komplementissahan on sekä plus että miinus nolla, mikä vähän hankaloittaa piirien suunnittelua ja myös tekee vertailut nollaan monimutkaiseksi, jos aina pitää vertailla erikseen plus ja miinus nollaan. On helpompi siis käyttää kahden komplementtia, jossa koko ongelma häviää. Lukualue kahden komplementtia käytettäessä ei ole symmetrinen, vaan pienimmän negatiivisen luvun itseisarvo on yhtä suurempi kuin suurin positiivinen luku.

Kahden komplementti

+57 = **0**011 1001
luku talletusmuoto

kahden komplementti

+127 = 0111 1111
...
1 = 0000 0001
0 = 0000 0000
-1 = 1111 1111
...
-128 = 1000 0000

Useimmiten käytössä tavallisille kokonaisluvuille

Vain yksi nolla

- yhden komplementissa +/- nolla
  (hankaloittaa piirejä ja vertailuja)

Helpot muunnokset arvon ja esitysmuodon välillä

- sama algoritmi (piiri) molempiin suuntiin!
  (komplementoi ja lisää yksi)

0011 1001	+57
1100 0110	komplementoi bitit
+1	lisää yksi
1100 0111	

- mikä on luvun -57 esitysmuoto?

- mikä on esitysmuodon 1100 0111 tarkoittama luku?

1100 0111	esitysmuoto, neg luku
0011 1000	komplementoi bitit
+1	lisää yksi
0011 1001	itseisarvo 57
	arvo -57

Kahden komplementin esitysmuodon yksi heikkous on vähän monimutkainen muunnos arvon ja esitystavan välillä. Muunnos on loppujen lopuksi aika yksinkertainen ja ehkä yllättäen liki samanlainen kumpaankin suuntaan. Muunnospiirejä tarvitaan siten vain yksi kappale, joka toimii sitten molempiin suuntiin. Esimerkeistä on jätetty yksityiskohtia pois esimerkiksi ylivuotobitin käsittelyn suhteen.

Liukuluvut

Tietokoneessa ei ole realilukuja tai rationaalilukuja

- tietokone ei laske samalla tavalla kuin matematiikassa yleensä ajatellaan
- tietokoneelle on oma liukulukumatemaatiikka

usein: 1.0 + 0.000000001 = 1.0

Aina on rajallinen esityksen tarkkuus

- lukuja Pii, SQRT(2) tai 1/3 ei voi esittää tarkalleen
- käytetään likiarvoja
- 32-bittisessä esitystavassa 1.000000000 = 1.000000001

Yleinen realilukuja vastaava esitysmuoto on liukulukuesitysmuoto

- 32 bittiä, noin 7-8 desimaalinumeron tarkkuus — float, real
- 64 bittiä, noin 16-17 desimaalinumeron tarkkuus — double, double real

On ehkä yllättävää, että tietokone ei laske samalla aritmetiikalla kuin mitä opimme koulussa. Tämä aiheutuu siitä, että puhdas matematiikka on abstraktio, mutta tietokone on vain yksinkertainen kone, jolla pitää olla niin sanotusti 'jalat maan pinnalla'.

Liukuluvut

Tietokoneessa ei ole realilukuja tai rationaalilukuja

- tietokone ei laske samalla tavalla kuin matematiikassa yleensä ajatellaan
- tietokoneelle on oma liukulukumatemaatiikka

Aina on rajallinen esityksen tarkkuus

- lukuja Pii, SQRT(2) tai 1/3 ei voi esittää tarkalleen
- käytetään likiarvoja
- 32-bittisessä esitystavassa 1.000000000 = 1.000000001

Pii = 3.1415927

1/3 = 0.33333333

3 * 1/3 = 0.99999999

Yleinen realilukuja vastaava esitysmuoto on liukulukuesitysmuoto

- 32 bittiä, noin 7-8 desimaalinumeron tarkkuus — float, real
- 64 bittiä, noin 16-17 desimaalinumeron tarkkuus — double, double real

Tietokoneessa kaikki luvut pitää aina tallettaa rajalliseen kenttään, esimerkiksi 32 bitin sanaan. Tästä seuraa, että lukujen esityksellä on aina jokin rajallinen tarkkuus, mikä ei oikein sovi yhteen abstraktin matematiikan kanssa. Esimerkiksi luku pii pitää aina esittää likiarvona. Toisaalta kannattaa pitää mielessä vanha insinöörivitsi, jossa kaksi henkilöä lähestyy toisiaan siten, että heidän välinen etäisyys puoliintuu aina minuutin välein. Puhtaan matematiikon mielestä he eivät koskaan kohtaa, mutta insinööriopiskelijan mielestä hyvin pian ollaan riittävän lähellä käytännön sovelluksiin.

Liukuluvut

Tietokoneessa ei ole realilukuja tai rationaalilukuja

- tietokone ei laske samalla tavalla kuin matematiikassa yleensä ajatellaan
- tietokoneelle on oma liukulukumatemaatiikka

Aina on rajallinen esityksen tarkkuus

- lukuja Pii, SQRT(2) tai 1/3 ei voi esittää tarkalleen
- käytetään likiarvoja
- 32-bittisessä esitystavassa 1.000000000 = 1.000000001

Yleinen realilukuja vastaava esitysmuoto on liukulukuesitysmuoto

- 32 bittiä, noin 7-8 desimaalinumeron tarkkuus — float, real
- 64 bittiä, noin 16-17 desimaalinumeron tarkkuus — double, double real

Matemaattisesti määriteltyjä realilukuja vastaa tietokoneessa oma lukutyyppinsä liukuluku, jonka ominaispiirteenä on rajallinen tiedon tarkkuus. Yleisesti käytössä olevan IEEE'n standardin mukaisissa 32-bittisissä liukuluvuissa on sitten noin 7 tai 8 desimaalinumeron tarkkuus, mikä ei siis vielä ole kovin paljon. Sillä tarkkuudella tulee jo aika paljon pyöristysvirheitä ja laskennan jatkuessa ne kasaantuvat ja voivat aiheuttaa suurenkin virheen. Esimerkiksi Kuwaitin sodan aikana liittoutuman ohjustentorjuntaohjukset menivät harhaan tällaisen pyöristysvirheen vuoksi. Virhe korjattiin boottaamalla systeemit kerran vuorokaudessa. Nykyiset ohjelmointikielet käyttävät liukuluvuista tyyppinimiä float, double tai real.

Liukulukujen esitys

$+1.23 = +1.23 * 10^{0}$

$+123.0 = +1.23 * 10^{2}$

$+0.123 = +1.23 * 10^{-1}$

$-0.000\ 000\ 000\ 123 = -1.23 * 10^{-10}$

$+123\ 000\ 000\ 000\ 000 = +1.23 * 10^{14}$

sign exponent mantissa, significant
etumerkki eksponentti mantissa, lukuarvo

Liukulukujen esitys perustuu ideaan, että jokainen realilukua muistuttava liukuluku annetaan kolmen kentän avulla. Tässä esimerkissä luku +1.23 annetaaan etumerkin ('+'), lukuarvon ('1.23') ja suuruusluokan (10 potenssiin 0) avulla.

Liukulukujen esitys

$+1.23 = +1.23 * 10^{0}$

$+123.0 = +1.23 * 10^{2}$

$+0.123 = +1.23 * 10^{-1}$

$-0.000\ 000\ 000\ 123 = -1.23 * 10^{-10}$

$+123\ 000\ 000\ 000\ 000 = +1.23 * 10^{14}$

Nyt jos luvun suuruusluokka muuttuu, mutta itse numerot säilyvät samana, niin ainoastaan suuruusluokka muuttuu esityksessä. Tässä suuruusluokka on 10 potenssiin 2 eli sadat. Olennaista esityksessä on, että numeroarvo (tässä 1.23) esitetään aina samalla tavalla suuruusluokasta riippumattomasti.

Liukulukujen esitys

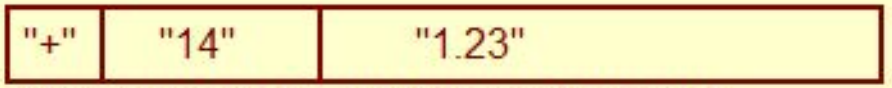
$$+1.23 = +1.23 * 10^{0}$$

$$+123.0 = +1.23 * 10^{2}$$

$$0.123 = 1.23 * 10^{-1}$$

$$-0.000\ 000\ 000\ 123 = -1.23 * 10^{-10}$$

$$+123\ 000\ 000\ 000\ 000 = +1.23 * 10^{14}$$



sign	exponent	mantissa, significant
etumerkki	eksponentti	mantissa, lukuarvo

Sama pätee myös ykköstä pienemmille luvuille. Tässäkin esimerkissä etumerkki ja numerot ovat samat kuin aikaisemmin, mutta suuruusluokkana on nyt 10 potenssiin -1 eli kymmeresosat. Numeroarvo pitää siis jakaa kymmenellä luvun arvon saamiseksi.

Liukulukujen esitys

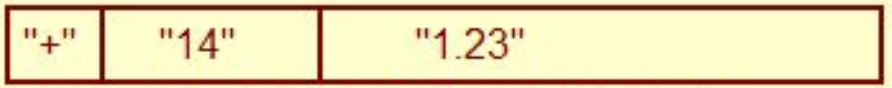
$+1.23 = +1.23 * 10^{0}$

$+123.0 = +1.23 * 10^{2}$

$+0.123 = +1.23 * 10^{-1}$

$-0.000\ 000\ 000\ 123 = -1.23 * 10^{-10}$

$+123\ 000\ 000\ 000\ 000 - +1.23 * 10^{14}$



sign exponent mantissa, significant
etumerkki eksponentti mantissa, lukuarvo

Tässä on hyvin pieni negatiivinen luku, jonka merkitsevät numerot ovat kuitenkin samat kuin aikaisemminkin.
Negatiivisuus näkyy esitysmuodon etumerkissä ja pienuus suuruusluokkakentässä.

Liukulukujen esitys

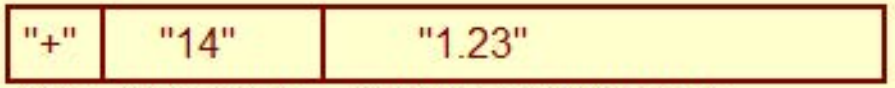
$+1.23 = +1.23 * 10^{0}$

$+123.0 = +1.23 * 10^{2}$

$+0.123 = +1.23 * 10^{-1}$

$-0.000\ 000\ 000\ 123 = -1.23 * 10^{-10}$

$+123\ 000\ 000\ 000\ 000 = +1.23 * 10^{14}$



sign	exponent	mantissa, significant
etumerkki	eksponentti	mantissa, lukuarvo

Tässä taas on hyvin suuri positiivinen luku, mutta senkin numeroarvo on edelleen vain tuo 123. Suuruus näkyy ainoastaan suuruusluokan esitystavassa.

Liukulukujen esitys

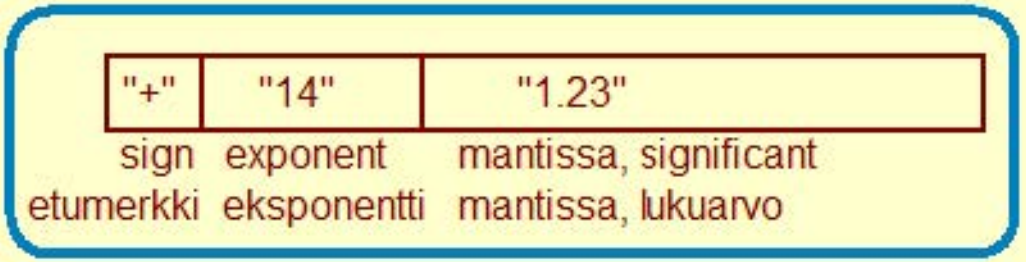
$+1.23 = +1.23 * 10^{0}$

$+123.0 = +1.23 * 10^{2}$

$+0.123 = +1.23 * 10^{-1}$

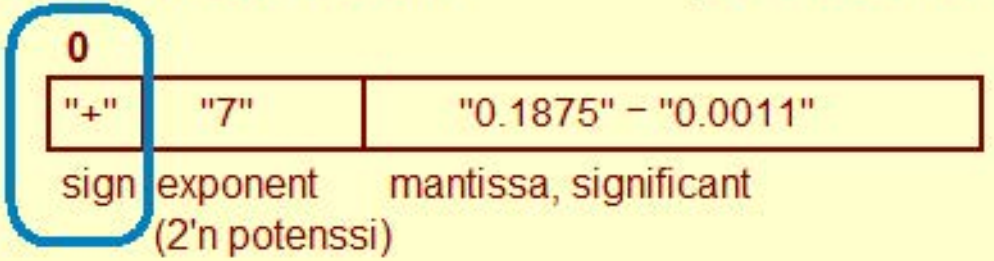
$-0.000\ 000\ 000\ 123 = -1.23 * 10^{-10}$

$+123\ 000\ 000\ 000\ 000 = +1.23 * 10^{14}$



Nämä kolme esitystavan kenttää pitää nyt tallettaa jollain tavoin annettuun esimerkiksi 32-bittiseen sanaan. Jokaiselle kentällä pitää olla jokin sovittu esitysmuoto ja pituus. Tässä esimerkissä kaikki annettiin 10-järjestelmän avulla, mutta todellinen toteutus perustuu tietenkin binäärijärjestelmään.

IEEE 32-bitin liukulukustandardi (IEEE 32-bit Floating Point Standard)



$= +24.0$
$= 11000.0_{2}$
$= 0.0011 * 2^{7}$

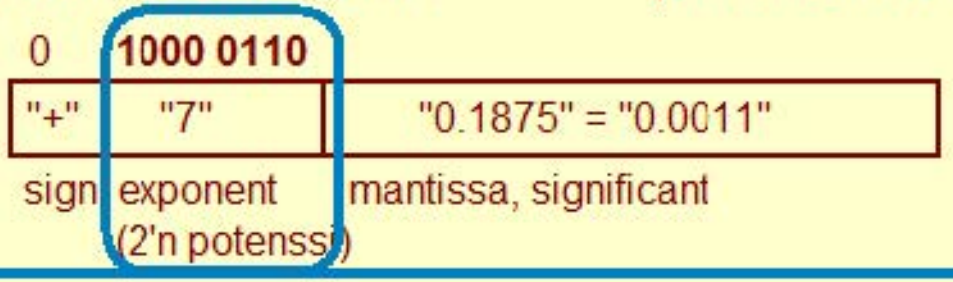
Etumerkki

- pituus: 1 bitti
- koodaus: '+' = 0, '-' = 1
- esitysmuodosta *S* saadaan arvo: $(-1)^{S}$

'+': $(-1)^{0} = +1$
'-': $(-1)^{1} = -1$

Lähes kaikki nykyiset suorittimet käyttävät samaa IEEE'n liukulukustandardia. Tämä on oikein hyvä asia, koska nyt eri järjestelmät antavat täsmälleen samat vastaukset samalla koodilla! Etumerkki koodataan yhdellä bitillä tavanomaiseen tapaan. Kun tietystä liukuluvun esitysmuodosta lasketaan sen lukuarvoa, etumerkkibitti voidaan muuttaa mukavasti plus/miinus ykköseksi oheisen kaavan avulla. Etumerkkibitin käyttö on tässä siis juuri samanlainen kuin sen käyttö kokonaislukujen etumerkkibittiesityksessä.

IEEE 32-bitin liukulukustandardi (IEEE 32-bit Floating Point Standard)



= +24.0
= 11000.0_{2}
= $0.0011 * 2^{7}$

Eksponentti

- piluus. 8 billiä
- koodaus: vakiolisäys 127 (biased form) talletusmuoto (8 bit)

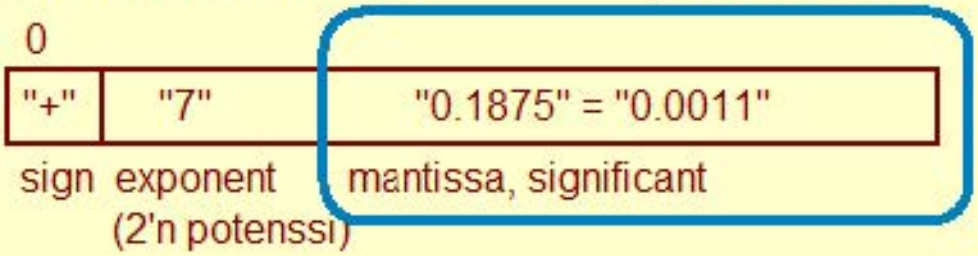
eksponentti = 7	→	7+127 = 134 = 1000 0110
eksponentti = 4	→	4+127 = 131 = 1000 0011
eksponentti = -1	→	-1+127 = 126 = 0111 1110
eksponentti = 0	→	0+127 = 127 = 0111 1111

- esitysmuodot 0 (0x00) ja 255 (0xFF) erikoistapauksia
  - laajennettu arvoalue, hyvin pienet luvut, NaN, **"4**
- talletettu esitysmuoto 1-254 vastaa arvoaluetta [-126, 127]
  - esitysmuodosta E saadaan arvo vähentämällä siitä 127

E arvo
134 - 127 = 7

Luvun eksponentin koodaamisessa käytetään aikaisemmin mainittua vakiolisäystä, jonka avulla kaikki eksponentin esitysmuodot ovat positiivisia kokonaislukuja. Eksponentin pituus on vain 8 bittiä, joten sen arvoalue on aika pieni [-126, +127]. Esitystavat 0x00 ja 0xFF on varattu erikoistapauksiin, joiden avulla meillä on esitysmuodot myös poikkeuksellisen pienille luvuille, määrittelemättömille luvuille ja plus/miinus äärettömälle. Joissakin tapauksissa näitä lukuja voi sitten käyttää myös aritmetiikkaoperaatioissa. Esimerkiksi, voi olla, että operaatio ääretön plus 1 on sallittu ja että tulos on ääretön!

IEEE 32-bitin liukulukustandardi (IEEE 32-bit Floating Point Standard)



= +24.0
= 11000.0_{2}
= $0.0011 * 2^{7}$

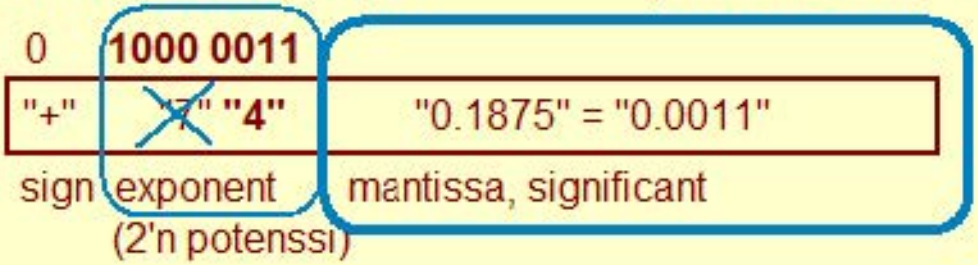
Mantissa

- pituus: 23 bittiä
- binääripiste on heti ensimmäisen bitin jälkeen

	mantissa	eksponentti	
	0.0011	"7"	alkuaan

Mantissan esitysmuoto on mielenkiintoisempi. Mantissahan on nyt aina pelkkä etumerkitön binäärilukuarvo, jossa on jossakin päin binääripiste. Mantissan esitys on standardoitu tilaa säästävällä tavalla. Ensinnäkin esitystapaan kuuluu, että kokonaisosan bittejä eli binääribitin vasemmalla puolella olevia bittejä on vain yksi kappale. Esimerkissä näin on valmiiksi, mutta muutoin bittejä voidaan joutua siirtämään oikealle tai vasemmalle tähän pääsemiseksi. Lukuarvon pitää tietenkin säilyä muuttumattomana, joten binääribitin siirtoa esimerkiksi oikealle yhden pykälän verran pitää kompensoida vähentämällä eksponenttia yhdellä. Ja päinvastoin tietenkin.

IEEE 32-bitin liukulukustandardi (IEEE 32-bit Floating Point Standard)



= +24.0
= 11000.0_{2}
= $0.0011 * 2^{7}$

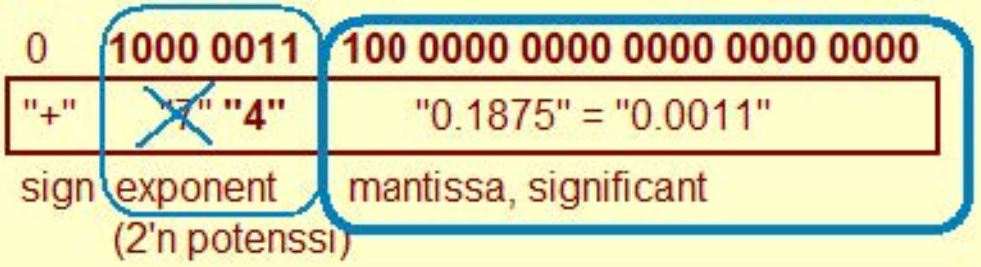
Mantissa

- pituus: 23 bittiä

	mantissa	eksponentti	
• binääripiste on heti ensimmäisen bitin jälkeen	0.0011	"7"	alkuaan
• mantissa on normalisoitu siten, että vasemmanpuolimmainen bitti on 1	1.1000	"4"	normalis

Seuraavaksi esitys normalisoidaan siten, että vasemmanpuolinen bitti on ykkönen. Esimerkissä binääripistettä pitää tämän takia siirtää oikealle 3 bittiä, joten eksponenttia pitää vähentää kolmella, jotta lukuarvo ei muuttuisi. Esitysmuodon eksponettikentän lukuarvo siis vaihtuu seiskasta neloseen.

IEEE 32-bitin liukulukustandardi (IEEE 32-bit Floating Point Standard)



= +24.0
= 11000.0_2
= $0.0011 * 2^7$

Mantissa

- pituus: 23 bittiä
- binääripiste on heti ensimmäisen bitin jälkeen
- mantissa on normalisoitu siten, että vasemmanpuolimmainen bitti on 1
- vasemmanpuolimmaista bittiä (eniten merkitsevää bittiä) ei talleteta
  - sen tiedetään olevan 1
  - piilobitti, implied bit

	mantissa	eksponentti	
	0.0011	"7"	alkuaan
	1.1000	"4"	normalis.
	1000	"4"	esitys

Viimeinen vaihe onkin sitten todella mielenkiintoinen. Kun kerran tiedämme kokonaisosabittien lukumäärän olevan 1 ja tämän yhden kokonaisosabitin arvon olevan 1, niin meidän ei tarvitse tallettaa kokonaisosan pituutta eikä sen arvoa. Talletamme siis ainoastaan binääripisteen jälkeen tulevat bitit. Tämän piilobitin idean keksi Konrad Zuse jo ennen toista maailmansotaa. Hän myös toteutti sen mekaanisessa laskimessaan, joka olisi voinut olla tärkeä merkkipylväs tietokoneen kehittelyssä, ellei Zuse olisi sattunut asumaan siihen aikaan vähän eristyksissä olevassa maassa. Piilobitin avulla saamme siis talletettua 24 bitin mantissan 23 bitin kenttään! Tilan säästöstä tietenkin maksetaan normalisoinnin aiheuttamana lisätyönä aina liukuluvuilla operoitaessa.

IEEE liukulukuesimerkkejä

23.0 = ?

Tarkastellaanpa IEEE:n liukulukuesitystä esimerkin avulla. Mikä on luvun 23.0 esitysmuoto 32 bitin liukulukustandardissa?

IEEE liukulukuesimerkkejä

23.0 = ?

$23.0 = +1\ 0111.0 * 2^{0} = +1.0111\ 0 * 2^{4} = ?$

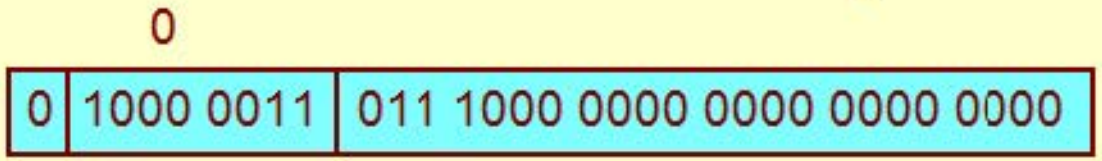
HUOM: Esimerkki käsittelee yleistä tapausta,
ei "0, ei "4, ei hyvin pieniä (epätarkkoja) lukuja

Luvun +0.0 esitysmuodoksi on sovittu 0x00000000
Luvun -0.0 esitysmuodoksi on sovittu 0x80000000
Luvun +4 esitysmuodoksi on sovittu 0x7F800000
Luvun -4 esitysmuodoksi on sovittu 0xFF800000

IEEE:n standardissa luku esitetään 3 kentän avulla, joten muutamme luvun 23.0 ensin binääri-esitysmuotoon, ja sitten normalisoituun binääriesitysmuotoon. Kuten edellä esitettiin, normalisoidussa esitysmuodossa luku esitetään etumerkin, ykkösen ja kakkosen välillä olevan itseisarvon ja suuruusluokan avulla. Etumerkki on tässä +, skaalattu itseisarvo 1.0111 ja suuruusluokka 2 potenssiin 4. Huomaa, että lukua 0.0 ei voi esittää tässä muodossa, vaan sille on sovittu ihan oma poikkeuksellinen esitystapansa.

IEEE liukulukuesimerkkejä

23.0 = ? $23.0 = +1\ 0111.0 * 2^{0} = +1.0111\ 0 * 2^{4} = ?$



Palataan nyt esimerkkiin. Esitysmuodossa ensimmäisenä on etumerkki. Plussa koodataan tässä nollaksi ja sijoitetaan paikalleen.

IEEE liukulukuesimerkkejä

23.0 = ? $23.0 = +1\ 0111.0 * 2^{0} = +1.0111\ 0 * 2^{4} = ?$

4+127=131

0	1000 0011	011 1000 0000 0000 0000 0000

etum. eksponentti mantissa

Suuruusluokka on seuraavana. Kakkosen potenssiin neljä lisätään 127 ja tulos 131 talletetaan etumerkittömänä kokonaislukuna 8 bitin kenttään. Tällä tavoin talletettava lukuarvo on aina vähintään nolla.

IEEE liukulukuesimerkkejä

23.0 = ? $23.0 = +1\ 0111.0 * 2^{0} = +1.0111\ 0 * 2^{4} = ?$

0111 0

0	1000 0011	011 1000 0000 0000 0000 0000

etum. eksponentti mantissa

Luvun skaalatusta itseisarvosta 1.0111 otetaan binääriosa ja talletetaan se mantissan arvoksi. Mantissan loppuosa täytetään nollilla.

IEEE liukulukuesimerkkejä

23.0 = ?

$23.0 = +1\ 0111.0 * 2^{0} = +1.0111\ 0 * 2^{4} = ?$

4 + 127 = 131 = 0x83

= 0x41B80000

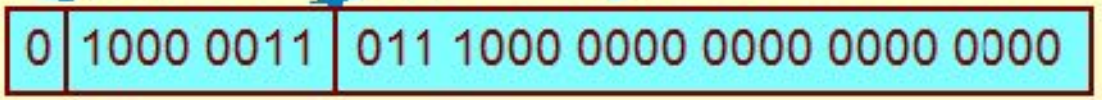
etum. eksponentti mantissa

Luvun 23.0 esitysmuoto on siten hexa 41B80000. Esitysmuoto ei ole niin hyvin suoraan hahmotettavissa, koska eksponentin 8 bittiä on osana kolmea eri heksadesimaalinumeroa. Ensimmäinen heksadesimaalinumero (4) käsittää sekä etumerkin että kolme ensimmäistä eksponentin bittiä.

IEEE liukulukuesimerkkejä

23.0 = ? 23.0 = +1 0111.0 * 2^{0} = +1.0111 0 * 2^{4} = ?

4 + 127 = 131 = 0x83



= 0x41B80000

etum. eksponentti mantissa

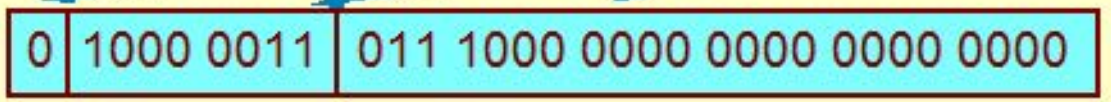
1.0 = ? 1.0 = +1.0000 * 2^{0} = ?

Otetaan toinen esimerkki, luku 1.0. Sekin jaetaan ensin etumerkkiin plus, normeerattuun itseisarvoon 1.0 ja suuruusluokkaan 2 potenssiin 0.

IEEE liukulukuesimerkkejä

23.0 = ? 23.0 = +1 0111.0 * 2^0 = +1.0111 0 * 2^4 = ?

4 + 127 = 131 = 0x83

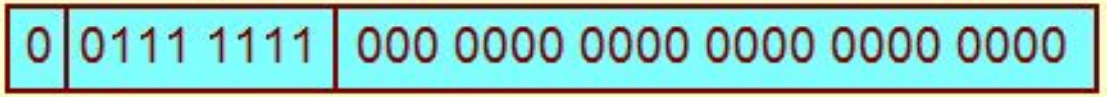


= 0x41B80000

etum. eksponentti mantissa

1.0 = ? 1.0 = +1.0000 * 2^0 = ?

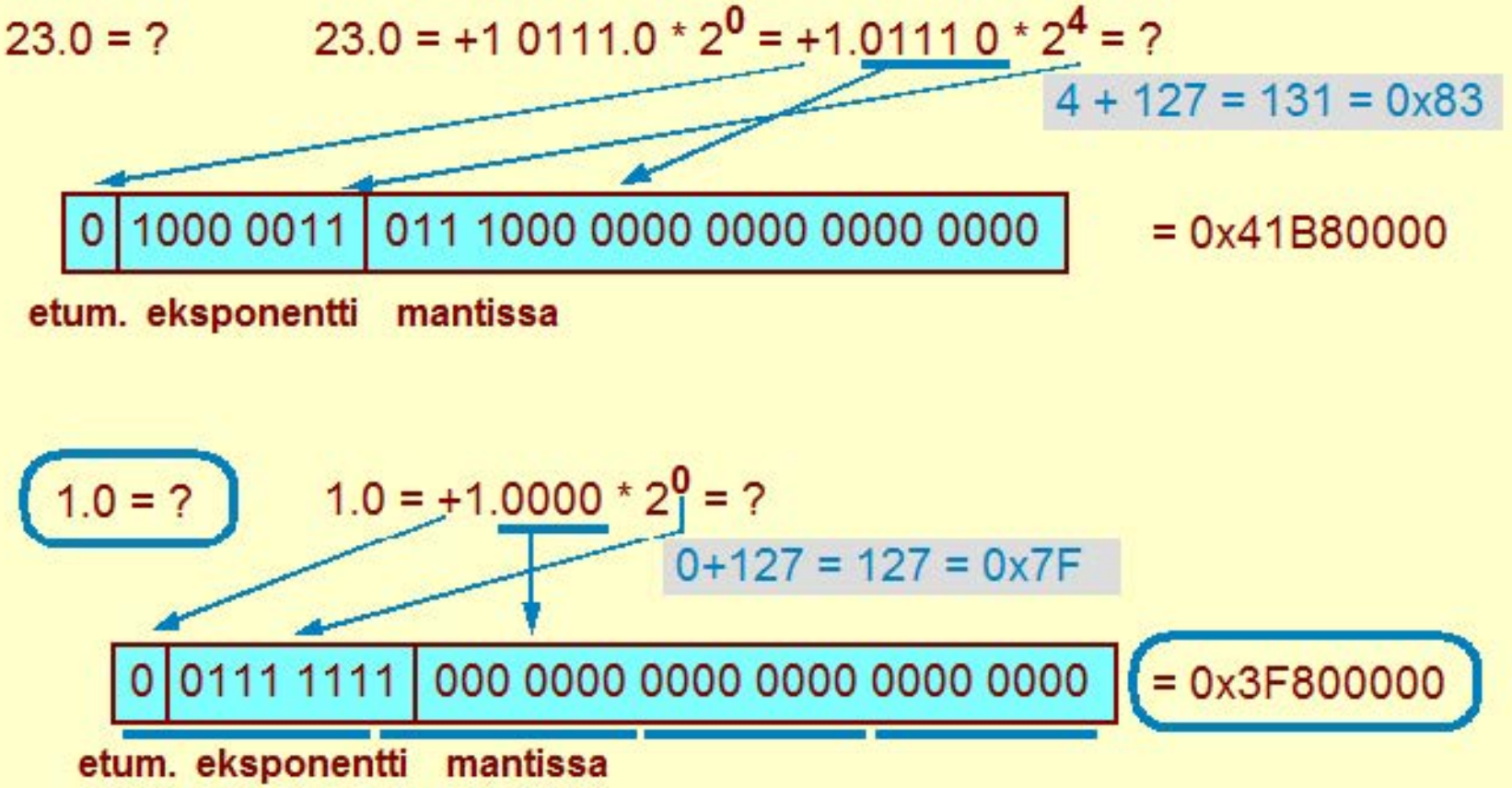
0+127 = 127 = 0x7F



etum. eksponentti mantissa

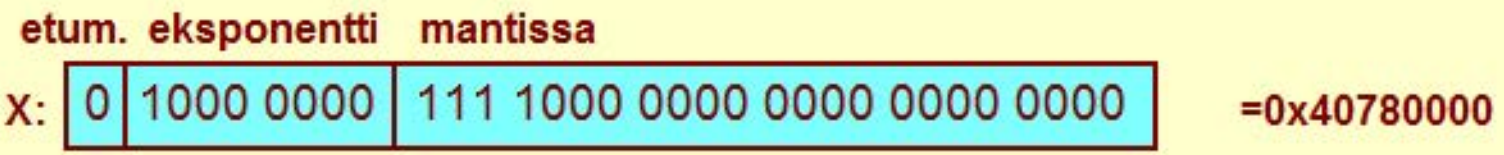
Luvun 1.0 kentät koodataan yksi kerrallaan esitystapaan. Huomaa, että exponentin esitystapana on pelkkä vakiolisäys 127, ja, että mantissan esitystapana on nyt pelkkiä nollia, kun 1.0:sta jäi piilobitti ykkönen pois.

IEEE liukulukuesimerkkejä



Luvun 1.0 esitysmuoto on siis hexa 3F800000, mikä ei tosiaankaan tuo lukua 1 heti mieleen.

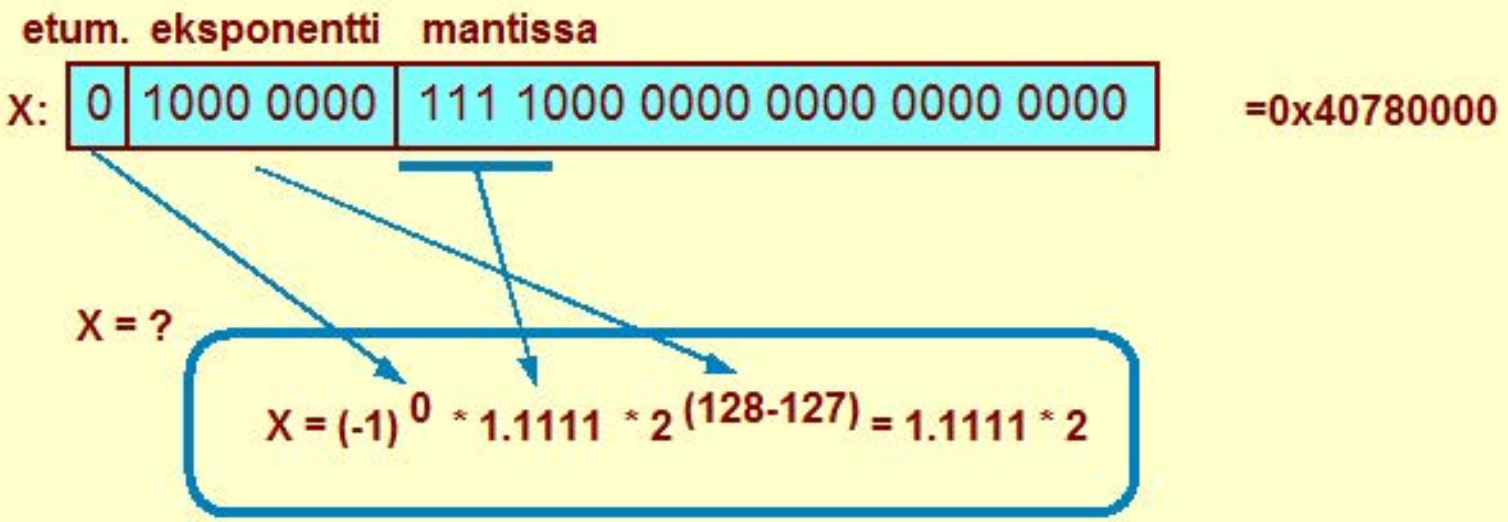
IEEE esitystapaa vastaava lukuarvo



X = ?

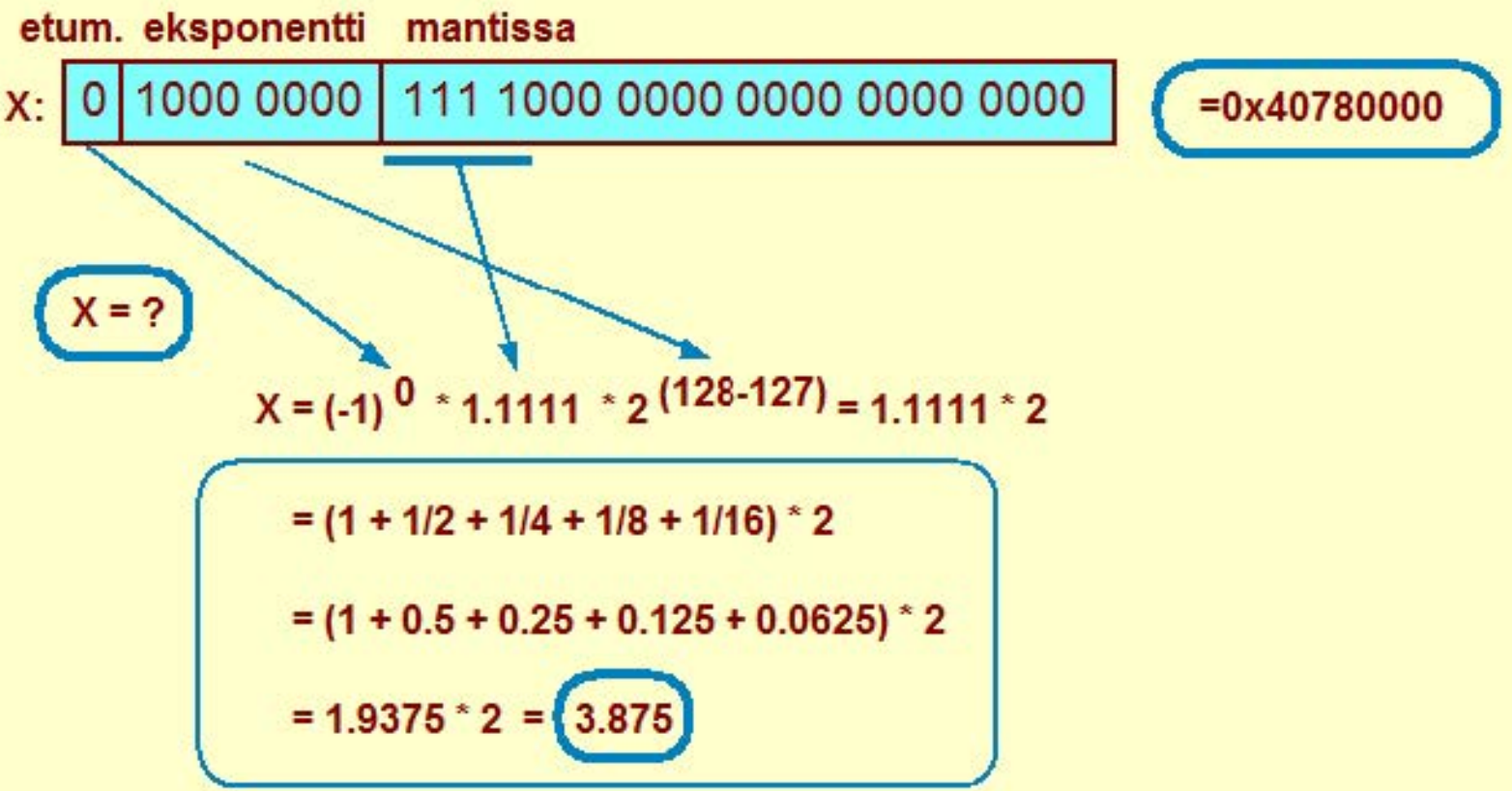
Tarkastellaan nyt toisen esimerkin avulla, kuinka liukulukuesitystavasta saadaan selville sen arvo eli mitä lukua esitystavalla tarkoitetaan. Esimerkissä liukulukumuuttujan X arvo on IEEE-standardin mukaisesti talletettuna 0x40780000. Mikä on muuttujan X nykyarvo?

IEEE esitystapaa vastaava lukuarvo



Kentät puretaan arvoiksi yksi kerrallaan. Etumerkkibitin 0 lukuarvo on -1 potenssiin 0 eli luku +1. Mantissaan laitetaan piilobitti paikalleen ja lopullinen mantissa on siten 1.1111. Exponentti on talletettu arvona 128, joten eksponentin arvo saadaan vähentämällä siitä siihen alkuaan lisätty 127. Exponentin arvo on siis 1.

IEEE esitystapaa vastaava lukuarvo



Binääriluku 1.1111 vastaa desimaaliarvoa 1.9375, mikä pitää vielä skaalata suuruusluokalla 2 potenssiin 1 eli kakkosella. Heksadesimaaliesitystä 40780000 vastaa siis lukuarvo 3.875, mikä siis on muuttujan X nykyinen arvo. Laskennan aikana tätä arvoa ei tietenkään tarvitse muuttaa 3.875:ksi missään vaiheessa. Toisaalta, esimerkiksi tulostettaessa tulostusrutiinit voivat purkaa arvon desimaaliesitykseen 3.875, koska se on ihmisen helpompi lukea.

Merkkien esitystapa

Yleensä 1 tavu (8 bittiä) per merkki

ASCII, 7 bittiä/merkki

ASCII = American Standard Code for Information Interchange

- 8. bitti (vas. puol.) tavussa voisi olla tarkistusbitti
- ei ä- tai ö-kirjaimia

'A' = 0x41, 'a' = 0x61, LF = 0x0A

EBCDIC, 8 bittiä/merkki

EBCDIC = Extended Binary Coded Decimal Interchange Code

- ei ä- tai ö-kirjaimia
- pohjautuu 6 bitin reikäkorttikoodistoon

'A' = 0xC1, 'a' = 0x81, LF = 0x25

ISO/IEC 8859-15 (eli 'ISO Latin9')

'A' = 0x41, 'a' = 0x61, LF = 0x0A

- 8 bittiä/merkki, 258 eri merkkiä käytössä
- ens. 128 merkkiä (0x00-0x7F) samat kuin ASCII'ssa
- seur. 32 merkkiä (0x80-0x9F) kontrollimerkkejä (VT100 escape)
- loput 96 merkkiä (0xA0-0xFF) Latin9 omia merkkejä
- mukana myös 'ä', 'ö', '£', '€'

'HTJ' = 0x89 (Horizontal Tab Set)

'€' = 0xA4, 'ä' = 0xE4, 'å' = 0xE5

Aakkosnumeeriset merkit esitetään tietenkin myös bitteinä. Tyypillisesti ainakin Suomessa käytössä olevat merkit esitetään sillä tavoin, että kukin merkki koodataan yhteen 8-bitin tavuun. Tämä on hyvin kätevää, koska yhdellä tavulla esitettävät 256 erilaista merkkiä tuntuvat riittävän useimpiin sovelluksiin. Joskus kuitenkin tarvitaan enemmän merkkejä, mutta puhutaan siitä vähän myöhemmin.

Merkkien esitystapa

Yleensä 1 tavu (8 bittiä) per merkki

ASCII, 7 bittiä/merkki

ASCII = American Standard Code for Information Interchange

- 8. bitti (vas. puol.) tavussa voisi olla tarkistusbitti
- ei ä- tai ö-kirjaimia

'A' = 0x41, 'a' = 0x61, LF = 0x0A

EBCDIC, 8 bittiä/merkki

EBCDIC = Extended Binary Coded Decimal Interchange Code

- ei ä- tai ö-kirjaimia
- pohjautuu 6 bitin reikäkorttikoodistoon

'A' = 0xC1, 'a' = 0x81, LF = 0x25

ISO/IEC 8859-15 (eli 'ISO Latin9')

'A' = 0x41, 'a' = 0x61, LF = 0x0A

- 8 bittiä/merkki, 258 eri merkkiä käytössä
- ens. 128 merkkiä (0x00-0x7F) samat kuin ASCII'ssa
- seur. 32 merkkiä (0x80-0x9F) kontrollimerkkejä (VT100 escape)
- loput 96 merkkiä (0xA0-0xFF) Latin9 omia merkkejä
- mukana myös 'ä', 'ö', '£', '€'

'HTJ' = 0x89 (Horizontal Tab Set)

'€' = 0xA4, 'ä' = 0xE4, 'å' = 0xE5

Eräs vanhimmista merkistöistä on Amerikkalainen ASCII, joka esiteltiin jo 1963. Siinä on vain 7 bittiä, joten 8-bittisissä tavuissa yhtä bittiä voitiin haluttaessa käyttää tarkistusbittinä. Käsittelemme tarkistusbitin käyttöä myöhemmin tällä kurssilla. Amerikkalaiseen tapaan merkistöön ei kuulunut suomalaisia tai eurooppalaisia merkkejä kuten ä tai ö. Urheiluselostuksista päätelleen jotkut tuloksia näyttävät ohjelmat käyttävät edelleen ASCII-koodistoa tai siitä johdettuja koodistoja, joista puuttuu esimerkiksi ä- ja ö-kirjaimet. Tämä oli tietenkin vitsi. Todellisuudessa kyseessähän on vain sovellusten huono laatu.

Merkkien esitystapa

Yleensä 1 tavu (8 bittiä) per merkki

ASCII, 7 bittiä/merkki

ASCII = American Standard Code for Information Interchange

- 8. bitti (vas. puol.) tavussa voisi olla tarkistusbitti
- ei ä- tai ö-kirjaimia

'A' = 0x41, 'a' = 0x61, LF = 0x0A

EBCDIC, 8 bittiä/merkki

EBCDIC = Extended Binary Coded Decimal Interchange Code

- ei ä- tai ö-kirjaimia
- pohjautuu 6 bitin reikäkorttikoodistoon

'A' = 0xC1, 'a' = 0x81, LF = 0x25

ISO/IEC 8859-15 (eli 'ISO Latin9')

'A' = 0x41, 'a' = 0x61, LF = 0x0A

- 8 bittiä/merkki, 258 eri merkkiä käytössä
- ens. 128 merkkiä (0x00-0x7F) samat kuin ASCII'ssa
- seur. 32 merkkiä (0x80-0x9F) kontrollimerkkejä (VT100 escape)
- loput 96 merkkiä (0xA0-0xFF) Latin9 omia merkkejä
- mukana myös 'ä', 'ö', '£', '€'

'HTJ' = 0x89 (Horizontal Tab Set)

'€' = 0xA4, 'ä' = 0xE4, 'å' = 0xE5

IBM esitteli oman EBCDIC-koodistonsa vuotta myöhemmin 1964 IBM/360 järjestelmän yhteydessä. Siinäkään ei ollut ä- tai ö-kirjaimia, tietenkään. EBCDIC'stä oli lopulta myös kansallisia versioita, jotka eivät olleet yhteensopivia keskenään tai muiden merkistöjen kanssa. EBCDIC pohjautuu IBM'n varhaisten tietokoneiden BCD (eli Binary Coded Decimal) koodistoon ja se oli suunniteltu erityisesti reikäkorttien käyttöön soveltuvaksi.

Merkkien esitystapa

Yleensä 1 tavu (8 bittiä) per merkki

ASCII, 7 bittiä/merkki

ASCII = American Standard Code for Information Interchange

- 8. bitti (vas. puol.) tavussa voisi olla tarkistusbitti
- ei ä- tai ö-kirjaimia

'A' = 0x41, 'a' = 0x61, LF = 0x0A

EBCDIC, 8 bittiä/merkki

EBCDIC = Extended Binary Coded Decimal Interchange Code

- ei ä- tai ö-kirjaimia
- pohjautuu 6 bitin reikäkorttikoodistoon

'A' = 0xC1, 'a' = 0x81, LF = 0x25

ISO/IEC 8859-15 (eli 'ISO Latin9')

'A' = 0x41, 'a' = 0x61, LF = 0x0A

- 8 bittiä/merkki, 258 eri merkkiä käytössä
- ens. 128 merkkiä (0x00-0x7F) samat kuin ASCII'ssa
- seur. 32 merkkiä (0x80-0x9F) kontrollimerkkejä (VT100 escape)
- loput 96 merkkiä (0xA0-0xFF) Latin9 omia merkkejä
- mukana myös 'ä', 'ö', '£', '€'

'HTJ' = 0x89 (Horizontal Tab Set)

'€' = 0xA4, 'ä' = 0xE4, 'å' = 0xE5

ISO Latin-9 vuodelta 1999 on ehkä nyt yleisimmin käytetty merkistö, ainakin Suomessa. Kukin merkki on 8-bittinen ja merkistöön sisältyy ä- ja ö-kirjaimien lisäksi myös Punta, Dollari ja Euro. Kyseessä on siis kyseisen standardin 15. versio, jonka pääasiallisena lisäarvona aikaisempiin Suomessa käytössä oleviin merkistöihin verrattuna oli Euro-merkki. ISO Latin-9 merkistö ei kuitenkaan ole suinkaan ratkaissut kaikkia merkistö-ongelmia, kuten verkkoa vähänkin selaamalla saa selville. Eri puolilla maailmaa on edelleen käytössä useita erilaisia merkistöjä.

UCS ja Unicode

UCS (Universal Character Set) = Unicode

- ISO:n standardoima (International Standards Organization) UCS
- Amerikkalaisen konsortiumin standardoima Unicode
- vihdoinkin universaali merkistöstandardi?

2 tavua eli 16 bittiä per merkki

- 65536 merkkiä käytössä, maailmassa käytössä noin 200000 symbolia

Aluekoodit

- 1 tai useampi 8-bittinen rivi varattu tietyn alueen omalle koodistolle
- Latin9 Zone: rivin 00 (eli merkit 0x0000-00FF) 8 viimeistä bittiä samassa järjestyksessä kuin Latin9 merkistössä '€' = 0xA4 = 0x00A4, 'a' = 0x41 = 0x0041
- Japanilainen Kanji merkistö, I-zone, merkit 0x4E00-9FFF, 20992 merkkiä

Tilanne on normalisoitumassa merkistöstandardien suhteen. UCS yhdistää kaikki merkistöt saman standardin alle. Standardointityö alkoi samoihin aikoihin eri puolilla maailmaa ja (yllättävää kyllä) standardointiorganisaatiot hyvin pian yhdistivät voimansa yhteiseen merkistöön. Merkistöllä on kehityshistoriansa vuoksi edelleen käytössä kaksi eri nimeä, UCS ja Unicode, mutta ne tarkoittavat samaa merkistöä.

UCS ja Unicode

UCS (Universal Character Set) = Unicode

- ISO:n standardoima (International Standards Organization) UCS
- Amerikkalaisen konsortiumin standardoima Unicode
- vihdoinkin universaali merkistöstandardi?

2 tavua eli 16 bittiä per merkki

- 65536 merkkiä käytössä, maailmassa käytössä noin 200000 symbolia

Aluekoodit

- 1 tai useampi 8-bittinen rivi varattu tietyn alueen omalle koodistolle
- Latin9 Zone: rivin 00 (eli merkit 0x0000-00FF) 8 viimeistä bittiä samassa järjestyksessä kuin Latin9 merkistössä '€' = 0xA4 = 0x00A4, 'a' = 0x41 = 0x0041
- Japanilainen Kanji merkistö, I-zone, merkit 0x4E00-9FFF, 20992 merkkiä

Koko maailmassa on laskettu eri kielialueilla olevan noin 200000 eri symbolia käytössä. Näistä 65K on sitten valittu uuteen 16-bittiseen UCS merkistöön. Merkin pituuden laajentaminen 16 bittiin tuntuu meistä suomalaisista vähän erikoiselta, mutta esimerkiksi Japanissa merkin pituus on ollut 16 bittiä jo kauan aikaa. Merkin pituudella on huomattava vaikutus esimerkiksi käyttöjärjestelmien tekemisessä, minkä kaikki Japanin markkinoilla toimivat länsimaiset yhtiöt ovat jo kauan aikaa sitten havainneet ja siihen sopeutuneet.

UCS ja Unicode

UCS (Universal Character Set) = Unicode

- ISO:n standardoima (International Standards Organization) UCS
- Amerikkalaisen konsortiumin standardoima Unicode
- vihdoinkin universaali merkistöstandardi?

2 tavua eli 16 bittiä per merkki

- 65536 merkkiä käytössä, maailmassa käytössä noin 200000 symbolia

Aluekoodit

- 1 tai useampi 8-bittinen rivi varattu tietyn alueen omalle koodistolle
- Latin9 Zone: rivin 00 (eli merkit 0x0000-00FF) 8 viimeistä bittiä samassa järjestyksessä kuin Latin9 merkistössä '€' = 0xA4 = 0x00A4, 'a' = 0x41 = 0x0041
- Japanilainen Kanji merkistö, I-zone, merkit 0x4E00-9FFF, 20992 merkkiä

Vaikka Suomessakin käytetään UCS'ää, niin käytännössä meillä on siitä vain yhden rivin (rivi 00) osajoukko, jossa oletusarvoisesti kunkin merkin 8 ensimmäistä bittiä ovat nollia. Tällaisia aluekoodeja on muuallakin maailmassa paljon, mistä tietenkin seuraa, että eri puolilla maailmaa edelleenkin käytetään erilaisia koodistoja. Nyt ne on kuitenkin standardoitu kaikki saman katon alle. Kuhunkin tiedostoon voidaan esimerkiksi liittää tieto siitä, mitä standardin mukaista merkistöä se käyttää.

UCS ja Unicod

UCS (Universa

- ISO:n standa
- Amerikkalais
- vihdoinkin un

2 tavua eli 16 b

- 65536 merkk

Aluekoodit

- 1 tai useamp
- Latin9 Zone: järjestyksessä

= 0x0041

- Japanilainen

C1 Controls and Latin-1 Supplement

	008	009	00A	00B	00C	00D	00E	00F
0	XXX 0080	DCS 0090	NB SP 00A0	° 00B0	À 00C0	Đ 00D0	à 00E0	ð 00F0
1	XXX 0081	PU1 0091	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	BPH 0082	PU2 0092	¢ 00A2	2 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	NBH 0083	STS 0093	£ 00A3	3 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	IND 0084	CCH 0094	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4

http://www.unicode.org/charts/

ISO Latin9 rivillä (rivi 0x00) on siis kaikki Suomessa käytetyt merkit. Tässä on näkyvissä niistä ihan pieni osajoukko, jossa kuitenkin on jo mukana muutamia vähän harvinaisempia merkkejäkin. Huomaa siis, että riviä 0x00 käytettäesssä merkit voidaan tallettaa aina 8-bittisinä. Tarvittaessa esitysmuoto voidaan sitten laajentaa 16-bittiseksi pistämällä alkuun rivin tunnuksen 0x00 eli 8 nolla-bittiä.

UCS ja

UCS (U
- ISO
- Am
- vihc

2 tavua
- 655

Aluekc
- 1 ta
- Lati
järje
- Jap

4E00 CJK Unified Ideographs

	4E0	4E1	4E2	4E3	4E4	4E5	4E6	4E7	4E8	4E9	4EA
0	一 4E00	丐 4E10	北 4E20	丰 4E30	乀 4E40	乐 4E50	习 4E60	买 4E70	亀 4E80	亏 4E90	亠 4EA0
1	丁 4E01	丑 4E11	两 4E21	丱 4E31	乁 4E41	乑 4E51	乡 4E61	乱 4E71	乾 4E81	云 4E91	亡 4EA1
2	丂 4E02	刃 4E12	丢 4E22	串 4E32	义 4E42	乒 4E52	乢 4E62	乲 4E72	亂 4E82	互 4E92	亢 4EA2
3	七 4E03	专 4E13	丣 4E23	弗 4E33	乃 4E43	乓 4E53	乣 4E63	乳 4E73	亃 4E83	亓 4E93	亣 4EA3

http://www.unicode.org/charts/

Japanilaisessa Kanji-merkistössä on siis yhteensä yli 20000 merkkiä ja Kanji'in sisältyykin siis 82 kappaletta 256-merkkistä UCS-riviä.

UCS ja Unicode

UCS (Universal C

- ISO:n standardo
- Amerikkalaisen
- vihdoinkin unive

2 tavua eli 16 bitt

- 65536 merkkiä

Aluekoodit

- 1 tai useampi 8
- Latin9 Zone: rivi
  järjestyksessä k

- Japanilainen Ka

Gujarati

	0A8	0A9	0AA	0AB	0AC	0AD	0AE	0AF
0		ઐ 0A90	ઠ 0AA0	ર 0AB0	ી 0AC0	ૐ 0AD0	ૠ 0AE0	
1	ઁ 0A81	ઑ 0A91	ડ 0AA1		ુ 0AC1		ૡ 0AE1	૱ 0AF1
2	ં 0A82		ઢ 0AA2	ણ 0AB2	ૂ 0AC2		ૢ 0AE2	
3	ઃ 0A83	ઓ 0A93	ણ 0AA3	ત 0AB3	ૃ 0AC3		ૣ 0AE3	

http://www.unicode.org/charts/

Vähän eksoottisempana esimerkkinä on tässä Intialainen Gujarati-kielen merkistö.

UCS ja Unicode

UCS (Universal C

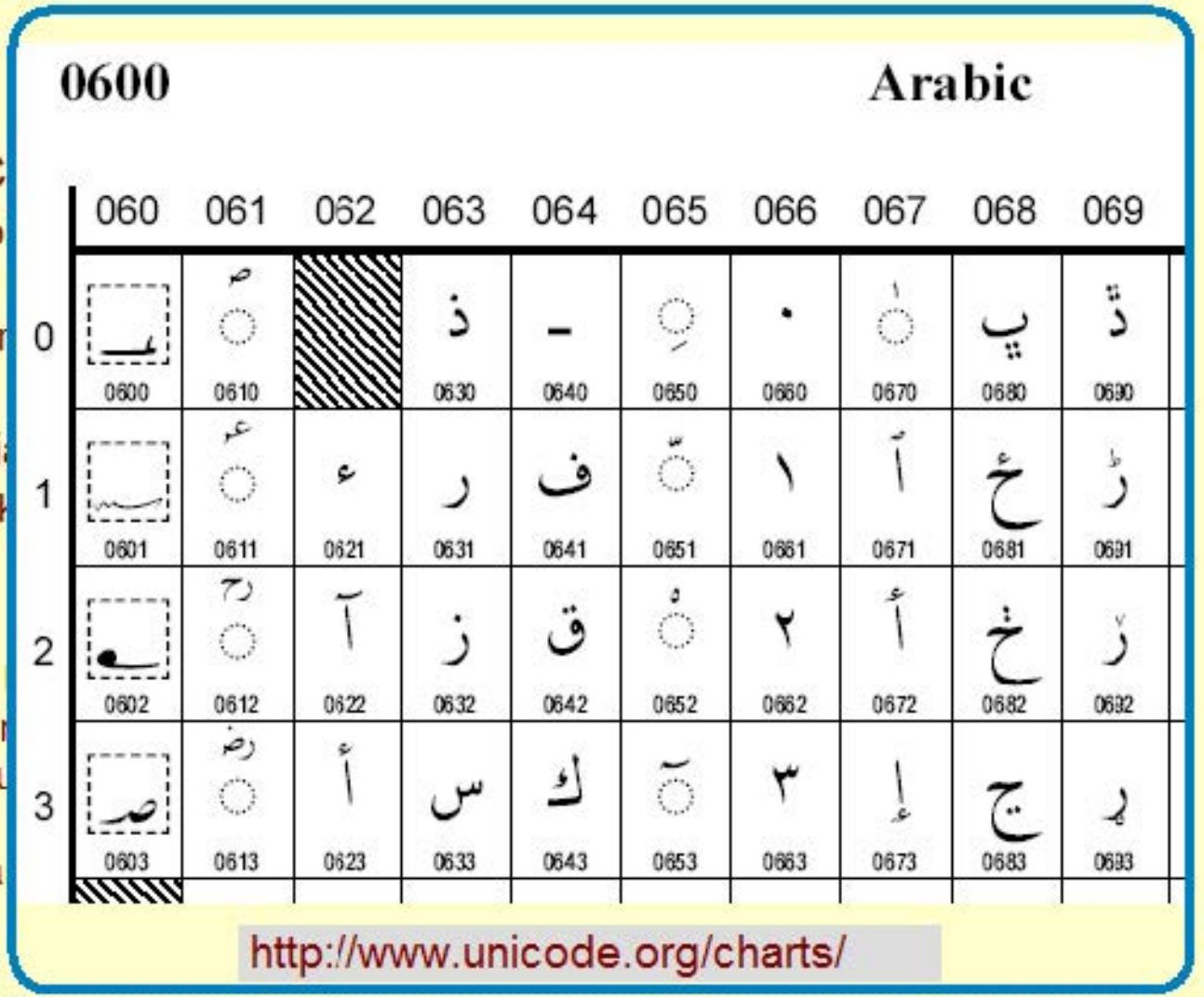
- ISO:n standardo
- Amerikkalaisen
- vihdoinkin univer

2 tavua eli 16 bitti

- 65536 merkkiä

Aluekoodit

- 1 tai useampi 8
- Latin9 Zone: rivi
  järjestyksessä ku
- Japanilainen Ka



Arabian kielelle on tietenkin myös oma rivinsä, rivi 0x06.

UCS ja Unicode

UCS (Universal Charac

- ISO:n standardoima (In
- Amerikkalaisen konsor
- vihdoinkin universaali m

2 tavua eli 16 bittiä per

- 65536 merkkiä käytöss

Aluekoodit

- 1 tai useampi 8-bittiner
- Latin9 Zone: rivin 00 (e
  järjestyksessä kuin Lati

- Japanilainen Kanji mer

Cherokee

	13A	13B	13C	13D	13E	13F
0	D 13A0	Ꭵ 13B0	G 13C0	Ꮴ 13D0	Ꮾ 13E0	ᏼ 13F0
1	R 13A1	Ꭻ 13B1	Ꭿ 13C1	Ꮵ 13D1	Ꮿ 13E1	Ᏽ 13F1
2	T 13A2	Ꭼ 13B2	h 13C2	R 13D2	P 13E2	Ᏺ 13F2
3	Ꭳ 13A3	W 13B3	Z 13C3	Ꮳ 13D3	Ꮹ 13E3	Ᏻ 13F3

http://www.unicode.org/charts/

0x0041

Viimeinen esimerkki on Amerikan Cherokee intiaaniheimon merkistö, joka löytyy UCS'n riviltä 0x13.

Merkkijonot

Yleensä peräkkäin talletettu jono merkkejä (tavuja tai tuplatavuja)

Merkkijono pituus koodattu jollain tavoin

- merkkijonon lopussa oleva erikoismerkki
  - C-kielessä ja UNIX-ympäristöissä: '\0' = 0x00
  - loppumerkki ei ole osa merkkijonoa, vaikka se talletetaankin muistiin
- merkkijono on tietue, jossa kentät pituus ja merkit

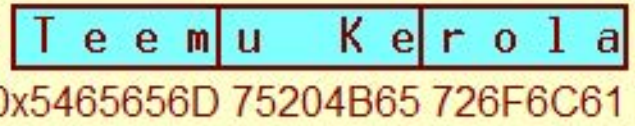
```
LOAD  R1, Str
LOAD  R2, =255  ; mask 0xFF
AND   R1, R2    ; 'm'

LOAD  R1, Str
LOAD  R2, =255  ; mask 0xFF
SHL   R2, =8
AND   R1, R2
SHR   R1, =8  ; 'e'
```

Merkkijonoja ei yleensä käsitellä omilla konekäskyillä, vaan niitä varten suunnitelluilla (käyttöjärjestelmän tai ohjelmointikielen) aliohjelmilla

- joissakin (vanhemmissa) koneissa on (esim.) strcopy ja strcomp konekäskyjä
  - vaatii tietyn muotoisen merkkijonon esitystavan
- yleensä merkkejä ja merkkijonoja käsitellään kokonaislukuoperaatioilla, jotka manipuloivat merkkien esitysmuotoja (kokonaislukuja)

Merkkijonot käsittävät itse merkkien lisäksi myös merkkijonon pituuden jollain tavalla koodattuna. Itse merkit talletetaan merkkijonoissa pakatussa muodossa siten, että joka (32 bitin) sanassa on neljä (8 bitin) merkkiä. Viimeisessä sanassa voi sitten olla myös täytebittejä, mutta kukin merkkijono talletetaan kokonaisiin sanoihin. Jos merkkijonon yksittäistä merkkiä halutaan käsitellä, se tulee eristää muista saman sanan merkeistä bittimanipulaatio-operaatioilla, käyttäen erityistä 1-biteistä koostuvaa 'maskia' ja AND- ja SHIFT-operaatioita.

Merkkijonot



Yleensä peräkkäin talletettu jono merkkejä (tavuja tai tuplatavuja)

Merkkijono pituus koodattu jollain tavoin

- merkkijonon lopussa oleva erikoismerkki
  - C-kielessä ja UNIX-ympäristöissä: '\0' = 0x00
  - loppumerkki ei ole osa merkkijonoa, vaikka se talletetaankin muistiin
- merkkijono on tietue, jossa kentät pituus ja merkit

"Teemu Kerola" = Teemu Kerola\0

Str: 0x5465656D 75204B65 726F6C61 **00**000000

Merkkijonoja ei yleensä käsitellä omilla konekäskyillä, vaan niitä varten suunnitelluilla (käyttöjärjestelmän tai ohjelmointikielen) aliohjelmilla

- joissakin (vanhemmissa) koneissa on (esim.) strcopy ja strcomp konekäskyjä
  - vaatii tietyn muotoisen merkkijonon esitystavan
- yleensä merkkejä ja merkkijonoja käsitellään kokonaislukuoperaatioilla, jotka manipuloivat merkkien esitysmuotoja (kokonaislukuja)

Merkkijonon loppuminen voidaan koodata erityismerkillä, mikä UNIX-järjestelmissä on yleisesti null-merkki '\0' eli 0x00. Menetelmällä on etuna, että merkkijono voi olla mielivaltaisen pitkä, koska merkkijonon pituutta ei ole eksplisiittisesti talletettu mihinkään rajalliseen tietoalkioon. Tämä sama piirre on osittain myös menetelmän heikkous, koska merkkijonoja käsiteltäessä ei koskaan voi etukäteen tietää, kuinka pitkä merkkijono on. Loppumerkin puuttuessa jonkin virheen vuoksi merkkijono onkin sitten liki 'äärettömän' pitkä, mistä hyvin pian aiheutuu käytännön ongelmia.

Merkkijonot

Yleensä peräkkäin talletettu jono merkkejä (tavuja tai tuplatavuja)

Merkkijono pituus koodattu jollain tavoin

- merkkijonon lopussa oleva erikoismerkki
  - C-kielessä ja UNIX-ympäristöissä: '\0' = 0x00
  - loppumerkki ei ole osa merkkijonoa, vaikka se talletetaankin muistiin
- merkkijono on tietue, jossa kentät pituus ja merkit

"Teemu Kerola" = {12, "Teemu Kerola"}

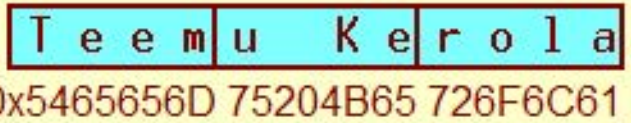
Str: 0x**0000000B** 5465656D 75204B65 726F6C61

Merkkijonoja ei yleensä käsitellä omilla konekäskyillä, vaan niitä varten suunnitelluilla (käyttöjärjestelmän tai ohjelmointikielen) aliohjelmilla

- joissakin (vanhemmissa) koneissa on (esim.) strcopy ja strcomp konekäskyjä
  - vaatii tietyn muotoisen merkkijonon esitystavan
- yleensä merkkejä ja merkkijonoja käsitellään kokonaislukuoperaatioilla, jotka manipuloivat merkkien esitysmuotoja (kokonaislukuja)

Toinen menetelmä on eksplisiittisesti kertoa kunkin merkkijonon yhteydessä, mitenkä monta merkkiä siihen kuuluu. Merkkijono on siis tietue, jossa on kaksi kenttää: pituus ja data eli itse merkit. Pituus koodataan esimerkiksi yhdellä sanalla ja itse merkit sitten tarvittavalla määrällä kokonaisia sanoja. Etuna tässä menetelmässä on etukäteen tarkistettavissa oleva tieto merkkijonon pituudesta. Huonona puolena on merkkijonon pituuden talletus, jokaesimerkissä vie aina kokonaisen sanan. Lyhyempääkin kenttää voidaan käyttää, mutta siitäkin tulee omia ongelmiaan pitkien merkkijonojen yhteydessä.

Merkkijonot



Yleensä peräkkäin talletettu jono merkkejä (tavuja tai tuplatavuja)

Merkkijono pituus koodattu jollain tavoin

- merkkijonon lopussa oleva erikoismerkki
  - C-kielessä ja UNIX-ympäristöissä: '\0' = 0x00
  - loppumerkki ei ole osa merkkijonoa, vaikka se talletetaankin muistiin
- merkkijono on tietue, jossa kentät pituus ja merkit

```
LOAD  R1, Str
LOAD  R2, =255  ; 0xFF
AND   R1, R2    ; 'm'
COMP R1, =109 ; 0x6D = 'm'
JEQU  SeOnM
```

Merkkijonoja ei yleensä käsitellä omilla konekäskyillä, vaan niitä varten suunnitelluilla (käyttöjärjestelmän tai ohjelmointikielen) aliohjelmilla

- joissakin (vanhemmissa) koneissa on (esim.) strcopy ja strcomp konekäskyjä
  - vaatii tietyn muotoisen merkkijonon esitystavan
- yleensä merkkejä ja merkkijonoja käsitellään kokonaislukuoperaatioilla, jotka manipuloivat merkkien esitysmuotoja (kokonaislukuja)

Vanhemmissa tietokoneissa oli usein omia konekäskyjä merkkien ja merkkijonojen manipulointiin. Nykyään näin ei useinkaan enää ole, vaan merkkejä ja merkkijonoja käsitellään tavallisilla kokonaislukukäskyillä. Tämä on huomattavasti joustavampi menetelmä nykytilanteessa, jossa useimpien laitteistojen tulee pystyä käyttämään hyvin monenlaisia eri merkistöjä.

Totuusarvojen talletus ja käyttö

Boolen totuusarvot True ja False

George Boole
Boolen algebra, 1854

Yleensä koodattu True=1 ja False=0

- varo: joissakin tapauksissa käytetty koodausta True=0, False=1
- totuusarvolauseke A and B kokonaislukulausekkeena A * B

Usein Boolen arvo per sana

- loput 31 bittiä nollia
- ohjelmissa käytetyt boolen muuttujat

Joskus pakattu muoto 32 arvoa per sana

- bitin "eristäminen" maskilla ja AND- ja SHIFT-operaatioilla

Ei omia konekäskyjä, manipulointi kokonaislukukäskyillä ja aliohjelmilla

- esim. halutun JTRUE käskyn asemesta kääntäjä käyttää JPOS käskyä

George Boole kehitti totuusarvoihin perustuvan algebransa jo 1854. Algebra on meille tärkeä kahdestakin syystä. Ensinnäkin, tietokoneen laitteisto perustuu binäärilogiikkaan, koska sitä on helppo toteuttaa ja ennen kaikkea sitä on helppo käsitellä Boolen algebran avulla. Laitteiston lisäksi useat ohjelmistot käyttävät paljonkin Boolen muuttujia sekä data-aineiston käsittelyssä että ohjelman kontrollilogiikan suunnittelussa. George Boole ansaitsee siis paljon kiitoksia meiltä tietojenkäsittelijöiltä, vaikka hän ei ajatellutkaan automaattista laskentaa algebraansa suunnitellessaan.

Totuusarvojen talletus ja käyttö

Boolen totuusarvot True ja False

Yleensä koodattu True=1 ja False=0

- varo: joissakin tapauksissa käytetty koodausta True=0, False=1
- totuusarvolauseke A and B kokonaislukulausekkeena A * B

Usein Boolen arvo per sana

- loput 31 bittiä nollia
- ohjelmissa käytetyt boolen muuttujat

Joskus pakattu muoto 32 arvoa per sana

- bitin "eristäminen" maskilla ja AND- ja SHIFT-

Ei omia konekäskyjä, manipulointi kokonaislukukäskyillä ja aliohjelmilla

- esim. halutun JTRUE käskyn asemesta kääntäj[illä]

```
; boolean muuttujat aX, bX (0 tai 1)
LOAD  R1, aX
ADD   R1, bX
STORE R1, aX ; aX = aX AND bX

LOAD  R1, aX
MUL   R1, bX
STORE R1, aX ; aX = aX OR bX

LOAD  R1, =1
SUB   R1, aX
STORE R1, aX ; aX = NOT (aX)
```


Totuusarvot koodataan yleensä siten, että arvoa tosi vastaa 1 ja arvoa epätosi arvo 0, koska tällöin voimme käyttää tavallisia kokonaislukujen aritmetiikkaoperaatioita totuusarvojen käsittelyyn, jolloin suorittimesta tulee yksinkertaisempi. Tämä ei tietenkään estä sitä, etteivätkö totuusarvot joissakin tapauksissa olisi kuitenkin koodattu juuri päin vastoin. Pitäkää varanne. Oletamme nyt jatkossa, että koodaus on juuri True=1 ja False=0.

Totuusarvojen talletus ja käyttö

Boolen totuusarvot True ja False

Yleensä koodattu True=1 ja False=0

- varo: joissakin tapauksissa käytetty koodausta True=0, False=1
- totuusarvolauseke A and B kokonaislukulausekkeena A * B

Usein Boolen arvo per sana

- loput 31 bittiä nollia
- ohjelmissa käytetyt boolen muuttujat

bA: **0x00000001**

bB: **0x00000000**

Joskus pakattu muoto 32 arvoa per sana

- bitin "eristäminen" maskilla ja AND- ja SHIFT-operaatioilla

Ei omia konekäskyjä, manipulointi kokonaislukukäskyillä ja aliohjelmilla

- esim. halutun JTRUE käskyn asemesta kääntäjä käyttää JPOS käskyä

Vaikka totuusarvojen arvojoukkoon kuuluu vain kaksi arvoa, niin useimmiten totuusarvotyyppiselle muuttujalle varataan käytännön syistä kokonainen esimerkiksi 32-bittinen sana. Tällä tavoin tilaa menee vähän hukkaan, mutta totuusarvon käyttö on nopeata, kun sana voidaan käyttää sellaisenaan.

Totuusarvojen talletus ja käyttö

Boolen totuusarvot True ja False

Yleensä koodattu True=1 ja False=0

- varo: joissakin tapauksissa käytetty koodausta True=0,
- totuusarvolauseke A and B kokonaislukulausekkeena A

Usein Boolen arvo per sana

- loput 31 bittiä nollia
- ohjelmissa käytetyt boolen muuttujat

Joskus pakattu muoto 32 arvoa per sana

- bitin "eristäminen" maskilla ja AND- ja SHIFT-operaatioilla

Ei omia konekäskyjä, manipulointi kokonaislukukäskyillä ja aliohjelmilla

- esim. halutun JTRUE käskyn asemesta kääntäjä käyttää

```
bArrA: 0x23A5B262

; set bit 15 in bArrA to 1
LOAD   R1, bArrA
LOAD   R2, =1  ; mask
SHL    R2, =15  ; mask bit 15
OR     R1, R2
STORE  R1, bArrA
```

```
; read bit 9 in bArrA
LOAD   R1, bArrA
LOAD   R2, =1  ; mask
SHL    R2, =9  ; mask bit 9
AND    R1, R2
SHR    R2, =9 ; bit 9 in bArrA
```

```
; bittimanipulaatio 32 bitille
LOAD   R1, bArrA
AND    R1, bArrB
STORE R1, bArrA
```


Jos totuusarvoja on paljon, esimerkiksi laskennan datajoukoissa, ne kannattaa pakata tiiviisti, hyödyntäen jokaista sanan 32 bittiä. Tilaa säästyy, mutta yksittäisten totuusarvojen käsittely tulee hankalammaksi, koska kukin käsiteltävä totuusarvo pitää aluksi eristää sanasta jo aikaisemmin tutulla maskin käytöllä. Toisaalta, konekäskyinä olevat bittimanipulaatiokäskyt on yleensä toteutettu juuri tällaisia pakattuja totuusarvoja varten ja ne tekevät halutut loogiset operaatiot kaikille operandien biteille. Tässä mielessä esimerkkikoneen AND-käsky on siis bittimanipulaatiokäsky eikä looginen operaatio -käsky.

Totuusarvojen talletus ja käyttö

Boolen totuusarvot True ja False

```
JTRUE  → JPOS
JFALSE → JZER
AND    → ADD (?)
OR     → MUL (?)
```

Yleensä koodattu True=1 ja False=0

- varo: joissakin tapauksissa käytetty koodausta True=0, False=1
- totuusarvolauseke A and B kokonaislukulausekkeena A * B

```
char *strcat (s, ct)
char *strchr(cs, c)
```

Usein Boolen arvo per sana

- loput 31 bittiä nollia
- ohjelmissa käytetyt boolen muuttujat

```
; boolean muuttujat aX (0 tai 1)
LOAD  R1, aX
JPOS  R1, Here ; jump if aX = TRUE
```

Joskus pakattu muoto 32 arvoa per sana

- bitin "eristäminen" maskilla ja AND- ja SHIFT-operaatioilla

```
; bA = bA AND bB
LOAD  R1, bA
AND   R2, bB
STORE R1, bA
```

Ei omia konekäskyjä, manipulointi kokonaislukukäskyillä ja aliohjelmilla

- esim. halutun JTRUE käskyn asemesta kääntäjä käyttää JPOS käskyä

Yksittäisiä totuusarvoja voidaan siis ihan yhtä hyvin käsitellä niiden esitysmuototasolla kokonaislukukäskyillä, joten käskykantaan ei tarvitse sisältyä erillisiä konekäskyjä tätä tietotyyppiä varten. Toisaalta, jos käskykantaan sisältyy bittimanipulaatiokäskyjä 32 bittisille sanoille, niin noita samoja käskyjä voidaan hyvin käyttää myös 1 bitin totuusarvoille, olettaen tietenkin, että boolen muuttujissa tosi on koodattu ykköseksi ja epätosi nollaksi.

Kuvien talletus

Monta kuvastandardia

- pakkaustiheys, yleisyys, siirrettävyys
- laskennan määrä pakatessa ja purettaessa

tiff, gif, bmp, jpeg, ps
png, pcd, psg
....

Kuvatiedoston alussa oleva otsake kertoo talletusformatin

Viiva- ja vektorikuvat

- kuva koodattu objekteina: punainen ympyrä, vihreä monikulmio, musta sinikäyrä
- tarkka zoomaus, vaatii laskentaa

Rasterikuvat

- kuva koodattu pisteinä: esim. 1024 x 1024 pistettä á 24 bittiä
- epätarkka zoomaus, suoraviivainen näyttö

Kuvien käsittely on huomattavasti monimutkaisempaa kuin aikaisemmin mainittujen tietotyyppien. Kuvat voidaan tallettaa moninaisten standardien mukaisesti, ja eri standardit eroavat toisistaan huomattavasti. Merkittäviä piirteitä kuvastandardeissa on niiden pakkaustiheys eli miten pieneen muistitilaan jokin kuva voidaan pakata kuvan laadun siitä merkittävästi huonontumatta. Lähes yhtä tärkeä piirre on levinneisyys. Pienestä tiedoston koosta ei ole hyötyä, jos vastaanottaja ei sitä voi lukea. Pakkaamisen kääntöpuoli on laskennan määrä, mikä joudutaan tekemään kuvaa talletettaessa ja sitä katsottaessa. Ei ole mukavaa, jos yhden kuvan purkamiseen menee 2 minuuttia suoritinaikaa.

Kuvien talletus

Monta kuvastandardia

- pakkaustiheys, yleisyys, siirrettävyys
- laskennan määrä pakatessa ja purettaessa

Kuvatiedoston alussa oleva otsake kertoo talletusformatin

PostScript

```
%!PS-Adobe-2.0
%%Creator: dvips(k) 5.86 Copyright 2005 Joku Yhtiö
%%Title: atari.dvi
%%Pages: 13
%%PageOrder: Ascend
%%BoundingBox: 0 0 596 842
%%DocumentFonts: Palatino-Bold Palatino-Roman Palatino-Italic
%%EndComments
%DVIPSWebPage: (www.abc-yhtio.com)
%DVIPSCommandLine: dvips atari.dvi -o
%DVIPSParameters: dpi=600, compressed
%DVIPSSource:  TeX output 2002.02.11:1208
%%BeginProcSet: texc.pro
```


Kuvatiedoston alussa on otsake, joka kertoo käytettävän formatin ja mahdollisesti myös joitakin kuvaformattiin liittyviä parametreja. Otsakkeen mukaan sitten esimerkiksi verkkoselain voi päätellä, mitä liitospalikkaa tulee käyttää tämän kuvan avaamiseksi.

Kuvien talletus

Monta kuvastandardia

- pakkaustiheys, yleisyys, siirrettävyys
- laskennan määrä pakatessa ja purettaessa

Kuvatiedoston alussa oleva otsake kertoo talletusformatin

Viiva- ja vektorikuvat

- kuva koodattu objekteina: punainen ympyrä, vihreä monikulmio, musta sinikäyrä
- tarkka zoomaus, vaatii laskentaa

Rasterikuvat

- kuva koodattu pisteinä: esim. 1024 x 1024 pistettä á 24 bittiä
- epätarkka zoomaus, suoraviivainen näyttö

PostScript

```
...
% arrowhead
45.000 slw
n 11431 7725 m 11581 7695 l
11431 7665 l 11461 7695 l
11431 7725 l
 cp gs 0.00 setgray ef gr
col0 s
% Polyline
15.000 slw
n 450 7605 m
 450 7785 l gs col0 s gr
...
```


Kuvaformatteja on kahta perustyyppiä. Viiva- ja vektorikuvissa kuva koostuu erilaisista matemaattisesti määritellyistä olioista, esimerkiksi viivoista, ympyröistä tai neliöistä, joilla kullakin on tietty sijainti, koko ja väri. Kuvaa voi tällöin tarkastella miten kaukaa tai läheltä tahansa ja siitä tarkasteluikkunassa näkyvän osan voi aina sitten laskea täsmälleen hyvin tarkkana kuvana. Kuvan käsittely ja tarkastelu vaatii aina aika paljon laskentaa, koska kuva on itse asiassa aina määritelty ainoastaan matemaattisten funktioiden avulla.

Kuvien talletus

Monta kuvastandardia

- pakkaustiheys, yleisyys, siirrettävyys
- laskennan määrä pakatessa ja purettaessa

Kuvatiedoston alussa oleva otsake kertoo talletusformatin

Viiva- ja vektorikuvat

- kuva koodattu objekteina: punainen ympyrä, vihreä monikulmio, musta sinikäyrä
- tarkka zoomaus, vaatii laskentaa

Rasterikuvat

- kuva koodattu pisteinä: esim. 1024 x 1024 pistettä á 24 bittiä
- epätarkka zoomaus, suoraviivainen näyttö

Rasterikuvat koostuvat tietystä joukosta pisteitä ja jokaisen pisteen väritys on sitten määritelty esimerkiksi 24 bitin avulla. Mitään muuta tietoa kuvasta ei ole. Tällaista kuvaa on helppo kääntää ja näyttää. Huonona puolena rasterikuvissa on niiden epätarkkuus zoomattaessa lähelle, jolloin erilliset pikselit tulevat lopulta näkyviin. Esimerkiksi digikameroissa alkuperäinen kuva tallentuu aluksi ns. raakana rasterikuvana kameran kennolle, mistä se kuvan oton jälkeen usein muunnetaan vähemmän vievään jpeg rasterimuotoon. Digitaalisessa zoomauksessa kuvan laatu heikkenee juuri tämän rasteroinnin vuoksi.

Kuvien käsittely

Kuvat yleensä pakattu mahdollisimman vähän tilaa vievään muotoon

- optimoitu tilan, ei laskennan määrän mukaan
- vähän levytilaa, nopea tiedon siirto verkon yli
- tradeoff: tilan määrä vs. kuvan laatu

JPEG 90% vai
JPEG 50% ?

Suoritti
käsittel
- käs
  kok

Erikois
konekä
- käs

JPEG 20%, 36KB

JPEG 80%, 200KB

Kuvien talletuksessa tärkeänä tekijänä on siis kuinka paljon tilaa kuva vie. Tiedoston koolla on merkitystä sekä kuvan tallettamisessa kovalevylle että sen siirtämisessä verkon ylitse koneelta toiselle. Kuvatiedostot ovat kooltaan usein niin suuria, että jo 20% tilansäästö voi olla käyttäjän kannalta merkittävä. Tässä on monta tradeoffia päällekkäin. Kuvan tiivis talletusmuoto säästää tilaa, mutta kuluttaa suoritinaikaa. Tiivis talletus voi myös merkittävästi heikentää kuvan laatua. Pienikin pakkaus voi helposti säästää paljon muistitilaa kuvan laadun siitä silti kärsimättä.

Kuvien käsittely

Kuvat yleensä pakattu mahdollisimman vähän tilaa vievään muotoon

- optimoitu tilan, ei laskennan määrän mukaan
- vähän levytilaa, nopea tiedon siirto verkon yli
- tradeoff: tilan määrä vs. kuvan laatu

Suorittimella ei puhtaita kuvankäsittelykäskyjä, käsittely aliohjelmilla

- käskykannassa voi olla kuvia varten optimoituja kokonaislukukäskyjä

Erikoissuorittimilla (näyttökorteilla) voi olla omia konekäskyjä kuvien käsittelyyn

- käskykanta muutenkin optimoitu (liikkuvan) kuvan käsittelyyn

Multimedia Extensions

Intelin MMX-käskyt 2D-grafiikkaa varten.

Tulkitse 64 bitin rekisterit siten, että ne sisältävät 8 kpl 8-bittistä kuva-alkiota.

MMX-käskyt laskevat kaikki 8 kuva-alkiota yhdellä kertaa.

Normaaleissa suorittimissa ei ole erikseen konekäskyjä kuvien käsittelyyn. Kuvien manipulointi tapahtuu esimerkiksi kokonaislukukäskyillä kuvien talletettua esitysmuotoa muokkaamalla. Kuitenkin esimerkiksi Intelin arkkitehtuureissa on jo kauan ollut erityisiä MMX-käskyjä, jotka on nimenomaan suunniteltu kuvien tallettamiseen käytettävien kokonaislukujoukkojen nopeaan muokkaamiseen.

Kuvien käsittely

Kuvat yleensä pakattu mahdollisimman vähän tilaa vievään muotoon

- optimoitu tilan, ei laskennan määrän mukaan
- vähän levytilaa, nopea tiedon siirto verkon yli
- tradeoff: tilan määrä vs. kuvan laatu

Suorittimella ei puhtaita kuvankäsittelykäskyjä, käsittely aliohjelmilla

- käskykannassa voi olla kuvia varten optimoituja kokonaislukukäskyjä

Erikoissuorittimilla (näyttökorteilla) voi olla omia konekäskyjä kuvien käsittelyyn

- käskykanta muutenkin optimoitu (liikkuvan) kuvan käsittelyyn

Näyttökorteilla olevilla grafiikkasuorittimilla tilanne on vähän erilainen, koska kyseiset suorittimet on nimen omaan optimoitu grafiikan laskentaan. Niissä on yleensä konekäskyjä jonkin tietyn grafiikkastandardin mukaiseen esitystapaan. Yleissuorittimella suorittava laiteajuri sitten muuttaa kaikki kuvat ja kuvienkäsittelypyynnöt näyttökortin grafiikkasuorittimelle sopivaksi.

Videokuva

Talletus kuva kerrallaan, esim. 25 kuvaa/sek

- 1 sekunti hyvälaatuista videokuvaa 20 MB

Talletus tilaa säästäen, eri keinoja yhdistellen

- kun seuraava kuva poikkeaa edellisestä vain vähän, talleta vain muutokset edelliseen
- talleta vain esim. joka viides kuva kokonaan, ja interpoloi (laske) näyttämisen yhteydessä välissä olevat 4 kuvaa
  - vaatii paljon laskentatehoa

Paljon eri standardeja

- MPEG, AVI, MOV, INDEO, FLI, GL, DVD, ...

Ei omia konekäskyjä, manipulointi aliohjelmilla

Erikoissuorittimissa (näytönohjaimissa) erikoistunut käskykanta

Liikkuva kuva on tietenkin monta kertaluokkaa vaativampi tiedon esitysmuoto. Yksinkertainen talletus ei yleensä ole mahdollista sen vaatiman suuren muistitilan vuoksi. Simppelissä tapauksessa videokuvassa on esimerkiksi 25 kuvaa sekunnissa ja kuvan koko vaihtelee tietenkin näytön koosta riippuen. Jos jokainen kuva talletetaan yksitellen, muistitilaa tarvitaan videolle esimerkiksi 20 MB sekunnissa, ja elokuvaa tai muuta korkeatasoista näyttöä varten vielä enemmän.

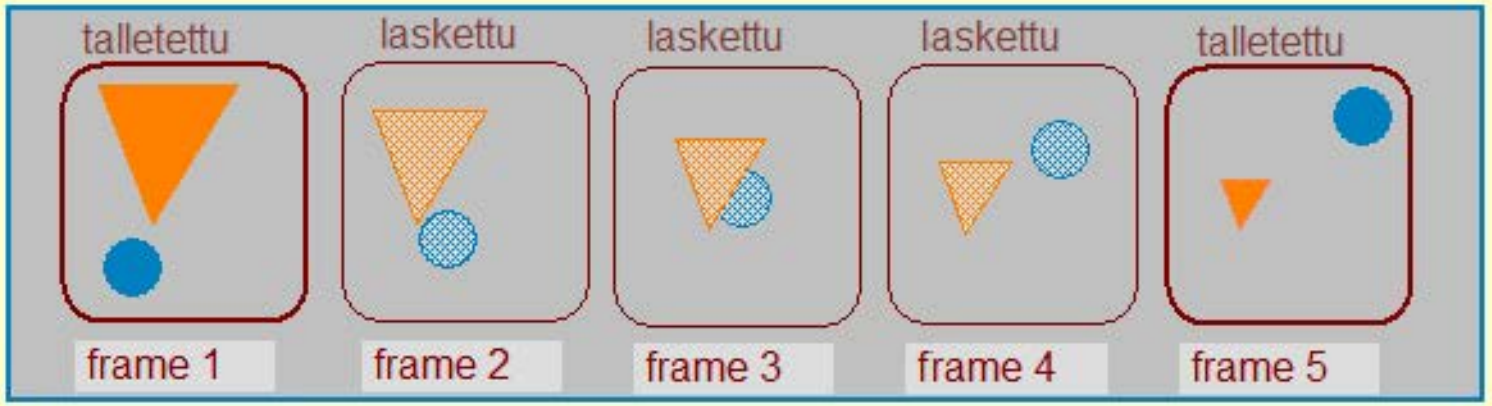
Videokuva

Talletus kuva kerrallaan, esim. 25 kuvaa/sek

- 1 sekunti hyvälaatuista videokuvaa 20 MB

Talletus tilaa säästäen, eri keinoja yhdistellen

- kun seuraava kuva poikkeaa edellisestä vain vähän, talleta vain muutokset edelliseen
- talleta vain esim. joka viides kuva kokonaan, ja interpoloi (laske) näyttämisen yhteydessä välissä olevat 4 kuvaa
  - vaatii paljon laskentatehoa



Yleisesti ottaen liikkuva kuva talletetaan aina usealla eri tavalla optimoidussa muodossa, mikä sitten pitää purkaa näyttöhetkellä. Tyypillinen optimointiperiaate on inkrementaalinen talletus, jossa vain kuvan muuttuneet osiot talletetaan. Animoinneissa tämä tarkoittaa, että tausta pysyy samana, mutta etualalla olevat kohteet muuttuvat. Toinen hyvä periaate on interpolointi, jossa talletetaan vain tärkeät ruudut sieltä täältä, mutta kuitenkin niin usein, että välissä olevat ruudut voidaan sitten laskea interpoloimalla.

Videokuva

Talletus kuva kerrallaan, esim. 25 kuvaa/sek

- 1 sekunti hyvälaatuista videokuvaa 20 MB

Talletus tilaa säästäen, eri keinoja yhdistellen

- kun seuraava kuva poikkeaa edellisestä vain vähän, talleta vain muutokset edelliseen
- talleta vain esim. joka viides kuva kokonaan, ja interpoloi (laske) näyttämisen yhteydessä välissä olevat 4 kuvaa
  - vaatii paljon laskentatehoa

Paljon eri standardeja

- MPEG, AVI, MOV, INDEO, FLI, GL, DVD, ...

MPEG = Moving Pictures Expert Group
AVI = Audio Visual Interleave

Ei omia konekäskyjä, manipulointi aliohjelmilla

Erikoissuorittimissa (näytönohjaimissa) erikoistunut käskykanta

Liikkuvaan kuvaan on monenlaisia standardeja, mutta niiden tarkempi kuvaus ei kuulu tälle kurssille.

Videokuva

Talletus kuva kerrallaan, esim. 25 kuvaa/sek

- 1 sekunti hyvälaatuista videokuvaa 20 MB

Talletus tilaa säästäen, eri keinoja yhdistellen

- kun seuraava kuva poikkeaa edellisestä vain vähän, talleta vain muutokset edelliseen
- talleta vain esim. joka viides kuva kokonaan, ja interpcloi (laske) näyttämisen yhteydessä välissä olevat 4 kuvaa
  - vaatii paljon laskentatehoa

Paljon eri standardeja

- MPEG, AVI, MOV, INDEO, FLI, GL, DVD, ...

Ei omia konekäskyjä, manipulointi aliohjelmilla

Erikoissuorittimissa (näytönohjaimissa) erikoistunut käskykanta

Tavallisella suorittimella ei ole puhtaasti videokuvan käsittelyyn tarkoitettuja konekäskyjä. Niissä voi kuitenkin olla kokonaislukukäskyjä, joita voidaan käyttää myös videokuvien esitysmuotojen käsittelyyn. Yleisesti ottaen, videokuvan jatkuva käsittely on liian vaativa tehtävä tavalliselle suorittimelle, koska 5 sekunnin videopätkän laskentaan kuluu helposti yli 5 sekuntia laskenta-aikaa, mistä aiheutuu selkeitä käytännön ongelmia.

Videokuva

Talletus kuva kerrallaan, esim. 25 kuvaa/sek

- 1 sekunti hyvälaatuista videokuvaa 20 MB

Talletus tilaa säästäen, eri keinoja yhdistellen

- kun seuraava kuva poikkeaa edellisestä vain vähän, talleta vain muutokset edelliseen
- talleta vain esim. joka viides kuva kokonaan, ja interpoloi (laske) näyttämisen yhteydessä välissä olevat 4 kuvaa
  - vaatii paljon laskentatehoa

Paljon eri standardeja

- MPEG, AVI, MOV, INDEO, FLI, GL, DVD, ...

Ei omia konekäskyjä, manipulointi aliohjelmilla

Erikoissuorittimissa (näytönohjaimissa) erikoistunut käskykanta

Yleinen ratkaisutapa liikkuvan kuvan laskentaan on oma erillinen sitä varten suunniteltu suoritin, joka sitten ei teekään muuta kuin kuvan käsittelyyn liittyvää laskentaa. Tällaiseen suorittimeen voi olla integroitu myös äänen prosessointi, mistä lisää ihan kohta.

2D- ja 3D-näytönohjaimet

Erityisesti kuvan/videokuvan käsittelemiseksi suunniteltu laite, piirikortti

Oma suoritin (GPU, Graphics Processor Unit)

- lukee videodataa (tavallisesta) muistista ja tallettaa kuvan omaan muistiinsa näyttöpuskuriin, josta monitori näyttää sen

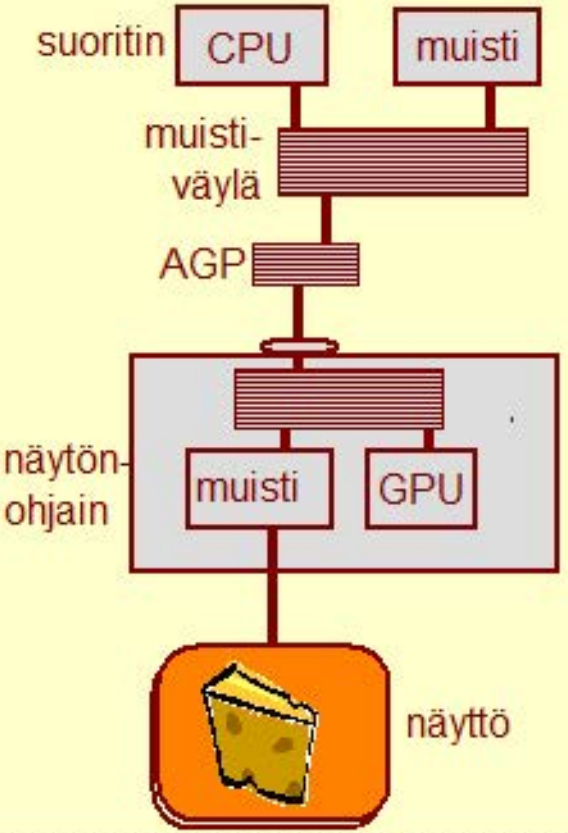
Paljon nopeaa 2-väyläistä (dual-port) muistia

- voi lukea/kirjoittaa samanaikaisesti
- esim. 4-128 MB VideoRAM

Nopea väylä suorittimelle

- esim. AGP (Accelerated Graphics Port)

Voi olla integroitu äänikortin/emolevyn kanssa



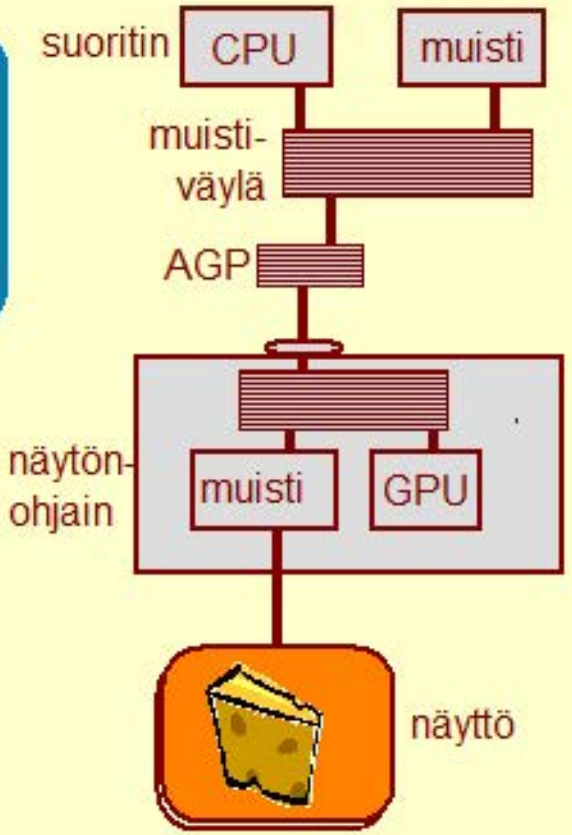
3D-grafiikkaa varten nykyisiin tietokoneisiin sisältyy yleensä joko parempi tai vielä parempi näytönohjain, jonka avulla kaikista hienoista toimintapeleistä saa realististia. Esimerkiksi näiden verkkoluentojen tekemiseen moista korttia ei tarvita, mutta onhan se mukava tietää, että tehoa löytyy tarvittaessa. Näytönohjaimella tarkoitetaan siis näytönohjainkorttia, joka liitetään laitteen väylähierarkiaan muiden laiteohjainkorttien tapaan. Tietokoneen näyttö kiinnitetään nyt tietenkin tähän korttiin sen sijaan, että se olisi kiinnitetty laitteessa muuten ehkä olevaan oletusarvoiseen, heikompitasoiseen perusnäytönohjaimeen.

2D- ja 3D-näytönohjaimet

Erityisesti kuvan/videokuvan käsittelemiseksi suunniteltu laite, piirikortti

Oma suoritin (GPU, Graphics Processor Unit)

- lukee videodataa (tavallisesta) muistista ja tallettaa kuvan omaan muistiinsa näyttöpuskuriin, josta monitori näyttää sen



Paljon nopeaa 2-väyläistä (dual-port) muistia

- voi lukea/kirjoittaa samanaikaisesti
- esim. 4-128 MB VideoRAM

Nopea väylä suorittimelle

- esim. AGP (Accelerated Graphics Port)

Voi olla integroitu äänikortin/emolevyn kanssa

3d-näytönohjaimeen sisältyy oma suoritin, joka voi hyvinkin olla samaa teholuokkaa kuin järjestelmän ns. 'tavallinen' suoritin. GPU on kuitenkin suunniteltu erityisesti grafiikkaoperaatioiden suorittamista varten ja se ei välttämättä ole niin hyvä muiden sovellusten tekemiseen. Vaativaa näyttöä käyttävät sovellukset ovat monimutkaisia suunnitella, koska niissä pitää erikseen miettiä, mitkä asiat tehdään tavallisella suorittimella ja mitkä asiat samanaikaisesti grafiikkasuorittimella.

2D- ja 3D-näytönohjaimet

Erityisesti kuvan/videokuvan käsittelemiseksi suunniteltu laite, piirikortti

Oma suoritin (GPU, Graphics Processor Unit)

- lukee videodataa (tavallisesta) muistista ja tallettaa kuvan omaan muistiinsa näyttöpuskuriin, josta monitori näyttää sen

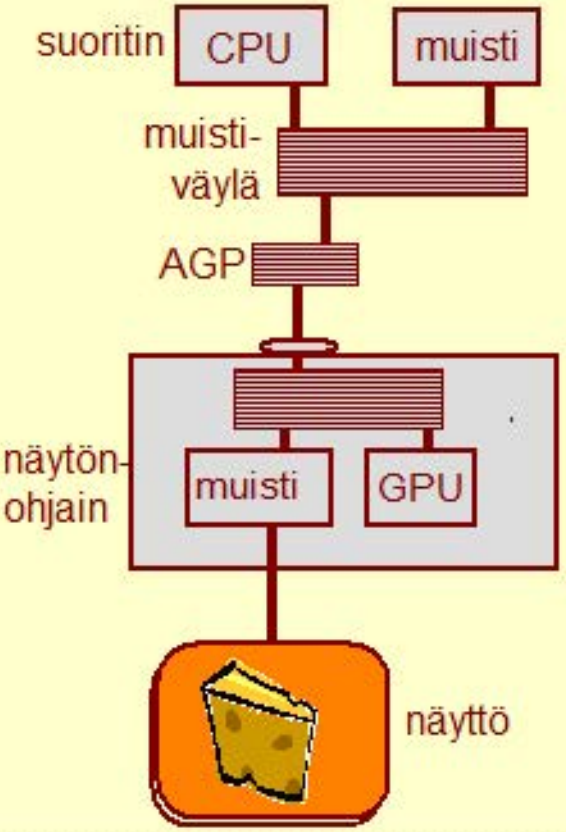
Paljon nopeaa 2-väyläistä (dual-port) muistia

- voi lukea/kirjoittaa samanaikaisesti
- esim. 4-128 MB VideoRAM

Nopea väylä suorittimelle

- esim. AGP (Accelerated Graphics Port)

Voi olla integroitu äänikortin/emolevyn kanssa



3D-ohjaimiin sisältyy paljon nopeata muistia, joka voi myös olla toteutettu 2-väyläisenä. Dual-port tarkoittaa, että muistiin voi toisaalta kirjoittaa yhteen puskuriin uutta näytettävää kuvaa ja toisaalta samaan aikaan näyttölaite voi lukea toisesta puskurista tällä hetkellä näytettävää kuvaa. Tavalliseen muistiin voi yhdellä kertaa olla vain yksi muistiviite (luku tai kirjoitus) kerrallaan.

2D- ja 3D-näytönohjaimet

Erityisesti kuvan/videokuvan käsittelemiseksi suunniteltu laite, piirikortti

Oma suoritin (GPU, Graphics Processor Unit)

- lukee videodataa (tavallisesta) muistista ja tallettaa kuvan omaan muistiinsa näyttöpuskuriin, josta monitori näyttää sen

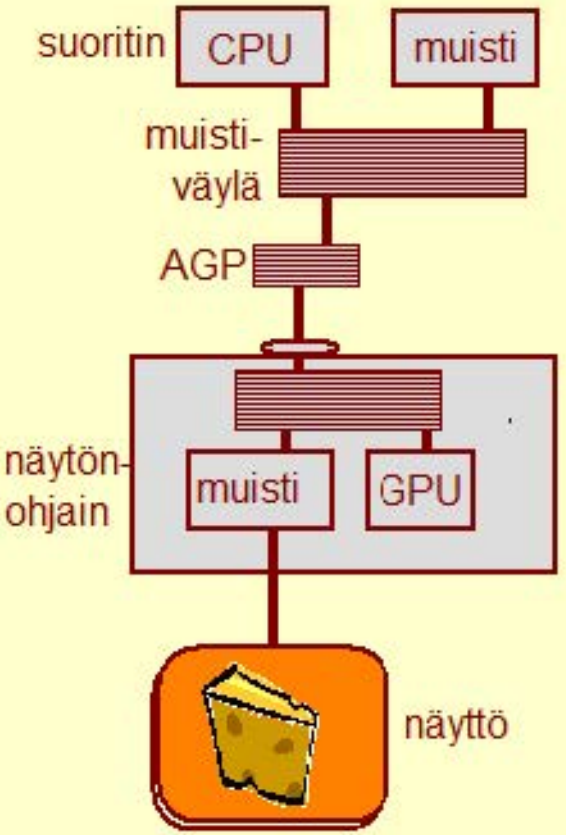
Paljon nopeaa 2-väyläistä (dual-port) muistia

- voi lukea/kirjoittaa samanaikaisesti
- esim. 4-128 MB VideoRAM

Nopea väylä suorittimelle

- esim. AGP (Accelerated Graphics Port)

Voi olla integroitu äänikortin/emolevyn kanssa



3D-ohjaimessa on tavallisia I/O-laitteita nopeampi väylä muistiin, koska grafiikkasovelluksiin vääjäämättä liittyy paljon kopiointia tavallisen muistin ja 3D-kortin oman muistin välillä.

2D- ja 3D-näytönohjaimet

Erityisesti kuvan/videokuvan käsittelemiseksi suunniteltu laite, piirikortti

Oma suoritin (GPU, Graphics Processor Unit)

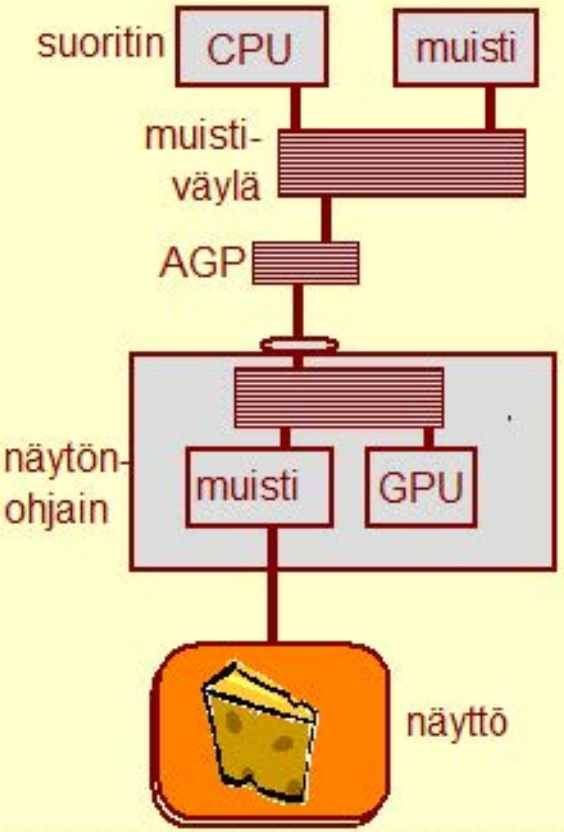
- lukee videodataa (tavallisesta) muistista ja tallettaa kuvan omaan muistiinsa näyttöpuskuriin, josta monitori näyttää sen

Paljon nopeaa 2-väyläistä (dual-port) muistia

- voi lukea/kirjoittaa samanaikaisesti
- esim. 4-128 MB VideoRAM

Nopea väylä suorittimelle

- esim. AGP (Accelerated Graphics Port)



Voi olla integroitu äänikortin/emolevyn kanssa

3D-kortti voi olla integroitu kohta esiteltävän äänikortin kanssa ns. multimediakortiksi tai sitten se voi olla valmiiksi integroituna järjestelmän emolevylle. Vaikka videokortti olisikin valmiiksi integroituna emolevylle, käyttäjä voi tietenkin aina installoida uuden paremman kortin sopivaan väyläpositioon ja sitten tietenkin liittää monitorinsa tähän uuteen supernopeaan ohjaimeen pelikokemuksensa täydellistämiseksi. Toisaalta, järjestelmän muiden laitteiden nopeudet antavat käytännön ylärajan sille, kuinka nopea grafiikkakortti kannattaa kyseiseen järjestelmään liittää.

Äänen talletus

Täydellinen äänidata

- 44100 näytettä/sek, 16b / näyte, 88KB / sek

Syntetisoitu ääni

- MIDI-käskyjä
- "soita nuotti N voimakkuudella V", ei kovin hyvä laululle...

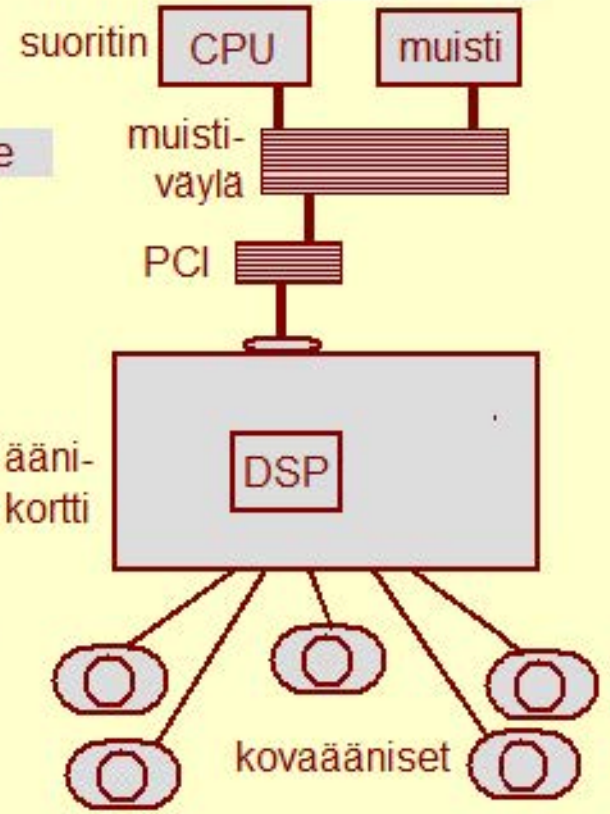
MIDI = Music Instrument Digital Interface

Paljon eri standardeja, eri pakkausmenetelmillä

- MIDI, MP3, WAV, M3U, AU, AIFF, ...

Ei omia konekäskyjä, manipulointi aliohjelmilla

Ääni- tai multimediakortit, joiden erikoissuorittimen käskykanta voi sisältää äänen käsittelyyn tarkoitettuja konekäskyjä



Äänen tallettamisessa on vähän samanlaisia lähestymistapoja kuin kuvankin kohdalla oli. Täydellinen äänidata perustuu sämplaukseen, jossa otetaan 44100 Hz taajuudella 16-bittisiä näytteitä äänestä, jolloin musiikkitiedoston kooksi tulee 88KB/sek. Tämä on aika paljon ja useimmiten liian paljon.

Äänen talletus

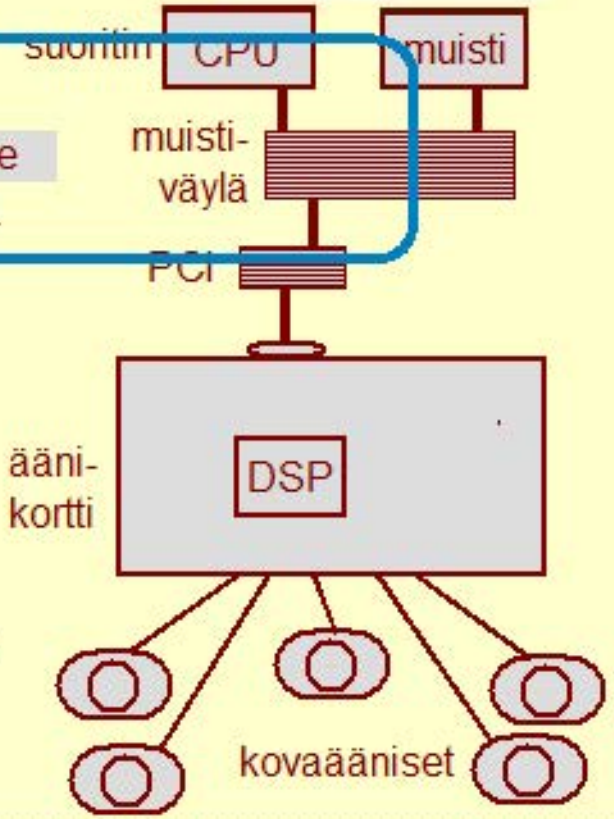
Täydellinen äänidata

- 44100 näytettä/sek, 16b / näyte, 88KB / sek

Syntetisoitu ääni

- MIDI-käskyjä
- "soita nuotti N voimakkuudella V", ei kovin hyvä laululle...

MIDI = Music Instrument Digital Interface



Paljon eri standardeja, eri pakkausmenetelmillä

- MIDI, MP3, WAV, M3U, AU, AIFF, ...

Ei omia konekäskyjä, manipulointi aliohjelmilla

Ääni- tai multimediakortit, joiden erikoissuorittimen käskykanta voi sisältää äänen käsittelyyn tarkoitettuja konekäskyjä

Toinen lähestymistapa muistuttaa kuvien talletusta erilaisten olioiden muodossa. MIDI-standardissa ääni kuvataan tavallaan nuotteina, joilla on tietty äänenkorkeus, tietty pituus ja tietty voimakkuus. Menetelmän etuna on sen selkeys, mutta haittoina erilaisten soitinten määrittelemättömyys. Eri laitteistot toistavat samat MIDI-tiedostot eri tavoin. Lisäksi standardi on suunniteltu instrumenteille, eikä ihmisen lauluäänelle.

Äänen talletus

Täydellinen äänidata

- 44100 näytettä/sek, 16b / näyte, 88KB / sek

Syntetisoitu ääni

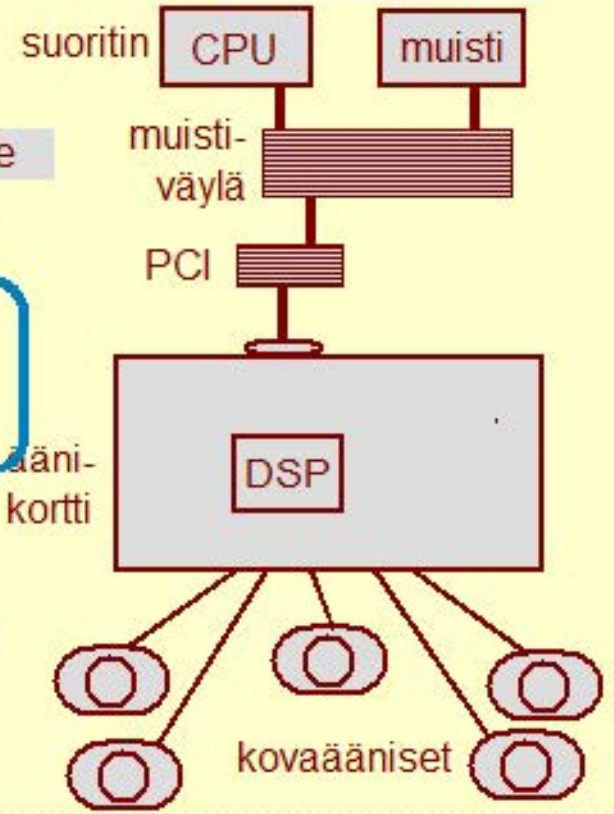
- MIDI-käskyjä MIDI = Music Instrument Digital Interface
- "soita nuotti N voimakkuudella V", ei kovin hyvä laululle...

Paljon eri standardeja, eri pakkausmenetelmillä

- MIDI, MP3, WAV, M3U, AU, AIFF, ...

Ei omia konekäskyjä, manipulointi aliohjelmilla

Ääni- tai multimediakortit, joiden erikoissuorittimen käskykanta voi sisältää äänen käsittelyyn tarkoitettuja konekäskyjä



Äänen talletus- ja pakkausmenetelmiä on kuvatiedon tapaan hyvin useita, ja menetelmissä on suuriakin eroja. Menetelmien tarkempi läpikäynti ei kuitenkaan kuulu tähän kurssiin.

Äänen talletus

Täydellinen äänidata

- 44100 näytettä/sek, 16b / näyte, 88KB / sek

Syntetisoitu ääni

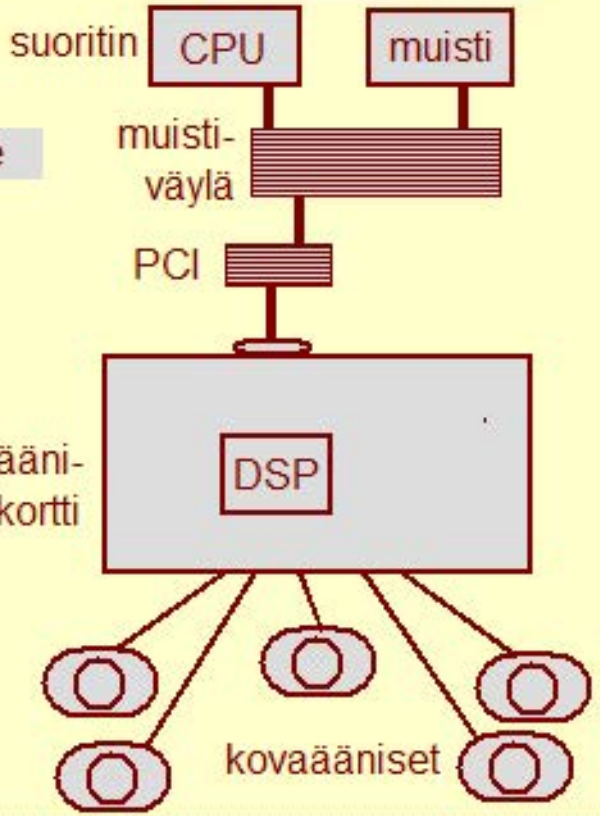
- MIDI-käskyjä MIDI = Music Instrument Digital Interface
- "soita nuotti N voimakkuudella V", ei kovin hyvä laululle...

Paljon eri standardeja, eri pakkausmenetelmillä

- MIDI, MP3, WAV, M3U, AU, AIFF, ...

Ei omia konekäskyjä, manipulointi aliohjelmilla

Ääni- tai multimediakortit, joiden erikoissuorittimen käskykanta voi sisältää äänen käsittelyyn tarkoitettuja konekäskyjä



Äänitiedon käsittelyyn ei ole suorittimessa omia konekäskyjä, vaan äänitietoa käsitellään kuvien tapaan sen esitystapatasolla kokonais- ja liukulukuaritmetiikkaa käyttäen.

Äänen talletus

Täydellinen äänidata

- 44100 näytettä/sek, 16b / näyte, 88KB / sek

Syntetisoitu ääni

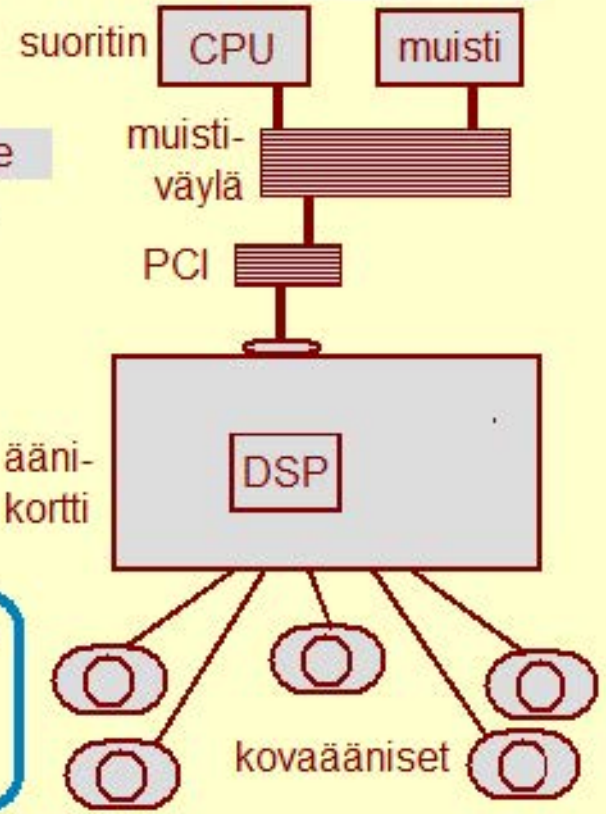
- MIDI-käskyjä
- "soita nuotti N voimakkuudella V", ei kovin hyvä laululle...

MIDI = Music Instrument Digital Interface

Paljon eri standardeja, eri pakkausmenetelmillä

- MIDI, MP3, WAV, M3U, AU, AIFF, ...

Ei omia konekäskyjä, manipulointi aliohjelmilla



Ääni- tai multimediakortit, joiden erikoissuorittimen käskykanta voi sisältää äänen käsittelyyn tarkoitettuja konekäskyjä

Äänikorteilla on DSP eli digitaalinen signaaliprosessori, minkä avulla digitoitu ääni muunnetaan vahvistimelle tai kovaäänisille sopivaan muotoon. Äänikortit on nykyään usein integroitu näytönohjainkortteihin erityisiksi multimediakorteiksi, joihin sitten liitetään sekä näyttö että kauittimet.

Muu tieto: maku, haju, tunto, tunteen palo, ...

Tähtien kirkkaus, hajut, veneen tyyppi, tunteen palo, etc etc

Visualizing and Classifying Odors Using a Similarity Matrix
Liran Carmel, Yehuda Koren and David Harel
Proc. 9th International Symposium on Olfaction and Electronic Nose (ISOEN'02), pp. 141-146 (2003)

Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science, Rehovot 76100, Israel

Abstract: The Lorentzian model is an analytic expression that describes the time response of electronic nose sensors. We show how this model can be utilized to calculate a normalized similarity index between any two measurements. The set of similarity indices is then used for two purposes: visualization of the data, and classification of new samples. The visualization is carried out using graph drawing tools, and the results are shown to bear some desired properties. The classification is done using a majority-decision type algorithm, and is demonstrated to have very low error rate.

Keywords: electronic noses, similarity index, feature extraction, Lorentzian model, graph drawing, visualization, classification

Aikaisemmin esittelimme yleisimmin käytössä olevat tietotyypit, mutta myös kaikki muu tieto voidaan esittää tietokoneessa. Näille tietotyypeille vaan ei ole vielä välttämättä yleisesti käytössä olevaa tiedonesitysmuotoa, jolloin tiedon käsittelijän pitää itse suunnitella sellainen. Jos samaa tietotyyppiä sitten käsitellään muuallakin, voidaan uusi tiedonesitysmuoto lopulta speksata jonkin standardin avulla.

Muu tieto: maku, haju, tunto, tunteen palo, ...

142

Tähtien kirkkaus, hajut, veneen tyyppi, tunteen palo, etc etc

Toteutus sovelluskohtaisesti, ei (vielä) yleisiä standardeja

- koodaa tieto, niin kuin parhaaksi näet
- diskreettiarvoinen data kokonaislukujen avulla
- jatkuva-arvoinen data liukulukujen avulla

47

Ei omia konekäskyjä, manipulointi omilla aliohjelmilla

- suunnittele tiedon koodaus siten, että sen manipulointi olisi helppoa

19

Esimerkki: sormenjäljet

-23.4E4

+5.6E34

6

Uusien tietotyyppien koodaus tehdään (tietenkin!) sovelluskohtaisesti. Jos luokiteltavan datan arvojoukko on diskreetti, niin koodauksessa on hyvä käyttää kokonaislukuja. Esimerkiksi junien luokittelussa veturityypin arvojoukko on hyvin rajallinen ja pieni kokonaislukujen joukko riittänee sen kuvaamiseen. Jatkuva-arvoinen data on parasta kuvata liukulukuina. Tällöin kahden lähellä olevan luokitteluarvon väliltä löytyy aina lisää lukuarvoja. Liukuluvut sopivat myös tilanteeseen, jossa luokitteluarvojen suuruusluokat vaihtelevat paljon. Esimerkiksi, tunteen palon voimakkuus voisi olla järkevää luokitella liulukujen avulla.

Muu tieto: maku, haju, tunto, tunteen palo, ...

Tähtien kirkkaus, hajut, veneen tyyppi, tunteen palo, etc etc

Toteutus sovelluskohtaisesti, ei (vielä) yleisiä standardeja

- koodaa tieto, niin kuin parhaaksi näet
- diskreettiarvoinen data kokonaislukujen avulla
- jatkuva-arvoinen data liukulukujen avulla

Ei omia konekäskyjä, manipulointi omilla aliohjelmilla

- suunnittele tiedon koodaus siten, että sen manipulointi olisi helppoa

Esimerkki: sormenjäljet

Tällaisen uuden tai monimutkaisen tietotyypin käsittelyyn ei tietenkään ole omia konekäskyjä, vaan tällaista tietoa käsitellään aina sen esitystapatasolla tavallisilla kokonaisluku- ja liukulukukäskyillä. Itse asiassa jo tietotyypin esitysmuotoa suunniteltaessa on tietenkin jo otettu huomioon, millä tavoin ja minkälaisessa laitteistossa tietoa tullaan käsittelemään. Esimerkiksi, avaruuteen lähetettävän luotaimen käsittelemä tieto pitää koodata sillä tavoin, että sitä voidaan hyvin tehokkaasti muokata mukanaolevalla laitteistolla ja että se voidaan nopeasti lähettää maahan käytettävissä olevalla järjestelmällä.

Muu tieto: maku, haju, tunto, tunteen palo, ...

FBI Fingerprint Format

Also Known As: FBI WSQ

Type Bitmap
Colors 8-bit grayscale
Compression Wavelet Scalar Quantization
Maximum Image Size 64Kx64K
Multiple Images Per File No
Numerical Format Big-endian
Originator U.S. Federal Bureau of Investigation
Platform All
Supporting Applications Many
See Also None

....
How much time it would take to visibly inspect each card for a match, even by a large army of trained researchers

Esimerkki: sormenjäljet

http://www.oreilly.com/www/centers/gff/formats/fbi/index.htm (19.1.2005)

Esimerkkinä vähän erikoisemmasta tietotyypistä on FBI:n sormenjälkistandardi. Mitä vain tietoa voidaan esittää koneessa ja yleisemmässä käytössä oleva tieto karnattaa standardoida käytännön syistä. Aikaisemmin mainittuja hajuja ei ole vielä standardoitu, mutta ei liene kaukana aika, jolloin tietokonepelaaja voi virtuaaliaseen ampumisesta johtuvan käden tärinän lisäksi hän myös haistaa virtuaaalivastustajan pelon tiukan paikan tuller.

Tiedon esitysmuodot

Tiedon tyypit ja tiedon esitys laitteistossa

Lukujärjestelmät

Kokonaisluvut
Liukuluvut
Merkit, merkkijonot
Totuusarvot
Kuvat
Äänet
Mikä tahansa muu tieto, esimerkiksi hajut

Olemme nyt käyneet läpi erilaisten tiedonesitystapojen luokittelun tietokonejärjestelmässä. Parhaiten tiedon käsittelyyn sopivat sellaiset tietotyypit, joita varten on omia konekäskyjä suorittimella. Kaikki muu tieto pitää koodata laitteiston ymmärtämään muotoon, käyttäen laitteiston toteuttamia tietotyyppejä, useimmiten kokonaislukuja ja liukulukuja. Luennon alussa meillä oli myös lyhyt kertaus erilaisista lukujärjestelmistä ja esitystapojen muuttamisesta lukujärjestelmien välillä. Konekäskyt tietotyyppinä ja rakenteellinen tieto jäivät käsittelemättä, mutta nämä puutteet paikataan seuraavalla luennolla.