1 Bitar, bytes och sånt

Detta kapitel får du i huvudsak läsa på egen hand. Jag har koncentrerat mig på Linux-systemet även om det mesta gäller även för andra system.

1.1 Binära tal

Datorer arbetar normalt med tal uttryckta i basen två och man talar därför om binära tal. En bit är en binär siffra (BInary Digit) dvs. ett av talen noll eller ett. Det binära talsystemet är ett positionssystem precis som vårt decimala system, som ju har basen tio. En siffras vikt (betydelse) avgörs således av dess position i talet. Om vi betraktar det decimala talet 347.2 så kan det ju skrivas

$$3 \times 10^2 + 4 \times 10^1 + 7 \times 10^0 + 2 \times 10^{-1}$$

så att vikten för en siffra i position p (om entalssiffran har position noll, tiotalssiffran position ett etc.) är 10^p. Om allmänt basen är β har vi β olika siffror, 0, 1, ..., β-1. En siffras vikt är β^p där p är siffrans position. Låt oss använda detta för att uttrycka det binära talet 1101.101 i decimal form. Vi skriver 1101.101_2 för att betona att talet är givet i basen två, 1101.101 tolkat i basen tio är ju ett helt annat tal.

$$1101.101_2 =$$

$$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 13.625$$

Skilj mellan siffror som är talen $0, 1, \ldots, \beta$-1 och tal som är sammansättningar av siffror. Siffror är tal, men tal är inte alltid siffror. Det decimala talet 145.77 är ingen siffra.
Åtta bitar bildar en byte och fyra bitar bildar en nibble. Ofta har man många bytes och man använder då prefix som i fysik och andra naturvetenskaper. I fysik står kilo för 1000, som i 1 kg (ett kilogram), 1 km (en kilometer) etc. Mega står för 10^6, 1 MW (en megawatt).

Dessa prefix har dock en liten annan betydelse i datorsammanhang. kilo är då $2^{10} = 1024$, mega $= 2^{20} = 1048576$, etc. Notera att 1024 är ungefär 1000 så att om vi vill göra överslagsberäkningar kan vi tänka i de vanliga tiopotenserna.

En Mbyte är till exempel 2^{20} bytes som är ungefär 10^6 bytes. Giga står för nästa tusental, $2^{30} \approx 10^9$. I vissa sammanhang finns de användning för mer än "Giga". Nästa tusental är Tera (och inte Terra som många skriver). Man skriver inte heller mb/s som vissa bredbandsannonsörer (milli-bit/s).
För att slippa tvetydigheten finns alternativa benämningar för tvåpotenserna.

namn	förkortning	värde
kibi	Ki	2^{10}
mebi	Mi	2^{20}
gibi	Gi	2^{30}
tebi	Ti	2^{40}
pebi	Pi	2^{50}
exbi	Ei	2^{60}
zebi	Zi	2^{70}
yobi	Yi	2^{80}

mebi = Mega binary etc. Så 1 Mi byte är 2^{20} byte, med andra ord. Dessa beteckningar verkar inte ha slagit igenom.

Stora binära tal tar stor plats att skriva ut. Ett tal på binär form kräver ungefär 3.3 så många siffror som talet skrivet i decimal form (eftersom $\log_2 10^m = m \log_2 10 = m \cdot 3.3219\ldots$). För att inte behöva skriva så mycket brukar man, i datorsammanhang, använda två andra talsystem, det oktala med basen åtta och det

hexadecimala med basen sexton. Det är enkelt att konvertera mellan dessa olika talsystem (eftersom baserna är tvåpotenser). I det oktala talsystemet har vi siffrorna noll till och med sju och när vi använder det hexadecimala talsystemet får vi hitta på sex nya siffror. Normalt använder man bokstäverna a - f för de nya siffrorna. Här följer en liten tabell som visar sambandet mellan de hexadecimala siffrorna och några andra talsystem:

Hexadecimalt	Decimalt	Binärt	Oktalt
0	0	0000	0
1	1	0001	1
2	2	0010	2
3	3	0011	3
4	4	0100	4
5	5	0101	5
6	6	0110	6
7	7	0111	7
8	8	1000	10
9	9	1001	11
a	10	1010	12
b	11	1011	13
c	12	1100	14
d	13	1101	15
e	14	1110	16
f	15	1111	17

Siffrorna, a-f, skrivs i bland med versaler.
Här följer ett exempel: $2bf.8$ hexadecimalt är lika med

$$2 \times 16^2 + 11 \times 16^1 + 15 \times 16^0 + 8 \times 16^{-1} = 703.5 \text{ decimalt.}$$

Det är enkelt att konvertera mellan det hexadecimala systemet och det binära. Låt oss konvertera talet ovan till det binära systemet. Vi ersätter då varje hexadecimal siffra med motsvarande binära tal (enligt tabellen ovan) och får 0010 1011 1111.1000 (där jag har skrivit ut fyra bitar för varje hexadecimal siffra) eller med onödiga nollor avlägsnade 1010111111.1. Låt oss nu anta att vi har ett tal på binär form och måste skriva det på hexadecimal form. Vi grupperar då bitarna fyra och fyra (till vänster respektive till höger om decimalpunkten), dvs. 0010 1011 1111.1000 i exemplet, därefter ersätter vi grupperna med motsvarande hexadecimala siffror och får 2bf.8. Det fungerar analogt att gå mellan det oktala- och det binära talsystemet fastän man får använda grupper om tre bitar. Så 1010111111.1 blir oktalt 1277.4 (grupperingen blir 1 010 111 111.100).

1.2 Teckenkoder

Även tecken (bokstäver, skiljetecken, tecknen 0-9 etc.) lagras med bitar (teckenkoder). Det finns flera standarder och den standard vi använde fram till ht 2009, på Linux-datorerna, heter ISO 8859-1, och den utnyttjar en byte för varje tecken. Detta ger upphov till $2^8 = 256$ olika kombinationer (binära tal) vilket räcker för många teckenuppsättningar, men inte för kinesiska t.ex. där man får använda två bytes ($2^{16} = 65536$ kombinationer). Här följer en tabell över några teckenkoder

Tecken	Teckenkod (hexadecimalt)
a	61
b	62
A	41
B	42
mellanslag	20
. (punkt)	2e
1	31

Notera att det är skillnad på talet ett och en textsträng som innehåller tecknet ett. ISO-standarden är en utvidgning av en äldre sjubitars-standard, ASCII (American Standard Code for Information Interchange), som definierar det engelska alfabetet, siffror, skiljetecken och en del specialtecken.

Under unix definieras radslut, i en textfil, av tecknet “newline” som har teckenkoden 10 (decimalt). I C-program skrivs tecknet `\n`. I ASCII-tabellen benämns tecknet `LF`, “linefeed”. På gamla terminaler (som skrev på papper) gjorde LF att valsen vred sig till nästa rad (som på en skrivmaskin).

“Carriage return”, `CR` i ASCII-tabellen, decimalt 13, gjorde att skrivhuvudet återgick till utgångspositionen.

På Windows-maskiner skrivs radslut med tvåteckenskombinationen `CR`, `LF`, vilket gör att man, under unix, kommer att se ett `CR`-tecken i slutet på varje rad (om man överför en Windowsfil till unixsystemet). Det kan se ut som:

```
program u14^M
  type nod^M
    character::data^M
    type(nod), pointer :: vaenster, hoeger^M
  end type nod^M
^M
contains^M
```

`^M` står för Ctrl-M (man trycker ned Ctrl-tangenten samtidigt som man trycker på M-tangenten). Unix-kommandon `dos2unix` samt `unix2dos` fixar konverteringen.

Om man vill avbryta ett program går det ofta bra att skriva tecknet Ctrl-C (decimalt 3). Ett annat specialtecken som kan lagras i en fil, är `HT`, “Horizontal Tab”, decimalt 9. Ett sådant tecken skrivs när man trycker på Tab-tangenten (eller Ctrl-I, `^I`).

En unix-fil avslutas *inte* med något speciellt filslutstecken. I stället detekteras filslut (EOF, end-of-file) via längd-information som finns lagrad i filsystemet. Filinformation är lagrad i en datstruktur som bland annat innehåller information om vem som äger filen, filrättigheter, modifieringsdatum, längd i bytes och en pekare till de ställen på disken där filens innehåll lagras.

2010 bytte vi från ISO 8859-1 till UTF-8 (Unicode Transformation Format) som också är en ISO-standard. UTF utgör en help grupp av standarder och en utmärkande skillnad mot den tidigare standarden, är att UTF-8 använder ett varierande antal bytes, en till fyra, för att lagra ett tecken. ASCII-teckenuppsättningen utgör en delmängd av UTF-8 och lagras med en byte per tecken. ÅÄÖåäö lagras dock med två bytes vardera. Detta får vissa konsekvenser när man listar en gammal fil på det nya systemet. Säg att vi har skapar filen `gammal` (med ISO 8859-1) som innehåller den enda raden:

```
De tre sista bokstäverna i svenska alfabetet är åäö.
```

Om vi listar filen i “pagern” `less` (en pager är ett program med vars hjälp man kan lista en fil, en skärmsida i taget) så får man utskriften (under UTF-8):

```
De tre sista bokst<E4>verna i svenska alfabetet <E4>r <E5><E4><F6>.
```

Av detta kan man sluta sig till att teckenkoderna för t.ex. `å` är hexadecimalt e5 i ISO 8859-1. Gör vi det omvända, listar en fil i UTF-8 på ett ISO 8859-1-system får vi:

```
De tre sista bokstÃ¤verna i svenska alfabetet Ã¤r Ã¥Ã¤Ã¶.
```

Så å, ö och ö tar två bytes vardera (man kan ta reda på teckenkodningen om man vill, men vi avstår från det).

Linux-kommandot `iconv` kan konvertera mellan olika teckenuppsättningar.

Nu några ord om hur mycket lagringsutrymme text kräver. Antag att vi vill lagra en pocketbok (utan illustrationer och på engelska) som en fil. Låt oss anta att boken består av 220 sidor, där varje sida innehåller 40 rader och en typisk rad har 60 tecken (bokstäver, skiljetecken). Antalet tecken per rad varierar, bland annat därför att böcker typsätts med ett proportionellt typsnitt (olika tecken upptar olika bredd på sidan; bokstaven m tar mer plats än bokstaven n till exempel).

Det blir ungefär $220 \times 40 \times 60 = 528000$ tecken som kräver ungefär 0.5 Mibyte för att lagras. En vanlig CD-skiva kan lagra minst 650 Mibyte, så ungefär 1300 böcker En vanlig hårddisk kanske rymmer 500 Gbyte, mer än 1000 000 böcker.
Det som tar plats när det gäller media är bilder, video och ljud. Text tar inte så mycket plats.

1.3 Datorer och programspråk

En dator är en maskin som kan utföra instruktioner av aritmetisk-logisk karaktär. Utmärkande för datorer är att de är snabba och lätt kan fås att upprepa sekvenser av instruktioner. Datorer är också utrustade med minnen för att lagra temporära data och program, men de har även minnen för långtidsförvaring.

En dators fysiska komponenter, "elektroniken", kallas hårdvara (eng. hardware). Instruktionerna samlas i program. Program, i största allmänhet, kallas mjukvara (eng. software).

Det finns hundratals programspråk, några som jag själv använder är Fortran, C, C++ och Java. MATLAB är inget programspråk, utan snarare en hel miljö för beräkningar. Man kan dock skriva program i MATLAB med MATLABs eget programspråk.

Datorn "förstår" bara ett språk, som man brukar kallas datorns maskinspråk. Olika datorfabrikat (Intel, Sun etc.) har olika maskinspråk. Detta gör att program skrivna i maskinspråk inte går att flytta mellan olika typer av system. Utmärkande för maskinspråket är att det är väldigt enkelt och är nära knutet till hårdvaran och att allt uttrycks med hjälp av binära tal (ettor och nollor). En del av programmet kanske ser ut så här (här skrivet i hexadecimal form):

```
74657874
002e7379
6d746162
002e7374
72746162
002e636f
```

Människor brukar aldrig programmera i maskinspråk eftersom det är så arbetskrävande. Det är också svårt att ändra i ett sådant program. I bland programmerar man i det så kallade assemblerspråket, som ligger strax "ovanför" maskinspråket. Man kan då använda namn på operationer i stället för att ange hexadecimala tal.

Följande assemblerkod adderar två tal, ungefär summa = a + b:

```
    ldd     [%o0], %f2
    ldd     [%o1], %f4
    faddd   %f2, %f4, %f2
    std     %f2, [%o2]
```

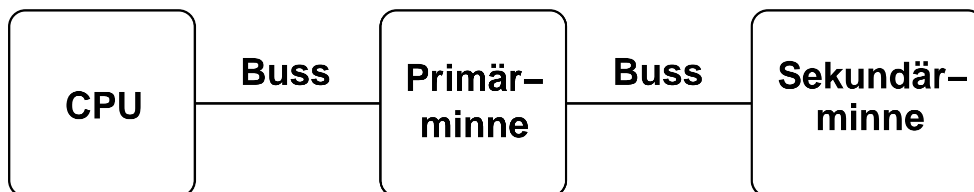
`faddd` beräknar summan. Vi kommer att återvända till detta exempel och då förstå lite bättre vad som händer. Det är tämligen jobbigt att programmera i assembler också, en enkel summaberäkning gav ju upphov till fyra assembler-rader. Så, assemblerprogrammering används normalt t.ex. vid vissa elektroniktillämpningar där det

är stora krav på snabbhet och kontroll.

För att det skall vara bekvämt att programmera och för att programmen skall bli portabla programmerar man normalt i så kallade högnivåspråk, som Fortran, C och C++. Program skrivna i dessa språk översätts sedan till maskinspråket. Översättningen görs av speciella program, så kallade kompilatorer (eng. compiler), man säger att man kompilerar. När översättningen är gjort kan man så "köra" programmet, man säger normalt att man exekverar programmet (eng. execute).

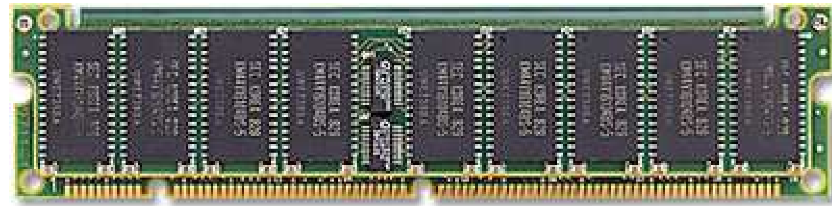
När det gäller vissa språk, som t.ex. en del av MATLAB-programmen, så interpreteras koden i stället. Detta görs av en så kallad interpretator. Programmet översätts då lite eftersom, och inte som när man kompilerar där hela programmet görs om till maskinkod på en gång. En vanlig analogi är den mellan en simultantolk (interpretatorn) som översätter mening för mening, och en översättare (kompilatorn) som översätter en hel bok. Java intar lite av en mellanställning. Man kompilerar först Javaprogrammet till så kallad bytekod (en enkel kod som inte är bunden till någon speciell dator; bytekoden är portabel med andra ord). Programmet exekveras sedan genom att bytekoden interpreteras (det kan också vara så att bytekoden kompileras till maskinkod som sedan exekveras).

Det skall också sägas att MATLAB har en "JIT-Accelerator" (JIT = Just-in Time) som översätter (en del) MATLAB-konstruktioner till maskinspråk.
Här kommer nu några rader om datorer varefter vi fortsätter med programspråken.
En enkel modell av en dator visas i följande bild (jag har inte ritat ut bildskärm, tangentbord osv.).



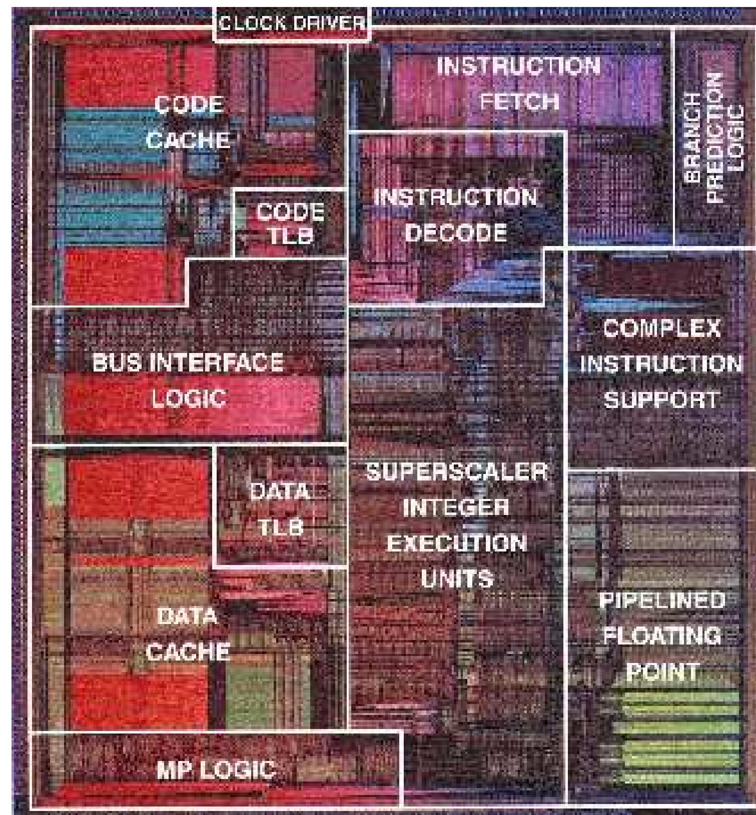
Sekundärminnet består i allmänhet av en eller flera hård-diskar (eller SSD, Solid-State Disk, en disk utan rörliga delar). En sådan, med avtaget lock, kan se ut som på bilden nedan (längden är 10-15 cm i verkligheten):

Disken består av en eller flera metallskivor som är belagda med magnetiserbart material. Skivorna roterar med några tusen varv per minut. Det går att läsa data från och skriva data på skivan. Detta arbete utförs med hjälp av armen (se bilden) som kan röra sig över skivans yta. I armens ände finns ett litet läs- och skrivhuvud. Dina filer lagras på diskar och data finns kvar även om du loggar ut från datorn. Det går att skriva över data som redan finns på skivan så om du ändrar i en fil så finns inte den gamla versionen kvar. En typisk hård-disk kan lagra 500 Gbyte - 1 Tbyte.
Primärminnet, ofta säger man bara minnet, består av elektroniska kretsar (inga rörliga delar) och det kan vara uppbyggt av kretsar som den på bilden nedan (längden är drygt 5 cm i verkligheten):



Primärminnet används för temporär lagring och minnesinnehållet byts normalt ut många gånger per sekund. Om man slår av datorn försvinner minnesinnehållet. Vanlig minnesstorlek är 1 Gbyte-16 Gbyte (åtminstone på hemdatorer).
CPUn (Central Processing Unit), processorn, är "hjärnan" i datorn. Det är här alla beräkningar sker, till exempel. Följande bild visar ovansidan och undersidan (med alla anslutningsstiften) av en CPU till en Sundator (sidlängden är drygt 5 cm i verkligheten):

Det mesta av bilden ovan är förpackning. Själva chipytan (kiselskivan) är mindre (10 mm sida kanske). Här är en uppförstorad bild av kiselskivan från en av Intels Pentiumprocessorer:



Bilden visar också funktionen hos olika delar av CPU-chipet. Vi återkommer strax till en del av dessa. En modern CPU innehåller flera tiotals miljoner transistorer.
De olika delarna av datorn, CPU, primär- och sekundärminne är ihopkopplade av så kallade bussar (bus på engelska). Du kan tänka dig en buss som ett knippe elektriska ledningar, 32 stycken kanske. Eftersom vi har parallella ledningar kan vi överföra data (bitar) snabbare än om vi har en enda ledningen där vi måste skicka bitarna efter varandra (seriellt, som man säger).
En dator är tämligen oanvändbar utan det stora programsystem som kallas operativsystem (brukar förkortas OS). Det OS vi använder på mattedatorerna är ett GNU/Linux-system från Red Hat. PC-datorer kör ofta något av Microsofts OS, t.ex. Windows XP.

Ett operativsystem har många uppgifter, t.ex.:

- Det förmedlar kontakten mellan användaren och hårdvaran.
- Det sköter filhanteringen och ser till att filer lagras på och hämtas från disken.
- Det fördelar resurser (tid och minne) mellan olika program som går i datorn.
- Det skyddar data mot otillåten åtkomst.
- Det ser till att vi kan ansluta till Internet.

Låt oss nu väldigt stora drag studera vad som händer när vi vill köra ett program i ett unix-system. Vi antar att programmet är kompilerat och att den så kallade exekverbara filen (maskinkoden) ligger i en fil på disken. I unix kör man program genom att skriva namnet på den exekverbara filen. När Du, till exempel, ger kommandot `ls` för att lista namnen på dina filer så körs ett kompilerat C-program. Den exekverbara filen heter `ls` och ligger i katalogen `/bin` i filsystemet.

När du skriver namnet på den exekverbara filen ser operativsystemet till att läsa in (eventuellt i omgångar) den exekverbara filen och placera koden (instruktionerna) i primärminnet. Därefter låter operativsystemet CPUn börja exekvera de instruktioner som ligger i minnet. CPUn kommer då att hämta instruktioner från minnet. Det görs av "Instruction Fetch"-enheten i processorn (se bilden ovan). När instruktionen har hämtats avkodas den (av Instruction Decode), dvs. CPUn tar reda på vilken typ av instruktion det är (addition, logiskt val etc).

Om det är en additionsoperation av decimaltal som skall utföras kommer operationen att exekveras av FPUn (Floating Point Unit). Om det är heltal som skall adderas tar instruktionen hand om av "integer execution units" (som det står i bilden). De övriga delarna i bilden tar vi inte upp i denna kurs. "Floating point numbers" är tal med decimalpunkt och eventuell exponentdel.

Man kan nu undra hur CPUn får tag i operanderna (de tal som skall adderas). Jo, primärminnet lagrar inte bara instruktionerna utan det kan även lagra data (tal, tecken etc.) som programmet skall arbeta med.

Tänk dig minnet som en uppsättning numrerade fack. Man brukar dock säga adress i stället för nummer. Varje fack svarar normalt mot en byte (minnet kan adresseras på byte-nivå). För att lagra ett heltal tar vi normalt fyra bytes och ett decimaltal kräver oftast åtta bytes. CPUn kan nu hämta ett tal från primärminnet och lagra det i ett så kallat register (minnesutrymme) i CPUn. Det finns inte speciellt många register, kanske 32-64 stycken. När så summan är beräknad kan resultatet lagras tillbaks till primärminnet. Vi är nu mogna att se på assembler-raderna igen:

```
ldd     [%o0], %f2
ldd     [%o1], %f4
faddd   %f2, %f4, %f2
std     %f2, [%o2]
```

`ldd` (load double) är en instruktion som hämtar åtta bytes från minnet. Minnesadressen som talet skall hämtas från anges, i exemplet, av innehållet i det register som heter `%o0` (konstigt namn) och talet placeras i register `%f2` som finns i FPUn. Därefter hämtas den andra termen och läggs i register `%f4`. FPUn adderar sedan ihop talen (`faddd` står för floating point add double) och summan läggs i `%f2`. Den sista instruktionen, `std` (store double) lagrar så värdet i den adress som anges av register `%o2`.

1.4 Intern representation av tal och tecken

1.4.1 Heltal

Vi skiljer mellan heltal (eng. integer) och flyttal (ungefär reella tal). I själva verket kan vi representera en delmängd av heltalen och en delmängd av de rationella talen. Det är vanlig att ett heltal lagras med fyra bytes (fast på moderna 64-bitars maskiner kan åtta bytes användas).

Detta ger oss

$$2^{4 \cdot 8} = 2^{32} = 4294967296 \text{ kombinationer.}$$

Vi vill normalt ha negativa tal samt noll vilket torde ge ett största heltal omkring 2^{31} (dvs. hälften av antalet tänkbara kombinationer). På mattemaskinerna (AMD64) gäller att vi kan lagra heltalen

$$-2^{31} = -2147483648, \ldots, -1, 0, 1, \ldots, 2147483647 = 2^{31} - 1$$

Varför får vi detta osymmetriska intervall? Jo, ett symmetriskt intervall med *en* nolla ger ett udda antal positiva (negativa) tal (men 2^{31} är ju inte udda).

För att hantera tal av olika tecken används det sk tvåkomplementsystemet på våra datorer. Låt oss anta att vi har fyra bitar för ett heltal (och inte fyra bytes) och att vi vill lagra talet -5 (0101_2). Tvåkomplementet

får vi om vi byter nollor mot ettor och vice versa samt adderar ett. Så 0101 blir $1010 + 1 = 1011$.
Här en tabell över heltalen i exempelsystemet:

Decimalt	Binärt (med tvåkomplement)
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

Negativa tal har en inledande etta. En fördel med detta system är att vi har *en* representation för nollan (andra system kan ha -0 och $+0$). Addition och subtraktion är enkelt och leder till enkla elektroniska kretsar.
$5_{10} + (-2_{10}) = 0101_2 + 1110_2 = 10011_2 \rightarrow 0011_2 = 3_{10}$.
Man stryker alltså helt fräckt den inledande ettan och får korrekt resultat. $-5_{10} + 2_{10} = 1011_2 + 0010_2 = 1101_2 = -3_{10}$.
Vi avstår från att visa att detta alltid fungerar.

Konstiga saker kan inträffa:

$$7_{10} + 1_{10} = 0111_2 + 0001_2 = 1000_2 = -8_{10}$$

Detta kallas heltalsoverflow och inträffar i riktiga program. Analogt gäller att

$$-8_{10} - 1_{10} = 1000_2 - 0001_2 = 0111_2 = 7_{10}$$

Motsvarande fenomen inträffar med fyra-bytes-heltal och C++.

1.4.2 Flyttal

Flyttal (eng. floating point numbers) försöker efterlikna de reella talen. Det som flyter är decimalpunkten, vilket gör att man kan ha tal med mycket varierande storleksordning. Exempel på flyttal är $-1.25 \cdot 10^{-73}$, 23, $5.77 \cdot 10^{56}$. Ett alternativ är "fixed-point" där man har ett fixt antal siffror till vänster om decimalpunkten och ett fixt antal decimaler.
De flyttal som vi använder i Matlab lagras med åtta byte, så 64 bitar. Detta säger oss att det finns många, men dock ändligt många flyttal. En bit används för tecken (0 för + och 1 för -). 11 bitar används för en binär exponent och resten av bitarna används för de binära decimalerna (mantissan). Vi räknar med drygt 16 decimala decimaler. Hur allt detta görs i detalj kommer vi att studera i kursen i numerisk analys.
Vi har ett största tal respektive ett minsta tal. Det finns ett minsta positivt tal liksom ett största negativt. Det finns två nollor. Vi räknar normalt inte exakt, utan råkar ut för avrundningsfel.

Flyttalskonstanterna ovan skrivs (i Matlab, C/C++, Java, Fortran, ...) `-1.25e-73`, `23` respektive `5.77e56`. Notera att `e` står för
exponent (inte för exponentialfunktionen, som heter `exp`).
Lite flyttalsräkning i Matlab (`%` markerar kommentarer):

```
>> format long e            % byt utskriftsformat
>> pi                       % fördefinierat
ans = 3.141592653589793e+00
%     3.1415926535897932384626433832795028841971693...

>> sin(pi)
ans = 1.224646799147353e-16 % inte noll!

>> exp(1)
ans = 2.718281828459046e+00
%     2.7182818284590452353602874713526624977572470...


>> sqrt(2)^2 - 2            % inte noll!
ans = 4.440892098500626e-16

>> (1 / 49) * 49 - 1        % inte noll!
ans = -1.110223024625157e-16

>> x = 1e308
x = 1.000000000000000e+308

>> 10 * x
ans = Inf                   % overflow

>> x = -1e308
x = -1.000000000000000e+308

>> 10 * x
ans = -Inf

>> x = -1e-323
x = -9.881312916824931e-324

>> 0.1 * x
ans = 0                     % underflow

>> 0 / 0
Warning: Divide by zero.
ans = NaN                   % Not-a-Number

>> sin(1 / 0)
Warning: Divide by zero.
ans = NaN

>> atan(1 / 0)
Warning: Divide by zero.
ans = 1.570796326794897e+00

>> atan(1 / 0) - pi / 2
Warning: Divide by zero.
ans = 0
```

```
>> sin(0) / 0
Warning: Divide by zero.
ans = NaN                    % inte ett, med andra ord
```

Eftersom flyttalen består av ändliga utvecklingar av binära decimaler så har vi en uppsättning rationella tal (med luckor emellan). Avståndet mellan två grannar är ungefär 10^{-16} kring ettan och omkring 10^{292} kring det största talet. Antalet flyttal är ungefär $1.8429 \cdot 10^{19} \approx 2^{64}$. Att testa en funktion för alla flyttal i dubbel precision är nästan omöjligt. 10^9 tester per sekund ger 584.4 år.

1.4.3 Byte-ordning

Datorns minne kan ses som en uppsättning numrerade fack, där varje fack omfattar en byte (minnet är byteadresserbart). Facknumret kallas adress. Om man har ett flyttal om åtta bytes så ordnas dessa bytes åt ett av två håll. Man talar om "big endian" (mest signifikanta byten kommer först, har lägsta adress) och "little endian" (den minst signifikanta byten har lägsta adress).

Namnen är hämtade från: Jonathan Swift, Gulliver's Travels (1726), Travels into Several Remote Nations of the World, in Four Parts, by Lemuel Gulliver, First a Surgeon, and Then a Captain of Several Ships.

> ... Which two mighty powers have, as I was going to tell you, been engaged in a most obstinate war for six-and-thirty moons past. It began upon the following occasion. It is allowed on all hands, that the primitive way of breaking eggs, before we eat them, was upon the larger end; but his present majesty's grandfather, while he was a boy, going to eat an egg, and breaking it according to the ancient practice, happened to cut one of his fingers. Whereupon the emperor his father published an edict, commanding all his subjects, upon great penalties, to break the smaller end of their eggs. The people so highly resented this law, that our histories tell us, there have been six rebellions raised on that account; wherein one emperor lost his life, and another his crown. These civil commotions were constantly fomented by the monarchs of Blefuscu; and when they were quelled, the exiles always fled for refuge to that empire. It is computed that eleven thousand persons have at several times suffered death, rather than submit to break their eggs at the smaller end. Many hundred large volumes have been published upon this controversy: but the books of the Big-endians have been long forbidden, ...

Detta användes senare i "On Holy Wars and a Plea for Peace" av Danny Cohen, COMPUTER, Oct 1981, pp.48-54
(skrivet 1980-04-01).

> We agree that the difference between sending eggs with the little- or the big-end first is trivial, but we insist that everyone must do it in the same way, to avoid anarchy. Since the difference is trivial we may choose either way, but a decision must be made.

Våra datorer är little-endian.
Det är viktigt att känna till byte-ordningen om man skall skicka data mellan datorer t.ex.
För detaljer, se: `www.gutenberg.org` för Gulliver's Travels och
`http://ftp.sunet.se/pub/Internet-documents/rfc/`
`ien/ien137.txt` för Danny Cohens artikel (RFC = request for comments, IEN = Internet Experiment Note, notera ftp).

Ett annat dator-relaterat ord från Gullivers resor är Yahoo. Från Webster: One of a race of filthy brutes resembling men but subject to the Houyhnhnms (hästar) in Swift's "Gulliver's Travels."

1.5 Flyttalsprestanda

I en dator finns en eller flera kristalloscillatorer som fungerar som klockor och håller takten i datorn. Olika delar kan gå i olika takt, men den viktigaste takten kallas datorns klockfrekvens (eng. clock frequency). På en modern dator är den några GHz (Giga-Hertz).

En typisk dator kan färdigställa en produkt och en summa (av flyttal) per clockcykel (tiden mellan två tick). Själv operationen kräver flera klockcykler, men olika delsteg (eng. stages) i operationen kan överlappa (som ett löpande band, kallas pipelining i datorsammanhang) så att en färdigt produkt blir färdigställd varje klockcykel. Tyvärr är dators primärminne mycket långsammare än CPUn, så flyttalsenheten får ofta vänta på att få tal att arbeta med (eng. pipeline stalls). Det gör att FPUn inte går på maxfart.
Cykeltiden är 1 dividerat med klockfrekvensen. Den mäts oftast i ns (nano-sekunder, eftersom $1/10^9 = 10^{-9}$.