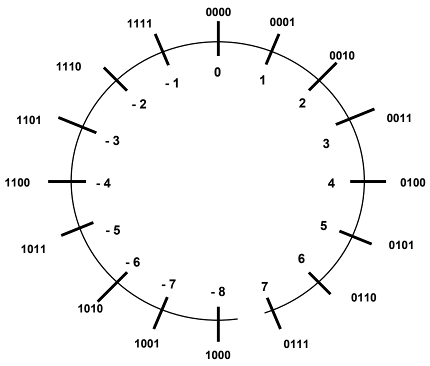
Tall



Tallsystemer

$123 = 01111011_2 = 7B_{16}$

Lærebokas kapittel 6

Posisjonstallsystemer

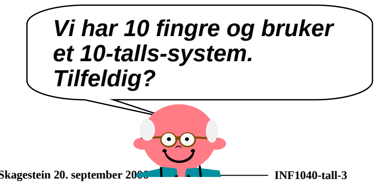
- **Vårt velkjente titallsystem er et posisjonssystem:**

  34567 = 30000
  \+ 4000
  \+ 500
  \+ 60
  \+ 7

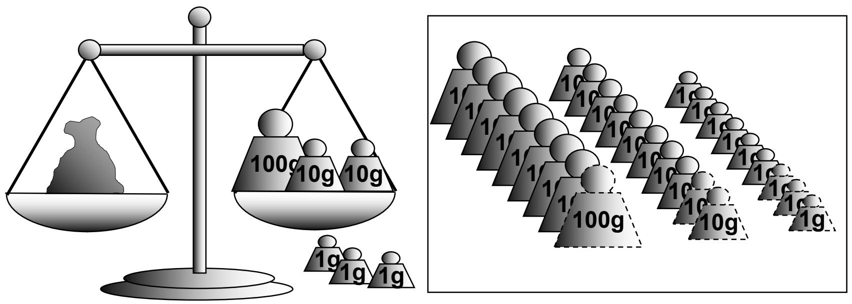
- **eller:**

  $34567 = (3 * 10^4) + (4 * 10^3) + (5 * 10^2) + (6 * 10^1) + (7 * 10^0)$

- **Potenser av 10**
  - $10^0 = 1$
  - $10^1 = 10$
  - $10^2 = 10*10 = 100$
  - $10^3 = 10*10*10 = 1000$
  - $10^4 = 10*10*10*10 = 10000$
  - osv



Veiing med skålvekt – titallsystemet



Loddsats titallssystemet

Den generelle formelen for titallsystemet

- Hvis n er antall siffer, er den generelle formelen for titallsystemet

  $x = s_1 * 10^{(n-1)} + s_2 * 10^{(n-2)} + \ldots + s_{n-1} * 10^{(1)} + s_n * 10^{(0)}$

  $= \sum_{k=1}^{n} s_k * 10^{(n-k)}$

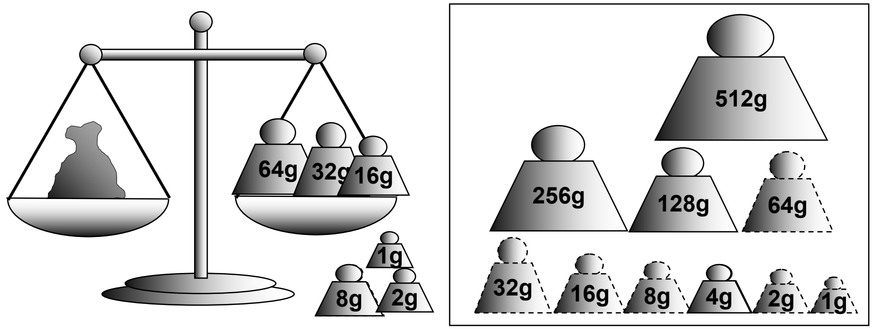
- Eksempel:

  x = 34567 dvs n = 5:

$$(\underset{s_1}{3} * 10^{\overset{5-1}{4}}) + (\underset{s_2}{4} * 10^{\overset{5-2}{3}}) + (\underset{s_3}{5} * 10^{\overset{5-3}{2}}) + (\underset{s_4}{6} * 10^{\overset{5-4}{1}}) + (\underset{s_5}{7} * 10^{\overset{5-5}{0}})$$

k = 1 2 3 4 5

Veiing med skålvekt – det binære tallsystemet



Loddsats
binærsystemet
(totallsystemet)

Det binære tallsystemet

- Har bare to siffer, 0 og 1
- Bygger på posisjonssystemet – som titallsystemet
- Posisjonenes verdi er potenser av 2
- Eksempel:

  $10101_2 = 10000_2$
  $+ 100_2$
  $+ 1_2$

  eller

  $10101_2 = (1 * 2^4) + (0 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0)$
  $= 16 + 4 + 1 = 21$

- Potenser av 2
  - $2^0 = 1$
  - $2^1 = 2$
  - $2^2 = 2*2 = 4$
  - $2^3 = 2*2*2 = 8$
  - $2^4 = 2*2*2*2 = 16$
  - osv

Den generelle formelen for det binære tallsystemet

- Hvis n er antall siffer, er den generelle formelen for det binære tallsystemet

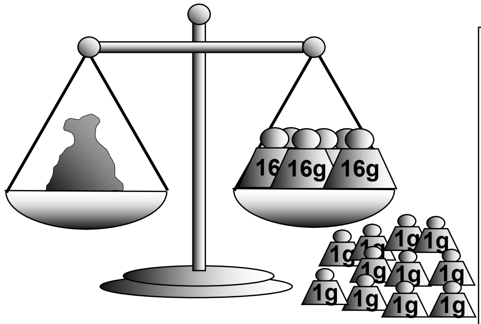
  $x = s_1 10_2^{(n-1)} + s_2 * 10_2^{(n-2)} + \ldots + s_{n-1} * 10_2^{(1)} + s_n * 10_2^{(0)}$

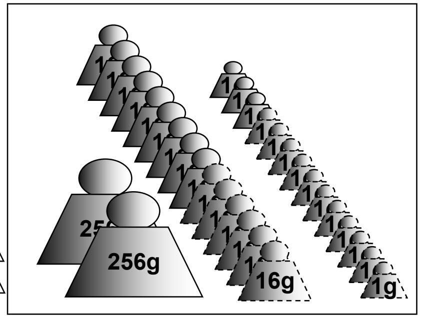
  $= s_1 * 2^{(n-1)} + s_2 * 2^{(n-2)} + \ldots + s_{n-1} * 2^{(1)} + s_n * 2^{(0)}$

  $= \sum_{k=1}^{n} s_k * 2^{(n-k)}$

- Dette er nøyaktig samme formel som for titallsystemet, bortsett fra at grunntallet 10 er byttet ut med grunntallet $10_2 = 2$

Veiing med skålvekt
– det heksadesimale tallsystemet





Loddsats
det heksadesimale systemet

Det heksadesimale tallsystemet

- **Har 16 siffer:**
  0 1 2 3 4 5 6 7 8 9 A B C D E F
- **Bygger på posisjonssystemet – som titallsystemet**
- **Posisjonenes verdi er potenser av 16**
- **Eksempel:**
  $A6C_{16} = A_{16}*100_{16} + 6_{16}*10_{16} + C_{16}*1_{16}$
  eller
  $A6C_{16} = (10 * 16^2) + (6 * 16^1) + (12 * 16^0)$
  $= 10*256 + 6*16 + 12 = 2668$

De heksadesimale sifrene

heksadesimalt siffer	verdi i titallsystemet
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

Den heksadesimale landeplage (?)

Mel.: Kjerringa med staven

C er tolv og D er tretten

E er fjorten, F er femten

seksten ganger seksten, det er to fem seks

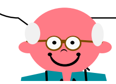
ganger seksten, det er førti nitti seks

(ganger seksten, det er seks fem fem tre seks)

A er ti, B er el've

Heksa B er el've

Nyttig heksadesimal kunnskap:
A = 10 16^2 = 256
B = 11 16^3 = 4096
C = 12 16^4 = 65536
D = 13
E = 14
F = 15



Den generelle formelen for det heksadesimale tallsystemet

- Hvis n er antall siffer, er den generelle formelen for det heksadesimale tallsystemet

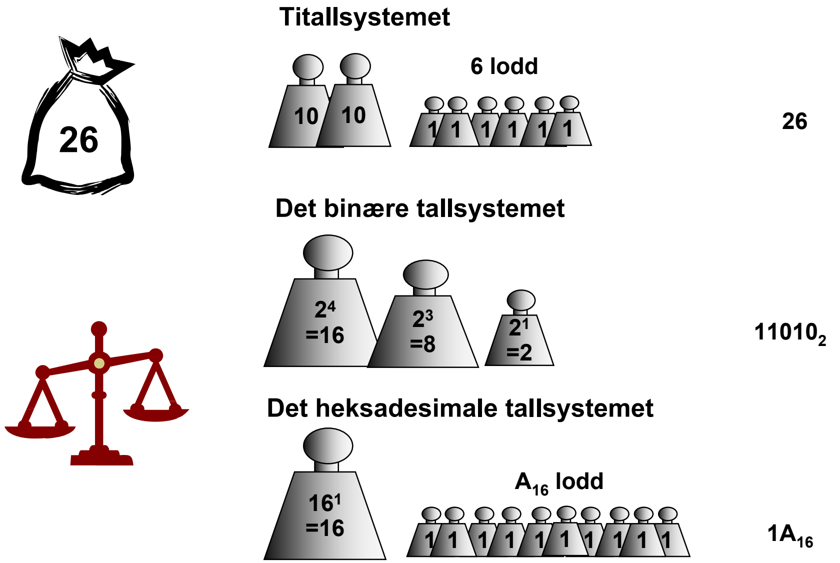
  $x = s_1 * 10_{16}^{(n-1)} + s_2 * 10_{16}^{(n-2)} + \ldots + s_{n-1} * 10_{16}^{(1)} + s_n * 10_{16}^{(0)}$

  $= s_1 * 16^{(n-1)} + s_2 * 16^{(n-2)} + \ldots + s_{n-1} * 16^{(1)} + s_n * 16^{(0)}$

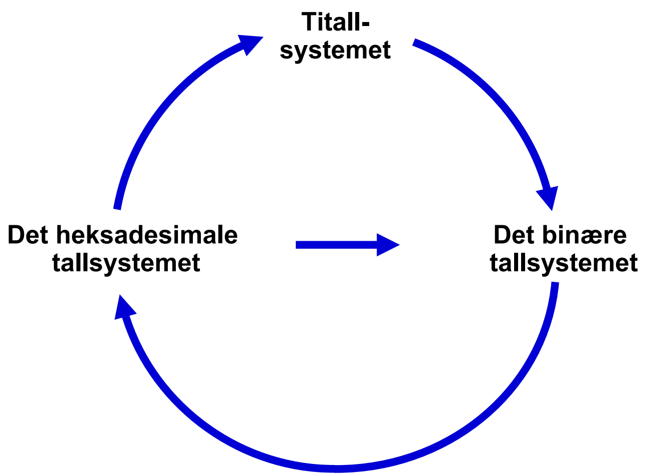
  $= \sum_{k=1}^{n} s_k * 16^{(n-k)}$

- Dette er nøyaktig samme formel som for titallsystemet, bortsett fra at grunntallet 10 er byttet ut med grunntallet $10_{16} = 16$

Tallsystemer - en veiingsanalogi



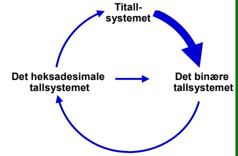
Nyttige konverteringer mellom tallsystemer



Hva er en algoritme?

- Algoritme (latin, opprinnelig arabisk): Metode eller formel for en utregning
- Har en bestemt “input”
- Har en bestemt “output”
- Har bestemte trinn
- Kan utføres av en maskin
- Algoritmen vil terminere hvis “input” er korrekt
- Vi bruker algoritmer for å konvertere mellom tallsystemene

Fra titallsystemet til det binære tallsystemet



En algoritme:

La tallet være x.
Heltallsdivider x med 2, kvotient gir ny x og rest r
– r er siste binærsiffer
Heltallsdivider ny x med 2, kvotient gir ny x og rest r
– r er nest siste binærsiffer
….
Fortsett til x blir 0

Eksempel:

x	x/2	rest
53	26	1
	13	0
	6	1
	3	0
	1	1
	0	1

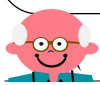
$53 = 110101_2$

Algoritmen:

Er tallet et oddetall eller et partall?

Er halvparten av tallet et oddetall eller et partall?

…



Fra titallsystemet til det binære tallsystemet

En alternativ algoritme:

La tallet være x.
Finn den største toerpotensen som er mindre enn x og sett en 1 i denne posisjonen
Trekk denne toerpotensen fra x og få en ny x
Finn den største toerpotensen som er mindre enn x og sett en 1 i denne posisjonen
…
Fortsett inntil x = 0

Eksempel:

x	2^k	x - 2^k
53		21
		5
		1
		0

2^k		
2^6	64	
2^5	32	1
2^4	16	1
2^3	8	
2^2	4	1
2^1	2	
2^0	1	1

0 0 $_2$

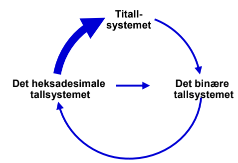
$53 = 110101_2$

Fra det binære til det heksadesimale tallsystemet

- Grupper de binære sifrene 4 og 4 (bakfra)
- Erstatt hver gruppe med det tilsvarende heksadesimale sifferet
- Eksempel:
  $0011\ 0101_2 = 3\ 5_{16}$
- For å komme fra det heksadesimale til det binære tallsystemet bruker vi tabellen motsatt vei

binær siffergruppe	heksadesimalt siffer
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Fra det heksadesimale tallsystemet til titallsystemet



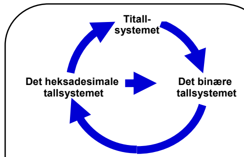
- Vi bruker formelen

$$s_1 \cdot 16^{(n-1)} + s_2 \cdot 16^{(n-2)} + \ldots + s_{n-1} \cdot 16^{(1)} + s_n \cdot 16^{(0)}$$

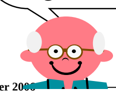
$$= \sum_{k=1}^{n} s_k \cdot 16^{(n-k)}$$

- Eksempel:

$35_{16} = 3 * 16 + 5 = 53$



Ringen er sluttet!



Det finnes mange konverteringstjenester på nettet, for eksempel

http://www.teach-at-home.com/fastfacts/numbers/Binary.asp

Ekstranummer:
Fra det binære tallsystemet til titallsystemet

- **Vi bruker formelen**

  $s_1 * 2^{(n-1)} + s_2 * 2^{(n-2)} + \ldots + s_{n-1} * 2^{(1)} + s_n * 2^{(0)}$

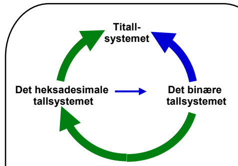
  $= \sum_{k=1}^{n} s_k * 2^{(n-k)}$

- **Eksempel:**

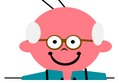
  110101_2

  $= 2^5 + 2^4 + 2^2 + 2^0$

  $= 32 + 16 + 4 + 1 = 53$



Enklere å gå via det heksadesimale systemet?



Representasjon av tall

Lærebokas kapittel 7

To måter å representere tall

- **Som binær tekst**

  Eksempel: '23' i ISO 8859-x og Unicode UTF-8 er

  U+32 U+33, altså $0011\ 0010\ 0011\ 0011_2$

  Brukes eksempelvis ved innlesing og utskrift,

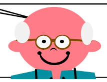
  i XML-dokumenter og i programmeringsspråket COBOL

- **Som binært tall (internt, "native")**

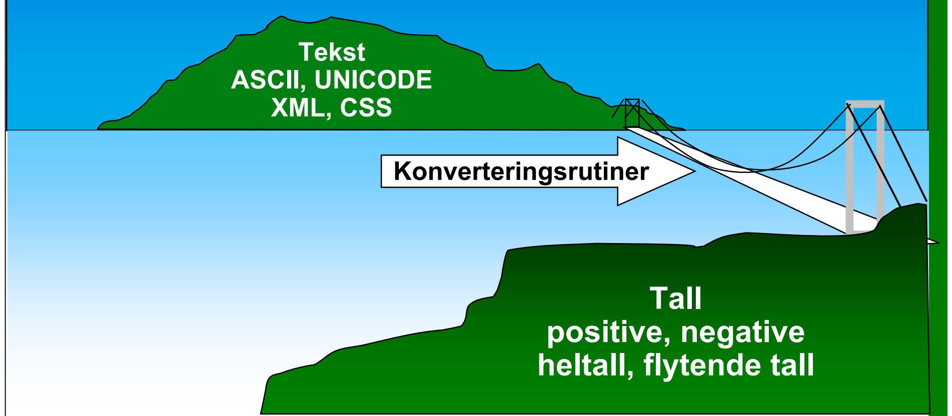
  Eksempel: $23 = 10111_2$

  Brukes som internt format ved lagring og beregninger

Vi skal her se nærmere på binære tall

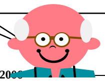


se også
http://courses.cs.vt.edu/~csonline/NumberSystems/Lessons/index.html



Binære regnestykker

- Enkel addisjonstabell
  - 0 + 0 = 0
  - 0 + 1 = 1 + 0 = 1
  - 1 + 1 = 0 med 1 i mente
- Enkel multiplikasjonstabell
  - 0 * 0 = 0
  - 0 * 1 = 0
  - 1 * 0 = 0
  - 1 * 1 = 1
- Et binært tall ganges med 2 ved å føye til en 0 bakerst



Binær addisjon - eksempel

$$
\begin{array}{rr}
53 = & 1\,1\,0\,1\,0\,1_2 \\
21 = & 1\,0\,1\,0\,1_2 \\
\hline
 & 1\,0\,0\,1\,0\,1\,0_2
\end{array}
$$

= 4 * 16 + 10 * 1 = 64 + 10 = 74

Binær multiplikasjon - eksempel

$53 = 1\,1\,0\,1\,0\,1_2$, $21 = 1\,0\,1\,0\,1_2$

$$
\begin{array}{r}
1\,1\,0\,1\,0\,1 * 1\,0\,1\,0\,1 \\
\hline
1\,1\,0\,1\,0\,1 \\
0\,0\,0\,0\,0\,0 \\
1\,1\,0\,1\,0\,1 \\
0\,0\,0\,0\,0\,0 \\
1\,1\,0\,1\,0\,1 \\
\hline
1\,0\,0\,0\,1\,0\,1\,1\,0\,0\,1_2
\end{array}
$$

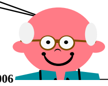
= 4 * 256 + 5 * 16 + 9 * 1 = 1024 + 80 + 9 = 1113

Overflyt ("overflow")

- I en datamaskin regner vi som regel med tallrepresentasjoner med et fastlagt antall biter (vanligvis 8, 16, 24, 32 eller 64)
- Dersom en aritmetisk operasjon fører til at resultatet faller utenfor det mulige tallområdet, har vi en overflyt ("overflow")
- Regneenheten sender da et signal til den omkringliggende programvaren slik at den kan ta seg av situasjonen (feilmelding)
- Eksempel – forutsetter 8 biters tallrepresentasjon

mente -> overflyt

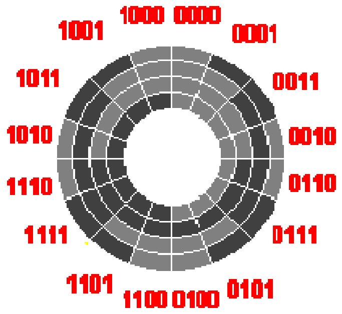
$$
\begin{array}{rr}
 & 1 \\
213 = & 1\,1\,0\,1\,0\,1\,0\,1_2 \\
106 = & 0\,1\,1\,0\,1\,0\,1\,0_2 \\
 & 1\,0\,0\,1\,1\,1\,1\,1_2
\end{array}
$$



Gray-kode

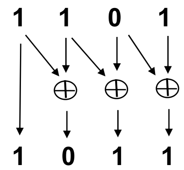
Gray-kode	Binært tallsystem	Titallsystem
0000	0000	0
0001	0001	1
0011	0010	2
0010	0011	3
0110	0100	4
0111	0101	5
0101	0110	6
0100	0111	7
1100	1000	8
1101	1001	9
1111	1010	10
1110	1011	11
1010	1100	12
1011	1101	13
1001	1110	14
1000	1111	15

Et ikke-posisjonssystem der representasjonen av et tall og det neste tallet i tallrekken atskiller seg i bare én bit

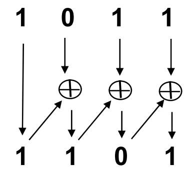


Konvertering binært tallsystem ↔ Gray-kode

Fra det binære tallsystemet til Gray-kode

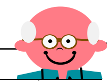


Fra Gray-kode til det binære tallsystemet



⊕ : ”exclusive or”-operasjonen
to like biter gir 0, to ulike biter gir 1

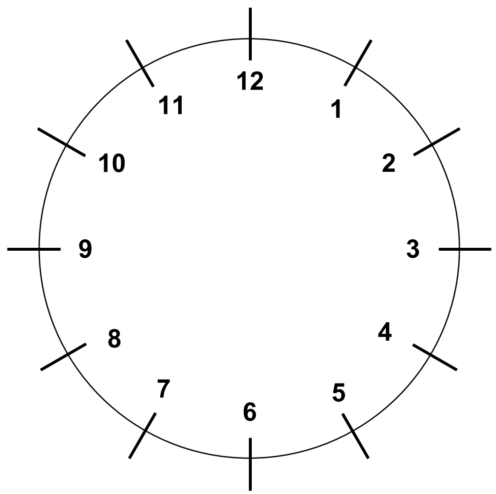
Huskeregel:
I begge konverteringer XOR’es med forrige bit i det *binære tallsystemet*!



Negative tall

- Mange representasjonsmuligheter, men stor fordel om vi kan bruke samme elektronikk for å regne som for positive tall
- Derfor representeres negative tall som *komplementer*
- Vi ser først på litt "klokkearitmetikk" (modulo-operasjoner)
- Deretter lager vi oss en 16-timers (4 biters) klokke, og blir enige om at tallene på venstre del av skiven (som alle starter med en binær 1) skal tolkes som negative
- Modulo-operatoren utføres ganske enkelt ved å se bort fra overflyten

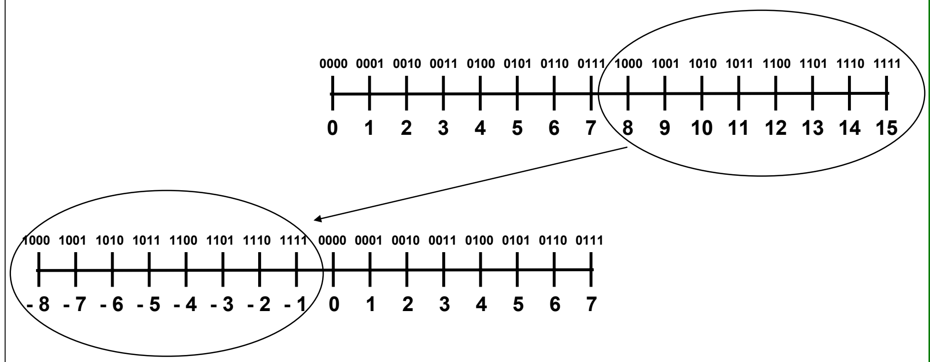
"Klokkearitmetikk"



Hvis klokka er 10, hvor mye er den om 5 timer?
Jo: (10+5)%12

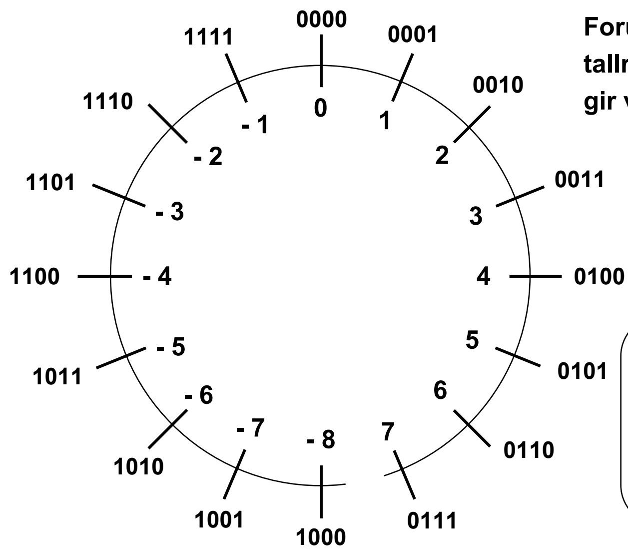
%: Modulo-operatoren (rest av heltallsdivisjon)

Negative tall som toer-komplement



Tenk deg en digital teller som står på 0.
Drei den i negativ retning.
Første nye tall som dukker opp er
komplementet til –1, dvs. 99999….

"Klokkearitmetikk" – toer-komplement



Forutsetter 4-biters tallrepresentasjon, gir verdiområdet [–8,…,7]

Hvis klokka er –2,
hvor mye er den om
5 timer?
Jo:
(1110+0101)%10000

Toer-komplementet

- Det binære toer-komplementet er lett å beregne:
  Ta et binært tall
  Erstatt alle 0 med 1, alle 1 med 0
  (legg merke til at vi må vite antall biter for tallrepresentasjonen)
  Legg til 1
- Eksempel:
  21 = 0001 0101 (forutsetter 8 biter)
  toer-komplementet er 1110 1010_2 + 1 = 1110 1011_2,
  dvs. 14 * 16 + 11 * 1 = 235

  Regne ut 53 + (–21):
  0011 0101_2 + 1110 1011_2 = ~~1~~0010 0000_2 = 2 * 16 = 32

Hva er det binære toer-komplementet til –1 ?

Noen observasjoner om toer-komplement

- Positive binære tall begynner på 0 (mest signifikante bit, MSB =0)
- Negative binære tall begynner på 1 (mest signifikante bit, MSB =1)
- Dersom vi adderer to tall og får mente som "renner over" i forkant, skal menten bare kastes
  – dette er en konsekvens av trikset med toer-komplement- representasjon
- Men…

  …dersom de to menteposisjonene lengst til venstre (inkludert menten som "renner over") er ulike, er vi kommet utenfor det tallområdet som kan representeres. Vi har altså en overflyt!

	overflyt!
mente	0100
4	0100
5	0101
	1001

- Dersom vi legger sammen et binært tall med dets toer-komplement, får vi 0 (selvfølgelig ☺)
- toer-komplementet av toer-komplementet av et tall er tallet
  Gjelder ikke for det "siste" negative tallet ("the weird number")

En annen vri – tall med "bias"

- Vi skal representere tallene [–128,…,127] (8 biters tallrepresentasjon)
- Vi legger en bias 128 til alle tallene, slik at vi istedenfor kan representere tallene [0,…,255] – og det er jo helt kurant
- Ved addisjon kommer bias med to ganger, så vi må trekke den fra igjen
- Eksempel (forutsetter 8 biters tallrepresentasjon og derfor bias 128):
  Vi skal addere 53 og –21.
  53 bias 128 = 181 = $1011\ 0101_2$
  -21 bias 128 = 107 = $0110\ 1011_2$
  181 + 107 – 128 = 10110101_2 + $0110\ 1011_2$ - $1000\ 0000_2$
  = $1010\ 0000_2$ = 10 * 16 = 160 = 32 bias 128
- Dette prinsippet brukes for eksponenten i flytende tall, se lysark INF1040-tall-42

Heltallstyper i Java

datatype	antall biter	minste tall	største tall
byte	8	– 128	127
short	16	– 32768	32767
int	32	– 2147483648	2147 483647
long	64	– 9223372036854775808	9223372036854775807

Flyttall

- Hva hvis heltallsområdet ikke er stort nok?
- Hva med desimaler etter komma?
- Svaret er flyttall!
- Vi ser først på den desimale verden:
  Et tall kan skrives som
  10 $^{\text{eksponent}}$ * mantisse
- Eksempler:
  10^2 * 0,5 = 100 * 0,5 = 50
  10^0 * 0,5 = 1 * 0,5 = 0,5
  10^{-1} * 0,5 = 0,1 * 0,5 = 0,05
  10^{-1} * – 0,5 = 0,1 * – 0,5 = – 0,05

Binære flyttall

- Et binært flyttall er basert på
  2 $^{\text{eksponent}}$ * mantisse
- Vi må representere eksponent og mantisse
  Begge må kunne være både positive og negative (og null)
- Mange representasjonsmuligheter, verden har imidlertid standardisert på IEEE 754
- To varianter:
  - 32-biter representasjon (single precision, datatype ”real”)
  - 64-biter representasjon (double precision, datatype ”double”)

  se http://stevehollasch.com/cgindex/coding/ieeefloat.html

Binære flyttall (forts.)

Single precision

fortegnsbit

eksponent med bias 127

mantisse

1	8	23

1. Ikke-lagret ledende 1 med antatt binærpunkt

Vi legger en bias til eksponenten slik at representasjonen aldri er negativ!

Double precision

fortegnsbit

eksponent med bias 1023

mantisse

1	11	52

1. Ikke-lagret ledende 1 med antatt binærpunkt

Mantissen er normalisert, slik at første bit alltid er 1. Derfor sparer vi plass ved ikke å lagre denne biten!

Flyttallsområder i IEEE 754 (og i Java)

datatype	antall biter	minste positive tall	største positive tall
real	32	$+2^{-126} = + \sim 10^{-44{,}85}$	$+ (2 - 2^{23}) * 2^{127} = + \sim 10^{38{,}53}$
double	64	$+2^{-1022} = + \sim 10^{-323{,}3}$	$+ (2 - 2^{52}) * 2^{1023} = + \sim 10^{308.3}$

og tilsvarende for negative tall

minste positive tall | største positive tall

0

overflow-område | underflow-område | overflow-område

Spesielle verdier:

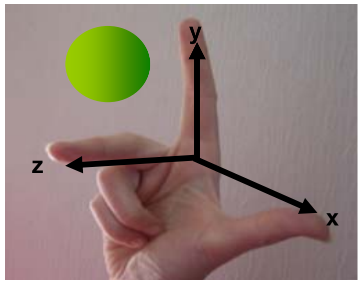
- **Null**: Både eksponent og mantisse er 0 (både +0 og –0)
- **Uendelig**: Eksponent med bare 1ere, mantisse med bare 0ere
- **Not A Number**: Eksponent med bare 1ere, mantisse ≠ 0
  - Mantisse som starter med 1 : Resultat av en udefinert operasjon (eksempel: 0/0)
  - Mantisse som starter med 0: Resultat av en ulovlig operasjon (eksempel: N/0)

Om presisjon og nøyaktighet

- **Presisjon ("precision"):**
  Hvor presist vi ønsker å representere et tall.
  Er direkte avhengig av hvor mange biter vi velger å bruke.
  - Heltall er alltid presist representert innenfor tallområdet.
  - Presisjonen for flyttall varierer,
    men er omvendt proporsjonal med tallets størrelse.
    Jo mindre tall, jo større presisjon!
- **Nøyaktighet ("accuracy"):**
  Hvor nøyaktig vi ønsker (eller greier) å måle eller observere en størrelse.

Siden mange tall ikke kan representeres eksakt, må de "snappes" til nærmeste representerbare tall. Dette kalles *diskretisering.*

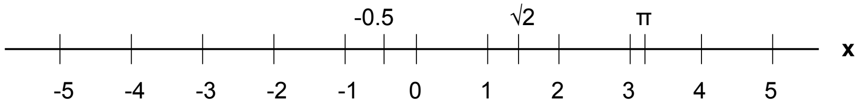
Ut i rommet



Lærebokas kapittel 8

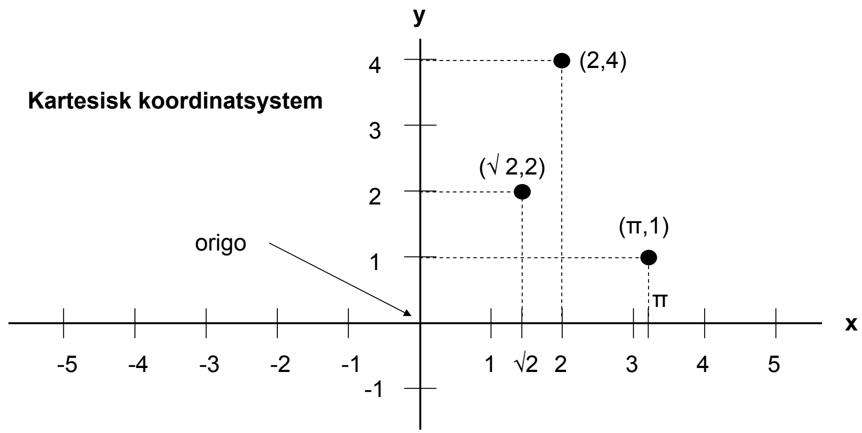
Punkter i endimensjonalt rom

- Et tall kan oppfattes som et punkt i et endimensjonalt rom
- Tallet er da en *koordinatverdi*
- Eksempler: Punktene -0.5, √2, π



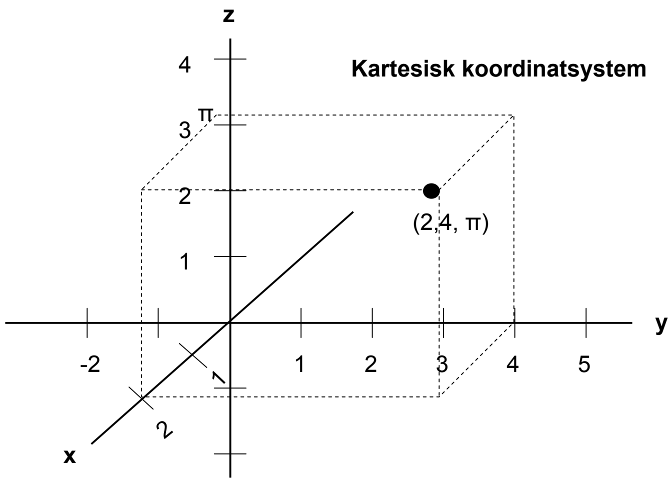
Punkter i todimensjonalt rom

- I et todimensjonalt rom trenger vi et *koordinatpar*.
- Eksempel: Punktet (√ 2,2)

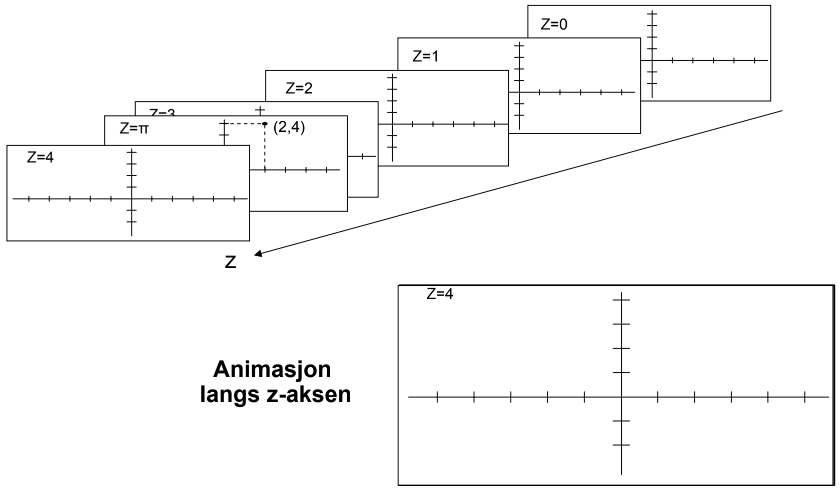


Punkter i tredimensjonalt rom

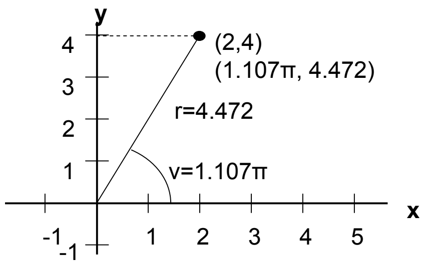
- I et todimensjonalt rom trenger vi et *koordinattrippel*.
- Eksempel: Punktet (2,4, π)



Tredimensjonalt rom vist som animasjon

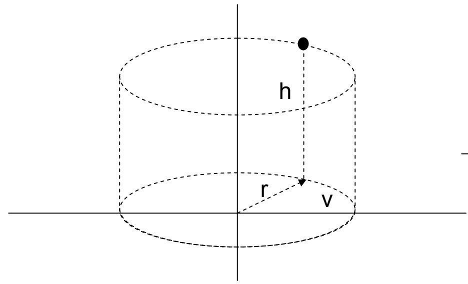


Alternativ til kartesiske koordinater – to dimensjoner

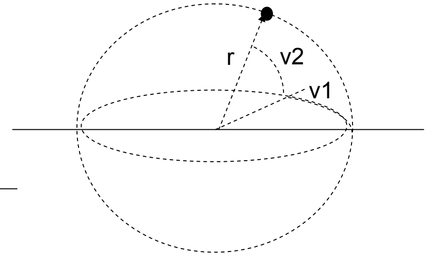


Polarkoordinater

Alternativer til kartesiske koordinater – tre dimensjoner



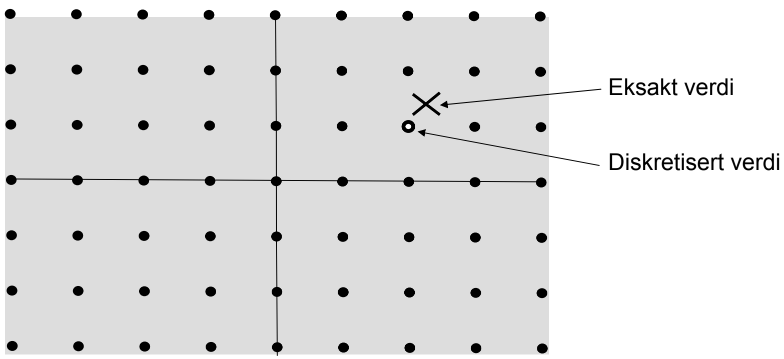
Sylinderkoordinater



Sfæriske koordinater

Diskretisering i rommet

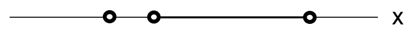
- **Punkter i flerdimensjonale rom må ”snappes” til nærmeste representerbare punkt – på samme måte som tall (punkter i det endimensjonale rom) ”snappes” til nærmeste representerbare tall**



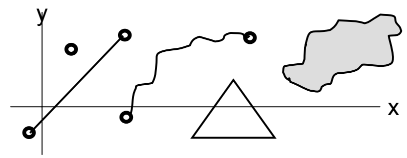
Geometrier i rommet

- **En geometri kan oppfattes som en punktsky med uendelig mange punkter (tett punktmengde).**
- **Dimensjonaliteten kan ikke være større enn rommets dimensjonalitet**

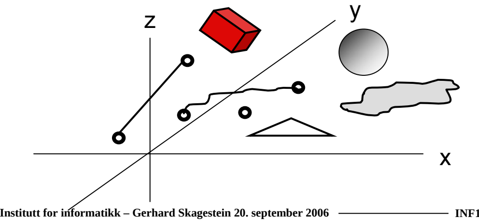
Endimensjonalt rom



Todimensjonalt rom



Tredimensjonalt rom

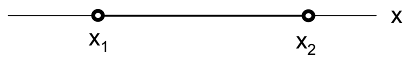


Representasjoner av geometri

- **En uendelig mengde punkter med uendelig presis beliggenhet kan ikke representeres i en datamaskin**
- **To ulike løsninger;**
  - **"Vektorrepresentasjon":**
    Representere noen viktige punkter, og avlede de øvrige punktene matematisk ved behov.
    Egnet for "regulære" geometrier.
  - **"Rasterrepresentasjon"**
    Bygge opp representasjonen av et endelig antall "punkter med utstrekning".
    Gir vanligvis bare en tilnærmet korrekt geometri.

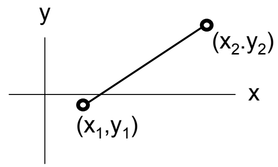
"Regulære" geometrier

Endimensjonalt rom

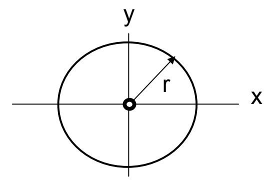


Rett linje

Todimensjonalt rom

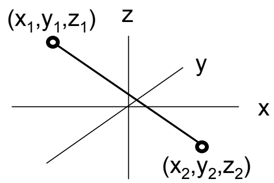


Rett linje

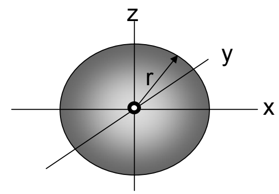


Sirkel

Tredimensjonalt rom

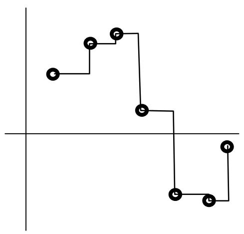


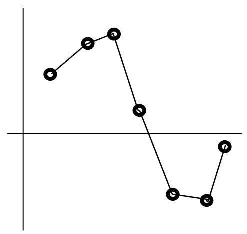
Rett linje

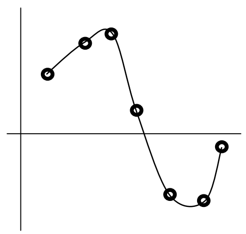


Kuleskall

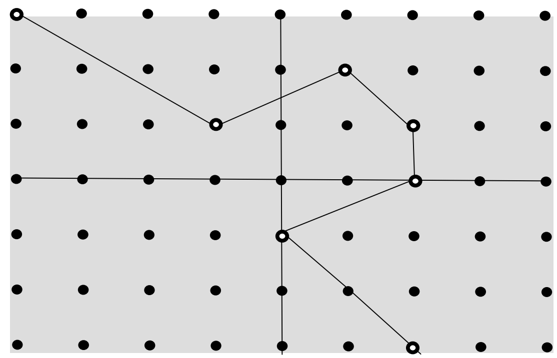
Interpolasjonsteknikker


Interpolasjon med konstant

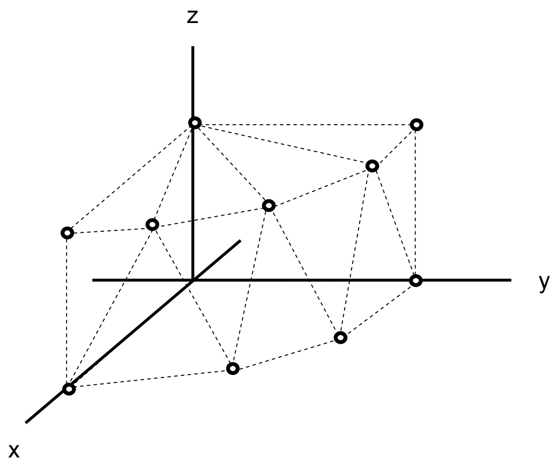

Lineær interpolasjon


Interpolasjon med glatting

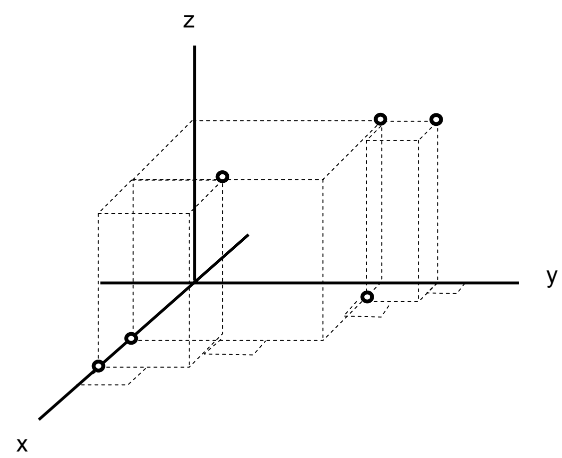
Lineær interpolasjon i to dimensjoner



Triangulated irregular network (TIN)

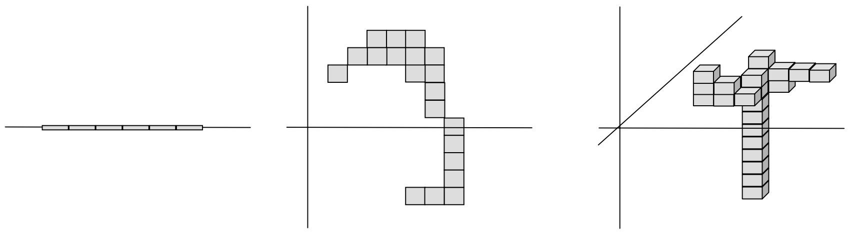


Quad-tre



Rasterrepresentasjon

I rasterrepresentasjon bygges geometrien opp av ”punkter med utstrekning”



Endimensjonalt raster **Todimensjonalt raster** **Tredimensjonalt raster**

Tall – oppsummering

- **Tall kan representeres tekstlig, som binære tall, i Gray-kode eller som flyttall.**
- **For negative binære heltall brukes vanligvis toer-komplementer. Andre alternativer er bruk av fortegnsbit eller bruk av bias,**
- **Tall kan betraktes som punkter i et rom – tallene fungerer da som koordinater.**
- **Punkter som ikke kan representeres eksakt, må ”snappes” til nærmeste representerbare punkt – såkalt diskretisering.**
- **Geometrier med utstrekning kan ikke ha høyere dimensjonalitet enn rommet de er plassert i.**
- **For geometrier med utstrekning kan vi bruke enten vektorrepresentasjon eller rasterrepresentasjon.**