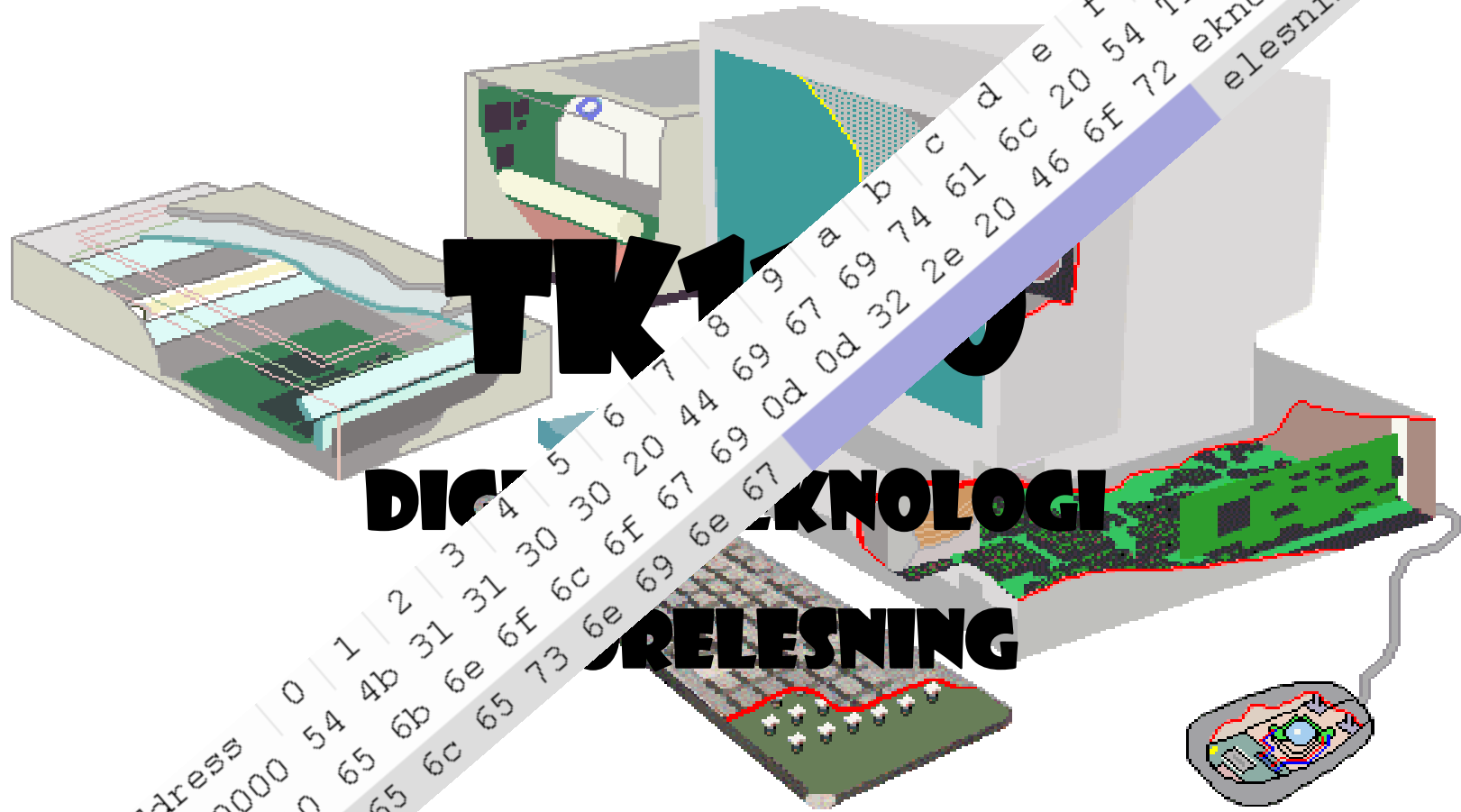




TK1100





TK1100

# Eksamen

---

- 5. oktober, 2 timer skriftlig eksamen
- 7. desember, 3 timer skriftlig eksamen



TK1100

# Forrige gang

- Intro
- Computer/Datamaskin
  - *en menneskelaget innretning som mottar data i en form, behandler disse og produserer ny (og nyttigere) informasjon bygget på de opprinnelige data*
  - TYPEN: Analog vs Digital, Spesialisert vs Generell, Elektronisk vs Mekanisk
- Tallsystemer ~ informasjonskoding
- Enheter: **bit**, **Byte**, **Hz**
  - Prefixer: **K**ibi ( $2^{10}=1024$ ), **M**ibi ( $2^{20}$ ), **G**ibi ( $2^{30}$ )
- Historikk:
  - 4 generasjoner HW
  - 3 behov (beregning, massedata, regulering)



TK1100

# Gjennomgås i dag

- Data-representasjon
- Tall-systemer: desimal, binær, hexadesimal (oktal)
  - Enkel beregninger
  - Presisjon og negative tall
  - Enkle beregninger
- Flyttall (litt om)
- (Koding/dekoding (begynner, vi skal ha mer))
- ØVING!!!



# Poeng!

TK1100

---

- Poenget er ikke i og for seg å regne...
- Vi vil forstå computere bedre...
- Computere er regnemaskiner
- Binæraritmetikk er dermed «computer-psykologi»



TK1100

---

# BINÆR KODING



TK1100

# Data - informasjon

- Data er bare en serie med tegn, som kan ligge lagret eller bli fremvist
- Informasjon har oftest med den tolkning og bruken som f.eks. mennesker gjør av disse tegnene og kombinasjonene av dem
  - Det er 29 forskjellige norske bokstaver (tegn) som kan settes sammen til ord og setninger med mening i. I tillegg benyttes ekstra hjelpetegn (,:!...), formatering (store og små bokstaver) og sifre (0,1,.....,9)
  - Til sammen bruker vi ca. 200 tegn i vår språk-krets
  - Hvert av disse tegnene kan få et nummer og settes i en tabell, den vanligste slike tabellen er ASCII



# ASCII Tabellen (7 bit)

TK110C

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>@</b>	96	60	140	&#96;	<b>`</b>
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>"</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>'</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>*</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>;</b>	91	5B	133	&#91;	<b>[</b>	123	7B	173	&#123;	<b>{</b>
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>	92	5C	134	&#92;	<b>\</b>	124	7C	174	&#124;	<b> </b>
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>	93	5D	135	&#93;	<b>]</b>	125	7D	175	&#125;	<b>}</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>	95	5F	137	&#95;	<b>_</b>	127	7F	177	&#127;	<b>DEL</b>










TK1100

# Typen av data/informasjon

- En (generell) datamaskin behandler 5 hovedtyper informasjon
- Numerisk
  - Tall-beregninger
- Karakterbasert (alfanummerisk)
  - Tekstmanipulasjon
- Visuell
  - Bilder
- Audio
  - Lyd
- Instruksjoner
  - Interne ordre til datamaskinen (CPU'en) om hva den skal gjøre

# Tolking av binære koder (noen få)

	0011 1110	0010 0000	0111 0010	0011 0111
(Hexadesimalt)	0x3E	0x20	0x72	0x37
32-bit heltall	1 042 313 783			
16-bit heltall	15 904		29 239	
32 bit flyttall	0.156686			
BCD	Umulig!	20	72	37
IPv4-adresse	62.32.114.55			
ASCII	>	mellomrom	r	7
Scancode(USB)	F5 	3 # 	F23	 .
UTF-16	桃		爷	
JVM bytekode	istore_3	Istore_2	frem	Istore
X86 opkode	DS:	AND	JNO	AAA

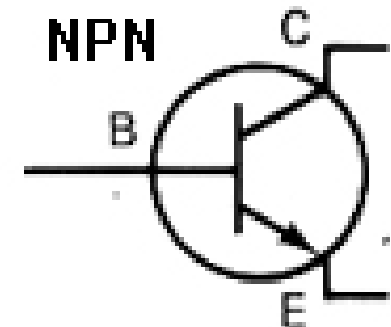
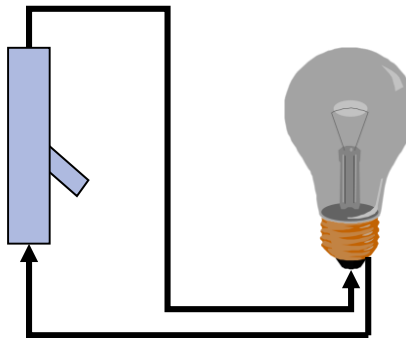
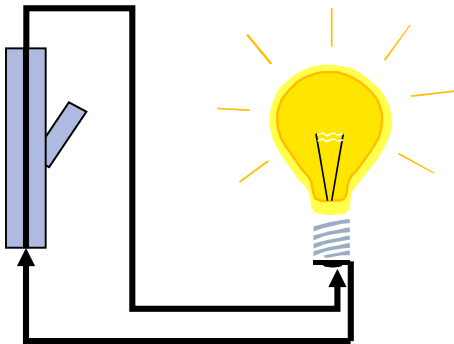
- Hvordan vet man hvilken tolkning som er riktig?
- Det bestemmer (bruken i) programmet!



TK1100

# Representasjon i en datamaskin

- Dagens digitale datamaskiner bruker binære tall
  - Trenger da bare to sifre: 0 og 1
- Dette gir muligheten for enkle logiske kretser
- og samtidig kan all slags informasjon representeres på denne måten



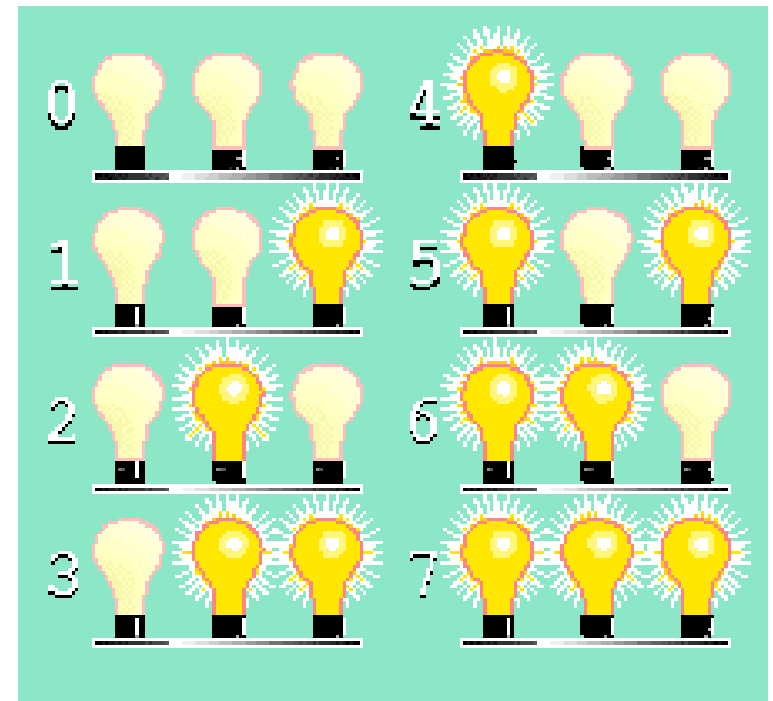


TK1100

# Binær tall-representasjon



- Med 3 lamper kan vi representere  $2^3 = 8$  kombinasjoner av av/på
- Med Av=0 og På=1 får vi
  - 0 = 000    4 = 100
  - 1 = 001    5 = 101
  - 2 = 010    6 = 110
  - 3 = 011    7 = 111
- Dette kan utvides til å bruke flere lamper (transistorer), f. eks. 8, 16, 32, 64, ....

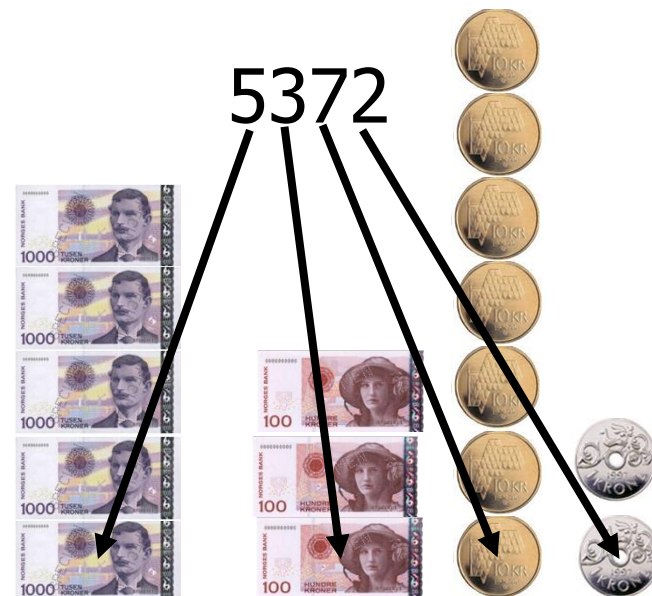




TK1100

# Desimale tall - posisjontall

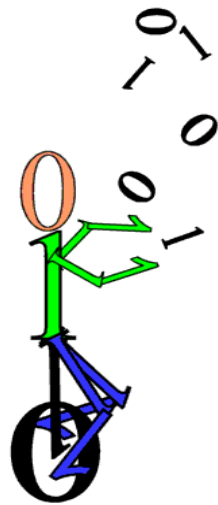
- Det «vanlige» (**desimale**) tallsystemet bruker **10 sifre**/symboler (0 – 9), mens det **binære** systemet bare bruker **2** sifre/symboler (0-1).
- Prinsippene bak det binære tallsystemet er imidlertid de samme som for det desimale systemet
  - **Posisjonen** til et siffer i et tall **avgjør** hvilken **verdi** sifferet representerer («tyngde»)





TK1100

# Tallsystemer



- Alle **posisjonelle** tallsystemer bruker en **base**
- Det **desimale** tallsystemet har base **10**
- Hver **posisjon** i tallet tilsvarer en **potens** av basen
- I prinsippet kan en bruke en hvilken som helst base
  - $407_{23} = 4 \cdot 23^2 + 0 \cdot 23^1 + 7 \cdot 23^0 = 2116 + 0 + 7 = 2123_{10}$
  - $\quad \quad = 6122_7$
- Det fine med posisjonelle tallsystemer er at fravær av en potens-verdi («vekt») kan representeres med **0**
  - **407** er ikke det samme som 47



TK1100

# Konvertering fra desimal til binær

$$\begin{array}{rcl} 851 & = & 512 + 339 = 2^9 + 339 \\ 339 & = & 256 + 83 = 2^8 + 83 \\ 83 & = & 64 + 19 = 2^6 + 19 \\ 19 & = & 16 + 3 = 2^4 + 3 \\ 3 & = & 2 + 1 = 2^1 + 1 \\ 1 & = & 2^0 \end{array}$$

$$851 = 2^9 + 2^8 + 2^6 + 2^4 + 2^1 + 2^0$$

$$851 = 1 \cdot 2^9 + 1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

$$851_{10} = 0000\ 0011\ 0101\ 0011_2$$

$2^0$	=	1
$2^1$	=	2
$2^2$	=	4
$2^3$	=	8
$2^4$	=	16
$2^5$	=	32
$2^6$	=	64
$2^7$	=	128
$2^8$	=	256
$2^9$	=	512
$2^{10}$	=	1024



TK1100

# Konvertering fra binær til decimal

$$\begin{aligned}1101010011 &= 1*2^9 + 1*2^8 + 0*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 \\&= 512 + 256 + 64 + 16 + 2 + 1 \\&= 851\end{aligned}$$



512	256	128	64	32	16	8	4	2	1
<b>1101010011</b>									
512	256	64	16					2	1

$2^0$	=	1
$2^1$	=	2
$2^2$	=	4
$2^3$	=	8
$2^4$	=	16
$2^5$	=	32
$2^6$	=	64
$2^7$	=	128
$2^8$	=	256
$2^9$	=	512
$2^{10}$	=	1024

**NB! Her mangler innledende nuller i 16 bit presisjon!**

0000 0011 0101 0011

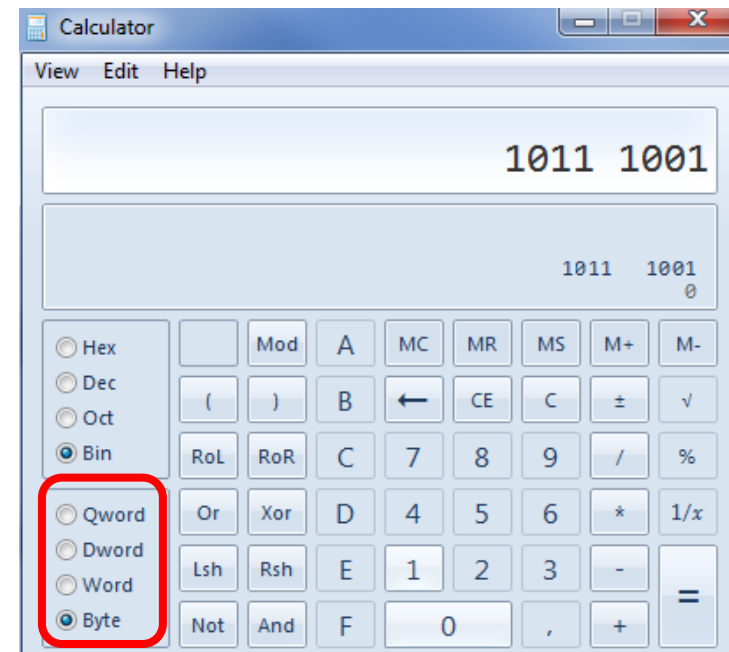




# PRESISJON

TK1100

- I computere lagres alt enten i RAM («minne»), i **registre** («minne») på CPU eller annet utstyr.
  - Disse har **adresser**/navn
- Både i minne og på CPU er **minste adresserbare enhet** en **Byte**
  - Det er ikke mulig å lagre en bit, minimum er 8!
- Microsoft m.fl. operer med enhetene:
  - **Byte**: 8 bit
  - **Word**: 16 bit
  - **Dword**: 32 bit
  - **Qword**: 64 bit





TK1100

# Regneoperasjoner - binærtall

## Addisjon

$$\begin{array}{r} \phantom{0000} 1011 \\ + 0001 1010 \\ \hline 0010 0101 \end{array}$$

Se  
også D1.0

## Multiplikasjon

$$\begin{array}{r} 10 * 10 \\ \hline 00 \\ 10 \\ \hline 0100 \end{array}$$

Se  
også D1.0

NB!

Å doble er dermed det samme som å legge til en null lengst til høyre...



TK1100

# Negative tall = toerkomplement

- Invertering bør ikke resultere i forskjell på +0 og -0
- Bruker 2'er komplement istedenfor 1'er komplement

0001 0011

1110 1100

1110 1101

1's komplement (flip (snu) alle bits)

2's komplement = 1's komplement + 1





TK1100

# Toerkomplement

- Toerkomplement fungerer bare forutsatt en bestemt **presisjon**, f.eks. 8 bit
  - Da blir 127 (0111 1111) det største tallet som finnes, -128 (1000 0000) det minste tallet som finnes.
  - -128 kalles også "tulletallet" (the silly number) fordi det ikke finnes noen positiv versjon av det.
  - Alle andre tall kan man skifte fortegn på ved å ta toerkomplementet.



TK1100

# Subtraksjon

- Å trekke fra er dermed *alltid* det samme som å legge til toerkomplementet

$$\begin{array}{rcll} 46 & = & 0010\ 1110 & = & 0010\ 1110 \\ -37 & = & -0010\ 0101 & = & +1101\ 1011 \\ \hline 9 & & & & \hline \hline \end{array}$$
$$\begin{array}{r} 1\ 0000\ 1001 \\ \hline \hline \end{array}$$

↙

**Overflow** (spillsiffer),

Det sløyfer vi!!

Pga 8 bit presisjon



TK1100

---

- Kunne konvertere mellom hexadesimale og binære sifre
- Vite forskjellen på Little og Big Endian, og noen konsekvenser av dette

## HEX OG REKKEFØLGE



TK1100

# Hexadesimale tall

- Bruker **16** som base
  - Må da finne opp 6 "nye" sifre
    - $A_{16}=10, B_{16}=11, C_{16}=12, D_{16}=13, E_{16}=14, F_{16}=15$
    - **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**
    - $11_{16} = 1*16 + 1*1 = 17_{10}$
- $$0A4C_{16} = \begin{array}{r} 0*16^3 + 10*16^2 + 4*16^1 + 12*16^0 \\ 0*4096 + 10*256 + 4*16 + 12*1 = 2636_{10} \end{array}$$
- Oversetter fra binær til hex ved å gruppere 4 og 4 binærtall sammen ("nibbler")
    - **1010 0011 1001 1111**<sub>2</sub> = 0x**A39F**<sub>16</sub>
    - $1010_2 = A_{16}, 0011_2 = 3_{16}, 1001_2 = 9_{16}, 1111_2 = F_{16}$



TK1100

# Hvorfor Hex?

- Færre sifre— **BRUKES FOR Å SKRIVE BINÆRTALL MER KOMPAKT**
- **Noteres** i Java og mange andre sammenhenger med prefix **0x**
  - F.eks **0x7F** = 127 = **0b0111 1111**
- Se også D1.0
- Det er lett å regne feil med Hex så vi går helst veien om binær!





TK1100

# Big vs little Endian

- Hvilken rekkefølge skal bits og bytes lagres og overføres i?
- For representasjoner som krever mer enn en byte har vi to muligheter
  - Starte med **minst signifikante** byte (LSB på lavest adresse)
  - Starte med **mest signifikante** byte (MSB på lavest adresse).
- I praksis betyr dette at i UTF-16 har f.eks "A" to forskjellige representasjoner
  - **Big Endian**: 0x00 41 (IMB, Mac inntil Intel)
  - **Little Endian**: 0x41 00 (Dette var/er vanligst på Intel/AMD)
  - Misforståelse vil erstatte **A** med 祇

RAM-adresse	Big Endian	Little Endian
001A3BF7		
001A3BF8	00	41
001A3BF9	41	00
001A3BFA		





TK1100

---

- Vite hvordan flyttall i et posisjontallsystem fungerer
- Kjenne til koding-standardene IEEE 754
- Kjenne til avrundingsproblemene forbundet med bruk av flyttall

# FLYTTALL



TK1100

# Flyttall

- Hvordan representerer man 5,625 binært?
- Som i et hvilket som helst annet posisjons-tallsystem?

$$\begin{aligned} 5.625 &= 5 \frac{5}{8} = 4 + 1 + \frac{1}{2} + \frac{1}{8} = \\ &1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} \\ &= 101,101 \end{aligned}$$

- Dette er helt tilsvarende at 103,57 er en notasjon for:

$$1*10^2 + 0*10^1 + 3*10^0 + 5*10^{-1} + 7*10^{-2}$$

- I.e. Å konvertere til "kommatal" er helt tilsvarende hva vi gjør i desimaltallsystemet



TK1100

# Ex: Konvertere 0,6875 til bin

Start tall	Doblet tall	Resultat så langt
<b>0.6875</b>	1.375	.1
0.375	0.75	.10
0.75	1.5	.101
0.5	1.0	<b>.1011</b>



TK1100

# IEEE 754 og x87

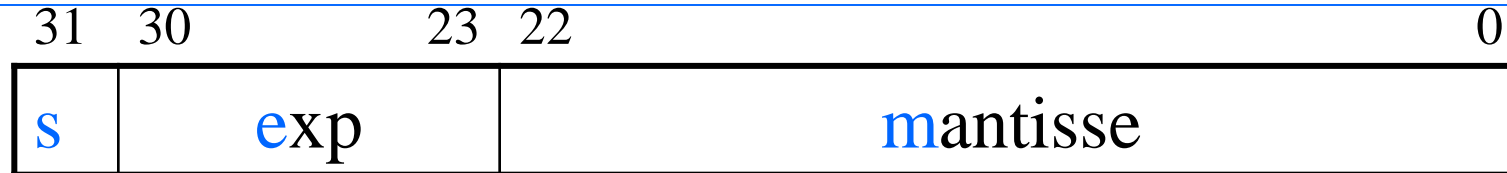
- Standarden IEEE 754 ble utviklet av W. Kahn i samvikling med Intel's utvikling av 8087 matte-koprosessoren
- Brukt i de aller fleste computere. Intels matte koprosessorer (innebygd i alle CPUer etter Pentium).
- IEEE definerer "single precision" (brukt av `float` i Java, 32 bit) og "double precision" (`double` i Java, 64 bit).
- Intels matte-"koprosessor" har også en tredje, høyere presisjon: "extended precision" (80 bit, brukes alltid til mellomregning i flyttallsenheten (FPU)).
- Nyeste versjon av standarden er IEEE 754-2008
  - Tillater også 128 bits flyttall (34 desimaler) m.m.m.





TK1100

# IEEE single precision



- Nøyaktig inntil ca 7 desimalplasser.
- s er en tegn-bit. 0 for positiv, 1 for negativ.
- exp (8 bit) er "biased" = virkelig eksponent + 7Fh. Verdiene 00h og FFh har spesielle betydninger.
- Mantissen er 23 bit – de første 23 bit etter 1 i signifikanden.



TK1100

## Ex: 5,8 i $\overline{\text{IEEE}}$ single precision?

- $5,8 = 101,1100\ 1100\ 1100\ 1100\ 1100\ 1100\ \dots$
- Med 23 tall etter desimaltegnet, har vi  $1.0111\ 0011\ 0011\ 0011\ 0011\ 001\ *2^{10}$
- Tegn bit'en er positiv = 0.
- **Justert** exponent er  $0x7F + 0x02 = 0x81$
- Vi får da:  
0100 0000 1011 1001 1001 1001 1001 1001  
(fortegn og mantisse er understreket)  
eller 40B99999h
- Merk: I C blir 5.8 representert som 40B9999A, fordi vi droppet en LSB som var en 1. Det er en bedre tilnærming til 5,8



- $0.5_{10} = 0.1_2$
- Med 23 siffer etter desimaltegnet, har vi  $1.000000000000000000000000 * 2^{-1}$
- Fortegnsbit'en er negativ = 1.
- Exponenten er 7Fh + -1h = 7Eh
- 1011 1111 0000 0000 0000 0000 0000 0000  
(BF000000h)

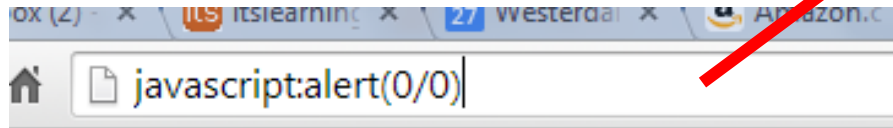
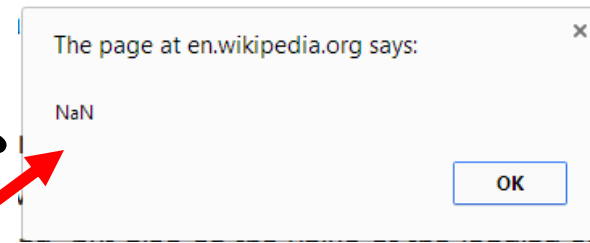




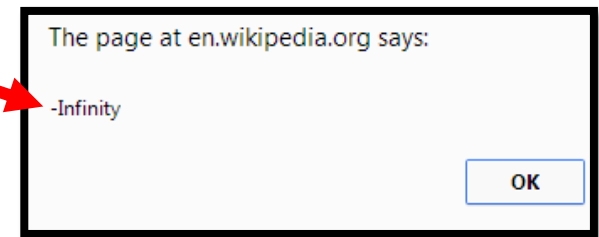
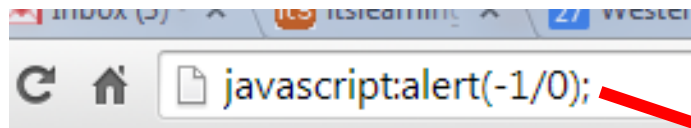


TK1100

# Not a Number



- IEEE754 har en egen koder for resultater som ikke vanlige er tall (**f**111 1111 1**k**xxx ...): NaN
- Denne finnes i flere formater:
  - Avhengig av om koden skal brukes videre eller ikke (**k**=1 er «quiet NaN»)
  - Avhengig av type (feil-)beregning som forårsaket NaN
- Det finnes også egne koder for positiv og negativ **uendelig** m.fl.





TK1100

# Flyttall kan være farlige!!

- En Ariadne-rakett styrtet pga av en flyttallsfeil
- F.eks.  $10\% = 0,1$  i desimaltallsystemet
  - Binært er det ikke mulig å skrive  $1/10$  med et endelig antall siffer, det blir  $0,000110011001100110011001100...$
  - Konverterer vi tilbake til desimaltall kan vi ende opp med  $1/10 = 0,0999999994$
  - I IEEE754 kan du angi hva slags avrunding som skal gjøres i hvilken retning, men må likevel alltid forholde deg til hvor mange siffer du faktisk kan stole på.



TK1100

---

# BCD

## OG ANDRE KODE-TABELLER



TK1100

# Kodetabeller

- **BCD** – kode (Binary Coded Decimals)
  - Hvert siffer i desimaltallet får sin 4 bits binærkode
  - $529_{10} = 010100101001_{\text{BCD}}$
  - Har også andre varianter der vi bruker 8 bit pr desimalsiffer, f.eks.
  - $529_{10} = 0000101\ 00000010\ 00001001_{\text{PBCD}}$
  - Intel/AMD-prosessorer har fremdeles egne instruksjoner for BCD-beregninger
- Ligner dermed på **alfanumerisk** koding
  - Alle sifre, bokstaver (små og store) og spesialtegn nummereres
  - EBCDIC, ASCII, ISO, Unicode
  - $M = 77_{\text{ASCII}} , \quad := 94_{\text{EBCDIC}}(5E_{16})$



# ASCII Tabellen (7 bit)

TK110C

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>@</b>	96	60	140	&#96;	<b>`</b>
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>"</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>'</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>*</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>;</b>	91	5B	133	&#91;	<b>[</b>	123	7B	173	&#123;	<b>{</b>
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>	92	5C	134	&#92;	<b>\</b>	124	7C	174	&#124;	<b> </b>
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>	93	5D	135	&#93;	<b>]</b>	125	7D	175	&#125;	<b>}</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>	95	5F	137	&#95;	<b>_</b>	127	7F	177	&#127;	<b>DEL</b>





TK1100

# ASCII (merk!)

- "A" = 0x41 -> 0100 0001  
"a" = 0x61 -> 0110 0001
  - En bit forskjell mellom store og små bokstaver
- Tallene **kodes** med: 0x30-0x39
  - Binærtallet 0000 0000 har ASCII-kode 0011 0000
- Mellomrom er 0x20
- Linjeskift er (i Windows) CarriageReturn LineFeed (0x0D 0x0A), i UNIX/Linux bare LineFeed, Mac OS brukte bare CR, mens OSX bruker LF (hovedsakelig)
- **Mer om dette og andre formater i neste forelesning**





TK1100

---

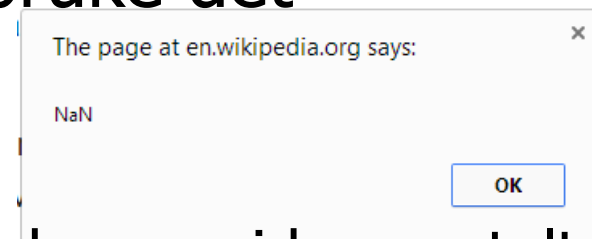
# OPPSUMMERING



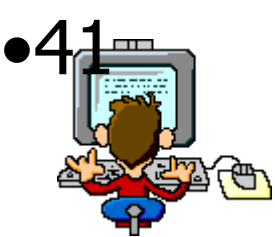
# Hva skal vi kunne

TK1100

- Hvordan et posisjonstallsystem er bygd opp
- Hvordan binærtallsystemet fungerer og hvorfor vi foretrekker det i datamaskiner
- Konvertere desimal  $\leftrightarrow$  binær  $\leftrightarrow$  hexadesimal
- Addere binærtall
- Finne toerkomplementet, og bruke det
- Vite at flyttall er annerledes!
  - Ikke aktuelt å gjøre for hånd...
  - IEEE 754-2008 er mer omfattende enn vi har omtalt







TK1100

---

Det finnes 10 typer mennesker i verden,  
de som kan binært og de som ikke kan det.



TK1100

---

# Neste gang?

- Mer om tegnsett
  - ASCII, ISO 8859-1, Windows 1252,..
  - Unicode
    - UTF-16 og UTF-8
- Koding av lyd og bilder
- Ulike filformater
- Komprimering



TK1100

---

0x3F