

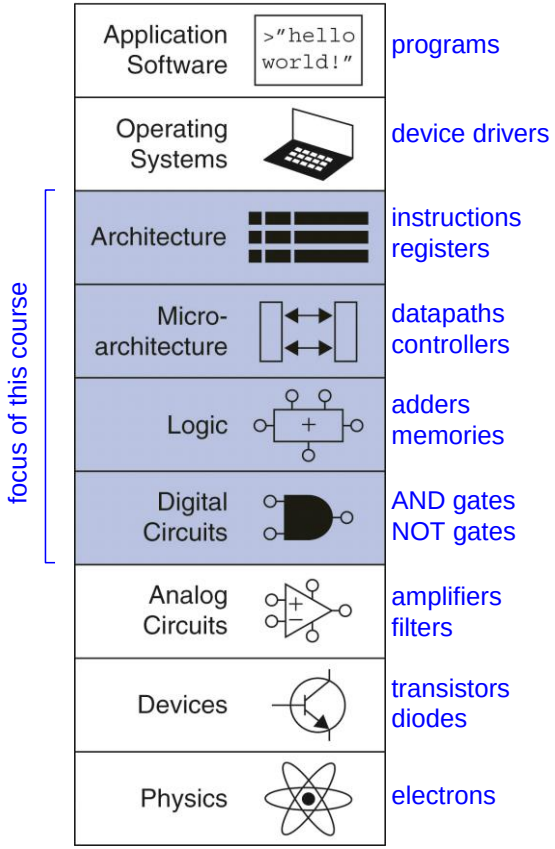
UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

IN2060 – Forelesning 1

Introduksjon til digitale systemer (Kap. 1)

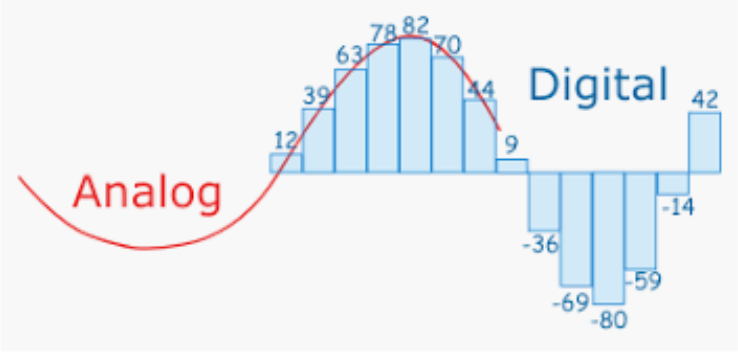
Tema

- Introduksjon til digitale systemer
- Den digitale abstraksjonen
- Tallsystemer
- Logiske operasjoner og porter



Hva menes med digital teknologi?

- Teknologi som opererer med digitale signaler, eller diskrete data.
- Vi skal se at det er mange fordeler med digitale systemer

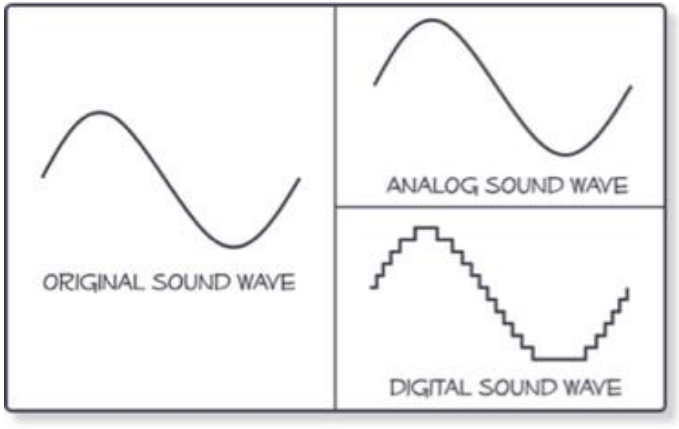


Digitale systemer

- Basert på binære tall (noen unntak):
  - Tall som kun er representert ved symbolene 0 og 1 (eller bit's).
  - Nøyaktighet er gitt av antall bit.
- Digitale system:
  - System som håndterer digitale tall (data)

Endelig oppløsning – ulempe?

- Verdier er som oftest analoge
  - Temperaturen til en kopp kaffe
  - Hastigheten til en bil
  - Lyd, farger, osv.



- I praksis kan analoge signaler konverteres til et 10 til 16 bits tall uten tap
  - Ikke merkbar forskjell.
  - Støy og målefeil begrenser den oppnåelige oppløsningen uansett.

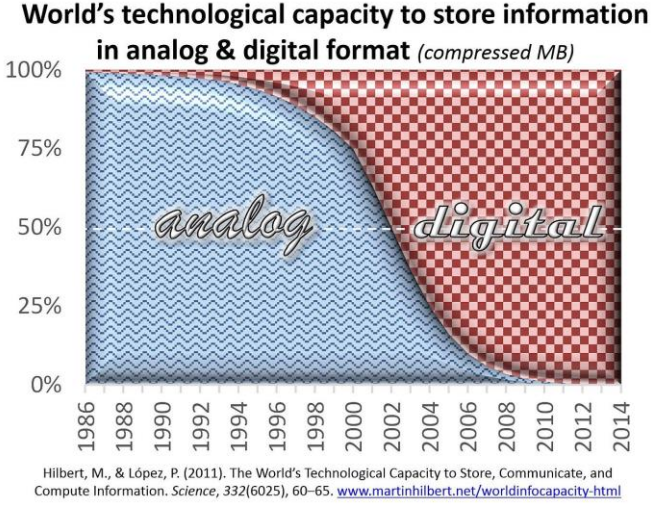
Fordelene med digital systemer

010…0101 ⟶ 010…0101
16.54368.. ⟶ 16.54337..

- Tapsfri lagring av signaller
- Tapsfri signalbehandling
- Kraftigere muligheter for manipulering
- Enklere design
- Billigere

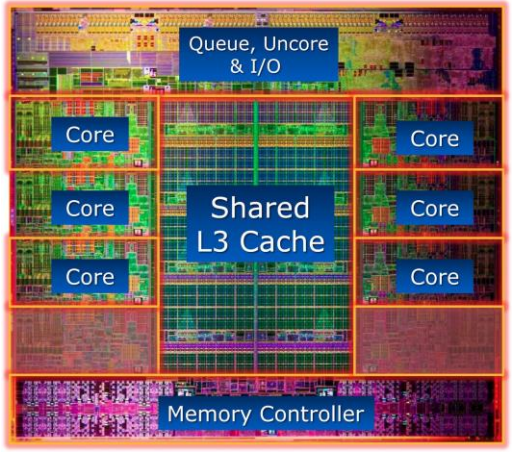
Den digitale revolusjonen (1950+)

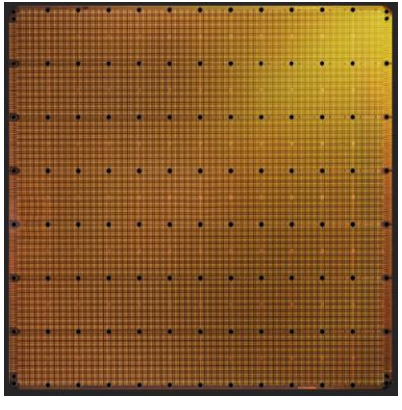
- Takket være digitale systemer har vi fått
  - Datamaskiner (~150++ MCU per hjem?)
  - Internett
  - Mobiltelefoner
  - Osv..
- Digitale systemer har tatt over for de fleste analoge systemer.
  - Lyd (CD 1982), Foto (Nikon D1 1999) TV (2009), Radio (2013) osv.
  - Eksamen (Inspera 2014-2018)



Enklere design?

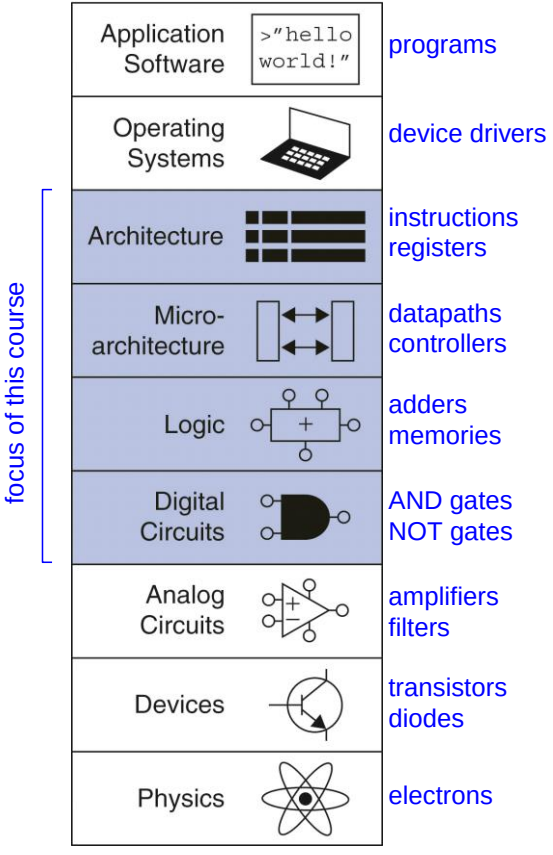
- Fordelen med digital design: slipper å bry oss om svært nøyaktig og dyr elektronikk.
- Kun jobbe med 0 og 1'ere.
- Enklere?
- Apple M3 Max (16-core, 64-bit)
  - 92,000,000,000 transistorer (2023)
- Cerebras WSE-3, **stor** AI Chip ~22x22 cm
  - 4,000,000,000,000 (900 000 AI-kjerner), 23 kW
  - Nisje chip, NVIDIA H100 mye mer brukt.
- Kunsten er å kunne håndtere mange av dem i store kompliserte systemer.
- **Abstraksjon**!





Digital abstraksjon

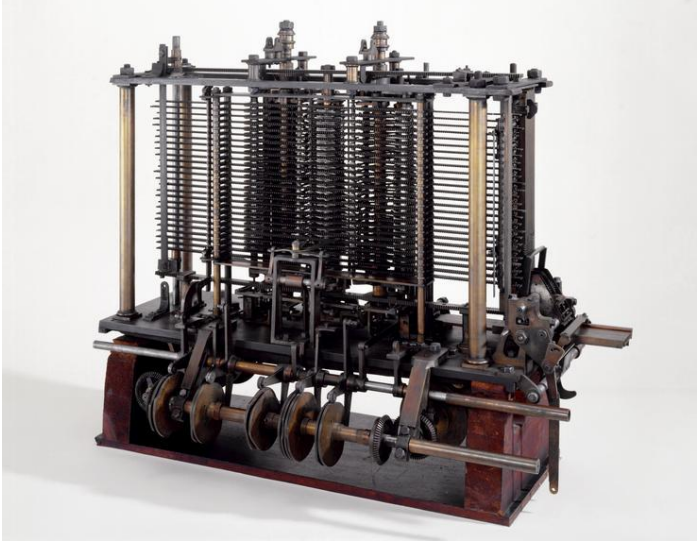
- Bruke ferdige moduler uten å tenke på hvordan de fungerer.
- Resultat:
  - Muliggjør konstruksjon av svært kompliserte systemer.



The Analytical Engine (~1837)



- Matematiker og pioner Charles Babbage.
- Antatt å være den første *foreslåtte* digitale maskinen. (Turing-complete).
- Konstruert av mekaniske tannhjul, der hvert hjul representerte en diskret verdi (0-9)
- Minne: “1000” tall (~16.2 kB)
- Babbage forsto at maskiner ville revolusjonere regning.



TALLSYSTEMER

Binære verdier

- To diskrete verdier, 0 eller 1.
  - 0, FALSE, lav
  - 1, TRUE, høy
- Typiske digitale kretser bruker spenningsverdier for å representere 1 og 0.
- Bit = Binary digit
- Hvordan representerer vi vanlige tall med bit?

Desimaltall

- Vi er vant med desimaltall fra dagliglivet
  - Eller *ti-tallsystemet*.
  - Består av symbolene/tallene 0,1,2,3,4,5,6,7,8,9. Disse settes sammen for å danne større tall.
  - Plasseringen bestemmer hvor stor "vekt" tallet har.
    - Neste posisjon har 10 ganger større vekt enn forrige posisjon (*ti*-tallsystemet).
- Eksempel
  - $(1423)_{10} = 1 \cdot 10^3 + 4 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 = (1423)_{10}$
  - $(1423)_{10} = 1000 + 400 + 20 + 3$

Binære tall

- Binære tall består av symbolene 0 og 1.
  - Neste plassering gir 2 ganger større verdi (to-tallsystemet).
- Eksempel

$$(101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (?)_{10}$$

$$(101)_2 = (4)_{10} + (0)_{10} + (1)_{10}$$

$$(101)_2 = (5)_{10}$$

Oktale tall

- Et oktalt tall er representert ved symbolene 0, 1, 2, ... 7.
- Kodingen er posisjonsbetinget med grunntall 8
- Eksempel

$$(252)_8 = 2 \cdot 8^2 + 5 \cdot 8^1 + 2 \cdot 8^0$$

$$(252)_8 = (128)_{10} + (40)_{10} + (2)_{10}$$

$$(252)_8 = (170)_{10}$$

Heksadesimale tall

- Et heksadesimalt tall er representert ved symbolene 0, 1, 2, ... 8, 9, A, B, C, D, E, F
- Kodingen er posisjonsbetinget med grunntall 16
- Eksempel

$$(2B9)_{16} = 2 \cdot 16^2 + 11 \cdot 16^1 + 9 \cdot 16^0$$

$$(2B9)_{16} = (697)_{10}$$

Heksadesimal	Desimal	Oktal	Binær
00	00	00	00000
01	01	01	00001
02	02	02	00010
03	03	03	00011
04	04	04	00100
05	05	05	00101
06	06	06	00110
07	07	07	00111
08	08	10	01000
09	09	11	01001
0A	10	12	01010
0B	11	13	01011
0C	12	14	01100
0D	13	15	01101
0E	14	16	01110
0F	15	17	01111
10	16	20	10000
11	17	21	10001
12	18	22	10010
13	19	23	10011
14	20	24	10100

Konvertering fra grunntall "r" til desimaltall

- Generelt

$$(...a_2a_1a_0,a_{-1}a_{-2}...)_r = ... + a_2 \cdot r^2 + a_1 \cdot r^1 + a_0 \cdot r^0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + ...$$

- Eksempel

$$(1A5,1C)_{16} = 1 \cdot 16^2 + 10 \cdot 16^1 + 5 \cdot 16^0 + 1 \cdot 16^{-1} + 12 \cdot 16^{-2}$$

$$(1A5,1C)_{16} = (421,1133)_{10}$$

Konvertering fra desimal til binær

To metoder:

- Metode 1:
  - Finn den største potens av 2 som passer, trekk fra, og repeter.
- Metode 2:
  - Del tallet på 2
  - Resten etter divisjon, multiplisert med 2 blir *lsb (least significant bit)*
  - Del det nye tallet på 2
  - Resten etter divisjon, multiplisert med 2 blir neste bit
  - Osv.

Eksempel:
Konvertering fra desimal til binær

Konverter tallet $(41)_{10}$ til binærtall

41/2	=	20 + 1/2	a0 = 1	lsb
20/2	=	10 + 0/2	a1 = 0	
10/2	=	5 + 0/2	a2 = 0	
5/2	=	2 + 1/2	a3 = 1	
2/2	=	1 + 0/2	a4 = 0	
1/2	=	0 + 1/2	a5 = 1	

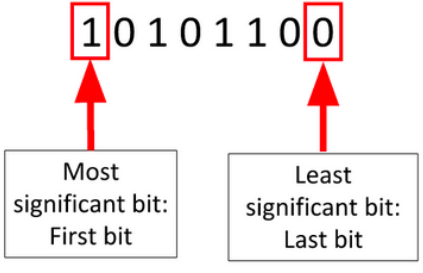
Dermed: $(41)_{10} = (101001)_2$

Konvertering fra desimal til det generelle grunntallet ”r”

- Gjenta prosedyren fra forrige side, men bytt ut grunntallet 2 med r. Resten multiplisert med r blir det aktuelle sifferet.

Noen navn

- Byte - gruppe på 8 bit.
  - Størrelsen på filer og objekter i data-hukommelsen blir vanligvis oppgitt i antall bytes.
- Chunks / words
  - Referer til word størrelsen som en bestemt datamaskinariktektur opererer på, f.eks. 32-bit eller 64-bit.
- Most og least signifikante bit. (msb og lsb)
- Most og least signifikante byte. (MSB og LSB)



Noen vanlige størrelser

- 2^{10} = 1 kilo ≈ 1000 (1024)
- 2^{20} = 1 mega ≈ 1 million (1,048,576)
- 2^{30} = 1 giga ≈ 1 billion (1,073,741,824)

- Så kB = 1024 byte (følger ikke SI-standard)
  - 1 KiB = 1024 B (IEC1998)

Addisjon

- Desimaltall

```
   11  ← carries
   3734
+  5168
------
   8902
```

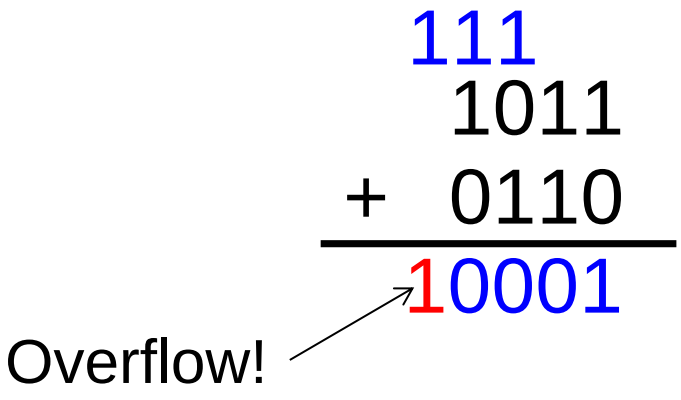
- Binærtall

```
   11  ← carries
   1011
+  0011
------
   1110
```

Prosedyren for binær addisjon er identisk med prosedyren for desimal addisjon

Overflow

- Digitale systemer opererer med et statisk antall bit.
- Overflow: når resultatet er for stort til å passe inn det tilgjengelig antall bit.



Negative tall / subtraksjon ?

- Hvordan representere negative tall?
- Og hvordan håndterer vi subtraksjon?

Se på to metoder

- Dele opp i Sign/Magnitude
- 2’er komplement (mest vanlig)

Sign/Magnitude representasjon

- Deler tallet i to
  - 1 fortegn bit, *N*-1 magnitude bits (101010….010)
- Fortegn bit er *msb*
  - Positivt tall: fortegn bit = 0
  - Negativt tall: fortegn bit = 1
- Eksempel: 4-bit representasjon av ± 6

  +6 = 0110

  -6 = 1110

Sign/Magnitude representasjon

To problemer:

- Summering fungerer ikke, for eksempel -6 + 6:

```
  1110
+ 0110
------
 10100 (feil!)
```

- Vi har også 2 representasjoner av 0 ($\pm$ 0):

```
1000
0000
```

2’er komplement

- Har ikke problemene til sign/magnetude metoden.
  - Summering fungerer
  - Kun en representasjon av 0
  - Den mest brukte metoden.

2’er komplement

- Det mest signifikante bit’et indikerer fortsatt fortegnet.

- Eksempel
  - 4 bit kan representere tallene –8 til +7

7	0 1 1 1
6	0 1 1 0
5	0 1 0 1
4	0 1 0 0
3	0 0 1 1
2	0 0 1 0
1	0 0 0 1
0	0 0 0 0
-1	1 1 1 1
-2	1 1 1 0
-3	1 1 0 1
-4	1 1 0 0
-5	1 0 1 1
-6	1 0 1 0
-7	1 0 0 1
-8	1 0 0 0

«Ta en» 2'er komplement

Forandrer fortegnet til et tall ved å invertere alle bittene og plusse på 1

Eksempel:

```
        1010
+       0001
-----------
-5  =   1011
```

Merk: dette fungerer begge veier.

Men hva skjer med -8?

7	0 1 1 1
6	0 1 1 0
5	0 1 0 1
4	0 1 0 0
3	0 0 1 1
2	0 0 1 0
1	0 0 0 1
0	0 0 0 0
-1	1 1 1 1
-2	1 1 1 0
-3	1 1 0 1
-4	1 1 0 0
-5	1 0 1 1
-6	1 0 1 0
-7	1 0 0 1
-8	1 0 0 0

Konvertere 2'er komplement til desimaltall

- Ingen forandring for positive tall.
- For negative tall må vi først "flippe" til et positivt tall for å tolke størrelsen.

Eksempel: Konverter 1001_2 til desimaltall.

1. 0110 inverterer
2. \+ 1 legger til 1

   $0111_2 = 7_{10}$

Dvs. $1001_2 = -7_{10}$

Binær subtraksjon

- 2'er komplement representasjon gir en enkel fremgangsmåte for subtraksjon:
  - Adderer tallene på vanlig måte.

- Eksempel:

```
          1 1
          0 1 1 0          6
+         1 1 1 0      +  -2
-------------------     -----
=  (1)  0 1 0 0      =     4
```

Går ut

Betyr positivt tall

Binær subtraksjon

- Eksempel 2

```
      0011          3
+     1000      +  -8
-------------     ----
=     1011      =  -5
```



Oppgave med 2'er komplemente tall

- Adder 6 og -6

```
   0110
+  1010
=
```

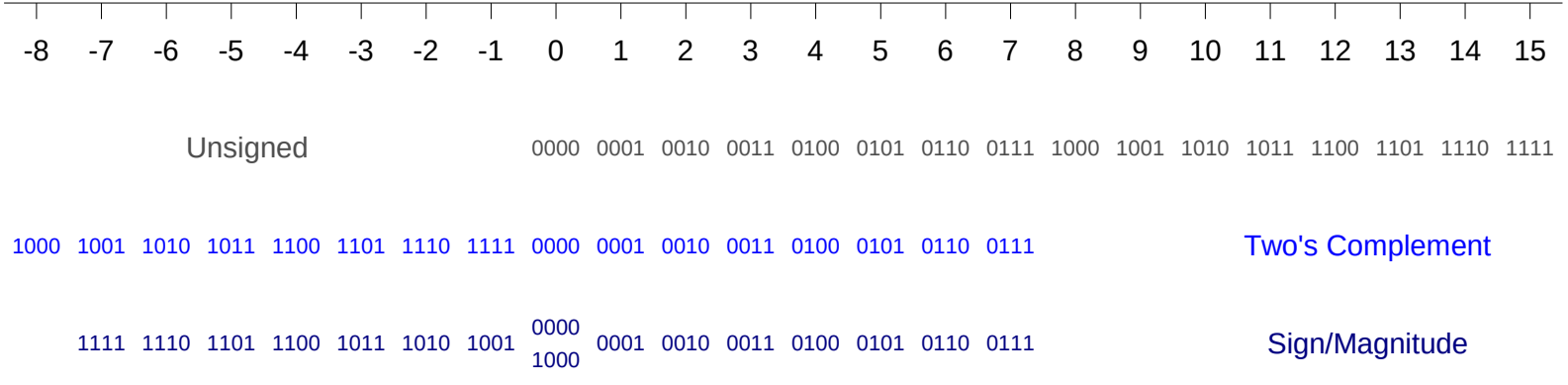
- Adder -2 og 3

```
   1110
+  0011
=
```

Tallsystem sammenligning

Number System	Range
Unsigned	$[0, 2^N-1]$
Sign/Magnitude	$[-(2^{N-1}-1), 2^{N-1}-1]$
Two's Complement	$[-2^{N-1}, 2^{N-1}-1]$

For 4-bit gir dette følgende tallrekker:



LOGISKE PORTER

Binær logikk - boolsk algebra

- Definerer:
  - Variablene: ”0” og ”1”
  - Og de binære basis funksjonene
    - AND ” • ”
    - OR ” + ”
    - NOT ” ´ ” (eller strek over, $A' = \bar{A}$)

Tre enkle operasjoner som utgjør basisen for *all digital logikk*.

Sannhetstabeller

AND

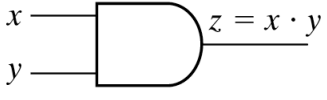
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

OR

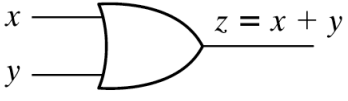
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

NOT

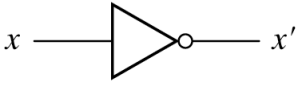
X	Y
0	1
1	0



(a) Two-input AND gate



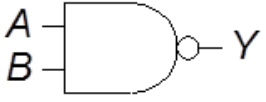
(b) Two-input OR gate



(c) NOT gate or inverter

Noen flere 2-inputs porter

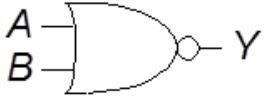
NAND



$Y = \overline{AB}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR

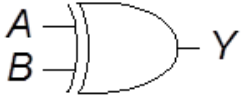


$Y = \overline{A + B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Noen flere 2-inputs porter forts.

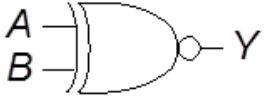
XOR



$Y = A \oplus B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

XNOR



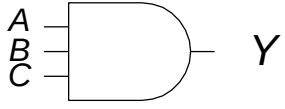
$Y = \overline{A \oplus B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

- Merk: Alle portene kan implementeres med AND, OR og NOT porter.

Flerinngangsporter

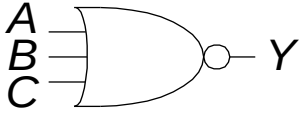
AND3



$Y = ABC$

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

NOR3

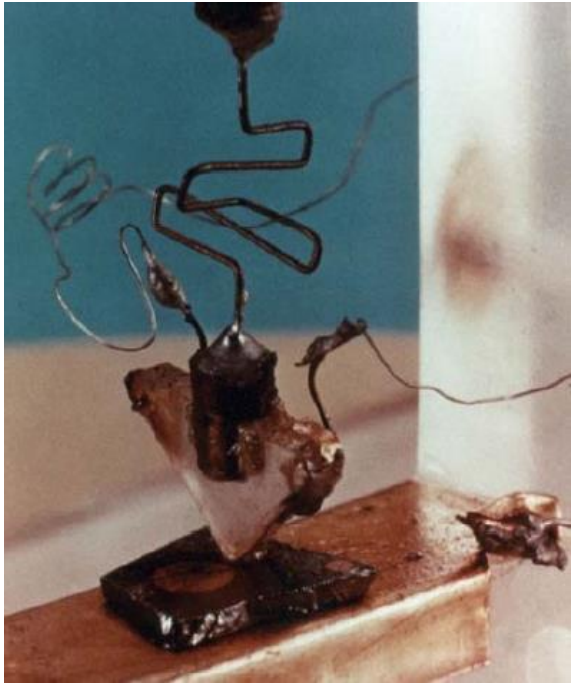


$Y = \overline{A+B+C}$

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

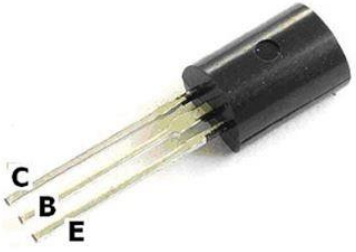
Hvordan implementeres disse portene?

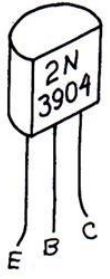
- Med *transistorer*.
- Første transistor be oppdaget ved et uhell i 1947.
- Erstattet store og strømkrevende radiorør.
- Kan fungere som av/på bryter.

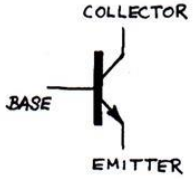


Transistor som strømbryter

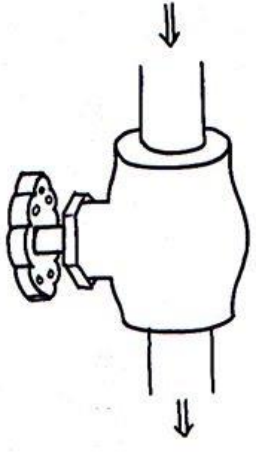
- Transistoren kan vi benytte for å slå av og på strøm.

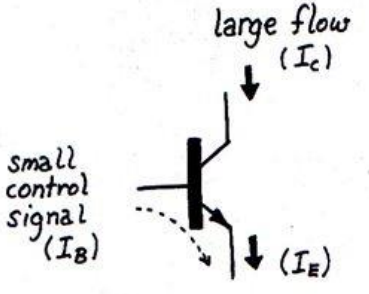






- Spenningen på B kontrollerer strømmen mellom C og E
- Tillsvarende som en vannkran.





Fra radiorør til integrerte kretser (mikrochip)

- Mindre størrelse
- Mindre strømforbruk
- Raskere!
- Billigere
- Mulighet for å lage store digitale systemer.
  - hva trengte man dette til?



1907



1947

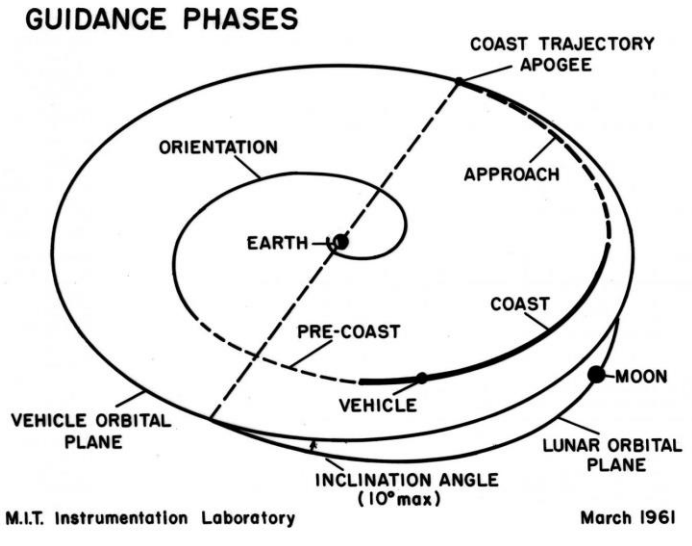


1963

Apollo – Hvordan navigere til månen?

- **Problem**: For vanskelig og kontraintuitivt for piloter å navigere/styre månelanding.
- De trengte en lett og kompakt maskin.
- Dette i en tid der datamaskiner
  - veldig store
  - innviklet å betjene (hullkort).

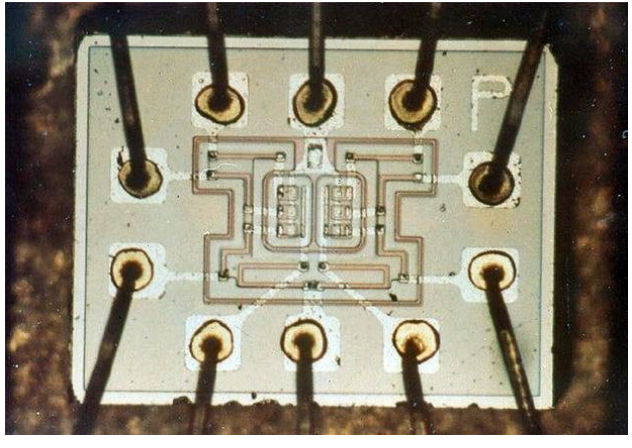
Du kan ikke bare peke på Månen og fly dit. Det krever nøyaktige baneberegninger basert på posisjon, hastighet og gravitasjon.



Løsning ~1966 - Apollo Guidance Computer

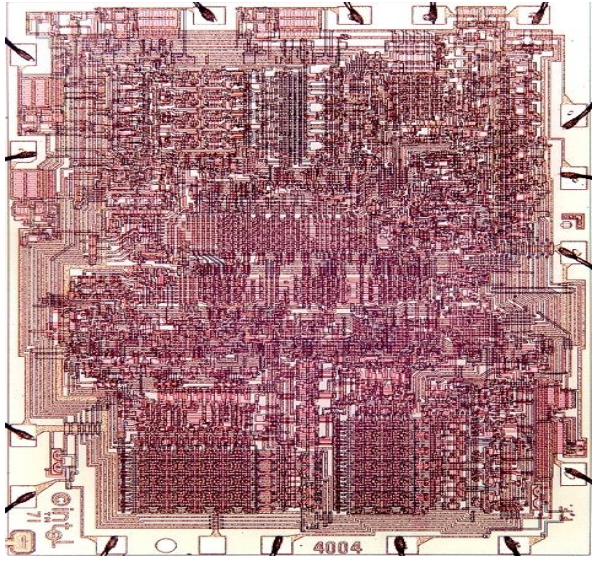
- Første datamaskin bygd med *integrerte kretser.*
- All logikk ble bygget av 3-inputs NOR porter. (Silicon Valley)
- Månelandingen ble en veldig god demonstrasjon for denne teknologien.
- Var med å starte en teknologisk revolusjon – **digital mikroelektronikk**.



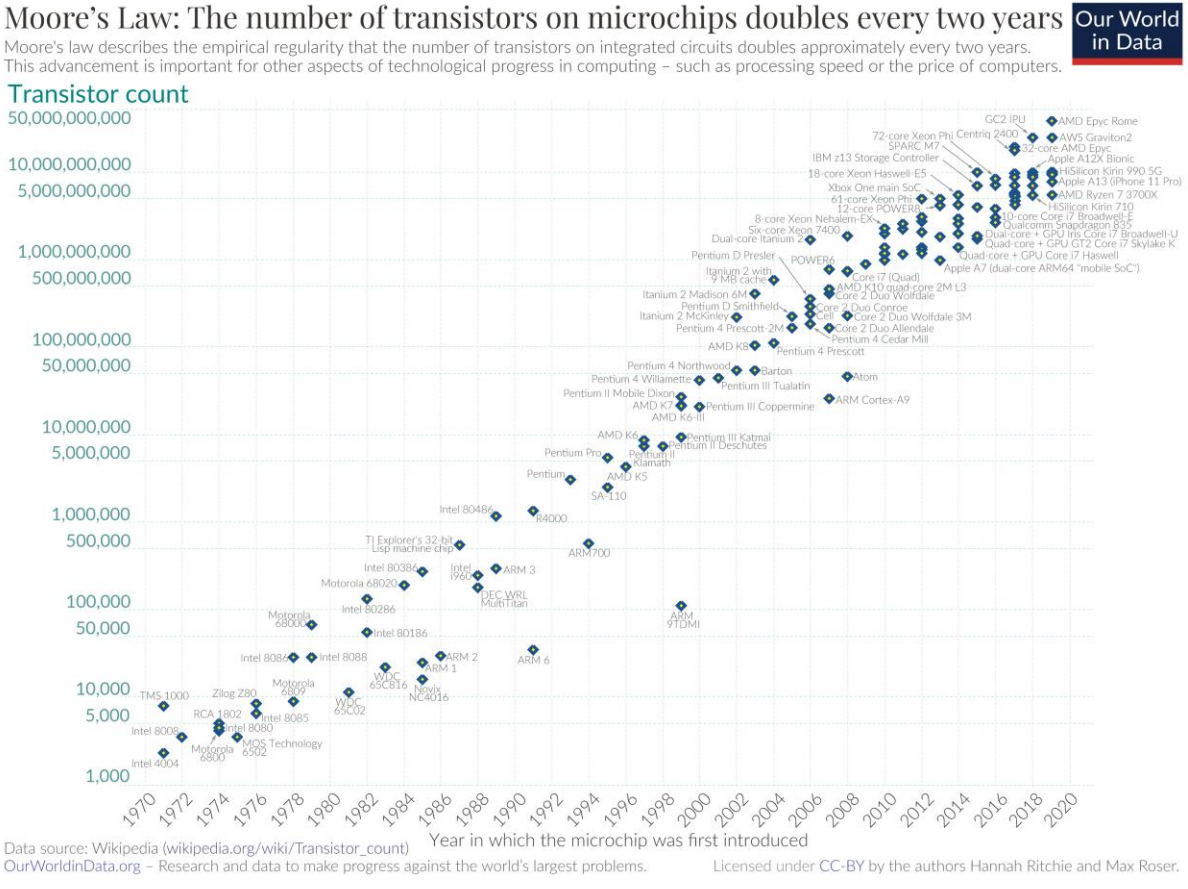


Intel 4004

- Første kommersielle tilgjengelige mikroprosessor (1971)
- 2300 transistorer
- 108kHz klokke
- Max 648 byte minne
- 4 bit bus



- Målet med IN2060 er å lære hvordan disse mikroprosessorene fungerer.



- Antall transistorer på et areal dobles hver 24. måned.
- Eksponentiell utvikling!

Neste forelesning

- Sette sammen de logiske portene til *kombinatorisk logikk*.
- Bruke boolsk algebra for å beskrive og forenkle kombinatorisk logikk.
- Viktig til neste gang
  - Forstå binære tall
  - Forstå de logiske portene AND, OR og NOT.
- Ikke nevnt så mye om fysiske detaljer
  - Elektriske spenninger, støy, strømforbruk og tidsforsinkelser.

Ukeoppgaver

- Exercies fra boka.
  - 1.7, 1.9, 1.12, 1.13, 1.17, 1.21, 1.23, 1.29, 1.33, 1.52, 1.60, 1.71, 1.74, 1.75
  - Ekstra: Prøv å lag en XOR port med AND, OR og NOT porter.