

Denne forelesningsøkten vil bli tatt opp og lagt ut i emnet i etterkant.

Hvis du ikke vil være med på opptaket:

 ^ Start Video	La være å delta med webkameraet ditt.
 ^ Unmute	La være å delta med mikrofonen din.
To: Marianne Sundby (Privately) Type message here...	Still spørsmål i Chat i stedet for som lyd. Hvis du ønsker kan spørsmålet også sendes privat til foreleser.



Høyskolen
Kristiania

Introduksjon

TK1100

Digital
teknologi

O'te forelesning

Gjennomgås i denne leksjonen

- Kursopplegget
- Datamaskinens bruksområder
- Datamaskinens bestanddeler
- Datamaskinen og PCens historie

Om Foreleser

Om Foreleser

Om Veiledere

Forbered deg på mye jobbing i TK fag!

- De første 4 leksjonene er det (for noen) vanskelig matematikk
- De neste 6 leksjonene er det MYE ny læring (TCP/IP ++)
- Det er viktig å være i alle forelesningene
- Dere må komme fysisk på øvingstimene, aktiv læring og interaksjon med medstudenter, lærer og veiledere er beste måten å lære!

Alle forelesningsdeck «slutter» midtveis – etter det er det samlet en del ekstra slider merket med ***For valgfritt egenstudie***. Disse slidene er ment for studenter som syntes dagens tema var spennende og vil lære mer, dette vil ikke være nødvendig for å bestå eksamen.

OBS: Anbefales sterkt for de mer tekniske studiene; Intelligente systemer, Programmering, og til en hvis grad Cybersikkerhet.

Kurs-opplegget

- 12 uker undervisning og øving
 - Undervisning kjøres i 3 grupper – og TK er på tirsdager
 - Gruppe 1 – Oslo: Forelesning T34-TAU-201 08.15 – 10.00, øving 10.15 – 12.00
 - Gruppe 2 – Oslo: Forelesning T34-TAU-201 15.15 – 17.00, øving 17.15 – 19.00
 - Gruppe 3 – Bergen: Forelesning BT-403 11.15 – 13.00, øving 13.15 – 15.00 (16. nov; 1 time senere)
 - Forelesning fysisk på Campus
 - Faget er bygget opp rundt 12 leksjoner med forelesning etterfulgt av øvingstimer
- Eksamens
 - 24 timer individuell hjemmeeksamen
 - Bestått / ikke bestått
- **Egenarbeid** er VIKTIG (aktiviser kunnskap!)

Egenarbeid

- Fagplanen sier 200 timer arbeid i dette emnet
- 48 timer forelesning og øving
- Feks 110 timer forbereitung og arbeid ifm forelesninger
- Feks 42 timer forbereitung til eksamen

Pensum?

- «Pensum» er fra Latin:
 - Pendere = en bestemt mengde ull man (typisk: en slave) skulle bearbeide i løpet av en dag.
 - M.a.o ARBEID som skal utføres
 - Pensum er alle **forelesninger** i emnet (det er ikke noen egen lærebok som må kjøpes)
 - Pensum er også å kunne besvare alle **oppgaver** som blir gitt i emnet i løpet av øvinger
 - Øvingsoppgavene er ofte en oppsummering av forelesning, og en del sett er multiple choice oppgaver av forskjellig oppbygging.
 - I tillegg vil det være en del mer åpne oppgaver, både som egenstudier og tekstoppgaver som ligner mer på hva dere får på eksamen
 - Der hvor det er relevant er det også praktiske øvingsoppgaver
-
- Pensum bok: INFORMATION TECHNOLOGY, SECOND EDITION (Richard Fox 2021)
 - Det ligger *kompendium* på emnesiden som kortfattet dekker flertallet av de viktigste kompetansemålene (forfattet av tidligere foreleser Bjørn Olav Listog

Pensum bok

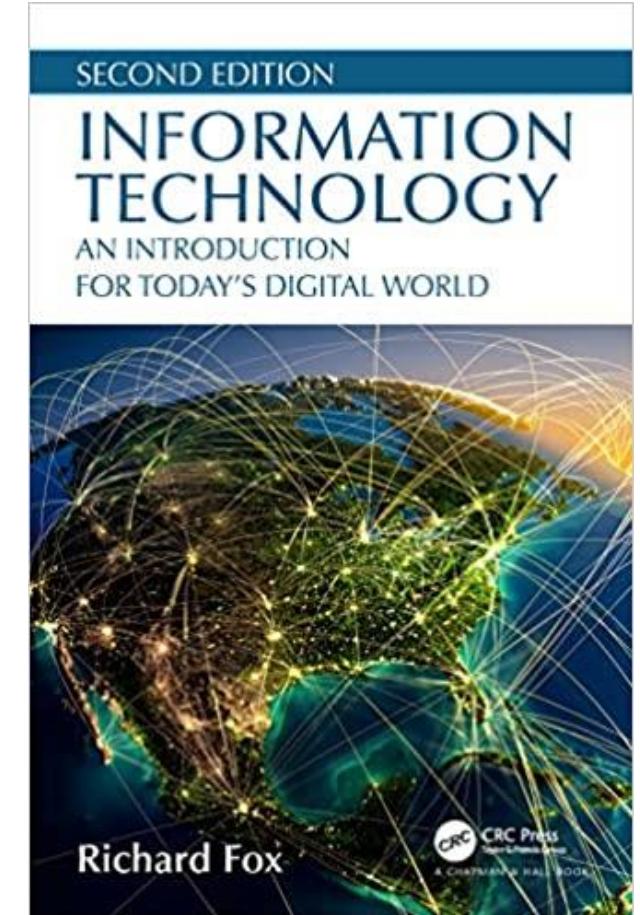
- INFORMATION TECHNOLOGY, SECOND EDITION
(Richard Fox 2021)

ISBN-13: 978-0367820213

ISBN-10: 0367820218

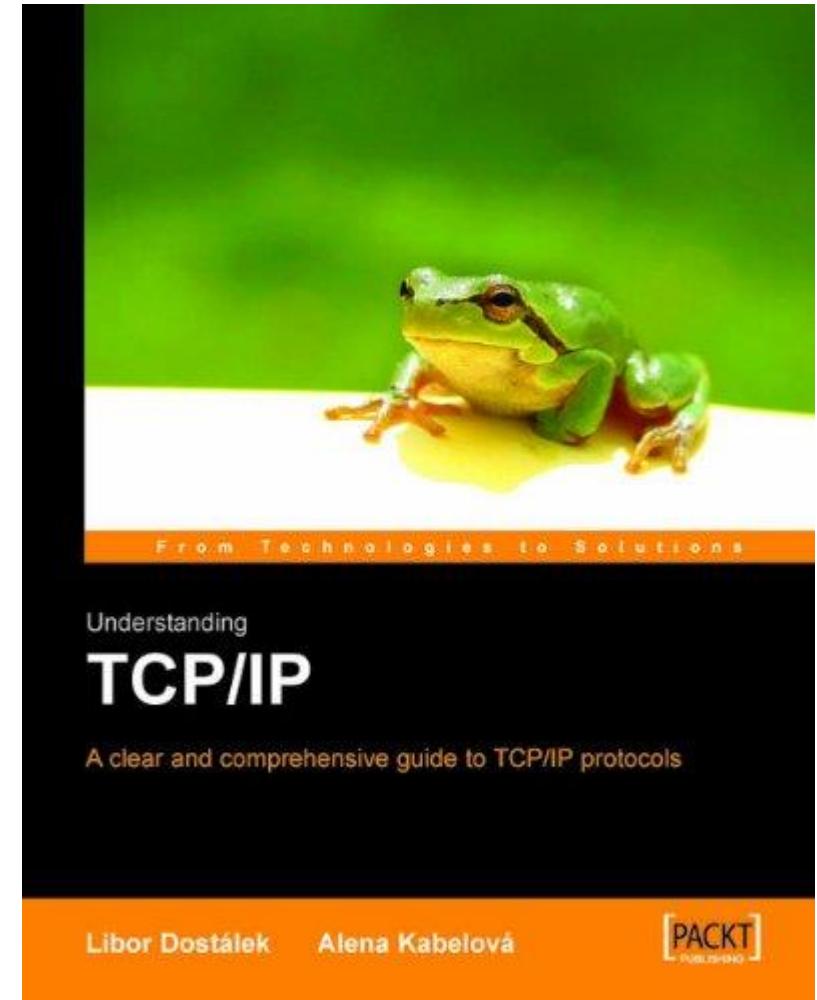
- Forelesningene er «pensum», så det vil ikke komme spørsmål som kun står i boken på eksamen
- Veldig god bok som dekker det meste av pensum; CPU and Memory, Storage and I/O, Binary Number System, Operating System, Computer Networks
- Kan kjøpes her:

<https://www.amazon.com/Information-Technology-Richard-Fox/dp/0367820218/>



Tilleggslitteratur

- Pensum boken er litt tynn på TCP/IP, de som er interessert i bok som går mer i dybden på nettverk kan finne mange gode bøker på Amazon
- Jeg anbefaler denne for de som studerer intelligente systemer og programering (og til de som ønsker å jobbe teknisk med Cybersikkerhet):
 - Understanding TCP/IP
- Men merk at denne igjen går mye dypere enn det som er pensum i dette faget. Ganske rimelig bok, men kun for de som ønsker å gå mer i dybden



Samarbeid utenfor klasserommet

- Pausene og veiledningstimene er beste kanal for «en til en» spørsmål
- På Canvas er det en samarbeidsmodul som heter «Diskusjoner», still spørsmål der og både medstudenter, veiledere og foreleser kan svare – og helst ønsker vi å få til en diskusjon rundt temaene
- Mattermost er også en kanal for å diskutere med hverandre og veiledere
- Hvis du er misfornøyd med noe ved skolen – ikke nøl med å ta kontakt med din tillitsvalgt, kan for mange føles tryggere enn å ta opp problemene direkte

Husk at veilederne får betalt for å jobbe i veiledningstimene, ikke forvent svar fra veiledere når du sender spørsmål klokken 23.51 på en lørdagskveld... ☺

Komptansemål

TEMA i emnet TK1100

Digital representasjon av data

- Tall, tekst, lyd, bilder, ...
- primitive datatyper (de maskinvaren støtter direkte: boolsk, heltall og flyttall)
- Alt annet representeres ved hjelp av de primitive typene og forhåndsdefinerte formater/standarder (= KODING!)

Maskin-arkitektur og –organisering

- Oppbygningen av maskinvaren og gangen i beregningene

Operativsystemet

- Hva, hvorfor og hvordan

Internett

- Oppbygging og protokoller

Oversikt over forelesninger

- Introduksjon til emnet (31. august)
- 1: Binære data (7. september)
- 2: Tegnsett, enkoding og media (14. september)
- 3: Datamaskin arkitektur (21. september)
- 4: Datamaskin oppbygging (28. september)
- 5: Operativsystem (5. oktober)
- 6: World Wide Web I (12. oktober)
- 7: Applikasjonslaget (19. oktober)
- 8: Transportlaget (26. oktober)
- 9: Nettverkslaget (2. november)
- 10: Linklaget (9. november)
- Repetisjonsforelesning (16. november)
- Eksamens

Forkortelser (bokstavkjeks)

- Det er mange forkortelser i data-verdenen!
- Og ofte benyttes samme ord som betegnelse på vidt forskjellige ting!

ISDN
RAID

SPOOL OS

CMOS

IBM

RAM

SNMP MB

HTML

CPU TCP

FAT

RISC

FORTRAN

Mål: Begreper

- For å beherske et fag må man (dessverre?) lære seg fagspråket
 - Bare slik kan man kommunisere med andre profesjonelle.
- Nivåer av kunnskap (forenklet)
 - i. Kunne ordene og forstå hva de betyr (**definere**)
 - ii. Kunne formulere egne påstander (teorier) og finne urimeligheter i andres (**forstå** teorier).
 - iii. Kunne lage ting selv (**anvende**).
 - iv. Kunne **vurdere** kvaliteten på andres arbeid.

Ex: Et begrep og en "teknikk"

- **System**
 - Et sett (en mengde) med komponenter som henger sammen på en slik måte at en endring i en komponent medfører en endring i en eller flere andre komponenter
- **Sort boks ("Black box")**
 - "Det vi tar for gitt, ikke bryr oss om hvordan innmaten fungerer i, og kun bryr oss om hva vi kan legge i og få ut av."
 - Påstand: De fleste brukere "black-boxer" PCen sin. (Ikke dere – lenger...)
 - Fakta: Jeg bruker bilen min som en black-box!

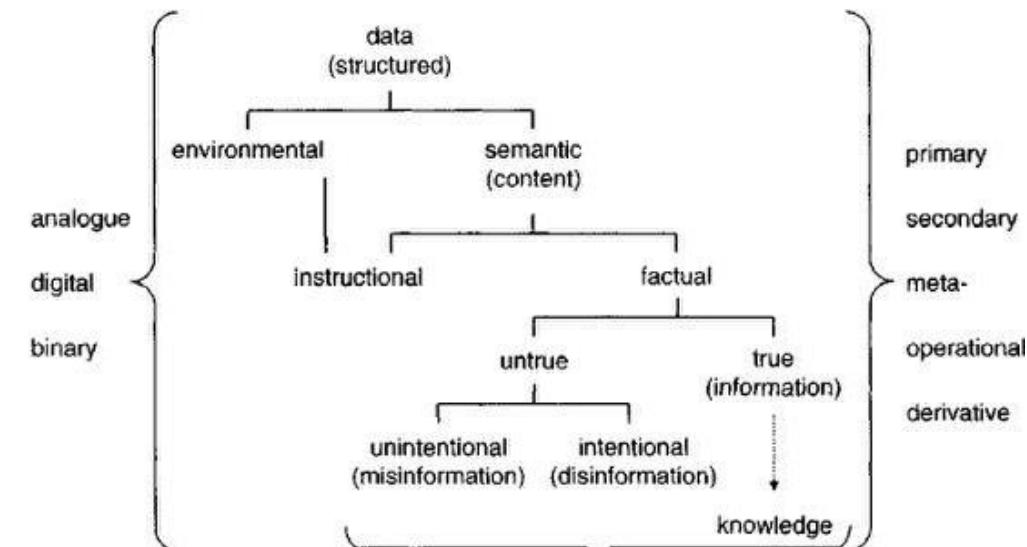
Data, informasjon, system

En **system**forståelse av et PC-system

- Ytelse er like mye å **flytte** data og instruksjoner raskt mellom komponenter, som å **prosessere** dem raskt
- Kjapp CPU på et hovedkort med trege og trange busser gir dermed (likevel) dårlig ytelse, og vice versa.
- Målet er et **balansert** system

Hva er informasjon?

- Brukes i mange mange forskjellige betydninger - to ytterpunkter er:
 - Informasjonsteoretisk (Shannon, 1948): Entropi (egen informasjon)
 $H = k \cdot \lg(N)$, N = antall tegn i tegnsettet, k antall tegn i meldingen.
 - Folkelig: data forstått i en sammenheng, "meningsfulle data"



Hva vi kommer til snakke om

- Generell bruk
 - Datamaskinen brukes innen mange forskjellige områder
- Elektronisk
 - Datamaskinen trenger elektrisitet for å fungere
- Digital
 - Datamaskinen er basert på binære logiske kretser (alt er **representert** som **tall** på bunnen)
- Menneske-laget
 - Datamaskinen er konstruert av mennesker for å gi resultater som mennesker er interessert i

Hovedmomenter med datamaskiner

- **Hastighet:** De utfører oppdrag meget fort
- **Pålitelighet:** De gjør ikke feil (bortsett fra når noe er galt med datamaskinen eller et menneske har gjort en tabbe, eller...!)
- **Lagringsevne:** De kan lagre store mengder informasjon (data) over lang tid
- **Pris:** De blir bedre og bedre til lavere pris
- **Størrelse:** De blir stadig mindre

Bruksområder for datamaskiner

- Brukes idag "overalt"
- Barnehager --> forskning
- Banker --> tungindustri
- Internett (email) --> mobiltelefon
- Trafikklys --> kjøleskap
-



Datamaskin (PC) med tilbehør



Digitalt?

Digitalisering

- I datamaskinen er alt representert som tall
 - Tall, bokstaver, bilder, lyd, film, instruksjoner...
- F.eks. så vil en MP3-fil være et langt tall som tolkes i forhold til et forhånds-definert format

Documents and Settings\blistog\Mine dokumenter\Min musikk\NiTimen.mp3																			
	0001	0203	0405	0607	0809	0A0B	0C0D	0E0F	0123456789ABCDEF	0x1920	10B4	473A	115A	FD5D	D2C8	CC5B	839C	A879	. ' G.. zý] ØÈÌ[fœ" y
0x1930	116D	4442	0C2F	267A	95EF	0FEB	0400	3B61		0x1940	5CE5	0E14	68AC	9A96	D963	CA0B	273D	F35D	.mDB./&z•i.ë...;a
0x1950	3E56	490D	9509	CD80	D1EB	A99E	5BDB	33A4		0x1960	BA21	2383	744B	7CBC	D4D0	E204	2DC7	CF2E	\å..h-š-ÙcÊ.'=6]
0x1970	EC25	E210	82F4	4085	5C5E	BEEF	120B	1AC6		0x1980	C457	44DD	EFB9	11BF	5121	FD37	E7C4	0708	>VI..•.í€Ñe@ž[Ù3*
0x1990	838C	B5C9	A0BE	B364	D440	0000	0000	08E4		0x19A0	7E0E	D12A	4CB5	9A7B	5103	B989	172B	8B98	°!#ftK ¾ÔÐâ.-çï.
0x19B0	D72C	49FA	3CAC	BC81	2889	69F3	484B	23D2		0x19C0	ABD3	E7FA	7277	FDBF	FFF8	FFF8	9264	4706	í®å., ð@...^\u00e3i...Æ
0x19D0	62DB	5350	A9E3	1AFO	4A49	7A1D	30A2	5E50		0x19E0	6D87	3EC4	A12D	C93A	AB68	A891	95B9	FFFE	ÄWDÝi.¿Q!ý7çÄ..
0x19F0	DEAF	BF66	A9C1	BD18	7C31	17D1	021B	406B		0x1A00	FFDF	A32B	FFDA	2018	7840	A868	5425	1F0D	~.Ñ*Lpš{Q.¹%..+<~
0x1A10	9923	4A64	8DC9	E046	358B	3378	3ADD	0C5A		0x1A20	1061	5DC4	3E69	11A8	E810	ACF3	4D52	D25C	×, Iú<-¾□(%i6HK#Ø
0x1A30	DE2D	00A1	064C	6456	F8BE	7329	9219	8E34										«Óçúrwýçýüýü'dG.	



Tallsystemer - desimaltall

- Desimaltall er basert på antall fingre (tær)
- Dusin, snes, og andre eldre måter å telle på
- Romertall
 - MCMLXXIV = 1974 (I=1, V=5, X=10, L=50, C=100, M=1000)
- 0 oppfunnet ca. 600 e. Kr. (indiske tall)
- Innført i Europa av Leonardo fra Pisa (Fibonacci) ca. 1200
- $1904 = 1*1000 + 9*100 + 0*10 + 4*1$
- $= 1*10^3 + 9*10^2 + 0*10^1 + 4*10^0$

Tallsystemer - binærtall

- Basert på to tilstander (digital; digit = finger eller tå)
 - finger oppe, finger nede
 - av, på (lavt signal, høyt signal)
 - 0, 1
- $25_{10} = 11001_2 = 1*16 + 1*8 + 0*4 + 0*2 + 1*1$
- $= 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0$
- Blir fort svært mange sifre
- Et binært siffer kalles **bit** (binary digit) (b)
- En gruppe på 8 binære sifre kalles **byte** (B)
 - (En byte består av to **nibble**)

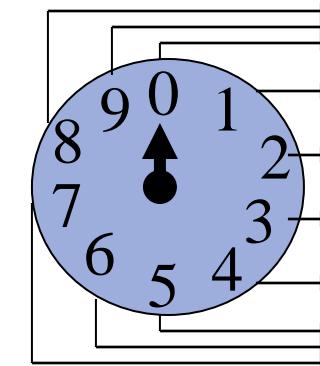
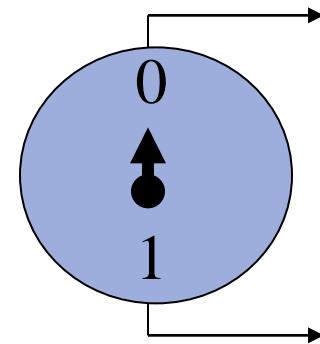
Å telle med binære tall

0	01	10	11
100	101	110	111
1000	1001	1010	1011
1100	1101	1110	1111

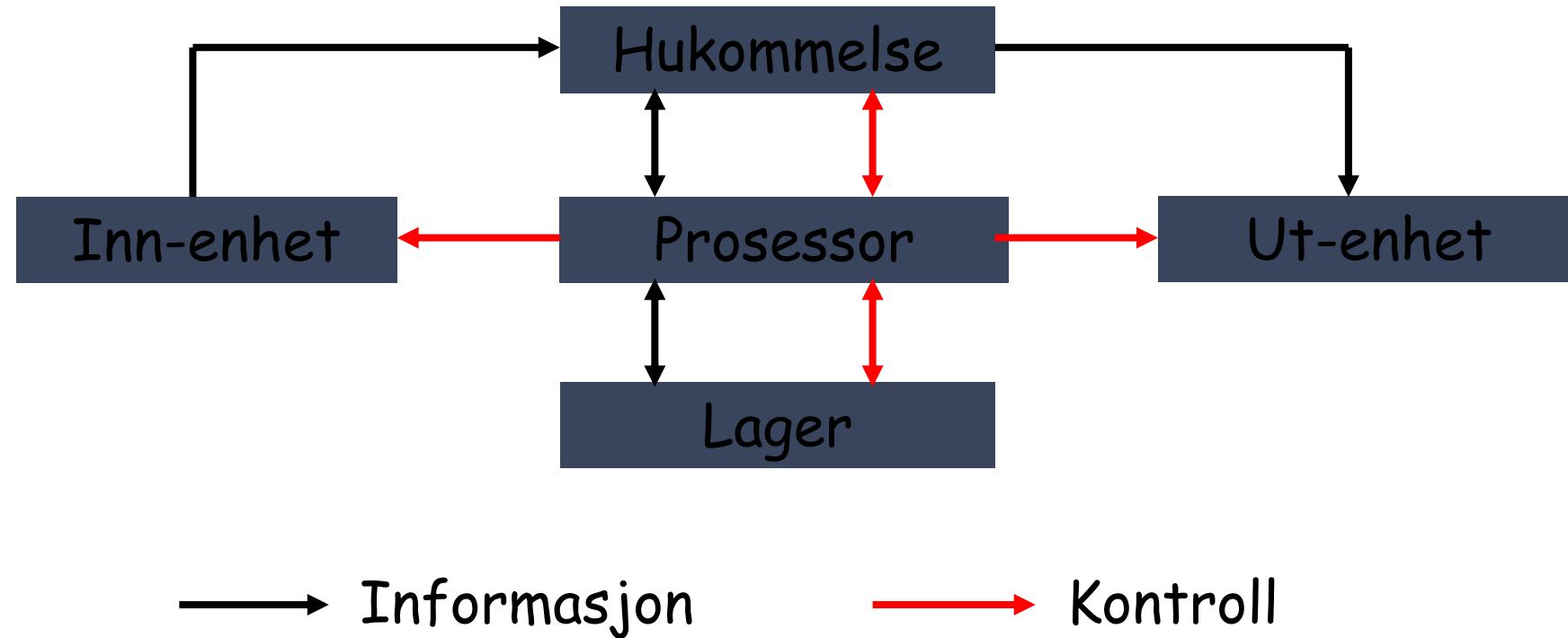
Computere

Hvorfor bruker datamaskiner binær logikk?

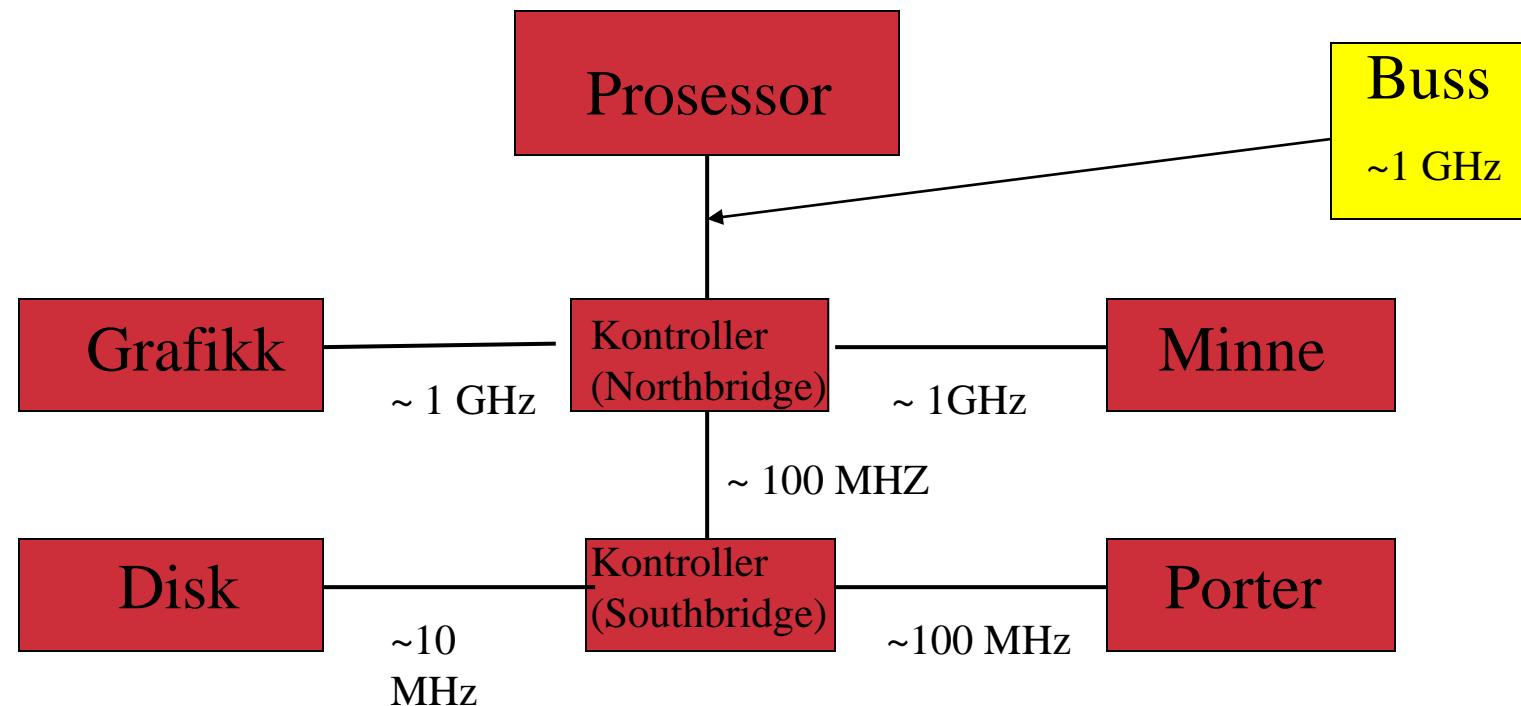
- Det binære systemet er enkelt og pålitelig
- Enkle kretser gir billige kretser
- Større kompleksitet i basis byggesteiner øker sannsynligheten for feil
- Det er det (teoretisk sett) mest effektive symbolsystemet som finnes!



Von Neumann arkitekturen



PCens hovedbestanddeler (eksempel)



Datamaskin - hovedinndeling

- Datasystemet kan tenkes delt i 2 hoved-deler
- Maskinvare (**hardware**)
 - *Elektroniske* og *elektro-mekaniske* komponenter som får computer-miljøet til å fungere
 - Input/Output, Lagring (HD, RAM), Beregning (CPU, GPU)
- Programvare (**software**)
 - **Applikasjoner** som utnytter de muligheter som maskinvaren gir for å utføre spesielle oppgaver
 - Applikasjoner er **programmer** som er laget i språk som datamaskinen "forstår"

Funksjonsorientert modell

- Lag 5 Brukerprogramnivå
- Lag 4 Kompilatornivå/interpreter
- Lag 3 Operativsystemnivå
- Lag 2 Instruksjonsnivå
- (Lag 1 Mikroinstruksjonsnivå)
- Lag 0 Digitalt *kretsnivå*

- Her fokuserer vi på hva slags oppgaver som løses på ulike nivåer.
- 0-2 er HW
- 3-5 er SW

Maskinvareorientert modell

Navn på nivået	Komponenter	IC tetthet	Informasjons-enheter	Tidsenheter
Portnivå (Gate)	Logiske kretser, Flip-Flops,..	SSI	Bit	$10^{-10} - 10^{-8}$ Sek
Registernivå	Registre, sekvensielle og kombinatoriske kretser	MSI	Ord (Words)	$10^{-9} - 10^{-6}$ Sek
Prosessornivå	CPU, Minne, busser, I/O	LSI/VLSI	Grupper av ord	$10^{-6} - 10^3$ Sek

Applikasjoner

- Tekstbehandling (Word, Notepad,)
- Regneark (Excel,....)
- Database-systemer (Oracle, Access,)
- Grafikk (PhotoEditor,)
- Nettleser (Chrome, Internet Explorer,)
- Elektronisk post (Outlook,)
- Spesial-programmer
 - Virus-sjekker, multi-media, programmerings-verktøy,

CPU - Central Processing Unit

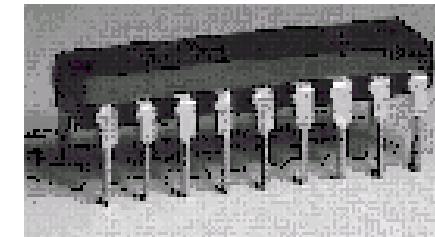
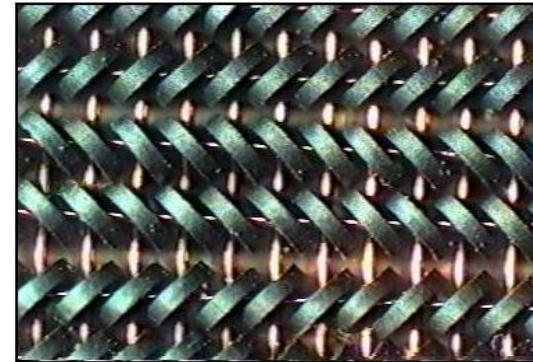
- **Prosessoren**
- Den "utførende" del av datamaskinen
- Kalles ofte datamaskinens «hjerne», men kan like gjerne oppfattes som en hovedmotor («regnemølle»).
- Består av kretser og elektronikk som kan utføre oppdrag (instruksjoner)
- Instruksjonene kan bearbeide data
- Begrepspar: **instruksjon** <-> **data**

Prosessorens virkemåte

- **Instruksjoner til prosessoren = programmering**
- Hentes inn i prosessoren fra minnet (RAM)
- Maskinspråk (IA32, IA64, m.fl.)
- Assemblerspråk (hver kommando tilsvarer en maskininstruksjon: `mov eax, [00AF3B13]`)
- Lavnivåspråk
 - C, C++ (og flere)
 - Kompileres over til maskin-instruksjoner
- Høynivåspråk
 - C#, JAVA (og veldig mange andre)
 - Interpreteres i et miljø (feks en virtual machine)
- Andre "språk"
 - Excel, SQL, skriptspråk

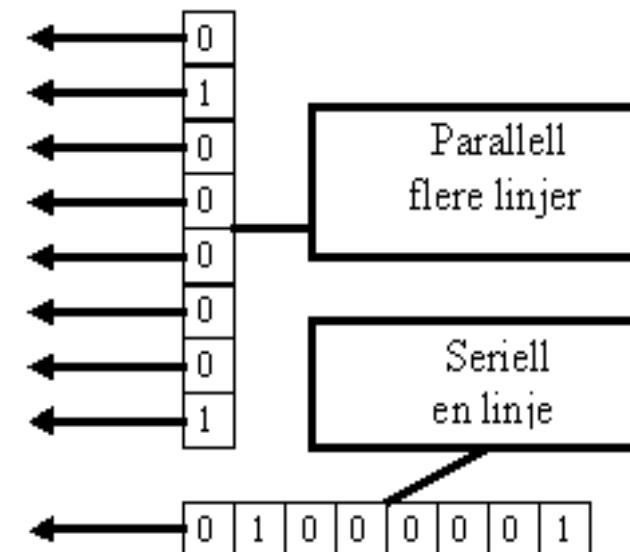
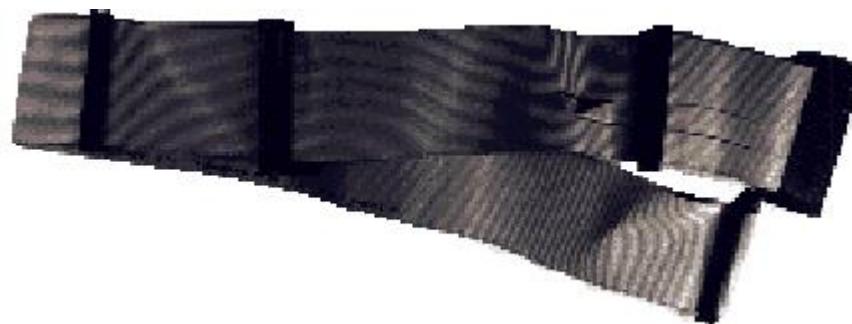
Minne

- Lagring av data og programmer
 - Ferritt-kjerner
 - RAM (Random Access Memory)
 - ROM (Read Only Memory), PROM, EPROM, Flash-RAM
 - «Virtuelt minne»
- Raskt mellomlager (cache)
 - Level 1 – internt (on-die, on-chip); ~KB
 - Level 2 – internt/eksternt; ~ MB
 - Level 3 – eksternt ~10 MB



Transportsystemet

- "Kabler" med parallelle "ledninger"
- Bredde * Hastighet = **Åndbredde/bitrate**
 - 64 bit (8 Byte) * 133 MHz = 1064 MiBps



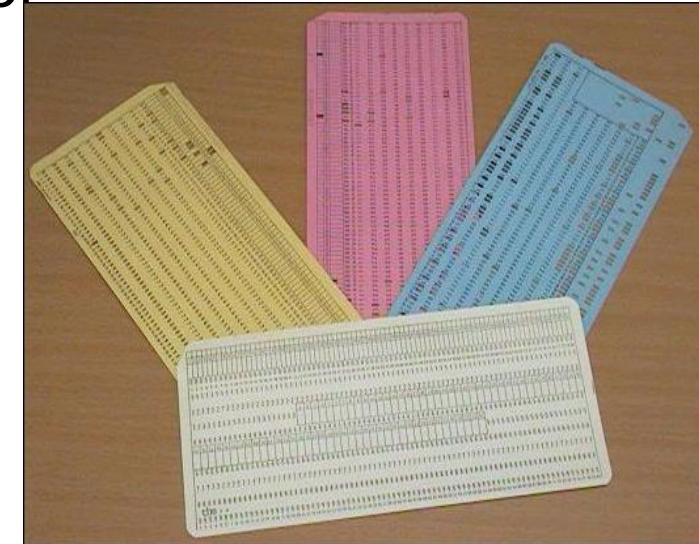
Masselager

- Brukes for permanent lagring av store datamengder

- Hullkort
- Hullbånd
- Magnetbånd (spolebånd, kassett)
- Diskett
- Disk
- USB stick
- CD, DVD

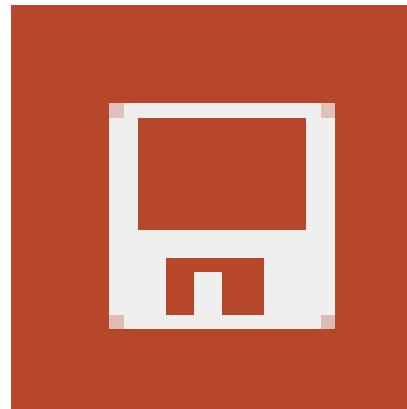
- Overføring til prosessoren via en kontroller

- IDE/EIDE, SATA, SCSI,...



Diskusjonsoppgave: Hva er dette?

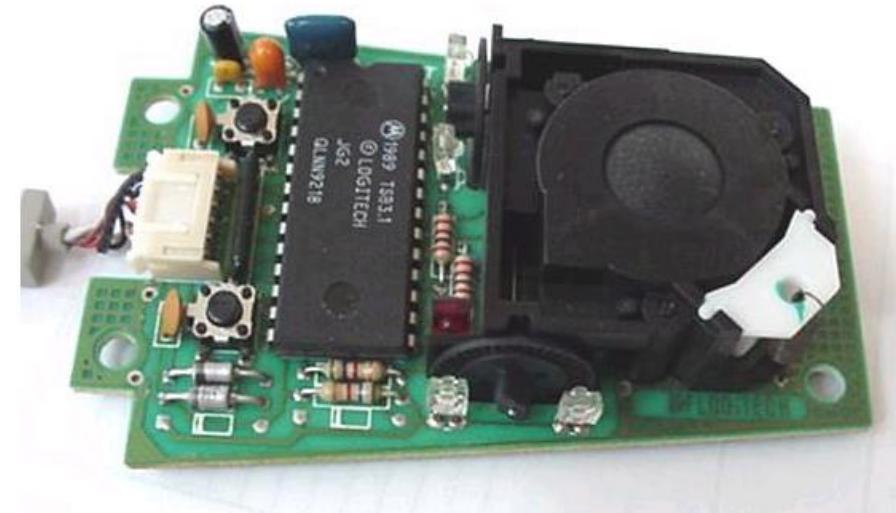
- Ikonet for lagring (Save) ser slik ut:



- Hva er dette bilde av, og hvorfor betyr det ikonet å lagre filen man har endret?

Periferiutstyr

- Skjerm
- Tastatur
- Mus
- Skriver
- Modem
- Annet utstyr
 - Høyttaler, mikrofon, joystick, plotter, scanner.....



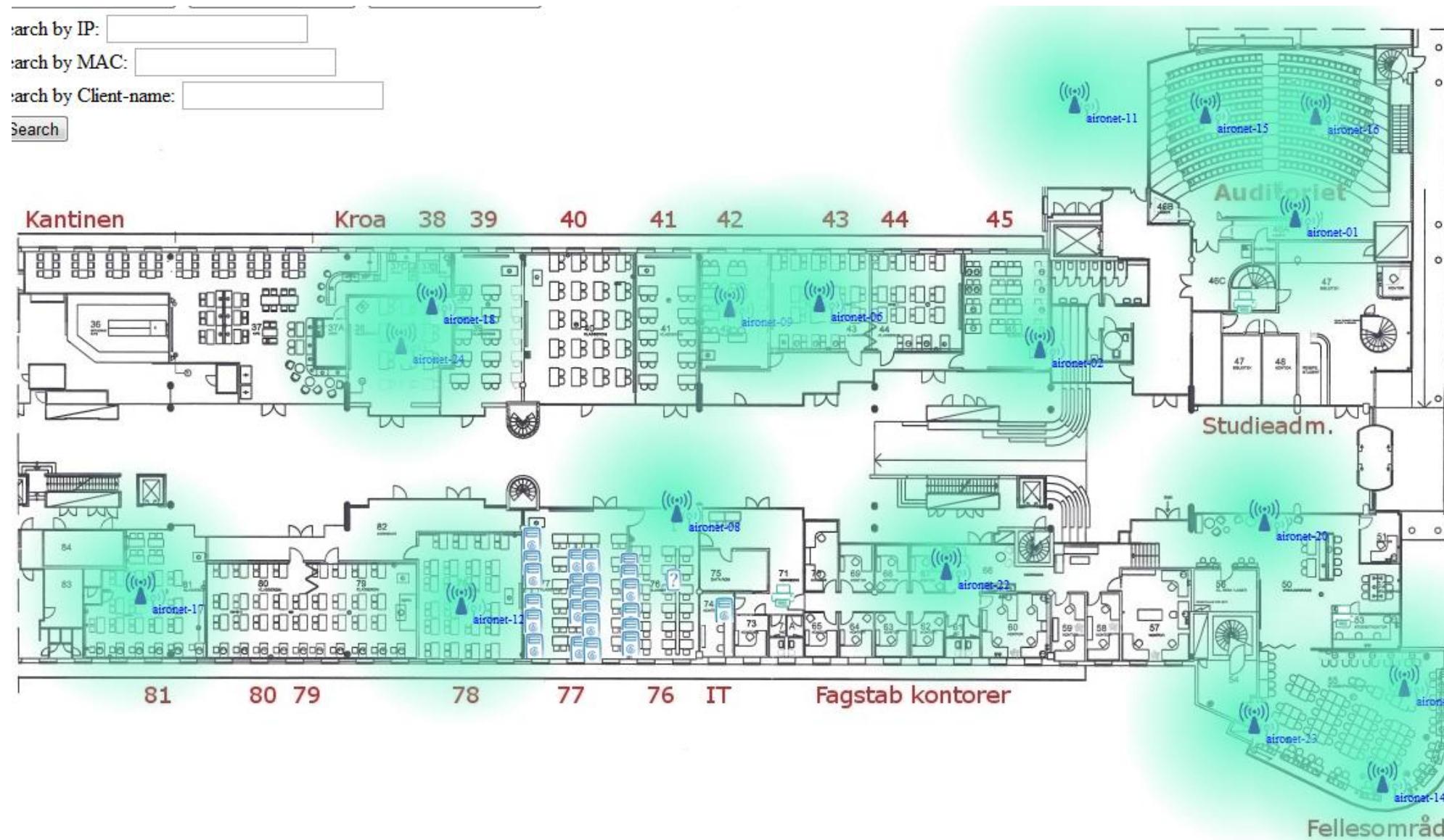
Operativsystemet

- Kjernen er det eneste som har full tilgang til Hardware!!!
- Gir et brukergrensesnitt
- Gir muligheter for sikkerhet og pålitelighet
- Kjører applikasjoner (tilbyr et API)
- Administrerer ressursene
 - Prosessor, hukommelse, eksterne lager, I/O-enheter
- Håndterer nettverk
- Ikke alle anvendelser av datamaskiner trenger et OS

Datamaskiner i nettverk

- Datamaskiner kan koples sammen i nettverk
 - Samme bygning = LAN (Local Area Network)
 - Hele verden = Internett (nettverk av nettverk)
- Kan overføre data mellom maskinene - protokoller
 - Tekst, bilder, musikk, mail,
- Strekker egne kabler (LAN), benytter generelt tilgjengelige linjer(telefon, kabel-tv,) eller bruker trådløse nett
- WWW (World Wide Web) er den mest brukte tjenesten på Internett = Filoverføring ihht HTTP-protokollen

WLAN skisse (fra gamle skolebygget)



Sluttema: Passord

Viktig sikkerhetstema: Passord

- En vanlig måte å måle kvaliteten på et passord er **bitstyrke**
- Uttrykker hvor mange forsøk en tilfeldig angriper **maximalt** trenger for å gjette ("brute force") passordet.
- Bitstyrke = $\lg_2(\text{forskjellige tegn mulige i passordet})^*\text{ antall tegn i passordet}$.
- F.eks. PIN-kode bruker 10 tegn (0-9) og 4 tegn => bitstyrke = $\lg_2(10)^*4 = 3,32^*4 = 13,28$
 - NB! Praktisk enhet pga **kombinatorisk eksplosjon**
 - $2^{\text{bitstyrke}} = \text{antall mulige passord som kan lages}$
- Anbefalt bitstyrke i våre dager er ca 80, mao ca tolv bokstaver og tegn!
 - I tillegg bør man selvsagt unngå alt som er knyttet til din egen person, alle vanlige ord (de som finnes i ordbøker) mm

“Sikre passord”

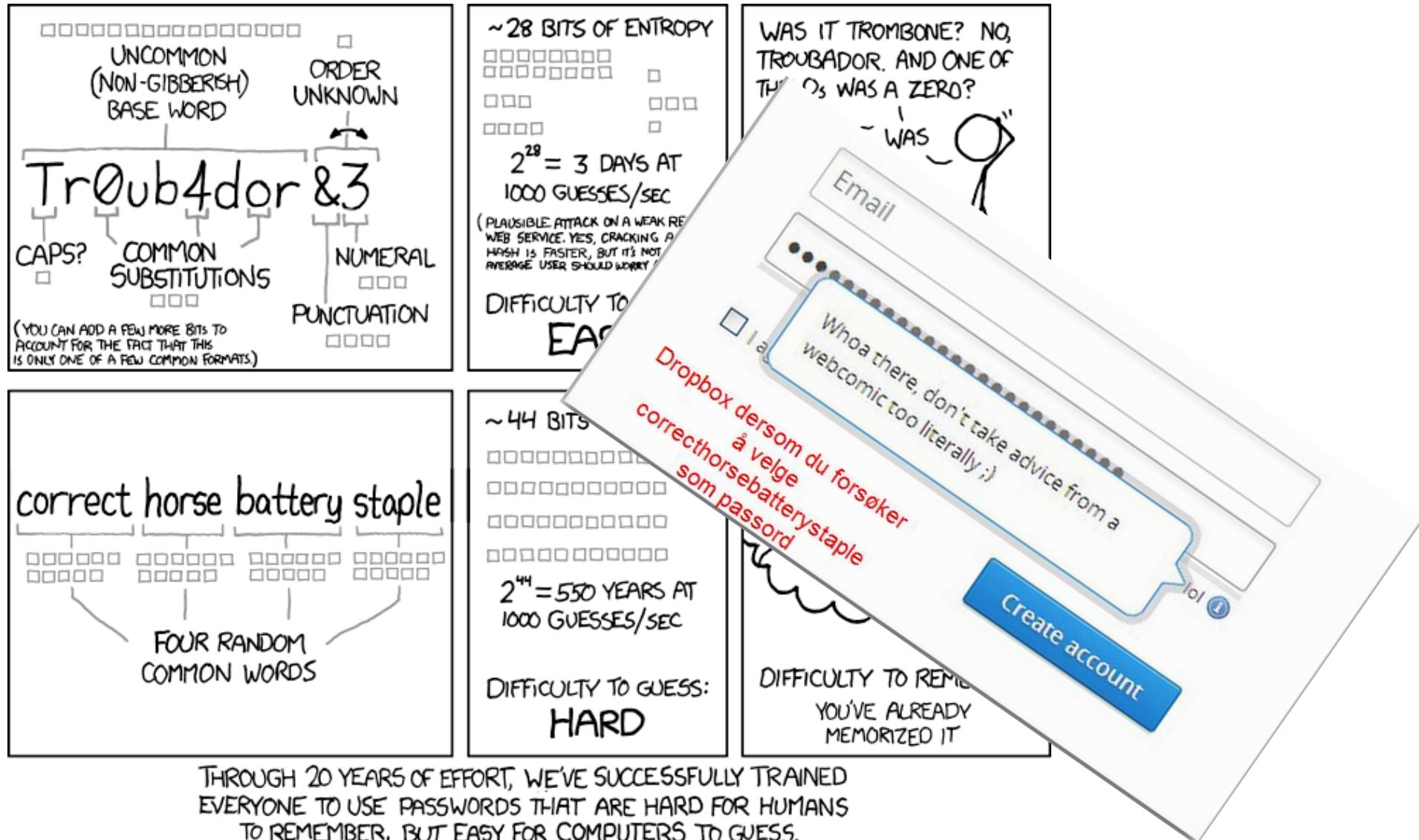
Bør inneholde tegn fra minst tre av gruppene under:

Group	Example
Lowercase letters	a, b, c, ...
Uppercase letters	A, B, C, ...
Numerals	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Non-alphanumeric (symbols)	() ` ~ ! @ # \$ % ^ & * - + = \ { } [] : ; " ' < > , . ? /
Unicode characters	€, Γ, f, and λ

Passfrase er enda bedre: "I re@lly want to buy 11 Dogs!"

Eller lær deg et fint dikt utenatt!!!

XKCD: kommenterer



Kombinatorisk eksplosjon?

AVSLUTNING

Oppsummering: «Teori»

- System
- Digitalisering
- Informasjon og data
- Data og Instruksjoner
- Hardware og software

Øvingstimer i emnet

- *Spørsmålsark (PDF), anbefales at dere jobber i grupper i øvingstimene*
- *Primært repetisjon av forelesningen for å sikre god innlæring*
- *En del øvinger er praktiske – gjør teorien litt lettere å forstå/huske*
- *Skriv ned svar – fyldig (dere trenger det om 3 mnd til eksamensforberedelser...)*
- *Søk forskjellige kilder på nettet for å lære temaene mer i dybden! Du må i dybden for å FORSTÅ temaet, ikke bare pugge foilene...*
- JA, det dukker opp «nytt stoff» i øvingene!!
- JA, dere kan få (direkte og indirekte) spørsmål fra øvingene på **eksamen**
 - Et par oppgaver på eksamen er praktiske oppgaver ☺

Neste uke

- Data-**representasjon**
- Tall-systemer /Boolsk algebra/Koding/dekoding
- Hjemmelekse
8-4-2-1, 8.4.2.1., 8 4 2 1 ->pugg!!!!!!!

128-64-32-16- 8- 4- 2- 1

1 0 1 0 0 1 1 0 =????

• Finn frem pen og papir!!!!

- Det meste av det vi gjør neste gang er mye enklere å få til på et ark, enn å forsøke å gjøre på «skjermen»

For valgfritt egenstudie

For de som ønsker å lære noen emner mer i dybden for å forstå det bedre er det her samlet noen ekstra temaer relatert til dagens undervisning, det må forventes en del egenarbeid for å forstå disse emnene.

Det vil ikke komme spørsmål på eksamen fra disse, og dette er altså ikke ansett for å være en del av pensum.

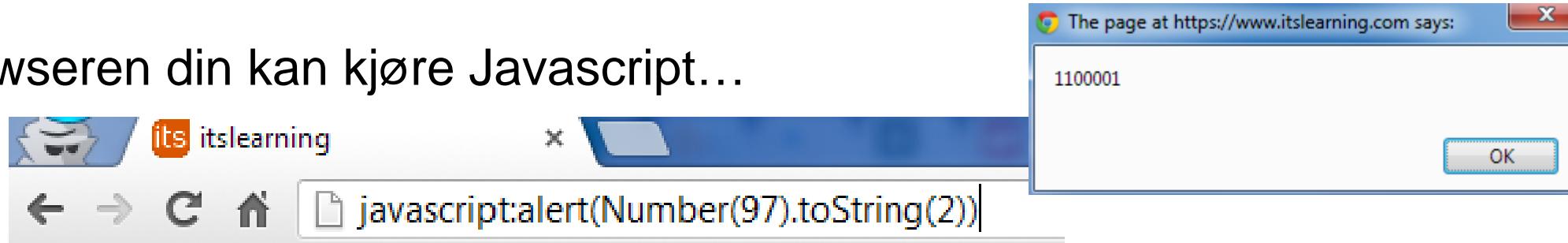
Flyttall

- Windows-kalkulatoren hevder at $(\sqrt{2})^2 - 2 = ?$
- Computere arbeider "alltid" med **endelig presisjon!!!**
- Flyttall er et kodingsformat!



To «binære tricks»

- Browseren din kan kjøre Javascript...



Enheter og størrelser

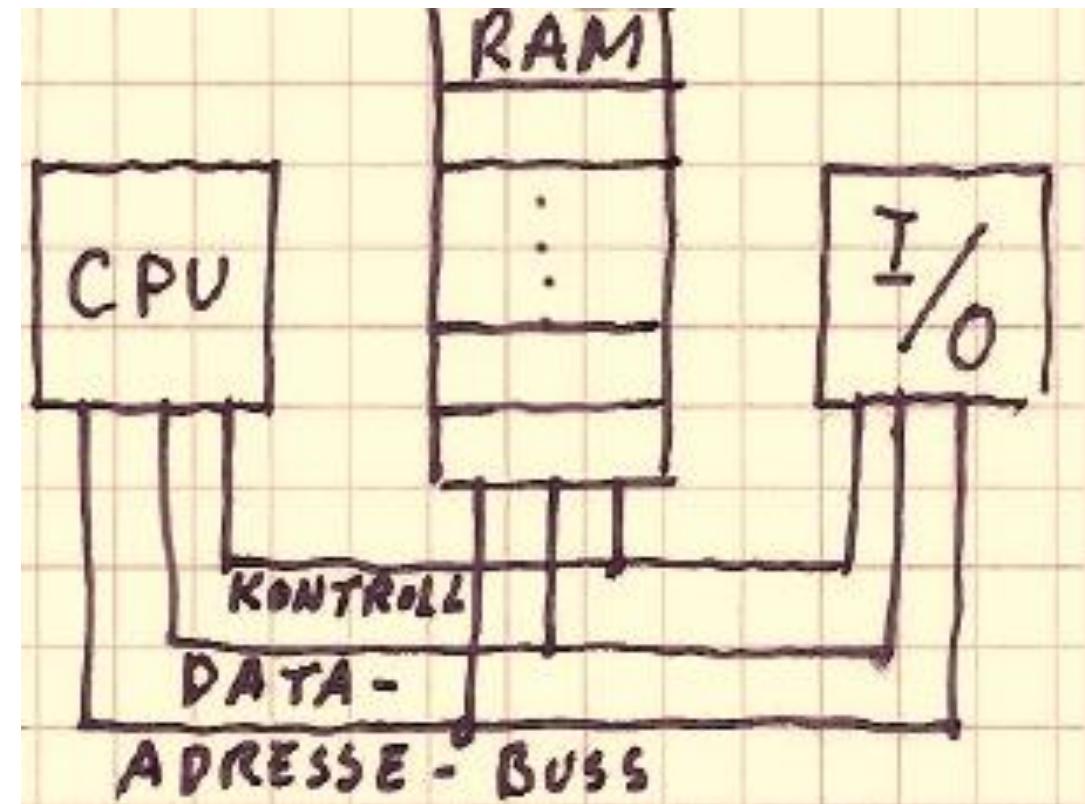
- **Bit (b)** - 0 eller 1
- **Byte (B)** = 8 bit
- $\text{Kilo} = 10^3 = 1000 \approx 1024 = 2^{10}$
 - 1 km = 1000 m, 1 mm = 1/1000 m
- For enkelhets skyld "jukser" vi litt !
 - k = 1000, Ki = 1024, (anta at K = 1024 også)
- Kilobyte/**KibiByte (KiB)** = 2^{10} byte = 1024 byte
- Megabyte/**MibiByte (MiB)** = 2^{20} byte = 1024 KB = 1048576 byte
- Gigabyte/**GibiByte (GiB)** = 2^{30} byte = 1024 MB = 1073741824 byte
- **Herz (Hz)** = hendelser pr. sekund
- **MIPS** = Mega instruksjoner pr sekund.
- **FLOPS** = Mega flyttalloperasjoner pr sekund
- **kbps** = 1000 bit / sekund (bitrate, «båndbredde»)

Multiples of bytes		v · d · e		
SI decimal prefixes		Binary	IEC binary prefixes	
Name (Symbol)	Value	usage	Name (Symbol)	Value
kilobyte (kB)	10^3	2^{10}	kibibyte (KiB)	2^{10}
megabyte (MB)	10^6	2^{20}	mebibyte (MiB)	2^{20}
gigabyte (GB)	10^9	2^{30}	gibibyte (GiB)	2^{30}
terabyte (TB)	10^{12}	2^{40}	tebibyte (TiB)	2^{40}
petabyte (PB)	10^{15}	2^{50}	pebibyte (PiB)	2^{50}
exabyte (EB)	10^{18}	2^{60}	exbibyte (EiB)	2^{60}
zettabyte (ZB)	10^{21}	2^{70}	zebibyte (ZiB)	2^{70}
yottabyte (YB)	10^{24}	2^{80}	yobibyte (YiB)	2^{80}

1 EiB tilsvarer
en 50.000 år
lang video
(DVD-kvalitet)!

Von Neumann (utvidet)

- Vi legger inn hovedtypene busser også
- Kun en prinsippskisse, men den hjelper deg å tenke klart



Litt Historie

- Den moderne datamaskinen kan ses på som en løsning på tre (historisk sett) forskjellige problemer:

1) Beregningsproblemet

- hvordan utføre kompliserte beregninger raskt og pålitelig

2) Massedataproblemet

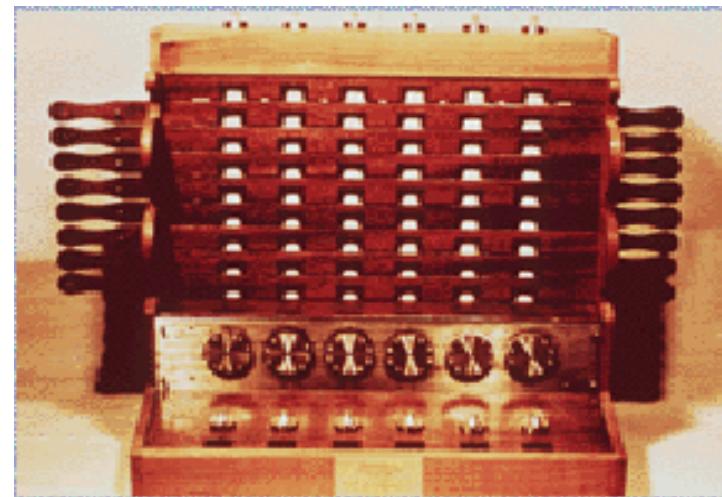
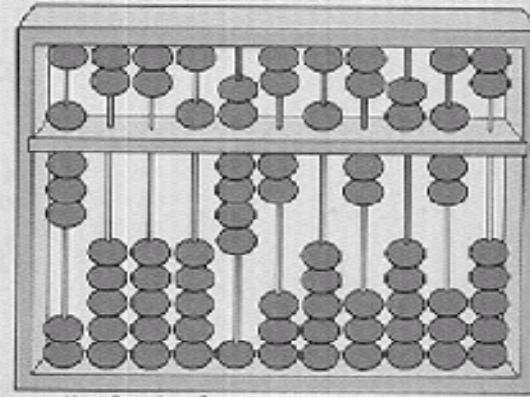
- Hvordan lagre og behandle store mengder data

3) Reguleringsproblemet

- Hvordan styre og automatiser industrielle o.a. prosesser

Regnemaskiner

- Abacus (ca -3500)
- "Tabeller"
 - John Napier, 1600 (staver)
 - Willian Oughtree, 1622 (regnestav)
- "Tannhjul-maskiner"
 - Pascal, 1642
 - Leibniz, 1694



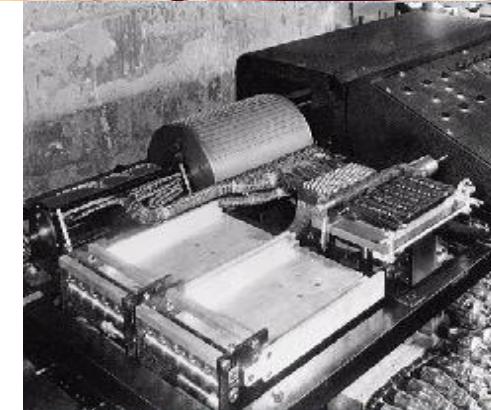
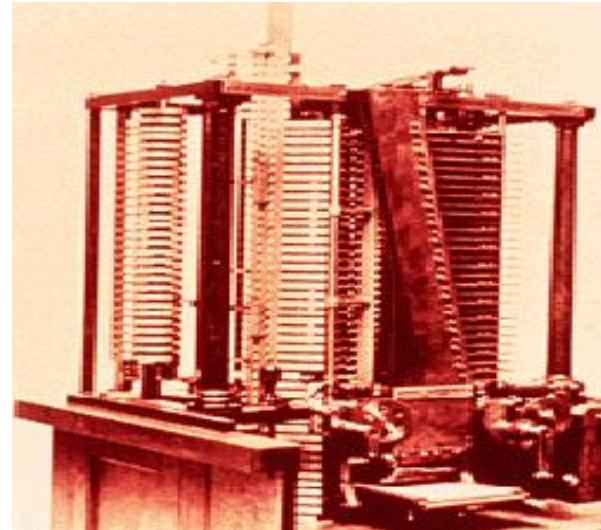
Regnemaskiner (2)

- **Mekaniske**

- Charles Babbage (1791 - 1871)
- Difference Engine - 1822
- Analytical Engine - 1833
- Augusta Ada King, countess of Lovelace

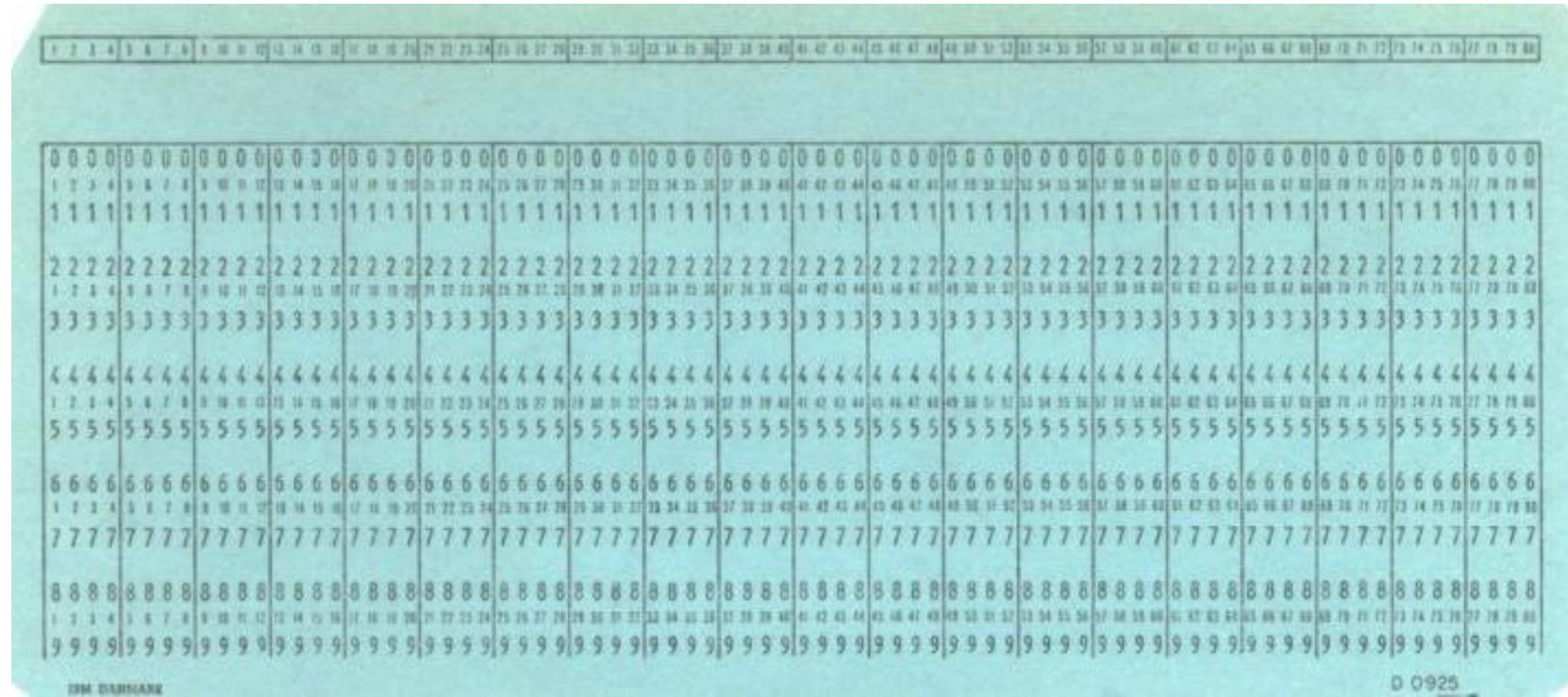
- **Elektromekaniske**

- Zuse, 1936-
- Atanasoff, 1940-



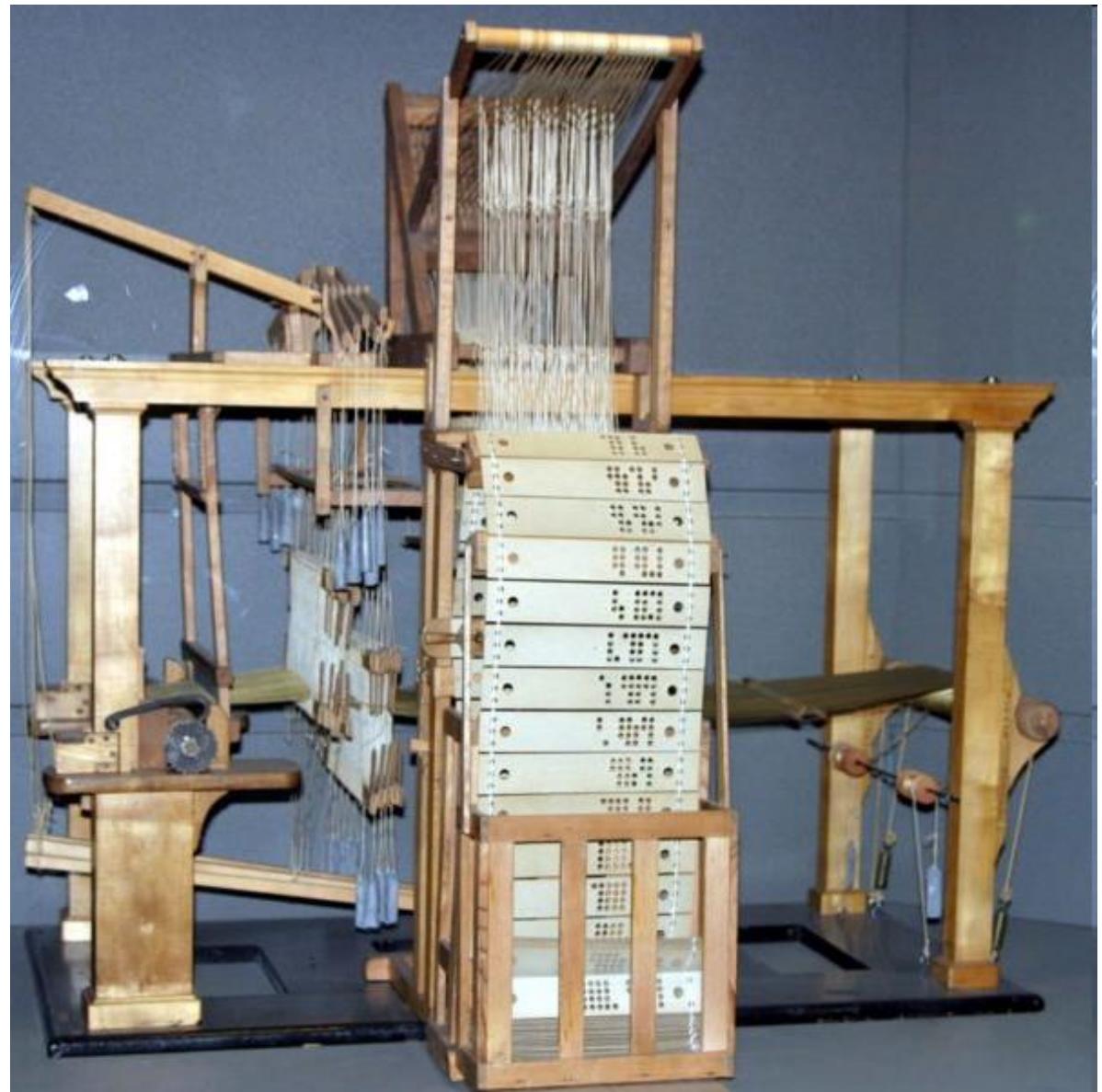
Massedata

- Herman Hollerith (hullkort) – 1890
 - Folketelling
 - Forløper til IBM



Regulering

- Jacquard-veven (ca. 1810)
 - hullkort/automatisering
- Maxwell, ca. 1880
 - teori/diff. lign for maskinstabilitet
- Norbert Wiener, 1946
 - kybernetikk (reguleringsteknikk)
- Mikrokontrollere overalt!
 - 1971 og utover

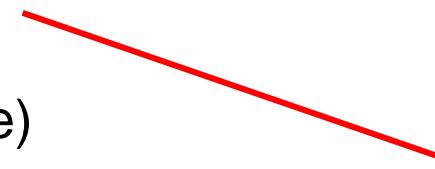


Typisk historisk fremstilling

- Inndeling i generasjoner/epoker basert på:
 - Ny hardware
 - Nye anvendelser
- «Koevolusjon av maskin- og myk-vare»

1. Generasjon (1940-1950)

- Elektromekaniske reléer og radiorør (vacuum tube)
- Conrad Zuse
 - Z1- 1936; Z4 – 1945
 - Plankalkül
- Alan Turing
 - Universell Turingmaskin - 1937
 - Colossus - 1943
- Howard H. Aiken
 - (Mark 1) - 1944
- John Presper Eckert & John W. Mauchly
 - ENIAC - 1944
- John von Neumann
 - EDVAC - 1945
 - UNIVAC - 1951



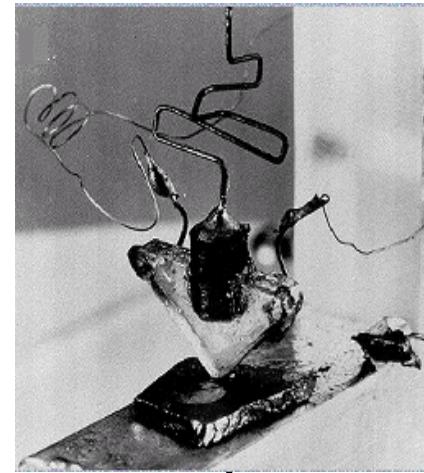
Sitat: Thomas Watson 1943

- Thomas Watson, president IBM, 1943:

“I think there is a world market for maybe five computers”

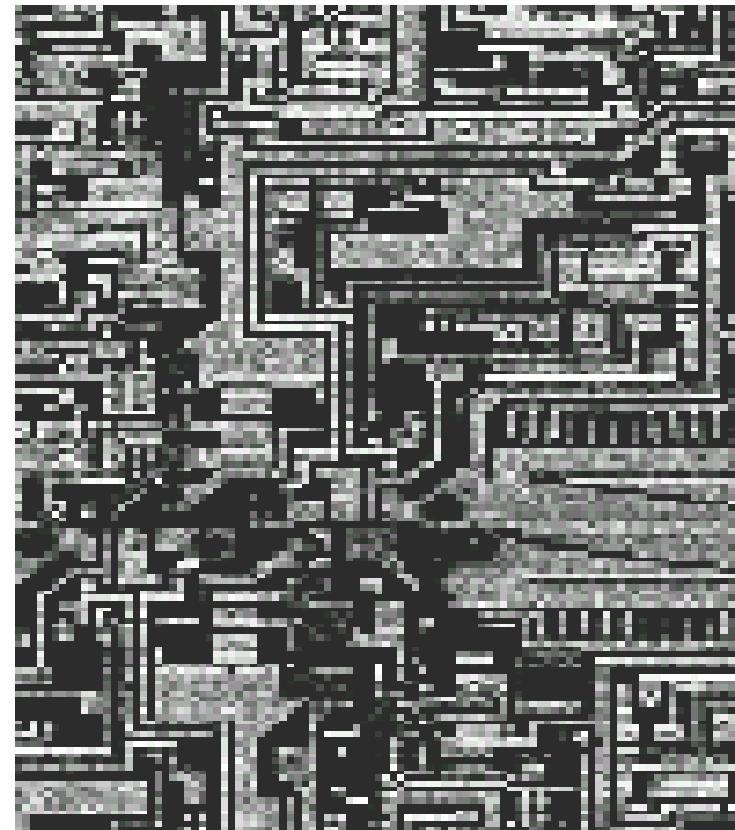
2. Generasjon (1950 - 1964)

- Transistor – 1947
- Den første kompilatoren (A-0, Grace Hopper) – 1951
- IBM 701 - 1953
- IBM og "de syv dvergene": Sperry-Rand, Burroughs, Control Data, Honeywell...
- Fortran - 1957
- Cobol, Lisp, Algol,...
- IBM 1401 - 1960. Datamaskinens "T-Ford"



3. Generasjon (1964 - 1971)

- Integrerert krets - 1958
- Basic (interpretert språk!) – 1964
- IBM 360 - 1964
- Intel – 1968
- Interaktive terminaler!!!
- ARPAnet - 1969
- PDP, DEC, Data General,..



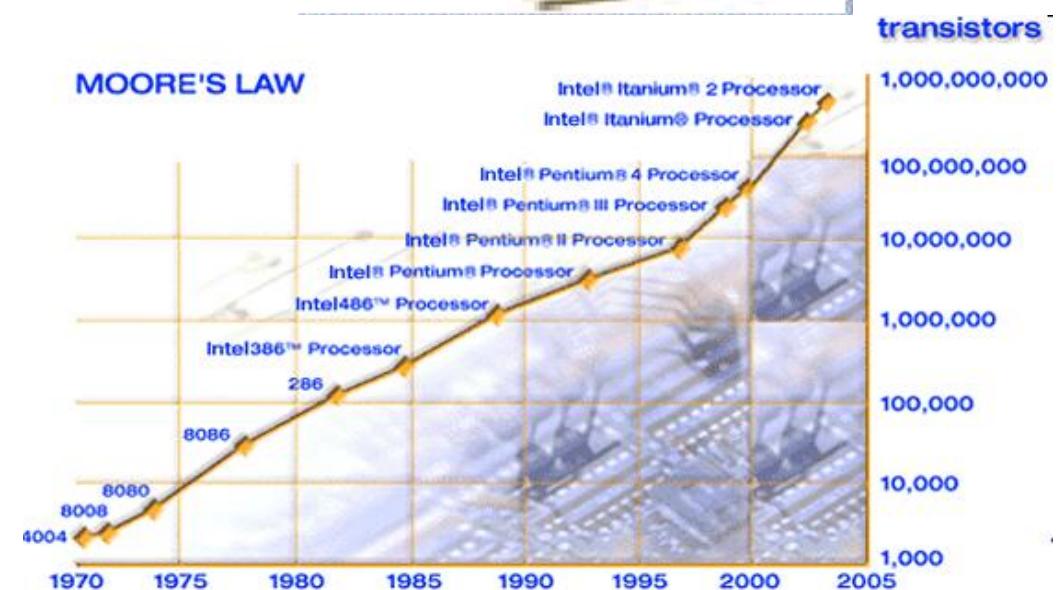
4. Generasjon (1971 -)

- **Mikroprosessor**
- Intel 4004 - 1971
- IBM 370 - 1971
 - MVS, VM
- UNIX - 1971
- Altair 8800 – 1975
- Apple, Radio Shack, Commodore
- VisiCalc - 1979
- WWW - 1991



Moderne PC (1981-)

- IBM PC, DOS - 1981
- Apple Macintosh - 1984
- Windows 1 - 1985; 3.0 - 1991
- Linux 1.0 – 1994
- Java 1.0 – 1995
- Moore's "lov"
 - Prosessor-”ytelsen” dobles (ca) hvert annet år
 - Antall transistorer pr cm² dobles ca hvert 2. år



Prosessor - CPU

- 1971 - Intel 4004
- 1973 - Intel 8008, 8080 (Altair)
- 1981 - IBM PC, 8086, 8088, 4 MHz
 - 8086: 16 bit databuss, 20 bit adressebuss = 1 MB
 - 8088: 8 bit databuss
- 1982 - 80286, 8 MHz
 - 24 bit adressebuss = 16 MB
- 1985/86 – 80386, 12-40 MHz
 - 1-4 GB virtuelt minne
 - 0,8 µm
 - Multitasking; Real Mode, Protected Mode

Prosessor - CPU

- 1985 - 80386, 12,5 - 30 MHz
 - 32 bit databuss og adressebuss(4 GB)
 - Virtual Real Mode
 - SX (1988); 16 bit ekstern databuss, 24 bit adressebuss
- 1989 - 80486, 25 - 50 MHz(DX2, DX4)
 - Innebygget flyttall ko-prosessor
 - 8 KiB cache
- **CISC** og **RISC**

Prosessor - CPU

- 1993 - Pentium, 75 MHz →
 - RISC nærmer seg CISC
 - Dual Independent Bus, L1 og L2 cache, Dynamic Execution
 - MMX (Multimedia = masseregning parallellt.
- 2001
 - Pentium 4 passerer 2 GHz
 - Intel *Itanium* (Merced), 64 bit prosessor → McKinley
 - AMD tar det private 64 bits markedet
- Ikke bare **Intel**
 - **AMD**, VIA (Cyrix), +++
 - Apple (Macintosh), Alpha, SPARC, +++
- Pr 2013 er Skjermkort i gang med å overta for supercomputere!

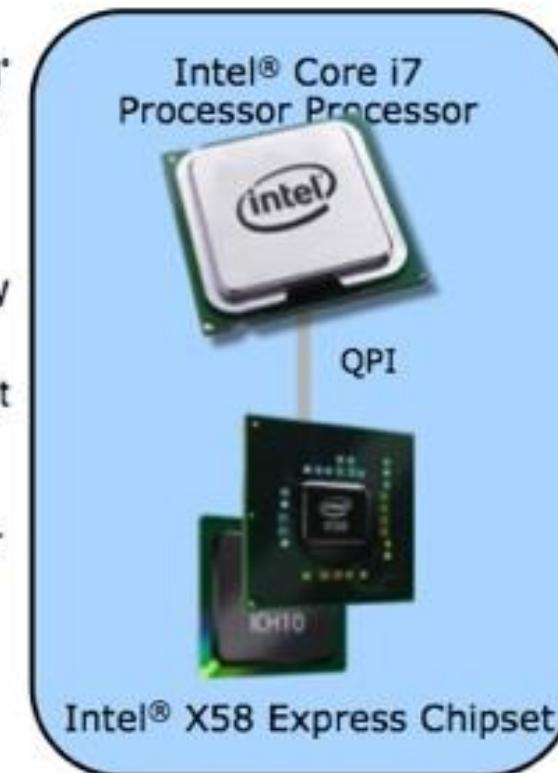
Intel Core i7

Intel® Core™ i7 Processor

- **Parallelisering**
 - 4 kjerner
- **Multimedia**
- 710 millioner transistorer
- ~ 3GHz
- 45 nm
- L1 & L2 pr kjerner, L3 felles

Performance/Features:

- 8 processing threads via Intel® Hyper-Threading Technology (Intel® HT)
- 4 cores
- Intel® Turbo Boost Technology operation
- Intel® QuickPath Interconnect (Intel® QPI) to Intel® X58 Express Chipset
- Integrated Memory Controller (IMC) – 3ch DDR3
- 7 more SSE4 instructions
- Overspeed Protection Removed



Socket:

- New LGA1366 Socket
- VRD 11.1

Platform Compatibility:

- Intel® X58 Express Chipset
- ICH10 / ICH10R

Targeted Segment:

- Extreme and performance demanding users
- Ultimate gaming, multimedia creation, compute intensive applications

DISKUTER

- Hva er problemet med denne («hardware-måten») fremstillingen av historien?

Denne forelesningsøkten vil bli tatt opp og lagt ut i emnet i etterkant.

Hvis du ikke vil være med på opptaket:

 ^ Start Video	La være å delta med webkameraet ditt.
 ^ Unmute	La være å delta med mikrofonen din.
To: Marianne Sundby (Privately) Type message here...	Still spørsmål i Chat i stedet for som lyd. Hvis du ønsker kan spørsmålet også sendes privat til foreleser.



Høyskolen
Kristiania

TK1100

DIGITAL TEKNOLOGI

1. FORMULERING



Forrige uke

- Introduksjon
- Computer/Datamaskin
 - *en menneskelaget innretning som mottar data i en form, behandler disse og produserer ny (og nyttigere) informasjon bygget på de opprinnelige data*
 - TYPER: Analog vs Digital, Spesialisert vs Generell, Elektronisk vs Mekanisk
- Tallsystemer ~ informasjonskoding

Denne leksjonen

- Data-representasjon
- Tall-systemer: desimal, binær, hexadesimal (oktal)
 - Enkle beregninger
 - Presisjon og negative tall
- Flyttall (litt om)
- (Koding/dekoding (begynner, vi skal ha mer))
- ØVING i dette temaet er viktig, regn oppgaver for å forstå!

Poenget!

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

Binær koding

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 hjelpe-tegn (,:!...), 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)

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

Typer 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

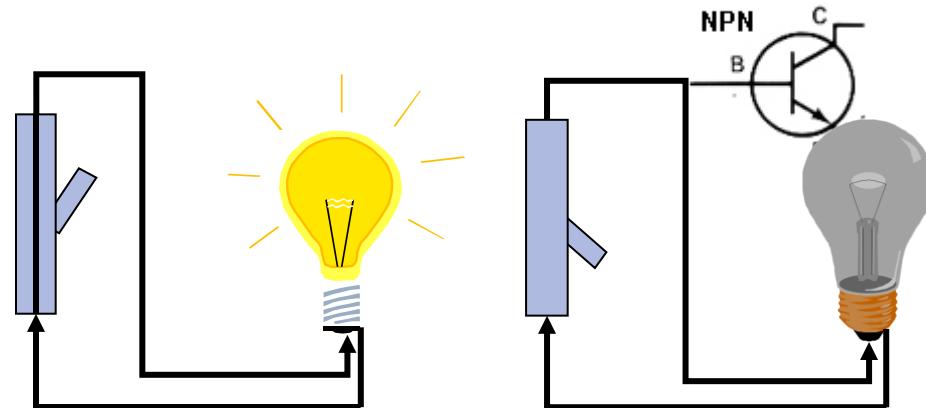
Tolking av binære koder (noen få)

	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	眺		爺	
JVM bytekode	istore_3	lstore_2	frem	lstore
X86 opkode	DS:	AND	JNO	AAA

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

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

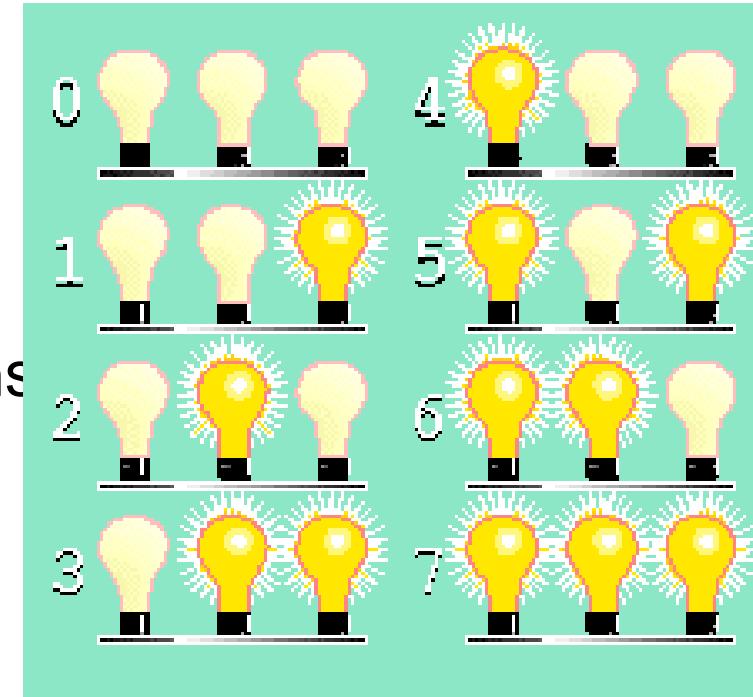


Binær tall-representasjon

- Med 3 lamper kan vi representere $2^3 = 8$ kombinasjoner av av/på
- Med Av=0 og På=1 får vi

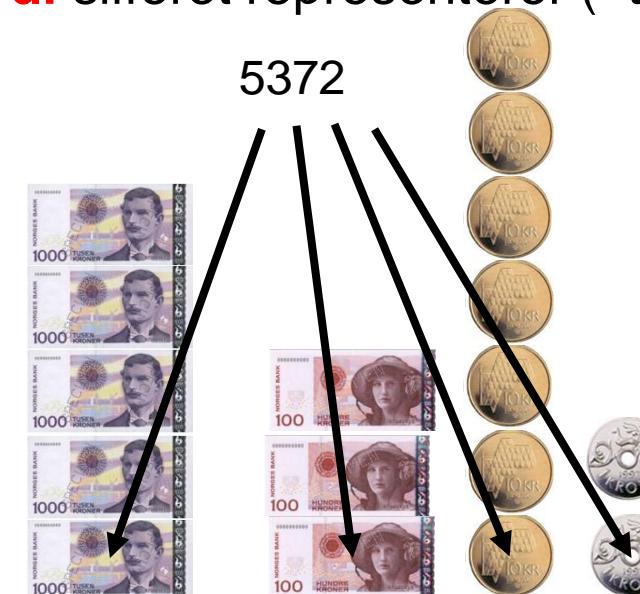
0 = 000	4 = 100
1 = 001	5 = 101
2 = 010	6 = 110
3 = 011	7 = 111

- Dette kan utvides til å bruke flere lamper (transistorer)



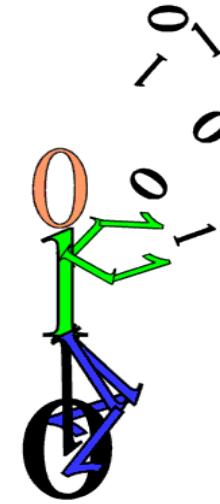
Desimale tall - posisjontall

- 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»)



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
 - $407_{23} = 4*23^2 + 0*23^1 + 7*23^0 = 2116 + 0 + 7 = 2123_{10}$
 - $= 6122_7$
- Det fine med posisjonelle tallsystemer er at fravær av en potens-verdi («vekt») kan representeres med **0**
 - 407 er ikke det samme som 47



Konvertering fra desimal til binær

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8$$

$$2^4 = 16$$

$$2^5 = 32$$

$$2^6 = 64$$

$$2^7 = 128$$

$$2^8 = 256$$

$$2^9 = 512$$

$$2^{10} = 1024$$

$$851 =$$

$$339 =$$

$$83 =$$

$$19 =$$

$$3 =$$

$$1 =$$

$$512 + 339 =$$

$$256 + 83 =$$

$$64 + 19 =$$

$$16 + 3 =$$

$$2 + 1 =$$

$$2 + 1 =$$

$$2^9 + 339$$

$$2^8 + 83$$

$$2^6 + 19$$

$$2^4 + 3$$

$$2^1 + 1$$

$$2^0$$

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

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

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

Konvertering fra binær til desimal

$$2^0 = 1 \quad 1101010011 = 1*2^9 + 1*2^8 + 0*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0$$

$$2^1 = 2$$

$$2^2 = 4$$

$$= 512 + 256 + 64 + 16 + 2 + 1$$

$$2^3 = 8$$

$$= 851$$

$$2^4 = 16$$

$$2^5 = 32$$

$$2^6 = 64$$

$$2^7 = 128$$

512 256 128 64 32 16 8 4 2 1

$$2^8 = 256$$

1 1 0 1 0 1 0 0 1 1

$$2^9 = 512$$

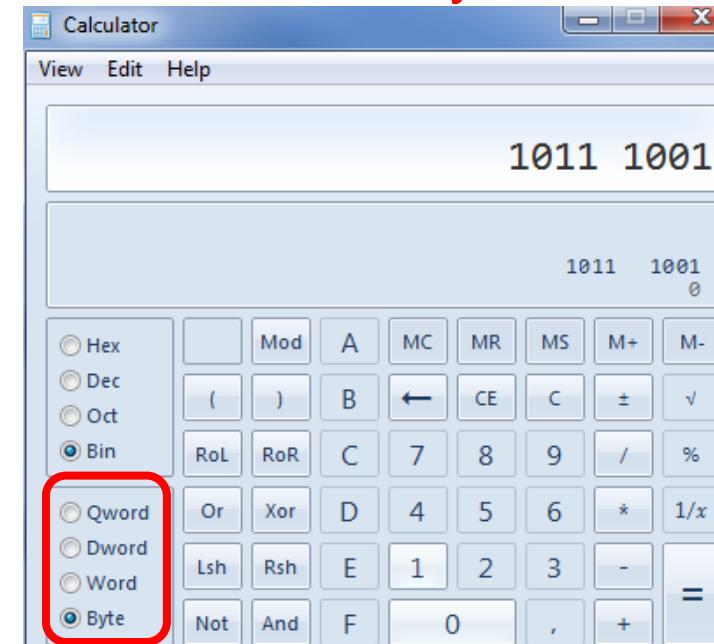
$$2^{10} = 1024$$

512 256 64 16 2 1

NB! Her mangler innledende nuller i 16 bit presisjon!

0000 0011 0101 0011

- I computere lagres alt enten i RAM («minne»), i **register** («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. opererer med enhetene:
 - **Byte**: 8 bit
 - **Word**: 16 bit
 - **Dword**: 32 bit
 - **Qword**: 64 bit



Regneoperasjoner - binærtall

Addisjon

$$\begin{array}{r} & 1 & 1 & 1 \\ 0000 & 1011 \\ + 0001 & 1010 \\ \hline 0010 & 0101 \end{array}$$

Multiplikasjon

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

NB!

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

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

1110 1101 2's komplement = 1's komplement + 1



- 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.

Subtraksjon

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

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

Overflow (spillsiffer),
Det sløyfer vi!!
Pga 8 bit presisjon

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**
 - $11_{16} = 1 \cdot 16 + 1 \cdot 1 = 17_{10}$,
- $0A4C_{16} = \begin{matrix} 0 \cdot 16^3 \\ 0 \cdot 4096 \end{matrix} + \begin{matrix} 10 \cdot 16^2 \\ + 10 \cdot 256 \end{matrix} + \begin{matrix} 4 \cdot 16^1 \\ 4 \cdot 16 \end{matrix} + \begin{matrix} 12 \cdot 16^0 \\ + 12 \cdot 1 \end{matrix} = 2636_{10}$
- Oversetter fra binær til hex ved å gruppere 4 og 4 binærtall sammen ("nibbler")
 - $1010\ 0011\ 1001\ 1111_2 = 0xA39F_{16}$
 - $1010_2 = A_{16}, 0011_2 = 3_{16}, 1001_2 = 9_{16}, 1111_2 = F_{16}$

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**
- Det er lett å regne feil med Hex
så vi går helst veien om binær!

ASCII og kode-tabeller

ASCII Tabellen (7 bit)

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

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) CarrigeReturn 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**

Oppsummering

Hva skal vi kunne

- 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 omfattende, se slide 38 for liten innføring



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

Neste leksjon?

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

0x3F

For valgfritt egenstudie

For de som ønsker å lære noen emner mer i dybden for å forstå det bedre er det her samlet noen ekstra temaer relatert til dagens undervisning, det må forventes en del egenarbeid for å forstå disse emnene.

Det vil ikke komme spørsmål på eksamen fra disse, og dette er altså ikke ansett for å være en del av pensum.

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 **nest 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
(IBM, Mac inntil Intel)
 - **Little Endian**: 0x41 00
(Dette var/er vanligst på Intel/AMD)
 - Misforståelse vil erstatte A med 犊

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

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

Flyttall

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

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

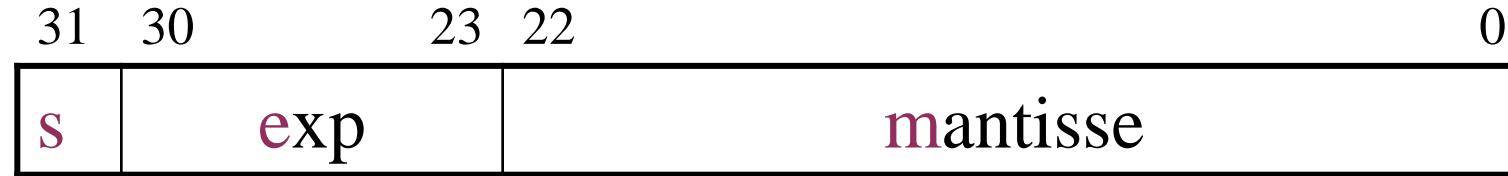
- Dette er helt tilsvarende at 103,57 er en notasjon for:
 $1 * 10^2 + 0 * 10^1 + 3 * 10^0 + 5 * 10^{-1} + 7 * 10^{-2}$
- I.e. Å konvertere til "kommatall" er helt tilsvarende hva vi gjør i desimaltallsystemet

Ex: Konvertere 0,6875 til bin

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

- Standarden IEEE 754 ble utviklet av W. Kahn i samvirkling 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.

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.

Ex: 5,8 i IEEE single precision?

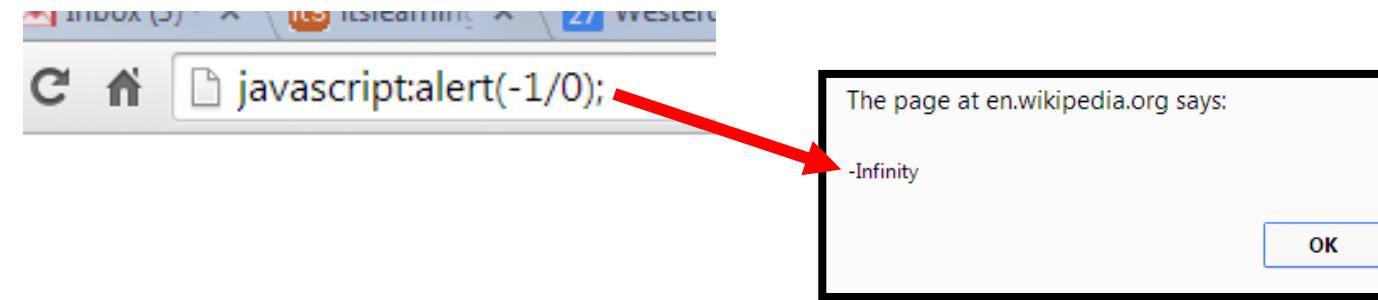
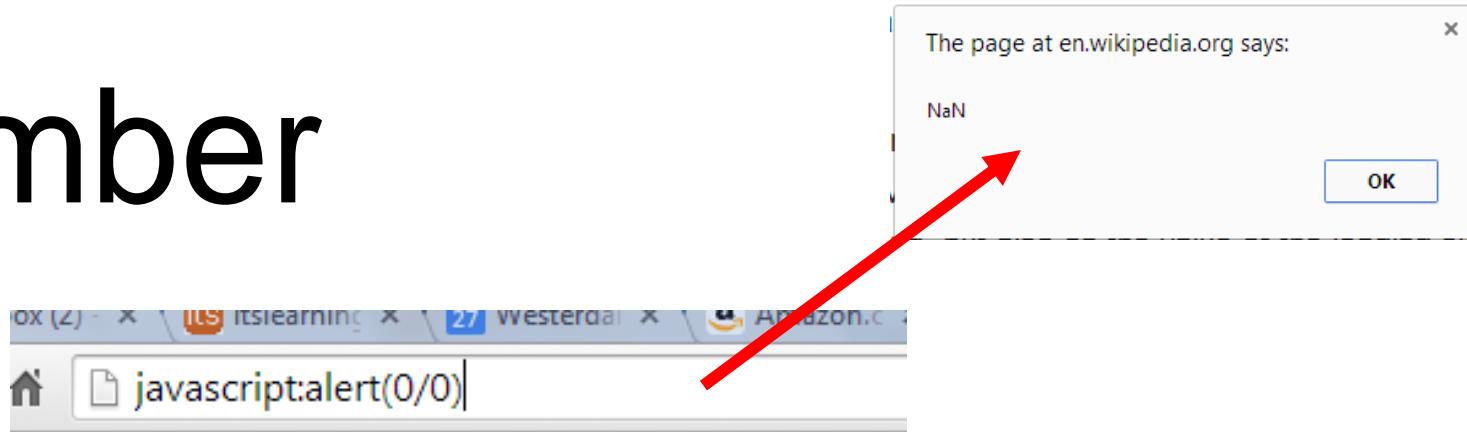
- $5,8 = 101,1100\ 1100\ 1100\ 1100\ 1100\ 1100\ \dots$
- Med 23 tall etter desimaltegnet, har vi
 $1.0111\ 0011\ 0011\ 0011\ 001\ *2^{10}$
- Tegn bit'en er positiv = 0.
- **Justert** exponent er $0x7F + 0x02 = 0x81$
- Vi får da:
0100 0000 1011 1001 1001 1001 1001 1001
(fortegn og mantisse er understreket)
eller 40B9999h
- 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

Ex: Hva er -0.5?

- $0.5_{10} = 0.1_2$
- Med 23 siffer etter desimaltegnet, har vi $1.0000000000000000000000000000000 * 2^{-1}$
- Fortegnsbit'en er negativ = 1.
- Exponenten er $7Fh + -1h = 7Eh$
- 1011 1111 0000 0000 0000 0000 0000 0000 (BF00000h)

Not a Number

- IEEE754 har en egen koder for resultater som ikke vanlige er tall (**f111 1111 1kxxxx ...**): 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.



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 $0,00011001100110011001100\dots$
 - Konverterer vi tilbake til desimaltall kan vi ende opp med $1/10=0,099999994$
 - 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.

Kodetabeller

- **BCD** – kode (Binary Coded Decimals)
 - Hvert siffer i desimaltallet får sin 4 bits binærkode
 - $529_{10} = 010100101001_{BCD}$
 - Har også andre varianter der vi bruker 8 bit pr desimalsiffer, f.eks.
 - $529_{10} = 0000101\ 00000010\ 00001001_{PBCD}$
 - 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
 - $M = 77_{ASCII}, \ : = 94_{EBCDIC}(5E_{16})$

Denne forelesningsøkten vil bli tatt opp og lagt ut i emnet i etterkant.

Hvis du ikke vil være med på opptaket:

 ^ Start Video	La være å delta med webkameraet ditt.
 ^ Unmute	La være å delta med mikrofonen din.
To: Marianne Sundby (Privately) Type message here...	Still spørsmål i Chat i stedet for som lyd. Hvis du ønsker kan spørsmålet også sendes privat til foreleser.



Høyskolen
Kristiania

TK1100

*Tegnsett
og
(litt) om andre media*

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

- Posisjontallsystemer
 - Basis, potens av basis
 - Konvertering
- (Primitive) **datatyper**
 - Heltall, (flyttall)
 - Binærtall og toerkomplement
 - Addisjon
 - Det CPUen bruker
 - Hexadesimale tall
 - Notasjon
 - Flyttall
 - Kodesystem, ikke «tallsystem»

I DAG

- Koding av bokstaver (**alfanumeriske** tegn)
- Kode-tabeller
 - **ASCII** (7 bit)
 - Mac Roman, **ISO 8859-1**, Windows 1252
 - **UNICODE**
 - UTF-16
 - UTF-8
- Andre media (litt)
 - Filformater
 - Bilde og lyd
- Komprimering

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)			F23	
UTF-16	眺		爷	
JVM bytekode	istore_3	lstore_2	frem	lstore
X86 opkode	DS:	AND	JNO	AAA

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

Hvorfor skal vi bry oss om tegnsett?

Dette er nøyaktig samme HTML/CSS-kode!
Det er bare hvilken **kodetabell** browseren
bruker for å velge **glyfer** som som er endret!

m.a.o. To **representasjoner** av samme **data**

(I eldre browsere var det lett å
endre dette manuelt ;-)

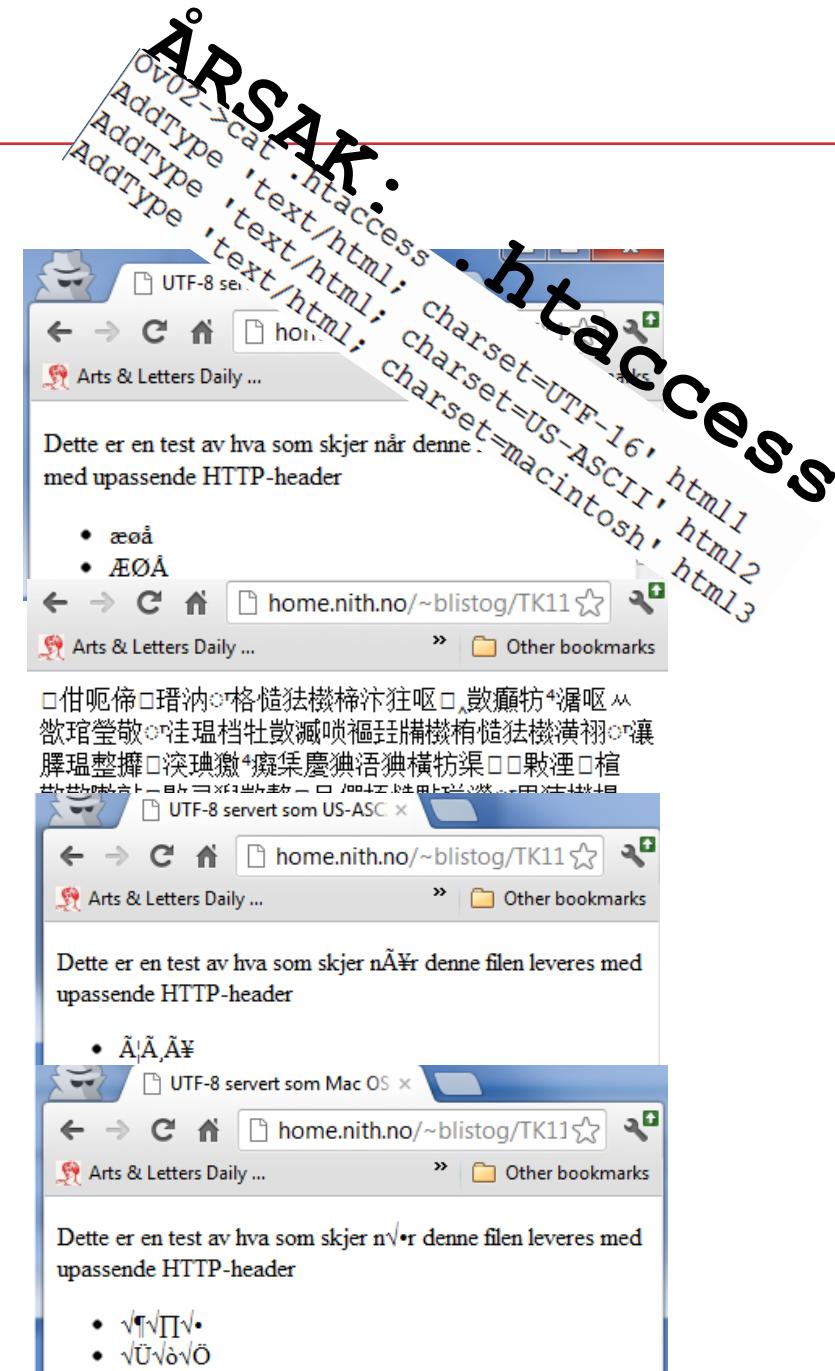
Encoding: UTF-8 → UTF-16LE



- Selve **HTML-koden** er en **ren text-fil**
 - Det kan gå (veldig) galt
 - **Browseren** bryr seg mest om hva **serveren** sier

```
GNU nano 2.2.4                                         File: test.html

<!DOCTYPE html>
<head>
<title>UTF-8 servert som UTF-8</title>
<meta charset="UTF-8">
</head>
<body>
<p>Dette er en test av hva som skjer når denne
filen leveres med upassende HTTP-header</p>
<ul>
<li>æøå</li>
<li>ÆØÅ</li>
</ul>
</body>
```



Moral (så langt)

- Text brukes til mye mer enn å bare lage/lagre bokstaver og utforme lesbare setninger
- Vi bruker det også til å kode andre former for adferd(program), data, struktur, utseende, m.m.
- Dersom man ikke passer på går det lett galt
 - Ulike systemer har ulike karakter-tabeller som default (standard)

ASCII Code Chart															
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1 DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2 !	”	#	\$	%	&	•	()	+	+	,	-	:	/	
3 0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4 @	A	B	C	D	E	F	G	H	I	J	K	L	M	N	0
5 P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	-
6 `	a	b	c	d	e	f	g	h	i	j	k	l	m	n	ø
7 P	q	r	s	t	u	v	w	x	y	z	{		}	-	DEL

ASCII

- 7-bit
- 32 kontroll-tegn
- 95/96 alfa-nummeriske tegn
- Inntil 2007 (?) mest brukt på web!

Glyf

- Det du ser på skjermen *er ikke* en ASCII-character
- Det er et **skjermbilde** som bestemmes av kodetabellen
- Betegnelse på et slikt skjermbilde er **glyf**
- Hvordan den *ser ut* bestemmes av **font-tabellen**



0100 0001



0	1	2	3
0 NUL	SOH	STX	ETX
1 DLE	DC1	DC2	DC3
2	!	"	#
3 0	1	2	3
4 @	A	B	C

ASCII-
tabell



Font-
Definisjonene
(Operativsystemet)

Å A-glyfen

ASCII (7 bit)

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

- **0x00-0x1F**: De 32 første kodene er **kontrolltegn**.
 - 0x07 er f.eks. beep i høytaler
 - Linjeskift er
 - i **Windows** CarrigeReturn (0x0D) **OG** LineFeed (0x0A),
 - i **Unix/Linux** og **OSX** bare LineFeed (0x0A)
 - Mellomrom-tegnet er **0x20**
 - Noensinne sett %20 e.l. i en URL?
- Tallene **kodes** med: **0x30-0x39**
 - Binærtallet **0000 0000** har ASCII-kode **0011 0000**
- "A" er **0x41** -> 0100 0001
"a" er **0x61** -> 0110 0001
 - En enkelt bit forskjell mellom store og små bokstaver
 - Numerisk sorteres dermed alle de store foran de små bokstavene



ASCII: Styrker og svakheter

- Hvert tegn er alltid 1 byte
- Aldri noe problem med Big/Little Endian
- Enkelt å flytte på = **interoperabelt**
- Støtter bare «amerikansk» alfabet!
- Alternative nasjonsspesifikke kodetabeller (og fonter..) ble dermed nødvendige.
- I HTML løste man dette med egne koder for «særnorske bokstaver»
 - Æ er Æ, mens æ er æ
 - Ø er Ø, mens ø er ø
 - Å er Å, mens å er å.

INTEROPERABELT

Windows 1252

ISO 8859-1

- «Europeiske tegn» manglet i ASCII
- Beholdt ASCII og utvidet til 8 bit
- Mange forskjellige løsninger på 1980-tallet (Code tables)
- Standarden for Vesteuropeiske skulle være ISO 8859-1 (Latin-1)

ISO 8859-1 (Latin-1) kodetabell

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0	NUL	DLE	space	0	@	P	`	p			no break space	°	À	Ð	à	ð
1	SOH	DC1	!	1	A	Q	a	q			i	±	Á	Ñ	á	ñ
2	STX	DC2	"	2	B	R	b	r			¢	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s			£	³	Ã	Ó	ã	ó
4	EOT	DC4	\$	4	D	T	d	t			¤	‘	Ä	Ö	ä	ö
5	ENQ	NAK	%	5	E	U	e	u			¥	µ	À	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v	un- defined		¡	¶	Æ	Ö	æ	ö
7	BEL	ETB	'	7	G	W	g	w			§	·	Ç	×	ç	÷
8	BS	CAN	(8	H	X	h	x			..	,	È	Ø	è	ø
9	HT	EM)	9	I	Y	i	y			©	¹	É	Ù	é	ù
A	LF	SUB	*	:	J	Z	j	z			¤	º	Ê	Ú	ê	ú
B	VT	ESC	+	;	K	[k	{			«	»	Ë	Ù	ë	û
C	FF	FS	,	<	L	\	l				¬	¼	Ì	Ü	ì	ü
D	CR	GS	-	=	M]	m	}			-	½	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~			®	¾	Î	Þ	î	þ
F	SI	US	/	?	O	_	o	DEL			-	¿	Ї	Þ	ї	ÿ

Windows 1252 kodetabell

Bygger på ISO 8859-1

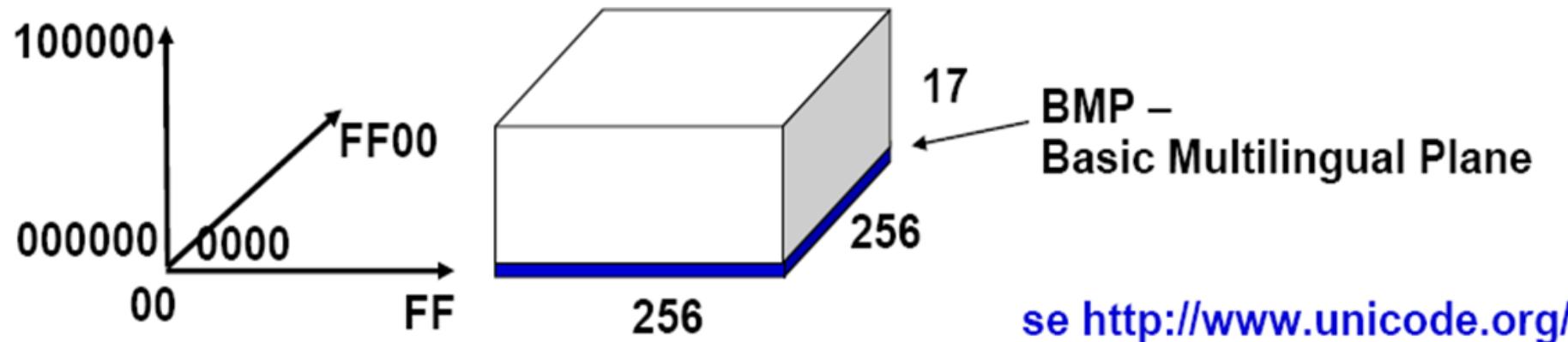
	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0	NUL	DLE	space	0	@	P	`	p	€	undef	no break space	°	À	Ð	à	ð
1	SOH	DC1	!	1	A	Q	a	q	undef	'	i	±	Á	Ñ	á	ñ
2	STX	DC2	"	2	B	R	b	r	,	'	ç	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s	f	"	£	³	Ã	Ó	ã	ó
4	EOT	DC4	\$	4	D	T	d	t	„	"	¤	‘	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u	...	•	¥	µ	Å	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v	†	-	!	¶	Æ	Ö	æ	ö
7	BEL	ETB	'	7	G	W	g	w	‡	—	§	·	Ç	x	ç	÷
8	BS	CAN	(8	H	X	h	x	^	~	..	,	È	Ø	è	ø
9	HT	EM)	9	I	Y	i	y	%oo	™	©	¹	É	Ù	é	ù
A	LF	SUB	*	:	J	Z	j	z	Š	š	a	º	Ê	Ú	ê	ú
B	VT	ESC	+	;	K	[k	{	<	>	«	»	Ë	Ù	ë	û
C	FF	FS	,	<	L	\	l		Œ	œ	¬	¼	Ì	Ü	ì	ü
D	CR	GS	-	=	M]	m	}	undef	undef	-	½	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~	Ž	ž	®	¾	Î	Þ	î	þ
F	SI	US	/	?	O	_	o	DEL	undef	ÿ	-	¿	Ï	ß	ï	ÿ

UNICODE

- Forsøk på å løse tegn-problemet en gang for alle
- Definerer bare kode-punkter, ikke hvordan de skal kodes binært
- 21 bit kodepunkt
- Faktisk binær kode bestemmes av Transformasjonsformatet
 - UTF-32, UTF-16, **UTF-8**, UCS-2, UTF-7 m.fl.
- Kodepunkter noteres typisk hexadesimalt, med U+ som prefix
 - U+0041 er kodepunktet for A

- Er ikke et tegnsett, men definerer koder for elementer som kan være med ("codespace").
- Sier ikke noe om HVORDAN disse tegnene skal kodes binært.
 - Flere muligheter: UTF-32 (hele), UTF-16 (deler), UTF-8 (hele+), Windows-1252 (litt), ISO-8859-1 (litt)
- Kodene U+0000 til U+007F er lik ASCII (7bit)
- Kodene U+0080 til U+00FF er lik ISO 8859-1
- Inkludere alle kjente tegn + Tengwar o.l.

UNICODE Struktur

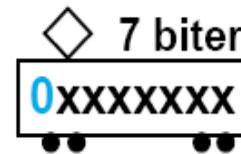


- Bruker (foreløpig) **21 bit** til kodepunkter
- Delt i 17 plan på 65536 kodepunkter i hvert
 - Plan 0: Basic Multilingual Plane (BMP)
 - Alle tegn vi støter på i «vanlige» språk
 - Plan 1: SMP
 - Historiske språk (f.eks hieroglyfer), musikk (noter), emotikoner ☺, spillkort mm
 - Plan 2: «Uvanlige» kinesiske tegn
 - Plan 14: Spesialtegn for taging av språk o.l.
 - Plan 15-16: Privat bruk
 - Firmalogoer o.l. kan registereres her.

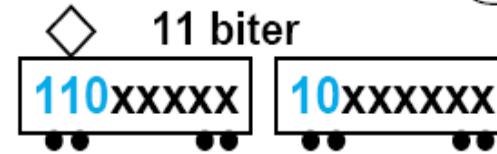
- Hvordan kodepunktene faktisk skal kodes er opp til bruker.
 - Mest brukt er UTF-8 og UCS-2 (subset av UTF-16)
-
- **UTF-16**
 - Blir for alle («våre vestlige») praktiske formål lik Unicode kodepunktet
 - UCS-2 er alltid to byte (brukes primært på Windows)
 - *UTF-16 kan* være mer enn to byte, er det stort sett ikke
 - Kodingsformatet for String i Java
 - **UTF-8**
 - U+0000 – U+007F er som i ASCII
 - Der etter benyttes to (eller flere) byte til å kode et tegn

Unicode UTF-8

- Enslig motorvogn = 0 + ASCII-kode
- Motorvogn i tog begynner alltid med et antall 1er-biter etterfulgt av en 0
- Antall 1er-biter i motorvognen = antall vogner i toget
- Vognene begynner alltid med 10
- Disse bitmønstrene brukes ikke for vanlige tegn i UTF-8



ASCII med en ekstra ledende 0 er kompatibel med UTF-8



Eksempel «Å» i UTF-8

- Tegnet «Å» har Unicode-punkt: U+00C5
- Binært: 0000 0000 1100 0101
- I UTF-8 må vi da legge dette inn i **to byte**:

Mest signifikante (**MSB**)
starter med 110

110x xxxx

1100 0011

«Padding»

Minst signifikante (**LSB**)
starter med 10

10xx xxxx

1000 0101

- UTF-8 koding av U+00C5 er **C385** ...

Fil formater

FILFORMATER

- En fil er (vanligvis) en **bitstrøm** som har blitt gitt et **navn**
- De aller fleste filformater består av
 1. meta-data og
 2. data
- Metadata beskriver hvordan filen skal behandles av programmet som skal bruke filen
- Metadata ligger vanligvis i en **HEADER**

Header

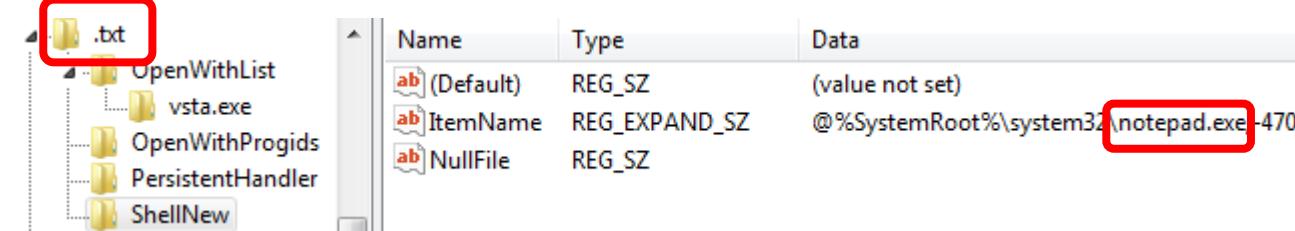
- beskriver dataene i filen

BODY

- Data vi skal bruke til noe

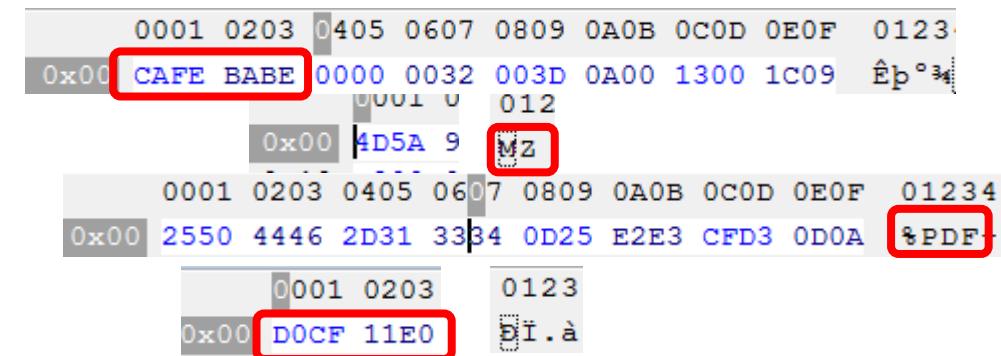
«Magic Number»

- Mange filformater starter med et «magisk tall» som fungerer som ID for filformatet
 - Windows bruker også fil-endelser som IDer og legger dem inn i Registry med info om hvilket program som åpner dem.



Name	Type	Data
ab (Default)	REG_SZ	(value not set)
ab ItemName	REG_EXPAND_SZ	@%SystemRoot%\system32\notepad.exe-470
ab NullFile	REG_SZ	

- OSX bruker et mer «distribuert» system
 - plister som kan endres med (bl.a.) `defaults` kommandoen
- Noen exemplar på magiske tall i filformat
 - Java **.class** filer:
 - Windows PE **.exe** filer:
 - Adobes PDF filer:
 - MS Office (gammelt):



0001 0203 0405 0607 0809 0A0B 0C0D 0E0F 0123	0x00 CAFE BABE 0000 0032 003D 0A00 1300 1C09 Ép °¾
0001 0203 0405 0607 0809 0A0B 0C0D 0E0F 012345	0x00 4D5A 9 MZ
0001 0203 0405 0607 0809 0A0B 0C0D 0E0F 012345	0x00 2550 4446 2D31 3334 0D25 E2E3 CFD3 0D0A %PDF-1
0001 0203 0405 0607 0809 0A0B 0C0D 0E0F 012345	0x00 D0CF 11E0

Binære vs text-filer

- **Text-filer** inneholder alltid binære koder som skal tolkes som text
 - «Alle» skjønner formatet
 - «selvforskjønende»
 - Enkle å overføre mellom ulike plattfomer
 - interoperabilitet
 - Kildekode, **XML**, HTML, CSS, SVG,...
 - Kan vanligvis gå ut fra at de består av byte.
- **Binære filer** inneholder binære koder som skal tolkes i henhold til et gitt format
 - Må vite filtype for å kunne tolke den korrekt
 - Kompilerte programmer, .class, .exe, .dll, .bmp, .pdf, .doc, ...
 - Kan ikke gå ut fra at de respektere noen faste **byte-grenser**.

Bilder og lyd

- Kan kun bli en smokebit i dag
- Bilder
 - Finnes flere hundre ulike formater, vi skiller mellom
 - bitmap/raster og vektor formater
 - Komprimering: tapsfri eller ikke?
- Lyd
 - Må samples for å digitaliseres
 - Nyquists samplingteorm
- Film
 - Container-format

Representasjon av bilder

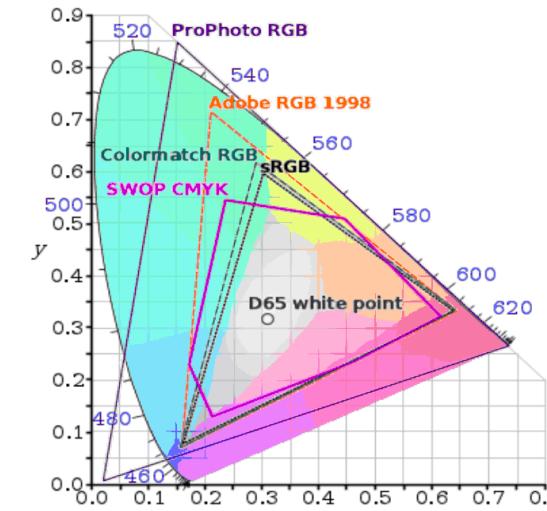
- Bildet deles opp i små elementer (**pixel**) som får en verdi

```
0101010101010101010101101001001000111110000
011010101010101010101001011010010110010100000110
10010101010101010101011010001010000101001010100
101101101011011010110101100110010110100010001001
0110100101101001011010100010011001001001011010010
100101101100101011010101110110011001010010101100
011010010011010110010010001001100110101010010001
010101101100101100100101110110011001010100100101
0101010101010011011010001001100010100001010100
10101010101011000100101100100011010011001110100001
0101010101010001000101000101101000010000001101
1101101010010100110100011010010011100101101000
101001010100100010100101100101101100001010000010
101011010001001001001011110101011010100101100
1010100001000100100101111010111100101001001001
010100101001000100101010111010101101001001000
1010010000100110011011111010111010101000100101
0100100101001000110110000111011101101010101000
0001000000010011001001111111110110111000000010
101000101010010011011000010101011101000010101000
000010000100101101010011111111111011101000101
00100010100110101010010001110111110100010010000
01001001011000100100100111101111010110100100101
1001001000011101001001001011111111011001001000
```

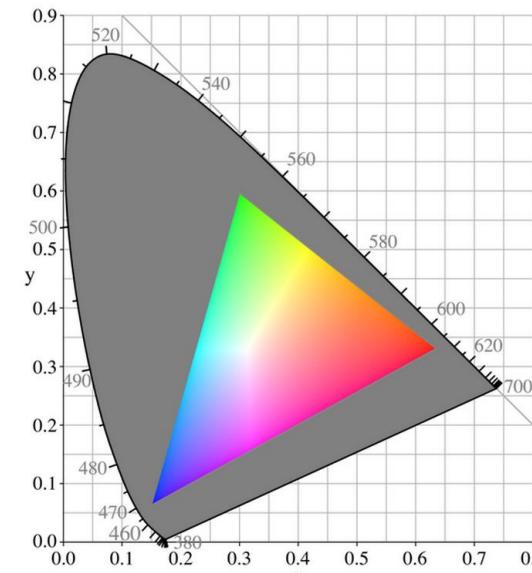


Farger: sRGB

- sRGB er standard farge-rom på Internett
 - Spesialtilpasset egenskapene til en CRT-monitor
 - En del nyanser lar seg ikke representere.
 - Typisk 1 byte hver for **R**, **G** og **B**; og gjerne en 1 byte for gjennomskinnlighet (**alfa-kanal**)

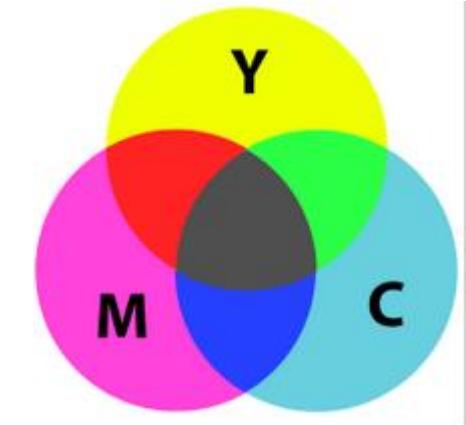


"CIE1931xy gamut comparison" by BenRG and cmglee
- http://commons.wikimedia.org/wiki/File:CIE1931xy_blank.svg.

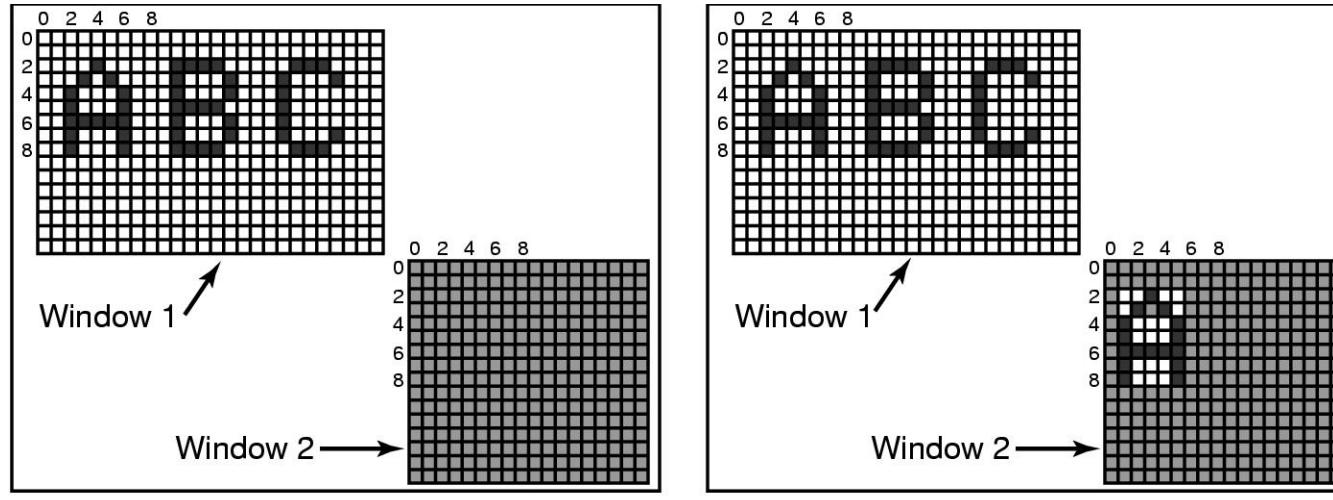


Utskrift: CMYK

- Farger fra skjerm er *additative*
 - Skjermen stråler ut den fargen vi vil se.
- Farger på utskrifter er *subtraktive*
 - «blekket» reflekterer den fargen vi vil se og absorberer de andre.
- Printerer o.l. benytter derfor ikke RGB, men ofte et CMYK-fargerom
 - K er sort, da blandingen ikke er intens nok...



Bitmap vs vektorgrafikk



(a)

Bitmap: koder og lagrer enkelt-pixlene slik de skal vises på skjermen

Vektor: Finner matematisk formel (ofte Bézier kurver) for å lage formen på billed-elementene du vil ha og lagrer bare formelen

20 pt: abcdefgh

53 pt: abcdefgh

81 pt: abcdefgh

Komprimering

- Ved hjelp av mønster-gjenkjenning kan datamengder som skal lagres gjøres mindre
- Eksempel:
 - En hest fra vest er best i test for de fleste (45 tegn)
 - En h❶ fra v❶ er b❶ i t❶ for de fl❶e ❶ = est (35 tegn)
- Det må benyttes egne programmer både for å komprimere data og pakke disse ut igjen
- Grafikk og multimedia kan ofte bli sterkt komprimert
- Populære formater er
 - gif, png, jpg, mpg,
 - zip, arj,

Typer komprimering

- **Tapsfri** versus med tap
- **Run-length** encoding
 - Lange, like sekvenser erstattes med en kode (f.eks. 583 0'er erstattes med 583*0)
- **Frekvensavhengig** koding (Huffman codes)
 - De vanligste kodene får kortest representasjon (jf Morse)
- **Relativ/differensiell** koding
 - Koder forskjell mellom påfølgende enheter istedetfor hver enkelt enhet (eksempel: stemme over GSM)
- **Ordbok**-koding
 - LZW (jf "ekstramateriale" foil som vedlegg, er adaptiv ordbok-koding: lager ordboken etterhvert som man koder meldingen)

Komprimere bilder

GIF

- Ordbokkoding, 256 farger pr pixel som lagres i ordbok, tap av fargedybde, en farge er "gjennomsiktig", egner seg for enkel animasjoner.

JPG

- Mange forskjellige komprimeringsteknikker, utnytter at øyet er mer følsomt for endring i lysstyrke enn farge, "baseline standard" komprimerer 10-30 ganger, egner seg for foto.

PNG

- Open source versjon av JPG, bruk = web

TIFF

- Komprimering minner om GIF, men brukes mye pga evne til å legge inn mye metadata, til arkivering

Komprimere lyd og fil: MPEG

- Film
 - Benytter diffrensiell koding, velger noen bilder (I-frames) som komprimeres i sin helhet (teknikk a la JPG), og *koder bare forskjell* for resten.
- Lyd (MP3 – MPEG Layer 3)
 - Fjerner detaljer som øret ikke hører, f.eks.
 - hører man i en periode *etter* en høy (intens) lyd ikke svakere lyder (tidsmaskering).
 - Nærliggende frekvenser med lavere intensitet blir "overdøvet" av de med høyere intensitet (frekvensmaskering)

Multimedia Container-formater

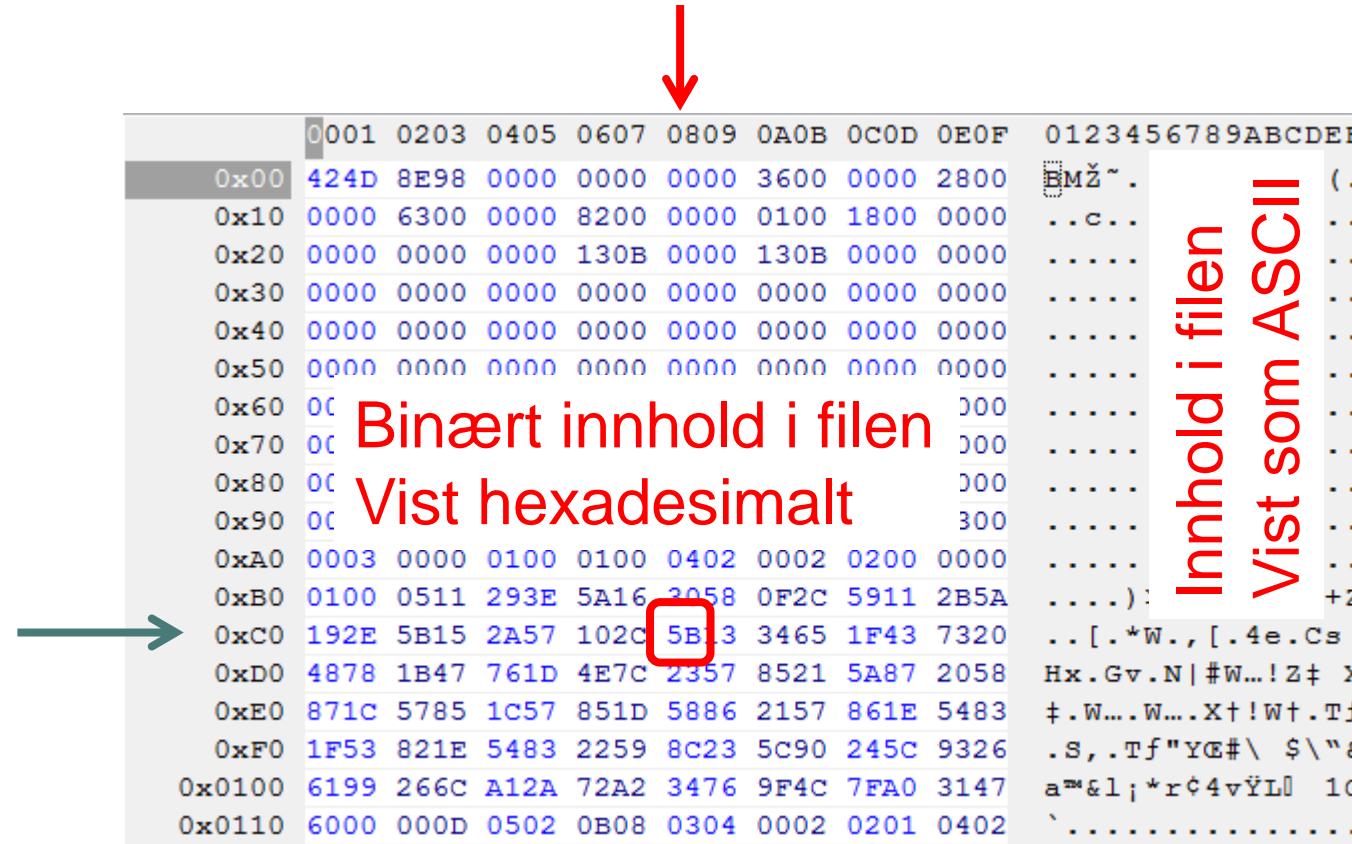
- De fleste **multimedia**-formater er egentlig **container/wrapper**-formater
 - De beskriver *hvordan* bilder og lyd som er pakket inn i formatet er kodet
 - Inneholder flere **separate data-strømmer**
- **Codec (coder-decoder)**
 - Programvaren som foretar den faktiske omkodingen til skjerm og høyttaler
- Eksempel **MP-4** (MPEG-4 Part 14)
 - Videreutvikling av Quicktime-formatet
 - Kan inneholde data-strømmer for
 - Metadata i MPX
 - Undertekster: flere formater
 - Film og lyd: Se f.eks. <http://www.mp4ra.org/codecs.html>
- At en fil faktisk lar seg avspille avhenger dermed at du har de korrekte codec'ene, ikke bare avspiller-programmet

Øvingstime

Hexeditor

- I øvingen til leksjon 0x02 skal vi bruke en HexEditor for å utforske oppbygningen av f.eks. et BMP-bilde..
 - En **Hexeditor** er programvare som viser deg og lar deg endre det binære innholdet i en fil
 - For å løse oppgaven **må** man samtidig følge med i et dokument som ligger i samme mappe som oppgaven og som beskriver den binære kodingen av bildet!
- **Windows** brukere kan f.ex. laste ned og bruke TinyHexer
 - http://texteditors.org/cgi-bin/wiki.pl?Tiny_Hexer
 - Det finnes også en HexEd-Plugin til Notepad++
- **OSX** brukere kan f.eks. bruk 0xED
- **Linux**-brukere kan f.ex. bruke DHEX

HexEditor (2)



	0001 0203 0405 0607 0809 0A0B 0C0D 0EOF	0123456789ABCDEF
0x00	4244D 8E98 0000 0000 0000 3600 0000 2800	BMŽ~.
0x10	0000 6300 0000 8200 0000 0100 1800 0000	...c...
0x20	0000 0000 0000 130B 0000 130B 0000 0000
0x30	0000 0000 0000 0000 0000 0000 0000 0000
0x40	0000 0000 0000 0000 0000 0000 0000 0000
0x50	0000 0000 0000 0000 0000 0000 0000 0000
0x60	0C 0000 0000 0000 0000 0000 0000 0000	000
0x70	0C 0000 0000 0000 0000 0000 0000 0000	000
0x80	0C 0000 0000 0000 0000 0000 0000 0000	000
0x90	0C 0000 0000 0000 0000 0000 0000 0000	300
0xA0	0003 0000 0100 0100 0402 0002 0200 0000
0xB0	0100 0511 293E 5A16 2058 0F2C 5911 2B5A): +Z
0xC0	192E 5B15 2A57 102C 5B13 3465 1F43 7320	...[.*W.,[.4e.Cs
0xD0	4878 1B47 761D 4E7C 2357 8521 5A87 2058	Hx.Gv.N #W...!Z‡ X
0xE0	871C 5785 1C57 851D 5886 2157 861E 5483	‡.W....W....X†!W†.Tf
0xF0	1F53 821E 5483 2259 8C23 5C90 245C 9326	.S,.Tf"Y€#\` \$\"&
0x0100	6199 266C A12A 72A2 3476 9F4C 7FA0 3147	a™&l;*r¢4vÝL 1G
0x0110	6000 000D 0502 0B08 0304 0002 0201 0402	`.....

På posisjon **0xC8** er det en byte med verdi **0x5B**
 $= 0101\ 1011 = 91_{10}$, som ASCII er det glyfen [

Instruksjoner til datamaskinen

- Instruksjoner til datamaskinen = programmering

NESTE UKE					
<u>HØY-NIVÅ</u>		ASSEMBLED	LOBBEG:	FERDIG:	DATA:
INPUT A		STORE A		0000 0000 0000 0000	0900
INPUT B		STORE B		0000 0001 0000 0000	0900
FOR I = A TO 0 STEP -1		LOAD I		0000 0001 0000 0000	0112
SUM = SUM + B		STORE I		0000 0001 0000 0000	0111
NEXT I		STORE A		0000 0001 0000 0000	040F
OUTPUT SUM		BRANCH FERDIG		0000 0010 0001 0001	0211
END		LOAD B		0000 0011 0000 1100	070C
		ADD SUM		0000 0011 0001 0010	0112
		STORE SUM		0000 0011 0001 0000	0310
		BRANCH LOBBEG		0000 0010 0001 0000	0210
	FERDIG:	LOAD SUM		0000 0101 0000 0100	0504
		OUT		0000 0001 0001 0000	0110
		STOP		0000 1010 0000 0000	0400
	I::	DM	0001	0000 0000 0000 0000	0000
	SUM:	DM	0000	0000 0000 0000 0001	0001
	A::	DM	0000	0000 0000 0000 0000	0000
	B::	DM	0000	0000 0000 0000 0000	0000
				0000 0000 0000 0000	0000

For valgfritt egenstudie

For de som ønsker å lære noen emner mer i dybden for å forstå det bedre er det her samlet noen ekstra temaer relatert til dagens undervisning, det må forventes en del egenarbeid for å forstå disse emnene.

Det vil ikke komme spørsmål på eksamen fra disse, og dette er altså ikke ansett for å være en del av pensum.

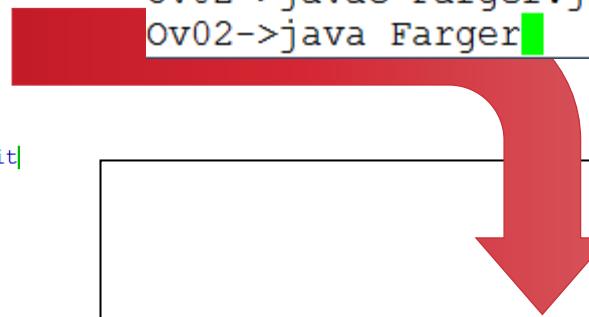
ANSI ESC-sekvenser og konsoll

- Windows konsollet (CMD.EXE) er det eneste som ikke støtter ANSI-escape sekvenser for å posisjonere og bruke farger.

```
public class Farger
{
    public static void main(String[] args)
    {
        //Sekvensene starter med ESC - som har ASCII-kode 27, som er 033
        //oktalt og 0x1B i hex; i Unix er oktalt vanlig brukt.
        //\033[2J Nuller skjermen
        System.out.print("\033[2J");
        //\033[13;40H plasser cursor ca midt i skjermbildet
        System.out.print("\033[13;40H");
        //\033[3x;4x sørger for text- og bakgrunnfarge
        //x-verdier 0-sort,1-rød,2-grønn,3-gul,4-blå,5-Magenta,6-Cyan,7-Hvit
        //25;H er koden for å flytte cursor til bånn av 25x80 konsollet
        System.out.println("\033[37;44m Hello World\033[25;H");
        //Fargekoden må så nullstilles igjen
        System.out.print("\033[0;0m");
    }
}
```

- I Linux og OSX er de derimot fremdeles støttet og brukt.

```
Ov02->javac Farger.java
Ov02->java Farger
```



```
Hello World!
```

Byte Order Marker

- Siden ett tegn kan kodes med mer enn en byte kan problem med *Endianess* oppstå
- Unicode-standarden innførte derfor **BOM**
 - Kode på starten av filen som viser hva slags rekkefølge byte legges inn i.
 - Mange moderne tekst-editorer legger automatisk inn BOM
- UTF-16 BOM: **U+FEFF**
 - Little **Endian** (UTF-16LE): 0xFF fulgt av 0xFE (þÿ)
 - Big **Endian** (UTF-16BE): 0xFE fulgt av 0xFF (ÿþ)
- UTF-8 BOM: **U+FEFF**
 - Blir da 0xEF 0xBB 0xBF (í»¿ som ISO-8859-1)
 - Leder lett til problemer, er ikke krevet av standarden og trengs egentlig ikke da innledende bits identifiserer plassering entydig.
 - Java støtter ikke BOM i UTF-8

Experiment i Word (Windows)

Setningen

Æ kodes som 00C6

blir når benytter Alt-X på hvert enkelt tegn i Word:

00C60020006B006F00640065007300200073006F006D00200030003000430036

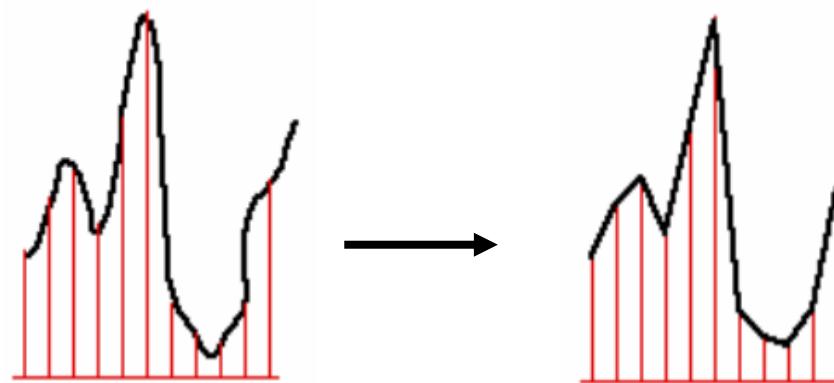
Windows programmering

- Alle Windows C APIer som omhandler tekst har 2 versjoner; A og W
- CreateFileA() dokumentert som ANSI
 - Støtter i utgangspunktet 1252 ASCII, men erfaring tilsier at å bruke annet enn 7-bit ASCII er å be om trøbbel
- CreateFileW() dokumentert som Unicode
 - Er egentlig UCS-2, vil alltid være 2 byte, lagres som W_CHAR i usermode (16 bit), og UNICODE_STRING i kernelmode

Representasjon av lyd



- Lyden blir digitalisert (samplet)
 - Hver liten lydbit blir representert av frekvens



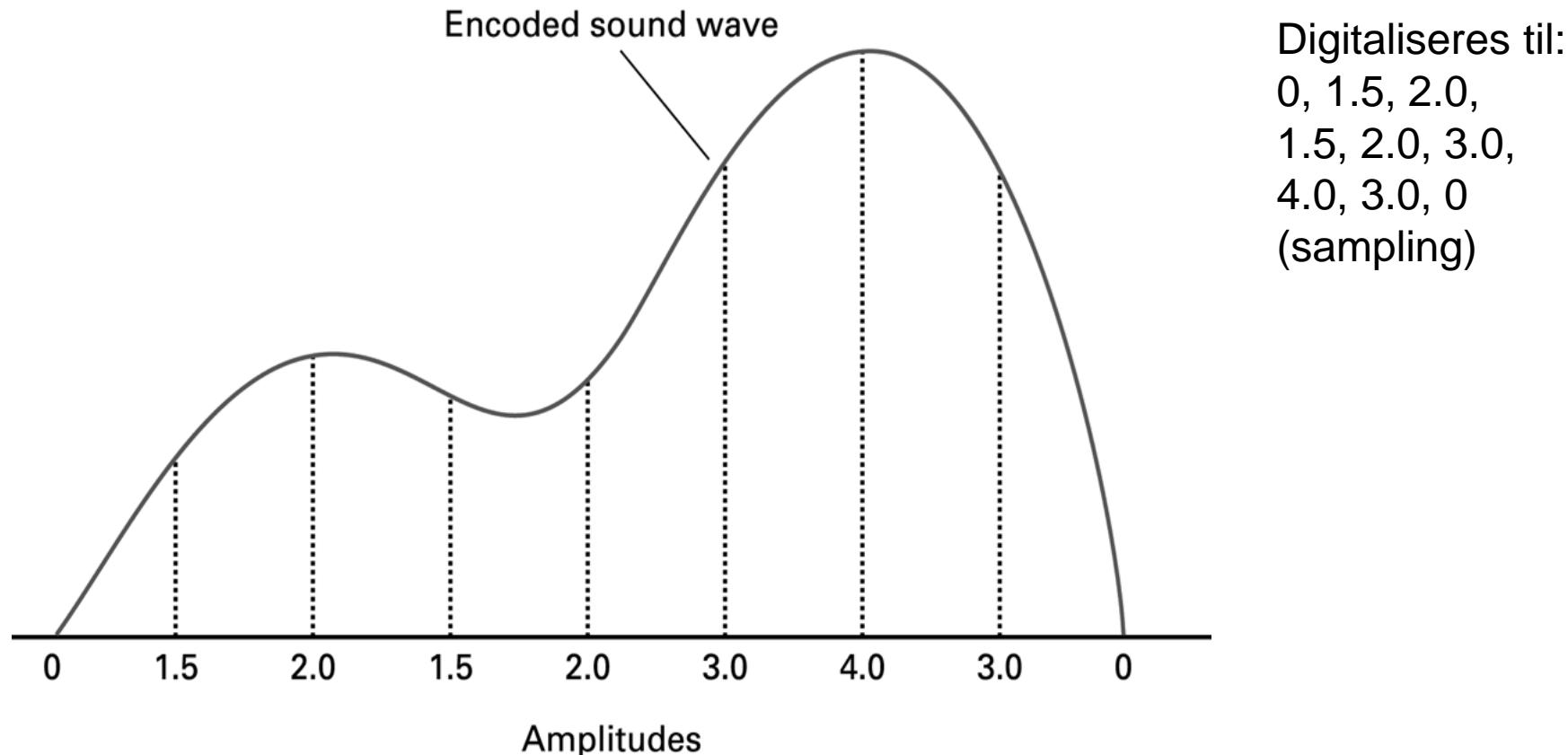
- Med lydkort i maskinen kan den lage multimedia presentasjoner
- Lyd kan lagres digitalt (f.eks. MP3)
- Tale kan digitaliseres og avspilles syntetisk
- Musikk kan lages med synthesizer (MIDI)



- Lydbølger detekteres av øret, omformes til nerveimpulser, og disse sendes til hørselsenteret i hjernen.
- Det ytre øret øker sensitiviteten med en faktor 2 – 3.
- Resonans i øregangen øker følsomheten for lyder ved 3 000 – 4 000 Hz. (viktig for oppfattelse av tale)
- Tromhinnen forsterker ca 15x
- I det indre øret sitter 16 000 – 20 000 hårceller som registererer lyd med forskjellige frekvenser.
- Vibrasjonene fra trykkbølger bøyer hårene, ionekanaler i bunnen av cellene åpnes, og en liten strømpuls sendes langs en nervefiber til hørselsenteret i hjernen.

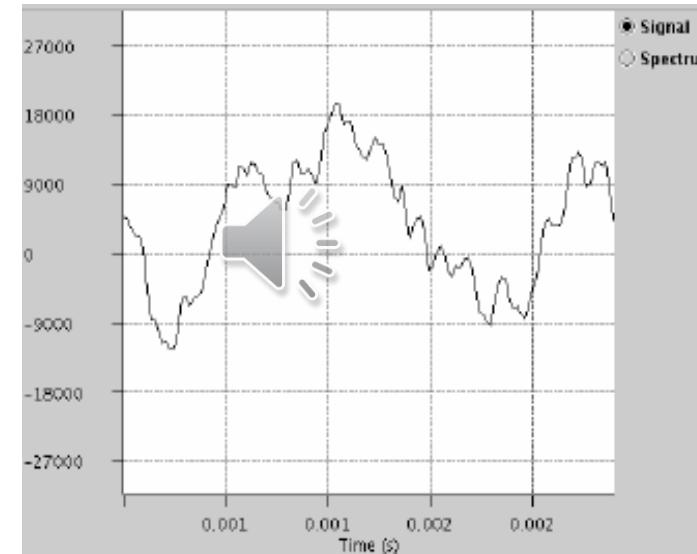
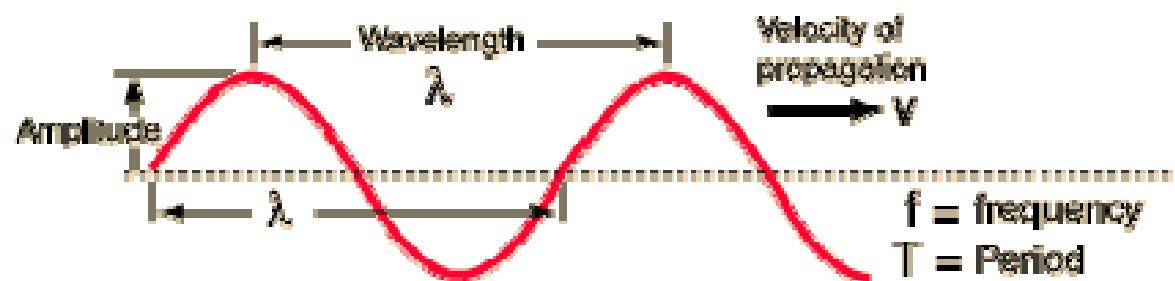


Lyd - digitalisering

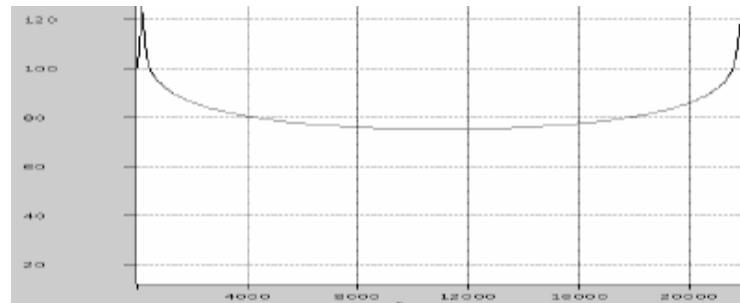
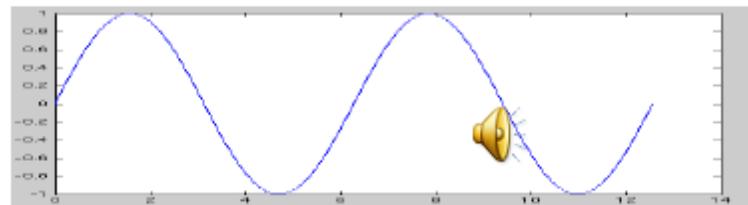


Lydbølger

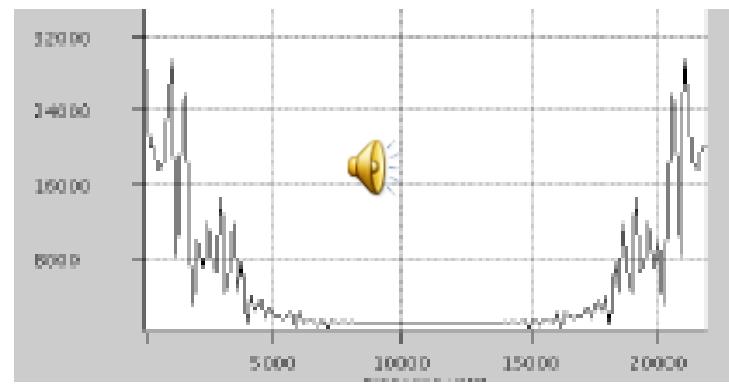
- **Nyquist-teoremet**
- Vi må **sample amplituden** med en **samplingfrekvens**, som for å få (perfekt) gjengivelse må være minst **dobel** så stor som en **største frekvenskomponenten** i lyden vi skal digitalisere



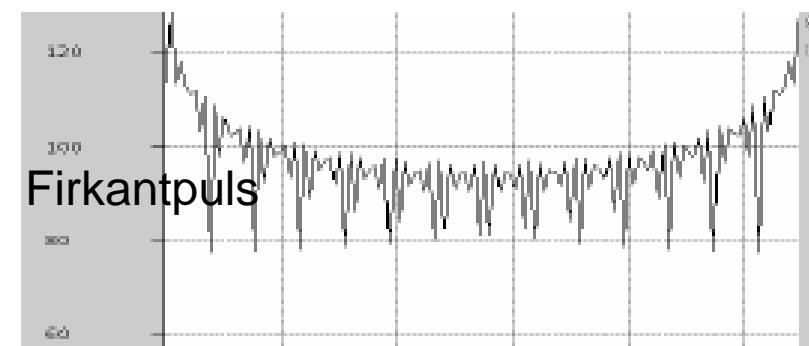
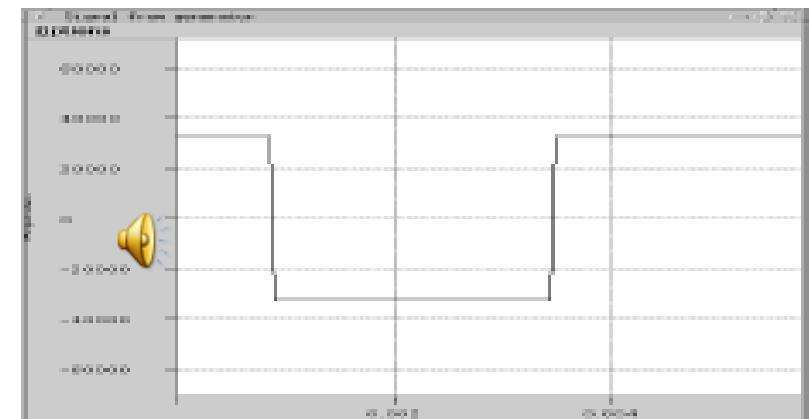
Frekvensspekter



Sinus

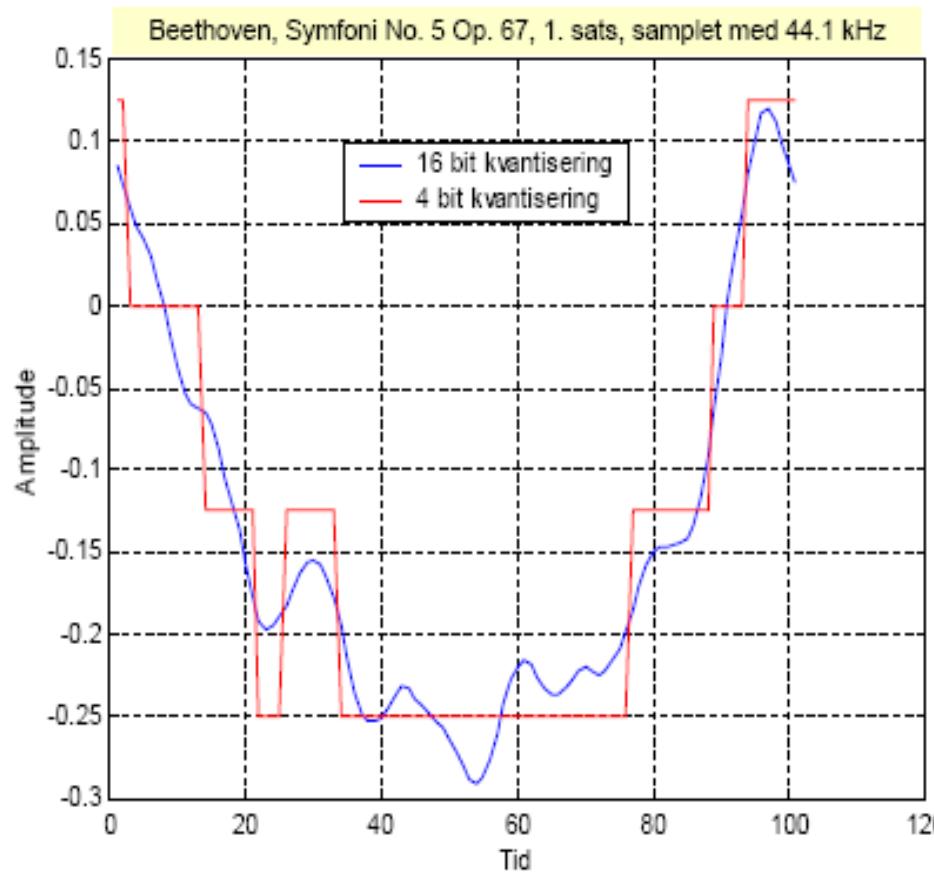


Messingblåser



Firkantpuls

Kvantisering eksempel



- Beethovens 5. kvantisert med hhv 16 (CD) og 4 bit (halvparten av telefon), samme samplingfrekvens
 - 16 bit->
 - 4 bit ->



- Lager ordboken mens vi koder (og dekoder).
Exempel:
 1. $xyx\ xyx\ xyx\ xyx \rightarrow \underline{\text{ordbok}}$: $x=0$, $y=1$, mellomrom = 2
 2. Koder: 0102, mellomrom viser at xyx er et ord \Rightarrow ordbok: $xyx = 3$
 3. 0102333 blir kodingen: komprimering $7/15 = 47\%$

NB! Ordboken *følger* av rekkefølgen, og trenger ikke følge med de komprimerte dataene!! Den kan lages på nytt med grunnlag i selve den omkodede meldingen

- Dekomprimering: 0102 \rightarrow xyx mellomrom \rightarrow $xyx = 3$, osv.

Denne forelesningsøkten vil bli tatt opp og lagt ut i emnet i etterkant.

Hvis du ikke vil være med på opptaket:

 ^ Start Video	La være å delta med webkameraet ditt.
 ^ Unmute	La være å delta med mikrofonen din.
To: Marianne Sundby (Privately) Type message here...	Still spørsmål i Chat i stedet for som lyd. Hvis du ønsker kan spørsmålet også sendes privat til foreleser.



Høyskolen
Kristiania

TK1100

Digital Teknologi

3'dje forelesning

Datamaskin- arkitektur



Læringsmål dekket så langt (1)

- definere begrepene **digital** og **analog**
- definere begrepene **system** og komponent og benytte disse i analyse
- definere datamaskin («computer»).
- definere og eksemplifisere begrepet **abstraksjon** i data teknisk forstand
- definere : Hz, bit, nibble, byte, word
- benytte prefixene: Ki, Mi, Gi, Ti både i desimal (k,M,G,T) og binær forstand

Læringsmål (2)

- forklare hvordan tall, tekst, lyd og bilder kan **representeres binært** og hvilke muligheter og begrensninger dette gir i forhold til analoge metoder.
- redegjøre for prinsippene bak et **posisjonstall**system og konvertere mellom **desimale**, **binære** og **hexadesimale** heltall
- redegjøre for konsekvensene av at i datamaskiner så operer man alltid med en begrenset, forhåndsgitt **presisjon** («ord-størrelse»)
- forstå prinsippet bak **toerkomplement** representasjon av heltall og konvertere mellom positive og negative heltall med en forhåndsgitt presisjon
- beskrive virkemåten til **IEEE 754** standarden for koding av **flyttall**, og når denne er ønskelig/nødvendig å bruke
- beskrive oppbygningen av ulike **kode-tabeller** for alfanumeriske tegn: **US-ASCII**, **ISO 8859-1**, **UNICODE** (UTF-8 og UTF-16) og benytte disse til å feilsøke vanlige tegnsett-problemer
- kunne forklare prinsippet bak Huffman-koding/komprimering og utføre enkel **ordbok-komprimering**
- forklare og vise forskjellen på grafikk-formater basert på raster- og vektor-grafikk.
- benytte en **hexeditor** til å inspirere og endre innholdet i filer i ulike formater

I dag: Datamaskinarkitektur

1. Boolsk algebra

- NOT, AND, OR, XOR
- Sekvensielle og kombinatoriske kretser.

2. Instruksjonsettarkitektur (ISA)

- Hvilke instruksjoner?
- Størrelse på ord (antall bit)
- Addresseringsmoduser
- Register
- Dataformater
- Hva CPU er egnet til å gjøre.

3. Organisering (mikroarkitektur på CPU)

- Veiene data flyttes (bussarkitektur)
- Hvordan data lagres, f.eks. Cache
- ...

4. Systemdesign (Neste gang!)

- Chipset (buss- og minne-kontrollere)
- Minneorganisering
- Virtuelt minne?
- DMA
- Hva og hvordan det er koplet sammen

Boolsk algebra

	x	y	
\wedge	0	1	
x	0	0	0
	1	0	1

	x	y	
\vee	0	1	
x	0	0	1
	1	1	1

	x	y	
\rightarrow	0	1	1
x	0	1	0
	1	0	1

	x	y	
\oplus	0	1	
x	0	0	1
	1	1	0

Figure 1. Truth tables



Figure 2. Logic gates



Figure 3. De Morgan equivalents

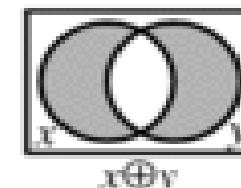
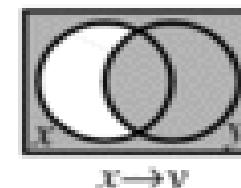
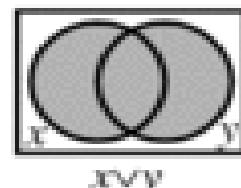
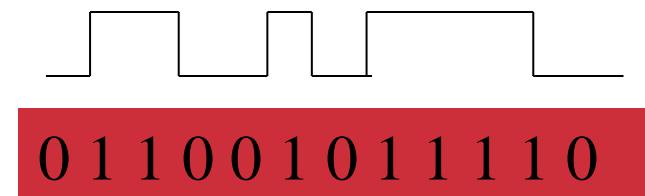


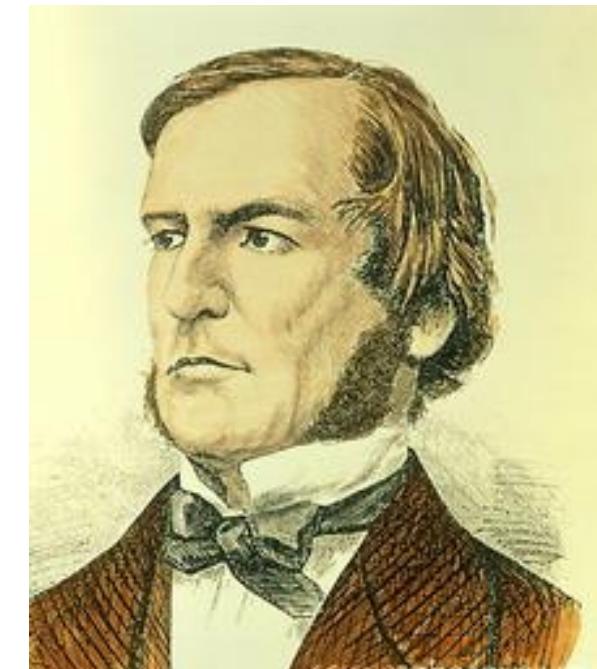
Figure 4. Venn diagrams

Bit og George Boole

- All logikk i **digital** elektronikk er basert på **bit**
- Bruker transistorer som byggeklosser
- Kan være 0 eller 1; høy eller lav spenning
- Logikken som brukes heter **Boolisk Algebra**
 - **George Boole** (1815-1864)

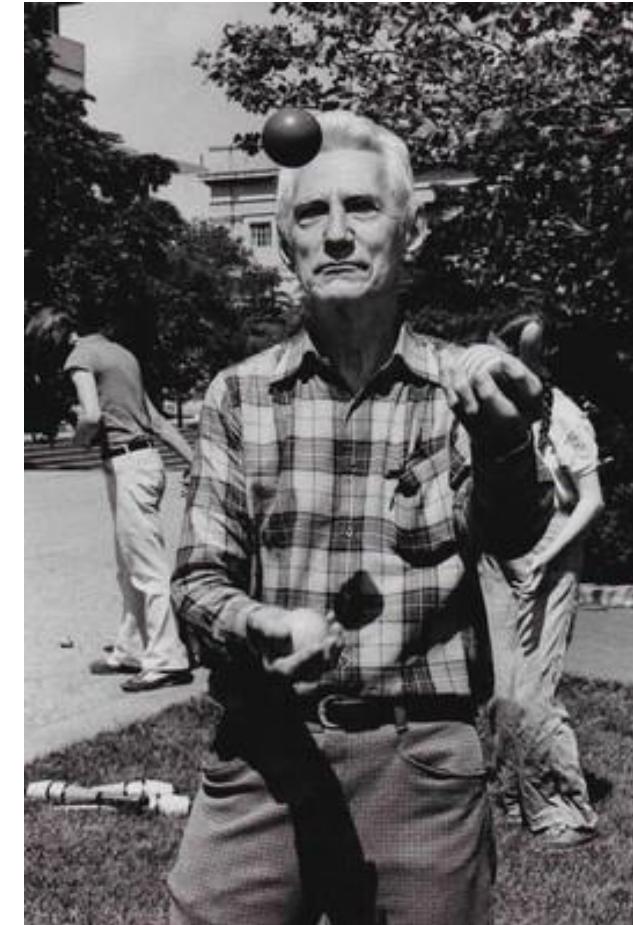


NB! Disse 12 bit'ene er normalt
kun et utsnitt av et 32/64 bit ord

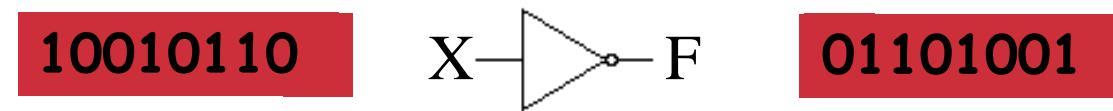


Boolsk algebra

- Består av 3 (4) grunnoperasjoner (porter, gates)
 - IKKE (**NOT**)
 - OG (**AND**)
 - ELLER (**OR**)
 - (EKSKLUSIV ELLER (**XOR**))
- ”Gjenoppdaget” i 1939 av Claude Elwood **Shannon**
- Data-elektronikk foretrekker oftest «negativ» logikk
 - NAND, NOR, XNOR



NOT

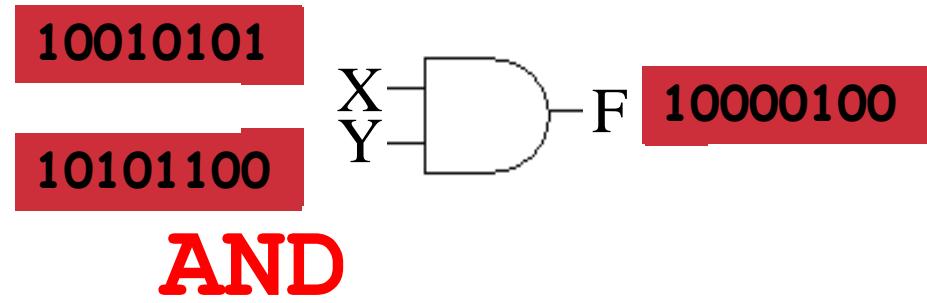


NOT

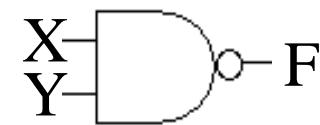
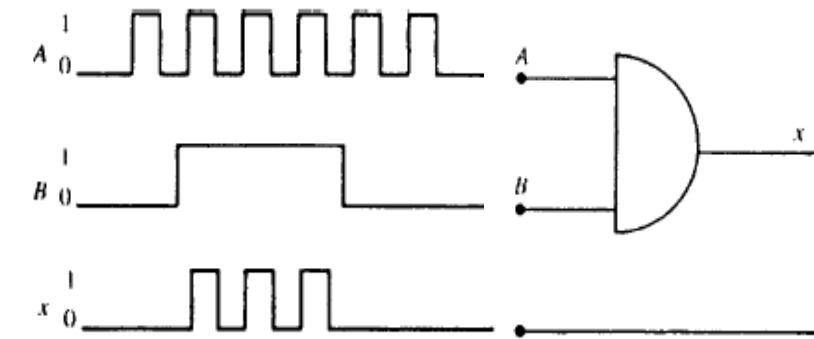
Sannhetstabell

X	F
0	1
1	0

AND og NAND



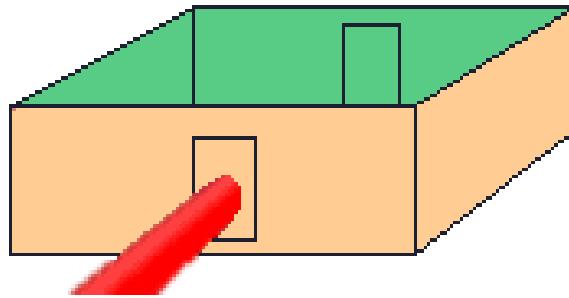
X	Y	F
0	0	0
0	1	0
1	0	0
1	1	1



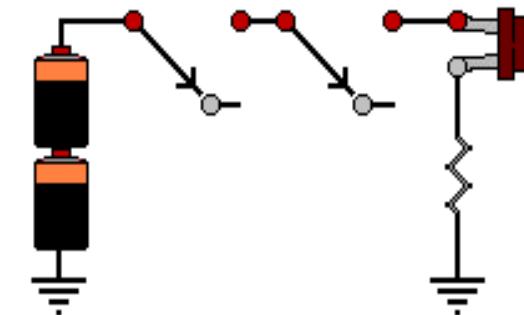
NAND

X	Y	F
0	0	1
0	1	1
1	0	1
1	1	0

Eksempel: AND



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



OR og NOR

10111001
10001110



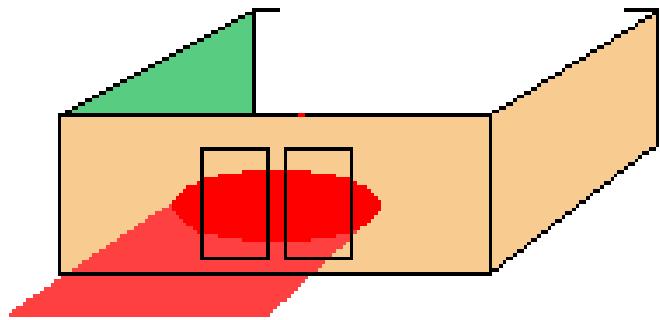
OR

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

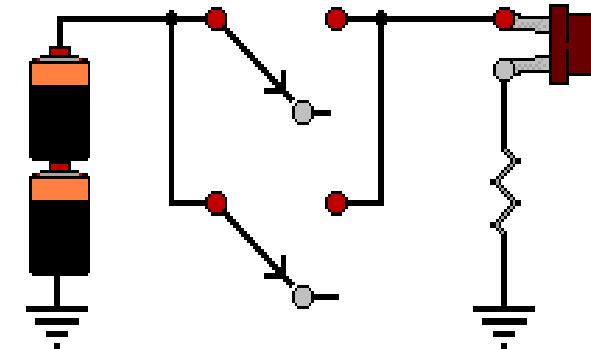
NOR

X	Y	F
0	0	1
0	1	0
1	0	0
1	1	0

Eksempel: OR



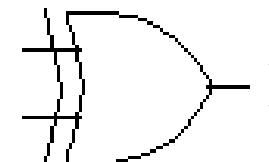
<i>A</i>	<i>B</i>	<i>C</i>
0	0	0
0	1	1
1	0	1
1	1	1



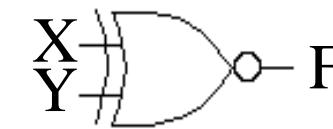
XOR og XNOR

10110010
11001110

X
Y



F
01111100



XOR

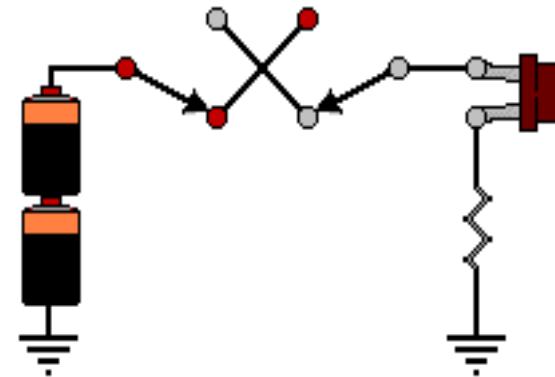
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

X	Y	F
0	0	1
0	1	0
1	0	0
1	1	1

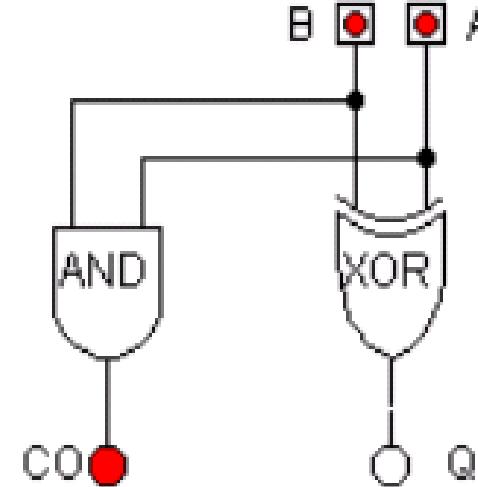
Eksempel: XOR

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



Praktisk eksempel: Half adder

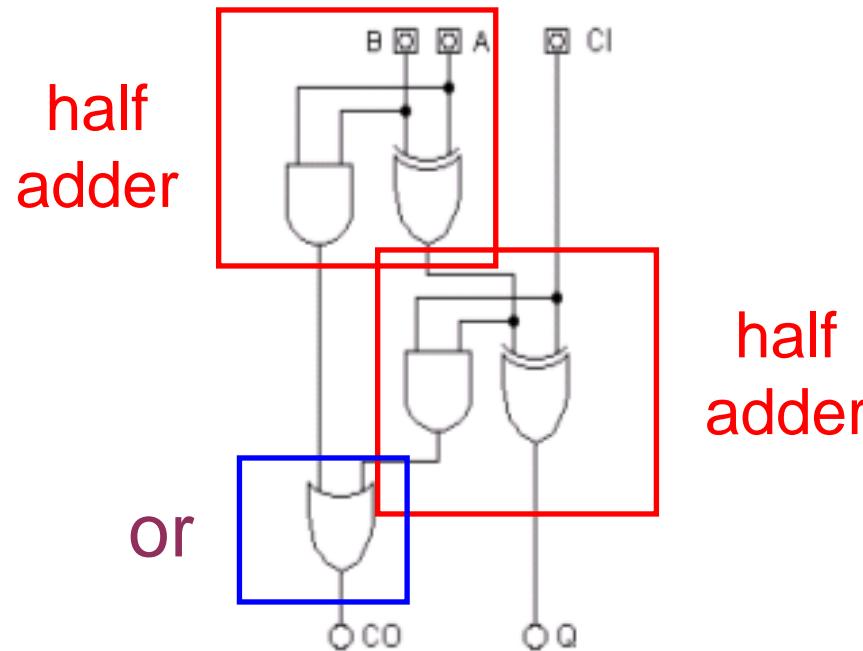
- Adderer to bit, A og B
- Får svaret i **Q** og menten i **CO**
 - $Q = A \text{ XOR } B$
 - $CO = A \text{ AND } B$



A	B	CO	Q
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

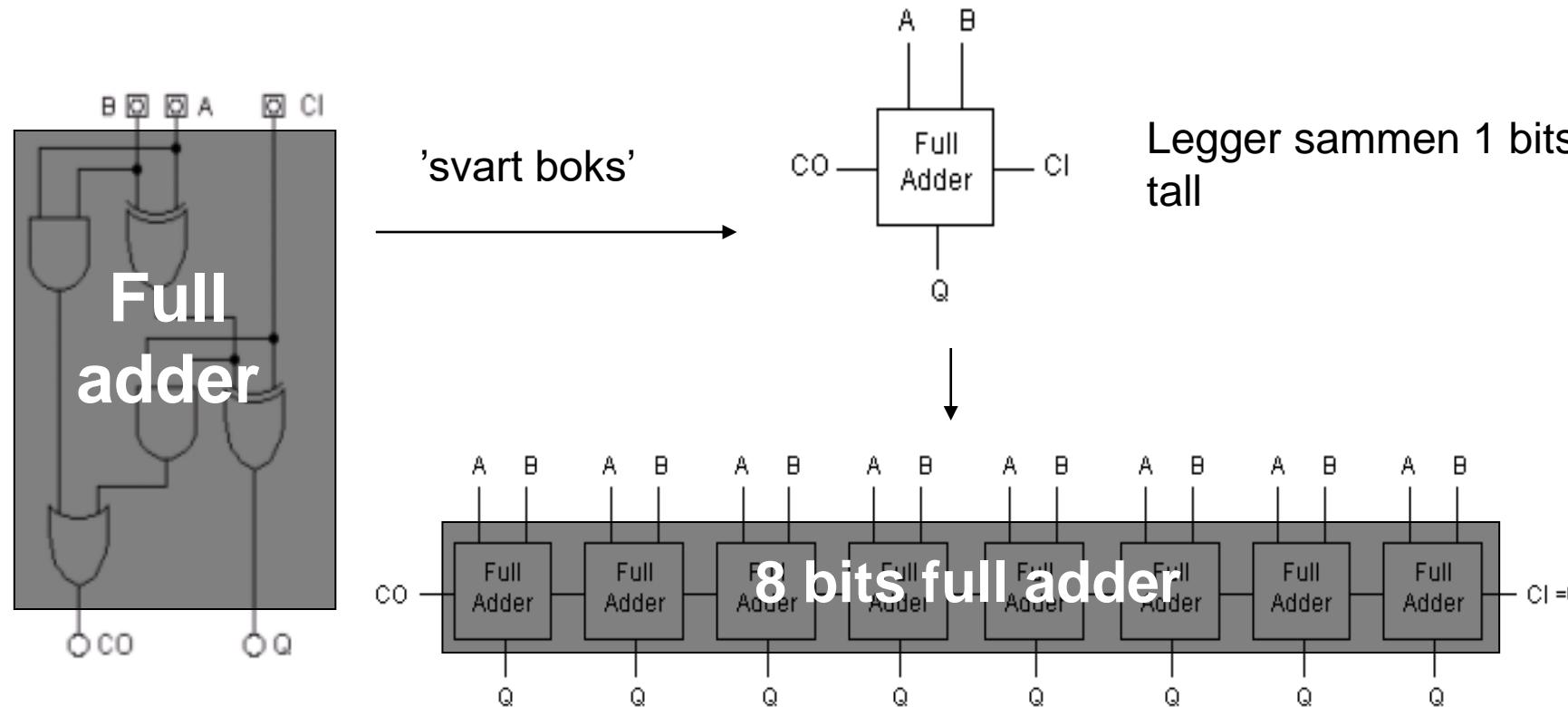
Praktisk eksempel: Full adder

- Legger sammen to bit, **A** og **B**, og eventuell mente inn på **CI**
- Får svaret i **Q** med menten ut i **CO**



A	B	CI	Q	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Praktisk eksempel: 8 bits full adder



8 fulle addere koplet sammen:
Legger sammen 8 bits tall (kalles
en barreladder, og er litt «treig»...)

Datamaskin arkitektur

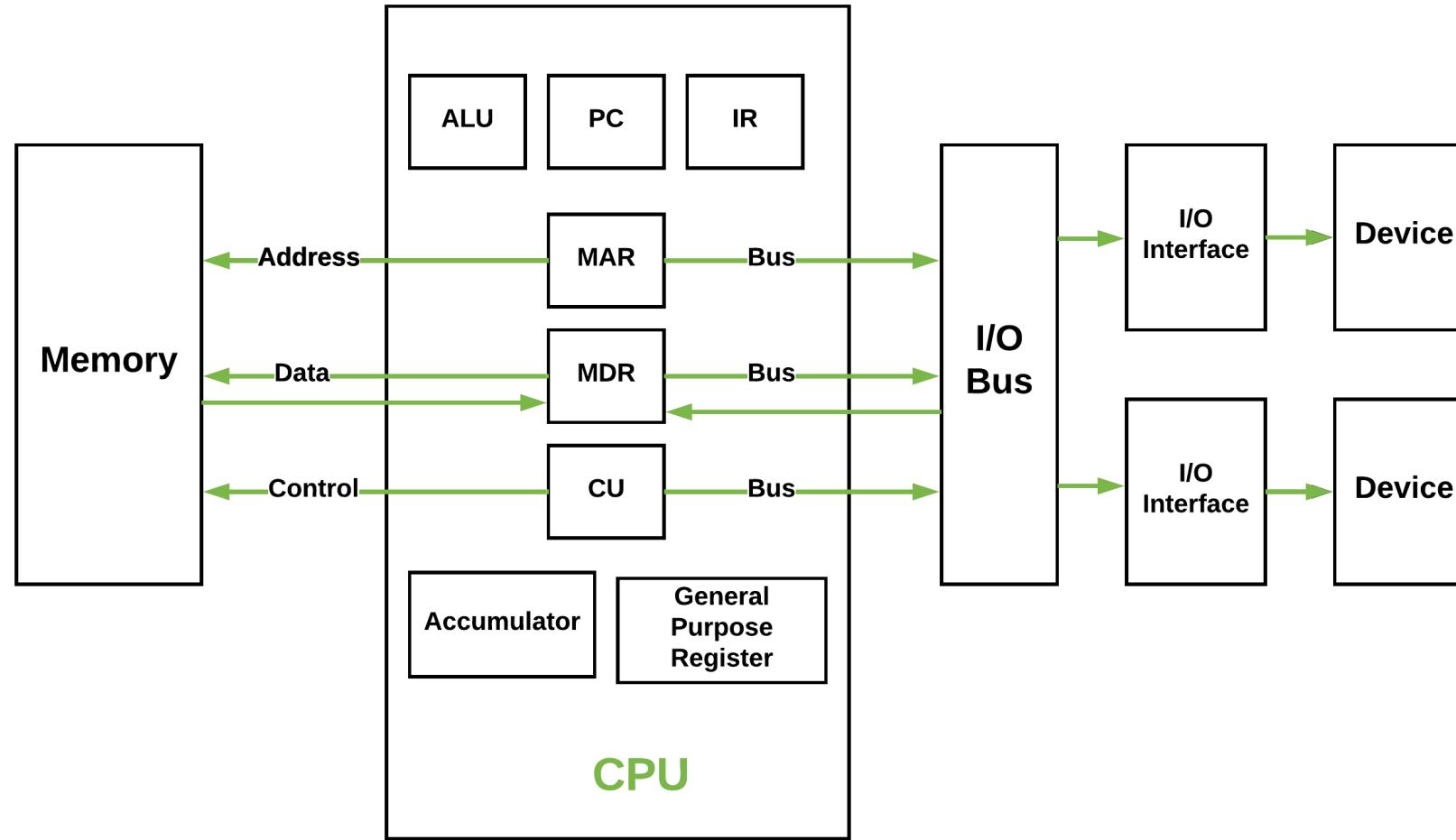
Enheter og størrelser

- **Bit (b)** - 0 eller 1
- **Byte (B)** = 8 bit
- $\text{Kilo} = 10^3 = 1000 \approx 1024 = 2^{10}$
 - 1 km = 1000 m, 1 mm = 1/1000 m
- For enkelhets skyld "jukser" vi litt !
 - k = 1000, Ki = 1024, (anta at K = 1024 også)
- Kilobyte/**KibiByte (KiB)** = 2^{10} byte = 1024 byte
- Megabyte/**MibiByte (MiB)** = 2^{20} byte = 1024 KB = 1048576 byte
- Gigabyte/**GibiByte (GiB)** = 2^{30} byte = 1024 MB = 1073741824 byte
- **Herz (Hz)** = hendelser pr. sekund
- **MIPS** = Mega instruksjoner pr sekund.
- **FLOPS** = Mega flyttalloperasjoner pr sekund
- **kbps** = 1000 bit / sekund (bitrate, «båndbredde»)

Multiples of bytes		v · d · e		
SI decimal prefixes		Binary	IEC binary prefixes	
Name (Symbol)	Value	usage	Name (Symbol)	Value
kilobyte (kB)	10^3	2^{10}	kibibyte (KiB)	2^{10}
megabyte (MB)	10^6	2^{20}	mebibyte (MiB)	2^{20}
gigabyte (GB)	10^9	2^{30}	gibibyte (GiB)	2^{30}
terabyte (TB)	10^{12}	2^{40}	tebibyte (TiB)	2^{40}
petabyte (PB)	10^{15}	2^{50}	pebibyte (PiB)	2^{50}
exabyte (EB)	10^{18}	2^{60}	exbibyte (EiB)	2^{60}
zettabyte (ZB)	10^{21}	2^{70}	zebibyte (ZiB)	2^{70}
yottabyte (YB)	10^{24}	2^{80}	yobibyte (YiB)	2^{80}

1 EiB tilsvarer
en 50.000 år
lang video
(DVD-kvalitet)!

Von Neumann modellen



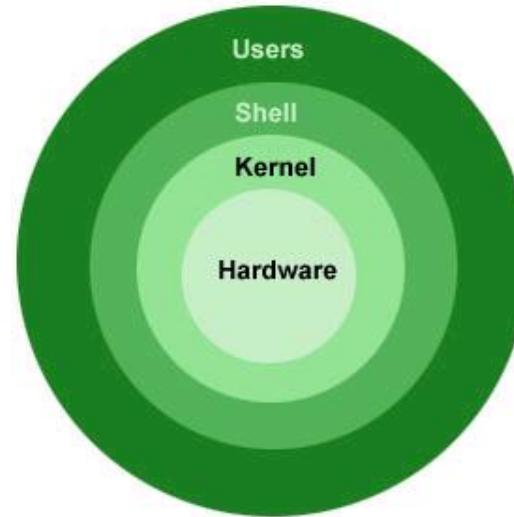
Software vs Hardware

Applikasjoner

- Shell + filbehandler
- Brukerprogrammer
- Kompilatorer

Hardware

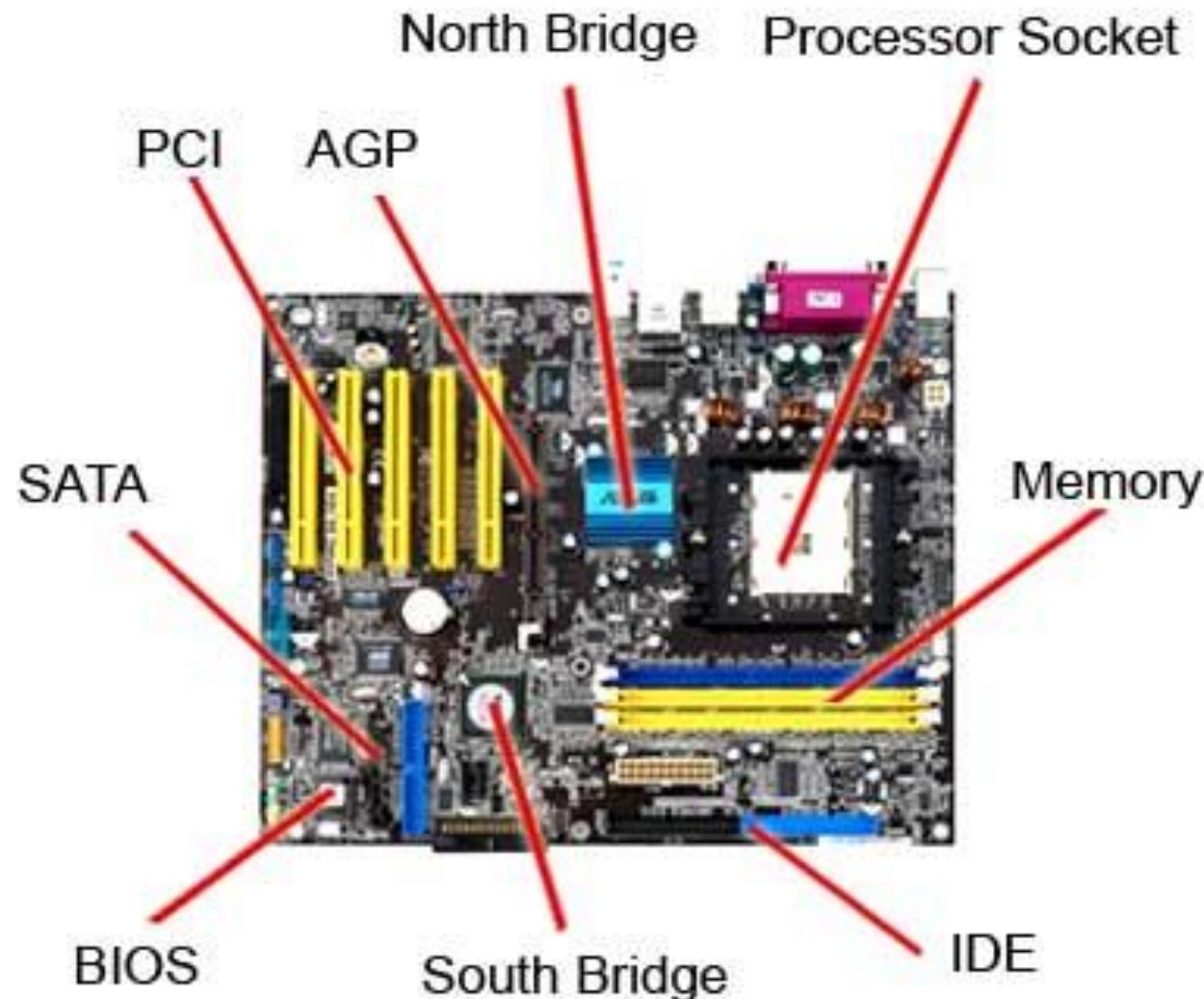
- CPU, RAM, I/O-kontrollere
- Periferibusser
- Tilleggskort
- Harddisk...



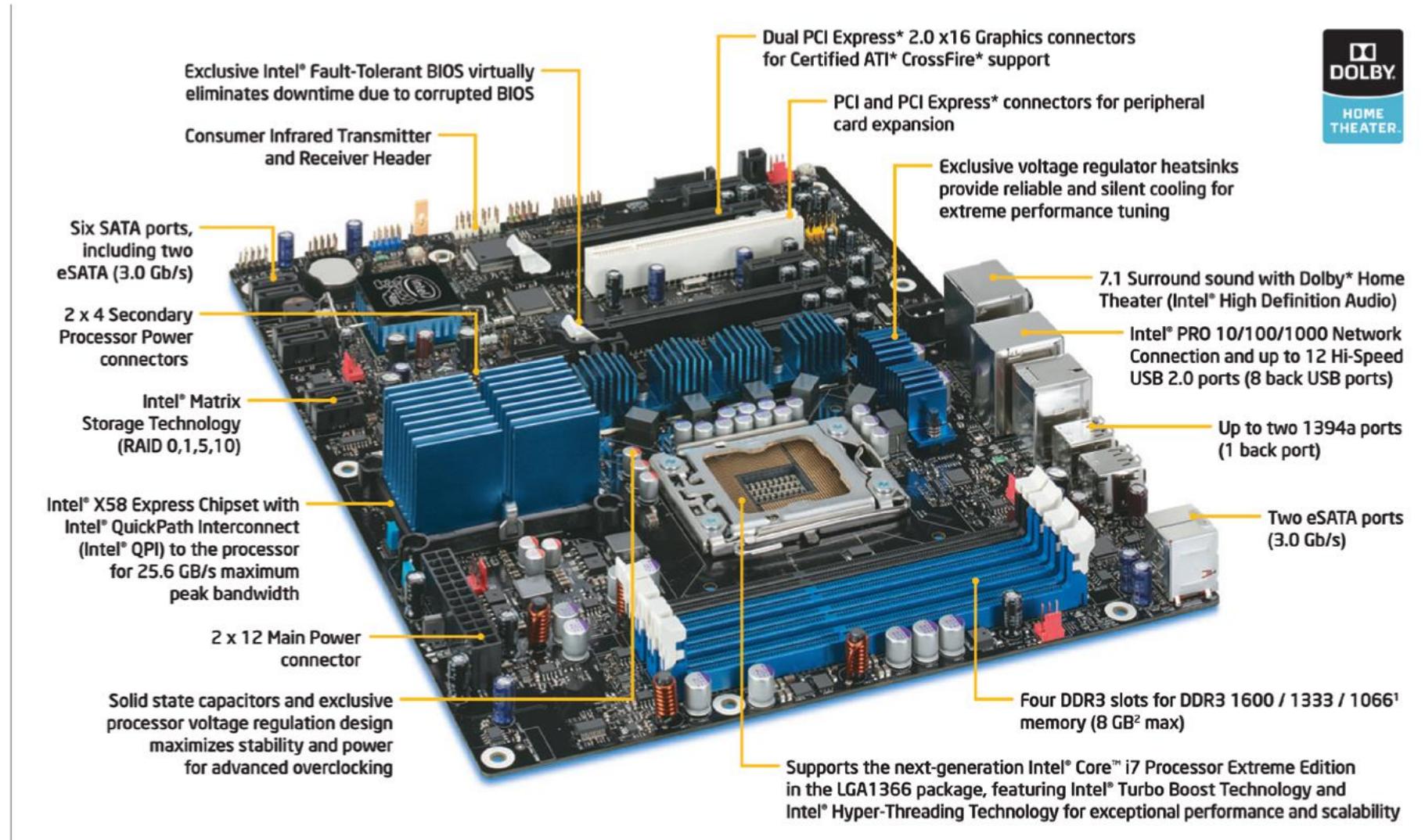
Operativsystem

- Administrerer tilgang til CPU, Minne, Utstyr og filer
- Gjør det enklere å bruke hardware
- Abstraksjonslag

Hovedkort ca 2004 (?)



Hovedkort 2012



Central Processing Unit

- Bruksområder
- Arbeidshastighet (klokkefrekvens)
- Instruksjonsett (ISA)
- Registrere
- Arithmetic Logical Unit og Float Point Unit
- Teknikker for å få opp ytelse: pipelining og parallelisering

Klassifikasjon

- Bruksområde
 - Embedded <-> Desktop <-> Tungregning
- Klokkefrekvens
- Instruksjonsett (**ISA**)
 - RISC vs CISC
- **Registerfilstørrelse**
- Antall funksjonelle enheter **ALU** og **FPU**
- Superskalaritet og parallelitet
- Socket-type
- Energibruk og kjølingsbehov

Produsenter

- Intel og AMD
 - Konstruerer og produserer CPUer, chipset m.m.
 - Fokus: **Desktop, server**, mobil, embedded
 - **C**omplex **I**nstruction **S**et **C**omputing
- MIPS og ARM
 - Konstruerer og lisenserer CPU-kjerner mm til ulike produsenter
 - Fokus: **Embedded, mobil**
 - **R**educed **I**nstruction **S**et **C**omputing

Tungregning



Tianhe-2

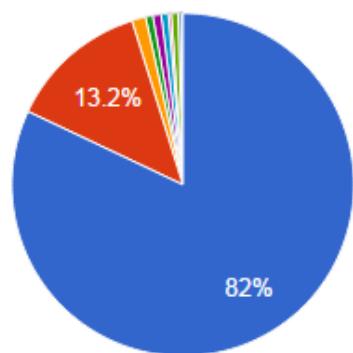
- 3.120.000 kjerner
- 33.863 Tflop/s
- 17.808 kW
- 1.024.000 GB RAM
- Kylin Linux

Verdens største
og kjappeste:
KINA

Tungregning

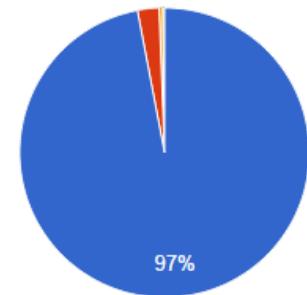
Pr juni 2014 var det ulike Intel Xeon-modeller, som kjører Linux i Clustere dominerer

Application Area Performance Share



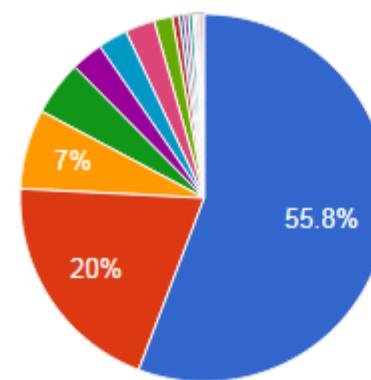
- Not Specified
- Research
- Weather and Climate Research
- Defense
- Energy
- Benchmarking
- Environment

Operating system Family System Share



- Linux
- Unix
- Windows
- Mixed

Processor Generation System Share



- Intel Xeon E5 (SandyBridge)
- Intel Xeon E5 (IvyBridge)
- Xeon 5600-series (Westmere-EP)
- Power BQC
- Opteron 6100-series "Magny-C..."
- POWER7
- Opteron 6200 Series "Interlagos"

Embedded systems

- Laget for bestemte formål
- Styres av en mikroprosessor/-**kontroller**
- Programvaren er ferdiglaget, ofte ett enkelt program, og ligger gjerne i firmware
 - Kan ofte konfigureres
 - Trenger (ofte) ikke OS
- Eksempler
 - Vaskemaskiner, hjemmeroutere, klokkeradioer, gitarstemmere, bankkort (smartcard), SIM-kort, ...



Mobiltelefoner

- Tradisjonelt benytter mobiltelefoner enkle og «billige» CPUer
 - **ARM** eller **MIPS** ISA og kjerne
 - Mobilprodusentene produserer så selv tilpassende versjoner
- Intel har ambisjoner i markedet.
- «Smart-telefoner» ligner (kanskje) mest på en fullverdig PC med eget GSM-kort



Desktop ≈ x86

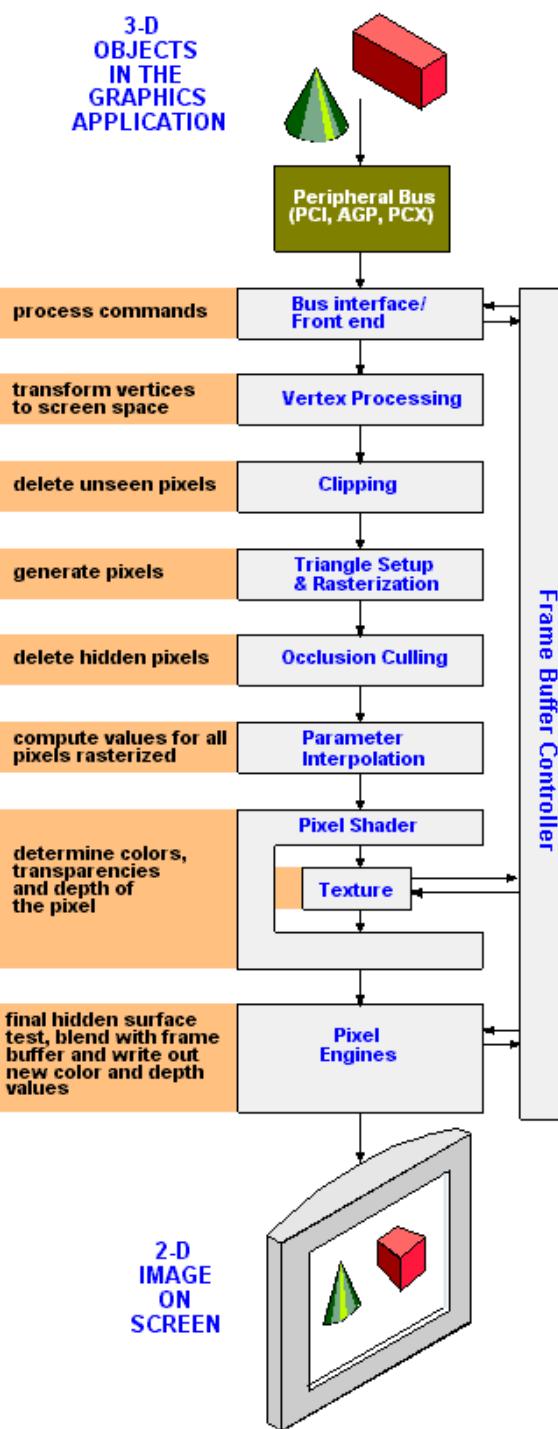
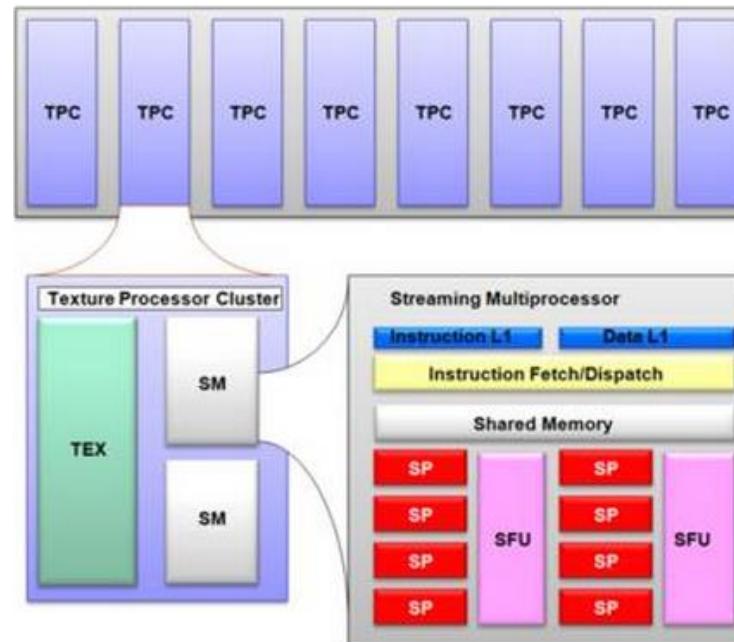
- **x86**
 - x86 er generisk betegnelse på CPUer med samme instruksjonsett som Intel har utviklet
- **Apple** startet med Motorola 68000 RISC CPU, fortsatte med IBM/Freescale PowerPC CPUer
 - Gikk over til x86 i 2005
- **Intel og AMD**
 - Videreutvikler fremdeles x86 serien

x86 instruksjonsett (CISC)

- **Data flytting**
 - Flytte data mellom registre og til/fra RAM
 - mov, push, pop, ...
- **Kontrolloverføring:**
 - Utføre betingelsessettninger (if, while) og metodekall
 - je, jg, jmp, call, ret
- **Aritmetikk/Logisk:**
 - Utføre operasjoner på data i registre
 - cmp, add, sub, inc, mul, imul, not, and, or, xor
- **Input/Output:** in, out
 - Skrive/lese til porter (og minneadresser)
- **Debug og Interrupt håndtering:**
 - int, sti, hlt, nop
- **Flyttall** har egne instruksjoner
- SIMD vektor og matriseinstruksjoner
 - MMX, 3D Now!, SSE 1-4
- Egne instruksjoner for å endre prosessortilstand (system)

Graphics Processing Unit

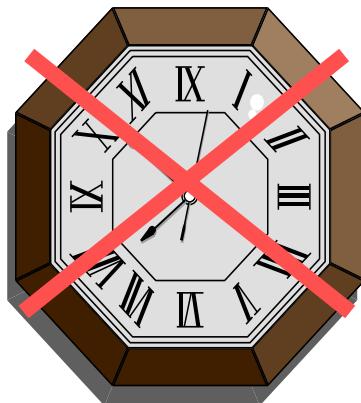
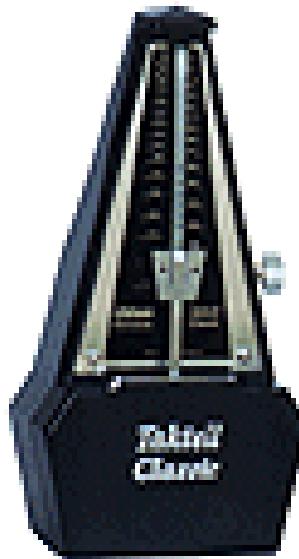
- Enten integrert på hovedkort, eller sitter på grafikkort
- Massivt parallel
- (nå) ~100vis av parallele regneenheter
- Regner på matriser (tabeller) av flyttall
- Transformerer matematiske modeller til skjermbilder (pixler)



Teknikker og begreper

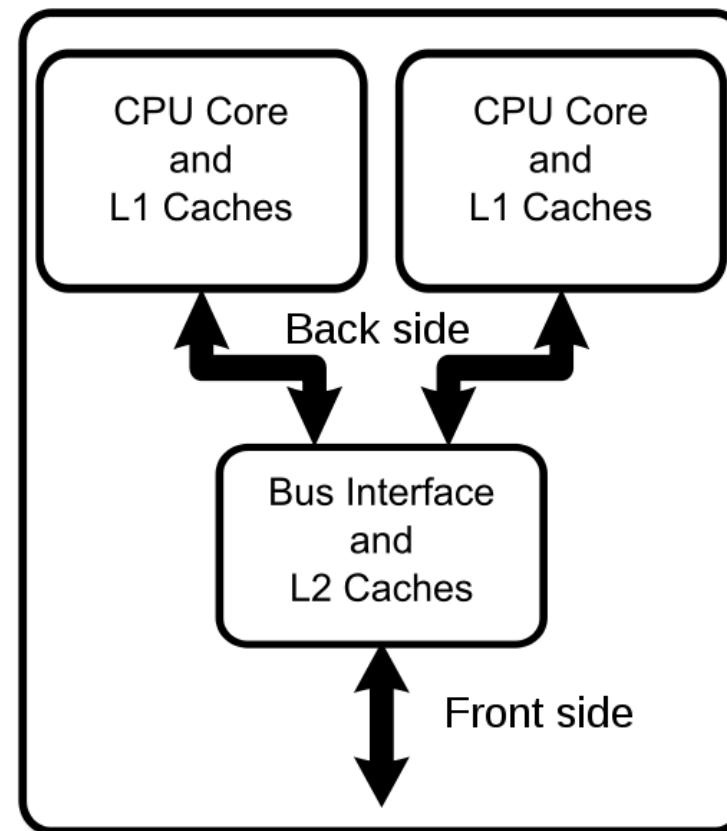
System-klokken (Hz)

- System-klokken holder takten, viser ikke tiden!



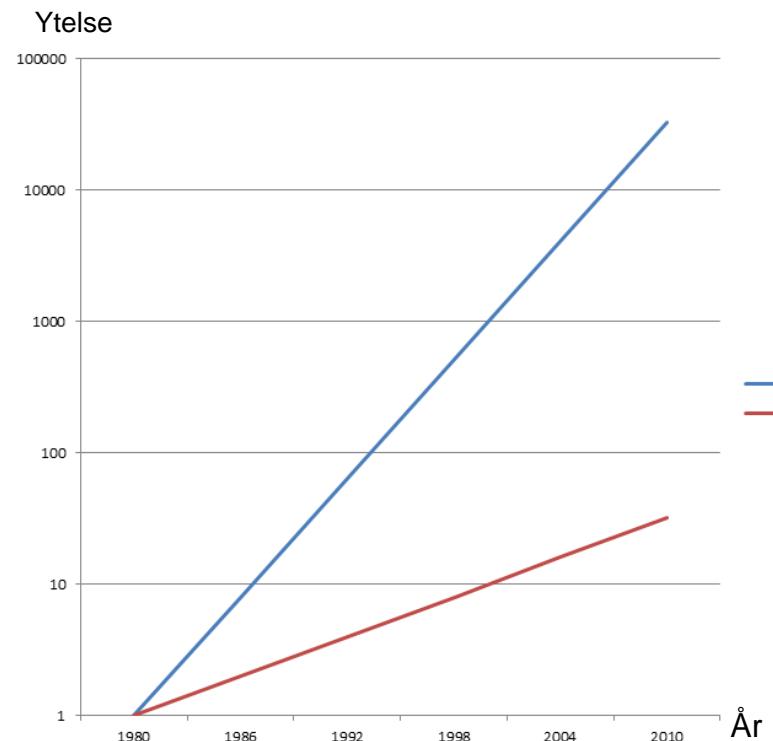
Flere «kjerner»

- CPU inneholder flere fullverdige «under-CPUer», som kan kjøre programvare parallellt.
- For at dette skal være vits i må man ha programmer som **kan** paralleliseres og ikke må kjøres rent sekvensielt!



CPU vs DRAM

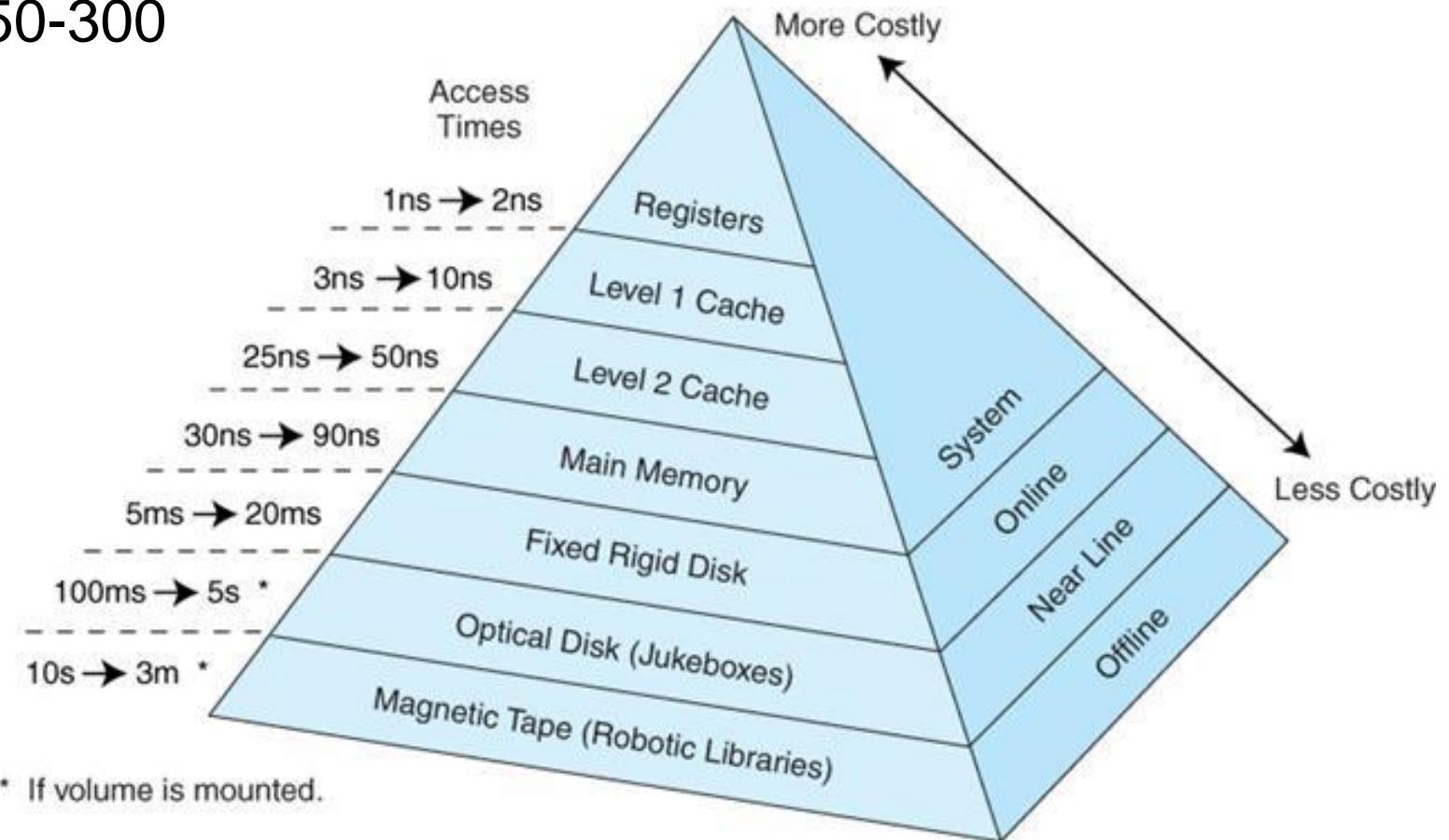
- CPU ytelse/hastighet ha doblet seg annet hvert år
- DRAM ytelse/hastighet hvert sjette år!



- Det fullstendige programmet ligger i RAM
 - Det er bare enkeltinstruksjoner og enkelta data som er på CPU
- ->PROBLEM!!!
- Utfordringen er å holde pipelinen full

Minne-hierarkiet

- Å gjøre noe i et register tar typisk en klokkepuls
- Å lese/skrive RAM typisk 150-300

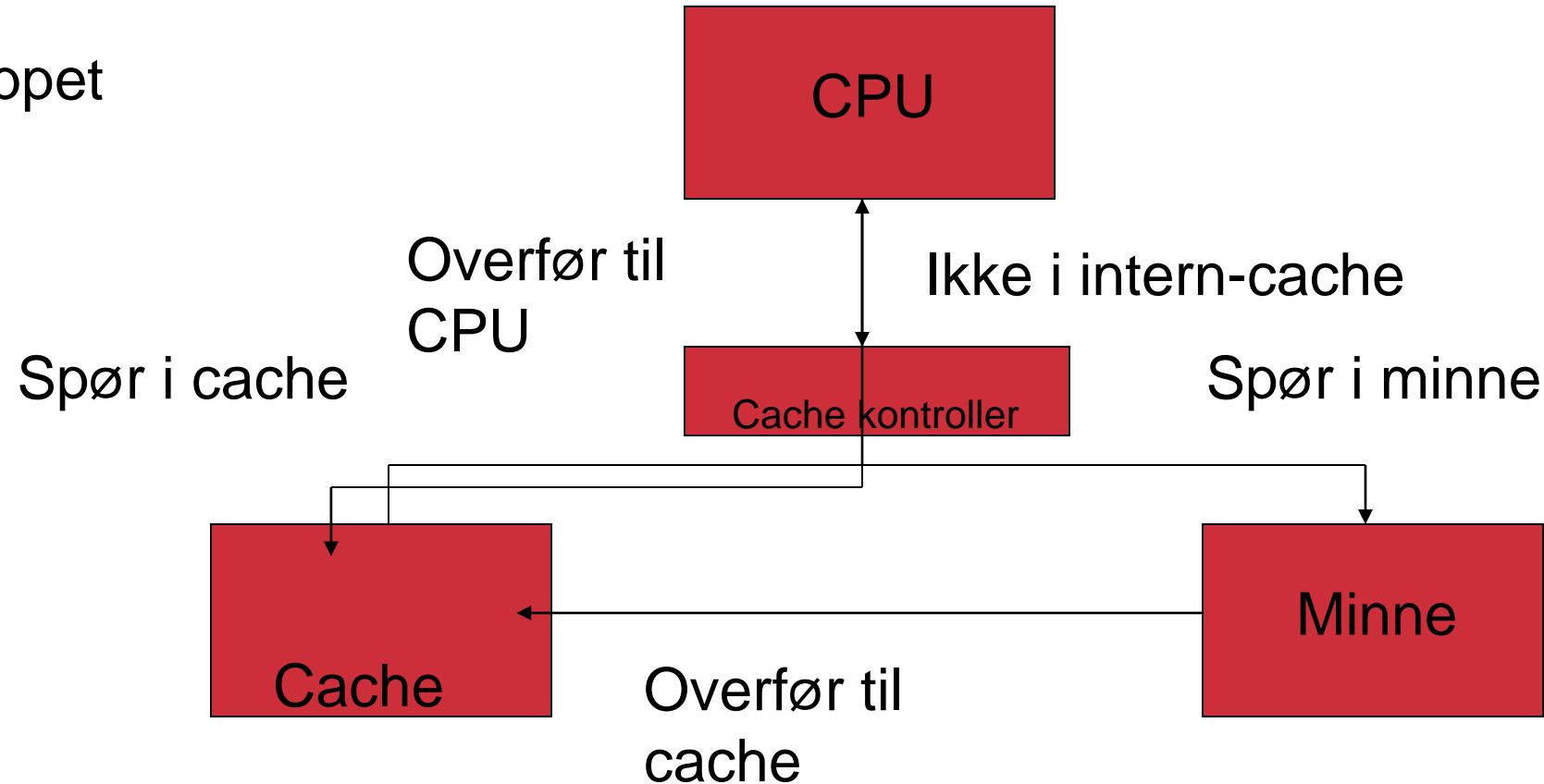


Cache

- **Lokalitetsprinsippet**
 - Ca 5 % av koden kjøres ca 95% av tiden
 - Instruksjoner som skal utføres er lagret i den rekkefølgen de skal utføres
- Cacheing benytter seg av dette ved at vi **mellomlagrer** sist innleste **instruksjoner** i **cache-minne**
 - Mye løkker medfører lett tilgjengelighet...
- + Pageing
 - Vi leser typisk ikke inn enkeltinstruksjoner til Cache, men minneblokker på 4 KiB

Cache

- I prinsippet



Forskjeller på x86 16, 32 og 64 bit?

- 16 bit arkitektur
 - 16 bits register-størrelse (word)
 - 20 bits adressering (max 1 MiB «RAM»)
 - Segmentert adressering (CS:IP f.eks.)
 - 16 bits ord (typisk 16 bit int f.eks.)
 - Ikke noe skille mellom kjøring av kjerne-kode i OS og vanlige programmer
- 32 bit (386->Pentium)
 - 32 bits register (ord-størrelse)
 - 32 bits adressering (4 GiB Virtuelt minn)
 - Flatt og støtte for virtuelt minne
 - 32 bits ord
 - Flyttall-støtte «on die»
 - Skille mellom kernel- og user-kjøring
- 64 bit (nå)
 - 64 bits registre og flere arbeidsregistre
 - Typisk 48 bits adressering (16 EiB virtuelt minne)
 - Ofte multi-kjerne
 - Enda flere instruksjoner...

Avslutning

Poeng

- CISC
 - COMPLEX Instruction Set Computer
 - Mange forskjellige instruksjoner
 - Instruksjoner har ulik lengde i bit/byte
- RISC
 - REDUCED Instruction Set Computer
 - Bare de instruksjonene som (erfaringsmessig) brukes mye.
 - Gir enklere elektronikk og dermed raskere utførelse.
 - Jf byte-kode på JVM
- Moderne Intel/AMD CPUer
 - "begge deler"
 - CISC eksternt
 - RISC mikroinstruksjoner på selve prosesoren

Flere poeng

- Prosessoren kjører instruksjonene i den rekkefølgen de ligger i minnet («**sekvensielt**»).
- På **ulike datatyper** benyttes helt **forskjellige instruksjoner**
 - 7 + 5:
 mov AX, 7
 add AX, 5
 resulterer i at det ligger 12 i AX-registeret
 - 7.0 + 5.0:
 FLD «minneadressen der 7.0 ligger»
 FLD 0012AC45
 FADD ST, ST(1)
 FSTP «adressen svaret skal legges»
 - 'Hei' + 'Hå!'
 mengdevis av instruksjoner som flytter begge strengene inn i felles minneområde og legge til slutt-markøren (typisk ASCII 0x00).

Flere poeng

- Hva er en variabel?
 - På instruksjonsnivå: **adressen** til et sted minnet (eller et register)
- Hvorfor har variabler «scope»
 - Å kalle en metode medfører at man begynner å kjøre instruksjoner et helt annet sted i minnet
 - Når man hopper tilbake til der metoden ble kalt forsvinner alt annet enn det som metoden returnerer (typisk en adresse til en verdi...)

Hva skal vi kunne?

- Boolsk algebra
 - NOT, AND, OR, XOR
- Hva en *instruksjon* er.
 - hvordan bygd opp
 - hvilke typer
- Von Neumann modellen (igjen)
 - mindre «black box»
- Forskjellen på CISC og RISC
- Litt om x86 og alternativer

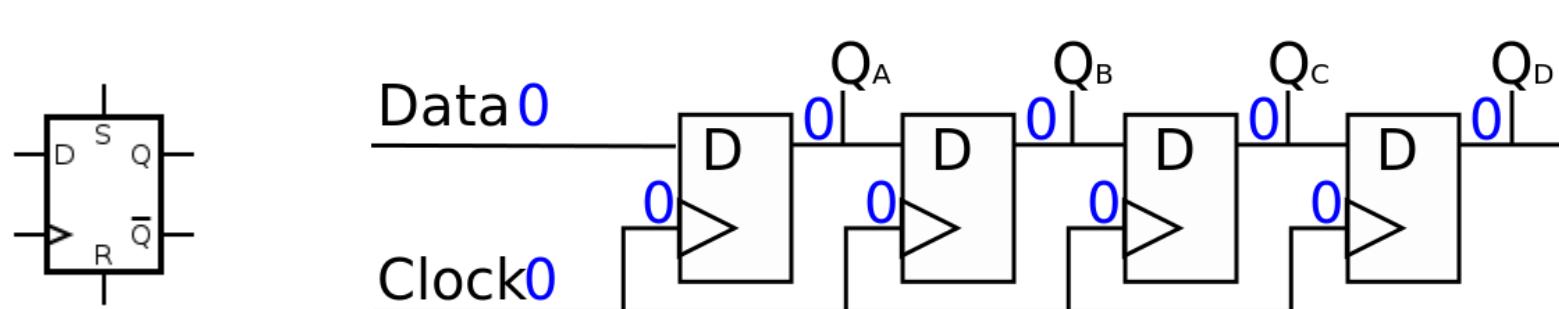
For valgfritt egenstudie

For de som ønsker å lære noen emner mer i dybden for å forstå det bedre er det her samlet noen ekstra temaer relatert til dagens undervisning, det må forventes en del egenarbeid for å forstå disse emnene.

Det vil ikke komme spørsmål på eksamen fra disse, og dette er altså ikke ansett for å være en del av pensum.

Sekvensiell vs Kombinatorisk

- Så langt har vi kun sett på det som kalles **kombinatorisk** logikk
 - **Output** bestemmes utelukkende av **Input** (og tid)
- **Sekvensiell** logikk har også «hukommelse»
 - Output bestemmes av Input **og** lagrede bits.
 - Brukes bl.a. til å bygge **register** (minne-celler)



Boolean som datatype

- I mange sammenhenger er det ikke samsvar mellom verdiene 0 og 1 og boolske datatyper.
 - Benyttes ofte til **utsagnslogikk**, der er vi interessert i «true» og «false», ikke 1/0
 - Utsagnet `4==4` har sannhetsverdi **true**
 - Utsagnet `3==5` har sannhetsverdi **false**
 - Hvordan «true» og «false» faktisk **kodes binært** er språk-/kompilator-avhengig
- I Java (og andre C-språk) så har man ulike logiske operatorer avhengig om det er sannhetsverdi eller bit-kombinering man vil utføre
 - `&` er **bitwise**, `&&` er **logisk**
 - Dermed blir **`3&4 = 0`** (`0011 & 0100 = 0000`),
 - mens **`3&&4`** gir «true», «syntax error», eller 4...
 - «*Dette kommer det mer av i både PG1100 og DB1100*» har sannhetsverdi **true**

Programmering

Bitwise AND	skrives	&
Bitwise OR	skrives	
Bitwise XOR	skrives	^
Bitwise NOT	skrives	~
Logisk NOT	skrives	!
Logisk AND	skrives	&&
Logisk OR	skrives	

LA OSS DESIGNE EN ENKEL DATAMASKIN!

- ENKEL!!! (Forstå prinsippene)
- Primitiv
- Vi lager ikke elektronikken
- Maskinen kan likevel gjøre noen ting
- Nærmere beskrivelse finner dere i **Kompendium 1**

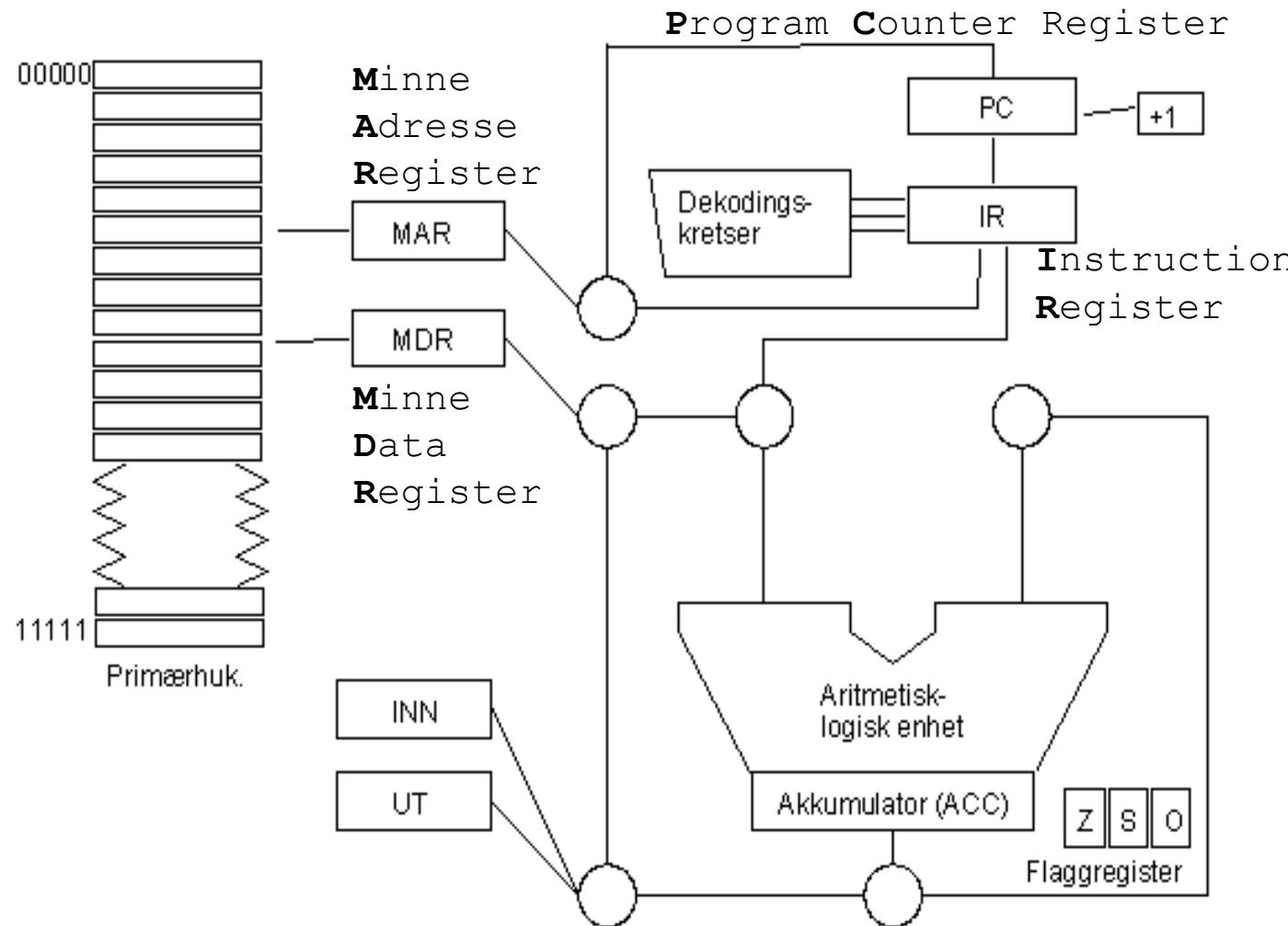
Hva var arkitektur?

- Hvordan bygger vi opp de *instruksjonene* som computeren skal utføre (eksekvere)
 - Hvilke typer instruksjoner skal vi ha?
 - Hvordan skal adressering foregå
- Når vi har bestemt instruksjonssett (**Instruction Set Architecture**) kan vi designe en CPU
- I praksis blir dette en vekselvirkning mellom maskin-design og ISA-design
 - Vi har ett transistor-budsjett

Register

- Alle bits på CPU må legges i et **register**
 - Et register er adresserbart minne (SRAM) som ligge inne på CPU
- I Assembly-språk har typisk registrene egne «navn» og roller
 - Registrene som brukes til å manipulere variabler og verdier omtales som **arbeidsregister**
- F.eks. i X86-64
 - **RIP** er registeret som inneholder adressen til hvor i minnet neste instruksjon skal hentes fra
 - **RAX** er registeret som oftest brukes til å lagre resultater i (akkumulator)
 - **RCX** brukes oftest til nedtelling (tenk for-løkke)
 - + mange flere (minnebehandling, kjøring,...)

PrimRisc v. 0.1 layout



- Akkumulator-basert
 - Kun **ett** arbeids-register
- 8 bit adressering
 - Bruker bare 5
- Load/store
 - Alt går via RAM
 - Enkelt
- Kan «egentlig» bare addere

Register-«filen»

- **PC** (Programteller).
 - Dette registeret vil til enhver tid inneholde adressen til den lokasjon i primærlageret som neste instruksjon skal hentes fra.
- **IR** (Instruksjonsregister).
 - Dette registeret oppbevarer instruksjoner etter at de har blitt hentet fra primærlageret og før de sendes over i dekoderkretsene.
- **MAR** (adresseregister).
 - Når vi ønsker å lese noe fra, eller legge noe inn i primærlageret må vi først plassere adressen i MAR.
- **MDR** (dataregister).
 - Når vi gjør en leseoperasjon fra primærlageret, plasseres resultatet her. Skal vi skrive til primærlageret, må vi først plassere det vi skal skrive i MDR
- **ACC** (Accumulator).
 - Dette registret brukes først og fremst i forbindelse med regneoperasjoner. Det er også mulig å lese innholdet av en minneadresse inn i akkumulatoren med instruksjonen LOAD, og lagre innholdet av akkumulatoren med instruksjon STORE.
- **FLAGG-registeret**
 - Trenger vi i forhold til å kunne gjøre betingede hopp (if, while og lignende)
 - Z-flagget (Zero-flag) blir satt til 1 hvis resultatet av en regneoperasjon blir null.
 - S-flagget (Sign-flag) blir satt til 1 hvis resultatet av en regneoperasjon blir negativt.
 - O-flagget (Overflow-flag) blir satt til 1 hvis resultatet av en regneoperasjon blir for stort (overflyt) for å få plass i ACC-registeret

Instruksjoner

Mnemonic	Opkode	Beskrivelse	Funksjon
STOP	0000	Stopp maskinen	
LOAD	0001	Les inn i akkumulator	$(ACC) \leftarrow (MDR)$
STORE	0010	Lagre akkumulator	$(MDR) \leftarrow (ACC)$
ADD	0011	Adder til akkumulator	$(ACC) \leftarrow (ACC) + (MDR)$
SUB	0100	Subtraher fra akkumulator	$(ACC) \leftarrow (ACC) - (MDR)$
BRANCH	0101	Hopp uansett	$(PC) \leftarrow IR<4:0>$
BRZERO	0110	Hopp hvis resultat = 0	$(PC) \leftarrow IR<4:0>$ eller $(PC) \leftarrow (PC) + 1$
BRNEG	0111	Hopp hvis resultat < 0	$(PC) \leftarrow IR<4:0>$ eller $(PC) \leftarrow (PC) + 1$
BROVFL	1000	Hopp hvis overflyt	$(PC) \leftarrow IR<4:0>$ eller $(PC) \leftarrow (PC) + 1$
IN	1001	Les inn fra tastatur	$(ACC) \leftarrow INN$
OUT	1010	Skriv til skjerm	$UT \leftarrow (ACC)$

- Alle instruksjoner er 16 bit (2 byte)

- 1 byte til **opkode**
 - Bruker bare 4 bit (i denne versjonen)
- 1 byte i instruksjonen til adresser
 - Bruker bare 5 bit (i denne versjonen)

Eksempel på et program i minnet

Minneadresse binært/(desimalt)	Maskinkode (opkode+adresse) binært/(desimalt)	Asembly-instruksjon
00000 (0)	0001 0000 0000 0100 (1) (4)	LOAD 4
00001 (1)	0011 0000 0000 0101 (3) (5)	ADD 5
00010 (2)	0010 0000 0000 0110 (2) (6)	STORE 6
00011 (3)	0011 0000 0000 0000 (0) (-)	STOP
00100 (4)	0000 0000 0000 0001 (0) (1)	
00101 (5)	0000 0000 0000 0010 (-) (2)	
00110 (6)	0000 0000 0000 0000 (-) (0)	

```
/* Last inn i akkumulatoren data fra adresse 4,  

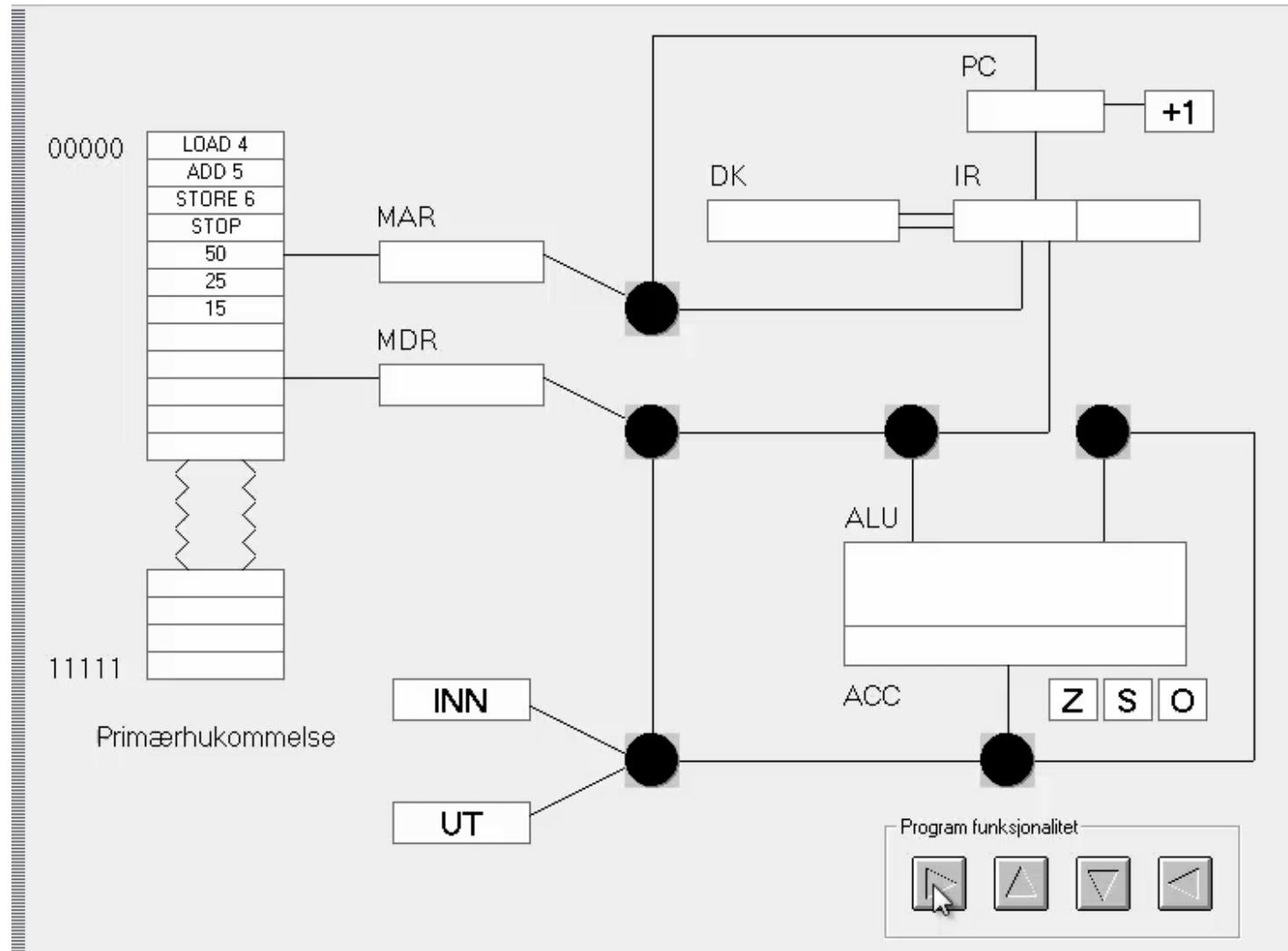
adder med data fra adresse 5,  

lagre på adressen som ligger på adresse 6,  

Avslutt kjøringen */
```

- Kan bruke en simulator av denne arkitekturen (buggy) på dagens øving
(Se http://youtu.be/igthwn_qGVI)

Test



Analyse/Kritikk av PrimRISC 1

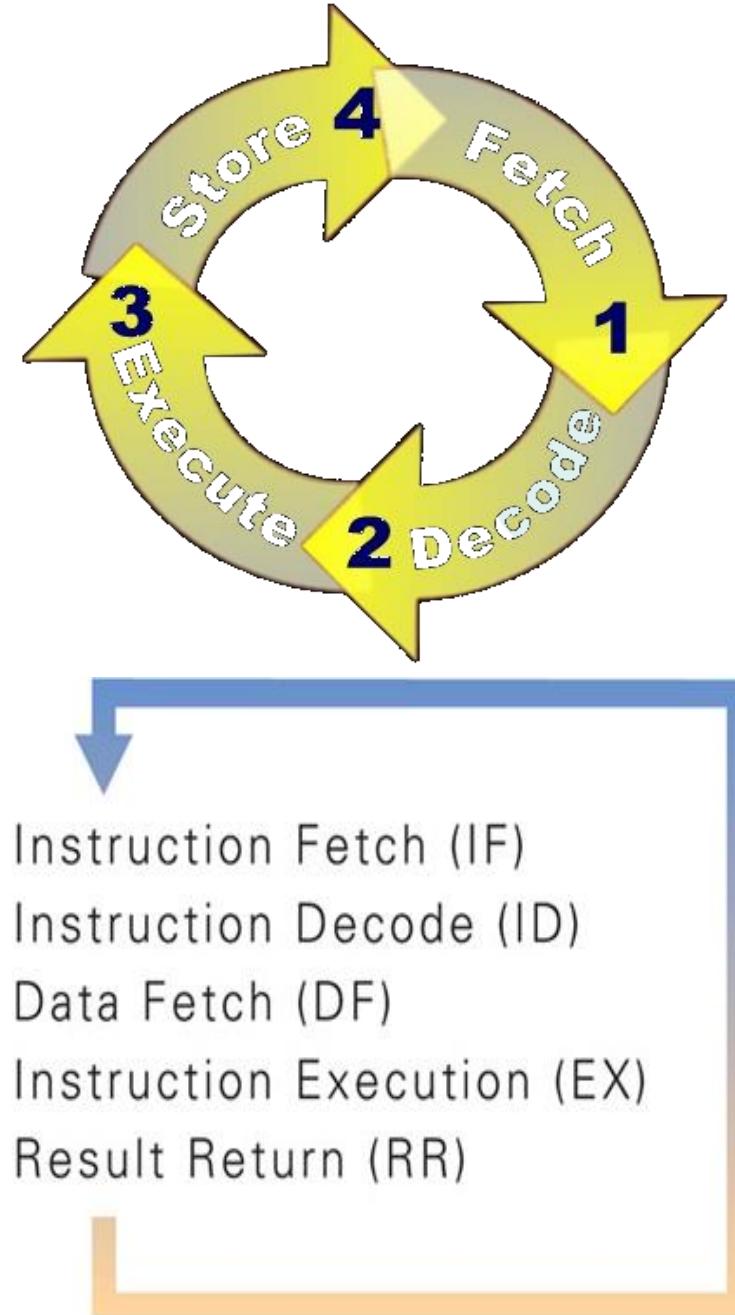
- Grei til å demonstrere en **Fetch-Execute** syklus og hvordan instruksjoner *kan* være bygd opp
- Mangler **Pipelining**
- Fattig instruksjonssett
 - 8 mot x86-64 sine ~1000
- Støtter ikke Systemprogrammering (OS), multitasking eller virtuelt minne
- «Ganske klein, men dog så enkel»

Kritikk 2

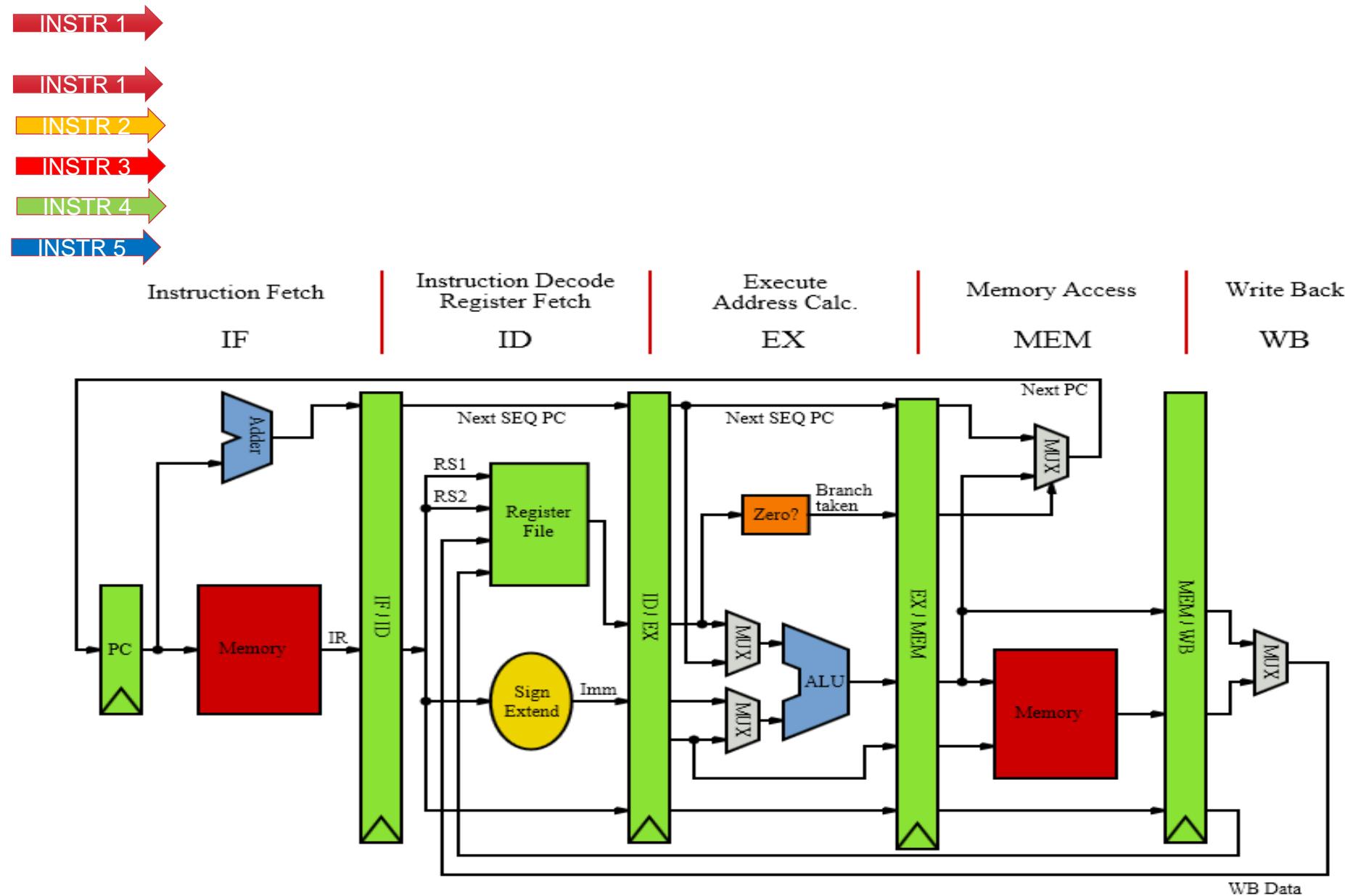
- **Har støtte for betingede hopp**
 - Kan implementere if, for, while o.l.
- **Mangler (skikkelig) HW-støtte for metoder!!!**
 - I Intel/AMD løses dette med egne registre (SS og SP) og instruksjoner (call) som støtter bruk av et eget minneområde som **stack**.
- **Har ikke støtte for interrupt!**
 - Mer om dette de to neste gangene..

Fetch-Execute-syklusen

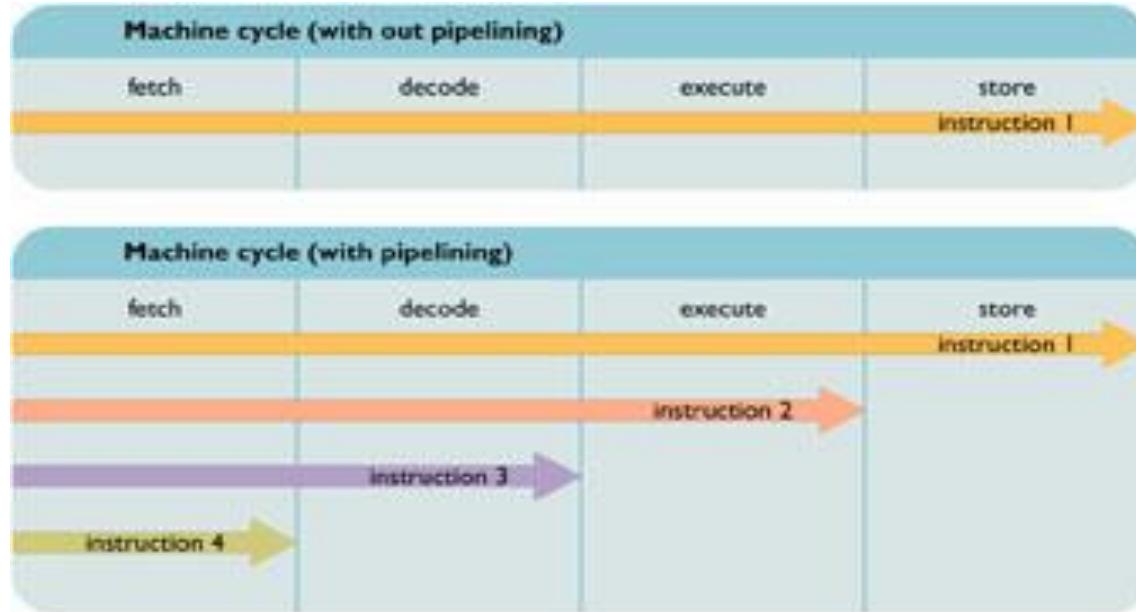
1. **Fetch**: hente instruksjon fra minne og inn i register
2. **Decode**: Tolke instruksjonen og sette opp kretsene som utfører den
3. **Execute**: Utføre instruksjonen på data/registre (endre tilstand)
4. **Store**: Lagre resultat (tilbake i minnet)



Ex: MIPS

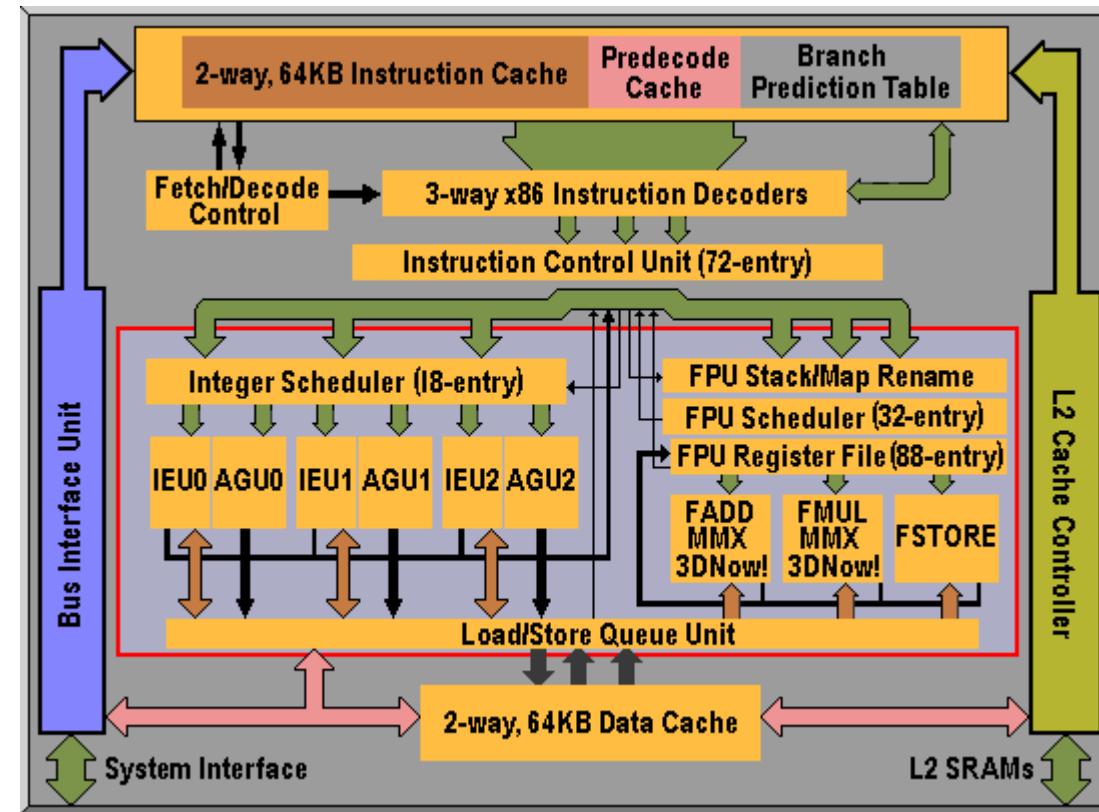


Pipelining



- Siden det er forskjellige enheter på CPU som henter, dekoder, utfører og lagrer kan disse kjøres i parallel («samlebånd»)
 - Kan starte å hente en ny instruksjon så fort denne er overlatt til dekoding
- I moderne Intel/AMD CPUer det «dype og lange» pipelines..
 - 15-25 trinn ->

Superskalaritet

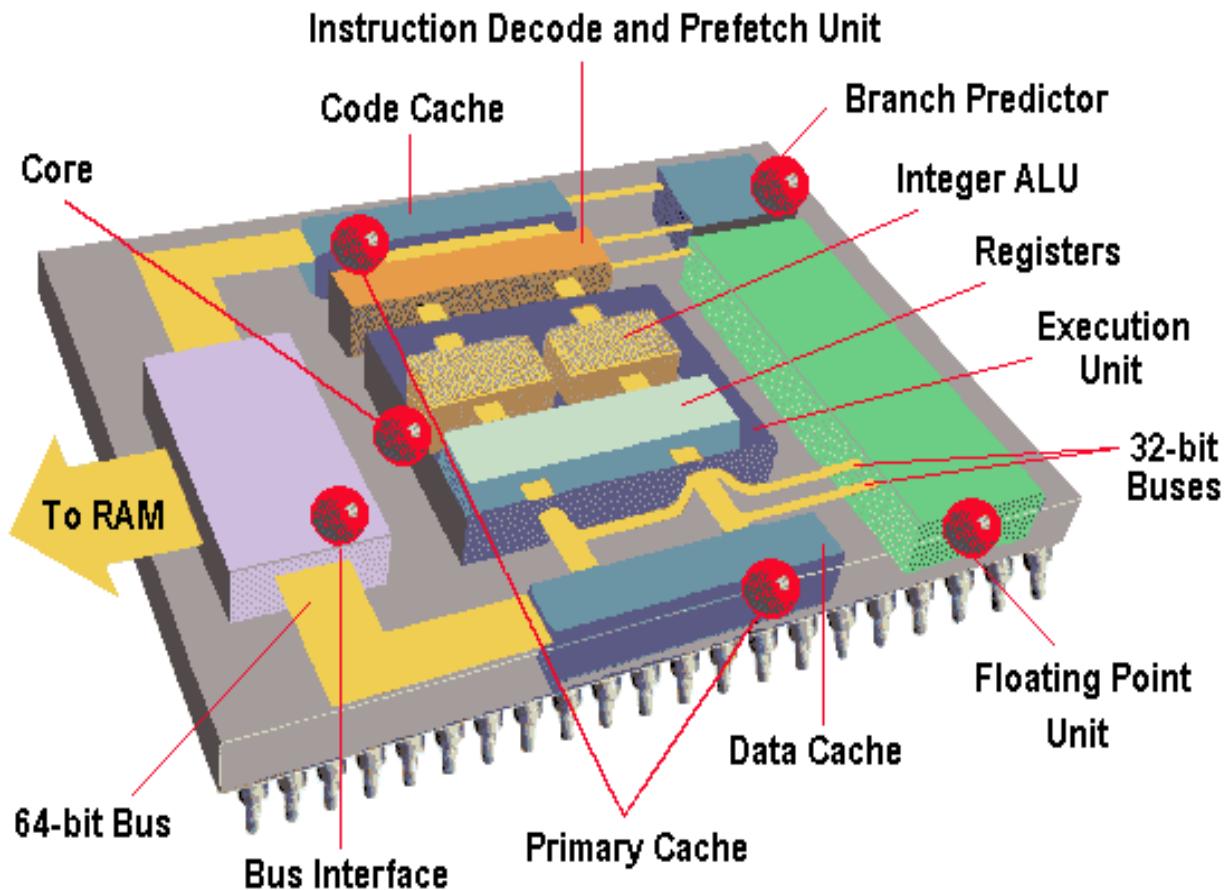


AMD Athlon hadde tre ALUer
og en SIMD flyttallenhet

- Superskalaritet er en form for parallelisering der vi har **flere** (parallele) **ALU** og **FPU** tilgjengelig på CPU
- Kan da utføre (enda) flere instruksjoner samtidig!

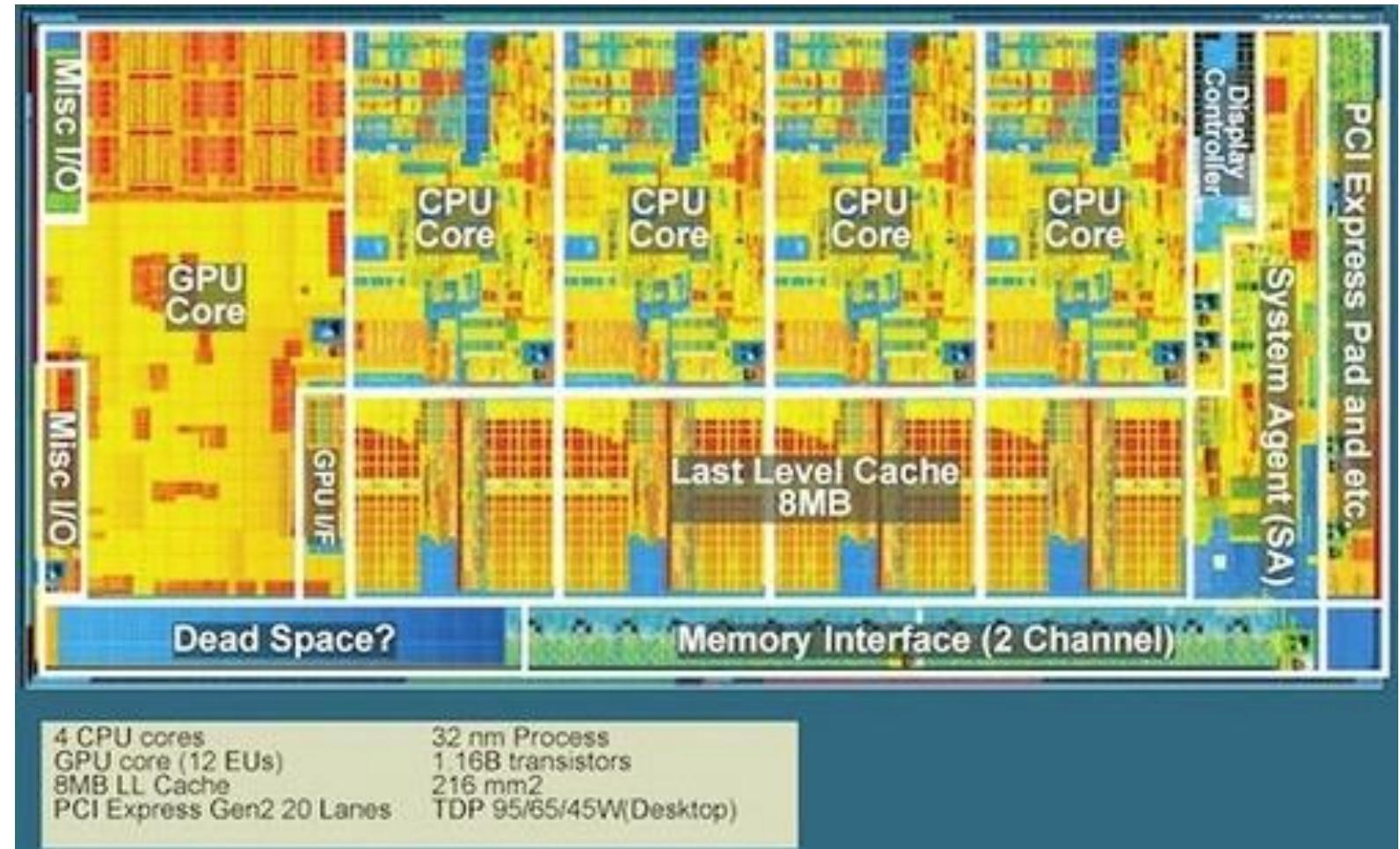
Pentium (IA32) layout (forenklet)

- Branch Prediction
 - «Gjetter» hvilke instruksjoner som skal kjøres etterpå og henter dem inn på CPU
- Cacheing
 - **Instruksjoner** i egen cache (skal aldri tilbake til RAM!)
 - **Data** i egen cache

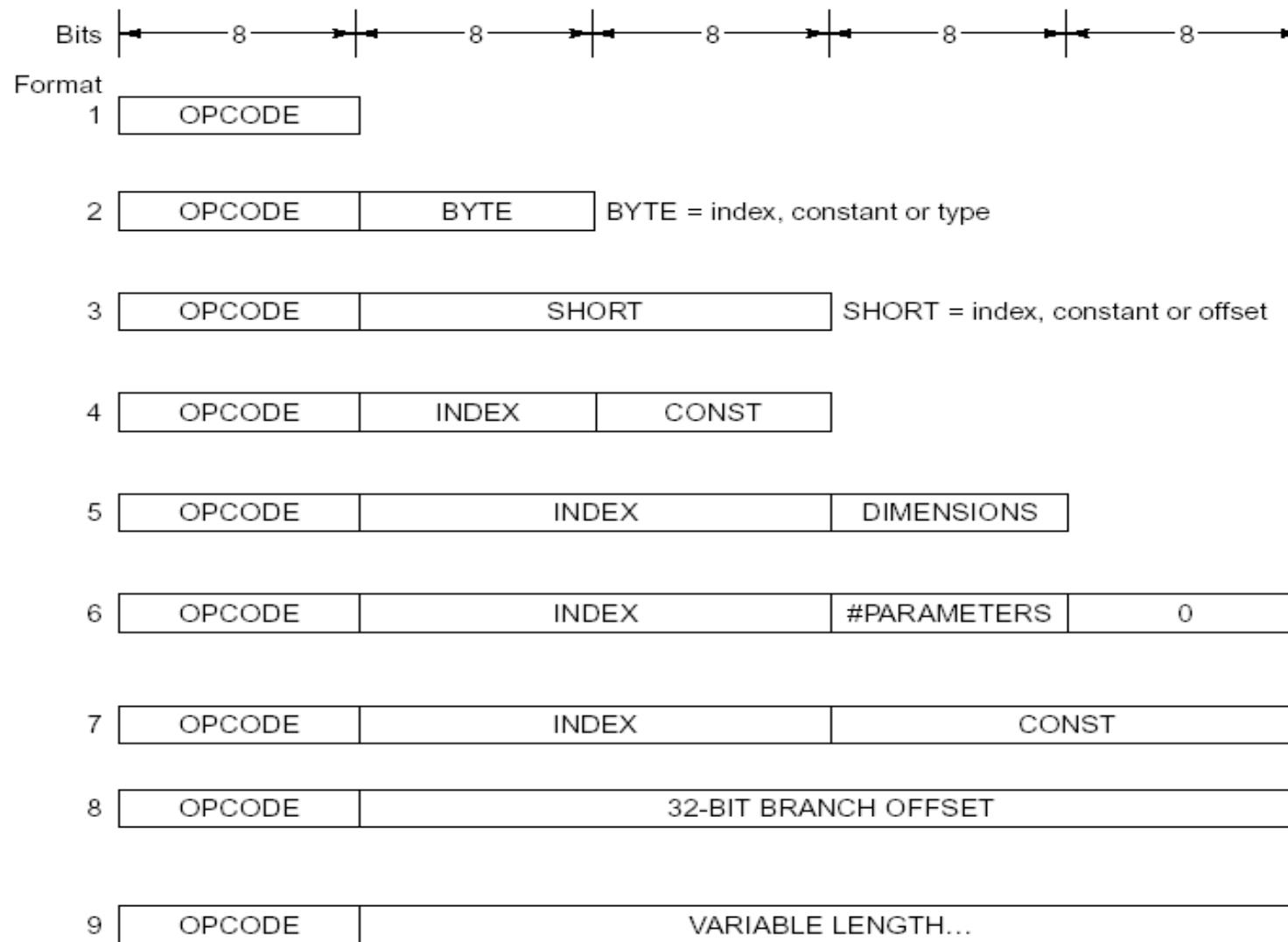


Sandy Bridge

- Flere «kjerner» -> parallelkjøring av program(-tråder)
- Flyttet Northbridge (og GPU) «on die»



JVM ISA: Format

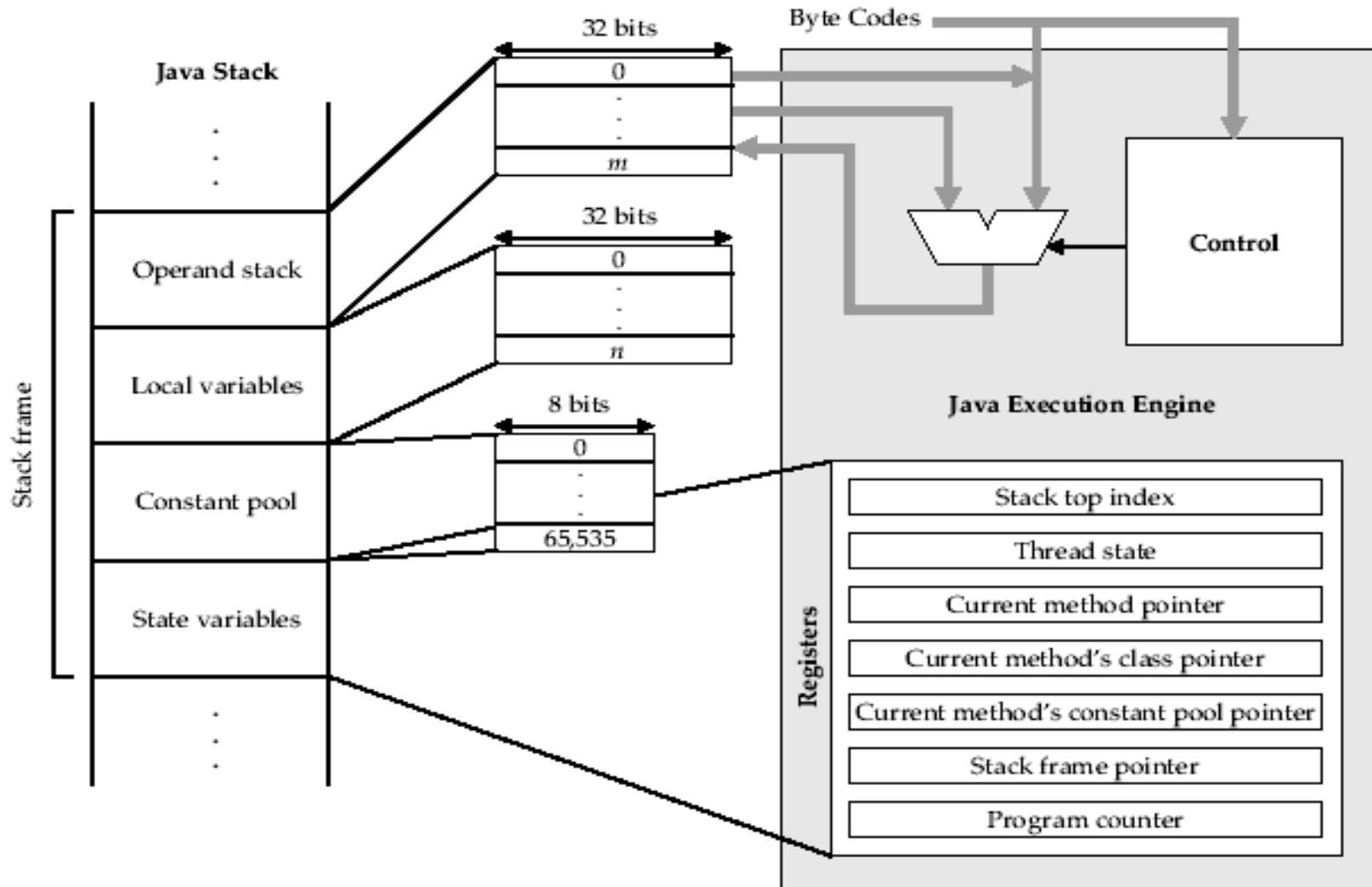


JVM: 226 instruksjoner

		Boolean/shift	Comparison
Loads		IIAND Boolean AND IIOR Boolean OR IIXOR Boolean EXCLUSIVE OR IISHL Shift left IISHR Shift right IU SHR Unsigned shift right	IF_ICMPREL OFFSET16 Conditional branch IF_ACMPEQ OFFSET16 Branch if two ptrs equal IF_ACMPNE OFFSET16 Branch if ptrs unequal IFREL OFFSET16 Test 1 value and branch IFNULL OFFSET16 Branch if ptr is null IFNONNULL OFFSET16 Branch if ptr is nonnull LCMP Compare two longs FCMPL Compare 2 floats for < FCMPG Compare 2 floats for > DCMPL Compare doubles for < DCMPG Compare doubles for >
Stores		x2y Convert x to y I2c Convert integer to char I2b Convert integer to byte	
Pushes		DUPxx Six instructions for duping POP Pop an int from stk and discard POP2 Pop two ints from stk and discard SWAP Swap top two ints on stack	
Arithmetic		IINC IND8,CON8 Increment local variable WIDE Wide prefix NOP No operation GETFIELD IND16 Read field from object PUTFIELD IND16 Write field to object GETSTATIC IND16 Get static field from class NEW IND16 Create a new object INSTANCEOF OFFSET16 Determine type of obj CHECKCAST IND16 Check object type ATHROW Throw exception LOOKUPSWITCH ... Sparse multiway branch TABLESWITCH ... Dense multiway branch MONITORENTER Enter a monitor MONITOREXIT Leave a monitor	
Transfer of control			INVOKEVIRTUAL IND16 Method invocation INVOKESTATIC IND16 Method invocation INVOKEINTERFACE ... Method invocation INVOKESPECIAL IND16 Method invocation JSR OFFSET16 Invoke finally clause typeRETURN Return value ARETURN Return pointer RETURN Return void RET IND8 Return from finally GOTO OFFSET16 Unconditional branch
Arrays			ANEWARRAY IND16 Create array of ptrs NEWARRAY ATYPE Create array of atype MULTINEWARRAY IND16, D Create multidim array ARRAYLENGTH Get array length

IND8/16 = index of local variable type, x, y = I, L, F, D
 CON8/16, D, ATYPE = constant OFFSET16 for branch

Eksempel: JVM (Java RunTime Enviroment)



Eksempel:

Kode og klasse

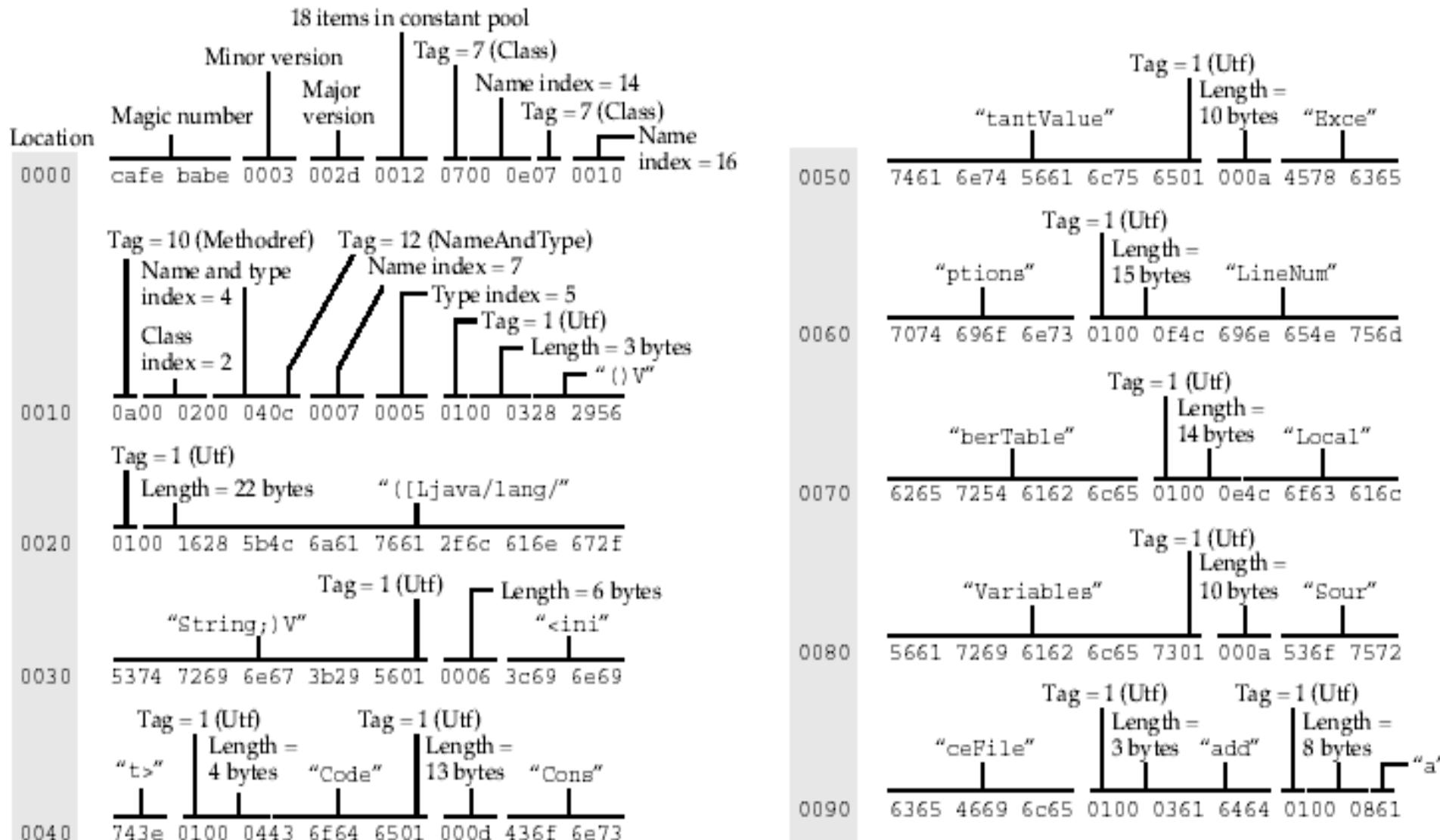
```

public class add {
    public static void main(String args[]) {
        int x=15, y=9, z=0;
        z = x + y;
    }
}

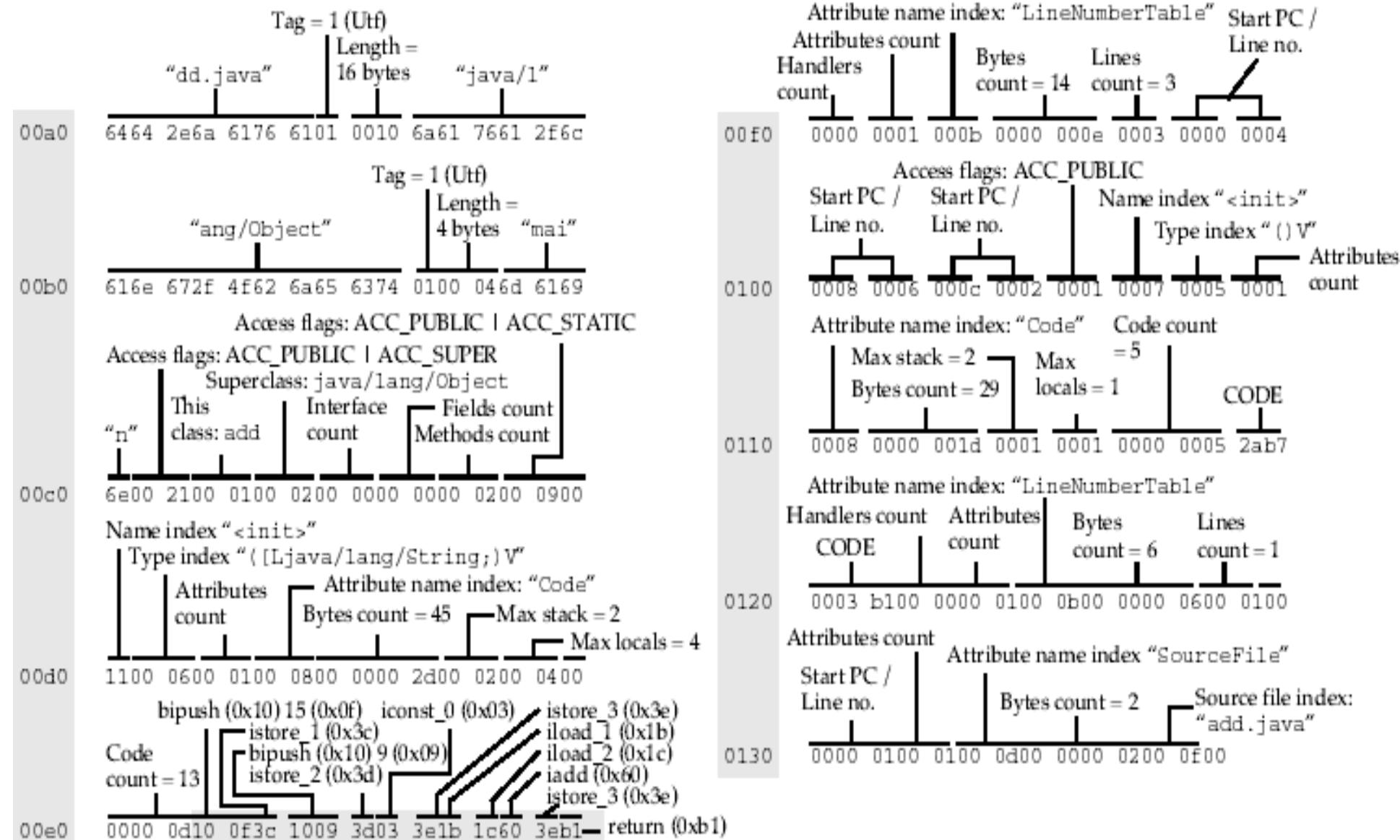
```

0000	cafe	babe	0003	002d	0012	0700	0e07	0010
0010	0a00	0200	040c	0007	0005	0100	0328	2956().V
0020	0100	1628	5b4c	6a61	7661	2f6c	616e	672f	...([Ljava/lang/
0030	5374	7269	6e67	3b29	5601	0006	3c69	6e69	String;)V...<ini
0040	743e	0100	0443	6f64	6501	000d	436f	6e73	t>...Code...Cons
0050	7461	6e74	5661	6c75	6501	000a	4578	6365	tantValue...Exce
0060	7074	696f	6e73	0100	0f4c	696e	654e	756d	ptions...LineNum
0070	6265	7254	6162	6c65	0100	0e4c	6f63	616c	berTable...Local
0080	5661	7269	6162	6c65	7301	000a	536f	7572	Variables...Sour
0090	6365	4669	6c65	0100	0361	6464	0100	0861	ceFile...add...a
00a0	6464	2e6a	6176	6101	0010	6a61	7661	2f6c	dd.java...java/l
00b0	616e	672f	4f62	6a65	6374	0100	046d	6169	ang/Object...mai
00c0	6e00	2100	0100	0200	0000	0000	0200	0900	n.....
00d0	1100	0600	0100	0800	0000	2d00	0200	0400
00e0	0000	0d10	0f3c	1009	3d03	3e1b	1c60	3eb1
00f0	0000	0001	000b	0000	000e	0003	0000	0004
0100	0008	0006	000c	0002	0001	0007	0005	0001
0110	0008	0000	001d	0001	0001	0000	0005	2ab7
0120	0003	b100	0000	0100	0b00	0000	0600	0100
0130	0000	0100	0100	0d00	0000	0200	0f00	0000

Klassefilen (1)



Klassefilen (2)



Byte-koden

<u>Location</u>	<u>Code</u>	<u>Mnemonic</u>	<u>Meaning</u>
0x00e3	0x10	bipush	Push next byte onto stack
0x00e4	0x0f	15	Argument to bipush
0x00e5	0x3c	istore_1	Pop stack to local variable 1
0x00e6	0x10	bipush	Push next byte onto stack
0x00e7	0x09	9	Argument to bipush
0x00e8	0x3d	istore_2	Pop stack to local variable 2
0x00e9	0x03	iconst_0	Push 0 onto stack
0x00ea	0x3e	istore_3	Pop stack to local variable 3
0x00eb	0x1b	iload_1	Push local variable 1 onto stack
0x00ec	0x1c	iload_2	Push local variable 2 onto stack
0x00ed	0x60	iadd	Add top two stack elements
0x00ee	0x3e	istore_3	Pop stack to local variable 3
0x00ef	0xb1	return	Return

Java bytecode vs x86 instruksjoner

- Java bytecode kan kalles et **RISC**-type instruksjonssett
- x86 (Intel, så vel som AMD) er **CISC**
- De to neste foilene viser:
 - De eldste instruksjonene som fremdeles er med (256 av dem, det fins mange, mange flere flere)
 - Eksempel på en IA-32 instruksjon

X86 – Grunnleggende Opkode-rom

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ADD rmb,rb	ADD rmv,rv	ADD rb,rmh	ADD rv,rmv	ADD AL,ib	ADD eAX,iv	PUSH ES	POP ES	OR rmb,rb	OR rmv,rv	OR rb,rmh	OR rv,rmv	OR AL,ib	OR eAX,iv	PUSH CS	386 space
1	ADC rmb,rb	ADC rmv,rv	ADC rb,rmh	ADC rv,rmv	ADC AL,ib	ADC eAX,iv	PUSII SS	POP SS	SBB rmb,rb	SBB rmv,rv	SBB rb,rmh	SBB rv,rmv	SBB AL,ib	SBB eAX,iv	PUSH DS	POP DS
2	AND rmb,rb	AND rmv,rv	AND rb,rmh	AND rv,rmv	AND AL,ib	AND eAX,iv	ES:	DAA	SUB rmb,rb	SUB rmv,rv	SUB rb,rmh	SUB rv,rmv	SUB AL,ib	SUB eAX,iv	CS:	DAS
3	XOR rmb,rb	XOR rmv,rv	XOR rb,rmh	XOR rv,rmv	XOR AL,ib	XOR eAX,iv	SS:	AAA	CMP rmb,rb	CMP rmv,rv	CMP rb,rmh	CMP rv,rmv	CMP AL,ib	CMP eAX,iv	DS:	AAS
4	INC eAX	INC eCX	INC eDX	INC eBX	INC eSP	INC eBP	INC eSI	INC eDI	DEC eAX	DEC eCX	DEC eDX	DEC eBX	DEC eSP	DEC eBP	DEC eSI	DEC eDI
5	PUSH eAX	PUSH eCX	PUSH eDX	PUSH eBX	PUSH eSP	PUSH eBP	PUSH eSI	PUSH eDI	POP eAX	POP eCX	POP eDX	POP eBX	POP eSP	POP eBP	POP eSI	POP eDI
6	PUSHA PUSHAD	POPA POPAD	BOUND rv,m2v	ARPL rm2,r2	FS:	GS:	OpLen	AdLen	PUSH iv	IMUL rmv,rv,iv	PUSH ib	IMUL rv,rmv,ib	INSB	INSD	OUTSB	OUTSD
7	JO ib	JNO ib	JB ib	JAE ib	JE ib	JNE ib	JBE ib	JA ib	JS ib	JNS ib	JP ib	JNP ib	JL ib	JGE ib	JLE ib	JG ib
8	Immed rmb,ib	Immed rmv,iv		Immed rmv,ib	TEST rmb,rb	TEST rmv,rv	XCHG rmb,rb	XCHG rmv,rv	MOV rmb,rb	MOV rmv,rv	MOV rb,rmh	MOV rv,rmv	MOV rm,segr	LEA rv, m	MOV segr,rm	POP rmv
9	NOP	XCHG eAX,eCX	XCHG eAX,eDX	XCHG eAX,eBX	XCHG eAX,eSP	XCHG eAX,eBP	XCHG eAX,eSI	XCHG eAX,eDI	CWDE CBW	CWQ CDQ	CALL FAR sm	FWAIT	PUSHF	POPF	SAHF	LAHF
A	MOV AL,[iv]	MOV eAX,[iv]	MOV [iv],AL	MOV [iv],eAX	MOVSB	MOVSD	CMPSB	CMPSD	TEST AL,rb	TEST eAX,iv	STOSB	STOSD	LODSB	LODSD	SCASB	SCASD
B	MOV AL,ib	MOV CL,ib	MOV DL,ib	MOV BL,ib	MOV AH,ib	MOV CH,ib	MOV DH,ib	MOV BH,ib	MOV eAX,iv	MOV eCX,iv	MOV eDX,iv	MOV eBX,iv	MOV eSP,iv	MOV eBP,iv	MOV eSI,iv	MOV eDI,iv
C	Shift rmb,ib	Shift rmv,ib	RET i2	RET	LES rm,rmp	LDS rm,rmp	MOV rmb,ib	MOV rmv,iv	ENTER i2,ib	LEAVE	RETF i2	RETF	INT 3	INT ib	INTO	IRET
D	Shift rmb,1	Shift rmv,1	Shift rmb,CL	Shift rmv,CL	AAM	AAD		XLATB	87 space	87 space	87 space	87 space	87 space	87 space	87 space	87 space
E	LOOPNE short	LOOP short	LOOP short	IeCXZ short	IN AL,[ib]	IN eAX,[ib]	OUT [ib],AL	OUT [ib],eAX	CALL iv	JMP iv	JMP FAR sm	JMP ib	IN AL,[DX]	IN eAX,[DX]	OUT [DX],AL	OUT [DX],eAX
F	LOCK		REPNE	REPE	HLT	CMC	Unary rmb	Unary rmv	CLC	STC	CLI	STI	CLD	STD	IncDec rmb	Indir rmv

MOV (386->) – mod-reg-r/m

d og **w** i opkoden definerer lengde og retning på operasjonen

Table 23: REG Bit Encodings

reg	w=0	16 bit mode w=1	32 bit mode w=1
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

[ebx] [ebp] ;Uses DS by default.
 [ebp] [ebx] ;Uses SS by default.
 [ebp*1] [ebx] ;Uses DS by default.
 [ebx] [ebp*1] ;Uses DS by default.
 [ebp] [ebx*1] ;Uses SS by default.
 [ebx*1] [ebp] ;Uses SS by default.
 es: [ebx] [ebp*1] ;Uses ES.

opcode	addressing mode									
1	0	0	0	1	0	d	w	mod	reg	r/m

displacement											
x	x	x	x	x	x	x	x	x	x	x	x

note: displacement may be zero, one, or two bytes long.

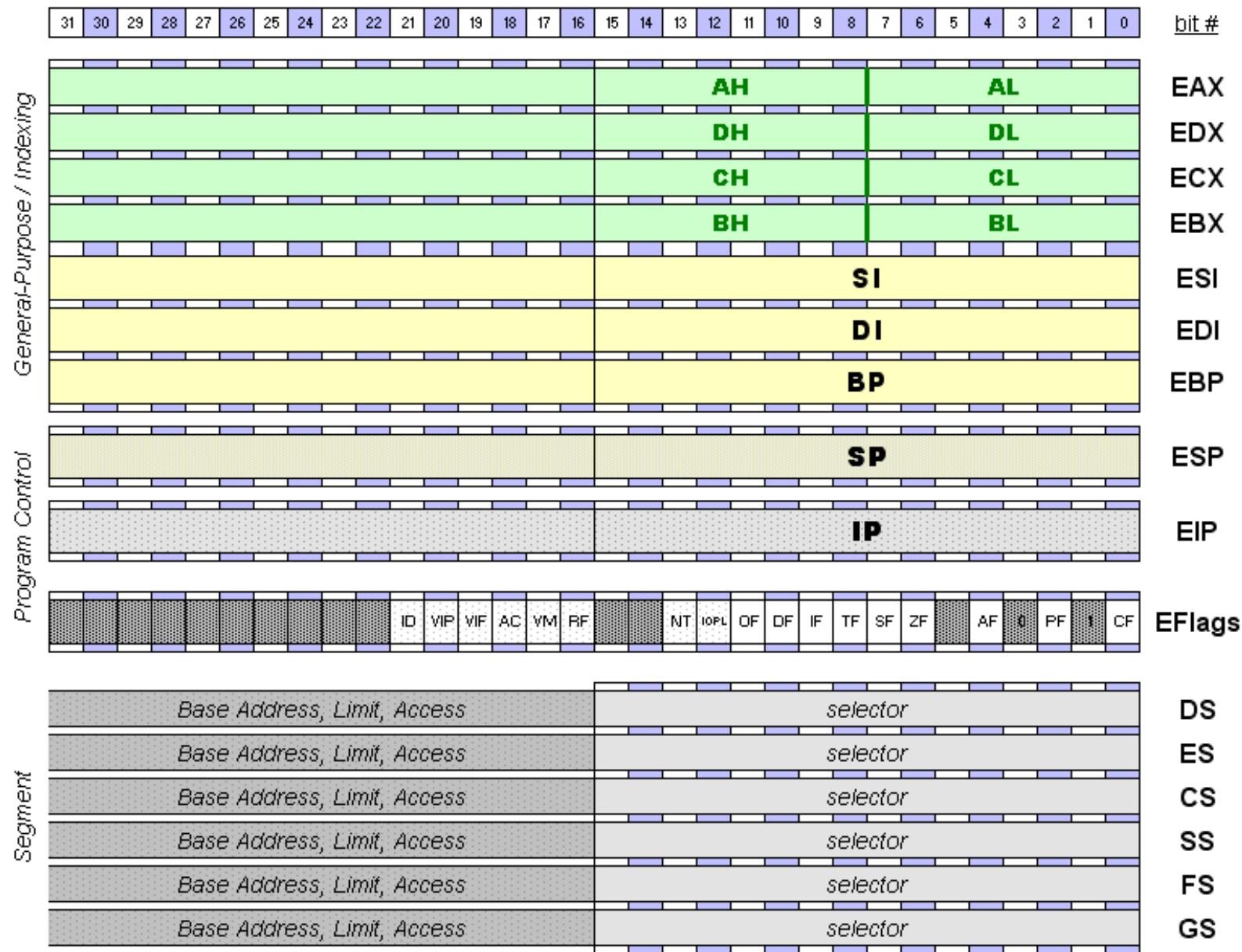
Table 24: MOD Encoding

MOD	Meaning
00	The r/m field denotes a register indirect memory addressing mode or a base/indexed addressing mode (see the encodings for r/m) <i>unless</i> the r/m field contains 110. If MOD=00 and r/m=110 the mod and r/m fields denote displacement-only (direct) addressing.
01	The r/m field denotes an indexed or base/indexed/displacement addressing mode. There is an eight bit signed displacement following the mod/reg/rm byte.
10	The r/m field denotes an indexed or base/indexed/displacement addressing mode. There is a 16 bit signed displacement (in 16 bit mode) or a 32 bit signed displacement (in 32 bit mode) following the mod/reg/rm byte .
11	The r/m field denotes a register and uses the same encoding as the <i>reg</i> field

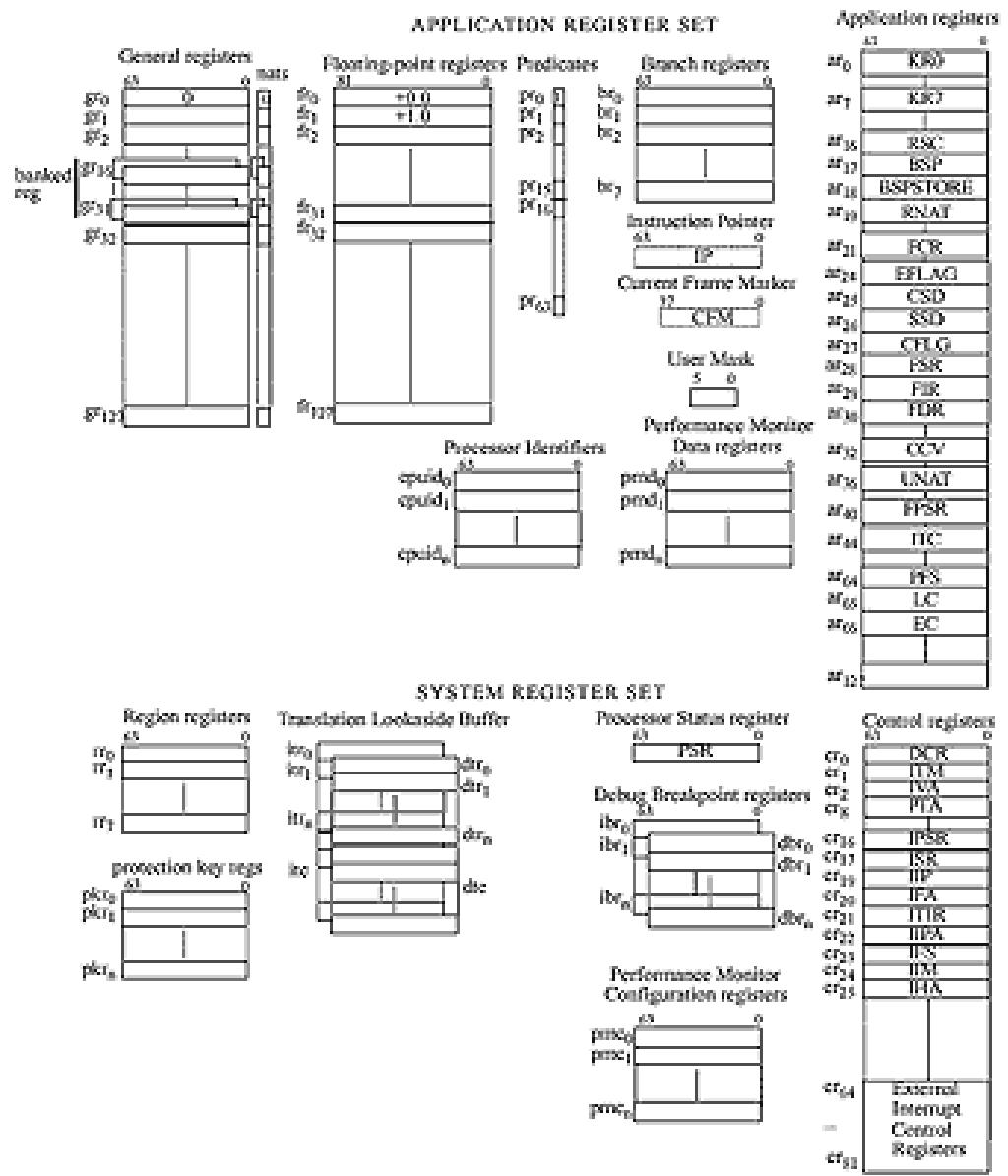
Table 25: R/M Field Encoding

R/M	Addressing mode (Assuming MOD=00, 01, or 10)
000	[BX+SI] or DISP[BX][SI] (depends on MOD)
001	[BX+DI] or DISP[BX+DI] (depends on MOD)
010	[BP+SI] or DISP[BP+SI] (depends on MOD)
011	[BP+DI] or DISP[BP+DI] (depends on MOD)
100	[SI] or DISP[SI] (depends on MOD)
101	[DI] or DISP[DI] (depends on MOD)
110	Displacement-only or DISP[BP] (depends on MOD)
111	[BX] or DISP[BX] (depends on MOD)

386 register (IA32) - brukerprogram



X86-64 registre



- Svært mange ulike registre
 - på hver kjerne!
- Ekstrem kompleksitet
- Hvordan forholder vi oss til kompleksiteten?
 - Abstraherer!

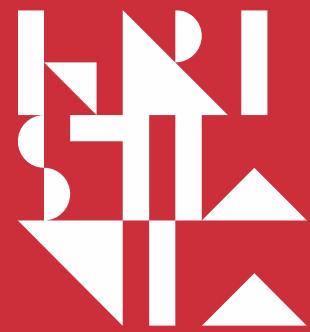
Abstraksjon

- et begrep eller en ide som ikke er forbundet med noen konkret instans
- å fjerne det unødvendige i en gitt sammenheng
- å fokusere på de trekkene som er felles
- Å «gjemme» kompleksitet
- Eksempel:
 - I et Java-program så kan du gjemme et komplisert program inne i en metode...
 - etter det trenger du bare å forholde deg til metoden
 - Høynivå-språk abstraherer bort all den maskinnære kompleksiteten
 - `System.out.print("Hallo World!")` involverer i realiteten noen tusen instruksjoner siden er et metode-kall fra `java.exe` til operativsystemets skrive-til-skjerm-rutiner.

Denne forelesningsøkten vil bli tatt opp og lagt ut i emnet i etterkant.

Hvis du ikke vil være med på opptaket:

 Start Video	La være å delta med webkameraet ditt.
 Unmute	La være å delta med mikrofonen din.
To: Marianne Sundby (Privately) Type message here...	Still spørsmål i Chat i stedet for som lyd. Hvis du ønsker kan spørsmålet også sendes privat til foreleser.



Høyskolen
Kristiania

TK1100 Digital Teknologi

Fjerde forelesning

Organisering

- Generelt
 - Digital vs analog, def computer
- Datatyper
 - Tall, text, bilder, lyde
- CPU
 - Oppbygging
 - Registre
 - Instruksjoner

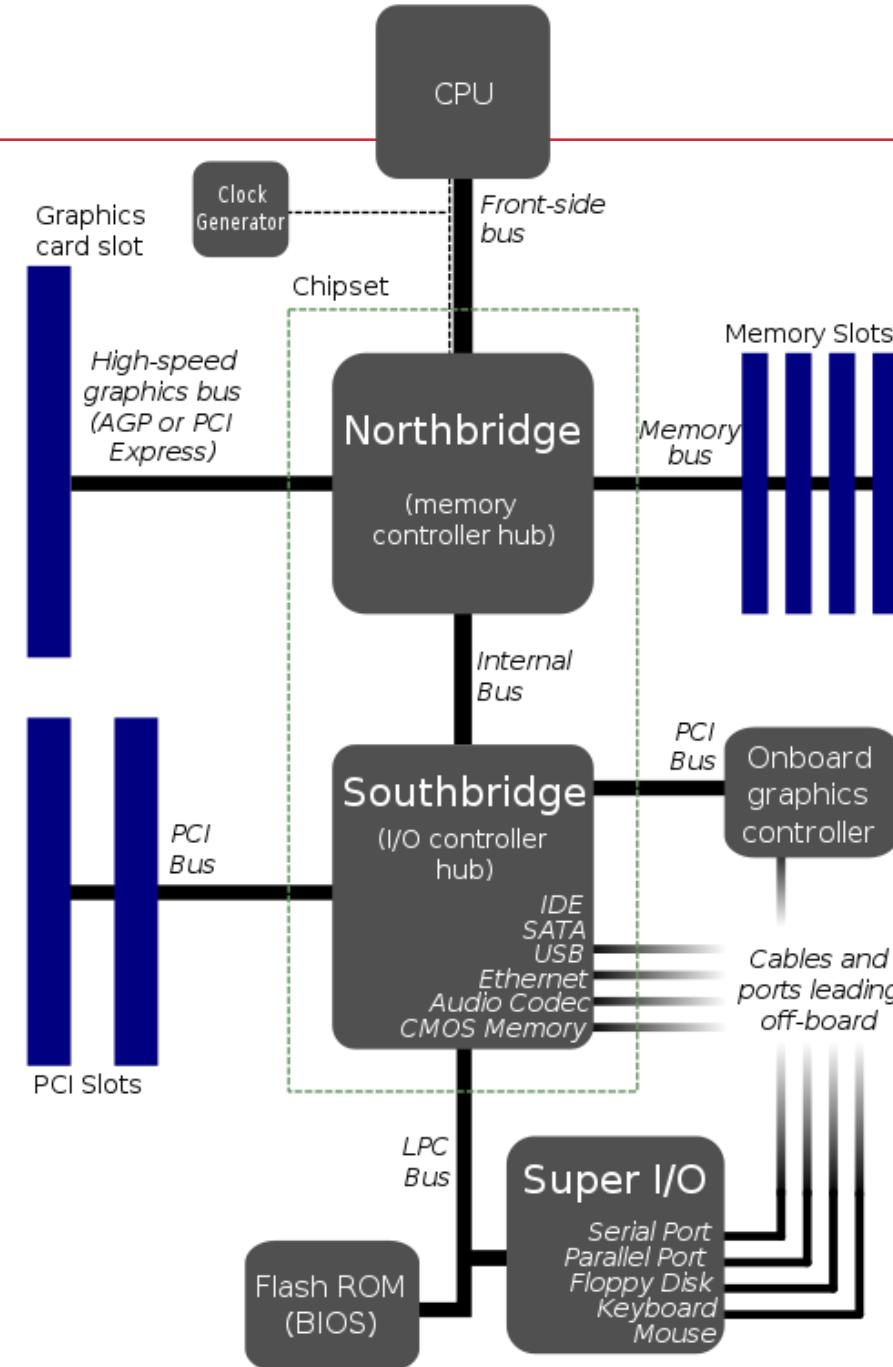
- Oppbygging av (IBM) PC systemer
 - Ikke de eneste, jf forrige forelesning
 - Ser ikke på ALT
 - Forhåpentligvis noe av det viktigste
- Viktigst
 - Forstå sammenheng
 - Ikke alle komponenter i detalj,
 - men hvilke type de er.
 - Hvordan de henger sammen med resten

Hovedkortet

- Grunnleggende lik **IBM PC** fra 1982
 - Mac pleide å ha en egen vei, men måtte endre seg da de gikk over til Intel
- Noen PC-systemer booter fremdeles opp i 16 bit modus
 - max 1 MiB RAM
 - begrenset instruksjonssett og minnemodell
 - kun BIOS-rutiner/interrupt
- Går over i 32/64 bit først når OS lastes inn.
- Flere og flere systemer er nå UEFI; men bakover kompatibilitet er ivaretatt

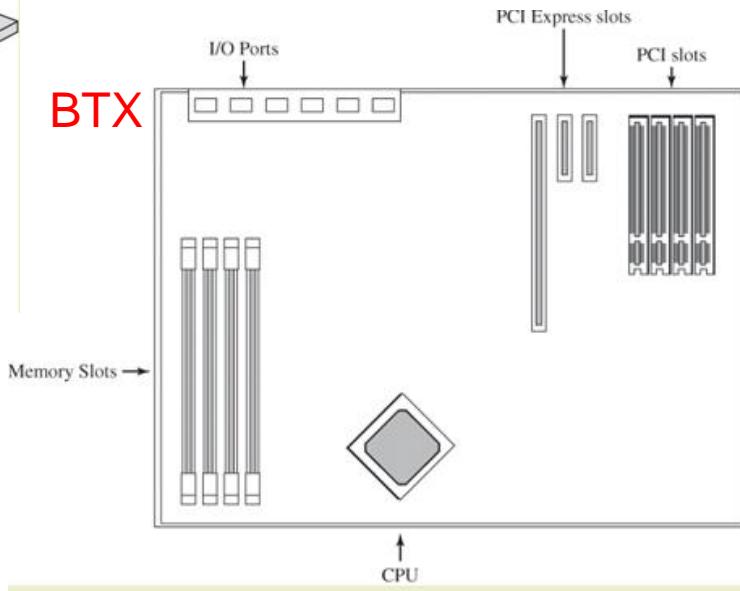
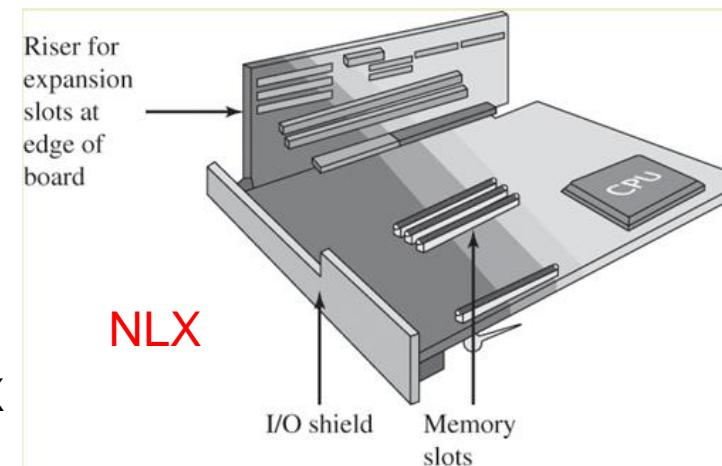
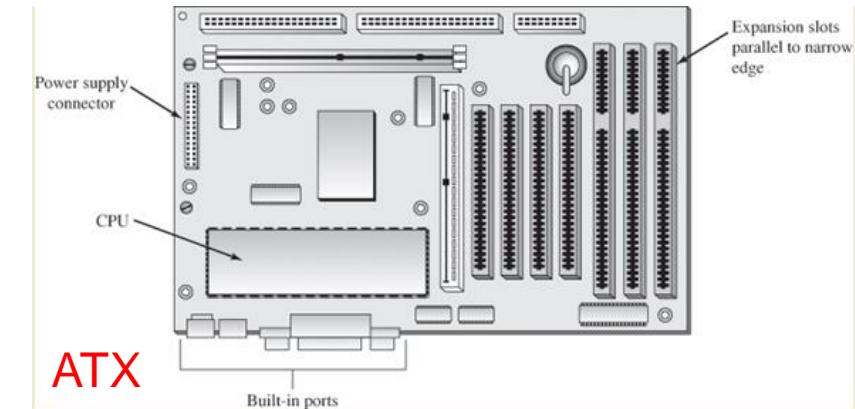
Hovedkort

- Kopler sammen
 - CPU
 - Chipset
 - Busser
 - Minnespor og RAM
 - Expansjons-spor og ekstrakort
 - Porter og «støpsler»



Formfaktor

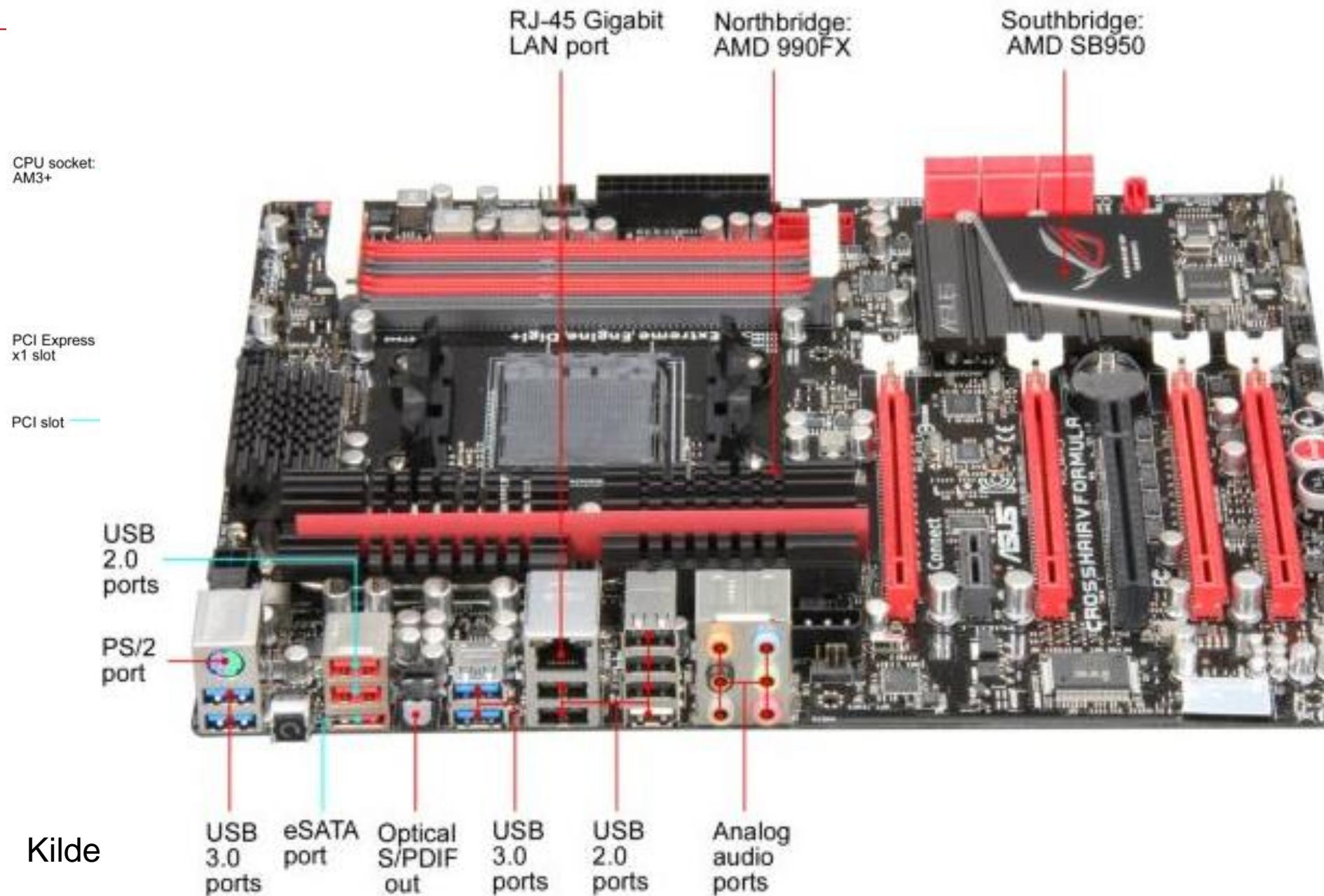
- Formfaktoren definerer fysisk størrelse, utforming og til en viss grad tilgjengelig funksjonalitet på hovedkortet
- ATX (12"x9.6")
 - Utvidelsesspor på bredkanten
 - Innebygde IO porter på en side
 - CPU orientert så den kan kjøles av strømforsynings-viften
 - CPU og RAM vinkelrett på utvidelsesspor
 - Tilleggs plass til ekstra kjøling
 - Micro ATX(9.6"x9.6")
 - Mindre maskiner
- NLX
 - Stående ekspansjon
- BTX
 - Skal gi bedre kjølingsmuligheter enn ATX
 - Tre ulike størrelser
 - Full (326 x 266 mm)
 - microBTX (264 x 266 mm)
 - picoBTX (203 x 266 mm)



Integrert og Ikke-

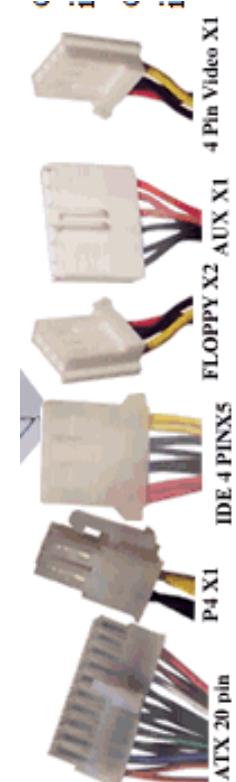
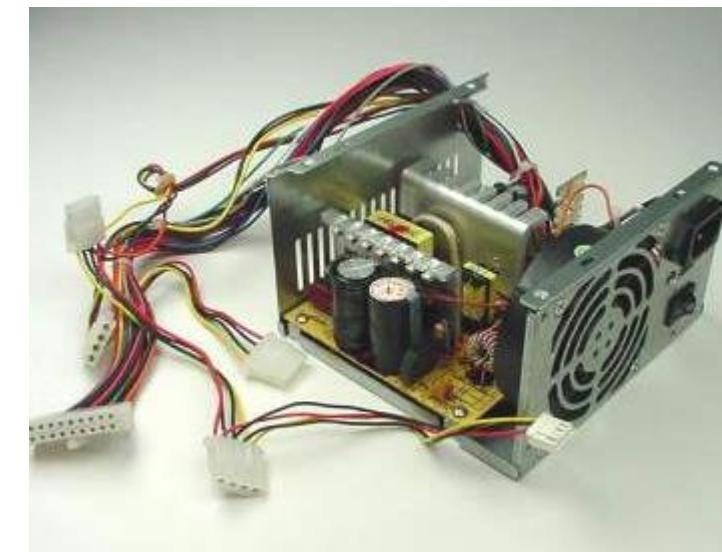
- Mange hovedkort er **integrerte** og kommer med CPU, skjermkort inkl GPU, nettverkskort, busskontrollere og alt ferdig innebygd i selve kortet
 - Typisk for f.eks. billige laptop'er m.fl.
- Ikke-integrerte hovedkort
 - Monterer tillegg som ekspansjonskort
 - Disk-kontroller, skjermkort, nettverkskort,...
- Stadig mer uklar distinksjon i våre dager...

Hovedkort ca 2011



Strømforsyning

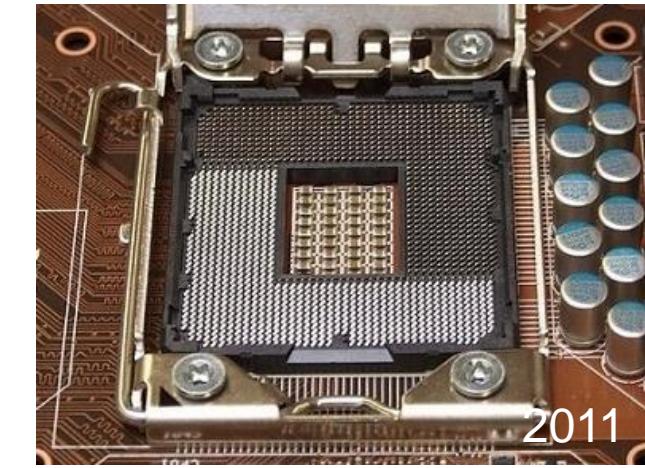
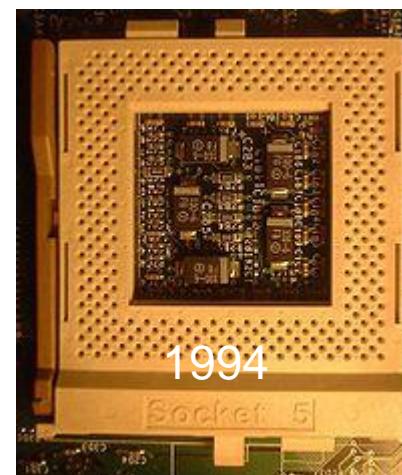
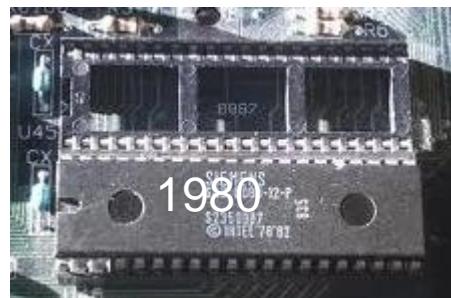
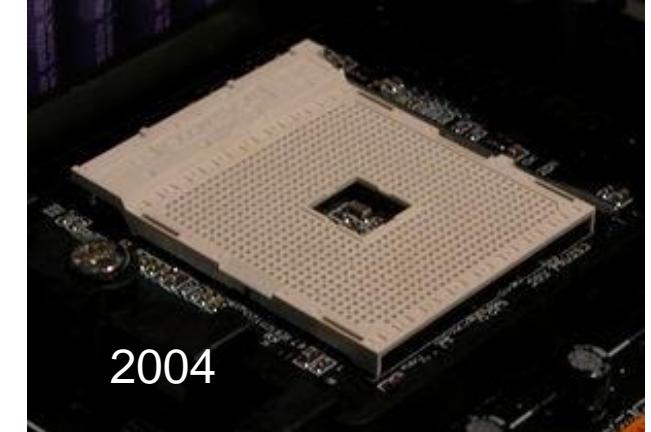
- Transformerer 220 V, 50Hz vekselstrøm (AC) til de spenningene likestrøm (DC) komponentene på hovedkortet trenger
 - 3,3 V
 - +/- 5 V
 - +/- 12V
- Typen strømforsyning du kan bruke avhenger bl.a. av formfaktor på hovedkortet



CPU

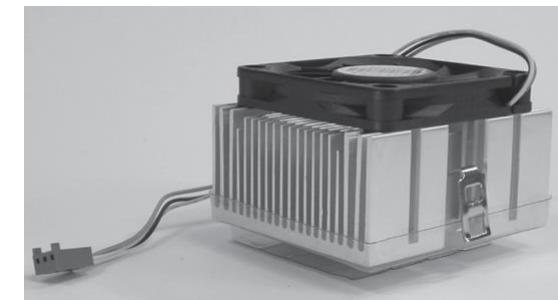
CPU Socket

- CPU tilkoples hovedkortet gjennom en **socket** (eller i et expansionspor)
- Ulike typer med ulike
 - antall plasser til CPU ben («pins»)
 - Hastighet på FSB (Front Side Bus)
- Avhenger av hovedkort



Kjøling av CPU

- Effekt ($W=J/s$) er energi pr tidsenhet
 - viser seg som varmeutvikling
 - $E=UI$, $U=RI$ (Ohms lov), $E=R|I|^2$
 - \Rightarrow Varmen stiger med kvadratet av strømmen
- Luftkjøling
 - Bruker vifte og/eller kjøleribber
- Væskekjøling
 - Pumper bort varmen med væske
 - Typisk vann
(eller flytende nitrogen)



Feilsøking: CPU

- Typiske problemer er kjøling eller strøm
- Symptomer
 - PC booter ikke, eller laster ikke OS
 - PC krasjer med enkelte applikasjoner
 - Plutselige POST-feil for mye forskjellig utstyr
- Varme-problem (typisk)
 - PC booter, men stopper/fryser/rebooter etter noen få minutter
 - Tiltak: bedre kjøling

RAM



- **BIT**
 - **Binary Digit**
 - På/av krets
 - 1 eller 0
- **BYTE**
 - 8 BIT
 - Kan lagre f.eks. en alfanumerisk karakter
 - Minste adresserbare enhet i RAM
- **ORD**
 - I computerarkitektur har dette flere betydninger
 - **Størrelsen på et register**
 - Det antall BIT CPUen kan prosessere som en enhet
 - Adresseringseenhet
 - BYTE, WORD (16), DWORD (32) og QWORD (64) er MicrosoftSpeak – ikke alle bruker disse betegnelsene

Typer Minne problemer

- Konfigurering
 - Mer minne enn PC, eller OS støtter
 - BIOS CMOS feil
- Hardware
 - Defekte brikker eller moduler
 - Ikke-kompatible moduler
- Installasjon
 - Ikke satt modulen riktig inn i sporet
 - Spor («socket») defekt, eller bør renses

Feilsøking av minne

- Andre komponenter kan forårsake problemer som ser ut som minneproblemer!
- Minne problemer viser seg typisk:
 - Første booting
 - Rett etter installering av
 - ny RAM-modul
 - ny software eller nytt OS
 - Etter ny hardware har blitt lagt til, eller gammel fjernet
 - Helt uten (åpenbar årsak)
 - Korrosjon («rust»)
 - Varme-skader

Vanlige RAM problemer

- PC booter ikke, POST beep kode
 - Minne satt inn, eller konfigurert, feil
- PC booter til blank skjerm
 - Minne-modul løsnet
 - Minne-modul av feil type
- POST minne-oppstelling feil
 - Feil RAM-modul type
- «Memory Error» melding
 - Kompabilitetsproblemer med gammelt og nyinnsett RAM
 - RAM-modul i gang med å gå i stykker
- ...

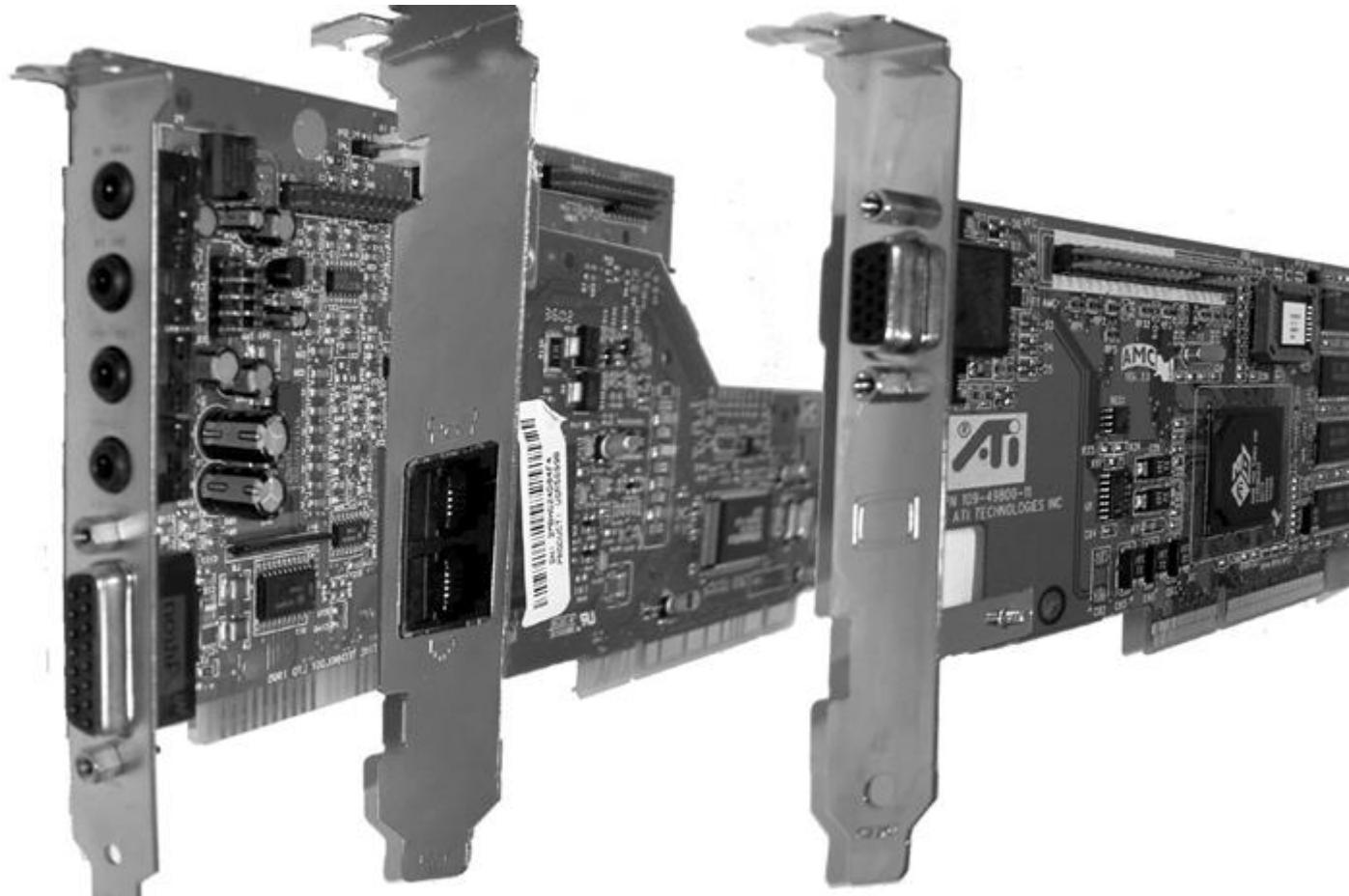
Perferi- busser & utstyr

- Nesten uendelig utvalg av I/O-enheter
- Vanlige konsepter
 - Port
 - Buss (*daisy chain* eller *delt direkte aksess*)
- Kontroller (*controller* eller *host adapter*)
- I/O-instruksjoner kontrollerer enheter
- Enheter har adresser, som brukes av
 - **Direkte** I/O-instruksjoner
 - **Minne-avbildet** I/O (*memory-mapped I/O*)
- Samarbeidet med OS foregår oftest gjennom en **driver** som oversetter OS-instruksjoner til Kontroller-instruksjoner

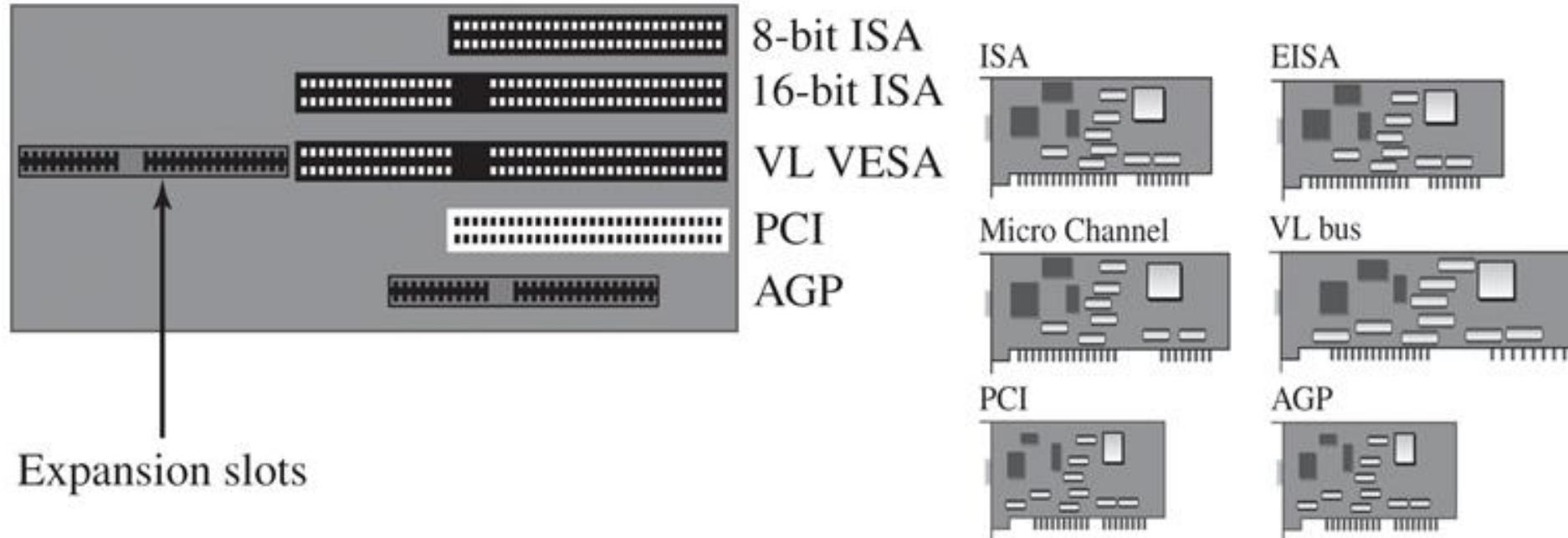
Ekspansjonsbusser

- Lar oss legg til ny funksjonalitet i form av kort og periferiutstyr
- Forbundet med spor på hovedkortet der vi monterer ekspansjonskortet
- Ulike teknologier
 - PCI, AGP, ISA, PCIe
- Utfører kommunikasjonen mellom kortkontrolleren og chipset'et

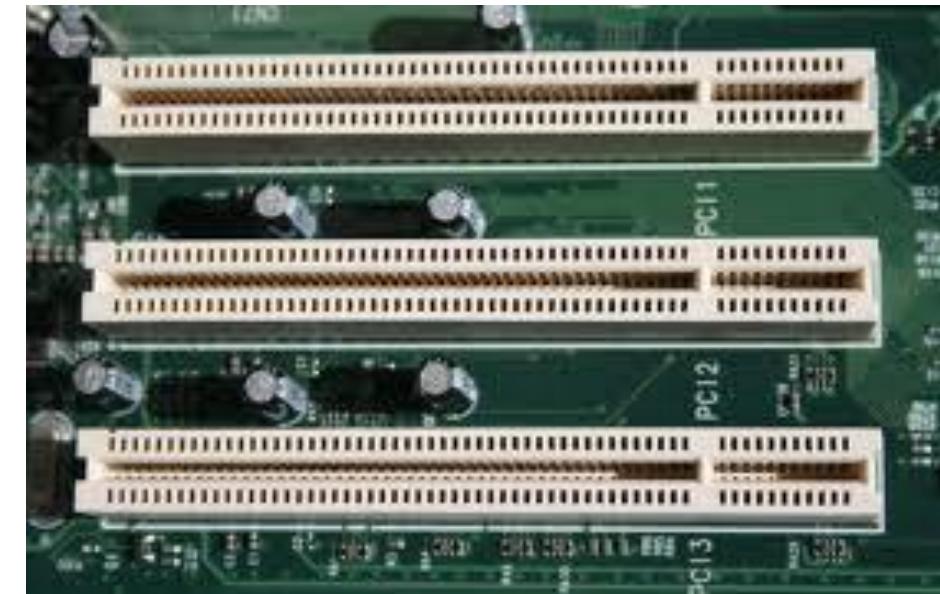
Ekspansjonskort



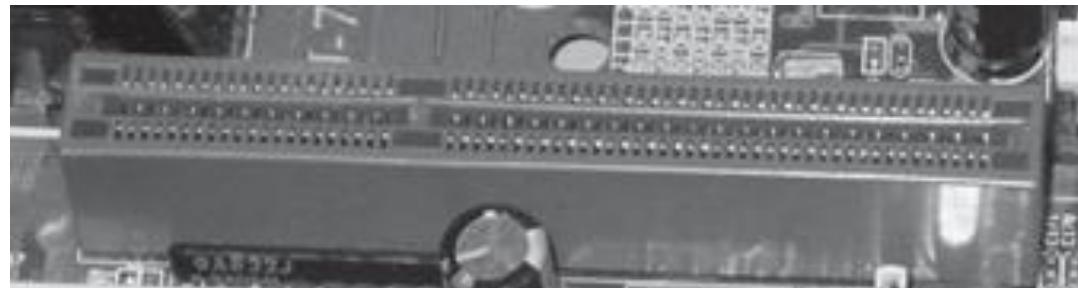
Gamle ekspansjonskort og -spor



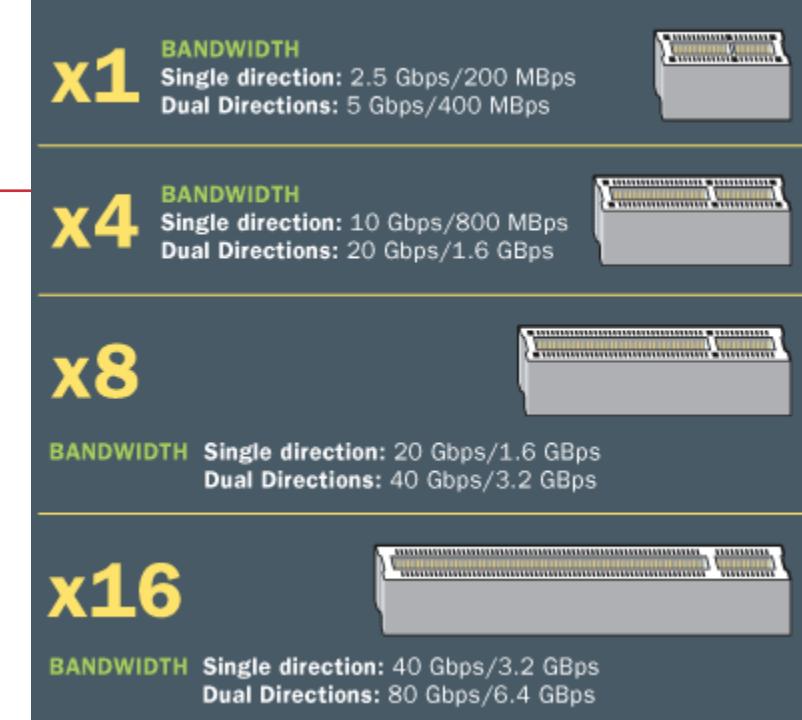
- 32 og 64 bit data
- Bitrate opp til 66 MHz, 533 MB/s (64 bit)
- CPU uavhengig
- Tilbakekompatibel med ISA, EISA
- PnP støtte



- Laget for å støtte 3D grafikk
- Koplet direkte til Northbridge
- 32 bit, 66 MHz
- Bitrate fra 266 MBps til 2133 MBps avhengig av versjon
- Versjonene 2X, 4X, 8X overførte data 2x, 4x og 8x pr klokkesyklus



- Høyhastighets **seriell** bussarkitektur
- Støtter flere kanaler
 - Ulike spor avh kanalantall
- Oppbygging
 - Punkt-til-punkt
 - Pakkebasert og routed a la nettverk
 - Adresser og data ligger i samme pakke
 - Høy bitrate
 - Hver kanal støtter duplex, 2,5 Gbps hver vei
 - **STØTTE FOR FLERE GRAFIKKORT**



Systemressurser

- Bussene benytter en eller flere typer systemressurser
 - I/O port-adresser
 - IRQer
 - DMA kanaler
 - Minne-adresser
- Ressurskonflikter kan oppstå når to ulike forsøker å benytte samme ressurs
- Plug-and-Play (PnP): Konfigurasjonstandard som automatisk tilordner systemressurser til ekspansjonskort og utstyr

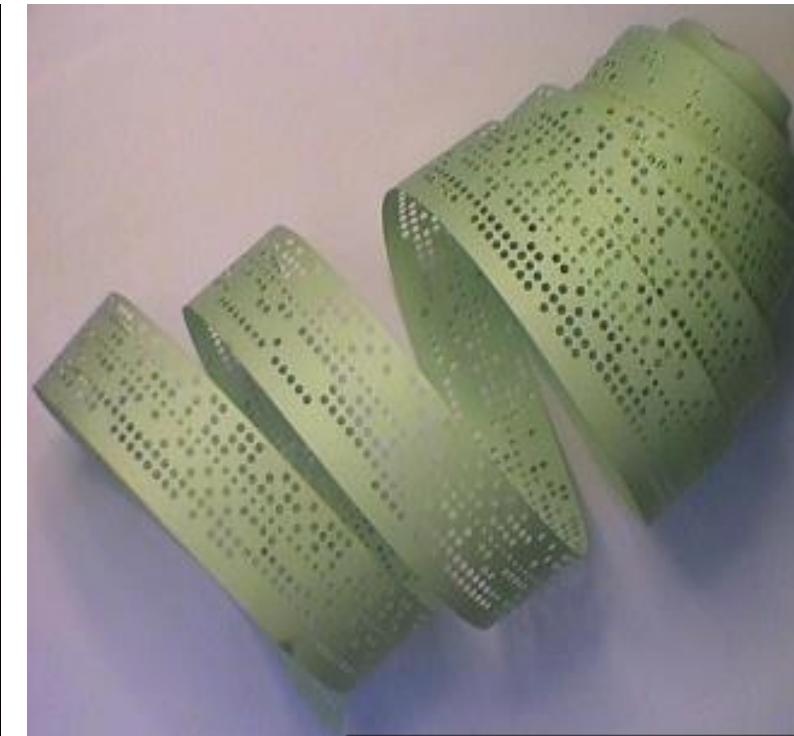
MASSELAGER

(Harddisk)

Masselager

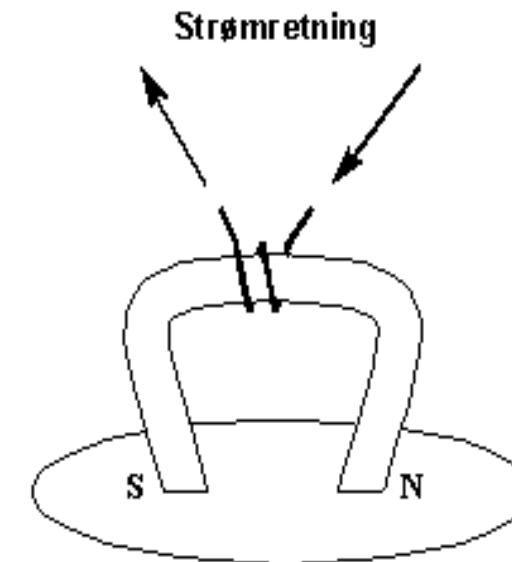
- Brukes for permanent lagring av store datamengder
 - Hullkort
 - Hullbånd
 - Diskett, Flash Mem
 - Zip, Jazz
 - Disk
 - CD, DVD
 - Magnetbånd(spolebånd, kassett)
- Overføring til prosessoren via en kontroller
 - IDE/EIDE, SATA, SCSI, ...

Papir som lagringsmedium



Magnetisk lagring/lesing

- Lagring
 - Når elektromagneten tilføres strøm magnetiseres overflaten under magneten
- Lesing
 - Når overflaten under magneten er magnetisert og passerer magneten \ det dannes(induseres) strøm
- Magnetisert område = 1
- Ikke magnetisert = 0



Organisering av data

- Data er i utgangspunktet en samling av 0-ere og 1-ere som i sin sammenheng representerer et dokument, bilde, musikk etc.
- Denne samlingen kalles en **fil**
- En fil har et **navn** som gjør at den er lett å identifisere
- I et identifikasjons-felt som legges sammen med filen finnes det foruten navnet også informasjon som størrelse på filen, når den ble opprettet, eier av filen,
- Jf Øving om BMP

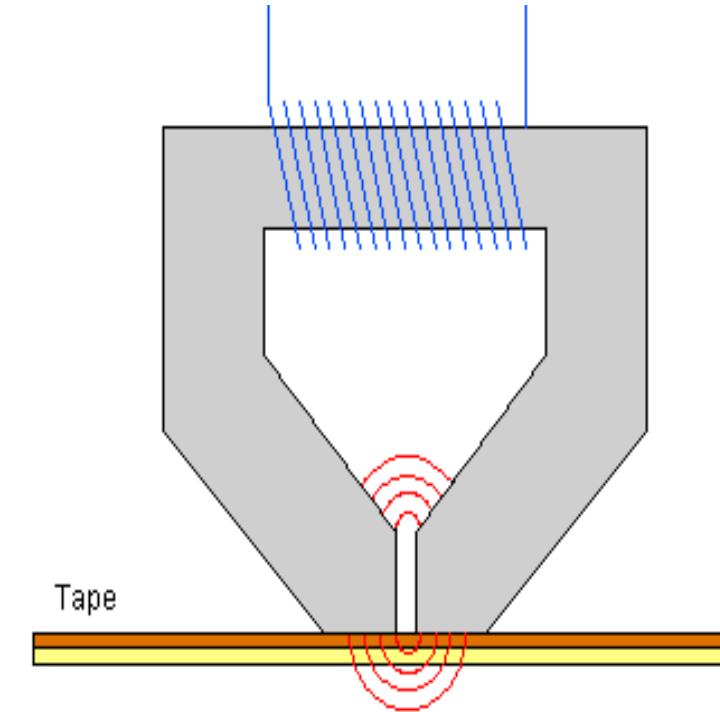
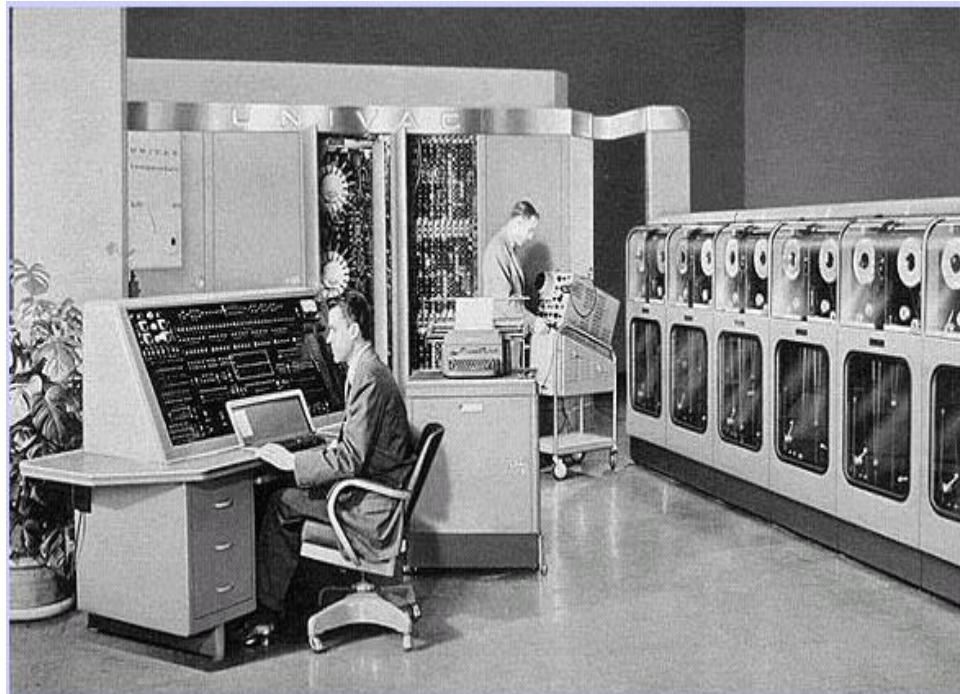
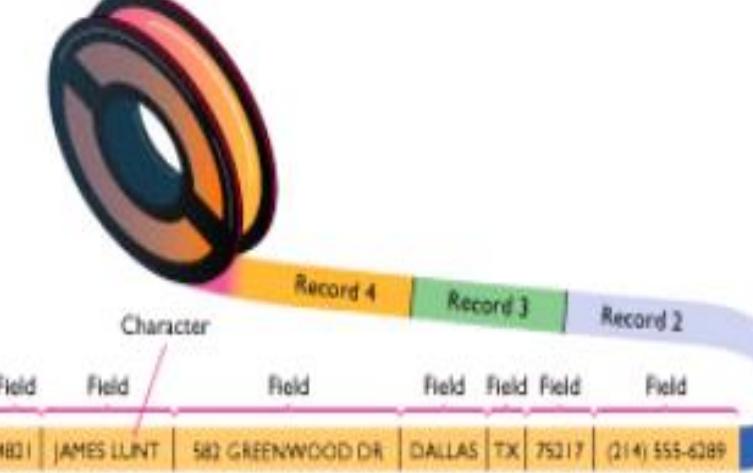
Disk



Disker

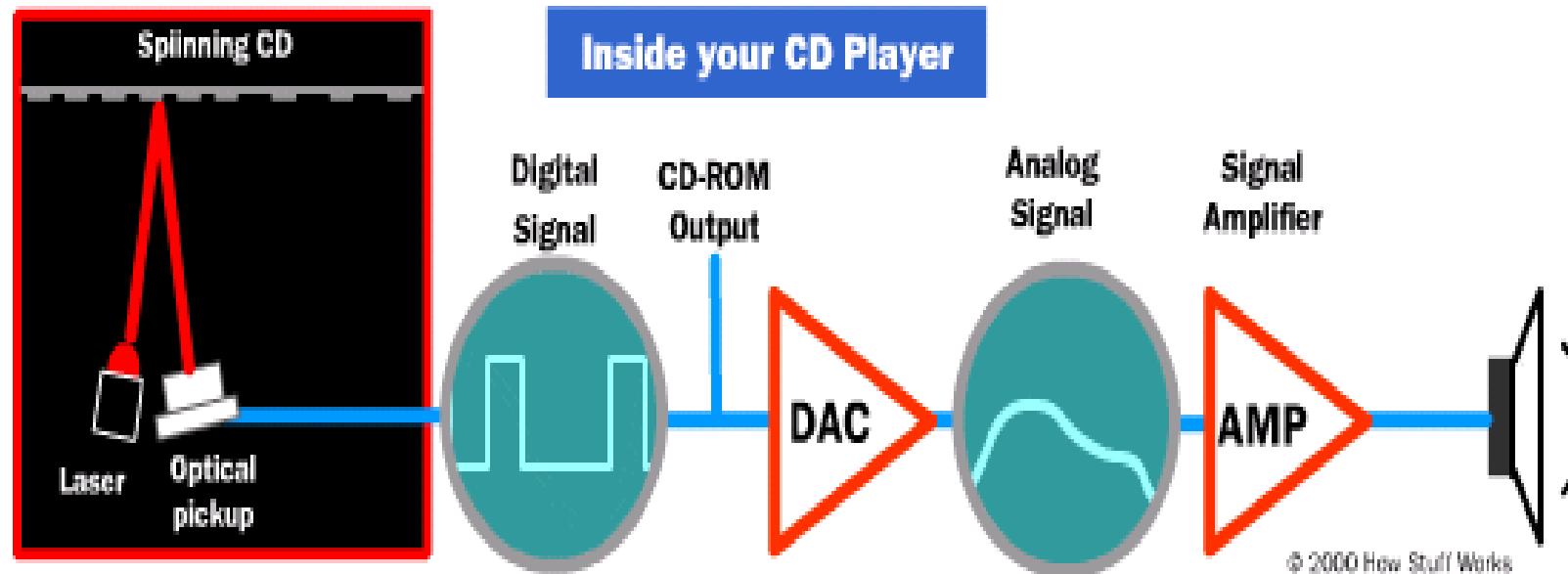
- Disker ...
 - Gir varig lagring
 - er *billigere* pr byte enn RAM (forløpig)
 - 😊 har *større kapasitet*
 - 😢 er mange størrelsesordener *tregere*
- Viktige parametere er
 - lagringsplass
 - I/O båndbredde
- Fordi...
 - ...det er en **kolossal** hastighetsforskjell (ms vs ns $\sim 10^6$) med RAM
 - ...disk I/O er svært ofte **viktigste** ytelses-flaskehals

Magnetbånd



CD/DVD/Blue-Ray

En CD kan lagre 700 MB, DVD 4 GB på hver side, Blue-Ray 25GB pr lag
(opp til 8 lag i RW)



Animasjon

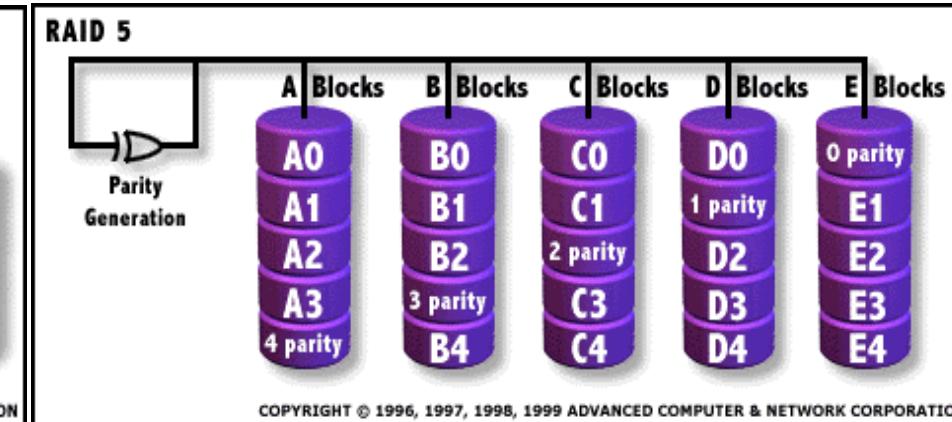
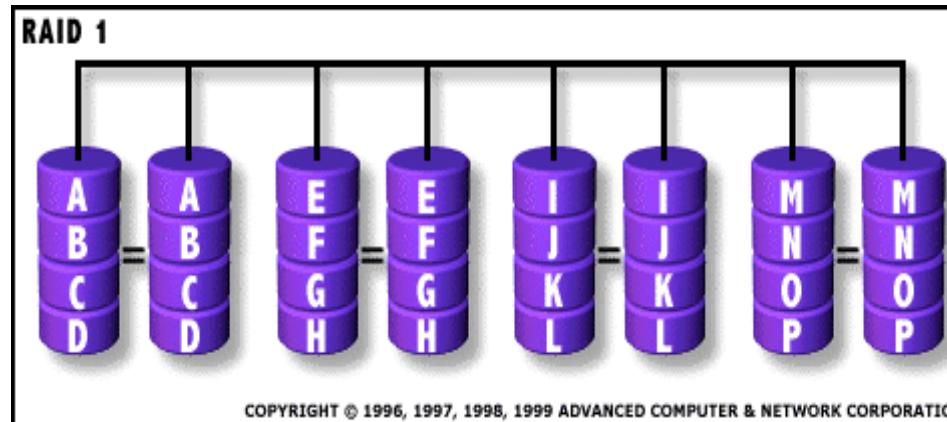
Sikkerhetskopi (backup)

- Alle typer lagringsmedia kan gå i stykker
- Viktige data bør derfor lagres på mer enn et sted
- Det er for sent å tenke på å ta backup **etter** et krasj
- Hjemmebrenning er blitt mer populært i det siste!
- Husk den legale/etiske siden av data-kopiering!!



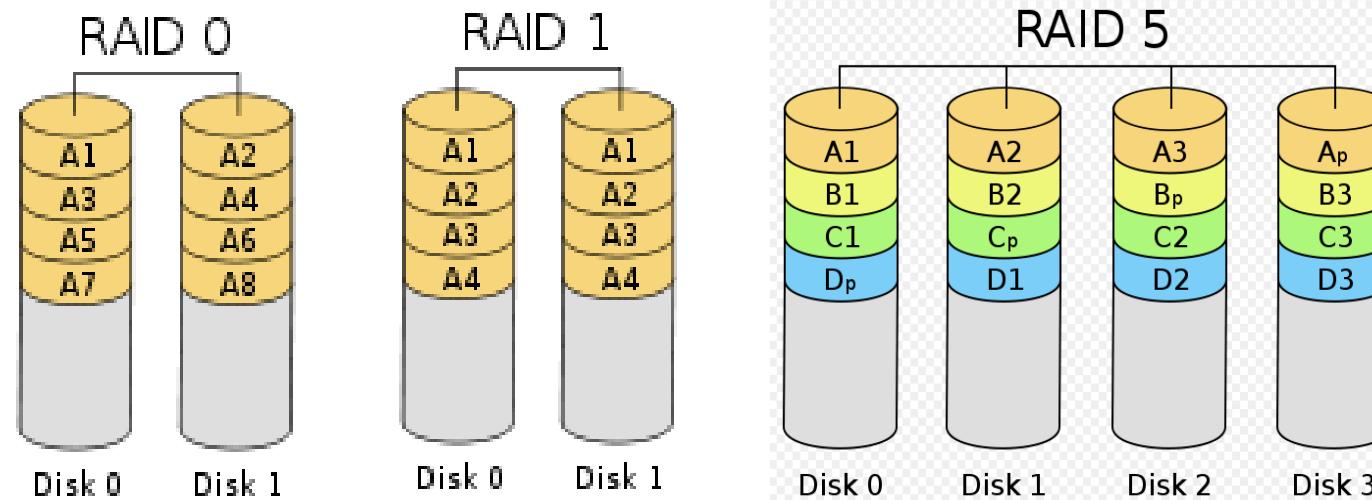
Beskyttelse mot feil og krasj (RAID)

- Store disk-banker kan la hver disk stå på egne ben, RAID0
 - eller generere fortløpende kopi av alle disker, RAID1
 - eller anvende andre sikringsteknikker (f. eks. striping)
- RAID5 lager en sjekksum (XOR) for to disker og lagrer denne på en tredje disk. Dette er godt nok hvis en disk faller ut.



RAID (Noen typer)

Betegnelse	Teknikk	
RAID 0	Striping	Raskere aksess
RAID 1	Speiling	Raskere aksess, større pålitelighet
RAID 5	Striping og distribuerte paritetsblokker	Raskere aksess og større pålitelighet



Typiske Harddisk problemer

- Tre vanlige årsaker
 - Feil på kontroller/adapter
 - Feil sammenkopling av adapter og disk
 - Feil på selve diskene
 - CMOS konfig feil
 - Ressurs konflikt
 - Korrumpt eller manglende Boot-partisjon
 - Virus i bootpartisjon
 - Defekt (mekaniske feil)

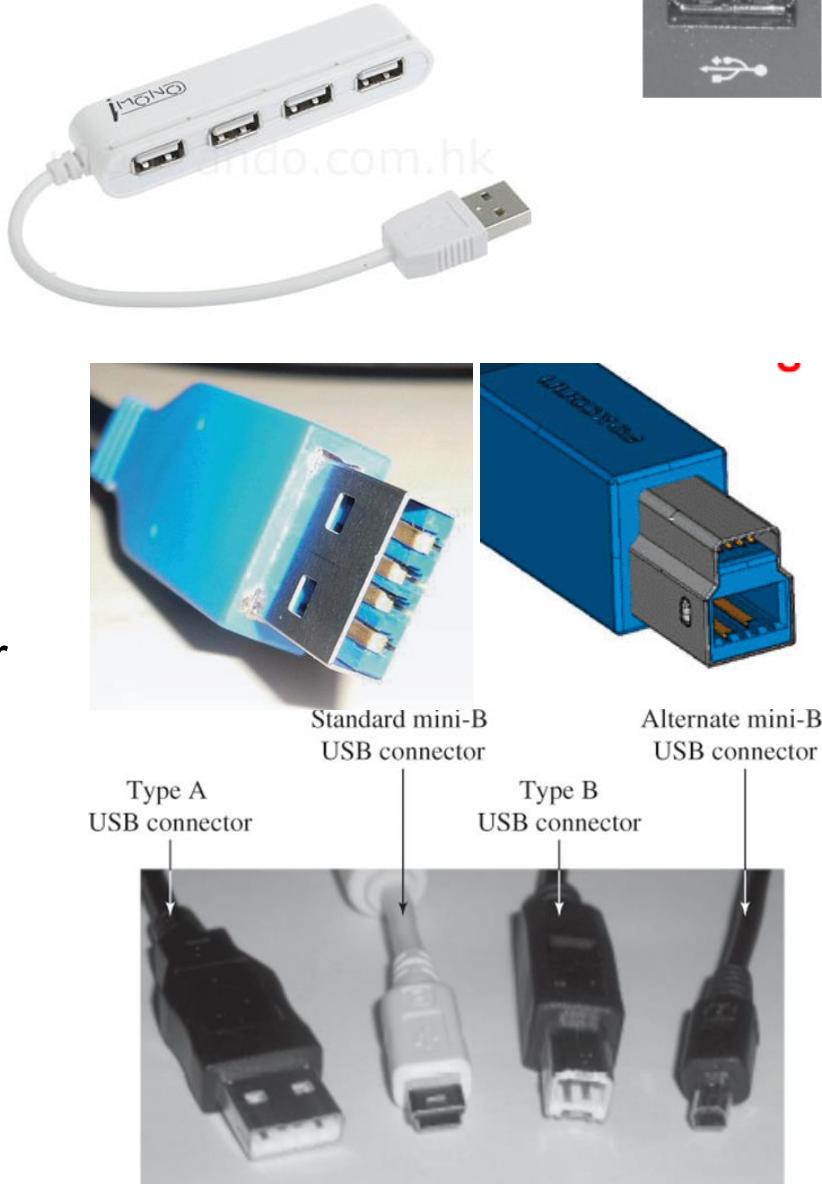
Typiske feilmeldinger

- Hard disk configuration error
 - Feil CMOS konfig parametre
 - Løs datakabel
- Hard disk 0 failure
 - Feil CMOS konfig
 - Dårlig forbindelse til strømforsyning
- Hard disk controller failure
 - Dårlig kabel forbindelse
 - Dårlig forbindelse til strømforsyning

USB og Flash-minne

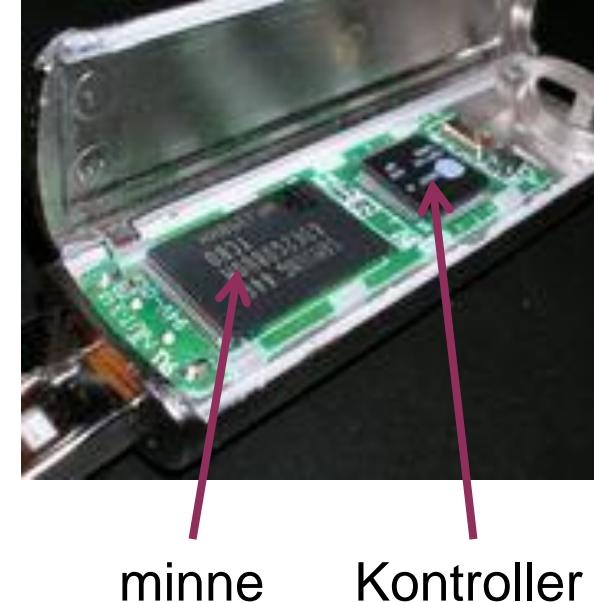
USB – Universal Serial Bus

- Støtter opp til 127 forskjellige tilkoplede enheter
 - Utvider med **hub**
- Høyhastighet dataoverføring
 - USB 1.1: 12 Mbps (max)
 - USB 2.0: 480 Mbps (max)
 - USB 3.0: 5 Gbps (max)
- Variable bitrate
 - Lavhastighetskanaler for tregt utstyr
- Fire hovedtyper dataoverføring
 - Bulk, interrupt, isokronisk, kontroll
- PnP og hotpluggbar
- Kan brukes til lading



Flash Memory

- To typer: "NOR" og "NAND"
- **NOR** (parallel-koplet)
 - Random Access lesing/skriving
 - Best til programmer
 - Mye brukt i mobiltelefoner
- **NAND** (seriekoplet)
 - Leses i blokker (som HD)
 - Billigere, flere transistorer på mindre areal
 - Raskere **å lese** fra enn å skrive til
 - Best til data
- Man kan (på begge typer) kun **slette blokker!**
- Holder til ca 100.000 skriv/slett
- 512MB -> 256 GB (2009)->1 TB



SSD: Solid State Drive

- (Oftest) samme lagringsteknologi som MinnePinne (NAND)
- 50-100 ggr raskere (kortere aksesstid) enn HardDisk pga
 - Ingen mekanikk, armer som må flyttes osv
- Begrenset levetid, akkurat som Flash Memory
- 10-20 ggr så dyrt pr GB
- Finnes opp til 4TB (?)

Periferiutstyr

Tastatur

- Typisk er 104-key IBM PC standard (qwerty)



Mus (optisk)

- LED eller LED-laser sender ut en stråle lys
- Lavoppløslig «kamera» (20x20 px ->) tar «bilder» ca hvert 1/10 sekund (ofte oftere)
- Chip måler/beregner forskjell mellom bilder og regner ut posisjonsendring



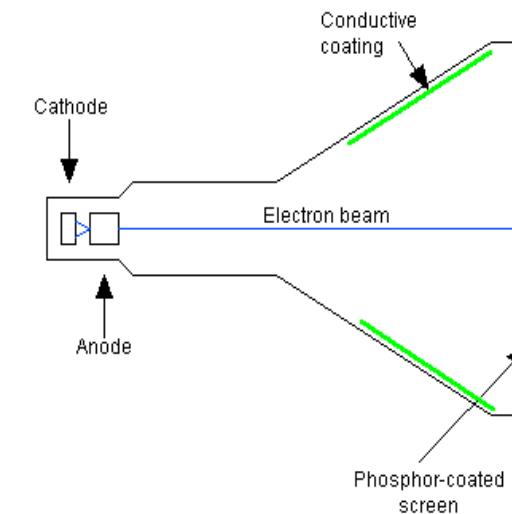
Skjerm

- To hovedtyper
- Katodestråle-rør (CRT – **C**athode **R**ay **T**ube)
 - Var vanligst lenge, nesten borte...
- Flat-panel
 - LCD – **L**iquid **C**rystal **D**isplay
 - Plasma-display, FED, LED, polymer-display, OLED

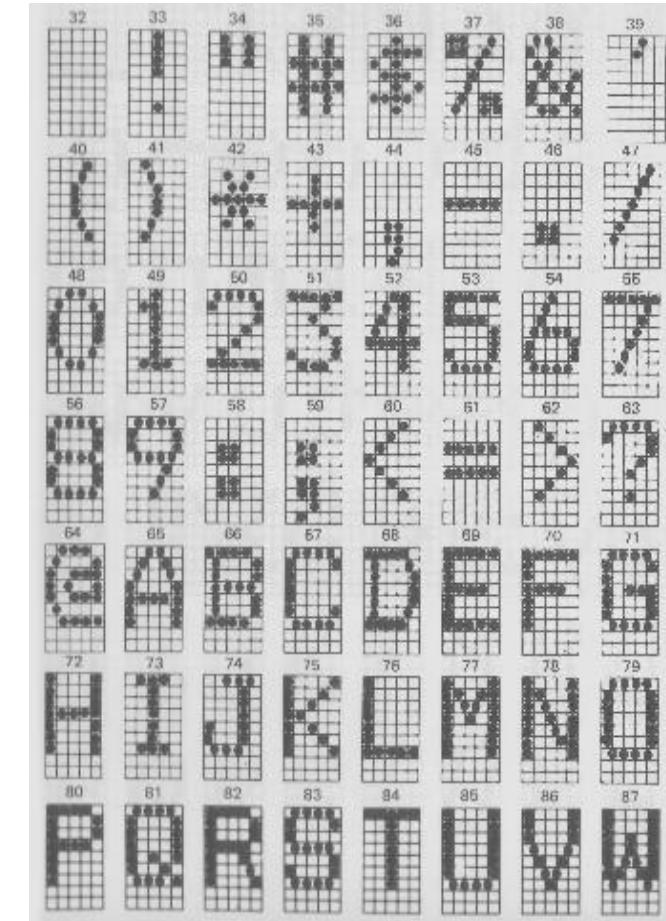
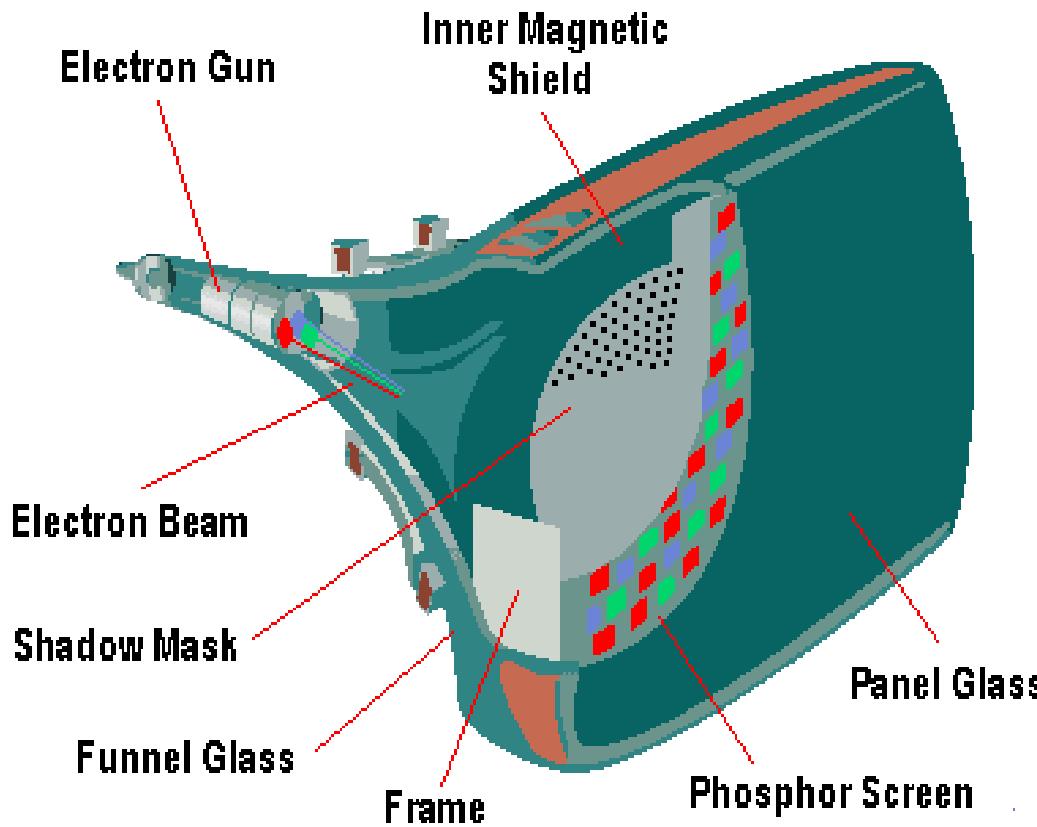


Skjerm

- Et bilde-element (pixel) består av 3 farger (rød, grønn og blå)
- Alle elementer på skjermen oppdateres flere ganger pr sekund
- **Skjermens størrelse oppgis som lengden av diagonalen**
- Styres av grafikk-kort med minne



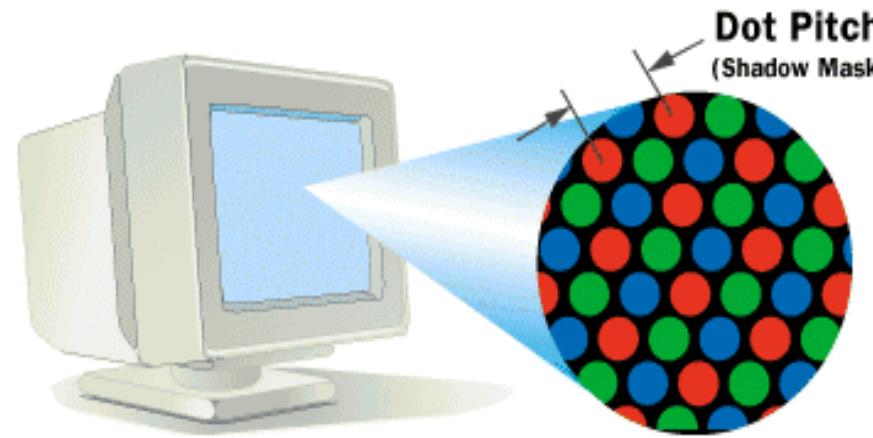
Skjerm



Skjerm-kort

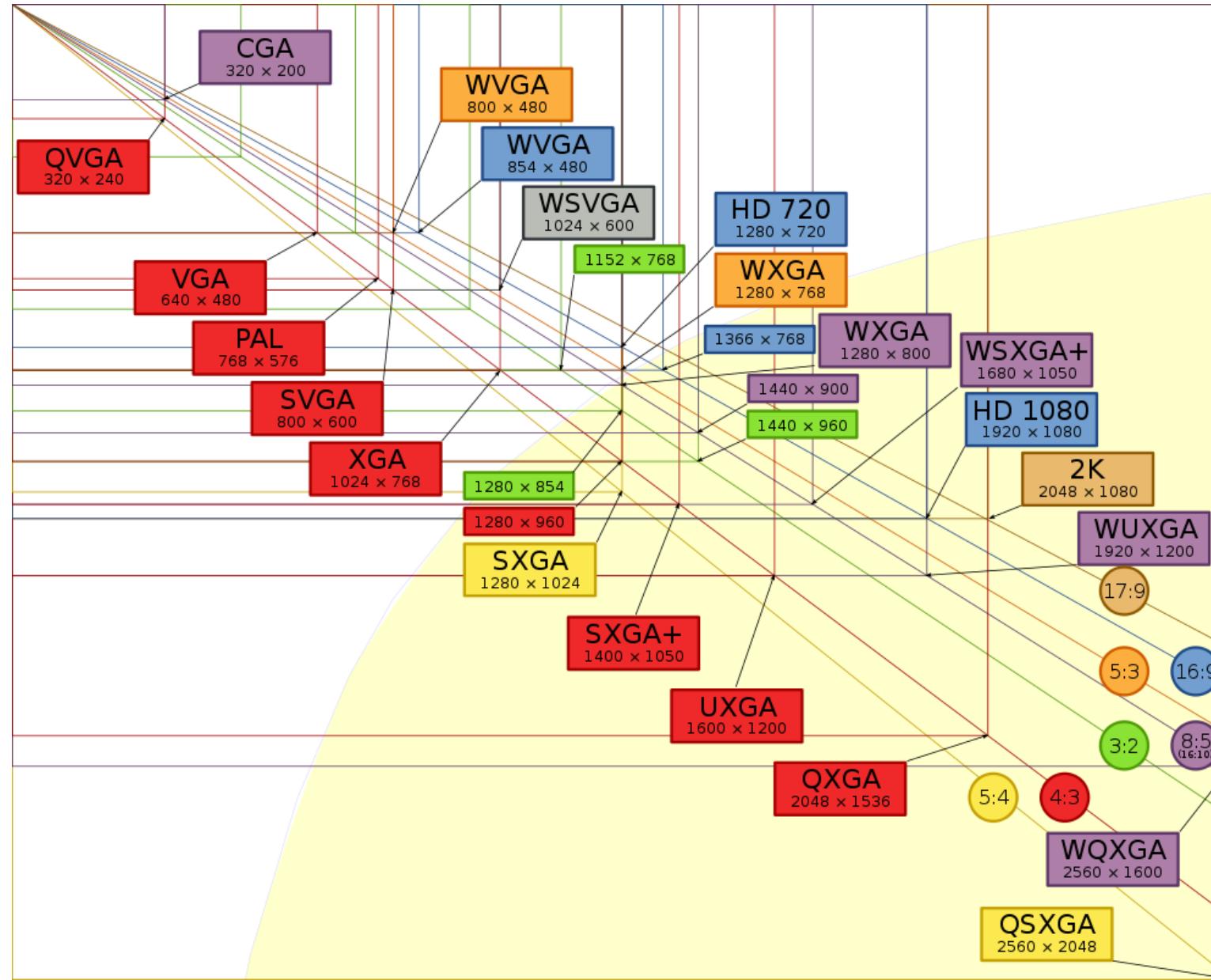
- Tolker og styrer det som skal ut på skjermen
- Egen hukommelse, bestemmer max oppløsning og fargedybde

Dot pitch; avstanden mellom to pixels (0,25 mm)



- Det er mange standarder for grafikk på skjerm
- Stort sett utvikler standardene seg etter utviklingen av skjerm-teknologien
- Eksempel: SVGA (**S**uper **V**ideo **G**raphics **A**rray)
 - **Oppløsning**: 800*600 → 1600*1200 pixler
 - **Fargedybde**: 256(8 bit) → ca. 16 millioner (24 bit)
 - **Oppfrisknings-rate**: 50 → 100 Hz
- Antall mulige samtidige farger er avhengig av størrelsen på video-minnet.

«Utallige» standarder (px, farger)

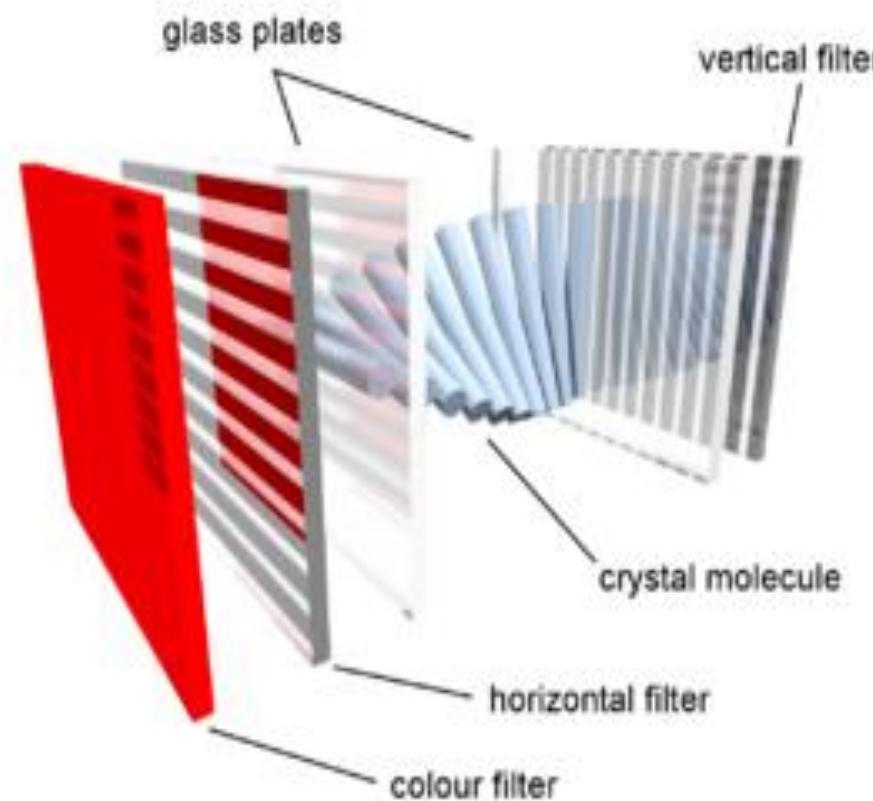


Flat-panel (LCD)

- Hver pixel har en adresse
- Bare pixler som endres blir oppdatert
- Meget stabilt bilde av god kvalitet
- Problematiske i sollys



Eksempel: En LCD-subpixel



- Lys fra bakgrunnstavle
- Polariserende filter (vertikalt)
- Flytende krystall som "vrir" polariseringen når du «setter strøm på»
- Polariserende filter (horisontalt)
- Farge

Hva skal vi kunne?

Hva skal vi kunne nå?

- Hva hovedkort og chipset er
- CPU (vs GPU) og hvorfor kjøling er nødvendig
- RAM: typer og roller
- Harddisk, adressering, partisjonering, paritet og RAID
- Periferi-utstyr, expansjons- og periferi-busser
- Prinsippet bak LCD-skjermer

Neste forelesning

- Operativsystem
 - Hva er det egentlig til?
 - Hvordan henger det sammen med alt vi har vært gjennom så langt
 - Hva er forskjellen på de ulike (Windows, Linux, OSX)
 - Hvordan arbeide i et shell

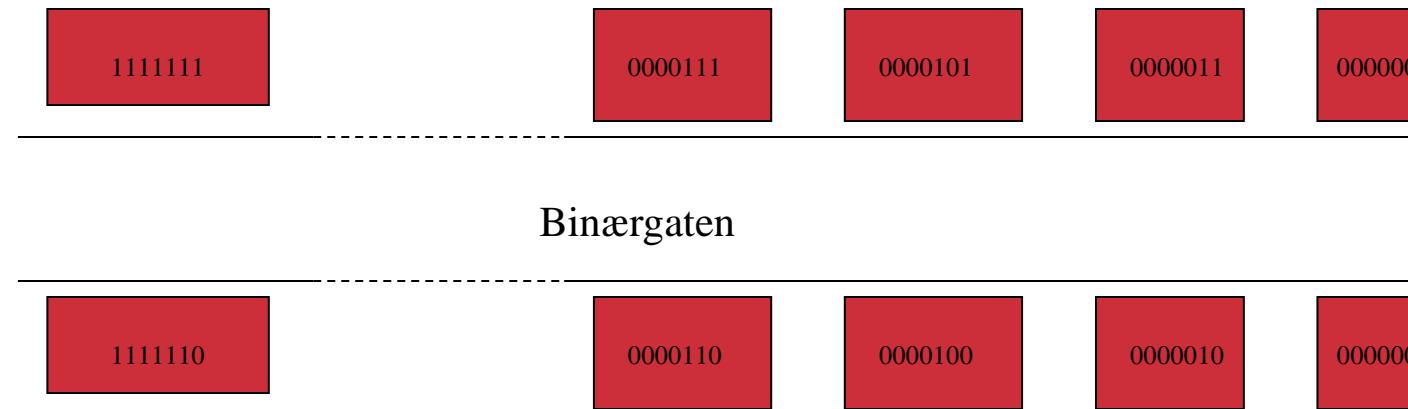
For valgfritt egenstudie

For de som ønsker å lære noen emner mer i dybden for å forstå det bedre er det her samlet noen ekstra temaer relatert til dagens undervisning, det må forventes en del egenarbeid for å forstå disse emnene.

Det vil ikke komme spørsmål på eksamen fra disse, og dette er altså ikke ansett for å være en del av pensum.

Diverse emner

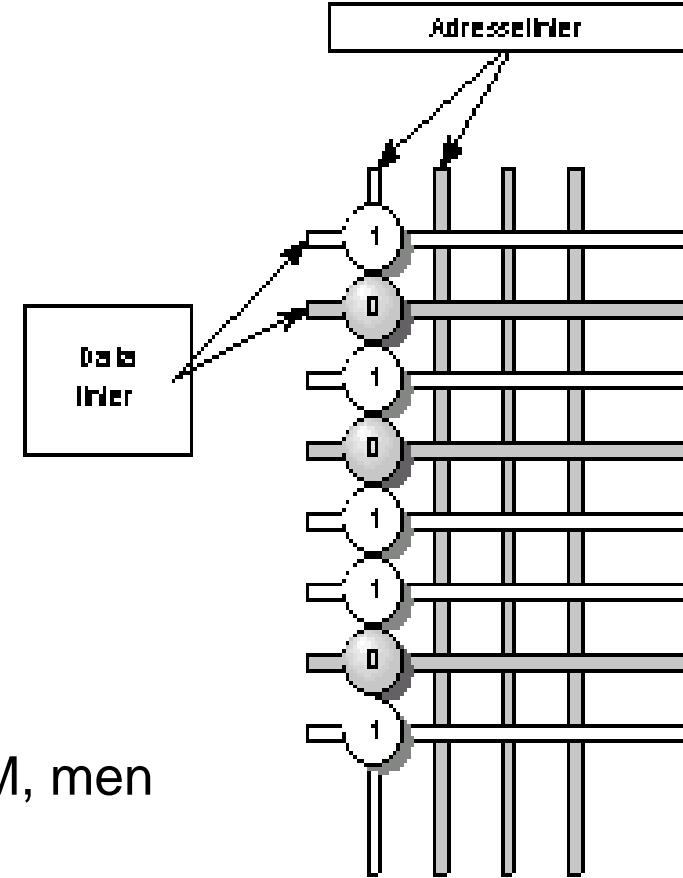
Adressering binært



Med 7 binære sifre(bit) kan $2^7=127+1=128$ = «hus» adresseres.

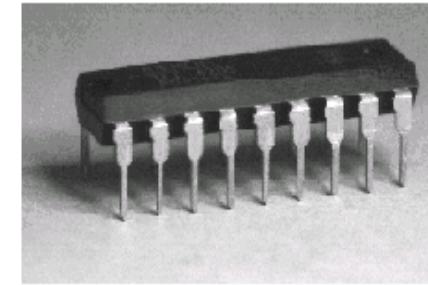
På hver adresse "bor" en byte.

- Adresser
 - 16 bit = 65536 = 64 KiB
 - 20 bit = 1MiB
 - 32 bit = 4 GiB
 - 64 bit = 16 EiB
- **DRAM** - Dynamic RAM
 - Må friskes opp; read/rewrite
- **SDRAM** - Synchronous DRAM
 - I takt med CPU
- **SRAM** - Static RAM
 - Må IKKE friskes opp, raskere en DRAM, men dyrere å produsere og trenger flere transistorer pr bit lagret



Fysisk organisering

- **DIP** (Dual In-line Pin) rett på hovedkortet



DIP

- **SIMM** (Single In-line Memory Module)

- Snappes på hovedkortet
- DRAM; 60 til 120 ns
- SRAM i cache; 20 ns
- Odde **Paritet**
 - **10010110 1**
 - **Brukes for å sjekke om feil har oppstått**



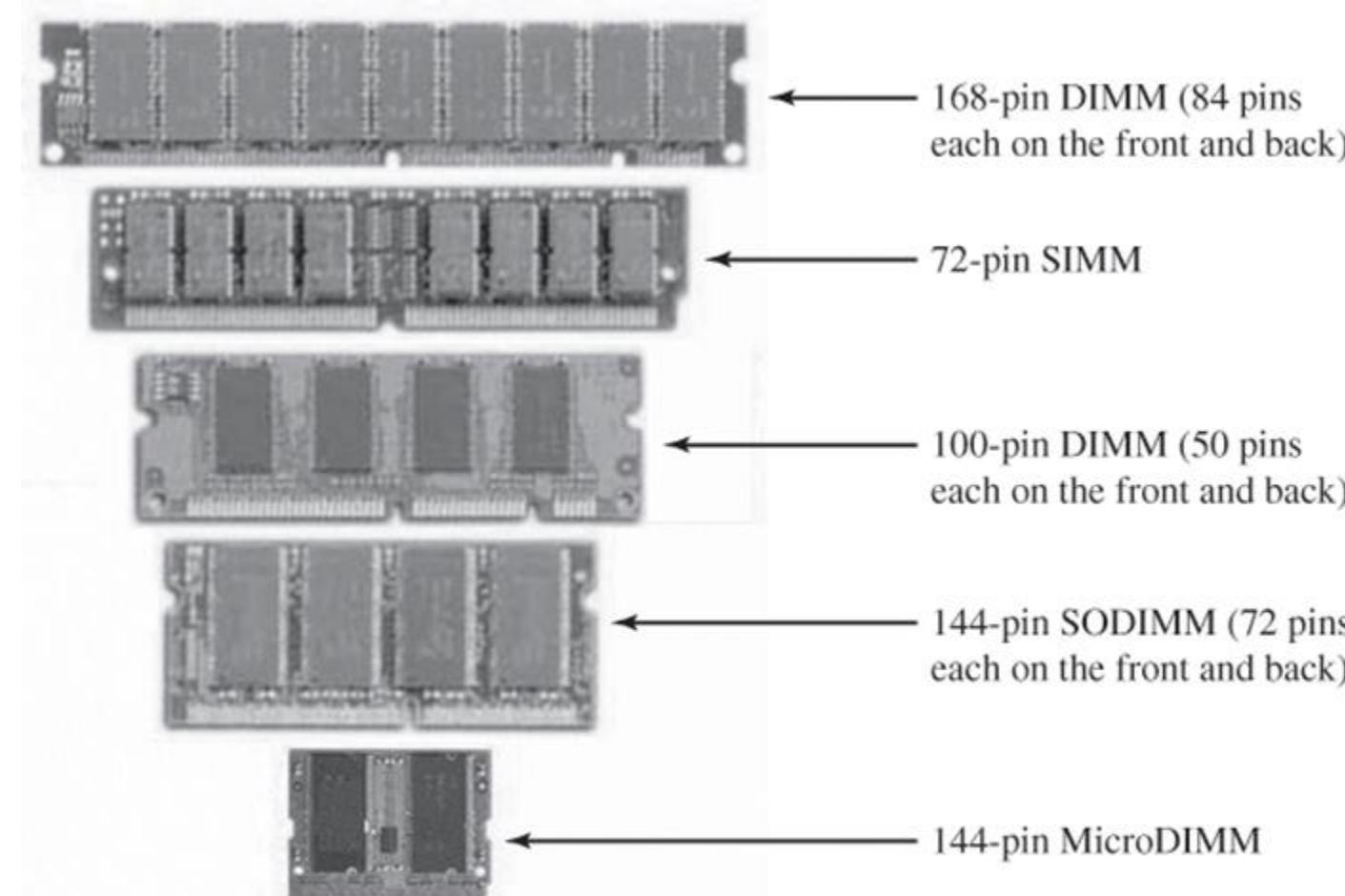
72 pin SIMM 4.25" x 1"

- **DIMM** (Dual In-line Memory Modu'')

- Kontaktpunkter på begge sider
- 64 bit busstilkoppling
 - 168, 184, eller 240 busstilkopplinger
- RDRAM er en variant
 - Finnes mange varianter mhp antall tilkoplingspunkter, type buss mm



DRAM formfaktorer (noen)

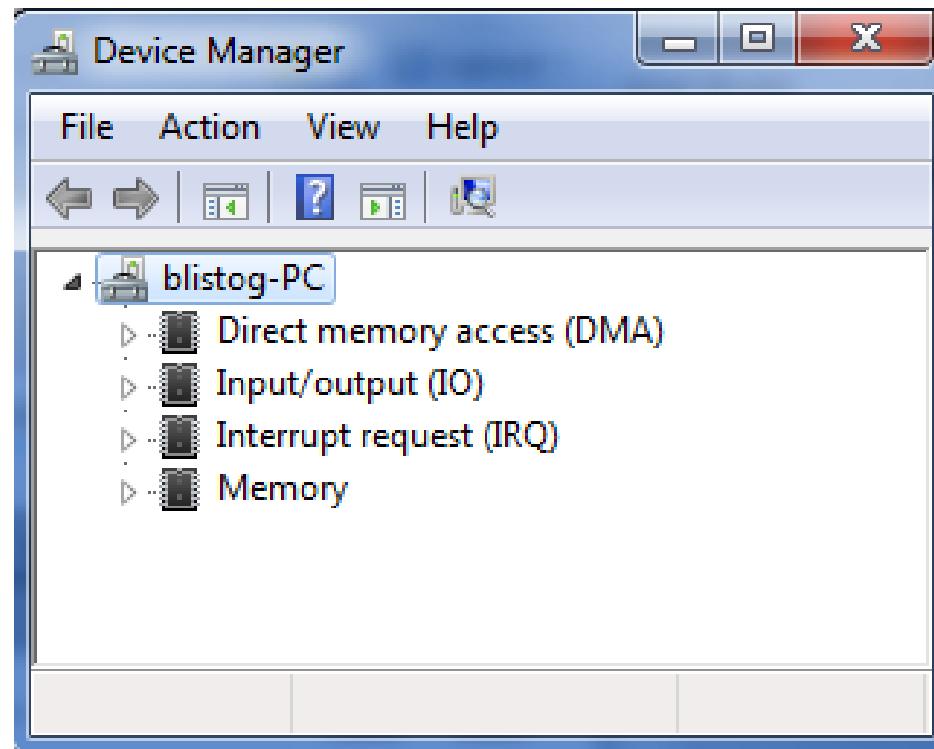


- Tidligere
 - Asynkront, fulgte ikke systemklokken
 - FPM, EDO, m.fl.
- Nåværende
 - Single Data Rate RAM
 - En les/skriv pr klokke-/buss-puls
 - Double Data Rate RAM (DDR, DDR2, DDR3)
 - RAMBUS
- RAM-ytelse avhenger mer av pakken og buss, enn av selve RAMen

DRAM egenskaper

- **CAS** (Column Adress Strobe) **latens**
 - Antall klokkesykluser det tar å flytte etterspurte data til RAM-modulens ben
 - F.eks. CL2 (to klokke sykluser)
- Dual eller quad **kanal** («channel»)
 - Hovedkort-avhengig – antall parallele RAM-kanaler
- **Gull vs tinn ben**
 - Bør være samme på brikke som hovedkort
- **Enkel vs dobbel rekke**
 - Dobbelt rekke («double-ranked») har to separate grupper av RAM-brikker som aksesserers hver for seg av RAM-kontrolleren.

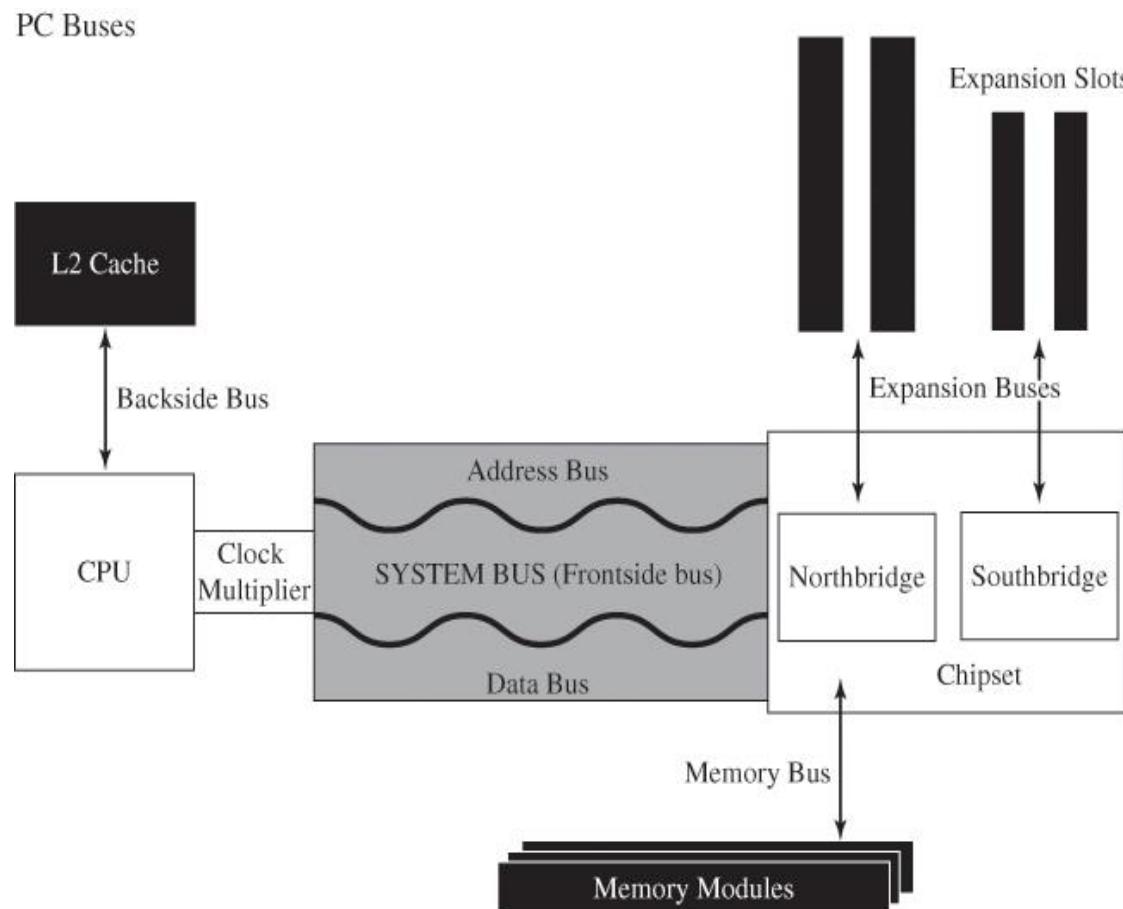
Win7: devmgmt.msc



- I win7/8/10 kan man inspisere hvilke ressurser i MMC snapin devmgmt.msc

Busser

Hovedkort: Busser



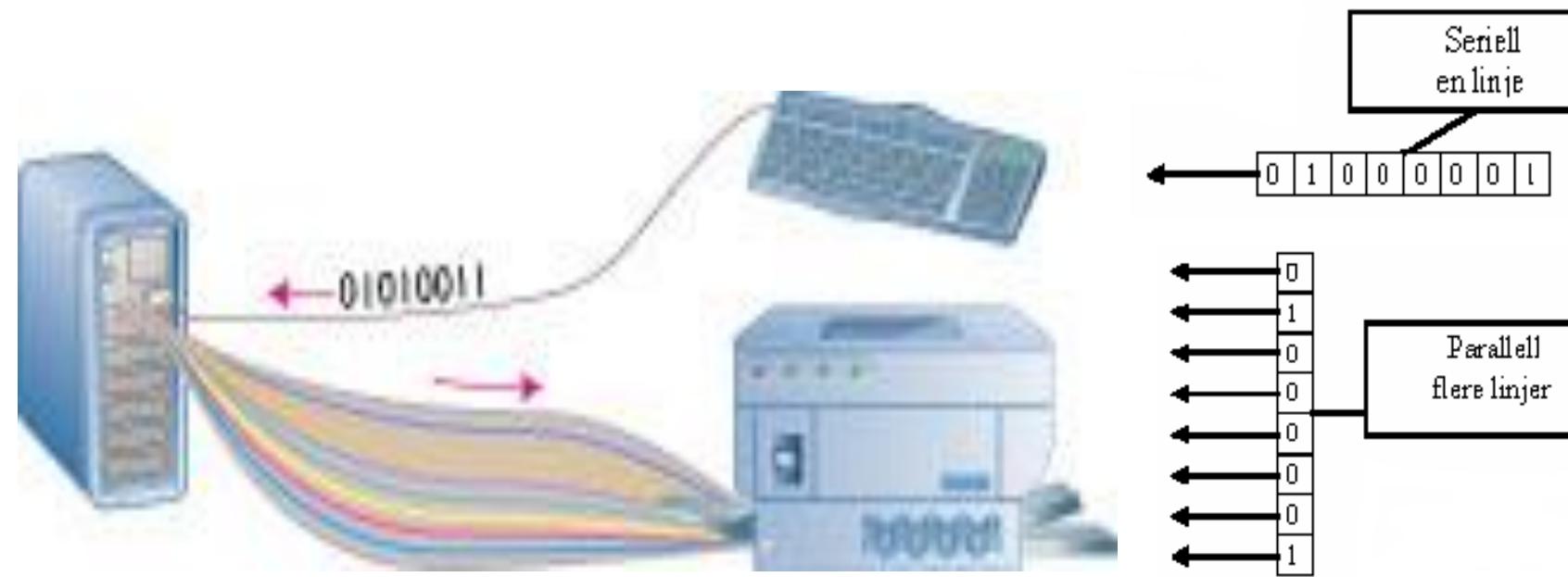
- Serielle vs parallell
- Data/Adresse/
Kontroll/Strøm
- Bredde (#bit), tempo (Hz)
- To hovedbusser:
 - System/FSB
 - forbinder CPU med Northbridge/RAM
 - Ekspansjon
 - Forbinder chipset med ekspansjons-slots

Chipset

- Kontrollerne som styrer utvekslingen av data/instruksjoner/avbrudd-signaler mellom CPU og annet utstyr benevnes gjerne som hovedkortets **chipset**
- De siste ti årene har dette typisk vært to hovedkontrollere
 - **Northbridge** (Intel: Memory Controller Hub)
 - CPU og minne (RAM)
 - Høyhastighets utstyr: grafikk-kort
 - Kommunikasjon med Southbridge
 - **Southbridge** (Intel: IOController Hub)
 - Periferiutstyr (typisk utenfor kabinettet)
 - Mindre viktige kontrollere
 - Sekundære og gammeldagse busser

Transportsystemet

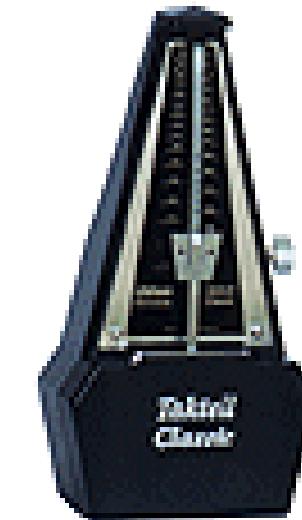
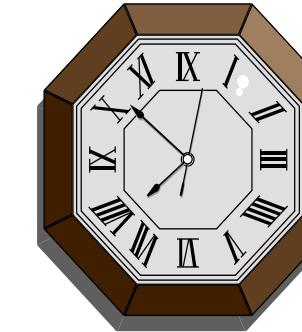
- "Kabler" med parallelle og serielle "ledninger"
- Bredde * Hastighet = Båndbredde



- **Adressering** av data
- **Frakt** av data
- **Synkronisering** av maskinens komponenter
- **Flytkontroll** mellom komponentene
- **Signalering** mellom komponentene
- **Strømtilførsel** til noen komponenter

- Bredde fra 20 til 64 bit
 - 32 bit gir mulighet for å adressere ($2^{32}=$) 4 GB
- Det finnes teknikker med å sende flere bit/byte om gangen
 - Trenger da mindre bredde på bussen
 - Trenger mer kontroll i komponentene

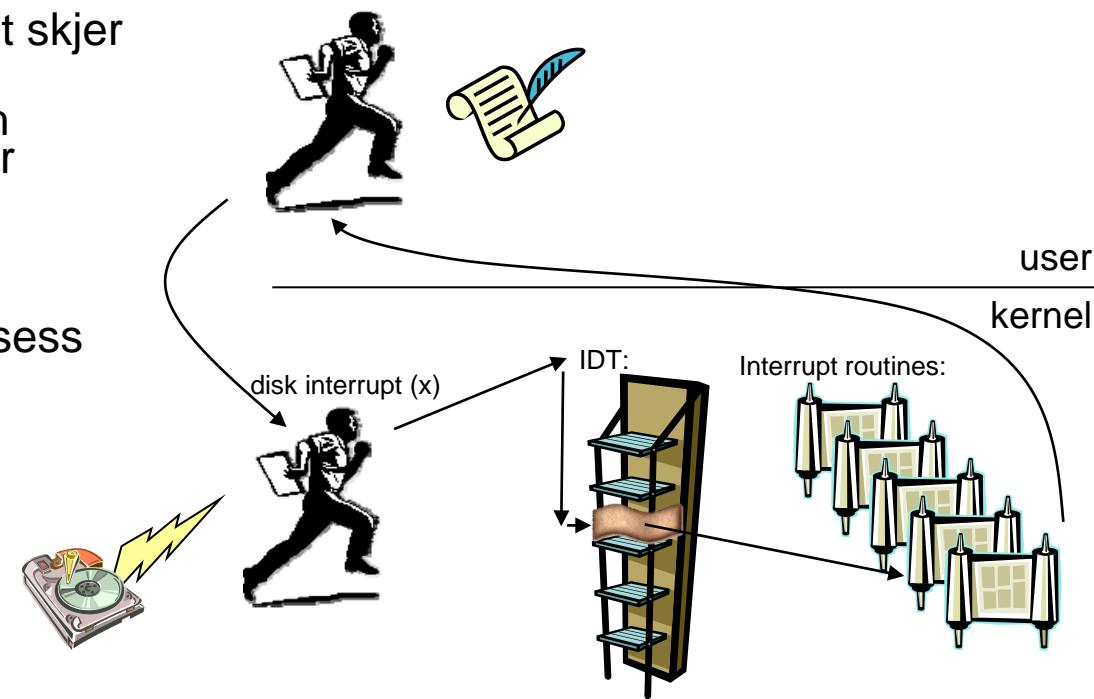
- Internt i maskinen vil det være flere klokker
 - CMOS (BIOS) har en "vanlig" (sanntids) klokke
 - Alle andre klokker er egentlig taktgivere
-
- **Synkrone** komponenter
 - Holder samme eller tilsvarende takt
 - **Asynkrone** komponenter
 - Holder hver sin egen takt



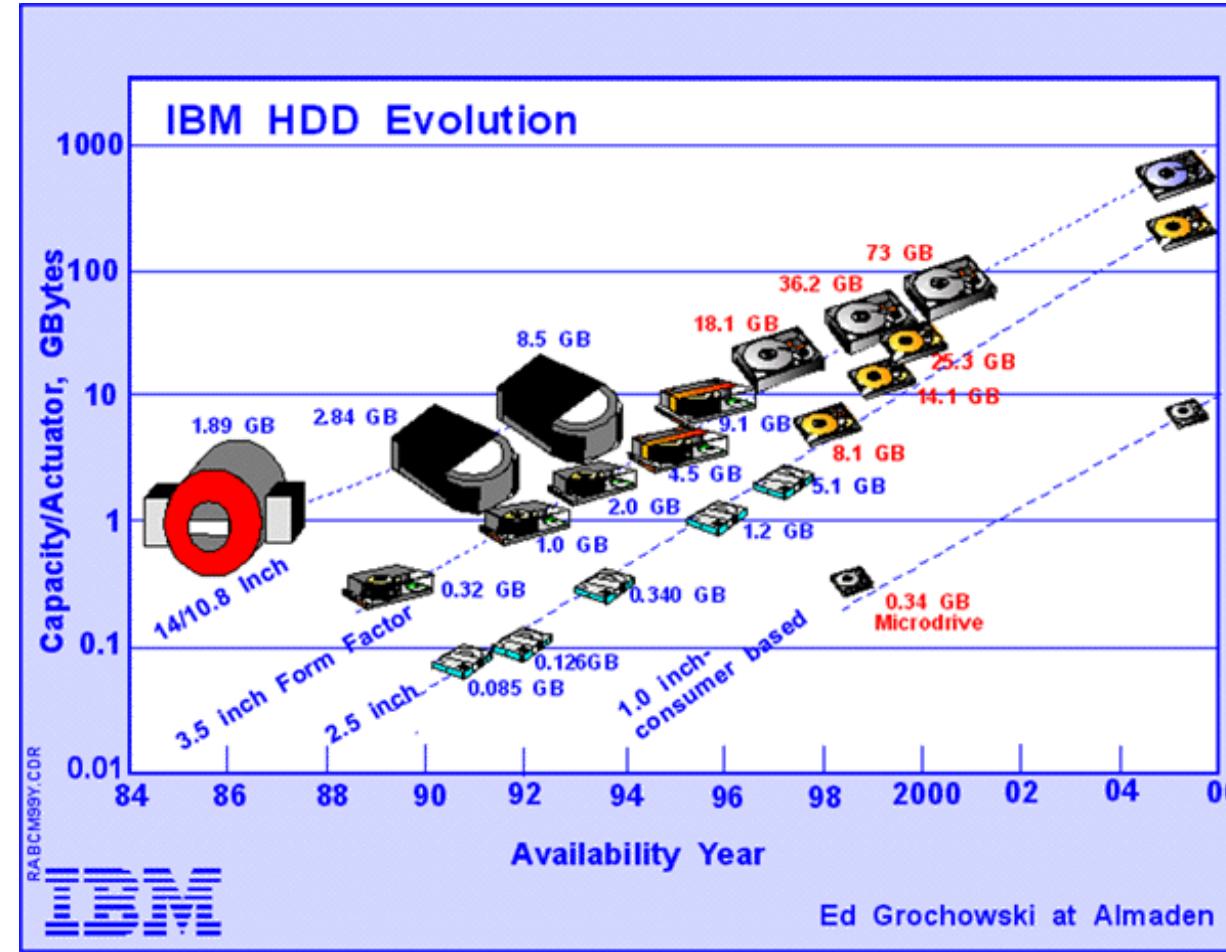
- Avbruddssignal er en beskjed om at noe har hendt
 - Det er mekanismen gjør at CPU og kontrollere får vite at noe/noen vil ha noe gjort
- IRQ - Interruption Request
- Slike signaler kan gis forskjellig prioritet
- Egne komponenter for registrering og viderebehandling

Interrupt (og Exception) Håndtering

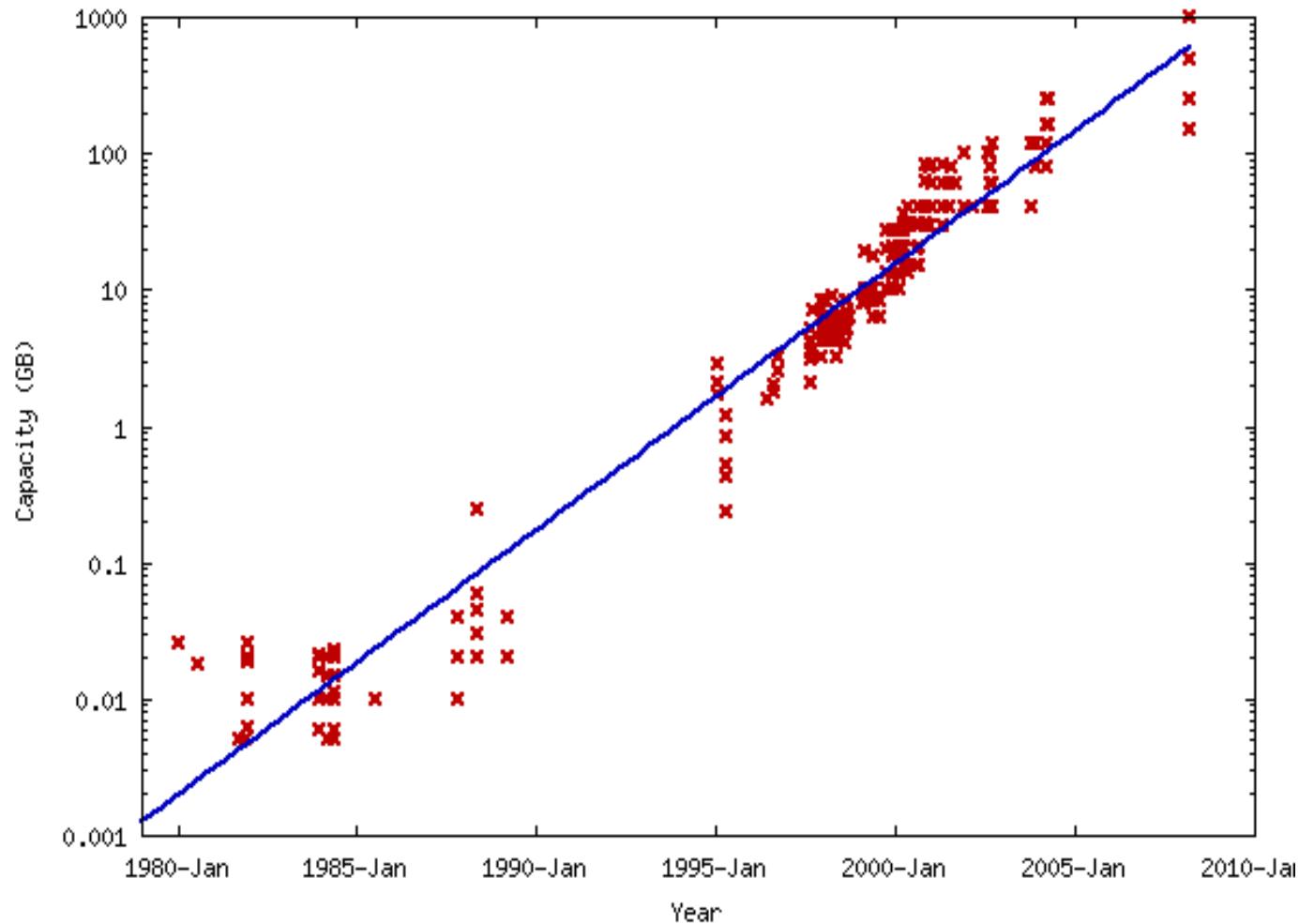
- IA-32 arkitekturen har en IDT med pekere til 256 ulike interrupt and exceptions
 - 32 (0 - 31) forhåndsdefinert og reservert
 - 224 (32 - 255) bruker/OS-definert
- Hvert interrupt har en en peker i Interrupt tabellen (IDT) og en unik index verdi som gir håndtering som følger
 1. Prosess kjører når interruptet skjer
 2. Lagre tilstand, context switch og finn riktig interrupt handler
 3. Eksekver interrupt handler
 4. Gjenopprett (interruptet) prosess
 5. Fortsett eksekvering



Lagrings-utvikling (1)

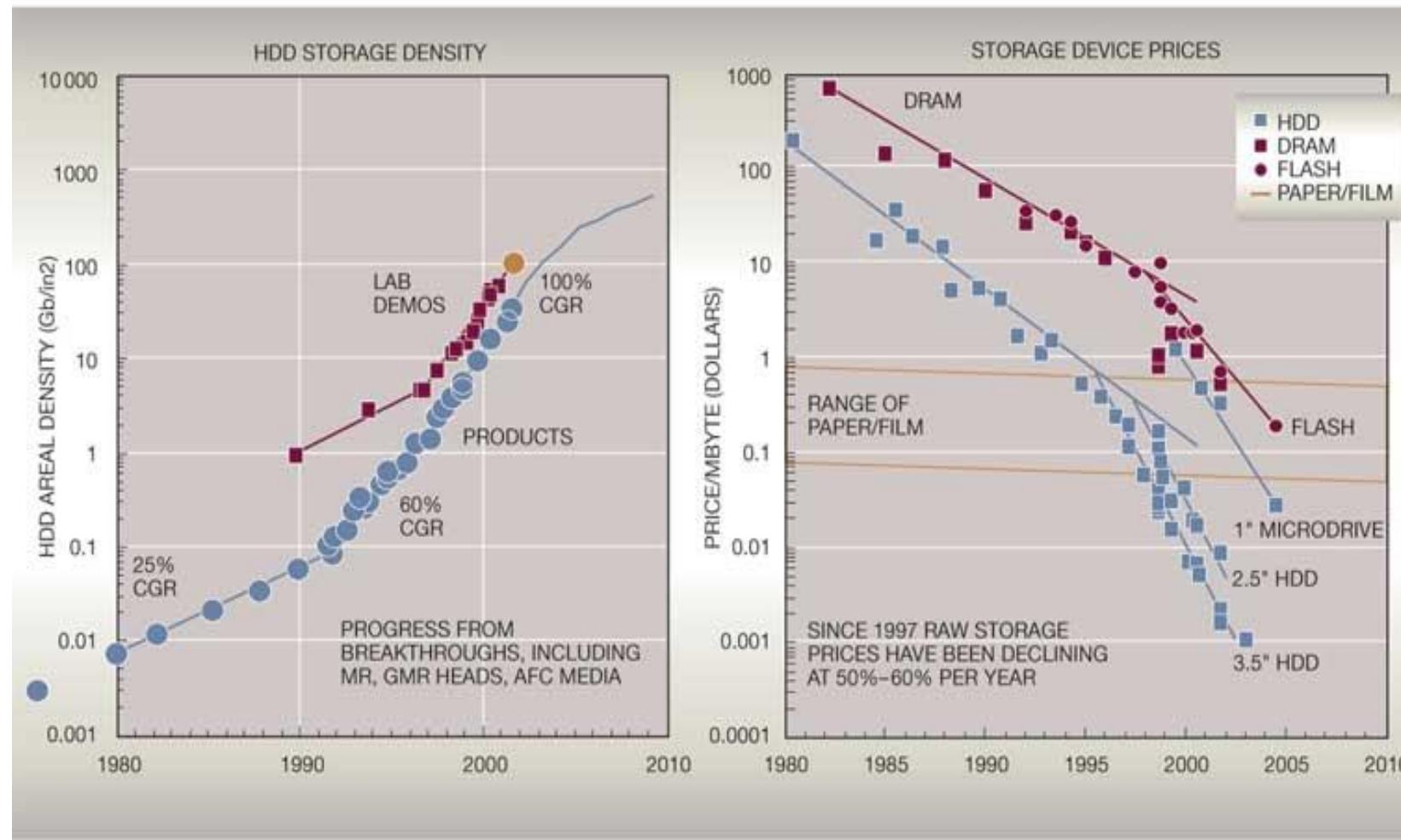


Lagringskapasitet (2)



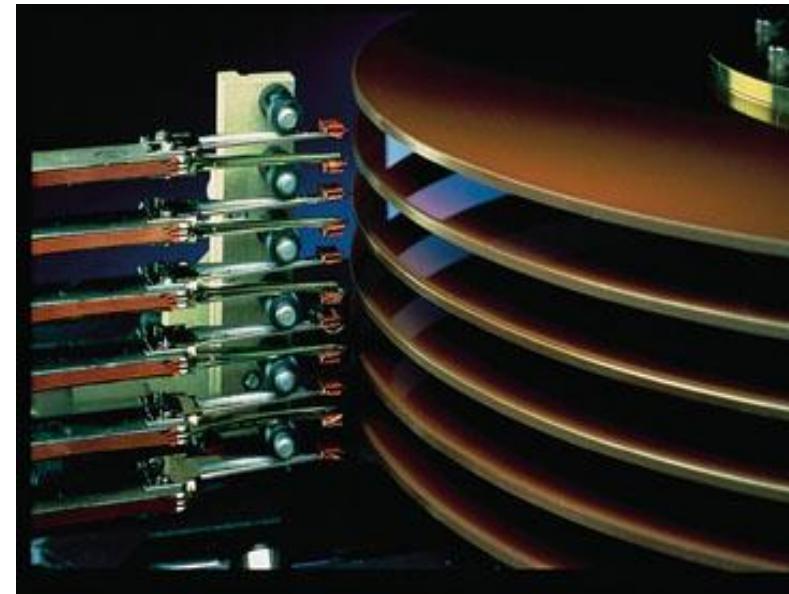
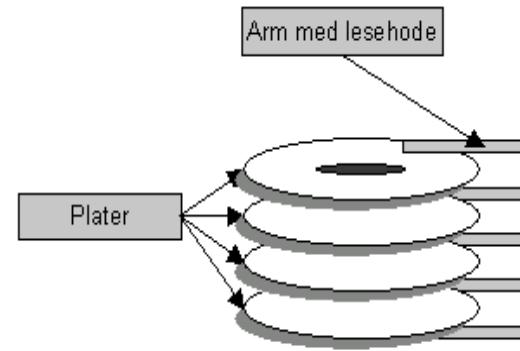
Tetthet og pris (3)

Figure 1 HDD storage density is improving at 100 percent per year (currently over 100 Gbit/in²). The price of storage is decreasing rapidly and is now significantly cheaper than paper or film.



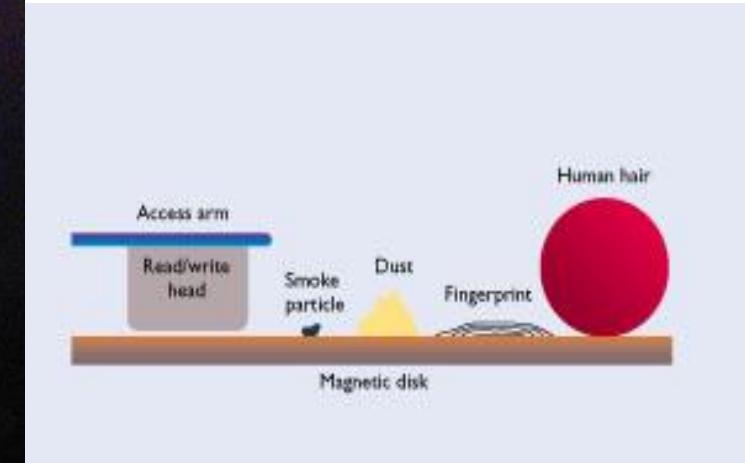
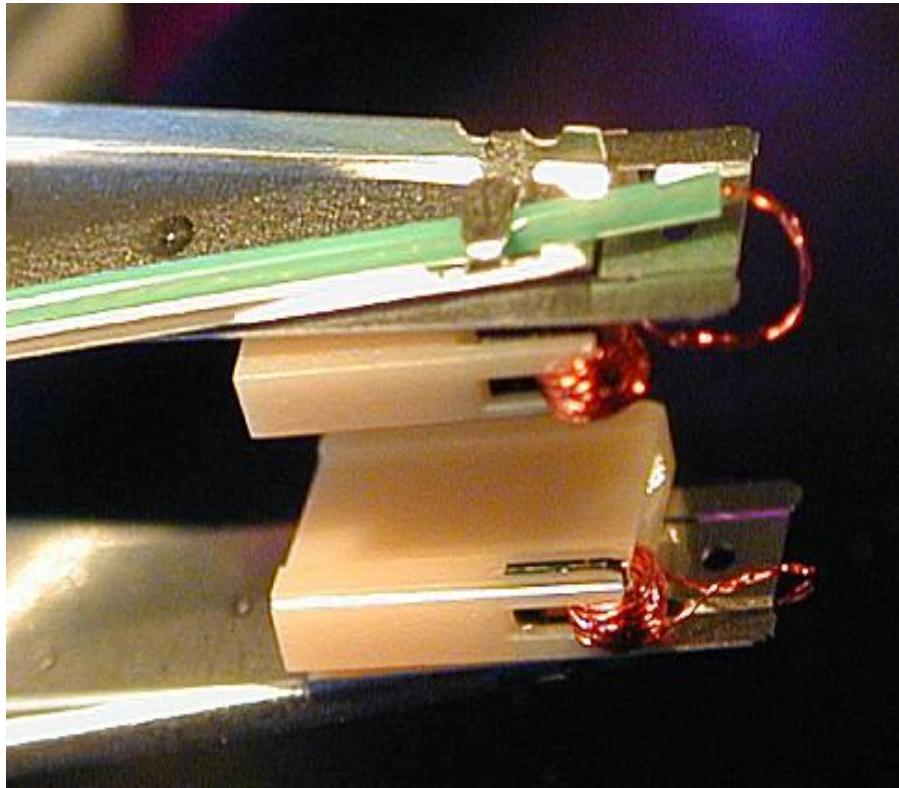
Diskens oppbygning

- Alle armene beveges samtidig
- Hver arm har 2 hoder
- Inn/ut 50 ganger/sek
- Spin: 3600-7200 (15000) RPM
- Søketid: 10-20 ms
- Datarate: 5-40 MB/sek
~300Mbps m/ SATA)



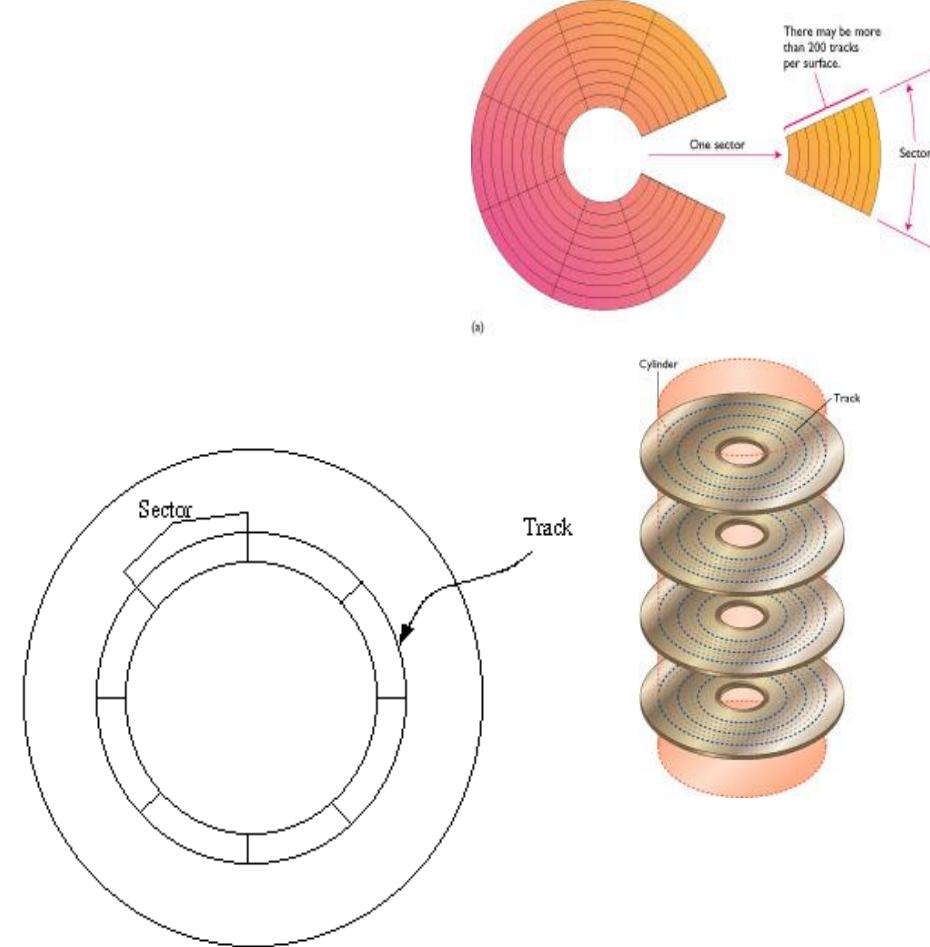
Lesehoder

- Lesehodene flyter på luften over platen(0,001 mm)



Diskens organisering

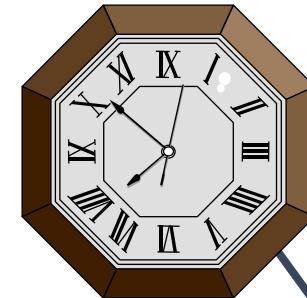
- **Spor (track)**
 - Det området lesehodet dekker på en rotasjon
- **Sektor**
 - Den minste delen av et spor som kan leses
- **Sylinder**
 - Alle spor i samme posisjon på disken



512 bytes pr sektor*51 sektorer pr track*723 sylinder*14 lesehoder=264305664 B=252 MB

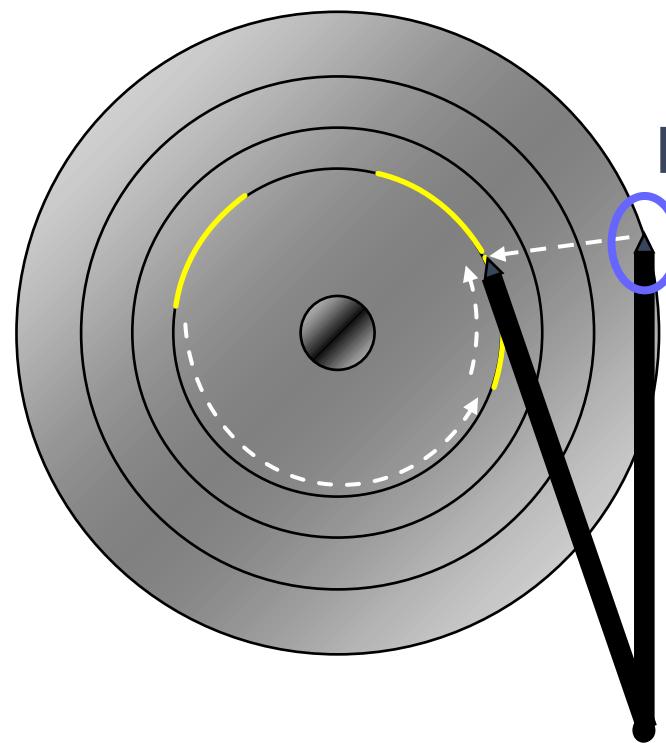
Disk Access Time

Vil ha
blokk X



blokk X
i minnet

Disk plate



Disk hode

Disk arm

Disk access tid =

Søketid

+ Rotasjonsforsinkelse

+ Overføringstid

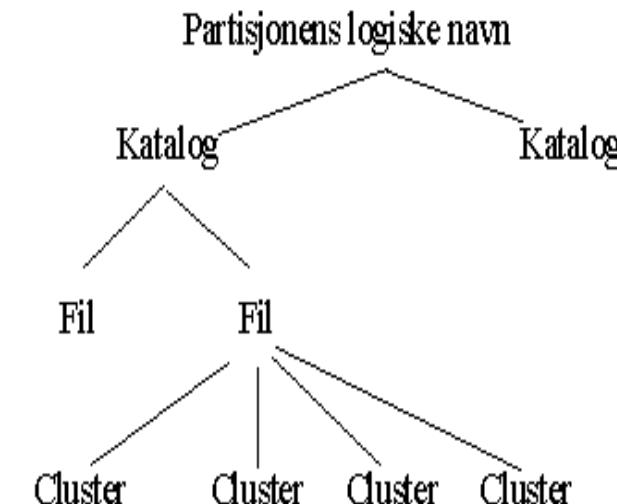
+ Andre forsinkelser

H-Disk adressering av sektorer

- Tidligere (og pga tilbakekompatibilitet) bruktes **CHS** (Cylinder, Head, Sector) eller Extended CHS
 - Opprinnelig begrenset til 1024 C, 16 H, 63 S = 528 MB
 - BIOS Interrupt 0x13 utvidet til 8G ved 0-254 Head
 - 24 bit adresse, 512 B sektor = 8G harddisk
- Nå brukes **LBA** (Logical Block Addressing)
 - Opprinnelig for SCSI, så for ATA/SATA
 - Sektorene har logiske lineære LBA-adresser 0,1,2,3...
 - C 0, H 0, S1 = LBA 0, ...
 - 32 bit adresser gir max størrelse 2TB
 - LBA støttes av alle moderne OS og BIOS
- I fremtiden ser **GUID** ut til å overta
 - Støttes av UEFI
 - 64 bits adresser
 - tillater i prinsippet diskene opp til 8 ZiB (forutsatt 512 Byte blokker)

Diskens logiske organisering (hierarki)

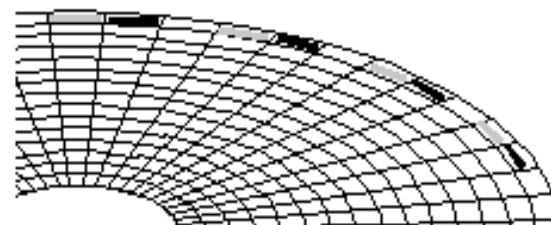
- **Cluster**
 - Minste enhet som leses av operativsystemet, minimum 1 sektor
- **Fil**
 - Minste enhet som refereres av operativsystemet, minimum 1 cluster
- **Katalog**
 - Fil for organisering av filer
- **Partisjon**
 - Organisering av kataloger; f. eks. C:



Lavnivå-formatering

- Lavnivåformatering gjøres av produsent
- Adresser skrives på hver sektor
- Disken initialiseres med nullverdier
- Defekte områder merkes
- Interleave-mønsteret settes
 - Eksempelet viser 3:1 interleave

— =Cluster 1
— =Cluster 2
— =Cluster 3



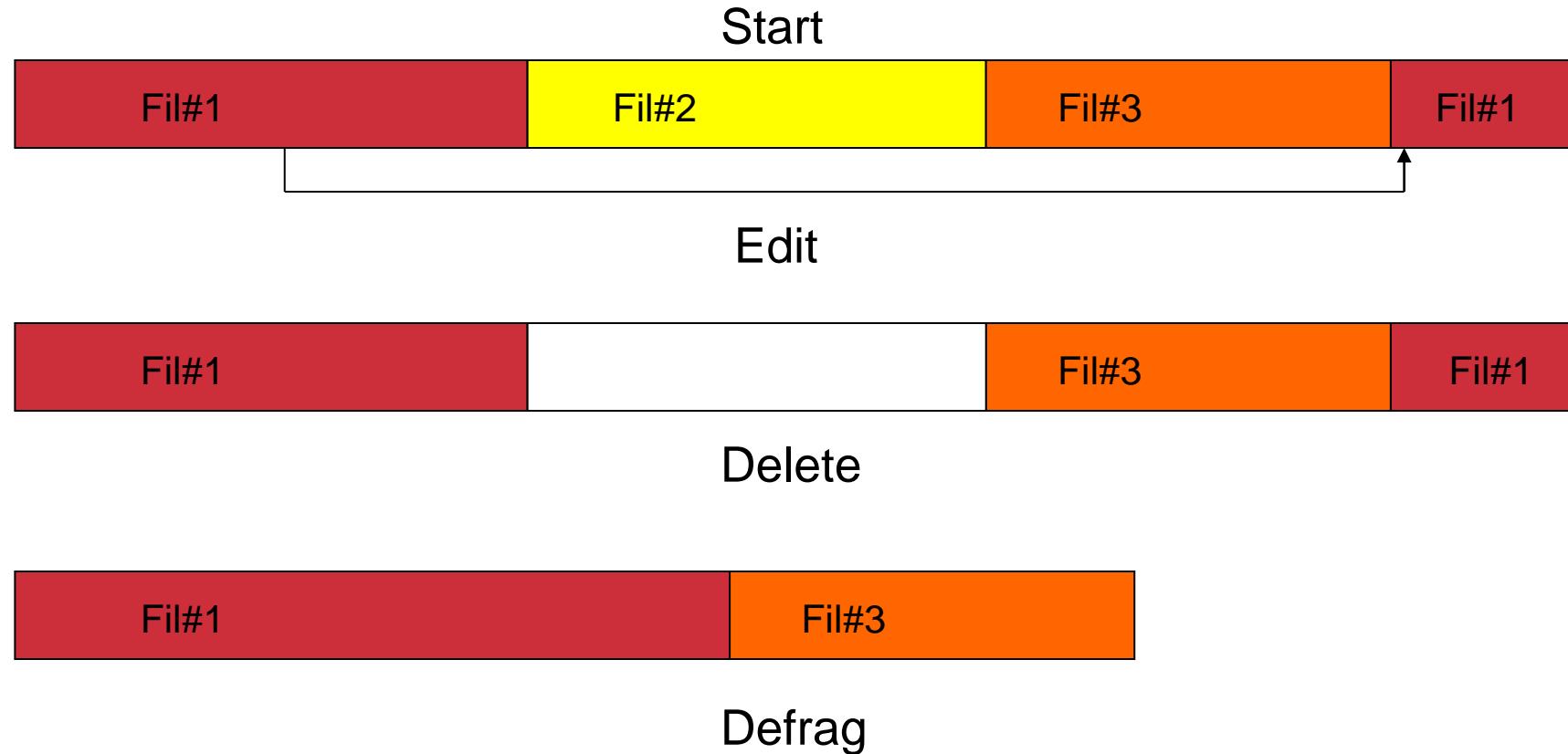
Partisjonering

- Partisjonering gjøres av leverandør (eller bruker)
- Gjøres fra «DOS» med programmet FDISK
- Lager **MBR** (Master Boot Record) på starten av disken
 - Brukes av BIOS for å starte operativsystemet
- Minimum 1 partisjon (primary)
 - Kan legge til secondary (extended) partisjoner
- Kan ikke ha to operativsystemer i *samme* partisjon (annet enn som Virtuell Maskin)

Høynivå-formatering

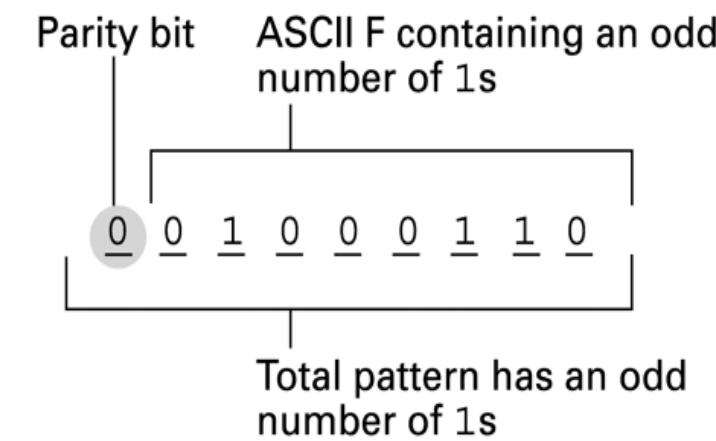
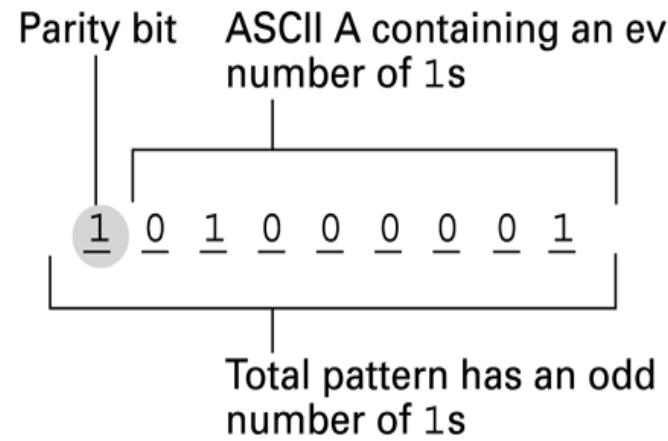
- Høynivå-formatering gjøres av leverandør (eller bruker)
- Kan gjøres fra DOS med programmet FORMAT
 - Får advarsel fra operativsystemet hvis det er primary partisjon
- Legger innhold fra MBR inn i oppstartsektor
 - Ikke alle disker (disketter) kan bootes
- Lager rot-katalog
- Definerer hvor mange sektorer det er i en cluster
- Bygger filallokeringstabeller (FAT)
 - FAT12, FAT16, FAT32, (NTFS, Ext3)

Fragmentering

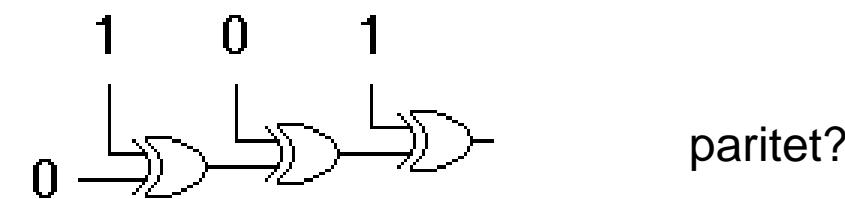


Paritetsbit

- For å kunne identifisere "bad blocks" (sektorer med feil) har hver sektor paritetsbyte som benyttes til å sjekke om feil har oppstått.
- Paritetssjekk var en del av 7bit ASCII:



- 1 paritetsbit kan bare oppdage oddetallsfeil!
- Bruker CRC-sjekksummer som gir mulighet til å oppdage langt flere feil (mer i nest siste forelesning)
- Paritet kan enkelt genereres med XOR!
- Ved bruk av smart koding kan man lage bedre sjekksummer. (jf personnummer)



Error Correcting Codes (ECC)

- Basert på Hamming-distansen mellom kodene

Symbol	Code
A	000000
B	001111
C	010011
D	011100
E	100110
F	101001
G	110101
H	111010

Eksempel

I dette kodesettet er det alltid minst tre bit forskjell mellom alle kodene

Hamming-distansen mellom A og B er 4, B og C 3, osv

ECC (2)

Character	Code	Pattern received	Distance between received pattern and code
A	0 0 0 0 0 0	0 1 0 1 0 0	2
B	0 0 1 1 1 1	0 1 0 1 0 0	4
C	0 1 0 0 1 1	0 1 0 1 0 0	3
D	0 1 1 1 0 0	0 1 0 1 0 0	1
E	1 0 0 1 1 0	0 1 0 1 0 0	3
F	1 0 1 0 0 1	0 1 0 1 0 0	5
G	1 1 0 1 0 1	0 1 0 1 0 0	2
H	1 1 1 0 1 0	0 1 0 1 0 0	4

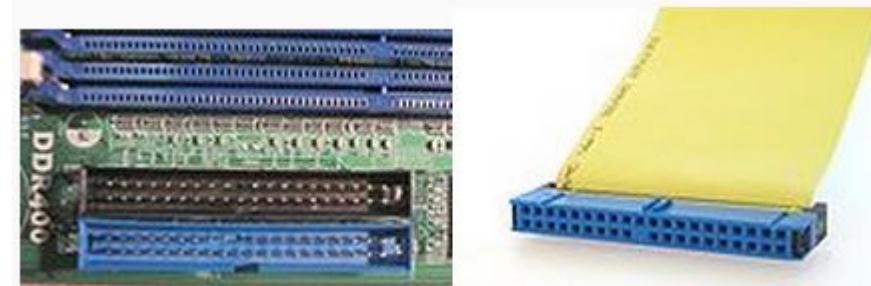
1 → Smallest distance

- Leser 010100, hvilken bokstav skulle det (sannsynligvis) vært?
 - Den med minst Hamming-distanse i kodesettet
- Med Hamming-distanse 3 i kodesettet kan man oppdage opp til to feilkodinger (bit-flips) og rette en.
- Med Hamming-distanse 5 i kodesettet vil man kunne oppdage opp til fire feil pr mønster og rette to

Busser/gr.snitt: ATA, SATA

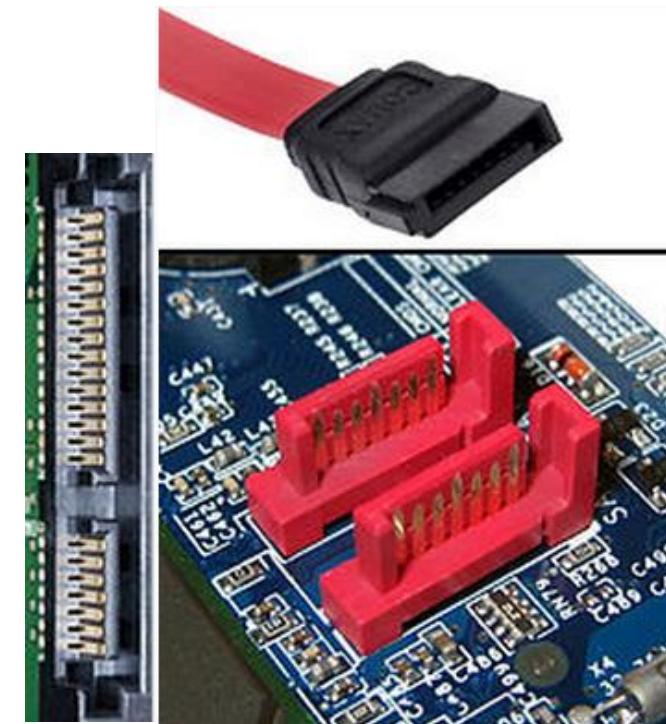
- (Parallel) ATA

- Versjoner 1-7
- Opprinnelig IDE
- Støttet LBA og DMA
- Kabel med 80 parallele ledere
- Max 133 MBps

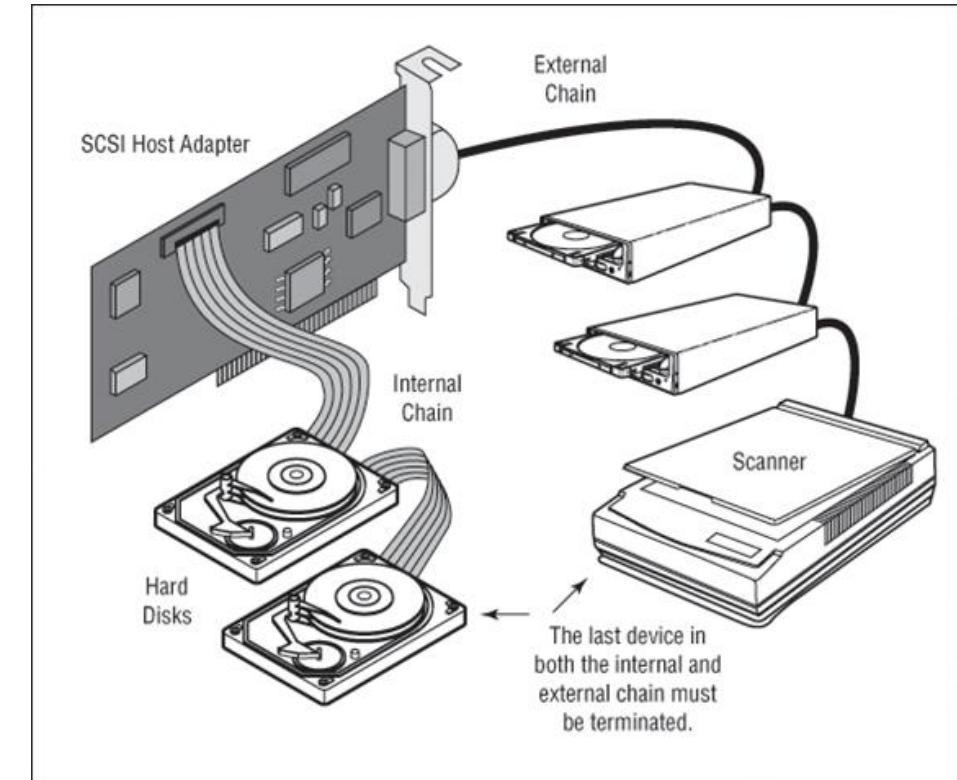


- SATA: **Seriell**

- Siste versjon støtter opp til 6 Gbps (v. 3.0)
- Kabler opp til 8 m
- **eSATA** for ekstern tilkopling
- HotPlug!



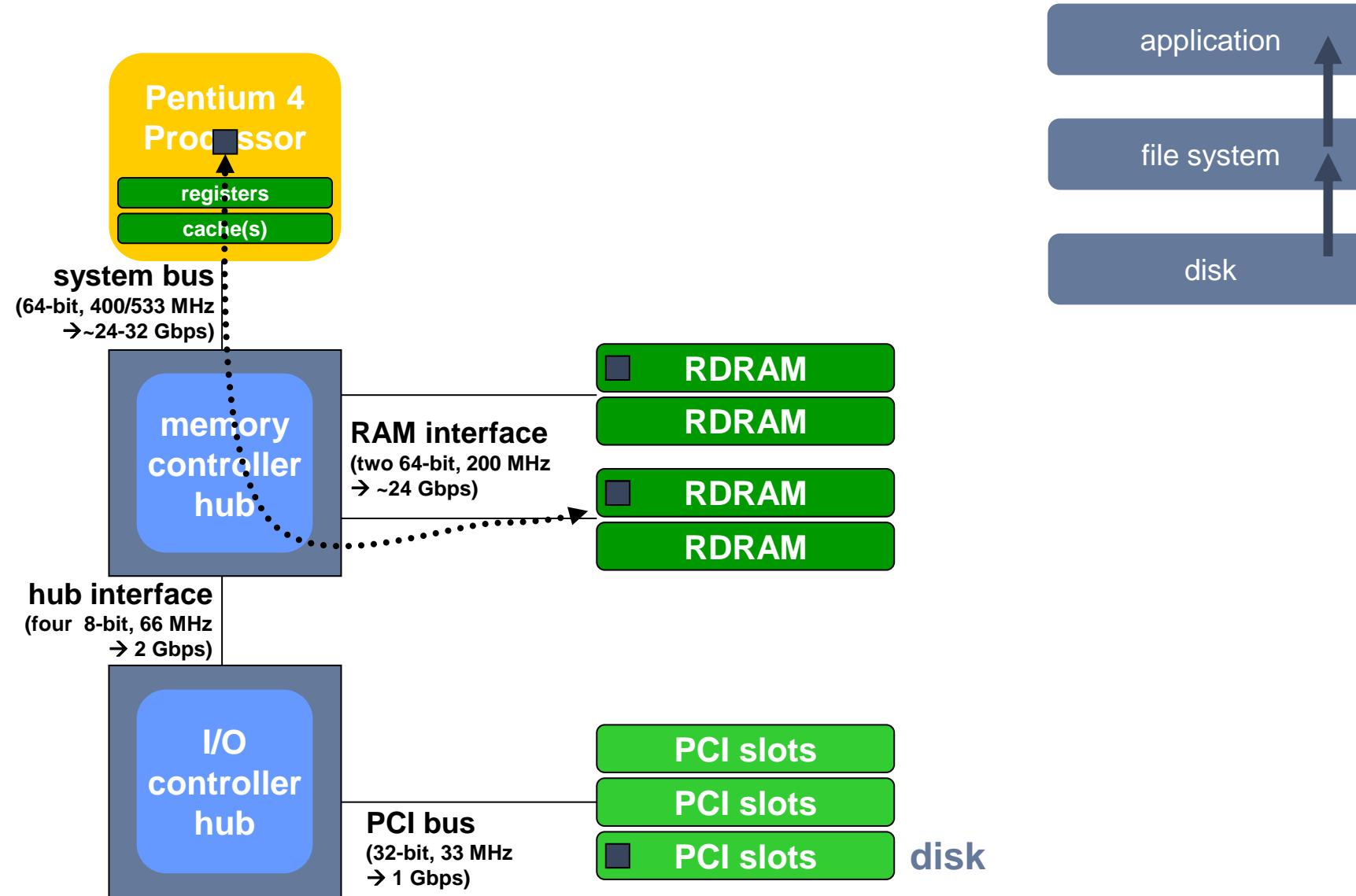
- Tillater seriekoppling av utstyr
 - Hver må ha egen ID
- Mest brukt på servere
- Versjoner 1-3
- En samling av ulike standarder!
 - iSCSI er f.eks. en kombinasjon av Etrhernet og SCSI



- I/O-enheter består ofte av
 - Mekanisk komponent ("selve enheten")
 - Elektronisk komponent
- Den **elektroniske** komponenten er **enhetskontrolleren** (*device controller*)
 - "En liten mikroprosessor"
 - Kan evt. håndtere flere enheter
- Kontrollerens oppgaver
 - Ta i mot instruksjoner
 - Vise status
 - Konverter seriell bitstrøm til blokk av byte
 - Utfør feilkorrigering etter behov
 - Gjør data tilgjengelig i primærminnet (RAM)

- Hver kontroller har noen **registre** som brukes for å oppbevare status og kommunisere med CPU
 - Kan skrive til disse for å kontrollere enheten
 - Kan lese andre for å finne tilstand til enheten
- Kan også ha data-buffer
 - Video RAM på skjermkort
 - Cache PÅ disk
- Kan skrive registre/buffer vha. I/O-porter
- Kan også bruke minne-avbildet (memory mapped) I/O

Ex: lese inn fra disk

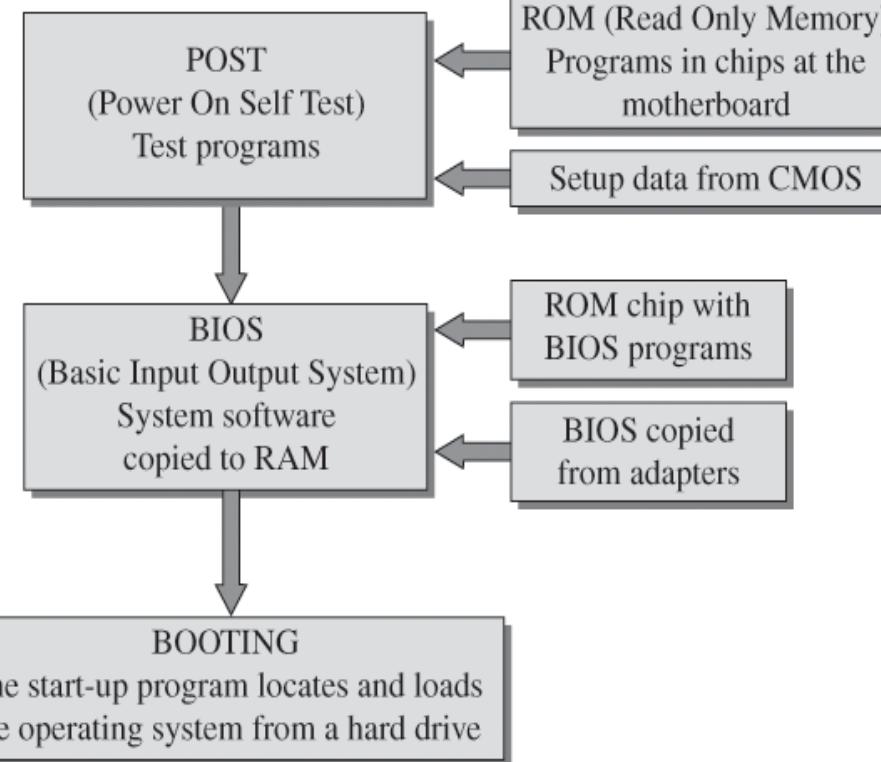


Oppstart

- **Programvare** for å kontrollere essensielle komponeneter
 - F.eks. Tastastur, skjerm, disker, serielle porter, ...
- Lagres på ROM brikke
- Tre hovedfunksjoner
 - Boote PCen
 - Teste og verifisere hardware
 - Levere grensesnitt mellom hardware og software.

Boot sekvensen

1. Strømforsyningen starter opp
2. Primærdelen av BIOS lastes inn i RAM
 - CPU leser inn startadressen til BIOS boot programmet
3. Type boot bekreftes
 - Cold boot: slås på
 - Warm boot: omstart/rest, kjører ikke POST when PC restarted or reset, does not run POST
4. Power On Self Test (POST) begynner
 - Feilmeldinger gis som "beep koder"
5. Video adapter programvaren lastes
6. BIOS for annet utstyr lastes
 - BIOS startup screen lastes
 - Feilmeldinger vises som text.
7. CMOS konfigurering testet
 - Sjekker at utstyr som er opplistet i CMOS er tilstede og fungerer.



8. Porter tildelt og utstyr konfigurert.
9. Hardware konfigurasjon bekreftet
10. BIOS viser oppsummeringskjerm
10. Operativ systemet lokalisert og lastes (MBR)

Boot mislykkes:

Når PCen ikke klarer å fullføre BIOS og laste OS

Typisk årsak er løse eller manglende komponenter, og/eller OS/driver-feil på disk.

Power On Self Test (POST)

- Feil-koder er (hovedsakelig) avhengig av BIOS produsent
- To typer
 - Beep-koder
 - Lange og korte toner i høyttaler
 - Feilmeldinger
 - Til skjerm
 - 1xx Motherboard, 2xx RAM, 3xx Tastatur, 17xx HD,...
 - 3-4 siffer med litt forklaring
 - «BIOS ROM checksum Error: System halted»
 - «Hard disk install failure»
 - «Memory Test Fail»
 - Enkelt BEEP i høyttaler når POST utført

- BIOS **setup programmet**: Brukes til å konfigurer innstillingar som **lagres** i CMOS
- CMOS innstillingar ≈ Hardware konfigurasjon
- SETUP:
 - Har typisk menyer for å sette:
 - Standard Innstillingar
 - Boot medium rekkefølge
 - System dato og tid
 - Hard disk, floppy, ..
 - Video display
 - Avanserte innstillingar: Hovedkort, CPU, chipset
 - Plug and Play (PnP) opsjoner, strømstyring, sikkerhet og passord, integrerte kontrollere for periferiutstyr.

Oppdatering av BIOS

- BIOS «bestemmer» hva slags utstyr PCen kan ha tilknyttet
- Oppgradering («flashing») av BIOS trengs av og til å for å løse spesielle problemer
 - Man kan ødelegge BIOS brikken ved å avbryte under oppdatering, eller forsøke å legge inn feil BIOS
- Erstattes nå av ulike typer firmware

- BIOS er for **16 bit** maskiner...
 - Støtter bare **1 MiB** minne
- Er i gang med å erstattes med **Unified Extensible Firmware Interface (UEFI)**
 - Støtter både 32 og 64 bits instruksjoner fra oppstart av
 - Støtter minimum 2GB minne
 - Støtter en ny adresseringsmåte for partisjoner og blokker på Harddisken (GPT i stedet for MBR) som gjør at større harddisker enn 2 TB kan brukes -> $8*2^{70}$