IN1020 - Summary

Tanusan Rajmohan - tanusanr@ulrik.uio.no

UNIVERSITY OF OSLO

Autumn 2017

Contents

1 Digital representasjon

Et *bit* ("binary digit") er et begrep for noe som har to tilstander. Disse kan være: (0, 1), (true/false), (yes/no), (0v, 5v). Lettere lagring i digital i forhold til analog og singalene sendes enklere med digital, robust signaloverføring.

1.1 Binære tallsystemet (0,1)

Et binært tall er representert ved symbolene 0 og 1.

Eks: $101_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 5_{10}$

Eks: desimal til bit:

Konverter tallet 100_{10} til binær:

$100/2 = 50 + 0/2 \rightarrow a_0\ 0$ (LSB)

$50/2 = 25 + 0/2 \rightarrow a_1\ 0$

$25/2 = 12 + 1/2 \rightarrow a_2\ 1$

$12/2 = 6 + 0/2 \rightarrow a_3\ 0$

$6/2 = 3 + 0/2 \rightarrow a_4\ 0$

$3/2 = 1 + 1/2 \rightarrow a_{54}\ 1$

$1/2 = 0 + 1/2 \rightarrow a_6\ 1$

100_{10}

100/2	0
50/2	0
25/2	1
12/2	0
6/2	0
3/2	1
1/2	1

} 1100100

1.2 Oktale tallsystemet (8 som base)

¸ Et oktalt tall er representert ved symbolene 0, 1, 2 $\rightarrow$ 7

Eks: $252_8 = 2 * 8^2 + 8 * 8^1 + 2 * 8^0 = 170_{10}$

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

1.3 Heksadesimalsystemet (0-9 + a-e)

Et heksadesimalt tall er representert ved symbolene 0, 1, 2, ... 8, 9,

A, B, C, D, E, F

Eks: $2B9_{16} = 2 * 16^2 + 11 * 16^1 + 9 * 16^0 = 69710$

Eksempel med desimal tall:

$1A5,1C_{16} = 1 * 16^2 + 10 * 16^1 + 5 * 16^0 + 1 * 16^- + 12 * 16^{-2}$

$= 421,109375_{10}$

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

Oktale, heksadesimale og binær telling

Heksadesimal	Desimal	Oktal	Binær	
0 0	0 0	0 0	0 0 0 0 0	
0 1	0 1	0 1	0 0 0 0 1	
0 2	0 2	0 2	0 0 0 1 0	
0 3	0 3	0 3	0 0 0 1 1	
0 4	0 4	0 4	0 0 1 0 0	
0 5	0 5	0 5	0 0 1 0 1	
0 6	0 6	0 6	0 0 1 1 0	
0 7	0 7	0 7	0 0 1 1 1	
0 8	0 8	1 0	0 1 0 0 0	
0 9	0 9	1 1	0 1 0 0 1	
0 A	1 0	1 2	0 1 0 1 0	
0 B	1 1	1 3	0 1 0 1 1	
0 C	**1 2**	**1 4**	**0 1 1 0 0**	Tallet $(12)_{des}$
0 D	1 3	1 5	0 1 1 0 1	
0 E	1 4	1 6	0 1 1 1 0	
0 F	1 5	1 7	0 1 1 1 1	
1 0	1 6	2 0	1 0 0 0 0	
1 1	1 7	2 1	1 0 0 0 1	
1 2	1 8	2 2	1 0 0 1 0	
1 3	1 9	2 3	1 0 0 1 1	
1 4	2 0	2 4	1 0 1 0 0	

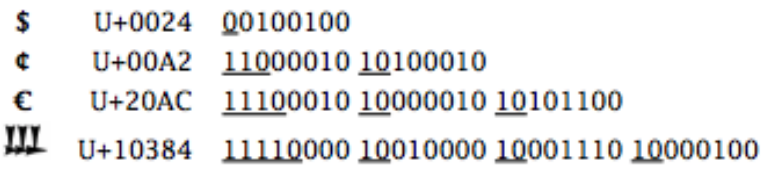
1.4 Tekst

Måten vi lagrer tegn i en datamaskin er vet å sette opp en tabell over tegn vi trenger og deretter gi hvert tegn et nummer. Færste vellykkede forsøket på standarisering var ASCII i 1963:

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	@	@	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	&#127;	DEL

1.4.1 Unicode

Unicode er en løsning som ble laget for tegn som ikke har plass i ASCII tabellen. Unicode skal omfatte alle skriftspråk som brukes eller har vært brukt. Det er plass til drøyt 1 000 000 tegn, men kun 136 755 tegn har blitt definert så langt. Unicode lagres som regel slik som **UTF-8**, da UTF-8 bruker 1 til 4 byte for å lagre et tegn.



1.5 Bilder

På en fargeskjerm består hvert bildepunkt ("pixel") av tre farger: rød, grønn og blå som kan lyse sterkt eller svakt.

1.5.1 Fargetabell

Vi bruker 3 byte ? 24bit til hvert piksel, men vi har bare 7 ulike farger i bildet. Da kan vi sette opp en tabell.

0	0 0 255	mørkeblå
1	135 206 255	mørkeblå
2	255 214 0	mørkeblå
3	0 0 255	mørkeblå
4	0 0 255	mørkeblå
5	0 0 0	mørkeblå
6	1 1 1	mørkeblå

I mange rasterbilder (et digitalt bilde bygd opp av rader og kolonner av bildeelementer/pixler) er det ofte mange like piksler på rad. Da kan vi lagre fargen og hvor mange slike piksler det er på rad. Formater som *PNG* og *GIF* bruker slike teknikker.

$$363\ 682 \text{ byte} \Rightarrow 45\ 483 \text{ byte} \Rightarrow 2\ 917 \text{ byte}.$$

1.5.2 JPEG-formatet

JPEG benytter dette til å lage en komprimert versjon av bildet. Vi kan få ytterligere reduksjon ved å senke kvaliteten.

Figure 1: Dette er en komprimering med **tap**. Det er umulig å komme tilbake til det opprinnelige bildet.

Ekte rasterbilde	40,7 MB
JPEG 100%	5,5 MB
JPEG 50%	0,94 MB
JPEG 25%	0,61 MB
JPEG 10%	0,34 MB
JPEG 5%	0,24 MB

1.5.3 Anbefalinger

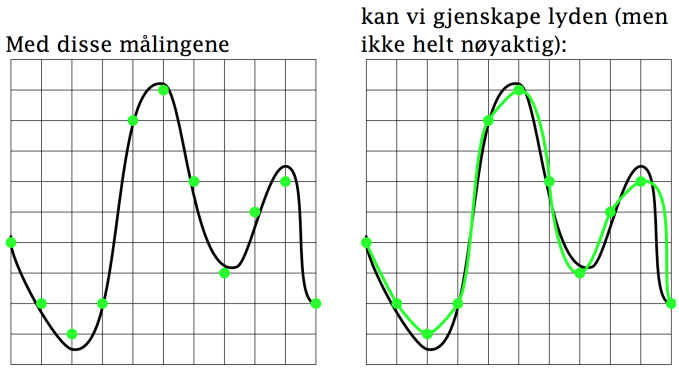
Bruk vektorgrafikk om mulig: SVG, EPS, PDF.

For fotografier bruk JPEG i så god kvalitet som mulig.

For annen rastergrafikk bruk PNG med så mange piksler som ulig.

1.6 Lyd

Lyd er bølger i luft, men de kan overføres som strøm. Vi kan lagre luden ved å måle strømmen med jevne mellomrom, men for å lagre digitalt må vi måle styrken i faste intervaller.



Kvaliteten på lyd blir da avhengig av hvor ofte vi måler og hvor mange trinn vi benytter til målingen.

1.6.1 CD

En vanlig CD har 74 minuuter spilletid med god lyd:

- 44 100 målinger ("samples") per sekund
- $2^{16} = 65\ 536$ intervaller (dvs 2 byte)
- 2 kanaler
- + 37% feilkorreksjonsdata.

En CD må derfor ha plass til 783 MB.

1.6.2 Er det mulig å spare plass?

Høyre og venstre kanal er stort sett nesten like. Det er lurere å lagre venstre kanal samt forskjellen. I stedet for å lagre hver måling med sin verdi, holder det å lagre forskjellen. Men: på grunn av feil og spoling må man av og til lagre den ekte verdien.
Enda mer plass kan vi spare om vi tar hensyn til hvordan vi mennesker hører:

- Vi kan ikke høre lyder under 20 Hz og over 20 000 Hz.
- Om vi hører en kraftig lyd med én frekvens, hører vi ikke litt svakere lyder med noe høyere frekvens.
- Etter å ha hørt en sterk lyd, hører vi dårligere en tid etterpå (inntil 0,2s).

Moderne standarder som **MP2** (brukt i DAB), **MP3**, **ACC** (brukt i DAB+, YouTube, iTunes, ...), utnytter dette og tillater til dels sterk komprimering:

Ekte CD	1410 kb/s
MP2	ca 256 kb/s
MP3	ca 160 kb/s
AAC	ca 128 kb/s

Men kvaliteten går ned om det komprimeres ennå mer.

1.7 Oppsummering

- Alt er bit i en datamaskin.
- Det finnes ulike typer tallverdier, og programmereren må velge riktig.
- Det er mange ulike tegnkodinger å forholde seg til (ennå).
- Rasterbilder og vektorbilder er nyttige til hvert sitt formål.
- Lydkoding med MP2, MP3 og AAC er blitt standard, men vi bør velge rett kvalitet.

2 Boolsk Algebra

Muliggjør design av komplekse digitale system, analyse av komplekse digitale system og forenkling av logiske uttrykk (Gir enklere fysisk implementasjon).

2.1 Binær addisjon

Prosedyren for binær addisjon er identisk med prosedyren for desimal addisjon.

Eksempel: Adder 5 og 13:

```
  1 1   1          0 5
  0 0 1 0 1       +1 3
+ 0 1 1 0 1       ----
= 1 0 0 1 0       =1 8
```

2.2 Negative binære tall

Mest vanlig representasjon: 2'er komplement

Lar mest signifikante bit være 1 for negative tall

Dette må være "avtalt" på forhånd

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

Setter minus foran et binært tall ved å invertere alle bittene og plusse på 1.

Eksempel: Finner -5

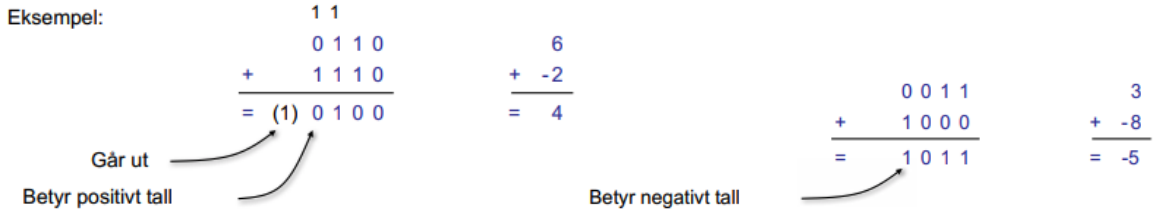
```
invertert 5:   1 0 1 0
             + 0 0 0 1
             ---------
-5:        =   1 0 1 1
```

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

2.3 Binær subtraksjon

Fremgangsmåte for tall representert ved 2'er komplement:

Adder tallene på vanlig måte.



2.4 Sannhetsverditabell

En sannhetsverditabell (eng: truth table) forteller hva sannhetsverdien til en sammensatt utsagnslogisk formel er pa bakgrunn av hvilke sannhetsverdier som er tilordnet utsagnsvariablene.

AND

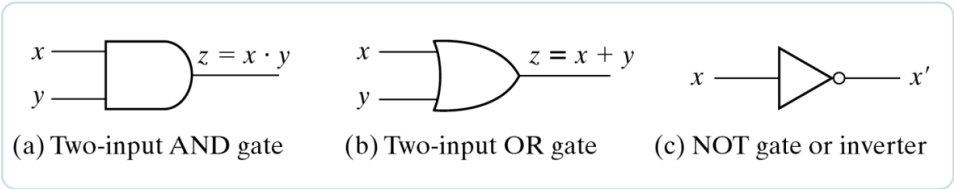
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

OR

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

NOT

X	Y
0	1
1	0



(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

NAND

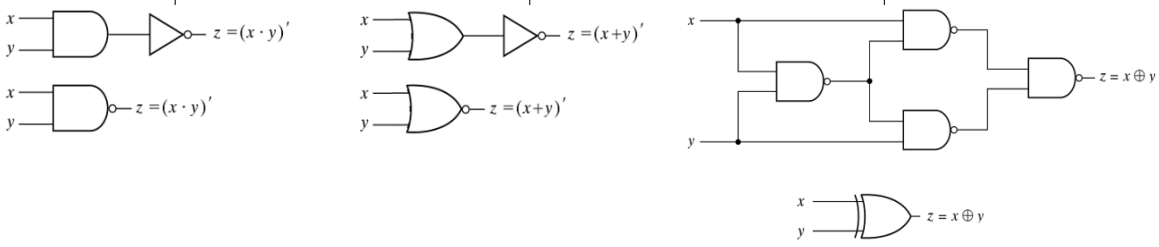
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

NOR

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

XOR

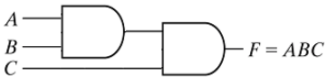
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0



2.5 Flerinputs-porter

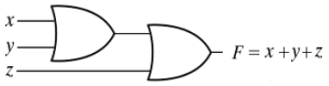
3-input AND

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



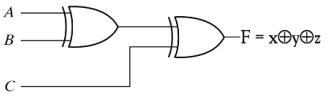
3-input OR

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



3-input XOR

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



Flerinputs-implementasjon av NAND/ NOR kan ikke lages direkte.

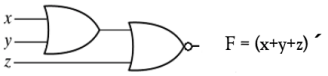
3-input NAND

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

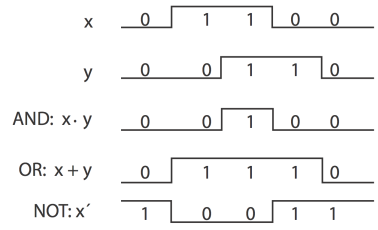
Mulig løsning

3-input NOR

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



Mulig løsning



2.6 DeMorgans teorem

$$(x \bullet y)' = x' + y'$$

$$(x + y)' = x' \bullet y'$$

På invertert form:

$$x \bullet y = (x' + y')'$$

$$x + y = (x' \bullet y')'$$

2.7 Regneregler

$x + 0 = x$	$x \bullet 1 = x$
$x + x' = 1$	$xx' = 0$
$x + y = y + x$	$xy = yx$
$x + (y+z) = (x+y) + z$	$x\,(yz) = (xy)z$
$x(y+z) = xy + xz$	$x + (yz) = (x+y)(x+z)$
$x + x = x$	$x \bullet x = x$
$x + 1 = 1$	$x \bullet 0 = 0$
$x + xy = x$	$x(x+y) = x$
$(x+y)' = x'y'$	$(xy)' = x' + y'$

2.8 Forenkling av uttrykk

$$F = x'y'z + x'yz + xy'$$

Prosedyre:

$$F = x'z(y' + y) + xy'$$
$$F = x'z \bullet 1 + xy'$$
$$F = x'z + xy'$$

Eksempel:

$F = x(x'+y)$
$F = xx'+xy$
$F = 0+xy$
$F = xy$

Eksempel:

$F = x+x'y$
$F = (x+x')(x+y)$
$F = 1(x+y)$
$F = x+y$

Eksempel:

$F = (x+y)(x+y')$
$F = x + (yy')$
$F = x + 0$
$F = x$

Eksempel:

F = xy+x'z+yz

F = xy+x'z+yz(x+x')

F = xy+x'z+xyz+x'yz

F = xy(1+z)+x'z(1+y)

F = xy+x'z

Eksempel:

F = (x+y)(x'+z)(y+z)

F = (x+y)(x'+z) Dualitet

2.8.1 Komplement av funksjon

Eksempel:

F = x'yz'+x'y'z

F' = (x'yz'+x'y'z)'

F' = (x'yz')'(x'y'z)'

F' = (x+y'+z)(x+y+z')

Eksempel:

F = x(y'z'+yz)

F' = (x(y'z'+yz))'

F' = x'+(y'z'+yz)'

F' = x'+(y'z')'(yz)'

F' = x'+(y+z)(y'+z')

Inverterer begge sider og bruker DeMorgan

2.9 Minterm

I en funksjon kan en binær variabel x opptre som x eller x̀. En funksjon kan være gitt på **"sum av produkt"** form, eks: F = xy + xȳ + x

Hvert "produktledd som inneholder **alle** variablene kalles en **minterm**. Mintermer har notasjon m_x, og det finnes 2^n mintermer, hvor n er antall variabler.

F(x,y,z) = S(m3, m6) = S(3, 6) = x̀ yz + xyz̀

For to variable finnes det 4 forskjellige mintermer: xy + xȳ + x̀ y + x̀ ȳ.

Mintermer brukes også i sannhetstabeller, summen som kommer ut er definert som mintermer og kan brukes i Karnaughdiagram for å forenkle utrykket.

2.10 Maksterm

En funksjon kan være gitt på **"produkt av sum"** form. Eks: F = (x+y)(x+ỳ)y.

Hvert "summeledd" som inneholder alle variablene kalles **maksterm**.

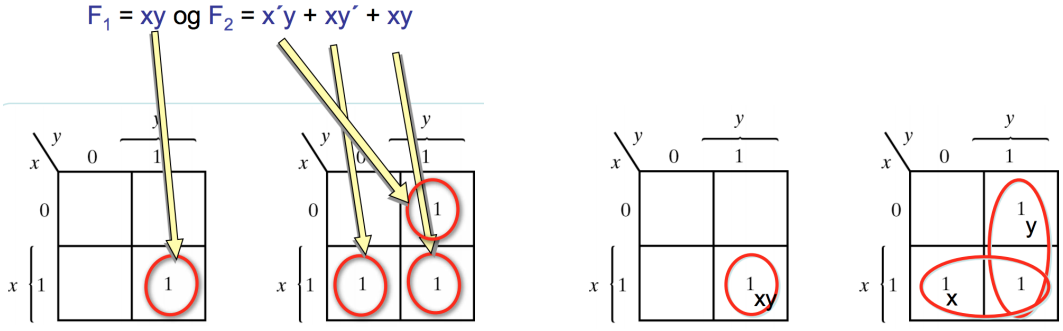
For to variable finnes det 4 forskjellige makstermer: (x+y) (x+ỳ) (x̀ +y) (x̀ + ỳ). For n variable finnes det 2^n forskjellige makstermer.

F(x,y,z) = ∏(M3, M6) = ∏(3, 6) = (x+ỳ +z̀)(x̀ +ỳ +z)

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

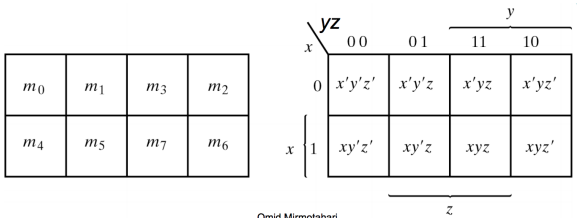
2.11 Karnaughdiagram

Karnaughdiagram er en måte å forenkle et utrykk på, dette gjøres ved hjelp av en tabell. Man starter med å legge inn tallet 1 for verdiene som er representert og lar den være blank for verdier som ikke er med i utrykket (kan også skrives som 0). Deretter grupperer man naborutene som inneholder "1" slik at vi får sammenhengende rektangler, velg så store grupper som mulig, men det må være en potens av 2 (2^n).



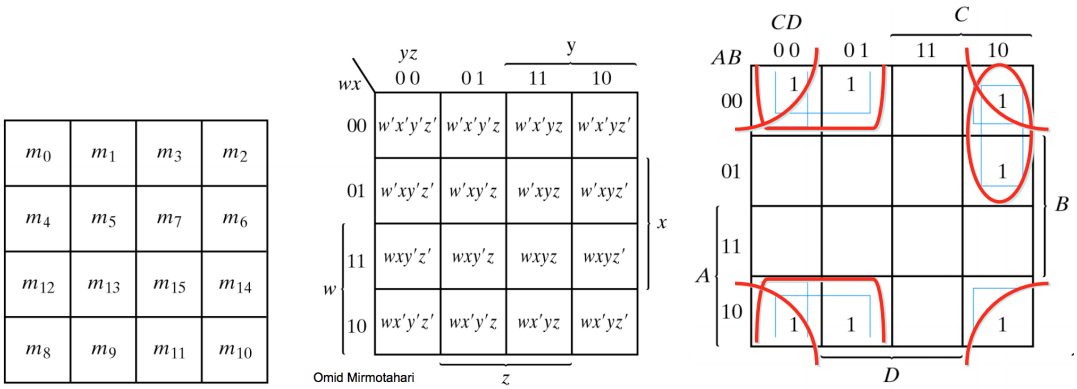
$F_1 = xy$ $F_2 = x + y$

2.11.1 3-variable Karnaugh



Mintermene plasseres slik at kun 1 variabel varierer i mellom hver vannrette/loddrette naborute.

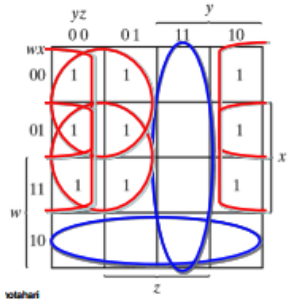
2.11.2 4-variable Karnaugh



Mintermene plasseres slik at kun 1 variabel varierer i mellom hver vannrette/loddrette naborute. Man kan også gruppere rektangler med ytterkantene av diagrammet, slik at man kan finne par rundt.

2.11.3 Utlesning av ”0”ere

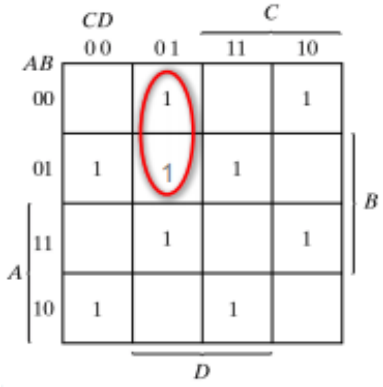
Ved å lese ut de tomme rutene (”0”erne) fra diagrammet får man F. Dette kan noen ganger gi en enklere funksjon, eksempel:
F = yz + wx`
F = (yz + wx`)`
Hadde vi lest ut ”1”ere ville vi fått:
F = xy` + w` y` + w` z + xz`.

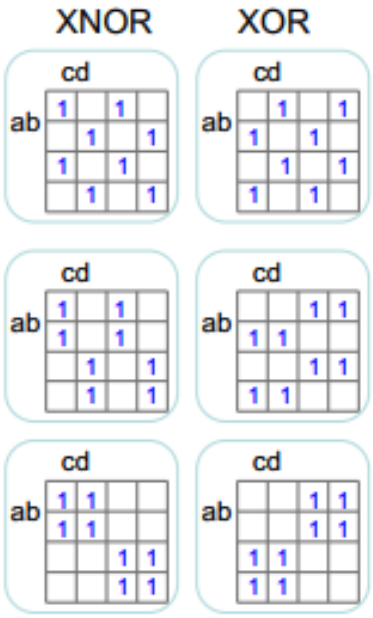


2.11.4 Utlesning av XOR

XOR funksjonen dekker maksimalt ”uheldige” ”1” er plasseringer i diagrammet. Har man XOR porter til rådighet bruker man disse. XOR representerer den odde funksjonen, det vil si de kombinasjonene av ingangene hvor det er odde antal 1’ere. XNOR er den inverterte funksjonen av XOR.

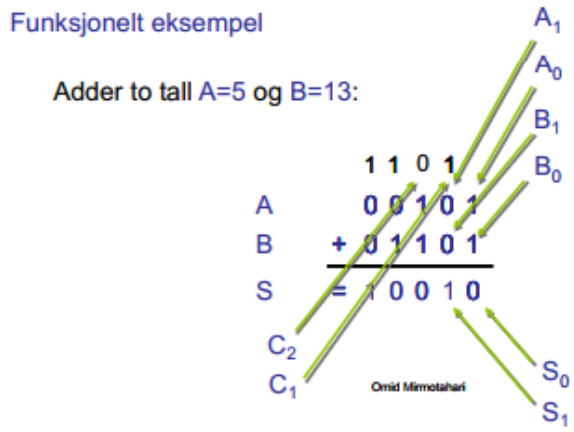
F = A⊕B⊕C⊕D + A` C` D



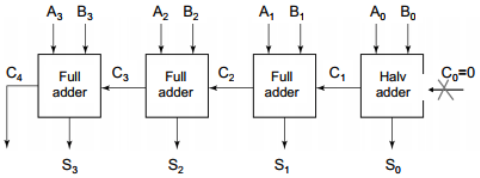


2.12 Binær adder

En av de mest brukte digitale kretsene, finees i pikroprosesser ALU / Xbox / mikserbord / digitalt kommunikasjonsutstyr / AD-DA omformere osv. Basis for addisjon, subtraksjon, multiplikasjon, divisjon og mange andre matematiske operasjoner.

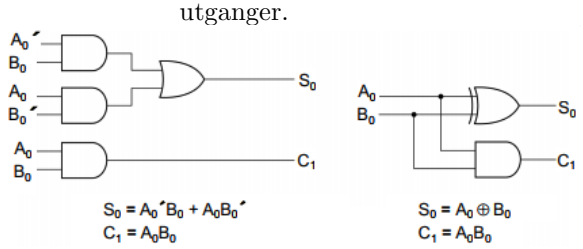


Et adder system, kan bli delt inn i to underkategorier:
Halvadder: Tar ikke mente inn
Fulladder: Tar mente inn



2.12.1 Halvadder (ingen mente inn)

Adderer sammen de to minst signifikante bittene A_0 og B_0. Elementet har 2 innganger og 2 utganger.



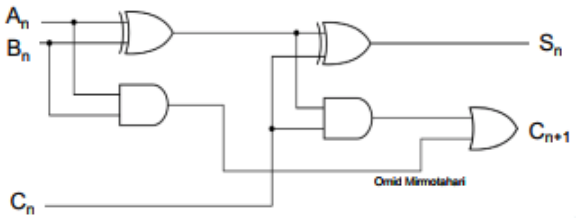
Sannhetstabell

A_0	B_0	S_0	C_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$S_0 = A_0\grave{} \; B_0 + A_0 \; B_0\grave{} = A_0 \oplus B_0$
$C_1 = A_0 \; B_0$

2.12.2 Fulladder (mente inn)

Adderer sammen bit A_n, B_n med eventuell mente inn. Elementet har 3 innganger og 2 utganger.



$$S_n = (A_n \oplus B_n) \oplus C_n$$
$$C_{n+1} = A_n\ B_n + (A_n \oplus B_n) \oplus C_n$$

Sannhetstabell

A_n	B_n	C_n	S_n	C_{n+1}
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

$S_0 = A_0\grave{}\ B_0 + A_0\ B_0\grave{} = A_0 \oplus B_0$
$C_1 = A_0\ B_0$

2.13 Komparator

Komparator sammenligner to tall A og B, og den har 3 utganger: A=B, A¡B og A¡B.

Utgang A>B slår til hvis:

(A_3>B_3) eller

(A_2>B_2 og A_3=B_3) eller

(A_1>B_1 og A_2=B_2 og A_3=B_3) eller

(A_0>B_0 og A_1=B_1 og A_2=B_2 og A_3=B_3)

Kan skrives:

$(A_3B_3\acute{}) + (A_2B_2\acute{})\ (A_3 \oplus B_3)\acute{} + (A_1B_1\acute{})\ (A_2 \oplus B_2)\acute{}(A_3 \oplus B_3)\acute{} +$
$(A_0B_0\acute{})(A_1 \oplus B_1)\acute{}(A_2 \oplus B_2)\acute{}(A_3 \oplus B_3)\acute{}$

Utgang A<B slår til hvis:

(A_3<B_3) eller

(A_2<B_2 og A_3=B_3) eller

(A_1<B_1 og A_2=B_2 og A_3=B_3) eller

(A_0<B_0 og A_1=B_1 og A_2=B_2 og A_3=B_3)

Kan skrives:

$(A_3\acute{}B_3) + (A_2\acute{}B_2)\ (A_3 \oplus B_3)\acute{} + (A_1\acute{}B_1)\ (A_2 \oplus B_2)\acute{}(A_3 \oplus B_3)\acute{} +$
$(A_0\acute{}B_0)(A_1 \oplus B_1)\acute{}(A_2 \oplus B_2)\acute{}(A_3 \oplus B_3)\acute{}$

Utgang A=B

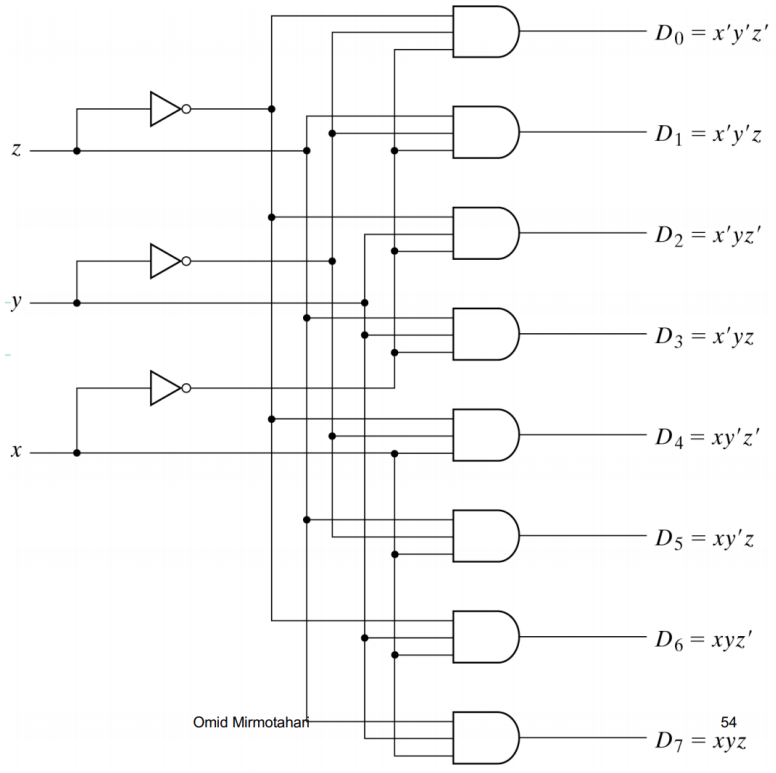
Slår til hvis A_0=B_0 og A_1=B_1 og A_2=B_2 og A_3=B_3

Kan skrives: $(A_0 \oplus B_0)\acute{}(A_1 \oplus B_1)\acute{}(A_2 \oplus B_2)\acute{}(A_3 \oplus B_3)\acute{}$

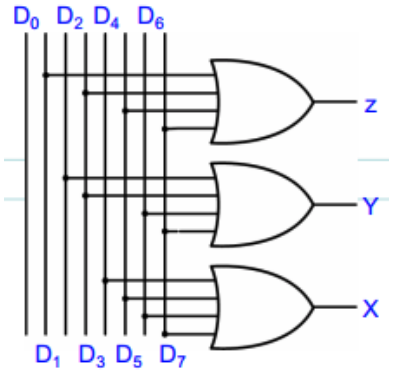
2.14 Dekoder

Dekoder tar inn et binært ord, gir ut alle mintermer. En dekoder kan generere generelle logiske funksjoner direkte fra mintermene på utgangen.

x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

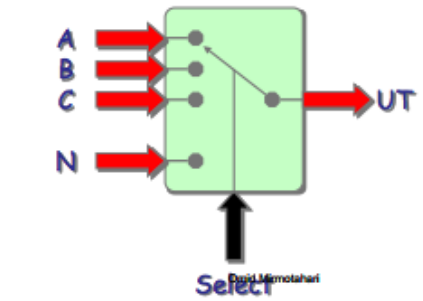


2.15 Enkoder

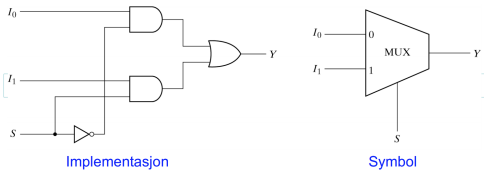


D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

2.16 Multiplekser

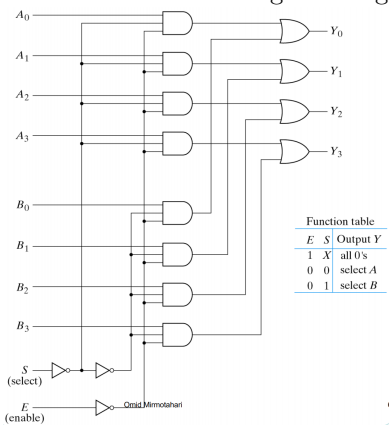


Eksempel: 2-1 MUX



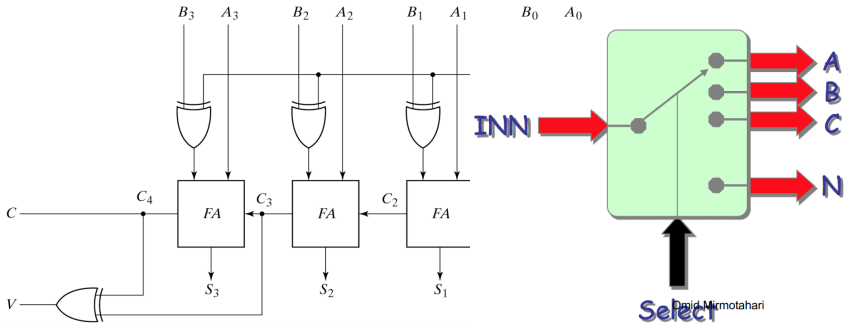
Implementasjon Symbol

Multiplekser (MUX), velger hvilke innganger som slippes ut. Hver inngang kan bestå av ett eller flere bit. Dette er altså en enhet som velger en av mange input signaler. Multiplekser brukes som regel til å øke mengden av data som blir sendt over nettverket innenfor en gitt tid og båndbredde.



2.17 Demultiplekser

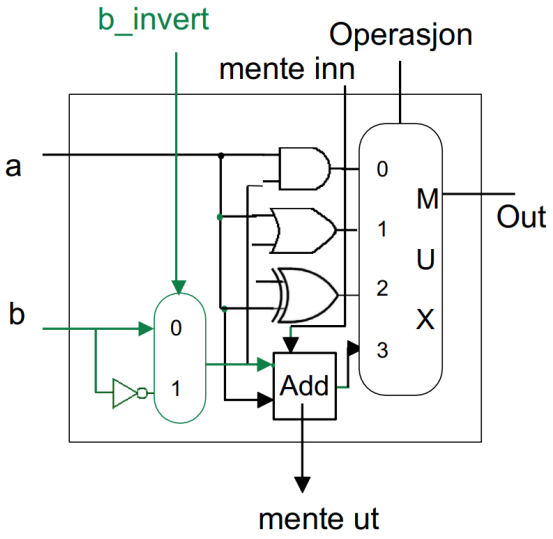
Motsatt av multiplekser, denne tar en inn og velger en av mange utganger.



M=0: adder / M=1: subtraktor / V: overflow bit

2.18 Aritmetisk logisk enhet (ALU)

Dette er en elektronisk krets som utfører aritmetiske og logiske operasjoner, CPU-er og GPU-er inneholder velid kompliserte og gjerne flere ALU-er. Eksempler på operasjoner: addisjon, subtraksjon, AND, OR, XOR osv. For å bygge en ALU trenger man følgende byggeblokker: fulladder, multiplekser, AND, OR og NOT.

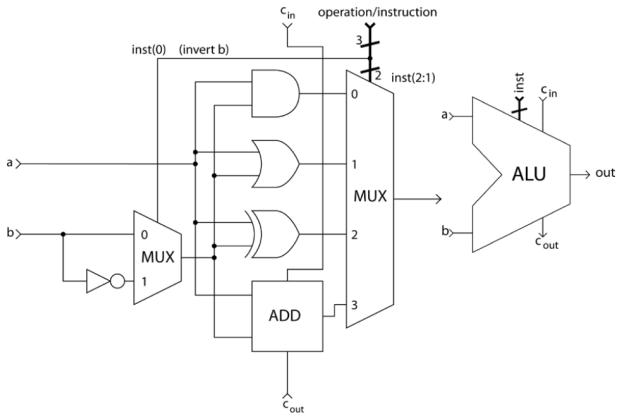


Operasjon = a2a1a0
a2 = b_invert
a1 = MSB MUX
a0 = LSB MUX

Operasjon	Out
000	ab
001	a+b
010	a XOR b
011	a SUM b
100	ab´
101	a+b´
110	a XOR b´
111	a SUB b

1-bit ALU

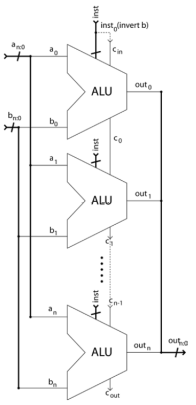
inst = a2a1a0
Inst(2) = a2 invert b
Inst(1) = a1 MSB MUX
Inst(0) = a0 LSB MUX

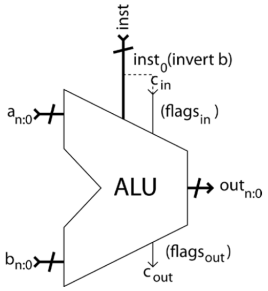


inst	computation
000	a · b
001	a · b´
010	a + b
011	a + b´
100	a ⊕ b
101	a ⊕ b´
110	a + b
111	a - b

ALUer kan designes til å håndtere flere funksjoner per trinn og mer komplekse.
Alternativt kan man bruke software aktivt til å designe slik at en ALU kan utføre komplekse operasjoner over flere trinn.
TRADE-OFF: hvor skal man plassere kompleksiteten, i harware eller software.
Ulemper med å sette kompleksitet i hardware er høyere kostnader i for eksempel strømforbruk, areal og produksjonskost.
ALU designet spiller en stor rolle med hensyn på CPU sin ytelse, da det mest komplekse operasjonen setter maksimal klokkefrekvens.

N-bit ALU





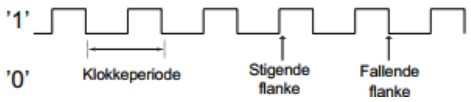
3 Sekvensiell Logikk

3.1 Definisjoner

Kombinatorisk logikk:
Utgangsverdiene er entydig gitt av nåværende kombinasjon av inngangsverdier.

Sekvensiell logikk: ⇒
Inneholder hukommelse (låsekretser). Utgangsverdiene er gitt av nåværende kombination av inngangsverdier, samt sekvensen (tidligere inngangs-/utgangsverdier).

- I **sykrone** sekvensielle kretser skjer endringen(e) i output **samtidig** med endringen i et **klokkesignal**.
- I **asynkrone** sekvensielle kretser skjer endringen(e) i output **uten** noe **klokkesignal**.
- Nesten alle kretser er synkrone.
- Et klokkesignal er et digitalt signal som veksler mellom '0' og '1' med fast takt.



3.1.1 Synkron logikk

I større digitale system har man behov for å synkronisere dataflyten. Til dette bruker vi et globalt klokkesignal. Uten global synkroniskering ville det vært total kaos.
Den omvendte av klokkeperioden kalles (klokke)frekvensen, altså

$$frekvens = \frac{1}{klokkeperioden}$$

Ønsker så høy klokkefrekvens som mulig, fordi hver enkelt operasjon da bruker så kort tid som mulig.

Maksimal klokkefrekvens bestemmes av flere faktorer, blant annet:

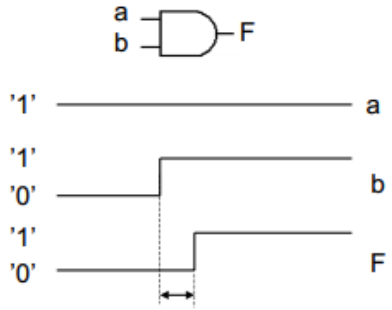
- Lengde på signalveiene
- Last

- Forsinkelse gjennom porter (delay)
- Teknologi.

NB: Hastighet er ikke direkte proporsjonal med klokkefrekvens.

3.2 Portforsinkelse / tidsforsinkelse

Logisk dybde: Antall porter et signal passerer fra inngang til utgang. Ved å redusere logisk dybde reduseres forsinkelsen gjennom kretsen.



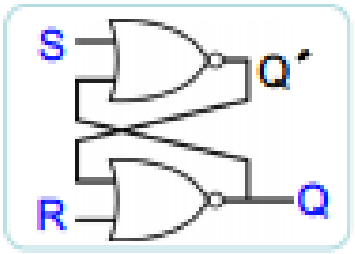
3.3 Latch

3.3.1 SR-latch - funksjonell beskrivelse

1. Kretsen skal sette Q til "1" hvis den får "1" på inngang S. Når inngang S går tilbake til "0" skal Q forbli på "1".
2. Kretsen skal resette Q til "0" når den får "1" på inngang R. Når inngang R går tilbake til "0" skal Q forbli på "0".
3. Tilstanden "1" på både S og R brukes normalt ikke.

SR

S R	Q
0 0	låst
0 1	0
1 0	1
1 1	0



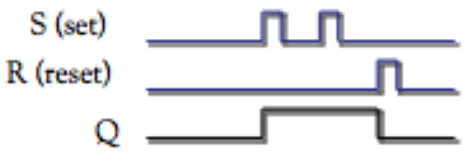
Øvre NOR

S Q	Q'
0 0	1
0 1	0
1 0	0
1 1	0

Nedre NOR

Q' R	Q
0 0	1
0 1	0
1 0	0
1 1	0

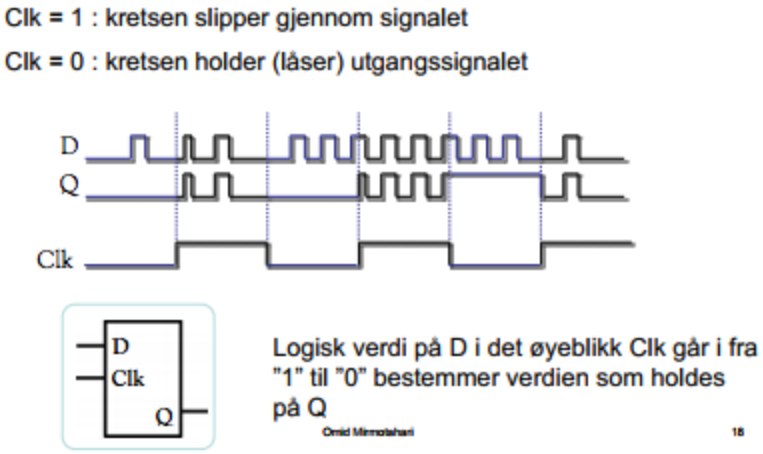
Signalet Q' er ikke invertert av Q for tilstand S=1, R=1



3.3.2 D-Latch

Dataflyten gjennom en D-latch kontrolleres av et klokkesignal.

1. Slipper gjennom et digital signal så lenge klokkeinngangen er “1” (transparent).
2. I det øyeblikket klokkeinngangen går fra “1” til “0” låser utgangen seg på sin nåværende verdi. Forandringer på inngangen vil ikke påvirke utgangsverdien så lenge klokkesignalet er “0”.



3.4 Flip-Flop

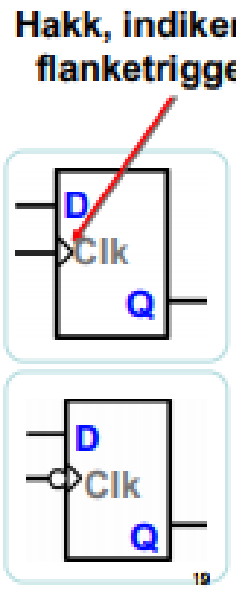
Flip-Flop’er kommer i to varianter:

- Positiv flanketrigget

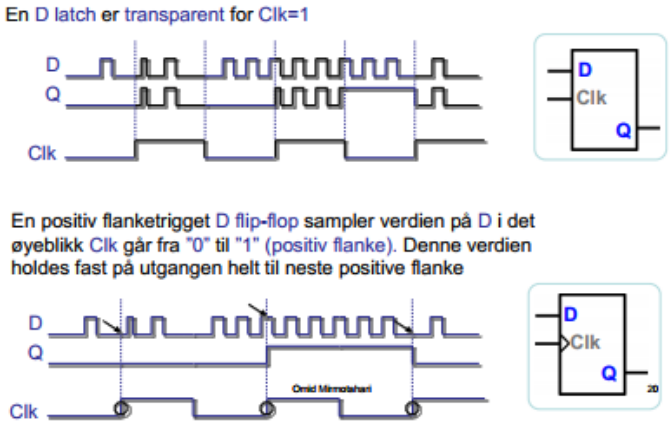
- Negativ flanketrigget

På en **positiv** flanketrigget Flip-Flop kan utgangen kun skifte verdi i det øyeblikk klokkesignalet går fra ”0” til ”1”.

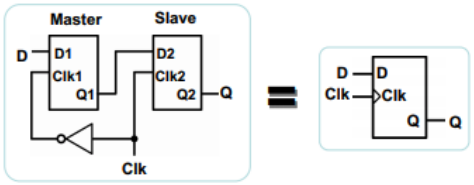
På en **negativ** flanketrigget Flip-Flop kan utgangen kun skifte verdi i det øyeblikk klokkesignalet går fra ”1” til ”0”.



3.4.1 D-Flip-Flop



En positiv flanketrigget D flip-flop kan lages av to D-latcher (Master-Slave).



Under Clk=0 er første D latch (master) transparent

Under Clk=1 er siste D latch (slave) transparent

3.4.2 JK Flip-Flop

En JK flip-flop har følgende egenskaper:

J=0, K=0: Utgang låst

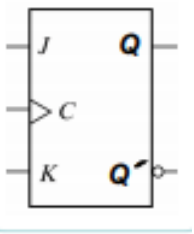
J=0, K=1: Resetter utgang til ”0”

J=1, K=0: Setter utgang til ”1”

J=1, K=1: Inverterer utgang Q → Q'

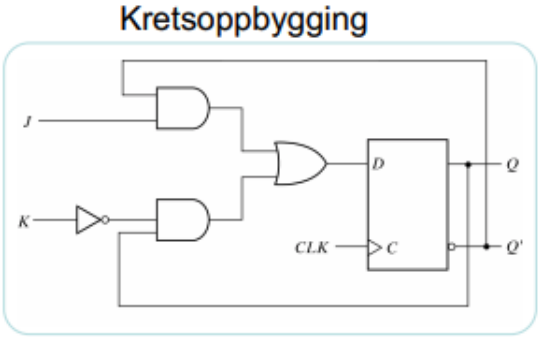
Utgangen kan kun forandre verdi på stigende klokkeflanke.

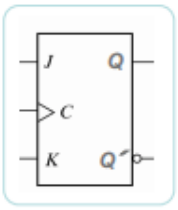
En JK flip-flop er den mest generelle flip-floppen vi har.



J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

Q(t+1) = JQ'(t) + K'Q(t)





3.4.3 T Flip-Flop

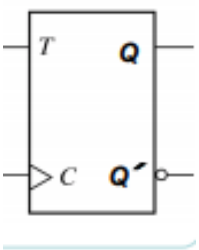
En T flip-flop har følgende egenskaper:

T=0: Utgang låst

T=1: Inverterer utgang Q → Q̄

Utgangen kan kun forandre verdi på stigende klokkeflanke.

Det er lett å lage tellere av T flip-flop'er.



T	Q(t+1)
0	Q(t)
1	Q´(t)

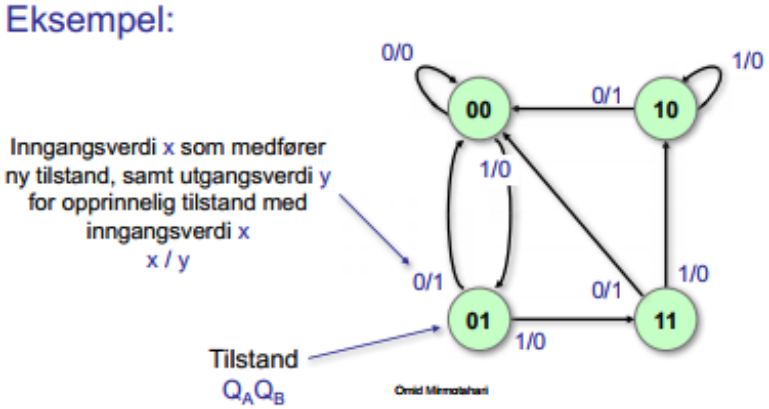
$Q(t+1) = T \oplus Q(t)$

3.5 Tilstandsmaskin

En tilstandsmaskin er et sekvensielt system som gjennomløper et sett med tilstander styrt av verdiene på inngangssignalene. Tilstanden systemet befinner seg i, pluss evt. inngangsverdier bestemmer utgangsverdiene. Tilstandsmaskins-konseptet gir en enkel og oversiktlig måte å designe avanserte system på.

3.5.1 Tilstandsdiagram

Tilstandsdiagram = grafisk illustrasjon av egenskapene til en tilstandsmaskin.



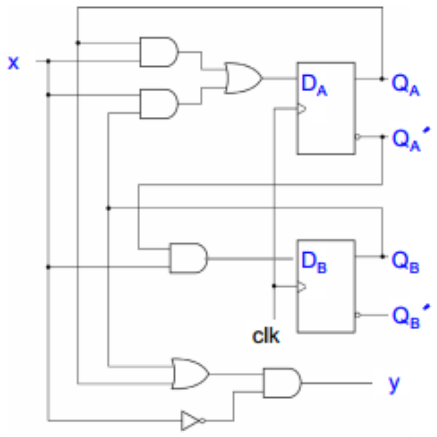
3.5.2 Tilstandstabell

Tilstandstabell = sannhetstabell for tilstandsmaskin.

Eksempel: En inngang, en utgang og 2 stk. D flip-flops

Nåværende tilstand		Inngang	Neste tilstand		Utgang for nåværende tilstand
Q_A	Q_B	x	Q_A	Q_B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Tilstandsmaskin der utgang y er en funksjon av tilstanden gitt av verdiene til Q_A og Q_B, samt inngangen x



4 Datamaskinarkitektur

4.1 Data- og instruksjonsbus

- Bus er kommunikasjonskanalen mellom registere, funksjonelle enheter (ALU), minne og I/O enheter.
- Bus kan deles mellom flere enheter, men kun en som kan sende av gangen.
- I en von Neumann arkitektur er det en bus mellom minne og CPU som skal både overføre instruksjon og data, dette vil da være flaskehalsen.
- Internt i en CPU er den en eller flere bus(er) som overfører data mellom interne registere.

4.1.1 Oppsummering av én-sykel implementasjon

Høy effektivitet oppnås gjennom:

1. **Alle instruksjoner bruker én klokkesykel**
   - Klokkeperioden må være like lang som den lengste forsinkelsen for enhver instruksjon gjennom data-path'en
     - Resultat: Raskere instruksjoner tvinges til å bruke mer tid enn nødvendig
     - Mulig løsning: Klokke med variabel periode
     - Vurdering: Vanskelig å implementere
2. **Operasjoner utføres samtidig ved å bruke flere identiske komponenter**
   - Trenger flere like komponenter som egentlig gjør samme jobb.
     - Resultat: Trenger mer plass på CPU'en og øker effektforbruket
     - Mulig løsning: Endre kontroll-logikk slik at samme komponent kan brukes til flere oppgaver.
     - Vurdering: Vanskelig, umulig eller lite effektivt hvis man krever at alle instruksjoner skal ta én klokkesykel.

3. **Instruksjonsminne og dataminne aksesseres samtidig**

   - Må kunne aksessere to forskjellig lokasjoner i samme minne-enhet samtidig.
     - Resultat: Må enten ha 2 separate minne-enheter, eller kunne adressere flere ord samtidig i én minne-enhet.
     - Mulig løsning: Operere med en minne-enhet for program, og en annen for data.
     - Vurdering: Lite fleksibelt ved blant annet dynamisk data-allokering, og mer plasskrevende (doble adressebusser osv).

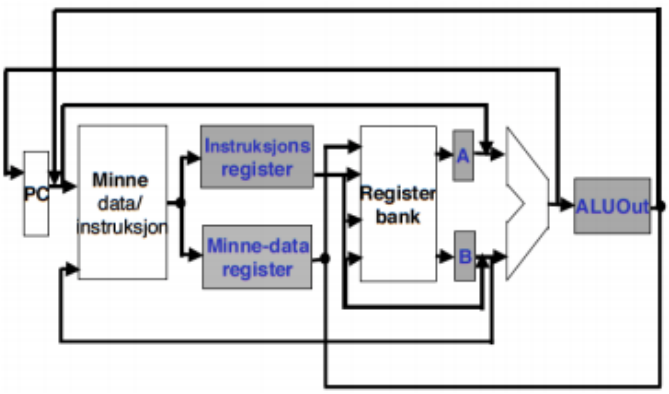
4.1.2 Forbedring av én-sykel designet

Innfører to nye strategier for å øke hastighet:

1. Multicycle
2. Pipelining

Multicucle

- Multi-cycle bruker mer enn én klokkesykel per instruksjon hvis nødvendig
- De ulike trinnene i en instruksjon kan dele samme komponent
- Trenger ikke separat data- og instruksjonsminne
- Trenger ekstra registre for å mellomlagre data



4.2 Pipelining

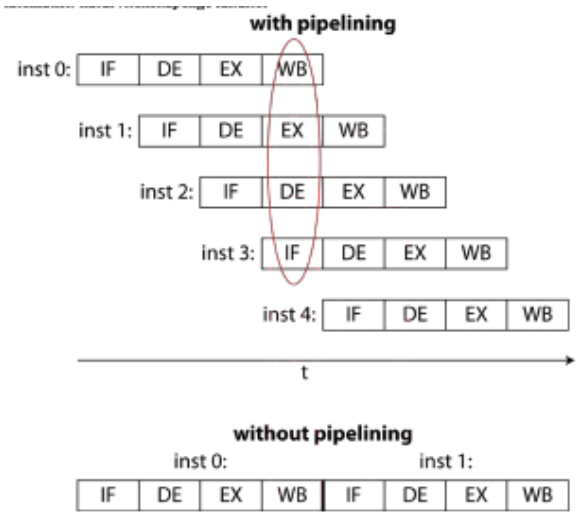
- Innfører samlebåndsprinsipp for eksekvering av instruksjoner
- Hver instruksjon må splittes opp i uavhengige deler (subinstruksjoner) som utføres etter hverandre
- Hver subinstruksjon kan utføres uavhengig av de andre subinstruksjonene
- Neste instruksjon settes igang før forrige instruksjon er helt ferdig.
- **NB!!** Hver instruksjon tar like lang tid å utføre, men prosessoren utfører flere instruksjoner i et gitt tidsrom!

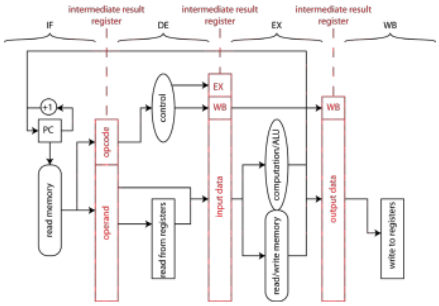
4.2.1 Pipelining instruksjonssett

Eksempelvis kan vi ha følgende subinstruksjoner i en pipeline

- IF: Instruction fetch (get the instruction)
- DE: decode and load (from a register)
- EX: Execute
- WB: Write back (write the result to a register)

PS! Read og write fra register/MEM kan gjøres i hver sin halvdel av en sykel (1/2 klokkeperiode)





4.2.2 Speed-up

Det å ha en 4 trinns pipeline betyr ikke at man får 4 ganger raskere prosessering. Det går alltid noe tid bort til administrering av instruksjoner.

Effektiv speed-up

En k-trinns pipeline som trenger én klokkesykel pr trinn med eksekverering av n instruksjoner vil ha en speed-up på:

$$\frac{kn}{k+n-1}$$

For veldig store programmer så vil vi nærme oss k.

4.3 Komplikasjoner ved pipelining - Hazards

- Til enhver tid kan det være subinstruksjoner fra opptil 4 instruksjoner i en pipeline
- Noen ganger er ikke alle subinstruksjonene gyldige
- Neste instruksjon kan ikke eksekveres rett etter hvis hopp-betingelsen slår til
- En slik situasjon kalles HAZARD. Vi har tre typer:

1. **Resource hazard**

   Kan oppstå hvis to subinstruksjoner i pipelinen ønsker å aksessere samme ressurs (eks minne).

   - Løsning 1: Designe slik at de ikke oppstår (!)
   - Løsning 2: Stoppe (STALL) pipelinen lenge nok til resurssen kan aksesseres sekvensielt. (lese inne en NOP?)
   - Løsning 3: Bruke lokale register som er organisert i en REGISTER FILE
   - Løsning 4: Bruke Harvard arkitekturen som har 2 separate minner for data og instruksjon.

   Andre resources kan også gi harzards, litt avhengig av hvordan CPU arkitekturen er bygget opp: minne, cache, register files, busser, ALU, osv.

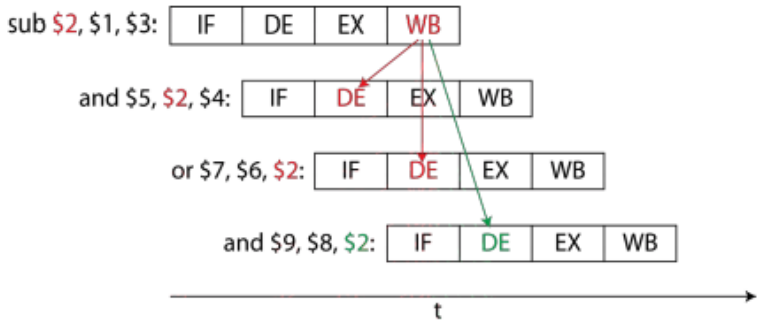
2. **Data hazard**

   Data hazard oppstår fordi to forskjellige instruksjoner trenger å aksessere samme data samtidig. - Trenger resultat fra forrige instruksjon før denne har produsert gyldig svar.

   – Løsning 1: Detektere avhengigheten i IF-stadiet av pipelinen for første instruksjon og stoppe (STALL) neste instruksjons EX-stadie til WB-stadiet for første instruksjon er ferdig.

   – Løsning 2: Snu om på rekkefølgen av instruksjonene slik at man ikke er avhengig av instruksjon rett før i pipelinen.

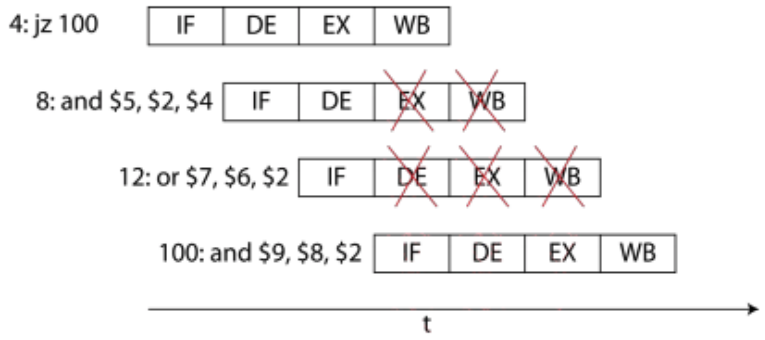
   – Løsning 3: Ha en snarvei (FORWARDING) i pipelinen, i dette tilfelle en direkte datapath som aktiviseres ved en hazard.



Løsning 3 er bedre fordi alt skjer i hardware og ikke i kompilatoren. Forwarding benyttes også mellom andre enheter i datapathen, feks mellom utgangen av MEM og inngangen til ALU.

3. Control Hazard

   I utgangspunktet har vi tenkt en pipeline som innhenter neste instruksjoner fortløpende. Men problemet oppstår når vi får en JUMP instruksjon. Da må vi tømme pipelinen for andre instruksjoner og lese inn den nye.



   - Løsning 1: Stoppe (STALL) ikke hent inn andre instruksjoner før det er helt klart at man ikke skal hoppe.
   - Løsning 2: Prøv å forutsi om man skal hoppe, eksekver som vanlig ved ikke-hopp, stop pipelinen som løsning 1 ved hopp.
   - Løsning 3: Dobling av hardware for deler av pipelinen. Dette for å kunne ha to parallelle pipelines som kan inneholde begge instruksjonsadressene. Når det er klart om instruksjonen er en hopp så "flushes" den andre.

5 Minnehierarki

5.1 Teknikker for hastighetsøkning

- Pipelining er viktig for å øke antall instruksjoner som utføres pr. sekund
- Andre teknikker som øker hastighet:
  - **Flere CPU i en maskin (multiprosessor)**
    - Ide: Istedenfor å la én CPU utføre instruksjoner, lar man flere CPUer samarbeide om den samme jobben.
    - Gir en teoretisk hastighetsøkning direkte proporsjonal med antall ekstra CPUer: det går n ganger raskere med n CPUer enn med én CPU.
    - I praksis ikke mulig av flere årsaker:
      - Ikke alltid mulig å dele opp et problem i like store deler som kan løses uavhengig av hverandre.
      - Det kreves administrasjon og koordinering av programeksekveringen før, under og etter at jobben er fordelt på de ulike CPUene, dvs ekstra overhead.
    - Endel applikasjoner egner seg for lastdeling, f.eks server- applikasjoner, dvs enkelt å parallellisere oppgavene.
  - **Økt klokkefrekvens og stor ordlenge på instruksjoner**
    - Økt klokkehastighet reduserer tiden hver enkelt instruksjon tar (Resultatet av teknologisk utvikling.)
    - Hastigheten til alle deler av en datamaskin øker ikke like raskt:
      - Interne data/adressebusser
      - Nettverk
    - Klokkehastigheten til en CPU øker mye raskere enn lese/skrivehastigheten til hukommelsen, mao: Effektive hastighets-forbedring blir langt mindre.
    - Bredere datapath og ordlengde gjør det mulig å behandle større tall i en operasjon og å lese/skrive mer data fra hukommelsen i én operasjon.

        * Det er en øvre grense for hvor mange bit som kan behandles i en operasjon

    - **Raskere disker**

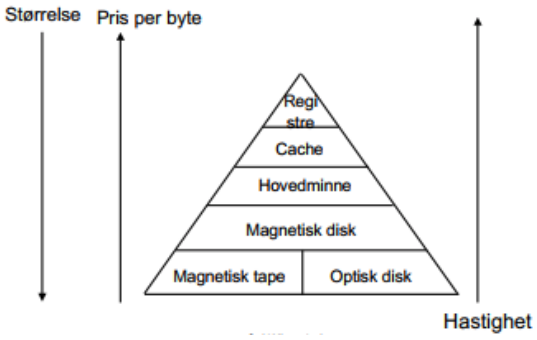
        * Programmer trenger å aksessere disker av og til ved f.eks innlesning og skriving av data, ved oppstart og virtuell hukommelse.
        * En harddisk ca 100 000 ganger langsommere enn en CPU, dvs at en CPU kan gjøre 100 000 instruksjoner, mens en harddsik gjør én lese/skriveoperasjon.
        * Teknikker finnes for å lese store blokker av data ad gangen, prøve å gjette hvilke data som skal leses neste gang og lagre disse i hurtigminne, etc
        * Harddisker blir både raskere og får mer kapasitet, men allikevel vil hyppig diskaksess være et betydelig problem hvis hastighet er viktig.

    - **Hurtigere og større RAM**

        * Ønsker ubegrenset med minne som er like raskt som CPU'en.
        * I praksis må man velge ut fra flere hensyn.
        * Tre parametre klassifiserer minnetyper:
            · Antall byte tilgjengelig per enhet
            · Pris per byte
            · Aksesshastighet

- Noen av disse er ren forbedring i teknologien, andre er forbedringer av selve arkitekturen.

5.2 Minnehierarki



5.2.1 Bruksområde

- Register - Intern kladdeblokk for CPU'en med rask aksess til innholdet.
- Cache - Hurtig mellomlager for både instruksjoner og data for å jevne ut hastighetsforskjellen mellom CPU'en og hurtigminnet.
- RAM - Buffer mellom eksternt lagringsmedium og CPU'en med rask både lese- og skrive aksess.
- SSD/Flash/Harddisk - Høykapasitetsmedium for program/data

5.2.2 Lagringskapasitet

- Register - Integrert på CPU'en, relativt få (32-128 stykker)
- Cache - Mellomlager internt (L1) eller i nærheten (L2,L3) av CPU'en, typiske kapasiteter er fra 10 KiloByte (L1) til 1 MegaByte (L2) og flere MegaByte (L3).
- RAM - Internt på hovedkortet i nærheten av CPU'en, størrelser opptil flere GigaByte.
- SSD/Flash/Harddisk - Ekstern eller intern lagringsenhet i maskinen med kapasitet opptil flere TeraByte.

5.2.3 Aksesshastighet

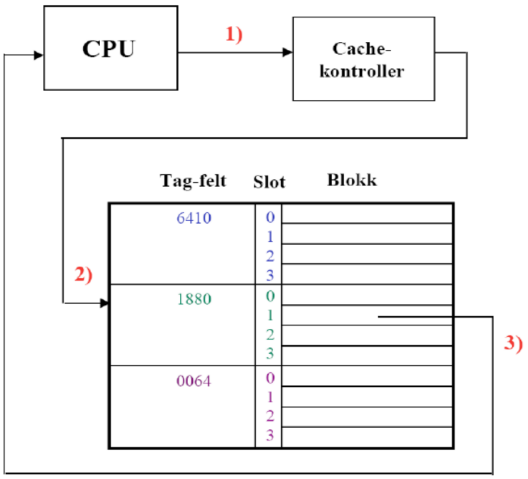
Registers	< 1ns	≈ 100 Byte
L1 (på CPU) cache	≈ 1ns	≈ 10 KB
L2,L3 (utenfor CPU) cache	2-10ns	≈ 1 MB
Hovedminne (RAM)	20-100ns	≈ 1 GB
SSD/Flash	100ns-1us	≈ 1 TB
Harddisk	1ms	≈ 1 TB

5.3 Cache

- Cache forbedrer von Neumann arkitekturens flaskehals med hensyn på minneaksessering.
- Cache ligger nærme CPU for å øke hastighet, men små størrelser for å redusere produksjonskost og derfor er det et trade-off diskusjon mellom størrelse og hastighet.
- Cache inneholder en kopi av et subsett av hurtigminne (RAM)
- CPU'en henter data og/eller instruksjoner fra cache istedenfor RAM.
- Siden cache er mindre enn RAM, må det bestemmes hvilke data/instruksjoner som skal ligge i cache
- Cache basere seg på at instruksjoner/data aksesseres:
  - Nær hverandre i tid
  - Nær hverandre i rom
- Hvis data/instruksjon CPU'en trenger ligger i cache, kalles det for cache hit
- Hvis data/instruksjon CPU'en trenger ikke ligger i cache, kalles det for cache miss.
- En essensiell del for utnyttelsen av cache er å kunne raskt sjekke for hit eller miss.

5.3.1 Cache hit

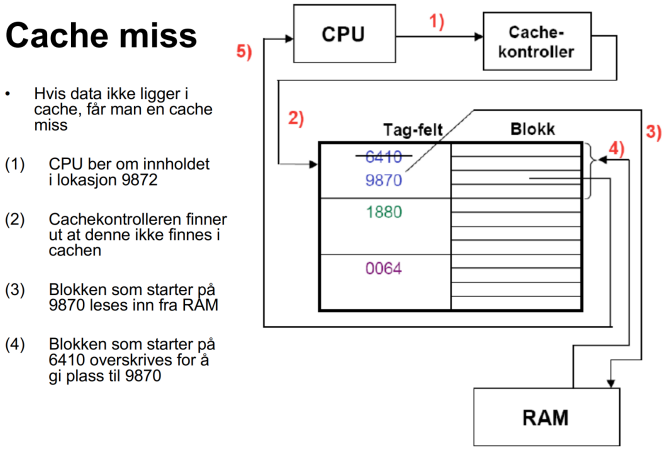
- Når en prosessor skal hente en instruksjon eller lese/skrive data, vet den ikke om den skal hente fra RAM eller cache. Hvis det prosessoren ber om ligger i cache, kalles det en cache hit



5.3.2 Cache miss

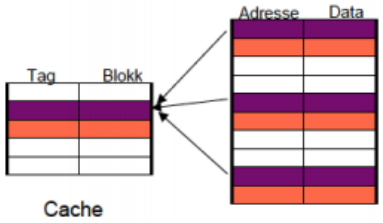
- Ved cache miss må data/instruksjon først kopieres fra hurtigminnet over i cache før CPU'en kan bruke det.

- Cache fylles fra hurtigminnet første gangen en bestemt lokasjon blir referert.



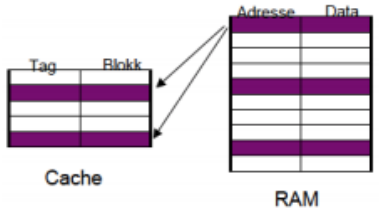
5.3.3 Mapping strategier for cache

1. **Direct-mapped cache**



- En bestemt blokk fra RAM kan bare plasseres i en bestemt blokk i cache. Flere RAM-lokasjoner må "konkurrere" om samme blokk i cache.
  - Fordel: Lett å sjekke om riktig blokk finnes i cache: Sjekk kun ett tag-felt og se om denne inneholder blokken man leter etter.
  - Ulempe: Høy miss-rate (selvom det er ledig plass andre steder, kan en blokk kun plasseres ett bestemt sted)

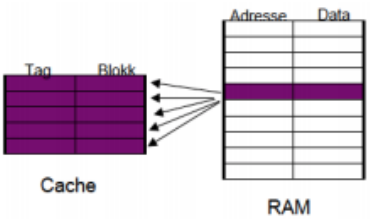
2. **Set-associative cache**



- En bestemt blokk fra RAM kan plasseres i et begrenset antall blokk-lokasjoner i cache.
  - Fordel: Lett å sjekke om riktig blokk finnes i cache: Søk gjennom et begrensset antall tag-felt og se om blokken man leter finnes i cache.
  - Ulempe: Lenger søketid enn ved direkte-mappet (med mindre man søker i parallell gjennom tagfeltene)

3. **Full associative cache**

   - En bestemt blokk fra RAM kan plasseres hvor som helst i cache.
     - Fordel: Cachen utnyttes meget godt, har minst sjanse for cache miss av alle de tre metodene.
     - Ulempe: Søket for å finne en blokk kan bli tidskrevende. Man kan lage mekanismer for å forenkle søk, f.eks. hashing, men dette kompliserer cache-kontrolleren og krever ekstra hardware.



Metodene skiller seg fra hverandre ved:

- Hvor enkle de er å implementere
- Hvor raskt det går å finne en blokk
- Hvor god utnyttelse man får av cache
- Hvor ofte man får en cache miss

5.4 Write Strategier

Write hit

- Det skal skrives til hukommelsen og blokken med lokasjonen det skal skrives til ligger i cache
- Data skrives til riktig lokasjon i blokken i cache

Write miss

- Det skal skrives til hukommelsen og blokken med lokasjonen det skal skrives til ligger ikke i cache

- Cache-kontrolleren må bestemme
  - Hvilken blokk i cache som kan/skal overskrives
  - Kopiere inn riktig blokk fra RAM til cache
  - Besørge skriving til riktig lokasjon i cache

Ved begge write-operasjoner vil ikke innholdet i RAM og innholdet i kopien av blokkene i cache være identiske → **ikke OK**

5.4.1 Write inkoherens

Hvis det bare skrives til cache og ikke RAM, vil ikke RAM inneholde gyldige data
Hvis en blokk det er skrevet til kastes ut fordi cache er full, mister man data
To strategier benyttes for å håndtere write- operasjonen korrekt:

1. Write-through
2. Write-back

5.4.2 Write-through

Etter hver gang det skrives til en blokk, skrives innholdet i blokken også tilbake til RAM.

1. Fordel: Enkelt å implementere i cache-kontrolleren
2. Ulempe: Hvis det skal skrives flere ganger rett etterhverandre til samme blokk må prosessoren vente mellom hver gang

5.4.3 Write-back

Innholdet i en blokk i cache kopieres kun tilbake til RAM hvis blokken skal overskrives i cache og det har blitt skrevet til den i cache
For raskt å detektere om en blokk har blitt skrevet til benyttes et kontroll-bit i cache (kalles dirty-bit'et).

- Fordel: Gir ingen hastighetsreduksjon ved flere påfølgende write-operasjoner til samme blokk siden CPU'en ikke trenger vente mellom hver skriving
- Ulempe: Hvis arkitekturen støtter direkte lesing/skriving mellom RAM og Input/Output-enheter uten å gå via CPU'en (kalt Direct Memory Access eller DMA), risikerer man inkonsistente data hvis ikke DMA- kontrolleren kommuniserer med cache-kontrolleren.

5.5 Replacement strategy

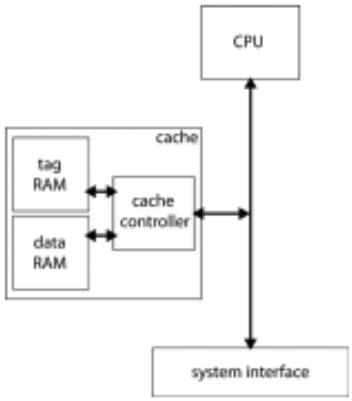
- Ved en cache miss må noen ganger en eksisterende blokk kastes ut for å gi plass til en ny blokk. Hvilken blokk som fjernes kan ha stor betydning for hastigheten til programmet som eksekveres.
- First-in-First-out (FIFO)
  - Blokkene er organisert som en ringbuffer.
  - Blokken som har ligget lengst i minne blir overstrevet (uavhengig om den er mye brukt)
- LRU (Least Recently Used):
  - Blokken som har ligget lengst i cache uten å ha blitt skrevet til eller lest skrives over, pga. Lokalitetsprinsippet
  - Ren LRU benyttes sjelden fordi det medfører mye administrasjon som i seg selv er tidkrevende (bl.a må et tidsstempel oppdateres hver gang en blokk aksesseres)
- Random: Kaster man ut en tilfeldig blokk når man trenger å frigi plass til en blokk
- Hybrid: Deler inn blokker i tidsgrupper, og kaster så ut en tidfeldig valgt fra den gruppen som har ligget lengst i cache uten å bli brukt.

5.6 Arkitektur

- Cache kan plasseres på (minst) to måter i forhold til CPU og RAM. (1) Look-aside, (2) Look-through
- De to organiseringene påvirker hvordan lesing håndteres ("read architecture")
- "Read architecture" og "write policy" er det viktigste karakteristika til cache
- Cache koherens: Hvorvidt innholdet i cache og RAM er identisk
- For å bevare koherens kan cache gjøre
  - "Snoop": Cache overvåker adresselinjer for å se etter transaksjoner som angår data den sitter på
  - "Snarf": Cache tar data fra datalinjer og kopierer dem inn
- Inkoherens kan være
  - "Dirty data": Data i cache er modifisert, men ikke tilsvarende data i RAM
  - "Stale data": Data i RAM er modifisert, men ikke tilsvarende data i cache.

5.6.1 Look-aside arkitektur

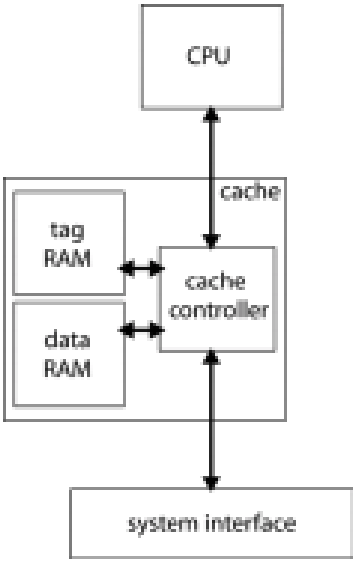
- Cache er plassert med RAM
- Både RAM og cache leser adresselinjene samtidig



- Når prosessorer starter en lesing, sjekker cache adressen (snooping) for å se om det er en "read hit"
- Hvis "read hit": Cache svarer prosessoren og stopper videre buss-syklus for å hindre lesing fra hovedminnet (RAM)
- Hvis "read miss": Hovedminnet (RAM) vil besvare henvendelsen fra prosessoren. Cache "snarfer" data slik at de er i cache neste gang prosessoren ber om dem.
- Fordel: Mindre komplisert enn look-thorugh, Bedre responstid (RAM og CPU ser samme buss-syklus)
- Ulempe: Prosessoren kan ikke aksessere cache hvis en annen enhet aksesserer RAM samtidig

5.6.2 Look-through

- Cache sitter fysisk mellom CPU og internbussen
- Cache-kontrolleren vil avgjøre om buss-syklus skal slippe videre til RAM



6 Hvordan jobber prosessoren i en datamaskin?

For å kunne kjøre et program må dette tolkes av maskinen. Dette kan gjøres ved hjelp av andre programmeringsspråk som C og C++. Disse språkene oversetter programmene til maskinkode slik at maskinkoden kan utføres av CPU-en i en datamaskin. I Python kan vi skrive en funksjon nineteen som gir verdien 19.

Maskinkoden for funksjonen nineteen er:
0x48 0xC7 0xC0 0x13 0x00 0x00 0x00 0xC3

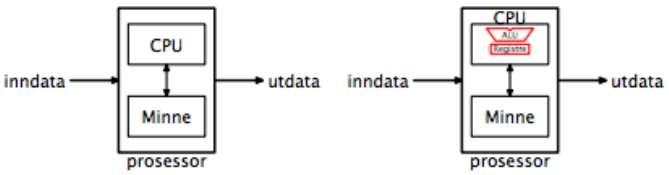
```
def nineteen():
    return 19

print("nineteen()", "=", nineteen())
```

Assemblerkode er huskekoder for instruksjonene, så vi skriver heller funksjonen vår slik:

```
movq    $19,%RAX
    #Hent 19.
ret
```

6.1 CPU-en

I dette faget tar de for seg en x64 CPU som er laget av Intel og AMD. Det finnes i de aller fleste personlige datamaskiner i dag.



Navn	Bruk
%RAX	"Accumulator", dvs svar
%RCX	"≪Counter≫
%RDX	"Data"
%RDI	"Destination index"
%RSI	"Source index"
%R8	"alt mulig"
%R9	"alt mulig"
%R10	"alt mulig"
%R11	"alt mulig"

6.2 Instruksjoner

Den aller viktigste instruksjonen heter *movq* og den kopierer data. Den kan kopiere en fast verdi (angitt med $ foran) til et register eller kopiere innholdet av ett register til et annet. Man må også huske å ha **"ret"** på slutten av koden slik at koden kan returnere og ikke bli hengende.

```
movq $4,%RAX
# Kopier verdien 4 til register %RAX
movq %RDX,%R8
# Kopier innholdet av register %RDX
til %R8
```

For å kunne teste en funksjon fra assembly filen må man ha et lite C-program (se under).

nineteen.s

```
# def nineteen(): Hent verdien 19.

        .globl  nineteen
nineteen:
        movq    $19,%rax  # Hent 19.
        ret
```

test-nineteen.c

```
#include <stdio.h>

extern long nineteen ();

int main (void)
{
  long res = nineteen();
  printf("nineteen() = %d\n", res);
}
```

Kompilering:
gcc -o test-nineteen test-nineteen.c nineteen.s
./test-nineteen
nineteen() = 19

Funksjonene kan også ha paramaetre, dvs verdier som brukes som grunnlag for beregningen. Parametrene ligger i disse registrene når funksjonen blir kalt (Linux):

↓

Navn	Bruk
movq	«Flytt en verdi»
ret	«Returner fra en funksjon»
negq	«Skift fortegn på en verdi»
addq	«Legg til en verdi»
subq	«Trekk fra en verdi»
imulq	«Multipliser med en verdi»
indivq	«Divider med en verdi»

Navn	Bruk
%RDI	«1. parameter»
%RSI	«2. parameter»
%RDX	«3. parameter»

Divisjon er litt sær, men følgende oppskrift fungerer:

1. Legg dividend (den verdien vi skal dele) i %RAX.
2. Legg divisor (den verdien vi skal dele på) i ett av registrene %R8, %R9, %R10 eller %R11.
3. Utfør de to instruksjonene: *cqo* og *indivq* *%rn* etter cqo.
4. Da kommer svaret i %RAX og resten fra divisjonen i %RDX.

6.2.1 Eksempler

```
def doble (v):
    return v + v

print("doble(88)", '=', doble(88))
```

I assemblerkode skriver vi

doble.s

```
# def doble(v): Beregn den doble verdien av v.

        .globl  doble
doble:
        movq    %rdi,%rax  # Hent v
        addq    %rdi,%rax  # og legg til v.
        ret
```

test-doble.c

```
#include <stdio.h>

extern long doble (long v);

int main (void)
{
  long res = doble(17);
  printf("doble(17) = %d\n", res);
}
```

Skriv en funksjon som beregner $a \times \frac{b}{c}$.

f.s

```
# def f(a, b, c): Beregn a * b / c

        .globl  f
f:
        movq    %rdi,%rax  # Hent a og
        imulq   %rsi,%rax  # gang med b
        movq    %rdx,%r8   # Hent c og
        cqo                #
        idivq   %r8        # del a*b med c.
        ret
```

test-f.c

```
#include <stdio.h>

extern long f (long a, long b, long c);

int main (void)
{
  long res = f(171, 211, 10);
  printf("f(171,211,10) = %d\n", res);
}
```

Instruksjoner

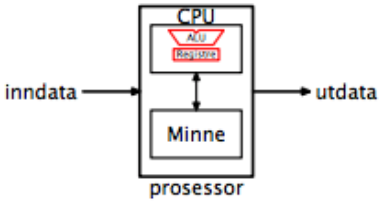
movq flytter data
negq snur fortegnet
addq legger sammen
subq trekker fra
imulq multipliserer
idivq dividerer (*sær!*)
cqo hjelp for *idivq*
andq bitvis AND
orq bitvis OR
xorq bitvis XOR
notq bitvis NOT

ret returnerer
jmp hopp
js hopp hvis $S = 1$
jns hopp hvis $S \neq 1$
jc hopp hvis $C = 1$
jnc hopp hvis $C \neq 1$
jz hopp hvis $Z = 1$
jnz hopp hvis $Z \neq 1$

7 Hvordan en prosessor arbeider, del 2

7.1 Logiske instruksjoner

Siden ALU-en er bygget opp av logiske porter, kan vi kanskje bruke den direkte?



Vi har disse instruksjonene:
andq foretar logiske AND på bit-ene i data:

	···	0	1	0	1	0	1	0	1
andq	···	0	0	0	0	1	1	1	1
=	···	0	0	0	0	0	1	0	1

orq foretar logisk OR.
xorq foretar logisk XOR.
notq foretar logisk NOT

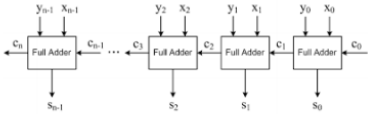
La oss lage en funksjon som sjekker om et tall er et oddetall; svaret skal være 1 om det er det, og 0 ellers.
Hva vet vi: I oddetall lagret binært er siste bit alltid 1.
Tenk fremgangsmåte:
- Vi har noen registre med gitte verdier.
- Vi har noen instruksjoner som kan være nyttige.
- Vi skal ha et gitt svar i korrekt register

```
        .globl odd
odd:
        movq %RDI, %RAX
        andq $1,%RAX        ret
```

7.2 Flagg

I tillegg til registrene vi har nevnt, finnes det noen 1-bits registre kalt flagg. Disse får sin verdi som bieffekt av instruksjonene, og de kan gi nyttig informasjon.

S («Sign») får en kopi av det øverste bit-et («fortegnsbit-et») etter operasjonen.



C («Carry») får det mentebit-et som forsvinner ut.

Z («Zero») settes til 1 hvis svaret er 0.

Denne funksjonen finner absoluttverdien til et heltall, dvs at hvis tallet er negativt, returnerer den den tilsvarende positive verdien.
Noen nye instruksjoner:
js: hopper til et annet sted i koden, men bare hvis S-flagget er 1.
jns: hopper til et annet sted i koden, men bare hvis S-flagget er 0.

```
        .globl  abs_val
abs_val:
        movq    %RDI,%RAX
        addq    $0,%RAX
        jns     positivt   # Jump if not S-flag
        negq    %RAX
positivt:
        ret
```

7.3 Variabler

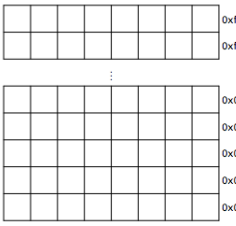
Ofte trenger man (mye!) mer plass enn registrene tilbyr. Da kan man legge data i minnet:

```
        .data
var1:   .quad 0
var2:   .quad 0
var3:   .quad 0
```

Instruksjonen movq kan flytte mellom registre og minnet også.

7.3.1 Minne

Minnet består av millioner av byte, og hver byte har sin adresse:



I en 64-bits prosessor jobber man med 8 og 8 byte.

Denne funksjonen lagrer tre verdier i minnet før den summerer dem.

```
        .globl  sum3
sum3:
        movq    %RDI,var1
        movq    %RSI,var2
        movq    %RDX,var3

        movq    var1,%RAX
        addq    var2,%RAX
        addq    var3,%RAX
        ret

        .data
var1:   .quad   0
var2:   .quad   0
var3:   .quad   0
```

```
$ more test-var3.c
#include <stdio.h>

extern long sum3 (long a, long b, long c);

int main (void)
{
  long res = sum3(228, 5, -33);
  printf("Res = %d\n", res);
}
$ gcc -o test-var3 test-var3.c var3.s
$ ./test-var3
Res = 200
```

7.4 Adresser

Grunnregel: Alt er bit!

1. Heltall lagres på binær form som bit i datamaskinen.
2. Alle cellene i minnet har sin adresse, og denne adressen er et heltall.
3. Konklusjon 1: Vi kan lagre adresser i minnet.
4. Konklusjon 2: Vi kan regne med adresser.

```
        .globl  get
get:
        movq    %RDI,%R10
        imulq   $8,%R10
        addq    $val,%R10
        movq    (%R10),%RAX
        ret

        .data
val:    .quad   22
        .quad   11
        .quad   34
        .quad   50
```

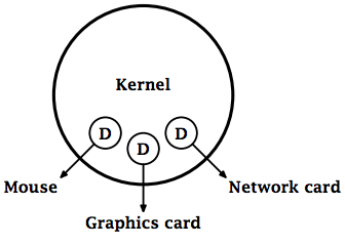
Array deklarasjon

```
$ more test-array.c
#include <stdio.h>

extern long get (long index);

int main (void)
{
  int i;

  for (i = 0;  i < 4;  i++) {
    long res = get(i);
    printf("%d: %d\n", i, get(i));
  }
}
$ gcc -o test-array test-array.c array.s
$ ./test-array
0: 22
1: 11
2: 34
3: 50
```

8 Operativsystemer

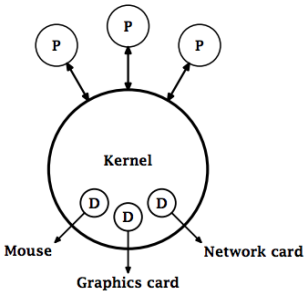
OS blir brukt til å kommunisere med brukeren, administrere kjøringen (dvs starte og stoppe programmer) og administrere lagring av filer.

Oppbygningen av et operativsystem



Kjernen styrer alt. Den kjører i «**supervisor mode**».
Kjernen kommuniserer med **periferenhetene** med spesialskrevne kodebiter kalt **drivere** (D).

Prosesser



Brukerprogrammene er **prosesser**. De kjører i «**protected mode**» og må kommunisere med kjernen for
- innlesing og utskrift
- kommunikasjon med andre prosesser

Parallellisering: Hvis datamaskinen har flere kjerner, kan prosessene kjøre på ulike kjerner og dermed gå fortere.

Tidsdeling: Når det er flere prosesser som skal kjøres på samme kjerne, vil OSet bytte på å kjøre dem (såkalt tidsdeling eller «timeslicing»).

8.0.1 Oppstart

Hvordan starter man et OS når man mangler alle mekanismene man trenger? Det skjer ved «bootstrapping» eller «booting».

Gammelt ordtak: «To pull oneself over a fence by one's bootstraps.»

1. Når prosessoren starter, begynner den et fast sted i koden kalt **BIOS** («Basic Input/Output System») som ligger i ROM. ROM = Read-only memory
2. BIOS er spesiallaget for den enkelte datamaskinmodellen. Det er et mini-mini-OS som kan lese fra disk eller nettverket.
3. BIOS leser en **Bootstrap loader** som er et mini-OS; det kan lese inn selve OS-et og starte det.

8.1 Prosesser

Alle prosesser har et unikt nummer: PID, kommandoen ps -ef vil vise alle prosessene på maskinen (linux). Man kan stoppe en prosess ved å trykke: "ctrl" + "c", dette vil stoppe prosessen man kjører. Hvis en prosess har sitt egt vindu på skjermen kan man bare lukke den ved å trykke på xpå toppen. Man kan også drepe en prosess med kommandoen "kill "PIDnr"".

8.1.1 Kommandotolkeren

Kommandotolkeren (≪shell-et≫) er én av prosessene; den vanligste i Linux heter bash (≪Bourne-again shell≫).
Kommandotolkeren leser kommandonavn som brukeren skriver, og oppretter en prosess som utfører den.

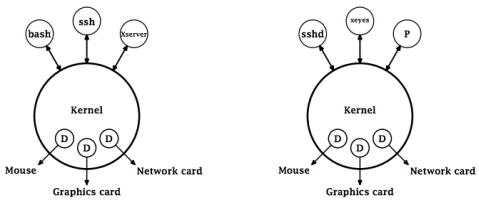
- Normalt vil bash vente til prosessen er ferdig før den ber om ny kommando.
- Hvis kommandolinjen slutter med &, venter den ikke.

8.2 Vindussystemer

Alle moderne OSer har et GUI (Graphical User Interface) for å

- ha mange vinduer på skjermen
- ha grafikk i vinduene (ikke bare tekst)
- få input fra mus (ikke bare tastatur)

Vindussystemet X kan kommunisere over nettet.



8.3 Filer

En fil er en samling byte lagret i eller ved datamaskinen (på disk, SSD (Solid State Drive) el).

OSer holder orden på:

- filens navn
- hvilken fysiske enhet den er lagret på, og
- hvor i mappestrukturen filen ligger

I bash er jeg alltid posisjonert et sted i fil-treet så jeg kan aksessere filer derfra.

pwd forteller hvor jeg er.

cd flytter meg.

NB! Alle mapper inneholder disse to filene:

. is mappen selv.

.. is «foreldremappen».

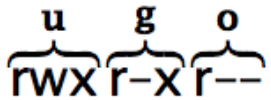
(Filer med navn som starter med "." blir ikke vist av ls med mindre man bruker opsjonen –a.)

8.3.1 Besyttelse av filer

For å forstå beskyttelsen av filer i Unix, må man vite at Unix opererer med tre kategorier brukere:
- **user** (forkortet «u») er filens eier, dvs den som opprettet den.
- **group** (forkortet «g») er gruppen tilordnet filen. (Mer om dette siden.)
- **other** (forkortet «o») er alle andre brukere.

Det finnes tre privilegier for filer:
- **read** (forkortet «r») er retten til å lese filen.
- **write** (forkortet «w») er retten til å skrive på filen, dvs endre den.
- **execute** (forkortet «x») er retten til å utføre den, dvs bruke den som et program.

Privilegiene angis i bolker på 3*3 bokstaver:



En bokstav angir at man har privilegiet, en «–» at man ikke har det.

9 Lagdeling i Internettarkitekturen

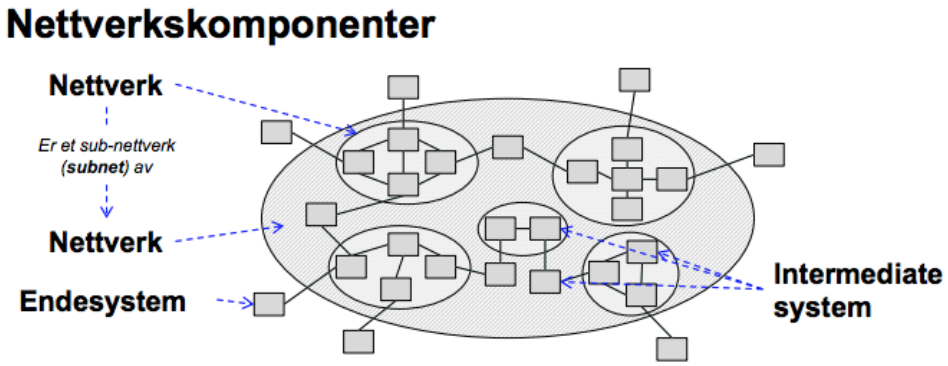
En protokoll definerer strukturen på beskjeder sendt over et nettverk.

Trenger i tillegg å adressere mange kompleksiteter...

- – Hvordan skal maskinvaren oppføre seg?
- Hvordan skal beskjeden finne frem?
- Er det noen garantier for levering?
- Hvordan håndtere kø, tap og andre problemer?

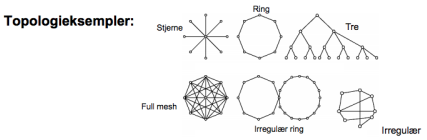
Endesystemer:
Endesystemer finnes helt ytterst i nettverket.
For eksempler: datamaskiner, mobiltelefoner, sensorer, skrivere

Intermediate system:
For eksempel:
- ruter, switch
- gateway
- repeater, bridge

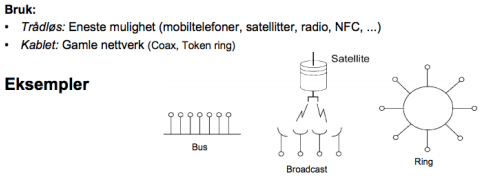


9.1 Nettverksstrukturer

Punkt til punkt:
- Flere forskjellige kabler, kabeltyper eller radiolinker som kommuniserer fra punkt til punkt.
- Kabel eller radiolink kobler alltid sammen to noder.
- En-til-en overføring.



Broadcast-systemer:
For eksempel:
- Nettverk som deler kommunikasjonsmedium.
- En sender, alle lytter (en-til-mange).



9.1.1 Nettverkstyper

Avstand mellom punktene	Lokasjon	Eksempel
0,1 m og lavere	Kretskort	Multi-core prosessorer
1 m	Systemer	NFC, BAN, PAN
10 m	Rom	LAN, SAN
100 m	Bygninger	
1 km	Campus	
10 km	Byer	MAN
100 km	Land	WAN
1.000 km	Kontinenter	
10.000 km +	Planeter	

- *NFC*: near field communication, *BAN*: body area network, *PAN*: personal area network
- *LAN*: Local Area Network: IEEE 802.3 (Ethernet), IEEE 802.11 ("WiFi", "WLAN"), ...
- *SAN*: storage area network (iSCSI, NVMe-oF)
- *MAN*: Metropolitan Area Network: DSL, EPON, ...
- *WAN*: Wide Area Network: Frame Relay, SDH, ATM, optiske nettverk (WDM)
- Interplanetært Internett: http://www.ipnsig.org/

9.2 Protokoller

Utfordring:

- Potensielt veldig komplisert med kommunikasjon til fremmede maskiner på nettet. - Interaksjon mellom forskjellige typer system og/eller nettverk.

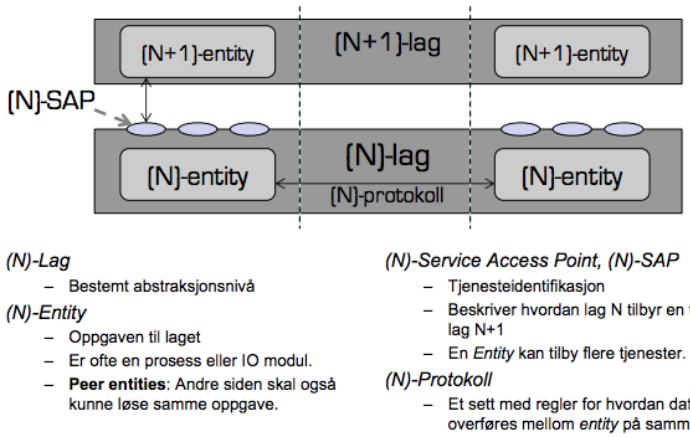
Forenkling:

- Introdusere standardiserte abstraksjonsnivåer
- Generelt: «modul», «lag» eller «nivå»

Eksempel:

- En biolog med en oversetter og en kryptert faks for å sende data over nettet.

9.2.1 Lag i nettverket (OSI)



Definerer:
- Formatet på beskjeden og header (konvolutt).
- Rekkefølgen på beskjedene.
- Utvekslingen av beskjeder mellom to eller flere kommunikasjonssystemer.
- Hva skal skje ved mottak eller sending av en beskjed.

Definerer ikke:
– Tjenestene tilbudt til laget over (N+1)
– Tjenestene brukt i laget under (N-1)-SAP

ISO Open Systems Interconnection – også kjent som OSI-modellen

- Modell for lagdelte kommunikasjonssystemer

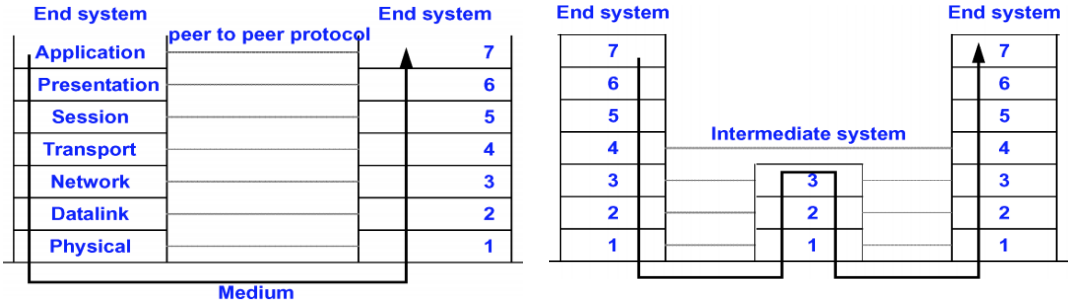
- Grunnleggende konsepter og terminologi

- Definerer syv lag med funksjonalitet

7	**Applikasjonslaget**
6	**Presentasjonslaget**
5	**Sesjonslaget**
4	**Transportlaget**
3	**Nettverkslaget**
2	**Linklaget**
1	**Fysisk lag**

9.2.2 Dataenheter i OSI-modellen

- "Beskjeder" fra applikasjonslaget defineres som «data units».
- Data units har forskjellige navn, avhengig av hvilket lag vi befinner oss i:
  - Packet: Brukes på transportlaget (kan inneholde fragmenter)
  - Datagram: Brukes på nettverkslaget
  - Frame: Brukes på linklaget, (ferdig "pakket", klar til å sendes ned til fysisk lag)
- OSI terminologi en for «beskjed» er en PDU
  - PDU: Protocol Data Unit
  - SDU: Service Data Unit = Nyttelasten i en PDU



9.3 TCP/IP

Forskjell på TCP/IP modellen og ISO-OSI

– Presentasjon, sesjon og applikasjonslagene slås sammen til ett lag.

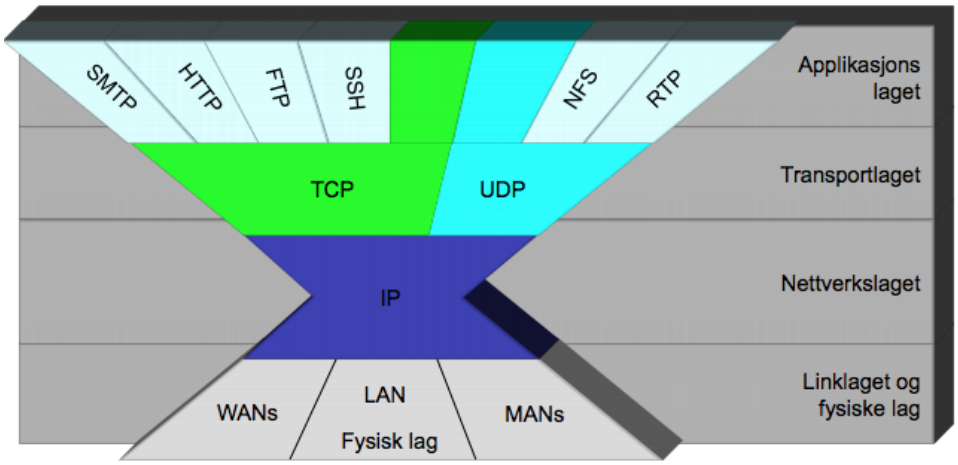
– Litt ut i fra hvem du spør så slås også linklaget og det fysiske laget sammen til ett lag kalt nettverksgrensesnittet.

5	Applikasjonslaget
4	Transportlaget
3	Nettverkslaget
1/2	Nettverksgrensesnitt

Forskjell på TCP/IP modellen og ISO-OSI
– Presentasjon, sesjon og applikasjonslagene slås sammen til ett lag.
– Litt ut i fra hvem du spør så slås også linklaget og det fysiske laget sammen til ett lag kalt nettverksgrensesnittet.

Lag		Funksjon
5	**Applikasjon**	Applikasjonsrelaterte tjenester, og eventuell funksjonalitet fra OSI lag 5 og 6.
4	**Transport**	Kobler sammen systemene ende-til-ende (TCP/UDP)
3	**Nettverk**	Rute data fra ende-til-ende systemer (IP)
2	**Link**	Pålitelig overføring mellom to noder
1	**Fysisk**	Sender bit ut på mediet (kablet eller trådløst)

9.4 Internet Protocol Stack



Husk: Internett har kun 5 lag (eller 4)

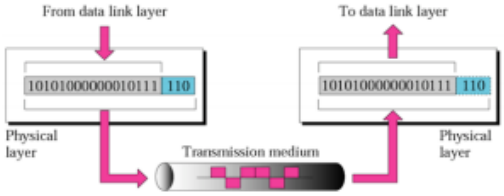
- Lag 5, 6 og 7 fra OSI er implementert som ett applikasjonslag.

På Internett blir ofte lag 3 (nettverk) og 4 (transport) blandet:

- Transportprotokoll TCP (eller UDP) og nettverksprotokoll IP
- Kan i flere tilfeller være utfordrerne å trekke linjen hvor TCP slutter og IP begynner.

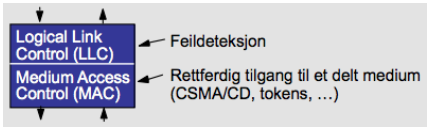
9.4.1 Lag 1 – Det fysiske laget

- Signalrepresentasjonen av bits:
- Sørger for at 1-bit også blir mottatt som 1-bit (og ikke et 0-bit):
- Mekanikk: Koblingstype, kabler/medium,..
- Elektronikk: spenning, bit-lengde,..
- Formelle regler for kommunikasjon:
    - Unidirectional (enveis) eller bidirectional (toveis) kommunikasjon
    - Hva skal markere starten og slutten på overføringer

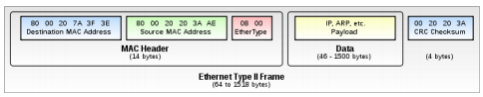


9.4.2 Lag 2 - Linklaget

- Pålitelig overføring mellom to enheter.
    - Pakker som overføres i linklaget kalles “frames”
    - Feildeteksjon or retting innenfor en frame
- En «switch» vil kun jobbe på lag 2
- Lag 2 vil kunne ha enkel flytkontroll
    - Rask sender, treg mottaker
- Medium Access Control (MAC)

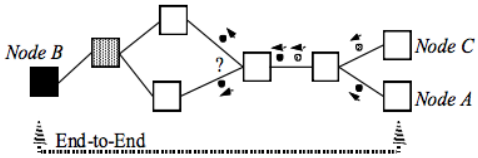


- Det vanligste linklagene er ”Ethernet”, og ”WiFi”. De er ganske like, men har noen forskjeller.
- Bruker en 6-byte adresse (48-bit) som ofte er lagret i nettverkskortet
    - MAC-adresse, brukes både på WiFi og Ethernet
    - Hver byte representeres med en heksadesimal verdi: 07:01:02:01:2C:4B

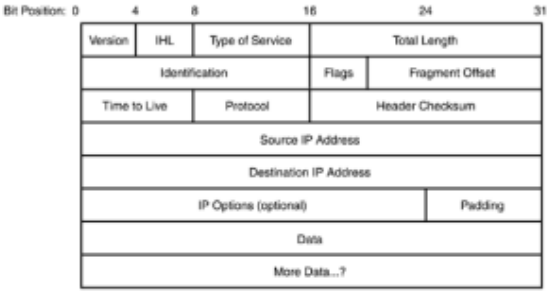


9.4.3 Lag 3 - Nettverkslaget

- Kobler sammen ende-til-ende systemer
- Ruting
    - Statisk, definert under tilkobling eller dynamisk
    - Meningskontroll (for mange pakker på en sti)
    - Tjenestekvalitet (QoS)
- En «ruter» jobber på lag 3
- Eksempler: IP (tilkoblingsløst), X.25 (tilkoblingsorientert)



- Den mest brukte nettverkslagsprotokollen i dag er Internet Protocol (IP). Den mest brukte versjonen er IPv4.
- Bruker en 32-bit adresse, ($4.3*10^9$) den nye versjonen IPv6 har 128-bit adresser ($3.4*10^{38}$)
    - Representeres med fire 8-bit heltall: 192.168.1.101



9.4.4 Lag 4 - Transportlaget

- **TCP**
  - Oppsett av forbindelse (3-way handshake)
  - Garanterer at pakkene leveres i riktig rekkefølge
  - Pålitelighet – Pakke sendes på nytt hvis kvittering (ACK) ikke kommer frem
  - Flytkontroll og meningskontroll
- TCP: HTTP, E-post, filoverføring, etc.

- **UDP**
  - Tilkoblingsløs forbindelse
  - Ingen garantier
  - «Best-effort» levering av data
- UDP: Strømming av video og lyd

Bruker «port» som en unik identifikator.

- Representeres med et 16-bit heltall

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

9.4.5 Lag 5 – Applikasjonslaget

- Lag med tjenester for applikasjoner: - Eksempler:

- – Nettlesere (WWW)
- – E-post
- – Filoverføring
- – P2P

10 Lagene spiller sammen

10.1 Lokalnettverk (LAN) og subnett

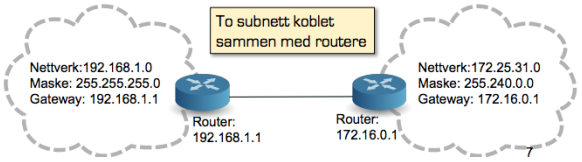
Internett er et sammenkobling av mindre, separate nettverk.

Koblet sammen med switch og/eller HUB (lag 2*)

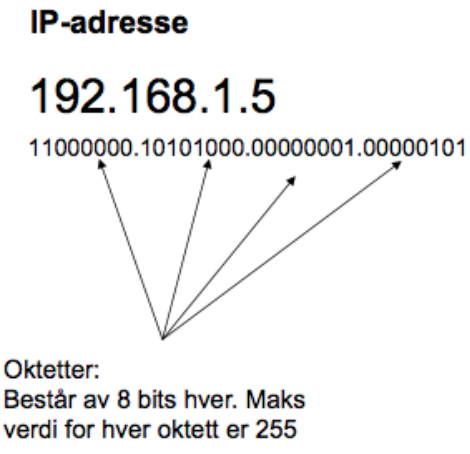
- Switch: filtrerer og videresender.

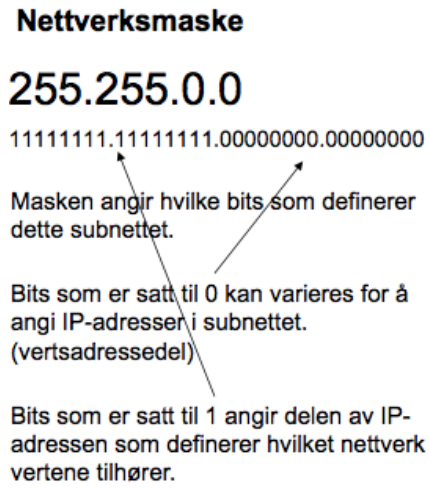
- HUB: Videresender det som kommer inn på alle porter

For å sende en pakke til en maskin utenfor ditt lokale nettverk må den sendes til en router som vet hvor den skal videresendes.

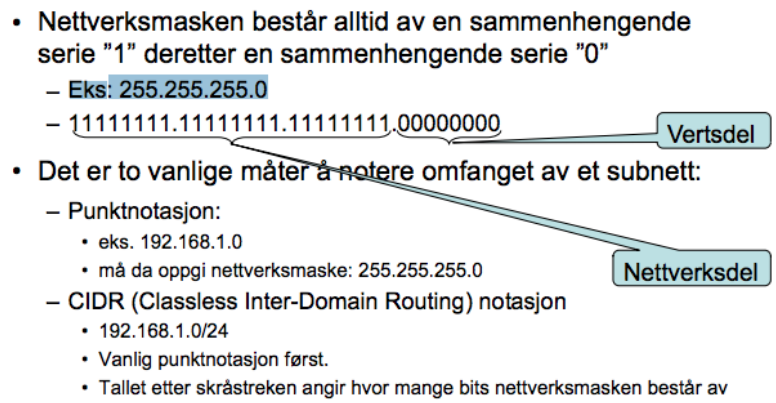


10.2 IP-adresser (IPv4)





10.2.1 CIDR- og punktnotasjon av subnett



10.2.2 Regne ut subnettet fra en IP + nettverksmaske

En maskin i nettet har IP 192.168.1.5= 11000000.10101000.00000001.00000101

Nettverksmasken er 255.255.255.0 = 11111111.11111111.11111111.00000000

For å finne subnettadressen til maskinen må du gjøre en bitvis AND operasjon mellom IP-adressen og nettverksmasken.

11000000.10101000.00000001.00000000 = 192.168.1.0

Dette er den første IP-adressen i subnettet og brukes til å identifisere subnett.

10.2.3 ARP – Koblingen mellom nettverk og IP

Nettverkskortene har en 6 byte lang media access control (MAC)-adresse som brukes til å identifisere maskinen innenfor et kringkastingsdomene (broadcast domain).

For at IP skal fungere, må avsenderen vite hvilken MAC-adresse pakken skal sendes til.

Address Resolution Protocol(ARP) kobler IP (Internett) og MAC (Linklaget).

For store kringkastingsdomener Om kringkastingsdomenet inneholder for mange enheter, kan det by på problemer:

– ARP går til alle maskiner i kringkastingsdomenet. Om domenet er for stort kan dette stjele kapasitet som burde ha vært brukt til å overføre data.

– DHCP-forespørsler går også til alle -¿ samme problem.

Avanserte switcher kan filtrere ARP og DHCP for å beskytte mot overbelastning fra kringkastet trafikk.

ARP kan også brukes til å finne duplikate IP-adresser:

– "Hvem har IP 0.0.0.0" sendes til MAC FF:FF:FF:FF:FF:FF

– Svar fra alle: IP kommer flere ganger = feil

10.2.4 Private IP-adresser

Private IP-adresser er adresser som er reservert for bruk I lukkede nettverk og nettverk med NAT mot Internett

Disse IPene skal ikke være direkte koblet mot Internett!

En hjemmeruter er vanligvis satt opp til å gi deg et LAN med et subnett fra en av disse segmentene.

RFC1918 name	IP address range	number of addresses	largest CIDR block (subnet mask)	host id size	mask bits
24-bit block	10.0.0.0 – 10.255.255.255	16,777,216	10.0.0.0/8 (255.0.0.0)	24 bits	8 bits
20-bit block	172.16.0.0 – 172.31.255.255	1,048,576	172.16.0.0/12 (255.240.0.0)	20 bits	12 bits
16-bit block	192.168.0.0 – 192.168.255.255	65,536	192.168.0.0/16 (255.255.0.0)	16 bits	16 bits

10.2.5 Ulemper med NAT (Network Address Translation)

Ekstra kompleksitet i nettverket

Routeren må bevare tilstanden til forbindelsene

Nye forbindelser må initieres fra innsiden av NAT-nettverket

Gjør det vanskelig å koble til utenifra

- Universal Plug and Play (UPnP)
    - Enheter på LAN kan automatisk åpne opp for forbindelser utenifra
- STUN / TURN ++
    - Bruke maskiner med gyldig Internett IP-adresse til å sette opp forbindelsen.
- Videresending av porter
    - f.eks 46.67.132.46:5000 -¿ 192.168.1.5:5000

- demilitarized zone (DMZ): 46.67.132.46:* -¿ 192.168.1.5:* 2

10.2.6 IPv6

Lang prosess med å få i drift siden det må støttes i alle noder fra ende-til-ende

128 bits IP-adresse (mot 32 i IPv4)

2^{128} (eller $3.4*10^{38}$) mulige adresser

10.3 Ruting i Internett (svært kort)

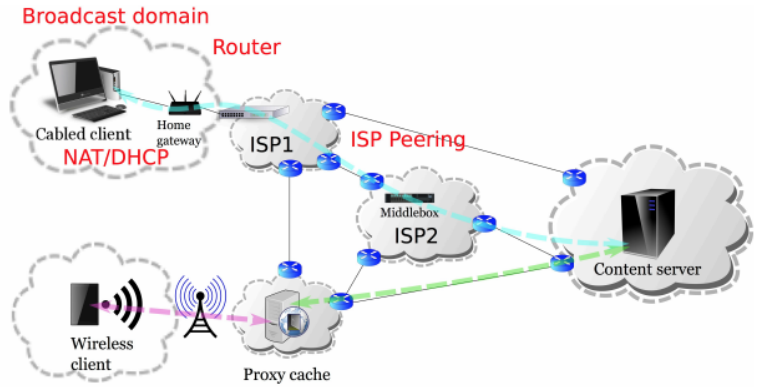
Målet for ruting: å videresende en datapakke slik at den til slutt når måladressen sin.

En Internet Service Provider (ISP) driver et nettverk og leverer datatransport til andre ISPer og sluttbrukere.

ISP Peering er når ISPer inngår avtaler om å videresende hverandres trafikk. Økonomiske prinsipper er da med på å bestemme hvor trafikken flyter.

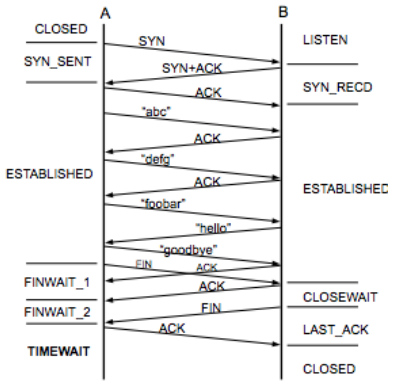
Border Gateway Protocol (BGP)– Rutingprotokoll som brukes mellom ISPer. Svært store selskaper kan også bruke BGP.

OSPF – eksempel på rutingalgoritme for mindre nettverk. Bygger et kart over mulige ruter til måladressene. Skalerer ikke for store nettverk.



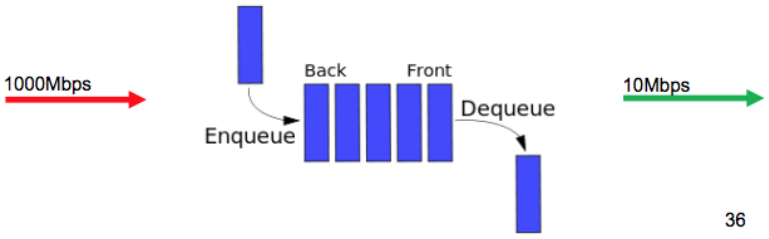
10.4 Transmission Control Protocol (TCP)

Forbindelsesorientert
Metningskontroll
Byte-strøm og levering i rekkefølge
Pålitelighet
– Implementert ved at bekreftelser på hver pakke sendes tilbake fra mottakeren
Feilsjekking av nyttelasten (sjekksum)

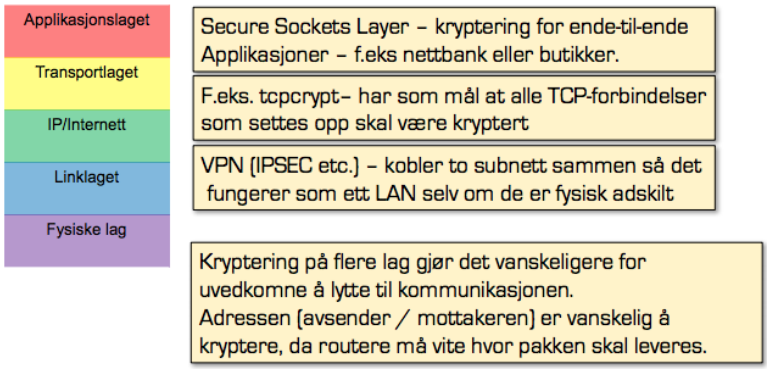


10.5 Metningskontroll

En ruter er i utgangspunktet bare en FIFO (first-in-first-out) kø.
Om det blir for mye trafikk over en gitt kø, vil det føre til stopp i trafikken. Dette kalles "congestion".
Om det er så mye traffikk at det blir full stopp for alle, kalles det "congestion collapse".
For å unngå dette bygde man inn mekanismer i TCP for å tilpasse senderaten til nettverksforholdene.
Ressursene deles teoretisk sett likt mellom forbindelser
For å kapre mer av kapasiteten: åpne flere forbindelser I paralell.
Ingen god metode for å fordele ressurser i Internett rettferdig pr. maskin eller pr. bruker i dag.



10.6 Kryptering / sikkerhet i lagene



10.7 Domain Name System (DNS)

Distribuert database med enkel klient/tjener arkitektur:

- UDP eller TCP port 53
- tjenere må bruke TCP (fra nylig)
- klienter som bruker TCP avvises ofte
  - reduserer lasten på DNS-tjeneren

10.7.1 Tjenerhierarki

Funksjonene til hver DNS-tjener

• Autoritet over en del av hierarkiet

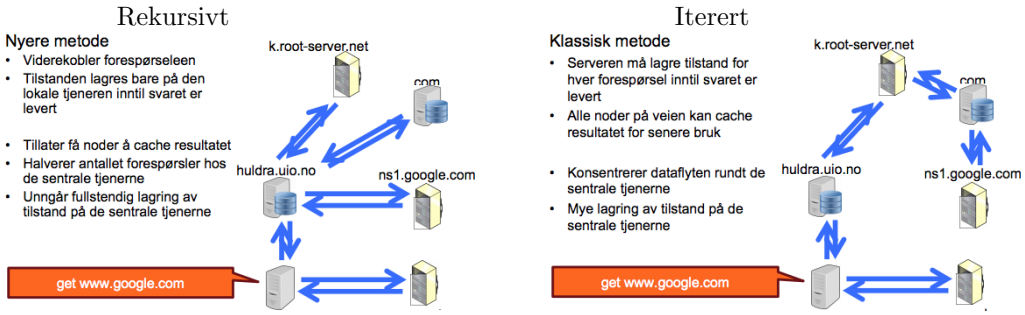
- Ikke behov for å lagre alle DNS-navn • Lagre alle oppføringene for maskiner/domener i sin

sone – Må replikeres for å sikre oppetid (minst 2 tjenere)

• Må kjenne adressene til rottjenerne

– Slå opp forespørsler på navn som den ikke lagrer selv Rottjenerne kjenner til alle TLDene (Top Level Domains)

10.7.2 Oppslag i DNS

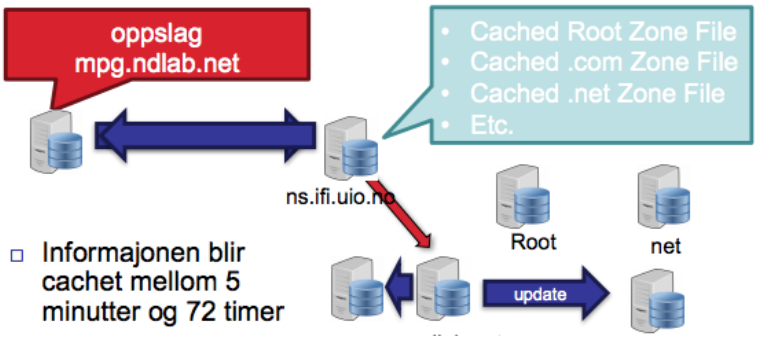


10.7.3 Caching vs. oppdaterte data

Caching reduserer forsinkelsen for et DNS-oppslag

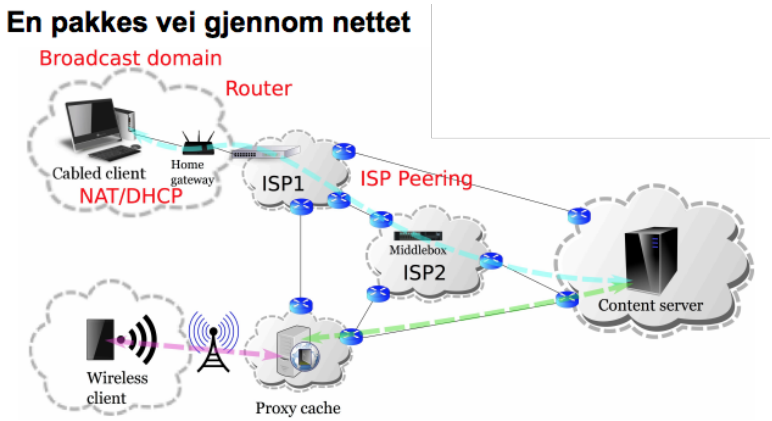
Caching reduserer lasten på DNS-tjenerne

Caching forsinker oppdateringer



10.8 Datapakker

En datapakkes vei fra sender til mottager: systemkall (send), OS/stack, nettverkskort, kabel/switch/GW, DSLAM/ISP, IX/peering partners osv.



11 Tjenester i Internett

Forbindelsesstrategier (push og pull)

Pull:

– Klienten ber tjeneren om en tjeneste (f.eks et dokument) Tradisjonelt den vanligste metoden Push

Tjeneren "dytter" en tjeneste (f.eks en beskjed) til klienten. Krever at det er en forbindelse fra før, eller at klienten lytter. Publish-subscribe: Variant av "Push" der tjeneren dytter ut beskjeder til en gruppe av abonnenter (subscribers)

11.1 Aksessmodeller

Klient-tjener

Tradisjonell kommunikasjonsmodell, lettfattelig abstraksjon

– Klienter ber om en tjeneste (oppretter en forbindelse)

– Tjenere leverer tjenesten (svarer på forespørselen)

Eksempler: Webklient (nettleser)/Webtjener, Mailklient/tjener, FTP

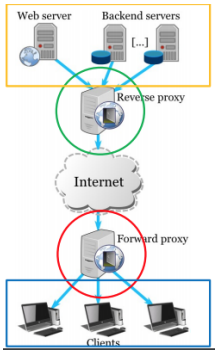
Peer-to-Peer (P2P)

P2P er den orginale modellen for internett, der alle noder er likeverdige, alle noder kan nå hverandre og eierskapet er distribuert.

Eksempler: Napster, BitTorrent, Kazaa osv. (fildeling - ulovlig).

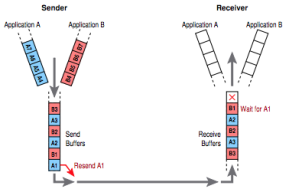
11.2 Proxy-cache

En "Forward Proxy" står nær klienten – og mellomlagrer data for klientene, slik at de ikke behøver å gå helt til kilden.
En "Reverse Proxy" står nær tjeneren – og mellomlagrer data fra én eller flere tjenere, slik at klienten slipper å gå helt til kilden(e).
+ Lastbalansering
+ Sparer nettverkskapasitet
+ Lavere forsinkelse



11.3 Head of line blocking

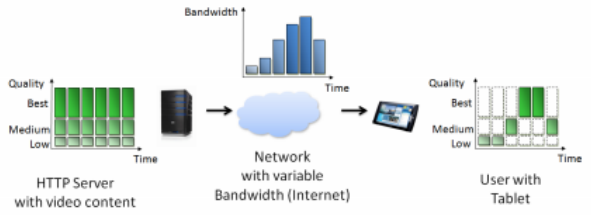
Mottageren må vente på et forsinket segment før mer data kan leveres. Oppstår ved avhengigheter
Tapte pakker må sendes på nytt.
Dette tar tid, og forsinker alle ventende bytes i køen



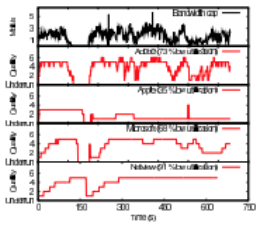
11.4 HTTP

Streaming:

Dynamisk, adaptiv streaming over HTTP (DASH)



- Dele videoen i segmenter: fullstendig uavhengige små filmer
- Velge varighet på segmentene: 2-10 sekunder vanlig
- Velge antall kvalitetslag
- Velge tilpasningsstrategi
  - Klienten velger, ikke tjeneren
  - Strategien hos avspilleren er det som utgjøre forskjellen

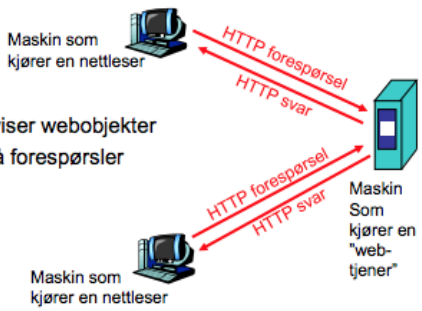


11.4.1 World Wide Web (www):HTTP-protokollen

HTTP: hypertext transfer protocol

- Applikasjonslagsprotokollen for Web
- Klient-/tjenermodell
  - *klient:* nettleser som spør etter, får og viser webobjekter
  - *tjener:* sender webobjekter som svar på forespørsler

- Tre hovedversjoner:
- HTTP/1.0 (1990)
- HTTP/1.1 (1999)
- HTTP/2 (2015)



11.4.2 HTTP-protokollen

HTTP: bruker TCP som transport:

- Klienten oppretter en TCP-forbindelse (socket) til tjeneren, port 80
- Tjeneren godtar TCP-forbindelsen fra klienten
- HTTP-meldinger (protokollmeldinger på applikasjonslaget) utveksles mellom nettleseren (HTTP-klient) og Webtjeneren (HTTP-tjener)
- TCP-forbindelsen lukkes

HTTP er "stateless"

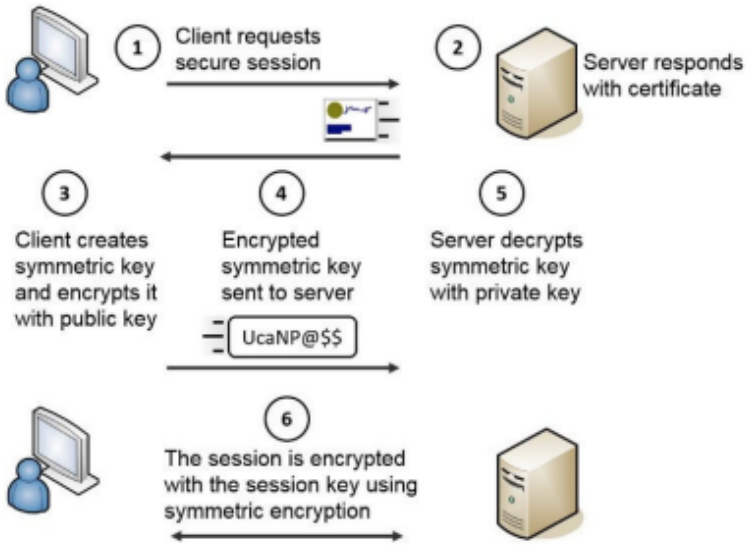
- Tjeneren sparer ikke på tilstandsinformasjon om tidligere forespørsler.

Protokoller som sparer på "tilstand" er komplekse!

- Tilstanden må vedlikeholdes

- Om en tjener eller klient "kræsjer", kan tilstanden bli ulik mellom dem. Da må den gjenopprette

11.4.3 HTTP med SSL (Secure Sockets Layer)



11.4.4 Persistente og ikke persistente forbindelser

Ikke-persistent

- HTTP/1.0: tjeneren leser forespørselen, svarer, lukker TCP-forbindelsen
- 2 RTTer for å hente objektet - TCP-forbindelse, forespørsel og overføring
- Hver overføring lider av TCP sin gradvise økning i senderate (slow start)
- Mange nettlesere åpner flere parallelle forbindelser

Persistent

- Standard for HTTP/1.1
- over samme TCP-forbindelse: tjeneren leser forespørselen, svarer, leser ny forespørsel...
- Klienten sender forespørsler for alle de objektene den trenger så fort den mottar hoveddokumentet (HTML)
- Færre RTTer, mindre slow start forbindelser

Persistent med "pipelining"

• Spør etter mange objekter på én gang (enda færre RTTer)

• Svaret kommer i serie etter hverandre i rekkefølgen forespørslene ankom tjeneren.

11.4.5 HTTP/1.x

response statuskoder

• 200 OK: Forespørselen var vellykket, objektet kommer senere i meldingen

• 301 Moved Permanently: Objektet har byttet plassering. Referanse til ny plassering kommer senere i meldingen.

• 400 Bad Request: Forespørselen var uforståelig for tjenere

• 404 Not Found: Dokumentet ble ikke funnet på tjeneren

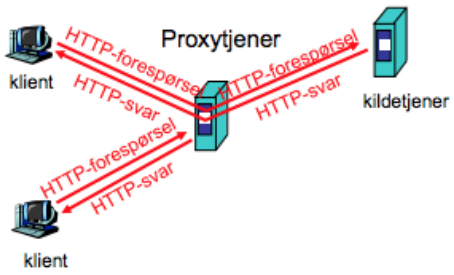
• 505 HTTP Version Not Supported

11.5 Cookies

Cookies tar vare på "tilstanden". Tjeneren lager et cookie #, tjeneren husker #, senere brukt til: autentisering og huske brukerpreferanser, tidligere valg. Produkter brukeren har sett på o.l. Tjeneren sender cookien til klienten i svarmeldingen Set-cookie: 1678453. Klienten legger ved cookien med etterfølgende forespørsler cookie: 1678453

11.6 HTTP Proxytjener

Mål: levere svar til klienten uten å gå helt til kilden
Brukeren konfigurerer nettleseren: Web skal gå via proxytjener
Klienten sender alle HTTPforespørslene til proxytjeneren
– Om objektet er i web cache: Proxytjeneren leverer objektet
– Om objektet ikke er i web cache, sender Proxytjeneren en forespørsel til kilden og lagrer svaret før den sender svaret til klienten.



Antagelse: cachen er nærmere klienten (f.eks i samme nettverk) =¿ raskere svar, mindre langdistansetrafikk.

11.7 Epost

Hovedkomponenter

– "mailklienter" Message User Agent (MUA)

– "mailtjenere": Mottak av meldinger / Videresending av meldinger

- Mail submission agent (MSA)
- Mail transfer agent (MTA)
- Mail delivery agent (MDA)
- Mail retrieval agent (MRA)
- Ofte realisert som én komponent kalt Message Handling Service (MHS)

MUA (eller "epostleser")

– Skrive, redigere, organisere og lese eposter

– utgående, innkommende lagres på eposttjener

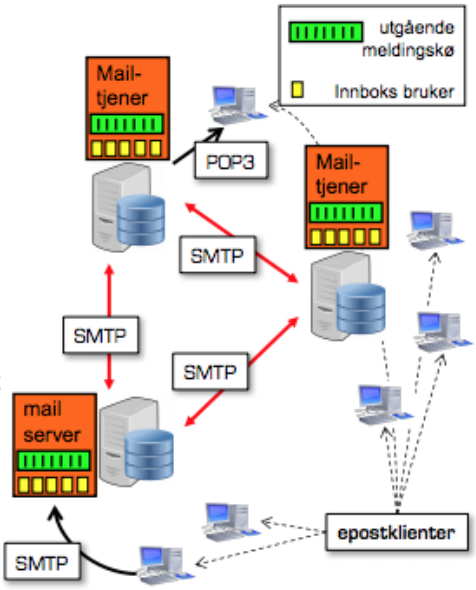
11.7.1 Epost: tjenere

Mailtjenere

- *mailbox* inneholder innkommende meldinger (hittil uleste) til brukeren
- *meldingskø* av utgående epostmeldinger (for sending)

Simple Mail Transfer Protocol (SMTP)

- Mellom eposttjenere for å sende epostmeldinger
- klient: tjener som skal sende en epost
- tjener: den som mottar eposten



Bruker TCP til å, pålitelig, overføre epost fra klient til tjener. Standardisert port: 25. Tre fase overføring: håndtrykk, overføring av beskjeder og avslutning.

11.8 MIME: Multipurpose Internet mail extension

```
Content-Type: type/subtype; parameters
```

Text

- eksempler på undertyper: **plain, html**

Image

- eksempler på undertyper: **jpeg, gif**

Audio

- eksempler på undertyper: **basic** (8-bit mu-law encoded), **32kadpcm** (32 kbps coding)

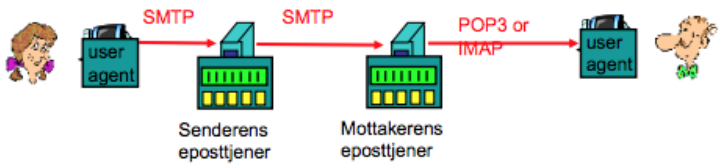
Video

- eksempler på undertyper: **mpeg, quicktime**

Application

- Andre data som må leveres til et eksternt program før det kan vises
- eksempler på undertyper: **msword, octet-stream**

11.8.1 Protokoller for mailtilgang



SMTP: leverer til mottagerens eposttjener
Mail access protocol: henter eposten fra tjeneren
- POP: Post Office Protocol
  - autorisering(agent <==> server) og nedlasting
- IMAP: Internet Mail Access Protocol (*Interim*➔*Interactive*➔*Internet*)
  - flere funksjoner (mer kompleks)
  - manipulere meldinger lagret på tjeneren
- HTTP: Hotmail , Yahoo! Gmail, etc.

11.9 Internet of Things (IoT)

Enheter i daglig bruk / husholdningen / arbeidsprosesser som leverer ekstra tjenester ved hjelp av Internett

Kan være med på å revolusjonere hverdagen

– Automatiske strømmålere

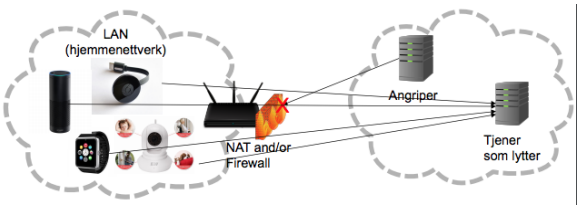
– Helseapplikasjoner

– Smarthus / Smartbyer

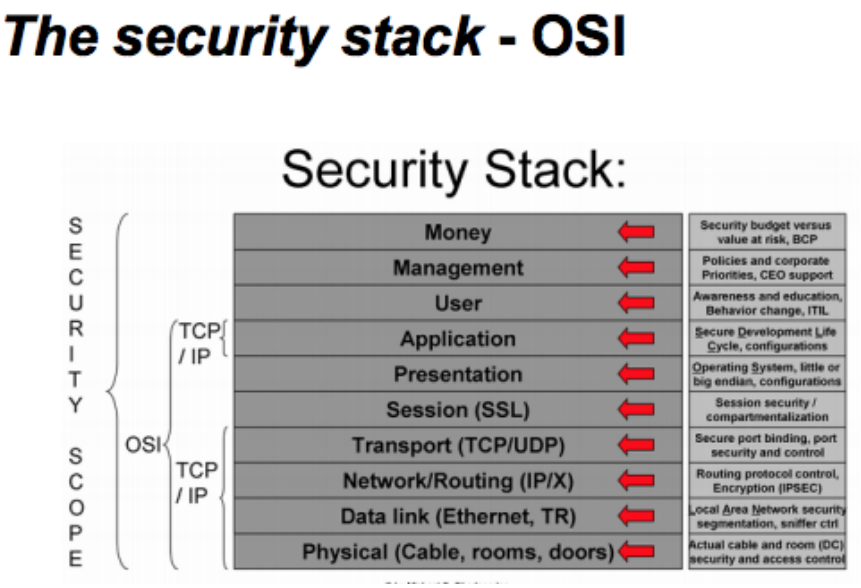
– Logistikk

Snakker med tjenere i "skyen"

Leverer data som brukes til analyse og tjenestelevering



12 Datateknologi og sikkerhet



12.1 Hva er sikkerhet?

Sikkerhet handler om å beskytte verdier mot skade og ødeleggelse

Verdi har i denne konteksten vid betydning:

– Helse, miljø, informasjon, sensitiv informasjon, penger, gjenstander, omdømme, etc.

Bekymringen er skadelige hendelser, tilsiktet eller utilsiktet

– Naturkatastrofer, krig, ulykker, skade påført med hensikt, osv.

Trusselbildet vil variere fra system til system.

12.1.1 Informasjonssikkerhet (ofte også omtalt som IKT-sikkerhet/IT-sikkerhet)

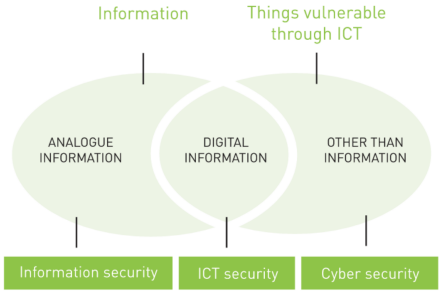
– Sikre informasjon

– Beskytte informasjonsressurser

- IT-utstyr, infrastruktur, datafiler, programvare

Cybersikkerhet: Sikre ting som er sårbare via IKT

12.1.2 Informasjonssikkerhet vs cybersikkerhet



God og dårlig oversettelse

Engelsk		Norsk	
• Security	→	• Sikkerhet	✓
• Safety	→	• Trygghet	
• Certainty	→	• Visshet	
• Security • Safety • Certainty	→	• Sikkerhet	✗

12.2 Økende behov for informasjonssikkerhet

Ville det ikke være mulig å løse alle sikkerhetsproblemer en gang for alle?

Nei, fordi:

– Hurtig innovasjon i nye IT-produkter og tjenester skaper også nye sårbarheter og insentiver for angrep

– Flere og flere tjenester er eksponert online

– Informasjonssikkerhet ignoreres ofte når utviklere har tidspress

– Kriminalitet følger nye tjenester og forretningsmodeller

– Forbrukere rangerer ny teknologi framfor sikker teknologi

– Mer effektive angrepsverktøy utvikles

– Økende antall trusler

Konklusjon: Informasjonssikkerhet er en kontinuerlig prosess for å oppdage og hinder trusler og å fjerne sårbarheter

12.2.1 Hvorfor dere skal lære om IT-sikkerhet

Bidra til å skape sikre(re) løsninger

– Utvikling av IT-løsninger uten tanke på sikkerhet vil føre til sårbare løsninger.

Være rustet til å rådgi andre (for eksempel bedriftsledere).

Å vurdere sikkerhet bør ligge nærmest som en ryggmargsrefleks i alt dere gjør etter endt utdanning ved Ifi.

12.3 Sikkerhetsmål og sikkerhetstiltak

Sikkerhetsmål:
– Er uavhengig av spesifikk implementering.
– Kan ivaretas med ulike sikkerhetstiltak.
Sikkerhetstiltak:
– Er basert på spesifikk implementering, ofte bundet til produkter.
– Er tiltak for å oppdage, forebygge eller gjenopprette.

Sikkerhetsmål for informasjonssikkerhet:

Opprettеholde CIA (Confidentiality, Integrity, Availability).

Autentifikasjon, det vil si å skape tillitt til ekteheten av en oppgitt identitet:

– En bruker

– Et system, en maskin

– Opprinnelsen til data eller til en melding

Uavviselighet, det vil si å sikre at mottager eller avsender ikke kan bestride å ha sendt eller mottatt en melding.

Sporbarhet, det vil si å sikre at en gitt hendelse kan spores til en gitt identitet.

12.3.1 CIA (eller KIT på norsk)

Konfidensialitet: Å sikre at kun de med rettmessige behov har tilgang til en ressurs.

– Sikkerhetstiltak: Kryptering, tilgangskontroll, perimetersikring

Integritet: Å sikre at en ressurs (eks. data eller kode) ikke endres eller slettes utilsiktet.

– Sikkerhetstiltak: Tilgangskontroll, endringskontroll, kryptografisk sjekksumalgoritme, perimetersikring

Tilgjengelighet: Å sikre at en ressurs er tilgjengelig for rett person til rett tid.

– Sikkerhetstiltak: Sikkerhetskopier, redundante tjenester, gode rutiner for hendelseshåndtering og gjenoppretting.

12.4 Autentisering

Identifikasjon: Hvem du påstår at du er.

- Brukernavn, biometri

Autentifikasjon: Legitimere at du er den du utgir deg for å være.

– Brukerautentisering, ofte basert på

- Noe du har (ting, gjenstand)
- Noe du kjenner (passord, pin)
- Noe du er (biometri)
- Eller en kombinasjon av disse

– System- eller maskinautentisering

- Ivaretas av kryptografiske autentiserings-protokoller (f.eks. systemsertifikater som verifiseres av tredjepart) som TLS, VPN og IPSec

– Dataopprinnelse

- Ivaretas gjennom kryptografiske mekanismer, PKI, digital signatur, elektronisk signatur

12.4.1 Brukerautentisering

Kunnskapsbasert:
– Den vanligste autentikatoren
– ≪En hemmelighet du kjenner≫
- Passord, PIN-kode, ol

– Utfordringer?
- Lett å glemme, lett å gjette, blir kanskje lagret i minne eller cache på datamaskinen

– Passordhygiene viktig
- Lengde, sammensetning, bruk av ord, passordfraser

Tokenbasert:
– ≪Noe du har≫
– Engangspassord
- Noe som genererer engangspassord på bakgrunn av en algoritme (tellerbasert eller klokkebasert)
- Kodegenerator, kodekort, SMS, mm.

– Kontaktløse brikker (RFID, RadioFrekves Id)
– Kryptografiske protokoller

Biometri:
– ≪Noe du er≫
– Ansiktsgjenkjenning
– Fingeravtrykk
– Retina/iris scanning
– Signatur (elektronisk eller skriftlig)
– Id-kort
– Utfordringer
    • Unikt nok?
    • Sikkert nok? Presentation attacks

Flerfaktor:
Når kun en autentikator ikke er sikkert nok. Passord kan gjettes, tokens kan mistes/stjeles. Derfor benyttes ofte en kombinasjon av flere mekanismer. – To-faktor autentisering
    • Oftest en kombinasjon av noe du vet og noe du har
    • Øker sikkerheten, men autentisering blir mer tungvint
    • Økende antall tjenester tilbyr 2FA
    • Økende antall applikasjoner og dingser for generering av engangspassord

12.5 Uavviselighet

Samme kategori som dataopprinnelse, men med et tilleggskrav om bevis for at en melding er mottatt.

Realiseres som regel ved bruk av digitale signaturer

12.6 Sporbarhet

Å kunne knytte en gitt identitet til en gitt hendelse.

Ivaretas gjennom

– Logging av alle hendelser

– Identifisere alle identiteter

– Perimeterforsvar, ikke gi uvedkommende tilgang

Ved innbrudd vil en angriper mest sannsynlig forsøke å fjerne spor.

12.7 Sikkerhetstiltak

ulike faser:
– Preventive (forebyggende) tiltak:
- Forhindre og avskrekke angrepsforsøk. Eksempel: kryptering av filer for konfidensialitet

– Detekterende tiltak:
- Varsle angrep som forsøkes eller som allerede er skjedd. Eksempel: Inntrengingsdeteksjon (IDS)

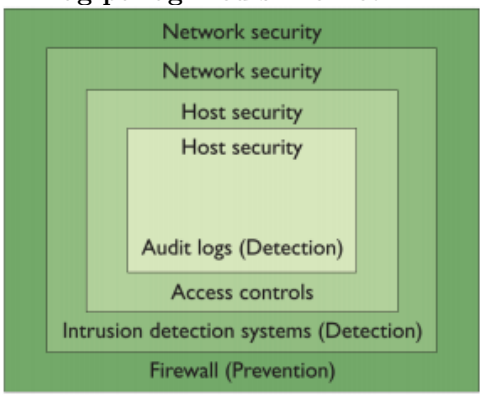
– Korrigerende tiltak:
- Gjenopprette skade på dataressurser etter angrep. Eksempel: Hente sikkerhetskopi av programmer og data ved tap/kompromittering av ressurser.

– Det er alltid nødvendig å benytte en kombinasjon av tiltak fra alle tre faser for å opprettholde dekkende beskyttelse

ulike tilstander:
– Informasjon kan befinne seg i ulike tilstander
– Ulike tilstander krever ulike sikkerhetstiltak:
- Under lagring
- Under overføring
- Når den er i bruk

– Informasjon må beskyttes i alle tilstander

Lag på lag med sikkerhet



Eller: Sikkerhet i alle ledd
Hvis en inntrenger kommer seg gjennom ett sikkerhetslag, vil du ønske å kunne stoppe dem i det neste. Sørg for:
- Fysisk sikring
- Sikker lagring
- Sikre datamaskiner og systemer
- Nettverkssikkerhet
- Sikker kommunikasjon
- IT-systemer uten kjente feil og sårbarheter
- Å ivareta personvern

12.8 Terminologi

Trussel (threat): Uønsket hendelse som kan inntreffe.

Angrep (attack): Noen forårsaker med hensikt en uønsket hendelse.

Sårbarhet (vulnerability): En sårbarhet i et system som gjør et angrep mulig.

Utnyttelse (exploit): Noe (en oppskrift, et program, en implementasjon) som kan utnytte en sårbarhet.

Risiko (risk): Sannsynligheten for et angrep multiplisert med tap i kroner.

12.9 Skadelig programvare, skadevare

(Engelsk: Malicious software eller malware)

• Samlebetegnelse på programvare som med hensikt er inkludert eller plassert i et datasystem for å gjøre skade.

• Skadevaren kan like gjerne utnytte sårbarheter i kjernen som i applikasjoner eller andre programmer.

• Skadevaren kan være avhengig av å være en del av et vertsprogram, eller er et enkeltstående program som kan kjøres som en egen prosess av operativsystemet.

• Noe skadevare formerer seg og infiserer andre systemer, som regel avhengig av en trigger.

12.9.1 Ulike typer skadevare

Gruppene er til dels overlappende, og ikke alltid presist definert:

Virus	Kjennetegnes ved at det sprer seg hver gang en fil med virus blir åpnet av bruker eller systemet. Eks. løsepengevirus.
Ormer	Skadevare som på egenhånd er i stand til å spre seg og finne nye ofre.
Trojaner	Skadevare som utgir seg for å være et nyttig program.
Bakdør	En ubeskyttet og ukjent åpning inn i et datasystem, laget av en angriper.
Spionvare	Programvare for å spionere på en bruker i håp om å få tak i f.eks. informasjon, passord, kontoinformasjon.
Logisk bombe	Skadevare som kjøres på et forhåndsgitt tidspunkt eller ved en bestemt handling eller hendelse.
Tastelogger (Keylogger)	Enhet eller programvare som er koblet til tastatur eller maskin og som lagrer alle tastetrykk.
Rootkit	Skadevare som endrer funksjoner i OS eller applikasjoner, med mål å ta kontroll over en maskin eller et system.

12.10 Datakriminalitet

Drivkraft
• Sabotasje
• Ideologi («hacktivisme»)
• Politikk
• Skade omdømme (konkurrerende virksomheter?)
• Økonomisk vinning (spam, selge exploits, utnytte kontoinformasjon, Få ting «gratis»)
• Utpressing (kryptovirus, unnlate å skade omdømme for en gitt sum)
• Anerkjennelse • «Fordi vi kan»
• Nysgjerrighet (nabo, venn, foreldrene dine?)

Hvem angriper?
Typiske angripere
• Nysgjerrige enkeltpersoner
• Venner og familie
• Uærlige personer — for personlig vinning, spare penger
• Hackers, crackers og script kiddies — for utfordring og anseelse
• Virksomheter/firmaer — forretningsvirksomhet og markedsføring (?)
• Organisert kriminalitet — for penger
• Myndigheter og sikkerhetsbyråer — NSA, SVR, GCHQ, DGSE, etc.
• Militær SIGINT — strategisk og taktisk virksomhet, cyber-forsvar
«Innsidebrukere» er ofte den største trusselen
• Ansatte, administratorer, tjenesteleverandører, kunder, familiemedlemmer
Hvilke angripere vi bryr oss om varierer
• Hvem bryr du deg om at leser dagboka eller e-posten din?

12.10.1 Datakriminalitet - hvordan

En angriper vil som regel:

• tilegne seg urettmessig tilgang til systemer. Utnytte systemet, ha tilgang til informasjon, forfalsker eller sletter informasjon.

• utilgjengeliggjøre systemer eller tjenester (DDoS eller DoS). Benytter som regel kompromitterte datamaskiner/systemer.

• avlytte eller forfalsker nettverkstrafikk for å få tilgang til eller manipulere informasjon.

• skape og utnytte tillitt (sosial manipulering) for å tilegne seg urettmessig tilgang eller gevinst

12.11 Sosial manipulering (Social engineering)

• Å skaffe seg tilgang ved å skape tillitt og deretter svindle. Utnytte menneskelig svakhet.

• Mest brukte, enkleste, og kanskje det eldste trikset...

• Berører alt fra privatpersoner til store virksomheter.

• Vanlige innfallsvinkler:

– E-post som inneholder eller lenker til skadevare (phishing):

* Fra noen du kjenner

* Fra kjent virksomhet (Posten, IKEA, banken din, etc)

* Om stor gevinst eller utbetaling som venter

– Kontaktes av noen som ber om informasjon (passord, kontoinformasjon) eller å gi dem tilgang til en ressurs (fysisk, et system eller datamaskinen din).

12.11.1 Sikkerhetstiltak

– Opplæring!

– Stopp. Tenk. Klikk. Angripere vil at du skal handle først, tenke etterpå.

– Virker det for godt til å være sant så er det mest sannsynlig det...

– Hold alltid operativsystem, drivere, programvare oppdatert med siste versjon.

12.12 Kryptering

Krypteringsalgoritmer:

– Symmetrisk

• En nøkkel, både sender og mottager må kjenne nøkkelen.

– Asymetrisk

• Nøkkelpar, privat + offentlig nøkkel.

– Sjekksumalgoritmer (hash-funksjoner)

• Enveiskryptering, med sjekksum.

• Ulike algoritmer med ulik styrke.

12.12.1 Krypteringsnøkler

Styrken i kryptografisk sikkerhet avhenger av

- Størrelsen/lengden på nøkkelen.
- Styrken i den kryptografiske algoritmen.
- Håndtering og beskyttelse av nøklene

Nøkkelhåndtering er basisen for sikker generering, lagring, distribusjon og destruksjon av nøkler.

En og samme nøkkel skal kun brukes til ett formål

- Eks. kryptering, autentisering, generering av digital signatur.

Nøkler klassifiseres gjerne:

- Offentlige, private, symmetriske..
- Hva er de ment å brukes til.

12.13 Public key infrastructure (PKI)

• PKI er et rammeverk for kryptering, autentisering og signering av dokumenter eller programvare.

• PKI er basert på asymmetrisk kryptering, med et nøkkelpar (offentlig og privat nøkkel).

• Nøkkelen er bakt inn i et elektronisk sertifikat som knytter nøkkel og identitet sammen.

• Et sertifikat utstedes av en betrodd virksomhet (Certificate Authority).

• Den private nøkkelen kan oppbevares på en datamaskin, på SIM-kortet i mobilen, på et smartkort, i banken (Bank ID).

• Den offentlige nøkkelen må spres til mottagere som skal kryptere meldinger

12.13.1 PKI i bruk

• Sikker kommunikasjon i helsevesenet:

- eResept, elektronisk sykemelding, etc
- Knytter melding til individ (signatur med personlig sertifikat for lege) og sikre konfidensialitet (kryptering av forsendelse).

• Publikumsrettede PKIer i Norge:

– BankID, Buypass, Commfides,

– Difi tilbyr en felles infrastruktur for innlogging til offentlige tjenester, basert på elektronisk signatur.

• Web PKI eller Browser PKI

– Nettlesere kjenner den offentlige nøkkelen til alle betrodde sertifikatutstedere (CA), og bruker denne til å forsikre seg om at sertifikatet er gyldig.

12.14 Nettverkssikkerhet

Tradisjonell sikkerhetsmodell: Stoler på endesystemene, nettverket anses upålitelig. • Endesystemer

sender og mottar meldinger til og fra nettverket, og nettverket kan levere, slette, endre eller forfalske meldingene.

• Metaforer: Upålitelig postmann, oppslagstavle som alle kan lese.

12.14.1 Sikkerhetstrusler

• Tradisjonelle trusler:

– Avlytting: Uvedkommende lytter til nettverkstrafikk og får tak i konfidensiell informasjon. Passive angrep.

– Forfalskning: Uvedkommende sender ikke-autentiske meldinger eller later som de er en annen. Aktive angrep.

– Data modifiseres (Man in the Middle, MitM): Uvedkommende griper inn i kommunikasjonen og modifiserer data. Aktive angrep.

- Berørte sikkerhetsmål:
  - Konfidensialitet
  - Integritet
  - Autentisering av opphav til data/melding.
- Andre trusler: (D)DoS, uautorisert bruk, ormer, etc.

12.14.2 Konsepter

Vi antar at alle virksomheter eier et nettverk.

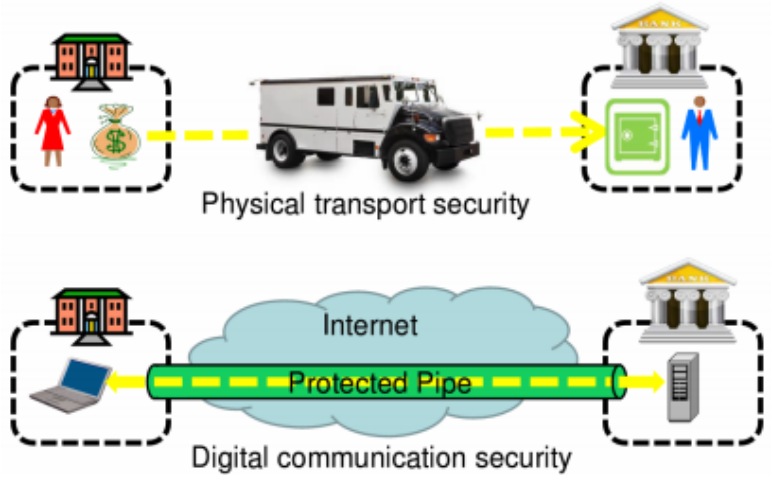
– Ønsker å beskytte sitt eget lokale nettverk.

– Ønsker å beskytte kommunikasjonen med andre nettverk. **Nettverkssikkerhet** kan dermed sies å bestå av to hovedområder:

– **Kommunikasjonssikkerhet**: Beskytte data i transporten mellom virksomheter (og endenoder).

– **Perimeterforsvar**: Beskytte en virksomhets nettverk mot uautorisert aksess fra omverdenen.

12.15 Kommunikasjonssikkerhet



12.16 Sikre protokoller

• En rekke protokoller for sikker kommunikasjon til ulike formål er spesifisert og implementert, mer eller mindre vellykket:

  – For sikker autentisering, integritet, konfidensialitet, nøkkelutveksling, etc.

• To av de mest benyttede protokollene er

  – Transport Layer Security (TLS)

• Sikkerhet i transportlaget (lag 4 i OSI-modellen), benyttes bl.a. omfattende for webtjenester (https, port 443).

• Basert på PKI, hver tjener må ha et tjenersertifikat og en privat nøkkel.

  – IP Security (IPSec)

• Sikkerhet i nettverkslaget (lag 3 i OSI-modellen), brukes bl.a. for å tilby VPN (Virtual Private Network), krever støttet i OS.

• Baserer seg på kryptering, autentisering og nøkkelhåndtering.

• Sårbarheter finnes dessverre likevel:

  – Protokollene er f.eks. ikke sikrere enn algoritmene de baserer seg på...

12.17 Perimeterforsvar

• Brannmur - førstelinjeforsvar

  – Avgjør hva som skal slippes inn i og/eller ut fra et lokalt nettverk.

  – Fungerer som en slags tilgangskontroll i nettverket, avverger uautorisert tilgang til/fra et privat nett.

  – Implementeres i programvare eller maskinvare (egen hardware laget for akkurat dette formålet)

  – Personlig brannmur: Programvare som beskytter maskinen det er installert på.

• Innbruddsdeteksjon (IDS)

– System som detekterer mistenkelig aktivitet, nettverksanalyse

– Kan detektere:

* Misbruk, både forsøk og suksessfulle innbrudd

* Skadevare (virus, ormer, trojanere)

* DoS-angrep

12.18 Trådfast vs trådløs forbindelse

Kommunikasjon i trådløse nett går via radiosignaler, og skaper derfor andre utfrordringer.

– Radiosignaler følger ikke fysiske grenser, hvilket betyr sårbarhet for avlytting også utenfor f.eks. en bygning.

– Uautorisert nettverkstilgang (internett, intranett).

– Falske aksesspunkter. En trådløs enhet vil koble seg til APet med sterkest signal.

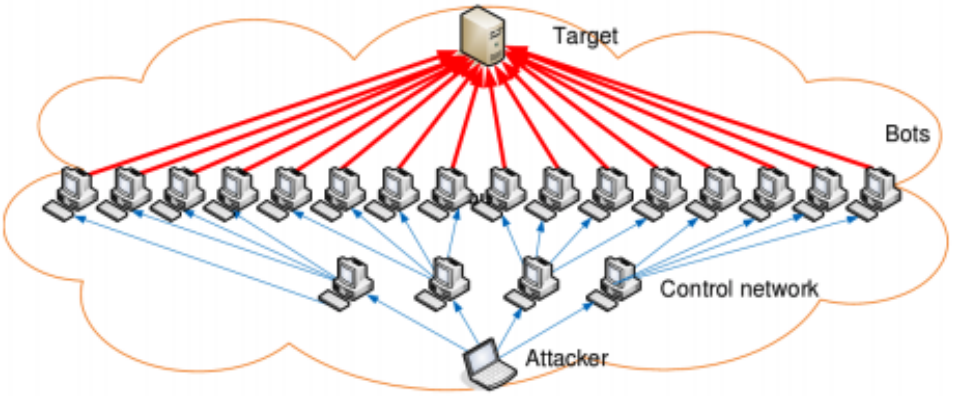
– Signal-forstyrrelser (tilsiktet eller utilsiktet) kan føre til utilgjengelighet (DoS og DDoS)

– Aksesspunkter kan kompromitteres, spesielt åpne nett/aksesspunkter.

12.19 DoS (Disk Operating System) og DDoS (distributed denial-of-service)

• Truer tjenesters tilgjengelighet.

• Angripere kontrollerer tusenvis av kompromitterte datamaskiner og kjører koordinerte angrep (packet flood) mot en tjeneste.



13 Datateknologi og sikkerhet II

Beskytte digital informasjon og verdier. Beskytte informasjonsressurser mot skadelige hendelser, tilsiktet og utilsiktet.

13.1 Kryptering

- Gjøre informasjon uleselig for uvedkommende
  - Hash-algoritmer (enveis-kryptering)
  - Symmetrisk (èn nøkkel)
  - Assymetrisk (nøkkelpar: Offentlig + privat nøkkel)
- Hvordan dele nøklene, og hvordan være sikre på at nøklene er ekte når de deles?
- Løsning: Et sertifikat som følger nøkkelen knytter sammen en ofentlig nøkkel og en identitet.
- PKI er et rammeverk for å håndtere bruk av digitale sertifikater.

13.2 DNS-sikkerhet

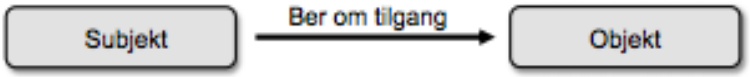
- Navneoppslag, knytter maskinnavn til IP-adresse
  - devilry.ifi.uio.no -¿ 129.240.65.28
  - En maskin forespør typisk en navnetjener om adresse.
  - Kommunikasjonen foregår i klartekst
- Sårbart:

- DNS-forfalskning
    - Uvedkommende introduserer uriktige opplysninger i navnetjener
- DNS-modifisering
    - Man in the Middle-angrep, forfalsker meldinger
- DNS-kapring
    - Endre hvilke navnetjener offerets maskin benytter (hacke maskinen).

Moral: Stol aldri ubegrenset på et domenenavn. Vanskelig å oppdage feil.

13.3 Tilgangskontroll

- Et subjekt vil utføre handling på objekt
    - Anne vil lese en fil



    - Per vil oppdatere kontobalansen på en bankkonto
    - En prosess vil åpne en nettverksforbindelse
- Tilgangskontroll = autentisering + autorisasjon
    - Autentisering = bekrefte identiteten til subjektet
    - Autorisasjon = sjekke at subjektet har tilgang til å utføre den ønskede handlingen på objektet etter forhåndsdefinerte regler.

13.4 Referansemonitor

- Referansemonitorens oppgave er å kontrollerer tilgang fra subjekt til objekt
    - Tilgang innvilges eller avvises
    - Følger policyen som er satt (eks. av administratorer eller brukere.
    - Logger hendelser til hendelseslogg (ivaretar sporbarhet).

13.5 Tilgangskontroll

i UNIX

• Brukere (brukernavn, UID) og grupper (gruppenavn, GID) er grunnleggende elementer, begge er permanente identiteter.
• Brukere har en brukerkonto.
• Brukere tilhører en eller flere grupper, en av gruppene er primærgruppe.
• Subjekter i UNIX:
  – En prosess
  – En prosess har en ekte UID og en effektiv UID. Tilsvarende for GID.
  – Ekte UID/GID: Arves fra foreldreprosessen, typisk UID/GID til innlogget bruker
  – Effektiv UID/GID: Arves også fra foreldreprosessen, eller fra programmet som kjøres.
• Objekter i UNIX:
  – Filer (programmer), mapper og devicer (maskinvareenheter, representert som filer)

Hvem angriper?

på flere nivåer

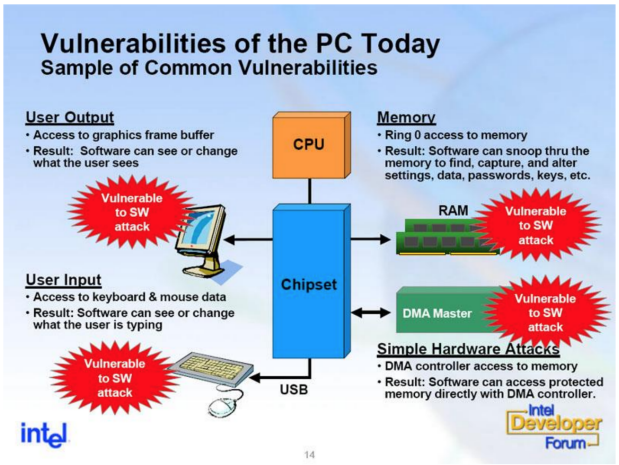
Eksempler:

• Brukere og passord kan kontrolleres i applikasjonslaget (OSI-modellens lag 7)

• Tilgang til ulike nettverksporter (til og/eller fra) kan kontrolleres i transportlaget (OSImodellens lag 4)

• Tilgang basert på IP-adresser (til og/eller fra) kan kontrolleres i nettverkslaget (OSImodellens lag 3)

13.6 Trussel mot datamaskiner i dag.



13.7 Sikkerhet i operativsystemet

Dere har tidligere lært:

• Kjernen i operativsystemet styrer alt.

• Kjernen kommuniserer med periferenhetene i datamaskinen gjennom drivere.

• Brukerprosesser må kommuniserer via kjernen for å kunne benytte periferenhetene.

• Minnet i en datamaskin benyttes (bl.a.) til mellomlagring.

• Kjerne og drivere er dataprogrammer som kan inneholde sårbarheter.

• Kjernen kjører i supervisor mode, og har tilgang til alt -¿ en sårbarhet i kjernen er relativt kritisk

• Svært viktig å holde både operativsystem (kjerne) og drivere oppdatert for å unngå at uvedkommende utnytter eventuelle sårbarheter.

13.8 Lagring av data

• Data kategoriserer gjerne etter hvilket beskyttelsesbehov det har:

  – Åpen informasjon

  – Informasjon med begrenset beskyttelsesbehov

  – Informasjon med sterkt beskyttelsesbehov

• Hvilke sikkerhetstiltak som iverksettes avhenger av beskyttelsesbehovet:

  – Kryptering, tilgangskontroll, perimeterforsvar, sikkerhetskopiering

13.9 Kryptering av data

• Kryptering av data bidrar til å sikre konfidensialitet

  – Laptop mistes/stjeles

  – Server/tjener kompromitteres

• Krypteringsstrategier

  – **Filkryptering**: Enkel kryptering, bruker har kun en nøkkel som en fil krypteres og dekrypteres med.

  – Kryptering av **filsystem**: Kryptering vil ofte være et attributt som kan velges, også for alle filer. Baserer seg gjerne på bruker-autentisering.

– **Fulldiskkryptering**: Alt på disken krypteres. Kan gjøres i programvare eller maskinvare (TPM)

• Lost decryption key = lost encrypted files...

– Sikkerhetskopier også nøkler!

13.10 Sikkerhetskopiering

• Strategi for å ta vare på data over tid, og kunne gjenopprette ved uønskede hendelser som endrer eller sletter data utilsiktet.

• Tar vare på versjoner av data over tid.

• Sikkerhetskopien må også sikres: Lagres trygt adskilt fra opprinnelige disker/data.

• Sikkerhetskopi =/= datasynkronisering

13.10.1 Sikker avhending av datautstyr

Destruere data:

• Å slette en fil markerer plassen på disk som ledig, men rådataene finnes fortsatt.

• Å overskrive en fil er heller ingen garanti: – Filsystemer organiserer ofte data på uortodokse måter.

• Overskrive all ledig diskplass med tilfeldig data?

• Magnetisk behandling (remanens). Tja.

• Fysisk destruksjon – varmebehandling over Curie temperatur. Heller ikke 100% garanti

13.11 Applikasjonssikkerhet

• Applikasjoner må designes og utvikles med tanke på å kunne brukes trygt

• I designfasen: Trusselmodellering:

– Hvilke trusler ser vi?

– Hvilke tiltak kan iverksettes for å motvirke truslene?

• I programmeringsfasen: – Unngå å skape/tilrettelegge sårbarheter

13.12 Buffer overflow

• En sammenhengende seksjon i minnet i datamaskinen, allokert til å inneholde alt fra en streng til en array med heltall (integer), kalles et buffer.

• En feil (sårbarhet) i et program, som gjør at programmet når det kjøres kan overskride bufferets grenser i minnet, kalles buffer overflow.

– Eks: Mulig å plassere en lang tekststreng i et buffer som i programkoden er definert til å kun ha plass til 20 tegn.

13.12.1 Utnyttelse av buffer overflow

• En buffer overflow kan krasje et program, føre til korrupte data, eller i verste fall utnyttes til exploits.

• Kreative og dyktige (men uærlige?) programmerere kan utnytte en slik sårbarhet:

– Plassere skadevare på steder i minnet hvor det ligger eksekverbar kode.

– Endre en funksjons returadresse til å isteden peke til adressen i minnet der skadevare er plassert.

• Hvordan sikre seg mot slik sårbarheter?

– Sjekke all input fra brukere.

– Programmeringsspråk uten innebygget sjekk av buffergrenser er utsatt (C og C++)

13.13 SQL injection

• SQL er et spørrespråk mot relasjonsdatabaser.
• Relasjonsdatabaser brukes svært ofte i bakkant av web-applikasjoner for lagring av data/informasjon.
• SQL injection:
– Å «lure» inn spørre-setninger sammen med informasjon som er gitt som input i et grensesnitt.
– Muliggjøres på grunn av «slapp» programmering. Input fra brukeren valideres ikke før det sendes videre som spørresetning til databasen.

SQL injection eksempel

Eks: tabellen «Brukere»:

BrukerId	Brukernavn	Passord
1001	anneh	ewtgfst45yh
1002	pederaa	67efg3ndfa

Kan få tak i informasjon man ikke skal ha tilgang til:

Oppgi brukerId: 1001 OR 1=1

Kan resulterer i følgende gyldige SQL-setning:

SELECT * FROM Bruker WHERE BrukerId = 1001 OR 1=1;

13.14 Personvernlovgivning

• Datatilsynet: ≪Personvern handler om rettet til et privatliv og retten til å bestemme over egne personopplysninger.≫

• Personvern reguleres av Personopplysningsloven.

• Noen punkter:

– Personopplysning = Opplysning og vurdering som kan knyttes til en enkeltperson.

– Formålsbegrensning: Personopplysninger skal kun behandles for spesifikke, uttrykkelige, angitte og legitime formål.

– Retten til innsyn i opplysninger om deg selv.

13.14.1 EUs Personvernforordning (GDPR)

• EUs nye forordning for personvern blir norsk lov i 2018:

– Gjelder et større geografisk område: Regelverket omfatter virksomheter utenfor EU som tilbyr varer og tjenester til EU-borgere.

– Virksomheter får et større ansvar for personvern, skal selv vurdere personvernkonsekvenser.

– Krav til åpenhet: Informasjon om behandling av personopplysninger skal gis den registrerte på en klar og forståelig måte.

– Nasjonale datatilsyn får en plikt til å samarbeide internasjonalt.

– Krav om innebygget personvern i alle nye løsninger.

13.14.2 Innebygget personvern

• Ta hensyn til personvern i alle utviklingsfaser av et system eller en løsning.

• Det minst personverninngripende alternativet skal være standarden i alle systemer og løsninger:

– Mengden personopplysninger

– Omfang av behandling

– Lagringstid

– Tilgjengelighet

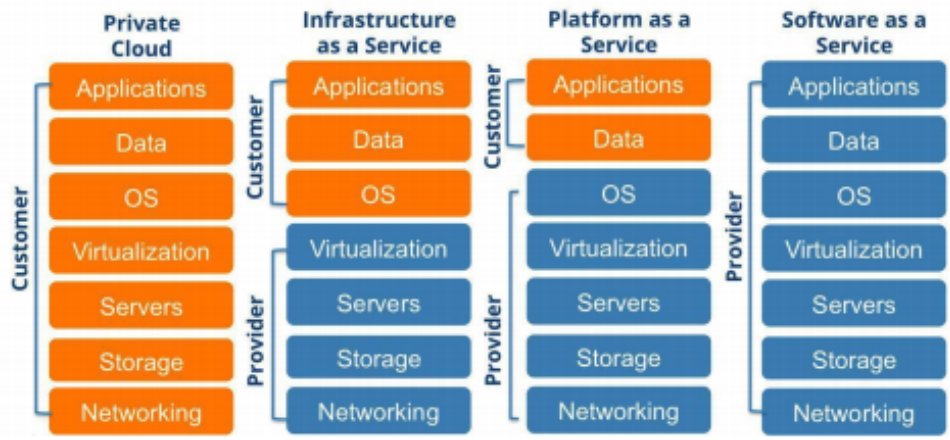
13.14.3 Innebygget sikkerhet

• Innebygd informasjonssikkerhet er et overordnet prinsipp i arbeidet med informasjonssikkerhet.

• Innebygd informasjonssikkerhet oppnås når informasjonssikkerhet er:

1. innarbeidet i virksomhetsstyringen og understøtter virksomhetens mål.
2. innarbeidet i virksomhetens prosesser og prosjekter fra starten av.
3. tatt hensyn til i hele livssyklusen til IKT-løsninger.
4. et tema alle ansatte i offentlig sektor kjenner til og vet hva innebærer for sine arbeidsoppgaver.

13.15 Skytjenester



13.15.1 Ikke gå i skyfella!

• **Fordeler**:

– Fordelene med skytjenester er blant annet reduserte infrastrukturkostnader, besparelser i form av betaling for faktisk bruk, og lokasjonsuavhengig tilgang.

• **Ulemper**:

– Dersom en virksomhet ønsker å benytte skytjenester til behandling av personopplysninger, er virksomheten juridisk ansvarlig for at personopplysningene prosesseres og lagres i samsvar med personvernregelverket.

13.16 Trusselmodellering

- Flere innfallsvinkler:
  - Hva er verdifullt i systemet, og hvordan kan det gå tapt?
  - Motivasjon for angrep: Hvem og hvorfor?
  - Hvilke deler inneholder systemet, og hvordan kan de feile?
  - Mottiltak: Hva kan gjøres for å forebygge og redusere angrep?
  - Lær av feil: Hva har gått galt tidligere?

13.16.1 Trusselbilde

13.16.2 Risikoanalyse

- Mange definisjoner, men:

Risiko = Sannsynlighet for angrep * kostnad eller tap

- I praksis litt meningsløst, hva betyr en risiko på 3,4?
- Hvor skal innsatsen legges?
  - Usannsynlig hendelse som medfører store tap?
  - Sannsynlige hendelser som har lavere kostnad?
- Sikkerhet kontra brukervennlighet

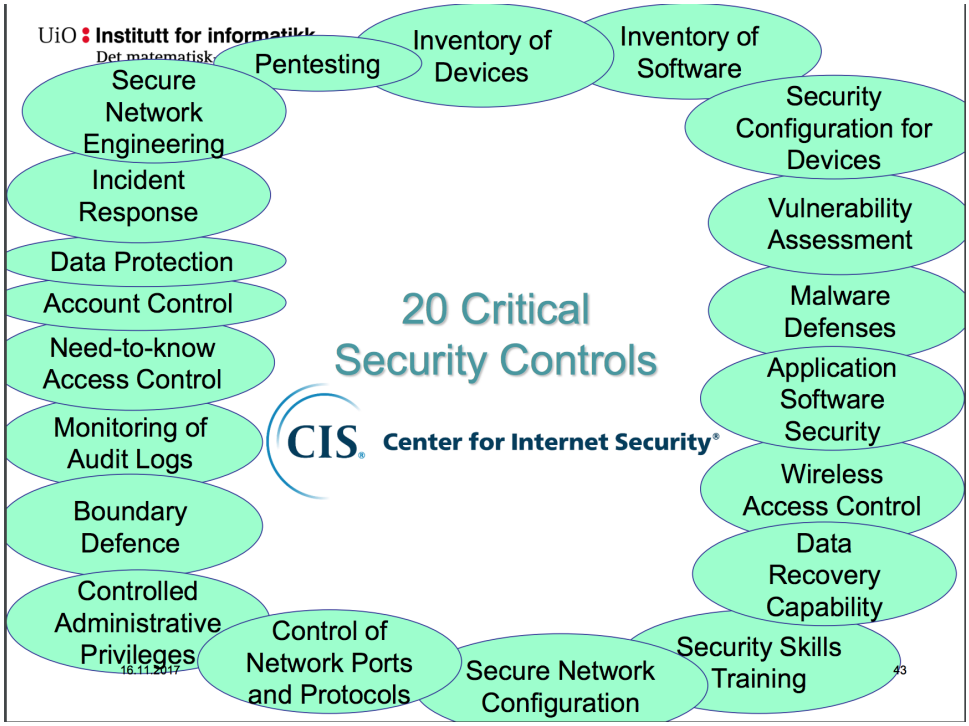
13.17 Styringsmodell for IT sikkerhet

«**Sikkerhetsstyring** er systematiske aktiviteter for å oppnå og opprettholde et sikkerhetsnivå i overensstemmelse med de mål og krav en organisasjon har satt seg»

Beskytte ressurser = skape verdi:

- Tillitt (fra kunder, forretningspartnere, investorer, ansatte)
- Rykte, merkevare
- Konkurransefortrinn
- Forebygge og redusere (økonomiske) tap
- Skape robusthet og sikre kontinuitet (håndtere hendelser og katastrofer)
- Øke selskapets verdi

13.18 20 CSC: Critical Security Controls



13.19 NSMs sikkerhets tips