UNIVERSITETET
I OSLO

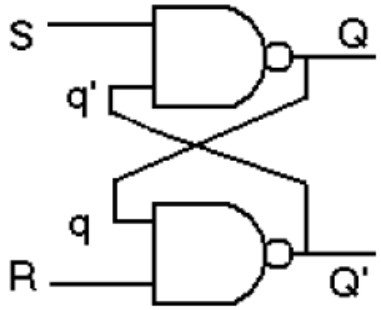
Dagens temaer

- Dagens temaer hentes fra kapittel 3 i læreboken
- Oppbygging av flip-flop'er og latcher
- Kort om 2-komplements form
- Binær addisjon/subtraksjon
- Aritmetisk-logisk enhet (ALU)
- Demo av 'Digital Works'

2008

Låsekrets: RS-Latch

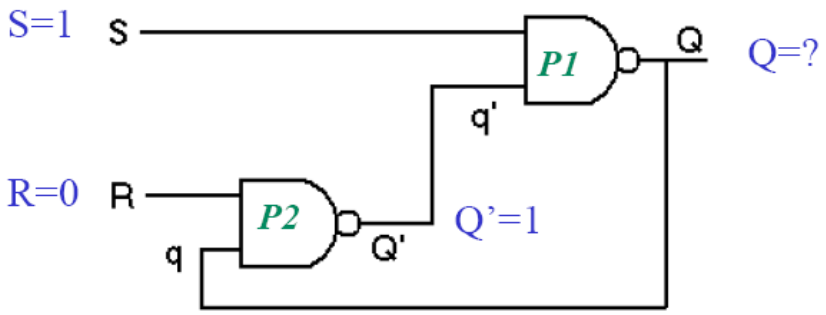
- Hukommelseselement med to innsignaler: S(et) og R(eset), og to utganger Q og Q'



- Q og Q' er definert som inverterte verdier av hverandre; utgangsverdien hentes normalt fra Q-utgangen alene.
- RS-latchen brukes sjelden direkte, (men andre typer hukommelsesceller bygger på den)
- RS-latchen er asynkron siden den ikke benytter et klokkesignal for å styre overgangen mellom to forskjellige verdier på utgangen skal finne sted.

Analyse av RS-Latch

- Analysèrer kretsen for S=1 og R=0

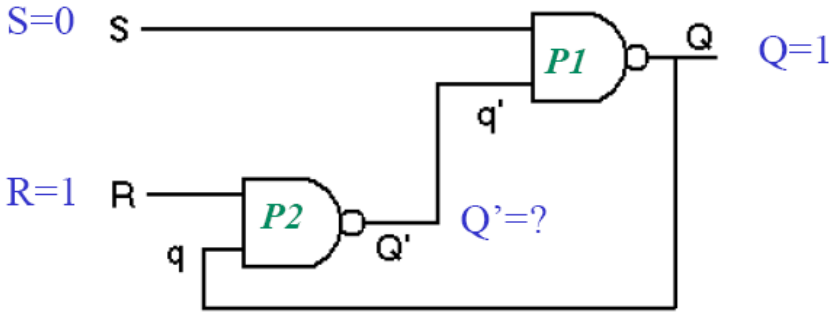


a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

- Kan ikke bestemme utgangen fra P1 (Q) uten å vite hva q' er
- Utgangen fra port P2 (Q') er 1, fordi 0 på en av inngangene til en NAND alltid vil gi 1 ut
- Dermed er Q' = q' = 1, og da blir utgangen fra port P1 (Q) lik 0.

Analyse av RS-Latch (forts.)

- Analysèrer kretsen for S=0 og R=1

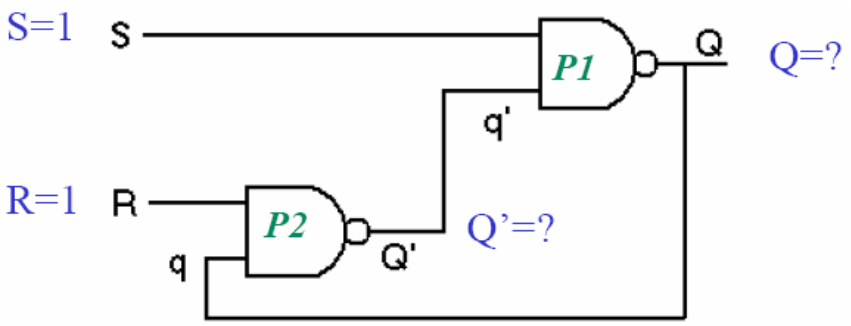


a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

- Kan ikke bestemme utgangen fra P2 (Q') uten å vite hva q er
- Utgangen fra port P1 (Q) er 1, fordi 0 på en av inngangene til en NAND alltid vil gi 1 ut
- Dermed er Q = q = 0, og da blir utgangen fra P2 (Q') lik 0.

Analyse av RS-Latch (forts.)

- Analysèrer kretsen for S=1 og R=1



a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

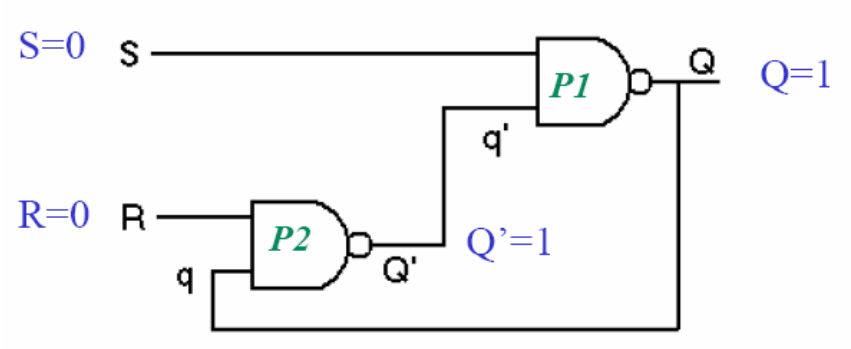
- Kan verken bestemme Q eller Q' fordi vi må kjenne den ene for å finne den andre.
- Siden Q (eller Q') både kan være 0 og 1, må vi vite hva forrige inputverdi (eller output) var:

S	R	Q	Q'	Kommentar
1	1	0	1	Etter S =1 og R = 0
1	1	1	0	Etter S =0 og R = 1

2008

Analyse av RS-Latch (forts.)

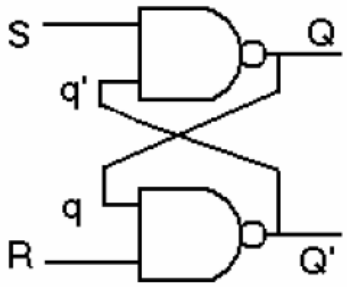
- Siste tilfelle: S=0 og R=0



a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

- Både Q og Q' er lik '1'. men er i konflikt med definisjonen av Q og Q'
- Ved bruk av RS-latchen i design bør denne kombinasjonen unngås (S = R = 0).

Oppsummering av RS-latch

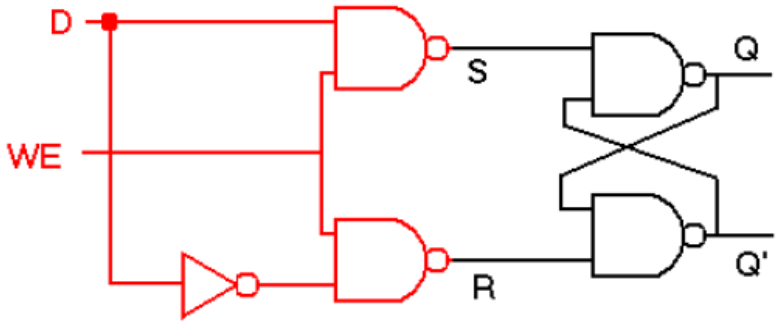


S	R	Q	Q'	Kommentar
1	0	0	1	
1	1	0	1	Etter S =1 og R = 0
0	1	1	0	
1	1	1	0	Etter S =0 og R = 1
0	0	1	1	

- RS-latch kan også konstrueres med NOR-porter (ukeoppgave)
- Ulemper ved RS-latch
  - Utgangen vil endre seg så fort inngangen endrer seg.
  - S = R = 0 gir en ikke-definert tilstand på utgangen; kan være vanskelig å kontrollere at ikke S og R er '0' samtidig

Utvidelse av RS-Latch: D-Latch

- Ved å utvide kretsen med to NAND-porter og en inverter
  - Forhindrer at Q endrer seg når WE (Write Enable) er 0, siden både S = R = 1, og dermed beholdes verdien til Q
  - Inverteren forhindrer at S=R=0 samtidig, dvs S+R=1



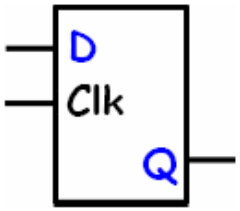
a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

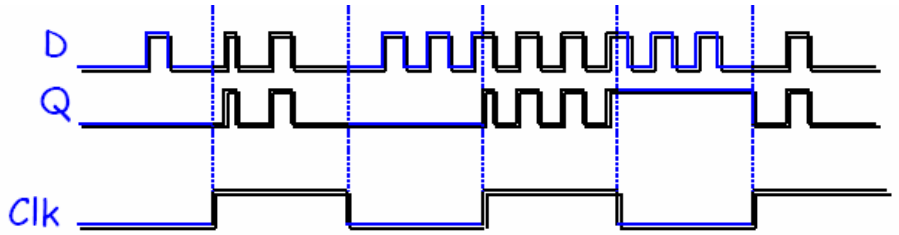
2008

Oppsummering D-Latch

W	D	S	R	Kommentar
0	0	1	1	Beholder lagret verdi
0	1	1	1	Beholder lagret verdi
1	0	1	0	Lagrer '0', dvs D
1	1	0	1	Lagrer '1', dvs D

- Når WE = 1 avleses D, og fører til at ENTEN S = 1 og R = 0 (D = 0) ELLER S = 0 og R= 1 (D = 1).
- Når WE går til ‘0’ igjen, vil verdien Q forbli uendret helt til WE blir ‘1’ og D eventuelt endrer verdi.



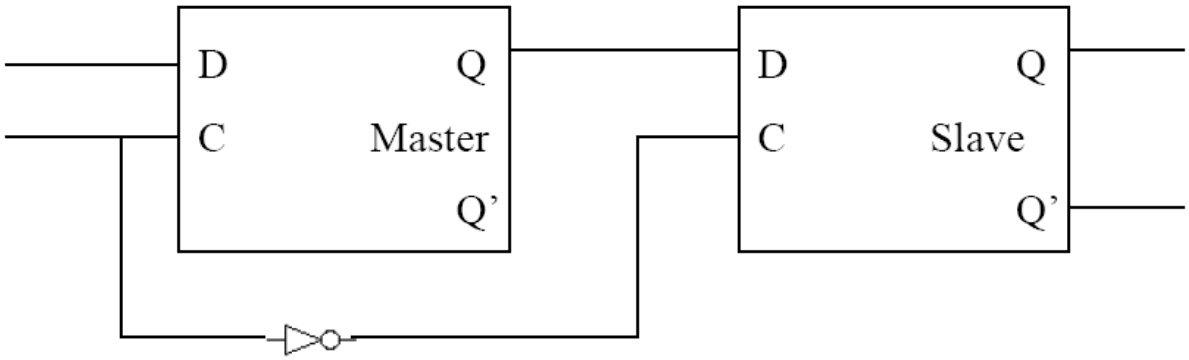


- D-latchen kalles **transparent** for WE=1 (i figuren over er WE=Clk)

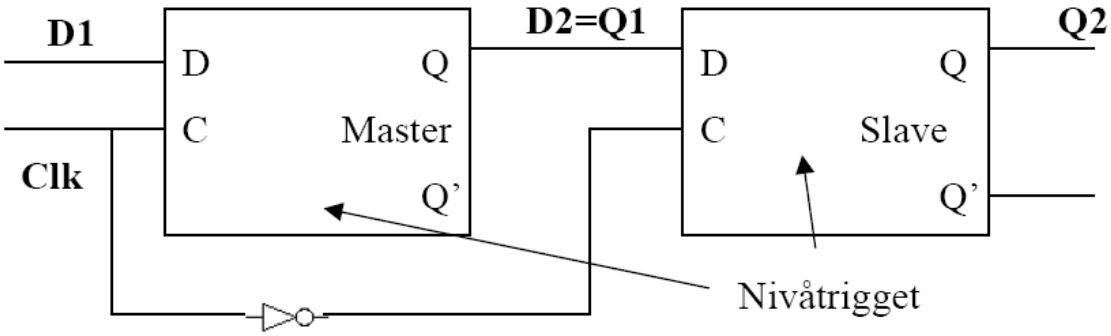
2008

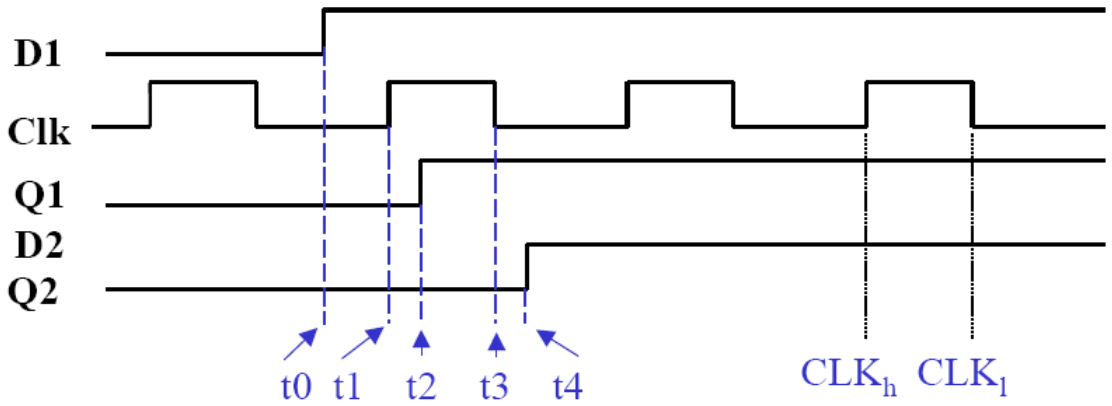
Utvidelse av D-Latch: kant-trigget D-flip-flop

- Ytterligere forbedring: Ønsker å kontrollere bedre når endringer skjer
- Går over fra nivå-trigging til kant-trigging, dvs når WE (CLK) går fra 0 til 1, eller fra 1 til 0
- Kobler to latcher i serie (den ene leser mens den andre er låst)



Timing i kant-trigget D-flip-flop





Timing i kant-trigget D-flip-flop (forts.)

- **t0**: D1 går fra lav til høy, men FF-D1 er låst
- **t1**: Clk endrer nivå fra lav til høy
- **t2**: Q1 og D2 endrer verdi fra lav til høy. FF-D2 er fortsatt låst
- **t3**: Klokkeinngangen på FF-D2 endrer nivå fra lav til høy
- **t4**: Q2 endrer verdi fra lav til høy
- (t2-t1) + (t4-t3) er den totale forsinkelsen gjennom flip-flop'en
- Lengden på høy klokkepuls må være større enn forsinkelsen gjennom flipflop'en, dvs. CLKh-CLKl > (t2-t1)
- t1-t0 > tx (tx kalles **setup time**)
- D1 kan ikke gå lav før etter en viss tid (**hold time**)
- Kravene setter begrensninger til maksimum tillatte klokkefrekvens
- Finnes også andre krav som må overholdes

Representasjon i binær 2-komplements form

- Brukes for å representere binære tall med fortegn. Det mest signifikate bit'et angir om tallet er positivt eller negativt

- **Positive tall**: Som før

- **Negative tall**: Representeres ved å invertere alle bit'ene i det tilsvarende positive tallet og legge til 1.

Eksempel: Konvertere 7 til –7 i binær 2-komplements form

1) $7_{10} = 0111_2$
2) Inverterer alle bit'ene hver for seg: 0111 → 1000
3) Legger til 1: 1000 + 1 = **1001**

Det vil si: 2-komplementet til 7 er 1001 og kan representere -7

Tar man 2-komplementet av dette tallet (1001):

4) Inverterer alle bitene: 1001 → 0110
5) Legger til 1: 0110 + 1 = **0111** $(=7_{10})$

Det vil si: Konverterer man til 2-komplementet to ganger kommer man tilbake til utgangspunktet!

2-komplements form (forts.)

- Ikke lenger mulig å lagre like store positive tall. Verdiområdet er endret fra $0 \rightarrow 2^n\text{-}1$ til $-2^{n-1} \rightarrow 2^{n-1} - 1$ for et n-bits binært tall.

Eksempel:

- 4 bit kan representere tall fra 0 til 15 uten fortegn, eller tall fra –8 til 7 på 2-komplements form (dvs. likevel 16 forskjellige verdier)

- Fordelen ved 2-komplements form er at addisjon og subtraksjon blir samme operasjon, dvs. addisjon!
  - Svaret (summen) blir i 2-komplements form

Binær addisjon

- Tilsvarer aritmetisk addisjon i titalls-systemet

Eksempel:

```
  0 0 0 1
+ 0 0 1 1
  0 1 0 0
```

- Notasjonen $a = a_{n-1}a_{n-2}..a_1a_0$ brukes for å angi enkelt-bit'ene i et tall a som er *n* bit langt.

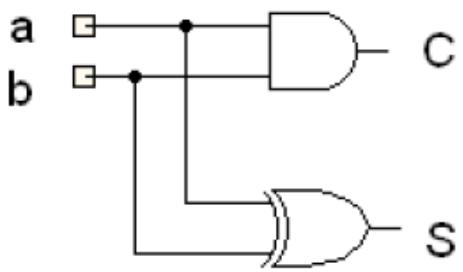
- Sannhetsverditabell for halvadder: (dvs. $a + b$)

a	*b*	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

S: Sum, C: Carry (mente)

$S = a'b + ab'$
$= a \oplus b$ (a XOR b)

$C = ab$



2008

Binær addisjon (forts.)

- Kretsen kalles halvadder fordi den ikke tar hensyn til eventuelle *mente*bit fra bitaddisjonen til høyre. Tar man hensyn til mentebit fra bitposisjon til høyre kalles kretsen en fulladder:

a_i	b_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S_i = a_i'b_i'C_i + a_i'b_iC_i' + a_ib_i'C_i' + a_ib_iC_i$$

$$C_{i+1} = a_i'b_iC_i + a_ib_i'C_i + a_ib_iC_i' + a_ib_iC_i$$

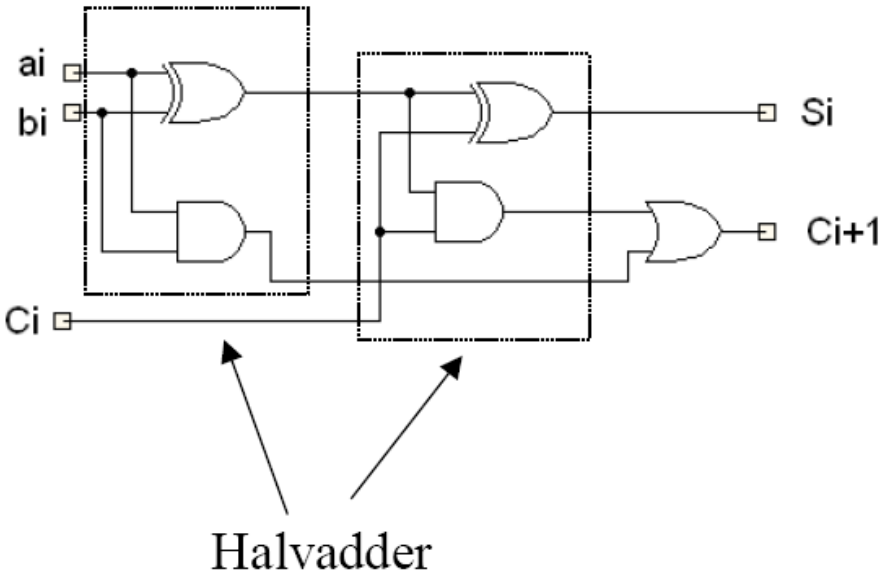
- C_i er mentebit'et som ble generert i posisjonen til høyre for posisjon i, mens C_{i+1} er mentebit'et som ble generert i posisjon i.

Binær addisjon (forts.)

- Skriver om uttrykkene for S_i og C_{i+1}:

$$S_i = a_i'b_i'C_i + a_i'b_iC_i' + a_ib_i'C_i' + a_ib_iC_i$$
$$= a_i' \cdot(b_i'C_i + b_iC_i') + a_i\cdot(b_i'C_i' + b_iC_i)$$
$$= a_i' \cdot(b_i'C_i + b_iC_i') + a_i\cdot(b_i'C_i + b_iC_i')'$$
$$= a_i \oplus(b_i'C_i + b_iC_i') = a_i \oplus b_i \oplus C_i$$

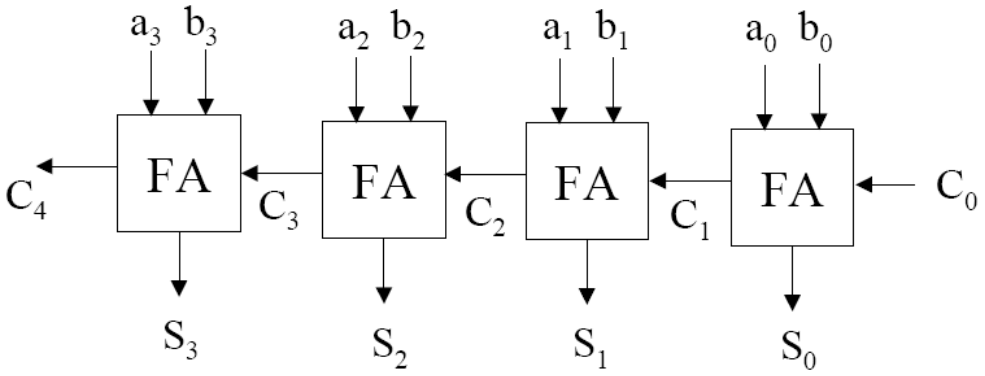
$$C_{i+1} = a_i'b_iC_i + a_ib_i'C_i + a_ib_iC_i' + a_ib_iC_i$$
$$= C_i \cdot(a_ib_i + a_ib_i') + a_ib_i \cdot(C_i' + C_i)$$
$$= \text{Ci} \cdot(a_i \oplus b_i) + a_ib_i$$



2008

Binær addisjon (forts.)

- For å addere tall med flere bit setter man sammen fulladdere:



- Svakhet: Mentepropagering (mest signifikante bitposisjon kan ikke beregnes før alle de mindre signifikante er beregnet)

Datapath og ALU

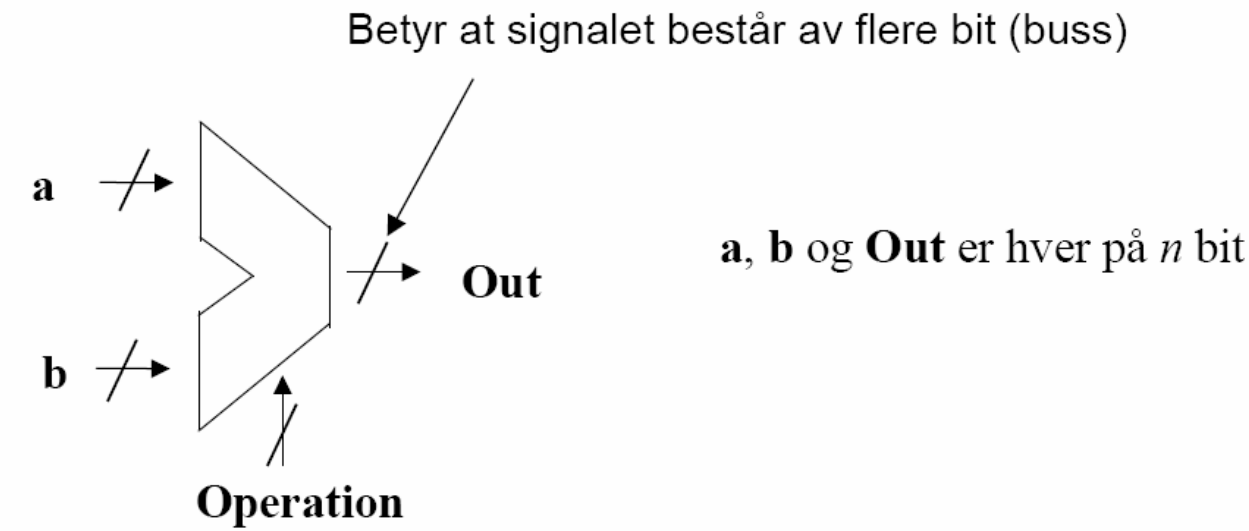
- **Datapath**: Den delen av en CPU hvor beregninger foretas (både aritmetiske og logiske, inkludert adresseberegninger)

- **Aritmetisk-Logiske Enhet** (**ALU**) utfører logiske og aritmetiske operasjoner på to input-variable i data path.

- En CPU har flere ALU'er slik at beregninger kan fortas i parallell, f.eks beregning av adresser (neste instruksjon) og utføring av selve instruksjoner

- Mye energi legges ned i å optimalisere designet av en ALU for å få maksimal ytelse

Datapath og ALU (forts.)

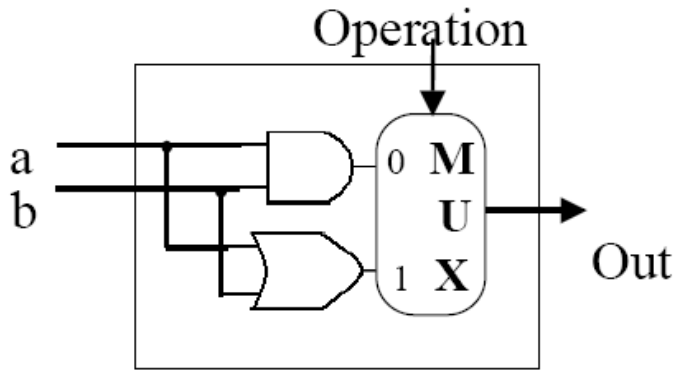


a, **b** og **Out** er hver på *n* bit

Antall bit i **Operation** er avhengig av antall ulike operasjoner som ALUen kan utføre

1-bits ALU

- 1-bits ALU som kan utføre AND eller OR
  - Kan designes med en MUX, AND og OR porter.

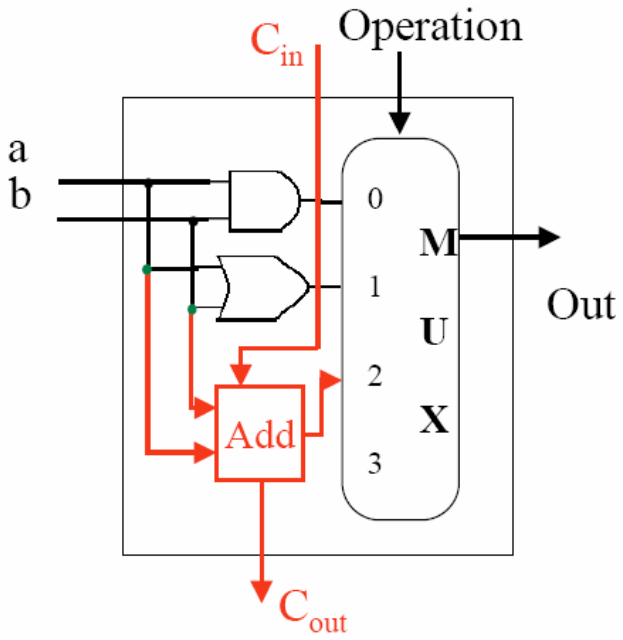


Operation	Out
0	a AND b
1	a OR b

- Funksjonen ALU'en utfører er mao. styrbar (kontrollert av Operation-signalet)

1-bits ALU (forts.)

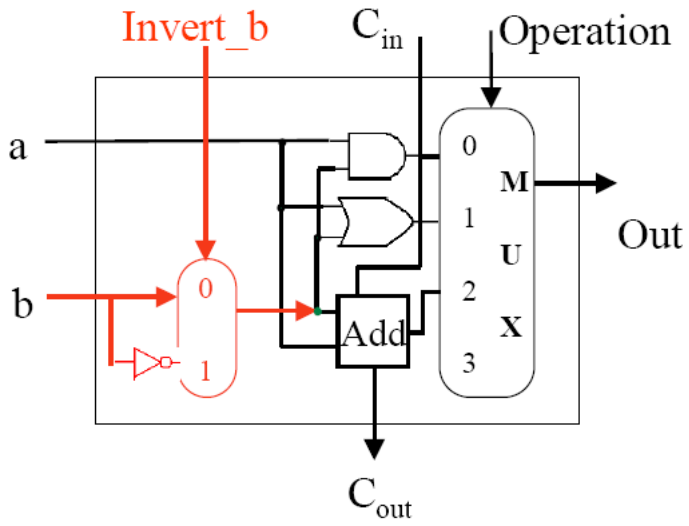
- 1-bits ALU som kan utføre AND, OR eller addisjon



Operation	Out
00	a AND b
01	a OR b
10	a + b + C_{in}
11	Ingen funksjon

1-bits ALU (forts.)

- 1-bits ALU som kan utføre AND, OR, addisjon eller subtraksjon



Invert_b	Operation	Out
0	00	a AND b
0	01	a OR b
0	10	$a + b + C_{in}$
0	11	Ingen funksjon
1	00	a AND b'
1	01	a OR b'
1	10	$a + b' + C_{in}$
1	11	Ingen funksjon

2008

n-bits ALU

- 1-bits ALU som kan utføre AND, OR, addisjon eller subtraksjon

•Setter sammen 1-bits ALU'er

- Legg merke sammenkoblingen av C_{in} og Invert_b på ALU0
  - for å gi riktig mente inn
  - å legge til '1' ved subtraksjon

  (forutsetter 2-komplements form)