UNIVERSITETET
I OSLO

Dagens temaer

- Dagens temaer hentes fra kapittel 3 i “Computer Organisation and Architecture”
- Kort repetisjon 2-komplements form
- Binær addisjon/subtraksjon
- Aritmetisk-logisk enhet (ALU)
- Sekvensiell logikk
- RS-latch

UNIVERSITETET
I OSLO

2-komplements representasjon

- Brukes for å representere binære tall med fortegn. Det mest signifikate bit'et angir om taller 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: 0111 → 1000

  3) Legger til 1: 1000+1 = 1001

UNIVERSITETET
I OSLO

2-komplements representasjon (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

- Fordelen ved 2-komplements forrm er at addisjon og subtraksjon blir samme operasjon, dvs addisjon!

Binær addisjon

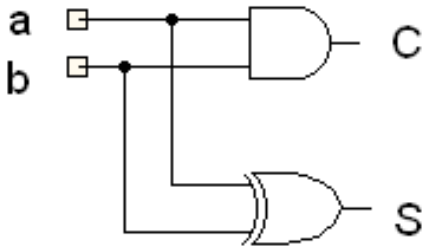
- Tilsvarer aritmetisk addisjon i titalls-systemet
- Eksempel

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

- 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:

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

S = a'b + ab'
C = ab



Binær addisjon (forts.)

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

a_i	b_i	C_i	S_i	C_{i+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

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

$$C_{i+1} = a_ib_i + a_iC_i + b_iC_i$$

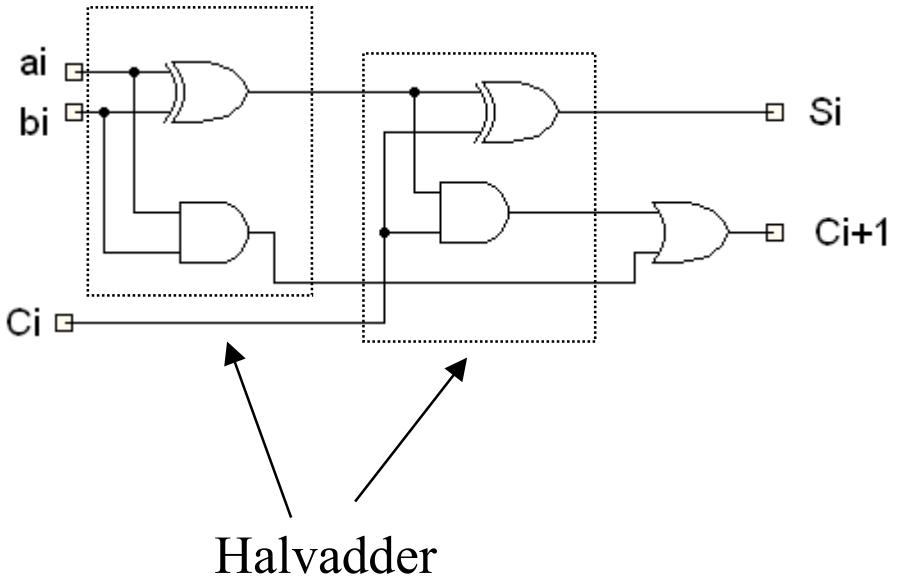
- C_i er mentebit'et som ble generert i posisjonen til *høyre* for posisjon *i*, mens C_{i+1} er mentebit'et som ble generert i posisjon *i*.

Binær addisjon (forts.)

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

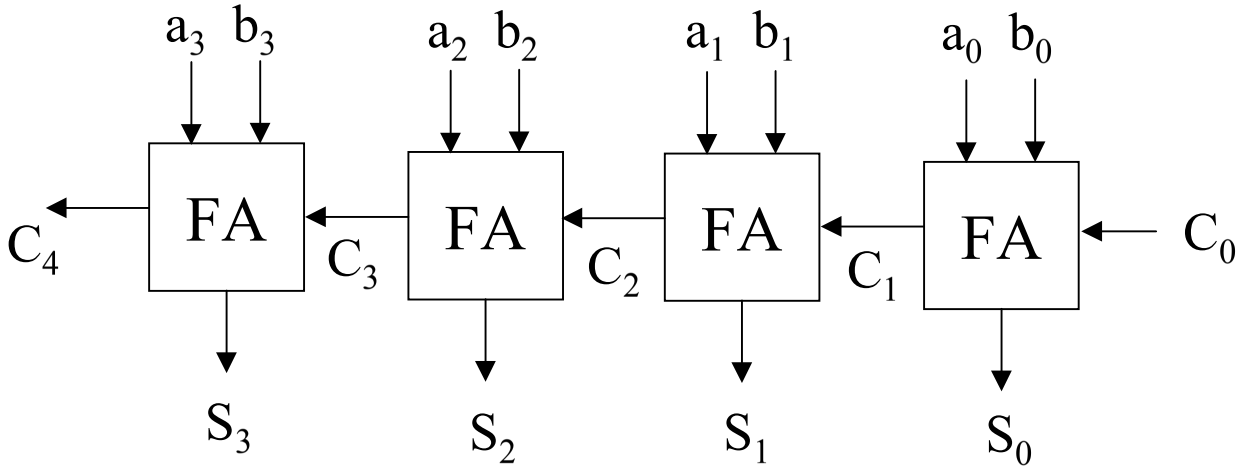
$$
\begin{aligned}
S_i &= a_i'b_i'C_i + a_i'b_iC_i' + a_ib_i'C_i' + a_ib_iC_i \\
&= a_i'(b_i'C_i + b_iC_i') + a_i(b_i'C_i' + b_iC_i) \\
&= a_i'(b_i'C_i + b_iC_i') + a_i(b_i'C_i + b_iC_i')' \\
&= a_i \otimes (b_i'C_i + b_iC_i') \\
&= a_i \otimes b_i \otimes C_i
\end{aligned}
$$

$$
\begin{aligned}
C_{i+1} &= a_ib_i + a_iC_i + b_iC_i \\
&= C_i(a_i + b_i) + a_ib_i \\
&= C_i(a_i \otimes b_i) + a_ib_i
\end{aligned}
$$



Binær addisjon (forts.)

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



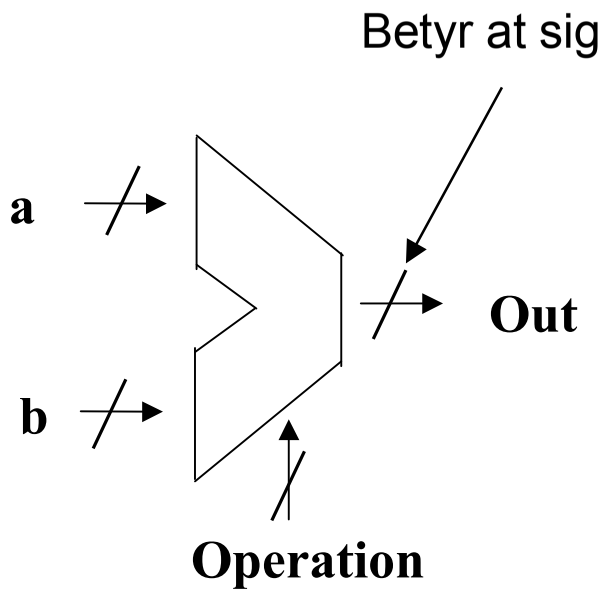
- Svakhet: Mentepropagering (siste bitposisjon kan ikke beregnes for de foregående er beregnet)

UNIVERSITETET
I OSLO

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.)

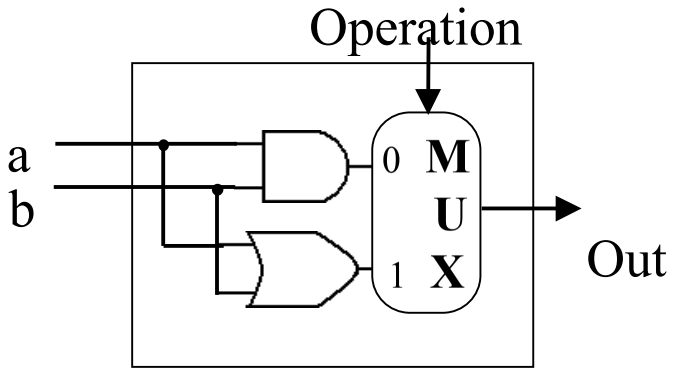


a, **b** og **Out** er hver på *n* bit

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.

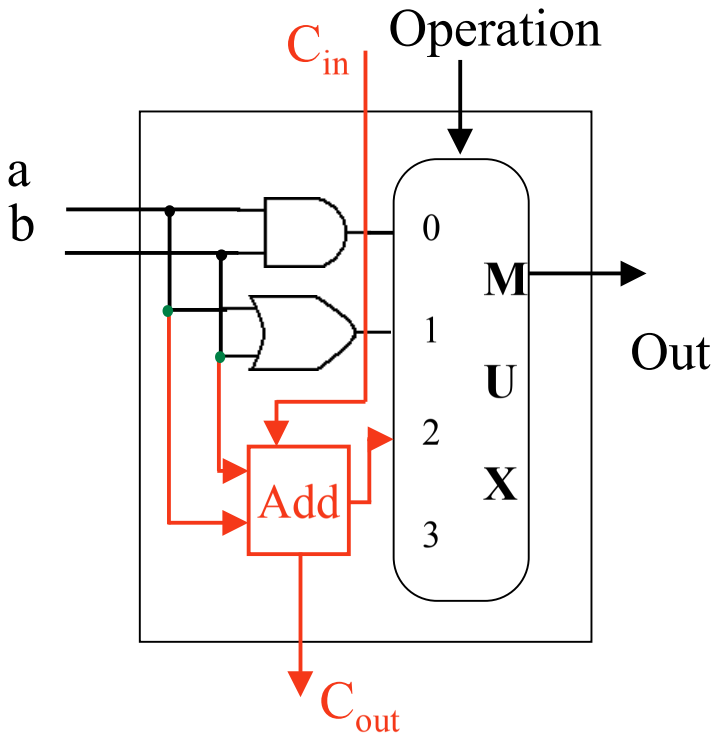


Operation	Out
0	a AND b
1	a OR b

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

1-bits ALU (forts.)

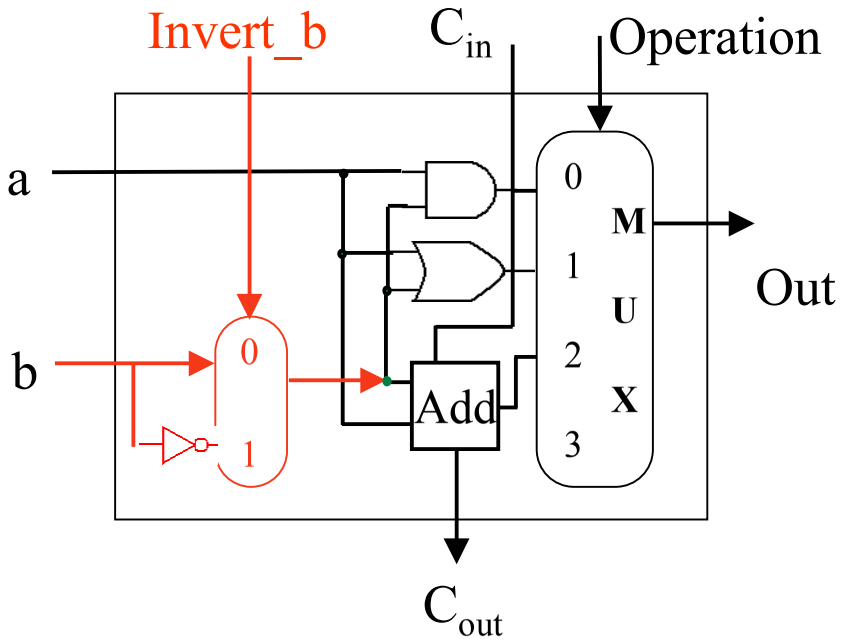
- 1-bits ALU som kan utføre AND, OR eller addisjon



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

1-bits ALU (forts.)

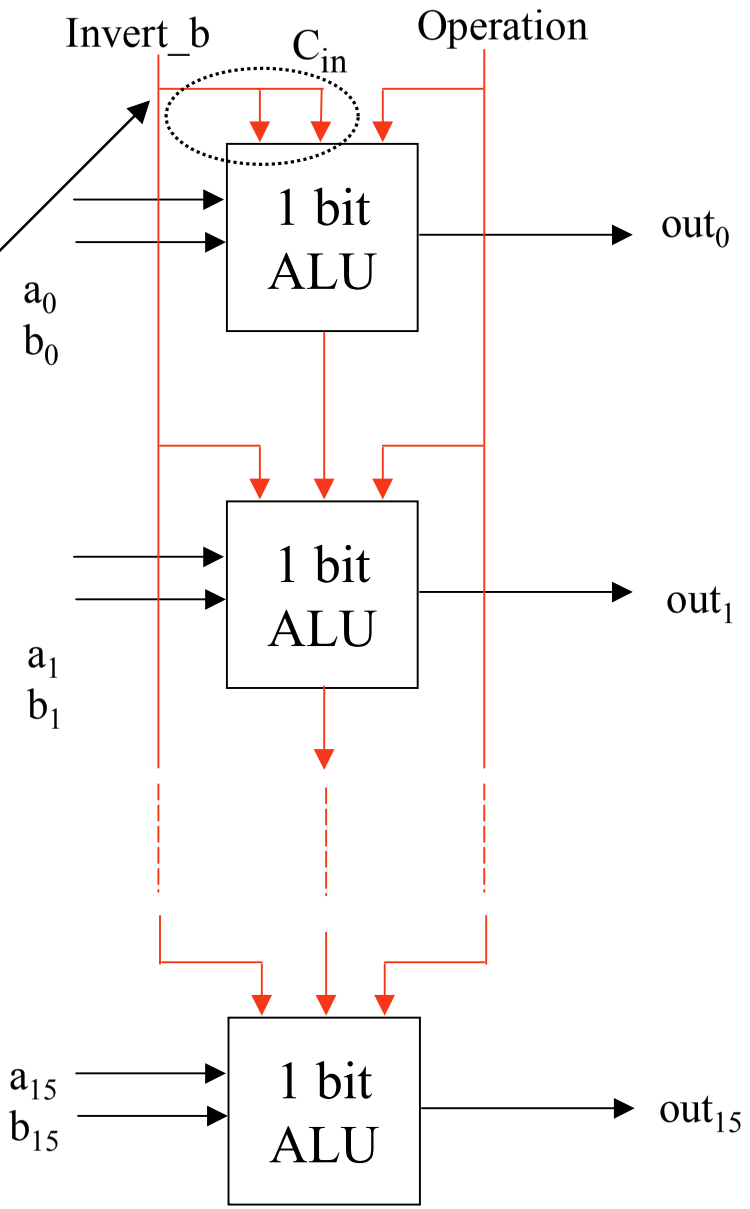
- 1-bits ALU som kan utføre AND, OR, addisjon eller subtraksjon



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

n-bits ALU

- Setter sammen 1-bits ALU'er
- Legg merke sammenkoblingen av C_{in} og Invert_b på ALU_0 for
  - å gi riktig mente inn
  - å legge til '1' ved subtraksjon (forutsetter 2-komplements form)



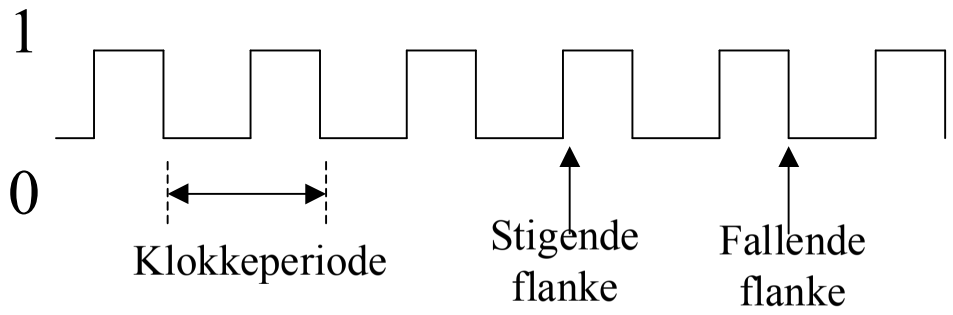
UNIVERSITETET
I OSLO

Sekvensiell logikk

- Hvis output fra en krets kun er avhenghig av nåværende input, kalles den kombinatorisk
- Hvis output fra en krets er avhengig av nåværende *og* tidligere input, kalles kretsen sekvesiell. Den må da inneholde minne eller hukommelse
- Alle datamaskiner inneholder en blanding av kombinatorisk logikk og hukommelse.
- Hukommelse finnes i mange varianter avhengig av hva de skal brukes til:
  - I RAM brukes spesialisert teknologi basert på lagring av elektriske ladinger (kondensatorer)
  - Inne i CPU'en brukes hukommelse basert på logiske porter.
  - En mengde andre typer som DVD, CD, FPGA, Flash, (E)PROM etc. (Kommer tilbake til disse senere i kurset)

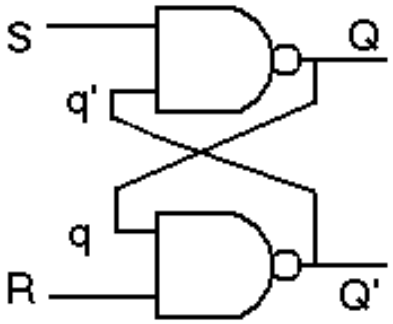
Sekvensiell logikk

- I *synkrone* sekvensielle kretser skjer endringen(e) i output samtidig med *endringen* i et *klokkesignal,* eller når klokkesignalet enten er '0' eller '1'
- I *asynkrone* sekvensielle kretser skjer endringene i output med en gang uten noe klokkesignal.
- Asynkrone kretser er raskere, men benyttes sjelden fordi de er mye vanskligere å designe og teste
- Klokkefrekvensen f = 1/klokkeperioden
  - Høyere frekvens betyr kortere tid mellom hver gang en endring kan skje
  - Pentium 4 kjører på over 3 GHz
  - Eksperimentelle transistorer kan skifte mellom '0' og '1' med en frekvens på 507 GHz!



RS-latch

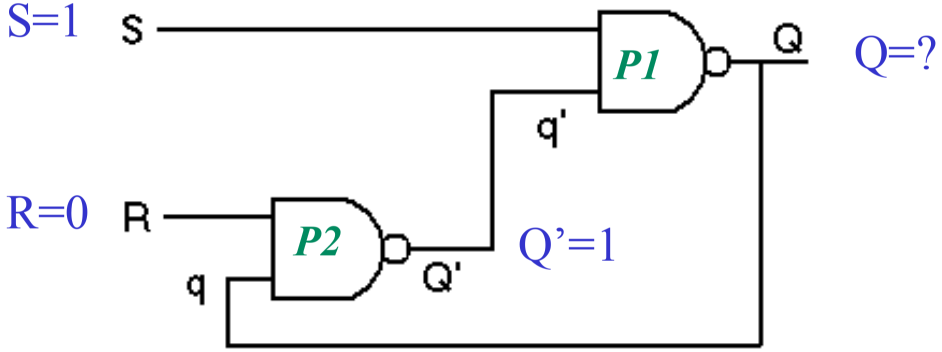
- Hukommelseselement med to innsignaler: S(et) og R(eset), og to utganger Q og Q'



- 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.

RS-latch (forts.)

- Analyserer kretsen for S = 1 og R = 0

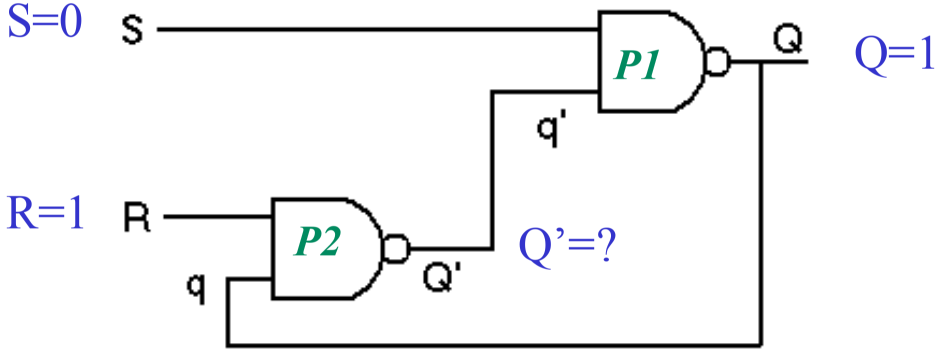


a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

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

RS-latch (forts.)

- Analyserer kretsen for S = 0 og R = 1

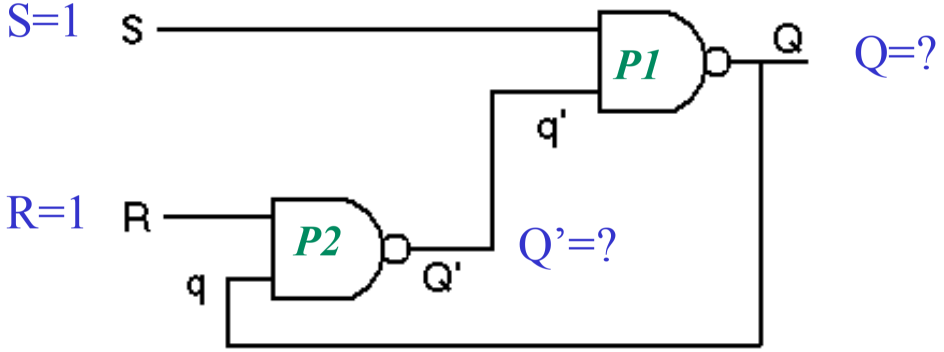


a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

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

RS-latch (forts.)

- Analyserer kretsen for S = 1 og R = 1



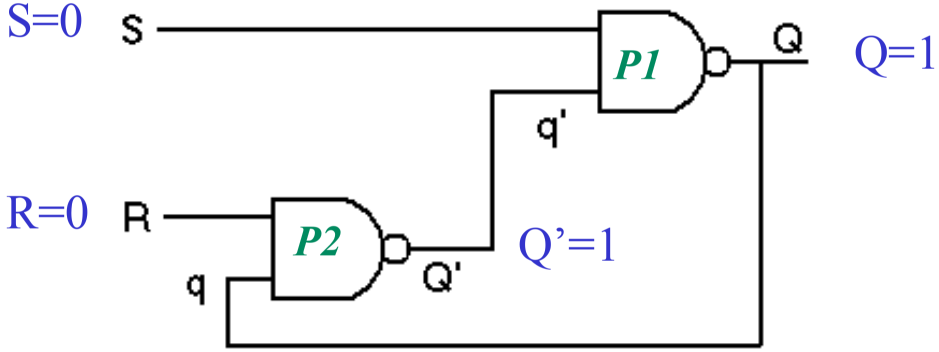
a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

- 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 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

RS-latch (forts.)

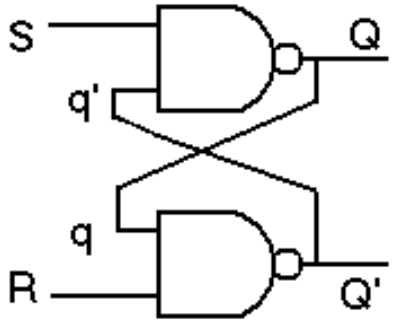
- Ssite tilfellet: S = 0 og R = 0



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'
- I design unngås denne kombinasjonen (S = R = 0).

Oppsummering RS-latch

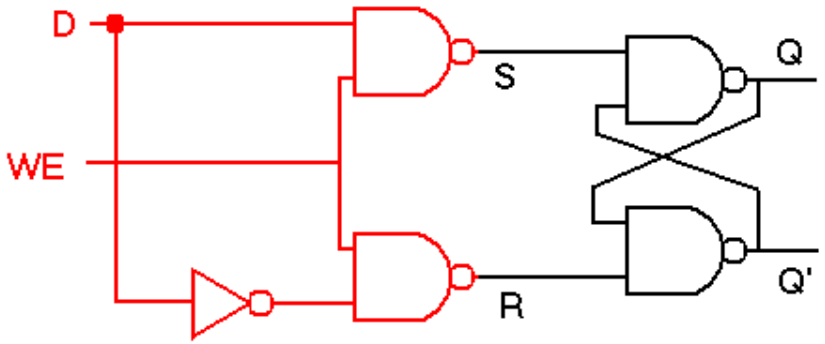


S	R	Q	Q'	Kommentar
1	0	0	1	
1	1	0	1	Etter S =1 og R = 0
0	1	1	0	
1	1	1	0	Etter S =0 og R = 1
0	0	1	1	

- RS-latch kan også konstrueres med NOR-porter (se 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

Forbedring av RS-latch

- Ved å utvide kretsen med 2 NAND-porter og en inverter:
  - Forhindrer man at utgangen 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



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

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