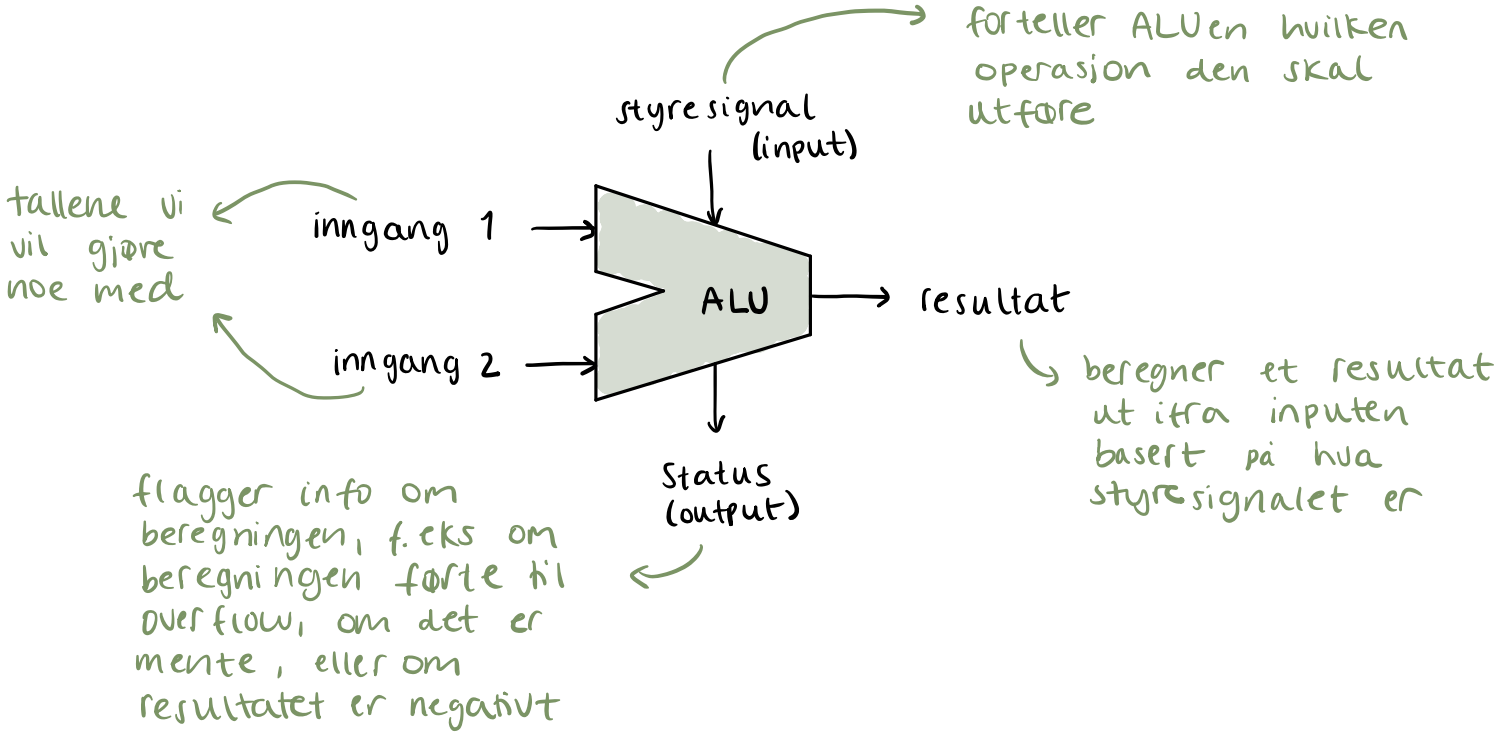
Oppsummering ALU, binær addisjon, abstraksjon og porter

ALU - aritmetisk logisk unit



→ ALUen i en vanlig datamaskin kan utføre mange type beregninger: addisjon, subtraksjon, multiplikasjon, og logiske operasjoner som OR, AND eller NOT.

Hva er ALUen egentlig laget av? Porter

→ Portene er basic "byggeblokker" for å lage kretser og kretser hjelper maskinen å beregne ting, for eksempel utføre addisjon mellom 2 tall.

→ Vi har følgende byggeblokker:

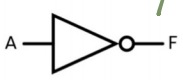
funksjon

port

INV - port

$F = A'$

A	F
0	1
1	0



→ sann kun når input er 0

AND - port

$F = AB$

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

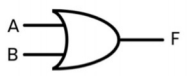


→ sann når alle inputene er 1

OR - port

$F = A + B$

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



→ minst 1 input må være sann for at output skal være sann

sannhetsverditabell

NOR - port

$F = (A + B)'$

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



→ motsatt av OR: Ingen input kan være sann for at output skal være sann

NAND - port

$F = (AB)'$

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

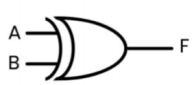


→ motsatt av AND: bare sann når inputene ikke er begge 1

XOR - port

$F = A \oplus B$

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



→ sann når inputene er ulike

XNOR - port

$F = (A \oplus B)'$

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1



→ motsatt av XOR: sann når begge inputene er like

→ setter vi noen av portene sammen kan vi danne kretser som beregner ting for oss, for eksempel en halvadder eller fulladder, som adderer tall.

Binær addisjon:

Vi skal fokusere på addisjon. Siden datamaskinens språk består av 0ere og 1ere, så er det **binær addisjon** det er snakk om her.

Hvordan utfører vi binær addisjon? På samme måte som i 10tallssystemet, men istedenfor 10 siffer, så har du 2.

MEN i binær addisjon så må du også passe på hvor stort resultatet ditt er. Om du adderer 2 tall på 1 bit hver så må resultatet også passe inn i 1 bit. Hvis ikke får vi **overflow**. Det skjer altså når resultatet vårt ikke passer inn i antall bit vi har tilgjengelig for resultatet.

De 4 måtene vi kan addere 2 binære siffer er:

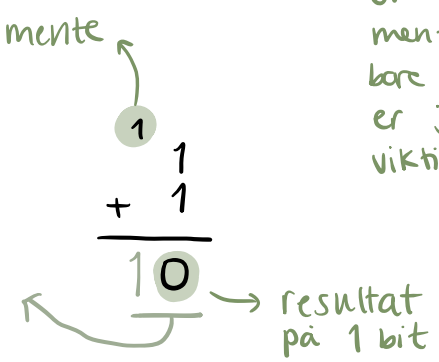
$0 + 0 = 0$ $\quad 1 + 0 = 1$

$0 + 1 = 1$ $\quad 1 + 1 = 0$ og 1 i mente

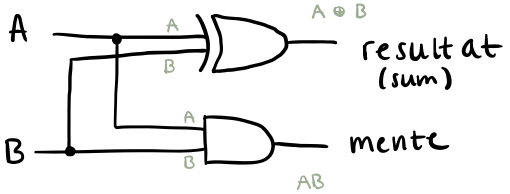
→ Mystisk?
- → i 10tallssystemet er $1+1=2$ men vi har ikke 2 binært
- → $2_{10} = 10_2$ (0 som resultat og 1 i mente), men 10_2 er 2 bits, mens resultatet vårt skal passe inn i 1 bit.

→ Om vi ikke tar med mente vil resultatet bare være 0, men det er jo helt feil ⇒ viktig å ta med mente

resultat på 2 bit, men vi kan IKKE skrive det slik fordi resultatet skal bare være på 1 bit, så vi må lagre det ekstra bitet i mente



Hvordan kan vi lage en krets som regner dette for oss?



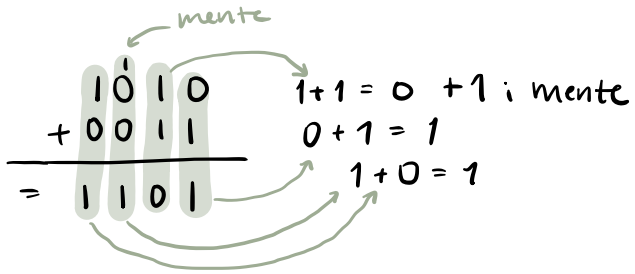
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

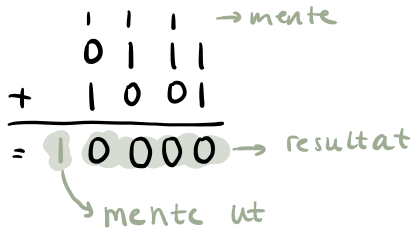
Når A og B er begge 1, så vil A + B = 0 og 1 i mente. Vi ser kretsen regna ut rett siden S = 0 og C = 1.

Denne halvadderen adderer bare 2 tall på 1 bit. Det er enkelt nok, men hva om vi vil addere tall på flere bits?

Først, la oss se på noen eksempler:

1)

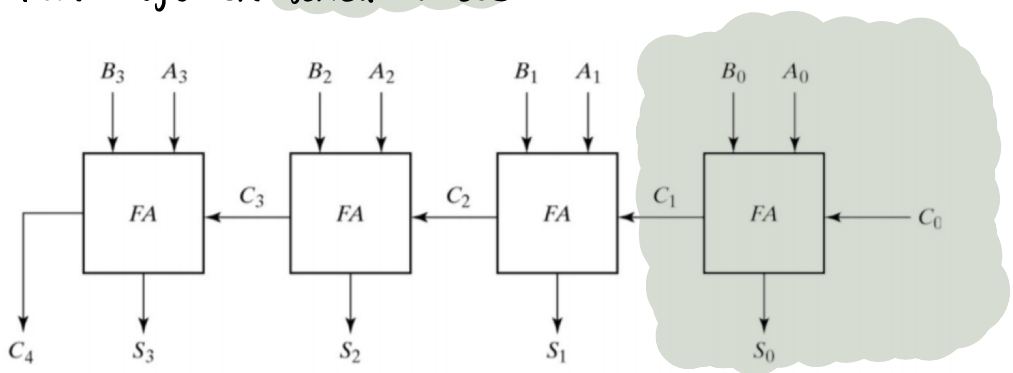




Hvordan kan vi lage en krets som gjør dette for oss?

→ vi kan ikke lenger bruke en halvadder, fordi vi må håndtere flere bits.

→ vi kan lage en **seriell adder**!



Legg merke til at vi nå har behov for å bruke både mente inn og ut, noe halvadderen ikke kan håndtere. Derfor bruker vi en fulladder for hver blokk.

Hver blokk er en adder som har mente inn, mente ut, har to inputs (2 1bits tall) og en output (1 bit)

→ så en fulladder har mente inn i tillegg til mente ut, til forskjell fra en halvadder som bare har mente ut.

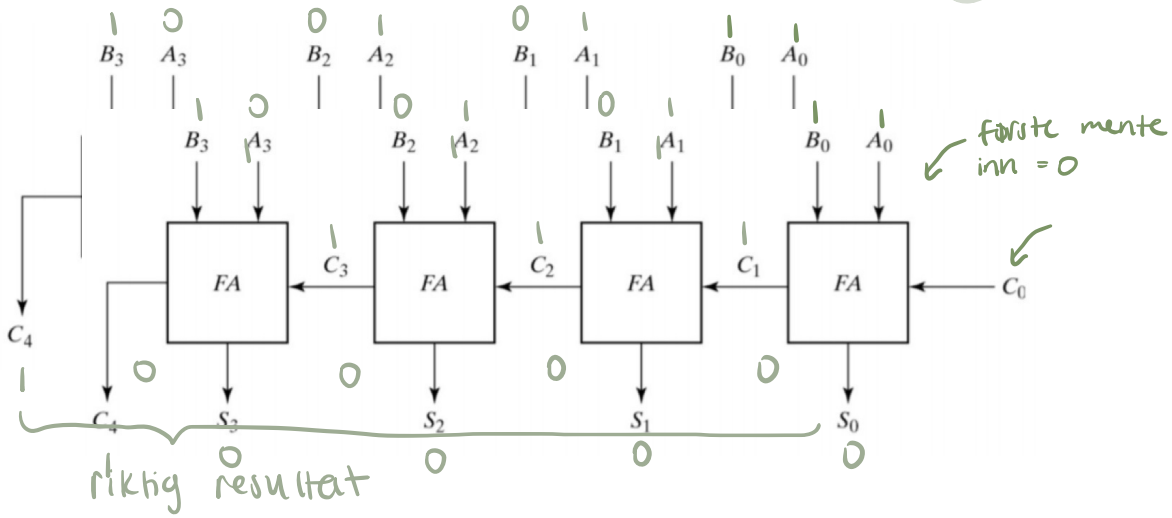
→ seriell adder = flere fulladdere på rekke

For eksempel, la oss se hvordan den 4-bits serielle adderen ville addet A og B:

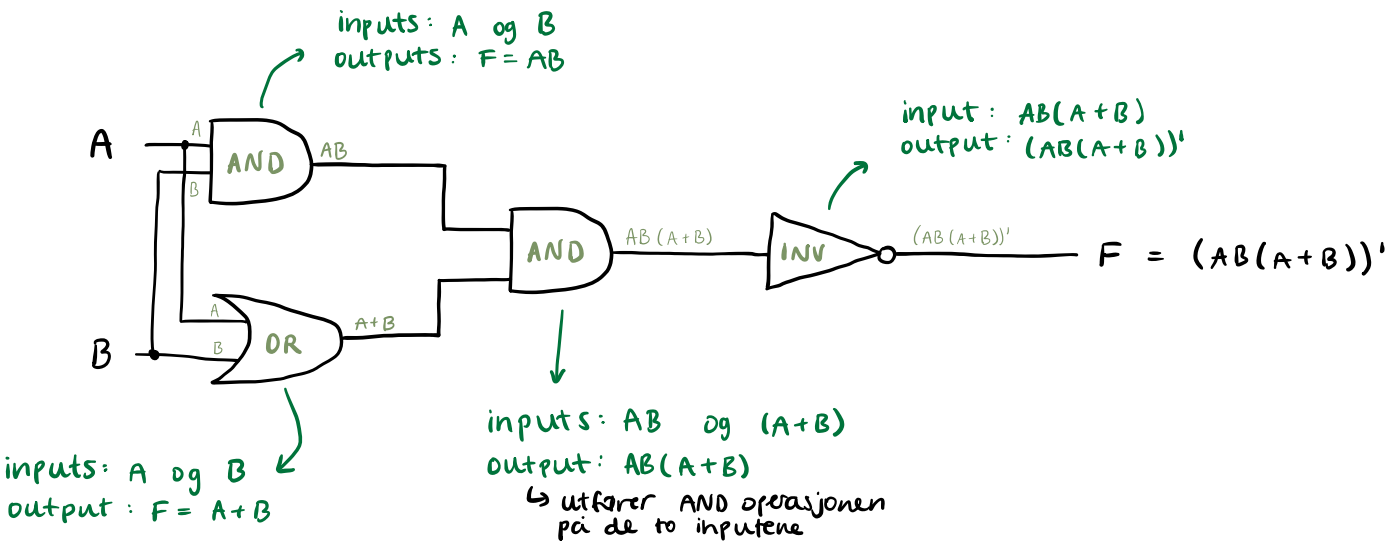
A = 0111 og B = 1001

A_3 A_0 B_3 B_0

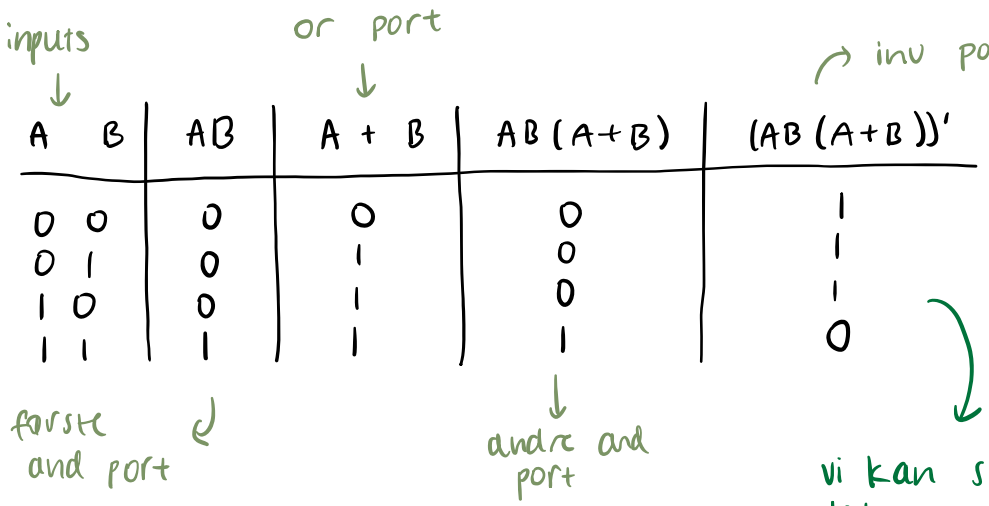
```
  0111
+ 1001
------
=10000
 ↑mente
```



Kretsanalyse:



La oss analysere kretsen med en sannhetsverditabell:



→ funksjonsuttrykk

Vi kan se at outputen blir det samme som en NAND port, som vil si at vi kan forkorte: hele kretsen kan kortes ned til en NAND port siden begge gjør det samme.

Vi kan også forenkle funksjonsuttrykket ved å bruke boolsk algebra:

$(AB(A+B))'$

$= (ABA + ABB)'$ (ganger ut parantesen)

$= (ABA + AB)'$ (bruker regel som sier at B and B = B)

$= (AB(A+1))'$ (faktoriserer ut AB siden det er felles faktor)

$= (AB \cdot 1)'$ (bruker regel som sier at A or 1 = 1)

$= (AB)'$ (bruker regel som sier at A and 1 = A)

↳ endte opp med funksjonsuttrykket for NAND!

↳ vi kan forkorte ned til en NAND port og $(AB(A+B))' = (AB)'$

Sannhetsverditabell for hver port:

Inputs		Truth Table Outputs For Each Gate					
A	B	AND	NAND	OR	NOR	EX-OR	EX-NOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

Regler for boolsk algebra:

1. Boolean Algebra simplification rules

1. $A+\overline{A}=1$	2. $A+A=A$
3. $A\cdot A=A$	4. $A\cdot\overline{A}=0$
5. $A\cdot(B+C)=A\cdot B+A\cdot C$	6. $A+0=A$
7. $A+1=1$	8. $A\cdot 1=A$
9. $A\cdot 0=0$	10. $A\cdot B=B\cdot A$
11. $A+B=B+A$	12. $B\cdot(A+\overline{A})=B$
13. $A+A\cdot B=A$	14. $A\cdot(A+B)=A$
15. $A+\overline{A}\cdot B=A+B$	16. $A\cdot(\overline{A}+B)=A\cdot B$
17. $\overline{A+B}=\overline{A}\cdot\overline{B}$	18. $\overline{A\cdot B}=\overline{A}+\overline{B}$