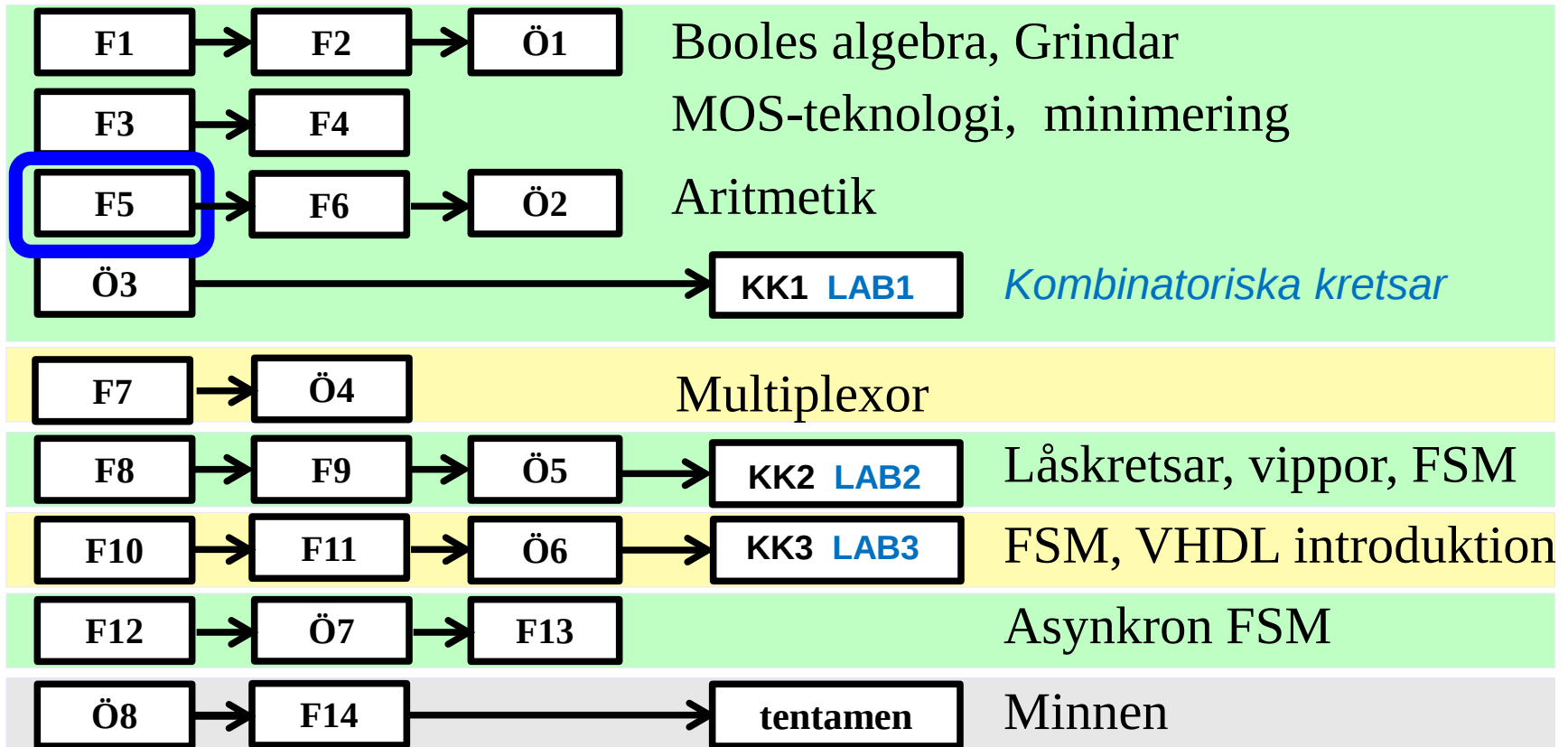


Digital Design IE1204

F5 Digital aritmetik I

`william@kth.se`

IE1204 Digital Design



*Föreläsningar och övningar bygger på varandra! Ta alltid igen det Du missat!
Läs på i förväg – delta i undervisningen – arbeta igenom materialet efteråt!*

Detta har hänt i kursen ...

Talsystem: Decimala, hexadecimala, oktala, binära

$$(175,5)_{10} = (AE.8)_{16} = (256.4)_8 = (10101110.1)_2$$

AND OR NOT EXOR EXNOR Sanningstabell, mintermer Maxtermer PS-form
SP-form deMorgans lag Bubbelgrindar Fullständig logik NAND NOR

CMOS grindar, standardkretsar

Minimering med Karnaughdiagrammet 2, 3, 4, 5, 6
variabler

Talrepresentation

**Ett tal kan representeras binärt på många sätt.
De vanligaste taltyperna som skall representeras
är:**

- **Heltal, positiva heltal (eng. integers)**
ett-komplementet, två-komplementet, sign-magnitude
- **Decimala tal med fix tal-område**
Fix-tal (eng. fixed-point)
- **Decimala tal i olika talområden**
Flyt-tal (eng. floating-point)

Heltal

Positiva Heltal:

$$\begin{array}{ccccccc} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \boxed{1} & \boxed{1} & \boxed{0} & \boxed{1} & \boxed{1} & \boxed{0} & \boxed{1} \end{array} = 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 = 109$$

Men hur representerar vi negativa tal ???

Sign-magnitude

Heltal:

S	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	1	0	1

$$= - (1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0) = -45$$

Magnituden (beloppet) av talet

Tecken-bit

Nackdel: två nollor (+/-) 0

1	0	0	0	0	0	0
0	0	0	0	0	0	0

1-komplement

De negativa talen är komplementet av de positiva talen. Bit för i det positiva talet bit inverteras för att ge den negativa motsvarigheten.

$$B = b_{N-1} b_{N-2} \dots b_1 b_0 \quad \text{där } b_i \in \{0, 1\}$$



Tecken-Bit
(Sign Bit)

Nackdelar:

- Två nollor (+/-) 0.
- Vid vissa additioner krävs justering av resultatet.

2-komplement heltal

Representation med 2-komplement

$$B = b_{N-1} b_{N-2} \dots b_1 b_0 \quad \text{där } b_i \in \{0, 1\}$$



Tecken-Bit
(Sign Bit)

Decimalvärde:

$$D(B) = -b_{N-1} 2^{N-1} + b_{N-2} 2^{N-2} + \dots + b_1 2^1 + b_0 2^0$$

Detta är den vanligaste representationen av tal ”med tecken”.

2-komplement heltal

Omvandlingsexempel:

$$B = b_{N-1} b_{N-2} \dots b_1 b_0 \quad \text{där } b_i \in \{0, 1\}$$



↑
Tecken-Bit
(Sign Bit)

Decimalvärde:

$$D(B) = -b_{N-1} 2^{N-1} + b_{N-2} 2^{N-2} + \dots + b_1 2^1 + b_0 2^0$$

$$= -128 + 127 = -1$$

Det är alltid det största talet som motsvarar -1.

Talkonvertering

positivt tal till negativt

Tvåkomplementmetoden

01111

+15

10000

invertera

10001

lägg till ett

10001

-15

Talkonvertering

negativt tal till positivt

Tvåkomplementmetoden

10001

-15

01110

invertera

01111

lägg till ett

01111

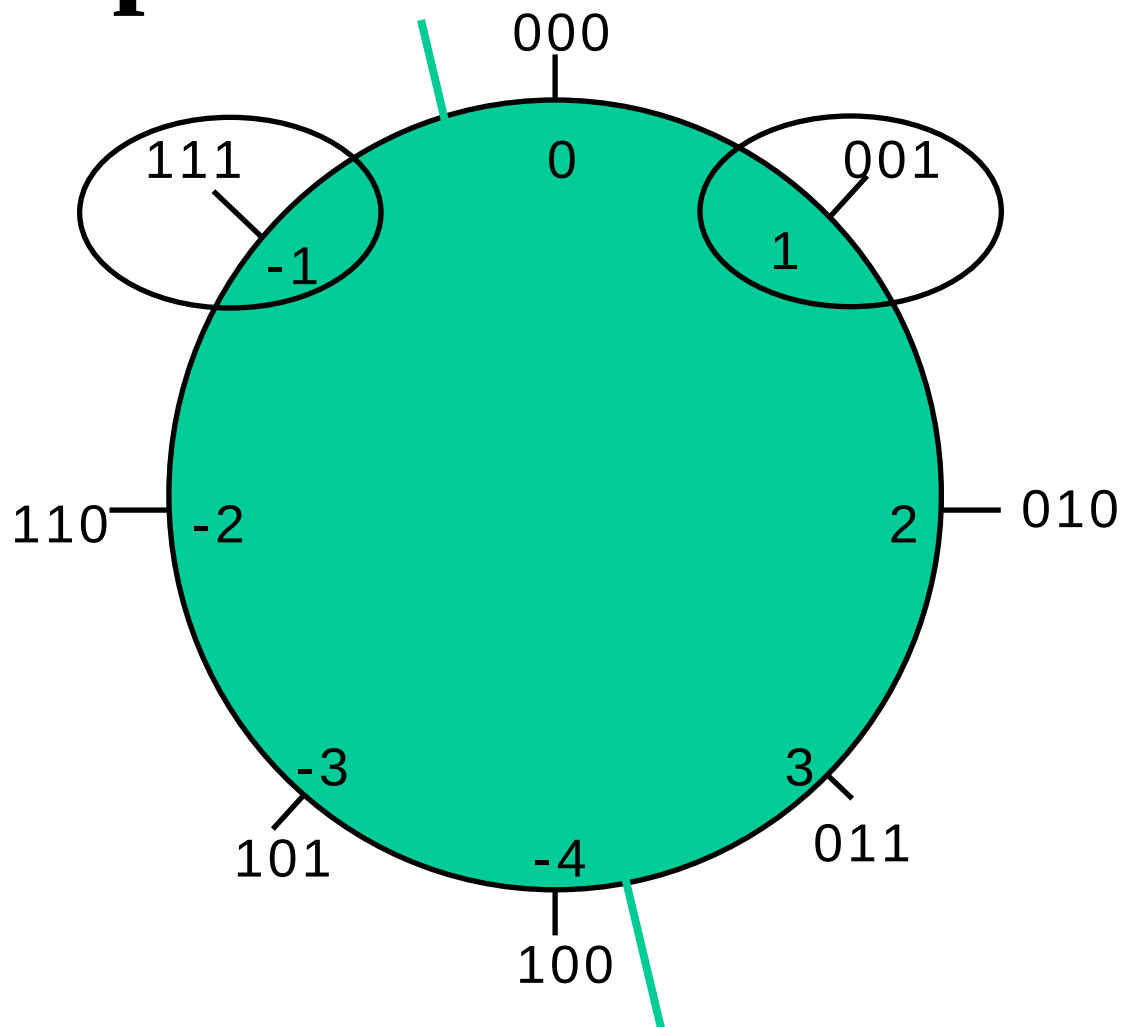
+15

Samma procedur i bägge riktningarna!

2-komplement heltal



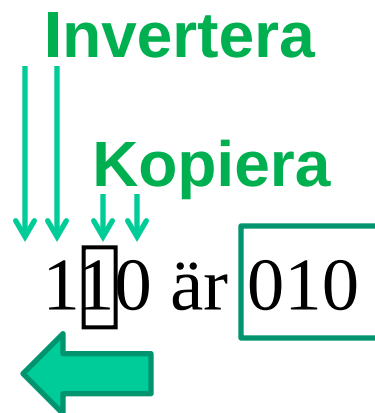
Dator-register är ”ringar”. Figuren visar ett trebitars-register. När man räknar med ”tal med tecken” är de negativa talen den vänstra halvan av ringen.



2-komplementet ”snabbt”

- För att lätt ta fram 2-komplementet av ett binärtal kan man använda följande förfarande:
 - Börja från höra sidan
 - Kopiera alla bitar från binärtalet som är 0 och den första 1:an
 - Invertera alla andra bitar

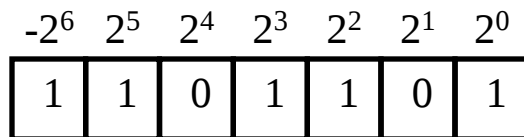
Exempel: 2-komplement från



Sign-extension

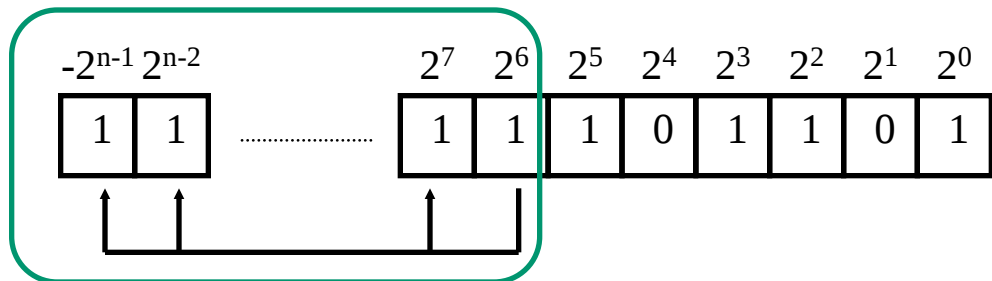
Vid beräkningar i datorer behöver man ofta öka antalet siffror (bitar) inför någon beräkning – hur gör man det med negativa tal?

Heltal:



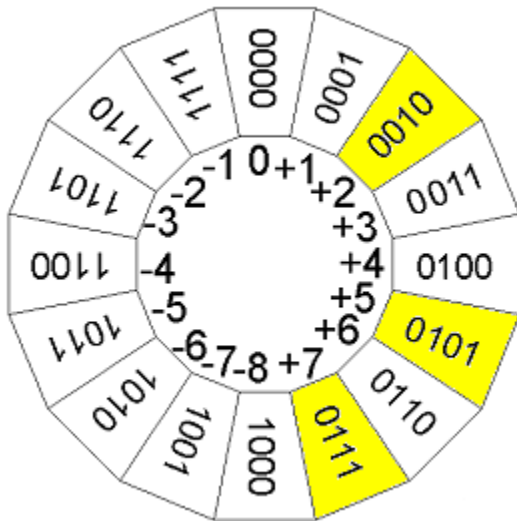
$$= -1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 = -45$$

Teckenbiten har negativ vikt



Om man vill utvidga talområdet genom att använda flera bitar kopierar man teckenbiten till de nya bitarna!

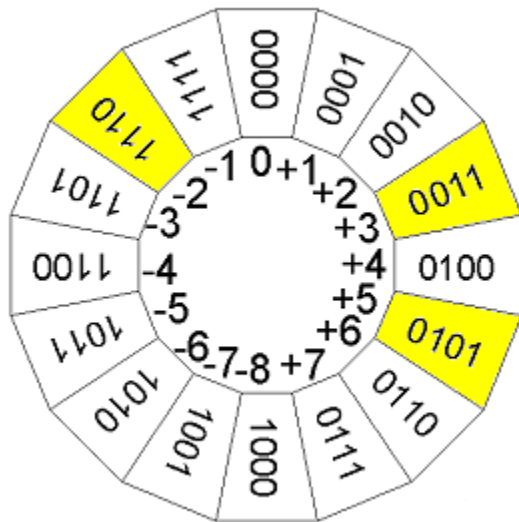
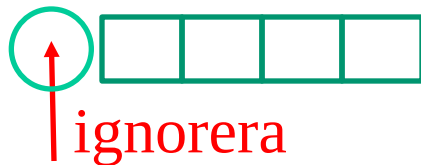
Addition (BV: sida 264)



$$\begin{array}{r}
 (+5) \\
 + \quad (+2) \\
 \hline
 (+7)
 \end{array}$$

$$\begin{array}{r}
 0101 \\
 + \quad 0010 \\
 \hline
 0111
 \end{array}$$

Addition (BV: sida 264)



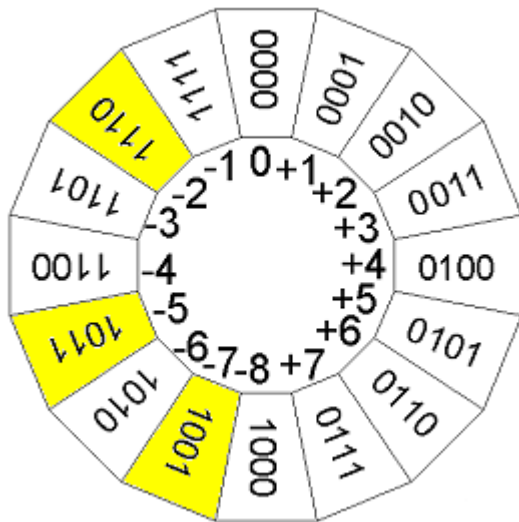
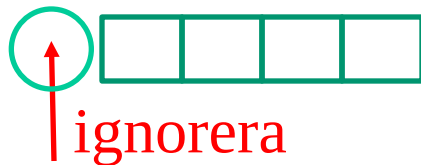
$$\begin{array}{r}
 (+5) \\
 + \quad (-2) \\
 \hline
 (+3)
 \end{array}$$

$$\begin{array}{r}
 \textcolor{red}{1} \text{ } \textcolor{red}{1} \\
 \hline
 0101 \\
 + \quad 1110 \\
 \hline
 \textcolor{red}{1} \quad 0011
 \end{array}$$

↑

Carry-biten kan ignoreras!

Addition (BV: sida 264)



$$\begin{array}{r} (-5) \\ + (-2) \\ \hline (-7) \end{array}$$

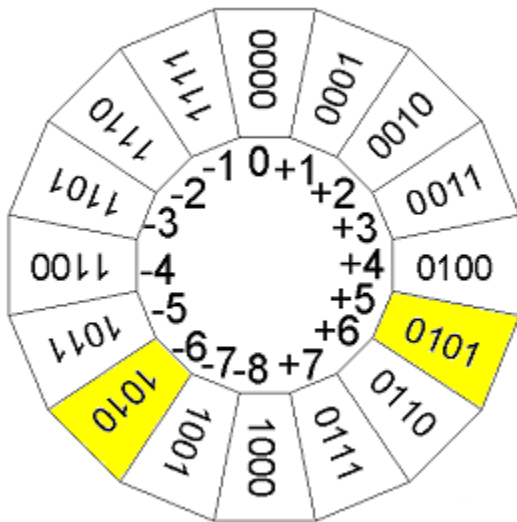
$$\begin{array}{r} \textcolor{red}{1} \text{ } 11 \\ \text{ } 1011 \\ + 1110 \\ \hline \textcolor{red}{1} \text{ } 1001 \end{array}$$

Carry-biten kan ignoreras!

Overflow



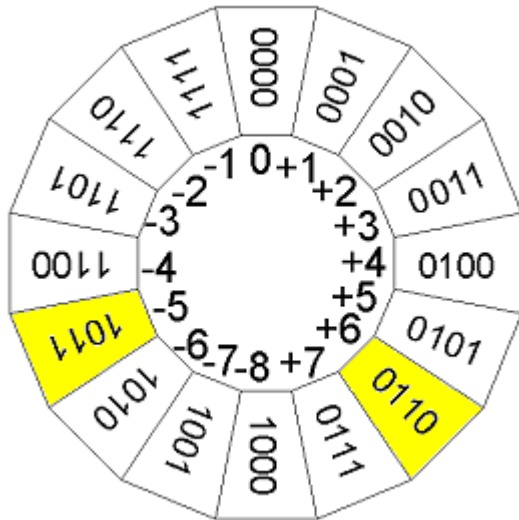
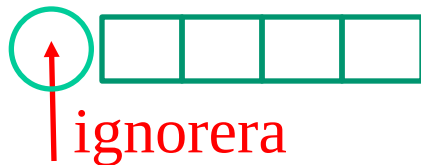
Overflow – teckenbiten stämmer *inte* överens med ingående tal...



$$\begin{array}{r}
 (+5) \\
 + \quad (+5) \\
 \hline
 (-6)
 \end{array}$$

$$\begin{array}{r}
 \begin{array}{cc} \underline{1} & \underline{1} \end{array} \\
 \begin{array}{ccc} 0 & 1 & 0 & 1 \\ + & 0 & 1 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 \end{array}
 \end{array}$$

Overflow 2



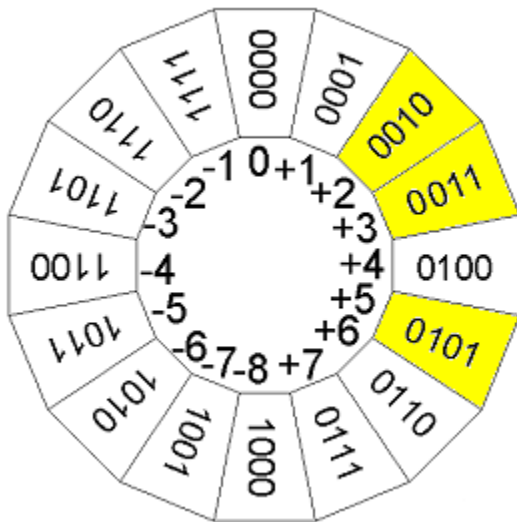
Overflow – teckenbiten stämmer *inte* överens med ingående tal...

$$\begin{array}{r}
 (-5) \\
 + (-5) \\
 \hline
 (+6)
 \end{array}
 \qquad
 \begin{array}{r}
 \overset{1}{\text{ }} \quad \overset{11}{\text{ }} \\
 \text{1011} \\
 + \text{1011} \\
 \hline
 \overset{1}{\text{ }} \text{0110}
 \end{array}$$

In the second calculation, the carry bit '1' at the top and the '0' in the result '0110' are highlighted with green boxes and red arrows, indicating they are the focus of the overflow discussion.

Carry-biten kan ignoreras!

Subtraktion (BV: sida 265)



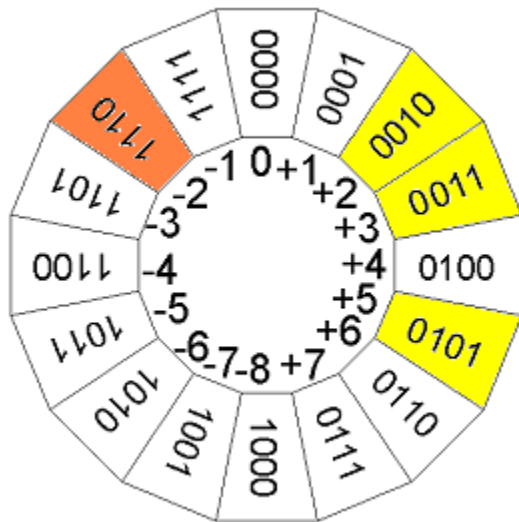
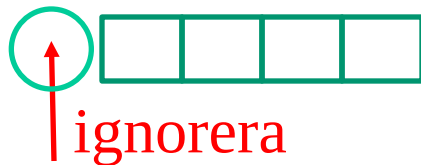
$$\begin{array}{r} (+5) \\ - (+2) \\ \hline (+3) \end{array}$$

"Låna" ett

$$\begin{array}{r} \text{10} \\ \text{0101} \\ - \text{0010} \\ \hline \text{????} \end{array}$$

Hur gör man subtraktionen på ett enkelt sätt?

Subtraktion (BV: sida 265)



$$\begin{array}{r} (+5) \\ - (+2) \\ \hline (+3) \end{array}$$

$$\begin{array}{r} 0101 \\ - 0010 \\ \hline \end{array}$$

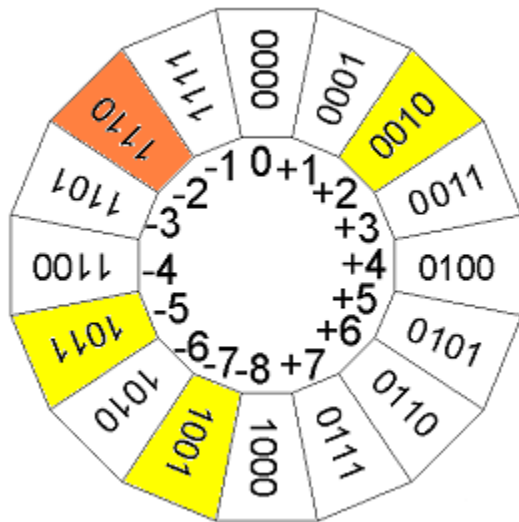
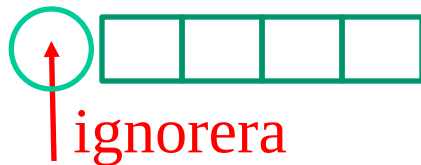
????

$$\begin{array}{r} \underline{1} \underline{1} \\ 0101 \\ + 1110 \\ \hline 1 \ 0011 \end{array}$$

Carry-biten kan ignoreras!

Gör en addition med 2-komplementet i stället!

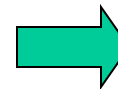
Subtraktion (BV: sida 265)



$$\begin{array}{r} (-5) \\ - (+2) \\ \hline (-7) \end{array}$$

$$\begin{array}{r} 1011 \\ - 0010 \\ \hline \end{array}$$

????

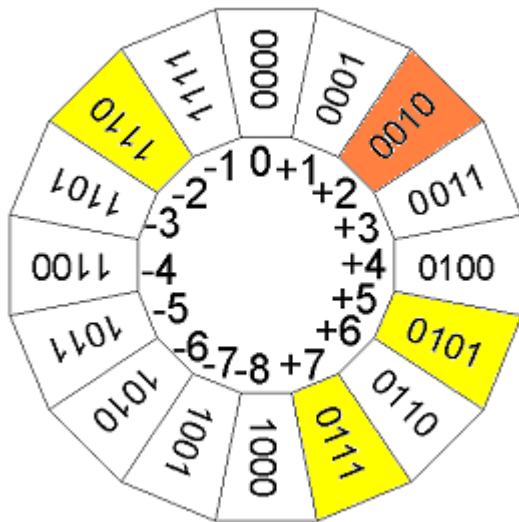


$$\begin{array}{r} \underline{1} \underline{1} \underline{1} \\ 1011 \\ + 1110 \\ \hline \end{array}$$

1 1001

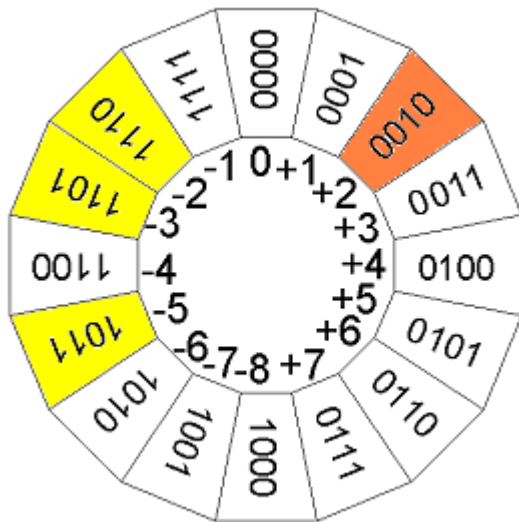
Carry-biten kan ignoreras!

Subtraktion (BV: sida 265)



$$\begin{array}{r}
 - \quad (+5) \\
 - \quad (-2) \\
 \hline
 (+7)
 \end{array}
 \quad
 \begin{array}{r}
 1011 \\
 - 1110 \\
 \hline
 \text{????}
 \end{array}
 \xrightarrow{\text{teal arrow}}
 \begin{array}{r}
 + \quad 0101 \\
 + \quad 0010 \\
 \hline
 0111
 \end{array}$$

Subtraktion (BV: sida 265)

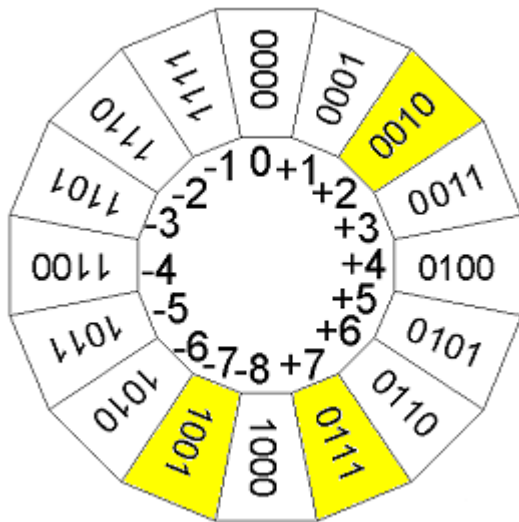


$$\begin{array}{r}
 (-5) \\
 - \quad (-2) \\
 \hline
 (-3)
 \end{array}
 \quad
 \begin{array}{r}
 1011 \\
 - \quad 1110 \\
 \hline
 \text{????}
 \end{array}
 \quad
 \begin{array}{r}
 \xrightarrow{\text{teal arrow}} \\
 \begin{array}{r}
 + \quad \overset{1}{\cancel{1}}011 \\
 \quad \quad 0010 \\
 \hline
 1101
 \end{array}
 \end{array}$$

2-komplement sammanfattning

- **Område:** -2^{N-1} upp till $2^{N-1} - 1$
- **Negation:** Invertera varje bit (det boolska komplementet), addera sedan 1.
- **Expansion av bit-längd:** Lägg till ytterligare bit positioner till vänster om teckenbiten, med samma värde som teckenbiten.
- **Overflow-regeln:** Om två nummer med samma tecken adderas, så har det blivit overflow om resultatet har ett motsatt tecken.
- **Subtraherings-regeln:** För att subtrahera B från A, ta två-komplementet av B och addera till A.

Alternativt sätt att detektera overflow (BV: sida 271)

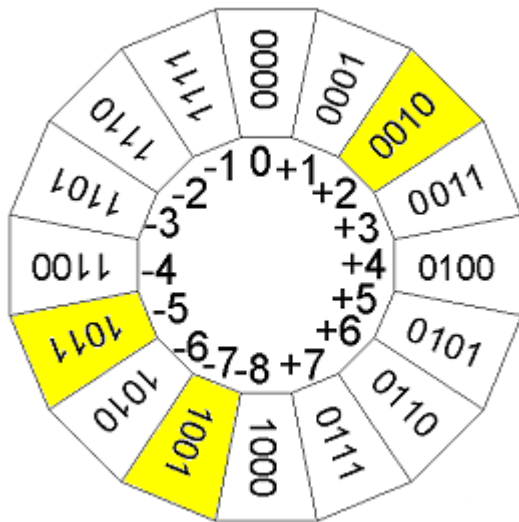


$$\begin{array}{r} (+7) \\ + (+2) \\ \hline (-7) \end{array}$$

$$\begin{array}{r} c_4=0 \quad c_3=1 \\ \quad \underline{0} \quad \underline{11} \\ \quad \underline{0111} \\ + \quad 0010 \\ \hline 1001 \end{array}$$

Overflow eftersom c_4 och c_3 är olika!

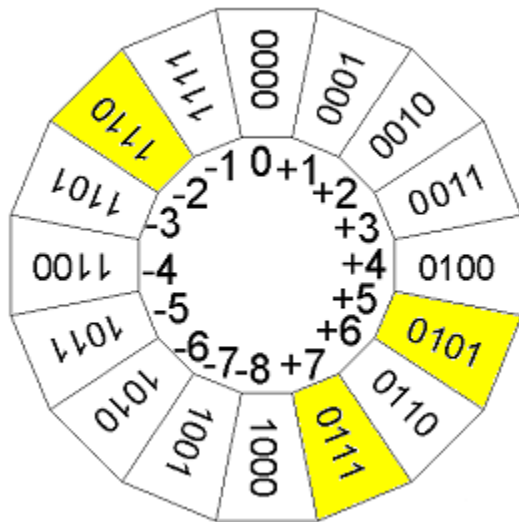
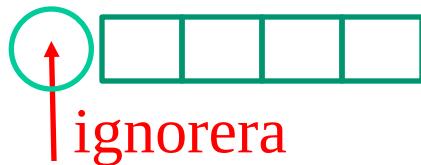
Alternativt sätt att detektera overflow (BV: sida 271)



$$\begin{array}{r}
 (-7) \\
 + \quad (+2) \\
 \hline
 (-5)
 \end{array}
 \qquad
 \begin{array}{r}
 c_4=0 \quad c_3=0 \\
 \quad \quad \quad \underline{0} \quad \underline{0} \\
 \quad \quad \quad 1001 \\
 + \quad 0010 \\
 \hline
 1011
 \end{array}$$

Inte **Overflow** eftersom c_4 och c_3 är lika!

Alternativt sätt att detektera overflow (BV: sida 271)



$$\begin{array}{r} (+7) \\ + (-2) \\ \hline (+5) \end{array}$$

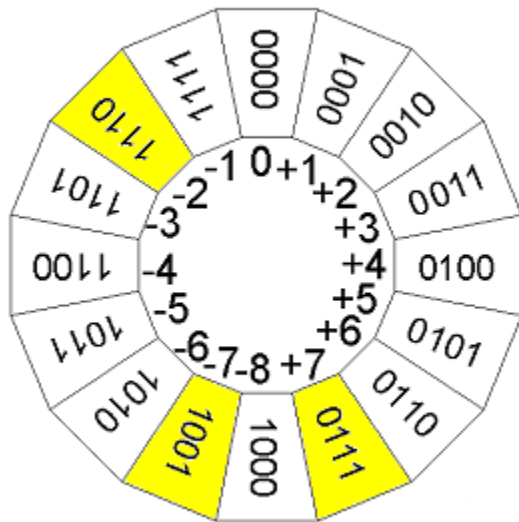
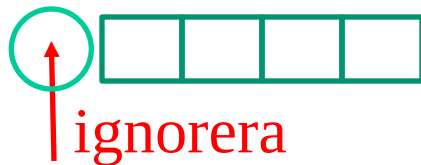
$c_4=1$ $c_3=1$
 $\downarrow$ $\swarrow$
1 11
 0111
+ 1110

1 0101

Carry-biten kan ignoreras!

Inte **Overflow** eftersom c_4 och c_3 är lika!

Alternativt sätt att detektera overflow (BV: sida 271)



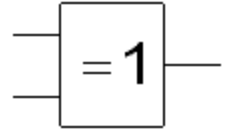
$$\begin{array}{r} (-7) \\ + (-2) \\ \hline (+7) \end{array}$$

$$\begin{array}{r} c_4=1 \quad c_3=0 \\ \underline{1} \quad \underline{0} \\ 1001 \\ + 1110 \\ \hline 1 \quad 0111 \end{array}$$

Carry-biten kan ignoreras!

Overflow eftersom c_4 och c_3 är olika!

Logik för att detektera overflow



*XOR testar
”olikhet”*

För 4-bit-tal

Overflow om c_3 och c_4 är *olika*

Annars är det inte overflow

$$\text{Overflow} = c_3 \bar{c}_4 + \bar{c}_3 c_4 = c_3 \oplus c_4$$

För n -bit-tal

$$\text{Overflow} = c_{n-1} \oplus c_n$$

En Snabbfråga ...

Talet +5 representeras med 0101 om vi använder 4 bitar.

Vilket tal motsvarar -5 om vi använder 8 bitars två-komplement representation?

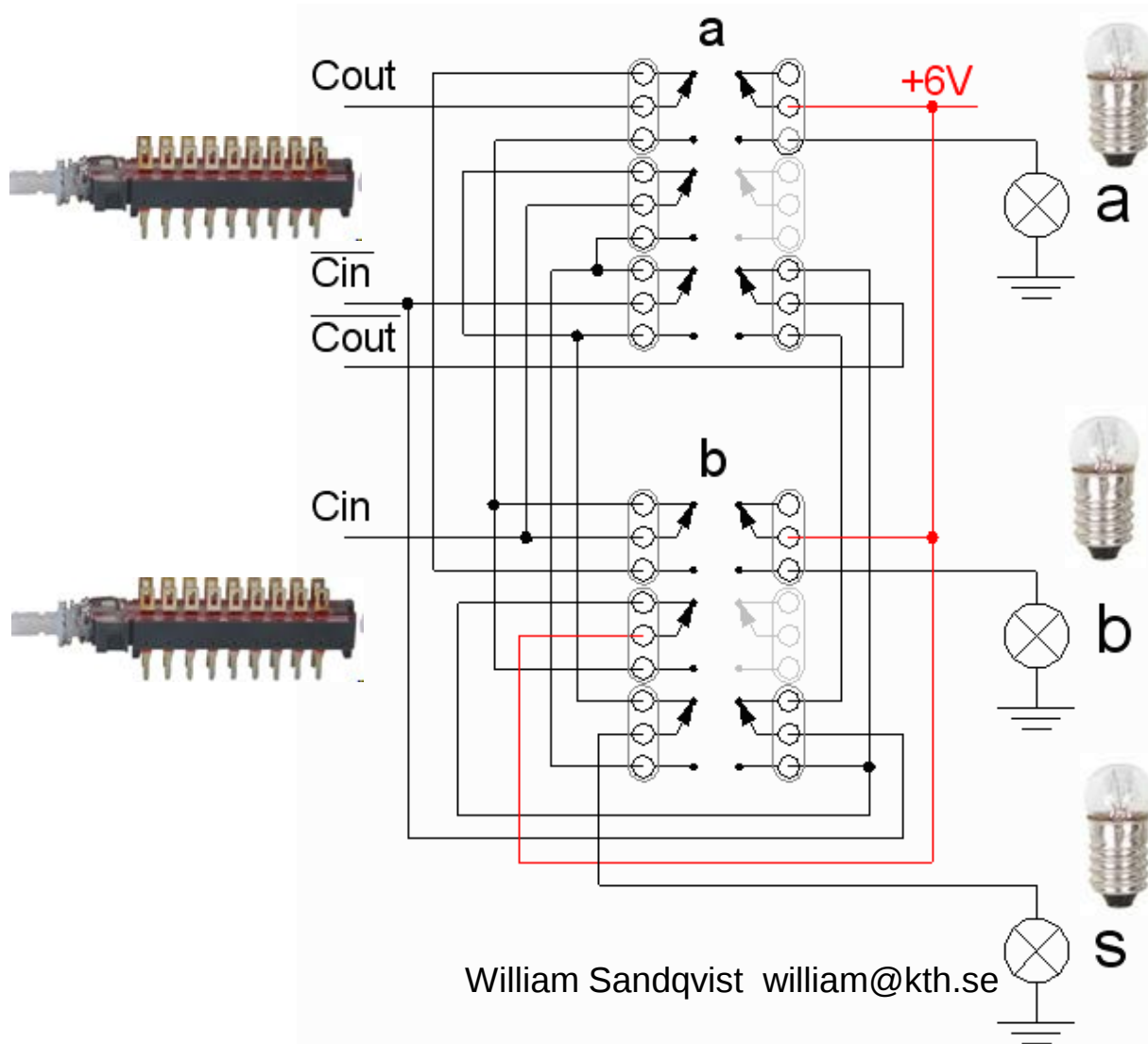
a: 1111 1011

b: 1111 1010

c: 1000 0101



1939 heladderare med kontaktnät

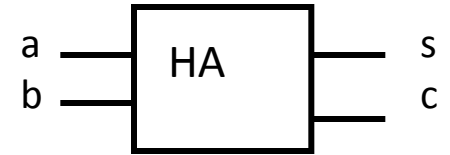


Konrad Zuse
Dual rail carry
full adder

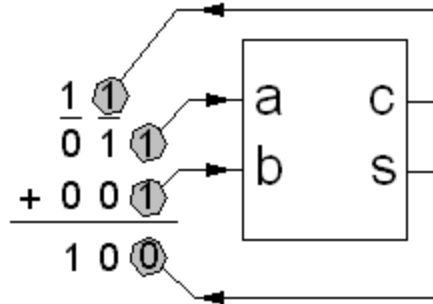
Hårdvara för aritmetik

- Nu när Hastighet/Komplexitet har betydelse!

Halv-adderaren (Half adder)



a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

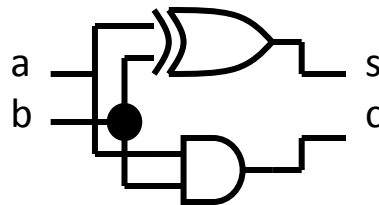


a \ b	0	1
0	0	0
1	0	1

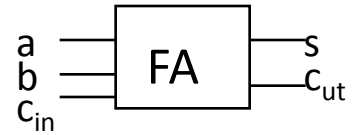
$$c = a \cdot b$$

a \ b	0	1
0	0	1
1	1	0

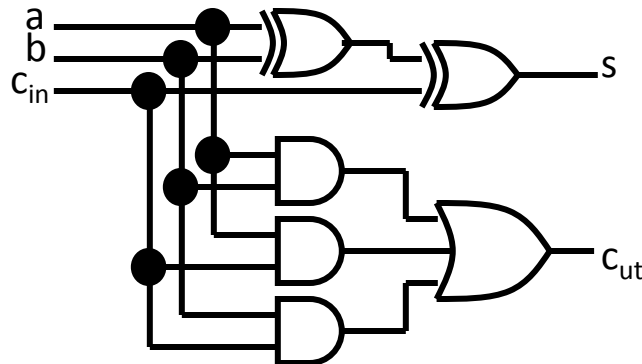
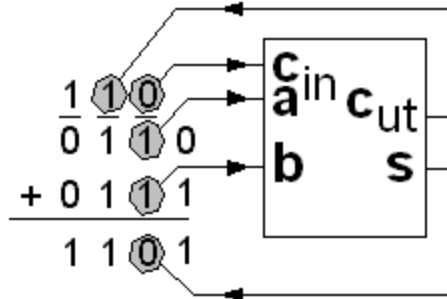
$$s = a \oplus b$$



Hel-adderaren (Full adder)



a	b	c_{in}	c_{ut}	s
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



c_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$s = a \oplus b \oplus c_{in}$$

ab	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$c_{ut} = a \cdot b + c_{in} \cdot a + c_{in} \cdot b$$

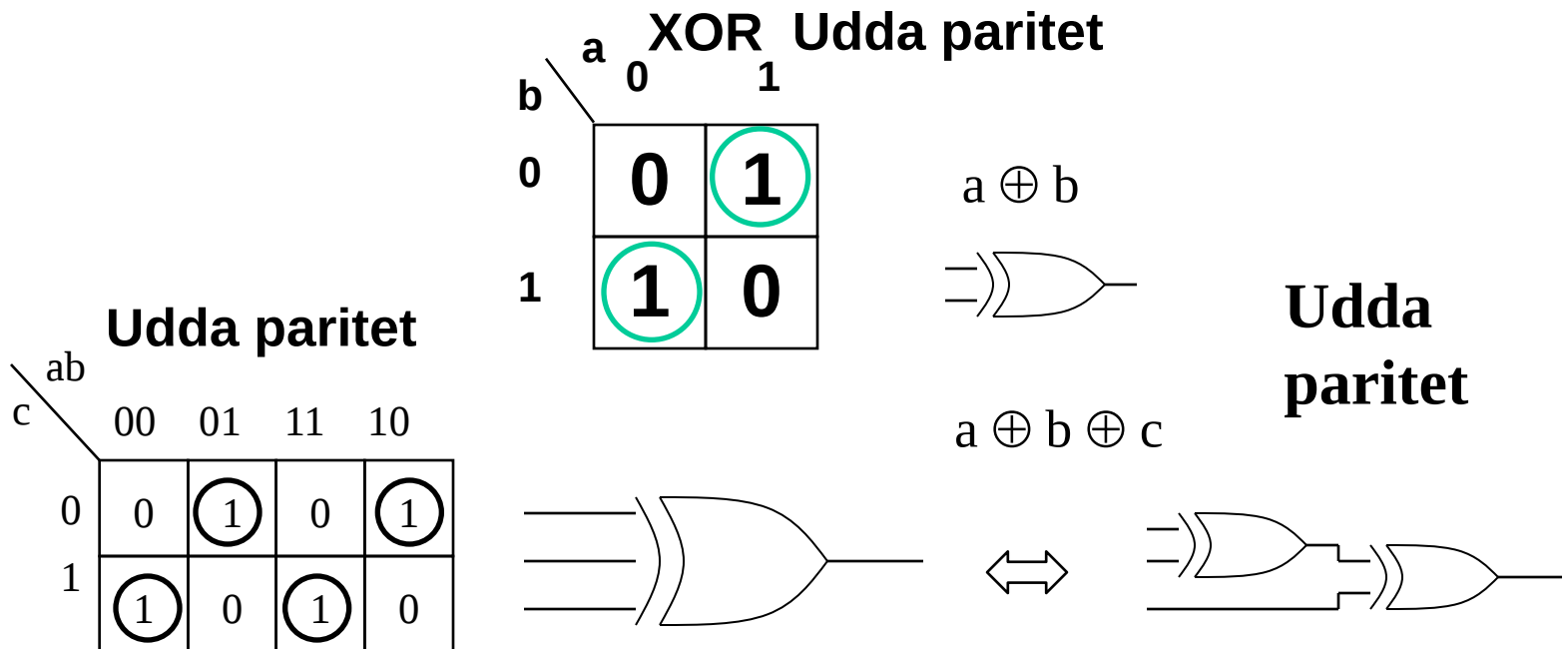
Summafunktionen?

c_{in} \	00	01	11	10
0	0	1	0	1
1	1	0	1	0

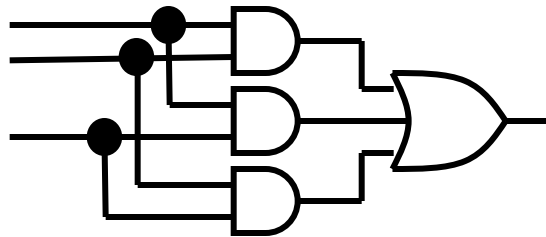
$s = a \oplus b \oplus c_{in}$

Summa = Udda paritet

Heladderarens summafunktion är den "udda"-paritetsfunktionen. Detta är XOR-funktionens naturliga utökning för fler variabler än två. **Udda paritet** är när antalet 1:or på ingångarna är ett udda tal.

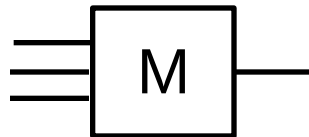


Carryfunktionen?



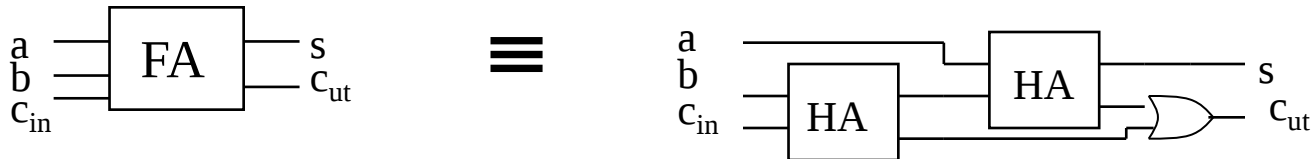
	00	01	11	10
0	0	0	1	0
1	0	1	1	1

- **Majoritetsfunktionen.** Utgången antar det som är flest 1/0 på ingångarna.



Hel-adderaren med två $\frac{1}{2}$ -adderare

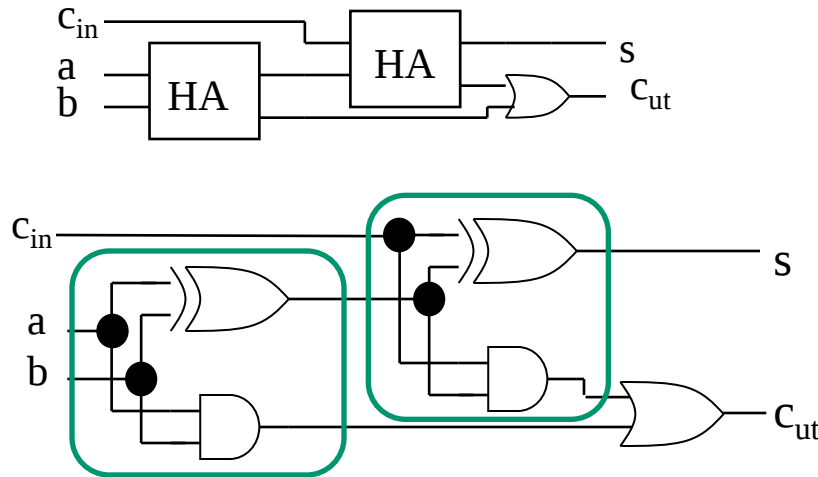
Vi kan även konstruera en hel-adderare mha två halv-adderare och en OR-grind



Dekomposition innebär att man ser kretsen som sammansatt av byggblock. Med hjälp av sådana kända byggblock kan man sedan bygga helt nya system **Komposition**.

Hel-adderaren $\frac{1}{2} + \frac{1}{2} = 1$

a	b	c _{in}	c _{ut}	s
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



c _{in} \ ab	ab			
	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$a \cdot b$

$(a \oplus b) \cdot c_{in}$

$c_{ut} = (a \oplus b) \cdot c_{in} + a \cdot b$

Man kan använda $(a \oplus b)$
till både s och c_{ut} !

$$s = (a \oplus b) \oplus c_{in}$$

$$c_{ut} = (a \oplus b) \cdot c_{in} + a \cdot b$$

c _{in} \ ab	ab			
	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$s = a \oplus b \oplus c_{in}$

Populär tatuering?



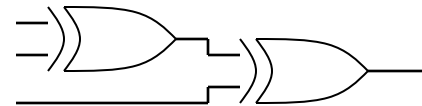
Tatueringar är för evigt! Tyvärr är detta inte den ”bästa” adderarkretsen, inte om man vill ha snabba datorer. Spännande fortsättning om adderarkretsar följer ...

(Paritetsfunktionen – trevägs ljuskontroll)

c_{in} \	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$s = a \oplus b \oplus c_{in}$

Udda paritet.

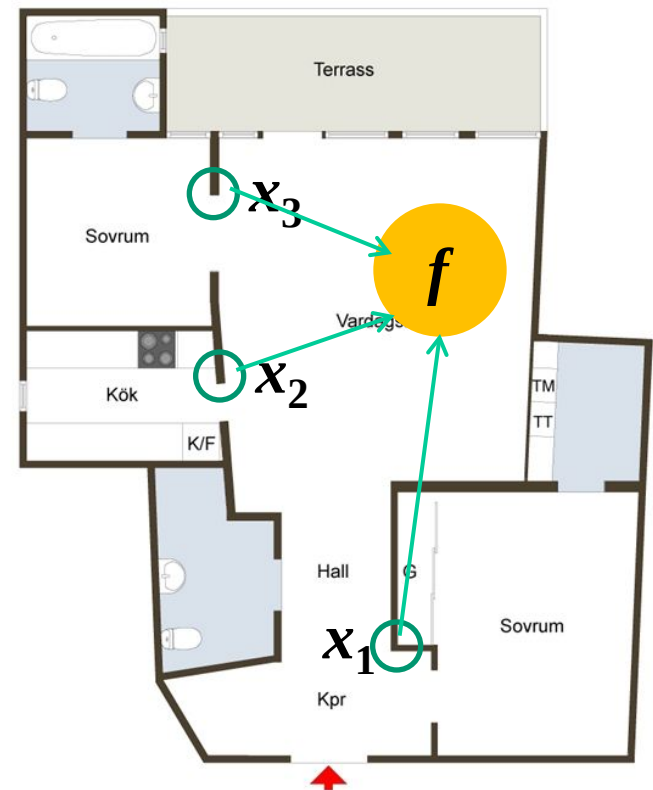


Trevägs ljuskontroll - *revisited*

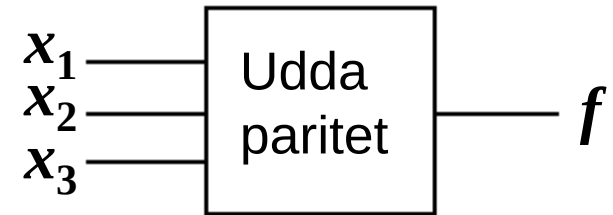
Brown/Vranesic: 2.8.1

Antag att vi behöver kunna tända/släcka vardagsrummet från tre olika ställen. Lösningen är paritetsfunktionen.

Paritetsfunktionen



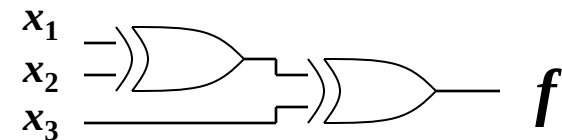
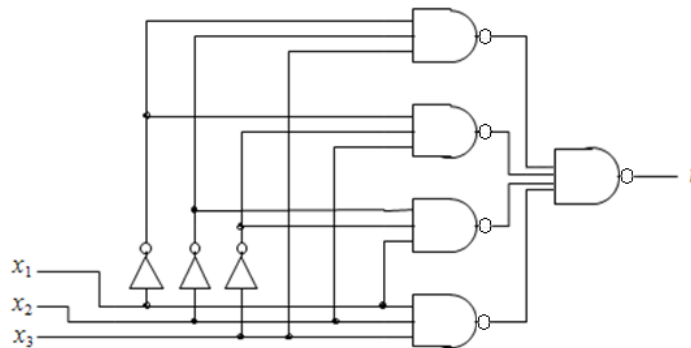
XOR eller NAND?



*Den tidigare lösningen
baserades på NAND-grindar.*

*XOR-grindar blir mycket
effektivare än NAND grindar!*

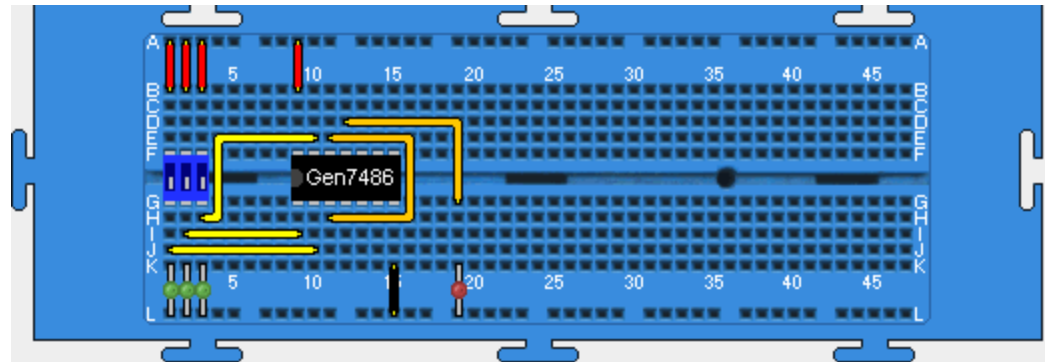
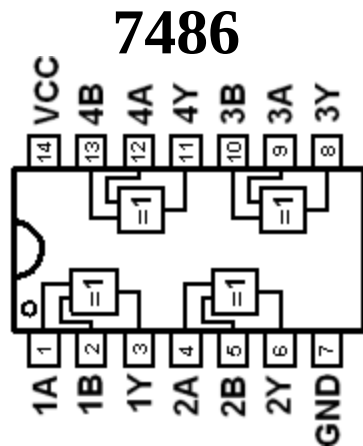
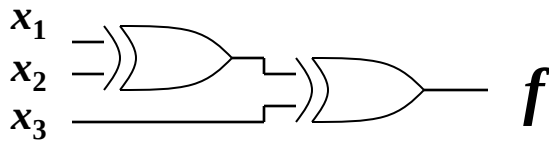
x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



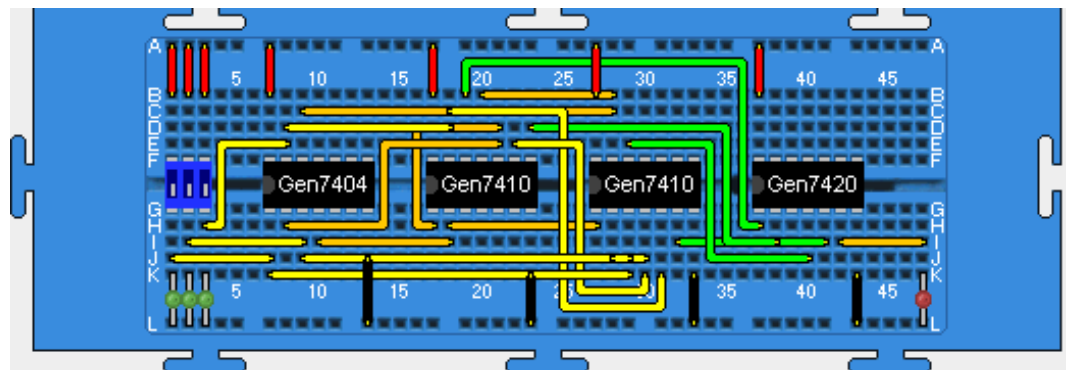
		f			
x_1	x_2 x_3	00	01	11	10
	0	0	1	0	1
1	1	1	0	1	0

Enklare med XOR-grindar

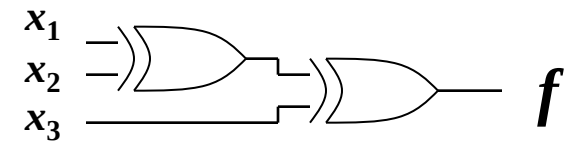
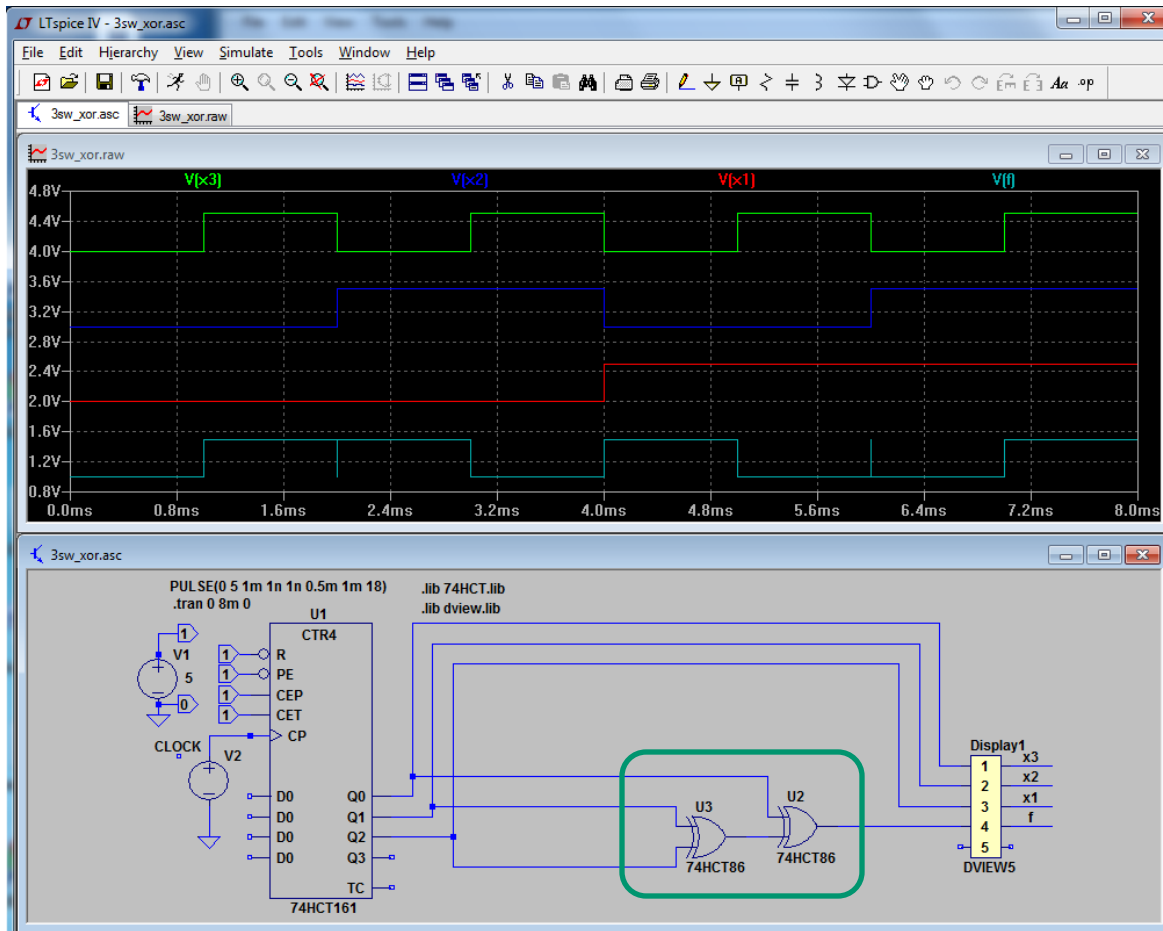
Med XOR-grindar:



(Med NAND-grindar:)



Enklare med XOR-grindar



x_1	x_2	x_3	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

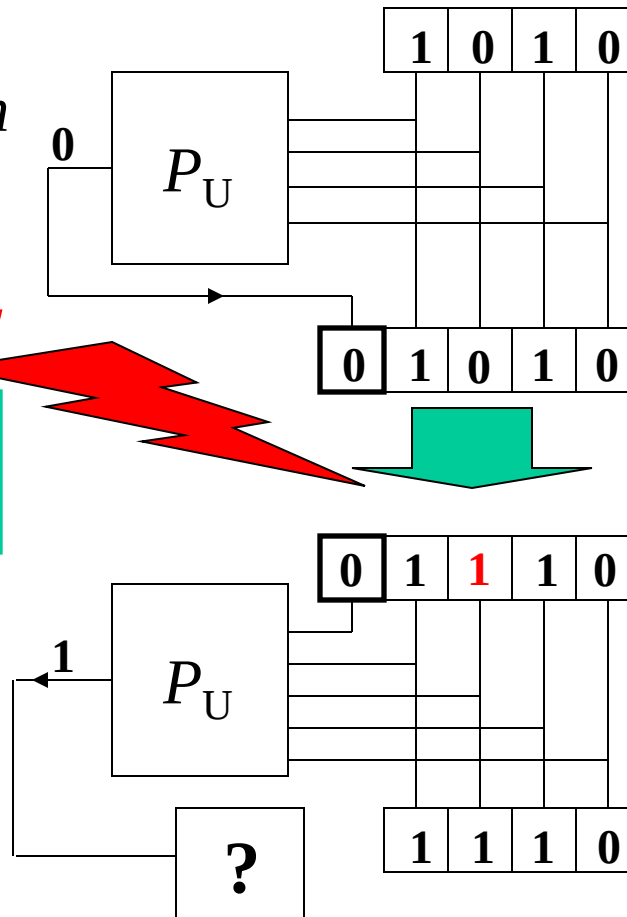
(Paritetscheck)

*Med paritetsfunktioner
kan man kontrollera om
data blivit stört eller ej.*

Störning!

*Data som överförs har
alltid jämn paritet!*

*LARM!
En bit ändrad!
Data har störts!*



Data - original

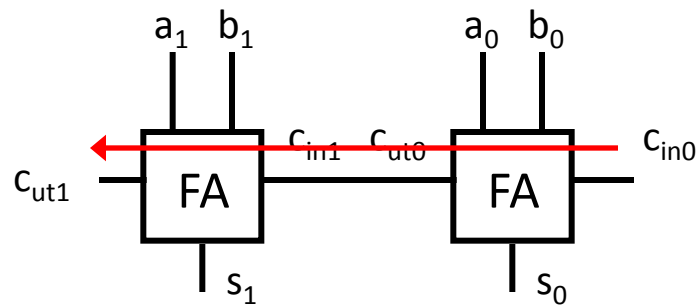
*Paritetsbit
läggs till*

*Paritetscheck
kontrollerar
om udda
antal "1:or"
Data – ev fel*

Mer **komposition**

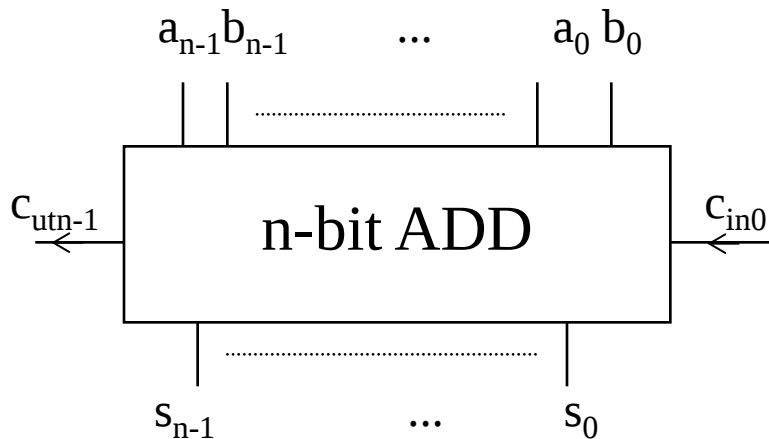
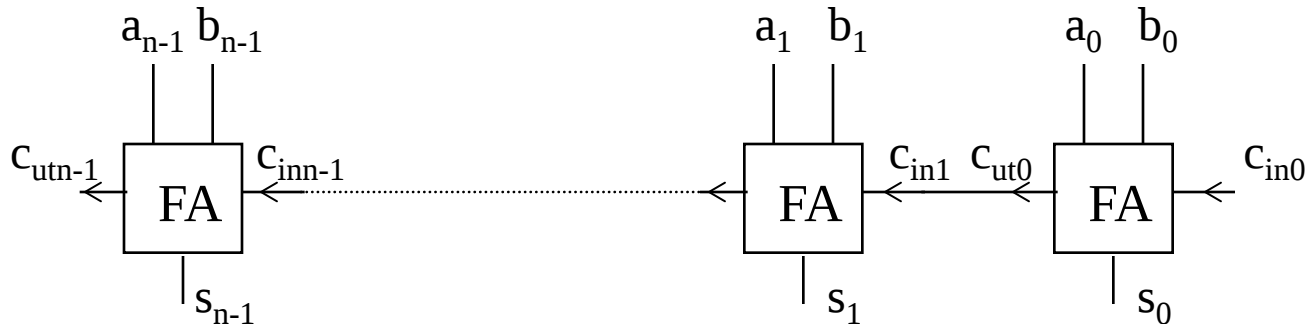
- Komposition kan även användas för att konstruera n -bit-adderare
- Man behöver n hel-adderare för att konstruera en n -bit-adderare (eller $2 \cdot n$ halvadderare)

Ripple-Carry Adder (RCA)



$$\begin{array}{r}
 \underline{C_{ut1}} \quad \underline{C_{ut0}} \quad \underline{C_{in0}} \\
 \quad \quad b_1 \quad b_0 \\
 + \quad \quad a_1 \quad a_0 \\
 \hline
 \quad \quad s_1 \quad s_0
 \end{array}$$

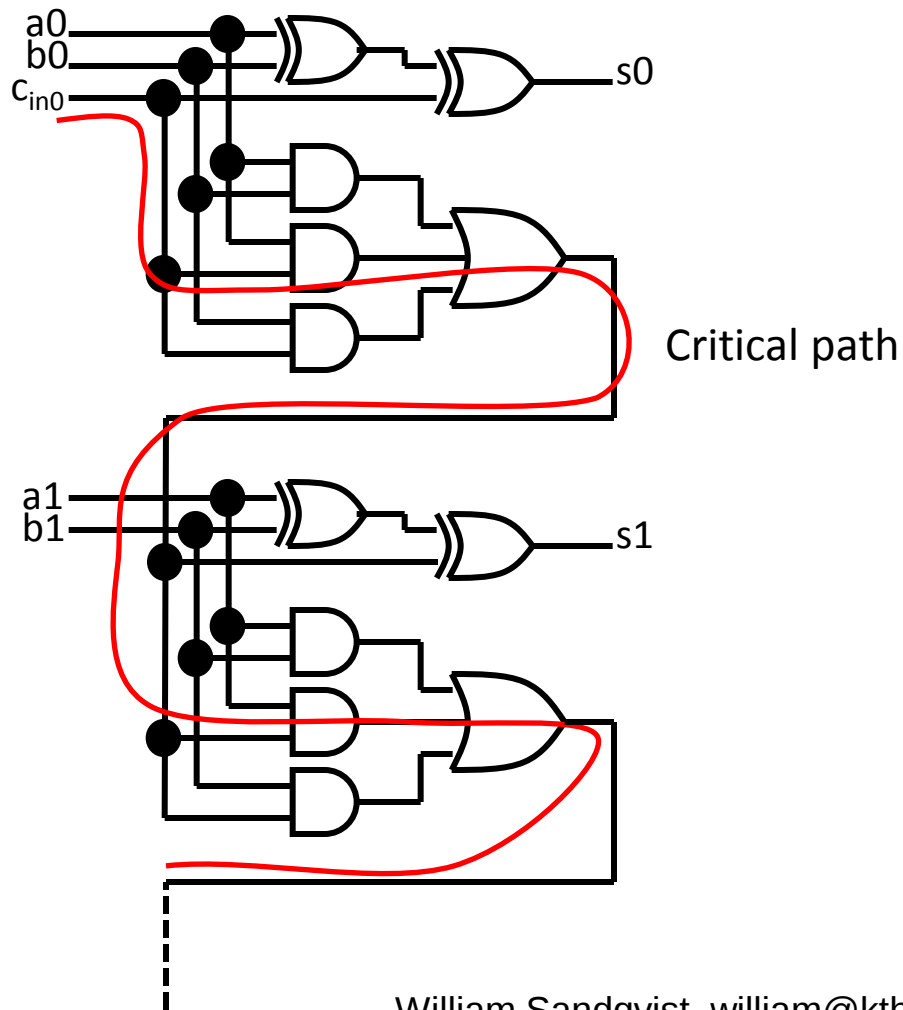
Ripple-Carry Adderare (RCA)



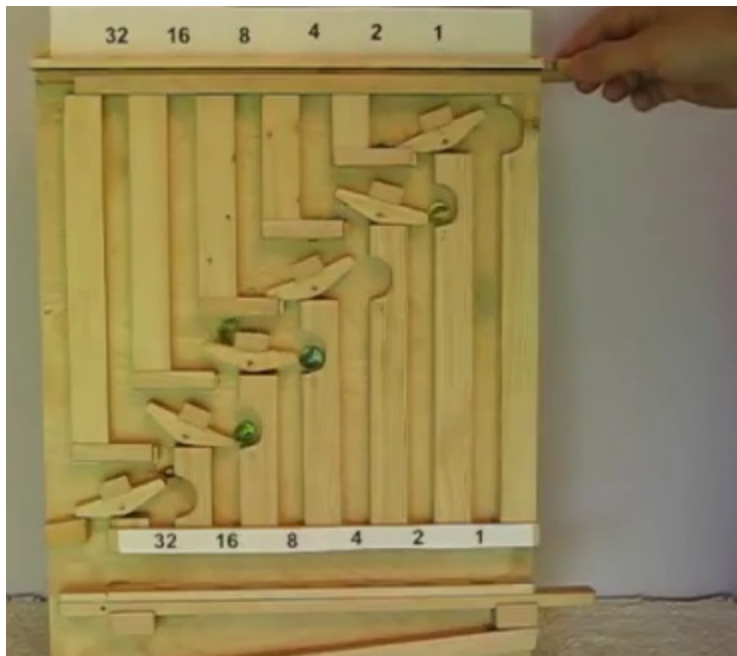
A_{FA} area för en Fulladder
 t_{FA} fördröjning genom en Fulladder

- Tidsfördröjning från c_{in0} till c_{outn-1} blir hela $n \cdot t_{FA}$
- Totala arean blir $n \cdot A_{FA}$

Ripple-Carry Adder (RCA)



Ripple effect = efterdyningar



You Tube

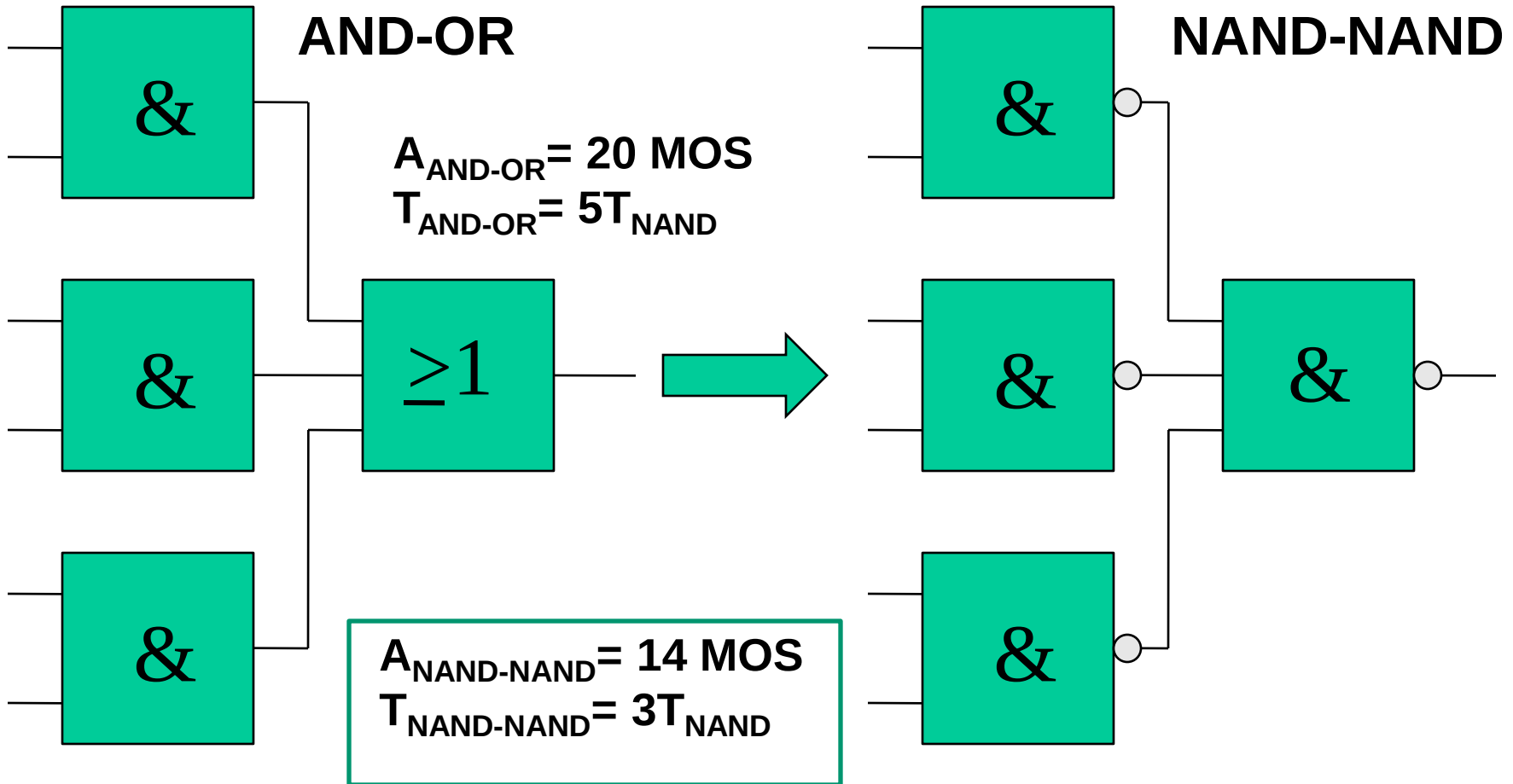
Marble adding machine

Matthiaswandel

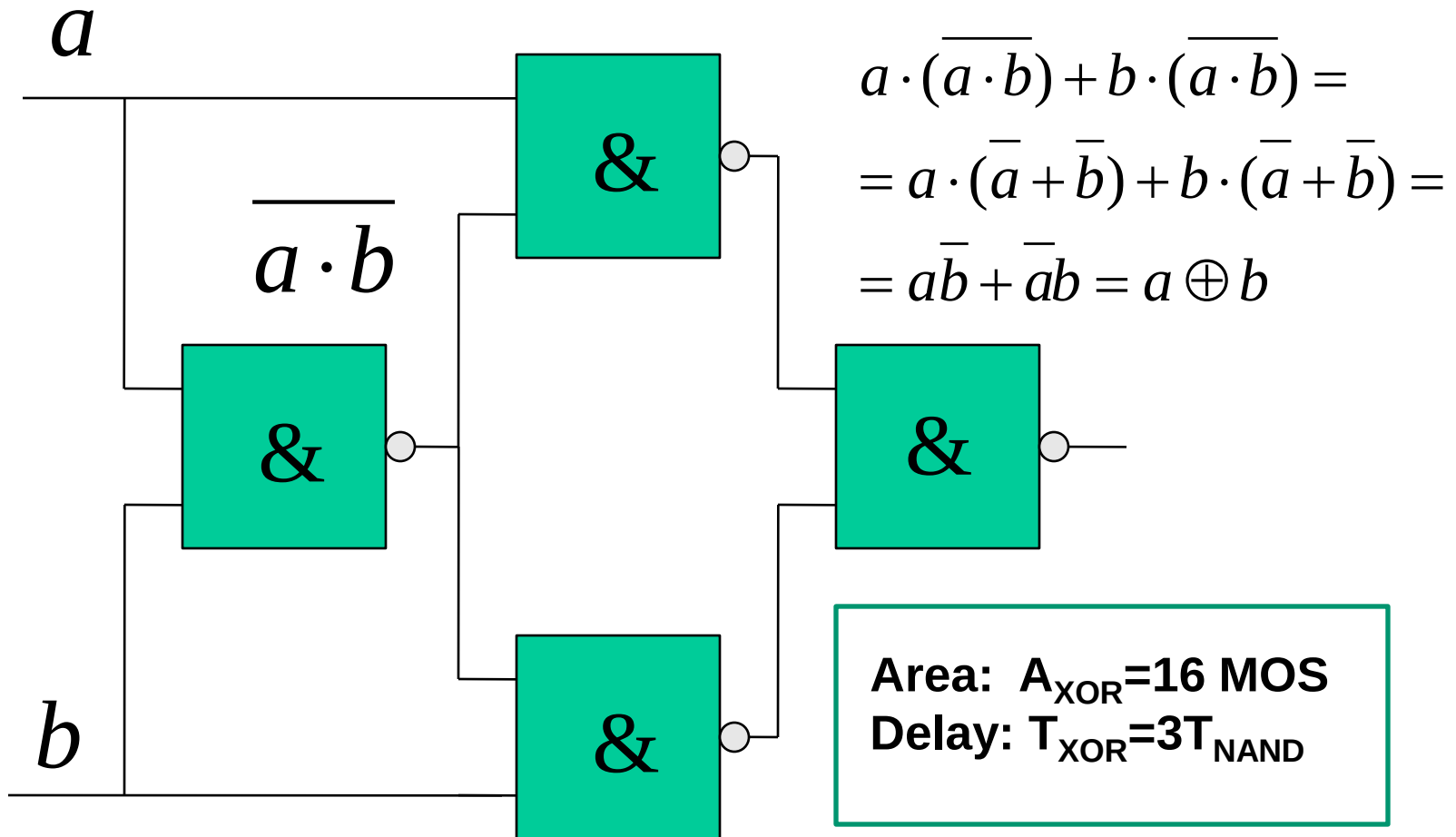


**Carry måste "rippa"
igenom hela adderaren!**

Heladderarens Carry-funktion



XOR med NAND



Kan vi konstruera en snabbare adderare?

- Fördröjningen i en ripple-adderare växer proportionellt med antalet bitar
- För 32 bitar kan fördröjningen blir mycket stor (≈ 65 gate delays)

generate- och propagate- funktionerna

Carry-kedjan kan beskrivas med två funktioner:

- **Generate** g_i (carry-out $c_{i+1} = 1$ ifall $g_i = 1$)

$$g_i = x_i y_i$$

- **Propagate** p_i (carry-out $c_{i+1} = 1$ ifall $c_i = 1$ och $x_i = 1$ eller $y_i = 1$)

$$p_i = x_i + y_i$$

$$p_i = x_i \oplus y_i$$

Fungerar också!

$$c_{i+1} = g_i + p_i c_i$$

$c_i \backslash x_i y_i$		00	01	11	10
0	0	0	0	1	0
1	0	1	1	1	1

$$c_{i+1} = x_i y_i + c_i x_i + c_i y_i$$

$$= x_i y_i + c_i (x_i + y_i)$$

$c_i \backslash x_i y_i$		00	01	11	10
0	0	0	0	1	0
1	0	1	1	1	1

$$c_{i+1} = x_i y_i + c_i (x_i \oplus y_i)$$

Carry-look-ahead funktion

- Carry-bit c_0

$$g_i = x_i y_i$$

$$c_1 = g_0 + p_0 c_0$$

- Carry-bit c_1

$$c_2 = g_1 + p_1 c_1$$

$$= g_1 + p_1 (g_0 + p_0 c_0)$$

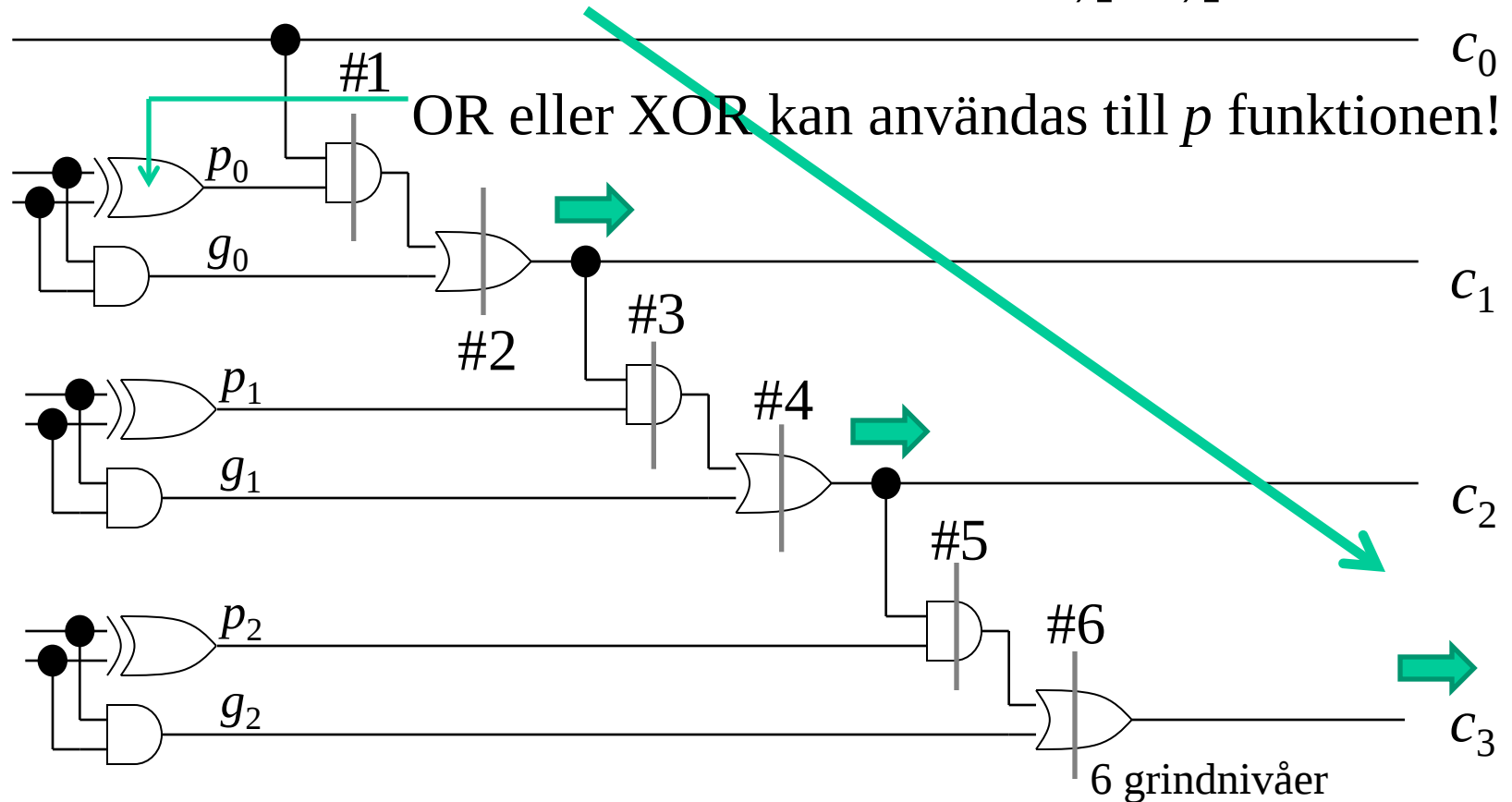
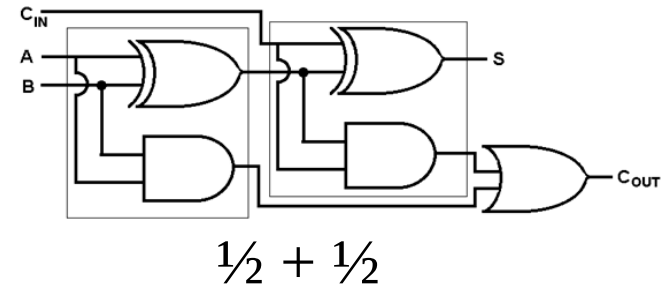
$$= g_1 + p_1 g_0 + p_1 p_0 c_0$$

Propagate funktionen
från "föregående" bitar
kan genereras parallellt

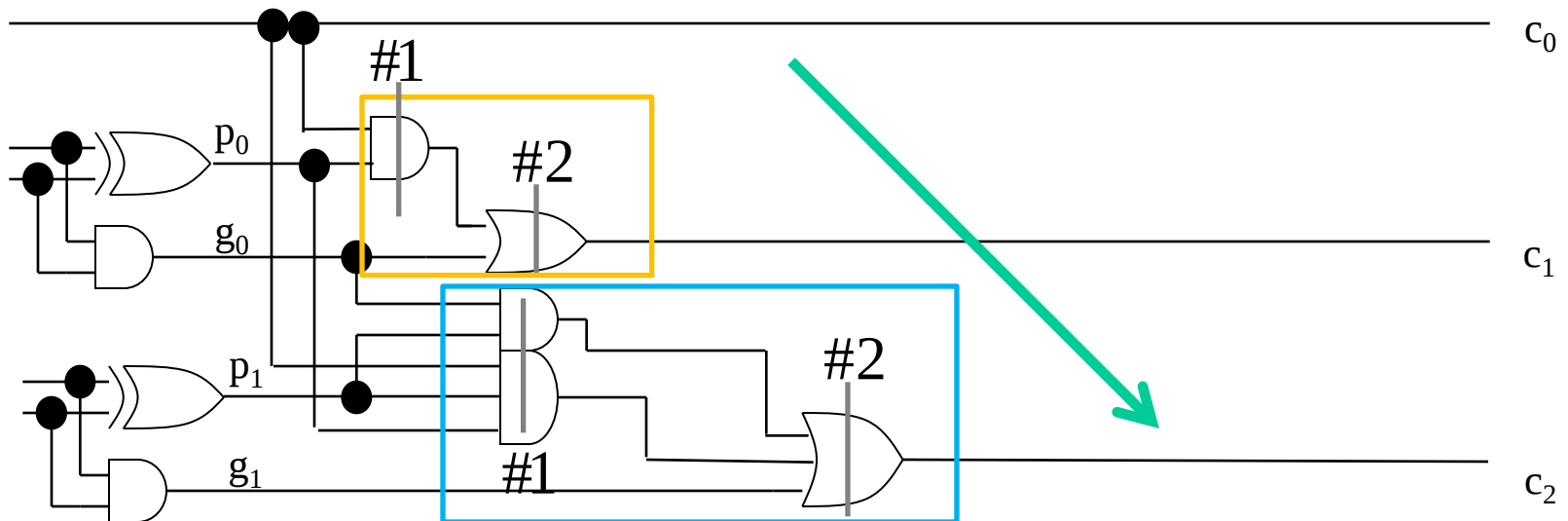
$$p_i = x_i + y_i$$

Bara två logiknivåer behövs ...

Carry-kedjan i ripple-adderaren



Snabbare implementering av Carry-kedjan



$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

- Två-nivåers nät

$(n = 8)$ med två logiknivåer

$$c_1 = g_0 + p_0 c_0$$

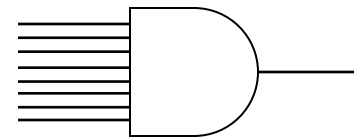
$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

**Snabbt, men otympligt med
så här många ingångar!**

·
·

Oops!

$$\begin{aligned} c_8 = & g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4 + p_7 p_6 p_5 p_4 g_3 + \\ & + p_7 p_6 p_5 p_4 p_3 g_2 + p_7 p_6 p_5 p_4 p_3 p_2 g_1 + \\ & + p_7 p_6 p_5 p_4 p_3 p_2 p_1 g_0 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0 c_0 \end{aligned}$$

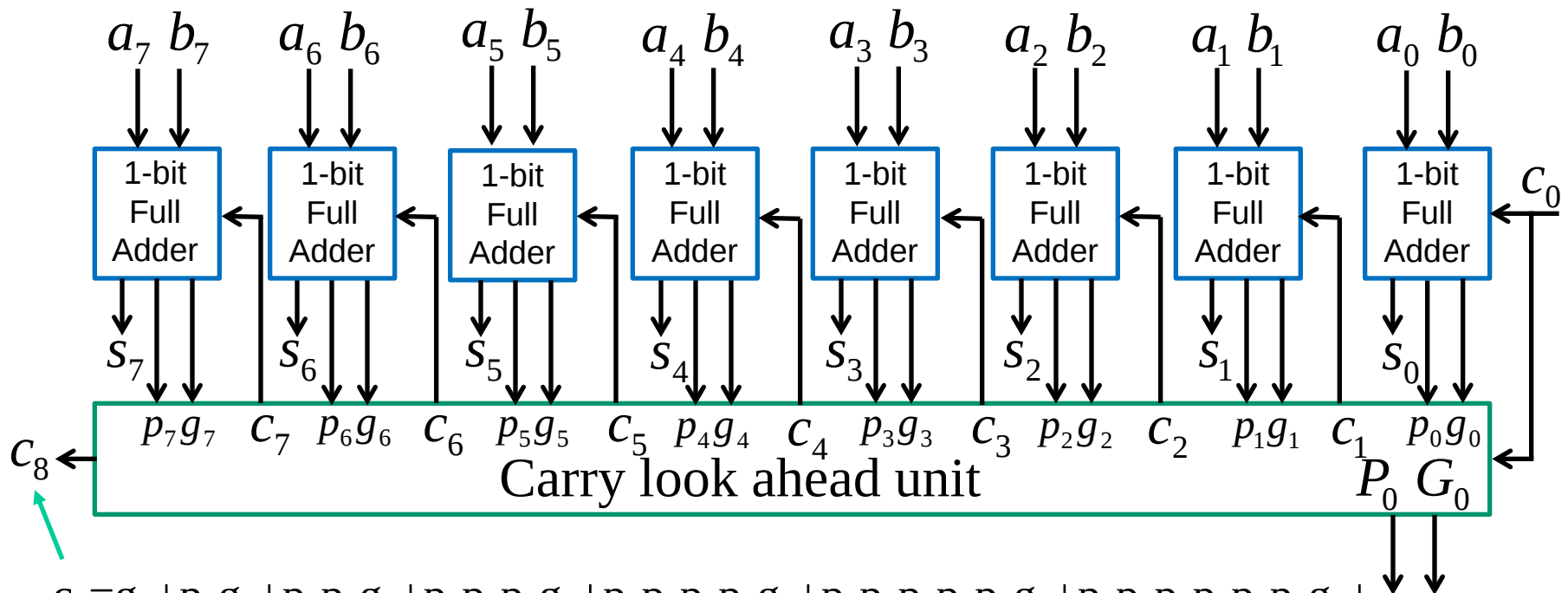


Det behövs ”specialgrindar” till detta ...

Carry-lookahead adder (CLA)

8-bit adder

$$c_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0 = a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 + b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$$

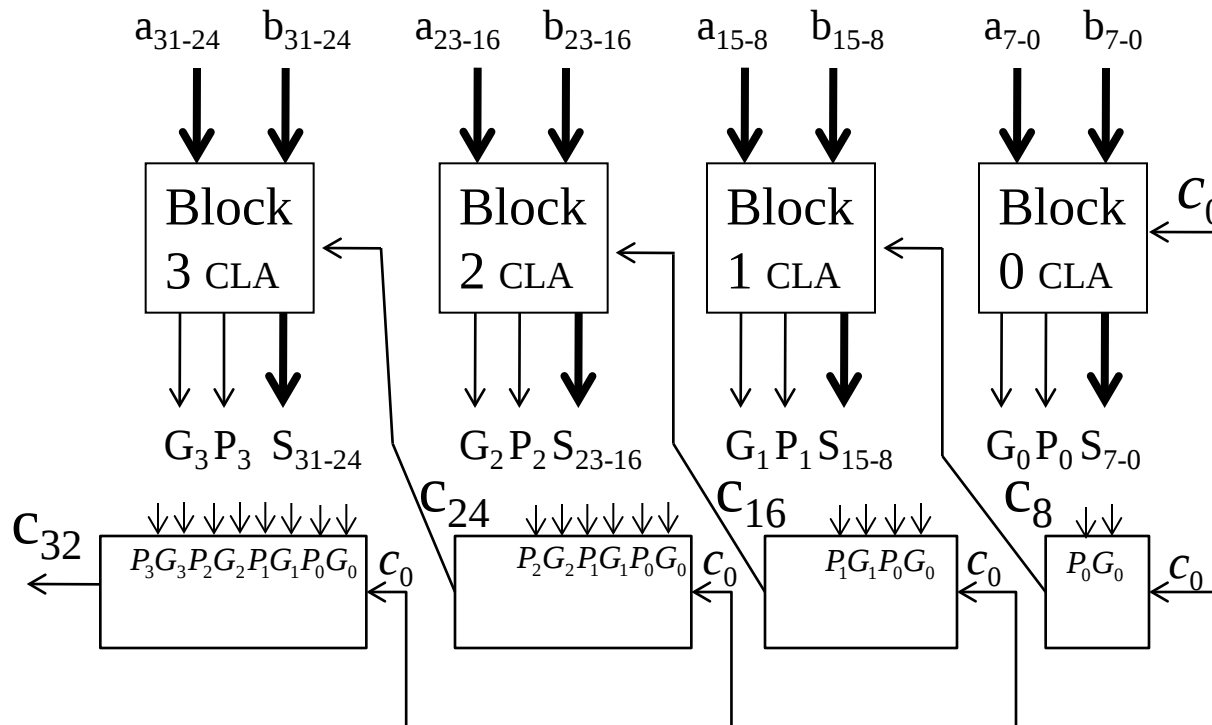


$$c_8 = g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4 + p_7 p_6 p_5 p_4 g_3 + p_7 p_6 p_5 p_4 p_3 g_2 + p_7 p_6 p_5 p_4 p_3 p_2 g_1 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 g_0 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0 c_0$$

Hierarkisk expansion (BV sid 277)

32-bit adder

32-bits adderare. Varje block består av en 8-bits CLA.



First level

four 8-bit adders with
Carry lookahead

Second level

four Carry lookahead units

Hierarkisk expansion

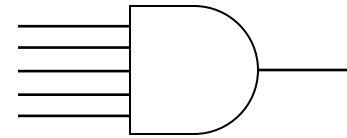
Carry-bitar från en andra nivåns Carry lookahead enheter

$$C_8 = G_0 + P_0 c_0$$

$$C_{16} = G_1 + P_1 G_0 + P_1 P_0 c_0$$

$$C_{24} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$$

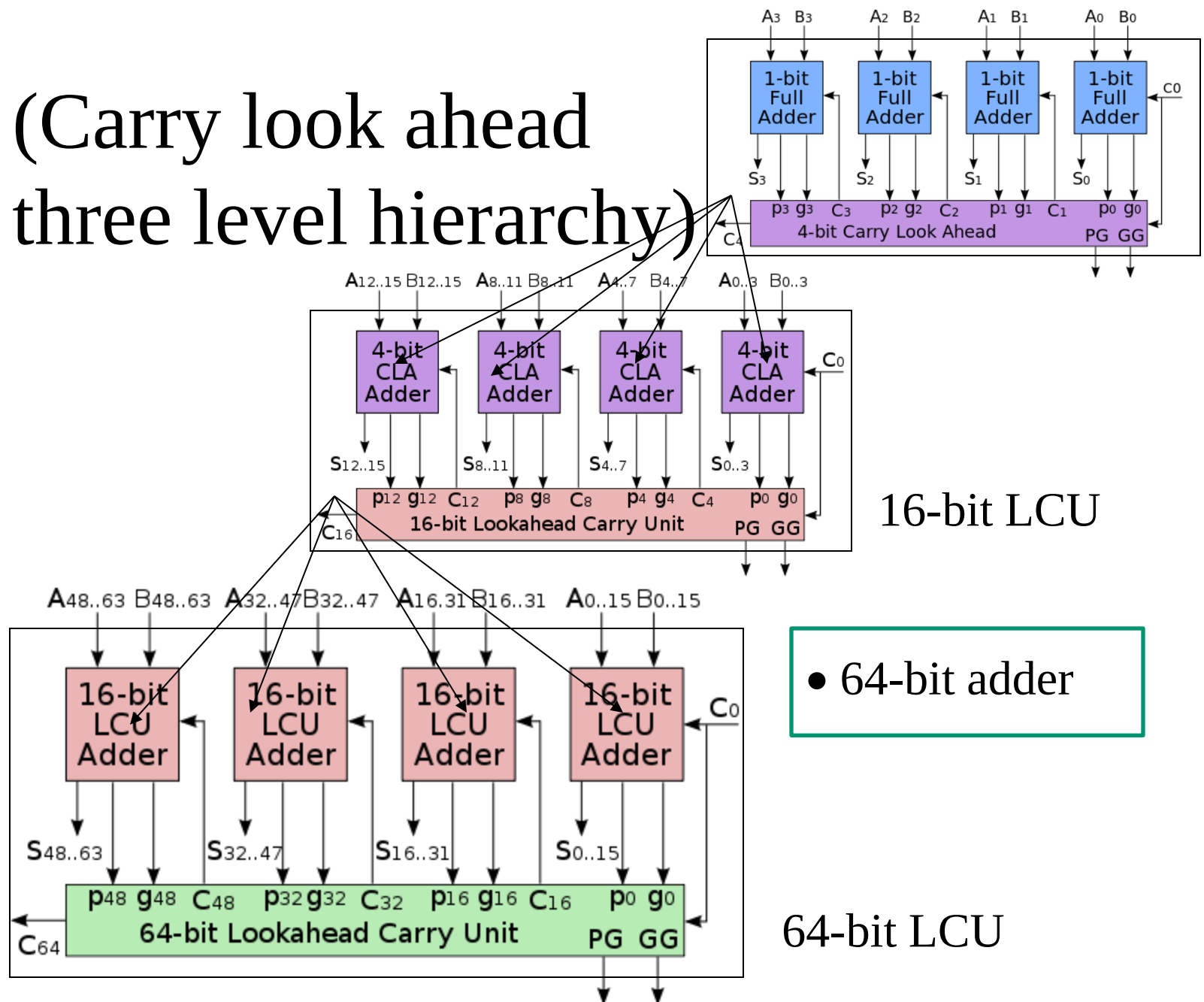
$$C_{32} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + \boxed{P_3 P_2 P_1 P_0 c_0}$$



etc.

Med Carry lookahead enheter i flera nivåer kan grindar med färre ingångar användas, men varje nivå bidrar med extra grindfördröjningar.

(Carry look ahead three level hierarchy)

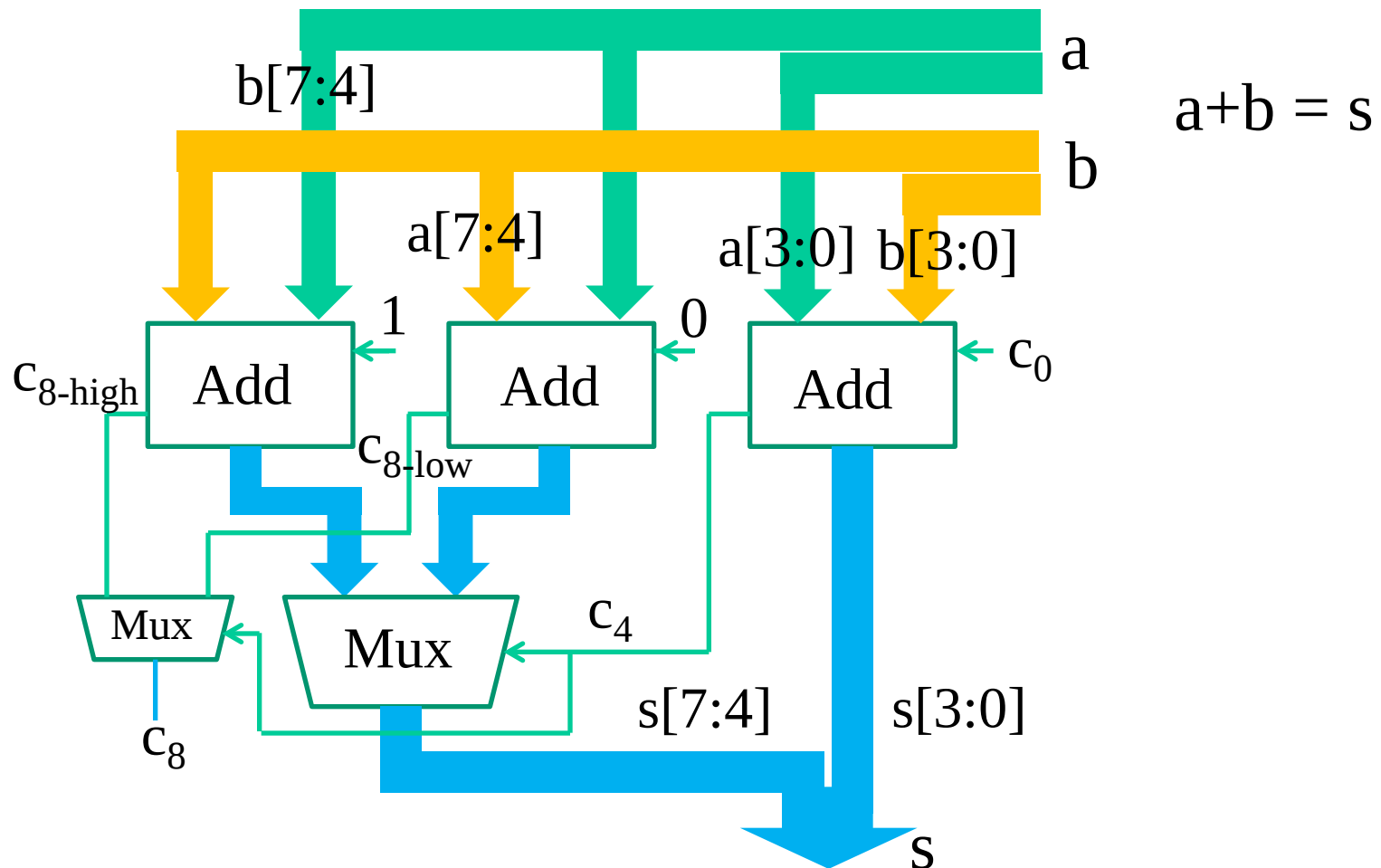


Carry-Select-Adder (CSA)

Idé

- Man delar upp en adderare i *två steg* med samma antal bitar
- För att snabba upp processen så räknar man ut resultatet av det andra steget i förväg för två fall
 - Carry-in = 0
 - Carry-in = 1
- När beräkningen av carry-biten är klar för det första steget, så väljer man resultatet av det andra steget beroende på carry-bitens värde!

8-bit Carry-Select-Adder



Jämförelse

- Ripple-Carry Adder

$$T(n) = n * 2.5 * T_{\text{NAND}}, \quad A = n * 12.5 A_{\text{NAND}}$$

$$T(4) = \mathbf{14} T_{\text{NAND}}, \quad A(4) = \mathbf{50} A_{\text{NAND}}$$

- Carry-Lookahead Adder (4bits)

$$T(4) = \sim \mathbf{8} T_{\text{NAND}}, \quad A(4) = 43 A_{\text{NAND}} + 4 * 4 * A_{\text{NAND}} = \sim \mathbf{60} A_{\text{NAND}}$$

- Carry-Select Adder (8 bits)

$$T(8) = \sim (\mathbf{8+4}) T_{\text{NAND}}, \quad A(8) = \sim (\mathbf{120+20}) A_{\text{NAND}}$$

Vilken är den bästa adderaren?

Det finns inget entydigt svar!

- Ripple-adderaren tar *minst plats* men är *långsam*
- Carry-lookahead-adderaren tar *mycket plats* men är *snabb*
- Carry-select-adderaren är en *kompromiss*

Man måste göra en *trade-off* mellan area och speed

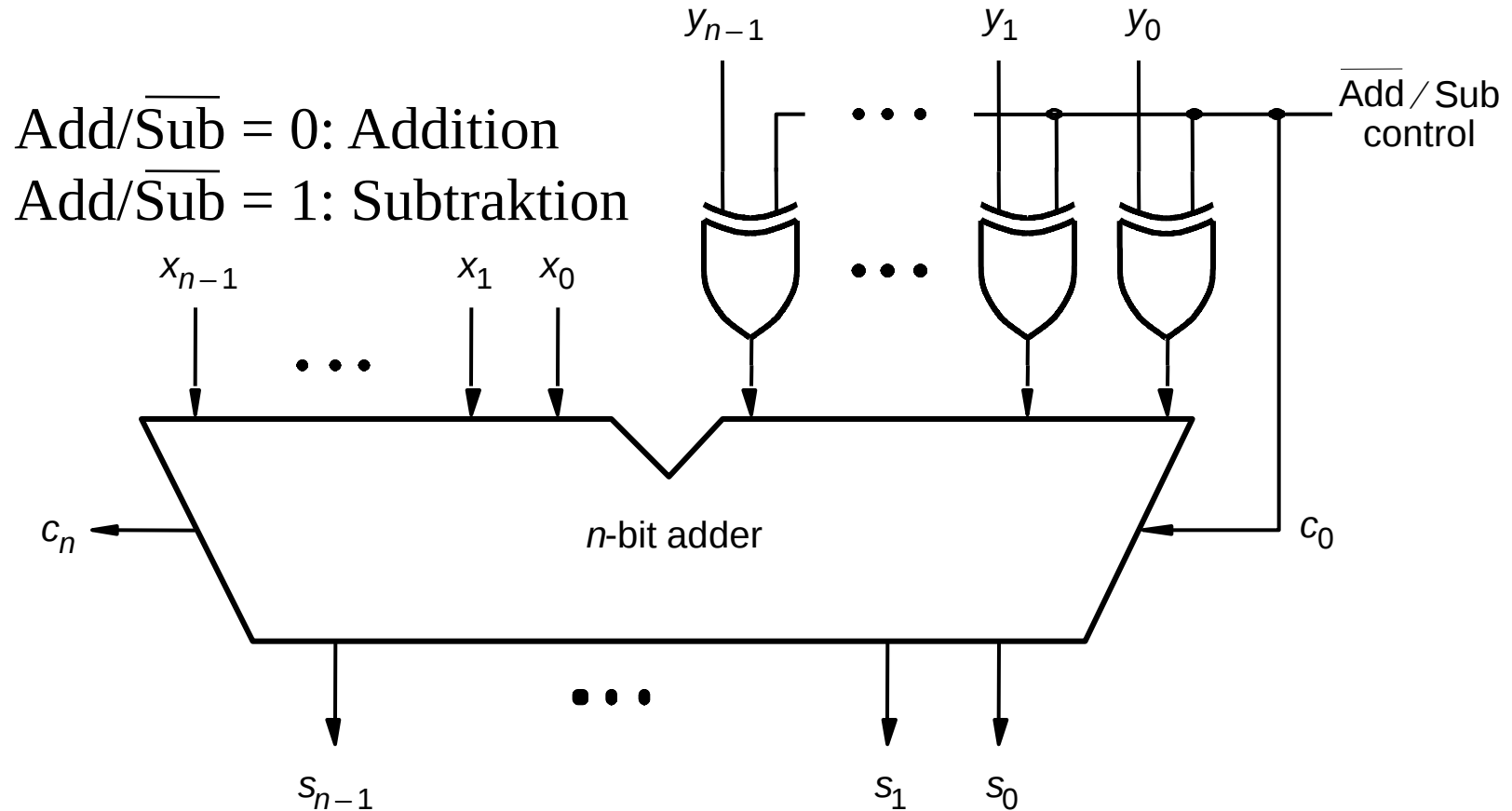
Subtraktion

Subtraktion kan göras genom addition med två komplementet

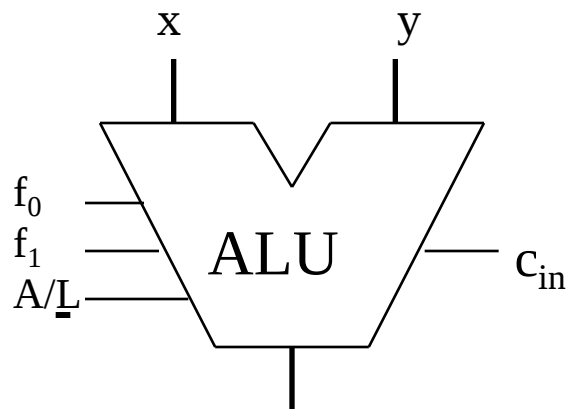
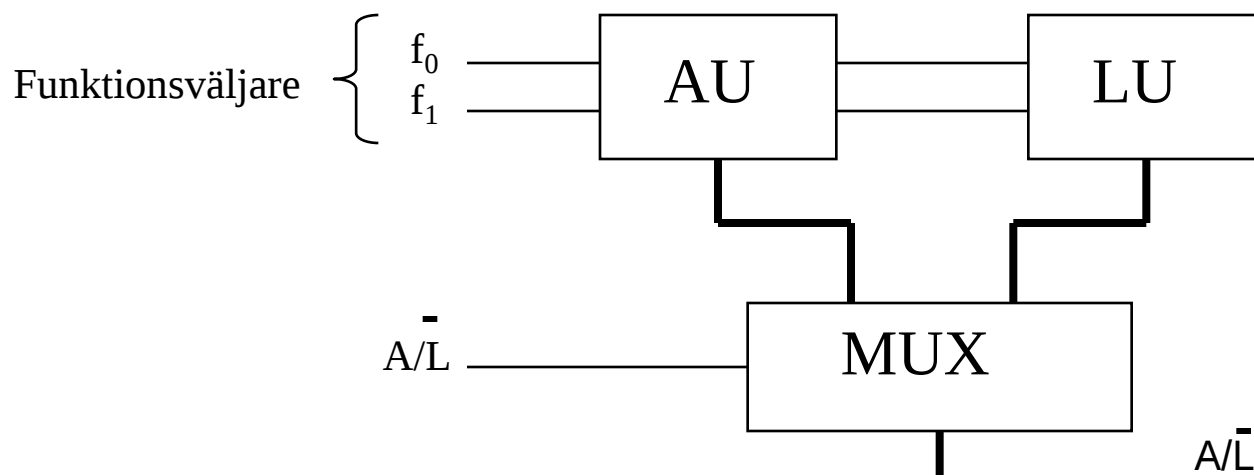
Invertera alla bitar av den andra operanden

Addera 1

Add/sub-enheter



Arithmetic Logic Unit (ALU)

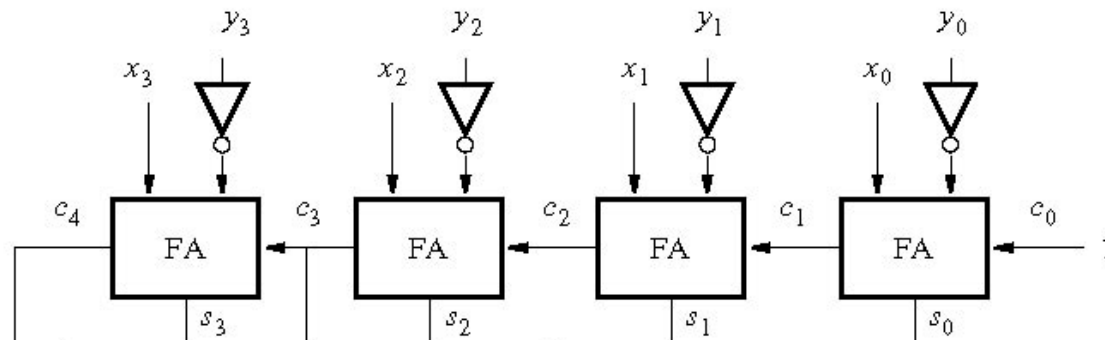


Processorer
brukar alltid ha
en **ALU**, inte
bara en adderare.

$\bar{A/L}$	f ₁	f ₀	Funktion
0	0	0	$x+y$
0	0	1	$x+y+c_{in}$
0	1	0	$x-y$
0	1	1	$x-y-\bar{c}_{in}$
1	0	0	$x \text{ or } y$
1	0	1	$x \text{ and } y$
1	1	0	$x \text{ xor } y$
1	1	1	$\text{inv } x$

Komparator

Komparatorn implementeras som subtraktionskrets



?

V
(overflow)

?

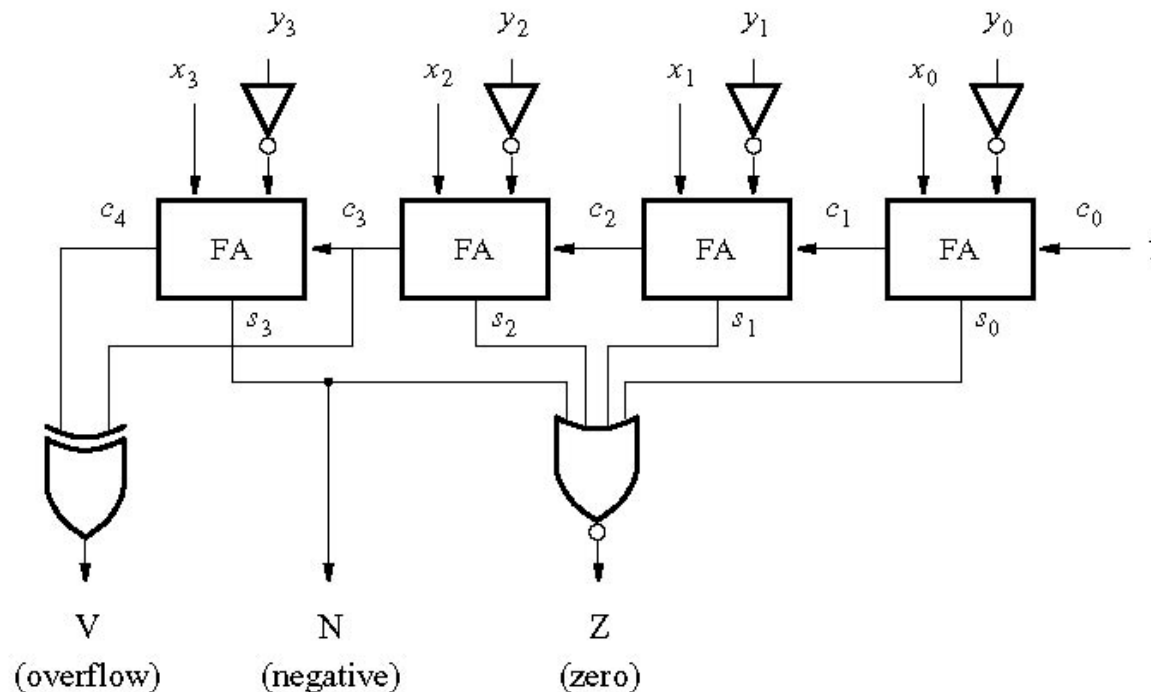
N
(negative)

?

Z
(zero)

Komparator

Komparatorn implementeras som subtraktionskrets



Sammanfattning

Addition och subtraktion av heltal

- Två-komplementet
- Subtraktion av ett tal implementeras som addition med dess två-komplement

Trade-Off: Area mot Speed

Olika Adder-strukturer

- Ripple-Carry Adder (RCA)
- Carry-Lookahead Adder (CLA)
- Carry-Select Adder (CSA)

