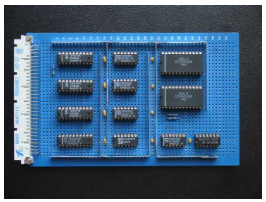
# Digital- och datorteknik



## Föreläsning #8

Biträdande professor Jan Jonsson

Institutionen för data- och informationsteknik
Chalmers tekniska högskola

# Aritmetik i digitala system

## Grindnät för addition:

Vi har sett att man bör kunna bygga en komponent (ett grindnät) för addition av en enskild bit, och sedan seriekoppla (kaskadkoppla) n stycken sådana komponenter för att erhålla en n-bitars adderare.

- Ett grindnät för bitaddition som inte tar hänsyn till inkommande minnessiffra kallas för en halvadderare (”half adder”).
- Ett grindnät för bitaddition som tar hänsyn till inkommande minnessiffra kallas för en heladderare (”full adder”).

| | $c_n$ | $c_{n-1}$ | | $c_{i+1}$ | $c_i$ | | $c_1$ | $c_0$ |
|---|---|---|---|---|---|---|---|---|
| | | $c_{n-1}$ | … | … | $x_i$ | … | $x_1$ | $x_0$ |
| + | | $y_{n-1}$ | … | … | $y_i$ | … | $y_1$ | $y_0$ |
| | | $s_{n-1}$ | … | … | $s_i$ | … | $s_1$ | $s_0$ |

# Aritmetik i digitala system

## Grindnät för addition:

### Härledning av grindnät för en halvadderare:

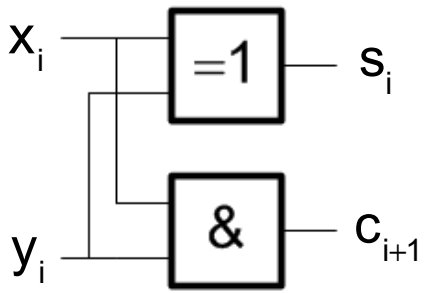
| $x_i$ | $y_i$ | $s_i$ | $c_{i+1}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

AND-funktion

XOR-funktion

$$s_i = x_i \oplus y_i$$

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



# Aritmetik i digitala system

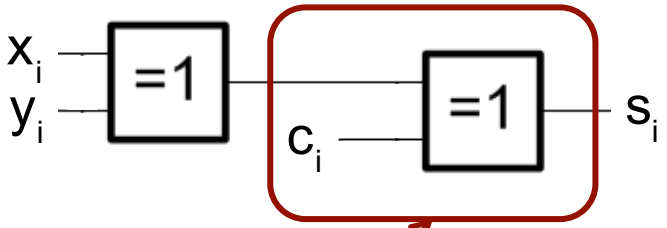
## Grindnät för addition:

### Härledning av grindnät för en heladderare:

Samma som halvadderare för $c_i = 0$

| $c_i$ | $x_i$ | $y_i$ | $s_i$ | $c_{i+1}$ |
|---|---|---|---|---|
| 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 |

$$s_i = c_i \oplus x_i \oplus y_i$$




Villkorlig inverterare

XOR-funktion för $c_i = 0$

NXOR-funktion för $c_i = 1$

# Aritmetik i digitala system

## Grindnät för addition:

### Härledning av grindnät för en heladderare:

Samma som halvadderare för $c_i = 0$

| $c_i$ | $x_i$ | $y_i$ | $s_i$ | $c_{i+1}$ |
|---|---|---|---|---|
| 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 |

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

Men: denna bit blir ’1’ även om vi väljer XOR i stället för OR.

Och vi har ju faktiskt redan denna XOR-funktion i halvadderaren.

# Aritmetik i digitala system

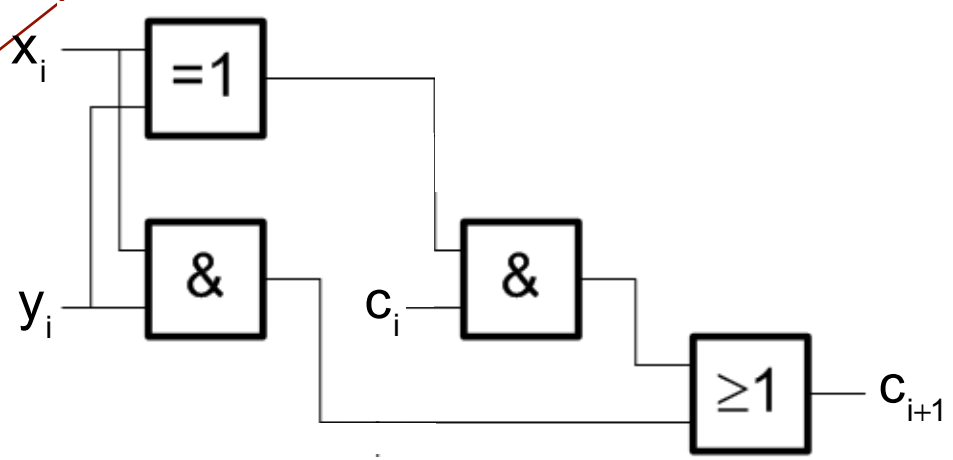
## Grindnät för addition:

### Härledning av grindnät för en heladderare:

Samma som halvadderare för $c_i = 0$

| $c_i$ | $x_i$ | $y_i$ | $s_i$ | $c_{i+1}$ |
|---|---|---|---|---|
| 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 |

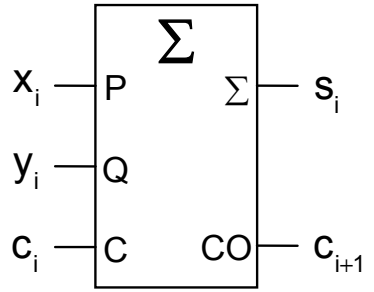
$$c_{i+1} = x_i * y_i + c_i(x_i \oplus y_i)$$



# Aritmetik i digitala system

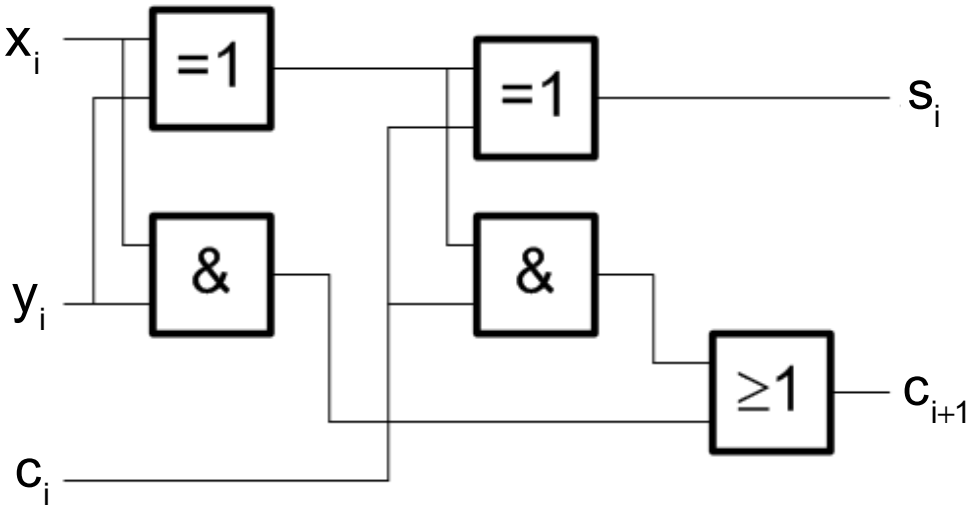
## Grindnät för addition:

### Härledning av grindnät för en heladderare:




| $c_i$ | $x_i$ | $y_i$ | $s_i$ | $c_{i+1}$ |
|---|---|---|---|---|
| 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 |

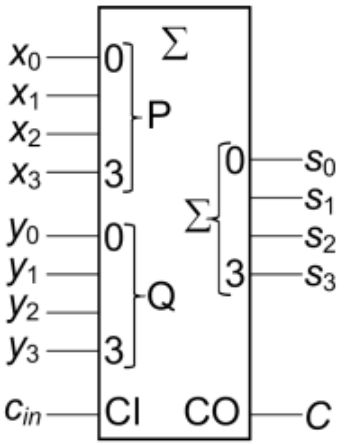
$$s_i = c_i \oplus x_i \oplus y_i \qquad c_{i+1} = x_i * y_i + c_i(x_i \oplus y_i)$$

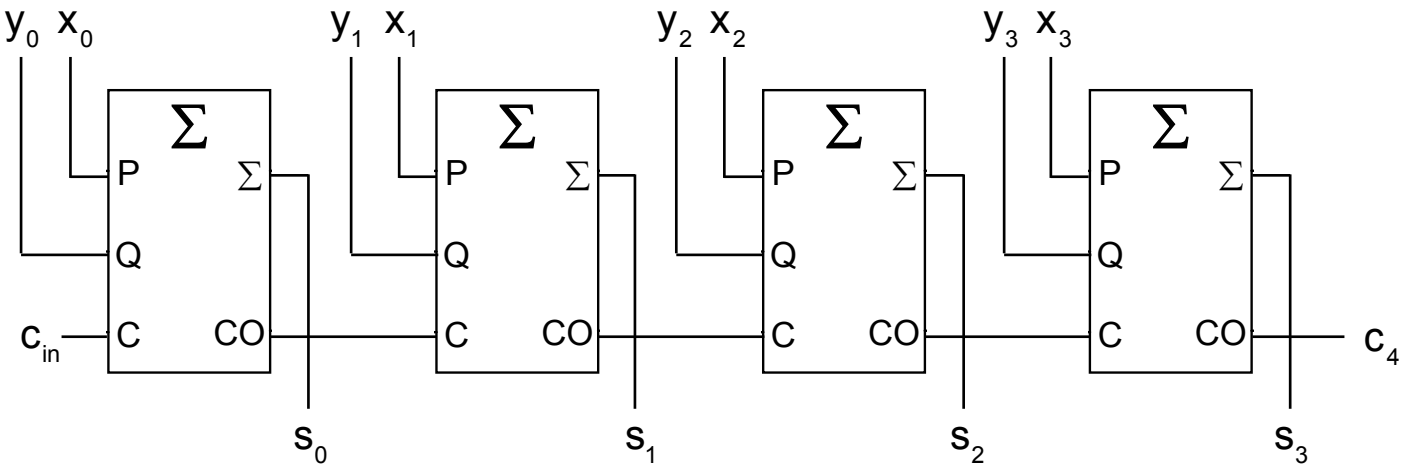


# Aritmetik i digitala system

## Konstruktion av 4-bitars adderare:

Vi kan nu kaskadkoppla fyra heladderare och få en 4-bitars adderare. Flera 4-bitars adderare kan sedan på samma sätt kopplas ihop för att bygga adderare för 8, 16 eller 32 bitars binärtal.

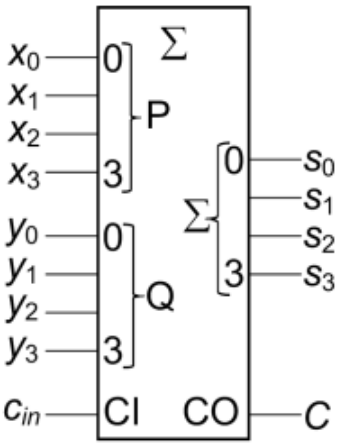


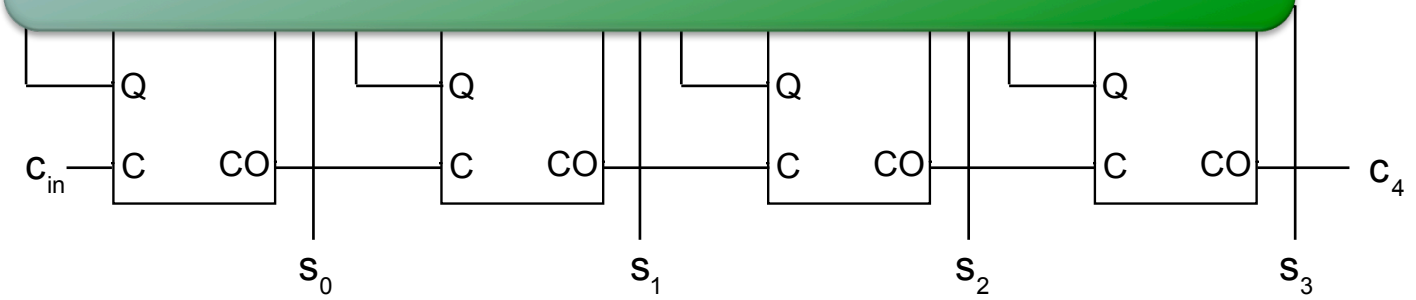
# Aritmetik i digitala system

## Konstruktion av 4-bitars adderare:

Vi kan nu kaskadkoppla fyra heladderare och få en 4-bitars adderare. Flera 4-bitars adderare kan sedan på samma sätt kopplas ihop för att bygga adderare för 8, 16 eller 32 bitars binärtal.




Vad bestämmer löptiden för adderaren?



# Aritmetik i digitala system

## Konstruktion av 4-bitars subtraktionsenhet:

Det vore praktiskt om vår 4-bitar adderare också kunde användas för subtraktion. Och vi har faktiskt redan sett hur detta skulle kunna gå till:

1-komplement

”carry-in”

$$Y_{2\text{-komplement}} = 2^n - Y = 2^n - 1 + 1 - Y = \boxed{(2^n - 1 - Y)} + 1$$

$$X - Y = X + (-Y) = X + Y_{2\text{–komplement}} = X + Y_{1\text{–komplement}} + 1$$

Vi kan alltså erhålla subtraktionen X − Y genom att addera X till de inverterade bitarna i Y (1-komplementet) samt lägga till 1 till summan via adderarens ”carry-in”.

# Aritmetik i digitala system

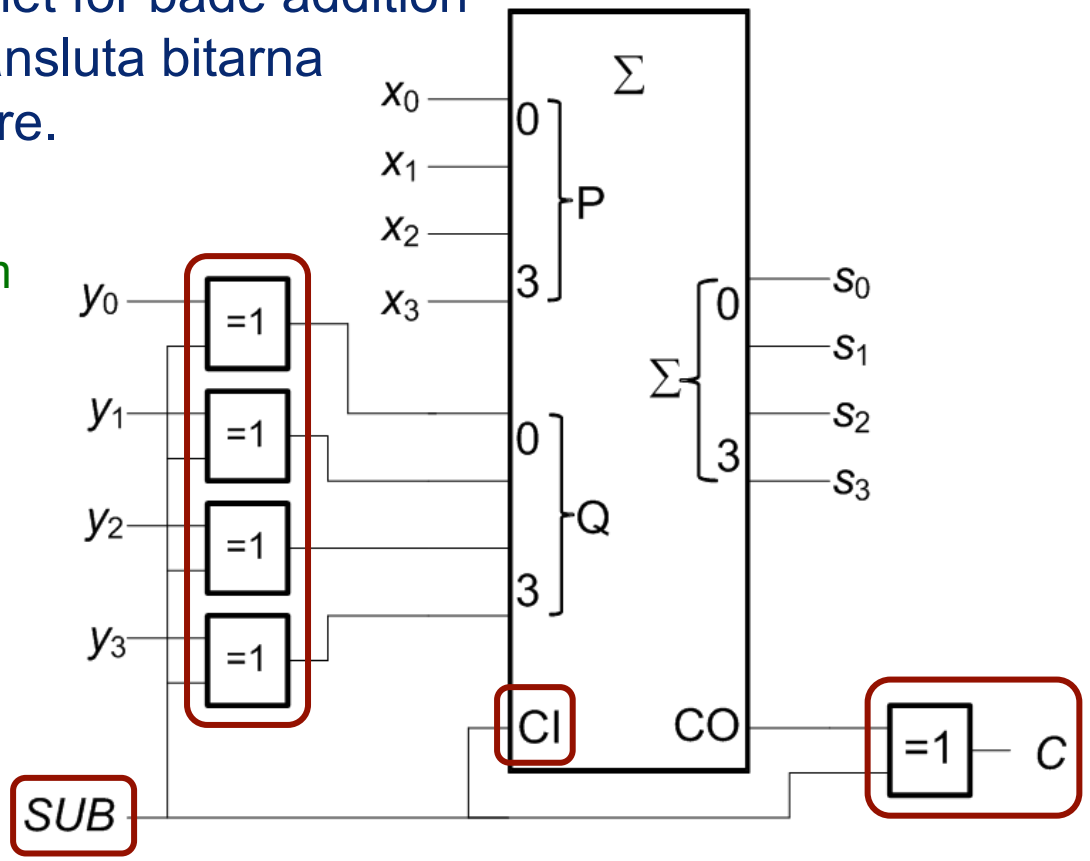
## Konstruktion av 4-bitars addition/subtraktionsenhet:

Vi kan nu konstruera en enhet för både addition och subtraktion genom att ansluta bitarna för Y via villkorliga inverterare.

De villkorliga inverterarna styrs av signalen SUB som väljer om kretsen skall utföra addition (SUB = 0) eller subtraktion (SUB = 1) med Y.

Signalen SUB sätter också rätt värde på ”carry-in”: 0 vid addition och 1 vid subtraktion.

Vi återkommer strax till den villkorliga inverteringen vid C.



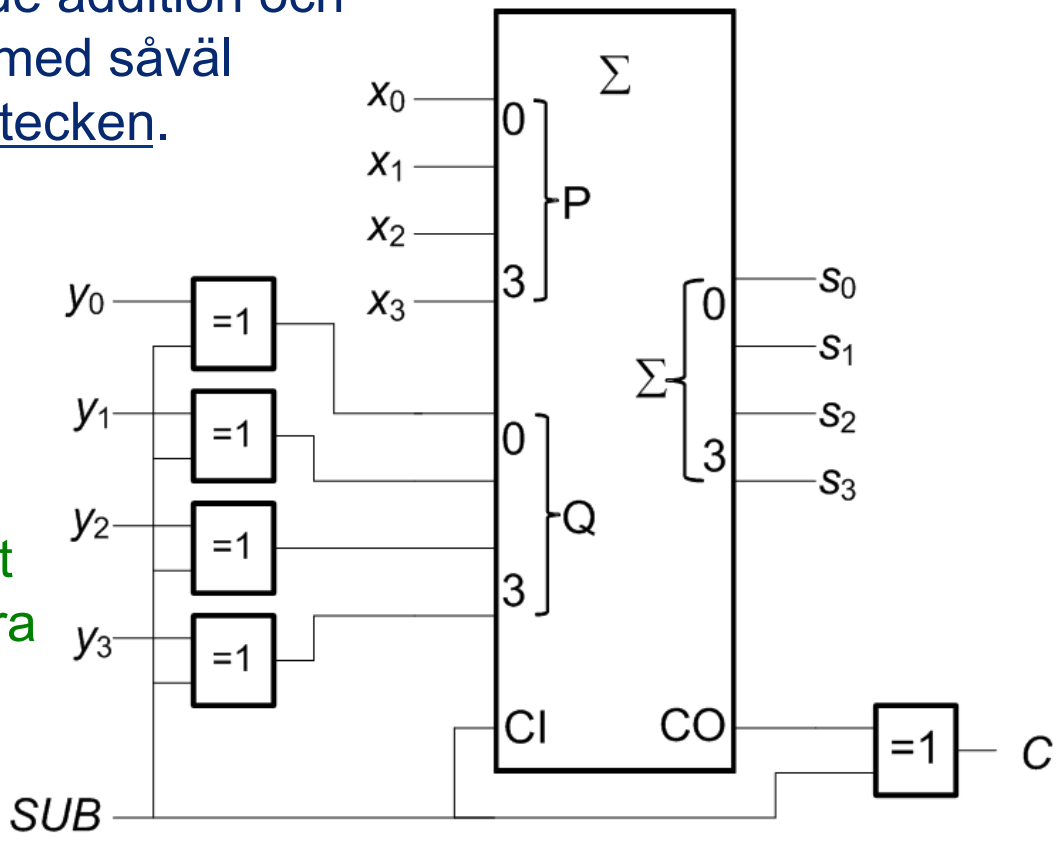
# Aritmetik i digitala system

## Tolkning av resultat

Vi har alltså en enhet för både addition och subtraktion som kan arbeta med såväl tal utan tecken som tal med tecken.

Fråga:
Hur vet enheten vilken typ av tal som används?

Svar:
Det vet den inte! Den måste därför alltid generera resultat som kan användas för endera typen av tal.



# Aritmetik i digitala system

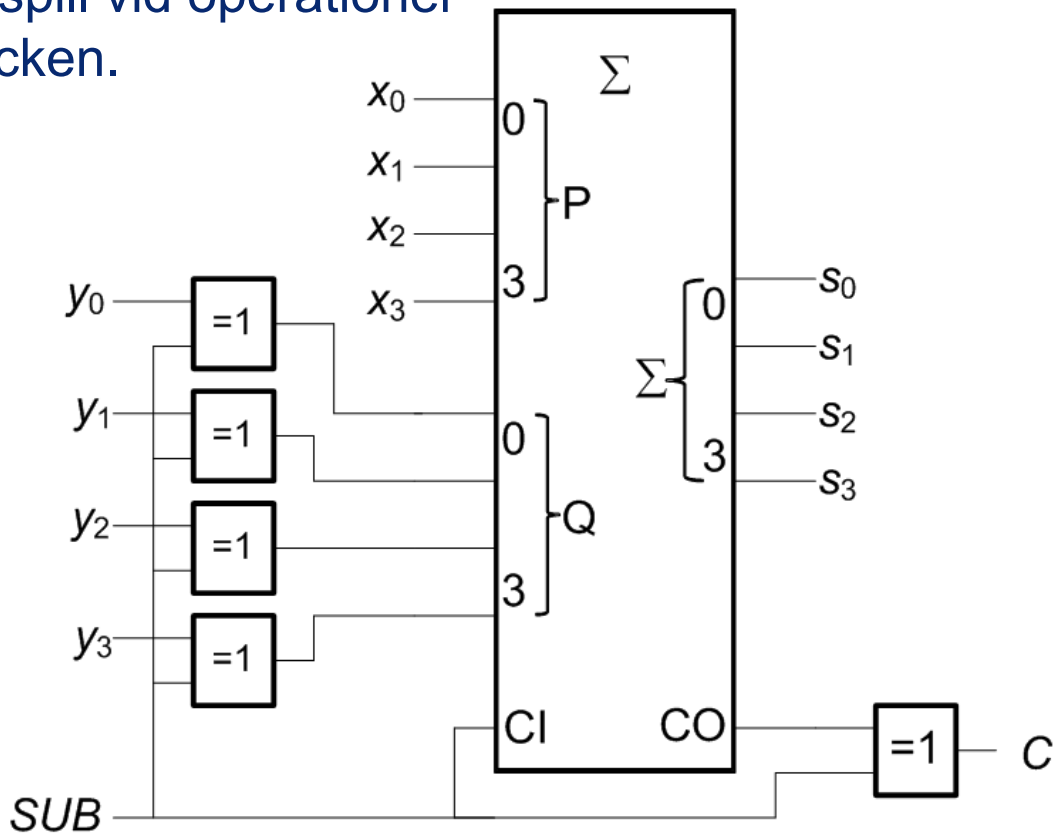
## Tolkning av resultat

Vi har en enhet som kan ge spill vid operationer på tal både med och utan tecken.

Fråga:
Hur vet man om resultatet har gett spill eller ej?

Svar:
Enheten måste generera information om spill för båda typer av tal. Det är sedan upp till användaren att tolka resultaten på rätt sätt.

Denna information om spill finns i s k flaggbitar.



# Aritmetik i digitala system

## Flaggbitar:

Flaggbitar används inte bara för att detektera spill utan också för att indikera andra egenskaper hos ett resultat. Som vi skall se senare kan programvaran i en dator konstrueras så att den vidtar olika åtgärder beroende på vilka egenskaper ett resultat har.

De vanligast förekommande flaggbitarna i ett digitalt system är

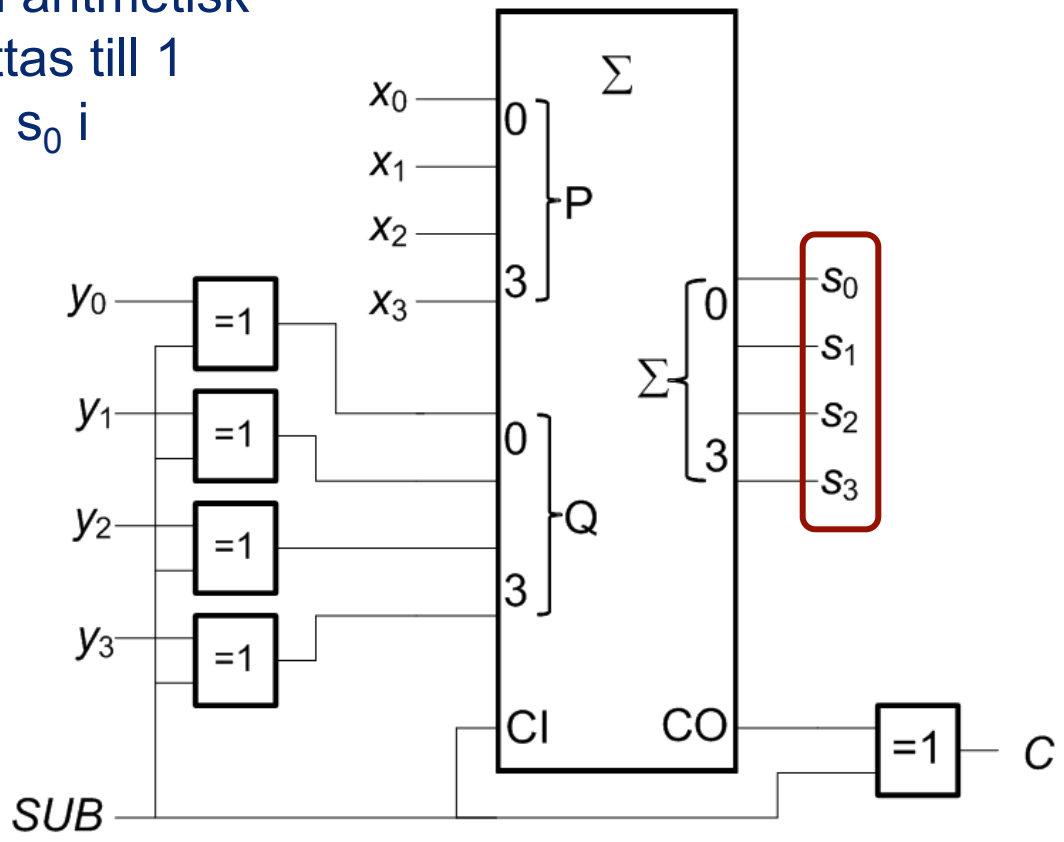
- Z (zero): Z-flaggan indikerar om resultatet av en operation blev 0.
- N (negative): N-flaggan indikerar om resultatet av en operation blev negativt (oftast bara av intresse vid aritmetik på tal med tecken)
- C (carry): C-flaggan indikerar om resultatet av en operation mellan tal utan tecken resulterade i spill.
- V (overflow): V-flaggan indikerar om resultatet av en operation mellan tal med tecken resulterade i spill.

# Aritmetik i digitala system

## Z-flaggan:

Baserat på resultatet från en aritmetisk operation skall Z-flaggan sättas till 1 när alla bitar $s_{n-1}, s_{n-2}, \ldots, s_1, s_0$ i resultatet är lika med 0.

$$Z = \overline{s_{n-1}} * \overline{s_{n-2}} * \ldots * \overline{s_1} * \overline{s_0}$$



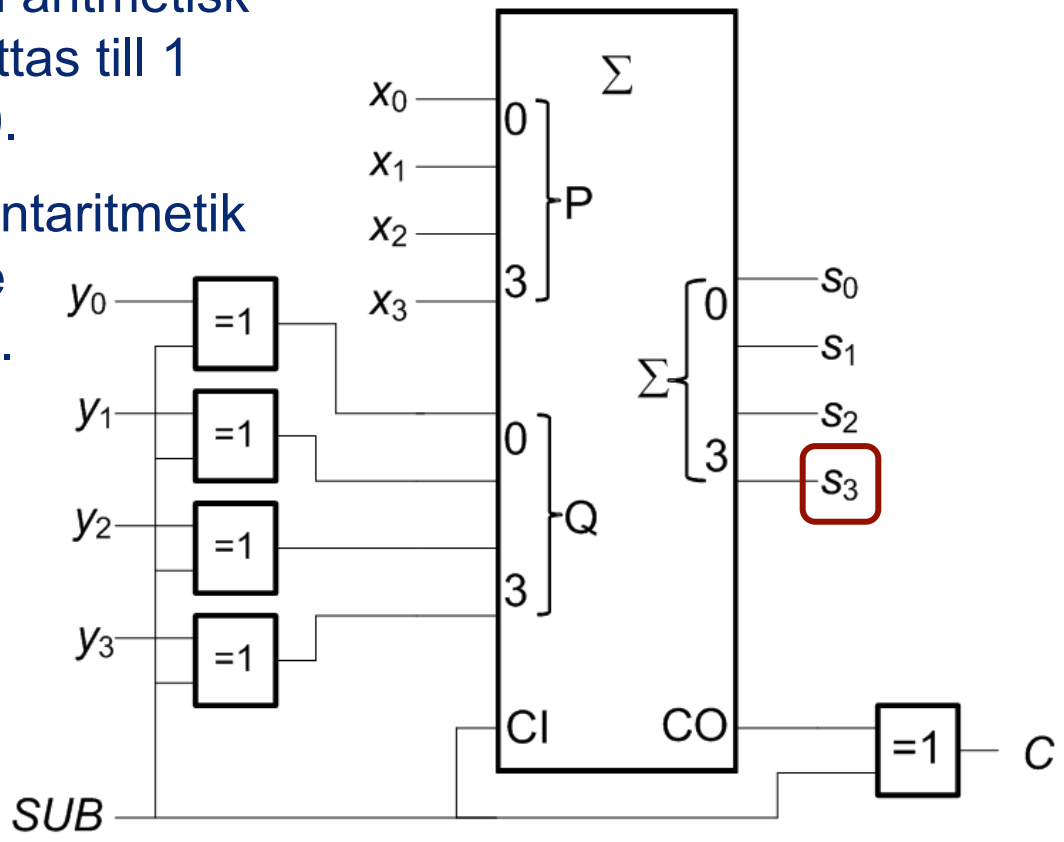
# Aritmetik i digitala system

## N-flaggan:

Baserat på resultatet från en aritmetisk operation skall N-flaggan sättas till 1 när resultatet är mindre än 0.

Då vi använder 2-komplementaritmetik vet vi att resultatet är mindre än 0 när teckenbiten $s_{n-1} = 1$.

$$N = s_{n-1}$$

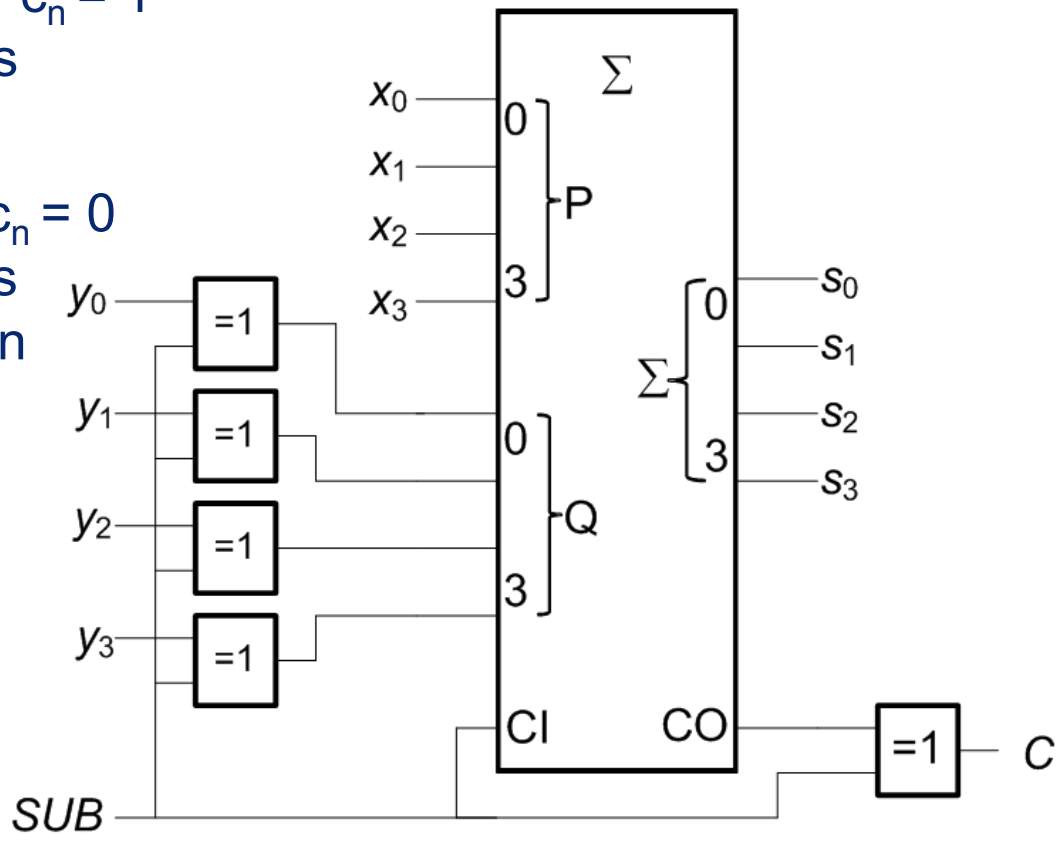


# Aritmetik i digitala system

## C-flaggan:

Vi såg tidigare att ”carry-out” $c_n = 1$ indikerar spill från en n-bitars addition av tal utan tecken.

Vi såg också att ”carry-out” $c_n = 0$ indikerar spill från en n-bitars subtraktion av tal utan tecken när man använder 2-komplementaritmetik.

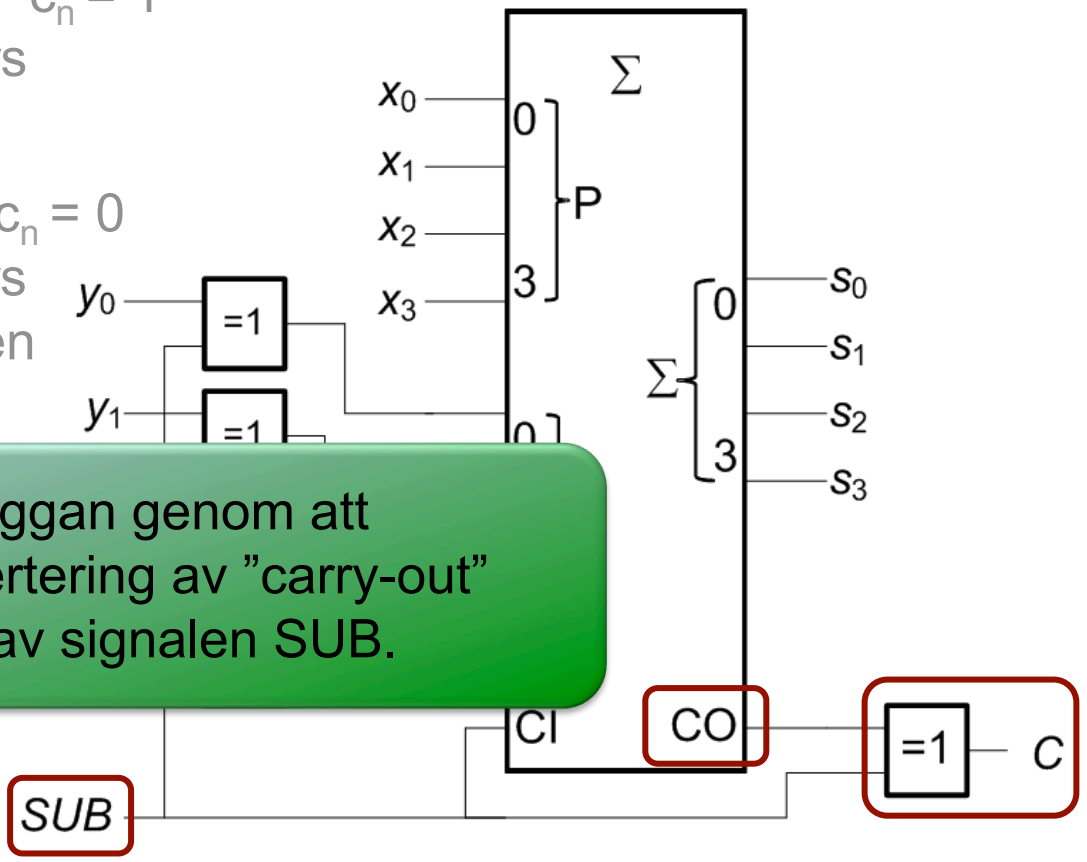


# Aritmetik i digitala system

## C-flaggan:

Vi såg tidigare att ”carry-out” $c_n = 1$ indikerar spill från en n-bitars addition av tal utan tecken.

Vi såg också att ”carry-out” $c_n = 0$ indikerar spill från en n-bitars subtraktion av tal utan tecken när man använder




Vi kan alltså erhålla C-flaggan genom att använda en villkorlig invertering av ”carry-out” $c_n$ från additionen, styrd av signalen SUB.

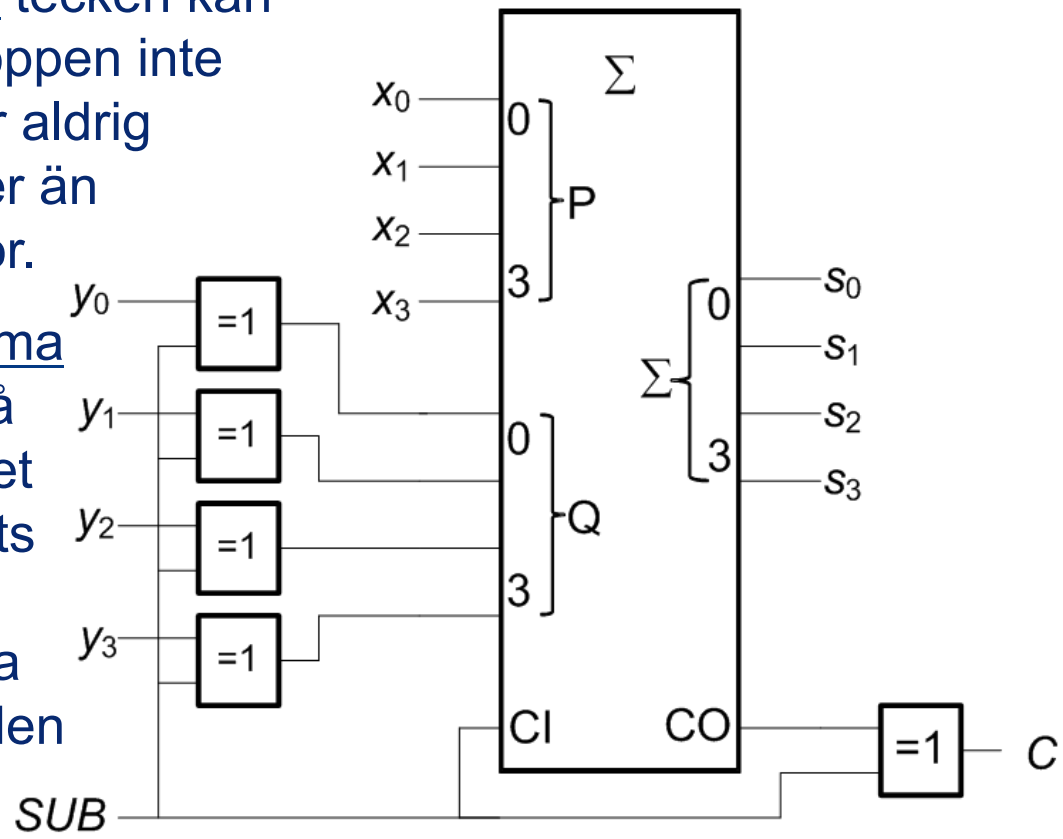
$$C = c_n \oplus SUB$$

# Aritmetik i digitala system

## V-flaggan:

Vid addition av tal med olika tecken kan spill aldrig uppträda, då beloppen inte adderas. Resultatet kommer aldrig närmare talområdets gränser än vad något av de två talen gör.

Vid addition av tal med samma tecken kan spill uppträda, då beloppen adderas. Resultatet kommer närmare talområdets gränser än vad något av de två talen gör och kan komma att ”spilla över” till andra änden av tallinjen.



# Aritmetik i digitala system

## V-flaggan:

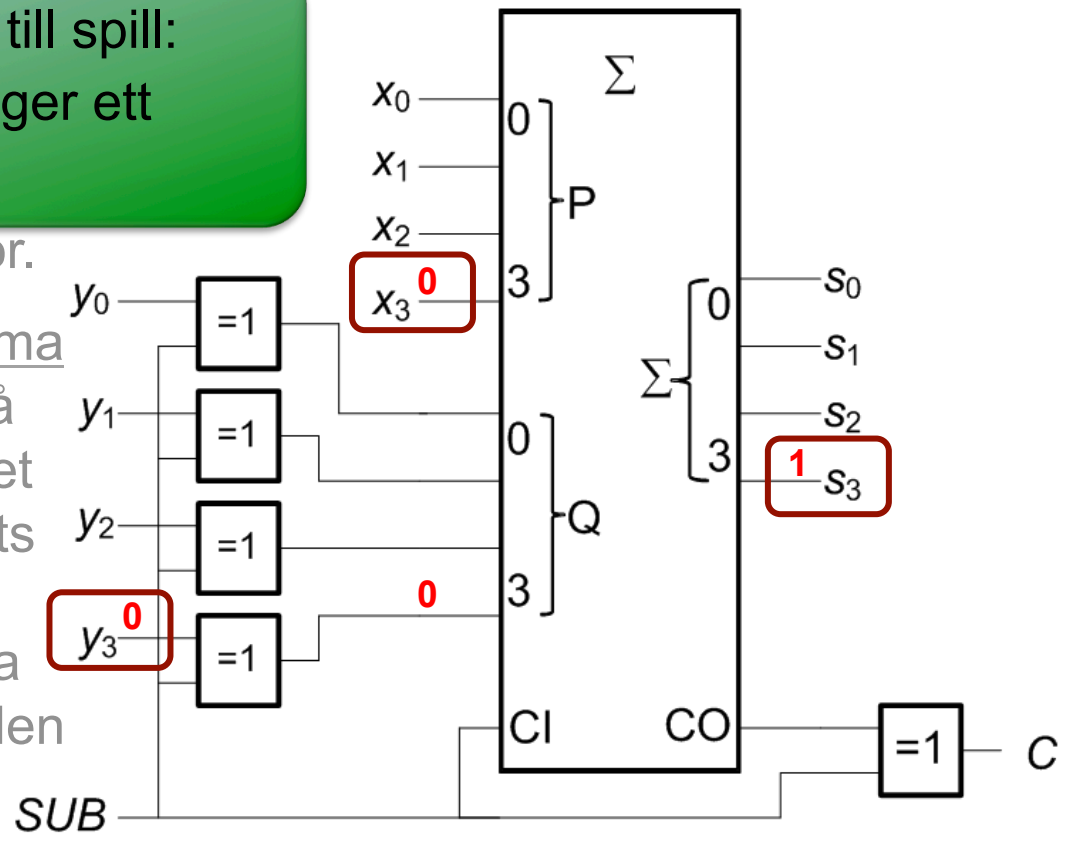
Villkor för spill: $\overline{x_{n-1}} * \overline{y_{n-1}} * s_{n-1}$

Detta ger oss två möjligheter till spill:

1. Addition av två positiva tal ger ett negativt resultat.

vad något av de två talen gör.

Vid addition av tal med samma tecken kan spill uppträda, då beloppen adderas. Resultatet kommer närmare talområdets gränser än vad något av de två talen gör och kan komma att ”spilla över” till andra änden av tallinjen.



# Aritmetik i digitala system

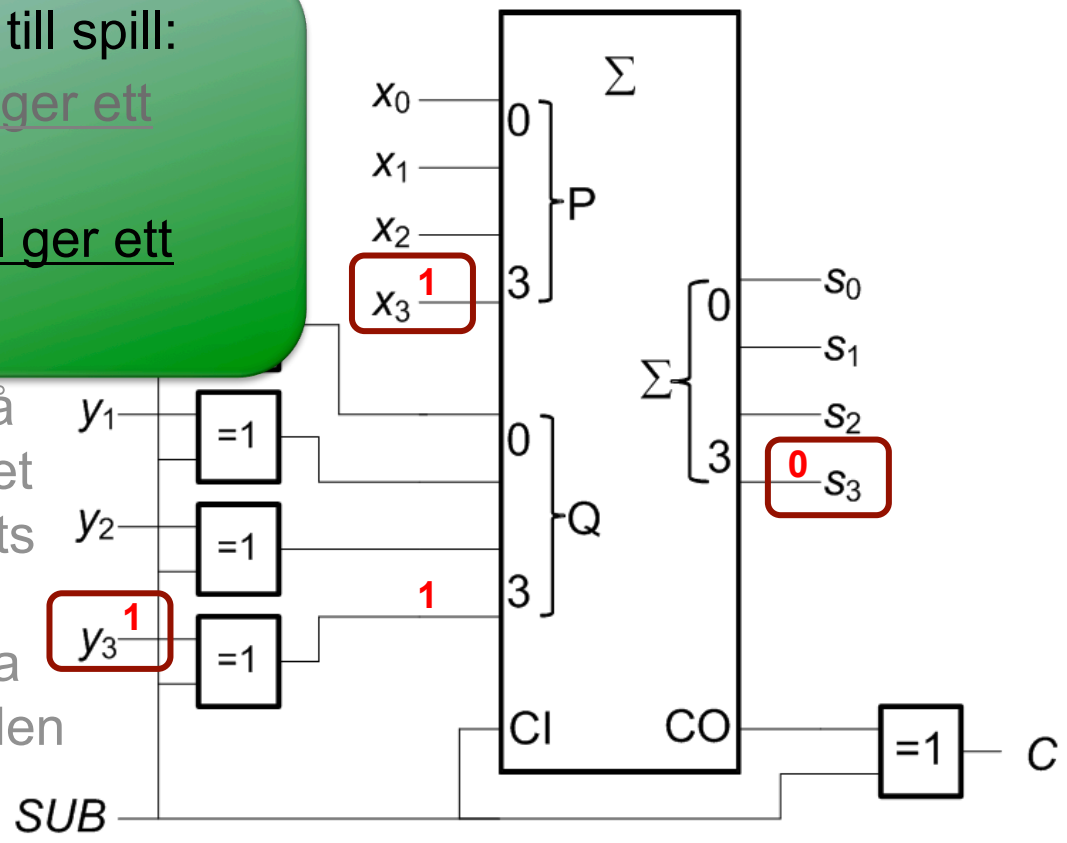
## V-flaggan:

Villkor för spill: $x_{n-1} * y_{n-1} * \overline{s_{n-1}}$

Detta ger oss två möjligheter till spill:

1. Addition av två positiva tal ger ett negativt resultat.
2. Addition av två negativa tal ger ett positivt resultat.

tecken kan spill uppträda, då beloppen adderas. Resultatet kommer närmare talområdets gränser än vad något av de två talen gör och kan komma att ”spilla över” till andra änden av tallinjen.



# Aritmetik i digitala system

## V-flaggan:

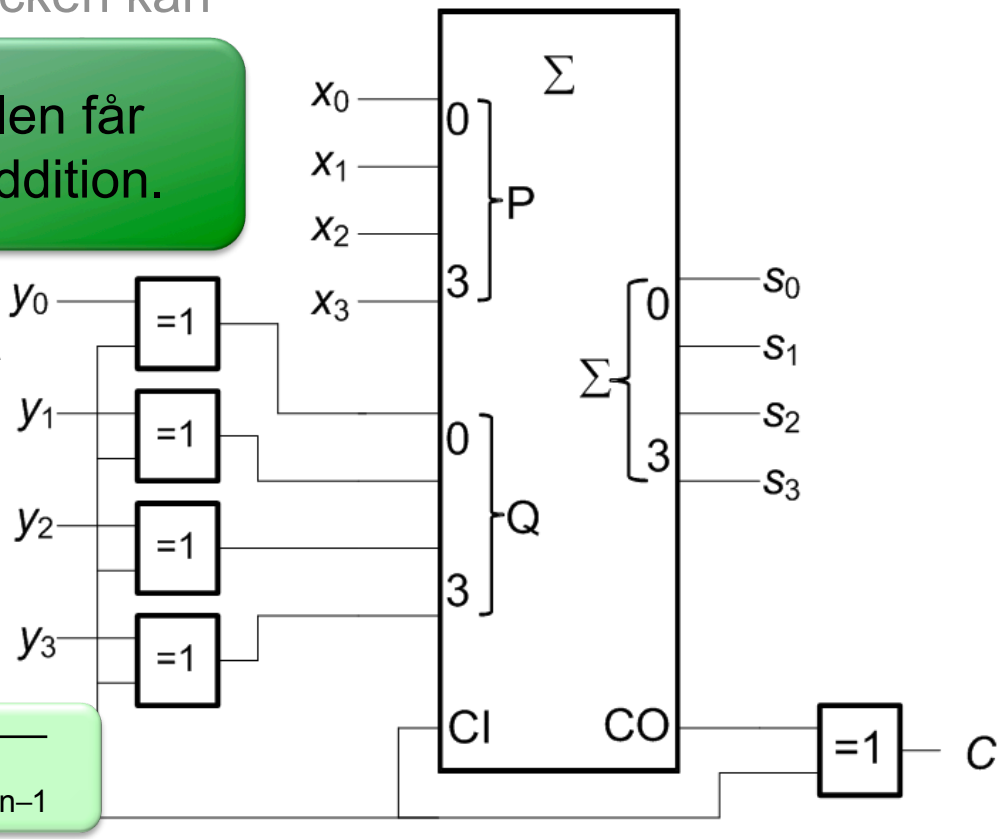
Vid addition av tal med olika tecken kan

Genom att kombinera de två fallen får vi ett slutligt villkor för spill vid addition.

vad något av de två talen gör.

Vid addition av tal med samma tecken kan spill uppträda, då beloppen adderas. Resultatet kommer närmare talområdets gränser än vad något av de två talen gör och kan komma

$$V = \overline{x_{n-1}} * \overline{y_{n-1}} * s_{n-1} + x_{n-1} * y_{n-1} * \overline{s_{n-1}}$$

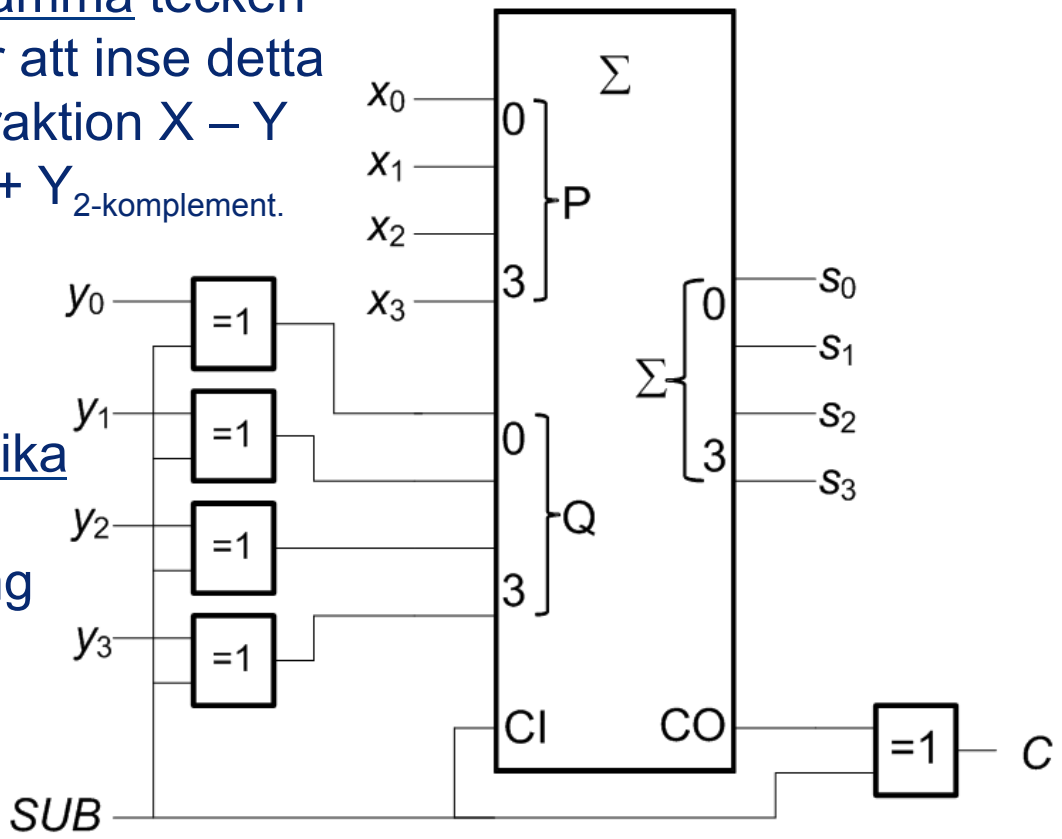


# Aritmetik i digitala system

## V-flaggan:

Vid subtraktion av tal med samma tecken kan spill aldrig uppträda. För att inse detta påminner vi oss om att subtraktion X – Y utförs medelst additionen X + $Y_{2\text{-komplement}}$. Då måste de två talen som adderas ha olika tecken, vilket vi vet inte kan ge spill.

Vid subtraktion av tal med olika tecken kan spill uppträda. Med ett liknande resonemang inser vi att de två talen som adderas har samma tecken, vilket vi vet kan orsaka spill.



# Aritmetik i digitala system

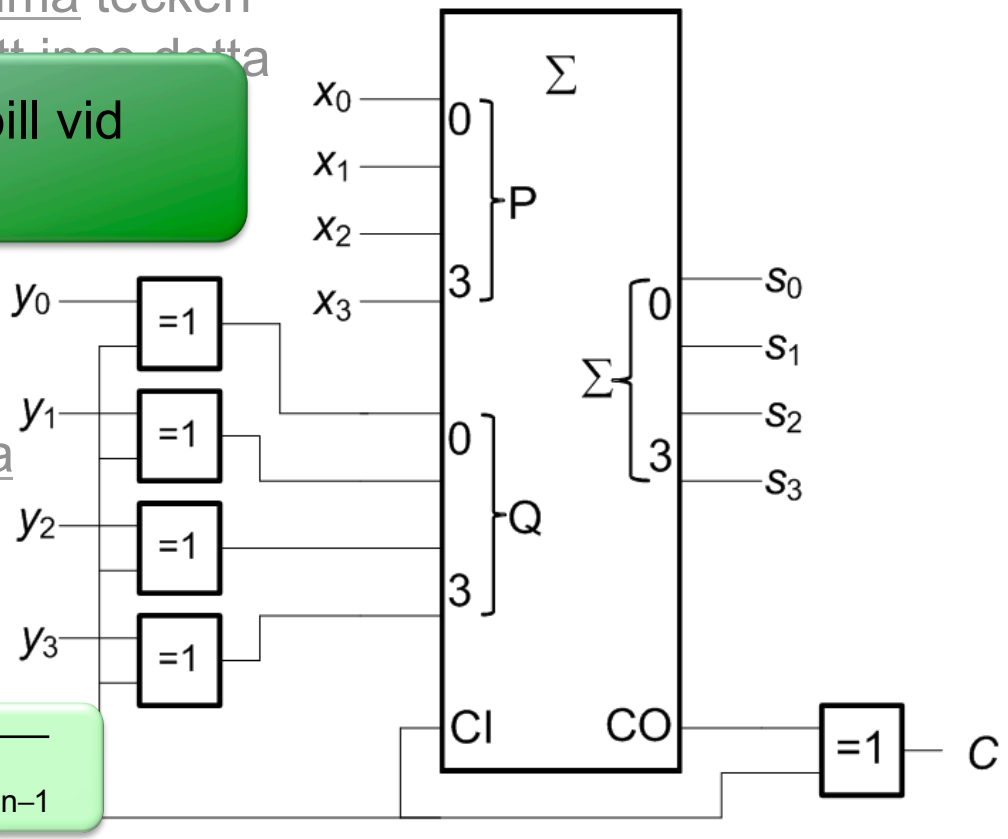
## V-flaggan:

Vid subtraktion av tal med samma tecken kan spill aldrig uppträda. För att inse detta

**Vi får alltså samma villkor för spill vid subtraktion som vid addition.**

Då måste de två talen som adderas ha olika tecken, vilket vi vet inte kan ge spill.

Vid subtraktion av tal med olika tecken kan spill uppträda. Med ett liknande resonemang inser vi att de två talen som adderas har samma tecken.




$$V = \overline{x_{n-1}} * \overline{y_{n-1}} * s_{n-1} + x_{n-1} * y_{n-1} * \overline{s_{n-1}}$$

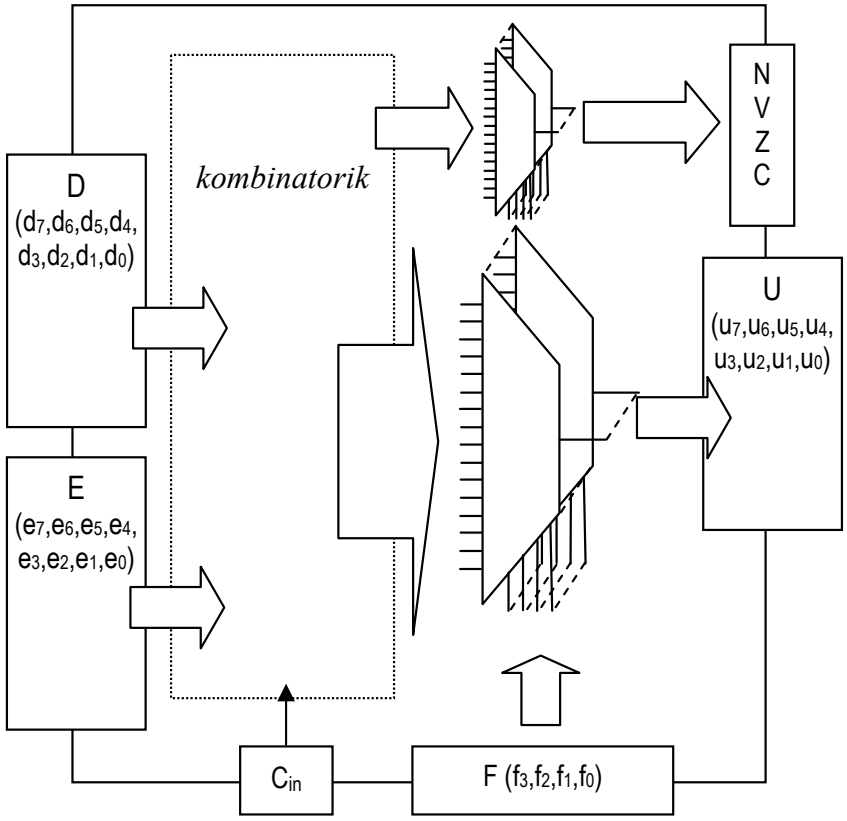
# Aritmetik-/logikenhet (ALU)

## Ett nytt kombinatoriskt nät:

Vår adderare kan nu, tillsammans med en uppsättning logikgrindar, användas för att bygga upp en aritmetik- och logikenhet (ALU).

Detta nya kombinatoriska nät är nödvändigt för att kunna utföra beräkningar i datorn.

Gränssnittet mot grindnätet utgörs av ett antal operationssignaler (F), ett par dataingångar (D och E), en datautgång (U) samt fyra flaggbitar (Z, N, C och V).



# Aritmetik-/logikenhet (ALU)

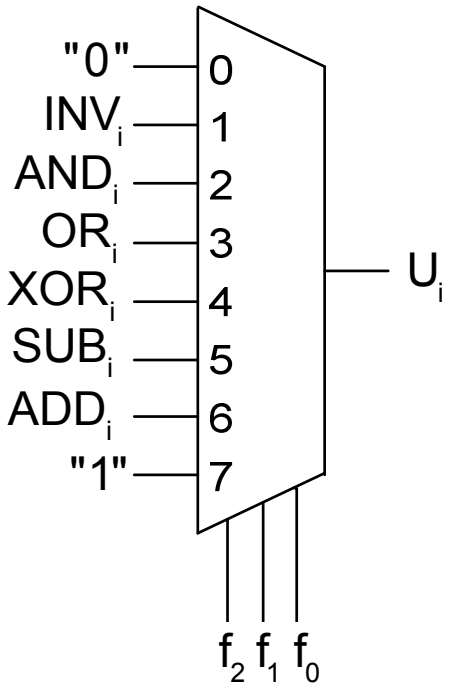
## ALU ”bitslice”:

Varje bit $U_i$ på datautgången ansluts till en väljare (”multiplexer”) som, styrd av operationssignalerna F, levererar en bit av resultatet (en s k ”bitslice”) från den valda operationen.

Exempel på operationer i en ALU:

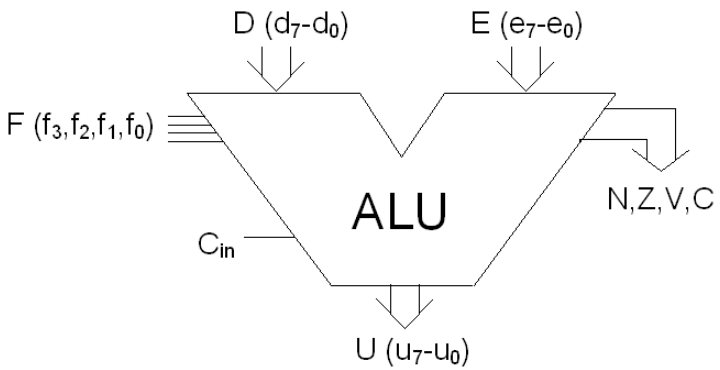
- addition och subtraktion
- bitvis XOR
- bitvis OR
- bitvis AND
- bitvis invertering
- bitkonstanter ”0” och ”1”

Exempel: Väljare för ”bitslice” med åtta operationer.



# Aritmetik-/logikenhet (ALU)

## ALU för laboration 1:




Den ALU ni möter i laboration 1 arbetar med 8-bitars maskintal, och innehåller de operationer vi visade för vår enkla ”bitslice”, samt ytterligare ett par varianter av addition.

| $f_3$ $f_2$ $f_1$ $f_0$ | U = f(D,E,F,$C_{in}$) | |
|---|---|---|
| | **Operation** | **Resultat** |
| 0 0 0 0 | Bitvis nollställning | 0 |
| 0 0 0 1 | | D |
| 0 0 1 0 | | E |
| 0 0 1 1 | Bitvis invertering | $D_{1k}$ |
| 0 1 0 0 | Bitvis invertering | $E_{1k}$ |
| 0 1 0 1 | Bitvis OR | D OR E |
| 0 1 1 0 | Bitvis AND | D AND E |
| 0 1 1 1 | Bitvis XOR | D XOR E |
| 1 0 0 0 | D + 0 + $C_{in}$ | D + $C_{in}$ |
| 1 0 0 1 | D + FFH + $C_{in}$ | D – 1 + $C_{in}$ |
| 1 0 1 0 | D + E + $C_{in}$ | D + E + $C_{in}$ |
| 1 0 1 1 | D + D + $C_{in}$ | 2D + $C_{in}$ |
| 1 1 0 0 | D +$E_{1k}$ + $C_{in}$ | D – E – 1 + $C_{in}$ |
| 1 1 0 1 | Bitvis nollställning | 0 |
| 1 1 1 0 | Bitvis nollställning | 0 |
| 1 1 1 1 | Bitvis ettställning | FFH |