F9: Aritmetiska operationer i hårdvara

- **Innehåll:**
  - **Binär addition**
  - **Parallella adderare**
  - **Representation av negativa tal**
  - **Subtraktion av binära tal**
  - **Multiplikation**

Binär addition

- **En-bits addition**
  - **Binär addition där + betyder aritmetisk addition**

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 → minnessiffra = 1

sanningstabell

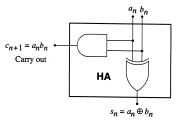
a_n	b_n	S_n	C_{n+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

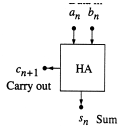
  - **Logiska uttryck för *s* och *c***

$$s_n = a_n \oplus b_n$$

$$c_{n+1} = a_n \cdot b_n$$

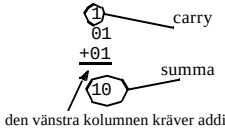
  - **Logikschema och symbol för halvadderare**





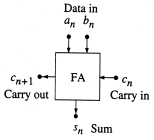
Binär addition av binära tal

- **Exempel: addition av två 2-bitars tal**



den vänstra kolumnen kräver addition av 3 bitar

- **Full-adderare**



a_n	b_n	c_n	s_n	c_{n+1}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$s_n = a_n \oplus b_n \oplus c_n$$

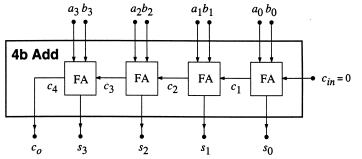
$$c_{n+1} = a_n b_n + c_n(a_n \oplus b_n)$$

Parallella adderare

- **Exempel: addition av två 4-bitars tal**

```
 0100   <-- c
  0101  <-- a
+ 0110  <-- b
-------
 01011  <-- s
```

- **Struktur för parallell adderare -- carry ripple adderare**



c_4	c_3	c_2	c_1	0	**c**
	a_3	a_2	a_1	a_0	**a**
	$+b_3$	$+b_2$	$+b_1$	$+b_0$	**b**

Representation av negativa tal

- **Teckenbit (signed-magnitude)**

sign-bit	b_7	b_6	b_5	b_4	. . .	b_1	b_0

En extra bit (sign-bit) införs för att representera vilket tecken talet har.

Positivt tal: sign-bit = 0

Negativt tal: sign-bit = 1

n-bit tal kan representera värden mellan $-(2^{n-1}-1)$ och $+(2^{n-1}-1)$ med två sätt att representera 0.

- **Exempel**

$01011101_2 = 93_{10}$ $\quad$ $11011101_2 = -93_{10}$

$00000000_2 = +0_{10}$ $\quad$ $10000000_2 = -0_{10}$

Svårt att implementera logiska kretsar som använder teckenbit.

Två-komplement representation (two's-Complement)

- **MSB fungerar som teckenbit - talet är negativt endast om MSB=1**

Till skillnad från ett tal utan tecken har MSB värdet -2^{n-1} (istället för $+2^{n-1}$).
Tal inom området $-(2^{n-1})$ till $+(2^{n-1}-1)$ kan representeras.

Byt tecken på ett tvåkomplement tal:

- Utför bitvis invertering
- Addera 1

Exempel (8 bitars tal):

$27_{10} = 00011011_2$

invertera → 11100100

+1

$-27_{10} = 11100101_2$

$-27_{10} = 11100101_2$

invertera → 00011010

+1

$27_{10} = 00011011_2$

$0_{10} = 00000000_2$

invertera → 11111111

+1

$0_{10} = 00000000_2$

$-128_{10} = 10000000_2$

01111111

+1

$-128_{10} = 10000000_2$

Addition och subtraktion av tal i två-komplement

Med tal i två-komplementform sker addition och subtraktion av positiva och negativa tal på samma sätt (i motsats till tal med teckenbit).

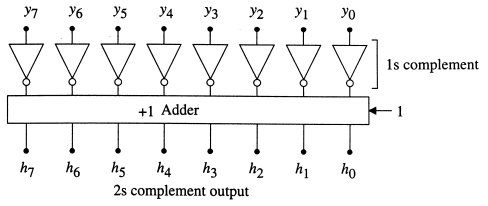
Exempel:

```
+2    0010        +3    0011
+2    0010        -4    1100
+4    0100        -1    1111

+7    0111        -3    1101
-2    1110        -4    1100
+5   10101        -7   11001
```

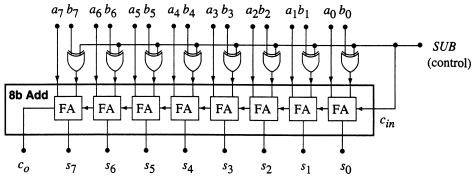

Kretsar för två-komplement

- **Skapa två-komplement**
  - **två_komplement(Y) = ett_komplement(Y) + 1**
  - **ett_komplement(Y) = $\overline{Y}$ (m.a.o invertering)**
- **Kretslösning**



Kombinerad adderare/subtraherare

- **Kretslösning**



då *SUB*=1 inverteras ***b*** och 1 adderas. Detta är samma som att ta 2-komplement av *b*.

- **XOR-grinden som "kontrollerbar" inverterare**

SUB	b_n	$f = SUB \oplus b_n$	
0	0	0	$= b_n$
0	1	1	
1	0	1	$= \overline{b}_n$
1	1	0	

då *SUB*=0 blir $f = b_n$

då *SUB*=1 blir $f = \overline{b}_n$

Multiplikation

- **Binär multiplikation med följande regler och implementation**

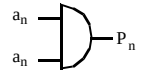
multiplikation

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

sanningstabell

a_n	b_n	P_n
0	0	0
0	1	0
1	0	0
1	1	1

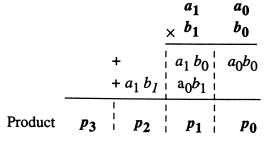
AND-grind



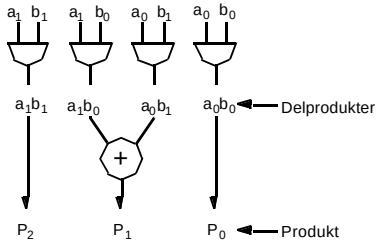
Principen för multiplikation av binära tal

- **Exempel: 2 x 2 bitars multiplikator**

Algoritm

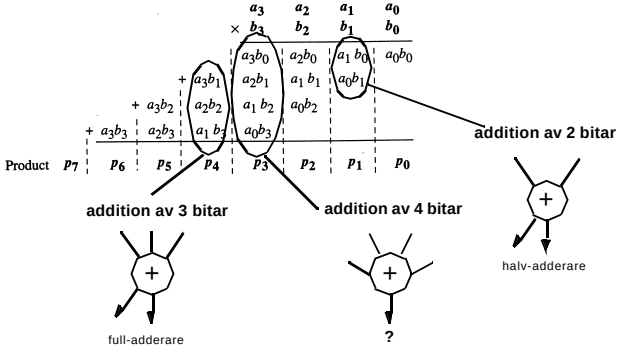


Hårdvaruarkitektur



Multiplikation av tal med många bitar

- **Exempel: 4 x 4 bitars multiplikation**

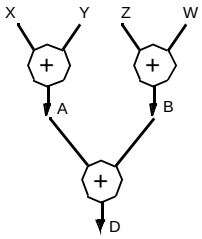


Addition av flera binära tal

- **Metod 1: *Vector merge***

Om ett stort antal binära tal ska adderas ihop så adderas talen ihop två och två i delsummor som i sin tur adderas samman tills det att en enda summa har erhållits.För att summera ihop två tal kan t.ex en *ripple carry adderare* användas.

Example: summering av fyra tal



Vektorna (binära tal) X, Y, Z, W ska summeras.

X och Y adderas parallellt med additionen av Z och W och delsummorna A och B erhålls.

Delsummorna A och B summeras och den sluliga summan D bildas (D = A + B = X + Y + Z+ W).

Exempel: *vector merge*

- **Addera talen X, Y, Z och W**

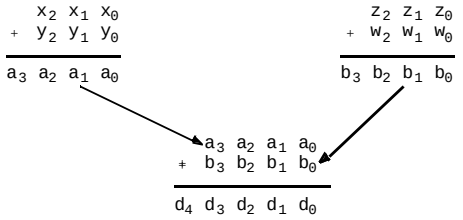
$X = (x_2, x_1, x_0)$; $Y = (y_2, y_1, y_0)$; $Z = (z_2, z_1, z_0)$; $W = (w_2, w_1, w_0)$

- **Delsummor**

$A = (a_3, a_2, a_1, a_0)$; $B = (b_3, b_2, b_1, b_0)$

- **Slutsumma**

$D = (d_4, d_3, d_2, d_1, d_0)$



Addition av flera binära tal

- **Metod 2: *Carry-Save Addition***

Till skillnad från metod 1 så bildas inte fullständiga delsummor. Istället bildar en addition en summa-vektor och en carry-vektor som inte förrän i sista steget adderas och bildar den slutliga summan.

1. Carry-save addition

$$\begin{array}{rrrr} & x_2 & x_1 & x_0 \\ & y_2 & y_1 & y_0 \\ + & z_2 & z_1 & z_0 \\ \hline & a_2 & a_1 & a_0 \\ & b_2 & b_1 & b_0 \end{array}$$

summa vektor
carry vektor

2. Carry-save addition

$$\begin{array}{rrrrr} & 0 & a_2 & a_1 & a_0 \\ & 0 & w_2 & w_1 & w_0 \\ + & b_2 & b_1 & b_0 & 0 \\ \hline & d_3 & d_2 & d_1 & d_0 \\ & 0 & e_2 & e_1 & e_0 \end{array}$$

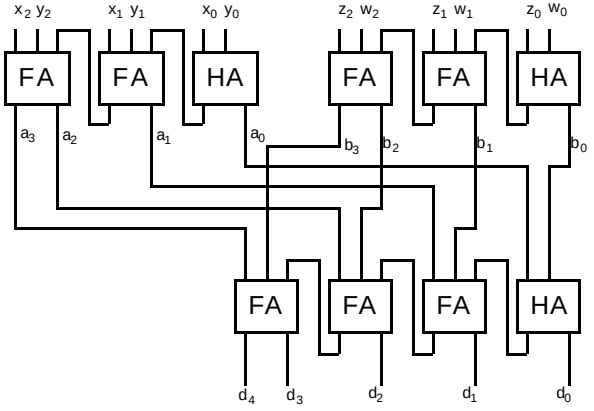
summa vektor
carry vektor

3. *Vector merge* addition (t.ex carry ripple adderare)

$$\begin{array}{rrrrrr} & 0 & d_3 & d_2 & d_1 & d_0 \\ + & 0 & e_2 & e_1 & e_0 & 0 \\ \hline & f_4 & f_3 & f_2 & f_1 & f_0 \end{array}$$

slutsumma

Implementering av *Vector merge* addition



Implementering av Carry-save addition