

Examples on negation

Represent -35 in 2's notation
using 8 bits.

Start with $+35$

$$+35 = 32 + 2 + 1$$

00100011

↳ sign bit.

↓ complement all bits

11011100

+1

$$-35 = \overline{11011101} \longrightarrow \text{Correct result.}$$

0 ← 0 ←

$$C_n = C_{n+1}$$

no OVF

Alternative approach

$a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$

↳ sign bit

$$\text{weight} = -2^7 = -128$$

$$-35 = -128 + x$$

↳ ?

$$\begin{aligned}x &= 128 - 35 \\ &= 93\end{aligned}$$

$$\begin{aligned}-35 &= -128 + 93 \\ &= -128 + 64 + 16 + 8 + 4 + 1\end{aligned}$$

$$\underline{-35 : \quad 11011101}$$

Addition notes

$s_n = a_n + b_n - C_n$ is the true approach

If $s_n = -1$ or 2 , result is OVF. However, it turns out

that if we use $s_n = a_n + b_n + C_n$ & generate a carry C_{n+1} , the following is true.

(a) Whenever $a_n + b_n - C_n = 0$ or 1 , so is $a_n + b_n + C_n$. And $C_n = C_{n+1}$

(b) Whenever $a_n + b_n - C_n = -1$ or 2 , $C_n \neq C_{n+1}$

Conclusion:

We can add numbers in 2's complement notation just like we add in unsigned notation. The detection of overflow is different.

If carry into sign position equals carry out of sign position, there is no OVF.

Else, there is an OVF.

Examples: $8 + 17$ in 2's notation using 6 bits.

$$A = +8: \quad 001000$$

$$B = +17: \quad 010001$$

$$\begin{array}{r} 001000 \\ + 010001 \\ \hline 011001 \\ \hline \end{array} = 25$$

$\leftarrow \leftarrow$
0 0

$$\textcircled{4} \quad \left. \begin{array}{l} A = -64 \\ B = -49 \end{array} \right\} 8 \text{ bits}$$

$$A = -128 + 64$$

$$11000000$$

$$B = -49 = -128 + 64 + 8 + 4 + 2 + 1$$

$$= 11001111$$

$$A = 11000000$$

$$B = 11001111$$

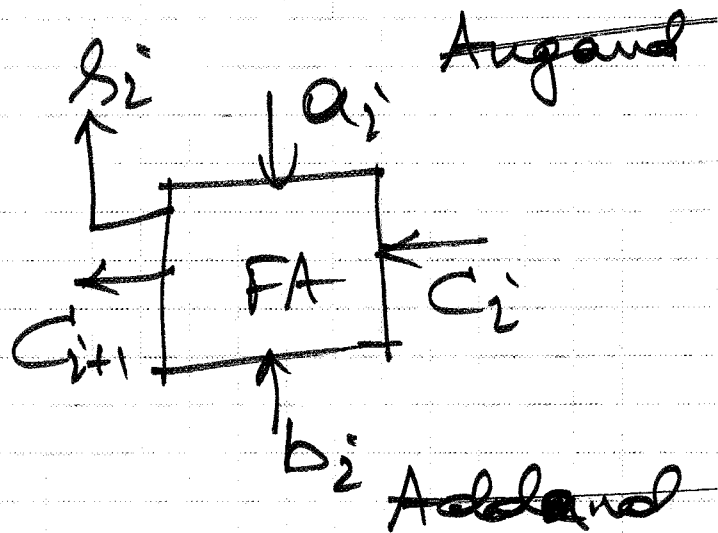
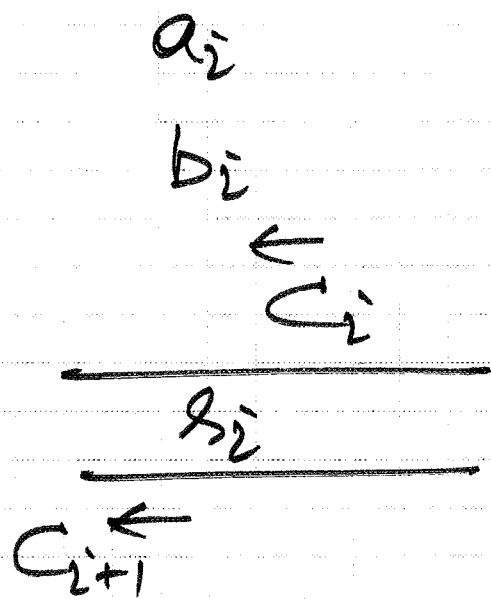
$$\hline 10001111 = -128 + 8 + 4 + 2 + 1$$

$$\begin{array}{c} \leftarrow \leftarrow \\ 1 \quad 1 \end{array} \Rightarrow \text{NO OVF}$$

$$= -113$$

~~✱~~

$$\begin{array}{r} -\cancel{128} \\ -64 \\ -49 \\ \hline -113 \end{array}$$



a_i	b_i	C_i	C_{i+1}	s_i
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