A few notes on circuits for addition and subtraction

Timothy Bourke

École normale supérieure

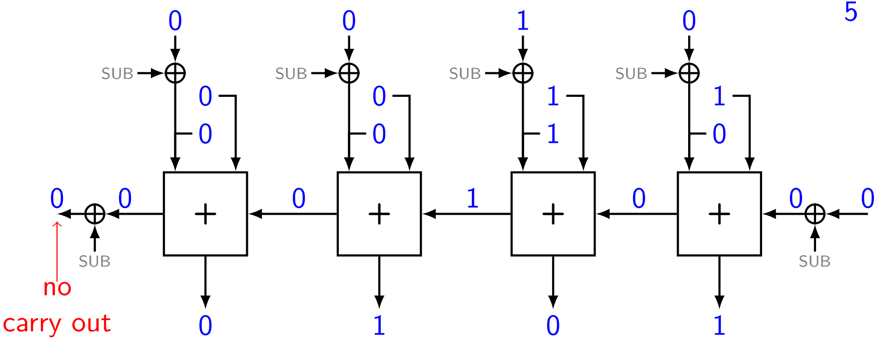
10 November 2020

1. Unsigned 4-bit arithmetic

- UINT_MIN $= 0$
- UINT_MAX $= 15$ $(2^4 - 1)$

Unsigned 4-bit arithmetic (1)

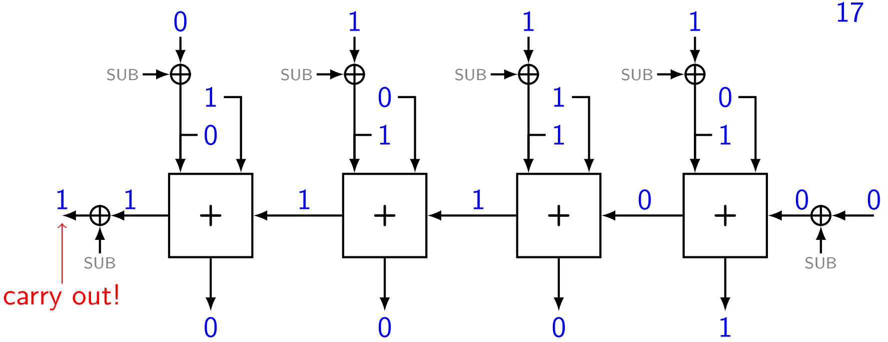
$$\begin{array}{rl} 3 & \\ 2 & + \\ \hline 5 & \end{array}$$



- We add two unsigned integers: $3 + 2$.
- The unsigned result is between 0 and 15 (inclusive).
- So there's no problem and the "carry out" flag is 0.

Unsigned 4-bit arithmetic (2)

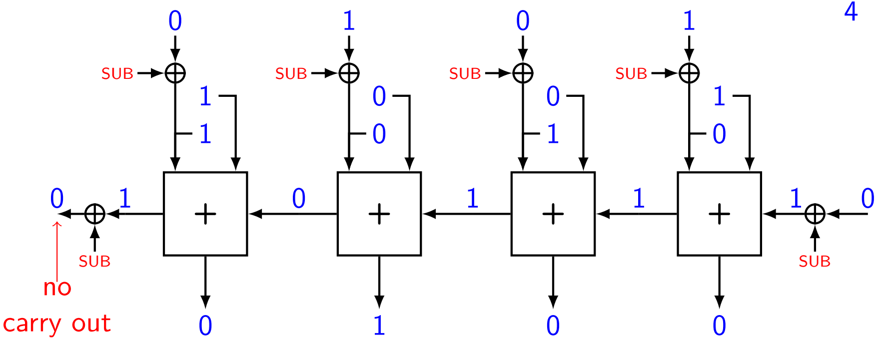
$$\begin{array}{rr} 10 & \\ 7 & + \\ \hline 17 & \end{array}$$



- We add two unsigned integers: $10 + 7$.
- The unsigned result is too big to be represented with 4 bits.
- So there's a problem and the "carry out" flag is 1.

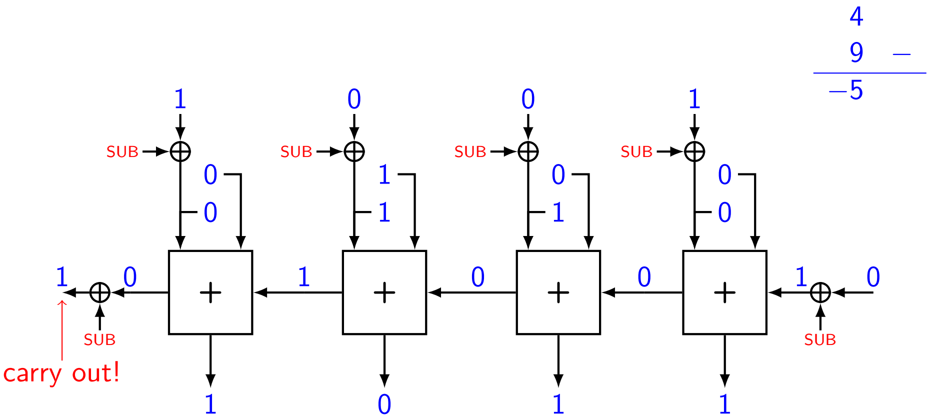
Unsigned 4-bit arithmetic (3)

$$\begin{array}{r} 9 \\ 5 \; - \\ \hline 4 \end{array}$$



- We substract two unsigned integers: $9 - 5$.
- The unsigned result is between 0 and 15 (inclusive).
- So there's no problem and the "carry out" flag is 0 (. . . as long as we XOR with SUB).

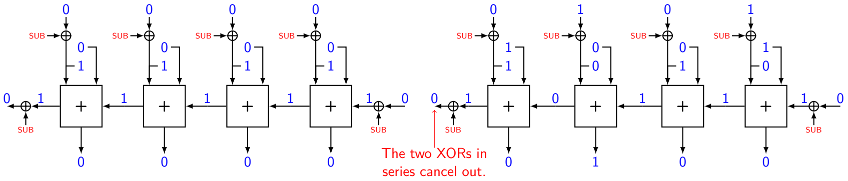
Unsigned 4-bit arithmetic (4)



- We subtract two unsigned integers: $4 - 9$.
- The result is negative and cannot be represented as an unsigned integer. (The result is correct modulo 2^4: $-5 + 16 = 11 = 1011_2$)
- So there's a problem and the "carry out" flag is 1 (... as long as we XOR with SUB).

Unsigned 4-bit arithmetic (3b)

$$\begin{array}{rr} 9 & \\ 5 & - \\ \hline 4 & \end{array}$$



- The XOR with SUB on the “carry out” of a 4-bit unit composes well with the “carry in” of another 4-bit unit to create an 8-bit unit.
- This extension still does not permit the representation of negative values as unsigned integers.

1. Signed 4-bit arithmetic

- `INT_MIN` = -8 (-2^3)
- `INT_MAX` = 7 ($2^3 - 1$)

Signed 4-bit arithmetic: overflow

- We use the same circuit to calculate with signed integers.
- It's the interpretation of the result and the flags that changes.
- The "carry out" flag is not used for signed arithemetic.
- A new "overflow" flag is needed.

- There are two operands a and b and a result s. So there are eight cases to consider:

a	b	s	ADD
pos.	pos.	pos.	OK
pos.	pos.	neg.	KO: Adding two positive numbers should never give a negative one!
pos.	neg.	pos.	OK
pos.	neg.	neg.	OK
neg.	pos.	pos.	OK
neg.	pos.	neg.	OK
neg.	neg.	pos.	KO: Adding two negative numbers should never give a positive one!
neg.	neg.	neg.	OK

(The calculation for pos.+ pos. is correct if one adds 2^n to it and similarly for neg.+ neg. if one subtracts 2^n from it.)

Signed 4-bit arithmetic: overflow

- We use the same circuit to calculate with signed integers.
- It's the interpretation of the result and the flags that changes.
- The "carry out" flag is not used for signed arithemetic.
- A new "overflow" flag is needed.

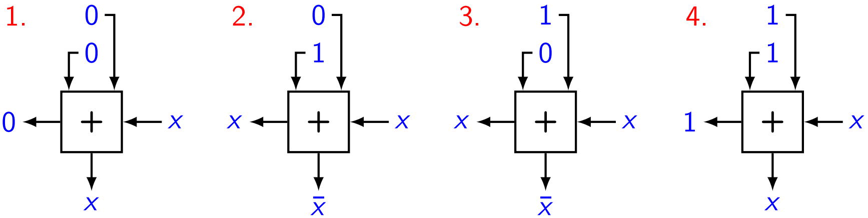
- Notice that in two's complement the most significant bit signifies whether a value is positive or negative:

a_3	b_3	s_3	ADD
0	0	0	overflow $= 0$
0	0	1	overflow $= 1$
0	1	0	overflow $= 0$
0	1	1	overflow $= 0$
1	0	0	overflow $= 0$
1	0	1	overflow $= 0$
1	1	0	overflow $= 1$
1	1	1	overflow $= 0$

Signed 4-bit arithmetic: overflow

a_3	b_3	s_3	ADD
0	0	0	overflow $= 0$
0	0	1	overflow $= 1$
0	1	0	overflow $= 0$
0	1	1	overflow $= 0$
1	0	0	overflow $= 0$
1	0	1	overflow $= 0$
1	1	0	overflow $= 1$
1	1	1	overflow $= 0$

- Consider the four cases for a_3, b_3, and s_3:

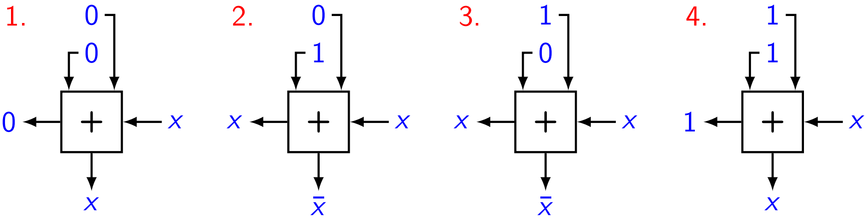


There is only a problem in case 1 when $x = 1$ and in case 2 when $x = 0$. An overflow must be signalled iff $\text{cout}_3 \neq \text{cin}_3$.

Signed 4-bit arithmetic: overflow

a_3	b_3	s_3	ADD
0	0	0	overflow = 0
0	0	1	overflow = 1
0	1	0	overflow = 0
0	1	1	overflow = 0
1	0	0	overflow = 0
1	0	1	overflow = 0
1	1	0	overflow = 1
1	1	1	overflow = 0

- Consider the four cases for a_3, b_3, and s_3:



There is only a problem in case 1 when $x = 1$ and in case 2 when $x = 0$. An overflow must be signalled iff $\text{cout}_3 \oplus \text{cin}_3 = 1$.

Signed 4-bit arithmetic: overflow

- Does the same definition work for subtraction?
- Consider the eight cases:

a	b	s	SUB
pos.	pos.	pos.	OK
pos.	pos.	neg.	OK
pos.	neg.	pos.	OK
pos.	neg.	neg.	KO: $x-(-y)=x+y\geq 0$!
neg.	pos.	pos.	KO: $(-x)-y\leq 0$!
neg.	pos.	neg.	OK
neg.	neg.	pos.	OK
neg.	neg.	neg.	OK

Signed 4-bit arithmetic: overflow

- Does the same definition work for subtraction?
- Consider the eight cases:

a_3	b_3	s_3	SUB
0	0	0	overflow $= 0$
0	0	1	overflow $= 0$
0	1	0	overflow $= 0$
0	1	1	overflow $= 1$
1	0	0	overflow $= 1$
1	0	1	overflow $= 0$
1	1	0	overflow $= 0$
1	1	1	overflow $= 0$

Signed 4-bit arithmetic: overflow

- Does the same definition work for subtraction?
- and then invert b_3:

a_3	$\bar{b}_3$	s_3	SUB
0	1	0	overflow $= 0$
0	1	1	overflow $= 0$
0	0	0	overflow $= 0$
0	0	1	overflow $= 1$
1	1	0	overflow $= 1$
1	1	1	overflow $= 0$
1	0	0	overflow $= 0$
1	0	1	overflow $= 0$

Signed 4-bit arithmetic: overflow

- Does the same definition work for subtraction?
- and then invert b_3:

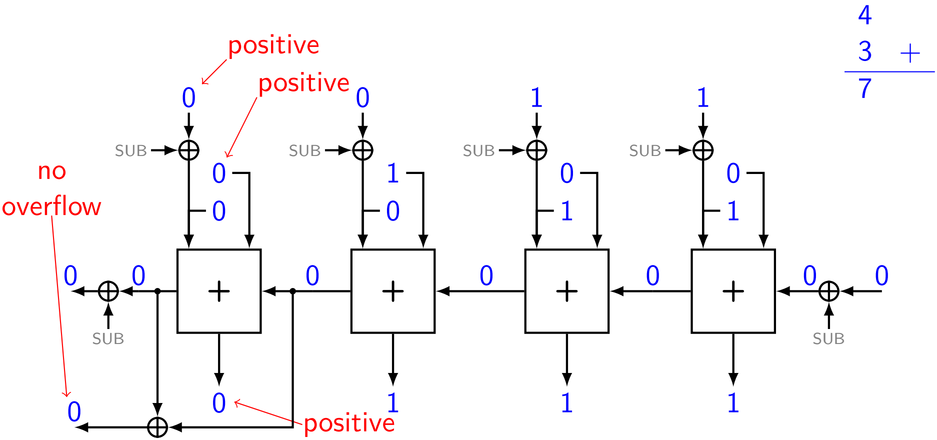
a_3	$\bar{b}_3$	s_3	SUB
0	1	0	overflow $= 0$
0	1	1	overflow $= 0$
0	0	0	overflow $= 0$
0	0	1	overflow $= 1$
1	1	0	overflow $= 1$
1	1	1	overflow $= 0$
1	0	0	overflow $= 0$
1	0	1	overflow $= 0$

- . . . after a slight reordering:

a_3	$\bar{b}_3$	s_3	SUB
0	0	0	overflow $= 0$
0	0	1	overflow $= 1$
0	1	0	overflow $= 0$
0	1	1	overflow $= 0$
1	0	0	overflow $= 0$
1	0	1	overflow $= 0$
1	1	0	overflow $= 1$
1	1	1	overflow $= 0$

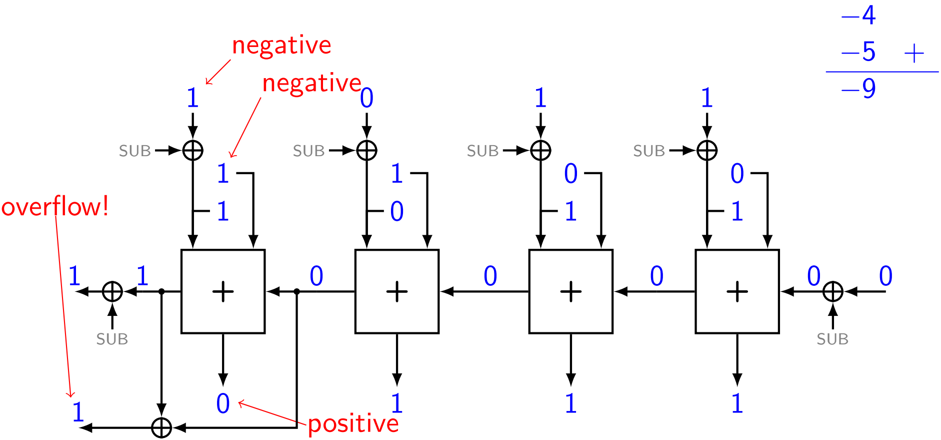
- We end up with the same table as before and thus the same definition.

Signed 4-bit arithmetic (1)



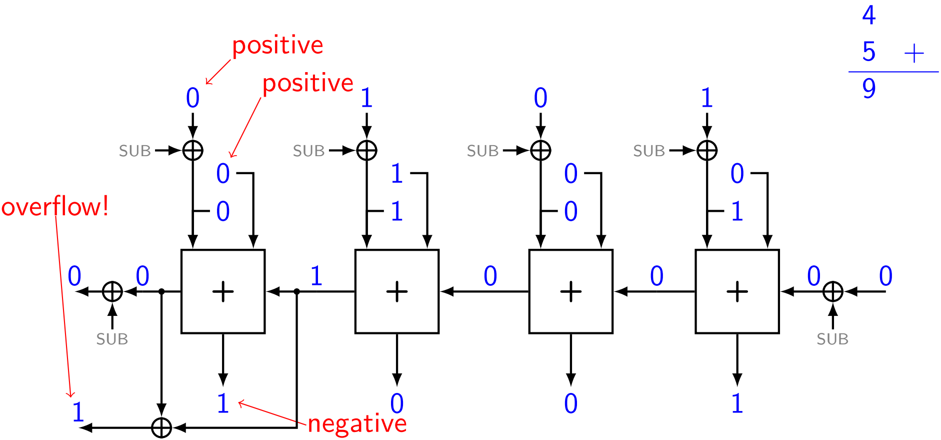
- We add two signed integers: $4 + 3$.
- The signed result is between -8 and 7 (inclusive).
- So there's no problem and the "overflow" flag is 0.

Signed 4-bit arithmetic (2)



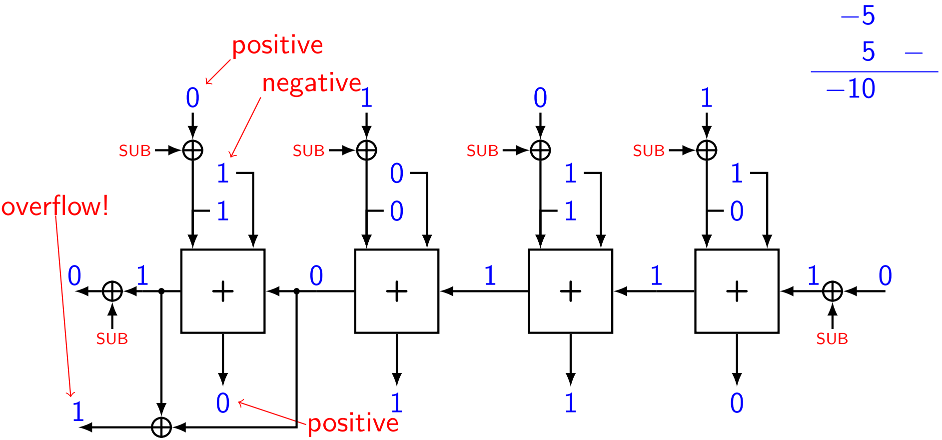
- We add two signed integers: $-4 + (-5)$.
- The signed result is too small to be represented in 4 bits.
- So there's a problem and the "overflow" flag is 1.

Signed 4-bit arithmetic (3)



- We add two signed integers: $4 + 5$.
- The signed result is too big to be represented in 4 bits.
- So there's a problem and the "overflow" flag is 1.

Signed 4-bit arithmetic (4)



- We subtract two signed integers: $-5 - 5$.
- The signed result is too small to be represented in 4 bits.
- So there's a problem and the “overflow” flag is 1.

- To read: Ian D. Allen's notes
  (`http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt`)