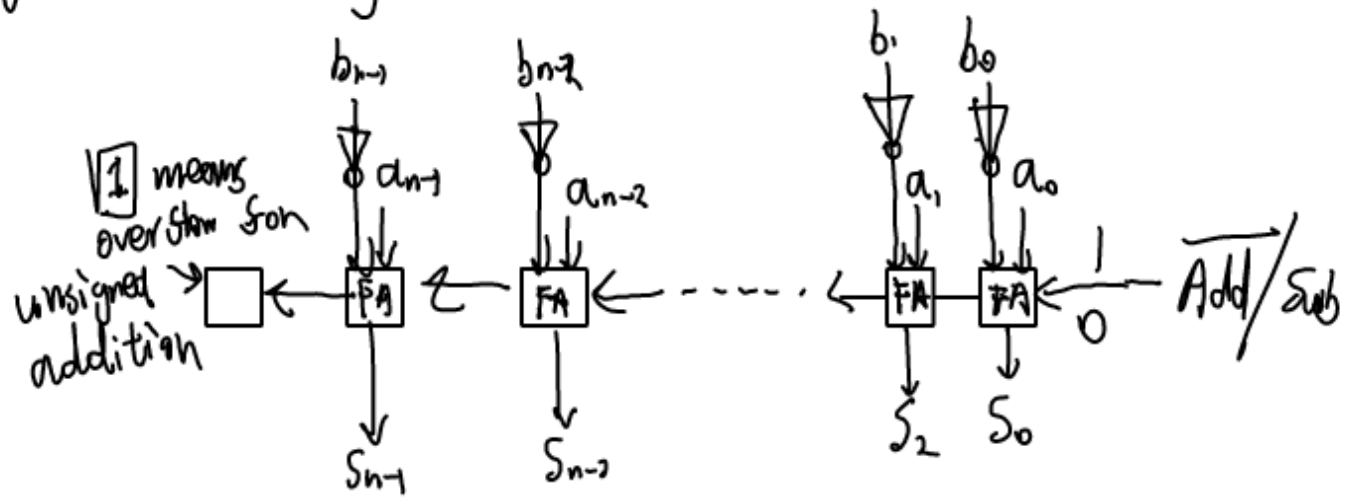
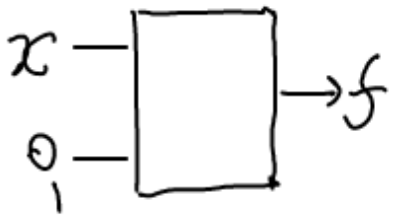
* Make a circuit that can both add & subtract unsigned #'s. Using stuff we already have.



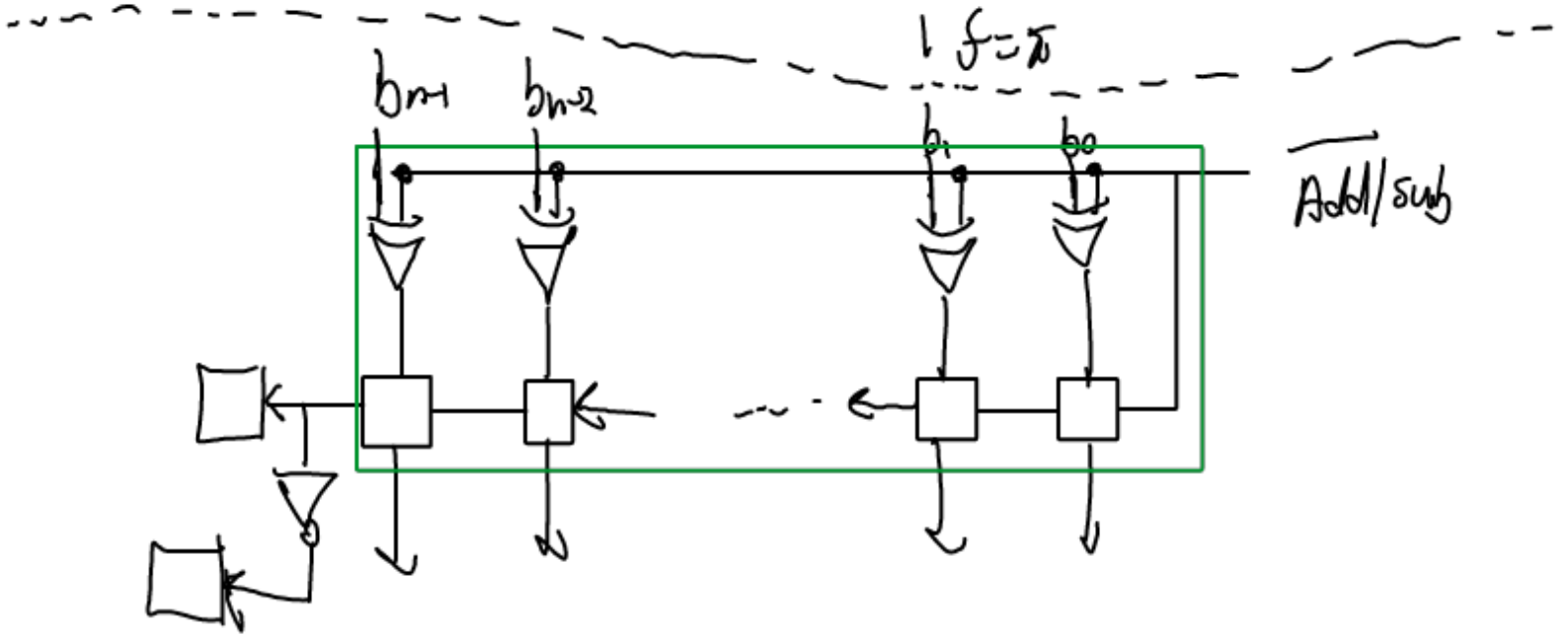
0 means add

1 means sub.



If $f = x$ use XOR

If $f = \bar{x}$



Signed Integers

Q. how to represent signed int's?

technique #1:
use sign bit + then other bits for magnitude.
(sign + magnitude)

0 000 1001 ↑ +9 in 8-bits

−9? 1 000 1001 = −9

We actually already have the correct soln.

⇒ use 2's complement for -ve #s.

0 000 1001 → 1111 0111 ← the actual value is "encoded" inside all the bits.

+9 −9

Not really a sign bit anymore, but a leading "1" does mean the # is negative.

1101 ← What is this #?

→ unsigned? 1+0+4+8 = 13

→ signed?

→ sign + mag? −5

→ 2's complement? 1 101 → 0011 (2's) → 3 (neg)

Overflow for signed #s.

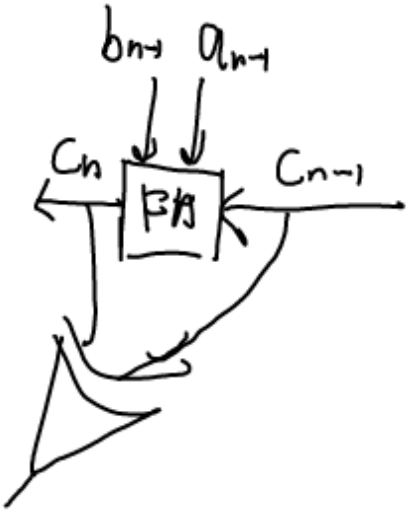
70 + 80 = 150

0100 0110 + 0101 0000 = 1 0010 0110

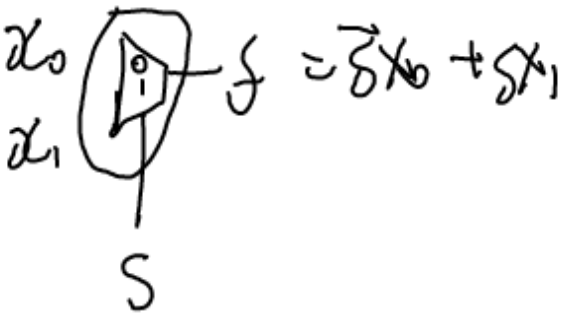
doesn't indicate overflow for signed ints.

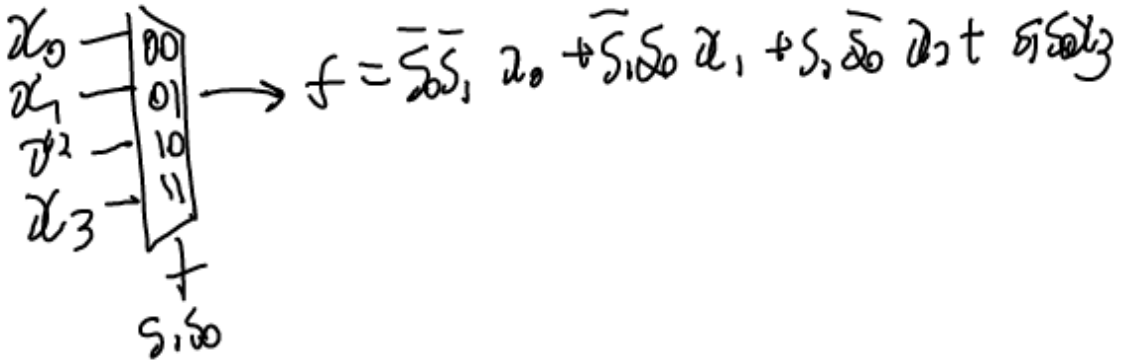
(−70) + (−80) = −150

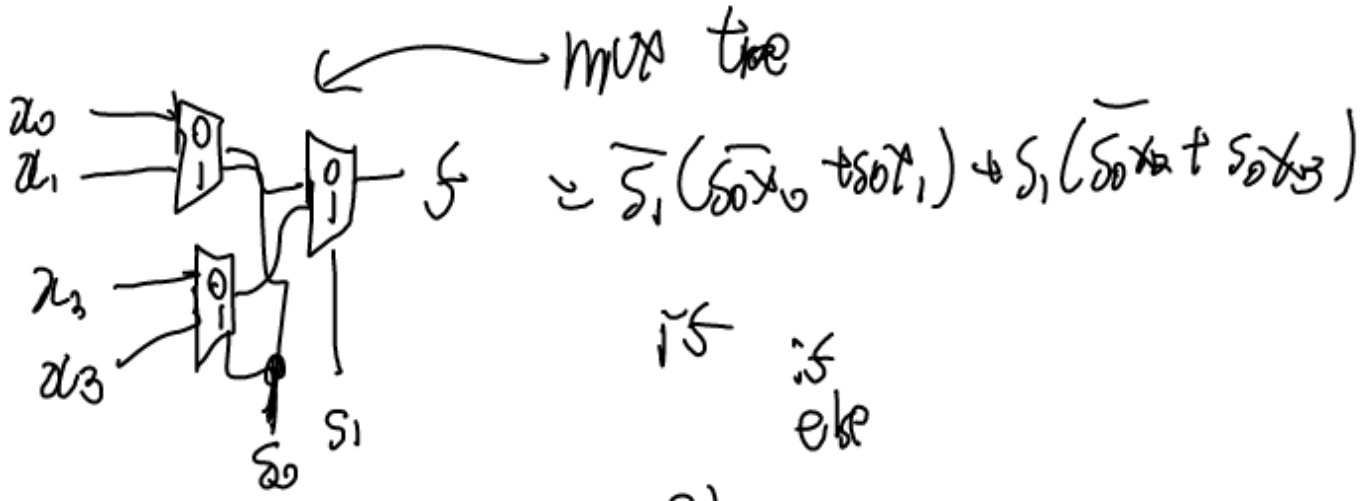
1011 1010 + 1011 0000 = 1 0111 0001



MUX:







MUX to impl. logic functions

* Any n-input function can be implemented with a mux with n-1 select lines.

x_0	x_1	x_2	δ
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

3 input δ using mux with 3 - select lines