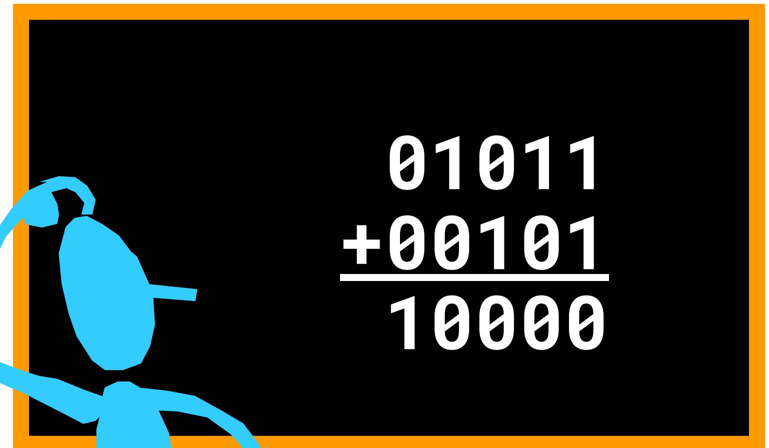
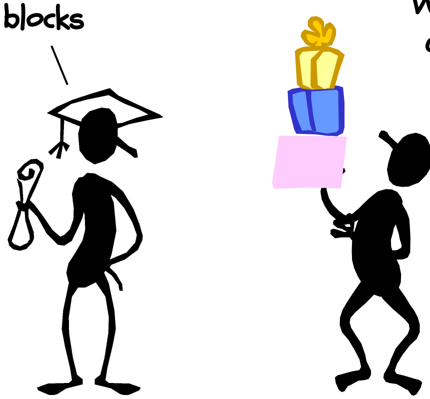
Arithmetic Circuits





- How to add and subtract using combinational logic
- Setting flags
- Adding faster

Review: Binary Representations

- Unsigned numbers, each increasingly significant bit has a weight of the next larger power of 2
- Signed 2's complement representation the most significant bit is a negative power of 2.

unsigned: $$v = \sum_{i=0}^{n-1} 2^i b_i$$

signed: $$v = -2^{n-1} b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i$$

2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	0

4294967254 -or- -42

- Why?
  - They are compatible. The same logic can be used for both
  - Only "adders" are needed for both addition and subtraction

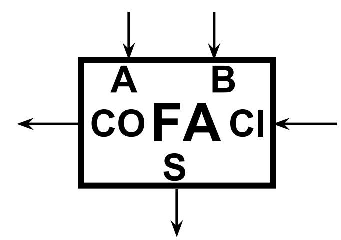
Binary Addition

Here's an example of binary addition as one might do it by "hand":

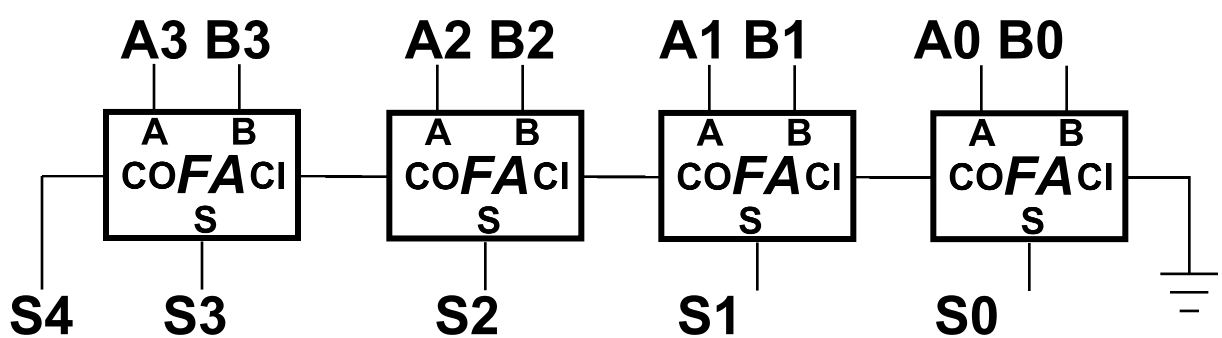
Adding two N-bit numbers produces an (N+1)-bit result

```
      1 1 0 1    <- Carries from previous column
A:     1101
B:  +  0101
      10010
```

Let's start by building a block to add one column:
This functional block is called a "Full-adder"



Then we can cascade them to add two numbers of any size...



Design of a "Full Adder"

1) Start with a truth table:

2) Write down equations for the "1" outputs

CO = (!CI & A & B) | (CI & !A & B)
| (CI & A & !B) | (CI & A & B)
S = (!CI & !A & B) | (!CI & A & !B)
| (CI & !A & !B) | (CI & A & B)

3) Simplifying a bit

CO = (CI & (A | B)) | (A & B)
S = CI ^ A ^ B

CO = (CI & (A ^ B)) | (A & B)
S = CI ^ (A ^ B)

C_i	A	B	C_o	S
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

As a Logic Diagram

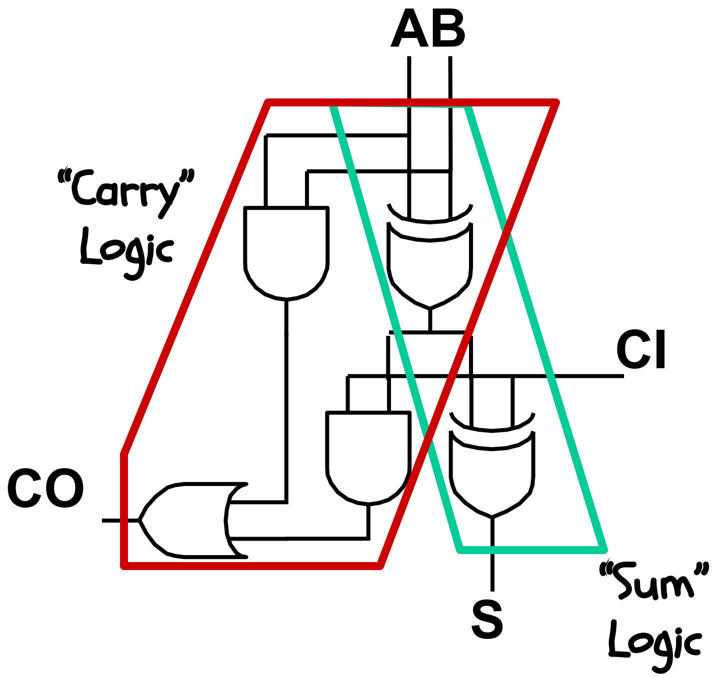
- Our equations:

  CO = (CI & (A ^ B)) | (A & B)

  S = CI ^ (A ^ B)

- A little tricky, but finally

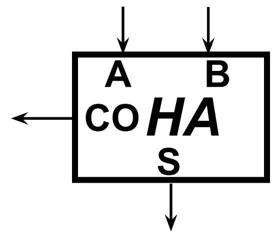
  Only 5 gates/bit



An Aside: Why Full Adder?

Suppose you don't want/need a carry-in?

Then you get a **"half adder"** with 2 inputs and 2 outputs:

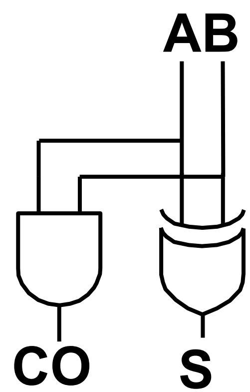


A	B	CO	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Half-adder equations:

  CO = A & B

  S = A ^ B

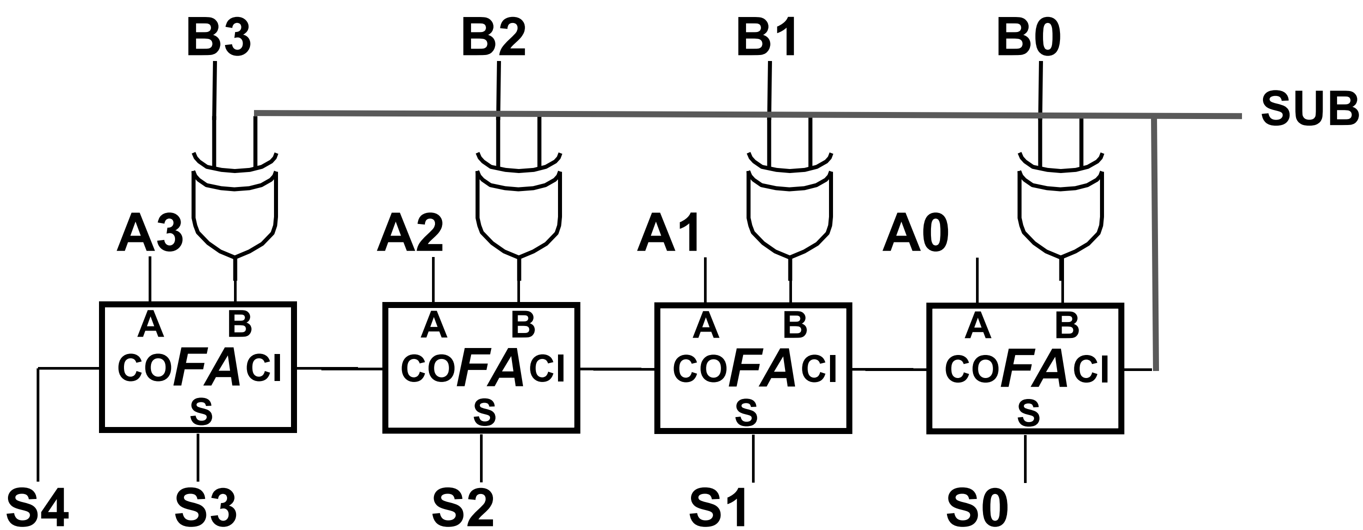


Subtraction: A - B = A + (-B)

- Recall the trick was to "complement and add 1"
- How to complement?

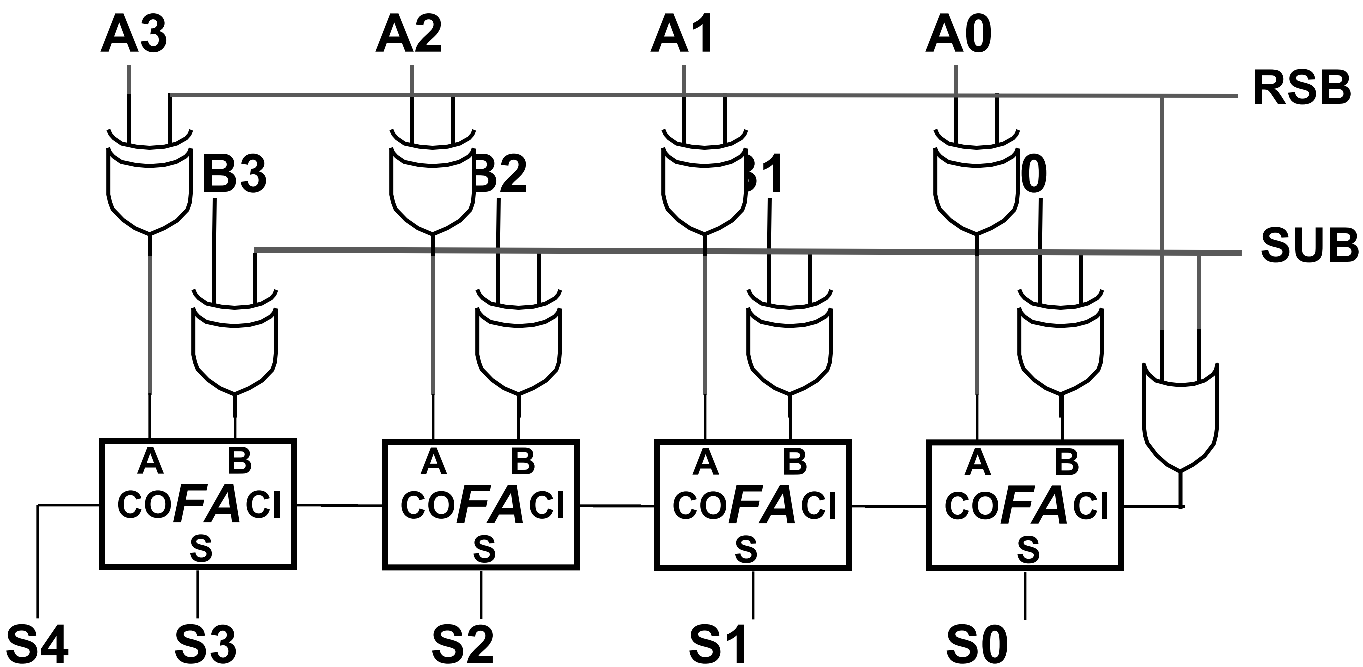
~ = bitwise complement

- So now a unit that can either add or subtract



Reverse Subtract: -A + B

- And with a few more XOR gates we can subtract either the A or the B operands



Condition Flags

Besides the sum, one often wants four other bits of information from an arithmetic unit, the condition flags.

Z (zero): result is = 0 — *big NOR gate*

N (negative): result is < 0 — *S_{31}*

C (carry): indicates the most significant bit produced a carry, e.g., "1 + (-1)" — *CO_{31} (of last FA)*

V (overflow): indicates an unexpected change in sign e.g., "$(2^{30} - 1) + 1$" — *$(A_{31} \& B_{31} \& !S_{31}) \mid (!A_{31} \& !B_{31} \& S_{31})$*

-- or --

$CO_{31} \wedge CO_{30}$

How condition flags are used in conditional execution

Signed comparison:

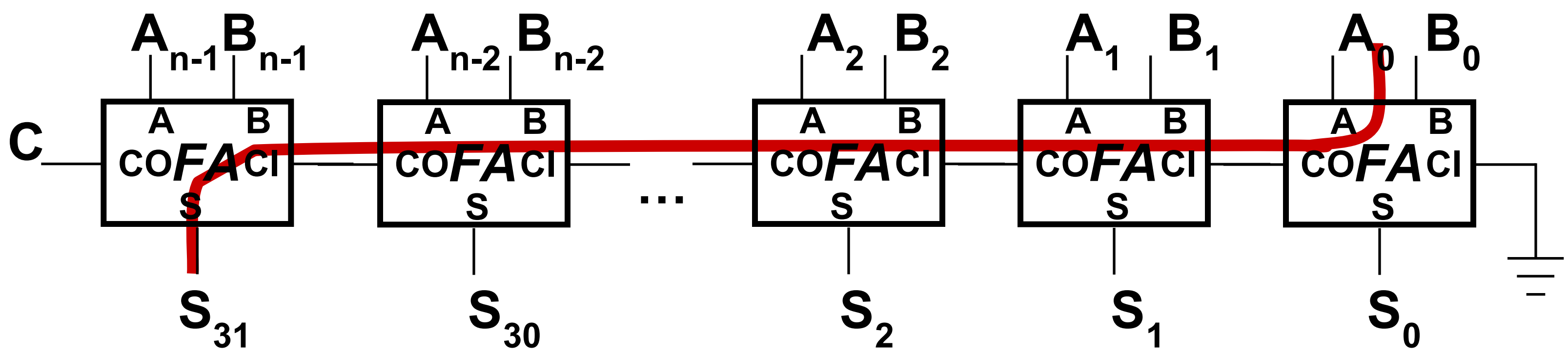
lt	N ^ V
le	Z | (N ^ V)
eq	Z
ne	!Z
ge	!(N ^ V)
gt	!(Z | (N ^ V))

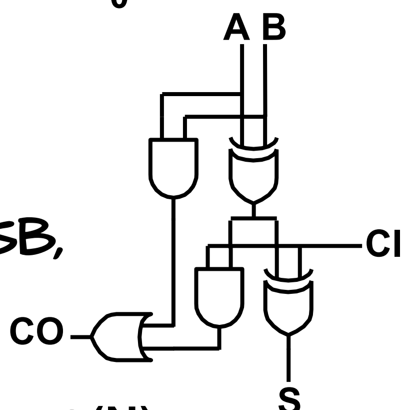
Unsigned comparison:

hi	C & !Z	
ls	!C | Z	
lo	!C	(same as cc)
hs	C	(same as cs)

How fast is an Add?

Determined by T_{pd} of the FA chain





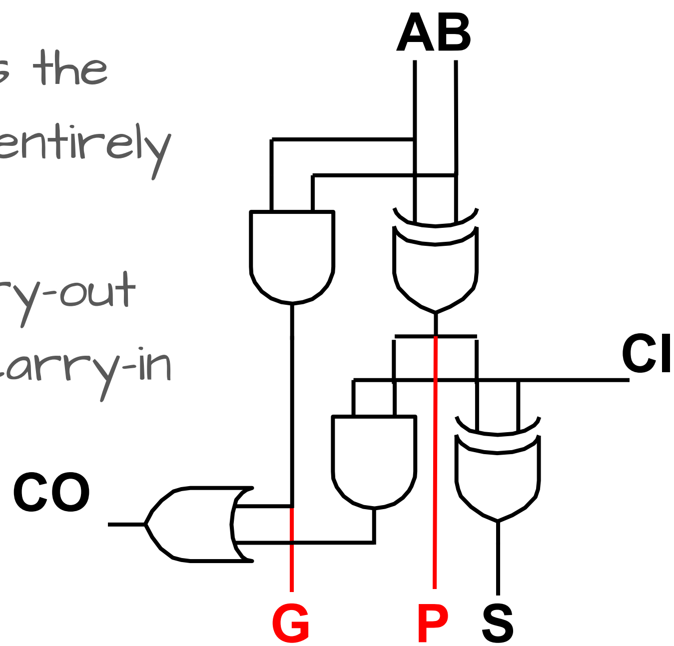
Worse-case path: carry propagation from LSB to MSB, e.g., when adding -1 to 1.

$$t_{PD} = (t_{PD,XOR} + t_{PD,AND} + t_{PD,OR}) + (N-2)*(t_{PD,OR} + t_{PD,AND}) + t_{PD,XOR} \approx \Theta(N)$$

We can add "much" faster

Using more gates we can speed up adding considerably if we add 2 "free" extra outputs from our adder

- P, Propagate, means the carry-out depends entirely on the carry-in
- G, generates a carry-out regardless of the carry-in

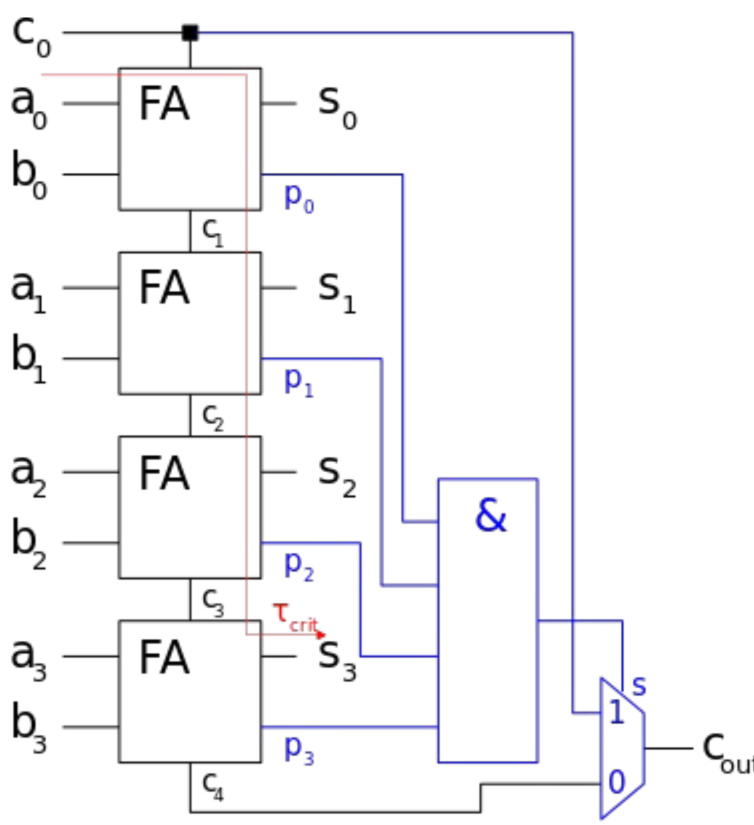


C_i	A	B	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Carry-skip adders

If all full adders in a contiguous block have their Propagate true, then the incoming carry-in can "skip" over the entire block!

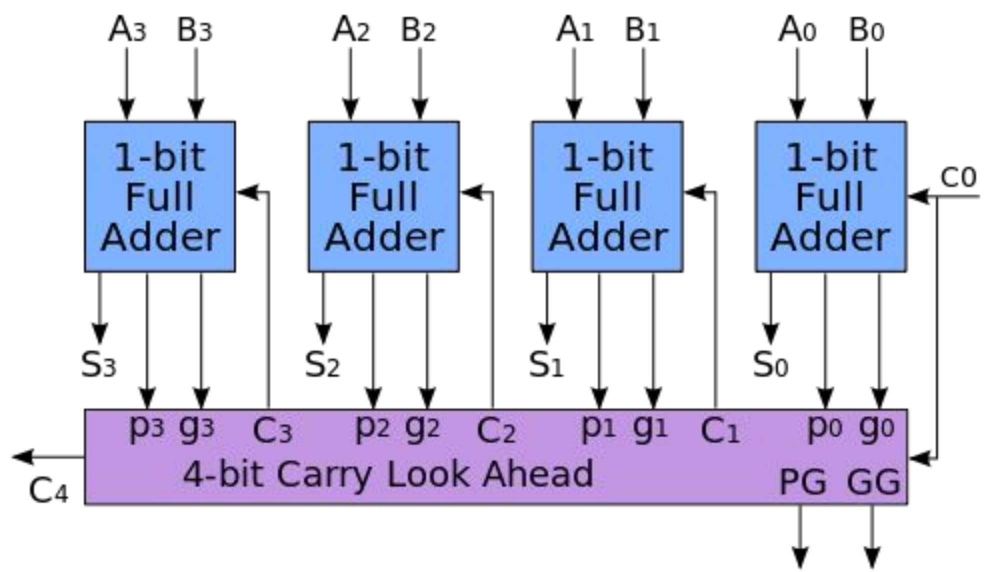
Requires extra AND gates and a MUX, but reduces the worst case add-time



Full Carry-Lookahead

The fastest adders use full carry look-ahead.

- Given the Ps and Gs of a block, one can simultaneously compute the carry-ins for all bits as well as the block using the 3-level SOP methods discussed last lecture.



- Results in an $\Theta(\log_2(N))$, T_{pd}, like an N-input AND gate, using ≈2x more gates

Next Time

We get shifty, no, Bool!