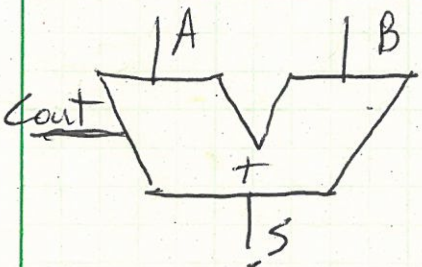
<u>Overview:</u>

- adders
  - half adder
  - full adder
  - carry propagate
- subtractors
- Comparators
  - equality
  - magnitude

Half Adder (1-Bit)



A B	Cout	S
00	0	0
01	0	1
10	0	1
11	1	0

$\Longrightarrow$ <u>Eqns:</u>

$Cout = AB$

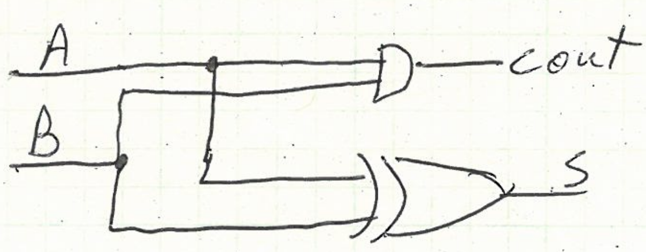
$S = A \oplus B$

Any Problems with this?

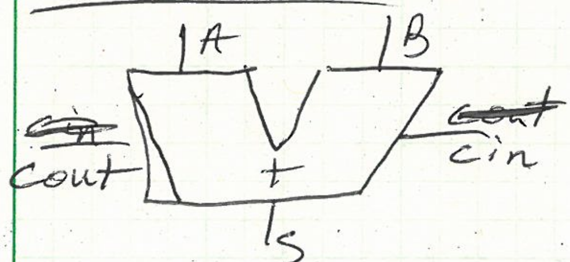
Limited in function. Doesn't contain circuitry to take/make use of cin/cout.

So we use a Full Adder

<u>Draw schematic:</u>



Full Adder: (1-Bit)

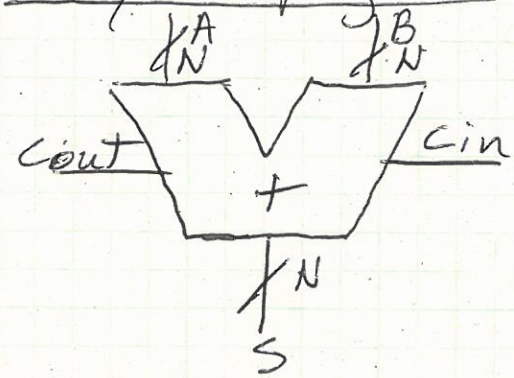


Cin A B	Cout	S
000	0	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	1	1

<u>Eqns:</u> $S = Cin \oplus A \oplus B$

$Cout = Cin\,B + AB + Cin\,A$

Carry Propogate

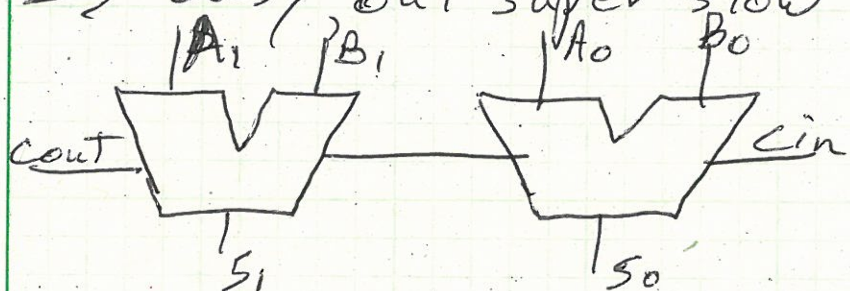


Types:
- ripple carry
- carry Lookahead
- prefix

Ripple Carry:

- N - Full adders chained
- delay grows Linearly

⇒ easy but super slow



Carry Lookahead

- Divides adder into blocks to determine carry out ~~asap~~ ASAP
- N > 16
  - we start to see timing gains
  - faster than Ripple carry

⇒ delay still grows Linearly

- Generate - if carry out produced (G) independently of Cin (means we will generate a carry)

  $G \triangleq AB$

- Propagate - Cout produced whenever there is Cin (means we will propogate (ie pass) a carry

  $P = A + B$

Examples:

1010 ← $C_{in} = 1$
+0101

Cout? yes! $P = A + B$

1000
+1000

Cout? yes! $G = AB$

1011
0101

Cout? yes! why?

because of multiplication. we will have a carry in the ith most Bit

$$C_i = A_iB_i + (A_i + B_i)C_{i-1} = G_i + P_iC_{i-1}$$

3210
1011 $G_0 = 1$
0101 $P_{1,2,3} = 1$

Since we generate ~~gar~~ carry in Bit 0 & then propogate through 1 → 3 we have a Cout!

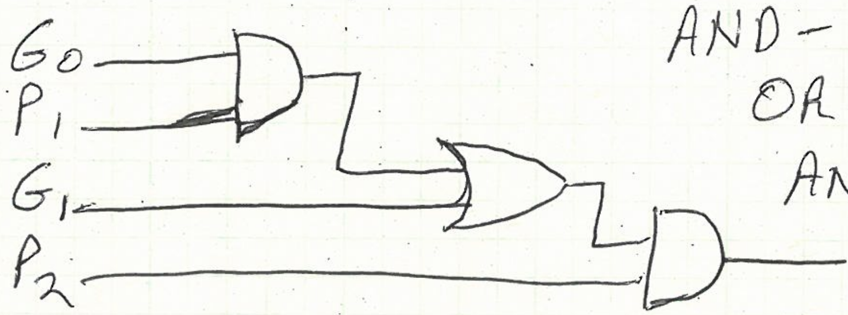
1010 $G_1 = 1$
+0110 $P_{2,3} = 1$

Cout? Yes!

1011
0111
← P P P_G G

1001 ← add $C_{in} = 1$
+0010

Cout? No!   No



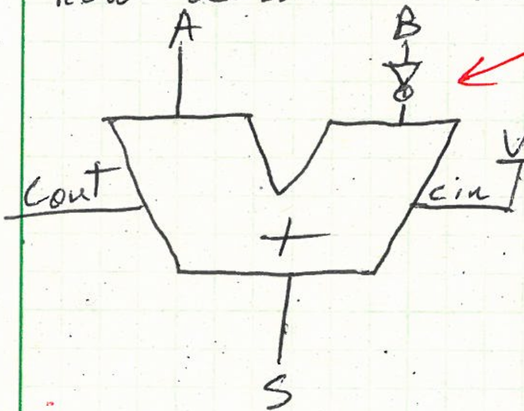
AND - next column propagate
OR - Next col Generate
AND - Next col Prop.

prefix:

- Look ahead by 2^x columns
- makes adder faster
- delays grow Logarithmically

Subtractor:

how do we subtract? using 2's complement.

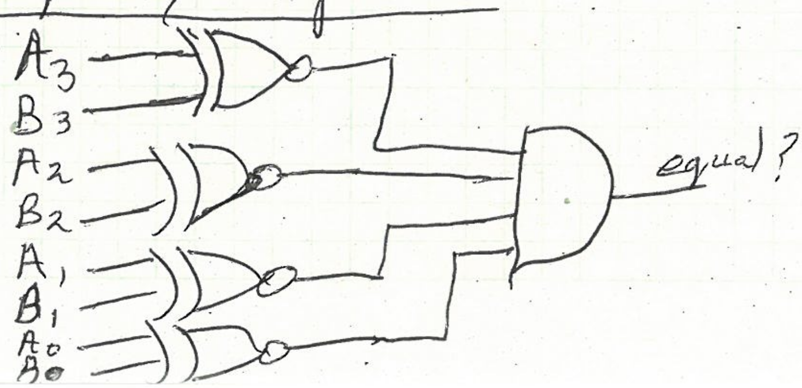


we can still use the same adder (Full adder) we used before

Comparator:

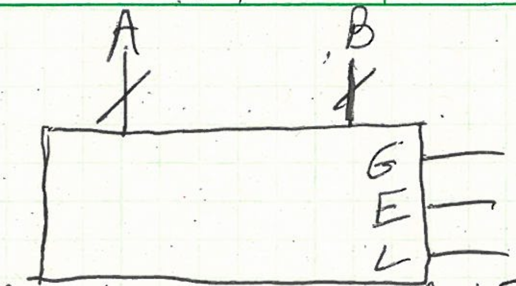
- equality
- magnitude

Equality Comparator:



XNOR:

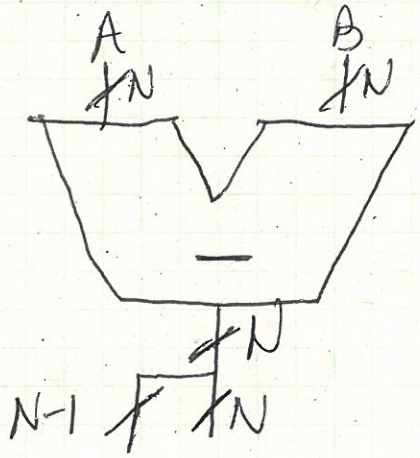
A B	Y
0 0	1
0 1	0
1 0	0
1 1	1



g=greater
e=equal
L=less

Why is this useful? How does it work

Boolean if statements in code



Implement this function using things we Learned today? No gates just Logic Functions

If a < 4
  z = y + 3
else
  z = y + Z