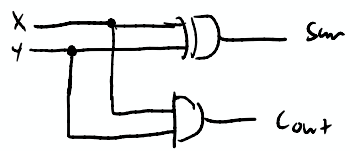
ARITHMETIC CIRCUITS

half-adder : add two bits

X	Y	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$S = X \oplus Y$

$C_{out} = XY$

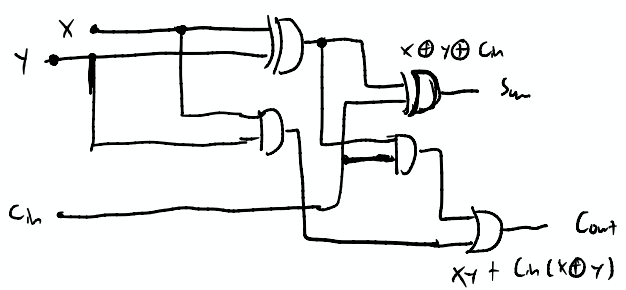
full-adder : add multiple bits w/ carry input from previous sums

X	Y	C_{in}	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S = X \oplus Y \oplus C_{in}$ (1 if odd, 0 if even)

$C_{out} = XY + C_{in}(X \oplus Y)$ (carry if $a+b+c \geq 2$)

- b if $x=1$ and $y=1$, carry = 1
- b if either x or y, and C_{in}, carry = 1
- b if $x(Y)(C_{in})$, same as XY

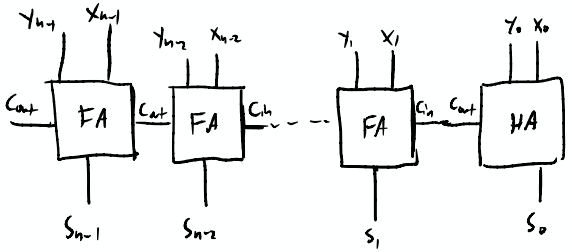


Ripple Carry Adders

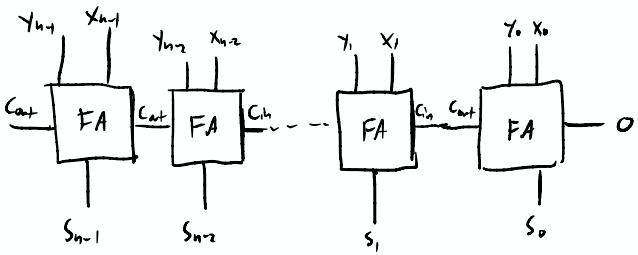
to add two n-bit numbers,

$X = (X_{n-1}, X_{n-2}, \ldots X_1, X_0)$

$Y = (Y_{n-1}, Y_{n-2}, \ldots, Y_1, Y_0)$



- HA = FA w/ carry 0.



performance analysis

- after a logic gate's input change, there is a propagation delay before the output changes
- to determine max delay, we need to find longest combinational path

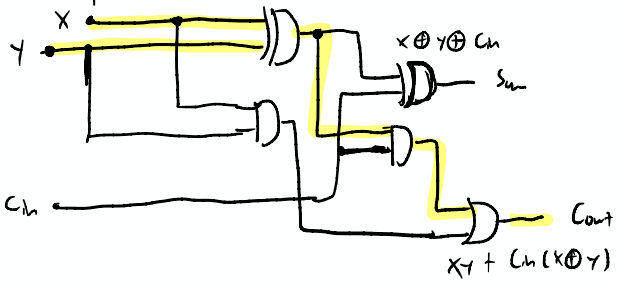
Assume all inputs are 0 initially for n-bit adder.

- S_{n-1} cannot compute its sum until S_{n-2} is computed.
- Similarly, S_1 cannot compute sum until S_0 is computed.
- In general, the i^{th} bit depends on all inputs from $i \to 0$.

∴ the longest potential path through ripple carry adder is from y_0/x_0 to C_{out} for n-1.

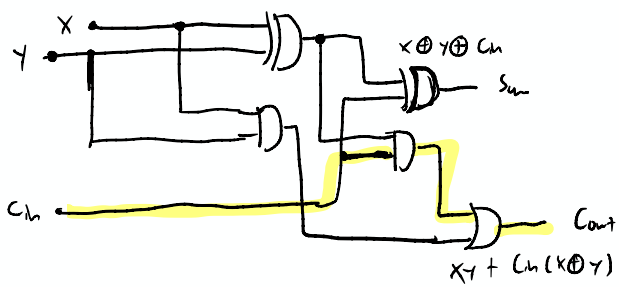
Assume we only use RCA made of full adders.

Recall a full adder:



to set to C_{out}, need to propagate through 1 XOR, 1 AND, 1 OR

for i^{th} bit, assume X and y already set. For C_{in} bit to propagate through, need to go via 1 AND, 1 OR



So the total delay is given by

$$(1 \text{ XOR} + 1 \text{ AND} + 1 \text{ OR}) + (n-1) * (1 \text{ AND} + 1 \text{ OR})$$

if delay of all gates are the same,

$$3 + (n-1)(2) = 2n + 1 \text{ gate delays.}$$

☆ performance is extremely bad for large n since values need to ripple down

borrowing

```
  1  1 1
  9 3 8 2
- 5 8 3 5 8
-----------
  3 5 3 9 4
```

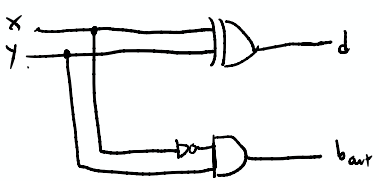
← the 1 means we take to subtract

half-Subtractor

- produces difference bit d and borrow bit b_{out}

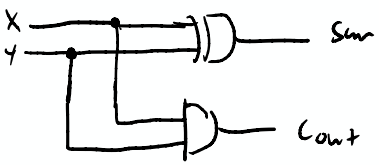
x	y	d	b_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

```
  0 1 10 1        10-1 (borrow)
  1 0 0 1 1 0      38
- 0 0 1 0 0 1    - 9
-------------    ----
  0 1 1 1 0 1     29
```



half subtractor

Vs.

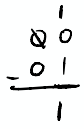


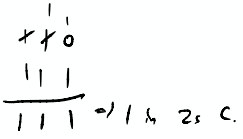
half-adder

full Subtractor

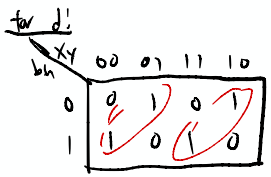
We need a circuit supporting multiple bit borrows

X	Y	b_{in}	d	b_{out}
0	0	0	0	0
0	1	0	1	1
1	0	0	1	0
1	1	0	0	0
0	0	1	1	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	1



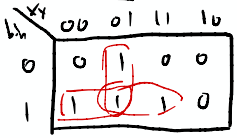


for d:



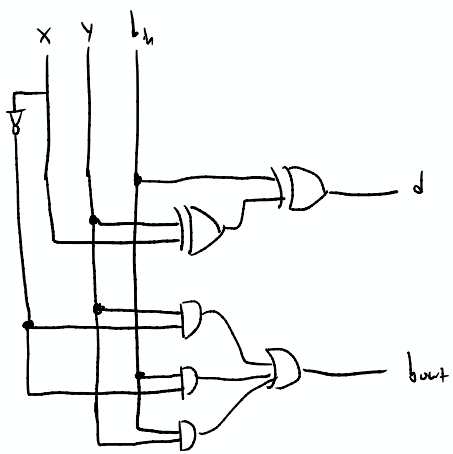
$= X \oplus Y \oplus b_{in}$

for b_{out}:



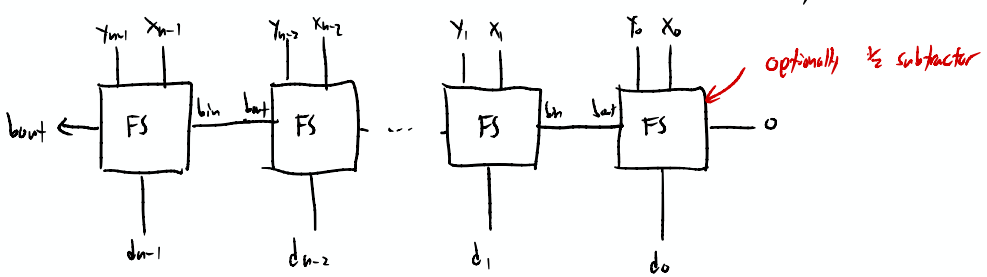
$= \bar{x}y + b_{in}\bar{x} + b_{in}y$

- if total disjoint is covered, can take xor of it map.



ripple subtractor

- to subtract two n-bit numbers, $Y = (Y_{n-1}, Y_{n-2}, \cdots Y_1, Y_0)$
  $X = (X_{n-1}, Y_{n-2}, \cdots X_1, X_0)$

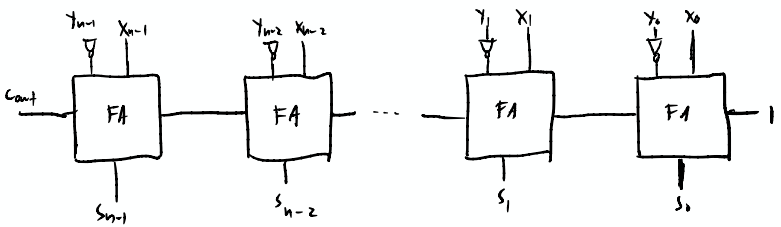


Combining addition & subtraction

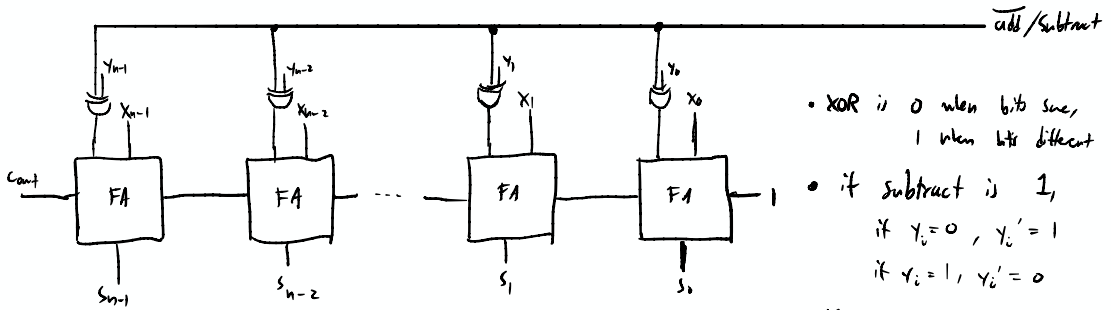
- we can add 2s Complement instead.
- taking 2s complement is equivalent to inverting each bit and adding 1.

  ex: 0100100 => 1011011 => 1011100

thus...



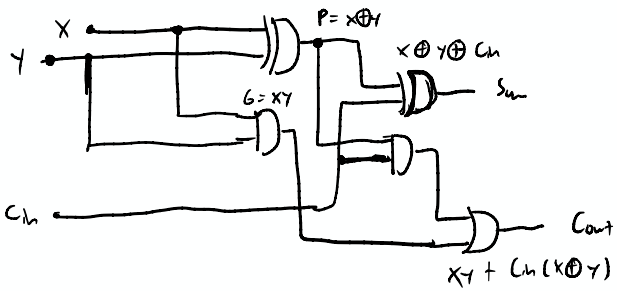
how do we do conditional addn. / subtraction?



- XOR is 0 when bits same, 1 when bits different
- if subtract is 1,
  - if $y_i = 0$, $y_i' = 1$
  - if $y_i = 1$, $y_i' = 0$
- if subtract = 0,
  - if $y_i = 0$, $y_i' = 0$
  - if $y_i = 1$, $y_i' = 1$

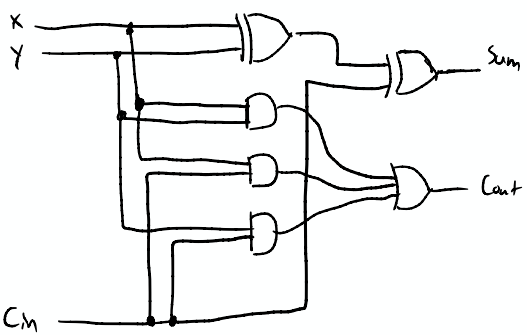
Carry-lookahead adder (CLA)

- ripple adder is slow since Cout needs to propagate n times.
- recall the full adder:



another way to write this circuit is

if XY, P = X+Y or X⊕Y. else P = X⊕Y.



$XY + XC_{in} + YC_{in}$

$= XY + C_{in}(X+Y)$

$= G + PC_{in}$

X	Y	XY	X+Y	X⊕Y	XY + X+Y	XY + X⊕Y
0	0	0	0	0	0	0
0	1	0	1	1	1	1
1	0	0	1	1	1	1
1	1	1	1	0	1	1

$XY + (X+Y) \Leftrightarrow XY + (X \oplus Y)$

$C_{out} = G + PC_{in}$, where $G = AB$ and $P = A+B$

- if G, Cout is generated immediately, w/o input from Cin
- if P is true, will overflow if $C_{in} = 1$. Otherwise safe.

We use generate/propagate model to fix this

Generate: Should a carry out be generated? (i.e. when sum > 1)

ex. in decimal, 52 + 67, 5 and 6 generates carry

in binary, A+B generates ⇔ A and B are 1.

⇒ $G(A,B) = A \cdot B$

Propagate: if no carry is generated, should we propagate it a carry coves from the right?

ex. in decimal, 37+62, 3 and 6 propagate it carry from ribht.

in binary, A+B propagates ⇔ at least A or B is 1 ($A+B+C_{in} > 1$)

⇒ $P(A,B) = A + B = A \oplus B$

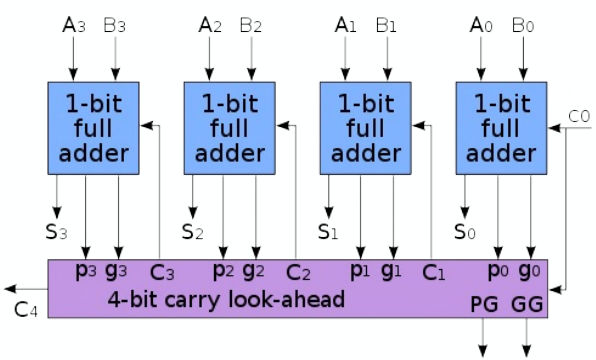
↳ OR is simpler to implement than XOR.

for the i^{th} bit in the addition, a carry will be generated iff addition generates or $i-1^{th}$ bit carries & i^{th} bit propagates. That is

$$C_{in}^{i+1} = C_{out}^{i} = G_i + C_{in}^{i-1} P_i$$

implementation

- For each bit, CLA determines if that bit pair will generate or propagate a carry.
  - notice how none of this is reliant on C_{in}
  - this allows for pre-processing for determining carry. When C_{in} arrives, the computation is instant.



$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 C_1$ $\Rightarrow$ Substituting, $C_2 = G_1 + G_0 P_1 + C_0 P_0 P_1$

$C_3 = G_2 + P_2 C_2$ $\quad C_3 = G_2 + G_1 P_2 + G_0 P_1 P_2 + C_0 P_0 P_1 P_2$

$C_4 = G_3 + P_3 C_3$ $\quad C_4 = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + C_0 P_0 P_1 P_2 P_3$

the 4-bit CLA can also be used in other circuits.

- Group Propagate (PG) and group generate (GG) are

$$PG = P_0 \cdot P_1 \cdot P_2 \cdot P_3$$

$$GG = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3$$

therefore, the final carry out for the group, CG, is

$$CG = GG + C_0 PG$$