

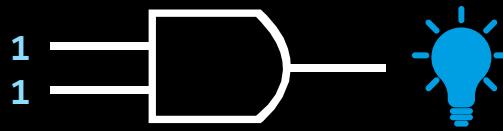
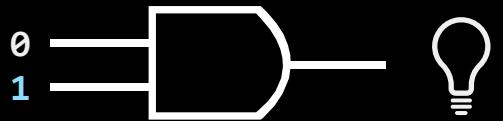
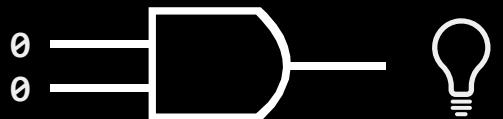
# LOGIC AND ARITHMETIC

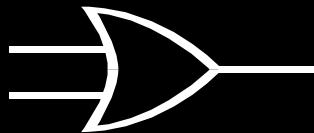
**logic gates** are the parts a computer is made of.

combined in the right way, they enable the basic arithmetic operations a computer can do: add, subtract, divide, and multiply.

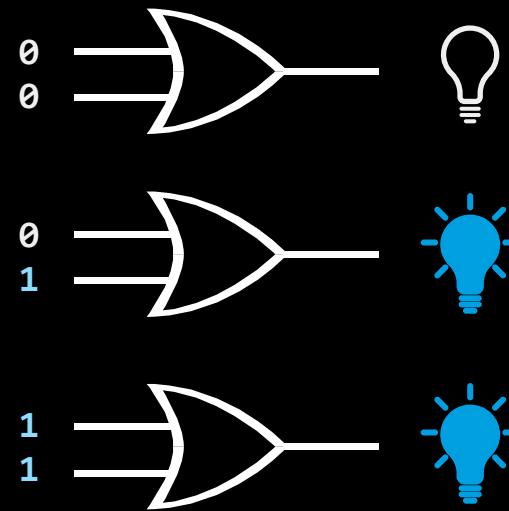


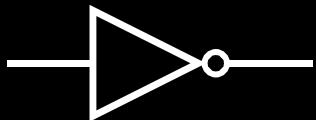
AND	0	1
0	0	0
1	0	1



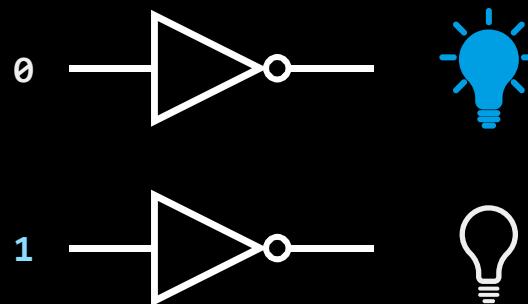


OR	0	1
0	0	1
1	1	1



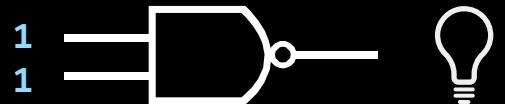
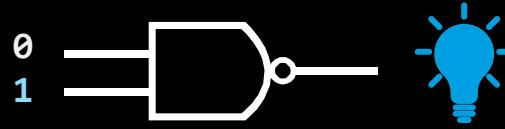
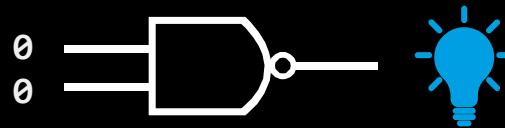


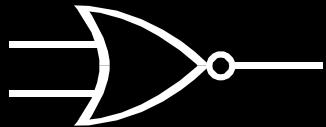
NOT	0	1
	1	0



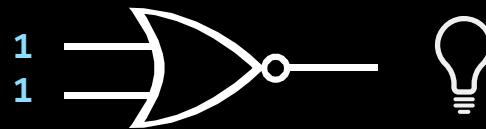
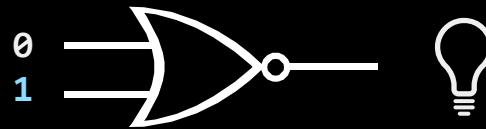
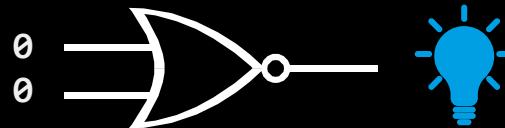


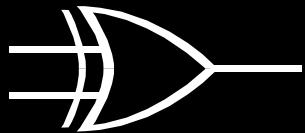
<b>NAND</b>	0	1
0	1	1
1	1	0



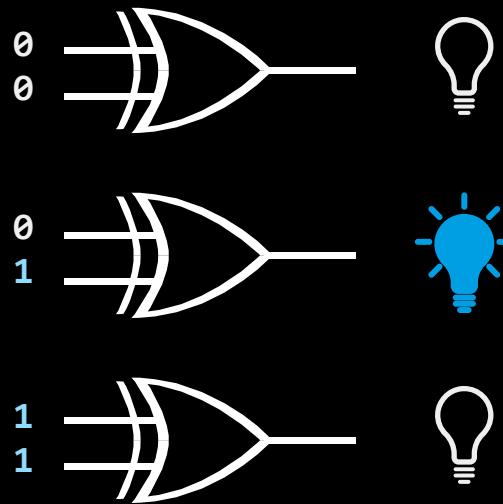


NOR	0	1
0	1	0
1	0	0



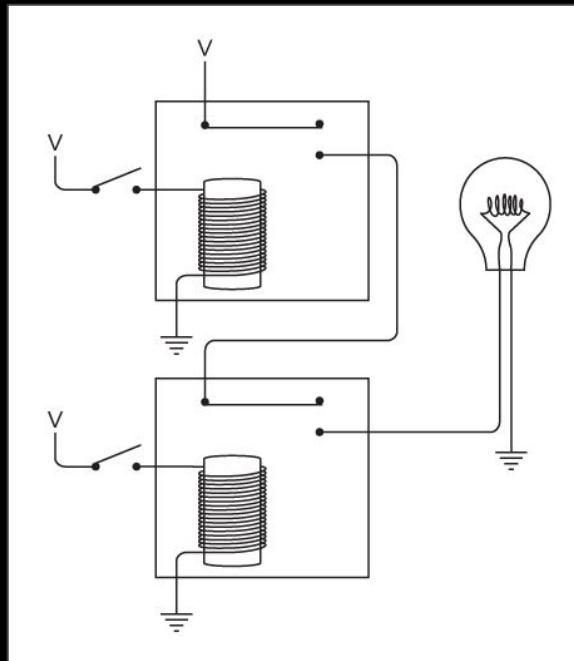


XOR	0	1
0	0	1
1	1	0

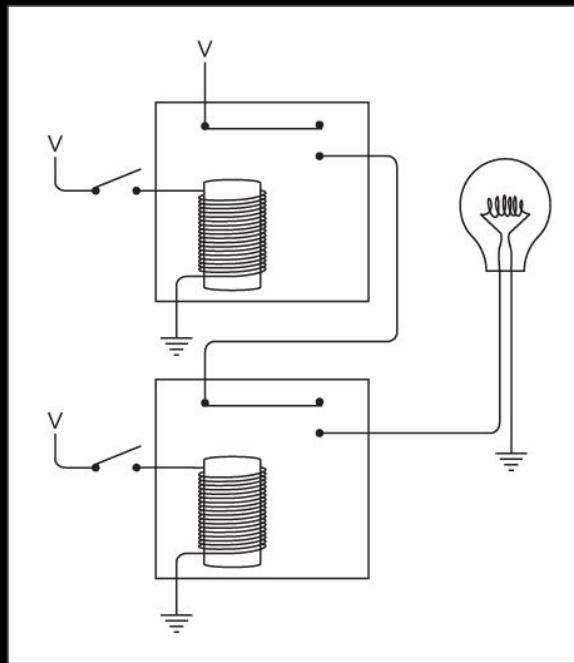


building a logic gate

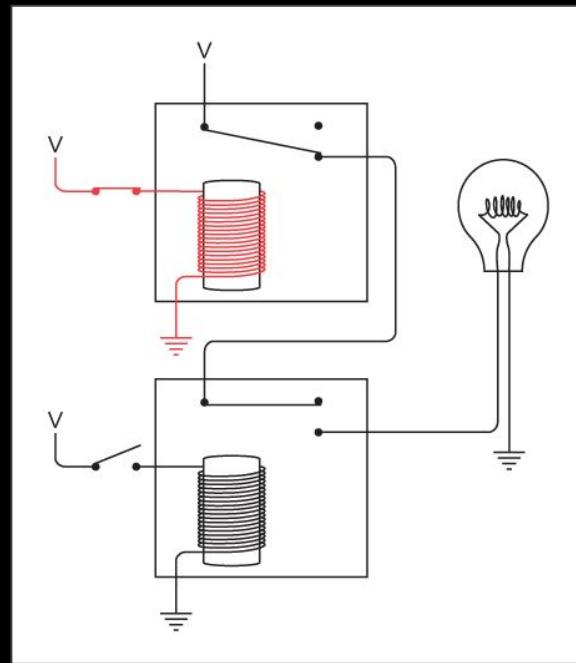
both inputs off



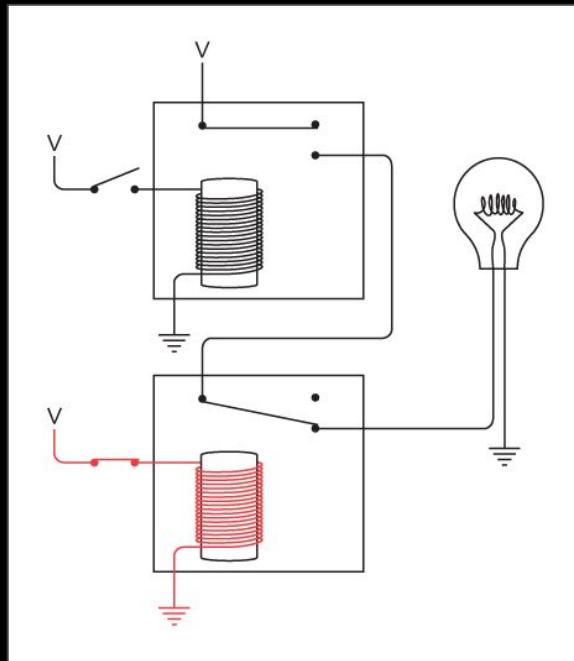
both inputs off



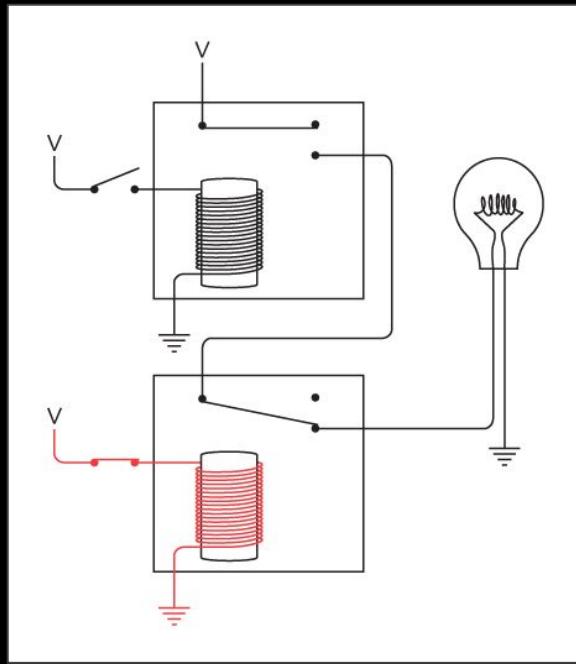
one input on



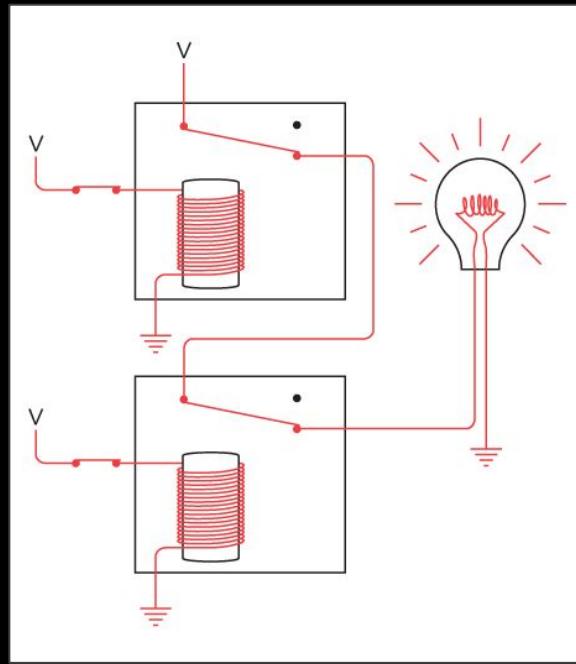
the other input on



the other input on



both inputs on



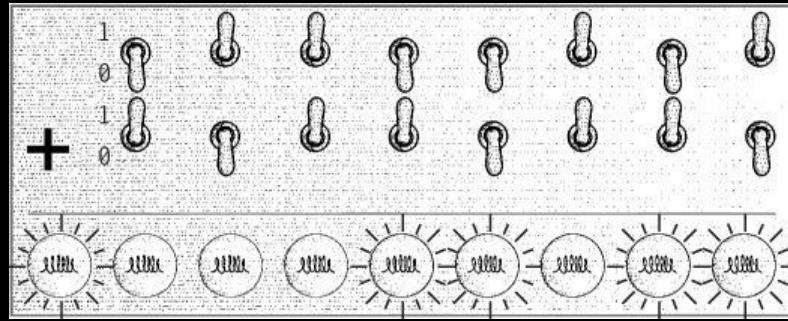
# binary addition

when you come right down to it, **addition** is just about the only thing that computers do.

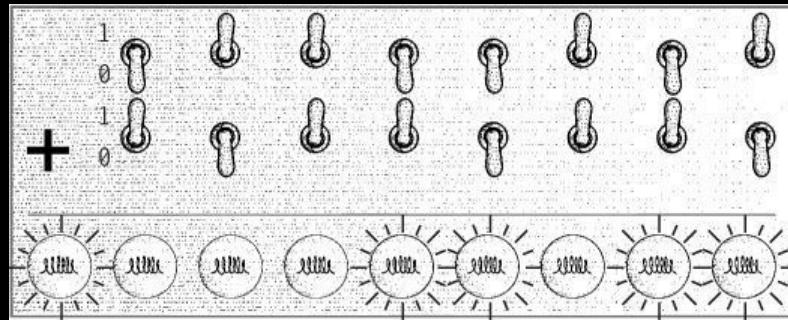
if we can build something that adds, we're well on our way to building something that uses addition to also subtract, multiply, divide, calculate mortgage payments, guide rockets to Mars, play chess, and foul up our phone bills.

(Charles Petzold)

# an 8-bit binary adding machine



# an 8-bit binary adding machine



maximum result:

$$255 + 255 = 510$$

OR

1 1111 1110

How can we add two binary numbers?

$$\begin{array}{r} 1 1 1 0 \\ + 1 0 1 0 \\ \hline \end{array}$$

How can we add two binary numbers?

$$\begin{array}{r} 1 1 1 0 \\ + 1 0 1 0 \\ \hline 0 \end{array}$$

How can we add two binary numbers?

$$\begin{array}{r} 1 1 1 0 \\ + 1 0 1 0 \\ \hline 0 0 \end{array}$$

How can we add two binary numbers?

$$\begin{array}{r} 1 1 1 0 \\ + 1 0 \underline{1} 0 \\ \hline 0 0 \end{array}$$

How can we add two binary numbers?

$$\begin{array}{r} 1 1 1 0 \\ + 1 0 \underline{1} 0 \\ \hline 0 0 0 \end{array}$$

How can we add two binary numbers?

$$\begin{array}{r} 1 1 1 0 \\ + 1 \underline{0} \underline{1} 0 \\ \hline 0 0 0 \end{array}$$

How can we add two binary numbers?

$$\begin{array}{r} 1 1 1 0 \\ + 1 \underline{0} \underline{1} 0 \\ \hline 1 0 0 0 \end{array}$$

How can we add two binary numbers?

$$\begin{array}{r} 1 1 1 0 \\ + 1 0 1 0 \\ \hline 1 1 0 0 0 \end{array}$$

adding two bits

+	0	1
0		
1		

+	0	1
0	0	
1		

$+$	$0$	$1$
$0$	$0$	$1$
$1$		

<b>+</b>	<b>0</b>	<b>1</b>
<b>0</b>	0	1
<b>1</b>	1	

+	0	1
0	0	1
1	1	10

+	0	1
0	0	1
1	1	10

OR

+	0	1
0	00	01
1	01	10

+	0	1
0	0	1
1	1	10

OR

+	0	1
0	00	01
1	01	10

the digit on the right, we call the **sum bit**  
 the left digit, we call ...?

+	0	1
0	0	1
1	1	10

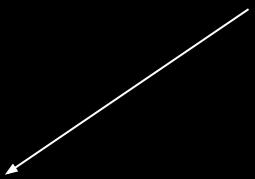
OR

+	0	1
0	00	01
1	01	10

the digit on the right, we call the **sum bit**  
the left digit, we call the **carry bit**

$+$	$0$	$1$
$0$	$0 \ 0$	$0 \ 1$
$1$	$0 \ 1$	$1 \ 0$

+	0	1
0	0 0	0 1
1	0 1	1 0



sum bit

+	0	1
0	0	1
1	1	0

+	0	1
0	0 0	0 1
1	0 1	1 0

sum bit

+	0	1
0	0	1
1	1	0

carry bit

+	0	1
0	0	0
1	0	1

+	0	1
0	0 0	0 1
1	0 1	1 0

sum bit

+	0	1
0	0	1
1	1	0

doesn't this look  
familiar?

+	0	1
0	0	0
1	0	1

carry bit

+	0	1
0	0 0	0 1
1	0 1	1 0

sum bit

+	0	1
0	0	1
1	1	0

doesn't this look  
familiar?

carry bit

+	0	1
0	0	0
1	0	1

AND	0	1
0	0	0
1	0	1

+	0	1
0	0 0	0 1
1	0 1	1 0

sum bit

+	0	1
0	0	1
1	1	0

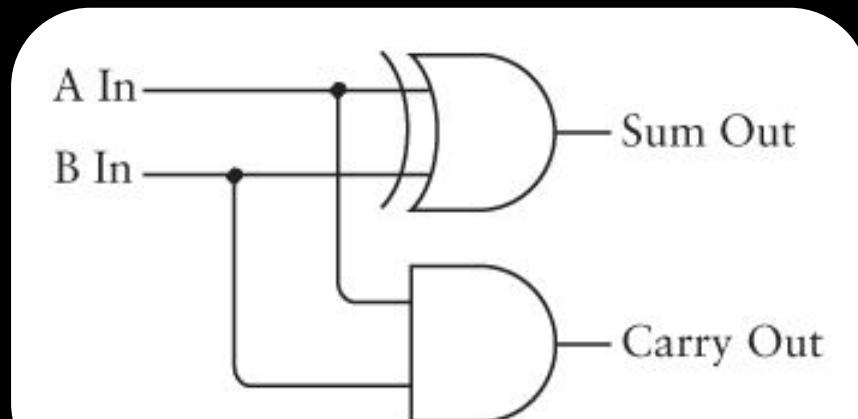
doesn't this look  
familiar?

XOR	0	1
0	0	1
1	1	0

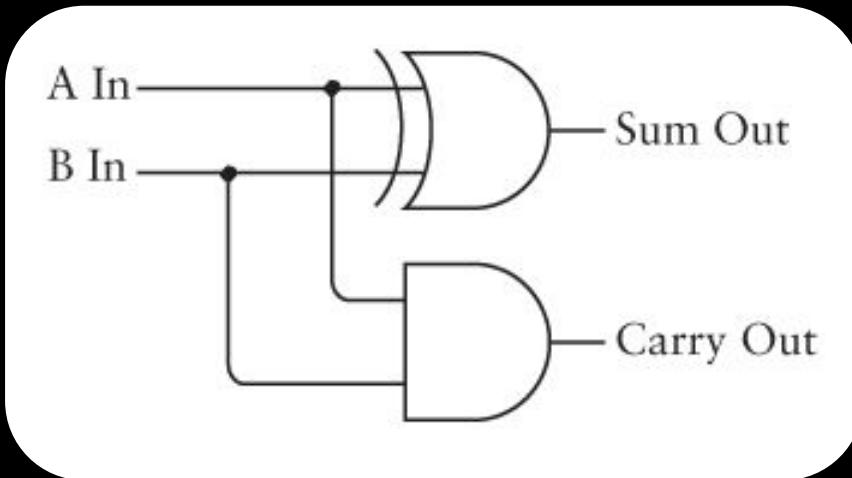
carry bit

+	0	1
0	0	0
1	0	1

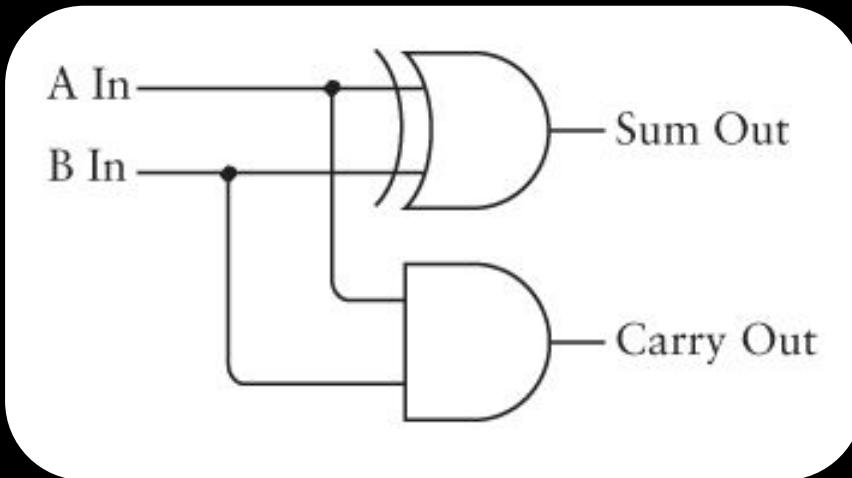
AND	0	1
0	0	0
1	0	1



bits A and B  
are inputs

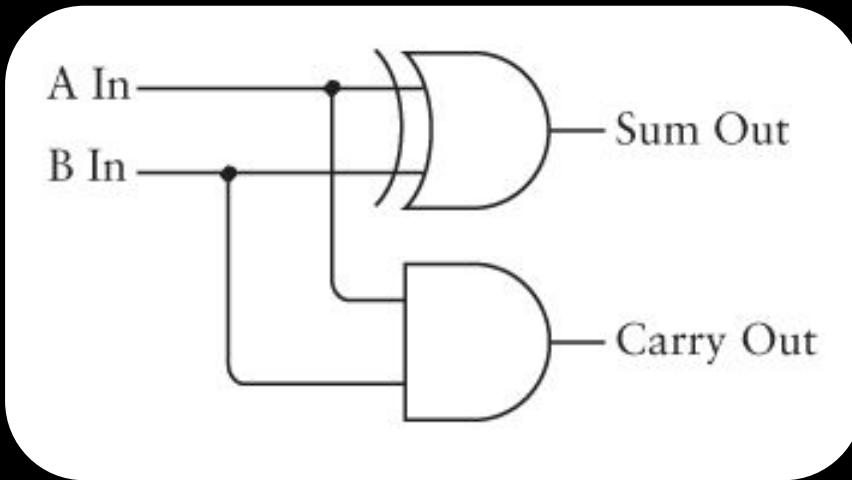


bits A and B  
are inputs



XOR gate  
computes sum bit

bits A and B  
are inputs

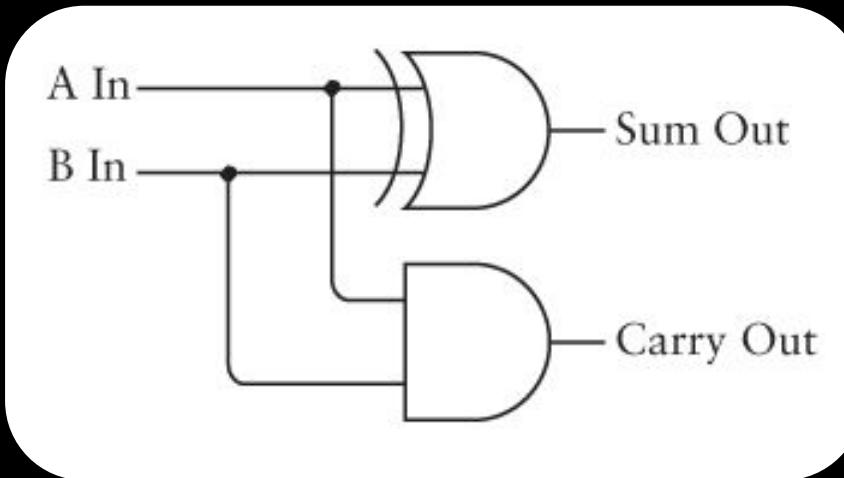


XOR gate  
computes sum bit

AND gate  
computes carry bit

## half adder

bits A and B  
are inputs

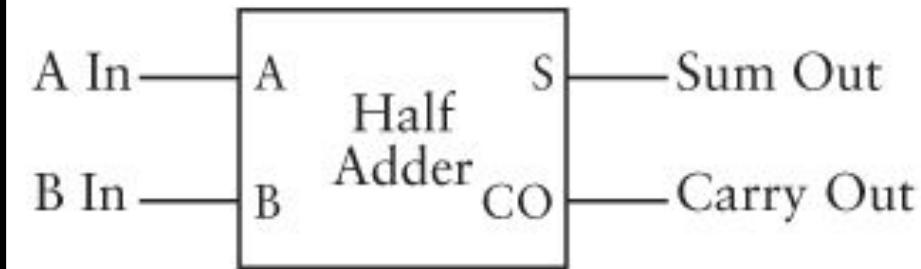


XOR gate  
computes sum bit

AND gate  
computes carry bit

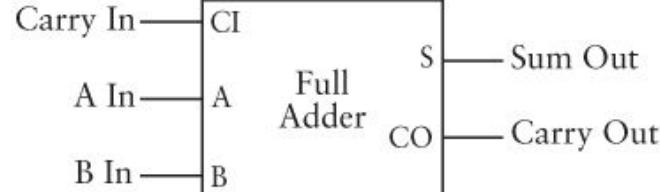
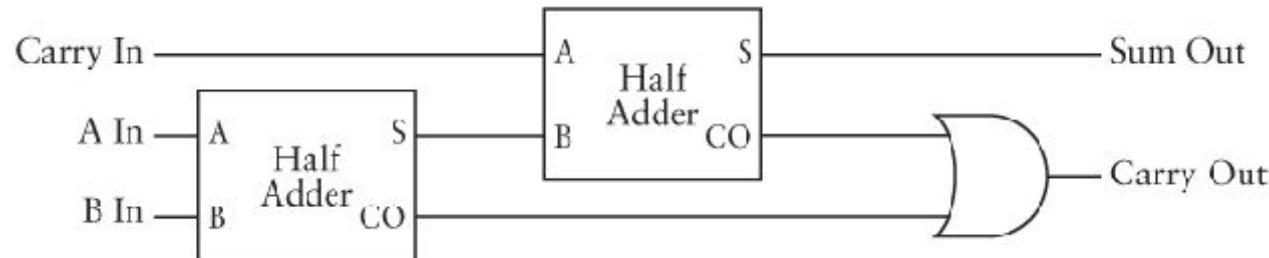


## half adder

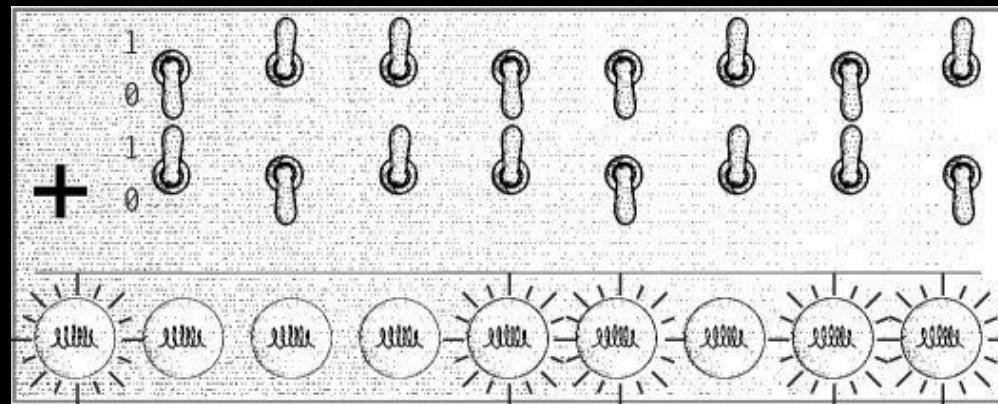


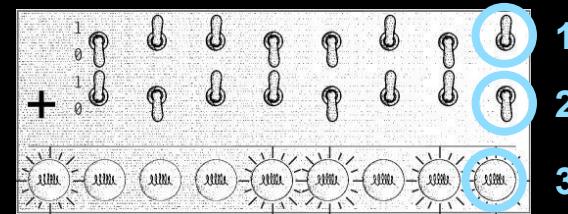
the half adder has no input for a carry bit.

two half adder wired together make a full adder.



so how to build this?



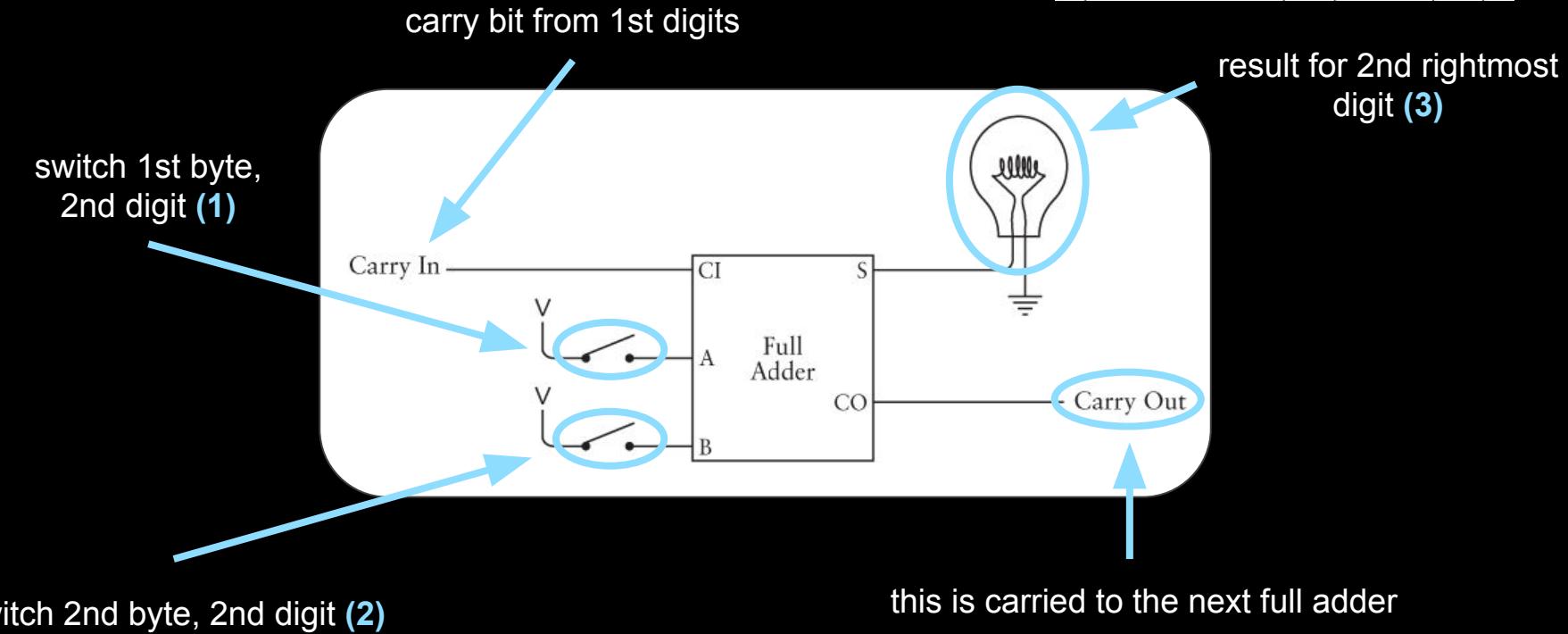
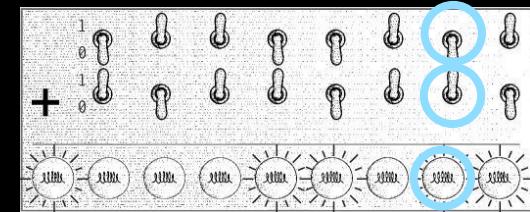


switch 1st byte,  
1st digit (1)

result for rightmost digit (3)

switch 2nd byte, 1st digit (2)

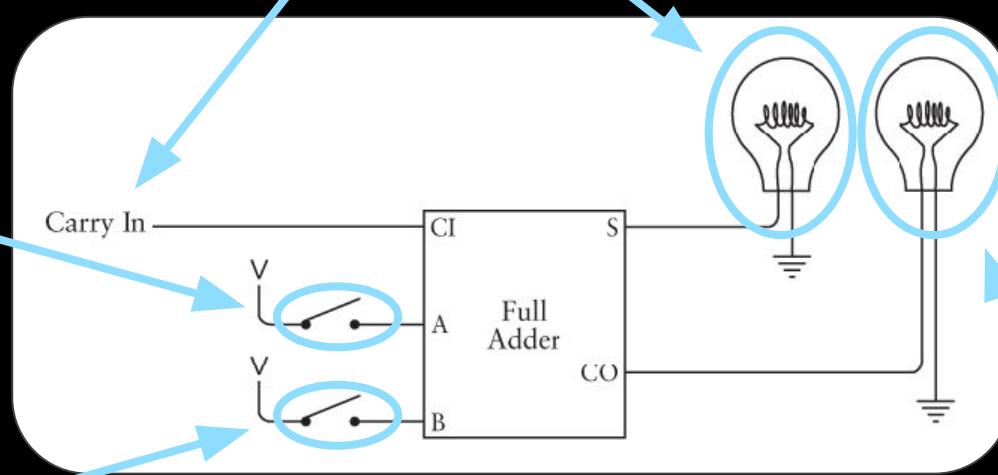
this is carried to the next full adder



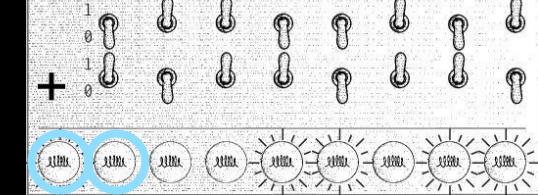
switch 1st byte,  
8th digit (1)

result for 8th digits (3)

carry bit from 7th digits

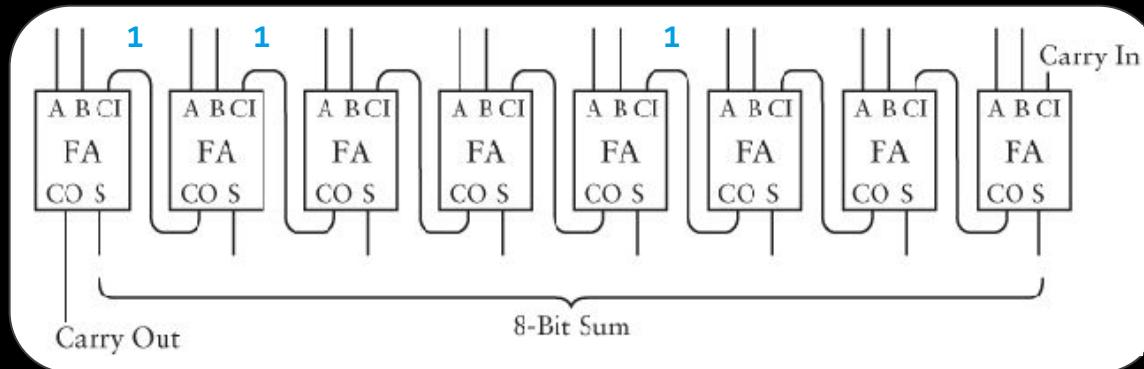


switch 2nd byte, 8th digit (2)



result for the 9th digit (4)

0	1	1	0	0	1	0	1
1	0	1	1	0	1	1	0



1	0	0	0	1	1	0	1	1
---	---	---	---	---	---	---	---	---