Introduction to Computer Architecture: How Computers Add

Fair Use Building and Research
http://fubarlabs.org

Introductions!

What is your name?

What brings you here?

Logic Basics: And, Or, Not

It's raining AND it's cloudy

It's sunny OR it's cloudy

It's sunny, NOT cloudy

It's sunny, AND it's NOT raining

Boolean Algebra

http://en.wikipedia.org/wiki/Boolean_algebra

... Boolean algebra is the algebra of truth values 0 and 1...

0 and 1

Yes and No

True and False

+ and -

Red and Black (or Blue)

High and Low

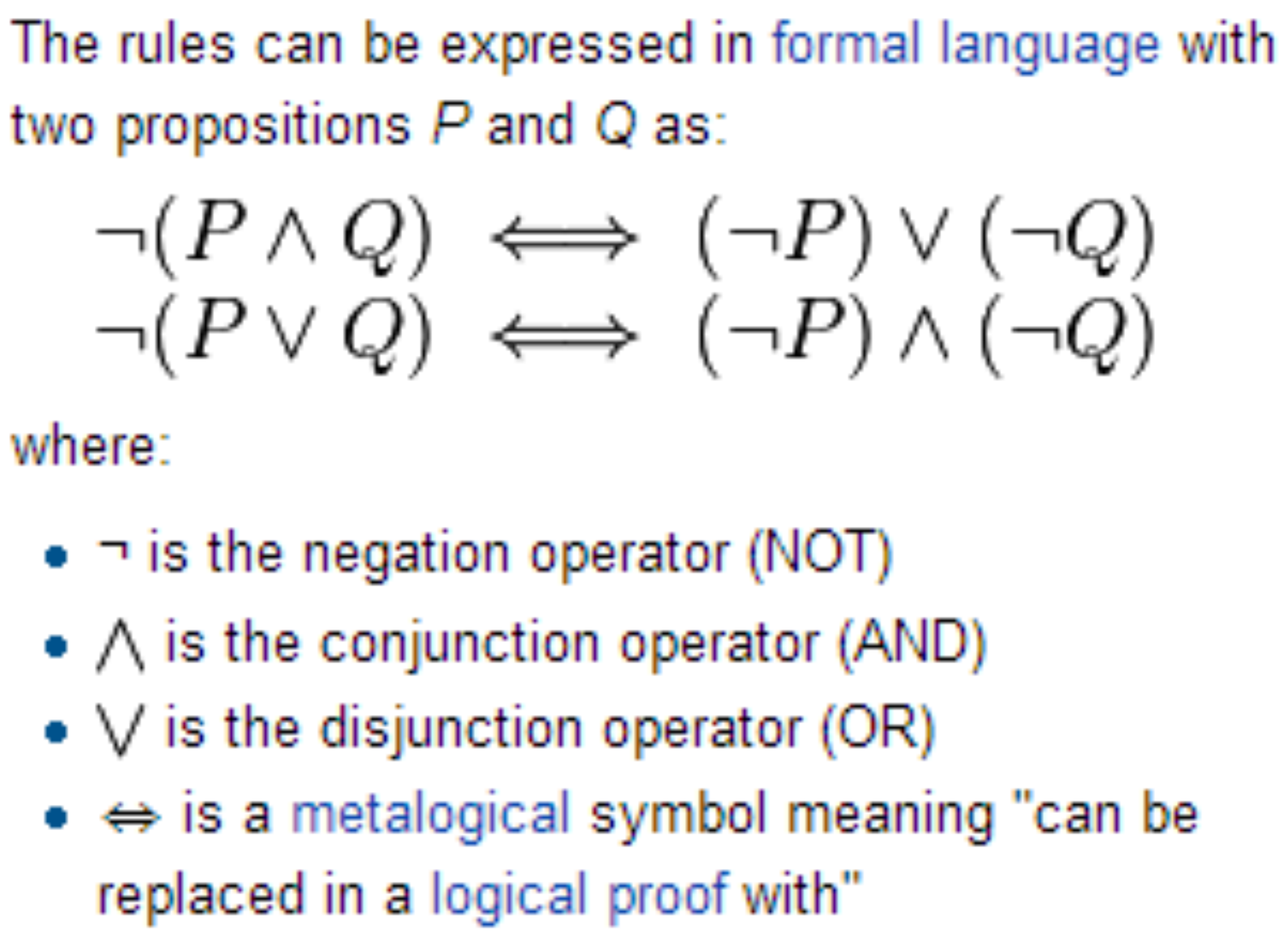
Boolean Algebra Conventions

(one of many!)

A + B	A or B
A.B ... or just AB	A and B
A'	Not A
(A + B)'	Not (A or B)
(AB)'	Not (A and B)
A'B'	(Not A) and (Not B)
(Sunny).(Raining')	Sunny and Not Raining

Another boolean convention

From: http://en.wikipedia.org/wiki/De_Morgan's_laws



The rules can be expressed in formal language with two propositions *P* and *Q* as:

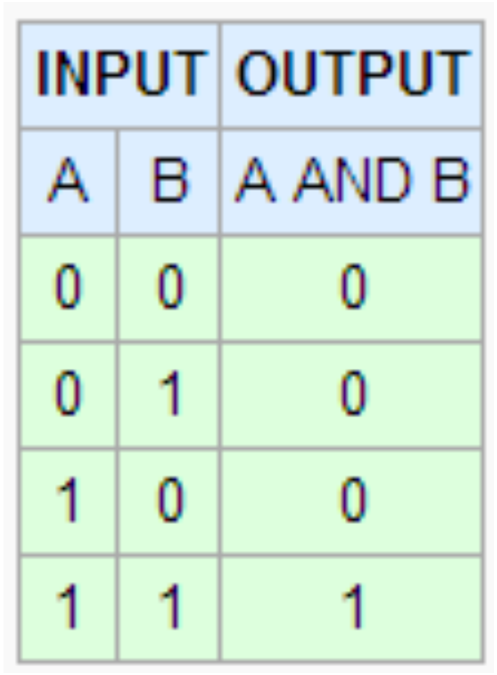
$$\neg(P \wedge Q) \iff (\neg P) \vee (\neg Q)$$
$$\neg(P \vee Q) \iff (\neg P) \wedge (\neg Q)$$

where:

- $\neg$ is the negation operator (NOT)
- $\wedge$ is the conjunction operator (AND)
- $\vee$ is the disjunction operator (OR)
- $\Leftrightarrow$ is a metalogical symbol meaning "can be replaced in a logical proof with"

Truth Tables

AND:



INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Truth Tables - (AB)' vs. (A'B')

A	B	A and B	Not (A and B)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

A	B	A'	B'	A' and B'
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Laws and Theorems of Boolean Algebra

from: http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/index.html

T1 : Commutative Law

(a) $A + B = B + A$

(b) $A\ B = B\ A$

T2 : Associate Law

(a) $(A + B) + C = A + (B + C)$

(b) $(A\ B)\ C = A\ (B\ C)$

T3 : Distributive Law

(a) $A\ (B + C) = A\ B + A\ C$

(b) $A + (B\ C) = (A + B)\ (A + C)$

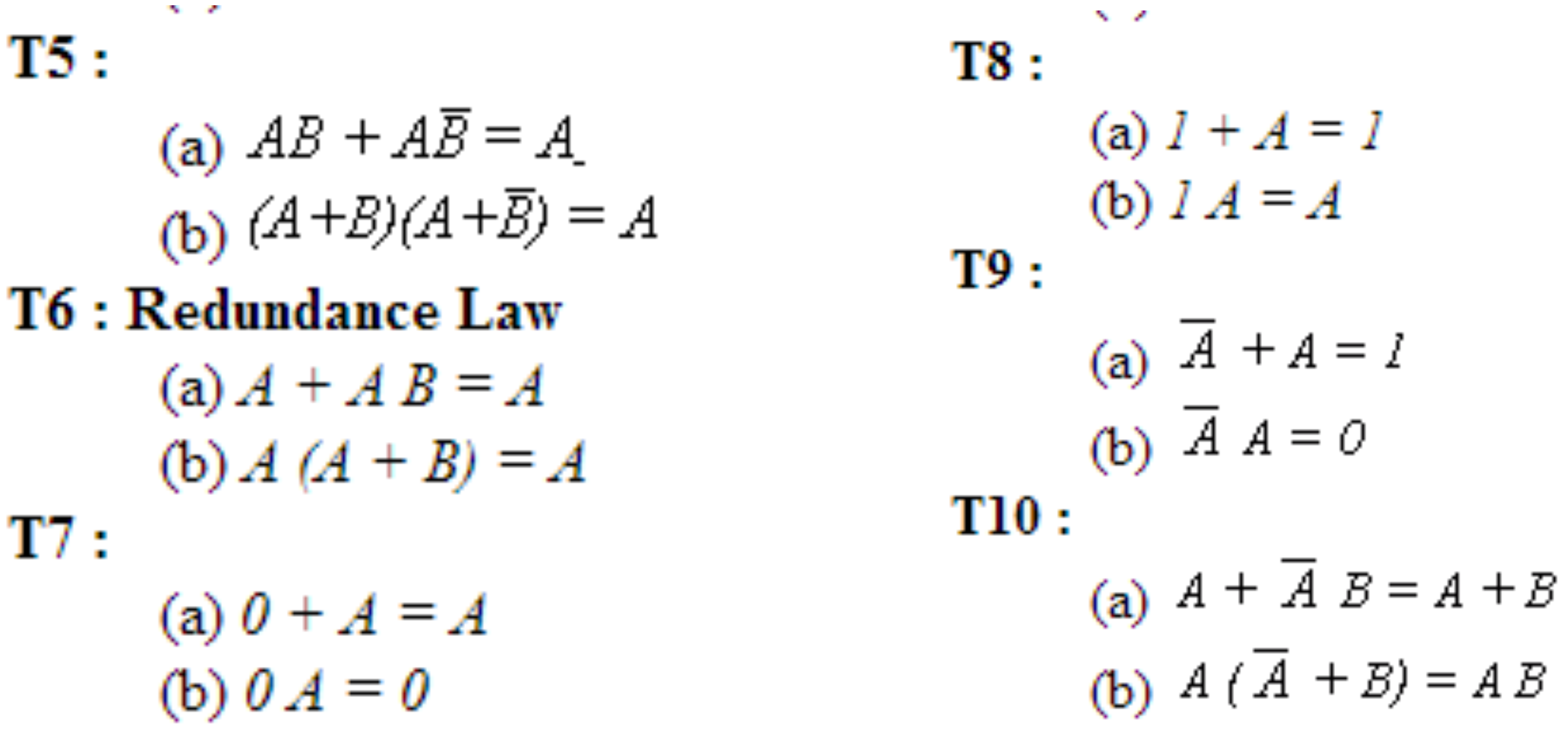
T4 : Identity Law

(a) $A + A = A$

(b) $A\ A = A$

... More laws, etc.

from: http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/index.html



T5 :

(a) $AB + A\overline{B} = A$

(b) $(A+B)(A+\overline{B}) = A$

T6 : Redundance Law

(a) $A + A\,B = A$

(b) $A\,(A + B) = A$

T7 :

(a) $0 + A = A$

(b) $0\,A = 0$

T8 :

(a) $1 + A = 1$

(b) $1\,A = A$

T9 :

(a) $\overline{A} + A = 1$

(b) $\overline{A}\,A = 0$

T10 :

(a) $A + \overline{A}\,B = A + B$

(b) $A\,(\overline{A} + B) = A\,B$

De Morgan's Law

$(AB)' = A' + B'$

$(A + B)' = A'B'$

Proofs with Truth Tables

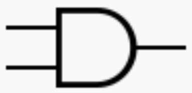
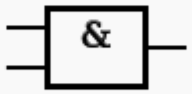
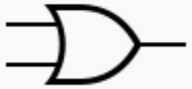
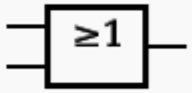
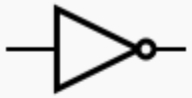
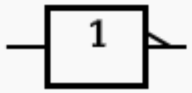
Prove T10(a):

A + A'B = A + B

A	B	A'	A + B	A'B	A + A'B
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	1	0	1
1	1	0	1	0	1

Basic Logic Gates

from: http://en.wikipedia.org/wiki/Logic_gate

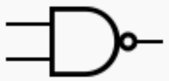
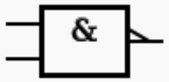
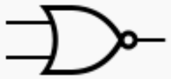
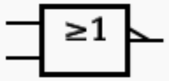
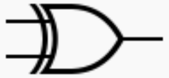
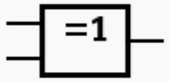
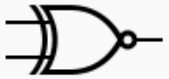
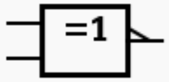
Type	Distinctive shape	Rectangular shape	Boolean algebra between A & B	Truth table
AND			$A \cdot B$	
OR			$A + B$	
NOT			$\overline{A}$	

INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

INPUT	OUTPUT
A	NOT A
0	1
1	0

NAND, etc.

NAND			$\overline{A \cdot B}$	see truth table below
NOR			$\overline{A + B}$	see truth table below
XOR			$A \oplus B$	see truth table below
XNOR			$\overline{A \oplus B}$ or $A \odot B$	see truth table below

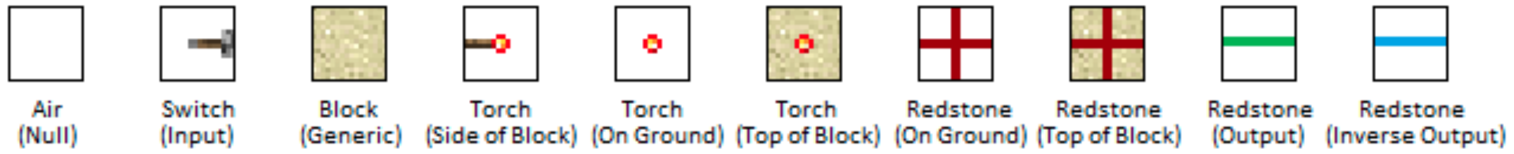
INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

INPUT		OUTPUT
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

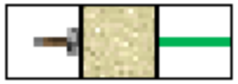
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

INPUT		OUTPUT
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

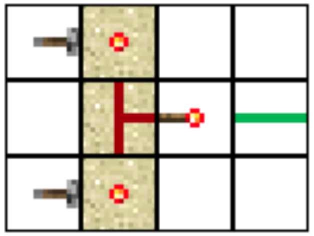
MineCraft Logic Gates



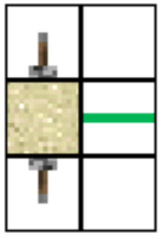
Input/Output Gate



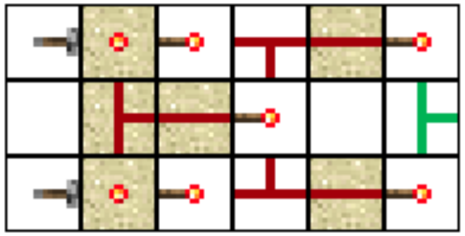
AND Gate



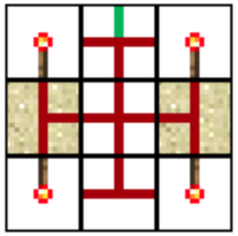
OR Gate



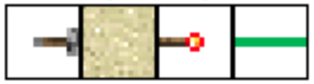
XOR Gate



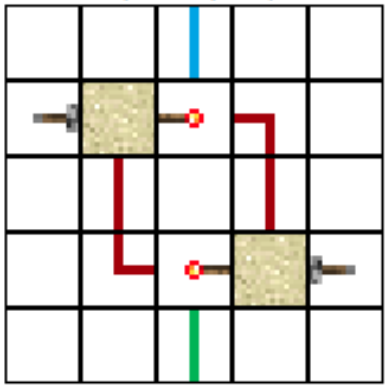
Rapid Pulser



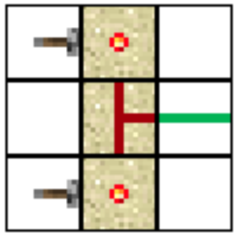
NOT Gate
(Inverter)



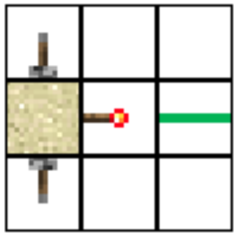
RS NOR Latch
(Memory Cell)



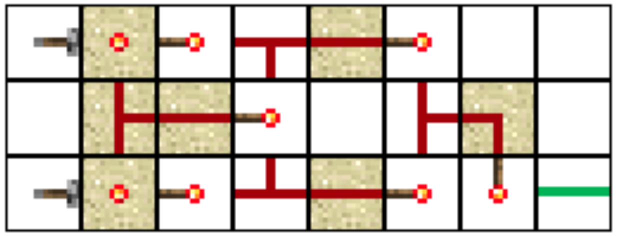
NAND Gate



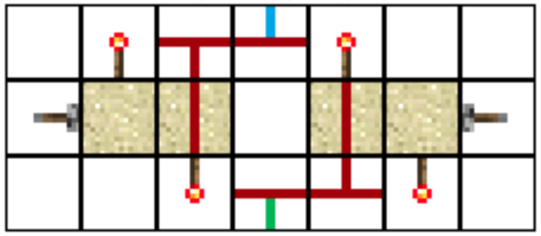
NOR Gate



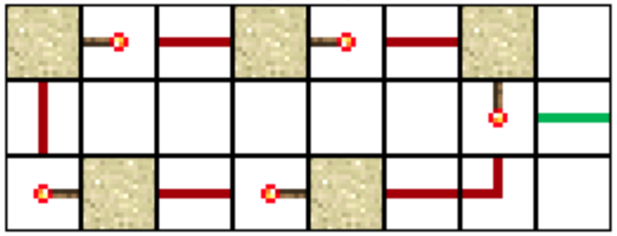
XNOR Gate



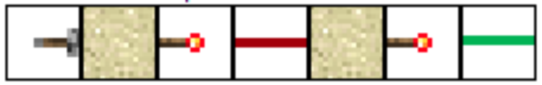
RS NAND Latch
(Memory Cell)



5-Clock
(Pulser)



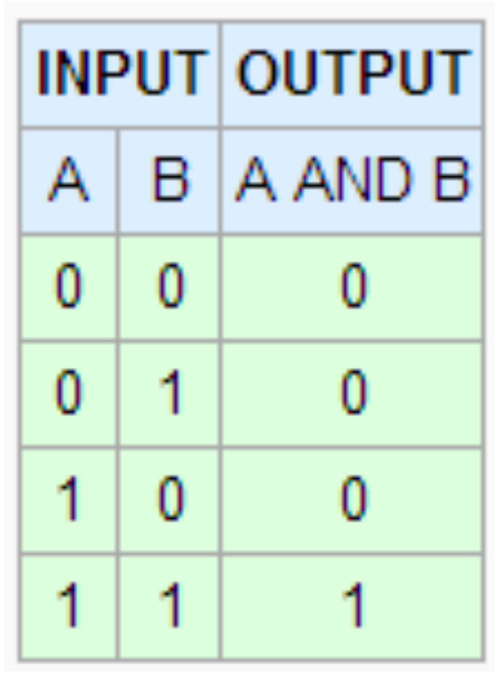
Repeater



NANDs are "universal gates"

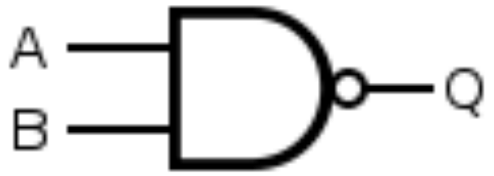
So are NORs -- http://en.wikipedia.org/wiki/NAND_logic

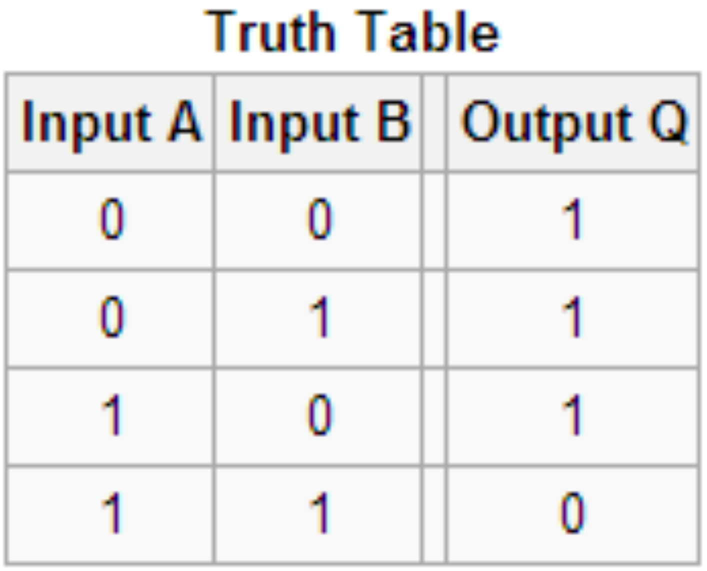
AND from earlier:



INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

NAND

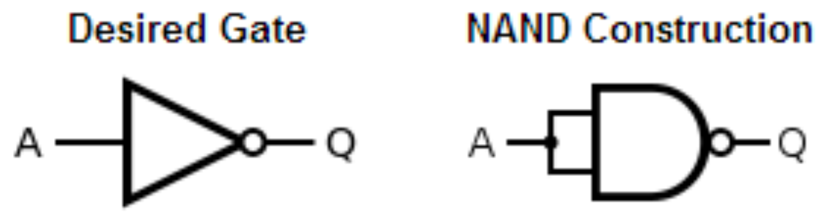




Truth Table

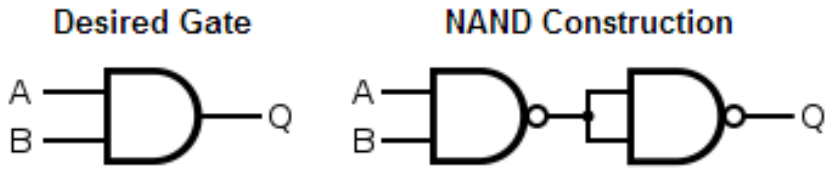
Input A	Input B	Output Q
0	0	1
0	1	1
1	0	1
1	1	0

More NAND Gates



Truth Table

Input A	Output Q
0	1
1	0

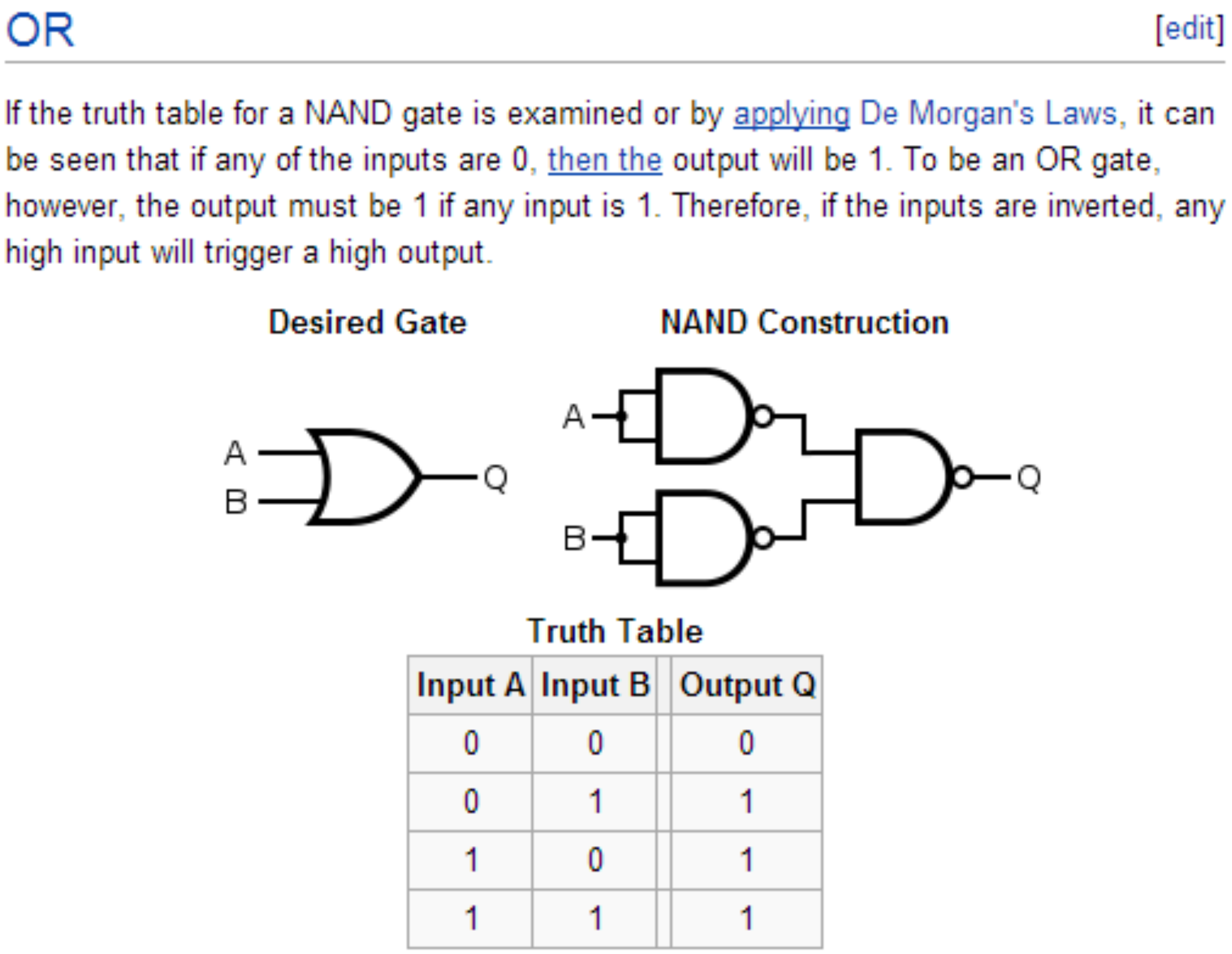


Truth Table

Input A	Input B	Output Q
0	0	0
0	1	0
1	0	0
1	1	1

$$F = (F')'$$

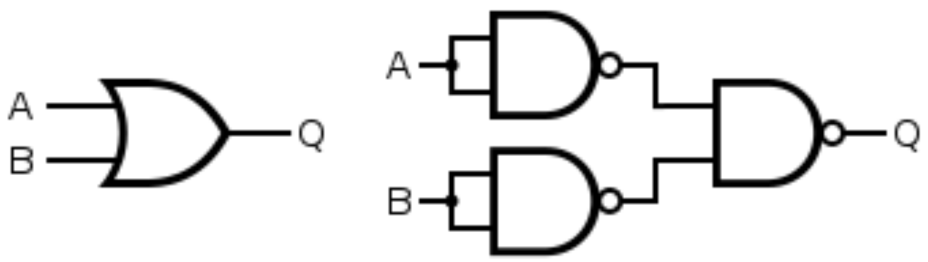
More NAND Gates



OR [edit]

If the truth table for a NAND gate is examined or by applying De Morgan's Laws, it can be seen that if any of the inputs are 0, then the output will be 1. To be an OR gate, however, the output must be 1 if any input is 1. Therefore, if the inputs are inverted, any high input will trigger a high output.

Desired Gate **NAND Construction**



Truth Table

Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	1

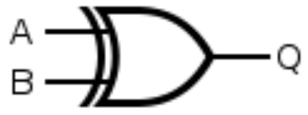
More NAND Gates

XOR

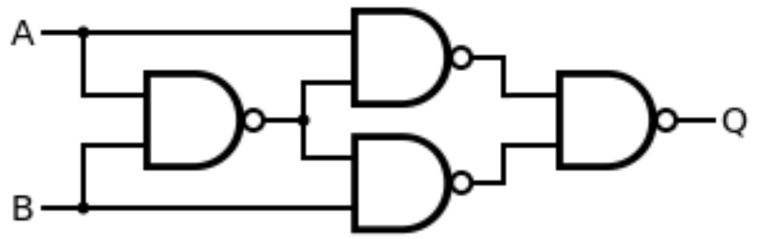
[edit]

An XOR gate is constructed similarly to an OR gate, except with an additional NAND gate inserted such that if both inputs are high, the inputs to the final NAND gate will also be high, and the output will be low. This effectively represents the formula: "NAND(A NAND (A NAND B)) NAND (B NAND (A NAND B))".

Desired Gate



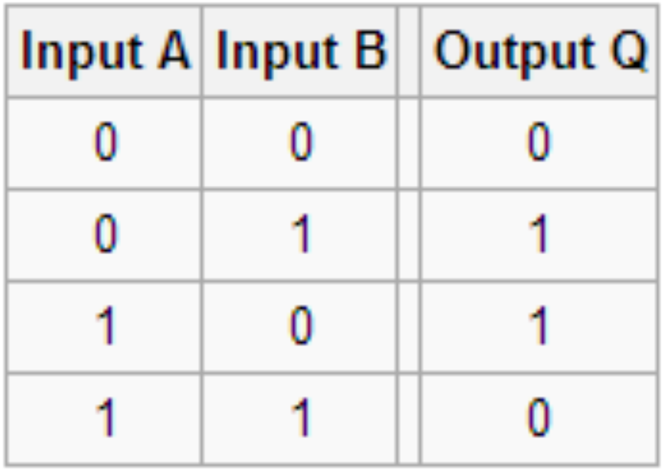
NAND Construction



Truth Table

Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	0

XOR - Truth Table to Gates

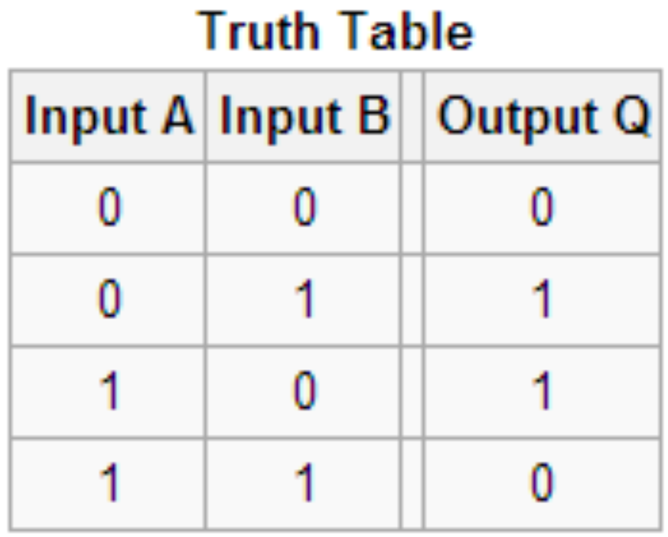

Truth Table

Input A	Input B		Output Q
0	0		0
0	1		1
1	0		1
1	1		0

WIKI Says:
 This effectively represents the formula:
NAND(A NAND (A NAND B)) NAND (B NAND (A NAND B))

OR:
((A.(AB)')'.(B.(A.B)')')

Karnaugh Maps



Truth Table

Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	0

A \ B	0	1
0		
1		

Karnaugh Maps - XOR

A \ B	0	1
0	0	1
1	1	0

Sum of Products: SoP

-- Work with the ones, OR them together

Product of Sums: PoS

-- Work with the zeros, AND them together

Karnaugh Maps - XOR

A \ B	0	1
0	0	1
1	1	0

Sum of Products: SoP (1s ORd)

F = (A'B) + (AB')

Product of Sums: PoS (0s ANDed)

F = (A' + B').(A + B)

Drawing XOR with Gates

$F = (A' + B').(A + B)$

Drawing XOR with NANDs

Simplifying XOR

$F = (A' + B').(A + B)$

First: De Morgan's Law!

$(AB)' = A' + B'$ ... $F = (AB)'.(A + B)$

...then **T3 : Distributive Law**

(a) $A (B + C) = A B + A C$

Simplifying XOR

F = (AB)'.(A + B)

...then **T3 : Distributive Law**

(a) *X (Y + Z) = XY + X Z*

F = (AB)'(A) + (AB)'(B)

F' = ((AB)'(A) + (AB)'(B))'

...then remember, F = (F')'

... and De Morgan's (other) Law (A + B)' = A'B'

Simplifying XOR

$F = (AB)'(A) + (AB)'(B)$

...then remember, $F = (F')'$

... and De Morgan's (other) Law $(A + B)' = A'B'$

$F'' = F = ((AB)'(A) + (AB)'(B))''$

$F = ((A(AB)')'.(B.(AB)')')'$

Simplifying XOR

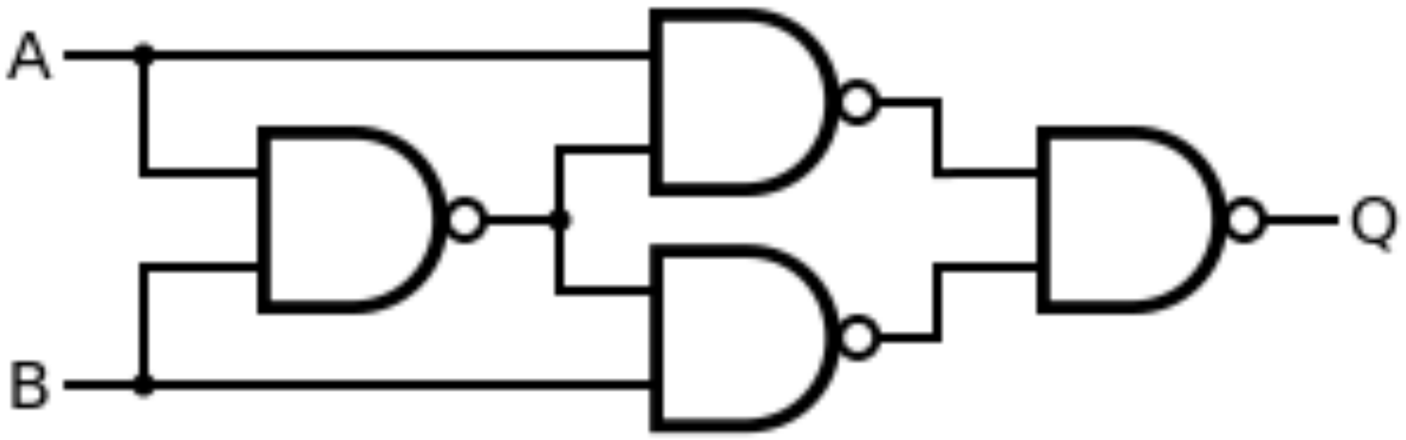
$F'' = F = ((AB)'(A) + (AB)'(B))''$

$F = ((A(AB)')'.(B.(AB)')')'$

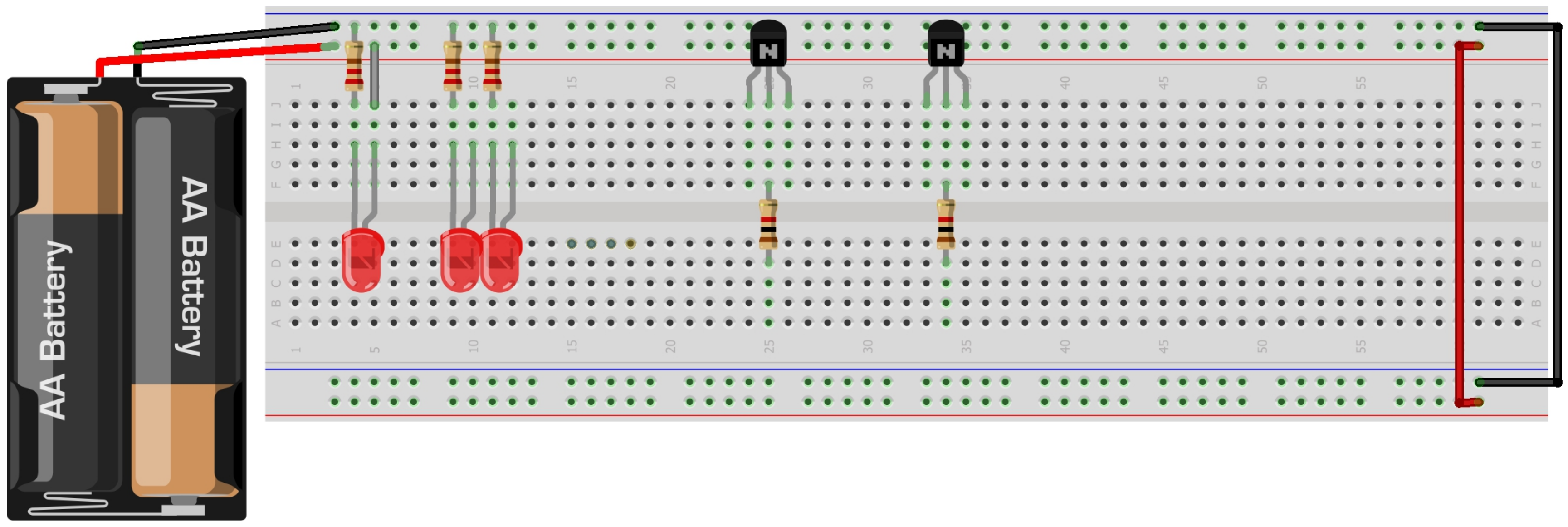
This effectively represents the formula: "NAND (A NAND (A NAND B)) NAND (B NAND (A NAND B))".

Simplifying XOR - Draw it

NAND Construction

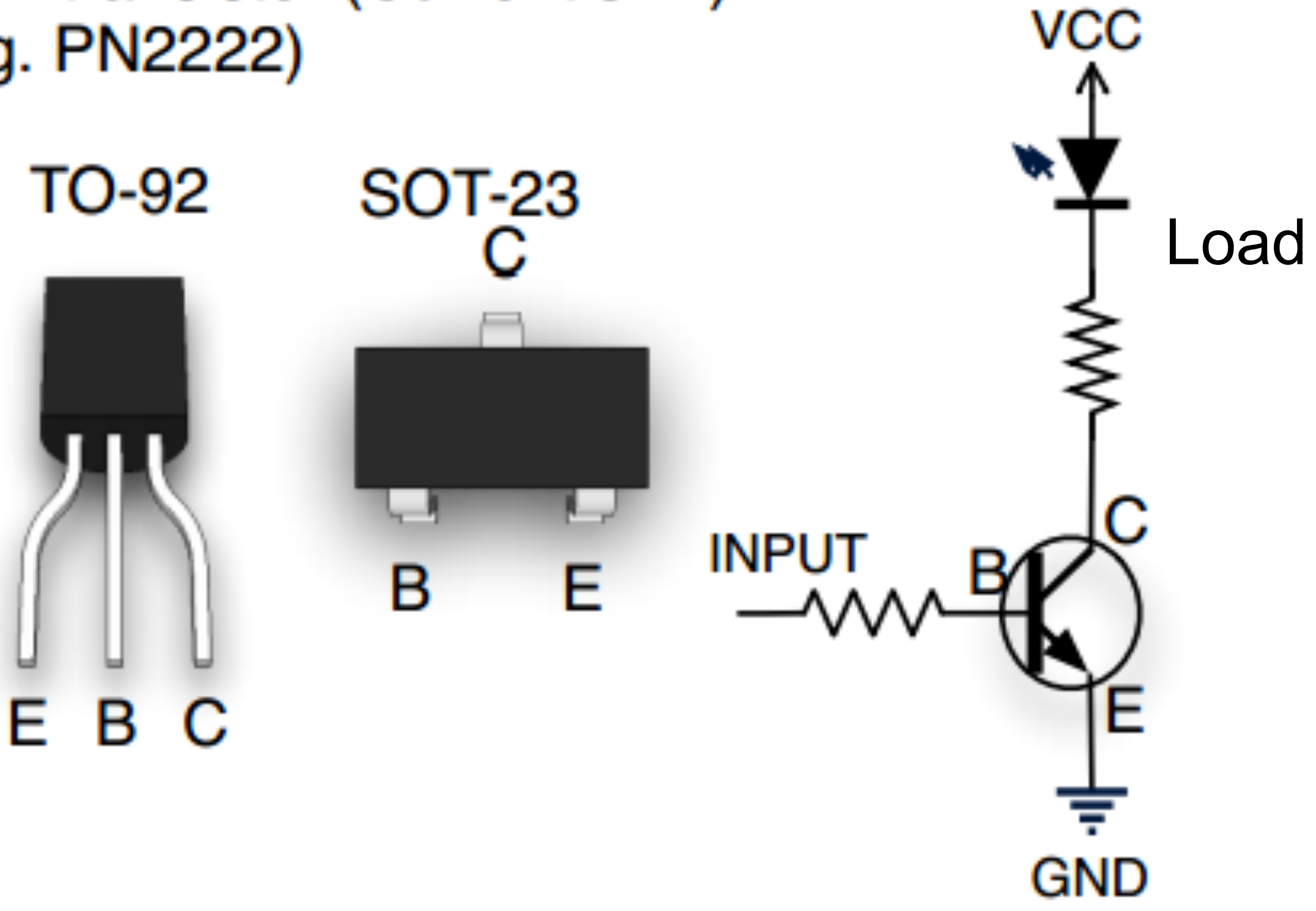


Setting up the breadboard

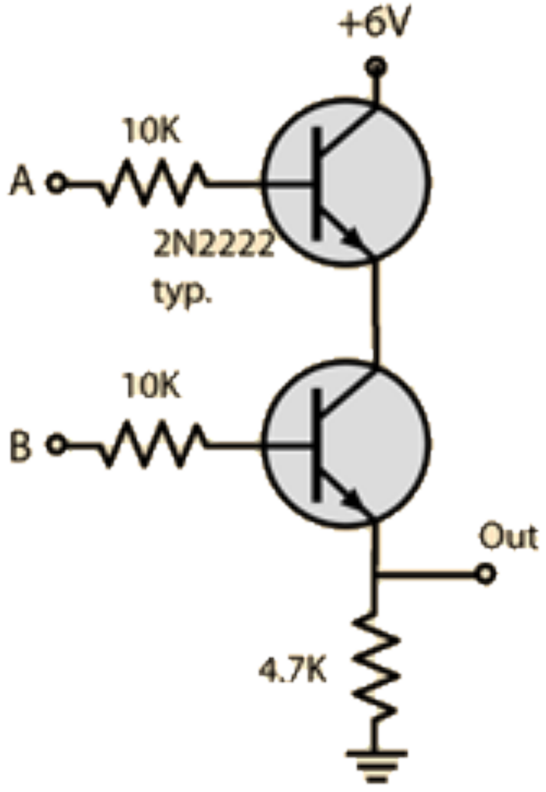


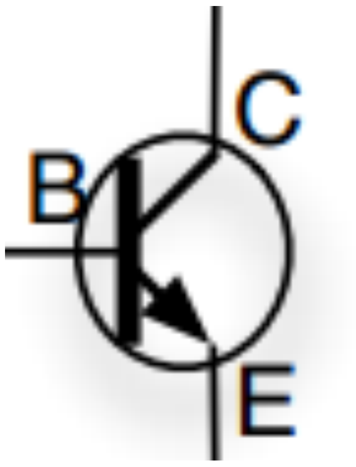
Transistors

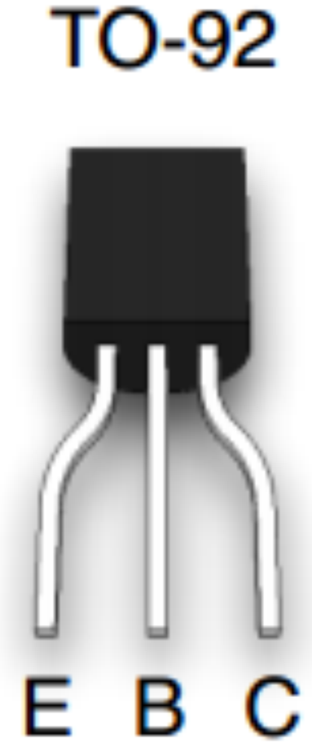
NPN transistor (Current sink)
(e.g. PN2222)



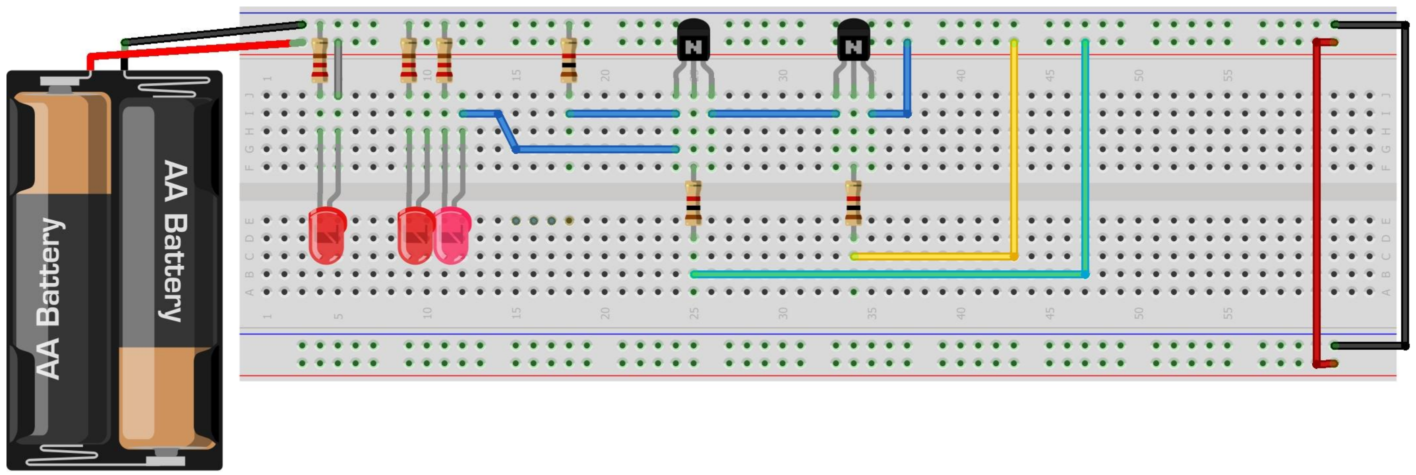
Transistor Gate - AND

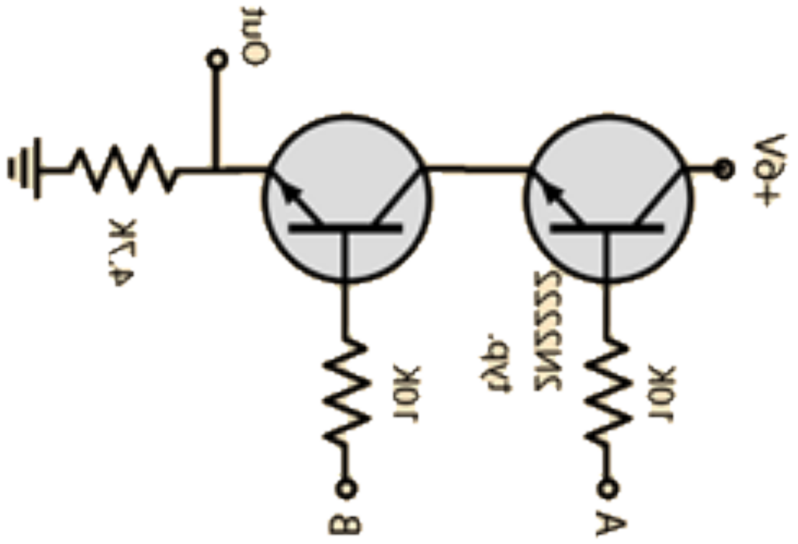




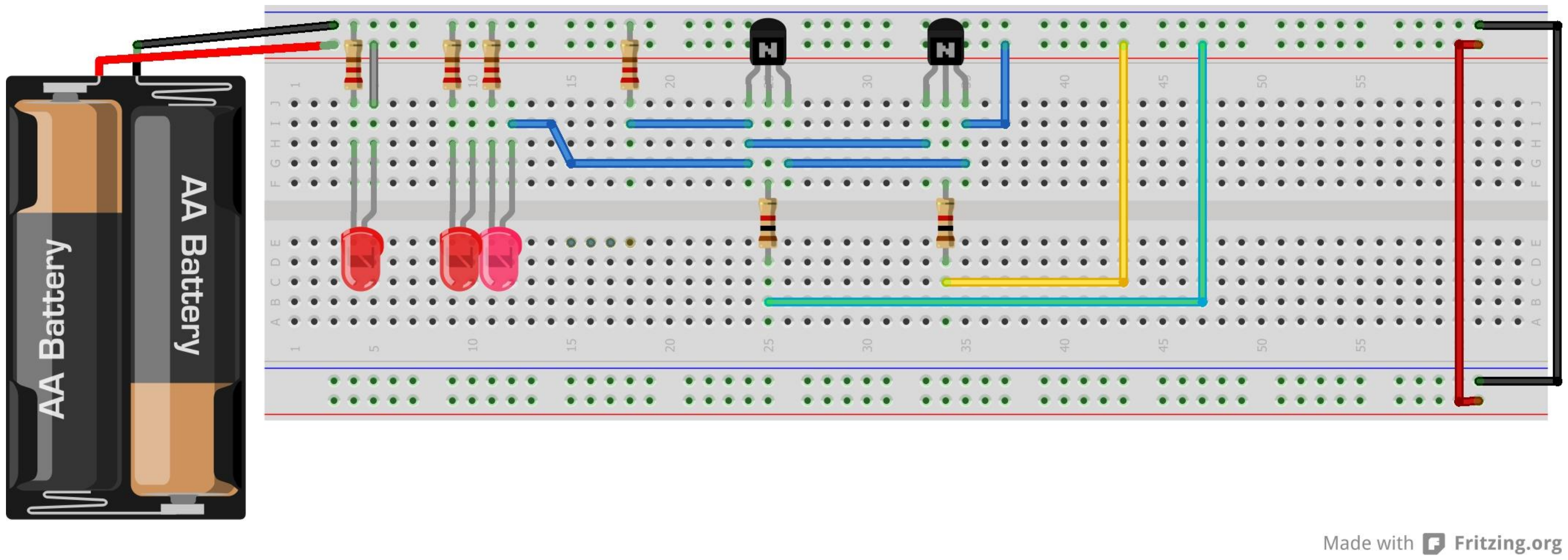


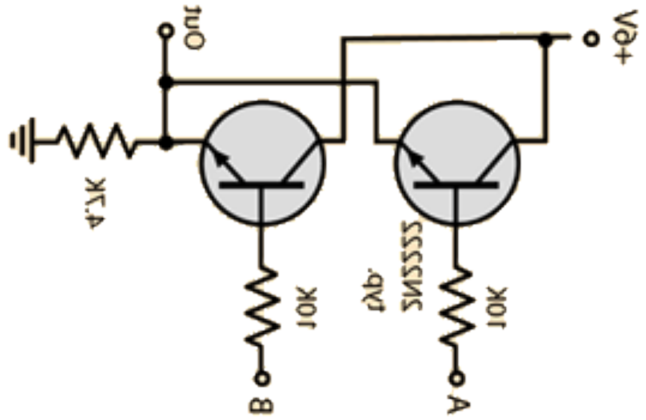
Let's build an AND



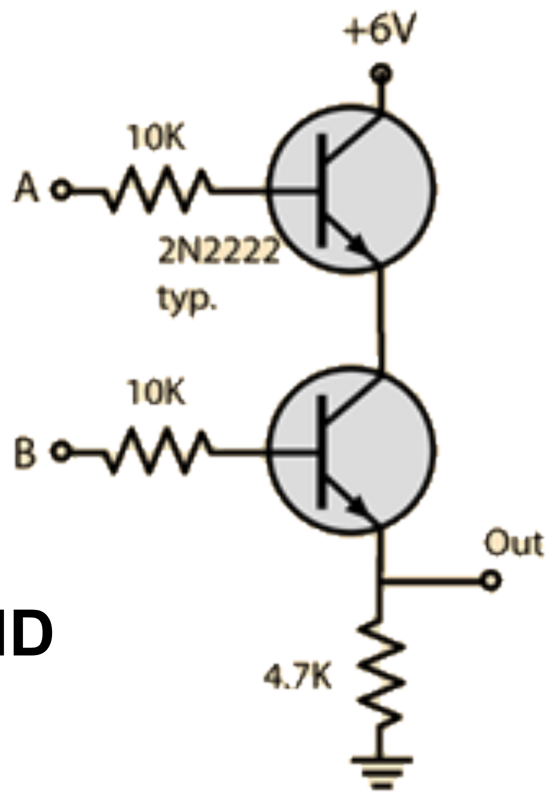


Let's build an OR

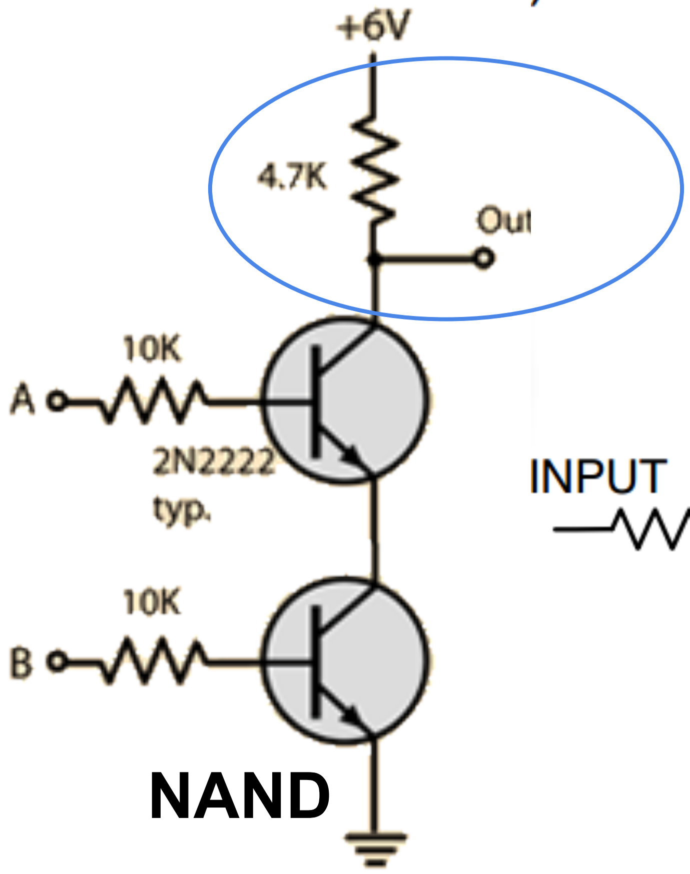




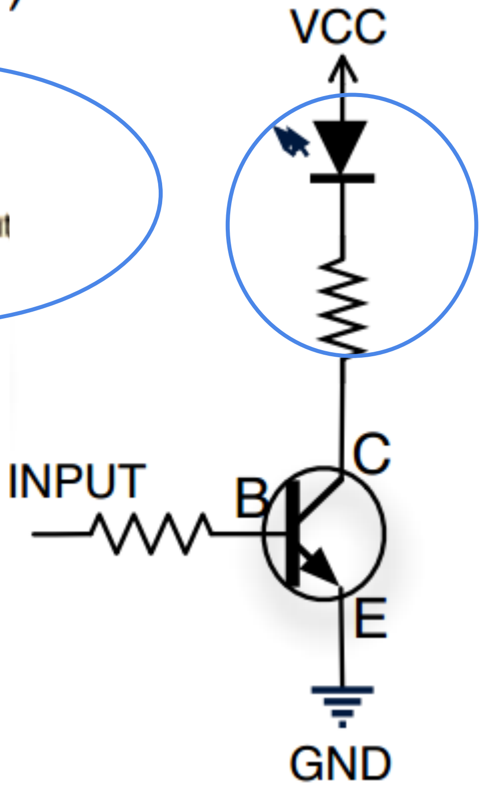
NANDs -- why they are common



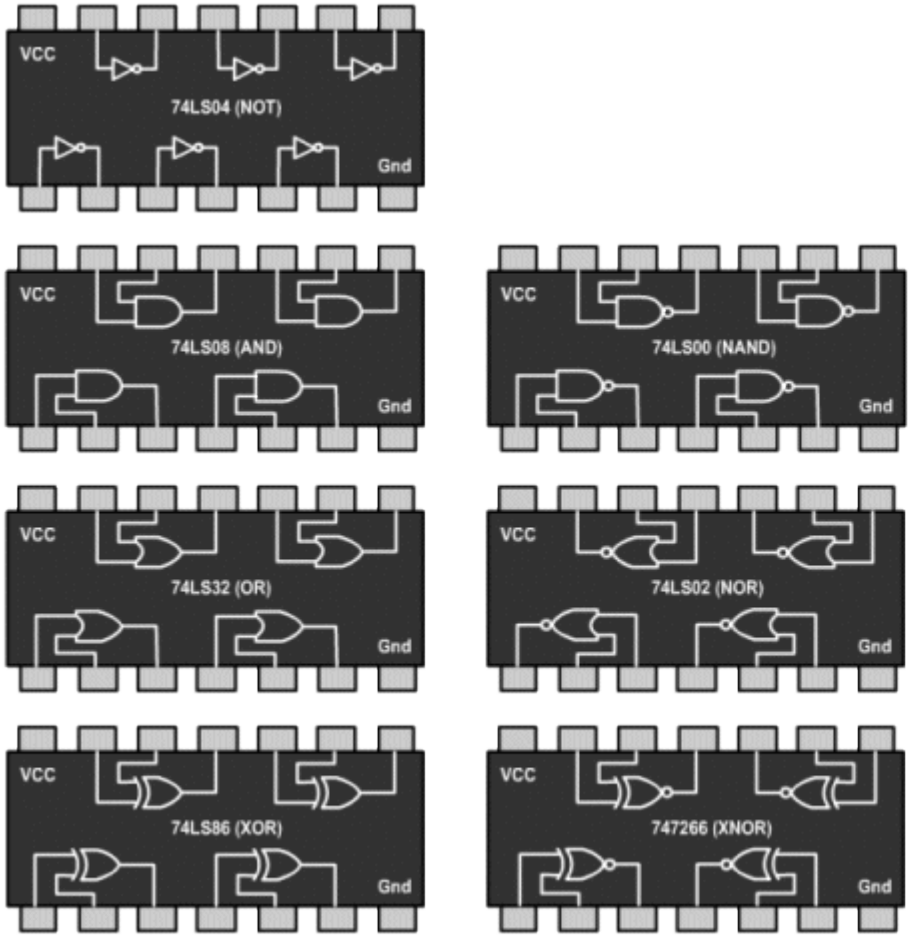
AND



NAND



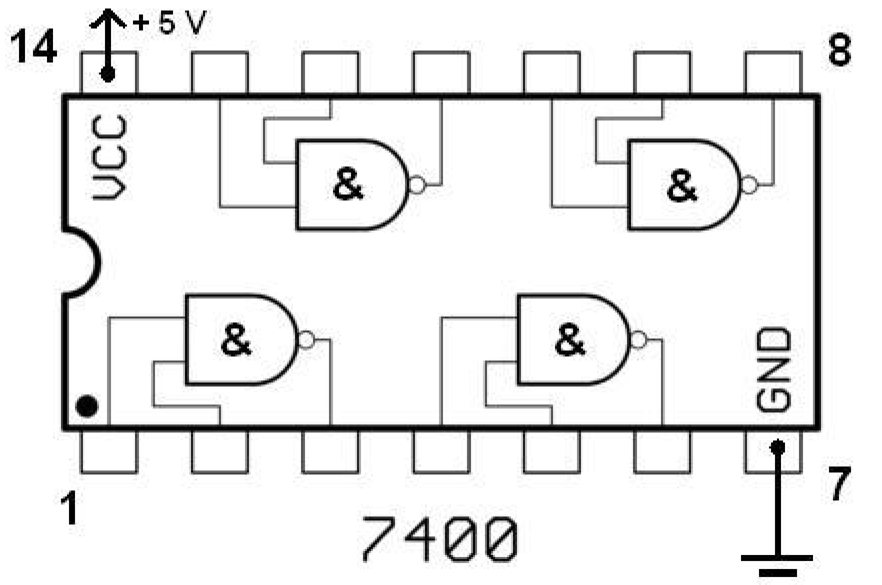
Presenting: The 7400 Series

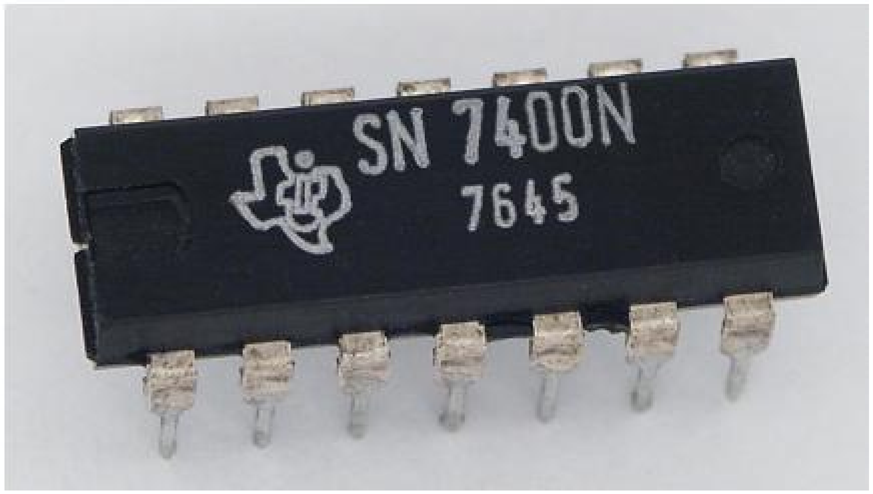


Note the "backwards" orientation of all four gates on the 74LS02 chip

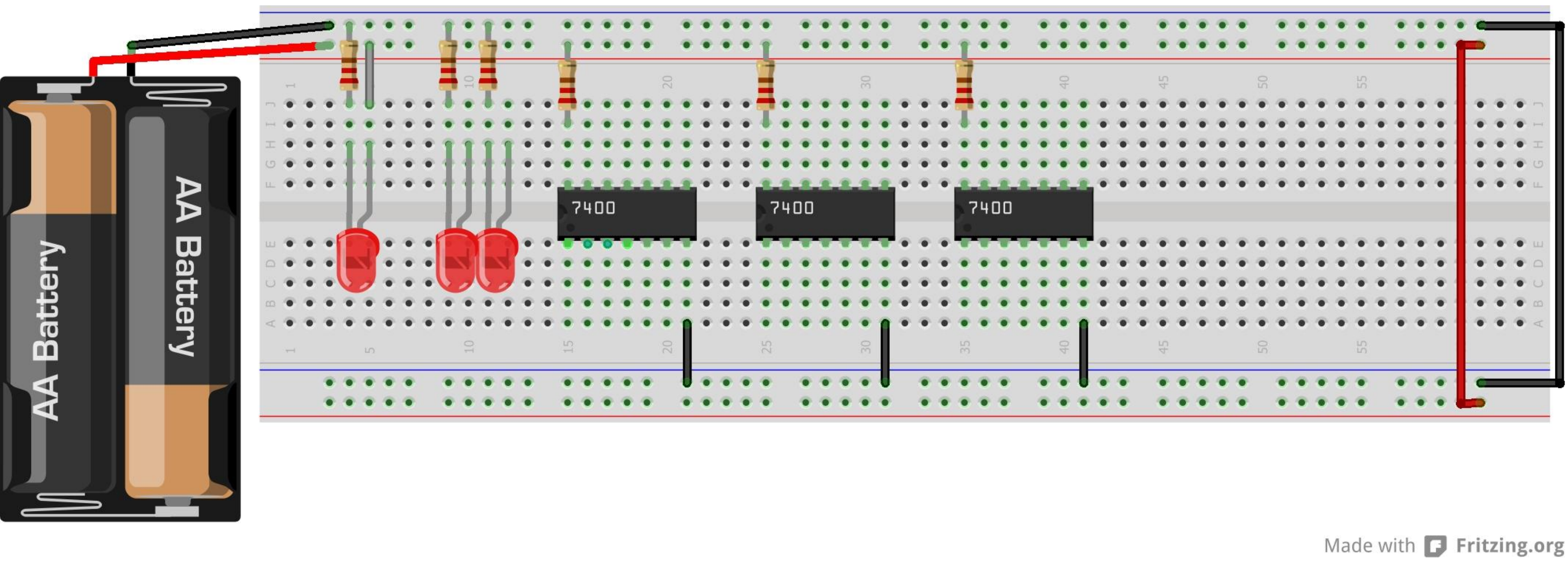
Note the "backwards" orientation of two of the four gates on the 747266 chip

7400 - Quad, dual input NAND gate

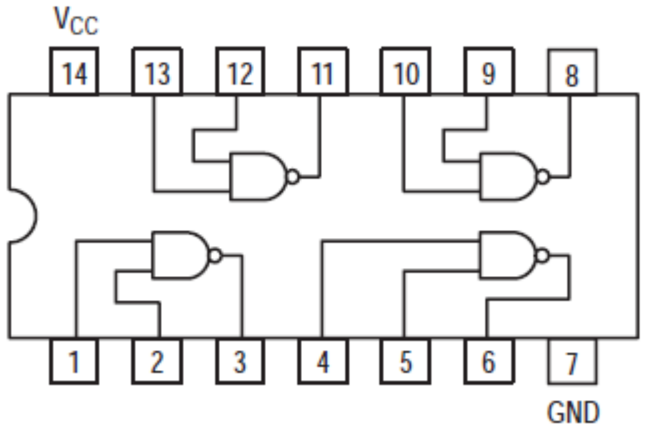


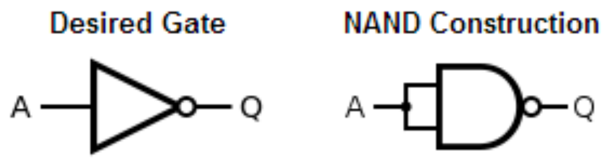


Set up the breadboard



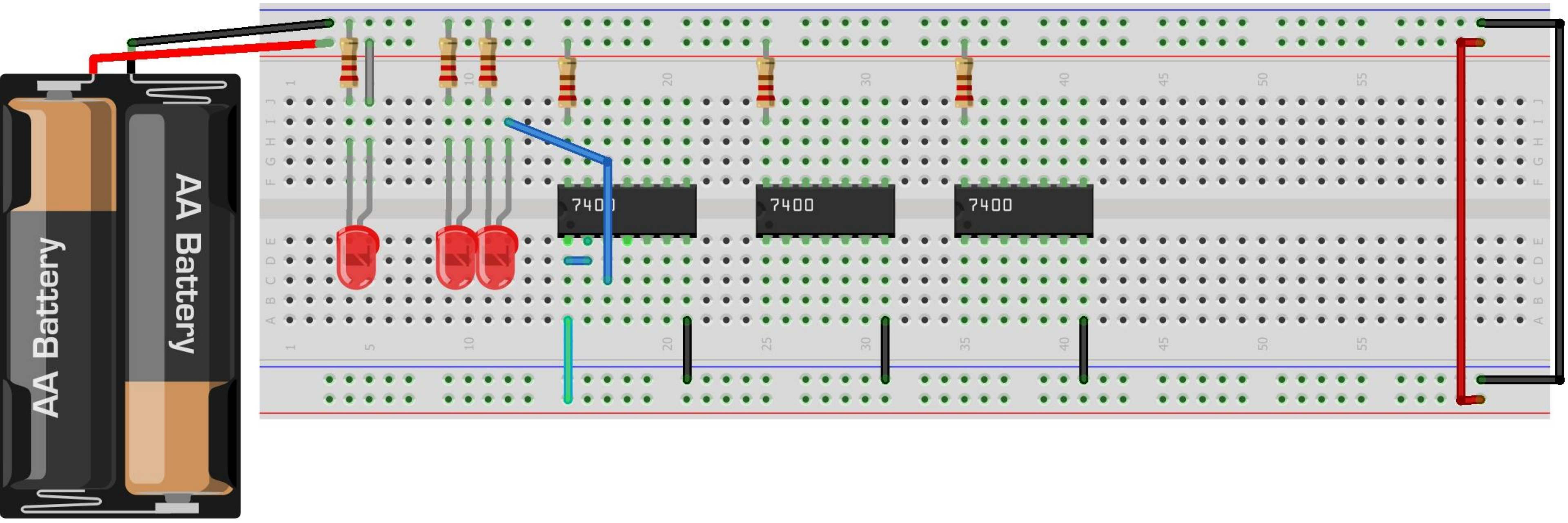
Build a NOT from NAND





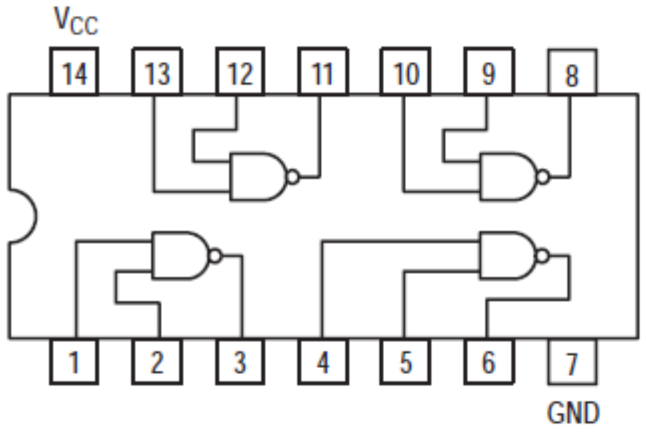
Truth Table

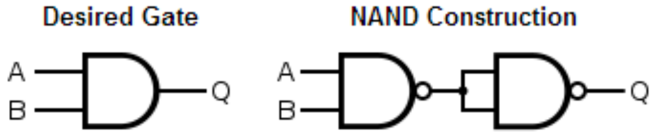
Input A	Output Q
0	1
1	0



Made with Fritzing.org

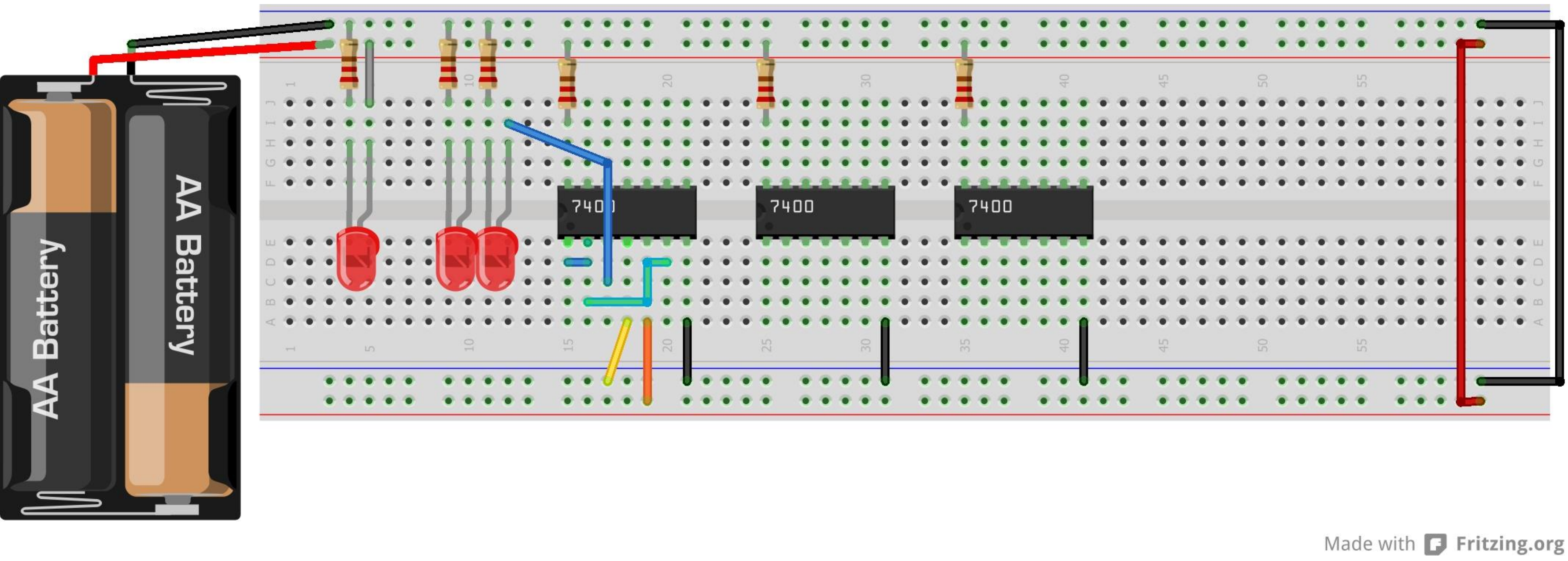
Build an AND from NAND





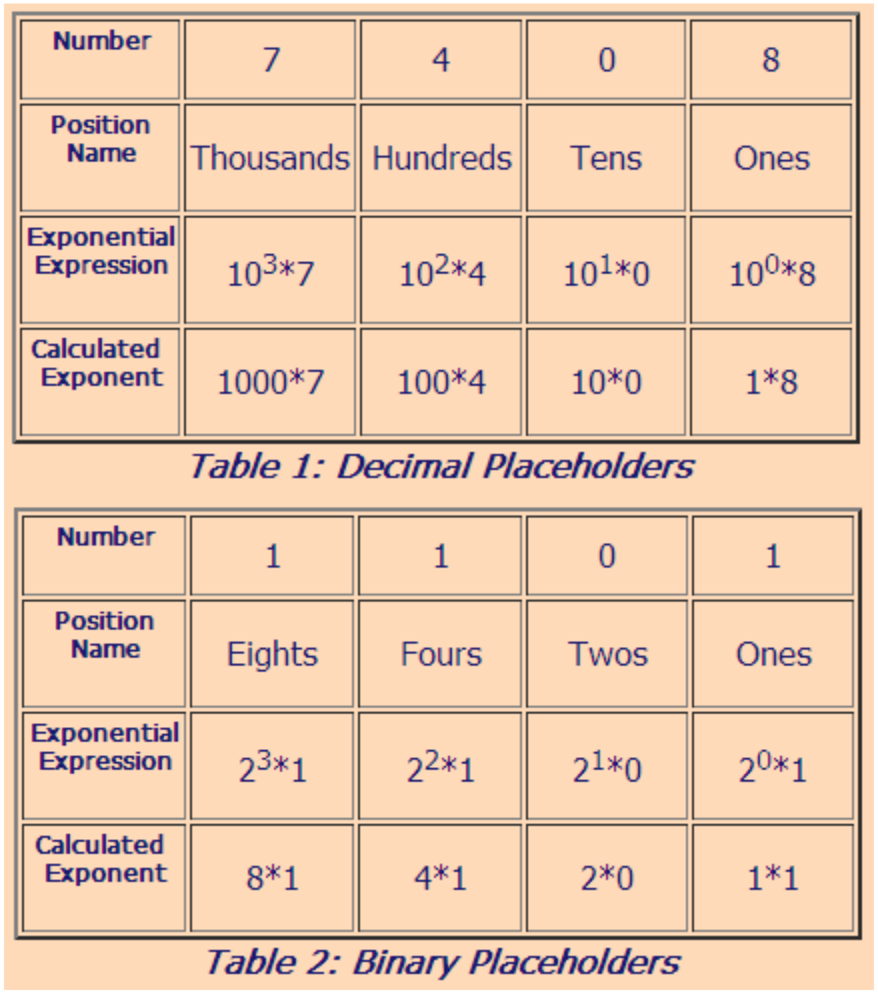
Truth Table

Input A	Input B	Output Q
0	0	0
0	1	0
1	0	0
1	1	1



Binary vs. Decimal

from: http://cs.iupui.edu/~n241/readings/binconv.html



Number	7	4	0	8
Position Name	Thousands	Hundreds	Tens	Ones
Exponential Expression	10^3*7	10^2*4	10^1*0	10^0*8
Calculated Exponent	1000*7	100*4	10*0	1*8

Table 1: Decimal Placeholders

Number	1	1	0	1
Position Name	Eights	Fours	Twos	Ones
Exponential Expression	2^3*1	2^2*1	2^1*0	2^0*1
Calculated Exponent	8*1	4*1	2*0	1*1

Table 2: Binary Placeholders

Binary Numbers

0000 = 0	1000 = 8
0001 = 1	1001 = 9
0010 = 2	1010 = 10
0011 = 3	1011 = 11
0100 = 4	1100 = 12
0101 = 5	1101 = 13
0110 = 6	1110 = 14
0111 = 7	1111 = 15

Binary Addition

(+ is back to add, for now!)

```
  0010  (2)
 +1011  (11)
 -----
  1101  (13)
```

Binary Addition - just 1 bit

(+ is back to add, for now!)

0 + 0 = 0

0 + 1 = 1

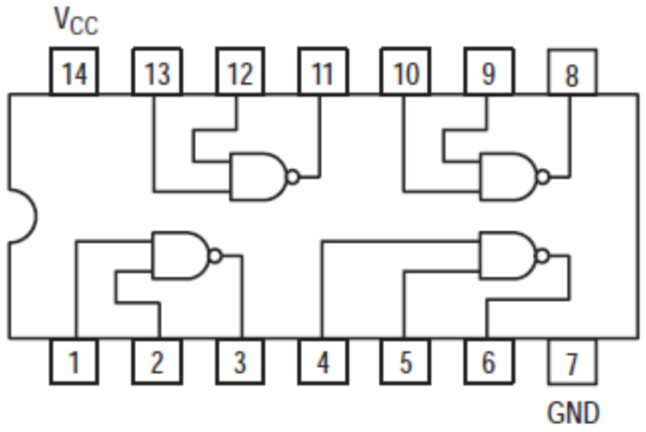
1 + 0 = 1

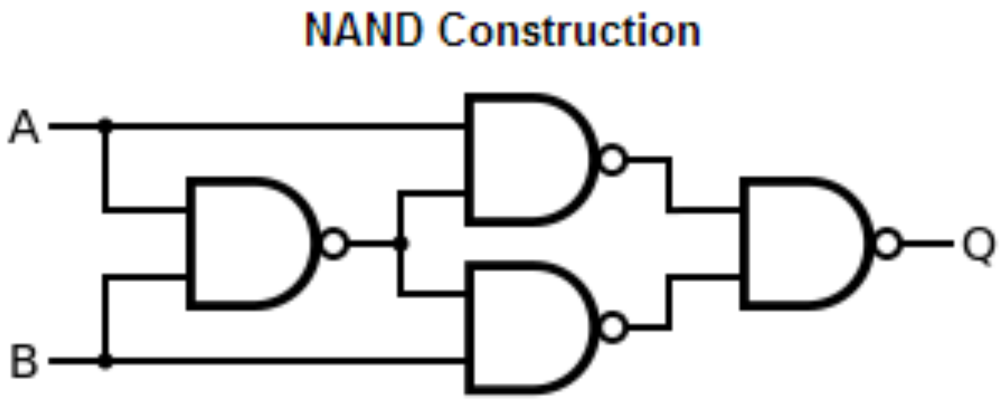
1 + 1 = 0   .... ?

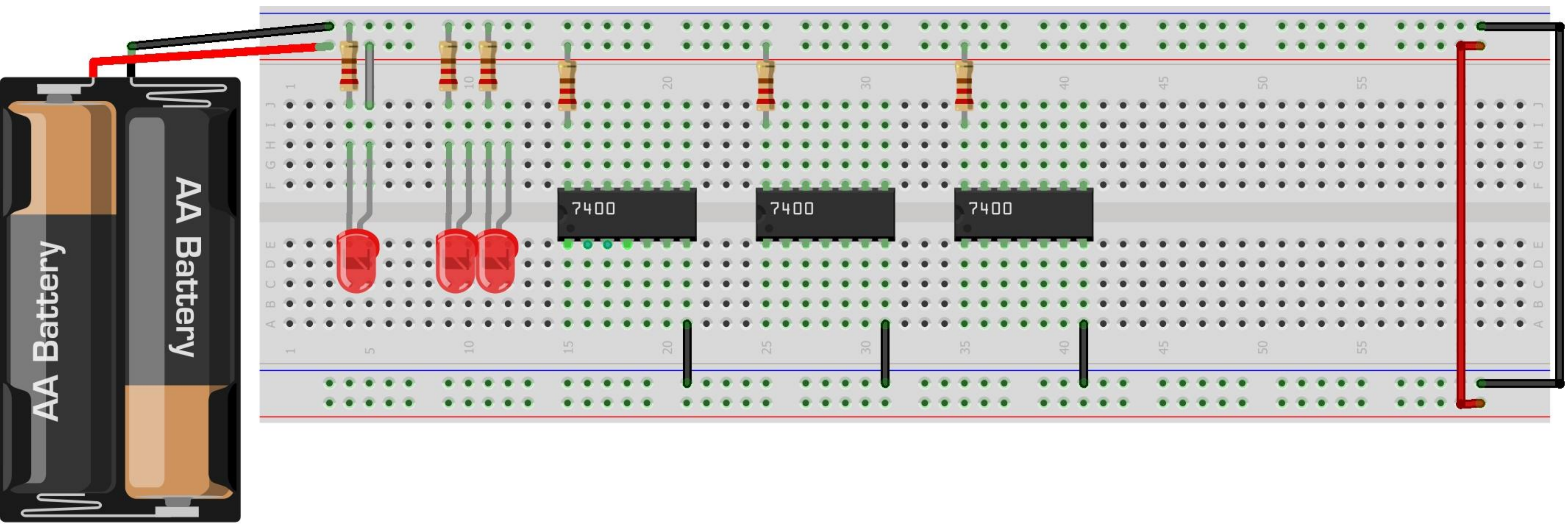
Binary Addition - just 1 bit - Truth Table

A	**B**	**S (Sum, a + b)**
0	0	0
0	1	1
1	0	1
1	1	0

Build the XOR!







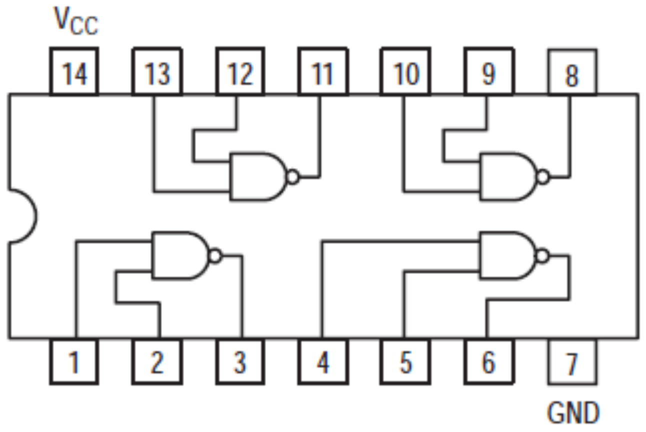
Made with Fritzing.org

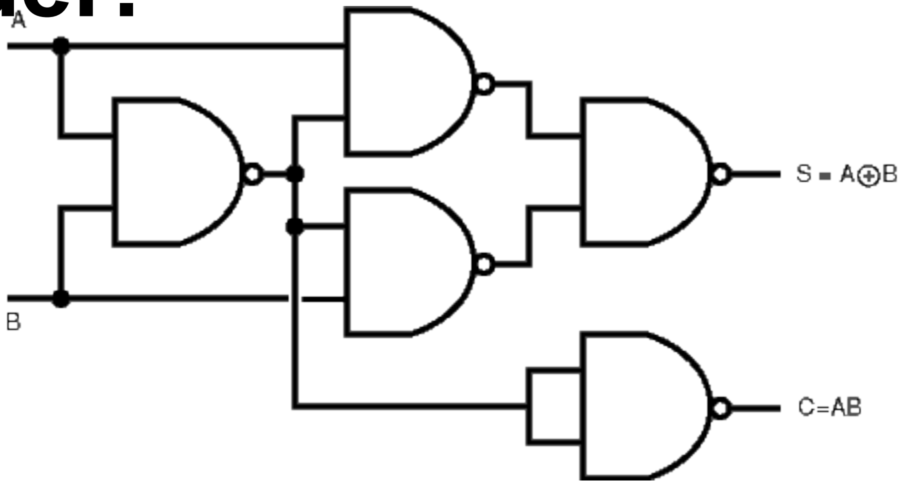
Binary Addition - just 1 bit - Truth Table

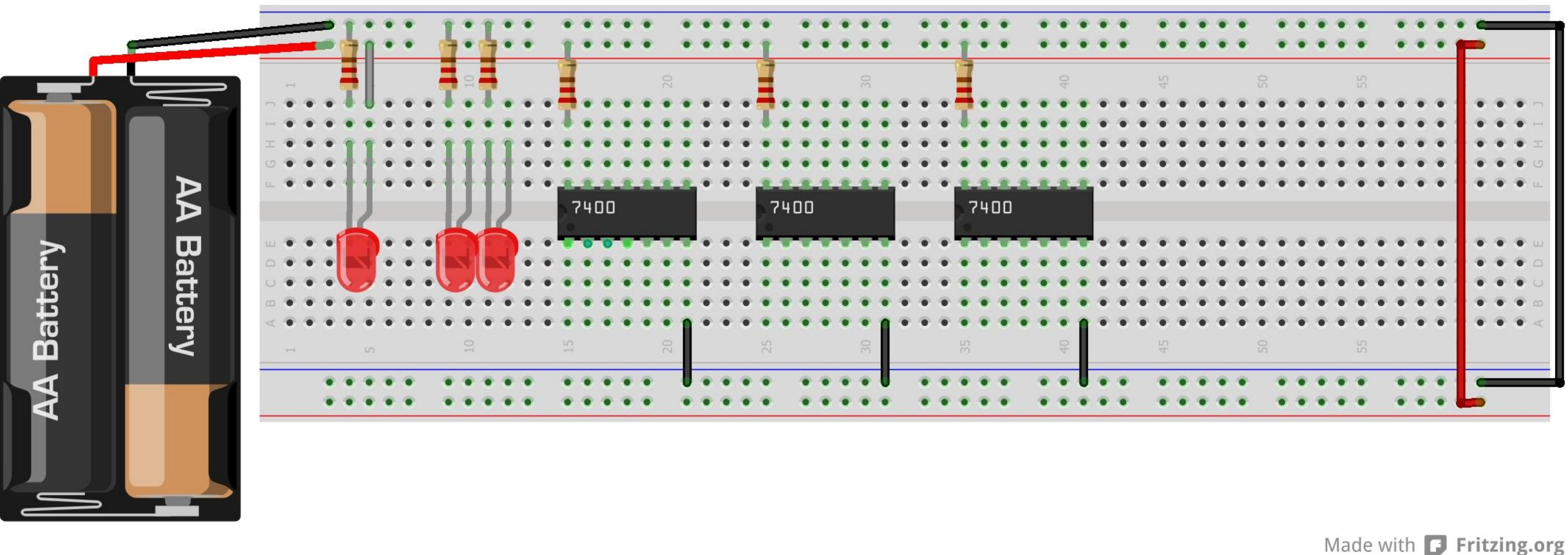
A	B	C (Carry)
0	0	0
0	1	0
1	0	0
1	1	1

$C = AB$ (Carry equals A and B)

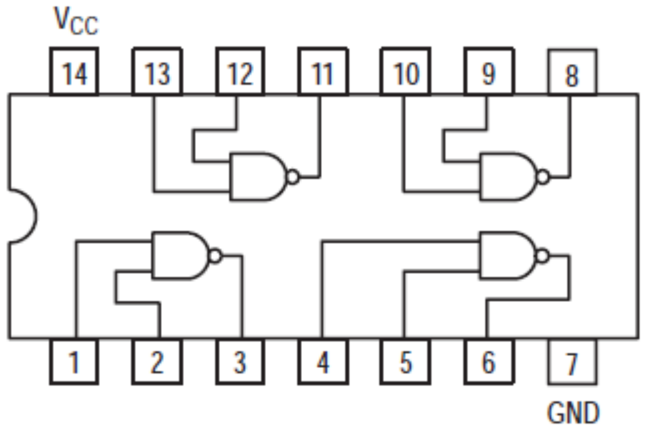
Build the Half Adder!

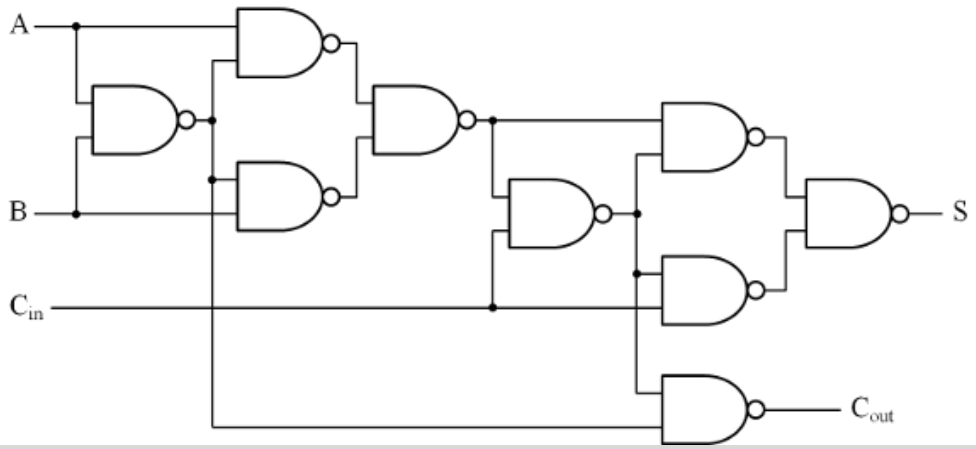


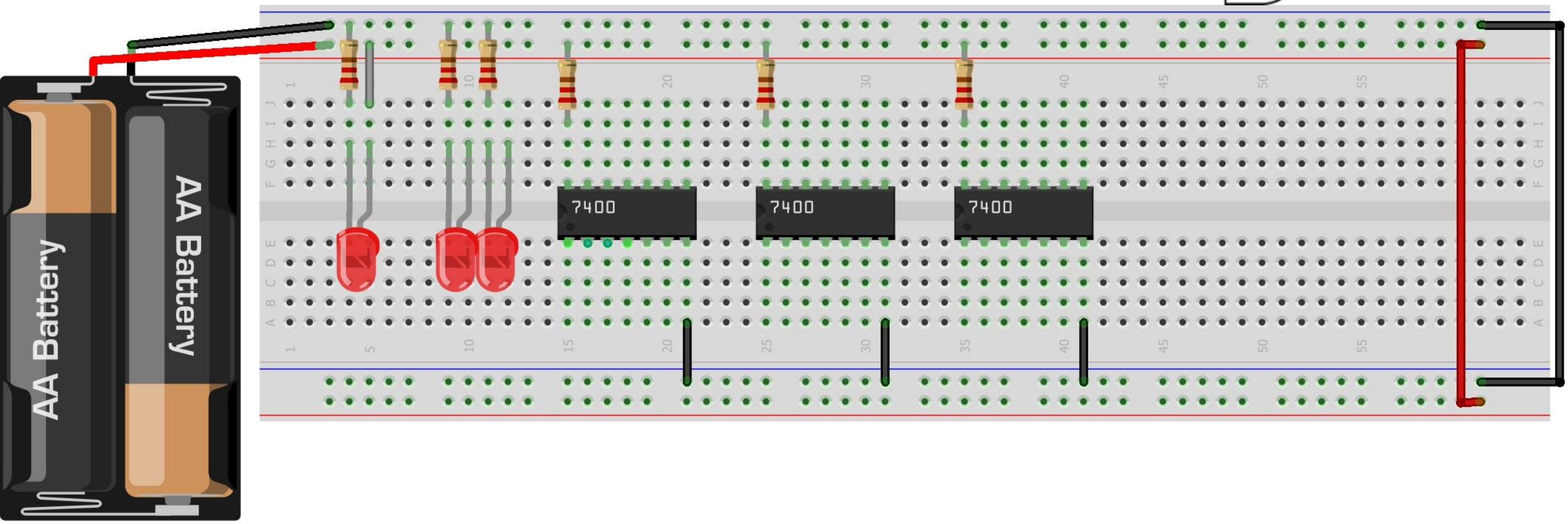




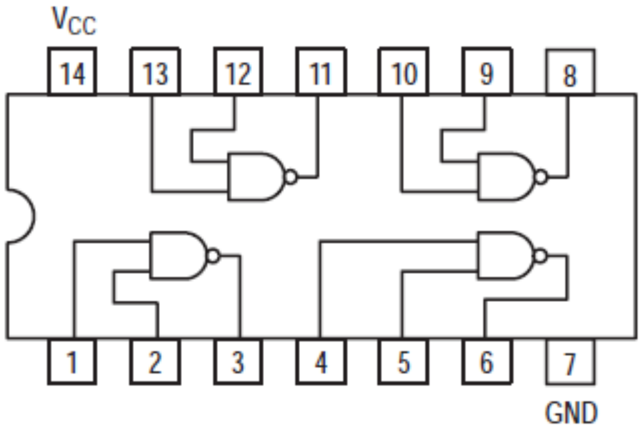
Build the Full Adder!

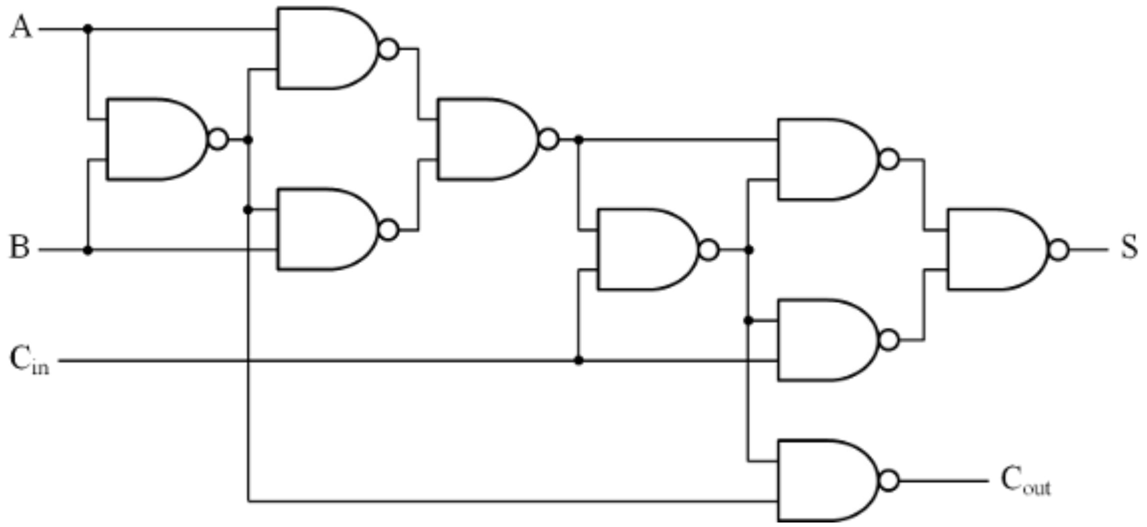






Build the Full Adder!





Now what?

http://hackaday.com

http://sparkfun.com

http://adafruit.com

http://arduino.cc

http://fubarlabs.org