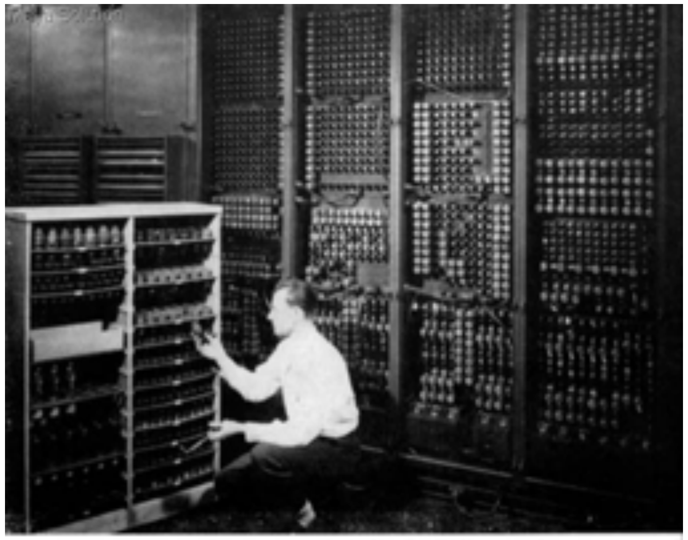
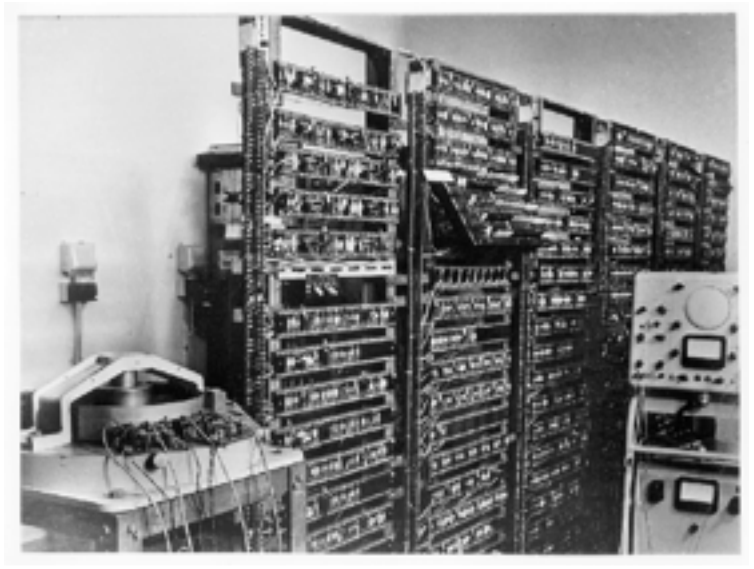
This podcast will demonstrate a logical approach as to how a computer adds through logical gates.

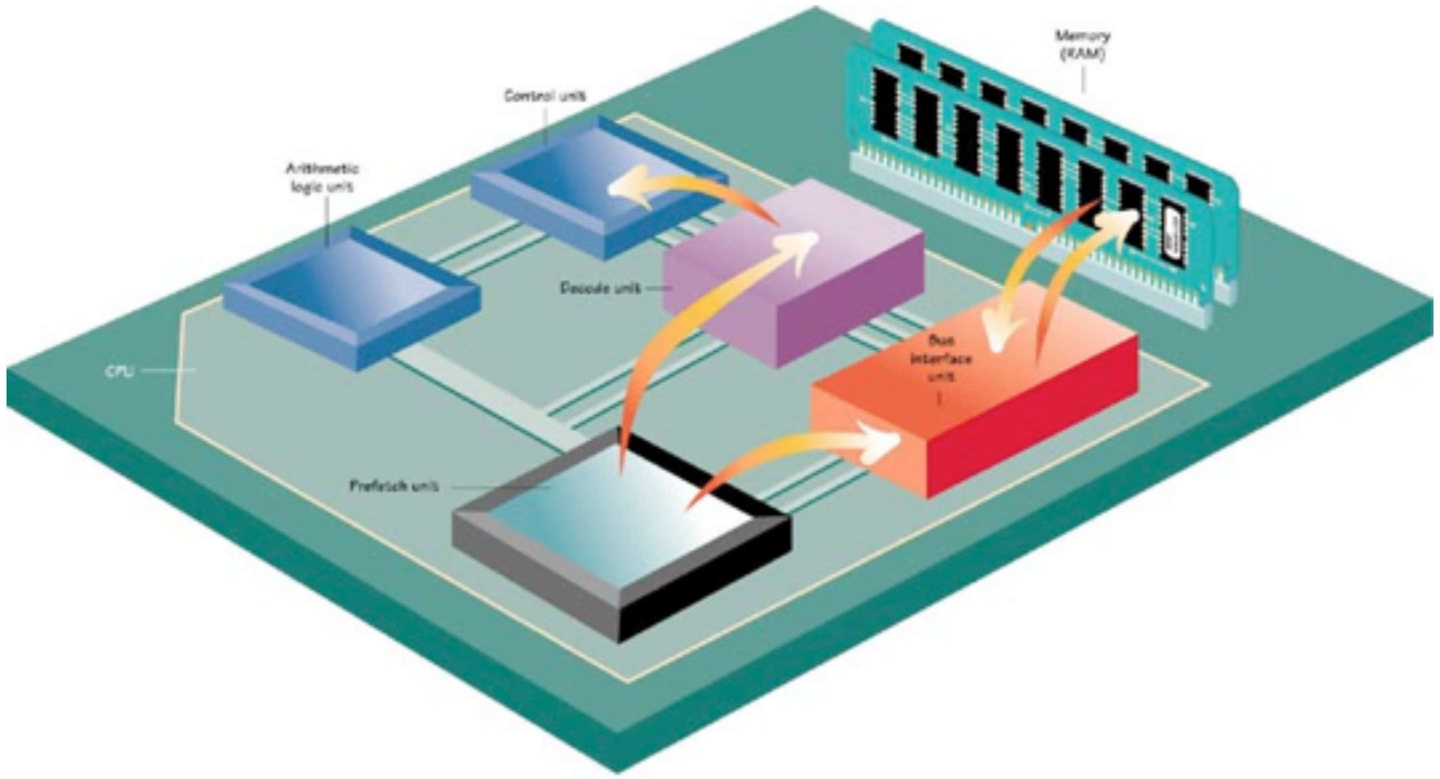
A computer is a programmable machine that receives input, stores and manipulates data, and provides output in a useful format.

Mechanical examples of computers have existed through much of recorded human history, the first electronic computers were developed in the mid 20th century and were the size of a large room pictured here. Computers using vacuum tubes as their electronic elements were in use throughout the 1950's but by the late 1960's were replaced by transistor-based machines, with integrated circuits and millions to billions of times more capable than the early machines, and occupy a fraction of the space.


Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.



In most cases, computer instructions are simple: add one number to another, move some data from one location to another, send a message to some external device, and so on.

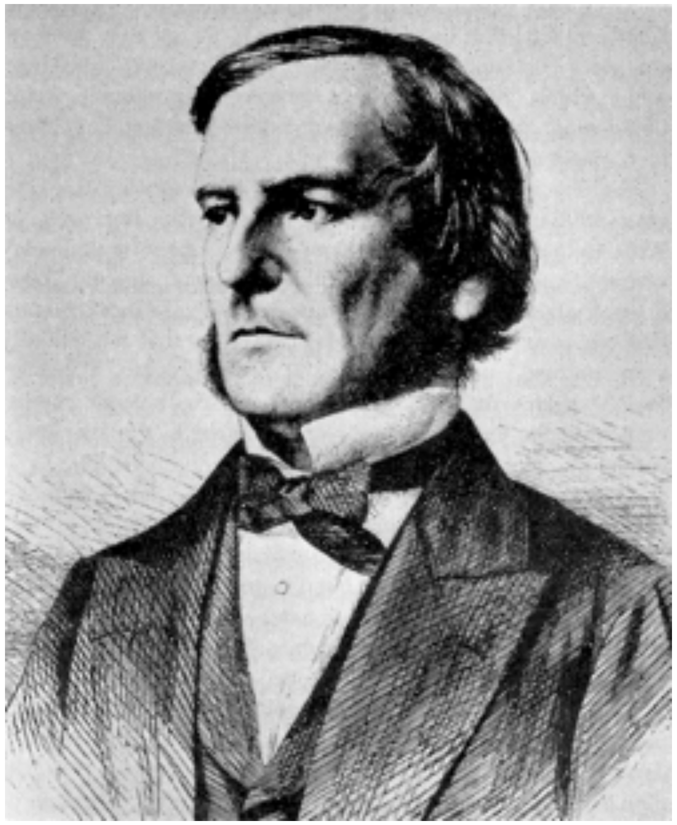


In class we looked at Number Systems and counting. As a quick refresher, we saw that computers are made up of many units of 0 and 1, the binary system. 1 is the highest digital possible so numbers in the computer are stored as for example 1010 or 10 in decimal. We also saw that these binary numbers can be seen as octal(8) or hexadecimal (16) numbers - in this case 1010 becomes 12 octal, or A hex.

You probable realize that the ‘standard’ PC code is in 8 bit bytes taking the hex system a stage further. You may also know that processors, and Windows software that runs on them, have progressed from 8 bits to 16 bits to 32 bits to 64 bits. Basically this means that the computer can work on 1, 2, 4 or 8 bytes at once.

We will look at a bit of math history that we studied in previous classes in regards to binary math which is called Boolean Algebra. Once again its really simple, but this will show you the math behind how a computer works, and why it is so important.

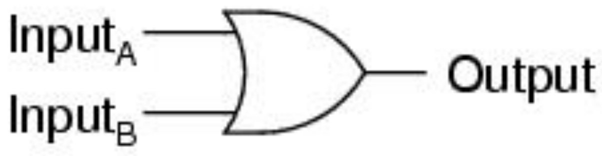
Boolean Algebra is named after George Boole, an English Mathematician in the 19th Century. He devised the logic used in digital computers more than a century before there was a computer to use it.



In Boolean Algebra, instead of + and - ., we use AND and OR to form our logic steps.

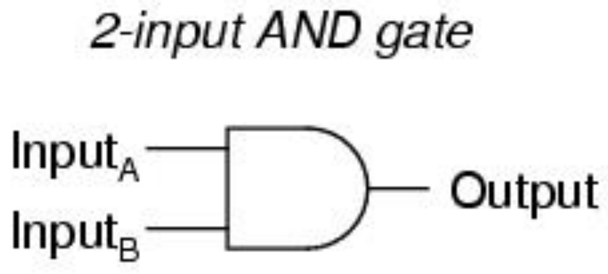
A OR B = Output which means if A or B is present, we get an Output of 1

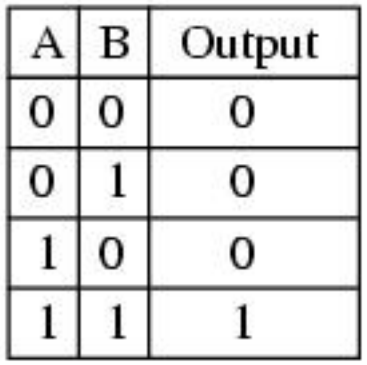
2-input OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

However,
A AND B = Output which means that both A and B need to be present to get an Output of 1.

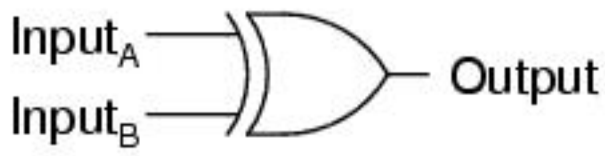


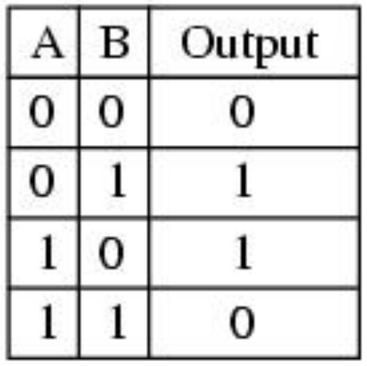


A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

We can also consider an XOR (EXclusive OR).
A XOR B = Output means that A or B BUT NOT BOTH must be present to get an Output of 1

Exclusive-OR gate

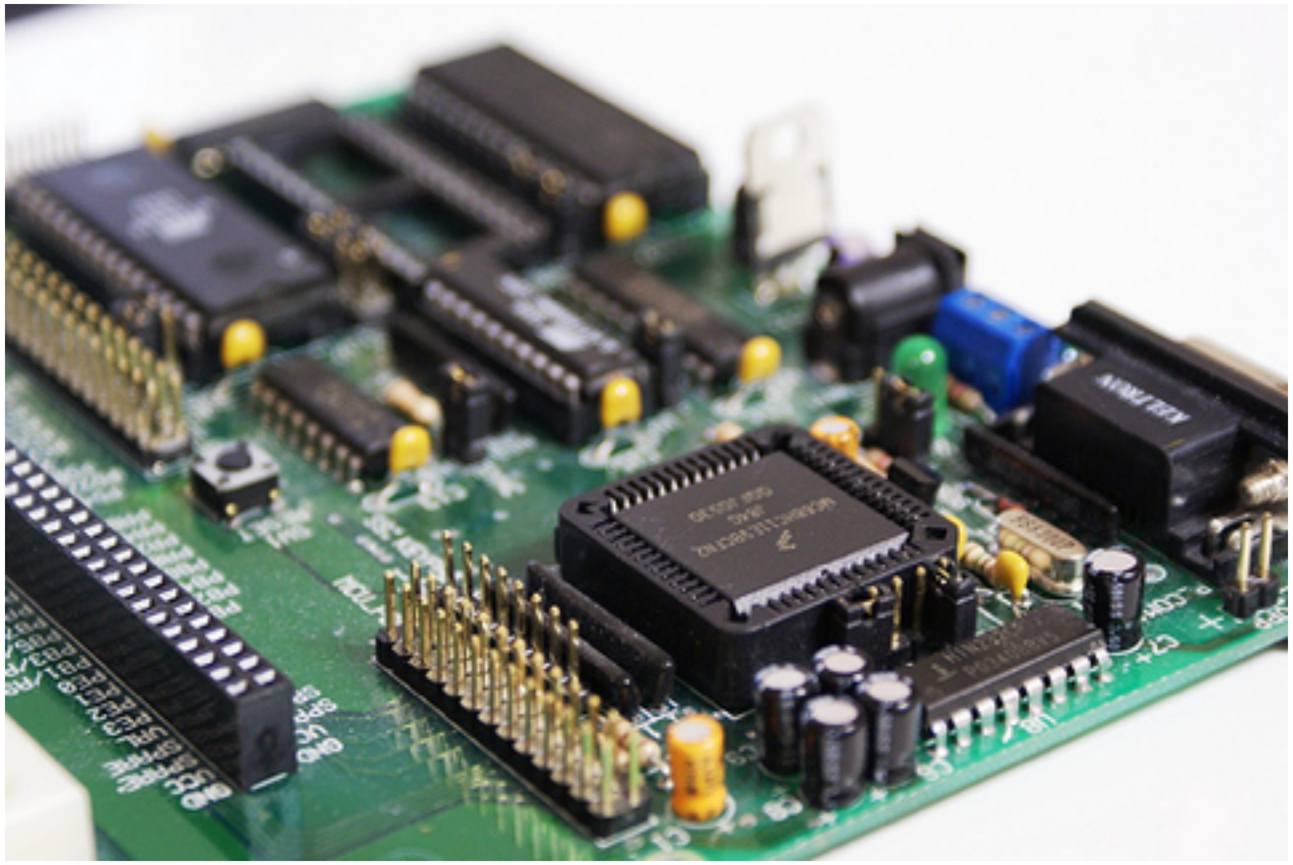




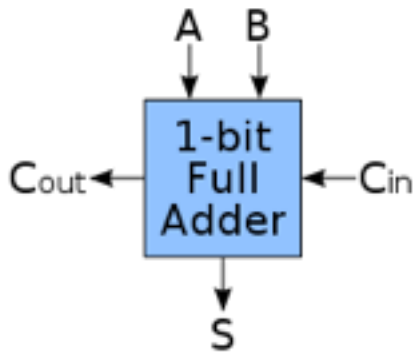
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Thats it! That’s all the math you need to understand how a computer counts.

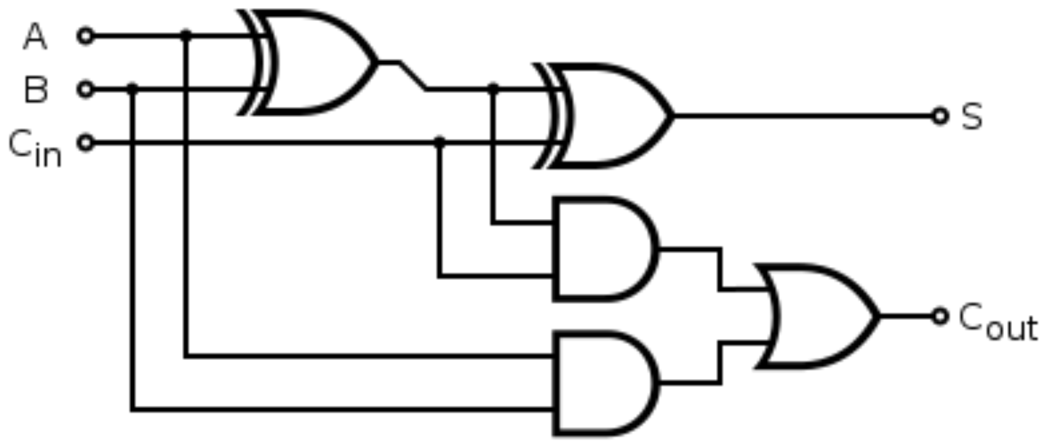
How do we use this logic in the computer? We make up a little electronic circuit called a Gate with transistors and capacitors, so we can work on our binary numbers stored in a register - which is just a bit of memory. We make an AND gate, OR gate, and an XOR gate.



To make an adder we must duplicate with a logic circuit the way we add in binary. To add 1+1 we need 3 inputs, one for each bit, and a carry in and 2 outputs, one for the result (1 or 0), and a carry out (1 or 0). In this case the carry input is not used. We use 2 XOR gates, 2 AND gates and an OR gate to make up the adder for 1 bit.



Schematic symbol for a 1-bit full adder with C_{in} and C_{out} drawn on sides of block to emphasize their use in a multi-bit adder

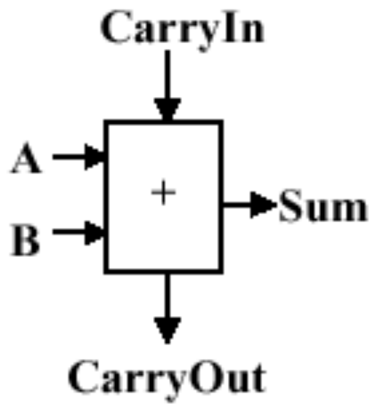


Example full adder circuit diagram

Now we go another step, and forget about gates, because now we have a Logic Block, an ADDER. Our computer is designed by using various combinations of logic blocks. As well as the adder we might have a multiplier (a series of adders) and other components.

Our ADDER block takes one bit (0 or 1) from each number to be added, plus the Carry bit (0 or 1) and produces an output of 0 or 1, and a carry of 0 or 1. A table of the input A, B and Carry, and output O and Carry, looks like this...

Symbol



Definition

A	B	CarryIn	CarryOut	Sum
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

CarryOut = (A*B*CarryIn) + (A*B*CarryIn) + (A*B*CarryIn) + (A*B*CarryIn)

= (B*CarryIn) + (A*CarryIn) + (A*B)

Sum = (A*B*CarryIn) + (A*B*CarryIn) + (A*B*CarryIn) + (A*B*CarryIn)

This is known as a Truth Table, it shows output state for any given input state.

Lets add 2+3 decimal. That is 010 plus 011 binary. We will need 3 ADDER blocks for decimal bit values of 1, 2 and 4.

The first ADDER takes the Least Significant Bit (decimal bit value 1) from each number. Input A will be 0, input B will be 1 with no carry - 0.

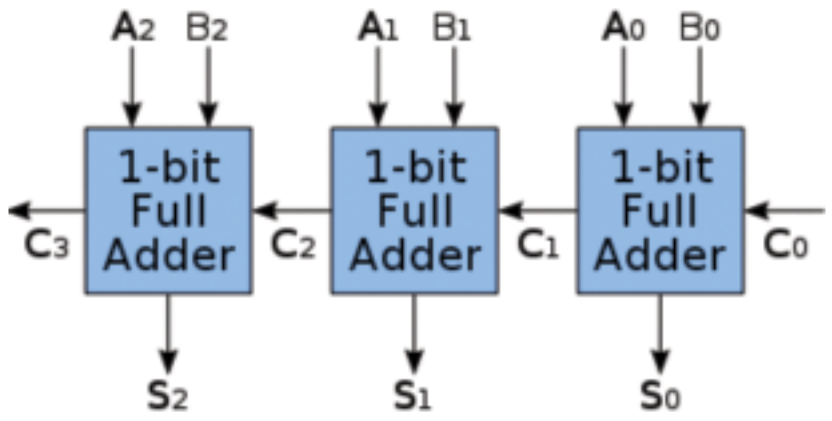
From the truth table this gives an output of 1 and a carry out of 0 (3rd row) . BIT 1 RESULT = 1.

At the same time the next ADDER (decimal bit value 2) has inputs of 1, 1 and a carry of 0, giving an output of 0 with a carry bit of 1 (4th row). BIT 2 RESULT = 0.

The next ADDER (decimal bit value 4) has inputs of 0, 0 and a carry of 1, giving an output of 1 with no carry - 0 (5th row). BIT 4 RESULT = 1.

So we have bits 4, 2, 1 as 101 or 4+1=5.

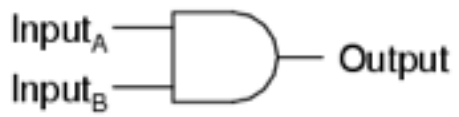
It seems like a laborious way to do it, but our computer can have 64 adders or more, adding simultaneously two large numbers billions of times a second. This is where the computer scores.



Exercise #1

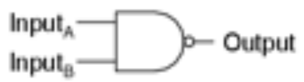
Given the following individual gates and their respective truth tables

2-input AND gate



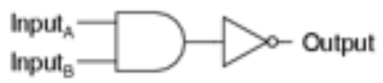
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

2-input NAND gate

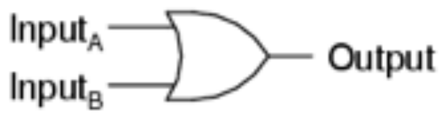


A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Equivalent gate circuit

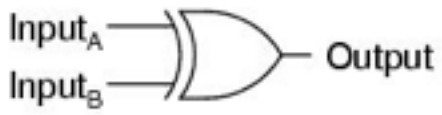


2-input OR gate



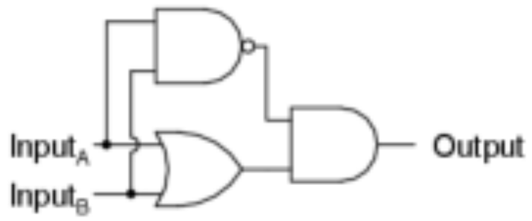
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

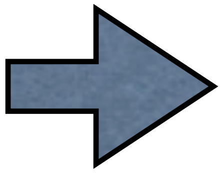
Exclusive-OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Find all the possible outputs for the following circuit





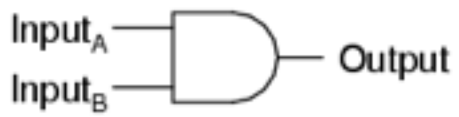
Develop the truth table

A	B
0	0
0	1
1	0
1	1

Exercise #2

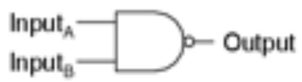
Given the following individual gates and their respective truth tables

2-input AND gate



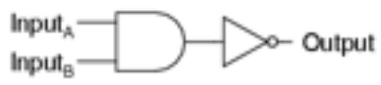
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

2-input NAND gate

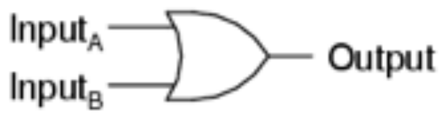


A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Equivalent gate circuit

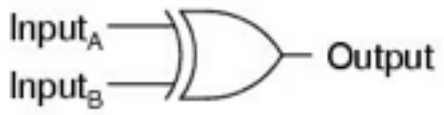


2-input OR gate



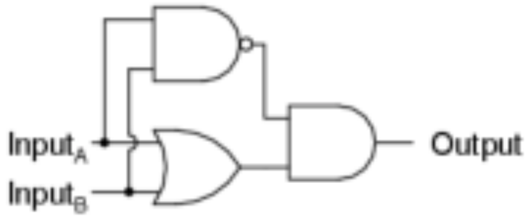
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Exclusive-OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

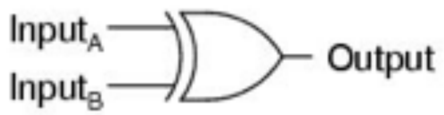
Find all the possible outputs
for the following circuit



Find the possible outputs

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-OR gate

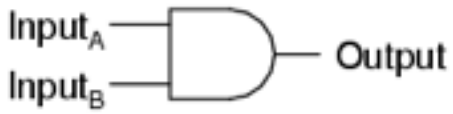


A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Exercise #2

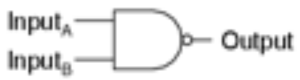
Given the following individual gates and their respective truth tables

2-input AND gate



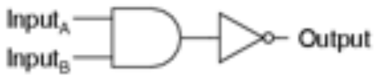
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

2-input NAND gate

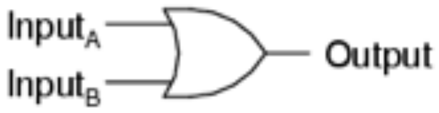


A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Equivalent gate circuit

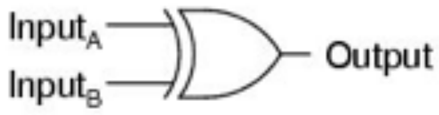


2-input OR gate



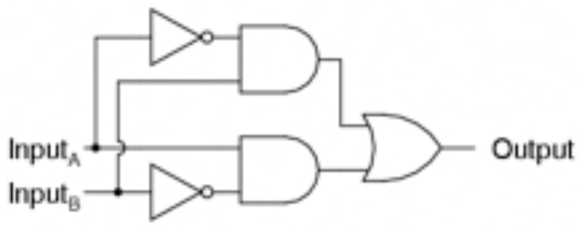
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Exclusive-OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Find all the possible outputs for the following circuit



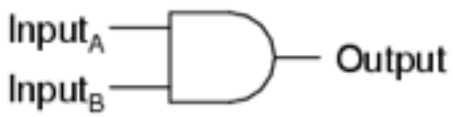
Develop the truth table

A	B
0	0
0	1
1	0
1	1

Exercise #2

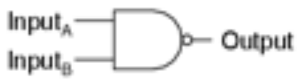
Given the following individual gates and their respective truth tables

2-input AND gate



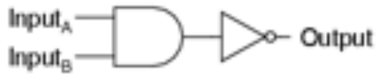
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

2-input NAND gate

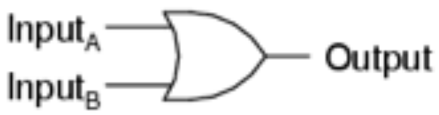


A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Equivalent gate circuit

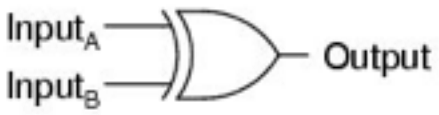


2-input OR gate



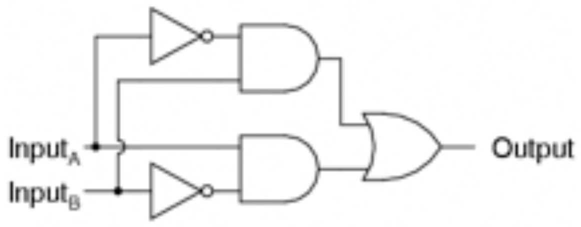
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Exclusive-OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

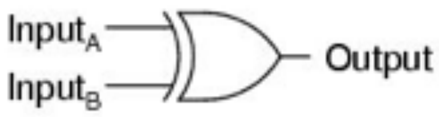
Find all the possible outputs for the following circuit



Develop the truth table

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0