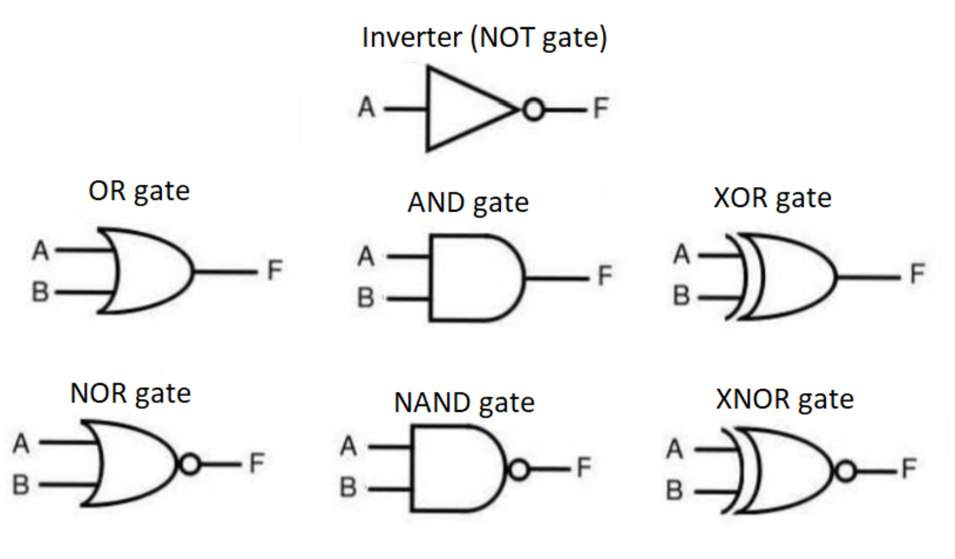
PORTS AND FUNCTIONS

Implementing logic functions in electrical circuits.



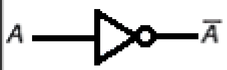
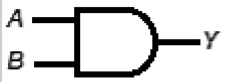
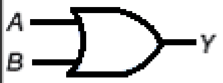
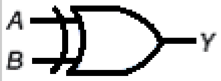
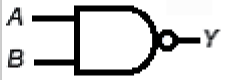
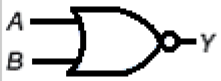
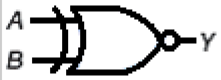
Mathematics as we know it since primary school is one of the most useful skills we have. We use it during daily chores, studies and work, sometimes even without realising. Electronic devices also cannot properly operate without performing simple mathematical and logical operations and that is why the logic gates were introduced. With them, addition and subtraction of bits was possible, as well as much more complex operations and devices, such as an ALU. In this chapter, we explain how logic gates work and how to use them in order to represent logical expressions.

REAL LIFE EXAMPLE

We know that computers use a binary system, which is a series of 1's and 0's that in some way allows us to play online computer games. With a very low level of abstraction, we can see that every program is composed of small components, which eventually lead to combinations of logic gates. Just like our nervous system. Some people can perform quite complicated things, like juggling 12 balls at the same time, but it's not something programmed in our brains. It's all a combination of electrical signals sent from our brain into some of 600 muscles in our body (which is extremely low, when compared to the amount of logic gates in a regular PC), that combined allows us to do great things. It's just like a computer uses millions of electrical signals to simulate a fight between 1.000 T-REX vs 80.000 chickens .

GATES TYPES

We distinguish 7 different gates. NOT, OR, AND, XOR, NOR, NAND, and XNOR. All possible outcomes, as well as the symbol used to represent the gate and logical expression, are presented in the following Table.

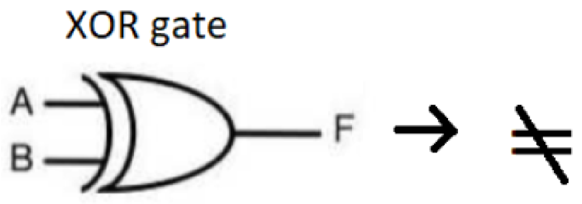
Gate Name	Symbol	Expression	TRUTH TABLE		
			INPUTS		OUTPUT
			B	A	Y
Inverter		$A = \overline{A}$		0	1
				1	0
AND		$A \cdot B = Y$	0	0	0
			0	1	0
			1	0	0
			1	1	1
OR		$A + B = Y$	0	0	0
			0	1	1
			1	0	1
			1	1	1
XOR		$A \oplus B = Y$	0	0	0
			0	1	1
			1	0	1
			1	1	0
NAND		$\overline{A \cdot B} = Y$	0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$\overline{A + B} = Y$	0	0	1
			0	1	0
			1	0	0
			1	1	0
XNOR		$\overline{A \oplus B} = Y$	0	0	1
			0	1	0
			1	0	0
			1	1	1

In general, we do not have to remember the Truth Table, but we need to understand the logic behind the gates.

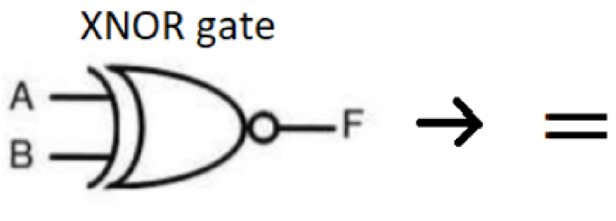
NOT gate (Inverter) - output is the opposite of the input (0 -> 1, 1 -> 0)
AND gate - only if both inputs are 1, there will be 1 on the output
OR gate - if there is at least one 1 on the input, there will be 1 on the output
XOR - Only if the inputs are of the opposite values, the output will be 1. It is an eXclusive OR gate, which means that if either A or B is 1, but not both, the output will be 1.
The **NOR**, **NAND** and **XNOR** gates are basically the opposite of the OR, AND and XNOR, meaning that if for a given input of OR gate, the output will be 1, for the same input in the NOR gate, the output will be 0. So it is possible to deduct the outcome for those three gates just by remembering the functioning of their "original" gate.

GATES AND MATHEMATICAL FUNCTION

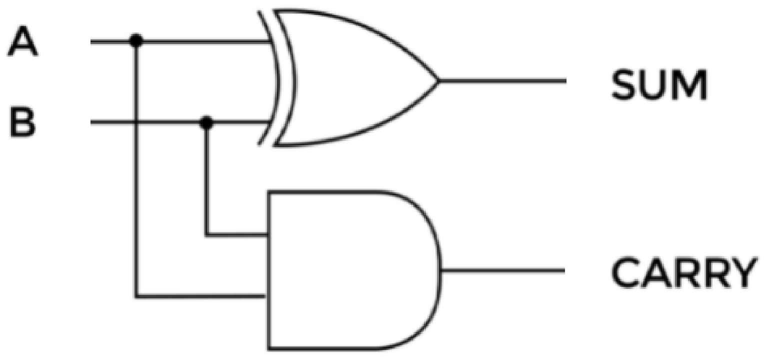
Some gates can be directly used to represent mathematical function.
XOR gate gives 1 as a result when two inputs are different, so it can be expressed as not equal sign.



What is more, XOR gate is used in adders to return the result.
XNOR in contrast to XOR returns 1 if both inputs are the same, so it can be expressed as the equality sign.



AND gate can be used in adders to return the carry.



LOGICAL EXPRESSIONS

Obviously, while creating logical circuits we will need more complicated Logical expressions, so it is good to know how to represent them with the use of the gates. Let's take for example the following expression

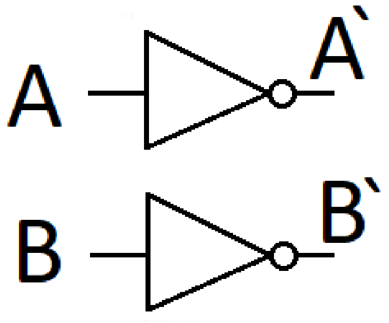
F = A`B`

We can start both from output to input, or input to output, whichever method suits you more. You just have to remember that the output should be on the right and the inputs on the left. In this example, we will follow from inputs to the output. At first, you should write them down and focus on them separately:

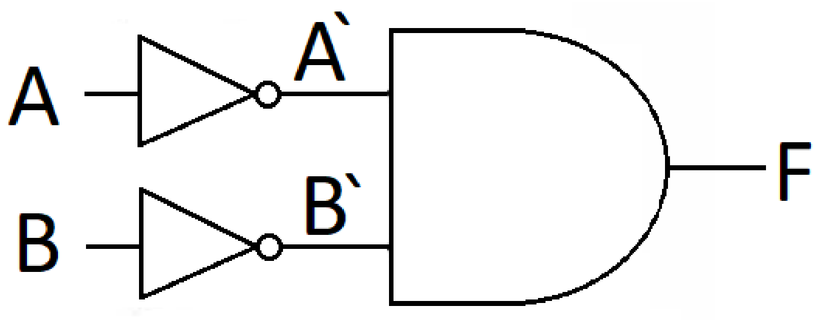
A

B

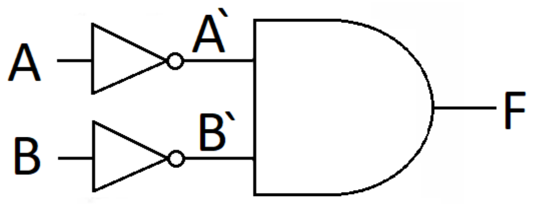
We can see, that both A and B should be negated (A`, B`), so we do that:

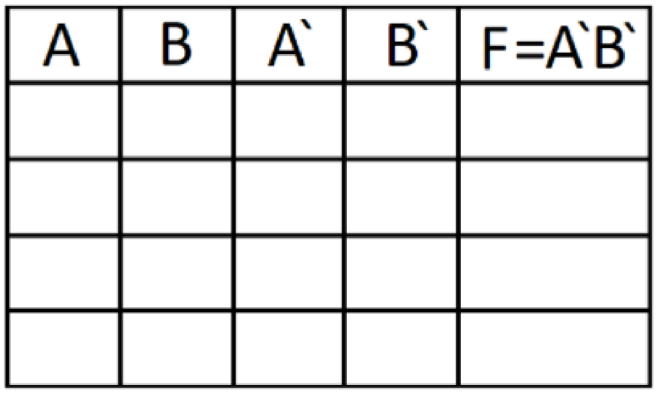


And finally, the inverted inputs should be connected by an AND gate (A`B`):



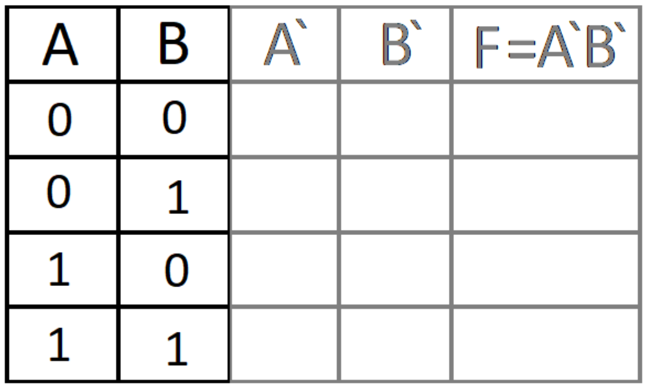
To get the outputs for this expression, you should fill up the Truth Table, following the input changes from left to right. Take your time, and solve for each pair of possible inputs. A good idea is to present the Table the same way it is presented in the graph:





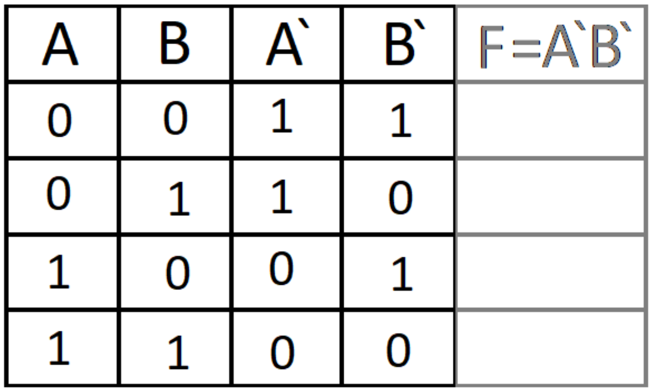
A	B	A`	B`	F=A`B`

At first, fill the possible inputs:



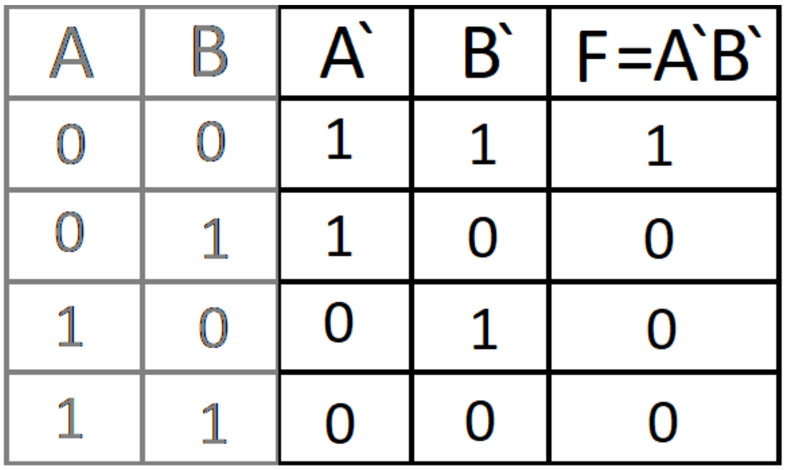
A	B	A`	B`	F=A`B`
0	0			
0	1			
1	0			
1	1			

Then do the negations:



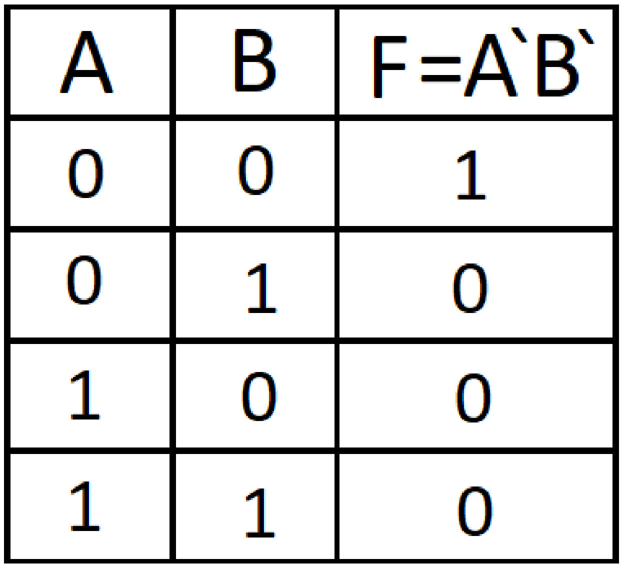
A	B	A`	B`	F=A`B`
0	0	1	1	
0	1	1	0	
1	0	0	1	
1	1	0	0	

In the last phase, you perform just the and function between A` and B`:



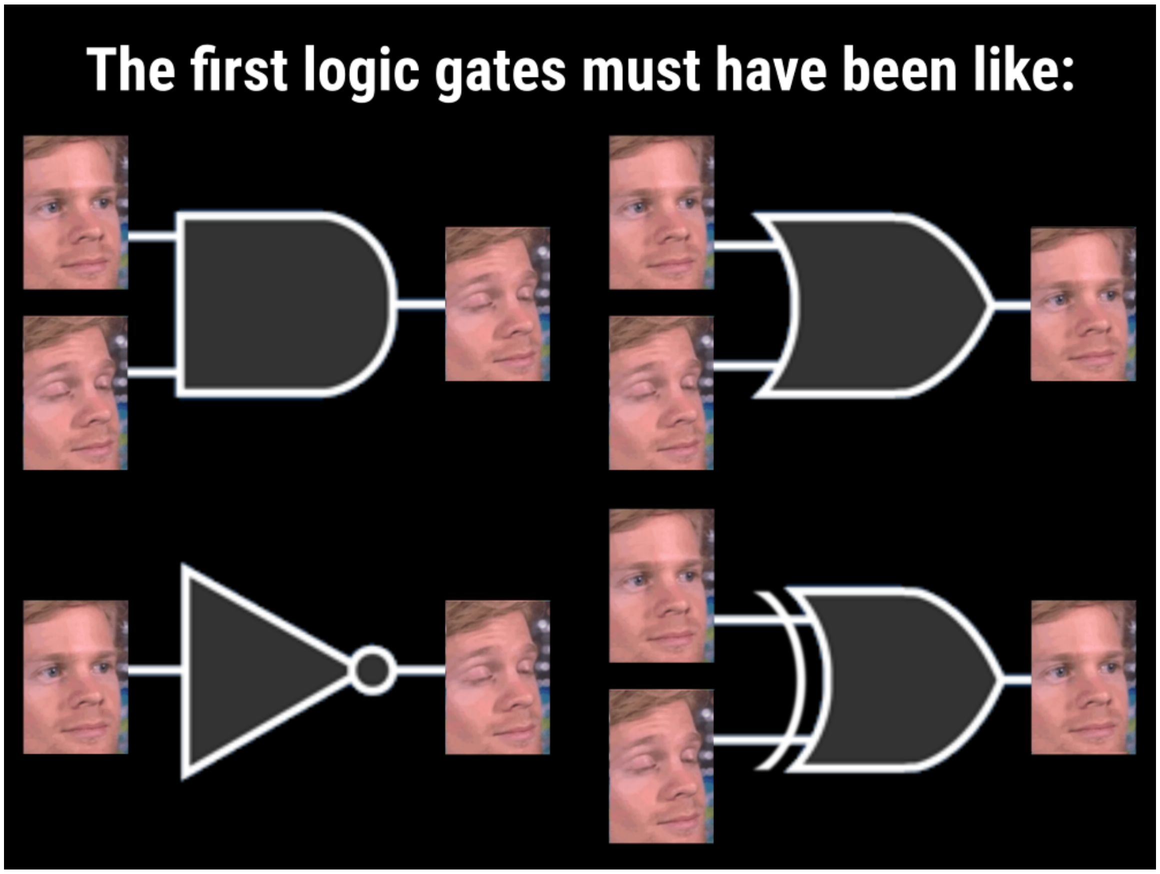
A	B	A`	B`	F=A`B`
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Just don’t forget, that the result is the combination of the input and outputs, so you can rewrite the table without the middle part:



A	B	F=A`B`
0	0	1
0	1	0
1	0	0
1	1	0

The same method may be applied to much more difficult expressions, so it is good for you to remember them.



https://www.reddit.com/r/LogicGateMemes/comments/d2nr07/the_first_logic_gates/