How big a number can we represent with a single bit? Think back to our last lesson on binary numbers. A single bit can represent a number in the range 0 to 1. A single bit in a computer can only represent a number in the range 0 to 1. We will start our discussion of arithmetic by only looking at numbers in the range 0 to 1.

Now think of the multiplication (×) operator, which use use in everyday maths. We can write down a table of all the possible results when we multiple two numbers, which can be 0 or 1, together. The table looks like this:

A	**B**	**A × B**
0	0	0
0	1	0
1	0	0
1	1	1

Compare it with the truth table for the AND operator. **They are the same! We can use an AND gate circuit to multiply two single bit numbers together!**

Now think of the addition (+) operator, which use use in everyday maths. We can write down a table of all the possible results when we add two numbers, which can be 0 or 1, together. The table looks like this, writing all the numbers in binary (base 2):

A	**B**	**A + B**
0	0	0_2
0	1	1_2
1	0	1_2
1	1	10_2

Compare it with the truth table for the OR operator. The ones column is **almost** the same as the truth table for the OR gate. Only the bottom row is different, with the OR gate giving a result of 1, whereas we need a result of 0.

The solution to this is to combine the basic AND, OR, NOT logic gates to produce a new logic gate, which gives the desired output.

We can do arithmetic with logic!

31/7/2018 Lesson 6: (Term 3, Week 2)

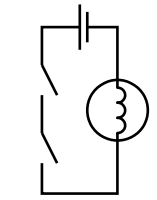
Reviewing Logic Gates

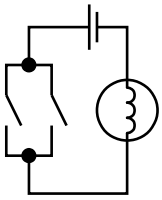
Remember our three basic logic operations from last week: AND OR, NOT. Let's review them by drawing their truth tables, a circuit that we used to implement them, and the symbols that we use to represent them on a circuit diagram.

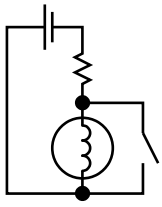
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

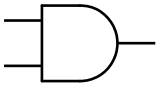
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

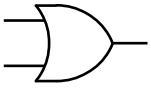
A	NOT A
0	1
1	0

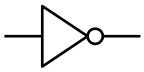












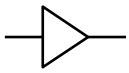
In our truth tables, we have used 1 and 0 to represent “true” and “false” respectively.

Pass the circuits for the AND, OR and NOT gates around. Have a look at the positions of the wires and components on each breadboard, and match them up with the circuit diagrams that I have drawn. They are called “circuits” as the electricity has to go in a circle, ending up where it started from. One of the simplest circuits we can build is a light bulb connected to a battery. You may be familiar with such a circuit. The circuit I built uses an LED and resistor in place of a light bulb. You can add a switch to the circuit, to turn the light on and off. It's the same as the lights in this room: close the switch, the electricity flows and the light turns on, open the switch and the light turns off. We make a logic gate just by adding another switch. In the AND gate the switches are in series, meaning that the electricity flows though both switches and both switches have to be closed for the light to turn on. In the OR gate, the switches are in parallel, so only one switch has to be closed for the light to turn on. The NOT gate is a little different. It's just a light connected to a battery. The switch is in parallel with the light, so when the switch closes the electricity flows through the switch instead of the light and the light turns off. The resistor prevents the switch from short-circuiting the battery.

Aside: LEDs have low resistance, so we have to place a resistor in series with them to avoid too much current going through them and burning them out. Light bulbs have a higher resistance so don't need a resistor with a light bulb.

Note the symbols at the bottom of the previous diagram. The AND gate has a round nose with a flat bottom. It has two inputs, corresponding to the two input columns of the truth table. We apply a high or low voltage to these inputs, which we talk about as being a “1” or “0” respectively. There is a single output, corresponding to the output column in the truth table and it is also a high or low voltage, corresponding to a “1” or “0”. The OR gate has a point nose with a curved bottom. The NOT gate is a triangle with a small circle at the output. The circle is important, as it indicates the NOT operation. The triangle without a circle is a type of gate called a YES gate or a BUFFER. Its truth table and symbol are below.

A	YES A
0	0
1	1



We can join these logic gates (AND, OR, NOT, YES) to form circuits called logic networks, so the voltage at the output of one gate drives the input of the next gate. When we do this we use an electrically controlled switch, called a transistor, in place of the strips of tin that we can push with our finger. This saves us having to sit there and look at a light and press the switch on the following gate. Despite the use of this electrically controlled switch, the principles are exactly the same as for our manually controlled gates made of tin and wire.

Combining Logic Gates (to do Arithmetic)

Lets figure out a truth table for a circuit with two inputs and two outputs, to add two numbers together. Our numbers are represented by a bit, so they can only have two values: 0 or 1. Let's start by considering all possible combinations of the numbers that we are adding together:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 2

Let's rewrite the above sums in binary, remembering our earlier lesson about binary numbers:

$0_2 + 0_2 = 0_2$
$0_2 + 1_2 = 1_2$
$1_2 + 0_2 = 1_2$
$1_2 + 1_2 = 10_2$

Remember that we can only use the digits 0 and 1 to count in binary. In the sun 1+1 with the answer 2, we don't have a binary digit for 2, so we have to move to the next column, which has a place value of two. The binary number 10_2 is the same as the decimal number 2.

When we write numbers, we can put a zero in front of them without changing their value. For example, the number 05 is just the number 5 isn't it? The same is true for binary numbers. Let's do this to the sums above, so all the answers have the same number of digits.

$0_2 + 0_2 = 00_2$
$0_2 + 1_2 = 01_2$
$1_2 + 0_2 = 01_2$

$1_2 + 1_2 = 10_2$

Let's just arrange the above sums and answers into a table. The numbers being added together go on the left, as inputs, and the answer goes on the right as output.

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

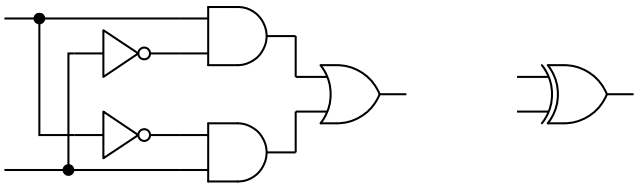
The above is a truth table for a circuit that adds two numbers together. In each row, the numbers in the A and B columns are added together to give a binary number, represented by the two bits in the C and S columns, the C column having a place value of 2 and the S column having a place value of 1.

How would we build a circuit to implement this truth table and add two numbers together? Compare the S with the truth table for the OR operator. The S column is **almost** the same as the truth table for the OR gate. Only the bottom row is different, with the OR gate giving a result of 1, whereas we need a result of 0.

The solution to this is to combine logic gates to produce a new circuit, which gives the desired output We call this circuit an EXCLUSIVE OR gate, often abbreviated to XOR. It is almost, but not quite, an OR gate. Below I have drawn the truth table for an XOR gate. You will see that it is like an OR gate, except that both inputs being 1 results on a 0 at the output. We can build an XOR gate by combining basic AND, OR and NOT gates. Such a logic network is also drawn below.

The XOR gate is so commonly used that we give it a symbol like the AND, OR and NOT gates. The symbol for an XOR gate is also shown below. The XOR symbol is like an OR gate, but it has an extra line on the bottom.

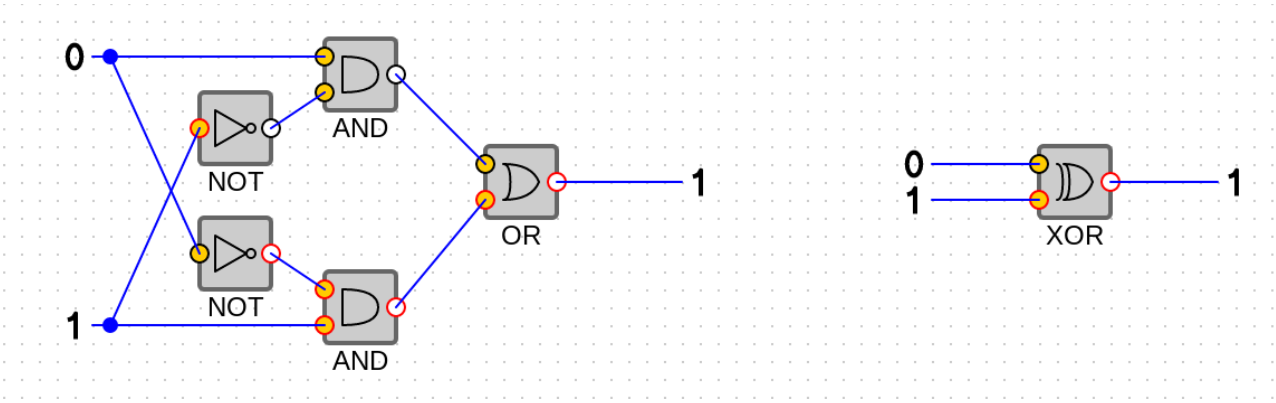
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



At this point we introduce the **SimcirJS** logic simulator. This program runs inside a web browser and allows you to draw a logic network and apply bits to its inputs. The program will then calculate the output of the network for the provided input bits. You can do this calculation yourself, on a sheet of paper, using your knowledge of AND, OR and NOT gates, but the simulator can do it faster. You can access the simulator program at the URL http://bugs.local/ on the teaching LAN or https://john.daltons.info/teaching/engineering/simcirjs/ on the Internet.

The simulator provides the gates we have introduced so far: AND, OR, NOT, along with XOR. It provides the NumSrc component to apply bits to inputs and the NumDsp to display the state of an output. There are also more complex blocks that we will cover in the future, such as half-adders, full-adders and seven segment displays. You can construct a logic network by dragging components from the toolbar into the drawing area, then clicking and dragging to draw wires between the inputs and outputs of the components you have placed. To delete a component, drag it back onto the toolbar. Additional instructions are on the SimcirJS page.

Start by drawing the network for the XOR circuit, which we have just discussed. Use the NumSrc component to apply bits to the inputs and the NumDsp component to display the state of the output. You can change the value of an input by clicking on the "1" or "0". Try all possible input combinations: 00, 01, 10 and 11 and observe the output to verify that the behaviour of the network matches the truth table for the XOR gate.



For this simple network, you should also do the calculation on a bit of paper to check that the simulator is telling the truth. Working from the gates closest to the inputs, use the truth table for each gate and the values of it inputs to write a "1" or "0" at the output of the gate. Repeat this procedure, working from the input of the network to its output, until you have written down the value of the output of the network. This value should match that given by the simulator.

The SimcirJS simulator provides an XOR component, which allows you to place an XOR network as a single component rather than having to draw the entire XOR network everytime you use it. Place an XOR component in the simulator and connect bits to its inputs, and a bit display to its output. As before, try every input combination: 00, 01, 10, 11 and verify that the behaviour of the XOR gate agrees with the XOR truth table and the network you just simulated.

Now consider the "C" column of the truth table that we drew earlier, for the circuit that can add two numbers together. Compare it to the truth table for the AND gate. It's the same! You can use an AND gate to generate the "C" column of the adder circuit.

Place an XOR gate and an AND gate as below in the simulator. Verify that the behaviour of the circuit matches the truth table for the adder circuit, and that it really can add the numbers "0" and "1" together. The output of the adder is a 2-bit binary number: the XOR gate generates the S output which represents the "1s" column and the AND gate generates the C output which represents the "2s" column.