**4CS015 – Workshop #3**

Right. We’ve nailed number systems (Or at least, we should have!) in the workshops and are now beginning to take a more in-depth look at logic in the lectures. We’ll see how these two things come together again in a few weeks time. This tutorial aims to help you understand how we work with the basic building blocks on paper. The workshop will show you how we build circuits using them.

## AND, OR and NOT gates

Although we looked at these as separate entities in the lecture, we only briefly discussed how they interact. Remember that the biggest trick to understanding logic circuits (or any electric circuit for that matter!) is to regard all of the methods that we have discussed so far as tools that enable us to twist and form outputs from given inputs. **It is us, as the designers, who dictate what goes in to a circuit and what comes out.** You should understand what you are trying to do in each circuit and then figure out *how* to do that with the tools you are given. This will hopefully become clearer in the next two lectures, but you need to understand the tools first.

The three sentences that will get you through all of your time working with logic are:

1. The output of an AND gate is **1** if the first input ***and*** the second input are **1**
2. The output of an OR gate is **1** if the first input, ***or*** the second input, ***or*** both inputs are **1**.
3. The output of a NOT gate is ***not*** the input.

AND gates and OR gates, by their very definition, require multiple inputs while a NOT gate (also known as an inverter) has only one. REMEMBER that ‘∙’ represents AND, ‘+’ means OR and ‘!’ means NOT (So does a bar over a variable/group of variables).

The most common method of representing a logic circuit, whether it’s a single gate or a more complex structure, is through a ***truth table***. A truth table shows ALL POSSIBLE inputs and their corresponding outputs.

We’re working in binary, so every input can either be a ‘0’ or a ‘1’. Thus a two input logic gate would have the form:

|  |  |  |
| --- | --- | --- |
| **INPUT 1** | **INPUT 2** | **OUTPUT** |
| 0 | 0 | ? |
| 0 | 1 | ? |
| 1 | 0 | ? |
| 1 | 1 | ? |

Using the sentences above, produce the truth tables for AND, OR and NOT gates.

We can now use these three truth tables to develop truth tables for more complex circuits or mathematics representations. We’ll cover circuits in the workshop and concentrate on mathematical representations here.

First we must remember that, as with any mathematical manipulation, there is a precedence of operators. The rules are simple:

1. Do the NOT operations first, then the AND, then the OR. Got it? NOT AND OR
2. Brackets work as normal
3. If a line goes over more than one variable, treat the variables as if they are in brackets

In the tutorial, create truth tables for the following:

1. !(A+B)
2. !A∙!B... What do you notice about these two truth tables?
3. !A+!B
4. !(A∙B)... What do you notice about these two truth tables?

We can create much more complex maths. None of which is very complicated, if you break it down. Crack on with these (you can use your knowledge of the three sentences, or the truth tables to solve these):

If A=1, B=1, C=0, D=1, E=0, G=1, calculate F

1. F= A∙B∙C
2. F= C∙A∙B
3. F= !A+B+C
4. F= A∙B+C
5. F= A+B∙C
6. F= A.B + (C +A) .D+E+G