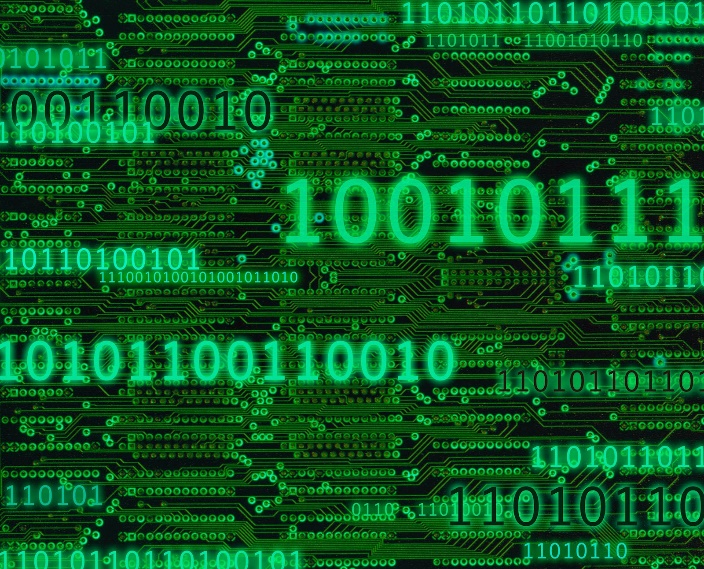
**P4- Produce a report that describes how data can be represented in control systems. You must describe coding data, logical operators and integer, fixed point and floating-point numbers**

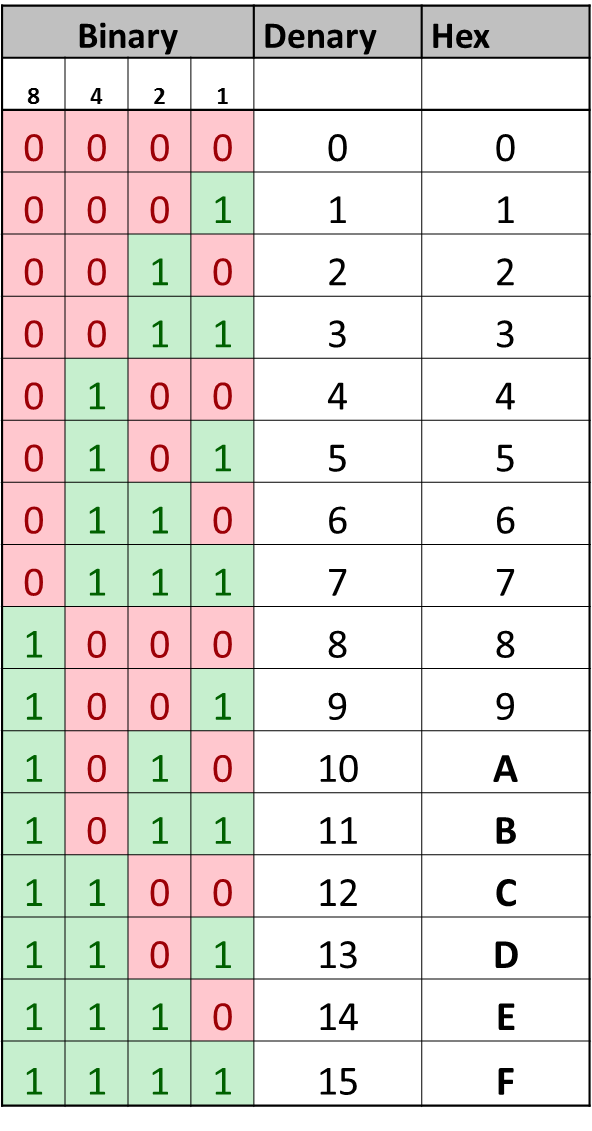
**Binary**

Binary is two digits that the computer uses in order to identify and communicate with each other. The two digits are zero and one. Computers operate in binary, this means they store data and perform tasks with zeros and ones. This table below demonstrates how to convert numbers to binary, zero and one.

The main way to work it out is to convert it by using the numbers– 128, 64, 32, 16, 8, 4, 2, and 1 - to get what you want. For example, if you want 13 as a denary, you should add the ones that make 12. Therefore, 8+4. They are represented as 1 and the others are 0s. It should look like this 01100. I only used 16, 8, 4, 2, and 1 for it, as I did not need the others. However if you do want to work out 256, you would need to double 128 and carry on as normal. The ones you added to make the number you put a one and the ones you did not you put a zero.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 27= | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| Decimal | **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| 72 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 65 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 85 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

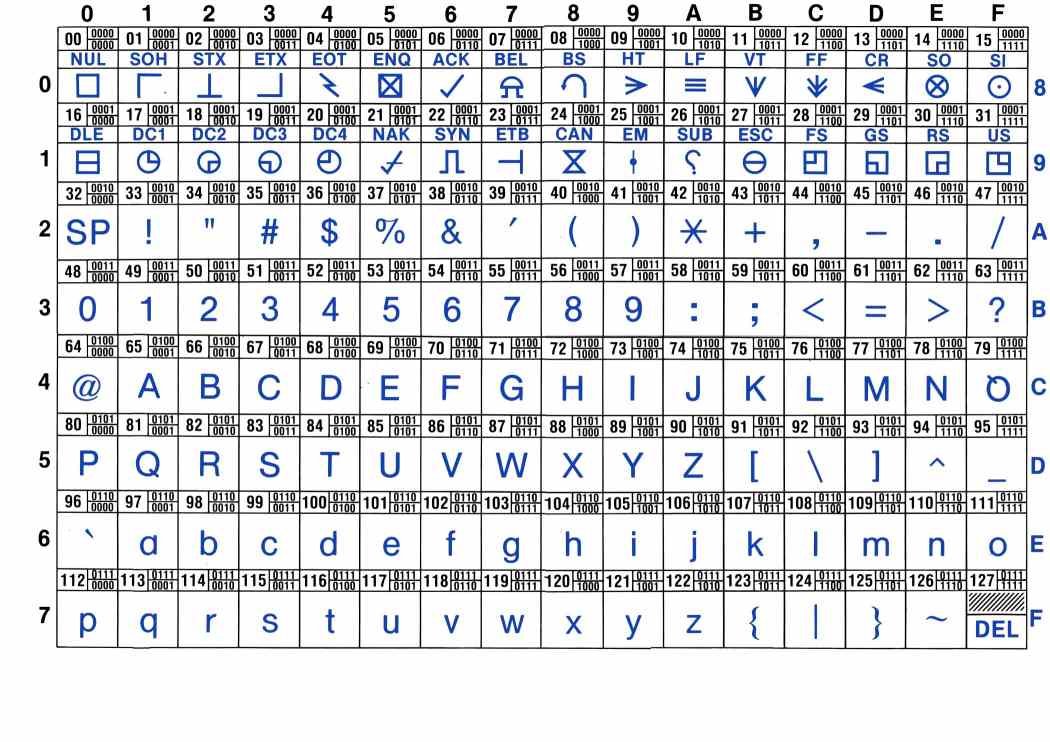


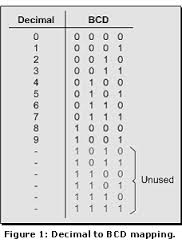


**Hexadecimal**

Hexadecimal refers to a base-16 digit, which consist of 16 numbers and letters; 0 to 9 is numbers and A to F is letters. Hexadecimal system is easier to convert to binary system because it gives you every byte (8 bits). Hexadecimal is easier to read than binary numbers. Before every hexadecimal number, *0X* prefix or an *H* suffix is placed. Hexadecimal numbers refer to them as this: *0x*7AE3. This converts to **0111 1010 1110 0011**. After every 4-digits, you have to include a space after it. This image represents how binary, denary and hexadecimal is calculated. The pattern for binary is simple. It starts from zero and each time it doubles. This is simple and something to remember if you are calculating it.

**ASCII**

The American Standard Code for Information Interchange (ASCII) is an American industry scheme, which is based on the English alphabet. It assigns letters, numbers and other characters. It ranges from 0-9 numbers, A to Z letters and different characters that are placed on the keyboard. It encodes 256 (28) configurations in an 8-bit byte. Different bytes are represented for different languages. 216 characters are represented to cover all the languages in the world. This was made to provide a code for every computer. This is used to code computers. An example could be C, the code for it is 01000011. Referring to the image below, it demonstrates how the ASCII standard table looks like. It looks complicated, but I will break it down for you. The symbols represent the keyboard keys, slightly above it, it shows the code for each one. For example 9, it would be 00111001.

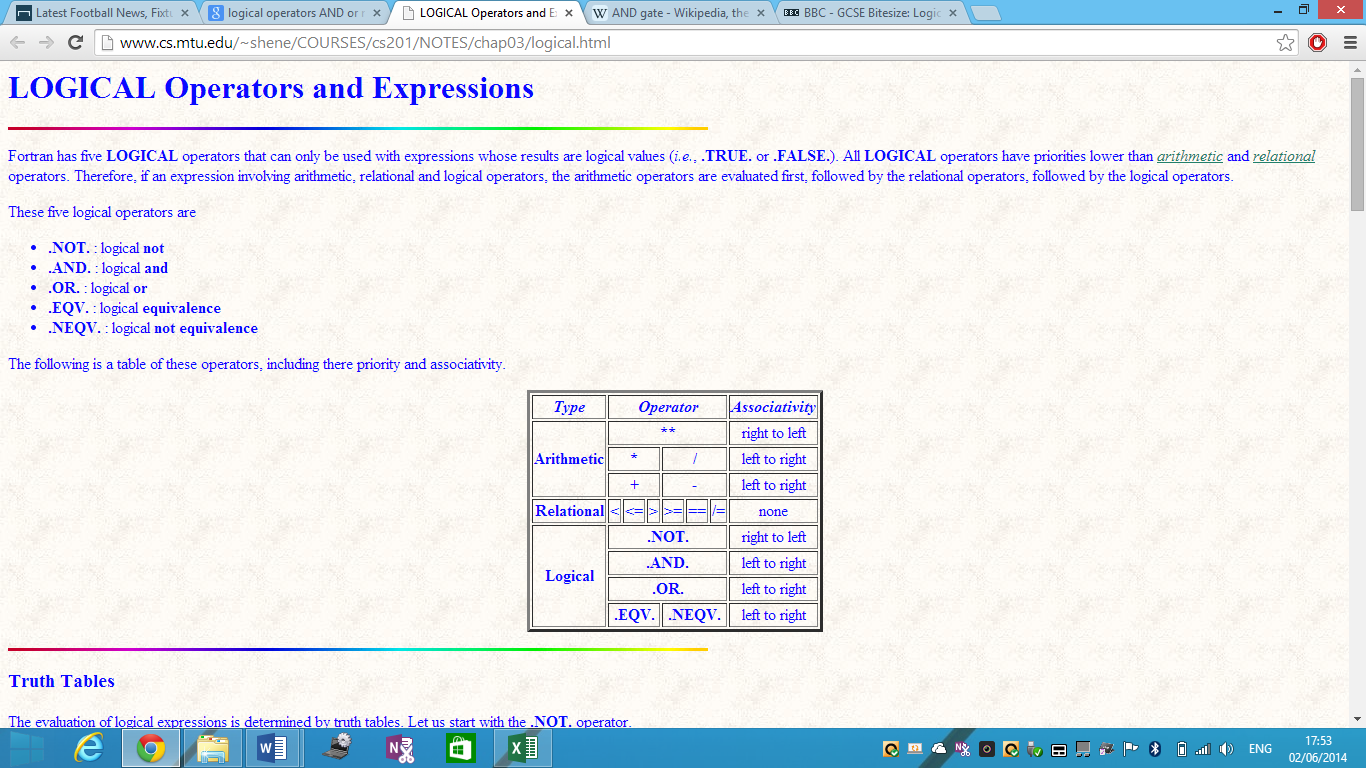
[](http://www.google.co.uk/url?sa=i&rct=j&q=&esrc=s&frm=1&source=images&cd=&cad=rja&uact=8&docid=uJTJejaojDDKdM&tbnid=6R4wMPdLv0KQJM:&ved=0CAUQjRw&url=http://www.edn.com/design/systems-design/4322431/Mind-boggling-math-BCD-binary-coded-decimal-&ei=wGNfU_TtOOyg7AbnpoCYAg&bvm=bv.65397613,d.ZGU&psig=AFQjCNHWjR-ZZhcCZ1FpM5QzvX2jJ5O4Ug&ust=1398846777282857) **Figure 1.1**

**BCD**

Binary Coded Decimal (BCD) is system where each decimal number is represented by 4-digits. For example, 324 would be represented as: 0011 0010 0100. Referring to figure 1.2, it demonstrates a straight pattern in our BCD base-10. Each time, the Os and 1s are doubling and so are the ones. This is how to remember the BCD if you are coding a computer.

Figure 1.2

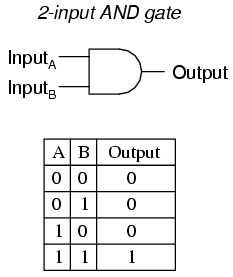
**Logical operators: AND; OR; NOT; other e.g. exclusive OR**

Logical operators are used in case of arithmetic, relational operators and logical operators. If these expressions are involved, they would be used followed by the logical operator. Referring to figure 1.3, it demonstrates how the logical operator operates. They are five logical operators, but I am going to be talking about the main three:

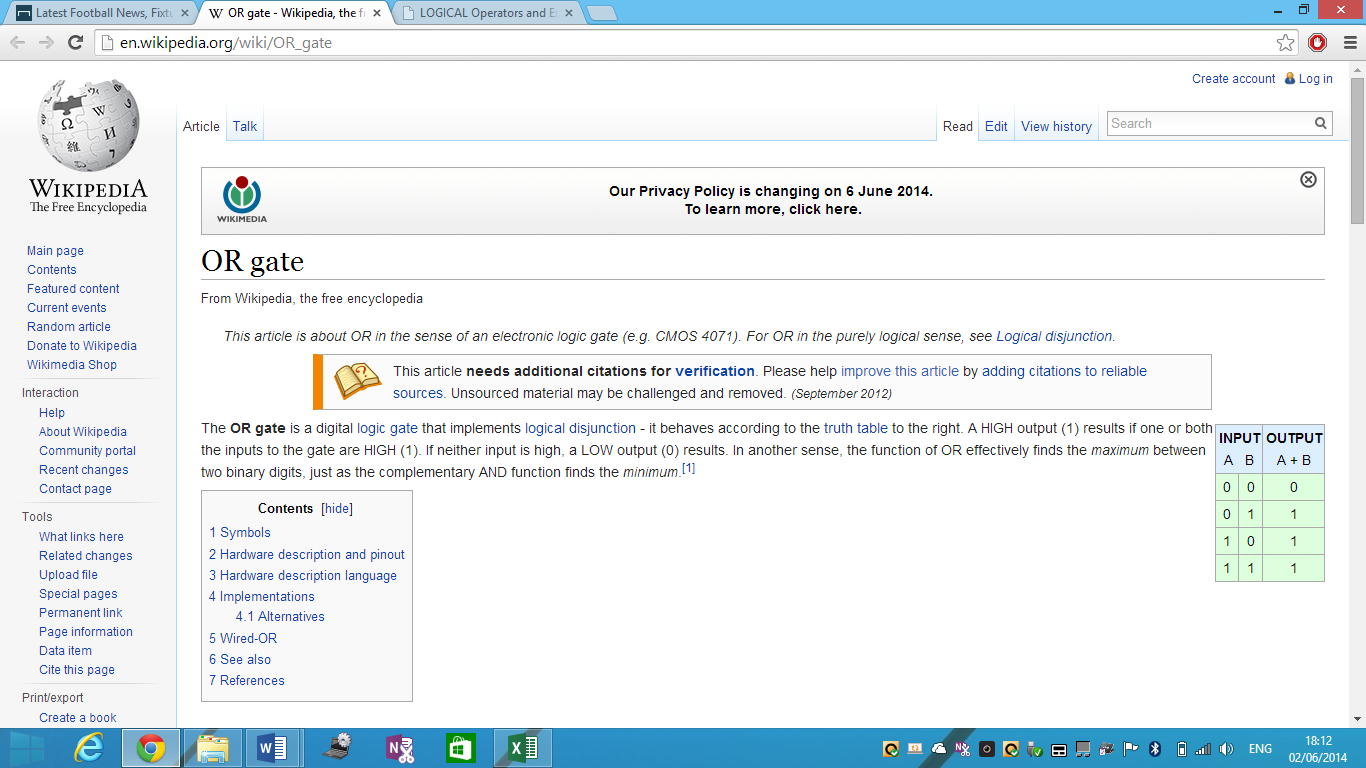
* AND Gate
* OR Gate
* NOT Gate
* Exclusive OR

**Fig 1.3**

**AND Gate**

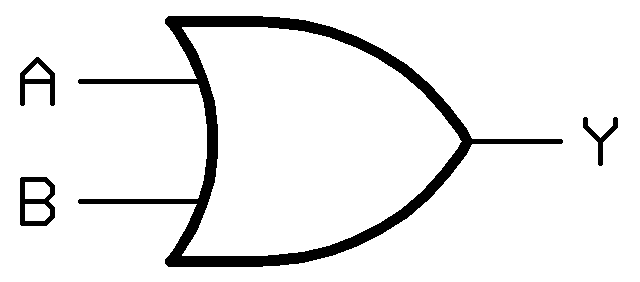
The **AND Gate** involves either if the two numbers are high. A high number is one and a low number is zero. Therefore, if both numbers are one, it would be an AND result. Symbols are represented for each gate and all of them are different. As you can see for figure 1.4, it shows how AND Gate works. The symbol is also present. Both of the inputs are inputted and if they are the same, they would be high (1). The table shows how it works. As shown above, any binary code is 4-digits, which is why there is four digits.

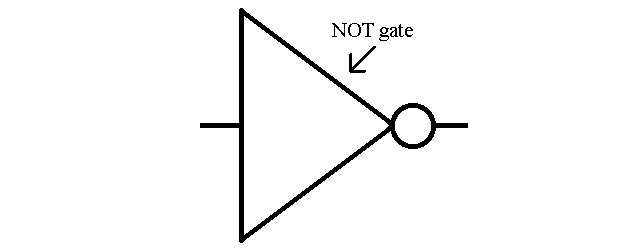
**Fig 1.4**

**OR Gate**

The **OR Gate** is different from AND Gate. Each gate acts similar, but the output and symbols are different. The OR Gate consists of the same adding as AND Gate, but they are different when they both are added up. An example could be that if A = zero and B = one the output would be one. In addition, if both of the numbers are high (1), the output would be zero. Both of the images show how OR Gate looks like and how they are shown.

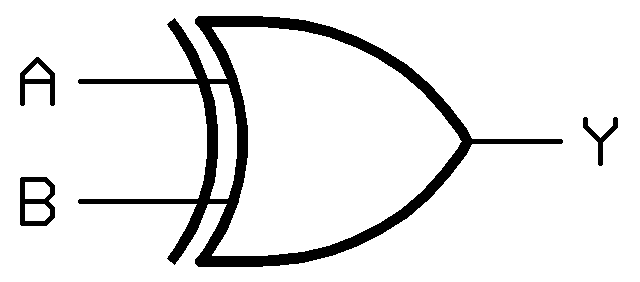
**Fig 1.5**



**NOT Gate**

The **NOT Gate** operates differently. This is the one that is different from OR Gate and AND Gate. The gate sends out the one as an output and if it receives zero, it puts that as an input. This gate can be used on integrated circuits.

**Exclusive OR**

Exclusive OR is another additional logical operation that operates which output is TRUE or FALSE. It can be written as ‘XOR Gate’. The figure 1.6 shows how Exclusive OR looks like. Both of them are inserted and it can tell whether it is false or true.

**Fig 1.6**

**Integer numbers**

Integer numbers are used in binary. It is a single number, but the main ones that consist in binary is zero and one, which is calcified as an integer number. Not only binary, but also the whole system that uses integer number. It could be the base-2 system, base-10 system and any other system that uses integer number. The base-2 and base-10 system uses not only 0s and 1s, but numbers that consist of up to 10.

**Fixed-point numbers**

Fixed-point numbers are numbers that represents usually base-2 or base-10. This is different from integer numbers because an integer is just a number. Fixed-point numbers are represented as such. For example, 1.23 can be represented as 1230 in a fixed-point data type. It is very different to integer number. They are more examples that are different, but I only named one of them.

**Floating-point numbers**

Floating-point is a method that represents an estimation used for real numbers. The real number can be anything – such as minus or a positive number and it can consist of the base used. It uses a method such as 3\*10. This is just an example of it. This is used to scale numbers.