## Converting numbers from Decimal to Binary

Let me show you a way to convert a number from decimal format to binary format. We divide the number by 2. We write the remainder as a digit of the binary number, then repeat the process with the quotient (answer). Let's do an example using the number 25:

| | | | | | | |
|---|---|---|---|---|---|---|
| 25 | ÷ | 2 | = | 12 | remainder | 1 |
| 12 | ÷ | 2 | = | 6 | remainder | 0 |
| 6 | ÷ | 2 | = | 3 | remainder | 0 |
| 3 | ÷ | 2 | = | 1 | remainder | 1 |
| 1 | ÷ | 2 | = | 0 | remainder | 1 |

You will see that the remainders give the digits of the binary number: $25_{10} = 11001_2$.

Why does this work? It uses the fact that an odd number will always have a “1” in the ones column when written as binary and an even number will always have a “0” in the ones column when written in binary. We are effectively asking “is the number odd or even?”, then subtracting the binary digit, dividing by 2 to move to the next column and repeating the process.

Let's relate this back to the explanation of bits in the last lesson. A bit is the answer to a yes/no question. It can also be used to represent the numbers 0 and 1. If we want to represent more complex information, we must use more that one bit. This is another way of saying that we turn the complex information into a collection of answers to yes/no questions. The binary numbers above can be viewed as a collection of answers to yes/no questions. For example. the number $11001_2$ can be viewed as answers to the following four questions:

Does it contain a 1? Yes.
Does it contain a 2? No.
Does it contain a 4? No.
Does it contain an 8? Yes.
Does it contain a 16? Yes.

Note: 5 questions corresponding to 5 bits and the digit 1 become a yes and the digit 0 becomes a no. Compare the yes/no answers with the 1/0 remainders above.

# 24/7/2018 Lesson 5: (Term 3, Week 1)

## Boolean Logic

In past lessons we have learnt about bits (yes/no decisions), how to use multiple bits to represent more complex information and binary numbers. Today we learn about logic.

There are three basic logical operators: AND, OR and NOT.

Compare these to the arithmetic operators that you are already familiar with in maths: +, −, × and ÷. These operators operate on numbers, and allow us to write equations involving numbers. For example: 3 + 4 = 7. The logical operators operate on **bits** and allow us to write equations involving bits. For example: true AND false = false.

The AND and OR operators take two inputs, just as × and + (and the other arithmetic operators) do and produce one bit as output. The NOT operator takes one bit as input and produces one bit as output.

Each logical operator is defined by a **truth table**. A truth table is just a table that lists all possible combinations of inputs, and what the output is for that combination of inputs. A two input function can have four input combinations, so there are fours rows in its truth table. (Each bit can be in 2 states and there are 2 inputs, so the number of combinations is 2×2=4.) A one input function, such as NOT, will have two rows in its truth table. The truth table for each of the basic logic operators is below. Note how every combination of inputs that you can think of appears on the table.

| A | B | A AND B |
|---|---|---|
| F | F | F |
| F | T | F |
| T | F | F |
| T | T | T |

| A | B | A OR B |
|---|---|---|
| F | F | F |
| F | T | T |
| T | F | T |
| T | T | T |

| A | NOT A |
|---|---|
| F | T |
| T | F |

Exercise: Build a circuit for each of AND, OR and NOT using switches and LEDs. Get the students to draw a truth table with the output column blank. Give the circuits to the students and allow them to experiment with them and fill in the output column for each table. Use the names A, B and C for each column. Use T and F for True (switch closed or light on) and False (switch open or light off).

Can you see, based on the truth tables, where each operator gets its name from?

AND: The output is is true only if A is true AND B is true.

OR: The output is is true only if A is true OR B is true.

NOT: The output is true if the input is false (not true), and false if the input is true (not false)
The names of the logic operators comes from the normal English usage of the words and, or and not.

Remember the three circuits that you just used to write down each truth table? As you can see, we can implement these logical operators as electrical circuits. These circuits are called **gates**. Computers contain lots of these gates, and you can even view a single gate circuit, like the one in my hand, as a very simple computer.

Here is an example:

Imagine that you are allowed to have chocolate if Mum AND Dad say you can have chocolate. You can use an AND gate to compute if you are allowed to have chocolate. Ask Mum if you can have chocolate. If she says yes, push one button on the AND gate circuit. Ask Dad if you can have chocolate. If he says yes, push the other button on the AND gate circuit. If the output lights up then you get chocolate!

Now the rules have changed: you can have chocolate if either Mum OR Dad agree to it. This time you use the OR gate, just like you did above. If the output lights up then you get chocolate, because at least one of your parents said you could!

Now imagine that there is only one piece of chocolate, so you can only have chocolate if your brother/sister does not have chocolate. Push the button if your brother of sister is eating chocolate. If the output of the NOT gate lights up then you get chocolate!

A computer contains lots of these logic gate circuits (perhaps billions of them). They are connected so the output of one gate can drive the input of another gate. To do this, the gates in a computer replace our mechanical switches with type of switch called a **transistor**, which can be electrically controlled, by the output of another gate. Despite this complexity, you can still understand a computer as a collection of simple gates.

Given that we can combine gates to make more complex circuits which do more complex things, here is a fact: We can build any logic circuit by using only AND gates and NOT gates. Alternatively, we can build any logic circuit by using only OR gates and NOT gates. The idea is that you need only two basic circuits, like the ones we have here, then you just use lots of them, over and over again. These basic, simple to understand, logic gates are building blocks with which you can build a seemingly complicated computer.

## Using Logic to do Arithmetic

Now let us change the representation of our bits: replacing false (F) and true (T) with 0 and 1 respectively. Remember that we can use any two symbols to represent bits.

The above truth tables now become:

| 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 |