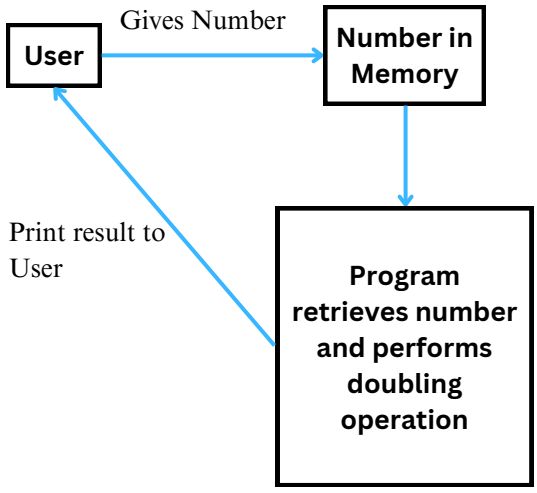
Computer Memory and Binary System

Introduction to Memory

In programs, we need data. The lifespan and scope of this data varies according to its purpose. **Lifespan** refers to the period of time we want data to exist. **Scope** is a reference to the level of access that our data has to other pieces of data.

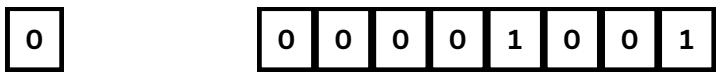
Consider wanting to write a simple program that asks a user for a number, doubles the number and then returns it to the user.

We can think of this as taking the data from the user and then putting it in its own little box inside of the computer's memory. Then we want to retrieve the number to perform the doubling operation on it before printing it to the screen for the user to see



Computer Memory Intro

To better understand how these processes are actually done, we want to consider a computer's memory which can be thought of as little boxes, each holding a single *state* as ***on*** or ***off***. Each box is a **bit** and a group of 8 bits is called a **byte**. 'On' can be characterized as a **1** and 'Off' can be characterized as a **0.**



Bit **1 Byte**

What do these 1's and 0's do?

The 1's and 0's are used to construct numbers and letters and everything we see in our programs. Grouping bits into bytes and then grouping bytes gives us sections that can be used for different data types. To fully understand them, we need to consider a different number system.

Number Systems

Did you know that the decimal number system (0 - 9) is not the only number system? There are many other systems that we want to be aware of and here we look at the **Binary Number System**

Binary Number System

In the **Binary Number System**, the only digits used are **0** and **1** (note how this links to bits in computer memory). These can be used to represent more than just the numbers '0' and '1' as long as we remember that the base of this system is **2** (2 digits allowed in the system). For comparison, the decimal number system has a base of 10 as there are 10 digits that it can use (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). In conversions, the base of the number system is also used

It is crucial to understand the **powers of 2** for this system to make sense

$2^0 = 1$	$2^4 = 16$
$2^1 = 2$	$2^5 = 32$
$2^2 = 4$	$2^6 = 64$
$2^3 = 8$	$2^7 = 128$

Consider a byte that looks like this:

0	0	0	0	0	0	1	1

This is not the number 11

Bytes hold numbers of the Binary System, but we can determine the Decimal number being represented if we consider what each bit (from right to left) represents by considering the powers of 2.

These are the 'place values' represented by each bit within a byte and the **max equivalent** decimal number represented by each bit within a single byte

Place Values →

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Max Decimal Equivalent →

128	64	32	16	8	4	2	1

Now if we remember that these bits can be turned on (1) and turned off (0), we can use them to create numbers.

Start with small numbers first:

Decimal	Binary	Binary in a Single Byte
0	0	0 0 0 0 0 0 0 0
1	1	$2^0 = 1$ 0 0 0 0 0 0 0 1
2	10	$2^1 = 2$ 0 0 0 0 0 0 1 0
3	11	$2^1 + 2^0 = 3$ 0 0 0 0 0 0 1 1

Take note that if multiple bits are turned on, then their total value is equal to the sum of all bits that are turned on

Consider a byte that looks like this: 0 0 0 1 0 1 1 1

To determine the equivalent decimal value of this number, we can use our place value comparisons:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	1	0	1	1	1

We calculate as: **Value** = $2^4 + 2^2 + 2^1 + 2^0$

Value = $16 + 4 + 2 + 1$

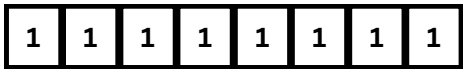
Value = 23

NOTE: In the above examples the word 'decimal' is referring to the **Decimal Number System** and not to Decimal Numbers such as 1.2467

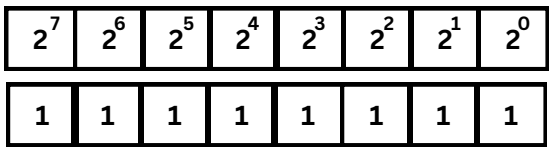
Remember that we want this section to link to Memory which will form a foundation for our Data Types to come in later notes.

Byte - Maximum Value

Now that we know how to convert the Binary Value held in a byte to its decimal equivalent, a great next step is to prove the maximum value that a single byte can hold.



Using our place values:



We calculate as: **Value =** $2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$

Value = $128 + 64 + 32 + 16 + 8 + 4 + 2 + 1$

Value = 255

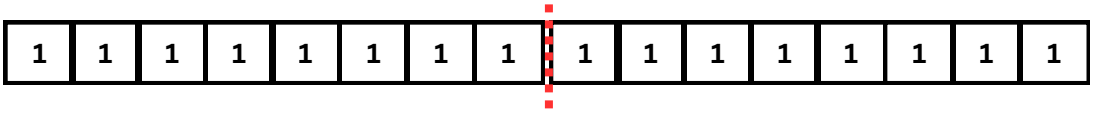
The maximum value for a byte is 255

What about numbers larger than 255?

The above activity demonstrates to us that a single byte can hold a maximum, decimal equivalent of **255**. This is a small number and is not always suited to the size of numbers we may need a program to work with. We need a way to create larger values (more specifically a way for the computer to store larger numbers in its memory)

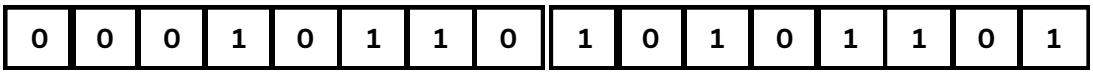
It turns that by combining bytes together, we can extend the number of bits available to hold a number

Consider two bytes next to each other:

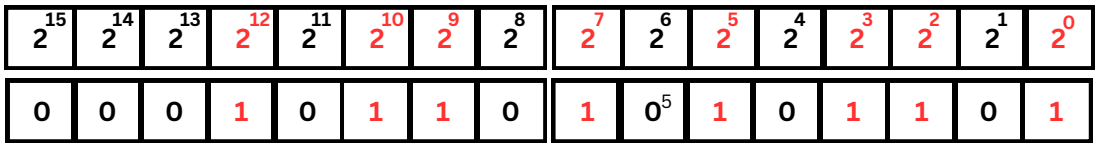


Now we have more bits, meaning we can store larger powers of 2 and therefore build larger numbers

Worked Example, what number is represented by the two bytes below?



Using place values:



Value = $2^{12} + 2^{10} + 2^{9} + 2^{7} + 2^{5} + 2^{3} + 2^{2} + 2^{0}$

Value = 4096 + 1024 + 512 + 128 + 32 + 8 + 4 + 1

Value = 5805

This demonstrates to us that by adding or linking more bytes together, we can create larger numbers. The more bytes we add, the larger the number is that the computer's memory can hold

Memory Usage and Limitation

Now we must consider what all of this means for a computer's memory and why memory is such an important factor when designing and running programs.

As we have seen - the larger the number, the more bytes of memory it needs. Consider that a computer's memory is **not infinite**. It only has so many bytes that it can allocate.

In addition, making a program:

- unnecessarily store values (or duplicates)
- store values in excessively large data types or data structures
- perform actions in an inefficient manner

makes it perform tasks with reduced performance and excessive memory use. As programmers we always want to make our programs efficient. It is worth noting that learning to program efficiently typically comes after learning to create a solution to a problem in the first place - i.e. when starting to program, focus on being able to write code that solves a problem and then learn to make it efficient. With time, being efficient becomes second nature (but we are **life-long learners** as programmers)