Data Representation

The following summary points are related to the learning objectives in the topic introduction:

- Data representation as positive binary numbers (using up to 32 bits) and negative binary numbers (using Two's Complement with up to 8 bits)
- Familiarisation and conversion between binary and decimal numbers
- Representation of numbers using floating point technique and range depending on size of mantissa and exponent
- Conversion of binary numbers to and from bit, byte, kilobyte, Megabyte,Gigabyte,Terabyte
- Unicode and limitations of ASCII when representing character sets
- Bit map methods of graphic representation including greyscale and colour
- Relationship of bit depth and colour (up to 24 bits)
- Vector graphic methods of graphic representation (using objects)
- Relative advantages and disadvantages of bit mapped and vector graphics
- Relationship between the bit depth and file size
- The need for data compression with outline of methods.

Why binary?

- All the logic circuits used in digital computers are based upon two-state logic. That is, quantities can only take one of two values, typically 0 or 1.
- These quantities will be represented internally by voltages on lines, zero voltage representing 0 and the operating voltage of the device representing 1.
- The reason two-state logic is used is because it is easy and economic to produce such devices.
- A switch that is ON has a voltage level of 5 Volts and a switch that is OFF holds a voltage level of 0Volts.
- It is convenient for us to use the Base 2 (Binary) number system to consider that ON represents Binary 1 and OFF represents Binary 0 (i.e. a two state system).
- Thus computers are known as two state (binary) machines which has advantages such as simple circuits (only two voltage levels to maintain), fewer rules of arithmetic along with compatibility with magnetic and optical storage (e.g. two states).

1 **Bit** is a	1 or 0
1 **Byte** (**b**) is a group of =>	8 **Bits**
1024 Bytes is =>	1 **Kilobyte** (1 **Kb**)
1024 Kilobytes is =>	1 **Megabyte** (1 **Mb**)
1024 Megabytes is =>	1 **Gigabyte** (1 **Gb**)
1024 Gigabytes is =>	1 **Terabyte** (1 **Tb**)

Number Representation

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0 <= each increasing place value is a multiple of **2**

128	64	32	16	8	4	2	1	
0	**1**	**0**	**0**	**1**	**1**	**1**	**0**	=> gives us the decimal number **78**

Integer Numbers

Integer numbers are all the **positive** and **negative whole numbers** - e.g. **-4, -3, -2, -1, 0, 1, 2, 3, 4,** ... etc.

- for **8 bits** the **number range** would be **0000 0000** to **1111 1111** which is **0** to **255** **(2^8-1)**

- for **16 bits** the **number range** would be **0000 0000 0000 0000** to **1111 1111 1111 1111** which is **0** to **65535** **(2^{16}-1)**

How does the computer represent Negative Integer numbers?

Using a **Sign Bit** - this is where the **most significant bit** (**msb** - the leftmost bit) in the number **represents** the **sign** of the number while the remaining bits represent the **size** of the number.

e.g. a **msb** of **1** could indicate that the number is **negative** while a **msb** of **0** could indicate that a number is **positive** => **1000 0011** could mean **-3** while **0000 0011** could mean **+3**

Problems

- does not give us a **symmetrical** number system about zero where each number in the range differs from the previous number by 1.
- This system also gives **two** values for **0** i.e. **0000 0000** and **1000 0000**
- Normal arithmetic operations do not give correct results.

Two's complement form

Changing a number to two's complement form :- e.g. => **0001 0111** **represents 23**

1) change all 1's to 0's (this is the **one's** complement) => **1110 1000**

2) add 1 to the result (this is the **two's** complement) => **1110 1001** **represents -23**

Advantages

- Gives us a **symmetrical** number system about zero where each number in the range differs from the previous number by 1.
- This system has one value for **0**
- Normal arithmetic operations give correct results.

Floating Point (Real) Numbers

This is the way that **very large numbers** and numbers with **decimal fraction parts** are **represented** (**Real** numbers)

Decimal numbers can be expressed in **standard form** as follows :-

356.78 => 3.5678 x 102

Standard form gives us a **compact** way of expressing **very large** and **very small** decimal numbers.

In **floating point format** a number such as **356.78** would be expressed slightly differently as **0.35678×10^3**

The **0.35678** part of the number is called the **mantissa** and the power of **3** is called the **exponent (power)**.

Similarly with Binary numbers :-

e.g. :- the **binary** number **11.0101** would be **represented** in **floating point** form as :-

sign bit					mantissa			sign bit	exponent	
+	.	1	1	0	1	0	1	+	1	0

In the previous the **exponent** is binary **10** which has the value **2** in decimal - indicating that the actual **position** of the point in the **mantissa** is **two places** to the **right** (due to the positive exponent).

- **Increasing** the number of bits for the **mantissa increases** the **accuracy/precision** of numbers that can be represented. (**and decreases the range**)

- **Increasing** the number of bits for the **exponent** increases the **range** of numbers that can be represented. (**and decreases the accuracy/precision**)

Text Representation

All the **Text Characters** in the **character set of a language** which includes **letters**, **digits**, **spaces**, **punctuation** and **non printing control characters** are each **represented** by a **unique binary code** (commonly an **8 bit** code- but see **Unicode** below)

ASCII

Every character in a character set uses **8 bits** (**1 byte**), this sentence would use **104** bytes.

A **common standard** is the **ASCII** code (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) which was initially designed for **communications** purposes.

- With an **accepted common standard** such as **ASCII** in place, **hardware** and **software** manufacturers know that their products can almost universally be used for the **transmission of data**.

Due to the **International nature of the Internet** it has been necessary to define the **Unicode standard**.
This is **more comprehensive than ASCII** to represent **foreign alphabets and character sets** such as Japanese and Arabic.

Unicode

This is a **16-bit code** which, in its present form, can **represent up to 40,000 characters** (more will be added - up to a maximum of 65,536).

Unicode-aware applications, e.g. Office 2000 allows you to edit and share documents in up to **80 different languages**.

Disadvantage

- Documents saved in **Unicode format** take up **twice as much space** as ASCII documents.

Graphics Representation

All images that are **displayed** on a computer **screen** are made up of **pixels** (picture elements). A **pixel** is the **smallest point** or **dot** on a computer **screen** display. **1 bit** (which can take values **0** or **1**) is used to display **two** colours only - **e.g.** black and white

For **more colours more bits per pixel** are needed - (you might recall a table similar to this on the introductory page).

Number of bits per pixel	Number of colours available
1	2^1 => **2 colours**
2	2^2 => **4 colours**
3	2^3 => **8 colours**
4	2^4 => **16 colours**
.....	.....
8	2^8 => **256 colours**
.....	.....
24	2^{24} => **16.7 million colours**

etc.

Some computer systems can display **16.7 million colours** , i.e. 24 bits per pixel => 2^{24} colours

Although **all images** on a computer screen are made up of a **grid or array of pixels**, i.e. the screen display shows a bitmapped image - note that **the images themselves** can **exist in different forms before they are displayed**. (e.g. vector images)

The **three (RGB) colours** are **each** represented in effect as an **8-bit greyscale** which defines their hue/saturation/lightness, (hence **3*8=24 bits**)

When **displaying images** on screen, it is very important to consider **screen and video RAM**. If we assume that a monitor is capable of displaying a full screen image at a maximum resolution of **1600 x 1200 pixels**, using **16.7 million colours**, this full screen image takes up **(1600*1200*24)/8 bytes, i.e. 5.76 Mb**.

This does not **necessarily** imply that the screen image occupies that **full amount of main memory**, since modern **graphics cards** come equipped with **generous amounts of their own RAM**. While 'good' graphics cards have **16**, **32**, or even **256 Mb** of RAM, note that **older cards with only 4 Mb are not capable** of displaying **photo-realistic images**.

- **storage overheads** for **3D images, rotation, lighting and moving video**, the memory requirements will be **much more than** the basic 5 - 6 Mb.
- **very memory hungry** - the **more detail** and **colour** displayed in the image on screen => the **more memory is needed** to store the image. (e.g. **higher resolution images** use **more** pixels)

E.g., an **A4 page** with a detailed **colour** image could use **many megabytes** of **storage** space.

Example of bit mapped image

Bit mapped images are produced by **Paint packages** or **Scanners/Cameras**

- If you **enlarge** these **images,** the effect will produce a **blocky image,** as you are simply **enlarging each element** that makes up the image.
- Also if you **move** one **bit mapped image over part** of another **bit mapped image** on screen, it will cause the foreground image to **overwrite** the background image as they are both trying to occupy the **same area of screen memory**.
- Editing individual images is **difficult and time consuming** as it has to be processed at **pixel level**!
- As a **bit mapped** image is **represented** as an **array of bits in memory**, when a bit mapped image is printed it **can only produce** the image **resolution** on paper that exists in memory.
- (e.g. an image which has been created at **300 dots per inch** can only reproduce a printed image of **300 dots per inch** resolution regardless of the **quality** of the printer.)

Vector Graphics (Object Oriented graphics) Representation

Vector images are **represented** by a **set of vectors** (**attributes**) which **describe** the image. These **vectors** which describe the image are used to **draw** the image **every time** it is used, moved or resized in a document. **Draw packages produce vector images.**

- **Vector** images are **limited** to **fairly simple shapes** which are made up of **simple** geometric shapes (squares, rectangles, circles, triangles, etc).-

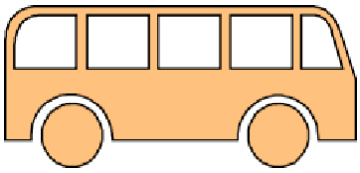
For a **circle** the **vectors** might be -

centre XY, radius R, line thickness T, color C, fill F, etc.

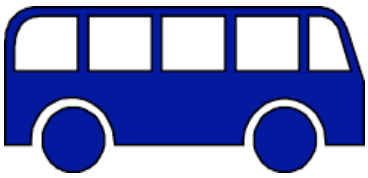
- Vector data is efficient with storage space as the vectors which describe an image take up very little storage space.
- The more complex an image the more of these basic vectors will be required to store the image, which will have memory implications.
- Each time a vector image is moved, reduced or enlarged on screen it is redrawn very fast.
- The vectors which describe an image take up very little storage space. Vector images can overlap each other on screen and when separated the two images are intact as they are redrawn every time they are moved.
- Vector images will display on screen and print out to the quality of the printer => a 2000 dpi printer will produce a 2000 dpi image. This is because the image is stored as a set of vectors and not an array of bits in memory.
- Scalable character sets (sometimes called vector fonts) can be resized without any loss of clarity as opposed to bit mapped fonts which become blocky when enlarged. The True Type fonts used in Windows are examples of scalable fonts.

Pixel (bit-mapped) *Versus* Vector-Based Graphics

You can see the **relative file sizes** for **bitmapped** and **vector** graphics below :
(The reason for the large file size in the bitmapped version is that the **pixel data for the entire rectangle on which the bus sits** has to be stored.)



Bit-mapped image file size 25.1 K



Vector file size 3.71 K

Vector graphics are the **most economical as regards storage space**.
However, both types of image may be **compressed**, with the **most effective compression ratios** being achieved for **pixel graphics**.

Vector graphics are also **device-resolution independent**

- if displayed or printed on a higher-than-normal **resolution** monitor or printer, an **improvement** in image **quality** will be apparent.
- **Pixel graphics**, on the other hand, will **deteriorate in quality if you enlarge them**, with the original pixels being represented by **larger** and larger **colour** blocks which **distort** the image.

Below you can see **vector graphics on the left**, the larger image being a **resized** version of the smaller. The other two heart images are**bitmapped**. Again the one on the **right** is a **resized** version.

You can see that the **black outline** of the larger bitmap is **thicker** and more **ragged** than the original.

It is not **usually possible** to **isolate** a **bitmap** image from the grid of background pixels, although **GIF** images, while being bitmapped, are stored with **transparency** information which allows the background colour to '**show through**' selected areas of the bitmap.

Both pixel-based and vector images may be **cropped**, i.e. unwanted parts of the image can be removed from their edges. The cropping action is usually **non-destructive**, i.e. you can reveal the cropped portion again by uncropping.