

Computing Studies

Software Design
and Development

Data Representation & Computer Architecture



&



Buckhaven High School



Version 1

Contents

Page 1	How to use this booklet.
Page 2	Introduction
Page 3	Binary
Page 4	Units of Binary
Page 5	Changing from One Unit to Another
Page 7	Using Binary to Store Integers
Page 11	Using Binary to Store Real Numbers
Page 12	Using Binary to Store Text
Page 13	Using Binary to Store Bit-Mapped Graphics
Page 16	Using Binary to Store Vector Graphics
Page 20	Machine Code
Page 21	Computer Architecture
Page 22	Memory Addresses
Page 23	Processor Components
Page 24	The Role of Buses in Processing
Page 25	Variables - How Computer Programs Store Data

How to use this booklet

This booklet has been written to cover the following content in National 4 and National 5 Computing.

	National 4 	National 5 
Low-level Operations and Computer Architecture	Use of binary to represent and store: <ul style="list-style-type: none"> • Positive Integers • Characters • Instructions (machine code) Units of storage <ul style="list-style-type: none"> • Bits • Byte • Kilobyte (Kb) • Megabyte (Mb) • Gigabyte (Gb) • Terabyte (Tb) • Petabyte (Pb) 	Use of binary to represent and store: <ul style="list-style-type: none"> • Integers • Real Numbers • Characters • Instructions (machine code) • Graphics Basic computer architecture: <ul style="list-style-type: none"> • Processor (registers, ALU, control unit) • Memory • Buses (data, address) • Interfaces
Data Types and Structures	String Numeric (integer) variables Graphical objects	String, character Numeric (integer and real) variables Boolean variables 1 Dimension (1D) arrays

The booklet is colour coded as shown above.

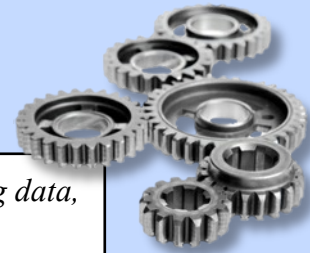
For assessment purposes, pupils working at National 4 level should revise only the N4 content. Pupils attempting National 5 assessments, coursework or final exam should study only N5 content. (N5 pupils may wish to revise N4 content anyway to improve their overall knowledge of the subject.)

Introduction

The word “compute” means to calculate or work out. A computer is simply a device that computes. Calculations on computers involve processing data, for example doing some arithmetic with numbers, sorting a list into order, moving the position of a character in a game.

Here is a dictionary definition of a computer:

Definition - An electronic device for storing and processing data, according to instructions given to it in a program.



A computer is a machine that carries out instructions given to it by a human. The instructions allow it to perform some useful functions as and when it is required. Without instructions (written by a programmer) a computer is a useless box of electronics that does nothing.



The key advantages computers have over humans are:

- computers work faster and more accurately than humans (the video “Fujitsu Motherboard Production” shows how fast and accurately a computer controlled system can work)
- they can store huge amounts of information that they never “forget”.

It might seem that computers can do almost anything. However, here are some other important things to remember:

- computers do have brains (processors) but can’t think for themselves
- they do not have human style intelligence so can’t consider problems
- computers don’t have feelings or “common sense”. This means that there are lots of everyday tasks that humans can perform that computers still cannot.

Task 1 - What can we do that computers can’t?

Identify three everyday activities that a human can perform but computers cannot (or are not very good at).

Write your answers on post-it notes and display them at the front of the class.

Your teacher will then discuss and remove the activities that a computer is capable performing.

Try to ensure your post-its are left on the board at the end.



VS



VS



Binary

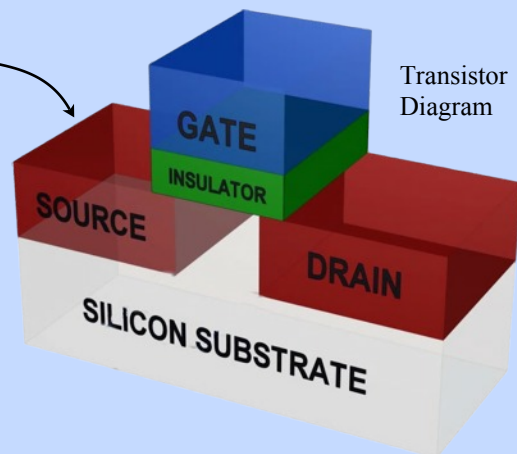
You have used software packages on a computer which handle text (word processing), numbers (spreadsheets) and graphics (photo editing and drawing). You may even have used packages which manipulate and store sound and video data.

In computing, **data** is the term given to the numbers, text, graphics, sound & video that computers process. In order to process it, data must be in a form that computers can understand.

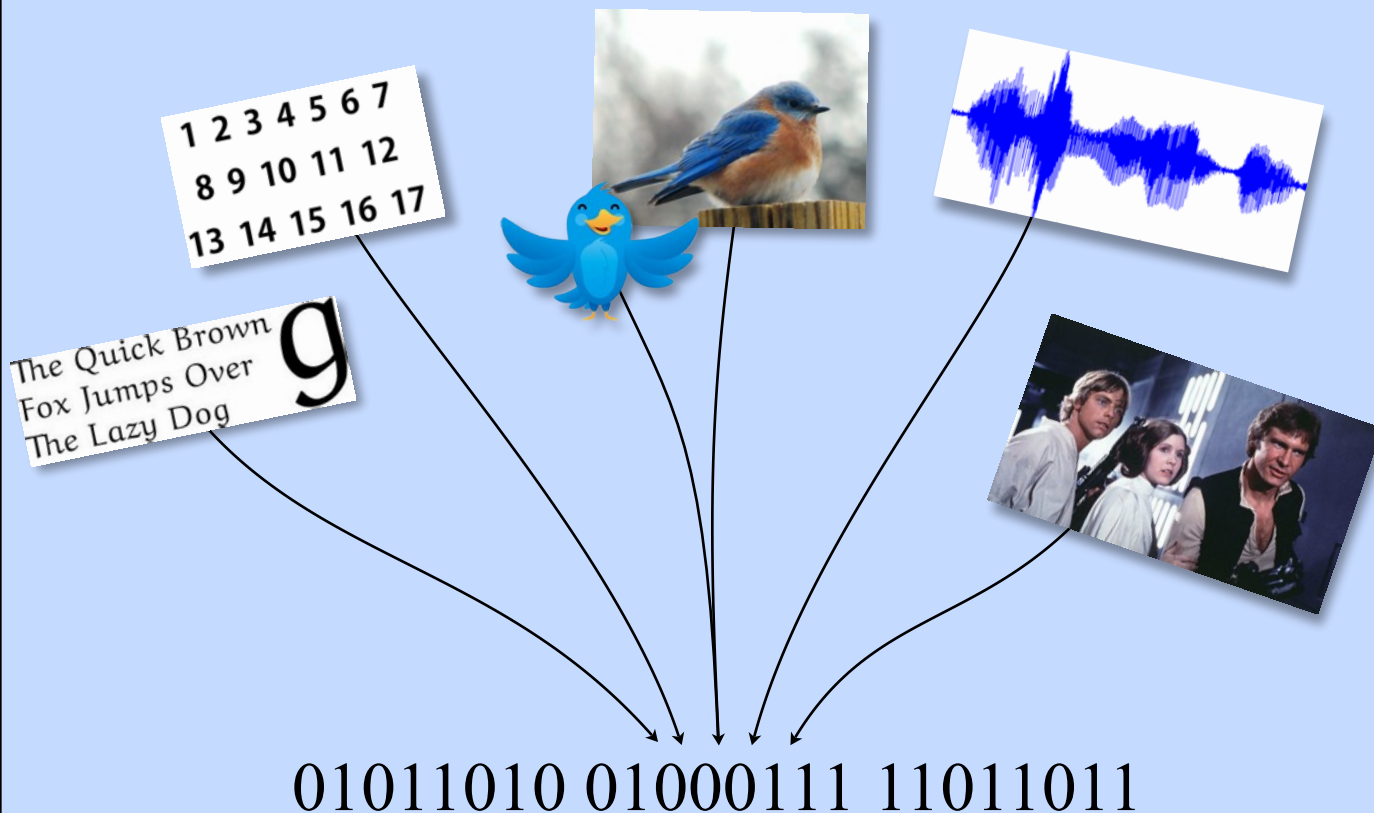
Computers work by switching circuits (transistors) on and off very quickly. All the data in a computer is stored and processed using these switches.

As a switch can only be on or off, computers are known as two-state machines. The numbers 1 and 0 are used to represent the on and off positions of the switches.

Everything that the computer has to do and all the data that it works with and stores is represented using a two digit number system - 1's and 0's. These are called **Binary digITS** (or **BITS** for short).



The following pages will explain how binary is used to store **numbers**, **text** and **graphics**.



Units of Binary

The software we use on the computer and the data we work with is represented as **bits** within the main memory. Early computers, like the 1975, Altair 8800 shown on the right, typically worked with groups of 8 bits (known as a **byte**) at a time.

When referring to memory and storage capacities, **bytes** became the basic unit of measurement. Computers today can process and store billions of bytes every second.

By 2015 it is estimated that 8 Zettabytes (8,000,000,000,000,000,000 bytes) of data will be stored on computer systems worldwide.

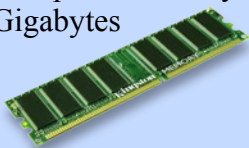


Measurement of Storage Capacity

The storage capacity of RAM, hard drives or any other storage device is usually quoted in Megabytes (MB) or Gigabytes (GB). But what do these terms mean?

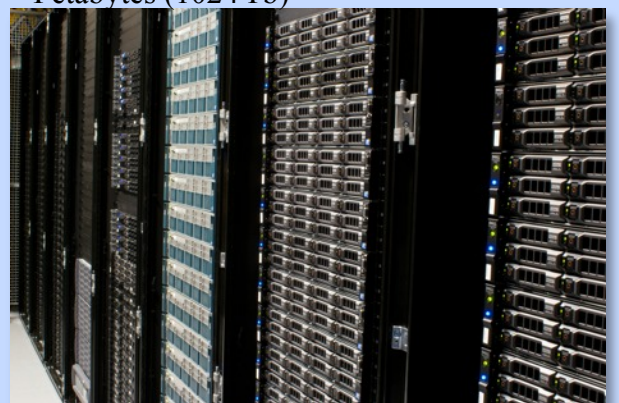
Unit	Description	
Bit	Short for Binary digIT. It is the smallest unit which can be defined in a computer. Bits (1s or 0s) correspond to simple switches being on or off.	
Byte	A byte is a group of 8 bits. Early computer worked with groups of 8 bits or a byte. Today's computers typically process groups of 64 bits (8 bytes) at a time.	
Kilobyte (Kb)	2^{10}	Exactly 1024 bytes. Kilo usually means 1000 of something but in binary 1024 is a round number. Text files and small graphic files are usually quoted in KB.
Megabyte (Mb)	2^{20}	Exactly 1024 Kilobytes. Mega usually means 1 million of something and in this case it is approximately 1 million bytes. Photos and music files are usually measured in MB.
Gigabyte (Gb)	2^{30}	Exactly 1024 Megabytes. The capacity of some storage devices (DVDs, USB Flash Drives) are measured in GB.
Terabyte (Tb)	2^{40}	Exactly 1024 Gigabytes. This measurement is now commonly used with newer hard disk drives, mainframe memory and server hard drives.

Computer Memory - Gigabytes



Mainframe Memory - Terabytes

File Server Storage (banks of hard disk drives) - Petabytes (1024 Tb)



Changing From One Unit to Another

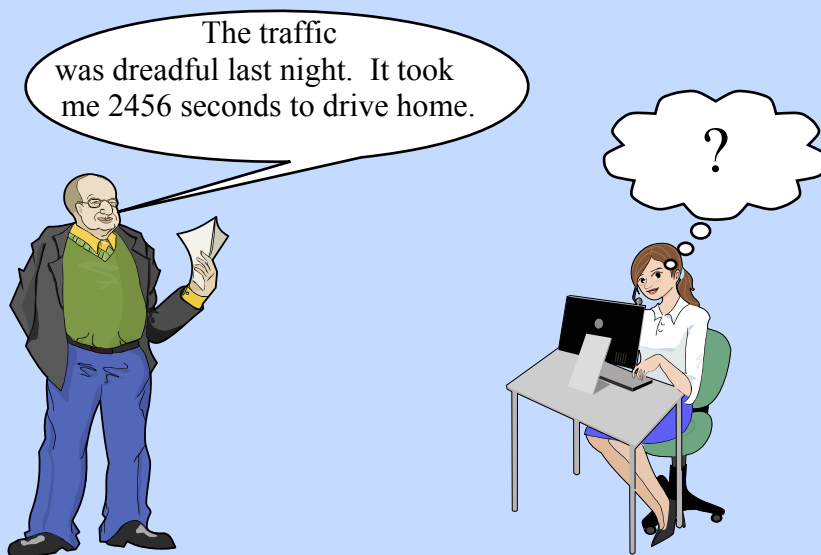
As well as knowing the order of the units (bits, bytes, Kb, Mb, Gb, Tb) it is important, when doing calculations in computing, to be able to change from one unit to another.

For example: A high definition movie might require 1,717,986,918 bytes of storage space.

If you were telling your friend that you had downloaded the above movie last night. You would be far more likely to say that the movie you downloaded was 1.6 Gb in size.

Using appropriate units is important.

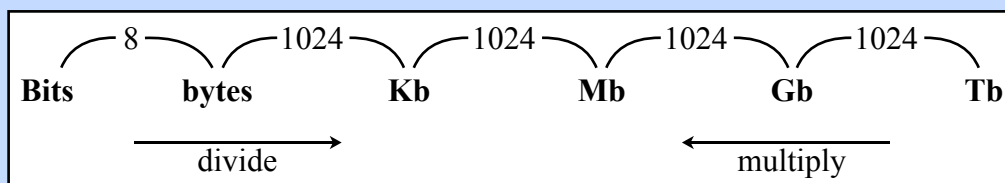
Without the correct units, information does not make sense.



To convert a small unit to a larger one we divide (for example changing bytes to Mb).

To convert a large unit to a smaller one we multiply (for example Tb to Mb)

What you multiply and divide by, depends on the number of places you are moving up or down. Use the chart below.



Kb to Gb would be two places to the right so you would divide by 1024 twice.

Example 1: Convert 4 Mb into bytes

We are moving two steps to the left
 $4 \times 1024 \times 1024 = 4,194,304$ bytes

Example 2: Convert 4096 Gb in Tb

We are moving one step to the right
 $4096 / 1024 = 4$ Tb

Example 3: Convert 3.5 Mb into bits

We are moving three steps to the left
 $3.5 \times 1024 \times 1024 \times 8 = 29,360,128$ bits

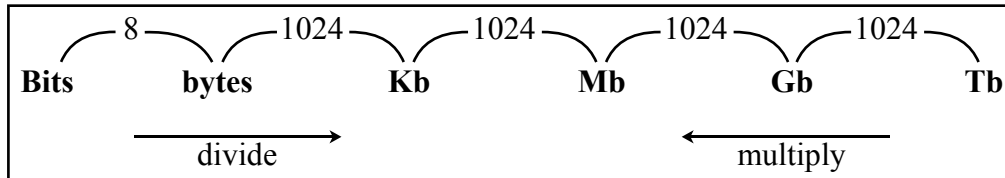
Example 4: Convert 68,719,476,736 bits into Gb

We are moving four steps to the right
 $68,719,476,736 / 8 / 1024 / 1024 / 1024 = 8$ Gb

Task 2 - Practise changing units

In each of the questions below you will have to change the units of the numbers in the questions.

Remember to use the conversion table



Submit your answers to your teacher on a piece of scrap paper.

- Q1 How many bytes are there in 64 bits?
 Q2 How many Kb are there in 8,192 bytes?
 Q3 How many Mb are there in 2 Gb?
 Q4 How many bits are there in 256 bytes?

Sometimes the same questions are worded differently.

- Q5 Convert 4096 Mb into Kb.
 Q6 Convert 1,048,576 Kb into Gb.
 Q7 Convert 4 Tb into Mb.
 Q8 Convert 12 Kb into bits.

The next set of questions require a bit of problem solving as well.

- Q9 Dave is offered two USB flash drives for £10 each. One is 10,240 Mb in size and the other is 12 Gb.
Which one should he buy?
 Q10 Wendy wishes to store 20 graphics, each of which 512Kb in size.
How many Mb of storage will she need to store all the files?
 Q11 Which is larger 163,840 bits or 22 Kb?
 Q12 How many 64Gb USB flash drives would you need to store a total of 1 Tb of data?

Using Binary to Store Integers

Storing any form of data on a computer creates problems. If the only data that a computer can store is 1s and 0s using transistors then how do we store numbers, text, graphics, sound and video?

Let's start by examining how positive, whole numbers (integers) are stored as they are the simplest to implement.

67 4 120344854
231 34670 0

To explore how integers are stored it's useful to first re-visit how we learned to count when we were in Primary 1.

Our number system (called denary) has 10 digits - 0, 1, 2, 3, 4, 5, 6, 7, 8 & 9

When we count we start in the right hand units column increasing the number until we run out of digits.

Number	U
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

The next column in our counting system is 10 for a reason. When we run out of digits we set the Units (U) column back to zero and note that we now have one 10 by placing the digit 1 in the next (10s) column.

We then keep counting.

Number	10s	U
10	1	0
11	1	1
12	1	2
13	1	3
14	1	4
15	1	5
16	1	6
17	1	7
18	1	8
19	1	9
20	2	0

Each time a column fills up we create a new column to store the next number.

We then return to filling up the units column and then the 10s, then the 100s and so on.

Number	1000s	100s	10s	U
999		9	9	9
1000	1	0	0	0
1001	1	0	0	1
1002	1	0	0	2
1003	1	0	0	3

When counting in binary we use exactly the same process but with only 2 digits, 0 and 1.

When counting in Binary it only takes two numbers before our units column is full (there is no 2!).

Number	U
0	0
1	1

↓

As before we create a new column but this time the next column is not 10. Instead the next column is 2. We place a 1 in the new 2s column to show that we have one 2, reset the units column back to 0 and start counting again.

Number	2s	U
0		0
1		1
2	1	0
3	1	1

↓

Every time we fill up all the columns (when every column is 1) we create a new column and set the others back to 0.

Counting in binary looks like this.

Number	16s	8s	4s	2s	U
0					0
1					1
2				1	0
3				1	1
4			1	0	0
5			1	0	1
6			1	1	0
7			1	1	1
8		1	0	0	0
9		1	0	0	1
10		1	0	1	0
11		1	0	1	1
12		1	1	0	0
13		1	1	0	1
14		1	1	1	0
15		1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1

↓

In our counting system (10 digits) each column is 10 times the previous column.

←

100000	10000	1000	100	10	U
--------	-------	------	-----	----	---

In binary (2 digits) each new column is double the previous column.

←

2048	1024	512	256	128	64	32	16	8	4	2	U
------	------	-----	-----	-----	----	----	----	---	---	---	---

At National 5 you are not expected to count in Binary but it's useful to understand the process.

You will be expected to convert a number from binary to denary (our 10 digit system) and back again.

1. Binary to Denary

To convert a binary number to a denary number, simply add up the columns where a 1 appears.

Example 1: Convert the binary number 01100100 into a denary value.

128	64	32	16	8	4	2	U
0	1	1	0	0	1	0	0
	64	32			4		

$64+32+4 = 100$

Example 2: Convert the binary number 11001001 into a denary value.

128	64	32	16	8	4	2	U
1	1	0	0	1	0	0	1
128	64			8			1

$128+64+8+1 = 201$

2. Denary to Binary

To convert a number from denary to binary we reverse the process and place 1s into the columns ensuring that they add up to the number we are looking for.

Example 1: Convert 94 into a binary number.

Begin by looking for the largest column that we can place a 1 into.

128	64	32	16	8	4	2	U
	1						

=64

We now need our remaining columns to add up to 30 ($94-64 = 30$ left). As the 32s column is too large, the next column we use will be the 16s column.

128	64	32	16	8	4	2	U
	1		1				

=80

We now have 14 left ($30-16 = 14$)

128	64	32	16	8	4	2	U
	1		1	1			

=88

Continue the process until you find the remaining columns.

128	64	32	16	8	4	2	U
	1		1	1	1		

=92

128	64	32	16	8	4	2	U
	1		1	1	1	1	

=94

Finally place zeros in all the columns you have not filled.

Answer = 01011110

128	64	32	16	8	4	2	U
0	1	0	1	1	1	1	0

=94

Example 2: Convert 237 into a binary number.

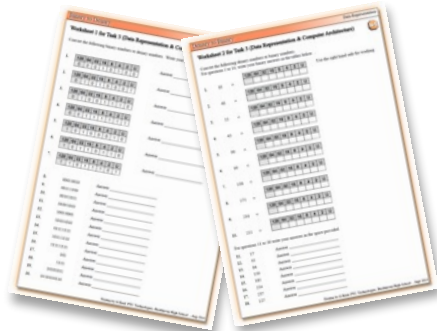
128	64	32	16	8	4	2	U	=128	(237-128 = 109 left)
1									
128	64	32	16	8	4	2	U	=192	(237-192 = 45 left)
1	1								
128	64	32	16	8	4	2	U	=224	(237-224 = 13 left)
1	1	1							
128	64	32	16	8	4	2	U	=232	(237-232 = 5 left)
1	1	1		1					
128	64	32	16	8	4	2	U	=236	(237-236 = 1 left)
1	1	1		1	1				
128	64	32	16	8	4	2	U	=237	Answer = 11101101
1	1	1	0	1	1	0	1		

Task 3 - Binary to Denary & Denary to Binary Practice

Collect the two worksheets titled “Binary to Denary” & “Denary to Binary” from your teacher.

Complete both sheets and submit them to your teacher for marking.

Ensure you show your working where required.



A useful fact to remember about binary is that 256 numbers can be stored using 1 byte (or 8 bits):

00000000 to 11111111

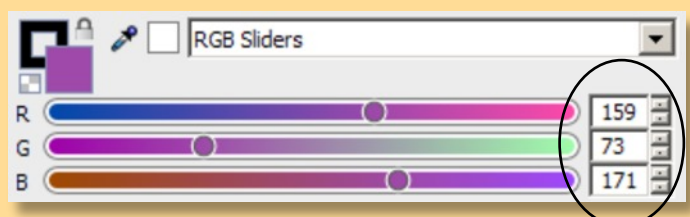
or 0 to 255 = 256 different numbers

A Practical Example of Integer Use

Colours are often stored by a computer using 3 numbers to represent red, green & blue (RGB).

The screenshot on the right shows how a colour can be selected in a graphics application by changing the rgb values.

When the colour is saved, it is stored as three 8 bit binary numbers (each one between 0 and 255).



For example R=159,G=73,B=171 would be stored as: 10011111 01001001 10101011

159 73 171

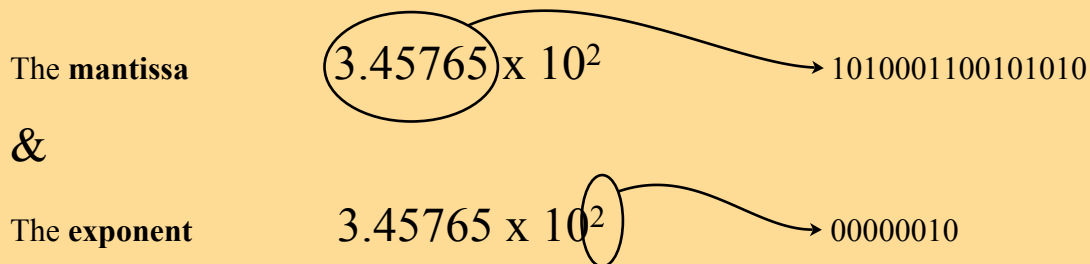
Using Binary to Store Real Numbers

Real number, or numbers with decimal places are stored using scientific notation.

For example, the number 345.765 would be stored as:

$$3.45765 \times 10^2$$

The computer then stores two separate integers with a set number of bits.



The complete number is then stored as one long integer - 101000110010101000000010

Note that the number of bits that a computer uses to store the mantissa and exponent has an effect on the number stored.

Accuracy

By reducing or increasing the numbers of bits used to store the mantissa we can affect the accuracy of the number. With increased bits, more decimal places can be stored.

8 bits	3.45
16 bits	3.45765
32 bits	3.45765234323

Size

By reducing or increasing the numbers of bits used to store the exponent we can affect the size of the number we can store.

4 bits	range of 0-15	10^0 to 10^{15}
8 bits	range of 0-255	10^0 to 10^{255}

What's Normal for Today

A common representation in today's computers uses 32 bits to store a real number split up as follows:

Mantissa - 23 bits

Exponent - 8 bits

Signed Bit (used to store if the number is positive or negative) - 1 bit

Using Binary to Store Text

Storing numbers using binary is easy as binary is a counting system for numbers. To store text characters we have to come up with a different solution.

Task 4 - A System for Storing Text

Split into pairs and collect a piece of scrap paper from your teacher.

Your task is as follows:

1. Design a method of storing a single character (A, v, Z etc) using a pattern of 1s and 0s.
2. Once you've decided how to store your characters, use your method to write a three letter binary message for your partner. Give you partner the coded binary message.
3. Now try to decode each others binary messages.

Could you decode the other person's message?

Unless you are extremely good at decoding messages (and very lucky) you will have discovered that it is nearly impossible to decode the message without knowing the method your partner used.

Task 4 simulates what happened in the early days of computing when methods of storing text were developed. The problem with everyone deciding how each character will be stored is that nobody can understand anybody else's codes. Any text you save can't be viewed by anyone using a different code.

As with many developments in technology, eventually most of the methods died out leaving only a few. From those few, one method now rules.

ASCII (American Standard Code for Information Interchange)

ASCII uses 8 bit binary numbers to represent text characters.

An 8 bit code allows 256 different characters to be stored:

- A-Z - 26 characters
- a-z - 26 characters
- Control Characters (return, tab etc) - 32 characters
- 0-9 - 10 characters
- Punctuation - approximately 20 characters
- Mathematical Symbols - approximately 50 characters

The remaining spaces in the 256 character code are used to store foreign alphabet letters.

234	11101010	ê
235	11101011	ë
236	11101100	ì
237	11101101	í
238	11101110	î

Denary	Binary	Character
51	00110011	3
52	00110100	4
53	00110101	5
54	00110110	6
55	00110111	7
56	00111000	8
57	00111001	9
58	00111010	:
59	00111011	;
60	00111100	<
61	00111101	=
62	00111110	>
63	00111111	?
64	01000000	@
65	01000001	A
66	01000010	B
67	01000011	C
68	01000100	D
69	01000101	E

UNICODE

Sometimes even 256 characters is not enough. Another commonly used standard for storing text is the 16 bit Unicode, capable of storing 65,536 different characters.

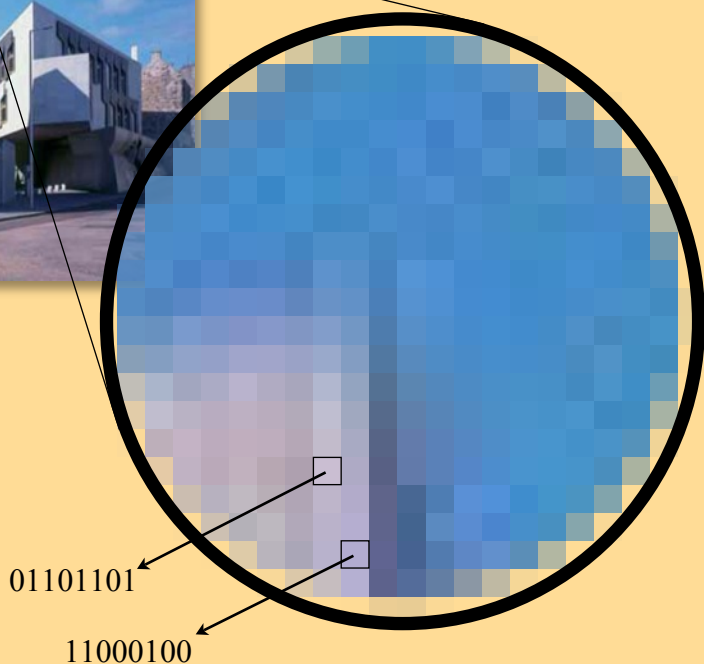
Using Binary to Store Bit-Mapped Graphics

A bit-map graphic is made up of rows and columns of pixels.



Zooming into a bit-mapped graphic allows you to see that each individual pixel has a single colour.

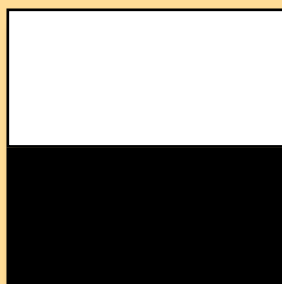
When a bit-map is stored by a computer, each pixel is stored as a binary number. The binary number represents the specific colour of that pixel.



The term “bit-map” comes from the fact that we are mapping each pixel to a sequence of bits.

The number of bits used to store each pixel’s colour is called its “**colour depth**”. The colour depth determines the maximum number of possible colours that each pixel can be.

2 possible colours



8 possible colours



256 possible colours



This is how colour depth works:

If each pixel = **1 bit**

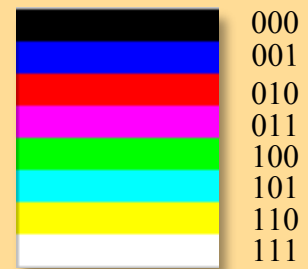
If only 1 bit is used to store the pixel, then only a single 0 or 1 can be used. This would allow to store only two different colours



Black and white graphics use 1 bit colour depth (0 = black, 1 = white)

If each pixel = **3 bits**

Now we can store binary values from 000 to 111 (0-7). With 3 bits, 8 different colours (for example - black, white, blue, red, green, yellow, magenta & cyan) can be stored.



If each pixel = **8 bits**

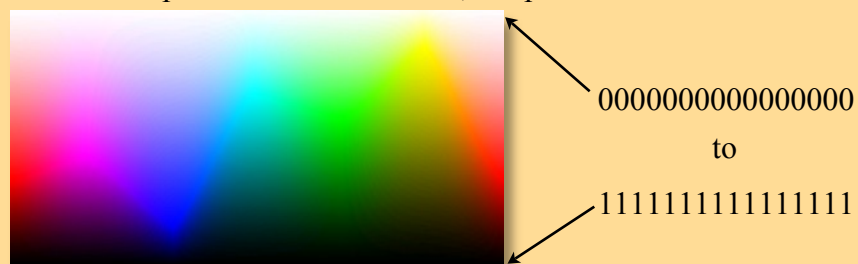
Now we can store 00000000 to 11111111 binary numbers (0-255)

With 256 colours we now have a far greater range.



If each pixel = **16 bits**

A colour depth of 16 bits allows 65,536 possible colours



If each pixel = **24 bits**

24 bit colour depth = 16,777,216 colours. (Too many to show!)

A colour depth of 24 bits is often called "**true colour**" as beyond 24 bit colour human eyes can no longer detect the difference between each individual colour.

If each pixel = **32 bits**

32 bit colour depth = 4,294,967,296 possible colours.

Different types of bit-map files store graphical data using a variety of different colour depths.

- a jpg graphic is always 24 bit colour.
- a gif file is always stored using 8 bit colour.
- a png file can vary its colour depth from 1 bit to 32 bit colour.

Calculating the Storage Requirements of a Bit-Mapped Graphic

To calculate the amount of storage space required to store a graphic we need two pieces of information.

800



1. The **resolution** of the graphic is used to calculate the number of pixels in the image.

$800 \times 360 = 480,000$ pixels

2. The **colour depth** tells us the number of bits used to store each pixel.

24 bits per pixel

We can then calculate the storage requirements of the graphic as follows.

$$\begin{aligned}\text{Storage Requirements} &= \text{Resolution} \times \text{Colour Depth} \\ &= 800 \times 360 \times 24 \text{ bits} \\ &= 11,520,000 \text{ bits} \\ &= 1,440,000 \text{ bytes} \\ &= 1,406.25 \text{ Kb} \\ &= \mathbf{1.37 \text{ Mb}}\end{aligned}$$

Note that the information you need is not always presented to you in a nice easy form.

For example: Consider this question. Calculate the storage requirements of a black and white graphic with a resolution of 5 inches x 3 inches at 300 dots per inch.

This example requires you to look harder for the information you need.

The colour depth is 1 bit - "black and white"

The resolution is 1500 x 900 - "5 inches x 3 inches at 300 dots (pixels) per inch"

Task 5 - Calculating the Size of Bit-mapped Graphics

Collect the worksheet titled “Bit-Mapped Graphics Calculations” from your teacher.

Complete the sheet and submit it to your teacher for marking.

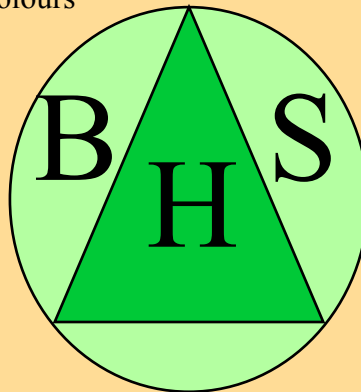
It is vital in these questions that you show your working where required.



Using Binary to Store Vector Graphics

A vector graphic is produced using shapes. The graphic may be simple with only a couple of simple shapes or be very complicated with thousands of intricate shapes.

Simple - Regular shapes with single colours



Complicated - Irregular shapes with shadows, textures, transparency, gradients, inner glows etc.

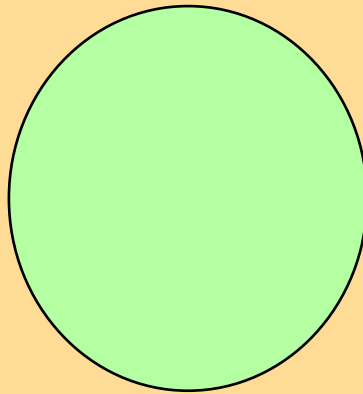


When a vector graphic is displayed, the computer system recreates the picture from the individual shapes. In order to do this it must have enough information stored on each shape to recreate it precisely.

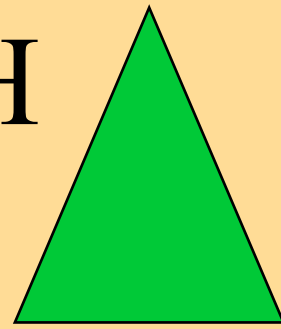
To store the simple vector example, the following graphic properties would have to be stored for each of the 5 shapes.



B

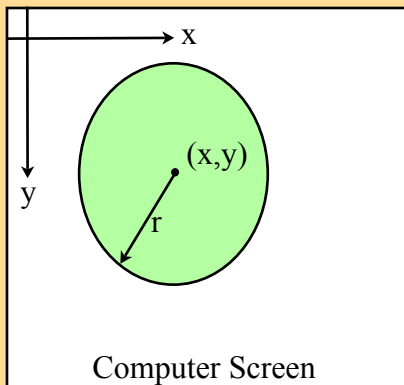


H



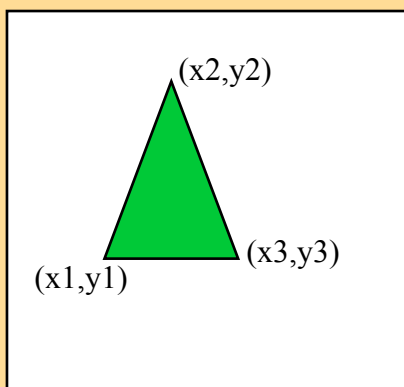
S

For the **circle** the following properties would be stored.



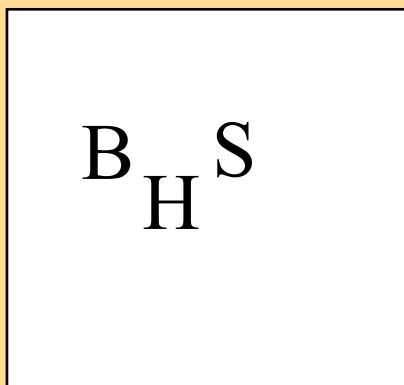
- The centre point of the circle (its position, x and y coordinate on screen)
- The size of the circle (its radius)
- The fill colour, light green (rgb - 180, 255, 161)
- The line thickness (1 point)
- The line colour, black (rgb - 0,0,0)
- The layer, back (layer 1)

The **triangle** would be stored slightly differently.



- The x,y coordinate of each point on the triangle
- The fill colour, dark green (rgb - 0, 201, 55)
- The line thickness (1 point)
- The line colour, black (rgb - 0,0,0)
- The layer, middle (layer 2)

The three **characters** are also shapes so we have to note their position, colour etc as well.



- The x,y coordinate of each letter
- The character itself (B, H or S)
- The fill colour, black (rgb - 0,0,0)
- The line thickness (0 point)
- The line colour, black (rgb - 0,0,0)
- The font
- The style (bold, underline, italics etc)
- The layer, front (layer 3)

Each of the above properties for each shape would be stored using binary. A common standard file type used to store vector graphic data is SVG (scalable vector graphic). The properties of each shape, once encoded in binary, are stored as a list in the vector graphic file.

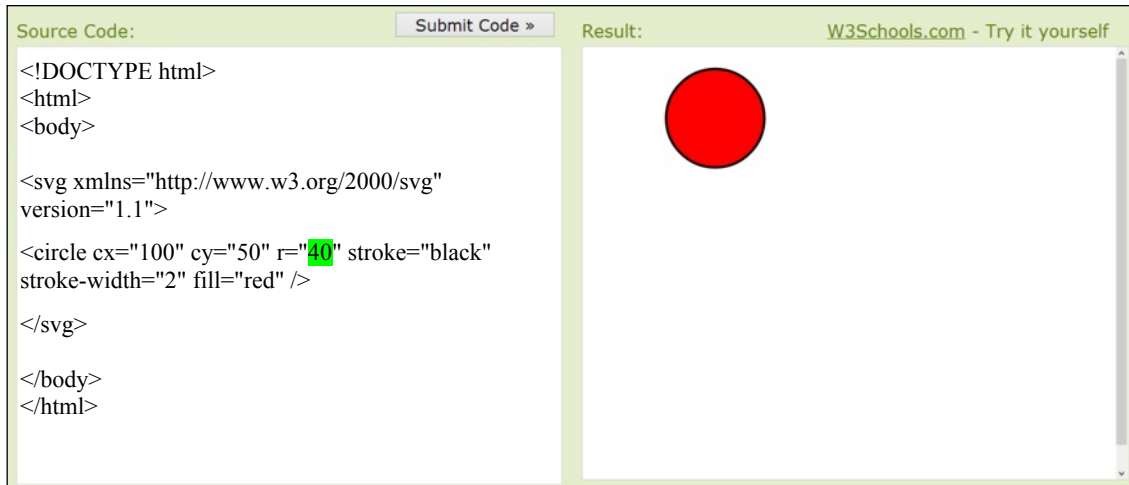
Task 6 - Editing SVG data

The website w3schools.com has some examples of SVG code embedded in HTML.

Open a browser and navigate to the URL below:

http://www.w3schools.com/svg/svg_examples.asp

Click on the “Circle” example, edit the radius of the circle as shown below and then click Submit Code.



```
<!DOCTYPE html>
<html>
<body>

<svg xmlns="http://www.w3.org/2000/svg"
version="1.1">

<circle cx="100" cy="50" r="40" stroke="black"
stroke-width="2" fill="red" />

</svg>

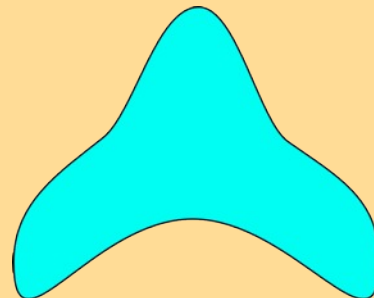
</body>
</html>
```

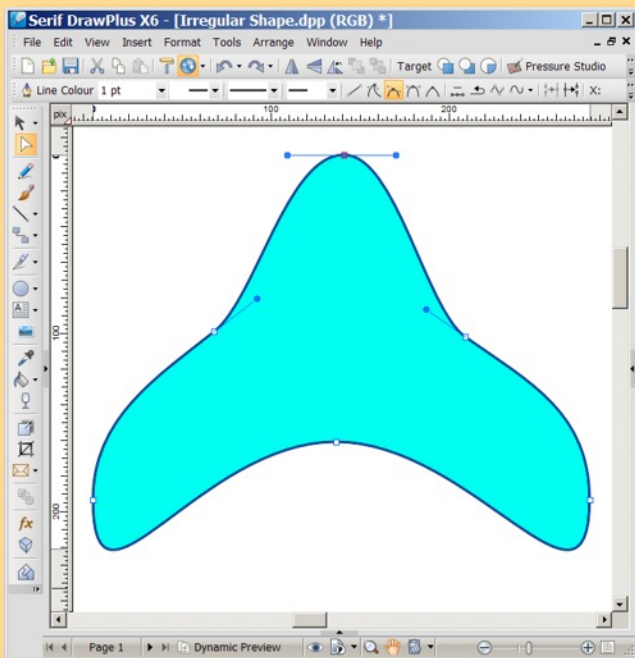
Over the next 15 minutes, try editing code in several different examples. Even without a detailed knowledge of SVG code you will find that you can work out what much of the code does.

To store a regular shape, like a circle, is easy as it will always have the same set of properties.

The fill colour, line colour etc of an irregular shape can be stored in exactly the same way as a regular shape using a simple list of properties.

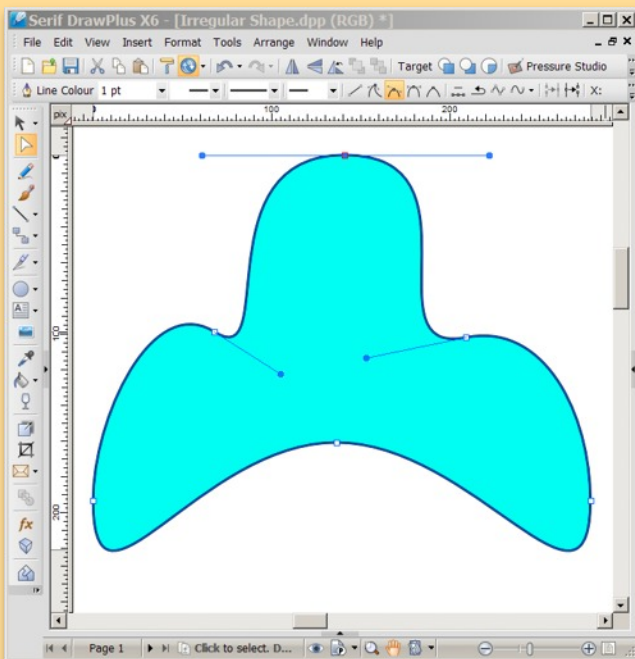
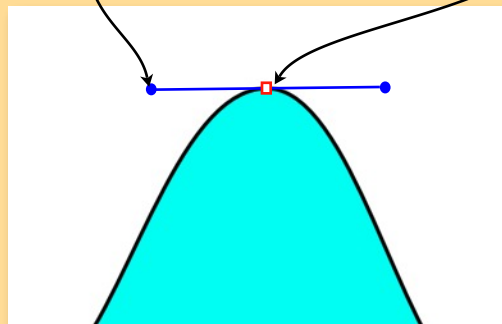
Storing the outline of the irregular shape requires a new approach. An irregular shape requires us to store the exact angle of each curve or corner in a way that will allow an application to redraw the shape once it's saved.





Each curve in a vector graphic object is determined by two attributes:

- set points on the curve called **nodes**
- control **handles** which can be used to adjust the angle at which the line enters and leaves the node.

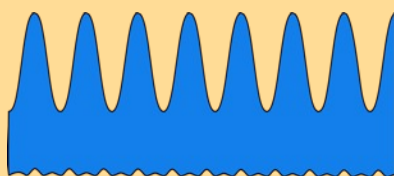
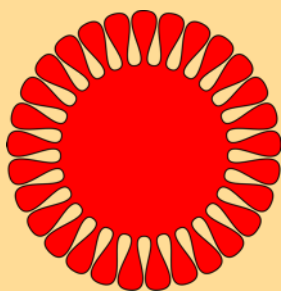


This example has the same 6 nodes as the example above and the nodes are in the same position as before.

By moving a few of the control handles (therefore changing the path of each curve) a completely different outline can be produced.

By storing the position of each node along with the curve angles, the outline of any shape no matter how complicated may be stored.

The examples below have hundreds or thousands of nodes and curves stored in each vector file.



Machine Code

We have learned that numbers, text and graphics are all stored as binary. To process all this data requires a computer program to provide instructions on how to calculate, move, store or display the binary values.

As everything processed in a computer system has to be in binary form it should come as no surprise now that program code is also stored as binary.

When programmers sit and write programs, they do not however write instructions in binary. Imagine how difficult it would be to understand, edit and find mistakes in long sequences of 1's and 0's.

```
010101010001010001001010100100100001011101010101010
101001010101010001010001001010100100100001011101010
00111010100101010100010100010010101001001000010111
11101010101001010101000101000100101010010010000101
01101010101010010101010001010001001010100100100001
01101010101010
```



It's much easier to write a program using an English based programming language and then translate it into binary so that the computer can then understand and process the code.

```
380 next p
390 print"gcrt="w/2/xm
400 nextz3: nextal
410 print"tau-avg="tt/kc
500 openl,4,7:cmdl
502 if z4=0 then printct$:print" "
505 if z4=0 then print"lens rim angle="rd"degrees" ";
506 if z4=0 then print"prism rim angle="pd"degrees"
510 if z4=0 then print"prism width="w"mils" ";
530 if z4=0 then print"refractive index="n2
550 if z4=0 then print"longitudinal incidence
angle="ld"degrees"
555 if z4=0 then print"average prism transmittance="
int(tt/kc*10000+.5)/10000
557 if z4=0 then print " ":print""
560 print"prism true thickness-theoretical thickness="
"dy"mils" ";
570 print"*** geometric concentration ratio="int
(w/2/xm*1000+.5)/1000;
572 print#1:close1
574 kc=0:zm=0:tt=0:z4=z4+1:nextdy
575 z4=0:gosub577:end
577 input"design thickness error in mils":dy
578 openl,4,7:cmdl
580 print " ":print"prism shape data"
600 rem prism shape
610 for p=pmtopm step pm/10
700 r=r0*(nl/nd-1)/(nl/nd*cos(p)-1)
710 x=r*sin(p)
720 y=r*cos(p):yp=y+dy
725 x=int(x*1000+.5)/1000:y=int(y*1000+.5)/1000:yp=int(y
p*1000+.5)/1000
730 print"x="x"mils y="y"mils yp="yp"mils"
740 nextp:printchr$(12)
750 print#1:close1
800 return
```

High Level Language (program written in English)

Low Level Language (translated binary version)

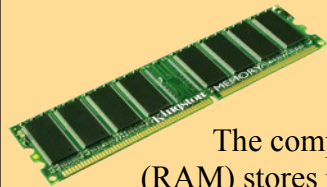
```
0101010101000101000100101001001000010111010101010101001010
1010100010100010010101001001000010110101000111010101010100
010100010010100100100001011111101010100101010100010100010
01010100100100001010110101010100101010100010100010010101001
001000010110101010100101010001010001001010100100100001011
010101010101001010101010101000101000100101010010000101101010
001110101001010100010100010010100100100001011111101010101
001010101000101000100101010010010000101011010101010010101
000101000100101010010010000101101010101001010100010100010
010101001001000010110101010101001010101010001010001001010
10010010000101101010001110101001010100010100010010101001001
000010111111010101001010100010100010010101001001000010101
101010101001010101000101000100101001001000010110101010101
0010101000010100010010101001001000010110101010101010010101
01010001010001001010010010000101101010001110101001010101000
101000100101001001000010111111010101010010101000101000100
10101001001000010101101010101001010101000101000100101010010
010000101101010101001010100010100010010101001001000010110
10101010101001010101010001010001001010100100100001011010100
01110101001010100010100010010101001001000010111111010101010
01010101000101000100101010010010000101011010101010010101010
00101000100101001001000010110101010101001010101010
```

Computer program code in binary form is called **machine code**.

Computer Architecture

To process data requires program instructions (machine code) and the data (binary) itself. The method used to process data stored in a computer systems has not changed since the early days of computing.

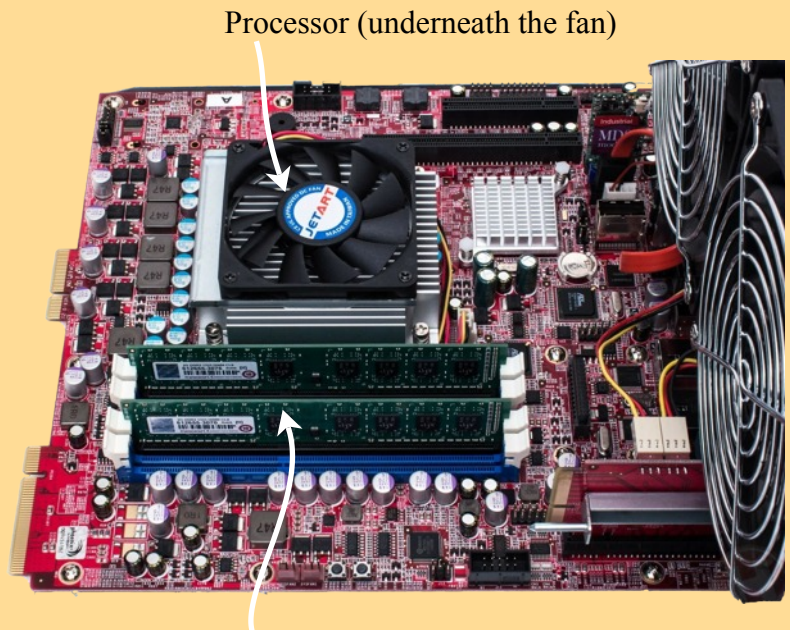
Processing instructions and data involves two of the main components in your computer system.



The computer's memory (RAM) stores program instructions and data while they are being used.



The processor decodes and carries out each program instruction. These instructions often involve fetching and processing the data stored in memory.

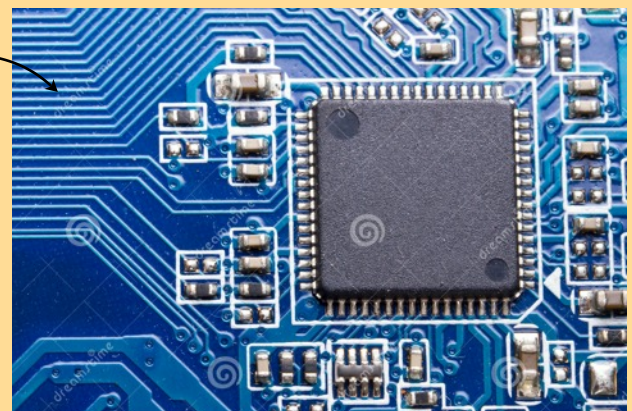


Memory modules (fitted into DIMM slots)

If program instructions and data are held in the computer's memory but are processed by the processor there must be a connection of some sort joining the two components.

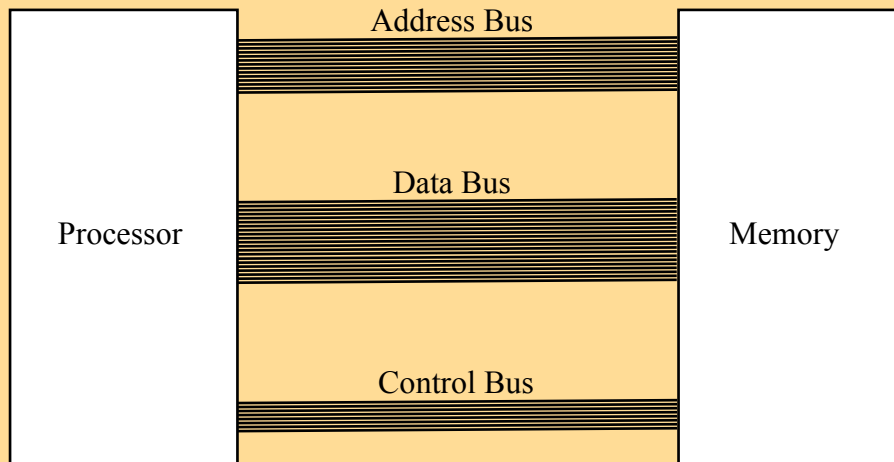
The surface of a motherboard is fitted with groups small, surface wires which provide physical connections between the different components on the motherboard.

The wires are manufactured on both sides of the motherboard but are easier to see on the bare underside of the circuit board.



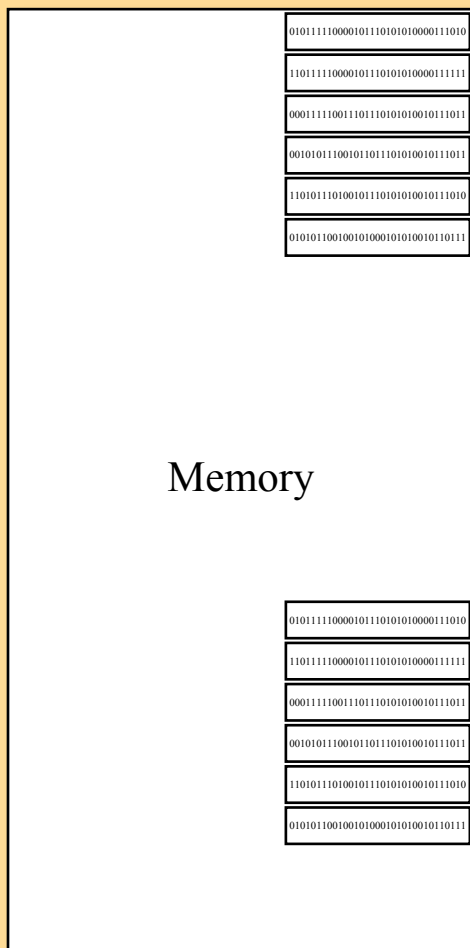
The groups of wires that connect the memory to the processor are called **buses**. There are three buses connecting the memory and processor (*address, data and control*) each with its own function.

To discuss this further we can represent the memory, processor and three buses in the form of a diagram.



Memory Addresses

When the processor fetches instructions and data from the memory it must know where to find them. The memory in a computer is organised into separate storage locations.



01011110000101110101010000111010

Each memory location will be capable of storing a set number of bits depending on the age and design of the computer. 64 bits is fairly common for a modern computer.

In a single RAM module there may be millions of memory locations each one containing a program instruction or item of data.

In order to find each memory location they are all numbered. Memory location numbers are known as **addresses**.

The processor uses the addresses to select the location which contains the program instruction or data to process next.

The concept of memory locations, each with its own unique address is called **addressability**.

Processor Components

A computer processor has three main components. The **ALU** (arithmetic logic unit), **Registers** and the **Control Unit**.

Registers

Registers are memory locations built into the processor in order to store temporary data during processing.

Temporary storage is required as it make take a processor several steps to complete a task.

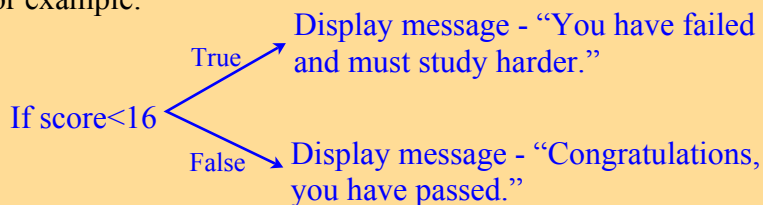
For example, if the processor was performing the following calculation $(5+4) - (6+1)$, it would...

- add the $5+4$ first and store the answer 9 in one of the registers
- add $6+1$ and store 7 in the registers
- fetch the two stored values from the registers
- subtract the $9-7$ to calculate the final result.

Arithmetic Logic Unit

The second part of the processor carries out any calculations (arithmetic) required and makes decisions (logic).

Decisions often involve doing some sort of comparison. For example:

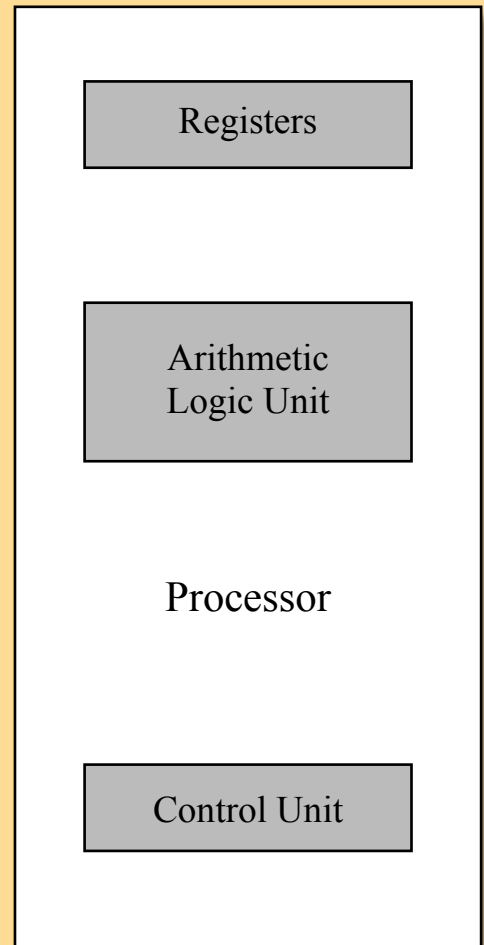


A computer program for the above example would have two sets of instructions (one for each message). The ALU would compare the “score” with the number 16 to decide which set of instructions should be processed next.

Control Unit

Modern computers are capable of performing billions of calculation every second. At these speeds it is import that all the events occur within a processor in the correct order.

The control unit is responsible for the timing of events within the processor. It does this by means of a clock pulse or by stopping and starting different processes.

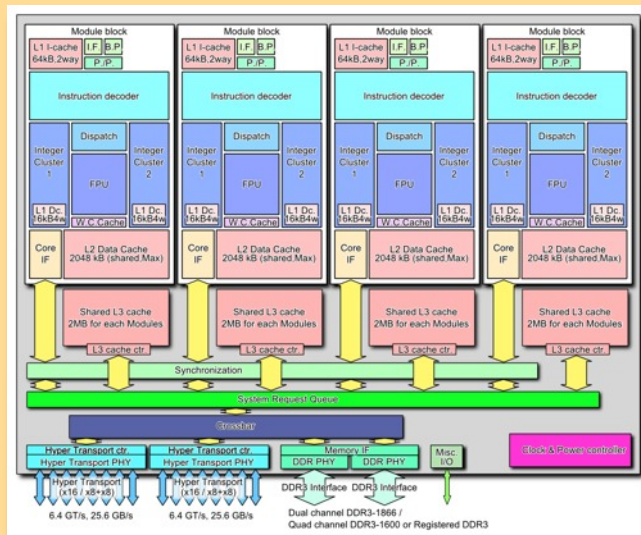


It's important to point out that our processor diagram is a very simplistic view of a modern processor which contains thousands of microscopic components.

This is a diagram of AMD's "Bulldozer" processor design released in 2011.

As you can see, the complexity of modern processor design is significantly different from our model.

Thankfully you do not need to learn this model for National 5!

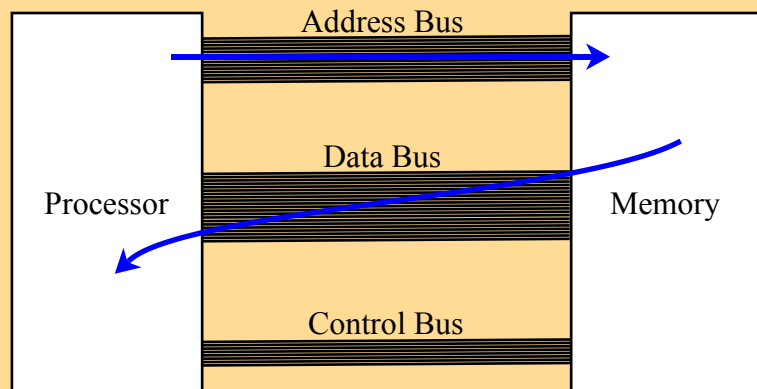


The Role of Buses in Processing

When instructions and data are transferred from the memory to the processor the following steps are carried out.

Memory Read

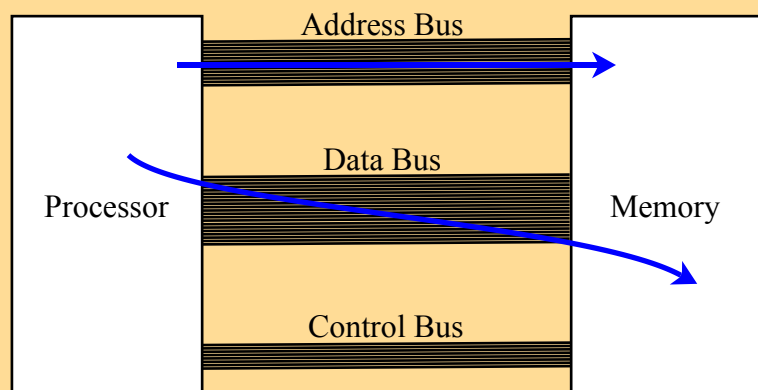
1. The address bus is used to select the address of the desired memory location.
2. The control bus sends a signal to activate the transfer.
3. The machine code instruction (or data) in the selected location is sent along the data bus to the processor.



When data is transferred from the processor back to the memory the following steps are carried out.

Memory Write

1. The address bus is used to select the desired memory location.
2. The control bus sends a signal to activate the transfer.
3. The data is sent along the data bus to the selected location.



Note - The address bus is one directional as it only ever sends address information from the processor to the memory. The data bus is bi-directional as data can travel to and from the processor.

Interfaces

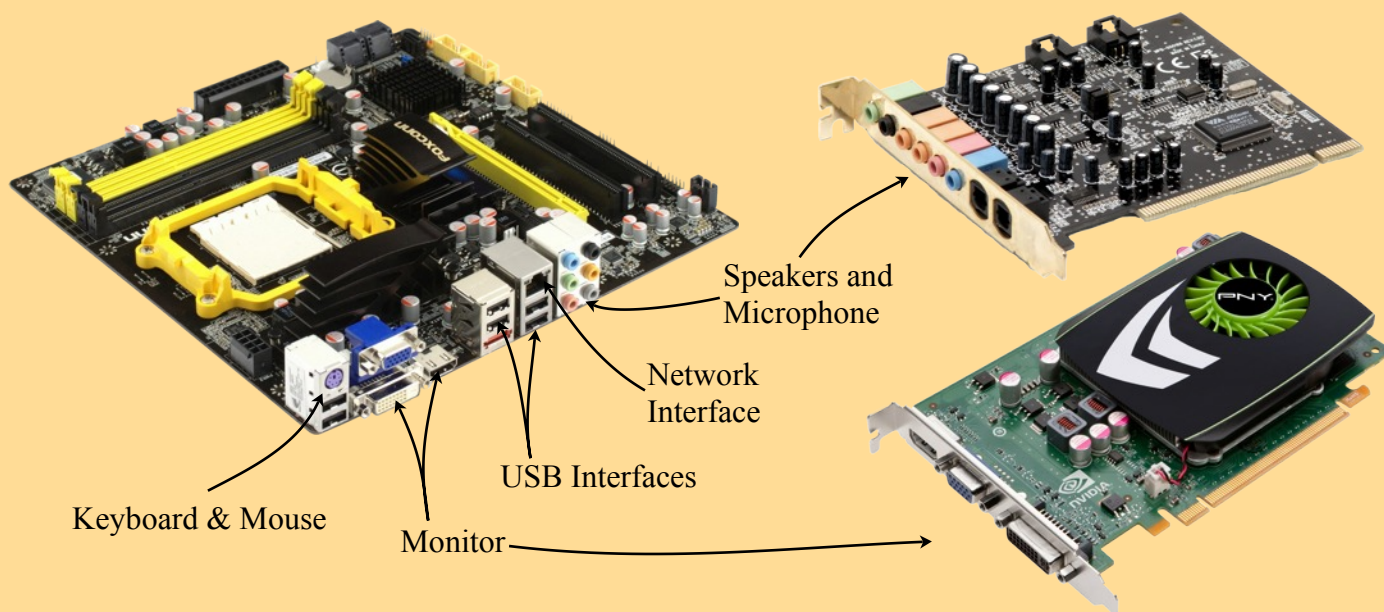
Processing often involves receiving data from peripheral devices or sending data to peripheral devices.



In a simple world, all these devices would work at the same speed, use the same type of connection, the same of electrical signals and format data the same way. Unfortunately with thousands of different manufactures producing computer peripherals this will never happen.

The role of an interface is to provide a bridge between the motherboard and peripheral devices, compensating for differences in the speed they work at and the methods they use to transfer data.

An interface may be a component on the motherboard itself or may be purchased as a separate specialised devices.



Separate interface devices (like the sound and graphics cards shown) will often have their own processing capability. This improves the processing power of your computer system as the main processor is freed up to concentrate on other tasks.

Task 7 - Computer Architecture Revision

Having a good understanding of facts will help you to answer problem solving questions in the National 5 exam.

Use the previous pages to research and type up answers for the following knowledge & understanding style questions.

1. Name and describe the function of the following internal processor components:
Arithmetic Logic Unit
Control Unit
Registers
2. State one difference between the data bus and the address bus..
3. Explain the need for each storage location in memory to be allocated a unique address.

Use the world wide web to research the answers for questions 4 & 5.

4. Together the memory and processor are called the CPU. State what the letters CPU stand for.
5. The iPhone 5S was the first mobile phone to have a 64 bit processor. What does the term “64 bit processor” tell us about the architecture of the phones CPU and buses?

Print and submit your answers to your teacher.

Variables - How Computer Programs Store Data

Computer programs are written to input, process and output data. To achieve this, the data being used or created by a program has to be stored in memory.

When declaring memory locations to store the data (called variables in programming) the program will allocate a different a numbers of locations depending on the type of data being stored.

Integers	32 or 64 bits may be allocated to storing a single integer. Depending on the size of each memory location this will usually equate to only 1 or 2 locations.
Real Numbers	Again 1 or 2 locations may be allocated to storing a real number. The memory locations may be split with part of the location being used to store the mantissa and part being used to store the exponent.
Characters	Using ASCII code only 8 bits are required to store a single character. One character will easily fit into a single 32 or 64 bit memory location in a modern computer.
Strings	A string is a list of characters (word or sentence) and will require multiple memory locations to store the data.
Arrays	An array is a structure that stores multiple values. The number of locations will depend on the type of data being stored and how many examples of that data. For example, an array of 1000 integers may require 1000 memory locations to be set aside.