Thomas Robinson

Unit 19, Assignment 1

The Devil is in the Data

Solihull College

September 2022

Task One P1

Numeric & Alphanumeric Data

Numeric Data

Complete the Sentence

Numeric data is stored in a computer system in the following forms shown below. These are used because…

Binary—Base 2—is used since the 1s and 0s represent the electrical signals used within the circuits that make up computers.

Octal and Hexadecimal—Base 8 and Base 16 respectively—are used since they can represent a larger number in a smaller number of characters for humans to read. A computer only ever interprets binary, so these systems are only used for convenience and are another way of displaying binary information.

Decimal ‘74’ in Different Number Systems

Denary: 74

100000 10^4	1000 10^3	100 10^2	10 10^1	1 10^0
0	**0**	**0**	**7**	**4**

Binary: 01001010

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

Octal: 112

512 8^3	64 8^2	8 8^1	1 8^0
0	**1**	**1**	**2**

Hexadecimal: 4A

4096 16^3	256 16^2	16 16^1	1 16^0
0	**0**	**4**	**A** (10)

Alphanumeric Encoding

Complete the Sentence

The characters on a computer keyboard are stored on a computer system in… ASCII Code/Unicode UTF-8 Encoding.

‘computer’ in ASCII decimal encoding

099	111	109	112	117	116	101	114
c	o	m	p	u	t	e	r

Task Two P2

Data Types

Sound

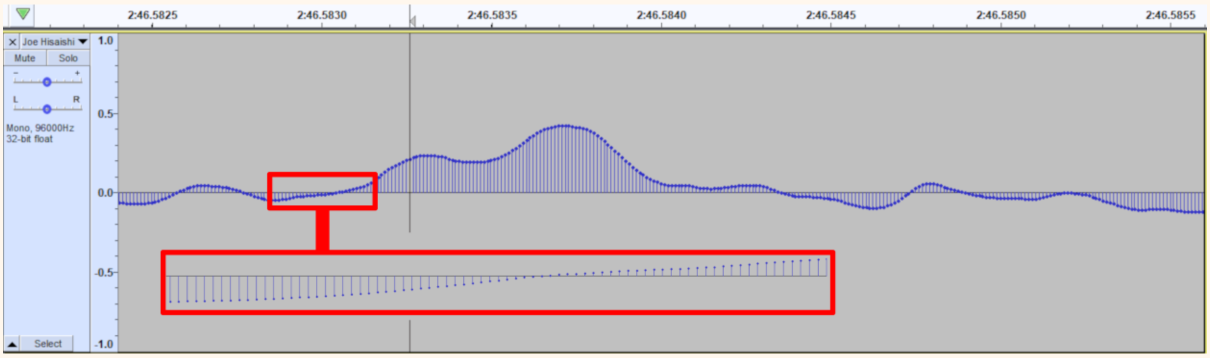


Image: A zoomed waveform being shown in the Audacity audio editing program. The red box contains an enlarged portion within which individual samples are more easily visible.

Amplitude & Time

In the real world, sound is produced by vibrations which cause waves to travel through the air. To represent this digitally, a microphone's transducer converts these sound waves into a voltage which is sampled a certain number of times each second. This is the 'amplitude' of the audio and the number of times a value is taken is referred to as the 'sample rate' of an audio recording. The sample rate is measured in Hz. In the screenshot above, the sample rate of the audio file is 96,000Hz which means each second there are 96,000 samples.

Bit-Depth

The amplitude values are quantised by an analogue-to-digital converter—this means to make them fit between two minimum/maximum values. This is shown in the screenshot on the Y axis where all values are between 1.0 and -1.0. The granularity available here is the 'bit depth' of the audio recording. The higher the bit-depth, the more faithful to the real frequency the recording will be. A 16-bit recording would be able to store 1 of 65536 (2^{16}) values each sample.

Storage

Audio filetypes and codecs dictate the amount and type of compression used. The most ubiquitous music file format—*mp3*—is a lossy format, which means some detail is lost to save space during compression. On the contrary, *flac* is a popular format which employs lossless compression which means there is no loss in quality.

Bitmap Graphics

Most images on a computer are broken up into pixels (picture elements). The most basic form of image storage is referred to as '*pix-map*' or '*bit-map*.' This refers to the fact that the file is made up of a long list of 1s and 0s representing every pixel.

Resolution

With any digital raster graphics format, the higher the number of pixels, the higher quality the image will be. This is referred to as the 'resolution' of an image and is written as the number of pixels available horizontally and vertically. For example, an average desktop computer monitor may have the resolution of 1920×1080. This is 1920 pixels along the X-axis and 1080 on the Y-axis for a total of 2,073,600 pixels.

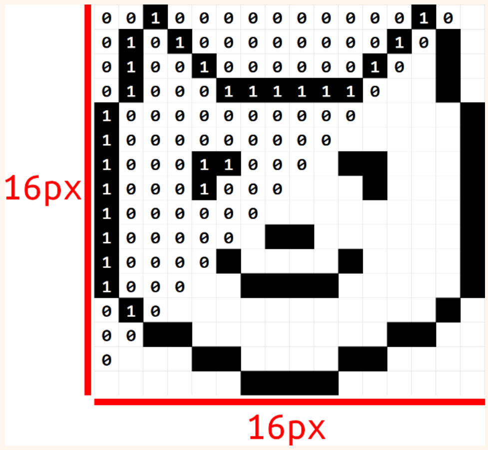
See below how the image of Puddles gets clearer with each increase in resolution. The higher the resolution, the more detail can be represented in a digital image.

25×25 50×50 125×125 500×500

In digital photography, image sizes are expressed as megapixel values—1 megapixel is 1,000,000 pixels. A megapixel value is calculated by multiplying the width and the height together, so an image that is 640×480 is made up of 307,200 pixels or 0.3 megapixels.

A high-end smartphone camera might be able to take 48-megapixel images whereas a lower quality webcam may only be able to capture 1-megapixel images.

Colour



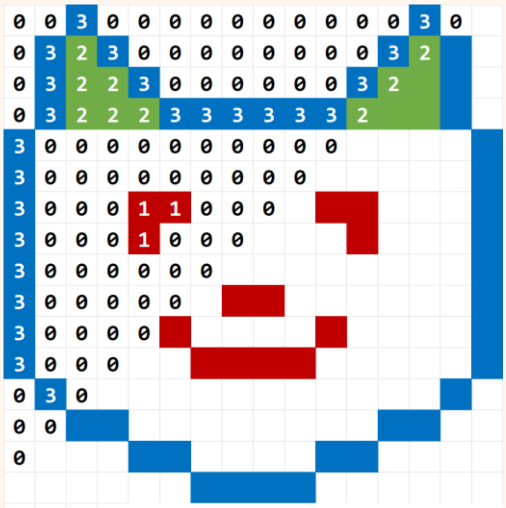
Each pixel contains information about the colour it needs to represent. In a 1-bit image, each pixel is represented by a single bit. This means that a pixel can either be 1 or 0 and results in a monochrome image.
In an 8-bit image, each pixel is represented by 8 bits of colour information, making it possible to store 256 (2^8) unique colours. The number of bits used to store this information is the 'bit depth' or 'colour depth' of the image. Modern displays can display 'true colour' which refers to a 24-bit colour depth—a total of 16,777,216 colours.

1-Bit
2 Colours

2-Bit
4 Colours

4-Bit
16 Colours

8-Bit
256 Colours



On the right, a 2-bit colour image is shown. A 2-bit image can show 4 different colours at once: from binary 00 to 11. In this case, 00 (0) is being used to show white, 01 (1) for red, 10 (2) for green, and 11 (3) for blue. This basic process of mapping colours to values is referred to as 'indexed colour.'

Metadata & Storage

Image data in a raster file is stored as long list of pixel data called a matrix. To display the image, the computer needs to know how many bits make up a pixel, what format the colour information takes and the size of the image. This auxiliary information is called 'metadata.'

As with audio files, bitmap graphic files can be uncompressed or employ either lossy or lossless compression algorithms. *jpg* files and *gif* files are examples of lossy formats, while *png* and *tiff* can both be lossless.

Task Three & Four P3, M1

Converting Numbers & Floating Point

Denary to Binary

123

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

252

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

9.125

9

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

.125

Calculation	Result	≥1?	Binary
0.125×2	0.25	no	**0**
0.25×2	0.5	no	**0**
0.5×2	1	yes	**1**
0.0×2	0	no	**0**
0.0×2	0	no	**0**

Normalising

$00001001.00100 \times 2^0$

1.001001×2^3

Exponent

3 + 127 = 130

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

Formatting

Sign	Exponent	Mantissa
0	**10000010**	**1001001**0000000000000000

Binary to Denary

1101010

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

2 + 8 + 32 + 64

= **106**

0111000

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

8 + 16 + 32

= **56**

011.011

011

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

1 + 2

= **3**

.011

0.5 2^{-1}	0.25 2^{-2}	0.125 2^{-3}	0.0625 2^{-4}
0	**1**	**1**	**0**

0.25 + 0.125

= **0.75**

3 + 0.375

= **3.375**

Binary to Hexadecimal

1101010

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

2 + 8 + 32 + 64
= **106**

4096 16^3	256 16^2	16 16^1	1 16^0
0	**0**	106 ÷ 16 = **6.625**	106 % 16 = **10**

6.625➔6, 10➔A
= 6A

0111000

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

8 + 16 + 32
= 56

4096 16^3	256 16^2	16 16^1	1 16^0
0	**0**	56 ÷ 16 = **3.5**	56 % 16 = **8**

3.5➔3, 8➔8
= **38**

1000111

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

1 + 2 + 4 + 64
= 71

4096 16^3	256 16^2	16 16^1	1 16^0
0	**0**	71 ÷ 16 = **4.4375**	71 % 16 = **7**

4.4375➔4, 7➔7
= **47**

Denary to Hexadecimal

123

4096 16^3	256 16^2	16 16^1	1 16^0
0	**0**	123 ÷ 16 = **7.6875**	123 % 16 = **11**

7.6875➔7, 11➔B

= **7B**

252

4096 16^3	256 16^2	16 16^1	1 16^0
0	**0**	252 ÷ 16 = **15.75**	252 % 16 = **12**

15.75➔F, 12➔C

= **FC**

541

4096 16^3	256 16^2	16 16^1	1 16^0
0	541 ÷ 256 = **2.1133**	541 % 256 = 29 29 ÷ 16 = **1.8125**	29 % 16 = **13**

2.21133➔2, 1.8125➔1, 13➔D

= **21D**

Binary to 32-Bit Floating Point

To store decimal numbers in a binary format, it is necessary to convert them to a standardised format. One of the options for doing this is floating point numbers, where numbers are stored as a fixed length with an exponent and a mantissa element.
This is covered by the IEEE 754 standard.

111110100.011111

Normalising

The first step to creating a floating-point number is to normalise it. This means moving the decimal point as far to the left as possible.
$111110100.011111 \times 2^0$
$1.11110100011111 \times 2^8$

Exponent

To calculate the exponent part of the number, we must add 127 and convert the result to binary.
8 + 127 = 135

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

Formatting

In a 32-bit number, the number must be 32 bits long.
The 'sign' is a single bit which indicates whether the number is positive or negative. 0 refers to a positive number and 1 to a negative number.
The exponent is 8 bits long.
The mantissa is 23 bits long. It is also known as the fraction portion.

Sign	Exponent	Mantissa
0	**100000111**	**11110100011111**000000000

1000011

Normalising

1000011×2^0

1.000011×2^6

Exponent

6 + 127 = 133

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

Formatting

Sign	Exponent	Mantissa
0	**10000101**	**000011**00000000000000000

110111000.100111

Normalising

$110111000.100111 \times 2^0$

$1.10111000100111 \times 2^8$

Exponent

8 + 127 = 135

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

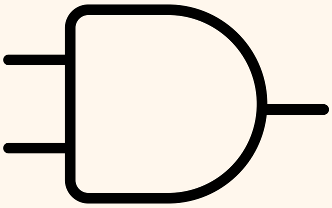
Formatting

Sign	Exponent	Mantissa
0	**10000111**	**11011100010011**100000000

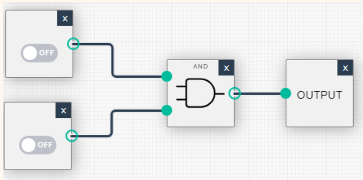
Task Five P4

Logic Gates & Truth Tables

AND

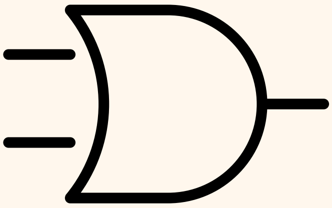


A	B	A∧B
0	0	0
0	1	0
1	0	0
1	1	1

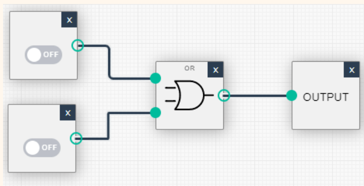


An AND gate takes two or more inputs and gives an output if they are all on.

OR



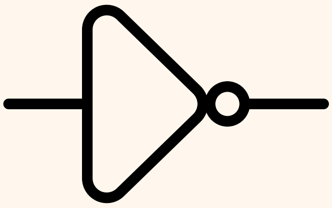
A	B	A∨B
0	0	0
0	1	1
1	0	1
1	1	1

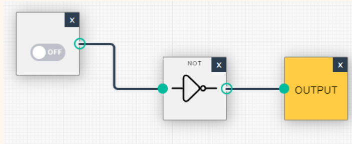


An OR gate takes two or more inputs and gives an output if any one of them is on.

NOT

A	¬A
0	1
1	0

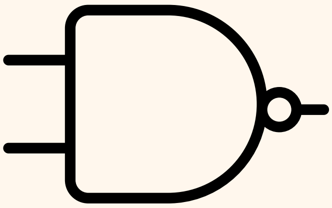


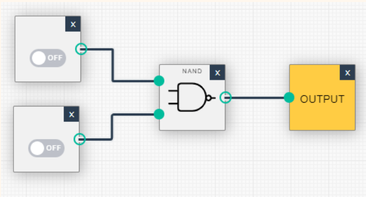


A NOT gate takes a single input and inverts it. If the input is on, then the output will be off and if the input is off, the output will be on.

NAND

A	B	¬[A∧B]
0	0	1
0	1	1
1	0	1
1	1	0

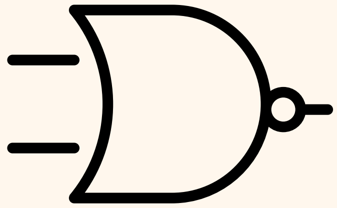


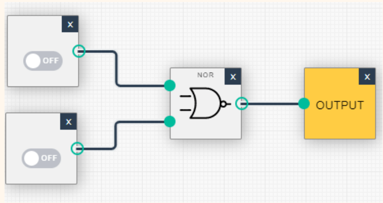


A NAND (NOT AND) gate is like an AND gate however the output is inverted. The output will be off when both inputs are on, but the output will be on if any input is off.

NOR

A	B	¬[A∨B]
0	0	1
0	1	0
1	0	0
1	1	0

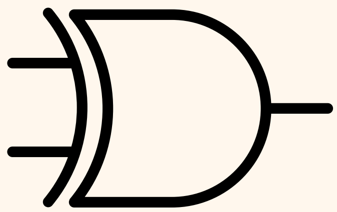


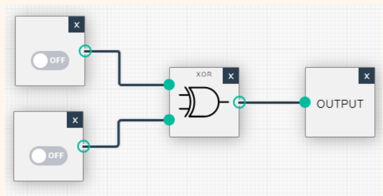


A NOT (NOT OR) gate will produce an output only if all its inputs are off. This is the inverse of an OR gate.

XOR

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0





An XOR (eXclusive OR) will only produce an output if a single input is on. If both or neither are on, then there will be no output.