Data Representation

Number bases

Denary (or decimal) is base-10 and is the number system we are most familiar with. We have the columns of units, tens, hundreds, thousands and so on. Base-10 means that we have 10 possible values (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) in each column.

Binary is base-2 and has 2 values, 0 and 1. It requires a greater number of digits in binary to represent a number than denary. This is how data and instructions are stored in a computer.

To calculate the maximum value for a given number of bits we use 2^n-1 where n is the number of bits. For example for 4 bits we have 2^4-1 which is 15.

Bits	Max value binary	Max value denary
1	1_2	1_{10}
2	11_2	3_{10}
3	111_2	7_{10}
4	1111_2	15_{10}
5	11111_2	31_{10}
6	111111_2	63_{10}
7	1111111_2	127_{10}
8	11111111_2	255_{10}

Hexadecimal is base-16. To make up the 16 values we use the ten denary numbers in addition to 6 letters (A, B, C, D, E, F).

Denary	Hex.	Binary
0_{10}	0_{16}	0000_2
1_{10}	1_{16}	0001_2
2_{10}	2_{16}	0010_2
3_{10}	3_{16}	0011_2
4_{10}	4_{16}	0100_2
5_{10}	5_{16}	0101_2
6_{10}	6_{16}	0110_2
7_{10}	7_{16}	0111_2

Denary	Hex.	Binary
8_{10}	8_{16}	1000_2
9_{10}	9_{16}	1001_2
10_{10}	A_{16}	1010_2
11_{10}	B_{16}	1011_2
12_{10}	C_{16}	1100_2
13_{10}	D_{16}	1101_2
14_{10}	E_{16}	1110_2
15_{10}	F_{16}	1111_2

Hexadecimal is used a lot in computing because it much easier to read than binary. There are far fewer characters than binary. So hexadecimal is often used in place of binary as a shorthand to save space. For instance, the hexadecimal number 7BA3D456 (8 digits) is 01111011101000111101010001010110 (32 digits) in binary which is hard to read.

Hexadecimal is better than denary at representing binary because hexadecimal is based on powers of 2.

Converting between number bases

Denary to binary conversion

1. Create a grid:

128	64	32	16	8	4	2	1

2. Add a 1 to the corresponding cell if number contributes to target number and 0 to all the other cells

Worked example: convert 24_{10} to binary.

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

$16_{10} + 8_{10} = 24_{10}$

The binary value is <u>11000_2</u> (we can ignore the preceding zeros)

Binary to denary conversion

Worked example: Convert 01011001_2 to denary

1. Create the grid:

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

2. Add up the cells that have a corresponding value of 1:
   64 + 16_{10} + 8_{10} + 1= <u>89_{10}</u>

Hexadecimal to denary conversion

1) Convert the two hex values separately to denary value
2) Multiply the first value by 16
3) Add the second value

Worked example: Covert $A3_{16}$ to denary

$A_{16} = 10_{10}$
$3_{16} = 3_{10}$
$(10_{10} \times 16_{10}) + 3_{10} =$ <u>163_{10}</u>

Denary to hexadecimal conversion

1) Integer divide the denary number by 16
2) Take the modulus 16 of the denary number
3) Convert the two numbers to the corresponding hex values.

Worked example: Convert 189_{10} to hex

$189_{10} / 16_{10} = 11_{10}$ remainder 15_{10}
$11_{10} = B_{16}$
$15_{10} = F_{16}$
$189_{10} =$ <u>BF_{16}</u>

Hexadecimal to binary conversion

1. Find the corresponding 4-bit binary number for the two numbers
2. Concatenate the two binary values to give the final binary value

Example: convert $C3_{16}$ to binary

$C_{16} = 12_{10} = 1100_2$
$3_{16} = 3_{10} = 0011_2$
<u>11000011_2</u>

Binary to hexadecimal conversion

1. Split the binary number into groups of 4 bits: 1110_2 1010_2
2. Find the corresponding Hex value for each of the 4-bit groups

Worked example: Convert 11101010_2 to hexadecimal

1110_2 |1010_2
$1110_2 = 14_{10} = E_{16}$
$1010_2 = 10_{10} = A_{16}$
<u>EA_{16}</u>

Units of Information

Unit	Symbol	Number of bytes
Kilobyte	KB	10^3 (1000)
Megabyte	MB	10^6 (1 million)
Gigabyte	GB	10^9 (1 billion)
Terabyte	TB	10^{12} (1 trillion)

A bit is the fundamental unit of binary numbers. A bit is a binary digit that can be either 0 or 1.

1 byte = 8 bits
1 nibble = 4 bits

Character Encoding

Character coding schemes allows text to be represented in the computer. One such coding scheme is **ASCII**. ASCII uses 7 bits to represent each character which means that a total of 128 characters can be represented.

Lower case letters	26
Upper case letters	26
Numbers	10
Symbols (e.g. comma, colon)	33
Control characters	33

ASCII encoded values for some characters

A	1000001_2	65_{10}
B	1000010_2	66_{10}
a	1100001_2	97_{10}
b	1100010_2	98_{10}
"0"	0110000_2	48_{10}
"1"	0110001_2	49_{10}

- ASCII has a limited character set (7 bits, 128 characters), but **Unicode** has 16 bits and allows many more (65K) characters.
- Unicode provides a unique character for different languages and different platforms.
- It allows us to represent different alphabets for instance Greek, Mandarin, Japanese, Emojis etc.
- Unicode and ASCII are the same up to 127.

Binary addition

Binary addition rules

$0_2 + 0_2 = 0_2$
$0_2 + 1_2 = 1_2$
$1_2 + 0_2 = 1_2$
$1_2 + 1_2 = 10_2$ (carry 1)
$1_2 + 1_2 + 1_2 = 11_2$ (carry 1)

Example

```
        1 0 1 0 1 0 0 1₂
        0 0 0 0 1 0 0 1₂
+       0 0 0 1 0 1 0 1₂
        1 1 0 0 0 1 1 1₂
carry     1 1 1       1
```

Binary Shift

The binary shift operator is used to perform multiplication and division of numbers by powers of 2

multiply/divide	x 16	x 8	x 4	x 2	/ 2	/ 4	/ 8
shift	<<4	<<3	<<2	<<1	>>1	>>2	>>3

Example: Apply shift operator to 1101_2 (13_{10})

Shift	Result	denary
<<1	11010_2	$13_{10} \times 2_{10} = 26_{10}$
<<2	110100_2	$13_{10} \times 4_{10} = 52_{10}$
>>1	110	$13_{10} // 2_{10} = 6_{10}$

Note that odd numbers are rounded down to the nearest integer when the right shift operator is applied.

Sound

Sample - Measure of the analogue signal at a given point in time

Sample rate - number of samples taken per second and is measured in Hertz.

Sample resolution - number of bits used to represent each sample

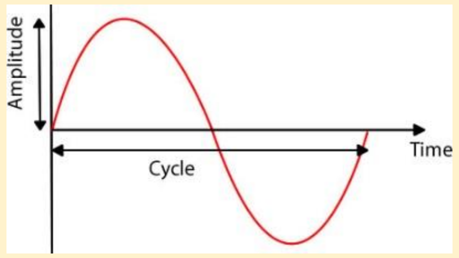
The size of sound files can be calculated using:

size of file = length (seconds) x sample rate x sampling resolution

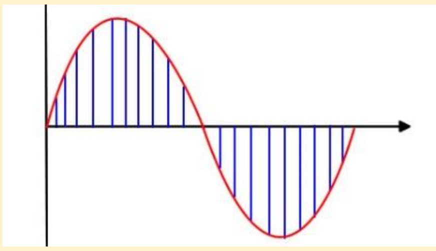
For sound to be stored digitally on a computer it needs to be converted from its continuous analogue form into a discrete binary values. The steps are:

1. Microphone detects the sound wave and converts it into an electrical (analogue) signal
2. The analogue signal is sampled at regular intervals
3. The samples are approximated to the nearest integer (quantised)
4. Each integer is encoded in binary with a fixed number of bits

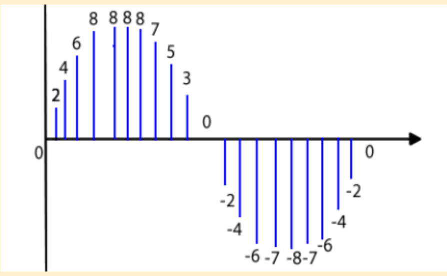
Original analogue signal



Sample signal at regular intervals



Integer values give to each sample



Encode as binary

0 2 4 6 8 8 8 8 7 5 3 0 ->

00000 00010 00100 01000

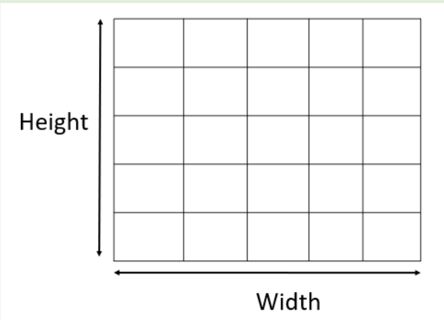
01000 01000 01000 00111

00101 00011 …

Images

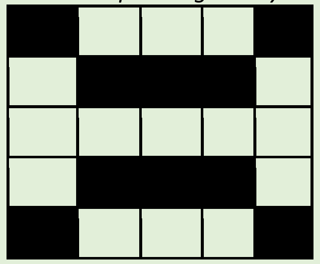
Bitmap images are made up from tiny dots called **pixels**. Each pixel will have a colour associated with it. An image can then be constructed from many of pixels which will have different colours arranged in rows and columns.

Total number of pixels in image = width in pixels x height in pixels



Colour depth is the number of bits used to represent each pixel in an image. If we have a black and white image it has two colours. Each pixel can be represented by a single pixel because a bit value of 0 is black and 1 is white.

Image and corresponding binary encoding



0111010001111111000101110

To represent more colours we can use more bits. For instance if we have 2-bits per pixel we can represent 4 colours because we know have 4 binary code combinations (00, 01, 10 11) where each code represents a different colour

Pixilation occurs when the image is overstretched. In these situations, the image looses quality and has a blocky and blurred appearance. This arises when the image is presented at too large a size and there are not enough pixels to reproduce the details in the image at this larger size.

Calculating the size of a bitmap image

File size in bits = width in pixels x height in pixels x colour depth

File size in bytes = width in pixels x height in pixels x colour depth / 8

Data Compression

The purpose of data compression is to make the files smaller which means that:

- Less time / less bandwidth to transfer data
- Take up less space on the disk

Given that there are 7 bits per ASCII character, the uncompressed size of an ASCII phrase is:

size = number of characters (including spaces) x 7

Run Length Encoding (RLE) is a compression method where sequences of the same values are stored in pairs of the value and the number of those values. For instance, the sequence:

0 0 0 1 1 0 1 1 1 1 0 1 1 1 1

would be represented as:

3 0 2 1 1 0 4 1 1 0 4 1

Huffman coding is a form of compression that allows us to use fewer bits for higher frequency data. More common letters are represented using fewer bits than less common letters. For instance, “a” and “e”, which occur in many words would be represented with fewer bit than “z” which occurs rarely.
This allows for much more effective compression than RLE.

The steps involved in Huffman encoding as are follows:

1. Do frequency table
2. Order table
3. Create the tree
4. Add 1, 0 to the branches
5. Encode letters
6. Encode message

Worked Example: How much smaller is the phrase henry horse encoded using Huffman encoding compared with its uncompressed size.

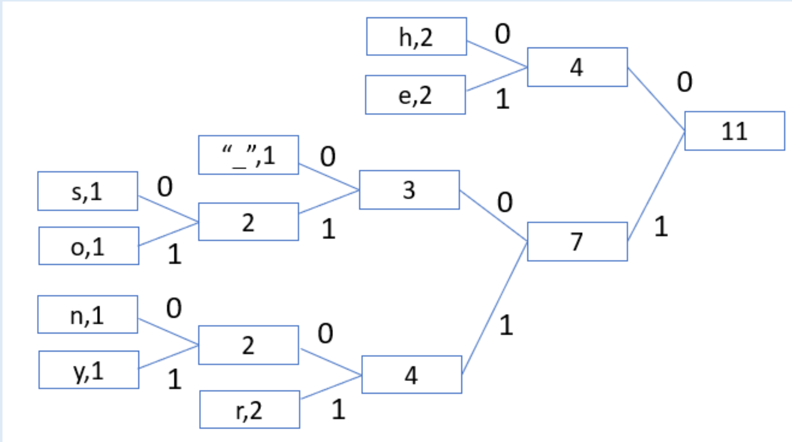
Calculate the uncompressed size

In the phrase *henry horse* there are 11 characters (including the space). Therefore the uncompressed size is 11 x 7 = <u>77 bits</u>

Generate ordered frequency table (steps 1 and 2)

letter	frequency
e	2
h	2
r	2
<space>	1
o	1
s	1
y	1
n	1

Create the tree and add 1 and 0 to branches (steps 3 and 4)



Encode letters

Letter	encoding
e	01
h	00
r	111
<space>	100
o	1011
s	1000
n	1100
y	1101

Encode message

00 01 1100 111 1101 100 00 1011 111 1000 01 = <u>33 bits</u>

Therefore by using compression we have reduced the size from 77 bits to 33 bits a saving of <u>44 bits</u>.