Name: Index: Reg. No.: Class: Date:

Chapter 1

Data Representation

Computers & Binary

Why does data have to be converted to binary to be processed by a computer?

- Data is processed in a computer using **logic gates** that only have **two states**
- The binary number system only has two digits (**1/0**), which means each digit can **represent a different state** (1 = on, 0 = off)
- All data must be converted to binary **before** a computer can **understand** and **process** it

Worked Example

Explain why computers process data in binary format **[2]**

Answers

- Computers process data using logic gates... **[1]**
- ... that can only have two states (1/0) **[1]**

Number Systems

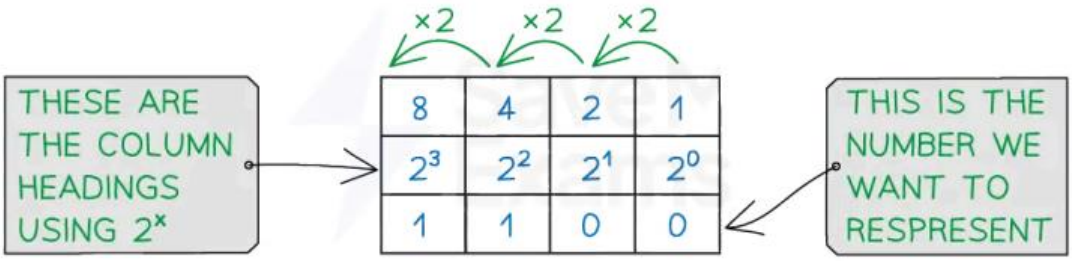
The Denary, Binary & Hexadecimal Number Systems

What is denary?

- Denary is a number system that is made up of **10 digits (0-9)**
- Denary is referred to as a **base-10** number system
- Each digit has a weight factor of **10 raised to a power**, the rightmost digit is 1s (10^0), the next digit to the left 10s (10^1) and so on
- Humans use the denary system for **counting, measuring** and **performing maths calculations**

What is binary?

- Binary is a number system that is made up of **two digits (1 and 0)**
- Binary is referred to as a **base-2** number system
- Each digit has a weight factor of **2 raised to a power**, the rightmost digit is 1s (2^0), the next digit to the left 2s (2^1) and so on
- Each time a new digit is added, the **column value is multiplied by 2**
- Using **combinations of the 2 digits** we can represent any number



- In this example, **Binary 1100 = (1 x 8) + (1 x 4) = 12**
- To represent bigger numbers we add more **b**inary dig**its** (bits)

32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Benefits:

- Computer can understand easily
- Faster processing because binary data matches the two states (1 and 0) used in logic circuits

Drawback:

- Difficult to understand by human
- Difficult to debug

Examiner Tips and Tricks

In the exam you are expected to be able to work with up to and including **16 bits**

The **largest denary number** that can be represented using **16 bits** is:

- **65,535** (Binary 1111111111111111)

What is hexadecimal?

- Hexadecimal is a number system that is made up of **16 digits**, 10 numbers (**0-9**) and 6 letters (**A-F**)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

- Hexadecimal is referred to as a **base-16** number system
- Each digit has a weight factor of **16 raised to a power**, the rightmost digit is 1s (16^0), the next digit to the left 16s (16^1)
- One hexadecimal digit can represent **four bits** of binary data

Converting Between Binary & Denary

Denary to Binary Conversion

Consider the conversion of the denary number, 142, into binary:

Method 1

The denary number 142 is made up of 128 + 8 + 4 + 2 (that is, 142 – 128 = 14; 14 – 8 = 6; 6 – 4 = 2; 2 – 2 = 0; in each stage, subtract the largest possible power of 2 and keep doing this until the value 0 is reached. This will give us the following 8-bit binary number:

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

Method 2

This method involves successive division by 2. Start with the denary number, 142, and divide it by 2. Write the result of the division including the remainder (even if it is 0) under the 142 (that is, 142 ÷ 2 = 71 remainder 0); then divide again by 2 (that is, 71 ÷ 2 = 35 remainder 1) and keep dividing until the result is zero. Finally write down all the remainders in reverse order:

2	142		
2	71	remainder:	0
2	35	remainder:	1
2	17	remainder:	1
2	8	remainder:	1
2	4	remainder:	0
2	2	remainder:	0
2	1	remainder:	0
	0	remainder:	1

read the remainders from bottom to top to get the binary number:

1 0 0 0 1 1 1 0

Binary to Denary Conversion

- To convert from binary to denary, count how many bits make up the value
- Write out the column headings for the number of bits given from right to left
- Multiply the binary values of 1s with headings
- Add together all the results

Example 1

Convert the binary number, 11101110, into a denary number.

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

(128x1) + (64x1) + (32x1) + (8x1) + (4x1) + (2x1)

The equivalent denary number is 128 + 64 + 32 + 8 + 4 + 2 = 238

Example 2

Convert the following binary number, 011110001011, into a denary number.

2048	1024	512	256	128	64	32	16	8	4	2	1
0	1	1	1	1	0	0	0	1	0	1	1

The equivalent denary number is 1024 + 512 + 256 + 128 + 8 + 2 + 1 = 1931

Examiner Tips and Tricks

If you are converting from binary to denary and the binary number ends in 1, the denary answer must be an odd number!

Converting Between Hexadecimal & Denary

Denary to Hexadecimal Conversion

Method 1 (denary to binary to hexadecimal)

- To convert the denary number **28** to hexadecimal, start by converting the denary number to binary

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

- **Split** the 8 bit binary number into **two nibbles** as shown below

8	4	2	1		8	4	2	1
0	0	0	1		1	1	0	0

- Convert each nibble to its denary value
- **0001 = 1** and **1100 = 12**
- Using the **comparison table**, the denary value 1 is also 1 in hexadecimal whereas denary value 12 is represented in hexadecimal as C
- Denary **28** is **1C** in hexadecimal

Method 2 (divide by 16)

Convert the denary number, 2004, into hexadecimal.

This method involves successive division by 16 until the value 0 is reached. We start by dividing the number 2004 by 16. The result of the division including the remainder (even if it is 0) is written under 2004 and then further divisions by 16 are carried out (that is, 2004 ÷ 16 = 125 remainder 4; 125 ÷ 16 = 7 remainder 13; 7 ÷ 16 = 0 remainder 7). The hexadecimal number is obtained from the remainders written in reverse order:

16	2004		
16	125	remainder:	4
16	7	remainder:	13
	0	remainder:	7

write the remainders from bottom to top to get the hexadecimal number:

7 D 4 (D=13)

Hexadecimal to Denary Conversion

Method 1 (hexadecimal to binary to denary)

- To convert the hexadecimal number **B9** to denary, take each hexadecimal digit and convert it from its **denary value** to **4 bit binary** (nibble)

B (11)					9			
8	4	2	1		8	4	2	1
1	0	1	1		1	0	0	1

- **Join** the two nibbles to make an **8 bit number** (byte)
- Convert from **binary to denary**

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

- (1 x 128) + (1 x 32) + (1 x 16) + (1 x 8) + (1 x 1) = 185
- Hexadecimal **B9** is **185** in denary

Method 2 (multiply by heading values)

- Take each of the hexadecimal digits and multiply it by the heading values
- Add all the resultant totals together to give the denary number

Convert the hexadecimal number, 4 5 A, into denary.

First of all we have to multiply each hex digit by its heading value:

256	16	1	
4	5	A	
(4 × 256 = 1024)	(5 × 16 = 80)	(10 × 1 = 10)	(NOTE: A = 10)

Then we have to add the three totals together (1024 + 80 + 10) to give the denary number:

1 1 1 4

Examiner Tips and Tricks

Remember that the exam is non-calculator, if you are not confident multiplying and dividing by 16 then use method 1 on both conversions

Converting Between Hexadecimal & Binary

Binary to Hexadecimal Conversion

- It is important before revising how to convert from binary to hexadecimal and vice versa that you fully understand the **binary** and **hexadecimal** number systems.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Example

- To convert the binary number 10110111 to **hexadecimal**, first split the 8 bit number into 2 binary **nibbles**

8	4	2	1		8	4	2	1
1	0	1	1		0	1	1	1

- For each nibble, convert the binary to it's denary value
- (1 x 8) + (1 x 2) + (1 x 1) = 11 (B)
- (1 x 4) + (1 x 2) + (1 x 1) = 7
- Join them together to make a 2 digit hexadecimal number
- Binary 10110111 is **B7** in hexadecimal

Hexadecimal to Binary Conversion

Example

- To convert the hexadecimal number **5F** to binary, first **split the digits apart** and convert each to a binary **nibble**

8	4	2	1	
0	1	0	1	= 5

8	4	2	1	
1	1	1	1	= 15 (F)

- Join the 2 binary nibbles together to create an 8 bit binary number

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

- Hexadecimal **5F** is **01011111** in binary

Uses of Hexadecimal

Why is hexadecimal used?

- It takes **fewer digits to represent a given value** in hexadecimal than in binary
  - 1 hexadecimal digit corresponds 4 bits (one nibble) and can represent 16 unique values (0-F)
- Easier to read and write as hexadecimal numbers are shorter than binary values
- Easier for humans to understand because it is shorter than binary values
- Fewer errors when reading or writing

- Examples of where hexadecimal can be seen:
  - MAC addresses
  - HTML colour codes
  - IPv6 addresses
  - Error codes

Binary Addition

What is binary addition?

- Binary addition is the process of **adding together two binary integers** (up to and including 8 bits)
- To be successful there are **5 golden rules** to apply:

Binary Addition	Binary Answer	Working
0 + 0 =	0	1s | 0 | = 0
0 + 1 =	1	1s | 1 | = 1
1 + 0 =	1	1s | 1 | = 1

1+1=	10	2s: 1, 1s: 0, = 2
1+1+1=	11	2s: 1, 1s: 1, = 3

- Like denary addition, start from the rightmost digit and move left

Example 1

Add 00100111 + 01001010

We will set this out showing carry and sum values:

```
  0 0 1 0 0 1 1 1
+ 0 1 0 0 1 0 1 0
    1 1 1          <- carry values
  0 1 1 1 0 0 0 1  <- sum values
```

Answer: **01110001**

column 1: 1 + 0 = 1 no carry
column 2: 1 + 1 = 0 carry 1
column 3: 1 + 0 + 1 = 0 carry 1
column 4: 0 + 1 + 1 = 0 carry 1
column 5: 0 + 0 + 1 = 1 no carry
column 6: 1 + 0 = 1 no carry
column 7: 0 + 1 = 1 no carry
column 8: 0 + 0 = 0 no carry

Example 2

a Convert 126 and 62 into binary.
b Add the two binary values in part a and check the result matches the addition of the two denary numbers

a 126 = 0 1 1 1 1 1 1 0 and 62 = 0 0 1 1 1 1 1 0

b

```
  0 1 1 1 1 1 1 0
+ 0 0 1 1 1 1 1 0
  1 1 1 1 1 1      <- carry values
  1 0 1 1 1 1 0 0  <- sum values
```

Answer: **10111100**

column 1: 0 + 0 = 0 no carry
column 2: 1 + 1 = 0 carry 1
column 3: 1 + 1 + 1 = 1 carry 1
column 4: 1 + 1 + 1 = 1 carry 1
column 5: 1 + 1 + 1 = 1 carry 1
column 6: 1 + 1 + 1 = 1 carry 1
column 7: 1 + 0 + 1 = 0 carry 1
column 8: 0 + 0 + 1 = 1 no carry

1 0 1 1 1 1 0 0 has the equivalent denary value of 128 + 32 + 16 + 8 + 4 = 188 which is the same as 126 + 62.

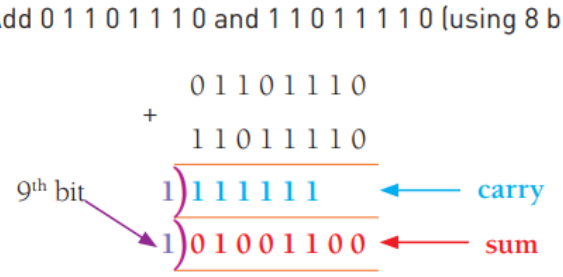
Examiner Tips and Tricks

Make sure any carried digits are clearly visible in your answer, there are marks available for working. Carries can be put above or below in the addition

Overflow & Binary Addition

What is an overflow error?

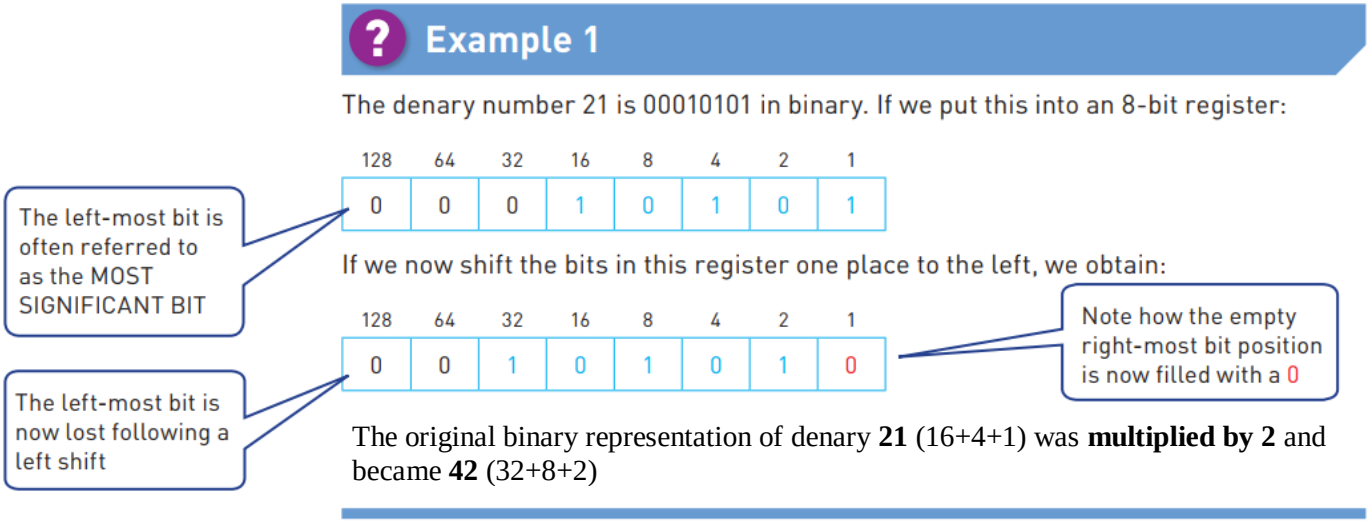
- An overflow error occurs when the result of a binary addition **exceeds the available bits**
- For example, if you took binary **11111111** (255) and tried to add **00000001** (1) this would cause an overflow error as the result would need a **9th** bit to represent the answer (256)



Binary Shifts

What is a logical binary shift?

- A logical binary shift is how a computer system performs **basic multiplication and division** on nonnegative values (0 and positive numbers)
- Binary digits are **moved left** or **right** a set number of times
- A left shift **multiplies** a binary number by 2^x (x is number of shifts)
- A right shift **divides** a binary number by 2^x (x is number of shifts)
- A shift can move more than one place at a time, the principle remains the same
- A left shift of 2 places would **multiply** the original binary representation of denary number by 4



Suppose we now shift the original number two places left:

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

The binary number 1010100 is **84** in denary – this is **multiplied by 4** (64+16+4)

And now suppose we shift the original number three places left:

1	0	1	0	1	0	0	0

The binary number 10101000 is **168** in denary – this is **multiplied by 2** (124+32+8)

So, let us consider what happens if we shift the original binary number 00010101 four places left:

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

Losing 1 bit following a shift operation will cause an error

The left-most **1-bit** has been **lost**. The result of binary number **00010101** after shifting **four places** of **left** is **336** in denary – this is **multiplied by 16** (256+64+16).

However, the denary value we got is **80** which is clearly **incorrect**.
This error is because we have **lost 1-bit** and **exceeded** the maximum number of left shifts possible using this register.

Example 3

a Write 24 as an 8-bit register.
b Show the result of a logical shift 2 places to the left.
c Show the result of a logical shift 3 places to the right.

a

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

b

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

64 + 32 = 96
OR
24 x 2^2 = 96

c

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

2 + 1 = 3
OR
24 ÷ 2^3 = 3

Two's Complement

What is two's complement?

- Two's complement is a method of using **signed binary values** to **represent negative numbers**
- Using two's complement the **left most bit** is designated the most significant bit (**MSB**)
- To represent negative numbers this bit **must equal 1**, turning the column value in to a **negative**
- Working with **8 bits**, the **128** column becomes **-128**

Writing positive binary numbers in two's complement format

Example 1

The following two examples show how we can write the following positive binary numbers in the two's complement format 19 and 4:

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

Converting positive binary numbers in the two's complement format to positive denary numbers

Example 3

Convert 01101110 in two's complement binary into denary:

−128	64	32	16	8	4	2	1
0	1	1	0	1	1	1	0

Writing negative binary numbers in two's complement format and converting to denary

Example 1

The following three examples show how we can write negative binary numbers in the two's complement format:

−128	64	32	16	8	4	2	1
1	0	0	1	0	0	1	1

By following our normal rules, each time a 1 appears in a column, the column value is added to the total. So, we can see that in denary this is: −128 + 16 + 2 + 1 = −109.

Converting negative denary numbers into binary numbers in two's complement format

Consider the number +67 in 8-bit (two's complement) binary format:

−128	64	32	16	8	4	2	1
0	1	0	0	0	0	1	1

Method 1

Now let's consider the number -67. One method of finding the binary equivalent to -67 is to simply put 1s in their correct places:

−128	64	32	16	8	4	2	1
1	0	1	1	1	1	0	1

−128 + 32 + 16 + 8 + 4 + 1 = −67

Method 2

However, looking at the two binary numbers above, there is another possible way to find the binary representation of a negative denary number:

first write the number as a positive binary value – in this case 67:	0 1 0 0 0 0 1 1
we then invert each binary value, which means swap the 1s and 0s around:	1 0 1 1 1 1 0 0
then add 1 to that number:	1
this gives us the binary for −67:	1 0 1 1 1 1 0 1

Text, Sound & Images

What is a character set?

- A character set is **list of characters and symbols** that can be **represented** by a computer system
- Each character is given a **unique** binary code
- The number of characters that can be represented is determined by the **number of bits used** by the character set
- Two common character sets are:
  - American Standard Code for Information Interchange (**ASCII**)
  - Universal Character Encoding (**UNICODE**)

What is ASCII?

- ASCII (American Standard Code for Information Interchange) is a character set and was an accepted standard for information interchange
- ASCII uses **7 bits**, providing maximum of **128 characters**
- ASCII only represents basic characters needed for **English**, limiting its use for other languages

Extended ASCII

- Extended ASCII uses **8 bits**, providing maximum of **256 characters**
- Extended ASCII provides essential characters such as mathematical operators and more recent symbols such as ©

Limitations of ASCII & extended ASCII

- ASCII has a **limited** number of characters which means it can only represent the **English alphabet**, **numbers** and some special characters
  - A, B, C, ……., Z
  - a, b, c ,.....,z
  - 0, 1, 2,....., 9
  - !, @, #, …..
- ASCII **cannot** represent characters from languages other than English
- ASCII **does not** include modern symbols or emojis common in today's digital communication

What is UNICODE?

- UNICODE is a **character set** and was created as a solution to the limitations of ASCII
- UNICODE uses a minimum of **16 bits**, providing a minimum of **65,536 characters**
- UNICODE can represent characters from **all the major languages** around the world

ASCII vs UNICODE

- ASCII uses 7 bits (or 8 bits in extended ASCII) but Unicode uses 16 bits or more.
- ASCII represents 128 (or 256 characters in extended ASCII) but Unicode represents 65,536 characters.
- ASCII used to represent characters in English language but Unicode used to represent characters from all world languages.
- ASCII uses less storage space than Unicode
- Unlike ASCII, Unicode includes emojis and symbols

How texts are represented in computer

- Character sets like ASCII and Unicode are used to store text.
- Each character is assigned a unique binary value.
- Each character's binary value is stored in sequence.
- The computer converts binary values back into readable text when displaying it.

Worked Example

The computer stores text using the ASCII character set

Part of the ASCII character set is shown:

Character	ASCII Denary Code
E	69
F	70
G	71
H	72

(a) Identify the character that will be represented by the ASCII denary code 76 **[1]**

(b) Identify a second character set **[1]**

Answers

(a) L (must be a capital)

(b) UNICODE

Representing Sound

How is sound sampled & stored in computer?

- The analogue sound is **captured using a microphone**.
- The microphone converts the sound into an **analogue electrical signal**.
- The computer **samples the sound wave by measuring the amplitude (height)** of the wave at regular intervals.
- Each sample will be represented by a **binary value**
- The **sample rate** is chosen.
- The **sample resolution** is chosen.
- Each sample (amplitude value) is **converted into binary** and **stored in computer.**

What is sample rate?

- Sample rate is the amount of **samples taken per second** of the analogue wave

What is sample resolution?

- Sample resolution is the **number of bits stored per sample** of sound

What effect do sample rate and sample resolution have?

- The accuracy of the recording and the file size increases as the **sample rate** and **resolution** increases.

Representing Images

What is a bitmap image?

- A bitmap image is made up of squares called **pixels**
- A pixel is the **smallest element** of a bitmap image
- Each pixel is **stored as a binary code**
- Binary codes are **unique** to the **colour** in each pixel
- A typical example of a bitmap image is a photograph

What is resolution?

- Resolution is the **total amount of pixels** that make up a bitmap image
- The resolution is calculated by **multiplying the height and width** of the image (in pixels)
- In general, the **higher the resolution the more detail in the image** (higher quality)

What is colour depth?

- Colour depth is the **number of bits used to represent each color**
- The colour depth is dependent on the number of colours needed in the image
- In general, **the higher the colour depth the more detail in the image** (higher quality)

What is the impact of resolution and colour depth?

- The file size and quality of the image increases as the **image resolution** and **colour depth** increase.

How images are represented in computer.

- Images are made up of pixels
- Each pixel stores one colour, which is represented using a binary value
- Set number of pixels wide by pixels high
- Each colour has a unique binary value
- The binary value of each pixel is stored in sequence, allowing the computer to reconstruct the image.

Worked Example

1. Define the term Pixel **[1]**
2. If an image has a colour depth of 2 bits, how many colours can the image represent? **[1]**
3. Describe the impact of changing an images resolution from 500×500 to 1000×1000 **[2]**

Answers

1. The smallest element of a bitmap image (1 square)
2. 4
3. - The image quality would be higher
   - The file size would be larger

Units of Data Storage

What are units of data storage?

- A unit of data is a **term** given to describe **different amounts of binary digits** stored on a digital device
- These are the units you need to know for IGCSE:

Name of memory	Size	Number of Bytes
1 nibble	4 bits	
1 Byte (B)	8 bits	
1 Kibibyte (KiB)	1024 Bytes	2^{10}
1 Mebibyte (MiB)	1024 KiB	2^{20}
I Gibibyte (GiB)	1024 MiB	2^{30}
1 Tebibyte (TiB)	1024 GiB	2^{40}
1 Pebibyte (PiB)	1024 TiB	2^{50}
1 Exbibyte (EiB)	1024 PiB	2^{60}

Converting between units

- It is often a requirement of the exam to be able to **convert between different units** of data, for example bytes to mebibytes (larger) or kibibytes to bytes (smaller)

	Name of memory	
Multiply by 8	Bit	**Divide by 8**
	Byte (B)	
Multiply by 1024	Kibibyte (KiB)	**Divide by 1024**
	Mebibyte (MiB)	
	Gibibyte (GiB)	
	Tebibyte (TiB)	
	Pebibyte (PiB)	
	Exbibyte (EiB)	

Calculating File Sizes

How do you calculate the size of a bitmap image?

- Calculating the size of a bitmap image can be carried out with either of the following formulas:
  - **Resolution x colour depth**
  - **Image width x image height x colour depth**

Example

A 16-bit colour image has a resolution of 512 pixels wide by 512 pixels high.

Calculate the file size of the image in kibibytes(Kib).

- **Step 1: Convert the colour depth bits to bytes**

  16/8 = 2 bytes

- **Step 2: Use the formula**

  Resolution x colour depth

  512 x 512 x 2

- **Step 3: Convert the values to 2 power numbers if possible to carry out calculations easy**

  $= 2^9 \times 2^9 \times 2^1$

  $= 2^{19}$ bytes

- **Step 4: Convert the values to KiB**

  $= 2^{19} / 2^{10}$ ~ we can write 1024 as 2^{10}

  $= 2^{9}$

  = 512 KiB

How do you calculate the size of a sound file?

- Calculating the size of a sound file is carried out with the following formula:
  - **Sample rate x duration x sample resolution**

Example

Sample rate: 100 samples per second

Duration: 60 seconds

Sample resolution: 24 number of bits stored per sample

- **Step 1: Convert the sample resolution bits to bytes**

  24 / 8 = 3 bytes

- **Step 2: Use the formula**

  Sample rate x duration x sample resolution

  100 x 60 x 3

- **Step 3: Convert the values to 2 power numbers if possible to carry out calculations easy**

  Not possible in this question as these numbers are not a power 2 numbers. So ignore this step.

- **Step 4: Solve the calculation**

  = 100 x 60 x 3

  = 18000 bytes

- **Step 5: Convert the values to KiB**

  = 18000 / 1024

  = 17 KiB

Compression

What is compression?

- Compression is **reducing the size of a file** so that it **takes up less space on secondary storage**
- The impact of compression is:
  - **Less storage space required**
  - **Less bandwidth required**
  - **Shorter transmission time**
- Compression can be achieved using two methods, **lossy** and **lossless**

What is lossy compression?

- Lossy compression is when **data is lost** in order to **reduce the size** on secondary storage
- Lossy compression is not able to **bring back to original file**
- Lossy can **greatly reduce the size of a file** but at the **expense of losing quality**
- Lossy is only suitable for data **where reducing quality is acceptable**, for example images, video and sound

- In the images above, lossy compression is applied to a photograph and dramatically reduces the file size
- Data has been removed and the overall quality has been reduced, however it is acceptable as it is difficult to visually see a difference
- Lossy compressed photographs take up less storage space which means you can store more and they are quicker to share across a network

Advantages and disadvantages of using lossy compression

Advantages	Disadvantages
Smaller file sizes	Loss of quality
Faster transmission speed	Cannot be used for text or program files, as lost data may render them unusable.
Less bandwidth required	Cannot be restored to original

How an image is compressed using lossy compression?

- A compression algorithm is used
- Identify the similar colours and unnoticed colours
- Remove those colours
- Reduce color depth
- Reduce image resolution
- Data is permanently removed

How a sound file is compressed using lossy compression?

- A compression algorithm is used
- Identify the unnecessary sounds like background noise/sounds human can't hear
- Remove those sounds
- Reduce the sample resolution
- Reduce the sample rate
- Data is permanently removed

How a video file is compressed using lossy compression?

- A compression algorithm is used
- Removes all the unnecessary data
- Reduce the colour depth
- Reduce the resolution
- Reduce the frame rate
- Data is permanently removed

What is lossless compression?

- Lossless compression is when data is **encoded** in order to **reduce the size on secondary storage**
- Lossless compression is **reversible**, the file **can be returned to its original state**
- Lossless can reduce the size of a file **but not as dramatically** as lossy
- Lossless can be used on all data but is more suitable for data where **a loss in quality is unacceptable**, for example documents
- In a document, lossless compression algorithms such as **run length encoding (RLE)** can be used to analyse the contents looking for **patterns** and **repetition**
- Lossless compressed documents **take up less storage space** which means you can **store more** and they are **quicker to share** across a network

What is Run Length Encoding?

- Run length encoding (**RLE**) is a form of lossless data compression that **changes identical elements into a single value** with a count
- For a text file, "**AAAABBBCCDAA**" is compressed to "**4A3B2C1D2A**"
- The string has four 'A's, followed by three 'B's, two 'C's, one 'D', and two 'A's
- RLE commonly used in **bitmap images** to compress **sequences of the same colour**
- For example, a line in an image with 5 red pixels followed by 3 blue pixels could be represented as "**5R3B**". RLE is only effective when there is a long run of a repeated units/bits
- When you open a lossless compressed document the **decompression process reverses the algorithms** and returns the data **back to its original state**

How a text file is compressed using lossless compression?

- A compression algorithm is used
- No data is removed in the process
- Repeated words are identified and indexed
- Index values are saved in a table to recover original file later
- Data can be fully reconstructed

Advantages and disadvantages of using lossless compression

Advantages	Disadvantages
No quality loss	Larger file sizes compared to lossy
The original file can be fully restored	Slower compression & decompression process
Less storage space require	May not provide sufficient size reduction for large multimedia files.

Worked Example

An email is sent containing a sound file.

Lossy compression is used to compress the sound file.

Explain two reasons why using **lossy** compression is beneficial. **[4]**

How to answer this question

- What are the differences between lossy and lossless?
- Can you state two differences? [2 marks]
- Can you say why each point is a benefit? [2 marks]

Answers

- Lossy will decrease the file size **[1]**
- ...so it can sent via email quicker **[1]**
- Lossy means data is lost **[1]**
- ...the difference is unlikely to be noticed by humans **[1]**