OCR Computer Science GCSE

1.2 – Memory and storage

Advanced Notes

1.2.1 Primary Storage (memory)

The Need for Primary Storage

Primary storage is the computer's workspace for actively running programs, and provides fast access to data and instructions currently in use by the CPU. Without it, computers would be significantly slower, as the CPU would need to constantly retrieve data from slower secondary storage like hard drives. Primary storage usually consists of RAM and ROM.

RAM and ROM

RAM stands for Random Access Memory, and it is a form of main memory. RAM holds the data and instructions that the computer is currently working with, such as the operating system, running applications, and open documents. RAM is volatile, meaning its contents are lost when the computer loses power (e.g., when turned off).

ROM stands for Read-Only Memory, and it is a form of main memory. It is typically used to store firmware that is essential for the computer to boot up and operate. As the name suggests, it is read-only; it cannot be written to or modified during normal operation. It is also non-volatile, meaning that it retains its contents even when the power is off.

Virtual Memory

Virtual Memory is needed when a computer's RAM is full and there are still more programs or data that need to be loaded. It allows the system to use part of the secondary storage (such as a hard drive or SSD) as if it were extra RAM. When this happens, the operating system moves data that is not currently needed from RAM to virtual memory (on the hard drive), creating space in RAM for new data. When the data in virtual memory is needed again, it is swapped back into RAM, possibly replacing other data. This process is slower than using RAM alone but allows the system to run more programs than it could with just physical RAM.

1.2.2 Secondary storage

The Need for Secondary Storage

Secondary storage is considered to be any non-volatile storage mechanism not directly accessible by the CPU. Secondary storage is needed so that data/files can be stored on a long-term basis, using non-volatile storage so that they are retained when the computer is switched off.

Types of Secondary Storage

Three types of secondary storage are solid-state, optical and magnetic.

Solid-state

Solid State Drives (SSDs), and other solid state technology like flash drives, use electrical circuits to persistently store data. They don't have any moving parts, so are capable of far higher read and write speeds than magnetic HDDs and are suitable for use in portable devices like phones and tablets.

Optical

Optical disks include CDs, DVDs and Blu-rays. They store information which can be read optically by a laser. They are typically used for the distribution of media as they have an incredibly cheap cost per disk. However, their low capacity does not make them suitable for storing large amounts of data (e.g., they are unsuitable for large-scale backups).

Magnetic

On a magnetic hard disk, data is represented by many, tiny magnetised regions. These magnetic hard disks often contain moving parts. Due to these moving parts and its cheap cost, magnetic storage is typically used to make backups of large amounts of data. However, they are not suited for portable devices.

Comparison of Secondary Storage Devices

	Hard-disk drive	Solid-state drive	Optical disk
Capacity	High capacity.	Relatively low capacity.	Very low capacity.
Read / write speeds	Good speeds.	Very high speeds.	Relatively low speeds.
Portability	Bulky, heavy and easily damaged by movement.	Lightweight and rarely damaged by movement.	Very small and lightweight, can be damaged by scratches and dirt.
Durability	Contains moving parts, prone to damage.	No moving parts, very durable.	Easily scratched or damaged.
Reliability	Fairly reliable but degrades over time.	Very reliable.	Less reliable - damage affects data easily.
Cost	Cheap per GB.	Expensive per GB.	Very cheap per disk, but expensive per GB.
Suitability	Good for desktop PCs and servers.	Good for laptops, phones and tablets.	Good for sharing and distributing small volumes of data.

1.2.3 Units

Binary

Binary is used by computer systems to store all data and instructions. This is because it has only two states, 0 or 1, which map directly to the two states of electronic components like transistors: on (1) or off (0). This simplicity makes it easier to design, build, and maintain computer hardware. Therefore, data needs to be converted into a binary format to be processed by a computer.

Each digit is a bit of data. You'll often come across the following prefixes used for decimal numbers, and you need to be able to convert between them.

Unit	Symbol	Relative size
Bit	b	1 bit
Nibble		4 bits
Byte	B	8 bits
Kilobyte	KB	1,000 bytes
Megabyte	MB	1,000 kilobytes
Gigabyte	GB	1,000 megabytes
Terabyte	TB	1,000 gigabytes
Petabyte	PB	1,000 terabytes

Note about prefixes: *the specification uses decimal (base-10) prefixes, as shown in the table above. These differ from binary prefixes (e.g. kibibyte = 1,024 bytes), but you only need to know about the prefixes in the table above for the OCR GCSE Computer Science (J277) exam.*

To calculate the file sizes of sound, images and text files, you can use the following formulas:

$$Sound\ file\ size = sample\ rate \times duration\ (s) \times bit\ depth$$

$$Image\ file\ size = colour\ depth \times image\ height\ (px) \times image\ width\ (px)$$

$$Text\ file\ size = bits\ per\ character \times number\ of\ characters$$

The coming pages will explain how sound, images and text files are represented using binary, and how the above formulas work.

1.2.4 Data storage

Decimal (base 10)

Decimal is the number base that humans use to count, perhaps because we have ten fingers. Decimal uses the ten digits 0 through to 9 to represent numbers.

Each digit in a decimal number has a place value based on powers of 10. The value of a digit depends on its position within the number. This is illustrated by the table below, which shows how the decimal number 237 is constructed using place values.

10^2	10^1	10^0
100	10	1
2	3	7

`237 = (2×100) + (3×10) + (7×1)`

Binary (base 2)

Each digit in a binary number has a place value based on powers of 2. This is illustrated by the table below, which shows how the decimal number 1011 is constructed using place values - making it equal to 11 in decimal.

2^3	2^2	2^1	2^0
8	4	2	1
1	0	1	1

`1011 = (1×8) + (0×4) + (1×2) + (1×1) = 11 (decimal)`

Most Significant and Least Significant Bit

The most significant bit is the bit with the highest value, which is the leftmost 1 in a binary number. The least significant bit is the bit with the lowest value, which is the rightmost bit, whether it is a 0 or 1, in a binary number.

Adding additional 0s to the left of a binary number does not change its value, e.g. 11010 is the same as 00011010.

Converting Decimal ↔ Binary

To convert binary → decimal:

You can convert from binary to decimal by using place value headers. Starting with one and increasing in powers of two, placing larger values to the left of smaller values. For example, the binary number `10110010` could have place value headers added as follows:

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	1	1	0	0	1	0

The binary number could then be converted to decimal by adding together all of the place values with a binary one below them.

$$128 + 32 + 16 + 2 = 178$$

So the binary number `10110010` is equivalent to the decimal number 178.

To convert decimal → binary:

When converting from decimal to binary, you use the same place value headers. Starting from the left hand side, you place a one if the value is less than or equal to your number, and a zero otherwise.

Once you've placed a one, you must subtract the value of that position from your number and continue as before, until your number becomes 0.

Let's say we're converting the number 53 to binary. First, write out your place value headers in powers of two. Keep going until you've written a value that is larger than your number. For 53, we're going to go up to 64.

64	32	16	8	4	2	1

Now, starting from the left, compare the place value to your number. 64 is greater than 53 so we place a 0 under 64.

64	32	16	8	4	2	1
0						

Moving to the right, we see that 32 is lower than 53, so we place a 1 under 32.

64	32	16	8	4	2	1
0	1					

Because we've placed a 1, we have to subtract 32 from 53 to find what's left to be represented. In this case, 53 - 32 = 21.

We move to the right again and find 16, which is lower than 21, so we place a 1 under 16.

64	32	16	8	4	2	1
0	1	1				

Again, because we've placed a 1, we have to calculate a new value. 21 - 16 = 5.

Moving right, we find 8. This is larger than 5 so we place a 0.

64	32	16	8	4	2	1
0	1	1	0			

After moving right again, we find 4. As 4 is lower than 5, we place a 1.

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

Having placed a 1, we must again calculate a new value. 5 - 4 = 1.

Moving right to find 2, we place a 0 as 2 is greater than 1.

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

Moving right for the last time, we have 1. 1 = 1 so we place a 1.

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

Now that we've placed a 0 or a 1 under each place value, we have our answer. Although it's acceptable to remove any leading 0s, it may be preferable to add 0s to the start of your answer to make it a whole number of bytes (a multiple of 8 bits).

$$53 = 0110101 = 110101 = 00110101$$

Binary addition

When adding binary numbers, there are three important rules to remember:

Binary add	Result	Carry
0 + 0	0	0
1 + 0	1	0
1 + 1	0	1 (carry)

You'll only be expected to add two binary numbers of up to 8 bits.

Note: an overflow error can occur where the result of a binary addition is too large to fit into the number of bits available. For example, if the result needs 9 bits and you only have 8 available, the extra (most-significant) bit would be lost and the final value would be incorrect.

Example

Add binary integers 1011 and 1110.

	1	0	1	1
+	1	1	1	0

Place the two binary numbers above each other so that the digits line up.

	1	0	1	1
+	1	1	1	0
				1

Starting from the least significant bits (the right hand side), add the values in each column and place the total below. For the first column (highlighted), rule 2 from above applies.

	1	0	1	1
+	1	1	1	0
		^{1}	0	1

Move on to the next column. This time rule 3 applies. In this case there is a carry digit. Place a 1 in small writing under the next most significant bit's column.

	1	0	1	1
+	1	1	1	0
	^{1}	0^{1}	0	1

On to the next column, where there is a 0, a 1 and a small 1. In this case, rule 3 applies again. Therefore the result is 10. Because 10 is two digits long, the 1 is written in small writing under the next most significant bit's column.

	1	0	1	1
+	1	1	1	0
1	1^{1}	0^{1}	0	1

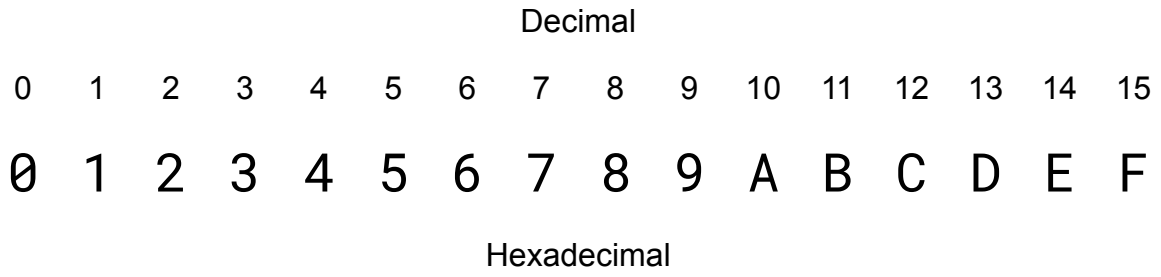
Moving on to the most significant column where there are three 1s. Rule 4 applies, so the result for this column is 11. The first digit of the result is written under the next most significant bit's column, but it can be written full size as there are no more columns to add.

1	1	0	0	1

Finally, the result is read off from the full size numbers at the bottom of each column. In this case, 1011 + 1110 = 11001.

Hexadecimal (base 16)

In contrast to decimal, hexadecimal uses the digits 0 through to 9 followed by the uppercase characters A to F to represent the decimal numbers 0 to 15.



Of all the number bases covered by this course, hexadecimal is the most compact. This means that it can represent the same number as binary or decimal while using far fewer digits. This makes it easier than binary to read or work with. Each character in hexadecimal represents four bits in binary.

Each digit in a decimal number has a place value based on powers of 16. This is illustrated by the table below, which shows how the hexadecimal value 2F is constructed using place values - making it equal to 47 in decimal.

16^1	16^0
16	1
2	15 (because F represents 15)

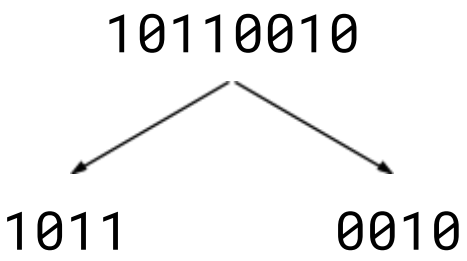
```
2F = (2×16) + (15×1)  = 47 (decimal)
```

Converting Binary ↔ Hexadecimal

To convert binary → hex:

In order to convert from binary to hexadecimal, the binary number must first be split into nibbles. A nibble is four binary bits, or half a byte.

For example, the binary number 10110010 would be split into two nibbles:



Each binary nibble is then converted to decimal as in the previous example:

8	4	2	1
1	0	1	1

8 + 2 + 1 = 11

8	4	2	1
0	0	1	0

2 = 2

Once each nibble has been converted to decimal, the decimal value can be converted to its hexadecimal equivalent like so:

11 = B

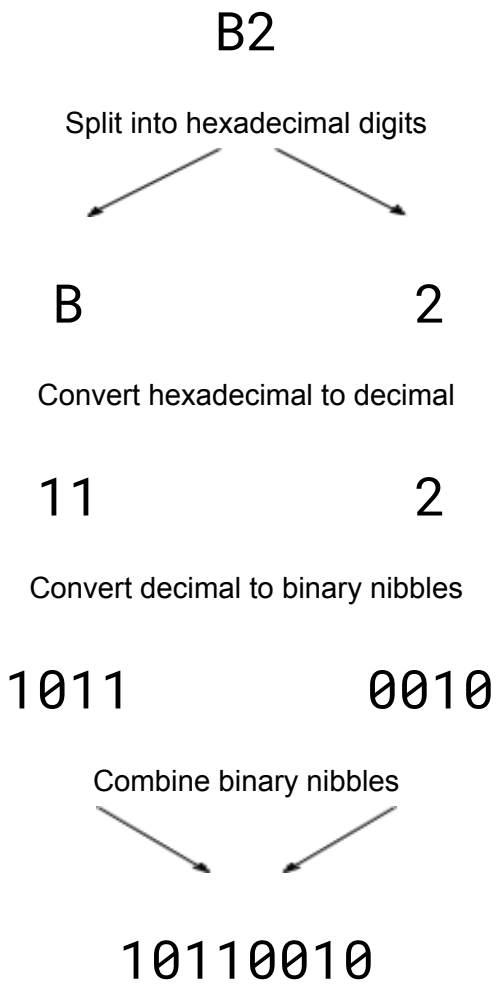
2 = 2

Finally, the hexadecimal digits are concatenated to form a hexadecimal representation:

10110010 = B2

To convert hex → binary:

First, convert each hexadecimal digit to a decimal digit and then to a binary nibble before combining the nibbles to form a single binary number.



Converting Decimal ↔ Hexadecimal

To convert decimal → hex:

Combining the steps above:

1. Begin by converting the decimal number into binary
2. Convert this binary number to hexadecimal

To convert hex → decimal:

Combining the steps above:

1. Begin by converting the hexadecimal number into binary.
2. Convert this binary number to decimal.

Binary Shifts

A binary shift involves moving the bits of a binary number left or right. Bits shifted from the end of the register are lost and zeros are shifted in at the opposite end of the register.

There are two types of binary shift:

- Left shift → moves all bits to the left (adds 0s on the right)
    - Same as multiplying by 2 for each place shifted
- Right shift → moves all bits to the right (adds 0s on the left)
    - Same as dividing by 2 for each place shifted

Example

In this example, we'll apply a binary left shift of 1 to the original binary number `00101100`. The effect of this is to multiply 44 by 2, making 88.

Original:	`00101100 (44)`
Shifted:	`01011000 (88)`

Why use binary shifts?

- To multiply or divide by powers of 2
- Used in low-level graphics, bitmasking, compression, and encryption

Characters

Character encoding

Character encoding is the process of converting characters (letters, numbers, symbols) into binary codes so that they can be stored and processed by a computer's hardware. This is necessary because computers can only store and process binary data.

A character set, such as ASCII or Unicode, is a collection of characters and their corresponding binary values. Every character is assigned a unique binary code, using a standard such as ASCII or Unicode. Character codes are grouped and they run in sequence. For example in ASCII 'A' is coded as 65, 'B' as 66, and so on, meaning that the codes for the other capital letters can be calculated once the code for 'A' is known. This pattern also applies to other groupings such as lower case letters and digits.

The number of characters stored is limited by the number of bits available, as each character must have a unique representation in binary, and the number of possible unique representations in binary depends on the number of bits.

The number of bits per character depends on the character set, with ASCII using 8 bits per character and Unicode using between 8 and 32. This allows Unicode to have a greater number of characters, as there are more unique codes possible with a greater number of bits. Therefore, Unicode is typically used when characters from other languages, and even emojis, need to be represented.

Calculating text file size

$$Text\ file\ size\ =\ bits\ per\ character \times number\ of\ characters$$

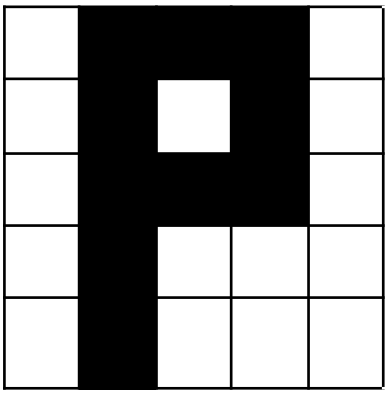
For example, if the string "Physics and Maths Tutor" was encoded using 7 bits per character, then as there are 23 characters, the text file size would be:

$$text\ file\ size\ =\ 7 \times 23\ =\ 161\ bits$$

Images

Digital images are made up of a series of tiny squares called pixels (short for “picture elements”). A pixel is a single point in an image. Each pixel has a colour value, and this is stored in binary.

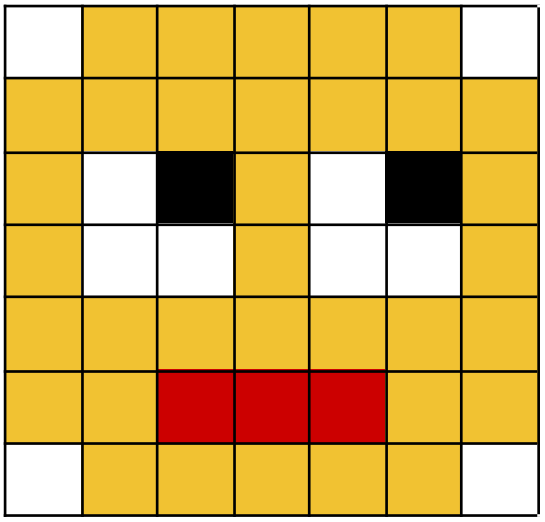
The value assigned to a pixel determines the colour of the pixel. The example below shows the binary representation of a simple image in which a 1 represents a black pixel and a 0 represents a white pixel.



```
0 1 1 1 0
0 1 0 1 0
0 1 1 1 0
0 1 0 0 0
0 1 0 0 0
```

The number of bits assigned to a pixel in an image is called its colour depth. In the example above, each pixel has been assigned one bit, allowing for 2 (2^1) different colours to be represented. If a colour depth of two bits were used, there would be four (2^2) different colours that each pixel could take, represented by the bit patterns 00, 01, 10 and 11.

The resolution refers to the number of pixels within an image. Resolution can be found by multiplying the image width in pixels by the image height in pixels.



```
00 11 11 11 11 11 00
11 11 11 11 11 11 11
11 00 01 11 00 01 11
11 00 00 11 00 00 11
11 11 11 11 11 11 11
11 11 10 10 10 11 11
00 11 11 11 11 11 00
```

Image metadata is data about an image such as: file format, resolution, colour depth, and sometimes details like the device used to capture the image.

Calculating image file size

$$Image\ file\ size\ =\ colour\ depth \times image\ height\ (px) \times image\ width\ (px)$$

For example, the picture of the face has 7 × 7 = 49 pixels, each of which is assigned two bits, so it requires 98 bits to be represented.

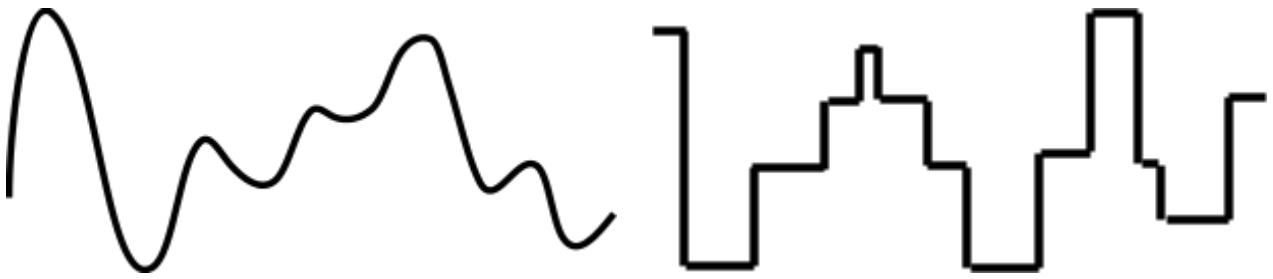
$$File\ size\ =\ 7 \times 7 \times 2\ =\ 98\ bits$$

Effect on image size, quality and file size

Looking at the equation above, higher widths, higher heights and higher colour depths will all increase an image's file size.

Increases in…	Effect on…
Resolution	Higher quality & higher file size
Colour depth	Higher quality & higher file size

Sound



Analogue signal

Digital signal

Sound is analogue, meaning that its signal is a continuous wave that can take any value. Computers cannot store continuous sound waves, so they take regular snapshots (samples) of the sound wave's amplitude. A sample is a measure of amplitude at a point in time - each sample is stored as a binary number.

The sampling rate is the number of samples taken in a second and is usually measured in hertz (1 hertz = 1 sample per second).

The bit depth is the number of bits available to store each sample.

Calculating sound file size

$$Sound\ file\ size\ =\ sample\ rate \times duration\ (s) \times bit\ depth$$

To calculate the file size of a sound file in bytes, divide the file size in bits by 8.

Example

For a sound file with a:

- Sample rate = 44,000 Hz
- Duration = 10 seconds
- Sample resolution = 16 bits

$$File\ size\ =\ 44000 \times 10 \times 16\ =\ 7040000\ bits$$

Effect on playback quality and file size

Increases in…	Effect on…
Sampling rate	Higher playback quality & larger file size
Bit depth	Higher playback quality & larger file size

An increase in both sampling rate and bit depth make the digital sound wave more accurate to the real analogue wave. This increases the quality of the audio.

1.2.5 Compression

Data compression is the process of reducing the file size of digital data without losing the original information (or with minimal acceptable loss). It is used to save storage space and speed up transmission.

Why compress data?

- Saves storage space
- Speeds up file transfer
- Reduces bandwidth usage
- Helps with faster downloads and streaming

Types of compression

Lossy compression
When using lossy compression, some information is lost in the process of reducing the file's size. This could cause the quality of the file to be slightly reduced; the compressed file can never be fully restored to the original.

Used for:

- Images
- Audio
- Video

Pros	Cons
✔ Smaller file size	✘ Loss of quality
✔ Faster to send/store	✘ Irreversible (original data gone)

Lossless compression
In contrast to lossy compression, there is no loss of information when using lossless compression. The size of a file can be reduced without decreasing its quality. Lossless compression methods use algorithms to find and compress patterns (e.g. repeated data).

Pros	Cons
✔ No loss of quality	✘ Less reduction in size compared to lossy
✔ Reversible	