

Information representation

A-Level Computer Science 9618 · Topic 1 revision notes

Everything from the 7 lessons of this topic on a few pages: the key idea of each lesson, the diagram to remember, the points to learn and the traps examiners set. Read it, cover it, test yourself.

- | | |
|---|--|
| 1. <u>Binary and decimal prefixes</u> | 5. <u>Bitmap and vector graphics</u> |
| 2. <u>Binary, denary, hexadecimal and BCD</u> | 6. <u>Sampled sound</u> |
| 3. <u>Two's complement and binary arithmetic</u> | 7. <u>Lossy and lossless compression, run-length</u> |
| 4. <u>Character sets: ASCII, extended ASCII and Unicode</u> | <u>encoding.</u> |

1 Binary and decimal prefixes

Syllabus 1.1

Decimal prefixes go up in powers of 10 (steps of 1000): kilo = 10^3 , mega = 10^6 , giga = 10^9 , tera = 10^{12} . **Binary prefixes** go up in powers of 2 (steps of 1024): kibi = 2^{10} , mebi = 2^{20} , gibi = 2^{30} , tebi = 2^{40} . So 1 KiB = 1024 bytes but 1 kB = 1000 bytes. Memory sizes are naturally powers of 2, which is why the binary prefixes exist.

decimal prefixes ($\times 1000$)

kilo kB 10^3

mega MB 10^6

giga GB 10^9

tera TB 10^{12}

binary prefixes ($\times 1024$)

kibi KiB 2^{10}

mebi MiB 2^{20}

gibi GiB 2^{30}

tebi TiB 2^{40}

the binary unit is always a little bigger: 1 KiB = 1024 bytes but 1 kB = 1000 bytes

Fig. 1: decimal prefixes multiply by 1000 each step; binary prefixes multiply by 1024

Worked example: Binary units

How many bytes are there in 4 MiB? How many bits is that?

1. 1 MiB = 2^{20} = 1 048 576 bytes.
2. $4 \times 1\,048\,576 = 4\,194\,304$ bytes.
3. Each byte is 8 bits: $4\,194\,304 \times 8 = 33\,554\,432$ bits.

Answer: 4 194 304 bytes = 33 554 432 bits

1000 or 1024.

- ✓ **Decimal:** kilo 10^3 , mega 10^6 , giga 10^9 , tera 10^{12} (steps of 1000).
- ✓ **Binary:** kibi 2^{10} , mebi 2^{20} , gibi 2^{30} , tebi 2^{40} (steps of 1024).
- ✓ **Switching ladders:** convert to bytes first, then divide by the other unit.
- ✓ **Bits and bytes:** 1 byte = 8 bits; B is bytes and b is bits.

Watch out!

Read the symbol carefully. MB means 10^6 bytes and MiB means 2^{20} bytes. If a question uses MiB, divide by 1024 at each step, not 1000.

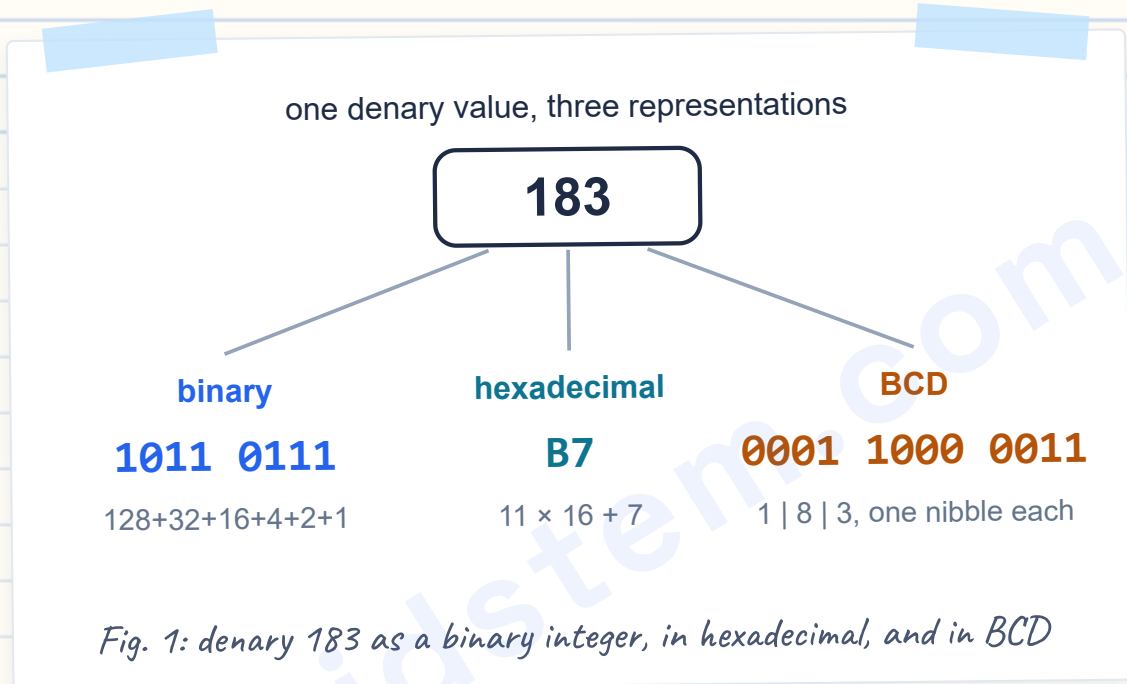
A capital B is bytes and a small b is bits. 100 Mbps is 100 000 000 bits per second, which is 12 500 000 bytes per second.

thelucidstem.com

2 Binary, denary, hexadecimal and BCD

Syllabus 1.1

Denary is base 10, **binary** is base 2 and **hexadecimal** is base 16 (digits 0 to 9 then A to F). One hex digit is exactly **4 bits**, so binary and hex convert a nibble at a time. **Binary Coded Decimal (BCD)** is different: each denary digit is stored separately in its own 4-bit nibble, so 59 is 0101 1001. Nibbles 1010 to 1111 are never valid in BCD.



Worked example: Denary to binary and hex

Convert 237 to 8-bit binary and to hexadecimal.

1. $237 = 128 + 64 + 32 + 8 + 4 + 1$.
2. Binary: 1110 1101.
3. 1110 = 14 = E and 1101 = 13 = D.

Answer: 1110 1101 = ED

Nibbles are the key.

- ✓ **Binary and hex:** one hex digit = one nibble; group from the right.
- ✓ **Hex to denary:** place values 4096, 256, 16, 1.
- ✓ **BCD:** one nibble per denary digit; 1010 to 1111 are invalid.
- ✓ **Uses:** BCD for displays and money; hex for memory addresses, dumps, colours and MAC addresses.

Watch out!

Do not convert the whole number to binary and call it BCD. In BCD, 59 is 0101 1001 (5 then 9), but 59 in pure binary is 0011 1011.

Hexadecimal is for humans. The computer never stores or processes hex: it converts it to binary. Saying "hex uses less memory" earns no marks.

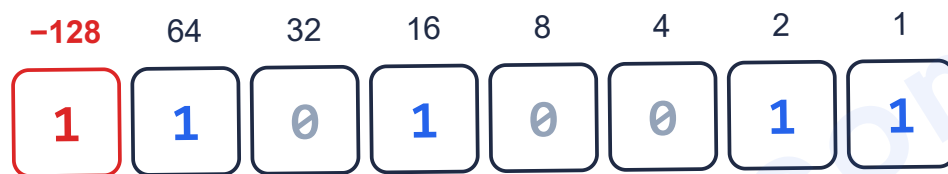
thelucidstem.com

3 Two's complement and binary arithmetic

Syllabus 1.1

In **two's complement** the most significant bit has a **negative place value**: in 8 bits it is -128 , so the range is **-128 to $+127$** . To negate a number, **invert every bit** (this alone gives the **one's complement**) and then **add 1**. Subtraction is done by adding the negative: $A - B = A + (-B)$. **Overflow** occurs when the true result is outside the range: adding two numbers with the same sign gives a result with the opposite sign.

8-bit two's complement: the leftmost place value is negative



$$-128 + 64 + 16 + 2 + 1 = -45$$

$$+45 = 00101101 \text{ invert: } 11010010 \text{ add 1: } 11010011$$

range in 8 bits: -128 to $+127$

Fig. 1: 11010011 in 8-bit two's complement is -45

Worked example: Denary to two's complement

Write -45 in 8-bit two's complement.

1. Write $+45$ in 8 bits: 00101101 .
2. Invert every bit (one's complement): 11010010 .
3. Add 1: 11010011 .
4. Check: $-128 + 64 + 16 + 2 + 1 = -45$.

Answer: 11010011

The sign bit is negative.

- ✓ **Two's complement:** leftmost place value is -128 in 8 bits; range -128 to $+127$.
- ✓ **Negate:** invert every bit, then add 1 (inverting alone gives one's complement).
- ✓ **Subtract:** $A - B = A + (-B)$; ignore a carry out of the leftmost bit.
- ✓ **Overflow:** two positives give a negative, or two negatives give a positive.

Watch out!

Inverting the bits is not the whole job for two's complement.

Forgetting to add 1 gives the one's complement, which is out by one.

A carry out of the leftmost bit is not the test for overflow in two's complement. Check the signs: overflow only when two positives give a negative or two negatives give a positive.

thelucidstem.com

4 Character sets: ASCII, extended ASCII and Unicode

Syllabus 1.1

A **character set** is the complete list of characters a computer can represent, with a unique binary **code** for each one. **ASCII** uses **7 bits** (128 characters: English letters, digits, punctuation and control codes). **Extended ASCII** uses **8 bits** (256 characters), adding accented letters and symbols. **Unicode** uses up to **32 bits** per character so it can represent every writing system and emoji; its first 128 codes match ASCII. Codes for letters are **consecutive**, so you can work out one code from another.

each character is stored as its code in the character set

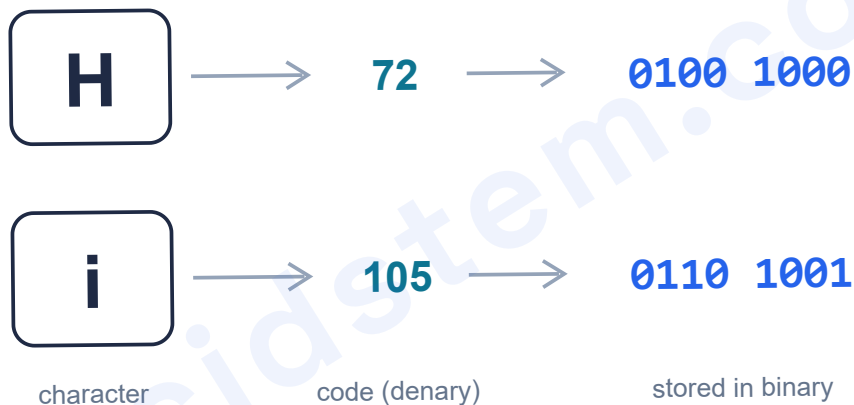


Fig. 1: the text "Hi" stored as two character codes

Worked example: Counting along

The ASCII code for 'A' is 65. Give the 8-bit binary code for 'G' and for 'g'.

1. G is 6 letters after A: $65 + 6 = 71$.
2. $71 = 64 + 4 + 2 + 1$, so 01000111.
3. Lower case is 32 more: $71 + 32 = 103 = 64 + 32 + 4 + 2 + 1$, so 01100111.

Answer: 'G' = 01000111, 'g' = 01100111

One code per character.

- ✓ **ASCII:** 7 bits, 128 characters; extended ASCII 8 bits, 256 characters.
- ✓ **Unicode:** up to 32 bits, every language and emoji; first 128 codes match ASCII.
- ✓ **Codes are consecutive:** count along from the code you are given; lower case = upper case + 32 in ASCII.
- ✓ **Storage:** number of characters $\times$ bits per character.

Watch out!

You will not be asked to memorise codes. The exam gives you one, such as 'A' = 65, and expects you to work out the others.

A digit stored as a character is not the same as the number. The string "7" is code 55; the integer 7 is 00000111.

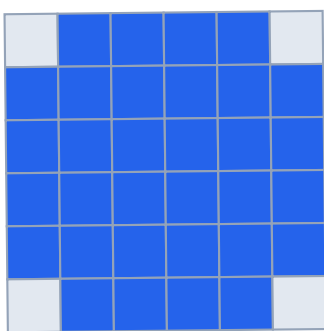
thelucidstem.com

5 Bitmap and vector graphics

Syllabus 1.2

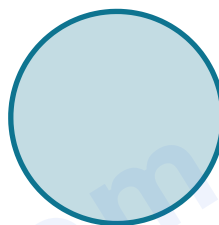
A **bitmap** is a grid of **pixels**; each pixel stores a binary colour code. The **colour depth** (bit depth) is the number of bits per pixel, so it allows 2^n colours. The **image resolution** is the number of pixels (width $\times$ height). A **file header** stores data about the image, such as its dimensions and colour depth. Estimated file size = **width $\times$ height $\times$ colour depth** bits. A **vector graphic** is a **drawing list** of **drawing objects**, each described by its **properties**, so it can be scaled without losing quality.

bitmap: a grid of pixels



6 $\times$ 6 pixels, each a colour code
enlarge it and the steps show

vector: a drawing list



object: circle
centre (250, 90)
radius 40
line colour, width 2
fill colour

properties describe the shape, so it is
redrawn sharply at any size

Fig. 1: a bitmap stores every pixel; a vector stores objects and their properties

Worked example: Full HD photo

Estimate the file size of a 1920 $\times$ 1080 bitmap with a colour depth of 24 bits, in MiB.

1. Number of pixels: $1920 \times 1080 = 2\,073\,600$.
2. 24 bits = 3 bytes per pixel: $2\,073\,600 \times 3 = 6\,220\,800$ bytes.
3. $6\,220\,800 \div 1024 = 6075$ KiB; $6075 \div 1024 = 5.93$ MiB.

Answer: about 5.93 MiB

Pixels times bits.

- ✓ **File size:** width $\times$ height $\times$ colour depth bits; $\div 8$ for bytes, $\div 1024$ for KiB.
- ✓ **Colour depth:** n bits give 2^n colours; find the smallest n that is enough.
- ✓ **Vector:** drawing list of drawing objects, each with properties; scales without pixelation.
- ✓ **Justify:** bitmap for photographs; vector for logos, diagrams and anything resized often.

Watch out!

Do not confuse image resolution with screen resolution. Image resolution belongs to the file; screen resolution belongs to the display.

Multiplying by the number of colours instead of the colour depth is the most common error. 256 colours means 8 bits per pixel, not 256.

"Vectors are smaller" is not always true. A complex scene with thousands of objects can need a larger vector file than a bitmap, and vectors cannot realistically store a photograph.

6 Sampled sound

Syllabus 1.2

Sound is **analogue data**: it varies continuously. To store it, an analogue-to-digital converter **samples** the amplitude of the wave at regular intervals and stores each sample as a binary integer, giving **digital data** made of discrete values. The **sampling rate** is the number of samples taken per second (measured in hertz). The **sampling resolution** is the number of bits used for each sample, giving 2^n possible amplitude levels. Raising either one makes the recording **more accurate** but the file **larger**. File size = sampling rate $\times$ sampling resolution $\times$ time ($\times$ number of channels) bits.

sampling an analogue wave at regular intervals

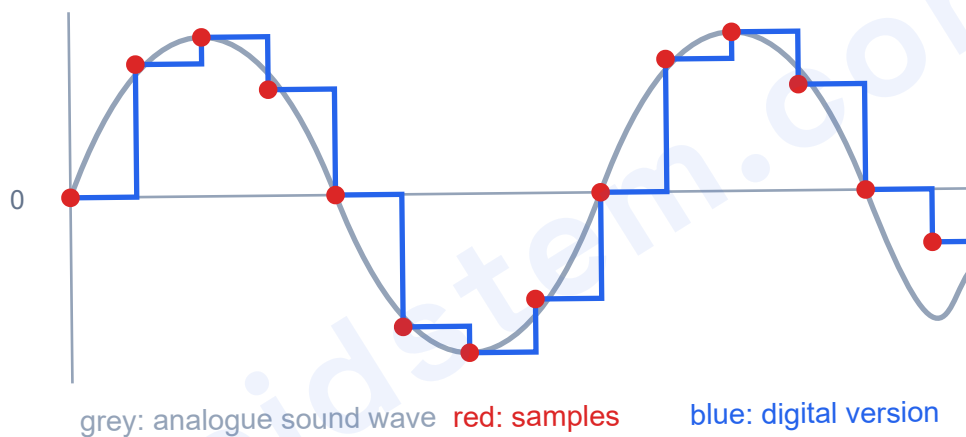


Fig. 1: the wave is measured at regular intervals and each sample is stored as a binary number

Worked example: CD-quality minute

Calculate the size of 60 seconds of mono sound sampled at 44 100 Hz with a 16-bit resolution, in MiB.

1. Bits: $44\,100 \times 16 \times 60 = 42\,336\,000$ bits.
2. $\div 8 = 5\,292\,000$ bytes.
3. $\div 1\,048\,576 = 5.05$ MiB (2 d.p.).

Answer: about 5.05 MiB (twice that in stereo)

Rate times resolution.

- ✓ **Sampling:** measure the amplitude at regular intervals; store each sample in binary.
- ✓ **Rate:** samples per second (Hz).
- Resolution:** bits per sample (2^n levels).
- ✓ **File size:** rate $\times$ resolution $\times$ seconds ($\times 2$ for stereo) bits.
- ✓ **Trade-off:** higher rate or resolution = more accurate but larger file.

Watch out!

Say what is measured: the amplitude (height) of the wave. "Sampling records the sound" is too vague to earn the mark.

Sampling resolution is not the same as sampling rate. Resolution is bits per sample; rate is samples per second. Mixing them up loses marks in every part of the question.

thelucidstem.com

7 Lossy and lossless compression, run-length encoding

Syllabus 1.3

Compression reduces file size so files need **less storage** and less **bandwidth**, and are **transmitted faster**. **Lossless** compression reduces the size with **no loss of data**: the original is rebuilt exactly. **Lossy** compression **permanently removes** data, usually detail that human senses notice least, giving much smaller files. **Run-length encoding (RLE)** is lossless: each run of identical values is replaced by a count and the value.

run-length encoding one row of 12 pixels



W W W W W B B B W W W W



5 W 3 B 4 W

12 values become 3 (count, colour) pairs = 6 values, and decode back exactly

Fig. 1: run-length encoding replaces each run of identical pixels with a count and a colour

Worked example: Text

Use RLE to compress `AAAAABBBCCCCCCCCD`. Each character and each count is stored in 1 byte. How many bytes are saved?

1. Runs: 5 A, 3 B, 8 C, 1 D.
2. Encoded: 5A 3B 8C 1D, which is 4 pairs = 8 bytes.
3. Original: 17 characters = 17 bytes. $17 - 8 = 9$ bytes saved.

Answer: 5A 3B 8C 1D, saving 9 bytes

Exact or close enough.

- ✓ **Why:** less storage, less bandwidth, faster transmission.
- ✓ **Lossless:** original rebuilt exactly (RLE, dictionary methods); for text, programs and critical data.
- ✓ **Lossy:** data permanently removed (perceptual coding, lower resolution or colour depth); for media.
- ✓ **RLE:** count and value for each run; only saves space when there are long runs.

Watch out!

A justification needs a reason linked to the scenario. "Lossy because it is smaller" earns less than "lossy, because the photo only needs to load quickly on a web page and the lost detail will not be noticed".

Never suggest lossy compression for a text file or program. Losing even one character changes the meaning, and a program would no longer run.