

Information representation AS

Read online at pastopic.com/cambridge/computer-science-9618/notes/information-representation

Last checked 30 September 2026

© 2026 Pastopic. Free for personal study. Please share the link, not the file.

What you need to know

By the end of this topic you should be able to:

1. **Use binary and decimal prefixes.** Know that kibi, mebi, gibi and tebi go up in powers of 1024 while kilo, mega, giga and tera go up in powers of 1000, and convert between them.
2. **Work in different number systems:** binary, denary, hexadecimal, Binary Coded Decimal (BCD), and one's and two's complement for negative binary numbers. Convert an integer from any of these to any other.
3. **Add and subtract in binary**, with positive and negative (two's complement) integers, and explain how **overflow** happens.
4. **Describe where BCD and hexadecimal are used** in practice, and why.
5. **Explain how characters are stored in binary** using a character set: ASCII, extended ASCII and Unicode. You never need to remember particular codes.
6. **Explain how a bitmap image is encoded**, using the terms pixel, file header, image resolution, screen resolution and colour depth (bit depth). Estimate a bitmap's file size, and explain what changing its resolution or colour depth does to its quality and file size.
7. **Explain how a vector graphic is encoded**, using the terms drawing object, property and drawing list, and justify choosing a bitmap or a vector graphic for a task.
8. **Explain how sound is sampled and encoded**, using the terms sampling, sampling rate, sampling resolution, analogue data and digital data, and the effect of changing the sampling rate and resolution on accuracy and file size.
9. **Explain why files are compressed**, tell lossy from lossless compression and justify one for a situation, and describe how a text file, a bitmap image, a vector graphic and a sound file can be compressed, including **run-length encoding (RLE)**.

Logical and arithmetic shifts, bit masks, parity checks and floating-point numbers belong to other topics (Processor fundamentals, Security, and the A Level topic Data representation).

Understand it

Bits, bytes and prefixes

A **bit** is one binary digit, 0 or 1. A **byte** is 8 bits. Four bits (half a byte) are a **nibble**. With n bits you can make 2^n different patterns, so 8 bits give $2^8 = 256$ patterns and 16 bits give 65 536.

Large amounts of data need prefixes, and there are two families:

Binary prefix	Size	Decimal prefix	Size
kibi (KiB)	$2^{10} = 1024$ bytes	kilo (kB)	$10^3 = 1000$ bytes
mebi (MiB)	$2^{20} = 1024$ KiB	mega (MB)	$10^6 = 1000$ kB
gibi (GiB)	$2^{30} = 1024$ MiB	giga (GB)	$10^9 = 1000$ MB
tebi (TiB)	$2^{40} = 1024$ GiB	tera (TB)	$10^{12} = 1000$ GB

So 6 KiB = $6 \times 1024 = 6144$ bytes, but 5 MB = 5 000 000 bytes. 4 TiB = 4096 GiB, while 4 TB = 4000 GB. Each binary unit is a little bigger than the decimal unit with the same letter.

Binary, denary and hexadecimal

Denary (base 10) uses the digits 0–9. **Binary** (base 2) uses 0 and 1, and each place is worth double the place to its right:

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

$10110101 = 128 + 32 + 16 + 4 + 1 = 181$. To go the other way, take the biggest place value that fits, write 1, subtract, and carry on: $181 - 128 = 53$, $53 - 32 = 21$, $21 - 16 = 5$, $5 - 4 = 1$, $1 - 1 = 0$.

Hexadecimal (base 16) uses 0–9 and then A–F for 10–15. Its place values are 1, 16, 256, 4096. One hex digit stands for exactly four bits, so to convert binary to hex, split the bits into nibbles **from the right** and replace each nibble:

- $1011\ 0101 \rightarrow B5$ (since $1011 = 11 = B$ and $0101 = 5$).
- 700 in 12-bit binary is 0010 1011 1100, which is 2BC.
- $3E9 = 3 \times 256 + 14 \times 16 + 9 = 768 + 224 + 9 = 1001$.

Hex is used because it is much shorter than binary, easier for people to read and copy without mistakes, and converts to binary instantly. Uses include **memory addresses** and memory dumps when debugging, **colour codes** on web pages (such as #FF8000), **MAC addresses**, error codes, and machine code listings.

Binary Coded Decimal (BCD)

In **BCD** each denary **digit** gets its own 4-bit nibble, instead of converting the whole number:

$$395 \rightarrow 0011 \ 1001 \ 0101$$

To read BCD, convert each nibble to a digit. Only 0000 to 1001 (0 to 9) are used, so a nibble from 1010 to 1111 means the value **cannot** be BCD.

BCD is used where every decimal digit must be exact or shown on its own:

- **money and financial calculations:** an amount such as 0.10 has no exact value in ordinary binary, so rounding errors build up; in BCD each decimal digit is stored exactly;
- **digital displays** (clocks, calculators, meters), where each digit drives its own part of the display;
- the **date and time** kept by a PC's real-time clock (BIOS), where digits are read out and shown directly.

Negative numbers: one's and two's complement

Computers use the leftmost bit (the **most significant bit**, MSB) to show the sign: 0 for positive, 1 for negative.

One's complement: write the positive number, then flip every bit. $105 = 01101001$, so $-105 = 10010110$. It has two zeros (00000000 and 11111111), which is one reason it is rarely used; the 8-bit range is -127 to $+127$.

Two's complement is what computers normally use. The MSB is worth a **negative** amount: -128 in 8 bits, -2048 in 12 bits.

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

$10010111 = -128 + 16 + 4 + 2 + 1 = -105$. Two ways to convert:

- **Denary to two's complement:** write the positive number in binary, **flip every bit, then add 1**. $105 = 01101001 \rightarrow 10010110 \rightarrow 10010111$.
- **Two's complement to denary:** add up the place values, using the negative MSB. Or, if the MSB is 1, flip and add 1 to get the size, then put a minus sign in front.

A positive number looks the same in two's complement as in unsigned binary; it just needs a 0 at the front. The ranges are:

Bits	Unsigned	Two's complement
8	0 to 255	-128 (10000000) to 127 (01111111)
12	0 to 4095	-2048 to 2047
n	0 to $2^n - 1$	-2^{n-1} to $2^{n-1} - 1$

Binary addition and subtraction

Add column by column from the right, carrying as in denary: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 1 = 10$ (write 0, carry 1), $1 + 1 + 1 = 11$ (write 1, carry 1).

$$\begin{array}{r} 01011011 \\ + 00110110 \\ \hline 10010001 \end{array}$$

This is $91 + 54 = 145$. As an unsigned number that is fine. But if the register holds **two's complement** numbers, 145 is bigger than the largest value, 127: the answer 10010001 reads as -11 . That is **overflow**: the true result cannot be represented in the number of bits available, so the stored answer is wrong. Overflow also happens when a carry comes out of the leftmost column of an unsigned sum, or when a result is below the most negative value.

Subtraction can be done by borrowing, column by column, or by **adding the two's complement** of the number you take away. For $97 - 45$: $45 = 00101101$, so $-45 = 11010011$, and

$$01100001 + 11010011 = (1) 00110100 = 52.$$

When you subtract this way, the carry out of the 8th bit is ignored: it is not an error.

Characters

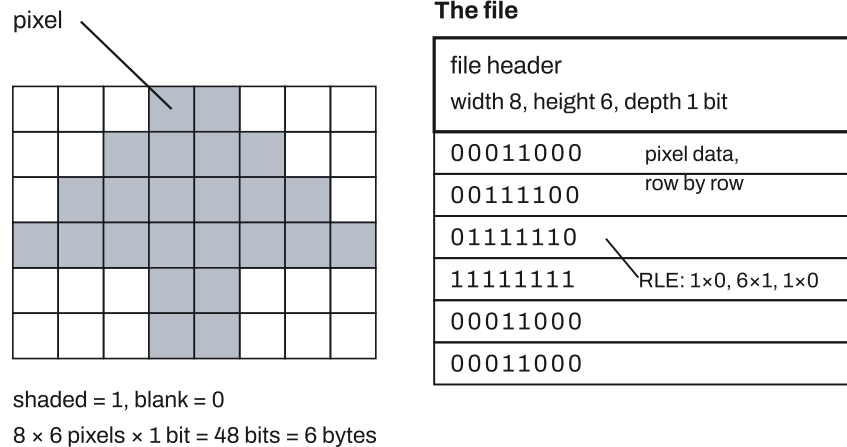
A **character set** is the list of characters a computer can represent, with a **unique binary code** for each one. Text is stored by replacing each character with its code, in order. "Cab" in ASCII is 01000011 01100001 01100010 (67, 97, 98).

Character set	Bits per character	Characters
ASCII	7	$2^7 = 128$: English letters, digits, punctuation, control codes
Extended ASCII	8	$2^8 = 256$: ASCII plus accented and other symbols
Unicode	up to 32 (16 is the usual answer)	over a million: every language, maths symbols, emojis

The first 128 Unicode codes are the same as ASCII. Codes for letters and digits run **in order**, so if 'A' is 65 then 'E' is 69, and 'a' is 32 more than 'A'. Unicode's advantage is range (every writing system and emojis); its drawback is that each character can take more bits, so text files are larger.

Bitmap images

A **bitmap** image is a grid of **pixels** (picture elements), the smallest part of the image, each one a single colour. Each colour has a binary code, and the file stores the code for every pixel in order, row by row.



- **Colour depth** (bit depth): the number of bits used for each pixel's colour. n bits give 2^n colours: 1 bit gives 2, 8 bits give 256, 24 bits give over 16 million.
- **Image resolution**: the number of pixels in the image, width $\times$ height (for example 1600×1200).
- **Screen resolution**: the number of pixels a monitor can show, width $\times$ height (also given as dots per inch when an image is displayed). Changing it changes how the image looks on that screen, **not** the image file.
- **File header**: data at the start of the file about the image (metadata), such as the file type, width, height, colour depth, file size and any compression used. The software needs it to rebuild the picture from the stream of bits.

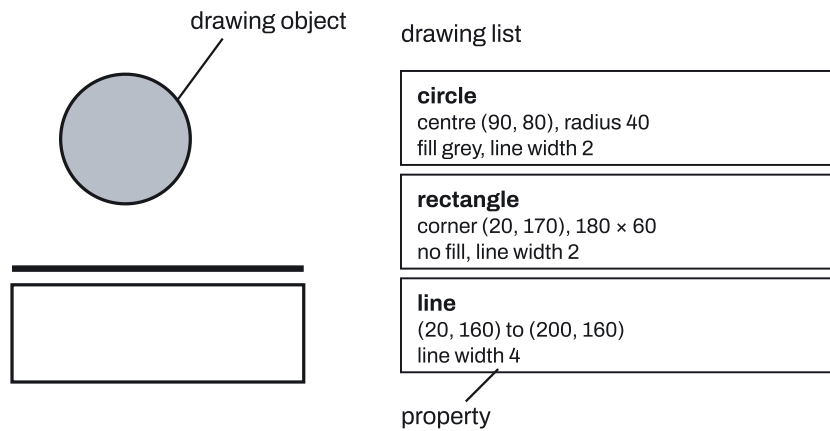
File size = width $\times$ height $\times$ colour depth bits. A 1600×1200 image with 24-bit colour is $1600 \times 1200 \times 24 \div 8 = 5\,760\,000$ bytes = 5.76 MB. This is an **estimate**: the real file is a little bigger because of the header.

Changing the elements:

- **More pixels** (higher resolution): more detail, a sharper image when enlarged, a bigger file. Fewer pixels: detail is lost, the image looks blocky (pixelated) when enlarged, a smaller file.
- **More bits per pixel** (higher colour depth): more colours, so shading is smoother and closer to the real scene, a bigger file. Fewer bits: fewer colours, banding and loss of detail, a smaller file. Halving the colour depth halves the pixel data.
- Changing the colour of one shape leaves the size unchanged: every pixel still uses the same number of bits.

Vector graphics

A **vector graphic** is stored as a description of shapes, not pixels.



- **Drawing object:** one shape in the picture (line, circle, rectangle, curve), defined by a formula or command rather than by pixels.
- **Property:** one piece of data about a drawing object that fixes its appearance, such as its position, size (radius, width), line colour, line width or fill colour.
- **Drawing list:** the list of all the drawing objects in the picture, with the commands and properties needed to draw each one.

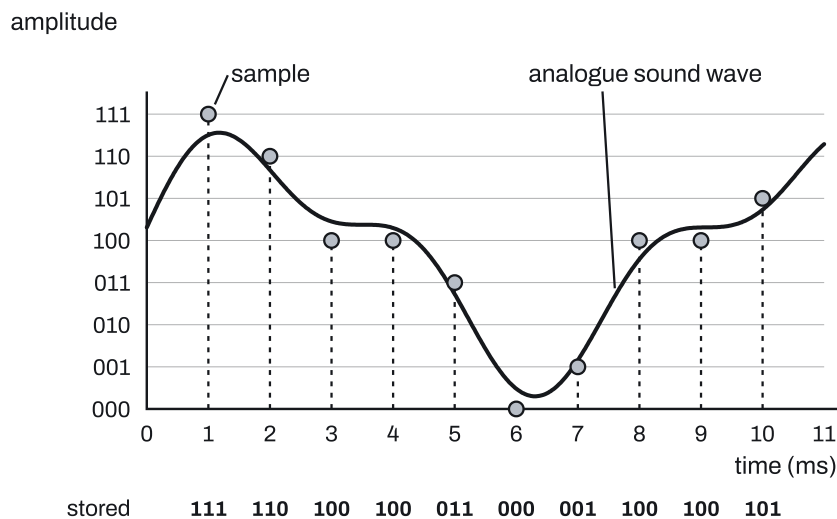
To display it, the computer works through the drawing list and draws each object at the size needed. So a vector graphic can be **scaled to any size without pixelating**, and its file is usually small, since it stores a few numbers per shape and does not depend on the size at which it is shown.

Which to choose:

- **Bitmap** for photographs and scanned images, and anything with detailed, varied colour: every pixel can differ, which shapes cannot describe.
- **Vector** for logos, diagrams, maps, fonts and plans made of shapes: they must look sharp at any size (a logo on a business card and on a poster), they can be edited object by object, and the file stays small.

Sound

Sound in the air is **analogue**: the amplitude changes smoothly and continuously, with infinitely many possible values. A computer stores **digital** data, a limited set of separate values. An analogue-to-digital converter changes one into the other by **sampling**.



- **Sampling:** measuring the amplitude of the sound wave at set, regular time intervals.
- **Sampling rate:** the number of samples taken per second, in hertz (Hz). CD audio uses 44 100 Hz (44.1 kHz).
- **Sampling resolution:** the number of bits used to store each sample. With n bits there are 2^n amplitude levels, and each sample is rounded to the nearest level.

Each sample's binary value is stored in sequence. **File size** = sampling rate $\times$ sampling resolution $\times$ number of seconds bits. For 10 s at 48 kHz with 16 bits: $48\,000 \times 16 \times 10 \div 8 = 960\,000$ bytes = 0.96 MB.

- **Higher sampling rate:** samples are closer together in time, so the recording follows the wave more closely (more accurate), and there are more samples, so a bigger file.
- **Higher sampling resolution:** more levels, so each sample is rounded less and the amplitude is more precise (more accurate, a larger dynamic range), and more bits per sample, so a bigger file.
- **Longer recording:** a bigger file, but **no change** in accuracy.

Compression

Compression makes a file smaller so it takes less storage space, is quicker to upload, download or email, uses less bandwidth and data allowance, and streams with less buffering.

- **Lossless** compression reduces the size so that the original file can be rebuilt **exactly**. It is needed for text, program code, spreadsheets and databases, where changing a single character would change the meaning or stop a program working.
- **Lossy** compression permanently removes data, usually data that people will not notice. The original cannot be rebuilt, but the file becomes much smaller than lossless could make it. It suits photos, music and video, especially for streaming and web pages.

Run-length encoding (RLE) is lossless. A run of identical, consecutive values is stored once, with a count of how many times it repeats.

- **Text:** AAAABBBCCD (10 characters) becomes 4A3B2C1D (8 characters).

- **Bitmap:** a row of pixels with colour codes 3C 3C 3C 3C 3C 7F 7F 00 (8 bytes) becomes the pairs 05 3C, 02 7F, 01 00 (6 bytes): count, then colour.
- RLE only helps when there are **long runs**. A photograph where neighbouring pixels are all slightly different has almost no runs, so each pixel becomes a count **and** a colour, and the file can get **bigger** (four different pixels: 4 bytes become 8).

How each kind of file can be compressed:

File	Lossless	Lossy
Text	RLE; replace repeated words or phrases with a short code and a table (dictionary) of what the codes mean	never used
Bitmap image	RLE on runs of same-coloured pixels	reduce the resolution or colour depth; remove fine detail and colour changes the eye does not notice
Vector graphic	compress the drawing list like text (repeated commands and properties stored once)	simplify shapes: fewer points on a curve, fewer decimal places in coordinates
Sound	RLE on repeated sample values (such as silence)	reduce the sampling rate or resolution; remove frequencies outside human hearing, and quiet sounds hidden by louder ones at the same time

Key facts and formulas

There is no formula list for 9618, so every entry here is "Learn it".

Formula	Status
n bits give 2^n different values; the least bits for N values is the smallest n with $2^n \geq N$	Learn it
1 KiB = 2^{10} = 1024 bytes, MiB = 2^{20} , GiB = 2^{30} , TiB = 2^{40} bytes	Learn it
1 kB = 10^3 bytes, MB = 10^6 , GB = 10^9 , TB = 10^{12} bytes	Learn it
1 byte = 8 bits; 1 nibble = 4 bits = 1 hex digit	Learn it
Hex place values: 4096, 256, 16, 1; A–F = 10–15	Learn it
Two's complement: MSB worth -2^{n-1} ; negate by flipping all bits and adding 1	Learn it

Formula	Status
n -bit ranges: unsigned 0 to $2^n - 1$; two's complement -2^{n-1} to $2^{n-1} - 1$	Learn it
BCD: each denary digit as 4 bits; valid nibbles 0000–1001	Learn it
Bitmap size (bits) = width $\times$ height $\times$ colour depth	Learn it
Sound size (bits) = sampling rate $\times$ sampling resolution $\times$ time in seconds	Learn it
Bits $\div 8$ = bytes; then $\div 1000$ per step for kB, MB, GB or $\div 1024$ per step for KiB, MiB, GiB	Learn it
ASCII 7 bits (128 characters); extended ASCII 8 bits (256); Unicode up to 32 bits (16 in most answers)	Learn it

How to do it

In the Paper 1 questions we have tagged for this topic (55 questions, 2021–2025, of which 3 are repeats on another paper variant), the types came up this often. 45 of the questions mix several types, so the counts add up to more than 55.

Question type	Questions
Convert between binary, denary and hexadecimal (including character codes)	26
Add or subtract in binary and explain overflow	17
Describe a compression method (RLE, lossy methods)	15
Define bitmap terms and explain the effects of resolution and colour depth	15
Convert to or from two's complement (and one's complement) and state its range	14
Justify compressing a file and choose lossy or lossless	14
Calculate a file size (bitmap, video, sound)	12
Explain sound sampling and the effect of sampling rate and resolution	12
Convert to or from BCD and justify where BCD is used	11
Describe character sets (ASCII, extended ASCII, Unicode)	9
Use binary and decimal prefixes	8
Define vector graphic terms and compare vector with bitmap	6
State the bits needed, or the number of values from n bits	6

1. Convert between binary, denary and hexadecimal (26 questions)

1. **Binary to denary:** write the place values over the bits ($\dots 256, 128, 64, \dots, 1$) and add the ones with a 1 under them.
2. **Denary to binary:** subtract the largest place value that fits, write 1 there, and repeat; fill the other places with 0. Give exactly the number of bits asked for (pad with 0s on the left).
3. **Binary to hex:** split into nibbles from the **right**, then convert each nibble ($1100 = C$). **Hex to binary:** each hex digit becomes 4 bits.
4. **Hex to denary:** multiply each digit by its place value: $B2D = 11 \times 256 + 2 \times 16 + 13 = 2861$. Or go through binary. **Denary to hex:** convert to binary first, then group.

Variations you will meet:

- **"Show your working":** write the place values you used, or the nibbles with their values.
- **Character-code tables.** You are given a character's code in one base and fill in the others (denary, 8-bit binary, hex). Treat each as a normal conversion. If you are told one code (for example 'P' is 80), count on for the letters after it: 'S' is 83.
- **Decoding bytes with a table.** Convert each received byte to denary, then look it up in the table given.
- **One register, several readings (9618/12/O/N/21 Q4):** the same 8 bits read as unsigned, as two's complement, as hex and as BCD give different answers. Work each one separately.

2. Add or subtract in binary and explain overflow (17 questions)

1. Write the numbers one above the other, bits lined up.
2. **Add** from the right; show every carry above or below the next column. With three numbers, a column can total 2 or 3 (write the right bit, carry 1), or more (carry to two columns); adding the first two, then the third, is safer.
3. **Subtract** either by borrowing, or by turning the number taken away into two's complement and adding (ignore the final carry out).
4. If one number is given in denary, convert it to binary first, with the same number of bits.
5. **Check** by converting the answer to denary.

Overflow (asked for as 1 or 2 marks): the result is too large (or too small) to be represented in the number of bits available in the register, so a bit is lost and the stored answer is wrong. Say which: bigger than the largest positive number, or smaller than the most negative number. If asked to "include any overflow bit", write the carry out in brackets in front: (1) 01101100.

3. Describe a compression method (15 questions)

1. **Name the method**, then **describe how it works**, for the file type in the question (text, bitmap, sound).
"Use JPEG" or "remove colours the eye can't see" on its own is not a method.
2. **RLE:** find runs of identical consecutive characters or pixels; store each run once with a count of how many times it repeats.

3. **Lossy image:** reduce the resolution (fewer pixels) or the colour depth (fewer bits per pixel); remove detail the eye does not notice.
4. **Lossy sound:** reduce the sampling rate (fewer samples each second) or the resolution (fewer bits per sample); remove sounds outside the range of human hearing.

Variations you will meet:

- **Complete an RLE table** of hex colour codes. Count each run and write count then colour; to decompress, write the colour count times.
- **Why RLE may not work** for a photo: few runs of identical pixels, so storing a count with every pixel can make the file larger. Give a short example.

4. Define bitmap terms and explain the effects of resolution and colour depth (15 questions)

1. Learn the five definitions (pixel, file header, image resolution, screen resolution, colour depth) from "Understand it" in your own words.
2. For **"explain the effect"**, give the effect **and** the reason: "fewer bits per pixel, so fewer colours can be shown, so detail and shading are lost" and "fewer bits per pixel, so less data, so a smaller file".
3. **How a bitmap is encoded:** made of pixels; each pixel is one colour; each colour has a unique binary code; the codes are stored in sequence.

Variations you will meet:

- **Tick the effect on file size:** lower colour depth decreases it; changing the **screen** resolution or the colour of a shape makes no change.
- **Items in the file header:** file type, width, height, colour depth, file size, compression method.
- **Why the real file is bigger than your estimate:** the header (metadata).
- **Two things to change to reduce file size:** colour depth and image resolution.

5. Convert to or from two's complement and state its range (14 questions)

1. **Negative denary to two's complement:** convert the size to binary with the right number of bits, flip every bit, add 1. Check by adding the place values with the negative MSB.
2. **Two's complement to denary:** if the MSB is 0, convert normally. If it is 1, use -128 (8 bits) or -2048 (12 bits) for the MSB and add the rest.
3. **Explain the method** (for 3 marks): name each step ("flip every bit, then add 1, then convert to denary and make it negative") and give the value.

Variations you will meet:

- **Range:** smallest 8-bit two's complement $10000000 = -128$; largest $01111111 = 127$; largest unsigned 8-bit $11111111 = 255$.
- **One's complement:** flip every bit of the positive number, and stop there (no adding 1).

- **Check a 12-bit answer:** $111101101010 = -2048 + 1024 + 512 + 256 + 64 + 32 + 8 + 2 = -150$.

6. Justify compressing a file and choose lossy or lossless (14 questions)

1. **Reasons to compress:** smaller file, so less storage space, quicker to send or download, less bandwidth, less of a data allowance, less buffering when streaming, fits email attachment limits. Each point usually needs its consequence for the person in the question ("... so the customer waits less for the photo to load").
2. **Choose:** there is normally **no mark for the choice**, only for the justification.
 - **Lossy** for photos, video and music, especially live streaming: a much bigger size reduction, and the lost data is barely noticed by the eye or ear.
 - **Lossless** for text and program files: every character must come back exactly.
3. Tie the justification to the situation: "the video is streamed live, so it must be sent quickly with little bandwidth".

7. Calculate a file size (12 questions)

1. Find the **bits per item**: the colour depth or sampling resolution. If you are given a number of colours, turn it into bits (256 colours $\rightarrow$ 8 bits); if you are given bytes, multiply by 8 (or keep bytes and skip the $\div 8$).
2. **Multiply:** width $\times$ height $\times$ bits (bitmap); $\times$ frames per second too for one second of video; sampling rate $\times$ resolution $\times$ seconds (sound, with minutes turned into seconds and kHz into Hz).
3. **Divide by 8** for bytes, then by 1000 per step for kB, MB, GB, or 1024 per step for KiB, MiB, GiB. The unit in the question decides.
4. Write the working as one calculation, for example 10 minutes of sound at 22.05 kHz with 8 bits:

$$\frac{22\,050 \times 8 \times 600}{8 \times 1000 \times 1000} = 13.23 \text{ MB.}$$

8. Explain sound sampling and the effect of sampling rate and resolution (12 questions)

1. **How sound is digitised:** the amplitude of the analogue wave is measured at set time intervals (a set number of times per second); each measurement is given a binary value from the available levels and the values are stored in sequence.
2. **Effects:** for each change, say what happens to **accuracy** and to **file size**, and why. Higher rate or resolution: more accurate and bigger. Longer recording: bigger, accuracy unchanged.
3. **Terms:** be precise. "Sampling rate is the number of samples taken per second", not "the rate of sampling".

9. Convert to or from BCD and justify where BCD is used (11 questions)

1. **Denary to BCD:** each digit becomes 4 bits: $572 \rightarrow 0101\ 0111\ 0010$. Keep leading zeros in each nibble.
2. **BCD to denary:** split into nibbles from the left, convert each to one digit. If you are given ordinary binary to write as BCD, convert to denary first.

3. **Application + justification** (2 marks): name a use and give the matching reason, such as "banking, because decimal amounts are stored exactly with no rounding errors" or "a digital clock, because each digit is displayed separately".
4. **"Why can't this be BCD?"**: a nibble is greater than 1001 (it would be a digit over 9).

10. Describe character sets (9 questions)

1. **How text is represented**: each character has a unique binary code in the character set; the text is stored as the codes of its characters, in order.
2. **Bits**: ASCII 7, extended ASCII 8, Unicode 16 (up to 32).
3. **Differences**: Unicode can represent far more characters (all languages, symbols, emojis); ASCII uses fewer bits per character, so its files are smaller.

11. Use binary and decimal prefixes (8 questions)

1. **State a difference**, with numbers: "a kibibyte is 1024 bytes, a kilobyte is 1000 bytes" or "tebi is a binary prefix (powers of 2), tera is a decimal prefix (powers of 10)". "A kibibyte is bigger" is not enough.
2. **Fill a blank** or **match equal values** by converting both to bytes.
3. **Pick the largest** of mixed units: convert all to bytes. $2500 \text{ KiB} = 2\,560\,000 \text{ bytes}$ is bigger than $2500 \text{ kB} = 2\,500\,000$, which beats $2 \text{ MiB} = 2\,097\,152$.

12. Define vector graphic terms and compare vector with bitmap (6 questions)

1. Learn drawing object, property and drawing list, each with an example from a simple picture ("the circle is a drawing object; its fill colour is a property").
2. **Benefits of vector**: scales without losing quality or pixelating, usually a smaller file, each object can be edited separately.
3. **Justify a choice** for a given image: photo $\rightarrow$ bitmap (continuous detail, every pixel different); logo or diagram $\rightarrow$ vector.

13. State the bits needed, or the number of values from n bits (6 questions)

1. **Values from n bits**: 2^n . 16 bits give 65 536.
2. **Bits needed for N values**: the smallest n with $2^n \geq N$. 26 letters need 5 bits ($2^5 = 32$); 300 values need 9 bits; 5 000 000 levels need 23 bits ($2^{23} = 8\,388\,608$, while $2^{22} = 4\,194\,304$ is too few).
3. **Bits for a stored value**: each hex digit is 4 bits (7A2 needs 12); a denary value needs enough bits for its size (40 000 needs 16, since $2^{15} = 32\,768$ is too small). Try 9618/13/O/N/25 Q2.

Worked examples

The bold codes show where marks are earned, as in Cambridge mark schemes. The number is how many marks.

- **M1** method mark: for a correct method, even if the numbers slip.
 - **A1** accuracy mark: for a correct answer; it needs the M mark before it.
 - **B1** mark for a correct result on its own, with no method needed.
-

Example 1

A register holds the 8-bit value 10110110.

- (a) Give the denary value if it is an unsigned integer. **[1]**
- (b) Give the denary value if it is a two's complement integer. **[1]**
- (c) Give the value in hexadecimal. **[1]**
- (d) State why the register's contents cannot be a BCD value. **[1]**
- (e) Convert 381 into BCD. **[1]**
- (f) Convert -90 into an 8-bit two's complement integer. **[1]**
- (a)** $128 + 32 + 16 + 4 + 2 = 182$. **B1**
- (b)** $-128 + 32 + 16 + 4 + 2 = -74$. (Check: flip to 01001001, add 1 to get 01001010 = 74, so the value is -74 .) **B1**
- (c)** $1011 = 11 = B$ and $0110 = 6$, so B6. **B1**
- (d)** The first nibble, 1011, is 11, which is not a denary digit. **B1**
- (e)** $3 \rightarrow 0011$, $8 \rightarrow 1000$, $1 \rightarrow 0001$: 0011 1000 0001. **B1**
- (f)** $90 = 01011010$; flip to 10100101; add 1: 10100110. **B1**

Example 2

- (a) The two's complement integers 01101101 and 01010110 are added in an 8-bit register. Complete the addition and explain the problem with the result. **[3]**
- (b) Subtract the denary number 38 from the two's complement integer 01011001 using binary. Show your working. **[2]**
- (a)**

$$\begin{array}{r} 01101101 \\ + 01010110 \\ \hline 11000011 \end{array}$$

with carries into the columns worth 8, 16, 32, 64 and 128. **B1** for 11000011.

The two numbers are 109 and 86, both positive, so the true answer is 195. That is more than 127, the largest 8-bit two's complement value, so it cannot be represented: **overflow**. **B1** The result's MSB is 1, so it reads as −61, which is wrong. **B1**

(b) $38 = 00100110$, so $-38 = 11011010$. **M1** for converting and using the two's complement (or for correct borrowing).

$$01011001 + 11011010 = (1) 00110011$$

Ignore the carry out: the answer is 00110011 ($89 - 38 = 51$). **A1**

Example 3

(a) A photograph is 3000 pixels wide and 2000 pixels high. It can show 65 536 different colours. Calculate an estimate of its file size in megabytes. Show your working. **[2]**

(b) Give this file size in mebibytes, to 2 decimal places. **[1]**

(c) A 3-minute song is recorded with a sampling rate of 44.1 kHz and a sampling resolution of 16 bits. Calculate its file size in megabytes, to 1 decimal place. Show your working. **[2]**

(d) Explain why the actual size of the photograph's file is larger than your answer to (a). **[1]**

(a) $65\,536 = 2^{16}$, so the colour depth is 16 bits (2 bytes).

$$\frac{3000 \times 2000 \times 16}{8 \times 1000 \times 1000} = 12 \text{ MB}$$

M1 for the working, **A1** for 12.

(b) $12\,000\,000 \div (1024 \times 1024) = 11.44 \text{ MiB}$. **B1**

(c) 3 minutes = 180 s and 44.1 kHz = 44 100 samples per second.

$$\frac{44\,100 \times 16 \times 180}{8 \times 1000 \times 1000} = 15.876 \approx 15.9 \text{ MB}$$

M1 A1

(d) The file also has a header (metadata such as the width, height and colour depth), which the estimate leaves out. **B1**

Example 4

A web designer stores a photograph of a beach and the HTML code for the page. Both are compressed before upload.

(a) Identify the type of compression most suitable for each file. Justify each choice. **[4]**

(b) One row of a simple banner image uses 8-bit colour. Its pixels, in hexadecimal, are

1F 1F 1F 1F 1F 1F 2A 2A 1F 1F 1F 1F

Show this row compressed with run-length encoding, and state the number of bytes saved. **[3]**

(a) Photograph: **lossy**. It reduces the file far more than lossless, so the page loads faster, and removing small colour changes is not noticed by the eye. **B1 B1**

HTML code: **lossless**. Every character must be restored exactly, or the page will not display properly. **B1 B1**

(No mark for the choice alone: the marks are for the justifications.)

(b) The runs are six 1F, two 2A and four 1F. **M1** for finding the runs.

06 1F 02 2A 04 1F **A1**

12 bytes become 6 bytes, so 6 bytes are saved (the row is halved). **A1**

Common mistakes

- **Mixing up the prefixes.** A kibibyte is 1024 **bytes**, not 1024 bits, and a kilobyte is 1000 bytes. Examiners see answers like "a kilobyte is 1000 bits", or kibi and kilo swapped round. "A kibibyte is bigger than a kilobyte" is too vague: give the numbers.
- **Forgetting to divide by 8.** Width × height × colour depth is in **bits**. Divide by 8 for bytes before converting to MB, and use 1000 or 1024 to match the unit asked for.
- **Reading BCD as ordinary binary.** 0110 0011 in BCD is 63 (6 then 3), not 99. And the reverse: converting an unsigned binary question to BCD, or giving denary when hex was asked for. Read which representation the question gives and which it wants.
- **Two's complement slips.** Flipping the bits but forgetting to add 1, or adding 1 for one's complement where it doesn't belong. Using +128 for the MSB. Always check by adding the place values.
- **Lost carries.** Show every carry; with three numbers a column can carry more than once. An answer with no working often loses the working mark.
- **Confusing the drawing list with a drawing object** (or with a library of available shapes). The drawing list is the list of every object in **this** picture with the commands to draw it.
- **Describing effects instead of methods.** "Use JPEG" or "the image looks worse" is not a lossy method. Name what is reduced (resolution, colour depth, sampling rate) and say how.
- **RLE described vaguely.** Say both parts: the value is stored **once** and the **number of times it repeats** is stored with it. For a question about text, talk about characters; for an image, pixels.
- **Mixing up the sound terms.** Sampling rate is samples per second; sampling resolution is bits per sample. Doubling the length of a recording changes the file size, not the accuracy.
- **Unsupported choices.** "Lossy" or "vector" earns nothing on its own; the marks are in the reason.

How to get the marks

- **One mark per point.** A 2-mark "explain" needs two separate, precise points; a point and its consequence often count as two ("fewer pixels" + "so less data is stored").
- **Show the working** for conversions and calculations when asked: place values, nibbles, carries, or the whole file-size expression. A correct method can still earn the working mark after a slip.
- **Units:** give the unit asked for (MB, KiB, gigabytes). A file size in bits when megabytes were asked for loses the answer mark.
- **Bit counts:** give exactly the number of bits asked for ("12-bit": 12 digits, leading zeros included). Grouping into nibbles helps you and the examiner.
- **"State" or "Identify"** needs a short answer; **"Describe"** needs the steps or features; **"Explain"** needs reasons (use "because" or "so"); **"Justify"** needs a reason linked to the situation in the question.
- **"Give two differences":** make each a comparison of both sides ("ASCII uses 7 or 8 bits per character, whereas Unicode uses up to 32").
- **Tables and matching:** one line or tick per row unless told otherwise; extra lines from one box cost the mark.
- **Justify a choice:** the choice itself is usually worth nothing; the justification must fit the file type and situation.

Quick check

1. Convert the 12-bit unsigned binary integer 110101111010 into hexadecimal and into denary.
2. Convert -300 into a 12-bit two's complement binary integer.
3. (a) How many bytes are in 5 KiB? (b) How many gigabytes are in 3 TB? (c) What is the least number of bits that gives 1000 different codes?
4. A voice message lasts 30 seconds, sampled at 8 kHz with a sampling resolution of 8 bits. Calculate its file size in kilobytes. State what happens to the file size and the accuracy if the sampling rate is doubled.
5. Compress the text WWWWWBWWWW with run-length encoding, and say whether RLE is lossy or lossless.

Answers

1. Nibbles 1101 0111 1010 $\rightarrow$ D7A. Denary: $13 \times 256 + 7 \times 16 + 10 = 3328 + 112 + 10 = 3450$.
2. $300 = 0001\ 0010\ 1100$. Flip: 1110 1101 0011. Add 1: 1110 1101 0100. Check: $-2048 + 1024 + 512 + 128 + 64 + 16 + 4 = -300$.
3. (a) $5 \times 1024 = 5120$ bytes. (b) $3 \times 1000 = 3000$ GB. (c) 10 bits: $2^9 = 512$ is too few and $2^{10} = 1024 \geq 1000$.
4. $\frac{8000 \times 8 \times 30}{8 \times 1000} = 240$ kB. Doubling the sampling rate doubles the number of samples, so the file doubles to 480 kB, and the recording follows the wave more closely, so it is more accurate.
5. 6W3B4W (13 characters become 6). RLE is lossless: the text can be rebuilt exactly.

Past questions to try

- **9618/12/O/N/21 Q4**: one register read as unsigned, two's complement, hex and BCD, then addition and subtraction.
- **9618/13/O/N/22 Q1**: effects on a sound recording's accuracy, and a sound file size.
- **9618/12/O/N/22 Q8**: effects on a bitmap's file size, vector benefits, and an RLE table in hex.
- **9618/11/M/J/23 Q1**: bitmap and vector terms, the file header, a file size and when RLE fails.