

Data representation

IGCSE Computer Science 0478 · Topic 1 revision notes

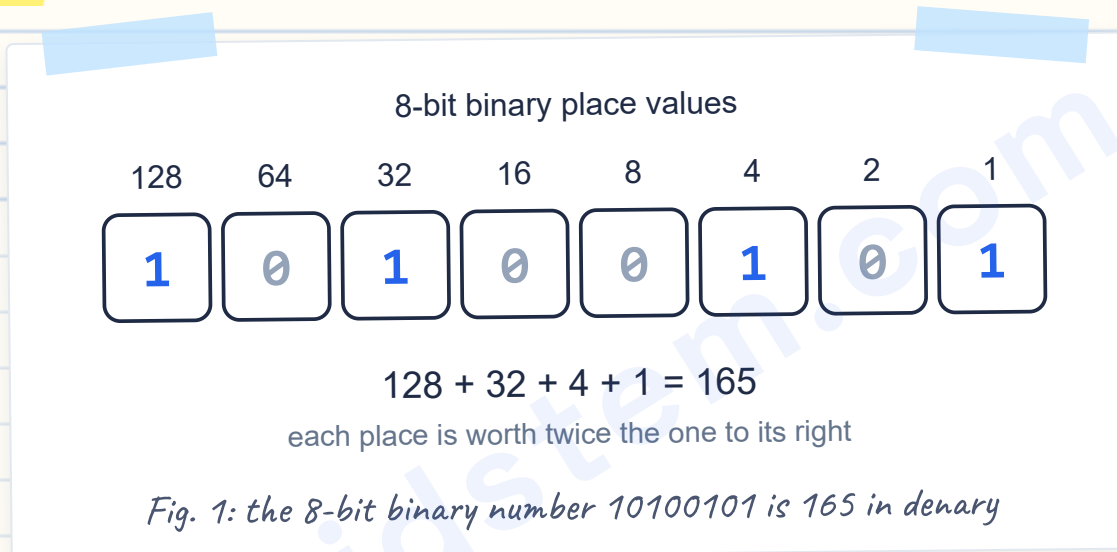
Everything from the 10 lessons of this topic on a few pages: the key idea of each lesson, the diagram to remember, the points to learn and the traps examiners set. Read it, cover it, test yourself.

- | | |
|--|---|
| 1. <u>Why computers use binary: binary and denary.</u> | 7. <u>Representing sound: sample rate and resolution</u> |
| 2. <u>Hexadecimal and where it is used</u> | 8. <u>Representing images: resolution and colour depth</u> |
| 3. <u>Binary addition and overflow</u> | 9. <u>Measuring data storage and calculating file sizes</u> |
| 4. <u>Logical binary shifts</u> | 10. <u>Lossy and lossless compression</u> |
| 5. <u>Two's complement for negative numbers</u> | |
| 6. <u>Representing text: ASCII and Unicode</u> | |

1 Why computers use binary: binary and denary

Syllabus 1.1

Computers process data using **logic gates** and store it in **registers**, both made of switches that are either off or on. So all data is represented in **binary** (base 2), which uses only the digits 0 and 1. People use **denary** (base 10). In binary each place value is **twice** the one to its right: 1, 2, 4, 8, 16, 32, 64, 128 ... A single binary digit is a **bit**; 8 bits make a **byte**.



Worked example: 8 bits

Convert the binary number 01011010 to denary.

1. Write the place values 128, 64, 32, 16, 8, 4, 2, 1 above the bits.
2. The 1s are under 64, 16, 8 and 2.
3. Add them: $64 + 16 + 8 + 2 = 90$.

Answer: 90

Place values first.

- ✓ **Place values:** 128, 64, 32, 16, 8, 4, 2, 1 for 8 bits; keep doubling for 16 bits.
- ✓ **Binary to denary:** add the place values of the 1s.
- ✓ **Denary to binary:** subtract the largest place values that fit, or divide by 2 and read the remainders upwards.
- ✓ **Check:** convert your answer back the other way.

Watch out!

"Computers use binary because it is easier for them" earns nothing. Say why: the computer is made of switches (logic gates and registers) with only two states, on and off.

Leaving out leading zeros changes nothing in value, but if the question says "8-bit" you must give all 8 bits: 00111010, not 111010.

thelucidstem.com

2 Hexadecimal and where it is used

Syllabus 1.1

Hexadecimal is base 16. It uses the digits 0 to 9 and the letters **A to F** for 10 to 15. One hex digit stands for exactly **4 bits** (a nibble), so binary converts to hex by splitting it into groups of 4 from the right. Hex is used because it is **shorter and easier for humans** to read and write than binary, with fewer mistakes, yet converts to binary simply. The computer itself still stores binary.

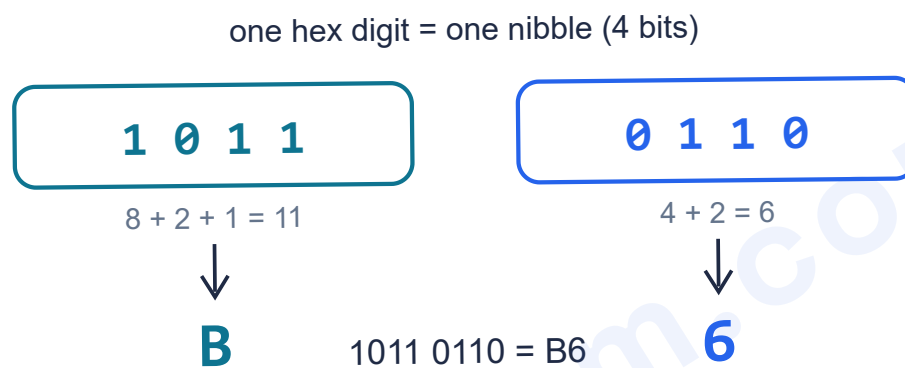


Fig. 1: each group of four bits becomes one hexadecimal digit

Worked example: Binary to hex

Convert 1101 0111 1010 to hexadecimal.

1. Split into nibbles from the right: 1101, 0111, 1010.
2. 1101 = 13 = D; 0111 = 7; 1010 = 10 = A.

Answer: D7A

Group in fours.

- ✓ **Digits:** 0 to 9, then A = 10 up to F = 15.
- ✓ **Binary to hex:** split into groups of 4 from the right; convert each group.
- ✓ **Hex to denary:** multiply each digit by 16 or 1 (or 256, 4096 for longer numbers) and add.
- ✓ **Why hex:** shorter, easier for humans to read and write, fewer errors; the computer still uses binary.

Watch out!

Do not write 12 as the hex digit for twelve. Each place holds a single character, so twelve is C; 12 in hex means $1 \times 16 + 2 = 18$.

Say that hex is easier for humans. Computers do not store or process hex: they convert it to binary.

thelucidstem.com

3 Binary addition and overflow

Syllabus 1.1

Add binary numbers in columns from the **right**, using four rules: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 1 = 0$ **carry 1**, and $1 + 1 + 1 = 1$ carry 1. Every register has a **predefined limit**: an 8-bit register can hold at most 255. If the true result is bigger than the limit, a carry is left over from the leftmost column and an **overflow error** occurs: the stored answer is wrong.

adding two 8-bit numbers, right to left

carries	1	1	1	1	1	1		
	0	1	0	1	1	0	1	1 91
+	0	0	1	1	0	1	1	0 54
	1	0	0	1	0	0	0	1 145

$1 + 1 = 0$ carry 1 $1 + 1 + 1 = 1$ carry 1

a carry out of the leftmost column means overflow

Fig. 1: $01011011 + 00110110 = 10010001$ ($91 + 54 = 145$), with the carries shown in red

Worked example: No overflow

Add 00101101 and 00011110 . Give the answer in 8-bit binary.

- Columns 1 and 2 (from the right): $1 + 0 = 1$; $0 + 1 = 1$.
- Column 3: $1 + 1 = 0$ carry 1. Column 4: $1 + 1 + 1 = 1$ carry 1.
- Column 5: $0 + 1 + 1 = 0$ carry 1. Column 6: $1 + 0 + 1 = 0$ carry 1. Column 7: $0 + 0 + 1 = 1$. Column 8: $0 + 0 = 0$.
- Check: $45 + 30 = 75$ and $01001011 = 64 + 8 + 2 + 1 = 75$.

Answer: 01001011

Right to left.

- ✓ **Rules:** $1 + 1 = 0$ carry 1; $1 + 1 + 1 = 1$ carry 1.
- ✓ **Show carries:** write each carry above the next column on the left.
- ✓ **Overflow:** the result is bigger than the register can hold (more than 255 for 8 bits).
- ✓ **Check:** convert the two numbers and the answer to denary.

Watch out!

Show your carries. If you only write the answer and it is wrong, you lose every mark; with the carries shown, you can still earn method marks.

Overflow is not "running out of memory". It is a single result that needs more bits than the register has.

thelucidstem.com

4 Logical binary shifts

Syllabus 1.1

A **logical shift** moves every bit in a register left or right by a number of places. Bits shifted off the end of the register are **lost** and **zeros** are shifted in at the opposite end. A **left shift** of n places **multiplies** a positive integer by 2^n ; a **right shift** of n places **divides** it by 2^n . If a 1 is lost, the answer is no longer correct: the most significant bits are lost in a left shift, the least significant bits in a right shift.

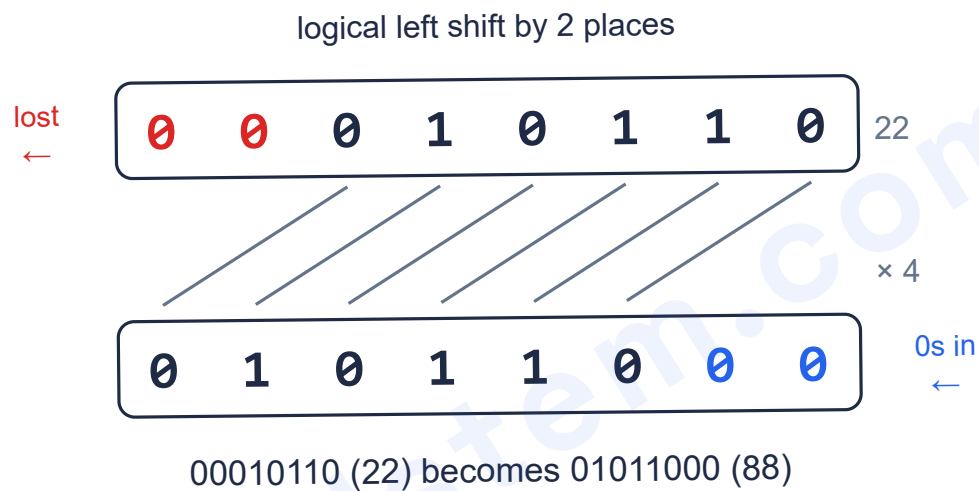


Fig. 1: a left shift of 2 places multiplies 22 by 4; two 0s are lost on the left and two 0s enter on the right

Worked example: Left shift 3 places

Perform a logical left shift of 3 places on 00001101. Give the result and its effect.

1. Move every bit 3 places left: the three leading 0s are lost.
2. Fill the 3 empty places on the right with 0: 01101000.
3. $00001101 = 13$ and $01101000 = 64 + 32 + 8 = 104 = 13 \times 8$.

Answer: 01101000: the number has been multiplied by 8

Lose, then fill.

- ✓ **Move:** shift every bit the given number of places.
- ✓ **Lose:** bits pushed past the end of the register are lost.
- ✓ **Fill:** 0s enter at the opposite end; keep exactly 8 bits.
- ✓ **Effect:** left n places $\times 2^n$, right n places $\div 2^n$, correct only if no 1s are lost.

Watch out!

Always give 8 bits after a shift. Writing 1101000 instead of 01101000 shows you have not kept the register size.

"Shift left 2" means multiply by 4, not by 2. The factor is 2 to the power of the number of places.

thelucidstem.com

5 Two's complement for negative numbers

Syllabus 1.1

In 8-bit **two's complement** the most significant bit has the place value **-128**; the other bits are 64, 32, 16, 8, 4, 2, 1 as usual. So a number starting with **1 is negative** and one starting with **0 is positive**. The range is **-128 to +127**. To make a negative number, write the positive version in 8 bits, **flip every bit**, then **add 1**.

8-bit two's complement place values

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

$$-128 + 64 + 32 + 8 + 4 = -20$$

most significant bit 1 = negative, 0 = positive

range -128 to +127

Fig. 1: in two's complement the leftmost bit is worth -128, so 11101100 is -20

Worked example: Two's complement to denary

Convert the 8-bit two's complement integer 10110110 to denary.

1. The leftmost bit is 1, so start with -128.
2. The other 1s are under 32, 16, 4 and 2: $32 + 16 + 4 + 2 = 54$.
3. $-128 + 54 = -74$.

Answer: -74

Minus 128 first.

- ✓ **Place values:** -128, 64, 32, 16, 8, 4, 2, 1.
- ✓ **Sign:** leftmost bit 1 means negative, 0 means positive.
- ✓ **Make a negative:** write the positive in 8 bits, flip every bit, add 1.
- ✓ **Check:** add up the place values, starting from -128.

Watch out!

Do not read the leftmost 1 as just a minus sign. 10000101 is $-128 + 5 = -123$, not -5 .

+128 cannot be stored in 8-bit two's complement. The largest positive value is +127.

thelucidstem.com

6 Representing text: ASCII and Unicode

Syllabus 1.2

Text must be **converted to binary** before a computer can process it. A **character set** gives every character a unique binary code. **ASCII** uses 7 bits (128 characters), and extended ASCII 8 bits (256 characters): enough for English letters, digits and symbols.

Unicode allows a far **greater range** of characters and symbols, including different languages and emojis, but it **needs more bits per character** (up to 32), so the same text takes more storage.

each character has a code in the character set

character	ASCII code	binary (8 bits)
H	72	01001000
i	105	01101001
!	33	00100001

Fig. 1: the text Hi! stored as three ASCII codes

Worked example: Codes in sequence

The ASCII code for A is 65. Give the 8-bit binary code for E.

1. E is 4 letters after A, so its code is $65 + 4 = 69$.
2. $69 = 64 + 4 + 1$.

Answer: 01000101

One code per character.

- ✓ **Character set:** every character has a unique binary code.
- ✓ **ASCII:** 7 bits, 128 characters (extended 8 bits, 256): English only.
- ✓ **Unicode:** many more characters (languages, emojis) but more bits per character.
- ✓ **Storage:** number of characters $\times$ bits per character.

Watch out!

The digit character 7 is not stored as the number 7. In ASCII it is code 55, so text digits cannot be used in a calculation until they are converted.

Saying Unicode is "better" earns nothing. Say it has a greater range of characters (other languages, emojis) but uses more bits per character, so files are bigger.

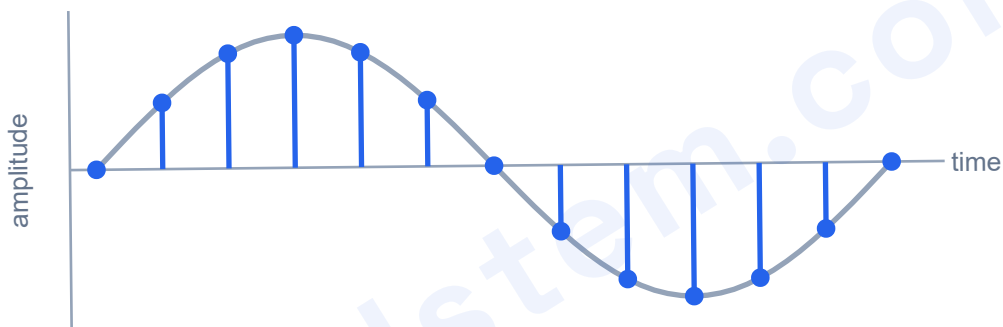
thelucidstem.com

7 Representing sound: sample rate and resolution

Syllabus 1.2

A sound wave is **sampled**: its amplitude (height) is measured at regular intervals and each measurement is stored as a binary number. The **sample rate** is the **number of samples taken in a second** (in hertz, Hz). The **sample resolution** is the **number of bits per sample**. Increasing either one makes the recording **more accurate** (closer to the original sound) but **increases the file size**.

sampling a sound wave at regular intervals



sample rate = samples per second (Hz)

sample resolution = bits used to store each sample

Fig. 1: each blue dot is one sample of the sound wave, stored as a binary number

Worked example: Size of a recording

A 1-minute recording uses a sample rate of 8000 Hz and a sample resolution of 8 bits. Calculate its size in KiB.

1. Length = 60 seconds.
2. Bits = $8000 \times 8 \times 60 = 3\,840\,000$ bits.
3. Bytes = $3\,840\,000 \div 8 = 480\,000$ bytes.
4. KiB = $480\,000 \div 1024 = 468.75$ KiB.

Answer: 468.75 KiB

Rate times resolution.

- ✓ **Sampling:** the amplitude of the wave is measured at regular intervals and stored in binary.
- ✓ **Sample rate:** samples per second (Hz).
- Resolution:** bits per sample.
- ✓ **Higher either:** more accurate recording, bigger file.
- ✓ **Size:** rate $\times$ resolution $\times$ seconds bits, then $\div 8$ and $\div 1024$ as needed.

Watch out!

Say what is measured: the amplitude (height) of the sound wave, at regular time intervals.

"The sound is recorded in binary" is too vague for a mark.

Doubling the resolution from 8 to 16 bits doubles the file size but gives 256 times as many levels (256 to 65 536). Do not mix the two up.

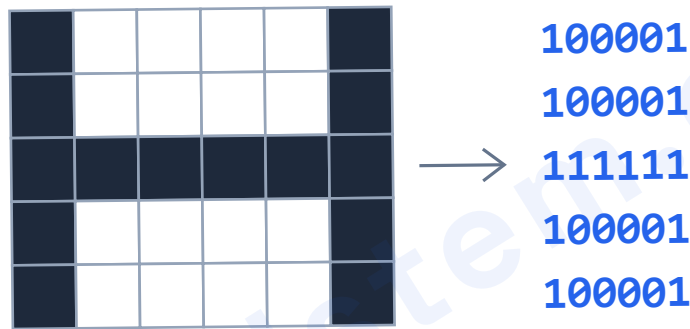
thelucidstem.com

8 Representing images: resolution and colour depth

Syllabus 1.2

An image is a series of **pixels** (picture elements), and the colour of each pixel is **converted to binary** so the computer can process it. The **resolution** is the **number of pixels** in the image (width $\times$ height). The **colour depth** is the **number of bits used to represent each colour**; n bits give 2^n possible colours. As resolution and colour depth increase, the **quality and the file size both increase**.

a 6×5 image with a colour depth of 1 bit



1 = black, 0 = white: one bit per pixel

$30 \text{ pixels} \times 1 \text{ bit} = 30 \text{ bits}$

Fig. 1: a tiny black and white image stored row by row as 1s and 0s

Worked example: Size of a photo

An image is 800 pixels wide and 600 pixels high with a colour depth of 24 bits. Calculate its file size in KiB.

1. $\text{Pixels} = 800 \times 600 = 480\,000$.
2. $\text{Bits} = 480\,000 \times 24 = 11\,520\,000 \text{ bits}$.
3. $\text{Bytes} = 11\,520\,000 \div 8 = 1\,440\,000 \text{ bytes}$.
4. $\text{KiB} = 1\,440\,000 \div 1024 = 1406.25 \text{ KiB}$.

Answer: 1406.25 KiB

Pixels times depth.

- ✓ **Resolution:** number of pixels (width × height).
- ✓ **Colour depth:** bits per pixel; n bits give 2^n colours.
- ✓ **Higher either:** better quality, bigger file.
- ✓ **Size:** width × height × colour depth bits, then $\div 8$ and $\div 1024$ as needed.

Watch out!

Colour depth is a number of *bits*, not a number of colours. "A colour depth of 256" is wrong; say 8 bits, which gives 256 colours.

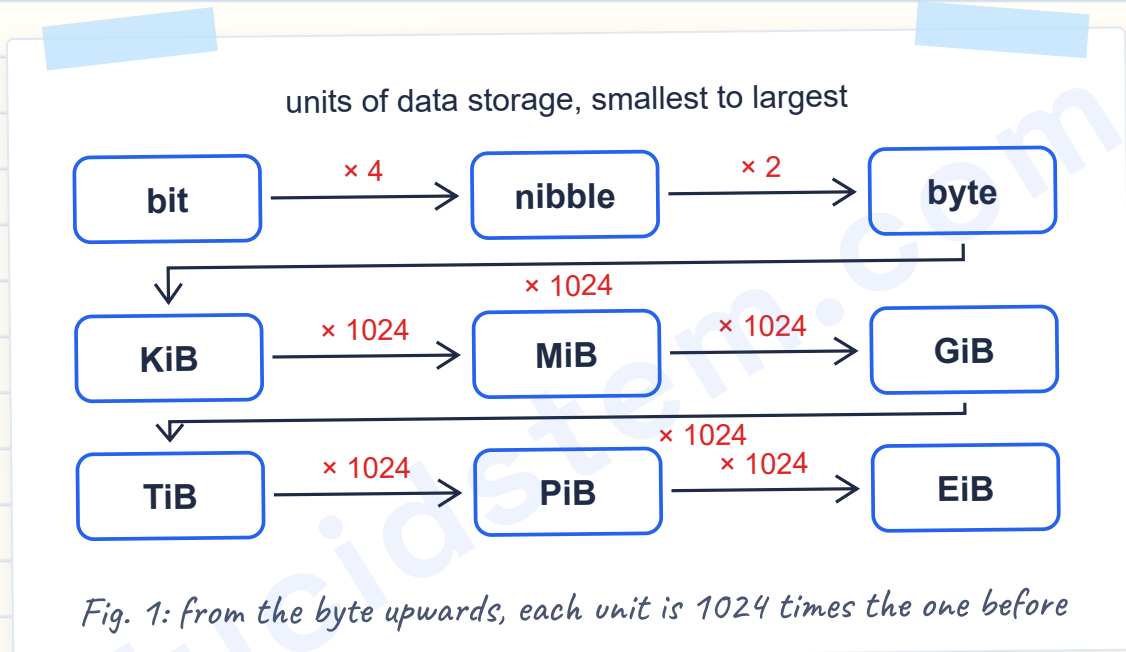
Resolution is the number of pixels in the image, such as 800×600 . It is not the size of the picture in centimetres.

thelucidstem.com

9 Measuring data storage and calculating file sizes

Syllabus 1.3

The smallest unit is the **bit**. There are **4 bits in a nibble** and **8 bits in a byte**. After that, each unit holds **1024** of the one before: kibibyte (**KiB**), mebibyte (**MiB**), gibibyte (**GiB**), tebibyte (**TiB**), pebibyte (**PiB**), exbibyte (**EiB**). Calculations must use **1024**, not **1000**, and answers must be in the **unit the question asks for**. Image size = width $\times$ height $\times$ colour depth; sound size = sample rate $\times$ resolution $\times$ seconds (both in bits).



Worked example: Bigger to smaller

Convert 3 GiB to MiB.

- 1 GiB = 1024 MiB, and MiB is the smaller unit, so multiply.
2. $3 \times 1024 = 3072$.

Answer: 3072 MiB

Bits first, then 1024.

- ✓ **Ladder:** bit, nibble (4 bits), byte (8 bits), then KiB, MiB, GiB, TiB, PiB, EiB, each $\times 1024$.
- ✓ **Image:** width $\times$ height $\times$ colour depth = bits.
- ✓ **Sound:** sample rate $\times$ resolution $\times$ seconds = bits.
- ✓ **Finish:** $\div 8$ for bytes, $\div 1024$ per step, and give the unit asked for.

Watch out!

The question decides the unit. If it asks for MiB and you stop at KiB, you lose the final mark even if the working is right.

Convert minutes to seconds before multiplying for a sound file.

Forgetting this makes the answer 60 times too small.

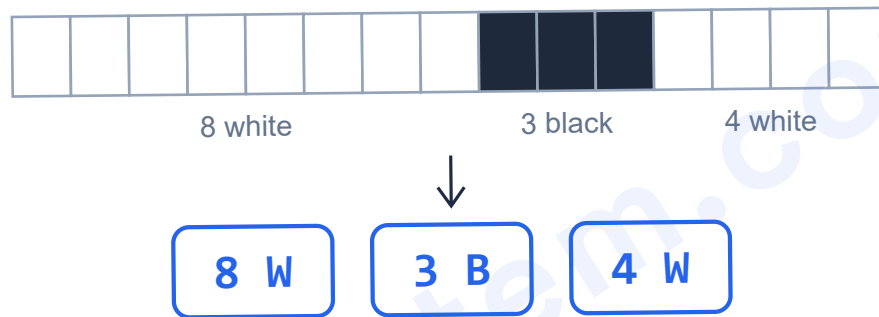
thelucidstem.com

10 Lossy and lossless compression

Syllabus 1.3

Compression reduces the size of a file, so it needs **less storage space**, **less bandwidth** and a **shorter transmission time**. **Lossy** compression **permanently removes data**, for example by reducing the resolution or colour depth of an image, or the sample rate or resolution of a sound; the original cannot be recovered. **Lossless** compression reduces the size **without permanent loss of data**, so the original file can be rebuilt exactly; **run length encoding (RLE)** is an example.

run length encoding (RLE) of one row of 15 pixels



15 values become 3 pairs: lossless, the row can be rebuilt exactly

Fig. 1: RLE stores each run of identical values as a count and the value

Worked example: Encode and count

Use RLE to encode WWWWBWWBWW. If each count and each character take 1 byte, how many bytes are saved?

1. Runs: 8 W, then 3 B, then 4 W.
2. Encoded: 8W 3B 4W, which is 3 pairs = 6 bytes.
3. Original: 15 characters = 15 bytes.
4. Saved: $15 - 6 = 9$ bytes.

Answer: 8W 3B 4W; 9 bytes saved

Lost or kept.

- ✓ **Why:** less storage, less bandwidth, shorter transmission time.
- ✓ **Lossy:** data permanently removed (resolution, colour depth, sample rate, sample resolution).
- ✓ **Lossless:** no permanent loss; the original is rebuilt exactly (RLE).
- ✓ **RLE:** each run of identical values becomes a count and the value.

Watch out!

"It makes the file smaller" only restates compression. For a mark, give the effect: less storage, less bandwidth or a shorter transmission time.

RLE does not always make a file smaller. ABCD encodes as 1A 1B 1C 1D, which is twice as long.

Never choose lossy compression for text or program files: a lost character or instruction changes the meaning or stops the program working.