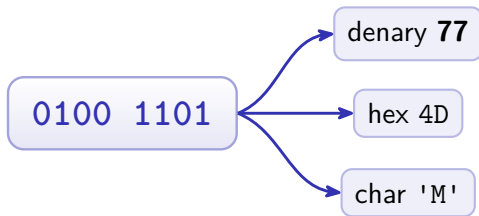


# Information Representation

## AS Computer Science ■ Topic 1

### A-Level Computer Science



## What we will cover

- 1 Numbers and characters 1.1
- 2 Prefixes, complements, BCD and hex 1.1.2
- 3 Images and sound: file sizes 1.2
- 4 Encoding a picture and a sound 1.2.1
- 5 Compression 1.3
- 6 Recap

# 1 Numbers and characters

---

|             |              |              |              |              |              |              |              |              |
|-------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| place value | 128          | 64           | 32           | 16           | 8            | 4            | 2            | 1            |
| bits        | <div>1</div> | <div>1</div> | <div>0</div> | <div>0</div> | <div>1</div> | <div>0</div> | <div>0</div> | <div>0</div> |
|             | 1100 = C     |              |              |              | 1000 = 8     |              |              |              |

$$200 = 128 + 64 + 8 = 11001000_2 = C8_{16}$$

# Sheet 1.1 – what you need to know

## You must be able to:

- convert between binary, denary and hex
- read and write **two's complement**, and give its range
- add in binary and recognise **overflow**
- use BCD, ASCII and Unicode

## What the exam asks

“Convert the 12-bit two's complement binary integer into denary.” [2]

“Write the smallest and the largest two's complement binary integers that can be represented in 8 bits.” [2]

“Convert the hexadecimal number C0F into denary.” [2]

*Practise on exercise sheet 1.1, then try the real exam questions on past-paper sheet 1.1.*

# Three number systems

Three ways to write the **same number** – here, thirteen:

**denary**

十进制 ▪ base 10  
digits 0–9

13

**binary**

二进制 ▪ base 2  
0 and 1

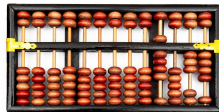
1101

**hexadecimal**

十六进制 ▪ base 16  
0–9, A–F

D

a **byte** 字节 = 8 **bits** 位; one hex digit = 4 bits



*An abacus counts by place value too*

## Converting between bases

- **denary** → **binary**: subtract the largest **place value** 位值
- **binary** → **hex**: group into **nibbles** 半字节
- **hex** → **denary**: digit × place value

|             |          |    |    |    |          |   |   |   |
|-------------|----------|----|----|----|----------|---|---|---|
| place value | 128      | 64 | 32 | 16 | 8        | 4 | 2 | 1 |
| bits        | 1        | 1  | 0  | 0  | 1        | 0 | 0 | 0 |
|             | 1100 = C |    |    |    | 1000 = 8 |   |   |   |

$$200 = 128 + 64 + 8 = 11001000_2 = C8_{16}$$

### Convert denary 200

$$200 = 128 + 64 + 8 \Rightarrow 1100\ 1000; 1100=C, 1000=8 \Rightarrow \text{hex } C8.$$

# Signed and unsigned integers

A bit pattern carries **no sign of its own** – the same byte is two numbers, read as **unsigned** 无符号 or as **signed** 有符号.

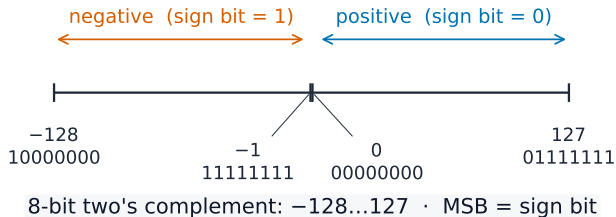
| 8 bits             | as unsigned<br>every bit a place value | as signed<br>top bit is the sign |
|--------------------|----------------------------------------|----------------------------------|
| 00000000           | 0                                      | 0                                |
| 01111111           | 127                                    | +127                             |
| 10000000           | 128                                    | -128                             |
| 11111111           | 255                                    | -1                               |
| range for $n$ bits | $0 \dots 2^n - 1$                      | $-2^{n-1} \dots 2^{n-1} - 1$     |

# Two's complement signed integers

- **sign bit** 符号位 = **most significant bit** 最高有效位 (1 = negative)
- read it: invert, +1, then negate
- range:  $-2^{n-1}$  to  $2^{n-1} - 1$

10110100

invert+1 = 01001100 = 76  $\Rightarrow$   
-76



## Addition & overflow

Add column by column from the right, carrying as in denary.

**Overflow** 溢出: the result needs more bits than the **register** 寄存器 can hold.

$$\begin{array}{r} 1111 \\ + 0001 \\ \hline \end{array}$$

overflow bit → 10000

## Worked examples – hex, bits and negative numbers

### Convert hex C0F into denary.

Place values 256, 16, 1:

$$12 \times 256 + 0 \times 16 + 15 \times 1 = 3087$$

### Fewest bits to store the value 1000?

$2^9 = 512$  is too few;  $2^{10} = 1024$  values reach 0–1023: **10 bits**.

### Smallest and largest 8-bit two's complement?

$1000\ 0000 = -128$ ;  $0111\ 1111 = +127$ .  
12 bits:  $-2048$  to  $+2047$ .

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

+45     0010 1101

invert   1101 0010

add 1   **1101 0011**

Check:  $-128 + 64 + 16 + 2 + 1 = -45$ .

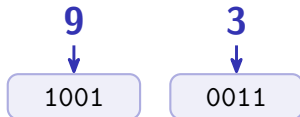
### Read the 12-bit 1111 0110 0100.

The MSB is worth  $-2048$ :

$$-2048 + 1024 + 512 + 256 + 64 + 32 + 4 = -156$$

# Binary Coded Decimal

In **BCD** 二进制十进数, each denary digit gets its own 4 bits.



In pure binary the same number is 0101 1101 – a **different pattern** of the same 8 bits.

- only 0000–1001 are valid; 1010–1111 unused
- each digit drives a display directly, and money avoids binary rounding



*Drives a 7-segment display 七段显示器*

# ASCII

Text = numbers – each character has a **code point** 码点 in a **character set** 字符集.

**ASCII**: 7 bits, 128 code points    ▪    **Extended ASCII**: 8 bits, 256

| character | code (denary) | binary   |
|-----------|---------------|----------|
| A         | 65            | 01000001 |
| a         | 97            | 01100001 |
| 0         | 48            | 00110000 |
| (space)   | 32            | 00100000 |

# Unicode

A universal character set – almost every script.

- **encodings** 编码: **UTF-8** (1–4 bytes), UTF-16, UTF-32
- **portable**, multilingual in one file
- larger files for English-only text

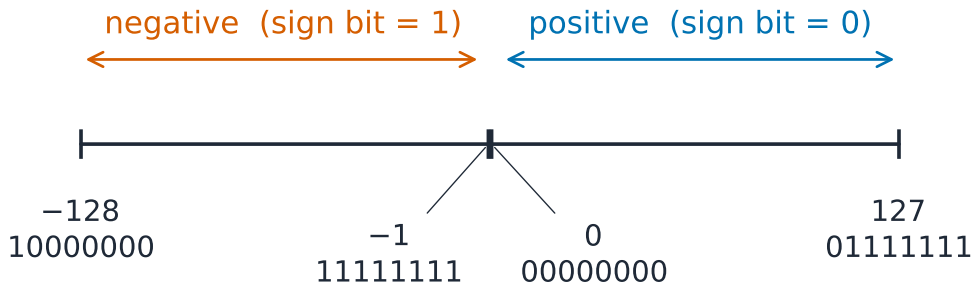
A → U+0041

中 → U+4E2D

é → U+00E9

one code point each

## 2 Prefixes, complements, BCD and hex



8-bit two's complement:  $-128 \dots 127$  · MSB = sign bit

## Sheet 1.1.2 – what you need to know

### You must be able to:

- use decimal and binary prefixes, in bytes and in bits
- subtract in binary; use one's complement
- read one bit pattern as unsigned, two's complement, hex and BCD

### What the exam asks

“Draw one line from each binary value to its equivalent value.” [5]

“State why the value in the register cannot be interpreted as BCD.” [1]

“Give one similarity and two differences between the ASCII and Unicode character sets.” [3]

*Practise on exercise sheet 1.1.2, then try the real exam questions on past-paper sheet 1.1.2.*

## Binary vs decimal prefixes

Two families look alike – powers of 10 against powers of 2. A 1 TB drive holds  $10^{12}$  bytes; an OS reporting in TiB shows a *smaller* number for the same drive.

| Decimal       | Binary                 |
|---------------|------------------------|
| kilo = $10^3$ | kibi = $2^{10} = 1024$ |
| mega = $10^6$ | mebi = $2^{20}$        |
| giga = $10^9$ | gibi = $2^{30}$        |

kilo



$10^3 = 1000$

kibi



$2^{10} = 1024$

same name, different size

## Prefixes counted in bits and bytes

|              | bytes                  | bits            |
|--------------|------------------------|-----------------|
| 1 kB (kilo)  | 1000                   | 8000            |
| 1 KiB (kibi) | 1024                   | 8192            |
| 1 MB         | $10^6$                 | $8 \times 10^6$ |
| 1 MiB        | $2^{20} = 1\,048\,576$ |                 |

Decimal prefixes step by 1000, binary prefixes by 1024. Multiply by 8 for bits.

**How many bytes in 4 GB and in 4 GiB?**

$$4 \text{ GB} = 4 \times 10^9 = 4\,000\,000\,000$$

$$4 \text{ GiB} = 4 \times 2^{30} = 4\,294\,967\,296$$

**Which is largest: 1000 kB, 980 KiB or 1 MB?**

$$1\,000\,000; \quad 980 \times 1024 = 1\,003\,520; \quad 1\,000\,000.$$

So **980 KiB**, the one with the smallest number.

## Subtraction by two's complement

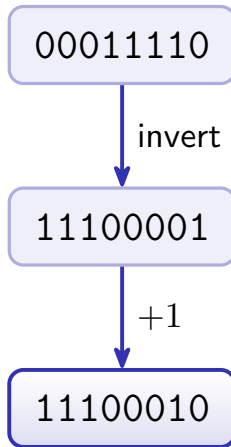
For  $A - B$ : form the **two's complement** 补码 of  $B$  (invert, add 1), add, drop the final carry.

**100 - 30 (8-bit)**

00011110  $\rightarrow$  11100010;

01100100 + 11100010

$\rightarrow$  01000110 = 70. ✓



## One's complement – the older scheme

**One's complement** 反码: negate by **only inverting** (no +1).

- $+30 = 00011110$ ,  $-30 = 11100001$
- drawback: **two zeros**

Two's complement has one zero and shares a circuit for + and -.

00000000 = +0

11111111 = -0

two zeros – wasteful

# One register, four readings

1001 0110

| read as          | value                    |
|------------------|--------------------------|
| unsigned binary  | $128 + 16 + 4 + 2 = 150$ |
| two's complement | $-128 + 22 = -106$       |
| hexadecimal      | 96                       |
| BCD              | 1001= 9, 0110= 6: 96     |

**“State why 0111 1100 cannot be interpreted as BCD.” [1]**

The nibble 1100 is 12: a BCD nibble holds only 0–9.

## Where BCD is used

Digital clocks and calculator displays (one nibble per digit); money, where denary fractions must be exact.

The bits never change; only the **reading** does.

## Hexadecimal in practice

One hex digit = **4 bits**, so one byte is exactly **two hex digits**.

Used for **memory addresses** 内存地址 (0x7FFE), colour codes (#FF8800) and MAC addresses (AC:DE:48:00:11:22).

Hex does **not** change the stored data – it only makes binary **readable for people**.

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
|---|---|---|---|

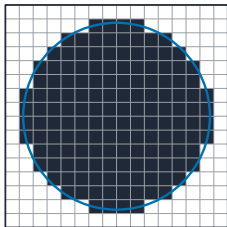
|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

    → AF  
nibble = A    nibble = F

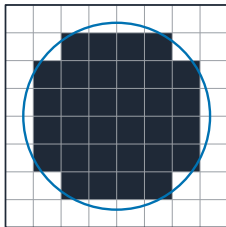
A byte is **two nibbles**; each nibble is **one hex digit**.

### 3 Images and sound: file sizes

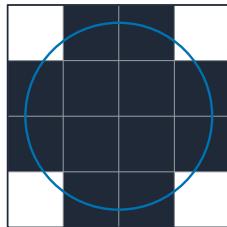
---



**A**  
 $16 \times 16$  pixels (high)



**B**  
 $8 \times 8$  pixels



**C**  
 $4 \times 4$  pixels (low)

fewer, larger pixels  $\Rightarrow$  lower resolution, less detail

## Sheet 1.2 – what you need to know

### You must be able to:

- calculate bitmap, sound and video file sizes
- give the effect of resolution and colour depth
- choose bitmap or vector for a job

### What the exam asks

“Calculate an estimate for the file size for one second of the video in gigabytes.” [2]

“Explain the effect of decreasing the bit depth on the image and on the image file.” [4]

“Convert the hexadecimal code for the character into denary.” [1]

*Practise on exercise sheet 1.2, then try the real exam questions on past-paper sheet 1.2.*

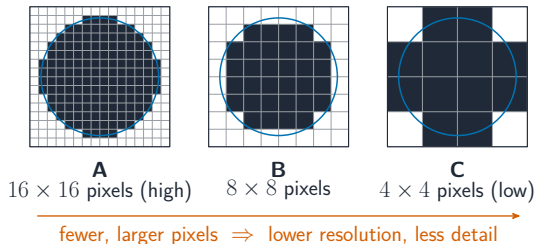
# Bitmap images

A **bitmap** 位图 stores every **pixel** 像素 in a grid, behind a **file header** 文件头 that records the resolution and colour depth.

- **resolution** 图像分辨率:  $w \times h$
  - **colour depth** 颜色深度: bits/pixel
- size in bits =  $w \times h \times \text{depth}$

**$3000 \times 2000$  at 24 bpp**

$1.44 \times 10^8 \text{ bits} \div 8 \div 1024^2 \approx$   
**17 MiB**



## Changing settings

Two settings decide a bitmap's file size: **image resolution** 图像分辨率 and **colour depth**.

| Change                    | File size | Quality                                  |
|---------------------------|-----------|------------------------------------------|
| <b>lower resolution</b>   | smaller   | less detail; <b>blocky</b> when enlarged |
| <b>lower colour depth</b> | smaller   | smooth shades show <b>banding</b>        |
| <b>higher either</b>      | larger    | better quality                           |

**Screen resolution** 屏幕分辨率 is a different thing: the monitor's own pixel grid.

# Colours, video, and why the real file is bigger

## Colours from $n$ bits

Maximum colours =  $2^n$ : 8 bits give 256; 24 bits give 16 777 216.

1000 colours need 10 bits ( $2^{10} = 1024$ ).

## Estimate one second of $1920 \times 1080$ video, 24-bit colour, 30 frames/s, in GB.

frame =  $1920 \times 1080 \times 24 \div 8 = 6\,220\,800$  B

second =  $30 \times 6\,220\,800 \approx 0.19$  GB

## Why is the real file larger than the estimate?

It also holds a **file header**: width, height, colour depth, file type, compression, other metadata.

## Convert the Unicode hex 2140 into denary.

$2 \times 4096 + 1 \times 256 + 4 \times 16 + 0 = 8512$

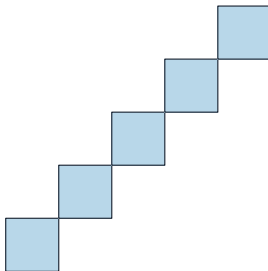
Sizes: multiply **everything** (pixels, depth, frames, seconds), then divide by 8 for bytes.

# Bitmap vs vector

| Use               | Better           |
|-------------------|------------------|
| Photograph        | <b>bitmap</b> 位图 |
| Painting, texture | bitmap           |
| Logo, icon        | vector           |
| Engineering dwg   | vector           |

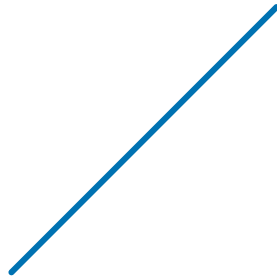
Bitmap wins on **smooth tonal detail** per area; vector **scales without blur**.

bitmap, enlarged



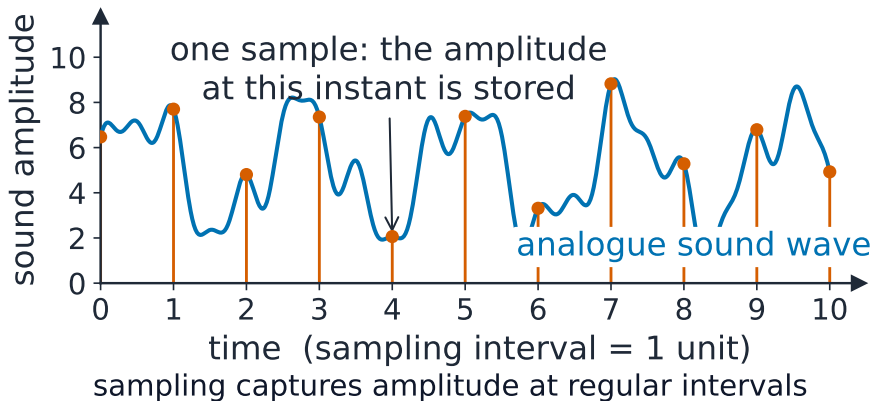
edges look jagged

vector, enlarged



edges stay smooth

## 4 Encoding a picture and a sound



## Sheet 1.2.1 – what you need to know

### You must be able to:

- describe how a bitmap and a vector graphic are encoded
- define colour depth, file header, image and screen resolution
- explain sampling rate and sampling resolution

### What the exam asks

“Describe how the data for a bitmapped image is encoded.” [3]

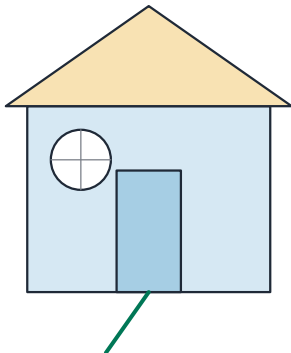
“Describe the contents of a vector graphic drawing list.” [2]

“Define the following bitmap terms. Colour depth; File header.” [2]

*Practise on exercise sheet 1.2.1, then try the real exam questions on past-paper sheet 1.2.1.*

# Vector graphics

A **vector graphic** 矢量图形 stores the **instructions** to draw the image – a **drawing list** 绘图列表 of **drawing objects** 绘图对象, each a geometric **primitive** 图元 with its own properties. Redrawn crisply at **any size**.



*the file is a list of shapes:*

- 
- |                                                                                   |                         |                                     |
|-----------------------------------------------------------------------------------|-------------------------|-------------------------------------|
|  | <b>triangle</b> — roof  | 3 points                            |
|  | <b>circle</b> — window  | $(0.75, 1.85)$ , $r\ 0.42$          |
|  | <b>rectangle</b> — body | $(0, 0)$ , $3.4 \times 2.6$         |
|  | <b>rectangle</b> — door | $(1.25, 0)$ , $0.9 \times 1.7$      |
|  | <b>line</b> — path      | $(1.7, 0) \rightarrow (1.1, -0.85)$ |

# Sampling sound

**Analogue data** 模拟数据 →

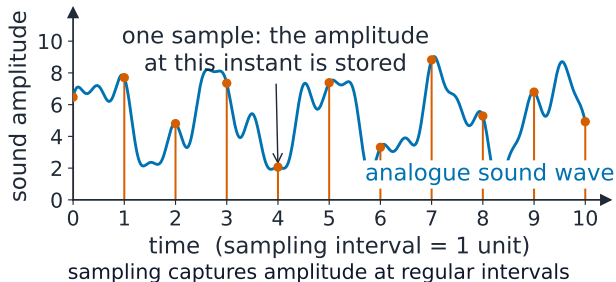
**digital data** 数字数据 by

**sampling** 采样:

- **sampling rate** 采样率 (CD 44.1 kHz)
- **sampling resolution** 采样分辨率 – bits per **amplitude** 振幅, set by **quantisation** 量化

**10 s stereo CD**

$44100 \times 16 \times 10 \times 2 \approx 1.68 \text{ MiB}$



# Describing the encoding: the answers that score

## “Describe how the data for a bitmapped image is encoded.” [3]

The image is divided into **pixels**; each pixel's colour is stored as a **binary value**; the number of bits per pixel is the **colour depth**; a **file header** records the width, height and colour depth.

## “Describe the contents of a vector graphic drawing list.” [2]

A list of **drawing objects** (shapes), each with its **properties**: coordinates, line colour and width, fill colour.

## Sampling rate

The number of samples taken **per second**, measured in **hertz** (Hz).

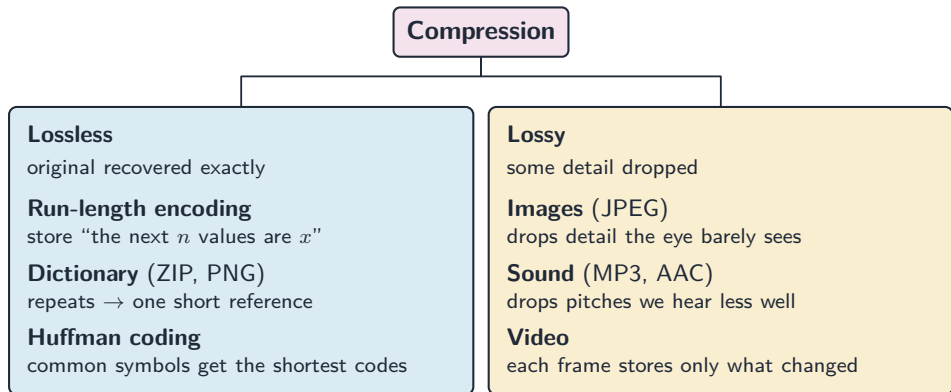
## Why not sample far above 44.1 kHz?

Human hearing stops near 20 kHz. Sampling at more than twice that already captures every audible frequency; more samples only make the **file larger**, with no difference anyone can hear.

Image resolution: the pixels in the image.  
Screen resolution: the pixels on the display.

# 5 Compression

---



## Sheet 1.3 – what you need to know

### You must be able to:

- tell **lossless** from **lossy**, with a reason
- apply run-length encoding and dictionary coding
- describe lossy methods for images and sound

### What the exam asks

“A sound file is compressed by reducing the sampling rate. State whether this is lossless or lossy. Justify your choice.” [1]

“Describe two lossy methods that Riya can use to compress the image.” [4]

“Calculate an estimate of the file size in megabytes.” [2]

*Practise on exercise sheet 1.3, then try the real exam questions on past-paper sheet 1.3.*

# Lossless vs lossy

**lossless 无损** – recovers data exactly (ZIP, PNG).

**lossy 有损** – drops detail for smaller files (JPEG, MP3).

**Compression 压缩** saves storage and **bandwidth 带宽**. Video does both at once: **spatial 空间** within each frame, **temporal 时间** between them – most frames store only what *changed*.

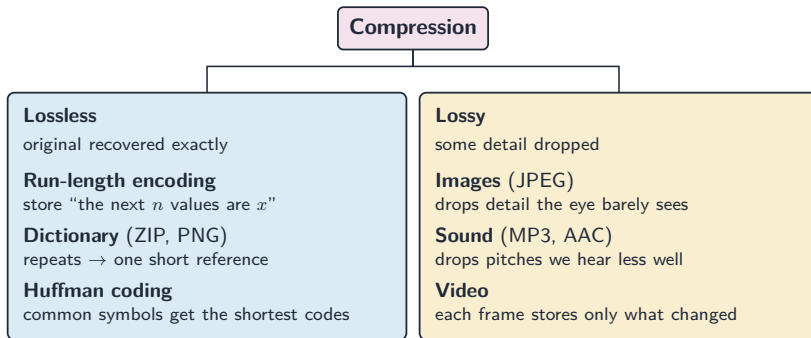
original  100%

lossless   $\approx 70\%$

lossy   $\approx 30\%$

same picture, three file sizes

# Compression methods



**lossless** 无损 recovers the original **exactly** – use it where every bit must survive (documents, code, medical images). **lossy** 有损 **drops** detail for much smaller files (JPEG, MP3, streaming video).

## Lossy methods you can name

**“Describe two lossy methods to compress the image.” [4]**

**Reduce the resolution:** fewer pixels to store.

**Reduce the colour depth:** fewer bits per pixel.

Or (JPEG) group neighbouring pixels of **similar colour** into one value, dropping detail the eye barely notices.

### Lossy sound

Lower the sampling rate or resolution; **perceptual coding** removes sounds people cannot hear (too quiet beside a loud one, or out of range).

**“The sampling rate is reduced: lossless or lossy? Justify.” [1]**

**Lossy:** the samples removed cannot be restored.

# Run-length encoding

Store each **run** 行程 of identical values once, as a **(count, value)** pair – **lossless** 无损, so the original comes back exactly. Best on **flat areas**; useless on **noisy** data, where a run of one costs more than the value it replaces.

| 8×8 bitmap | binary   | RLE code |
|------------|----------|----------|
|            | 00000000 | 8W       |
|            | 01111110 | 1W 6B 1W |
|            | 01100000 | 1W 2B 5W |
|            | 01111100 | 1W 5B 2W |
|            | 01100000 | 1W 2B 5W |
|            | 01100000 | 1W 2B 5W |
|            | 01100000 | 1W 2B 5W |
|            | 00000000 | 8W       |

**Key:** 1 = black (B), 0 = white (W). RLE stores *count + colour*: e.g. 01111110 → 1W 6B 1W

# Run-length encoding by hand



5W 3B 1W

9 pixels  $\rightarrow$  3 pairs

**Text:** AAAABBBCCD

4A3B2C1D: 10 characters become 8.

- each run becomes a (count, value) pair
- a run of **one** costs two symbols: ABAB would **grow**
- video: **spatial** compression works within a frame; **temporal** stores only the changes between frames

RLE is lossless: decoding gives back every pixel.

# Dictionary coding

**Dictionary methods** 字典压缩 store each repeated **sequence** 序列 once in a **dictionary**; every occurrence then becomes its short **index** 索引. **Lossless** 无损, and best where whole sequences repeat – text, source code, flat colour in a PNG.

**source**    ABC ABC ABC XYZ    12 characters

**dictionary**    1 → ABC    2 → XYZ    stored once each

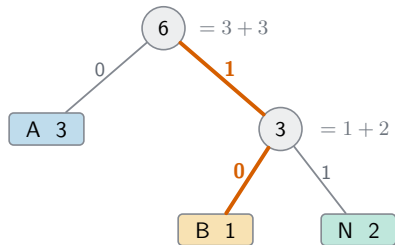
**encoded**    1 1 1 2    4 tokens + dictionary

---

The decoder rebuilds the dictionary as it reads, so the original returns exactly — **lossless** (ZIP, PNG).

# Huffman coding

Give the **commonest** symbol the **shortest** code – a **variable-length code** 变长编码, still **lossless** 无损. No code starts another, so the decoder needs no separator, and the average code length approaches the data's **entropy** 信息熵.



Walk **down from the root**, writing a bit at each branch: left = 0, right = 1.

**A** = 0

**B** = 10    right, then left

**N** = 11

each circle joins the two smallest counts

BANANA becomes 100110110 — **10 bits**, against 12 for fixed 2-bit codes: the commonest letter is nearest the root.

6

Recap

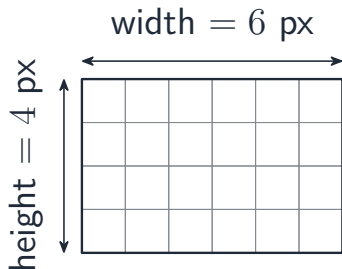
---

## Exam tips

- Show working for base conversions: denary  $\rightarrow$  binary by place values, binary  $\rightarrow$  hexadecimal in **nibbles** (groups of 4 bits).
- For **two's complement** the MSB is negative; to negate, **invert and add 1**; watch for **overflow** when the sign bit flips wrongly.
- Distinguish **bitmap** (pixels; file size = width  $\times$  height  $\times$  colour depth) from **vector** (drawing commands; scales without loss).
- Sound file size depends on **sample rate**  $\times$  **bit depth**  $\times$  **time** – more of each means better quality but a bigger file.
- Compare **lossless vs lossy** compression and give a use for each.

## Topic 1 – key takeaways

- 1 denary / binary / hex; one hex digit = 4 bits
- 2 two's complement for signed integers
- 3 text = code points; images **bitmap**/**vector**; sound **sampled**
- 4 **compression**: lossless or lossy



$$\text{pixels} = 6 \times 4 = 24$$

$$\text{size} = 24 \times 8 = \mathbf{192} \text{ bits}$$

(**8** = colour depth, bits per pixel)

# Check yourself

- 1 Convert denary 173 to binary and hex.
- 2 8-bit two's-complement  $11111011 = ?$
- 3  $1920 \times 1080$  at 24 bpp – size in MiB?
- 4 Why is JPEG lossy but PNG lossless?



16 pixels

$\Rightarrow$  **6W 4B 6W**

(3 runs, not 16)

Answers: 10101101/AD ■ -5 ■  $\approx 5.9$  MiB ■ JPEG drops unseen detail; PNG keeps every pixel.