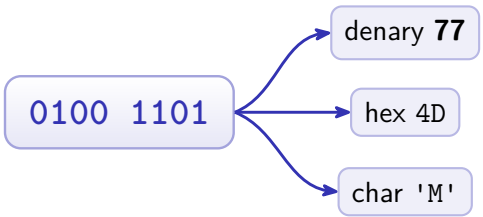
Information Representation

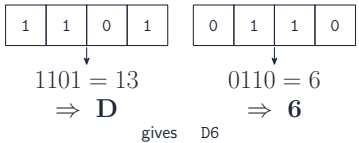
AS Computer Science ▪ Topic 1

A-Level Computer Science



What we will cover

1. Number systems
2. Binary arithmetic
3. BCD & hexadecimal
4. Character codes
5. Images
6. Sound
7. Compression



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

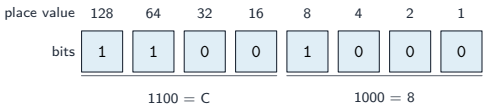
1101

abacus

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

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



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

Convert denary 200

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

Binary vs decimal prefixes

Two families look alike – powers of 10 vs powers of 2.

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

same name, different size

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.

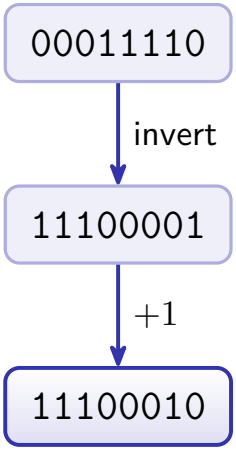
$$
\begin{array}{r}
1111 \\
+\ 0001 \\
\hline
10000 \leftarrow \text{overflow bit}
\end{array}
$$

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→11100010;
01100100+11100010
→ 01000110= 70. ✓

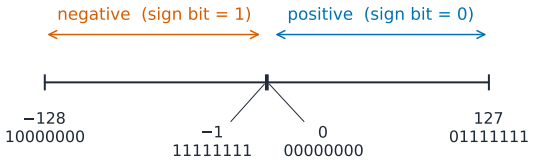


Two's complement signed integers

- **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= $76 \Rightarrow -76$



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

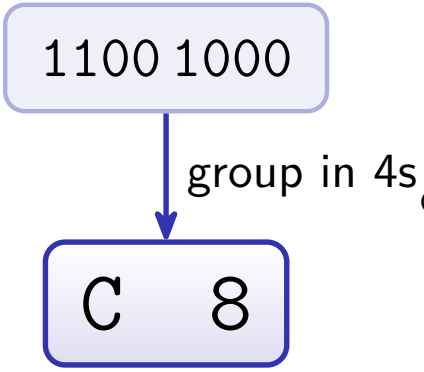
Binary Coded Decimal

In **BCD** 二进码十进数, each denary digit is its own 4 bits.

- $93 = 1001\ \ 0011$ (not binary 93)
- only 0000–1001 are valid
- clocks, calculators, currency

*Drives a **7-segment display** 七段显示器*

Hexadecimal in practice



One hex digit = 4 bits – compact binary:

- **memory address** 内存地址 `0x7FFE`
- colour `#FF8800`
- MAC `AC:DE:48:00:11:22`

one byte = two hex digits

ASCII

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

- **ASCII**: 7 bits, 128 points
- **Extended**: 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

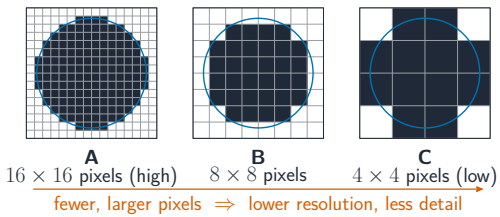
one code point each

Bitmap images

A **bitmap** 位图 stores every **pixel** 像素 in a grid.

- **image resolution** 图像分辨率: w × h
- **colour depth** 颜色深度: bits per pixel

size = w × h × depth

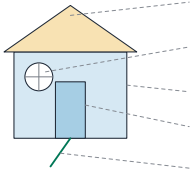


Fewer pixels = less detail

Vector graphics

A **vector graphic** 矢量图形 is a list of **primitives** 图元, not pixels:

- shapes – lines, circles, polygons
- each with properties – position, colour, width
- redraws crisply at any size



Stored as a *list of shapes*, each with attributes:

triangle – roof
3 points, orange fill

circle – window
centre $(0.75, 1.85)$, $r = 0.42$

rectangle – body
corner $(0, 0)$, 3.4×2.6

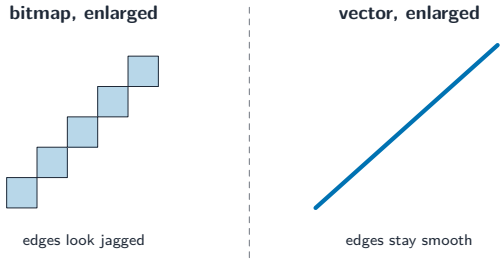
rectangle – door

line – path
start, end points

Bitmap vs vector

Use	Better
Photograph	bitmap
Logo, icon	vector
Engineering drawing	vector

Vector scales without blur.

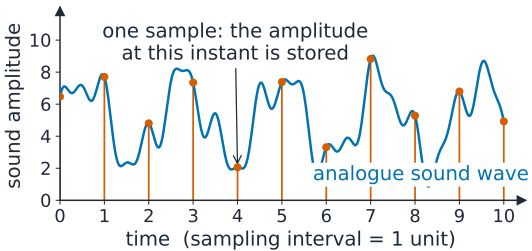


Enlarged: bitmap jagged, vector smooth

Sampling sound

Analogue data 模拟数据 → **digital data** 数字数据 by **sampling** 采样:

- **sampling rate** 采样率 (CD $44.1\,\mathrm{kHz}$)
- **sampling resolution** 采样分辨率 – bits per **amplitude** 振幅

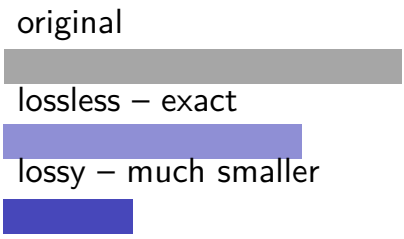


Lossless vs lossy

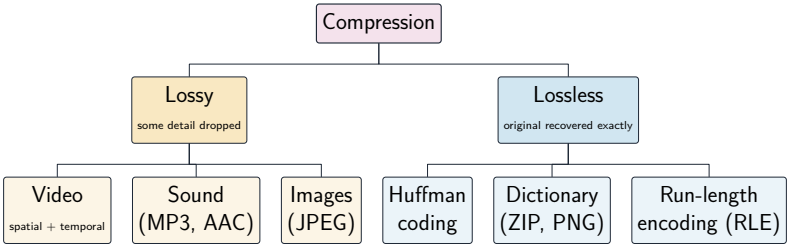
lossless 无损 – recovers data exactly (ZIP, PNG).

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

> **Compression** 压缩 saves storage and **bandwidth** 带宽.



Compression methods

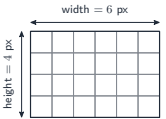


- **run-length encoding** 行程编码
- **dictionary methods** 字典编码
- **Huffman coding** 霍夫曼编码

Video adds **spatial** 空间 + **temporal** 时间 redundancy between frames.

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



pixels $= 6 \times 4 = 24$

size $= 24 \times \mathbf{8} = \mathbf{192}$ 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. 1920×1080 at 24 bpp – size in MiB?
4. Why is JPEG lossy but PNG lossless?

16 pixels
⇒ **6W 4B 6W**
(3 runs, not 16)

Answers: `10101101/AD` ▪ -5 ▪ $\approx 5.9\,\mathrm{MiB}$ ▪ JPEG drops unseen detail; PNG keeps every pixel.