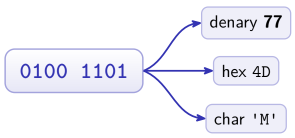
Information Representation

AS Computer Science ▪ Topic 1

A-Level Computer Science



What we will cover

1 Number systems

place value	128	64	32	16	8	4	2	1
bits	1	1	0	0	1	0	0	0

1100 = C   1000 = 8

$$200 = 128 + 64 + 8 = 11001000_2 = \mathtt{C8}_{16}$$

Three number systems

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

denary	**binary**	**hexadecimal**
十进制 ▪ base 10	二进制 ▪ base 2	十六进制 ▪ base 16
digits 0–9	0 and 1	0–9, A–F
13	1101	D

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

place value	128	64	32	16	8	4	2	1
bits	1	1	0	0	1	0	0	0

1100 = C   1000 = 8

$$200 = 128 + 64 + 8 = 11001000_2 = \mathtt{C8}_{16}$$

Convert denary 200

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

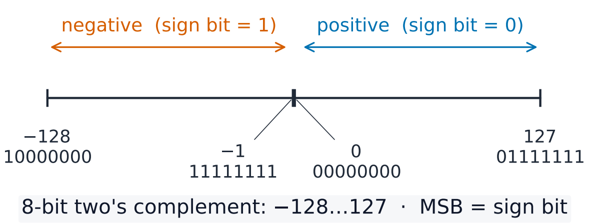
Binary vs decimal prefixes

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

2 Binary arithmetic



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 every bit a place value	as signed top bit is the sign
00000000	0	0
01111111	127	$+127$
10000000	128	-128
11111111	255	-1
range for n bits	$0\dots2^n-1$	$-2^{n-1}\dots2^{n-1}-1$

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.

```
   1111
+  0001
-------
  10000
```

overflow bit → 10000

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$=70$. ✓

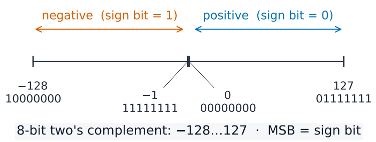
00011110 → invert → 11100001 → $+1$ → 11100010

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

3 BCD & hexadecimal

Binary Coded Decimal

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

9	3
1001	0011

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

Hexadecimal in practice

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

11001000 ⟶ 1100 1000 ⟶ C 8

group in 4s   nibble → hex

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

4 Character codes

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

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

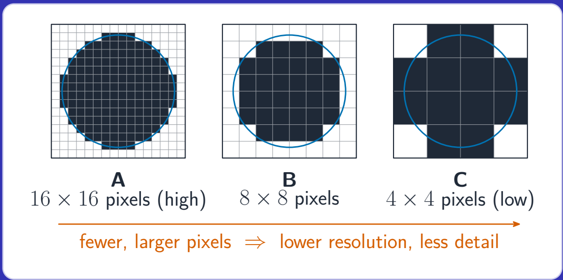
A ⟶ U+0041

中 ⟶ U+4E2D

é ⟶ U+00E9

one code point each

5 Images

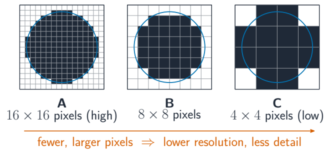


Bitmap images

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

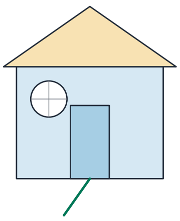
- **resolution** 图像分辨率: w × h
- **colour depth** 颜色深度: bits/pixel

  size = w × h × depth



Vector graphics

A **vector graphic** 矢量图形 is a list of **primitives** 图元, not pixels: shapes with properties, redrawn crisply at any size.



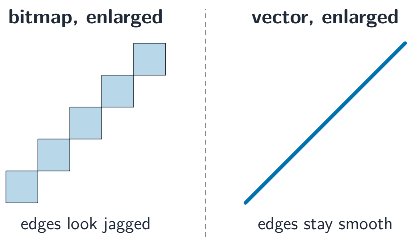
the file is a list of shapes:

triangle — roof	3 points
circle — window	$(0.75, 1.85)$, $r\,0.42$
rectangle — body	$(0, 0)$, 3.4×2.6
rectangle — door	$(1.25, 0)$, 0.9×1.7
line — path	$(1.7, 0) \to (1.1, -0.85)$

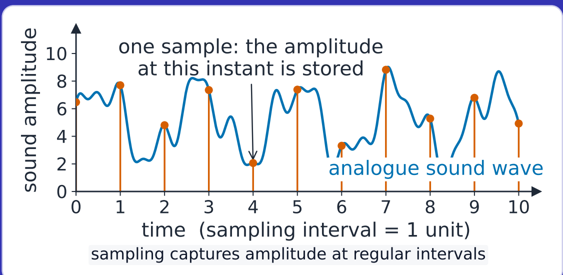
Bitmap vs vector

Use	Better
Photograph	**bitmap** 位图
Logo, icon	vector
Engineering dwg	vector

Vector scales without blur.



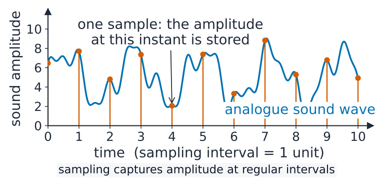
6 Sound



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



7 Compression

Compression

Lossless
original recovered exactly
Run-length encoding
store "the next n values are x"
Dictionary (ZIP, PNG)
repeats → one short reference
Huffman coding
common symbols get the shortest codes

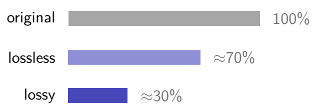
Lossy
some detail dropped
Images (JPEG)
drops detail the eye barely sees
Sound (MP3, AAC)
drops pitches we hear less well
Video
each frame stores only what changed

Lossless vs lossy

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

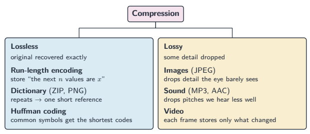
lossy 有损 – drops detail for smaller files (JPEG, MP3).

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



same picture, three file sizes

Compression methods



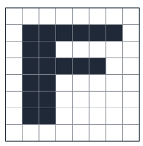
Which one? Lossless where every bit must survive (documents, code, medical images); lossy where size matters more than detail (streaming video and audio).

- **lossless** 无损 – the original data is recovered exactly (text, programs, ZIP/PNG).
- **lossy** 有损 – some detail is dropped for much smaller files (JPEG, MP3, video).

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

Dictionary coding

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

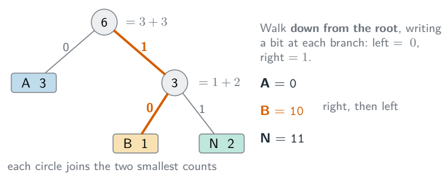
dictionary `1` → `ABC` `2` → `XYZ` — stored once each

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.

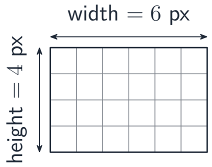


BANANA becomes 10 0 11 0 11 0 — **10 bits**, against 12 for fixed 2-bit codes: the commonest letter is nearest the root.

8 Recap

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



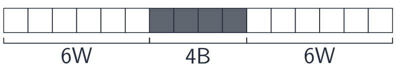
pixels $= 6 \times 4 = 24$

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

($\mathbf{8}$ = 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

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

(3 runs, not 16)

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