Course Companion

IGCSE Computer Science

Every topic handout in one volume

全部专题讲义合集

exercises.lol

Contents 目录

Data representation

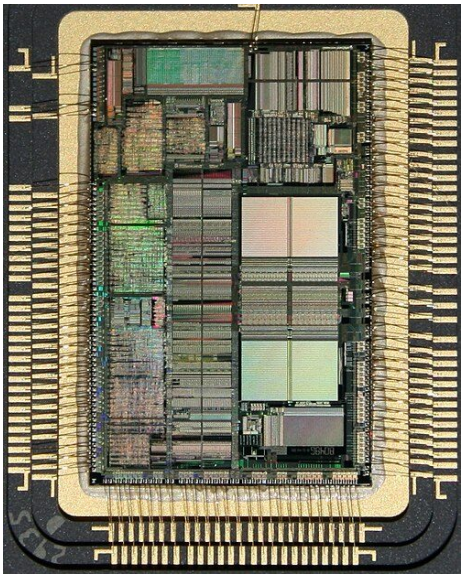
IGCSE Computer Science

Why computers use binary

A computer can only work with two states: on and off. You write these as 1 and 0. A system that uses only two digits 数字 is called **binary** 二进制 (base 2).

Computers represent all data — numbers, text, sound and images — as binary strings of 0s and 1s

Every kind of data 数据 — numbers, text, sound and images — must be changed into binary before a computer can use it. The computer processes this binary using **logic gates** 逻辑门, and stores it in **registers** 寄存器 (small, fast stores inside the processor 处理器).



A microprocessor holds millions of tiny transistors, each a switch that is on (1) or off (0) — the physical basis of binary

Number systems

A **number system** 数制 is a way of writing numbers using a fixed set of digits. You need three of them.

System	Base	Digits used
Denary	10	0–9
Binary	2	0 and 1
Hexadecimal	16	0–9 then A–F

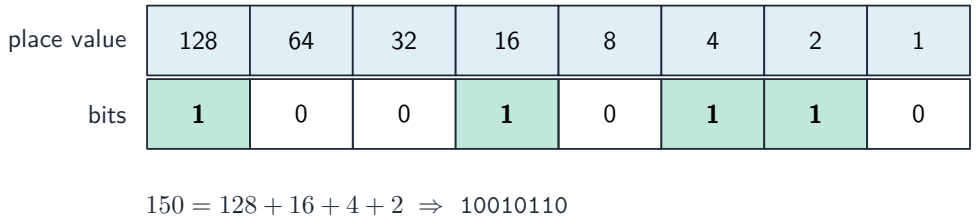
- **denary** 十进制 is the normal counting system (also called decimal).
- **binary** uses only 0 and 1.
- **hexadecimal** 十六进制 (hex) uses sixteen digits: 0–9, then A, B, C, D, E, F stand for 10, 11, 12, 13, 14, 15.

The **base** 基数 tells you how many different digits a system uses.

Place value

Each column in a number has a **place value** 位值. In binary the place values double from right to left. For an 8-bit number they are:

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



An 8-bit place-value chart: the 1s sit under the values that add up to 150

One **bit** 位 is a single 0 or 1. Eight bits make one **byte** 字节. Four bits (half a byte) is a **nibble** 半字节.

Converting between number systems

Denary → binary. Write the place values. Put a 1 under each value you need so they add up to your number; put 0 under the rest.

Example: change denary 150 to binary. $150 = 128 + 16 + 4 + 2$.

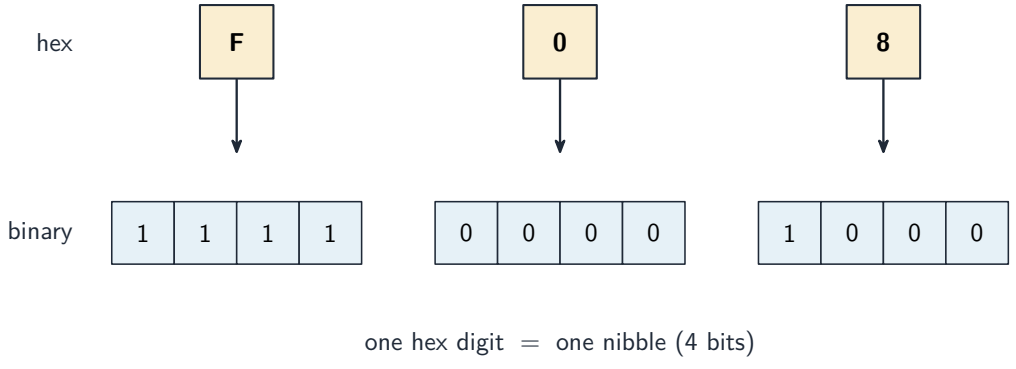
```
128 64 32 16 8 4 2 1
  1  0  0  1 0 1 1 0
```

So 150 = `10010110`.

Binary → denary. Add the place values where there is a 1. `10010110` $= 128 + 16 + 4 + 2 = 150$.

Hexadecimal → binary. Change each hex digit into its own 4-bit group (a nibble).

Example: hex F08. $F = 1111$, $0 = 0000$, $8 = 1000$, so F08 = `1111 0000 1000`.



Each hex digit maps to its own 4-bit nibble — F08 = 1111 0000 1000

Binary → hexadecimal. Group the bits into nibbles of 4, starting from the right. Change each nibble to one hex digit.

Denary → hexadecimal. The easy way is to change to binary first, then binary to hex.

This table helps with the hex letters:

Denary	Binary	Hex
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Cambridge questions use binary numbers up to 16 bits long.

Worked example. Convert denary 100 to 8-bit binary, then to hexadecimal.

$100 = 64 + 32 + 4$, so the binary is `01100100`. Splitting into nibbles, `0110 0100` = 6 and 4, so the hexadecimal is `64`.

Why hexadecimal is used

Hex is shorter than binary and easier for people to read and write. One hex digit replaces 4 binary digits, so you make fewer mistakes. The value does not change — hex is just a shorter way to show the same binary.

Computer scientists use hex for:

- MAC addresses and IPv6 addresses
- colour codes in HTML (for example `#FF0000` is red)
- **memory addresses** 内存地址 and error codes
- showing the contents of memory (a "memory dump")

Binary addition

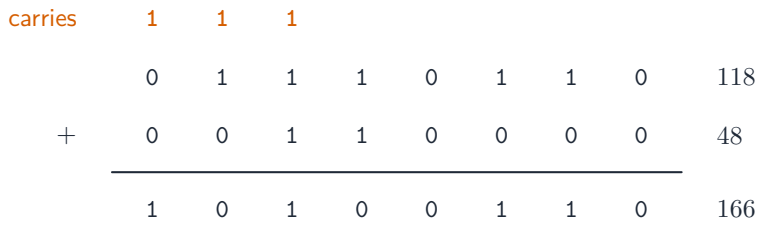
You can add two 8-bit binary numbers, column by column from the right, just like denary. The rules for one column are:

A	B	Result bit	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

When a carry also comes in, $1 + 1 + 1 = 1$ with a carry of 1.

Example: add `01110110` (118) and `00110000` (48).

```
  0 1 1 1 0 1 1 0    (118)
+ 0 0 1 1 0 0 0 0    (48)
--------------------
  1 0 1 0 0 1 1 0    (166)
```

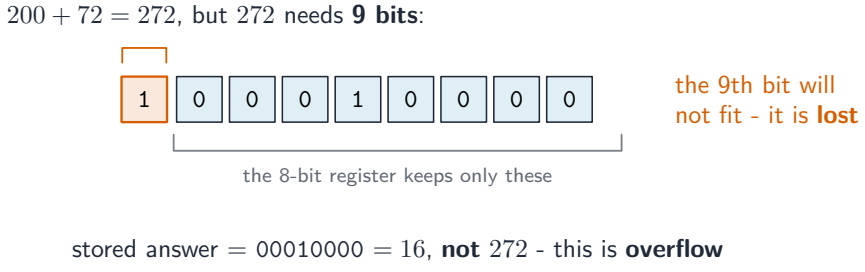


Adding column by column; the carries ripple to the left. 118 + 48 = 166

Overflow

An 8-bit register can hold denary values from 0 to 255 only. If an addition gives a result above 255, the answer needs a 9th bit. The register cannot hold this extra bit, so it is lost. This is called **overflow** 溢出 (an overflow error). It happens when a value goes outside the limit the register can store.

Example: `11001000` (200) + `01001000` (72) = 272. In binary that is `1 00010000`, which needs 9 bits. The leading 1 will not fit in 8 bits, so the stored answer is wrong.



Adding 200 and 72 needs 9 bits, but an 8-bit register drops the ninth, so the answer is wrong

Logical binary shift

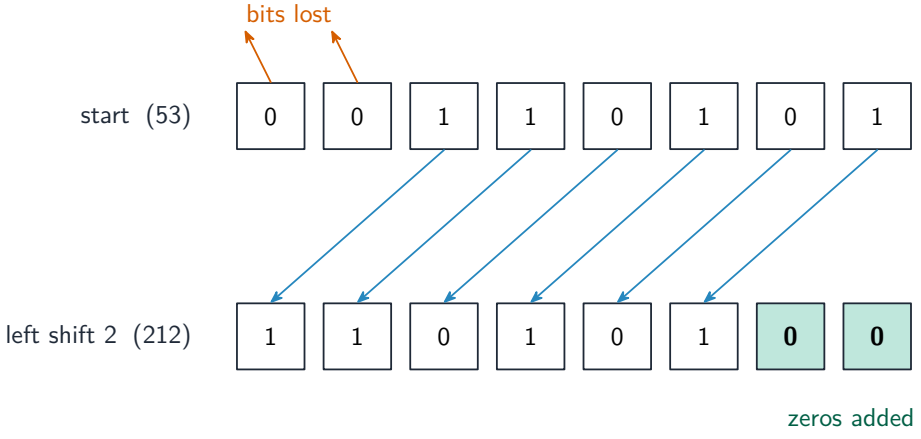
A **logical binary shift** 逻辑二进制移位 moves all the bits left or right by a number of places.

- Bits that move off the end of the register are **lost**.
- **Zeros** are added at the empty end.

A **left shift** multiplies the number by 2 for each place moved. A **right shift** divides it by 2 for each place; the right-most bits (the **least significant bit(s)** 最低有效位) are lost.

Example: left shift `00110101` (53) by 2 places.

```
start:          0 0 1 1 0 1 0 1
left shift 2:   1 1 0 1 0 1 0 0
```



A left shift of 2: every bit moves 2 places left, the top bits are lost and zeros fill the right

The result is `11010100` (212), which is 53×4. The two left-most bits were lost and two zeros came in on the right. If a 1 is pushed off the end, that information is gone for good.

Two's complement

So far the numbers were positive. **Two's complement** 补码 lets an 8-bit register hold negative numbers too.

In two's complement, the left-most bit (the **most significant bit** 最高有效位, or MSB) has a *negative* place value:

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

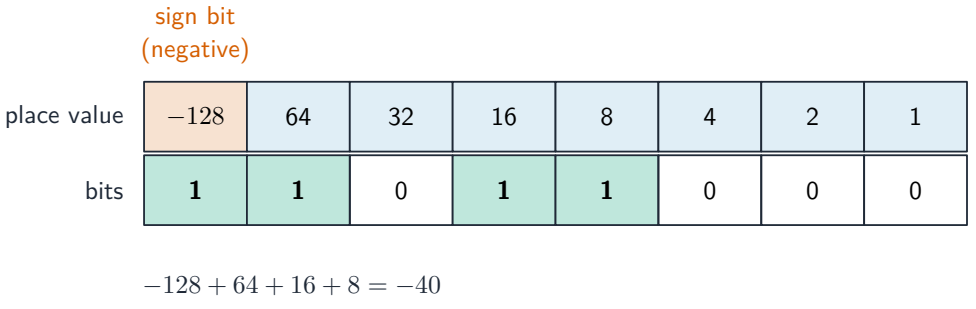
- If the MSB is 0, the number is positive.
- If the MSB is 1, the number is negative.

To make a positive number negative: write the positive binary, flip every bit ($0 \leftrightarrow 1$), then add 1.

Example: make -40.

- $+40$ = `00101000`
- flip the bits = `11010111`
- add 1 = `11011000`

So -40 = `11011000`. Check by adding the place values: $-128 + 64 + 16 + 8 = -40$.



The most significant bit is worth −128, so 11011000 = −128 + 64 + 16 + 8 = −40

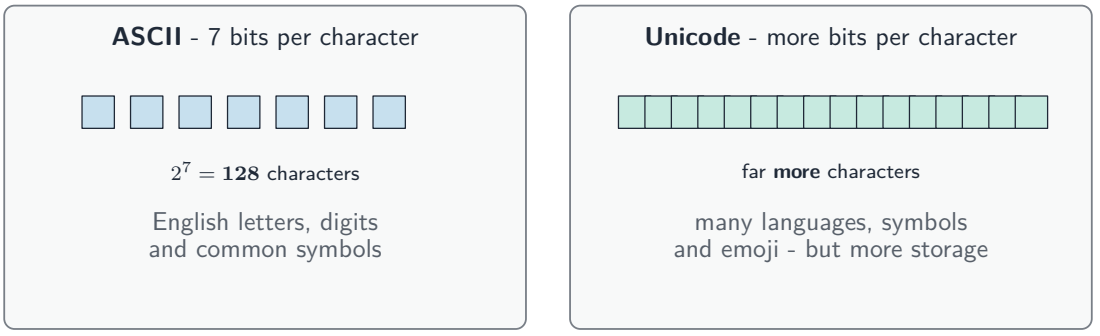
To read a negative two's complement number, just add the place values (the MSB counts as -128). The range of an 8-bit two's complement number is -128 to $+127$.

Representing text

Computers store text by giving every character a number, then storing that number in binary. The set of characters a computer can use, together with their numbers, is a **character set** 字符集.

- **ASCII** uses 7 bits per character, so it has 128 different characters. This is enough for English letters, digits and common symbols.
- **Unicode** uses more bits per character. It can represent far more characters — many languages, plus symbols and **emoji** 表情符号.

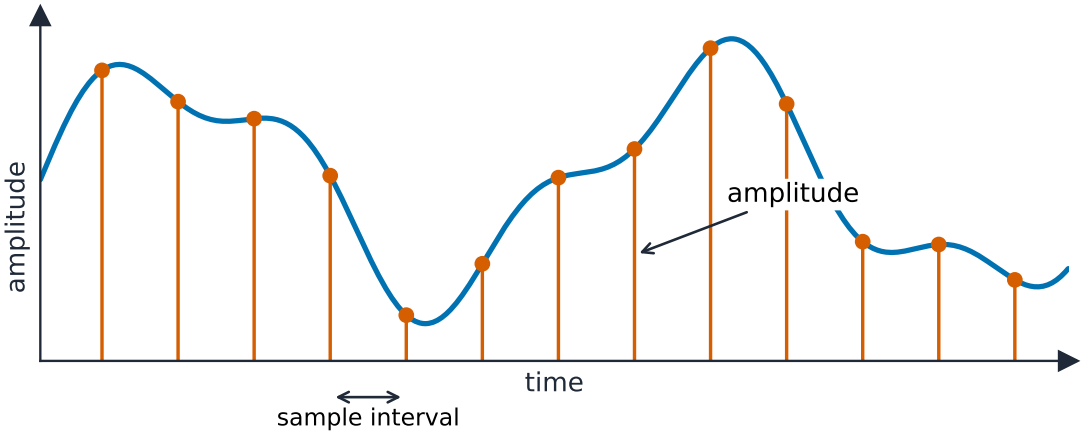
Because Unicode has more characters, it needs more bits per character than ASCII, so the same text takes more storage 存储.



ASCII uses 7 bits for 128 characters; Unicode uses more bits for far more characters but more storage

Representing sound

A **sound wave** 声波 is smooth and always changing. To store it, the computer measures the height of the wave at regular moments. This is called **sampling** 采样, and each measurement is a sample.



sample rate = samples/s · Nyquist $\geq 2f_{max}$

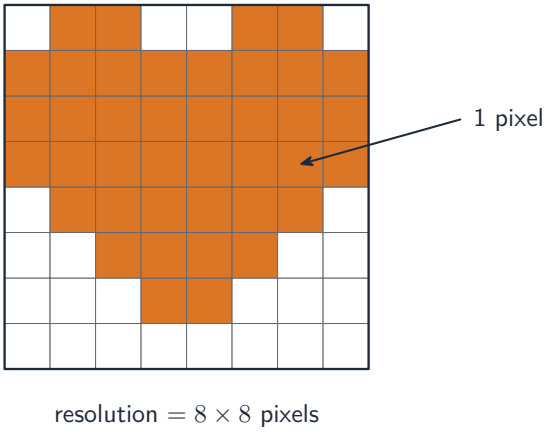
Sampling records the wave's height (amplitude) at regular moments

- **sample rate** 采样率 is the number of samples taken each second (measured in Hz).
- **sample resolution** 采样分辨率 is the number of bits used for each sample. The height of the wave at a sample point is its **amplitude** 振幅.

A higher sample rate and a higher sample resolution give a more accurate recording, but a larger file.

Representing images

A computer image is made of a grid of small dots called **pixels** 像素.



A bitmap is a grid of pixels; the resolution is how many pixels it has

- **resolution** 分辨率 is the number of pixels in the image (for example 1920×1080).
- **colour depth** 颜色深度 is the number of bits used to store the colour of each pixel.

A higher resolution and a higher colour depth give a better-quality image, but a larger file.

Measuring data storage

Data storage is measured in the units below. A nibble is 4 bits and a byte is 8 bits; from the kibibyte upward, each unit is 1024 times the one before it (because $1024 = 2^{10}$, which fits binary).

Unit	Equals
bit	a single 0 or 1
nibble	4 bits
byte	8 bits
kibibyte (KiB)	1024 bytes
mebibyte (MiB)	1024 KiB
gibibyte (GiB)	1024 MiB
tebibyte (TiB)	1024 GiB
pebibyte (PiB)	1024 TiB
exbibyte (EiB)	1024 PiB

Hard-disk platters: storage is measured in bytes — knowing file size needs width × height × colour depth for images

Calculating file size

Image file size (in bits) = resolution × colour depth = width × height × colour depth.

Example: an image is 1024 × 1024 pixels with a colour depth of 2 bytes (= 16 bits).

- bits = $1024 \times 1024 \times 16 = 16\,777\,216$ bits
- bytes = $\div 8 = 2\,097\,152$ bytes
- KiB = $\div 1024 = 2048$ KiB
- MiB = $\div 1024 = 2$ MiB

Sound file size (in bits) = sample rate × sample resolution × length in seconds.

Always divide by 1024 (not 1000) to change to KiB, MiB and so on. Give your answer in the unit the question asks for.

Worked example. A sound is recorded for 30 seconds at a sample rate of 8,000 Hz with a sample resolution of 16 bits. Find the file size in kibibytes (KiB).

- bits = $8\,000 \times 16 \times 30 = 3\,840\,000$ bits
- bytes = $3\,840\,000 \div 8 = 480\,000$ bytes
- KiB = $480\,000 \div 1024 \approx 469$ KiB

Compression

Compression 压缩 makes a file smaller. A smaller file:

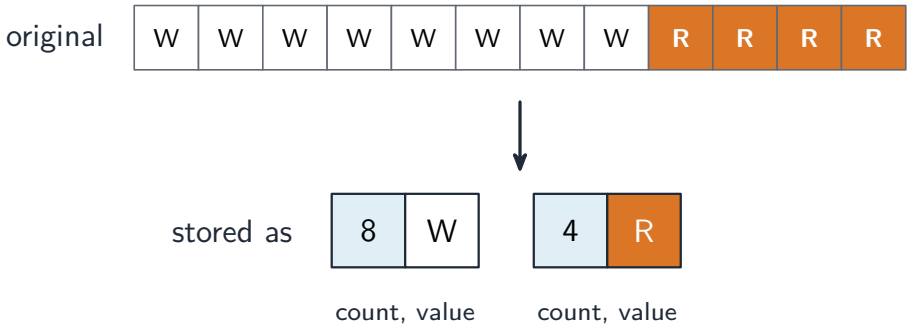
- uses less storage space,
- needs less **bandwidth** 带宽 (the amount of data a connection can carry),
- takes a shorter time to send (a shorter **transmission** 传输 time).

There are two types.

Lossless compression

Lossless 无损 compression makes the file smaller with **no permanent loss** of data. The original file can be rebuilt exactly.

One method is **run-length encoding** 行程编码 (RLE). It replaces a run of repeated values with one copy of the value and a count of how many times it repeats. For example `WWWWWWWW` (8 whites) is stored as "8 W". This works well when data has many repeats.



Run-length encoding stores each run once as a count and a value

Lossy compression

Lossy 有损 compression makes the file much smaller by **permanently removing** some data. The removed data cannot be got back. For example:

- reducing the resolution or colour depth of an image,
- reducing the sample rate or sample resolution of a sound.

Use lossless when you must keep every detail (text and program files). Use lossy for photos, music and video, where a small loss of quality is worth a much smaller file.

Exam tips

- Convert denary → binary by subtracting the place values (128, 64, 32 …); binary → denary by adding the place values that hold a 1.
- To convert to hex, group the binary into **nibbles** of 4 bits from the right; each nibble is exactly one hex digit.
- **Overflow** happens when a result needs more bits than the register has (an 8-bit register only holds 0–255), so the extra bit is lost.

- File size in bits: for an image, width × height × colour depth; for sound, sample rate × resolution × seconds. Divide by 8 for bytes, then by 1024 for each larger unit.
- **Lossless** compression keeps every bit (text; run-length encoding); **lossy** permanently removes data (photos, music) for a much smaller file.

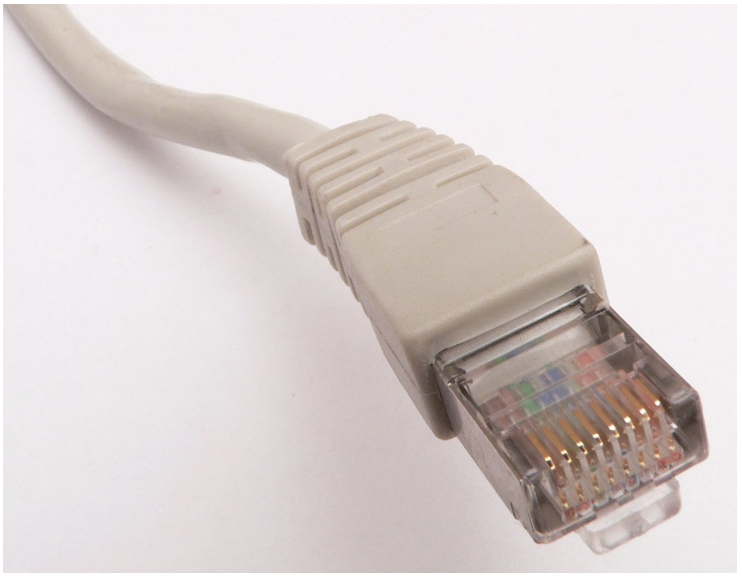
Data transmission

IGCSE Computer Science

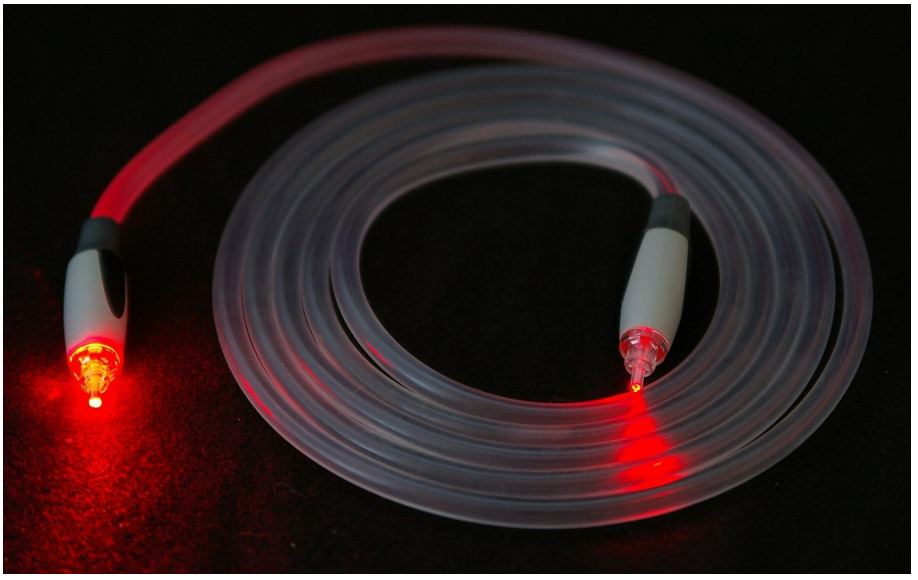
What is data transmission?

Data transmission 数据传输 means moving data from one place to another — for example from your computer to a website, or from a phone to a printer.

Data often travels along a cable. Two common ones are shown below.



An Ethernet cable with an RJ45 plug, often used to carry data on a wired network



A fibre-optic cable carries data as pulses of light along thin glass fibres

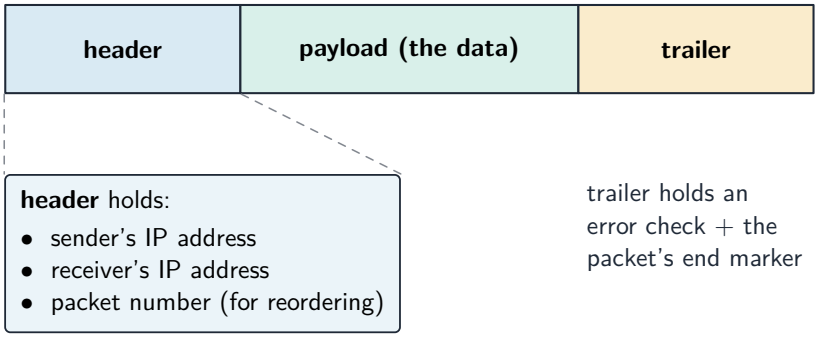
Before it is sent, the data is broken down into small, equal-sized pieces called **packets** 数据包. Each packet travels separately and they are put back together at the other end.

Using packets has benefits:

- if one packet is lost or damaged, only that packet is sent again, not the whole file;
- packets can take different routes, so a busy or broken route can be avoided;
- many users can share the same connection at the same time.

The structure of a packet

Every packet has three parts.



A packet has a header, a payload (the data) and a trailer; the header carries the addresses and packet number

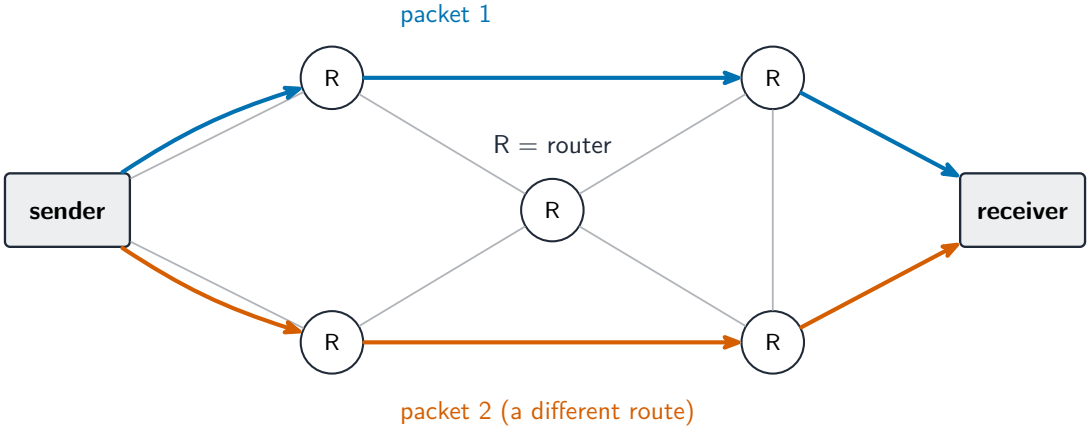
Part	What it holds
packet header 包头	control information (see below)
payload 有效载荷	the actual data being carried
trailer 尾部	shows where the packet ends, plus an error check

The **packet header** holds:

- the sender's IP address 地址 (where the packet came from),
- the receiver's IP address (where it is going),
- the **packet number** (so the packets can be put back in the right order).

Packet switching

Packet switching 数据包交换 is the way packets are sent across a network.



Routers forward packets, and different packets can take different routes to the same receiver

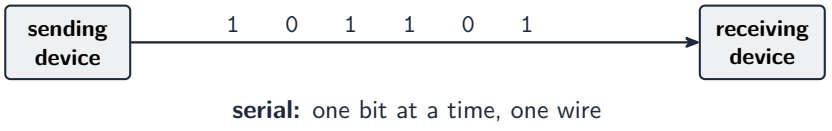
- Special devices called **routers** 路由器 read each packet's header and choose the best route for it.
- Each packet may take its **own path**, so packets can travel different ways.
- Packets may **arrive out of order**.
- When the last packet has arrived, the packets are **reordered** using their packet numbers.

Methods of data transmission

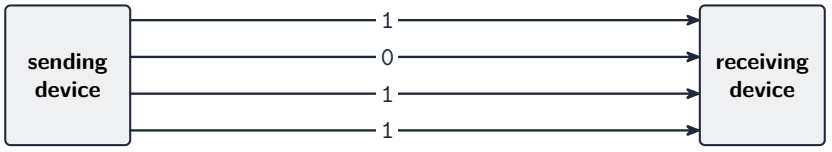
Serial and parallel

	Serial 串行	**Parallel 并行**
How	one bit at a time, down one wire	several bits at once, down several wires
Distance	good over long distances	good over short distances only
Cost	cheaper (fewer wires)	more expensive (more wires)
Errors	bits stay in order	bits can arrive at slightly different times over long wires

Serial sends one bit after another along a single wire. **Parallel** sends several bits at the same time along several wires, so it is faster over short distances.



serial: one bit at a time, one wire

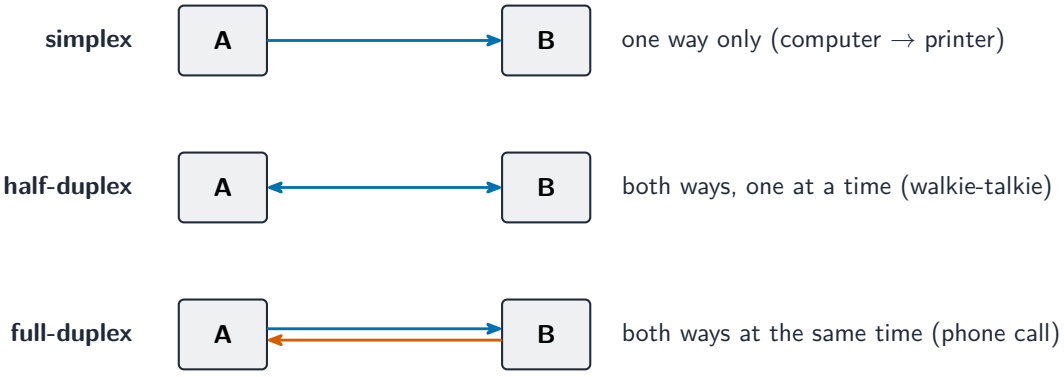


parallel: several bits at once, several wires

Serial sends one bit at a time down one wire; parallel sends several bits at once down several wires

Simplex, half-duplex and full-duplex

These describe the *direction* data can travel.



Simplex is one way only; half-duplex is both ways but one at a time; full-duplex is both ways at once

Method	Direction
simplex 单工	one direction only (e.g. computer → printer)
half-duplex 半双工	both directions, but only one at a time (e.g. walkie-talkie)
full-duplex 全双工	both directions at the same time (e.g. phone call)

Choosing a method

The choice depends on the **distance**, the **speed** needed, the **cost**, and whether data must travel both ways at once.

USB

The **universal serial bus** 通用串行总线 (USB) is a common **interface** 接口 (a connection point) for joining devices to a computer.

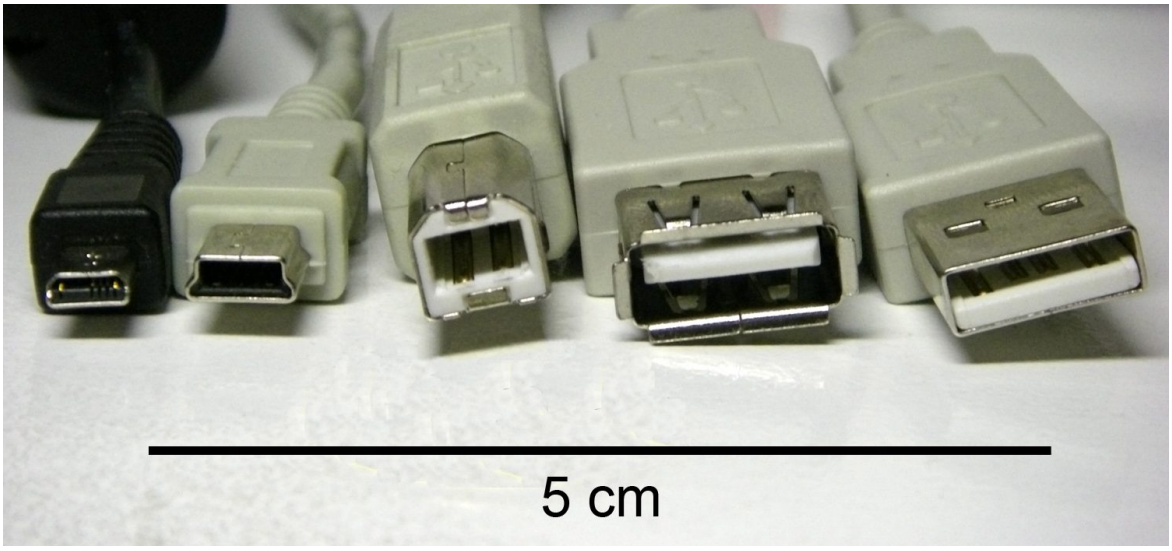
Benefits:

- the same connector is used by many devices (one standard);
- the device is detected automatically and the correct driver is loaded;
- it carries power, so it can charge or run a device;
- the connector only fits one way, so it is hard to plug in wrongly.

Drawbacks:

- the cable can only be a limited length (a few metres);

- older USB versions are slower than newer ones;
- transfer speeds can be lower than some other connection types.



A few of the USB connector shapes. All use the same USB standard, so devices are detected automatically.

Why we check for errors

While data travels, **interference** 干扰 can change it. Three things can go wrong:

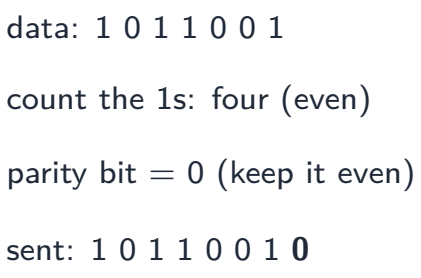
- **data loss** — some bits do not arrive;
- **data gain** — extra bits are added;
- **data change** — some bits are flipped.

So the receiver checks whether the data arrived correctly.

Error detection methods

Parity check

A **parity check** 奇偶校验 adds one extra bit, the **parity bit** 奇偶校验位, to each byte. There are two types:



A parity bit is added to make the number of 1s even (or odd)

- **even parity** — the total number of 1s in the byte (including the parity bit) must be **even**;
- **odd parity** — the total number of 1s must be **odd**.

Example (even parity): the data `1011001` has four 1s, which is already even, so the parity bit is `0`:

```
parity bit + data
    0        1011001
```

To find an error: the receiver counts the 1s. If even parity was agreed but a byte arrives with an **odd** number of 1s, an error has happened during transmission.

A parity check cannot find every error. If two bits flip, the count may still look correct.

Worked example. A single parity bit tells you *that* an error happened, but not *where*. A **parity block** 奇偶校验块 can find the exact bit. Several bytes are sent together, each with a row parity bit, and one extra **parity byte** 奇偶校验字节 at the bottom holds a parity bit for each column. Even parity is used below, and one bit was flipped during transmission. Find it.

```
          b1 b2 b3 b4 b5 | parity
 byte 1    1  0  1  1  0 |   1
 byte 2    0  1  1  0  1 |   1
```

```
byte 3    1  1  1  1  0 |   1    <- row: odd number of 1s
byte 4    0  0  1  1  0 |   0
------------------------------
parity    0  0  1  1  1 |   1
               ^ column b3: odd number of 1s
```

Row *byte 3* breaks even parity, and column *b3* breaks even parity. The flipped bit is where they cross: **byte 3, bit b3**. Changing it back from 1 to 0 repairs the data. A single parity bit only says an error happened; a parity block says exactly where, so the receiver can even fix it.

Checksum

A **checksum** 校验和 is a value worked out from all the data before sending. The receiver works out the checksum again from the data it received. If the two values do not match, an error has occurred and the data is resent.

Echo check

In an **echo check** 回送校验, the receiver sends the data back to the sender. The sender compares it with the original. If they differ, an error happened. (This needs the data to be sent twice, so it is slow.)

Automatic Repeat reQuest (ARQ)

Automatic Repeat reQuest 自动重传请求 (ARQ) uses messages called **acknowledgements** 确认.

- The receiver sends a **positive acknowledgement** if a packet arrived correctly, or a **negative acknowledgement** if it did not.
- The sender starts a **timeout** 超时 timer. If no positive acknowledgement comes back in time, the sender sends the packet again.

Check digit

A **check digit** 校验码 is an extra digit placed at the end of a number, worked out from the other digits. It is used to find mistakes when a number is **typed in**.

When the number is entered, the check digit is calculated again and compared. It can catch:

- an **incorrect digit** entered,
- a **transposition error** 换位错误 (two digits swapped, e.g. 21 typed as 12),
- a digit **left out or an extra digit** added,

- a **phonetic error** (a digit that sounds similar, e.g. 13 and 30).

Check digits are used in **barcodes** 条形码 and ISBNs (book numbers).

Encryption

Why we need encryption

Encryption 加密 scrambles data so that it cannot be understood if it is **intercepted** 拦截 (caught) by the wrong person. The readable data is called **plaintext** 明文; after encryption it becomes **ciphertext** 密文.

Encryption does **not** stop data from being intercepted. It only stops the data from being understood.

Symmetric encryption

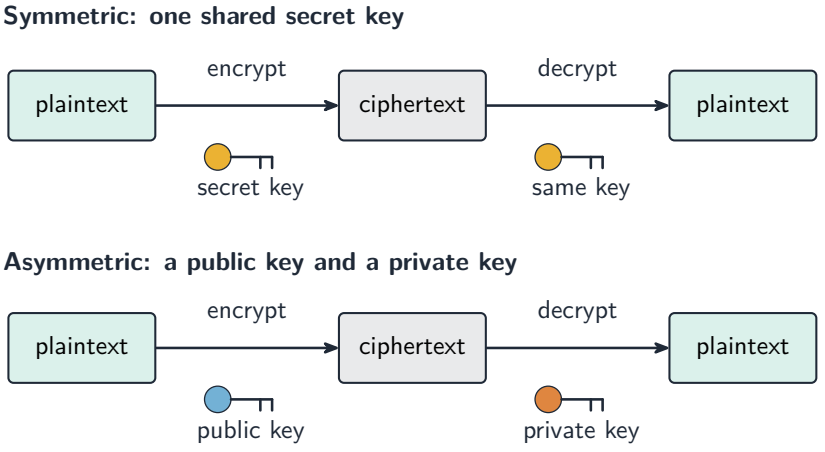
Symmetric encryption 对称加密 uses a single **key** 密钥 to both encrypt and decrypt the data. The sender and receiver must share this same secret key. The risk is that the key itself could be stolen while being shared.

Asymmetric encryption

Asymmetric encryption 非对称加密 uses two different keys:

- a **public key** 公钥 that anyone can have, used to encrypt;
- a **private key** 私钥 that is kept secret, used to decrypt.

Data encrypted with the public key can only be decrypted with the matching private key, so the key never has to be shared. This is safer than sharing one secret key.



Symmetric uses one shared key; asymmetric uses a public key to encrypt and a private key to decrypt

Exam tips

- Data is sent in **packets** (a header with the addresses and packet number, a payload, and a trailer). Packets can take different routes and arrive out of order, then are reordered by their packet numbers.
- Match the transmission types: **serial** (one bit at a time, good over long distances) vs **parallel** (several bits at once, short distances); **simplex / half-duplex / full-duplex** describe the direction.
- A single **parity bit** only shows *that* an error happened; a **parity block** locates the exact wrong bit — the row and the column that both fail.
- Learn the error-detection methods and what each does: parity check, checksum, echo check, ARQ, and the check digit (for typed-in numbers).
- **Symmetric** encryption uses one shared key; **asymmetric** uses a public key to encrypt and a private key to decrypt, so no secret key is ever shared.

Hardware

IGCSE Computer Science

The central processing unit (CPU)

The **central processing unit** 中央处理器 (CPU) is the part of the computer that processes data and carries out **instructions** 指令. It runs programs by repeating a cycle, millions of times each second.

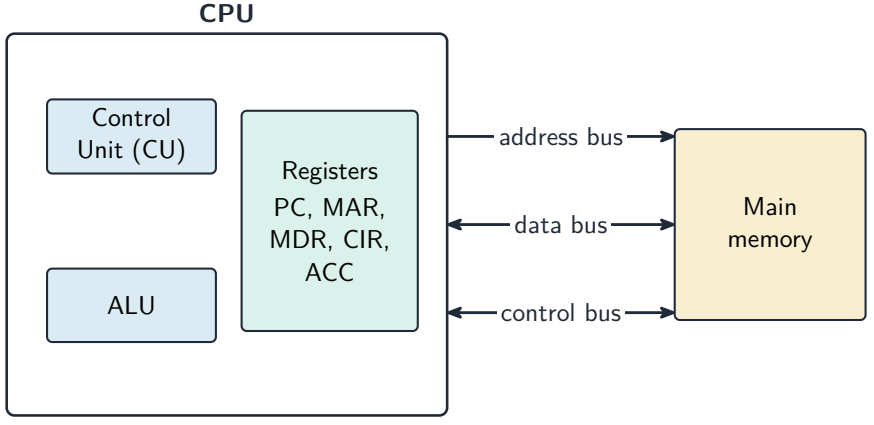
A central processing unit (CPU): the chip that processes data and runs the instructions

Parts of the CPU

Part	What it does
arithmetic logic unit 算术逻辑单元 (ALU)	does calculations (add, subtract) and logical operations (compare)
control unit 控制单元 (CU)	sends control signals to manage all the parts and tell them what to do
registers 寄存器	very small, very fast stores that hold one piece of data at a time

The parts are joined by **buses** 总线 — sets of wires that carry information:

- the **address bus** 地址总线 carries memory addresses (one direction);
- the **data bus** 数据总线 carries data and instructions (both directions);
- the **control bus** 控制总线 carries control signals.



The CPU holds the control unit, the ALU and the registers; three buses link it to main memory

Special registers

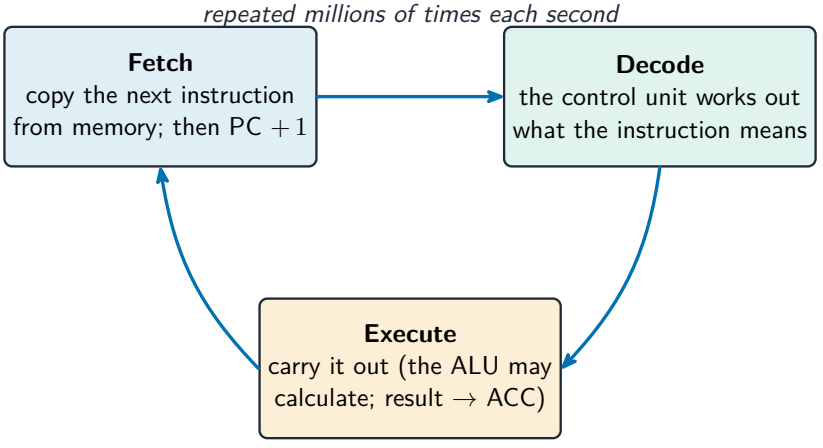
These registers are used during the fetch–execute cycle.

Register	Purpose
program counter 程序计数器 (PC)	holds the address of the next instruction
memory address register 内存地址寄存器 (MAR)	holds the address to read from or write to
memory data register 内存数据寄存器 (MDR)	holds the data or instruction just fetched
current instruction register 当前指令寄存器 (CIR)	holds the instruction now being carried out
accumulator 累加器 (ACC)	holds the result of the ALU's calculations

The fetch–execute cycle

The **fetch–execute cycle** 取指执行周期 has three stages, repeated again and again:

1. **Fetch** — the instruction is copied from memory 内存 into the CPU (using the PC, MAR and MDR). The PC then increases by 1.
2. **Decode** — the control unit works out what the instruction means.
3. **Execute** — the instruction is carried out (the ALU may do a calculation, with the result going to the accumulator).



The processor repeats three stages: fetch the instruction, decode it, then execute it

Cache memory

Cache 高速缓存 is a small amount of very fast memory inside or close to the CPU. It stores data and instructions that the CPU uses often. The CPU can read from cache much faster than from main memory, so the computer runs faster.

What affects CPU performance

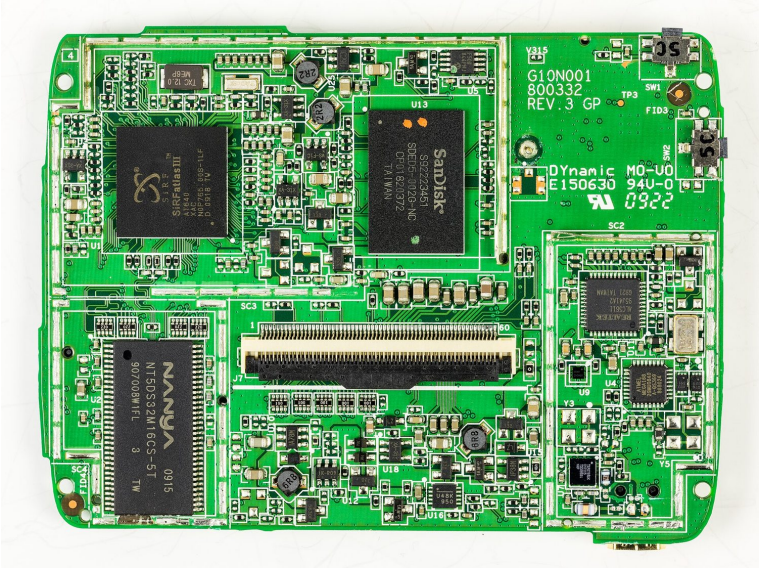
Feature	Effect
number of **cores** 核心	each core can run instructions on its own, so more cores can do more work at the same time
cache size	a larger cache holds more data ready for the CPU, so it waits less
clock speed 时钟速度	the number of cycles per second; a higher clock speed runs more instructions per second

Worked example. Two computers are identical except that A has a 3.0 GHz single-core CPU with 2 MB of cache, and B has a 2.4 GHz quad-core CPU with 8 MB of cache. Which is better for video editing, and why? Video editing splits into many tasks that can run at the same time, so B's **four cores** can process several streams in parallel while A can only work on one at a time. B's larger **cache** also keeps more

frequently used data close to the CPU, so it has to fetch from RAM less often. B is the better choice, even though A has the higher clock speed. Never answer on clock speed alone: weigh **cores, cache and clock speed** together, and tie each one to the job being done.

Embedded systems

An **embedded system** 嵌入式系统 is a small computer built inside a larger device to do one fixed job. It is dedicated to that task, is usually small and low-cost, and has no general operating system. Examples: a washing machine, a microwave, a set-top box.



The mainboard from inside a satnav: an embedded system is a small dedicated computer like this, built into a device to do one fixed job

Input devices

An input device sends data *into* the computer.

Device	How it works
barcode scanner 条形码扫描器	shines light at the bars and reads the pattern reflected back
QR code scanner 二维码扫描器	a camera reads a square pattern of black-and-white blocks
keyboard	pressing a key sends a code for that character
optical mouse 光电鼠标	a light and a sensor track movement across a surface
microphone	turns sound waves into an electrical signal
digital camera	a lens focuses light onto a sensor that records pixels
sensors 传感器	measure a physical quantity (such as temperature or light) and send it as data
touch screen 触摸屏	detects where your finger touches the screen

A keyboard: each key press sends a code for that character

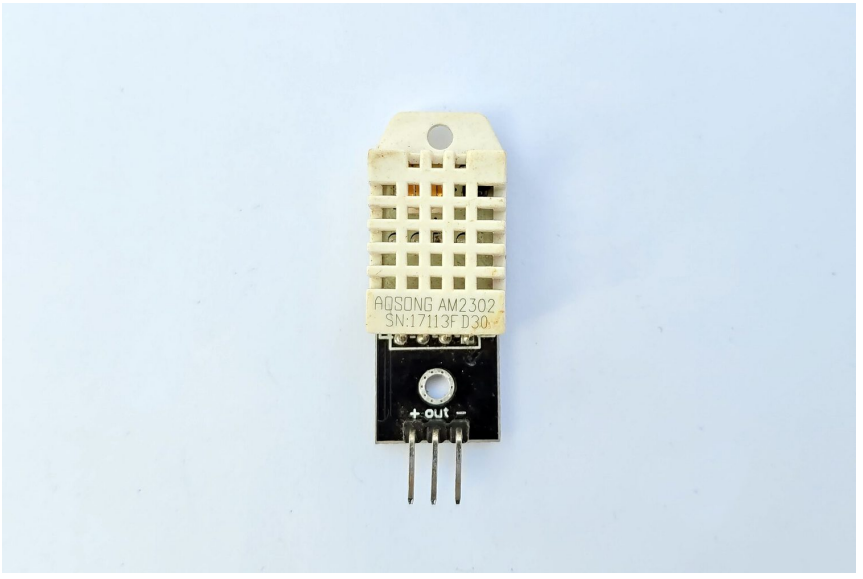
An optical mouse: a light and a sensor track its movement on the desk

A flatbed scanner: it shines light across a page to copy it into the computer

A barcode scanner reads the pattern of bars and sends it to the computer

A digital camera: a lens focuses light onto a sensor that records the pixels

A microphone turns sound waves into an electrical signal



A sensor measures a physical quantity (here, temperature) and sends it as data

A touch screen can work in three ways:

- **resistive** 电阻式 — two layers are pressed together where you touch;
- **capacitive** 电容式 — it senses the tiny electric charge of your finger;
- **infrared** 红外线 — your finger breaks a grid of light beams.

A touch screen senses where your finger touches it

Output devices

An output device sends data *out* of the computer to the user.

Device	How it works
actuator 执行器	turns an electrical signal into movement (e.g. a motor or valve)
LCD/LED screen	shows images using a grid of tiny coloured dots
projector 投影仪	shines an enlarged image onto a wall or screen
inkjet printer 喷墨打印机	sprays tiny drops of ink onto paper
laser printer 激光打印机	uses a laser and powder (toner) fixed onto paper by heat
3D printer	builds a solid object layer by layer
speaker 扬声器	turns an electrical signal into sound

An inkjet printer sprays tiny drops of ink onto the paper

A laser printer uses a laser and powder (toner) fixed onto paper by heat

A 3D printer builds a solid object layer by layer

An LCD/LED monitor shows images using a grid of tiny coloured dots

A speaker turns an electrical signal into sound

Headphones turn an electrical signal into sound for one listener

An actuator such as a motor turns an electrical signal into movement

Primary storage: RAM and ROM

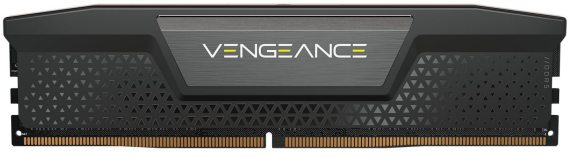
Primary storage 主存储器 is memory the CPU can use directly. There are two kinds.

RAM	ROM
volatile	non-volatile
can change	cannot change
running programs	startup (BIOS)

RAM is volatile and holds running programs; ROM is non-volatile and read-only

	RAM	**ROM**
Full name	**random access memory** 随机存取存储器	**read only memory** 只读存储器
Read/write	read and write	read only
Keeps data without power?	no — it is **volatile** 易失性	yes — it is **non-volatile** 非易失性
Holds	programs and data in use now	start-up instructions (how to boot the computer)

Both are needed: ROM tells the computer how to start, then RAM holds the programs you run.



A stick of RAM (random access memory): it holds the programs and data in use now

Secondary storage

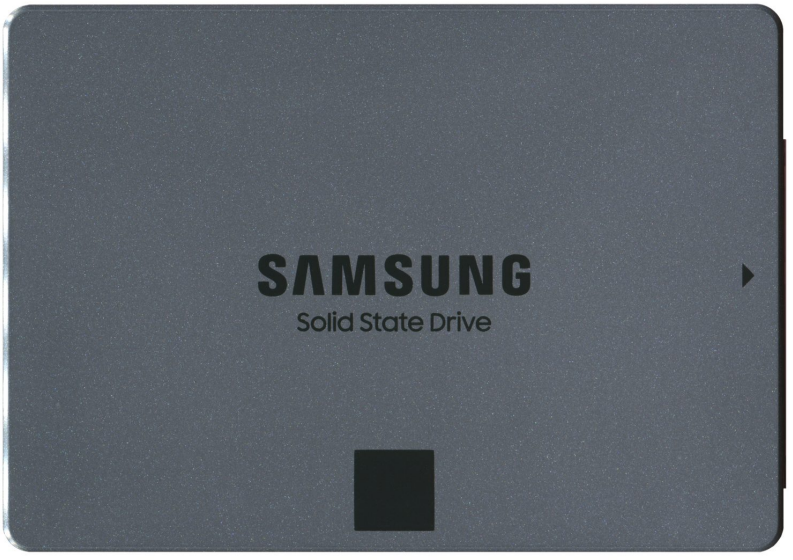
Secondary storage 辅助存储器 keeps data permanently, even when the power is off. It is non-volatile and is used for long-term storage. There are three types.

Type	How it stores data	Examples
magnetic storage 磁存储	data is stored as magnetised spots on spinning disks	hard disk drive (HDD), magnetic tape
optical storage 光存储	data is stored as marks read by a laser	CD, DVD, Blu-ray
solid state storage 固态存储	data is stored in **flash memory** 闪存, with no moving parts	SSD, USB flash drive, memory card

Inside a hard disk drive (HDD): the shiny disk spins and the arm reads the magnetised data

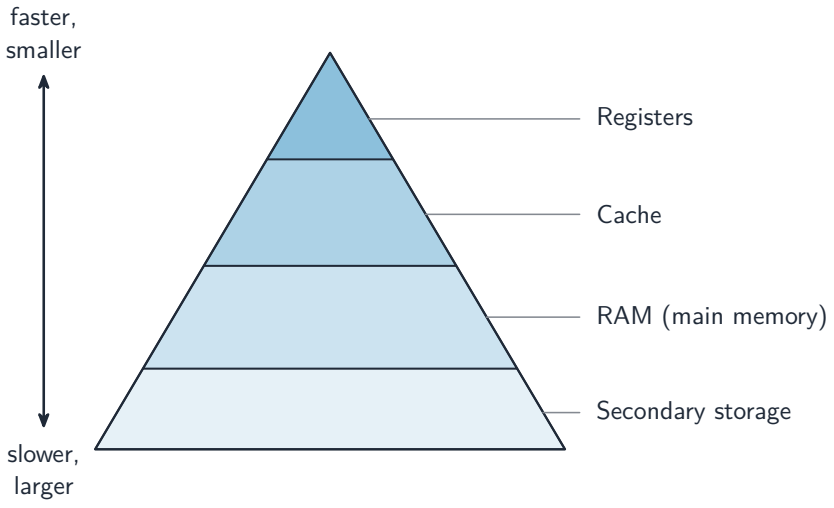
Magnetic tape stores data on a long magnetic strip; it is cheap for large backups

An optical disc such as a CD or DVD: a laser reads the marks on its shiny surface



A solid state drive (SSD) stores data in flash memory chips and has no moving parts

A USB flash drive stores data in flash memory and has no moving parts

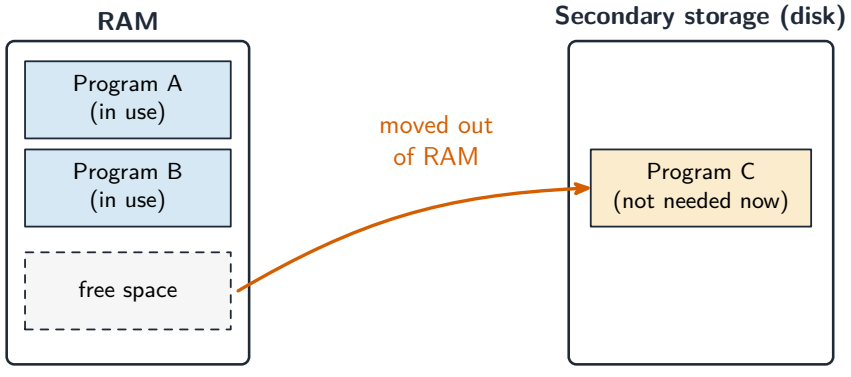


Storage forms a hierarchy: registers and cache are fastest but smallest; secondary storage is slowest but largest

Virtual memory

When the RAM becomes full, the computer can use part of the secondary storage as extra, pretend RAM. This is called **virtual memory** 虚拟内存.

Data that is not needed right now is moved out of RAM onto the disk, which frees space in RAM for other programs. This lets you run more programs than the RAM alone could hold. It is slower, because secondary storage is much slower than RAM.



This frees RAM, so more programs can run than RAM alone could hold — but it is slower, because the disk is much slower than RAM.

When RAM is full, data not in use is moved to the disk to free RAM — slower, but it lets more programs run

Cloud storage

Cloud storage 云存储 keeps your data on remote physical servers, reached over the internet, instead of on your own device.

- **Advantages**: reach it from anywhere on any device; no local storage hardware to buy or maintain; easy to share files and to back up automatically.
- **Disadvantages**: needs a working internet connection; an ongoing subscription cost; security and privacy depend on the provider; you rely on that provider staying available and in business.

Data-centre server racks: cloud storage keeps your files on remote servers you reach over the network

Network hardware

Network interface card (NIC)

A **network interface card** 网络接口卡 (NIC) is the hardware that lets a device join a network and send and receive data. Each NIC has a built-in MAC address.

MAC address and IP address

- A **MAC address** (media access control 介质访问控制 address) is a number that identifies one physical device. The maker sets it, and it does not normally change. It is written in hexadecimal.
- An **IP address** (internet protocol 网际协议 address) is a number that identifies a device on a network. It is given by the network and can change.

So a MAC address stays with the device, while an IP address depends on the network the device is using.

There are two versions of IP address. **IPv4** is 32-bit, written as four denary numbers 0–255 (for example `192.168.0.1`); the newer **IPv6** is 128-bit, written as eight groups of hexadecimal separated by colons, giving vastly more addresses. An address is also either **static** 静态 (fixed, set by hand and never changing) or **dynamic** 动态 (handed out by the router/network each time the device connects, so it can change).

Router

A **router** 路由器 connects different networks together — for example, your home network to the internet. It reads the IP address in each packet and forwards it towards the right network.

A Wi-Fi router: network hardware forwards packets between devices and the wider Internet

Exam tips

- Learn the **fetch–decode–execute** cycle and the special registers: the **program counter** holds the address of the *next* instruction, the MAR/MDR carry the address/data, and the **accumulator** holds the ALU's result.
- CPU performance improves with more **cores**, a larger **cache**, and a higher **clock speed**.
- **RAM** is volatile (it loses its data with no power) and holds programs in use; **ROM** is non-volatile and read-only (the start-up instructions).

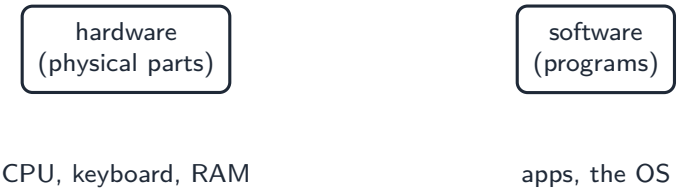
- Match the secondary-storage types: **magnetic** (HDD, tape), **optical** (CD/DVD, read by a laser), **solid state** (SSD, USB, flash — no moving parts).
- A **MAC address** identifies the physical device and does not normally change; an **IP address** is given by the network and can change.

Software

IGCSE Computer Science

Hardware and software

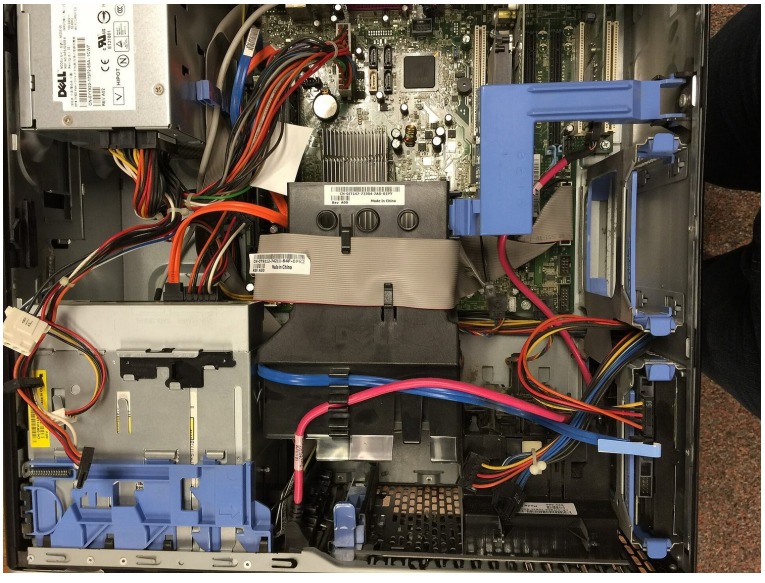
A computer system is made of two parts that work together.



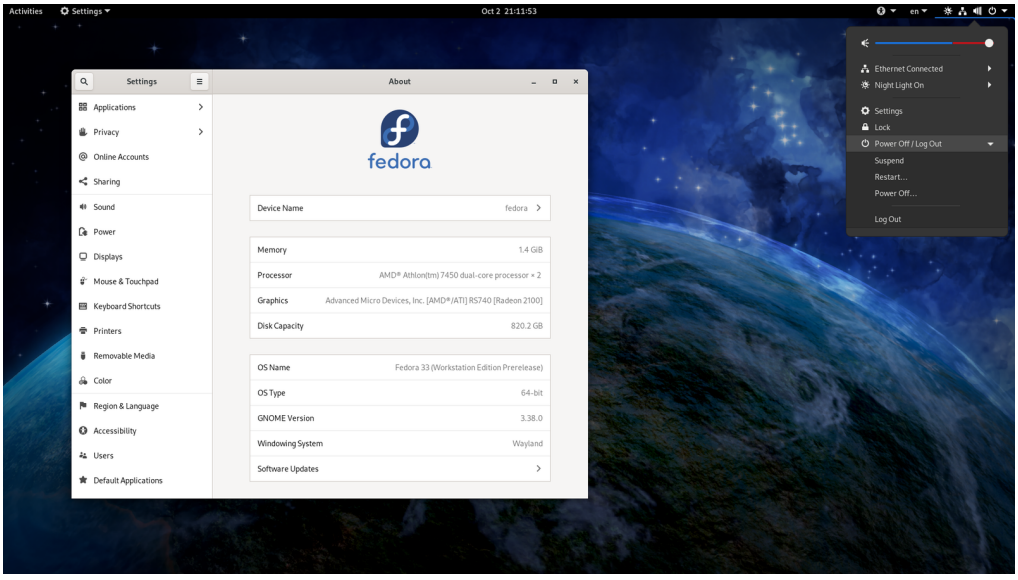
Hardware is the physical parts; software is the programs

- **Hardware** 硬件 is the **physical** 物理的 parts of the computer — the parts you can touch. Examples: the CPU, memory, keyboard, screen and hard disk.
- **Software** 软件 is the **programs** 程序 that control the computer and tell the hardware what to do.

Hardware cannot do anything useful on its own. Software needs hardware to run on. Both are needed.



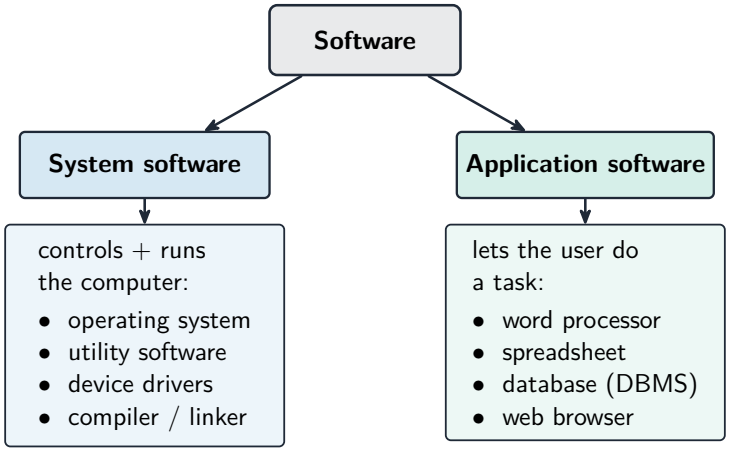
The motherboard and the parts attached to it are hardware — the physical parts you can touch



An operating system is software — the programs that tell the hardware what to do

Types of software

Software is split into two types.



Software divides into system software (runs the computer) and application software (lets the user do a task)

System software

System software 系统软件 controls the computer itself and gives a base for other programs to run on. Examples:

- **operating system** 操作系统 (see below);
- **compiler** 编译器 and **linker** 链接器 (tools that turn programs into a form the computer can run);
- **device driver** 设备驱动程序 — software that lets the operating system control a piece of hardware;
- **utility software** 实用程序 — small tools that look after the computer (for example anti-virus, backup, disk clean-up).

Below the operating system sits **firmware** 固件 – software stored permanently on ROM or flash that runs the moment the computer is switched on. The **BIOS** and the **bootloader** 引导程序 are firmware: they test the hardware and then load the operating system. So the layers stack up **hardware → firmware → operating system → application software**, each running on the layer below.

Application software

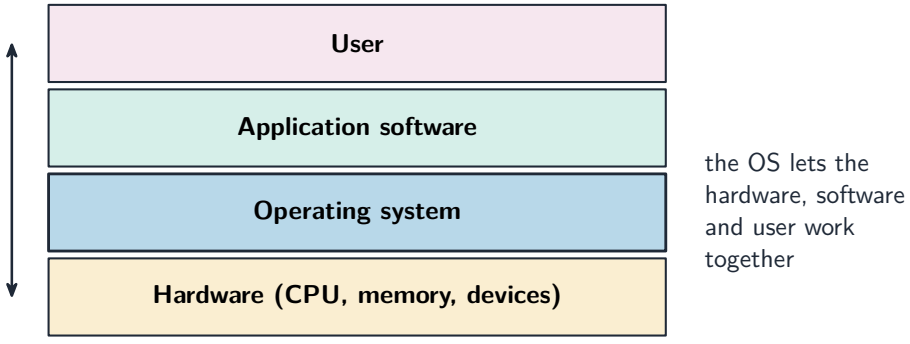
Application software 应用软件 lets the user do a useful task. Examples:

- **word processing** 文字处理 software (writing documents);
- **spreadsheet** 电子表格 software (working with numbers in a grid);
- **database management system** 数据库管理系统 (storing and searching data);
- **web browser** 网页浏览器 (viewing web pages).

	System software	**Application software**
Job	runs and manages the computer	helps the user do a task
Used by	the computer (in the background)	the user (directly)
Examples	operating system, drivers, utilities	word processor, browser, games

The operating system

An **operating system** (OS) is the main system software. It controls the whole computer and lets the hardware, the application software and the user work together. Without an OS the computer is very hard to use.



The operating system sits between the hardware and the application software and user, letting them work together

The OS has many roles:

- **managing files** — saving, opening, naming, moving and deleting files;
- **handling interrupts** (see below);
- **providing a user interface** 用户界面 so the user can control the computer;
- **managing memory** — deciding what is kept in RAM and where;
- **managing peripherals** 外围设备 and their drivers — controlling devices like printers;
- **managing multitasking** 多任务处理 — letting several programs run at once;
- **providing a platform** 平台 for running application software;
- **managing security** — user accounts, passwords and access rights.

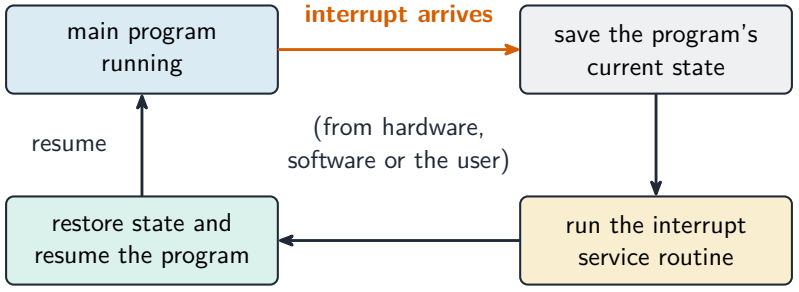
Interrupts

An **interrupt** 中断 is a signal sent to the **processor** 处理器 to tell it that something needs attention now.

An interrupt can come from three sources:

- **hardware** — for example a key is pressed, or a printer runs out of paper;
- **software** — for example a program tries to divide by zero;
- **the user** — for example the user presses a “cancel” key.

When an interrupt arrives, the processor **stops** what it is doing, saves its place, and **services** (deals with) the interrupt. When that is done, it goes back to what it was doing before. This lets the computer react quickly to important events.



On an interrupt the CPU saves its place, runs the interrupt service routine, then restores its place and carries on

Programming languages

All computers process data using the **central processing unit** 中央处理器 (CPU). To make the CPU do work, people write **programs** in a **programming language** 编程语言. There are two kinds.

High-level languages

A **high-level language** 高级语言 is written in words that look a bit like English (for example Python). It is:

- **easy** for people to read, write and fix;
- **independent of the computer** — the same program can run on different types of computer.

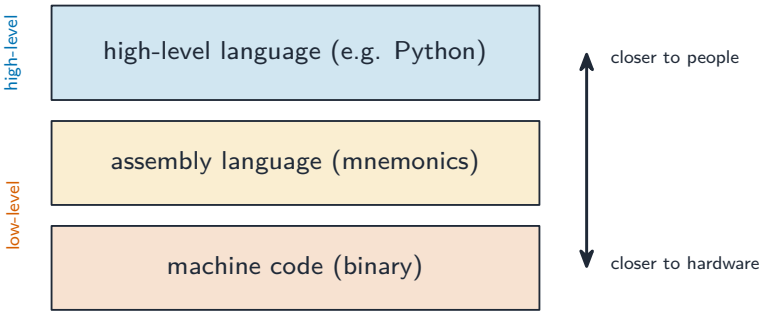
Low-level languages

A **low-level language** 低级语言 is close to what the hardware actually uses. It includes:

- **machine code** 机器码 — instructions written in binary (0s and 1s), which the CPU runs directly;
- **assembly language** 汇编语言 — machine code written with short word codes (mnemonics) instead of binary.

A low-level language relates to the **specific** 特定的 type of CPU, so a program written for one CPU may not run on another. It is used when the programmer needs full control

of the hardware or very fast, small code.



High-level languages are close to human language and run on any computer; low-level languages (assembly and machine code) are close to the hardware and tied to one type of CPU

Translators

The CPU can only run machine code. A high-level program must first be changed into machine code. The software that does this is a **translator** 翻译程序. There are three types: a compiler and an interpreter (for high-level code), and an assembler (for low-level code).

Compiler

A **compiler** translates the **whole** program into machine code **before** it is run. After translating, the machine code can be run many times without translating again.

- translation happens once, in full;
- it reports all the errors together, at the end of translating;
- the finished program runs fast and does not need the compiler to run.

Interpreter

An **interpreter** 解释器 translates and runs the program **one line at a time**.

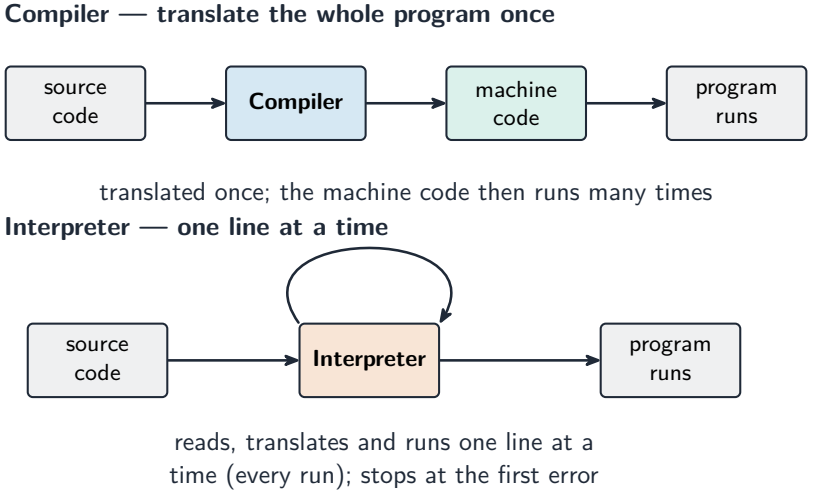
- it stops at the **first error** it finds, which helps when testing;
- the program runs more slowly, because it is translated each time it runs;
- the interpreter must be present every time the program runs.

	Compiler	**Interpreter**
Translates	the whole program at once	one line at a time
Errors	all reported at the end	stops at the first error
Speed of finished program	faster	slower
Needed to run later?	no	yes

Use a **compiler** when you want to share a fast, finished program. Use an **interpreter** when you are writing and testing a program and want to find errors easily.

Assembler

An **assembler** 汇编器 translates a **low-level assembly-language** program (written in mnemonics such as `LDA`, `ADD`) into machine code. Each mnemonic becomes **one** machine instruction – a direct one-to-one translation, unlike a compiler or interpreter, which work on high-level code.



A compiler translates the whole program once into machine code; an interpreter translates and runs one line at a time

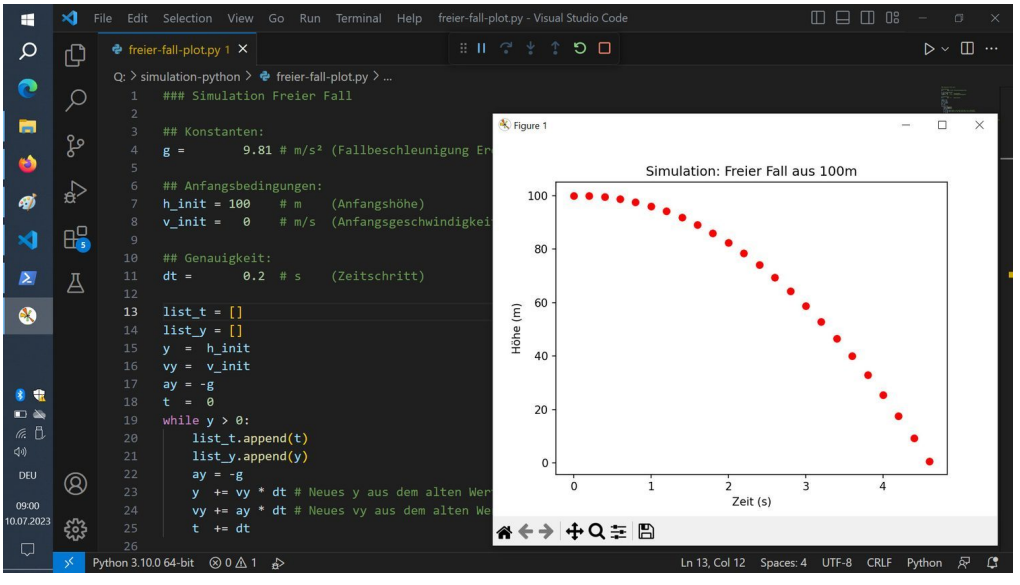
Worked example. A team is writing a program. While developing it they want to find and fix errors quickly; when they ship it they want customers to run it fast

without seeing the source code. Which translator suits each stage? While developing, use an **interpreter**: it translates and runs one line at a time and stops at the first error, so a mistake is easy to locate. To ship, use a **compiler**: it translates the whole program once into machine code, which then runs quickly with no translator present, and the customer receives only the executable rather than the source. The two are not rivals - name the **stage** and give the reason, because "a compiler is faster" on its own earns nothing.

Integrated development environment (IDE)

An **integrated development environment** 集成开发环境 (IDE) is software that helps you write programs. It puts many useful tools in one place:

- a **code editor** 代码编辑器 — for typing in your program;
- a **run time environment** 运行时环境 — for running the program to test it;
- a **translator** — a compiler or interpreter to turn your code into machine code;
- **error diagnostics** 错误诊断 — messages that help you find and understand mistakes;
- **auto-completion** 自动补全 and **auto-correction** 自动纠错 — the IDE finishes or fixes code as you type;
- **prettyprinting** 美化打印 — laying out the code neatly with colour and indentation so it is easy to read.



Source code in an editor: an IDE combines editor, debugger and run tools in one place

Exam tips

- **Hardware** is the physical parts; **software** is the programs. **System software** runs the computer (OS, drivers, utilities); **application software** does a task for the user.
- A **compiler** translates the whole program at once (the finished program runs fast, and all errors are reported at the end); an **interpreter** translates line by line (it stops at the first error, runs slower, and is needed every time).
- **High-level** languages are close to English and run on any computer; **low-level** languages (assembly, machine code) are tied to one type of CPU.
- An **interrupt** is a signal that makes the CPU stop, save its place, service the event, then carry on where it left off.
- Learn the operating system's roles: managing files, memory, peripherals, multitasking, security and the user interface.

The internet and its uses

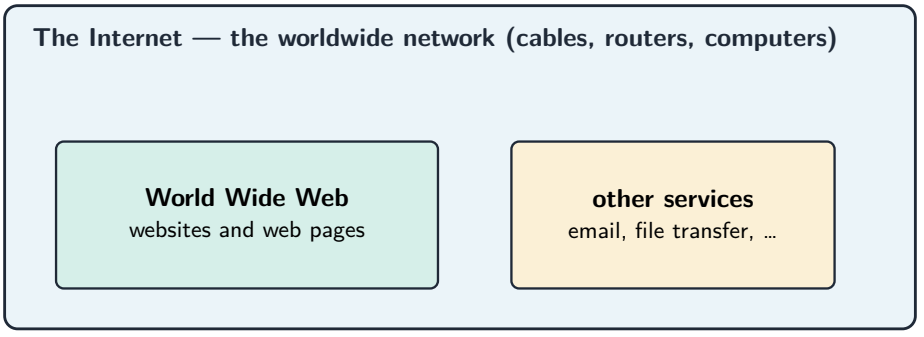
IGCSE Computer Science

The internet and the world wide web

People often mix up these two terms, but they are not the same.

- The **internet** 互联网 is the **infrastructure** 基础设施 — the huge worldwide network of computers and the cables, routers and connections that join them.
- The **world wide web** 万维网 (WWW) is a collection of **websites** 网站 and web pages that you view using the internet.

So the internet is the network; the web is one of the things you use *on* that network.

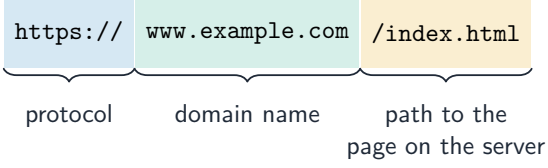


The internet is the worldwide network; the world wide web is one of the things you use on it

A router joins your home network to the internet and sends data to the right place

URL

A **uniform resource locator** 统一资源定位符 (URL) is a text-based address for a web page. You type it into a **web browser** 网页浏览器 to visit a page, for example `https://www.example.com/index.html`.



A URL has three parts: the protocol, the domain name, and the path to the page on the server

The web browser

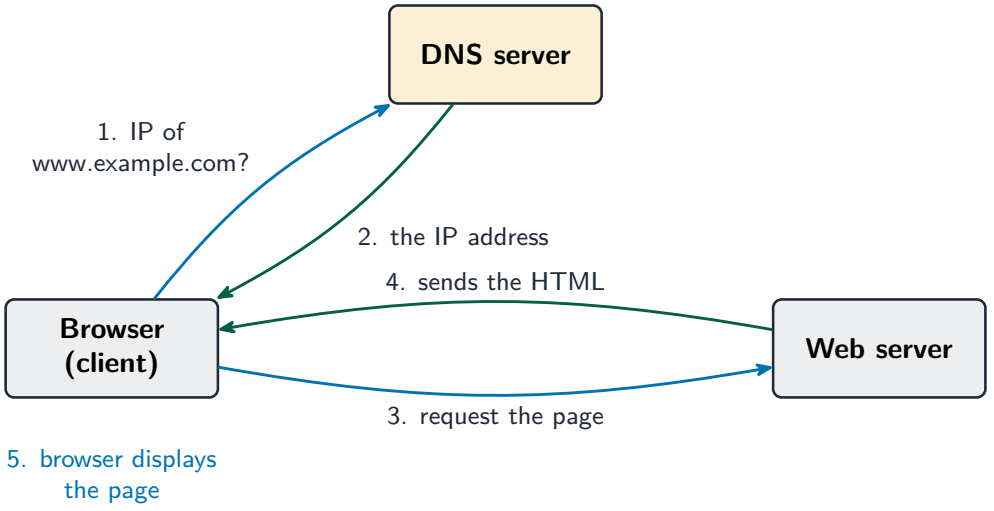
A web browser is the program you use to view web pages. Its role includes:

- **rendering** 渲染 the **hypertext markup language** 超文本标记语言 (HTML) — turning the page's code into what you see;
- **displaying web pages** on the screen;
- **managing how a web page is presented** — its layout, text and images.

How a web page reaches you

When you type a URL and press enter, several steps happen:

1. The browser needs the website's **IP address** 网际协议地址, but a URL uses a name, not a number.
2. The browser asks a **domain name service** 域名服务 (DNS) — a system that stores the IP address for each web address.
3. The DNS finds the matching IP address and sends it back to the browser.
4. The browser uses the IP address to contact the **web server** 网页服务器 that stores the page.
5. The web server sends the page's HTML back to the browser.
6. The browser turns the HTML into the page and displays it.



The browser asks a DNS for the site's IP address, then requests the page from the web server, which returns the HTML

If the DNS does not have the address, it asks another DNS server until the address is found.

A data centre full of servers: powerful computers that store websites and send them to you

Worked example. Break down the URL `https://www.example.com/books/index.html`. `https` is the **protocol**, the set of rules used to transfer the page (the `s` means the transfer is encrypted). `www.example.com` is the **domain name**, which identifies the web server and which a DNS server turns into an IP address. `/books/index.html` is the **file path and file name** of the resource stored on that server. Name each part together with its **job**: an answer that says "the first bit" and "the last bit" describes the URL without ever explaining it.

Cookies

A **cookie** is a small text file that a website stores on your device. It lets the website remember things about you. There are two types.

a cookie

a small text file

stored on your computer

remembers settings / login

A cookie is a small text file stored on your computer

Type	Kept when you close the browser?	Used for
session 会话 cookie	no — deleted when you close the browser	short-term data, e.g. items in a shopping basket
persistent 持久 cookie	yes — saved on the device	remembering you between visits

Cookies are used to:

- **save personal details** so you do not type them again (such as login or address);
- **track user preferences** 用户偏好, such as your language or the things you like, so the site can suggest content.

Digital currency

A **digital currency** 数字货币 is money that exists only in **electronic form** 电子形式. There are no coins or notes. It is stored and moved between people using computers, and can be used to pay for things online.

A problem with digital money is trust: how do you stop someone spending the same money twice, or changing the records? Blockchain solves this.

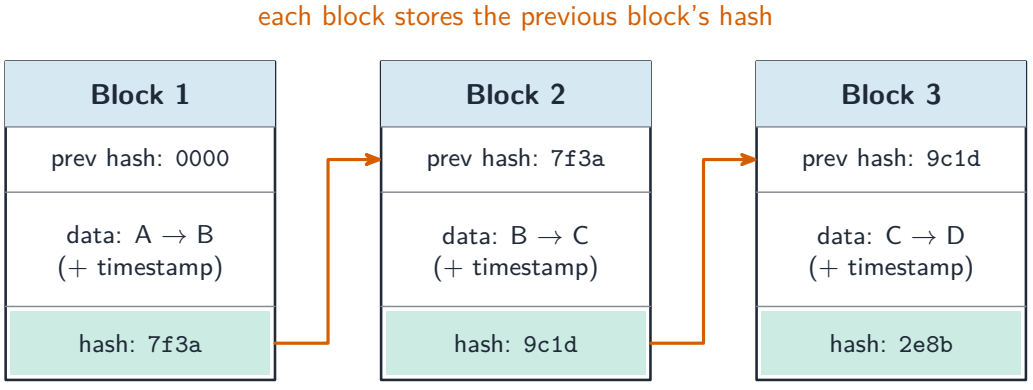
Blockchain

A **blockchain** 区块链 is a **digital ledger** 数字账本 — a shared record of every **transaction** 交易. The record is copied across many computers, so no single person controls it and it is very hard to change.

The transactions are kept in blocks joined in a chain. Each block holds:

- the **data** of the transaction (who paid whom, and how much);
- a **timestamp** 时间戳 (the date and time);
- a **hash value** 哈希值 — a special code worked out from the block's contents.

Each block also stores the hash of the block before it. If someone changes an old block, its hash changes, so it no longer matches the next block. This breaks the chain and the change is spotted at once. This is how blockchain tracks transactions safely.



Each block stores the previous block's hash, chaining the blocks; changing one block breaks the chain

Cyber security threats

Cyber security 网络安全 means keeping computers, networks and data safe from attack. You must know these threats.

Threat	What it is
brute force attack 暴力破解	trying many passwords very quickly until the right one is found
data interception 数据拦截	"listening in" to steal data while it travels across a network
distributed denial of service 分布式拒绝服务 (DDoS) attack	flooding a server with so many requests that it cannot respond, so the website goes down
hacking 黑客攻击	gaining access to a computer system without permission
malware 恶意软件	harmful software (see the table below)
pharming 域名欺骗	secret code sends you to a fake website even when you type the correct address
phishing 网络钓鱼	fake emails or messages that trick you into giving away private details
social engineering 社会工程	tricking a person (not a computer) into breaking security, e.g. pretending to be the boss

Types of malware

Malware is any software made to harm a computer. The main types are:

Malware	What it does
virus 病毒	attaches to a file and copies itself when the file is opened; can delete or damage data
worm 蠕虫	copies itself across a network on its own, without needing a file to be opened
Trojan horse 特洛伊木马	pretends to be useful software, but harms the computer once installed
spyware 间谍软件	secretly records what you do, such as the keys you press
adware 广告软件	floods you with unwanted adverts
ransomware 勒索软件	locks your files and demands payment to unlock them

Impact of the threats

These attacks can **steal personal data**, **delete or change files**, **stop a service from working**, cost money, and make users lose trust in a company. For example, a DDoS attack can make an online shop unusable, so it loses sales.

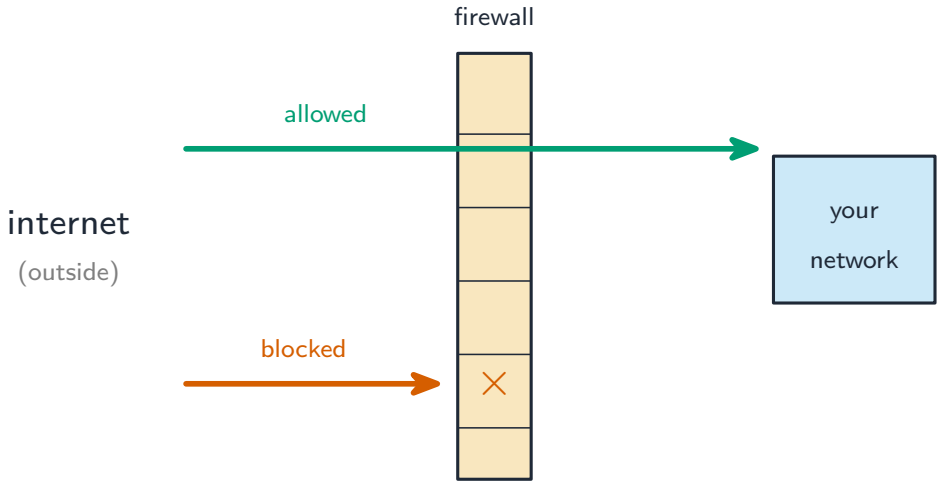
A padlock: cyber security locks data and systems against malware, phishing and unauthorised access

Keeping data safe

You can protect data using these methods.

- **access levels** 访问权限 — each user can only see or change the data they need, nothing more.
- **anti-malware** 反恶意软件 software, including **anti-virus** 杀毒软件 and **anti-spyware** 反间谍软件 — scans for and removes harmful software.
- **authentication** 身份验证 — proving who you are before you get access. Methods include:
  - **biometrics** 生物识别 (fingerprint or face),
  - a **password** 密码,
  - **two-step verification** 两步验证 (a code sent to your phone as well as a password).
- **automating software updates** — updates fix weak points (bugs) in software automatically.
- **checking the spelling and tone** of messages — bad spelling or an odd, urgent tone can show a message is fake (phishing).

- **checking the URL attached to a link** — a wrong or strange web address warns you the link is unsafe.
- **firewall** 防火墙 — checks data going in and out of a network and blocks anything not allowed.
- **privacy settings** 隐私设置 — control who can see your information online.



A firewall sits between the outside internet and your network, checking each connection and blocking any traffic that is not allowed

- **proxy server** 代理服务器 — sits between the user and the internet, hiding the user's IP address and filtering out unsafe content.

Exam tips

- The **internet** is the worldwide network (the infrastructure); the **world wide web** is the pages you view on it — do not mix the two up.
- Learn how a page loads: the browser asks a **DNS** for the site's IP address, then requests the page from the **web server**, which returns the HTML.
- Match the malware types: **virus** (attaches to a file), **worm** (spreads by itself across a network), **Trojan horse** (pretends to be useful), **spyware**, **ransomware**.
- Match each threat to a defence: phishing/pharming → check the URL; brute force → strong passwords + two-step verification; interception → encryption; hacking/DDoS → a firewall.
- A **session** cookie is deleted when you close the browser; a **persistent** cookie is saved to remember you between visits.

Automated and emerging technologies

IGCSE Computer Science

Automated systems

An **automated system** 自动化系统 is a mix of software and hardware that senses and responds to data in its **environment** 环境, with no need for **human intervention** 人工干预. In other words, it works on its own.

Examples of automated systems:

- a central heating 中央供暖 system;
- a chemical process 化学过程 in a factory;
- a **greenhouse** 温室 (for growing plants);
- a **car park barrier** 停车场道闸.

Sensors, microprocessors and actuators

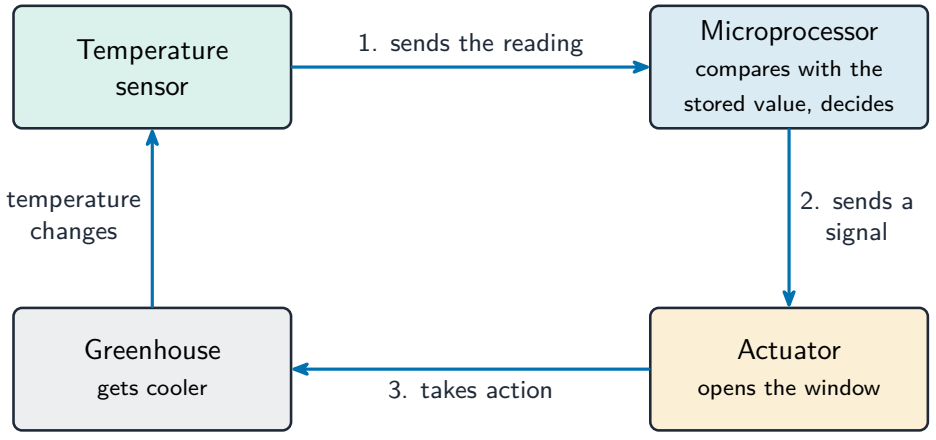
Three parts work together in an automated system.

Part	Job
sensor 传感器	measures a physical quantity (such as temperature or light) and sends the data
microprocessor 微处理器	compares the data with stored values and makes decisions
actuator 执行器	receives a signal and causes movement or action (such as opening a valve)

The cycle works like this:

1. The sensor sends data to the microprocessor.
2. The microprocessor compares this data with **stored values** 存储值 and makes a decision.
3. The microprocessor sends signals to the actuators to take action.

For example, in a greenhouse: a temperature sensor reads the heat; the microprocessor compares it with the wanted value; if it is too hot, the microprocessor signals an actuator to open a window.



A sensor feeds the microprocessor, which compares the reading with a stored value and signals an actuator; the loop then repeats

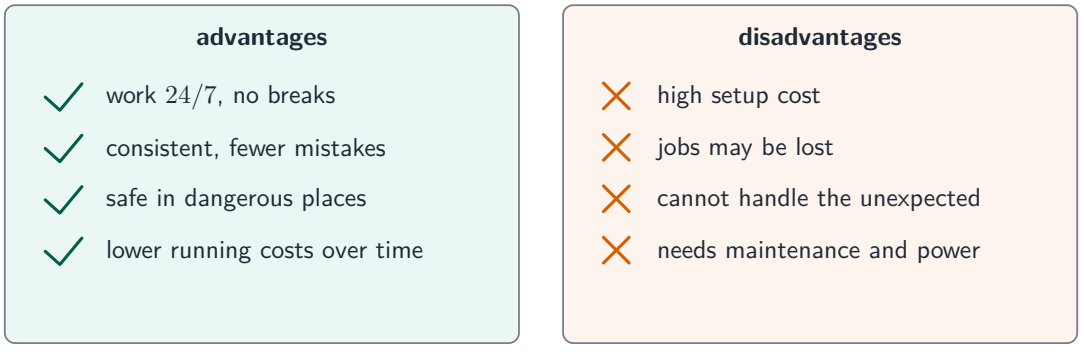
The syllabus names many **sensor types** 传感器类型, each reading one physical quantity – for example an **accelerometer** 加速度计 (movement or tilt), a **humidity** 湿度 sensor (water in the air), and a **proximity** 接近 sensor (a nearby object):

Sensor	Measures	Typical use
temperature	heat	ovens, greenhouses, heating
light	brightness	automatic lights, cameras
pressure	force on a surface	alarm floor mats, touchscreens
infra-red / proximity	a nearby object	automatic doors, parking sensors
acoustic (sound)	sound level	noise monitors
humidity / moisture	water in air or soil	greenhouses, irrigation
gas	a gas being present	smoke and CO alarms
pH	acidity	pools, fish tanks
accelerometer	movement or tilt	phones, airbag triggers
magnetic field	magnetism	door contacts, compasses
flow / level	fluid flow or depth	pipes, tanks

To pick a sensor for a scenario, choose the one that reads the **right physical quantity**: a fish tank needs pH and temperature sensors; an automatic door needs an infra-red / proximity sensor.

Advantages and disadvantages

Automated systems are used in many situations, such as industry, transport, agriculture 农业 (farming), weather (gathering data), gaming and lighting.



The advantages and disadvantages of automated systems

A camera drone: it can fly itself along a set route to collect data

Advantages	Disadvantages
work all day and night, without rest	cost a lot to buy and set up
faster and more **consistent** 一致的 than people	can break down and need expert repair
can work in places unsafe for people	may replace people's jobs
fewer human mistakes	cannot easily react to a situation they were not built for

Worked example. Describe how an automatic greenhouse holds the temperature at 25 °C. A **temperature sensor** continuously measures the temperature and sends its reading, converted to digital by an ADC, to the **microprocessor**. The microprocessor **compares** the reading with the stored value of 25 °C. If the temperature is above it, the microprocessor signals an **actuator** to open a window or switch on a fan; if it is below, it switches on a heater. The whole loop then **repeats continuously**. Two marks nearly always sit on the words "compares with a stored (pre-set) value" and "the process repeats" - a description that stops at "the sensor tells the computer" leaves them behind.

Robotics

Robotics 机器人技术 is the branch of technology that deals with the design, building and operation of **robots** 机器人.

A robot has these characteristics:

- a **physical structure** 物理结构 (a mechanical 机械的 body, such as arms);
- **electrical components** 电子元件 (motors, sensors and wiring);
- **programmable instructions** 可编程指令 — it follows a program that can be changed.

What robots do

Robots can perform many roles, for example:

- **factory equipment** 工厂设备 — building cars, welding, moving heavy parts;
- **domestic appliances** 家用电器 — such as a robotic **vacuum cleaner** 吸尘器.

An industrial robot arm. It follows a program and can be reprogrammed to do a new job.

A robot vacuum cleaner: a home robot that drives itself around to clean the floor

Advantages	Disadvantages
do dull or dangerous jobs	expensive to buy and maintain
work quickly and accurately	can replace human workers
do not get tired or bored	a robot has a **lack of independent decision-making** 缺乏独立决策 — it only does what it is programmed to do

A key limit of a robot is that it cannot think for itself. It cannot make its own choices when something unexpected happens.

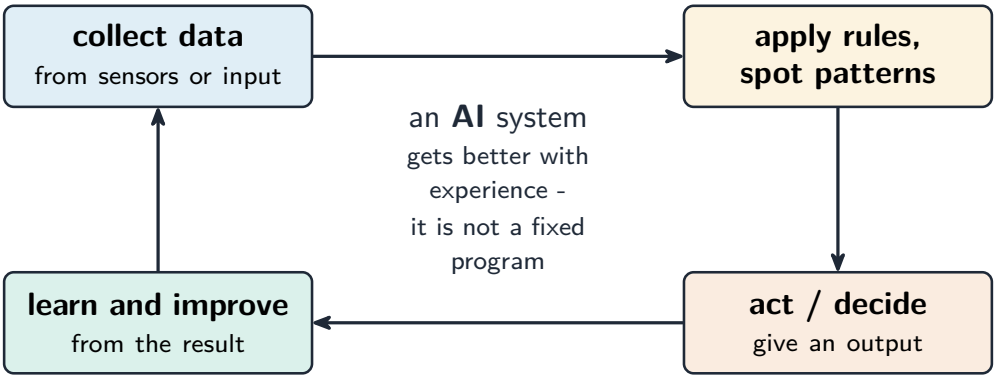
Artificial intelligence

Artificial intelligence 人工智能 (AI) is the **simulation** 模拟 of human intelligence by computer systems. This means a computer doing tasks that normally need human thinking.

Characteristics of AI

AI systems usually have these features:

- they **collect data** and the **rules** 规则 for using that data;
- they have the ability to **reason** 推理 (work things out using the rules);
- they have the ability to **draw conclusions** 得出结论, which may be **approximate** 近似的 (a best guess) or **definite** 确定的 (certain);
- they have the ability to **learn** 学习 from data;
- they have the ability to **adapt** 适应 — to change their behaviour as they get new data.



An AI system collects data, applies rules, acts, then learns and improves

Examples of AI

- **expert systems** 专家系统 — software that gives advice like a human expert (for example, helping a doctor with a diagnosis);
- **natural language processing** 自然语言处理 — understanding human speech or text;
- **self-driving cars** 自动驾驶汽车.

A self-driving car: sensors on the car let a computer steer it with no driver

Inside an expert system

An **expert system** stores human expertise and reasons from it. It has four parts:

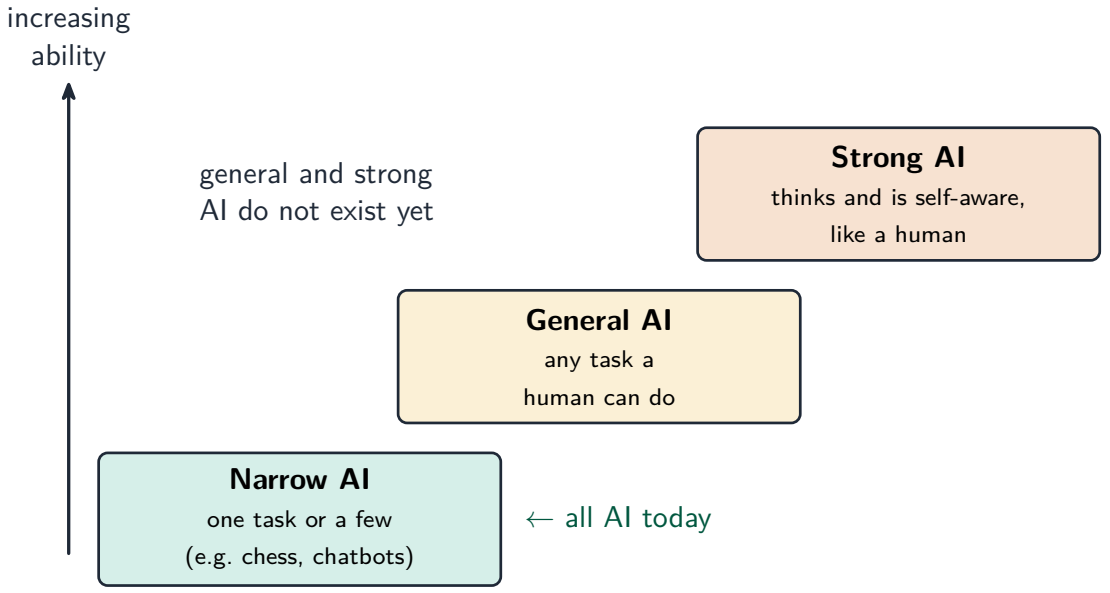
- a **knowledge base** 知识库 – the stored facts about the subject;
- a **rule base** 规则库 – the "if… then…" rules a human expert would apply;
- an **inference engine** 推理引擎 – applies the rules to the facts to reach a conclusion;
- a **user interface** 用户界面 – asks the user questions and shows the result.

The interface collects facts from the user; the inference engine runs the rule base against the knowledge base; and the system outputs a diagnosis or a probability – as used to help a doctor diagnose an illness or to find a car fault.

Machine learning

Machine learning 机器学习 is a program that improves its own performance from **experience**, without being explicitly reprogrammed. It finds patterns in data and adapts: a search engine gets better at ranking results, a voice assistant recognises spoken commands more accurately, and a robot vacuum gradually learns the layout of a room.

Narrow, general and strong AI



Narrow AI does one task (all AI today); general AI could do any human task; strong AI would be self-aware — neither exists yet

Type	What it means
narrow AI 弱人工智能	can do only one task or a small set of tasks (for example, a chess program). All AI today is narrow AI.
general AI 通用人工智能	could do any task a human can do, switching between many different tasks.
strong AI 强人工智能	would think and be aware like a real human mind. This does not exist yet.

Exam tips

- Learn the control loop: a **sensor** measures a quantity → the **microprocessor** compares it with a stored value → an **actuator** takes action; then the loop repeats.
- Do not confuse the parts: a **sensor** sends data *in*; an **actuator** causes movement *out*.
- A robot follows **programmable instructions** and cannot make its own decisions when something unexpected happens.

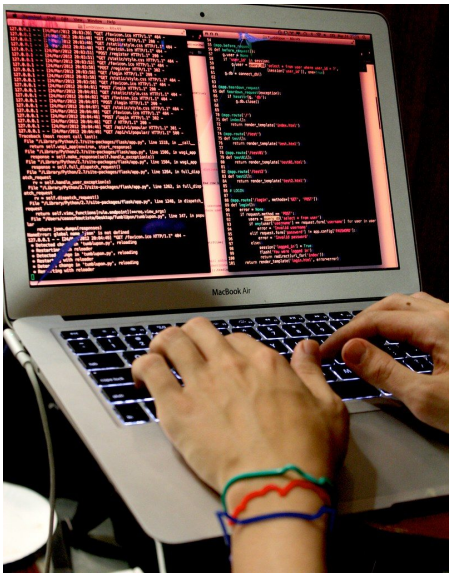
- **AI** simulates human thinking: it collects data and rules, reasons, draws conclusions, and can learn and adapt.
- Know the AI levels: **narrow** (one task; all AI today), **general** (any human task), **strong** (self-aware; does not exist yet).

Topic 07

IGCSE Computer Science

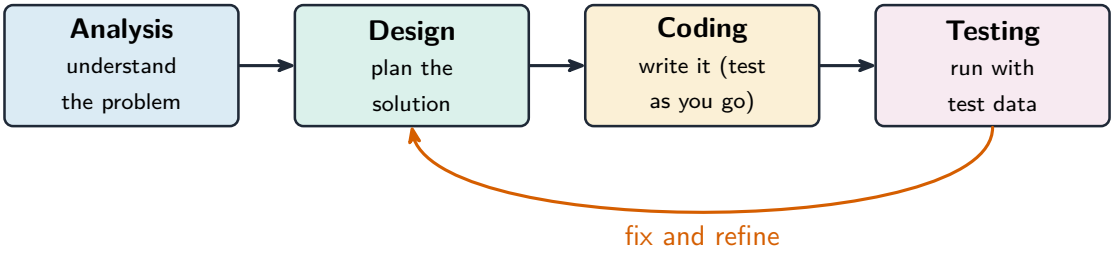
The program development life cycle

The **program development life cycle** 程序开发生命周期 is the set of stages used to make a program. There are four stages.

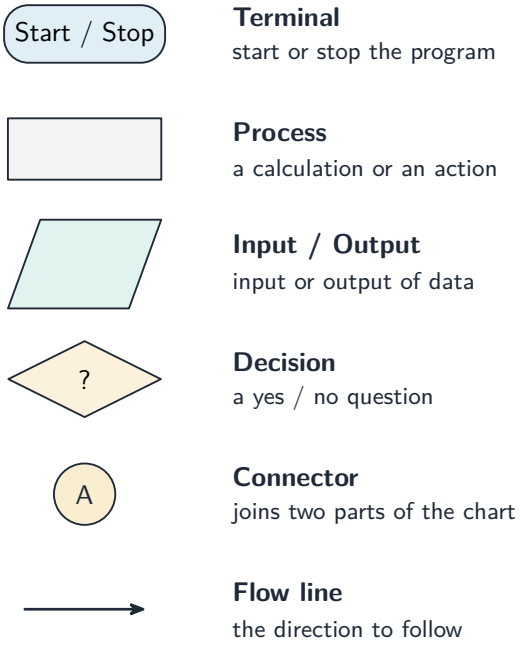


Software is written by programmers, who follow the development life cycle

Stage	What you do
analysis 分析	study the problem and work out what is needed
design 设计	plan how the program will work
coding 编码	write the program code and test it as you go
testing 测试	run the finished program with test data to find errors



The four stages of program development; testing feeds back to fix and refine the design



A program flowchart sets out the steps and decisions of a program during the design stage

Analysis

In analysis you understand the problem. Two key skills help:

- **abstraction** 抽象 — keep only the important details and ignore the rest;
- **decomposition** 分解 — break a big problem into smaller, easier parts.

Design

In design you plan the solution, often using decomposition. You can show the parts as **sub-systems** 子系统 in a **structure diagram** 结构图 (a chart that splits a system into smaller boxes).

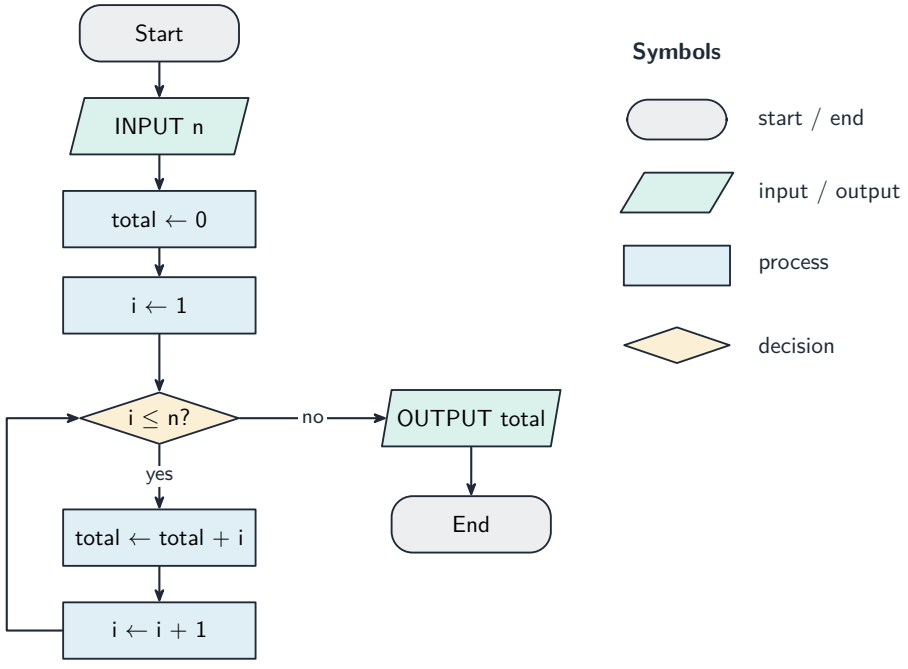
Coding and testing

In **coding** you write the program code. You use **iterative testing** 迭代测试 — test small parts again and again as you build them. In **testing** you run the whole program with **test data** 测试数据 to check it works.

Design tools

You can plan a solution in three main ways.

- a **structure diagram** — shows the parts of a system and how they fit together;
- a **flowchart** 流程图 — a diagram using boxes and arrows to show the steps in order;
- **pseudocode** 伪代码 — steps written in simple, code-like English (not a real language).



A flowchart for the sum algorithm, using the standard symbols (start/end, input/output, process, decision)

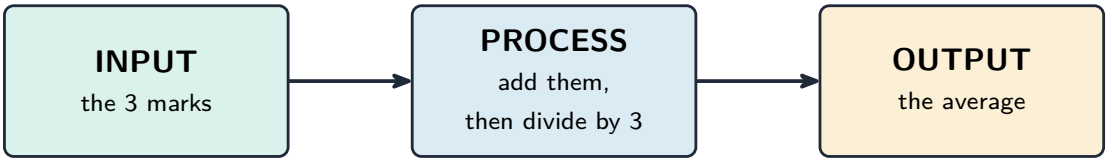
Algorithms

An **algorithm** 算法 is a set of steps, in the right order, that solves a problem. Every algorithm can be split into three parts:

- **input** 输入 — the data that goes in;
- **processing** 处理 — the work done on the data;
- **output** 输出 — the result that comes out.

This is called decomposition into inputs, processes and outputs. For example, for "find the average of three marks": the inputs are the three marks; the processing is adding

them and dividing by 3; the output is the average.



Every algorithm decomposes into input, processing and output — here, finding the average of three marks

Validation and verification

When data is entered, you check it to reduce mistakes.

Validation 验证 checks that the data is **sensible** and follows the rules. It cannot check that the data is true, only that it is allowed.

Validation check	What it checks
range check 范围检查	the value is between a lowest and highest allowed value
length check 长度检查	the number of characters is allowed (e.g. a password ≥ 8)
type check 类型检查	the data is the right type (e.g. a number, not letters)
presence check 存在性检查	something has actually been entered (not left blank)
format check 格式检查	the data is in the right pattern (e.g. a date as dd/mm/yyyy)
check digit 校验码	an extra digit confirms a number was entered correctly

Verification 核实 checks that data was **copied or entered correctly** (no mistakes while typing it in). Two methods:

- **visual check** 目视检查 — a person compares the typed data with the original;
- **double entry** 双重输入 — the data is entered twice and the two copies are compared.

Trace tables

A **trace table** 追踪表 records the value of each variable as an algorithm runs, step by step. It helps you:

count	total	output
1	5	
2	12	
3	20	20

A trace table records each variable's value as the program runs

- check that an algorithm works correctly;
- work out **what an algorithm does** by following it with given data.

Example: trace this algorithm with the input 5.

```
INPUT n
total ← 0
FOR i ← 1 TO n
    total ← total + i
NEXT i
OUTPUT total
```

i	total	OUTPUT
1	1	
2	3	
3	6	
4	10	
5	15	15

The trace shows the algorithm adds up 1 to n. With input 5 the output is 15.

Worked example. Trace this algorithm and give the output.

```
x ← 20
count ← 0
WHILE x > 1
    x ← x DIV 2
    count ← count + 1
```

```
ENDWHILE
OUTPUT count
```

`DIV` gives only the whole-number part of a division. Take one row per pass: x becomes 10 (count 1), then 5 (count 2), then 2 (count 3), then 1 (count 4). Now `x > 1` is false, so the loop stops and the output is **4**. Two habits protect these marks: test the condition **before** each pass rather than after, and write a new row for **every** pass - trying to hold the values in your head is what makes traces go wrong.

Test data

Test data is data you use to test a program. There are four types you must know.

Type	Meaning	Example (age 0–120 allowed)
normal 正常数据	sensible data that should be accepted	25
abnormal 异常数据	wrong data that should be rejected	-4 or "cat"
extreme 极端数据	the largest and smallest values still allowed	0 and 120
boundary 边界数据	the values on each side of a limit (one allowed, one not)	120 and 121

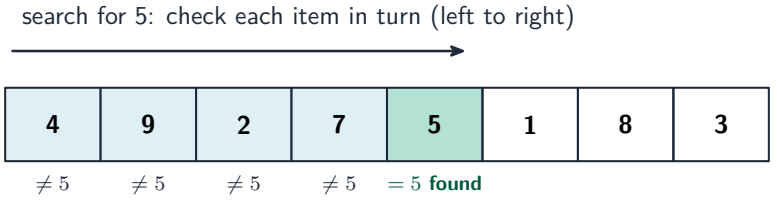
Standard methods of solution

You must know these common algorithms.

Linear search

A **linear search** 线性查找 checks each item in a list, one by one, until it finds the value it wants or reaches the end.

```
found ← FALSE
FOR i ← 0 TO 9
    IF list[i] = searchValue THEN
        found ← TRUE
    ENDIF
NEXT i
OUTPUT found
```

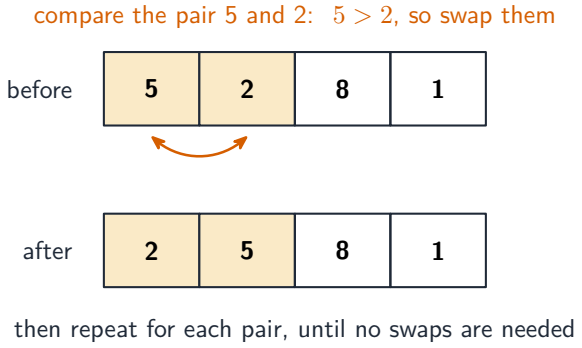


Linear search checks each item in turn from the start until it finds the value

Bubble sort

A **bubble sort** 冒泡排序 puts a list in order. It compares each pair of side-by-side items and swaps them if they are in the wrong order. It repeats this until no more swaps are needed.

```
FOR i ← 0 TO 8
    IF list[i] > list[i + 1] THEN
        temp ← list[i]
        list[i] ← list[i + 1]
        list[i + 1] ← temp
    ENDIF
NEXT i
```



Bubble sort compares each side-by-side pair and swaps them if they are out of order, repeating until sorted

Totalling and counting

- **totalling** 求和 — keep adding values to a running total (`total ← total + value`).
- **counting** 计数 — add 1 to a counter each time something happens (`count ← count + 1`).

Maximum, minimum and average

- to find the **maximum** 最大值: keep the largest value seen so far.
- to find the **minimum** 最小值: keep the smallest value seen so far.
- to find the **average** 平均值: divide the total by how many values there are.

```
total ← 0
FOR i ← 0 TO 9
    total ← total + list[i]
NEXT i
average ← total / 10
OUTPUT average
```

Exam tips

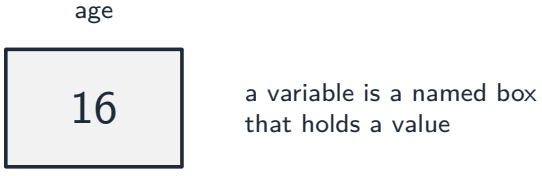
- Learn the four life-cycle stages: analysis → design → coding → testing. **Abstraction** keeps only the important details; **decomposition** breaks a problem into smaller parts.
- **Validation** checks data is *sensible* (range, length, type, presence, format checks); **verification** checks it was *copied correctly* (a visual check or double entry).
- Learn the four test-data types: **normal** (accepted), **abnormal** (rejected), **extreme** (the largest/smallest still allowed), **boundary** (the values either side of a limit).
- To work out what an algorithm does, fill in a **trace table** — write down every variable's value at each step.
- Know the standard algorithms: **linear search** (check each item in turn) and **bubble sort** (swap side-by-side pairs until no swaps are needed).

Programming

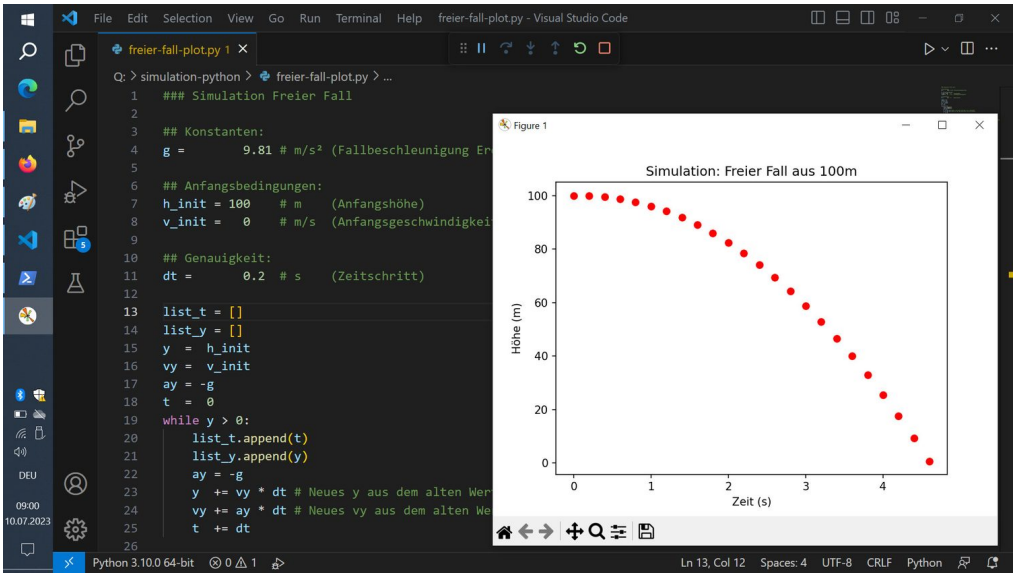
IGCSE Computer Science

Variables and constants

A **variable** 变量 is a named store that holds a value which **can change** while the program runs. A **constant** 常量 is a named store whose value is **fixed** and does not change.



A variable is a named store whose value can change



Variables and constants are named stores for values, set in the program's code

You should **declare** 声明 them (say their name and type) before use:

```
DECLARE score : INTEGER
DECLARE name : STRING
CONSTANT Pi = 3.142
```

While a program runs, its variables are held in the computer's memory (RAM)

Use a constant for a value that never changes (like Pi), so it is set in one place and is easy to read.

Data types

A **data type** 数据类型 says what kind of value a variable holds. You must know five basic types.

integer	whole number: 7
real	decimal: 3.5
char	one letter: 'A'
string	text: "hello"
boolean	true or false

The basic data types: integer, real, char, string and boolean

Type	Holds	Example
integer 整数	a whole number	`42`, `-7`
real 实数	a number with a decimal point	`3.14`, `-0.5`
char 字符	a single character	`'A'`, `'?'`
string 字符串	a sequence of characters	`"Hello"`
Boolean 布尔值	one of two values	`TRUE` or `FALSE`

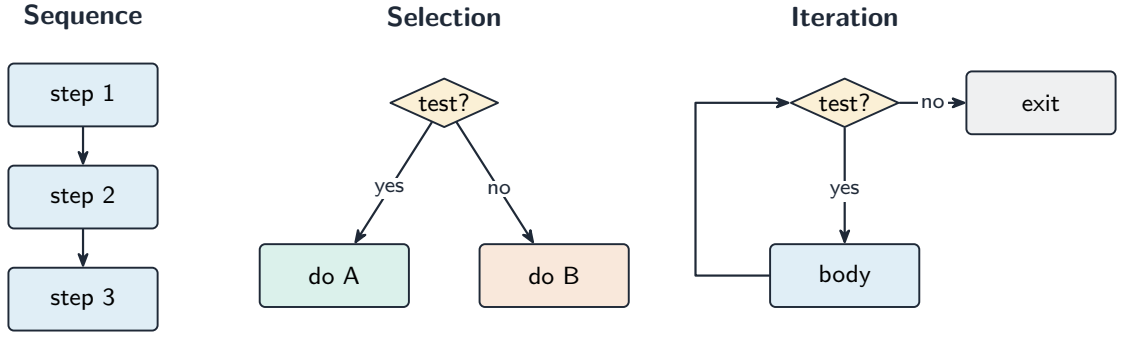
Input and output

Input 输入 reads a value from the user. **Output** 输出 shows a value on the screen.

```
OUTPUT "What is your name?"
INPUT name
OUTPUT "Hello ", name
```

The three basic structures

Every program is built from three control structures.



The three control structures: sequence runs steps in order, selection chooses a branch, iteration repeats a body

Sequence

Sequence 顺序 means the steps run one after another, in order, from top to bottom.

```
INPUT length
INPUT width
area ← length * width
OUTPUT area
```

Selection

Selection 选择 chooses which steps to run, based on a condition. Use an `IF` statement, or a `CASE` statement when there are many choices.

```
IF score >= 50 THEN
    OUTPUT "Pass"
ELSE
    OUTPUT "Fail"
ENDIF
```

```
CASE OF grade
    'A' : OUTPUT "Excellent"
    'B' : OUTPUT "Good"
    OTHERWISE : OUTPUT "Keep trying"
ENDCASE
```

Iteration

Iteration 迭代 (a loop 循环) repeats steps. There are three kinds.

A **count-controlled loop** 计数循环 repeats a fixed number of times:

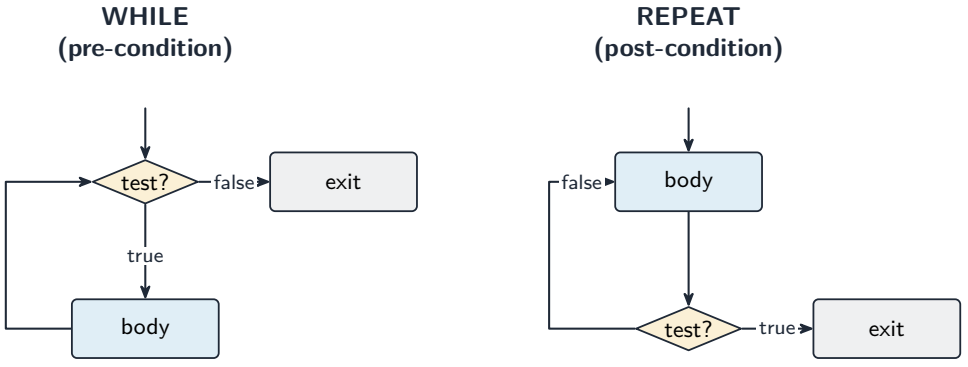
```
FOR i ← 1 TO 5
    OUTPUT "Hello"
NEXT i
```

A **pre-condition loop** 前测循环 checks the condition **before** each repeat, so it may run zero times:

```
WHILE answer <> "stop" DO
    INPUT answer
ENDWHILE
```

A **post-condition loop** 后测循环 checks the condition **after** each repeat, so it always runs **at least once**:

```
REPEAT
    INPUT password
UNTIL password = "secret"
```



tested first ⇒ may run **zero** times

tested last ⇒ runs **at least once**

A pre-condition (WHILE) loop tests before the body, so it may run zero times; a post-condition (REPEAT) loop tests after, so it runs at least once

Totalling and counting

- **totalling** 求和 — keep adding values to a running total: `total ← total + value`.
- **counting** 计数 — add 1 to a counter each time: `count ← count + 1`.

`count = count + 1`	count up by 1
`total = total + value`	add to a running total

A counter counts up by 1; a total builds a running sum

```
total <- 0
count <- 0
FOR each value (4, 7, 5)
  total <- total + value
  count <- count + 1
```

total keeps a running sum;
count adds 1 each pass -
a **trace table** follows both

value	total	count
start	0	0
4	4	1
7	11	2
5	16	3

A trace table follows a totalling and counting loop pass by pass

```
total ← 0
FOR i ← 1 TO 10
    INPUT mark
    total ← total + mark
NEXT i
OUTPUT total
```

Worked example. A program must read 5 marks, then output the total and the average.

```
total ← 0
FOR i ← 1 TO 5
    INPUT mark
    total ← total + mark
NEXT i
average ← total / 5
OUTPUT total, average
```

Two lines carry the marks. `total ← 0` must sit **before** the loop: put it inside and the total resets on every pass, so the program outputs only the last mark. And `average ← total / 5` must sit **after** the loop, because the total is not complete until every mark has been added. Initialise before, calculate after - that ordering is what the question is really testing.

Operators

Arithmetic operators

Operator	Meaning
`+ - * /`	add, subtract, multiply, divide
`^`	raised to the power of
`MOD`	the **remainder** 余数 after division
`DIV`	the whole-number part of a division

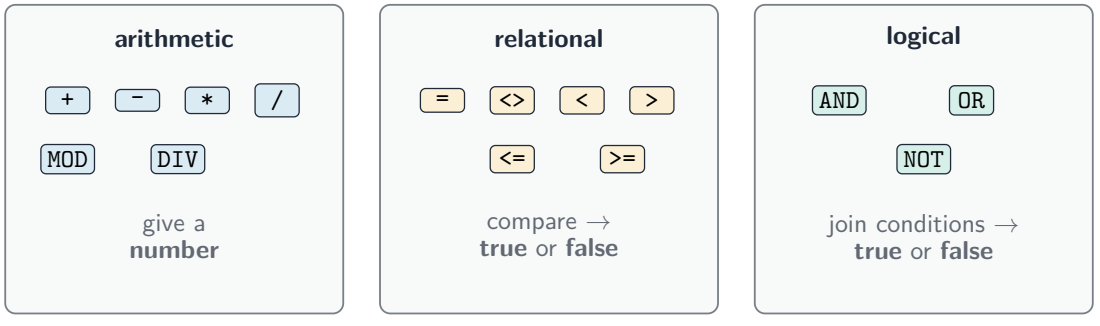
For example, `17 MOD 5` is `2`, and `17 DIV 5` is `3`.

Relational operators

These compare two values and give a Boolean result: `=`, `<`, `<=`, `>`, `>=`, and `<>` (not equal to).

Logical operators

These join conditions: `AND` (both must be true), `OR` (at least one must be true), `NOT` (reverses true/false).



The three families of operators: arithmetic, relational and logical

```
IF age >= 13 AND age <= 19 THEN
    OUTPUT "Teenager"
```

```
ENDIF
```

String handling

A string is made of characters. Useful operations:

- **length** 长度 — the number of characters in the string;
- **substring** 子串 — a smaller part taken from the string;
- **upper case** 大写 — change letters to capitals;
- **lower case** 小写 — change letters to small letters.

```
name ← "Computer"
OUTPUT LENGTH(name)          // 8
OUTPUT SUBSTRING(name, 1, 4) // "Comp"
OUTPUT UCASE(name)           // "COMPUTER"
OUTPUT LCASE(name)           // "computer"
```

(The first character may be counted as position 0 or position 1, depending on the language.)

Nested statements

A **nested statement** 嵌套语句 is one control structure placed **inside** another. You can nest selection and iteration.

```
FOR i ← 1 TO 3
    IF i MOD 2 = 0 THEN
        OUTPUT i, " is even"
    ELSE
        OUTPUT i, " is odd"
    ENDIF
NEXT i
```

You will not have to write more than three levels of nesting.

Procedures and functions

To avoid repeating code, you can break a program into named blocks.

- A **procedure** 过程 is a named block of code that does a task. You run it with `CALL`.

- A **function** 函数 is like a procedure, but it **returns a value** back to where it was called.

A **parameter** 参数 is a value passed into a procedure or function (up to three parameters).

```
PROCEDURE Greet(personName : STRING)
    OUTPUT "Hello ", personName
ENDPROCEDURE

CALL Greet("Sam")
```

```
FUNCTION Square(n : INTEGER) RETURNS INTEGER
    RETURN n * n
ENDFUNCTION

answer ← Square(5)   // answer is 25
```

Local and global variables

- A **local variable** 局部变量 is declared inside a procedure or function. It can only be used there.
- A **global variable** 全局变量 is declared in the main program. It can be used anywhere.

Local variables are safer, because they cannot be changed by mistake from another part of the program.

Library routines

A **library routine** 库程序 is a ready-made piece of code you can use. You must know these:

Routine	What it does
`MOD`	gives the remainder of a division
`DIV`	gives the whole-number part of a division
`ROUND`	rounds a real number to a number of decimal places
`RANDOM`	gives a random number

Writing a maintainable program

A **maintainable** 可维护的 program is easy for other people to read and change later. To make one:

- use **meaningful identifiers** 有意义的标识符 — clear names for variables, constants, arrays, procedures and functions (`totalScore`, not `x`);
- add **comments** 注释 to explain what parts of the code do;
- use procedures and functions to split the work into small blocks.

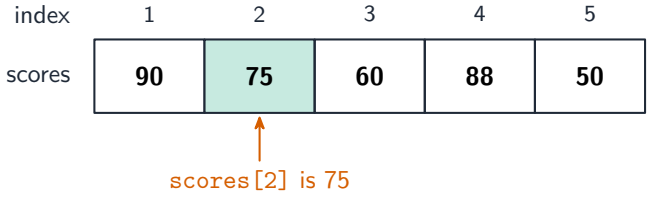
Arrays

An **array** 数组 is a single variable that holds **many values** of the same type, found by an **index** 索引 (a position number).

One-dimensional arrays

A **one-dimensional (1D) array** 一维数组 is like a single list.

```
DECLARE scores : ARRAY[1:5] OF INTEGER
scores[1] ← 90
scores[2] ← 75
OUTPUT scores[1]
```



A 1D array holds many values in one variable; each is found by its index

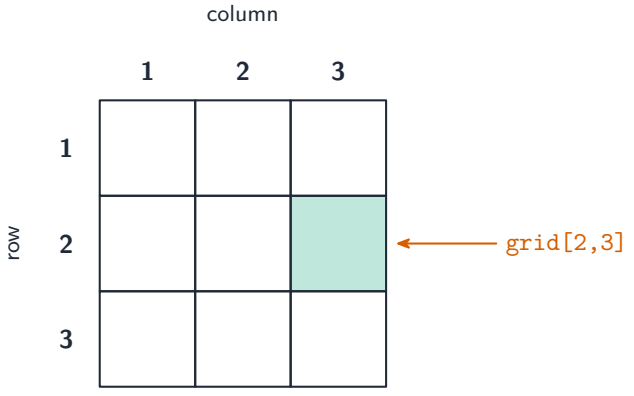
You can use a loop to fill or read an array:

```
FOR i ← 1 TO 5
    INPUT scores[i]
NEXT i
```

Two-dimensional arrays

A **two-dimensional (2D) array** 二维数组 is like a table with rows and columns. It uses two indexes.

```
DECLARE grid : ARRAY[1:3, 1:3] OF INTEGER
grid[1, 1] ← 5
grid[2, 3] ← 8
```



A 2D array is a table; each value is found by two indexes, [row, column]

You read a 2D array with a loop inside a loop (nested iteration).

File handling

A program can store data in a **file** 文件 so it is kept after the program stops. You must **open** the file, use it, then **close** it.

```
OPENFILE "data.txt" FOR WRITE
WRITEFILE "data.txt", "Hello"
CLOSEFILE "data.txt"

OPENFILE "data.txt" FOR READ
READFILE "data.txt", line
CLOSEFILE "data.txt"
```

You can read and write single items of data or a whole line of text. Always close a file when you have finished with it.

Exam tips

- A **variable** can change while the program runs; a **constant** is fixed. Learn the five data types: integer, real, char, string, Boolean.
- Every program is built from three structures: **sequence**, **selection** (IF / CASE), and **iteration** (loops).
- A **WHILE** loop tests *before* the body, so it may run zero times; a **REPEAT ... UNTIL** loop tests *after*, so it always runs at least once.
- `MOD` gives the remainder and `DIV` the whole-number part of a division: `17 MOD 5` is 2, `17 DIV 5` is 3.
- A **function** returns a value; a **procedure** does not. A **local** variable works only inside its block; a **global** one works anywhere.

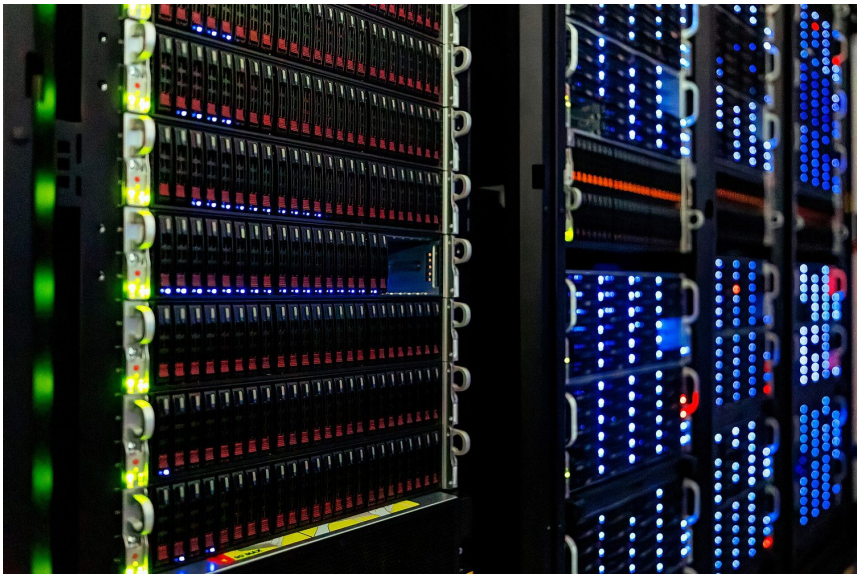
Databases

IGCSE Computer Science

What is a database?

A **database** 数据库 is an organised store of data, kept so that it is easy to search, sort and update. At IGCSE you work with a **single-table database** 单表数据库 — all the data is held in one table.

A library card catalogue is a database on paper — records you can search and sort by index



Large modern databases are stored on servers in data centres

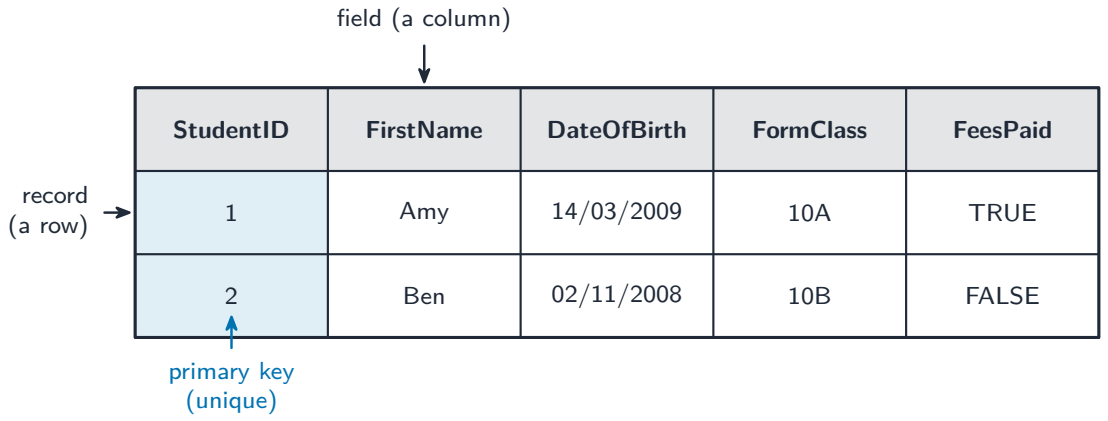
Records and fields

A database table is made of records and fields.

- A **record** 记录 is one row in the table — all the data about one thing (for example one student).
- A **field** 字段 is one column in the table — one piece of data that every record has (for example "First name").

StudentID	FirstName	DateOfBirth	FormClass	FeesPaid
1	Amy	14/03/2009	10A	TRUE
2	Ben	02/11/2008	10B	FALSE

Here each **row** is a record, and each **column** is a field.



Each row is a record and each column is a field; the primary key (StudentID) is unique for every record

Data types for fields

Each field stores one **data type** 数据类型. You choose the type that best fits the data.

text	a name
number	an age
date/time	a birthday
boolean	yes / no

Field types: text, number, date/time and boolean

Data type	Used for	Example
text/alphanumeric 文本	letters, digits and symbols	“10A”, “Amy”
character 字符	a single character	’M’
Boolean 布尔值	one of two values	TRUE / FALSE
integer 整数	a whole number	42
real 实数	a number with a decimal point	3.5
date/time 日期时间	a date or a time	14/03/2009

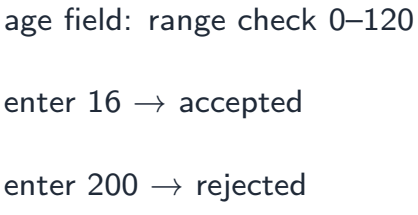
The primary key

A **primary key** 主键 is a field that holds a **unique** 唯一的 value for every record. No two records can have the same primary key, so it lets you pick out exactly one record.

In the table above, `StudentID` is a good primary key because every student has a different number. A field like `FormClass` would be a bad primary key, because many students share the same class.

Validation in a database

When data is put into a database, **validation** 验证 checks make sure it is sensible — for example a range check on an age field, or a presence check so a field is not left empty. (You saw these checks in topic 7.)



A range check accepts sensible values and rejects the rest

Structured Query Language (SQL)

Structured Query Language 结构化查询语言 (SQL) is a language used to **query** 查询 a database — to pick out the records you want. You must understand and complete SQL scripts.

SELECT, FROM and WHERE

- `SELECT` says **which fields** to show.
- `FROM` says **which table** to use.
- `WHERE` gives a **condition** 条件, so only matching records are shown.

```
SELECT name
FROM students
WHERE age > 15
```

gets the names of students older than 15

SELECT picks fields, FROM names the table, WHERE sets the condition

```
SELECT FirstName, FormClass
FROM Student
WHERE FeesPaid = TRUE;
```

This shows the first name and class of every student who has paid the fees.

Use `*` to select **all** fields:

```
SELECT *
FROM Student
WHERE FormClass = '10A';
```

ORDER BY

`ORDER BY` sorts the results. Use `ASC` for **ascending** 升序 (smallest first, A→Z) or `DESC` for **descending** 降序 (largest first, Z→A).

```
SELECT FirstName, DateOfBirth
FROM Student
ORDER BY DateOfBirth ASC;
```

AND and OR

Join conditions with `AND` (both must be true) or `OR` (at least one must be true).

```
SELECT FirstName
FROM Student
WHERE FormClass = '10A' AND FeesPaid = FALSE;
```

SUM and COUNT

- `SUM` adds up the values in a number field.
- `COUNT` counts how many records match.

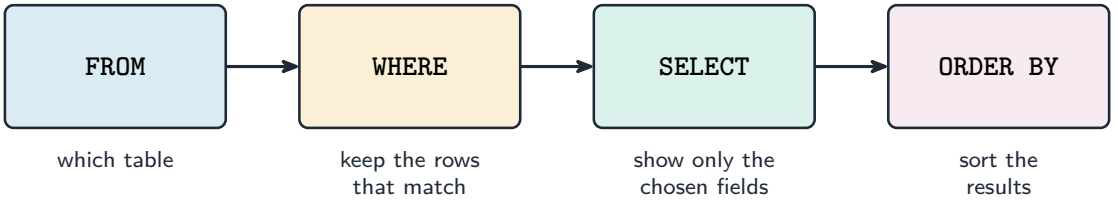
```
SELECT COUNT(StudentID)
FROM Student
WHERE FeesPaid = FALSE;
```

This counts how many students have not paid. `SUM` works the same way but adds a number field instead of counting rows.

Working out the output

To find the output of an SQL script, read it in this order:

1. `FROM` — which table;
2. `WHERE` — keep only the records that match the condition;
3. `SELECT` — show only the chosen fields;
4. `ORDER BY` — put the results in order.



Read an SQL query in this order: FROM (which table), WHERE (which rows), SELECT (which fields), ORDER BY (sort)

Following these steps, you can write down exactly which rows and columns the query returns.

Worked example. A `Book` table has the fields `Title`, `Author`, `Price` and `InStock`. Write a query showing the title and price of every book by Orwell that is in stock, cheapest first.

```
SELECT Title, Price
FROM Book
WHERE Author = 'Orwell' AND InStock = TRUE
ORDER BY Price ASC;
```

Build it in the reading order: `FROM` names the table; `WHERE` keeps only the matching records, and because there are two conditions they need `AND`; `SELECT` shows only the

two fields asked for; `ORDER BY … ASC` sorts them. Text values go in **quotes**, and only the fields the question asks for belong in `SELECT` - adding `Author` just because you filtered on it is the commonest way to lose a mark here.

Exam tips

- A **record** is a row (all the data about one thing); a **field** is a column (one item that every record has).
- A **primary key** must be *unique* for every record, so it picks out exactly one record (StudentID, not FormClass).
- Learn the SQL parts: `SELECT` (which fields), `FROM` (which table), `WHERE` (the condition), `ORDER BY` (sort, ASC or DESC).
- Read a query in the order **FROM → WHERE → SELECT → ORDER BY** to work out its output.
- `COUNT` counts the matching records; `SUM` adds up a number field.

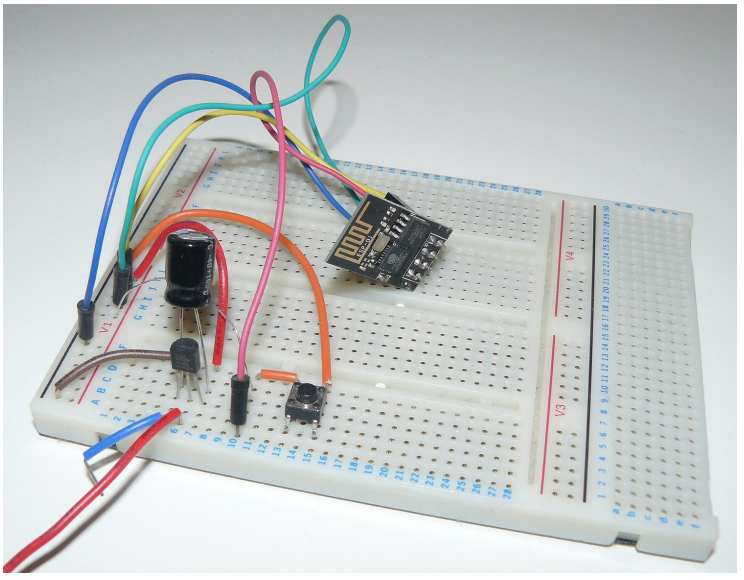
Boolean logic

IGCSE Computer Science

What is Boolean logic?

Boolean logic 布尔逻辑 works with values that are either **true** or **false**. In electronics these are shown as **1** (true) and **0** (false). A **logic gate** 逻辑门 takes one or more of these inputs and gives one output, following a fixed rule.

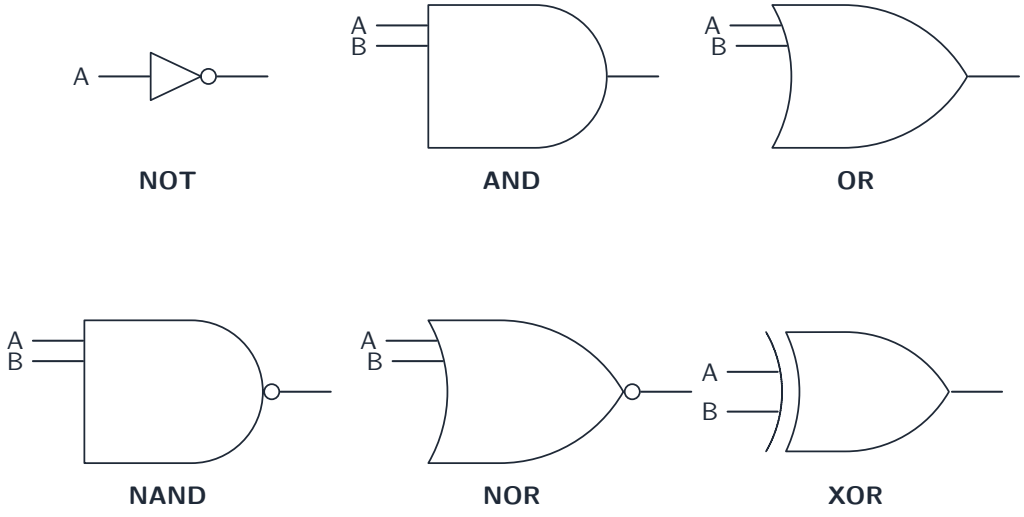
A **truth table** 真值表 lists every possible set of inputs and the output for each. You build it by writing out all the input combinations.



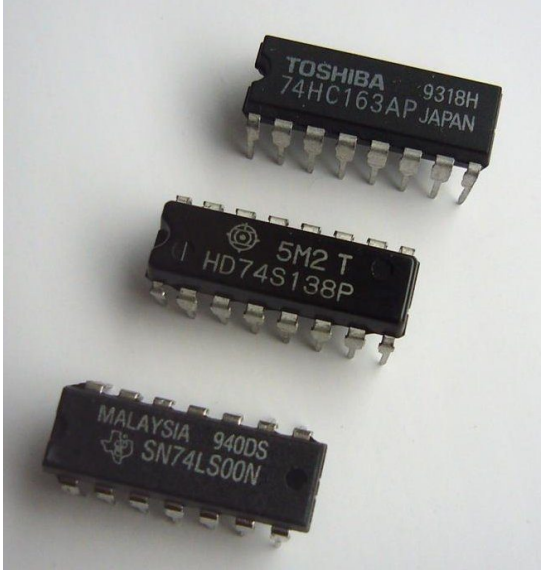
Logic gates are built from electronic circuits like this one, where each gate switches on 1s and 0s

The six logic gates

You must know six gates. `NOT` has one input; all the others have two inputs (A and B).



The six logic gates. A small circle on the output means the result is inverted (NOT, NAND, NOR)



A real logic chip: inside are logic gates like the ones on this page

NOT gate

The **NOT gate** 非门 reverses the input. Output is 1 when the input is 0.

A	Output
0	1
1	0

AND gate

The **AND gate** 与门 gives output 1 **only when both** inputs are 1.

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

AND

AND outputs 1 only when both inputs are 1

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR gate

The **OR gate** 或门 gives output 1 when **at least one** input is 1.

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

OR

OR outputs 1 when either input is 1

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

NAND gate

The **NAND gate** 与非门 is AND followed by NOT. The output is the **opposite** of AND.

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate

The **NOR gate** 或非门 is OR followed by NOT. The output is the **opposite** of OR.

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

XOR gate

The **XOR gate** 异或门 (exclusive OR) gives output 1 when the inputs are **different**.

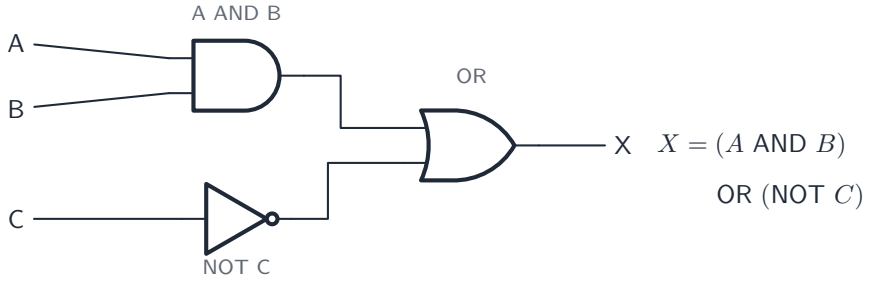
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Logic expressions

A **logic expression** 逻辑表达式 writes a circuit using letters and gate words. The usual way to write the gates:

Gate	In words
NOT A	NOT A
A AND B	A AND B
A OR B	A OR B

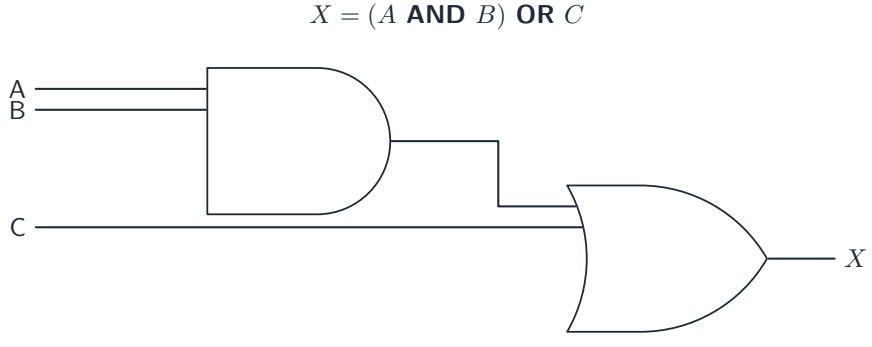
For example, the expression `(A AND B) OR (NOT C)` means: do A AND B, do NOT C, then OR the two results together.



The expression X = (A AND B) OR (NOT C) drawn as a logic circuit

Logic circuits

A **logic circuit** 逻辑电路 joins gates together to carry out a task. The output of one gate can become the input of another. At IGCSE a circuit has up to **three inputs** and **one output**.



Building the circuit for X = (A AND B) OR C — the AND gate's output feeds the OR gate

You must be able to move between four forms:

- a **problem statement** 问题陈述 (a description in words),
- a logic expression,
- a logic circuit,
- a truth table.

From a problem statement to a circuit

Read the statement and pick out the conditions and the logic words (and, or, not). For example:

> An alarm (X) sounds when the door is open (A) AND the system is switched on (B).

This is `X = A AND B`, so you draw one AND gate with inputs A and B.

Completing a truth table from a circuit or expression

To fill in a truth table:

1. Write **all** the input combinations. For three inputs there are 8 rows (000 up to 111).
2. Work out each gate's output in order, one column at a time.
3. The last column is the final output.

3 inputs $\Rightarrow 2 \times 2 \times 2 = \mathbf{8}$ rows

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

list **every** combination - count up in binary, from 000 to 111

the last column flips every row, the next every two rows, the first every four

Three inputs give eight rows: every combination counted in binary

A	B	C	A AND B	(A AND B) OR C
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Adding a middle "working" column for each gate makes the final output easy to fill in. Always draw the circuit exactly as the statement says, **without simplifying** it.

Worked example. Complete the truth table for `X = (A AND B) OR (NOT C)` for the row A = 1, B = 0, C = 0. Work **outwards from the brackets**, one gate at a time. First `A AND B` = 1 AND 0 = **0**, because AND needs both inputs to be 1. Next `NOT C` = NOT 0 = **1**. Finally OR the two results: 0 OR 1 = **1**. So X = **1**. Give each intermediate gate **its own column** rather than trying to do the whole expression in one step: with three inputs there are $2^3 = 8$ rows, and those intermediate columns are where the method marks live even if the final answer slips.

Exam tips

- Learn all six gates and their truth tables: **NOT**, **AND**, **OR**, **NAND** (NOT AND), **NOR** (NOT OR), **XOR** (output 1 when the inputs are different).
- Build a truth table with *all* input rows (2 inputs → 4 rows, 3 inputs → 8), counting up in binary, and add a working column for each gate.
- Turn a problem statement into a logic expression by picking out the AND / OR / NOT words, then draw it exactly as written — **do not simplify** it.
- NAND and NOR give the *opposite* output to AND and OR; a small circle on a gate's output means the result is inverted.