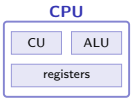
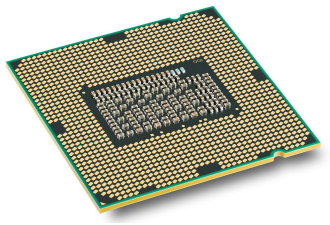
Processor Fundamentals

AS Computer Science ▪ Topic 4

A-Level Computer Science

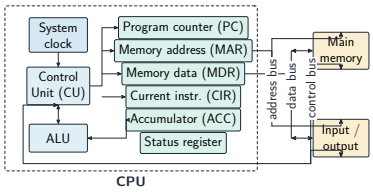


What we will cover



1 Von Neumann and the CPU

Von Neumann architecture



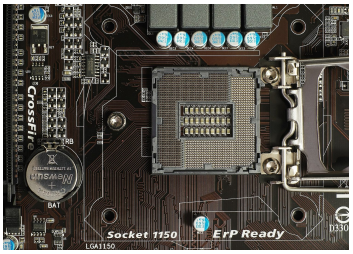
One single memory holds both program and data – the **stored program** 存储程序.

- CPU fetches and runs one instruction at a time
- change the program ⇒ change behaviour, no rewiring
- instructions run in order unless a branch changes the flow

The CPU's main parts

- **ALU** 算术逻辑单元 – arithmetic & logic
- **control unit** 控制单元 – decodes, sends control signals
- **clock** 时钟频率 – keeps the CPU in step
- **registers** 寄存器 – tiny, very fast stores, including the **general-purpose registers** 通用寄存器 for temporary values

All of it sits inside one small chip.



Special-purpose registers

Six **special purpose registers** 专用寄存器 do the work of the fetch–execute cycle:

PC
program counter: next instruction's address

MAR
memory address register: address being accessed

MDR
memory data register: data to/from memory

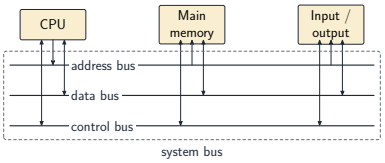
CIR
current instruction register: what is being decoded

ACC
accumulator: the value the ALU works on

Status
status register: flags – carry, zero, negative

Data moves are written as register transfers, e.g. `MAR ← [PC]`.

The system buses



- **address bus** 地址总线 – address, one-way
- **data bus** 数据总线 – data, two-way
- **control bus** 控制总线 – signals, two-way

Three sets of parallel wires. An n-bit address bus reaches 2^n locations.

What actually makes a computer faster

clock speed
时钟频率
pulses per second, e.g. 3.8 GHz

cores
核心
a multi-core CPU runs several threads at once

word size
字长
32- or 64-bit: how much the CPU handles at once

cache memory
高速缓存
a closer, faster store than the **immediate access store** 立即存取存储器

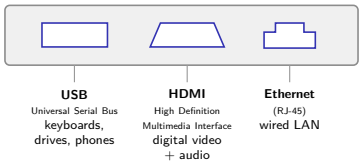
amount of RAM
随机存取存储器
too little and the OS must **page** 页 to disk

secondary storage
辅助存储器
an SSD loads programs far faster than an HDD

Each instruction takes a fixed number of clock cycles. Match the specs to the workload: a quad-core beats a dual-core on parallel work, but higher per-core speed wins on single-threaded work.

Ports connect the peripherals

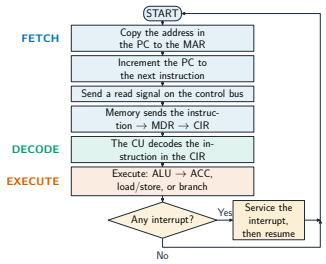
A **port** 端口 is a physical socket for a **peripheral** 外围设备. Different ports carry different signals, so an HDMI cable will not fit a USB socket.



Also on the case: **VGA** (older *analogue* video) and audio jacks. **USB-C** is unusual – it carries video, data *and* power.

2 Fetch-execute and interrupts

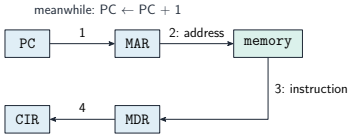
The fetch-execute cycle



- fetch: PC → MAR, increment PC, read → MDR → CIR
- decode: the CU reads the instruction
- execute: ALU / load-store / branch

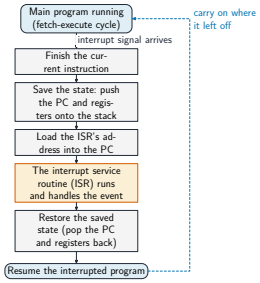
Then check for interrupts and repeat.

The register transfers in a fetch



1. `MAR ← [PC]`
2. `PC ← [PC] + 1`
3. read signal on control bus
4. `MDR ← [memory]`
5. `CIR ← [MDR]`

Interrupts



An **interrupt** 中断 pauses the cycle for an urgent event:

1. finish the current instruction
2. save the state (PC, registers)
3. run the **ISR** 中断服务程序
4. restore and carry on

Interrupts, and what the CU decodes

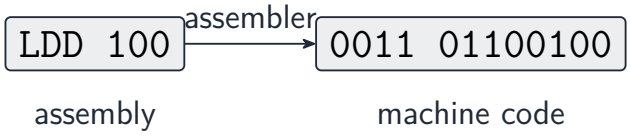
decode	the CU decodes the instruction in the CIR – what operation, and which operands or addresses
an interrupt 中断	a signal that pauses the normal cycle so the CPU can handle an urgent event – a key press, a packet arriving, a hardware fault
what happens	the current registers are saved, and the **interrupt service routine** 中断服务程序 runs
then	the saved state is restored and the interrupted program continues as if nothing happened
priorities	a higher-priority interrupt can interrupt a lower one; equal ones queue
why it matters	the system responds without the CPU constantly checking devices – and it is how the OS multitasks

Polling wastes cycles asking "anything yet?". An interrupt lets the device speak, which is why every modern system uses them.

3 Assembly and addressing

Assembly and machine code

The **processor** 处理器 runs **machine code** 机器码 (bit patterns). **Assembly language** 汇编语言 is readable, one line per instruction, using **mnemonics** 助记符 (`LDD`, `ADD`, `JMP`). A **two-pass assembler** 两遍汇编器: pass 1 builds a **symbol table** 符号表 of **labels** 标签; pass 2 generates code (handling forward references).

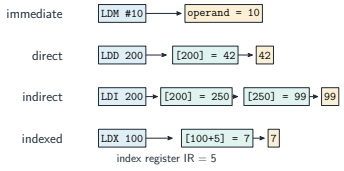


The instruction set, and the two-pass assembler

data movement	`LDD`, `LDM`, `LDI`, `LDX`, `STO`, `MOV`
arithmetic	`ADD`, `SUB`, `INC`, `DEC`
logic and shifts	`AND`, `OR`, `XOR`, `LSL`, `LSR` – and a **cyclic shift** 循环移位, where the bit leaving one end re-enters the other
comparison and jumps	`CMP`, `JMP`, `JPE`, `JPN`
the assembler's job	translate mnemonics to machine code, resolving labels to addresses
why two passes	to handle **forward references** 前向引用 – a jump to a label that is defined later in the source

Pass one builds the symbol table; pass two writes the code. One pass alone cannot know an address it has not yet reached.

Addressing modes



How the CPU finds the operand:

- **immediate addressing** 立即寻址: the operand *is* the value
- **direct addressing** 直接寻址: the value at that address
- **indirect addressing** 间接寻址: the address of an address
- **indexed addressing** 变址寻址: address + an index register
- **relative addressing** 相对寻址: an offset from the current position

Worked example: trace the program

```
LDD 200
ADD 201
STO 202
```

[200] = 5 [201] = 7 [202] = ?

LDD load ▪ ADD add ▪ STO store

after…	ACC	[202]
LDD 200	5	–
ADD 201	12	–
STO 202	12	**12**

A **trace table** 跟踪表 records every register after each instruction – exactly what exam trace questions ask for.

4 Binary shifts and masking

Binary shifts

A **logical shift** 逻辑移位 fills with 0:

- left by 1 = ×2
- right by 1 = ÷2 (integer)

An arithmetic right shift keeps the sign bit.

LSL #1	00001011 ⟶ 00010110	×2	(0 in on the right)
LSR #1	00001011 ⟶ 00000101	÷2	(0 in on the left)
ASR #1	10110100 ⟶ 11011010		sign bit kept (stays negative)

Logical vs arithmetic right shift

Both move every bit one place to the right. They differ only in what enters on the left – and that single bit decides whether a negative number survives.

start	1 1 1 1 0 0 0 0	= 240 unsigned, −16 signed	
LSR #1	0 1 1 1 1 0 0 0	= 120	halves the **unsigned** value
ASR #1	1 1 1 1 1 0 0 0	= −8	halves the **signed** value

The two results differ in **one bit** — the one that entered on the left. **LSR** always brings in a 0; **ASR** copies the sign bit, so a negative number stays negative and $-16 \div 2$ comes out right.

> **ASR** copies the sign bit, so a negative number stays negative.

Bit masking

One **bit** 位 per signal; use a **mask** 掩码:

- **set**: OR a 1 in that bit
- **clear**: AND a 0 in that bit
- **toggle**: XOR a 1 in that bit

set bit 2		clear bit 6		toggle bit 3	
	01001000		01001000		01001000
OR	00000100	AND	10111111	XOR	00001000
=	01001100	=	00001000	=	01000000

Bit masking: the three operations

test bit *n*	R AND the mask, then check whether the result is non-zero
set bit *n*	R OR the mask – ones in the mask force ones in the result
clear bit *n*	R AND the *inverted* mask – zeros force zeros, ones leave the rest alone
flip bit *n*	R XOR the mask
building a mask	a 1 in the position you care about and 0 everywhere else – 00001000 for bit 3
where it is used	monitoring and control: one byte can carry eight independent sensor or actuator states

> Each operation is chosen for what it leaves *unchanged*. AND clears, OR sets, XOR flips – and in every case the other bits survive.

Worked example – four addressing modes

Memory: location 200=250, 250=99, 105=7. The index register holds 5. What is in the accumulator after each instruction?

- `LDM #200` – **immediate**: the operand *is* the number. Accumulator = **200**.
- `LDD 200` – **direct**: go to location 200 and take what is there. = **250**.
- `LDI 200` – **indirect**: location 200 holds 250, which is another *address*, so go on to 250. = **99**.
- `LDX 100` – **indexed**: add the index register, $100 + 5 = 105$, and read location 105. = **7**.

Count the hops: immediate none, direct one, indirect two, indexed one after an addition.

Exam tips

- Learn the **fetch-execute cycle** in register-transfer terms (PC, MAR, MDR, CIR, ACC) and what increments the PC.
- Name each register's job; the **address bus is one-way**, the data bus is two-way.
- Distinguish the **addressing modes** (immediate, direct, indirect, indexed) – a frequent question.
- Explain how clock speed, number of cores, cache size and word length affect **performance**.
- For a **binary shift**, state whether it is logical or arithmetic; a left shift multiplies by 2, a right shift divides by 2.

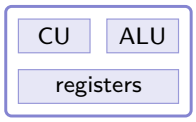
5 Recap

Topic 4 – key takeaways

1. **Von Neumann** – one memory for program & data
2. **Buses** – address (one-way), data, control
3. **Cycle** – fetch → decode → execute; interrupts pause it
4. **Assembly** – mnemonics; addressing modes; shifts & masks

Check yourself

1. Which register holds the next instruction's address?
2. Which bus is one-way?
3. What does an interrupt make the CPU do first?
4. 00000110 after LSL #1 – value?



Answers **1.** the PC **2.** the address bus **3.** finish the current instruction, then save state **4.** 00001100 = 12