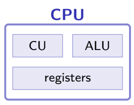
Processor Fundamentals

AS Computer Science ▪ Topic 4

A-Level Computer Science



What we will cover

1 Von Neumann & the CPU

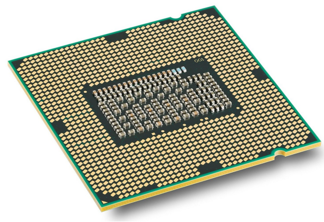
2 Buses & performance

3 Fetch-execute & interrupts

4 Assembly & addressing

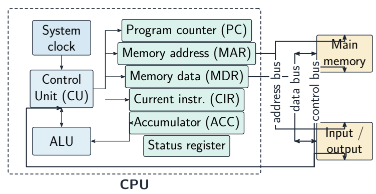
5 Binary shifts & masking

6 Recap



1 Von Neumann and the CPU

Von Neumann architecture

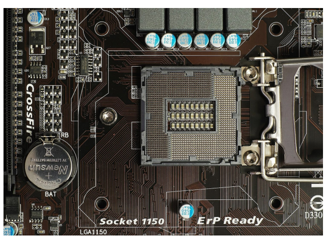


One single memory holds both program and data – the **stored program** 存储程序.

- CPU fetches and runs one instruction at a time
- change the program ⇒ change behaviour, no rewiring

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



Special-purpose registers

PC
next instruction's address

MAR
address being accessed

MDR
data to/from memory

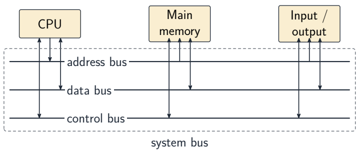
CIR
instruction being decoded

ACC
value the ALU works on

Status
flags: carry, zero, ···

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

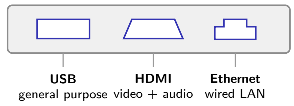
An n-bit address bus reaches 2^n locations.

Performance and ports

Performance depends on:

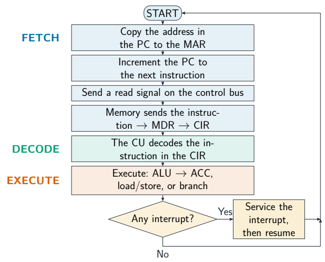
- **clock speed** 时钟频率 & **cores** 核心
- **word size** 字长 (32 vs 64-bit)
- RAM, **cache** 高速缓存, bus width, SSD vs HDD

Ports 端口 connect **peripherals** 外围设备:



2 Fetch-execute and interrupts

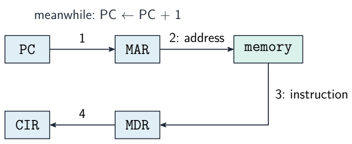
The fetch-execute cycle



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

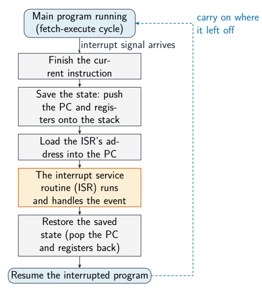
Then check for interrupts and repeat.

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



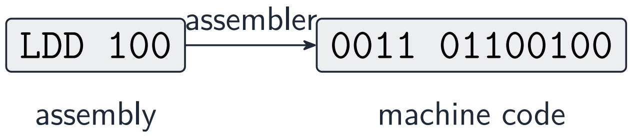
An **interrupt** 中断 pauses the cycle for an urgent event:

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

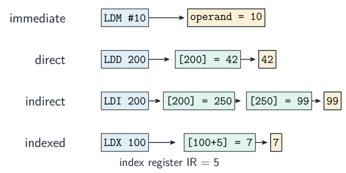
3 Assembly and addressing

Assembly and machine code

The CPU runs **machine code** 机器码 (bit patterns). **Assembly** 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).



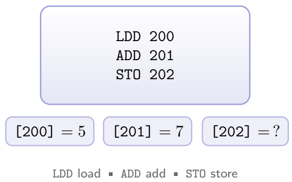
Addressing modes



How the CPU finds the operand:

- **immediate** 立即寻址: the value itself
- **direct** 直接寻址: the value at an address
- **indirect** 间接寻址: address of an address
- **indexed** 变址寻址: address + index register

Worked example: trace the program



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 $=\times 2$
- right by 1 $=\div 2$ (integer)

An arithmetic right shift keeps the sign bit.

LSL #1	00001011 → 00010110		$\times 2$	(0 in on the right)
LSR #1	00001011 → 00000101		$\div 2$	(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.

- **left shift by 1** ('LSL #1') – bits move left, a 0 enters on the right; for an unsigned number this is $\times$ **2**.
- **right shift by 1** ('LSR #1') – bits move right, a 0 enters on the left; for an unsigned number this is **integer** $\div$ **2**.

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

```
    01001000
OR  00000100
=   01001100
```

clear bit 6

```
    01001000
AND 10111111
=   00001000
```

toggle bit 3

```
    01001000
XOR 00001000
=   01000000
```

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?

CU | ALU
registers

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