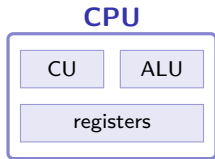


Processor Fundamentals

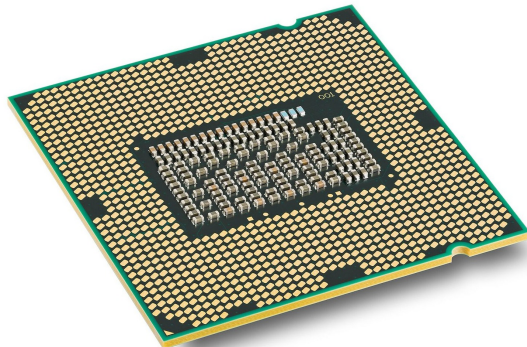
AS Computer Science ■ Topic 4

A-Level Computer Science



What we will cover

- 1 CPU architecture 4.1
- 2 Fetch, RTN and interrupts 4.1.1
- 3 Assembly language 4.2
- 4 Bit manipulation 4.3
- 5 Arithmetic and cyclic shifts 4.3.1
- 6 Recap



1

CPU architecture

Sheet 4.1 – what you need to know

You must be able to:

- describe the Von Neumann model and every special-purpose register
- explain the buses and what makes a CPU faster
- compare ports such as USB, HDMI and VGA

What the exam asks

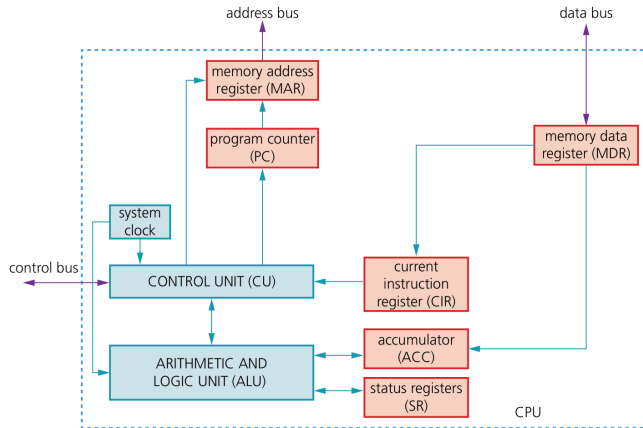
“Complete the table by writing the three incorrect statement numbers and the corrected statements.” [3]

“Explain how the CU and the system clock work together.” [2]

“Explain the benefits of connecting the monitor using the HDMI port instead of the VGA port.” [4]

Practise on exercise sheet 4.1, then try the real exam questions on past-paper sheet 4.1.

Von Neumann architecture

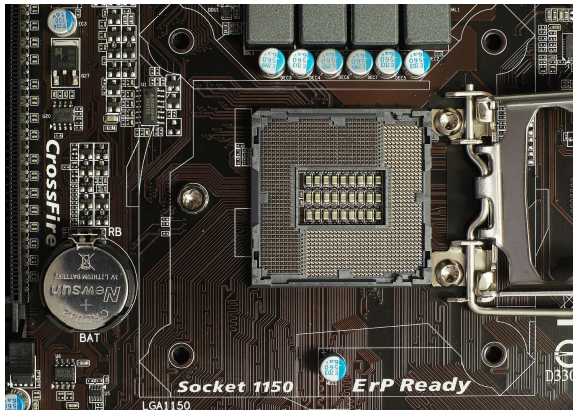


The **stored-program** 存储程序 model uses one memory for instructions and data.

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.

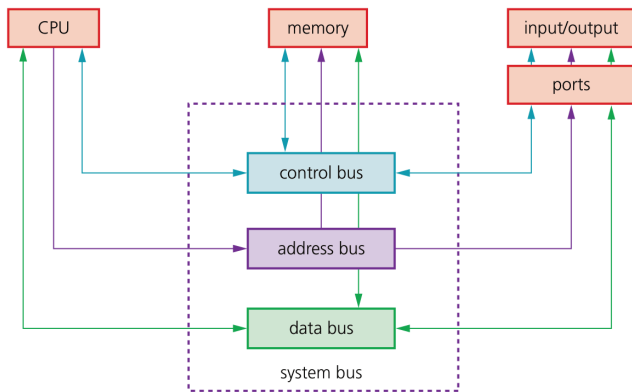


Special-purpose registers

Register	What it holds
PC	Address of the next instruction
MAR	Address of the memory location being accessed
MDR	Data or an instruction transferred to/from memory
CIR	The current instruction being decoded/executed
ACC	Operands and results used by the ALU
SR	Status flags: carry, zero, negative and overflow

The **index register** 变址寄存器 (IX) holds an offset added to an address.

The system buses



- **address bus** 地址总线: addresses, one-way
- **data bus** 数据总线: data and instructions, two-way
- **control bus** 控制总线: control signals, two-way

An n -bit address bus selects 2^n memory locations.

What actually makes a computer faster

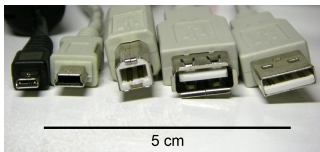
Factor	Why it helps
clock speed	more cycles per second, so more instructions per second
number of cores	helps when software divides its work between them
cache memory	fast, nearby storage cuts slow main-memory access
data bus width	a wider bus transfers more bits at once
word size	more bits processed together
RAM and storage	enough RAM avoids paging; an SSD loads faster than an HDD

Real performance also depends on the workload and the architecture.

Ports connect the peripherals

A **port** 端口 is a socket for a **peripheral** 外围设备. Match the connector and signal type.

USB



Data in both directions;
power. Keyboards, drives
and phones.

USB: plug-and-play. USB-C can also carry video when supported.

HDMI



Digital video and audio.
Monitors, TVs and
projectors.

VGA



Analogue video; no audio.
Older monitors and
projectors.

Clock, cores, registers and ports: the answers

“Explain how the CU and the system clock work together.” [2]

The clock sends regular **pulses** along the control bus; the CU sends its **control signals** in step with them, so each stage of the cycle happens on a tick.

Why do more cores not always mean more speed?

Not all software can be split into **parallel** tasks; the cores must **communicate** and share data, which costs time.

General-purpose vs special-purpose registers

General: hold any temporary values the program is working on.

Special: one fixed job each (PC, MAR, MDR, CIR, ACC, IX, SR).

“Explain the benefits of HDMI instead of VGA.” [4]

HDMI is **digital**: no conversion, so a sharper picture; higher **resolution**; it carries **sound** too; it can **encrypt** the signal.

2

Fetch, RTN and interrupts

Sheet 4.1.1 – what you need to know

You must be able to:

- describe the fetch–execute cycle in register transfer notation
- say which bus carries what, including control signals
- explain how an interrupt is detected and handled

What the exam asks

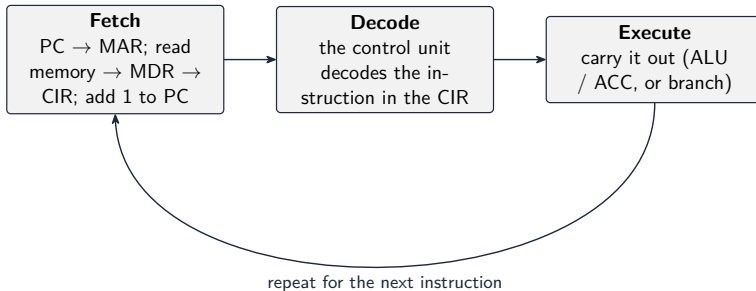
“Tick one box in each row to identify the system bus used by each CPU component.” [1]

“Identify two causes of a software interrupt.” [2]

“Describe each of the given steps.
 $PC \leftarrow [PC] + 1$ / $MDR \leftarrow [[MAR]]$ / $MAR \leftarrow [PC]$ ” [3]

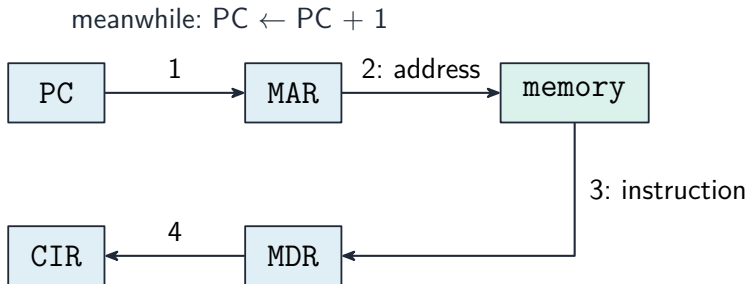
Practise on exercise sheet 4.1.1, then try the real exam questions on past-paper sheet 4.1.1.

The fetch-execute cycle



Each instruction is fetched, decoded and executed. The CPU then checks for interrupts before the next instruction.

The register transfers in a fetch



The address goes out through the MAR; the instruction returns through the MDR. Send a read signal, copy the instruction to the CIR, and increment the PC.

Reading register transfer notation

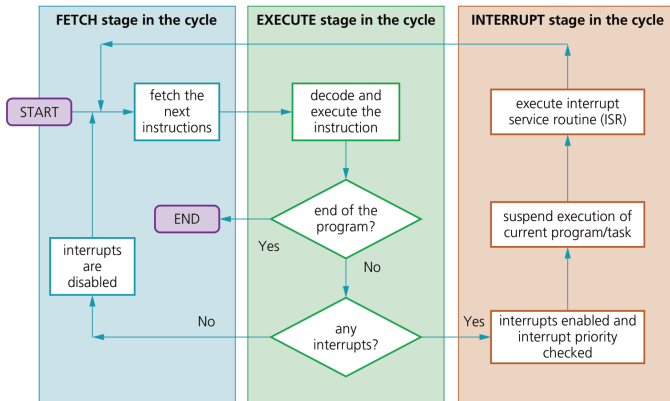
step	meaning
$MAR \leftarrow [PC]$	copy the contents of the PC into the MAR
$PC \leftarrow [PC] + 1$	increment the PC
$MDR \leftarrow [[MAR]]$	copy the contents of the address held in the MAR into the MDR
$CIR \leftarrow [MDR]$	copy the instruction into the CIR

[X] means “the contents of X”.
[[MAR]] goes one step further: the contents of the memory location whose address is in the MAR.

“Describe each of the given steps.”
[3]

Name the register **and** say what is copied where; “the PC goes to the MAR” loses the mark.

Interrupts



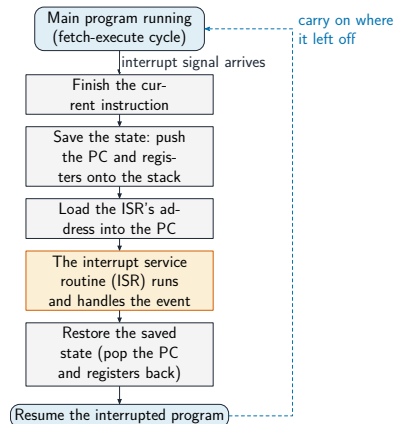
Finish the instruction; save the registers; run the ISR; restore the state and resume.

Interrupts, and what the CU decodes

Decode: the CU decodes the instruction in the **CIR** – which operation, which operands.

Interrupt: a signal requesting attention – a key press, a timer, a fault.

Finish the current instruction, **save the registers**, run the **ISR** 中断服务程序, restore and resume. A higher priority can interrupt a lower one.



The control bus, interrupt causes and the stack

Signals on the control bus

memory **read** and **write**, the **clock**, **interrupt** requests, bus request and grant, reset.

“Identify two causes of a software interrupt.” [2]

Division by zero; two processes trying to use the **same memory**; a program asking the OS for a service.

(Hardware: a key press, the printer out of paper, a timer.)

When are interrupts checked?

At the **end** of each fetch–execute cycle, before the next fetch.

Why save the registers on a stack?

The PC and other registers are **pushed** so the interrupted program can resume **exactly where it stopped**; they are popped after the ISR. A stack also lets a higher-priority interrupt interrupt the ISR.

3

Assembly language

Sheet 4.2 – what you need to know

You must be able to:

- explain the assembler and the five addressing modes
- read operands written in denary, binary and hex
- trace a program with compares and jumps

What the exam asks

“State what is meant by relative addressing.” [1]

“Complete the table by writing the contents of the ACC after the execution of each instruction.”
[4]

“Complete each statement by writing the name of the appropriate instruction group.”
[3]

Practise on exercise sheet 4.2, then try the real exam questions on past-paper sheet 4.2.

Assembly and machine code

Assembly language 汇编语言 uses readable **mnemonics** 助记符, such as LDD, ADD and JMP.

Assembly		Machine code
LDD 100	$\xrightarrow{\text{assembler}}$	0011 01100100

The processor executes the binary instruction, not the mnemonic.

In this illustrative encoding, 0011 is the opcode and 01100100 is the address 100.

The instruction set, and the two-pass assembler

Instruction group	Examples
Data movement	LDD, LDM, LDI, LDX, STO, MOV
Arithmetic	ADD, SUB, INC, DEC
Logic and shifts	AND, OR, XOR, LSL, LSR; cyclic shift
Compare and jump	CMP, JMP, JPE, JPN

Pass 1: build a **symbol table** 符号表 of labels and addresses.

Pass 2: translate instructions and replace labels with addresses.

Two passes resolve a **forward reference** 前向引用: a jump to a label defined later in the source.

Addressing modes

The addressing mode tells the CPU how to find the **operand** 操作数.

Mode	Where the operand comes from
------	------------------------------

Immediate	The value is in the instruction.
-----------	----------------------------------

Direct	Read the memory address in the instruction.
--------	---

Indirect	Read an address from memory, then follow it.
----------	--

Indexed	Add IX to the base address, then read memory.
---------	---

Relative	Add an offset to the current instruction position.
----------	--

Worked example – four addressing modes

Given: $[200] = 250$, $[250] = 99$, $[105] = 7$, and $IX = 5$.

Instruction	Mode	Route to the value	ACC
LDM #200	Immediate	Take the value 200	200
LDD 200	Direct	$200 \rightarrow 250$	250
LDI 200	Indirect	$200 \rightarrow 250 \rightarrow 99$	99
LDX 100	Indexed	$100 + 5 = 105 \rightarrow 7$	7

Count memory reads: immediate 0; direct 1; indirect 2; indexed 1 after adding IX.

Operands: denary, binary and hex

operand	means
#13	the denary value 13
B00001101	the binary value 13
&0D	the hexadecimal value 13
13 (no prefix)	the address 13

CMP compares the ACC with a value and sets a flag. It does [not](#) change the ACC.

ACC holds 1011 0101. Give the ACC after each.

AND B00001111	0000 0101
XOR &F1	0100 0100
CMP #181	1011 0101 (unchanged)

“State what is meant by relative addressing.” [1]

The address is found by adding an [offset](#) to the address of the current instruction.

Worked example: trace the program

Initially: memory location 200 holds 5; location 201 holds 7.

	After instruction	ACC	[202]
LDD 200			
ADD 201	LDD 200	5	–
STO 202	ADD 201	12	–
	STO 202	12	12

A **trace table** 跟踪表 records the values after each instruction.

Worked example – a trace with a loop

```

100  LDD 200
101  CMP #5
102  JPE 105
103  INC ACC
104  JMP 101
105  STO 201
106  END

```

[200] = 4

instruction	compare	ACC	[201]
LDD 200		4	
CMP #5	false	4	
JPE 105	not taken	4	
INC ACC		5	
JMP 101		5	
CMP #5	true	5	
JPE 105	taken	5	
STO 201		5	5

Why write in assembly? Direct control of the hardware, and fast, small code for devices with little memory.

4

Bit manipulation

Sheet 4.3 – what you need to know

You must be able to:

- give the result of logical and arithmetic shifts
- test, set, clear and toggle bits with a mask
- write the instruction itself, operand and all

What the exam asks

“Write the bit manipulation instruction that can be used to set the least significant bit to 1 in an 8-bit register.” [1]

“Show the result of an arithmetic right shift of 3 places on 10011110.” [1]

“Explain how bit manipulation can be used to clear the data in an 8-bit register.” [3]

Practise on exercise sheet 4.3, then try the real exam questions on past-paper sheet 4.3.

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 $\longrightarrow$ 00010110 $\times 2$ (0 in on the right)

LSR #1 00001011 $\longrightarrow$ 00000101 $\div 2$ (0 in on the left)

ASR #1 10110100 $\longrightarrow$ 11011010 sign bit kept (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

$$\begin{array}{r} 01001000 \\ \text{OR } 00000100 \\ \hline = 01001100 \end{array}$$

clear bit 6

$$\begin{array}{r} 01001000 \\ \text{AND } 10111111 \\ \hline = 00001000 \end{array}$$

toggle bit 3

$$\begin{array}{r} 01001000 \\ \text{XOR } 00001000 \\ \hline = 01000000 \end{array}$$

Bit masks: test, set, clear and toggle

To act on bit 3, use the mask 00001000 (bit 0 is on the right).

Task	Operation	Effect
Test bit 3	R AND 00001000	Non-zero means the bit is set.
Set bit 3	R OR 00001000	Force bit 3 to 1.
Clear bit 3	R AND 11110111	Force bit 3 to 0.
Toggle bit 3	R XOR 00001000	Flip bit 3.

The other bits stay unchanged. Masks let one byte hold eight separate sensor or actuator states.

Writing the bit-manipulation instruction

task (8-bit ACC)	instruction
set the LSB to 1	OR #1
clear bit 6	AND B10111111
clear the whole register	AND #0
flip every bit but the LSB	XOR &FE
test bit 7	AND B10000000

“Explain how bit manipulation can clear an 8-bit register, and write the instruction.” [3]

AND each bit with 0: anything AND 0 is 0, so every bit becomes 0.
AND #0 (or AND B00000000).

Arithmetic right shift of 3 on
1001 1110

Copy the sign bit in: 1111 0011
−98 → −13 (rounded down from −12.25).

5

Arithmetic and cyclic shifts

Sheet 4.3.1 – what you need to know

You must be able to:

- apply LSL, LSR, ASR and cyclic shifts by several places
- say when a shift is no longer a multiply or a divide
- combine a shift with a mask

What the exam asks

“Show the result of an arithmetic right shift of 3 places on 10011110.” [1]

“Write the bit manipulation instruction that can be used to set the least significant bit to 1.” [1]

“Explain how bit manipulation can be used to clear the data in an 8-bit register.” [3]

Practise on exercise sheet 4.3.1, then try the real exam questions on past-paper sheet 4.3.

Logical vs arithmetic right shift

Both start with 11110000: -16 in 8-bit two's complement.

Operation	Resulting bits	Signed value
Logical right shift	01111000	+120
Arithmetic right shift	11111000	-8

- LSR inserts a zero at the left; it treats the bits as unsigned.
- ASR copies the sign bit, preserving the negative sign.

Cyclic shifts, lost bits and rounding

Cyclic left shift by 3 of 1011 0010

Bits leaving the left end **re-enter** at the right:

1011 0010 $\rightarrow$ 0110 0101 $\rightarrow$ 1100 1010 $\rightarrow$
1001 0101

Eight cyclic shifts give back the original byte.

When a shift stops being a multiply

0110 0000 (96) LSL 2 $\rightarrow$ 1000 0000: a 1 fell off the left, so the answer is 128, not 384.

ASR rounds towards $-\infty$

$-7 = 1111\ 1001$; ASR 1 $\rightarrow 1111\ 1100 = -4$, not -3.5 or -3 .

Shift and mask together: split 1011 0110

high nibble: LSR #4 $\rightarrow 0000\ 1011 = 11$

low nibble: AND &0F $\rightarrow 0000\ 0110 = 6$

A logical shift loses bits; a cyclic shift keeps every one.

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.

Image credits

- CPU architecture, system buses and interrupt flowchart:
Cambridge International AS and A Level Computer Science, Hodder; Figures 4.1, 4.2 and 4.6.
- USB connectors: Techtonic, public domain.
- HDMI connector: Evan-Amos, public domain.
- VGA connector: Swift.Hg, CC BY-SA 3.0; resized for the slide.

6

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

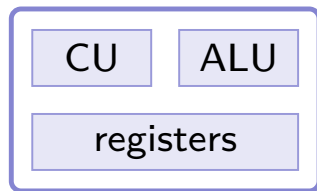
4

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. $00001100 = 12$