Solutions

Each answer shows the steps, then the **result**.

2.1

- both the **program (instructions) and the data** are held together in the **same memory**

2.2

- PC holds the address of the **next** instruction
- MAR holds the address currently being read from or written to memory

2.3

- a special-purpose register has a **fixed role** in the cycle (e.g. PC, MAR)
- a general-purpose register holds **temporary values** chosen by the program

2.4

- the **address bus** is one-way
- the CPU only ever **sends** an address to memory — memory never sends an address back

3.1

- **fetch:** the address in the PC is copied to the MAR; the instruction is read from memory into the MDR, then the CIR; the PC is increased by 1
- **decode:** the control unit decodes the instruction in the CIR
- **execute:** the instruction is carried out (ALU/ACC, or a branch)

3.2

(a) the ALU performs **arithmetic and logic** operations

- the control unit **decodes instructions and sends control signals**

(b) any two of: clock speed; number of cores; cache size; bus width

3.3

- an interrupt is a signal that makes the CPU **pause** the current program
- it runs a routine to handle an urgent event, then **returns** to where it left off

3.4

- the MAR holds the **address** of the instruction to fetch
- the MDR holds the **instruction/data** read back from that address

3.5

- a wider address bus has more lines, so it can carry **more distinct addresses**
- the CPU can then directly address **more memory** (each extra line doubles the addressable memory)

3.6

(a) **PC** — holds the address of the **next** instruction to be fetched

- **MAR** — holds the address currently being read from or written to
- **MDR** — holds the data or instruction just fetched from, or about to be written to, that address
- **CIR** — holds the instruction currently being decoded and executed

(b) each core can fetch and execute its own instructions, so several instructions run **at the same time**

- the program must be written to divide its work between cores, and parts of it are inherently sequential
- the cores also compete for the same memory and bus, so the gain is less than the core count suggests

(c) cache holds recently and frequently used instructions and data close to the processor

- it is far faster to access than main memory, so more cache means fewer slow main-memory fetches

(d) any three: DRAM stores each bit in a **capacitor** that must be **refreshed**, SRAM uses flip-flops and does not; SRAM is **faster**; SRAM is more expensive per bit and takes more space, so DRAM gives more capacity for the money

- **SRAM** is used for cache

(e) a wider data bus carries **more bits per transfer**

- each fetch moves more data and fewer transfers are needed for the same work, raising throughput without any change in clock speed