

Solutions

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

2.1

- any one: RISC has **few simple fixed-length** instructions, CISC has **many complex variable-length** ones; in RISC only load/store access memory

2.2

- it **overlaps the stages** of consecutive instructions so more than one is processed at once (one finishes per cycle when full)

2.3

- Single Instruction, Multiple Data**
- example: a **GPU** / graphics card / CPU vector unit

2.4

- software** that behaves like a separate (virtual) computer, running its own operating system on a host machine

2.5

- without: each instruction uses all 4 cycles: $10 \times 4 = 40$ **cycles**
- with: cycles = stages + (instructions - 1) = $4 + (10 - 1) = 13$ **cycles**

2.6 any two:

- a crash or fault in the new OS **stays inside the VM**; the host is not harmed
- no **extra hardware** is needed to run it
- it can run **beside** the current OS, for comparison

3.1

- any two: **fixed-length** instructions (each stage takes equal time); **simple** instructions (each ~1 cycle); register-to-register operations (only load/store touch memory), so stages are predictable

3.2

- while one instruction is being executed, the **next is being decoded and another fetched**
- so instructions overlap and throughput rises to ~one per cycle
- a **data** hazard (or control/branch hazard) can stall it

3.3

- SISD, SIMD, MISD, MIMD**

3.4

- any one: **thousands of processors**; each with its **own (distributed) memory**; they communicate by **message passing**; it is MIMD

3.5

- any **four**: a small set of **simple** instructions; **fixed-length** instructions; only **load/store** instructions access memory; **many general-purpose registers**; a **hardwired** control unit; designed for **pipelining** (about one instruction per cycle)

3.6 any four:

- a **large** set of **complex** instructions
- instructions of **variable length**
- many addressing modes**
- instructions take **many clock cycles**
- a **microprogrammed** control unit
- fewer** registers; many instructions can **access memory directly**

3.7

- SIMD: **one instruction** is carried out on **many data items at the same time**
- SIMD: by several processing units in step, e.g. an **array/vector processor**, a GPU
- MIMD: **several processors**, each runs **different instructions** on **different data**
- MIMD: the processors work **independently** (asynchronously), e.g. a **multi-core** processor

3.8

(a)

- benefits (any two): **several OSe**s on one machine; a crash is **contained** in the VM; **test** software / a new OS safely; run **legacy** software
- limitations (any two): **slower** than running on the real hardware; needs a lot of **memory/processing**; some hardware **cannot be emulated**; complex to set up

(b)

- host OS: runs **directly on the hardware**
- host OS: runs the VM software and **allocates the real resources** (memory, processor time) to each guest
- guest OS: runs **inside the virtual machine**, as if it controlled real hardware
- guest OS: its requests for hardware are **passed to / handled by** the VM software and host

3.9

(a)

- cycles = stages + (instructions - 1)
- = $5 + (20 - 1) = 24$ **cycles**

(b) any four:

- the processor **checks for an interrupt** at the end of each fetch–execute cycle
- if the interrupt has **higher priority**, the current **register contents are saved** (on the stack)
- the **address of the ISR** is loaded into the **PC** (from the interrupt vector table)
- the **ISR runs** to deal with the interrupt
- the saved registers are **restored** and the original task **continues**

(c) any three:

- several instructions are **part-way through** the pipeline when the interrupt arrives
- they can be **flushed/discarded** (and fetched again after the ISR) ...
- ... or the instructions already in the pipeline are **allowed to finish** first
- the state is **saved**, and the **ISR is loaded into the pipeline**