ANSWERS

A9.8 Performance, pipelining and parallel processors

Computer architecture · A level · OCR H446 1.1.1, AQA 7517 4.7.3.7, Eduqas A500QS 2.1 · about 25 min

What this lesson is about

Cores, cache, clock speed, word length and bus widths; pipelining; CISC and RISC; multicore systems and GPUs.

- Factors affecting performance
- Pipelining
- CISC and RISC
- Multicore and parallel systems
- GPUs

Questions

6 marks in all

1. A three-stage pipeline runs 10 instructions with no branches. How many ticks does it take? [1 mark]

> **Answer:** 12. n + s - 1 = 10 + 3 - 1.

2. Which is a feature of RISC processors rather than CISC? [1 mark]

- ☐ **A** A small set of simple, fixed-length instructions that are easy to pipeline
- ☐ **B** Many complex instructions of different lengths
- ☐ **C** Fewer instructions per program
- ☐ **D** More of the work done in hardware

> **Answer:** A. RISC keeps instructions simple and uniform; the compiler builds complex operations from them.

3. Why does a larger cache usually improve performance? [1 mark]

- ☐ **A** More instructions and data can be found in fast memory, so fewer slow fetches from main memory are needed
- ☐ **B** It increases the clock speed
- ☐ **C** It adds more cores
- ☐ **D** It widens the address bus

> **Answer:** A. Link the factor to what gets faster: fewer trips to slower RAM.

4. Which task is best suited to a GPU? [1 mark]

- ☐ **A** Training a machine learning model
- ☐ **B** Running an operating system's scheduler
- ☐ **C** Handling a keyboard interrupt
- ☐ **D** Following a long chain of if statements

> **Answer:** A. Training is dominated by the same matrix calculations on huge amounts of data, which GPU cores do in parallel.

5. What happens in a pipeline when a branch is taken? [1 mark]

- ☐ **A** The instructions already fetched and decoded after the branch are flushed
- ☐ **B** The pipeline speeds up
- ☐ **C** The branch is ignored
- ☐ **D** The clock speed doubles

> **Answer:** A. Those instructions were the wrong ones, so the work on them is wasted.

6. What does this program print? [1 mark]

```
def speedup(p, cores):
    return 1 / ((1 - p) + p / cores)

print(round(speedup(0.5, 2), 2))
print(round(speedup(0.5, 100), 2))
```

> **Answer:**
>
> ```
> 1.33
> 1.98
> ```
>
> Half the job is serial, so even a hundred cores cannot quite double the speed.

The task: a pipeline, tick by tick

Simulate a pipeline. `instructions` is the list `["I1", "I2", "I3", "I4"]` and `stages` is the list `["F", "D", "E"]`, in pipeline order. Each tick, every instruction moves on one stage: I1 is fetched at tick 1, decoded at tick 2 and executed at tick 3, and each later instruction follows one tick behind the one before it. A stage with no instruction in it holds nothing. - For every tick until the last instruction leaves the last stage, print one line in the form `tick 2: F=I2 D=I1 E=-`, with each stage's name, `=`, and the instruction it holds or `-` for an empty stage, separated by single spaces. - Then print `pipelined: 6 ticks` and `not pipelined: 12 ticks`, working both numbers out from the lengths of the two lists. The robot does not move.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

instructions = ["I1", "I2", "I3", "I4"]
stages = ["F", "D", "E"]
```

The hint students can ask for: Number the instructions and the stages from 0. At a given tick, work out which instruction number each stage would hold by counting back from the tick, and check whether that number is a real instruction. The total number of ticks comes from when the last instruction reaches the last stage.

A solution

```
from bugbot import *
connect()
instructions = ["I1", "I2", "I3", "I4"]
stages = ["F", "D", "E"]
n = len(instructions)
s = len(stages)
total = n + s - 1
for tick in range(1, total + 1):
    parts = []
    for k in range(s):
        i = tick - 1 - k
        parts.append(stages[k] + "=" + (instructions[i] if 0 <= i < n else "-"))
    print(f"tick {tick}: " + " ".join(parts))
print(f"pipelined: {total} ticks")
print(f"not pipelined: {n * s} ticks")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.