3.3 Processor and memory overview

Name: _____ Class: _____ Date: _____ **Total: 24 marks**

KEY WORDS fetch-decode-execute cycle - - • cache

Objective

Trace the cycle; reason about the hierarchy.

1 Key knowledge and worked examples

■ The loop and the pyramid

Fetch (PC → memory → IR) → decode → execute; PC → PC + 4 ✓. Registers (~ns) → L1/L2/L3 → DRAM (~100 ns) → SSD; one cache line carries 64 adjacent bytes ✓.

■ Worked example: layout beats arithmetic

Row-major array sum: consecutive bytes ride the same cache line; transposed access at the same op count is many times slower ✓ – the hierarchy, not the clock, decides.

2 Practice

2.1 Which register receives the fetched instruction? [3]

2.2 By how much does the PC advance per instruction (RISC)? [3]

2.3 Name the two locality kinds. [3]

3 Exam-style questions

3.1 Trace `LOAD R1, 0(R2)`, `ADD R3, R1, R1`, `STORE R3, 0(R2)` through fetch-decode-execute.

For each instruction, state the register (or memory) that changes and its new content, given R2 holds 100 and mem[100] = 7. Give the PC values before and after each of the three instructions. [8]

3.2 A program sums a 4096 × 4096 integer array twice: once row-major, once column-major; the machine has a 64-byte cache line.

Explain why the row-major pass loads roughly one cache line per 16 elements while the column-major pass loads one line per element, and estimate the miss-ratio contrast. State which of temporal or spatial locality each pass exploits. Give one code change that repairs the column-major pass, and one hardware reason why a bigger cache alone cannot fully hide the difference. [7]

Solutions

2.1

- The IR (instruction register) – PC supplies the address ✓.

2.2

- 4 (one word) ✓.

2.3

- Temporal (recently used again soon) and spatial (neighbours used soon) ✓.

3.1

- LOAD: R1 ← mem[100] = 7; PC 0 → 4 ✓.
- ADD: R3 ← 7 + 7 = 14; PC 4 → 8 ✓.
- STORE: mem[100] ← 14; PC 8 → 12 ✓.
- Only these change: the loop names its own deltas ✓.

3.2

- 64-byte line, 4-byte ints: 16 consecutive elements per line – row-major reuses each line 16×, column-major strides 16 KB and misses nearly every element ✓.
- Row-major exploits spatial locality; the reuse-of-line is spatial, not temporal ✓.
- Loop-interchange (reorder to keep the fast index inner) fixes the program; a bigger cache cannot hold the working set of a strided sweep – capacity misses persist ✓.