Computer Organization and Architecture

Chapter 6 – RISC, CISC Architecture and Pipelining

Exercise and Exam-Oriented Questions and Answers

Section A – Short Answer Questions

Q1. Define RISC architecture.

Ans. RISC (Reduced Instruction Set Computer) is a processor design approach that uses a small set of simple, fixed-length instructions, most of which execute in a single clock cycle, allowing efficient pipelining and higher throughput.

Q2. Define CISC architecture.

Ans. CISC (Complex Instruction Set Computer) is a processor design approach that uses a large set of complex, variable-length instructions, some of which can perform multi-step operations such as memory access combined with arithmetic, in a single instruction.

Q3. What is meant by CPI?

Ans. CPI stands for Cycles Per Instruction — the average number of clock cycles required to execute one instruction. RISC processors aim for a CPI close to 1, while CISC processors often have a CPI greater than 1 due to complex, multi-cycle instructions.

Q4. What is a load-store architecture?

Ans. A load-store architecture is one in which only the LOAD and STORE instructions are permitted to access main memory; all other instructions (arithmetic, logical, etc.) operate exclusively on values already held in registers. It is a defining feature of RISC processors.

Q5. Define parallel processing.

Ans. Parallel processing is the simultaneous execution of multiple instructions, tasks, or data operations using multiple functional units or processors, with the goal of increasing overall computational throughput.

Q6. On what basis did Flynn classify computer architectures?

Ans. Flynn classified computer architectures on the basis of the number of concurrent instruction streams and data streams a system can handle, giving four categories: SISD, SIMD, MISD, and MIMD.

Q7. Define instruction pipelining.

Ans. Instruction pipelining is a technique in which the execution of an instruction is broken down into a sequence of independent stages (such as fetch, decode, execute, memory access, and write back), allowing multiple instructions to be processed simultaneously at different stages, thereby overlapping their execution.

Q8. What is a pipeline stage (segment)?

Ans. A pipeline stage, or segment, is one of the sequential steps into which the processing of an instruction is divided, with each stage implemented by a dedicated piece of hardware that performs a specific part of instruction execution, such as fetch or decode.

Q9. Define speed-up due to pipelining.

Ans. Speed-up due to pipelining is the ratio of the time taken to execute a set of instructions without pipelining to the time taken to execute the same instructions with pipelining. For a k-stage pipeline executing n instructions, $S = (n \times k) / (k + n - 1)$, which approaches k as n becomes large.

Q10. What is pipeline stalling (idling)?

Ans. Pipeline stalling, or idling, occurs when one or more pipeline stages are forced to wait and perform no useful work, usually because a hazard prevents the next instruction from proceeding, which reduces the effective throughput of the pipeline.

Q11. What is a pipeline hazard?

Ans. A pipeline hazard is any situation that prevents the next instruction from executing in its designated clock cycle, potentially causing incorrect results or forcing the pipeline to stall. The three main types are structural, data, and control hazards.

Q12. What is operand forwarding (bypassing)?

Ans. Operand forwarding, or bypassing, is a hardware technique in which the result of one instruction is passed directly from the stage that produces it to an earlier stage of a dependent instruction that needs it, instead of waiting for the result to be written back to the register file, thereby reducing data-hazard stalls.

Q13. What is a branch delay slot?

Ans. A branch delay slot is the instruction position (or positions) immediately following a branch instruction in the pipeline, which is always executed regardless of the branch outcome. Compilers try to fill it with a useful, independent instruction instead of a NOP, to avoid wasting a pipeline cycle.

Q14. Differentiate between structural and data hazards in one line each.

Ans. A structural hazard occurs when two instructions need the same hardware resource at the same time, whereas a data hazard occurs when an instruction needs a value that a previous, still-executing instruction has not yet produced.

Section B – Long / Descriptive Answer Questions

Q1. Explain the characteristic features of RISC architecture.

Ans. RISC (Reduced Instruction Set Computer) architecture is built around the idea of keeping the processor's hardware simple so that most instructions execute quickly and uniformly, which in turn makes deep, efficient pipelining possible.

Its main characteristic features are: (i) a small, simple instruction set in which every instruction performs one basic operation; (ii) a fixed-length, uniform instruction format that simplifies fetch and decode; (iii) single-cycle execution for most instructions, giving a CPI close to 1; (iv) a load-store architecture, where only LOAD and STORE instructions touch memory; (v) a large general-purpose register file, which reduces the need for memory accesses; (vi) a hardwired (rather than micro-programmed) control unit for speed; and (vii) very few, simple addressing modes.

Because of these features, RISC processors rely heavily on the compiler to schedule instructions optimally and make efficient use of registers, shifting complexity from hardware to software.

Q2. Compare RISC and CISC architectures.

Ans. RISC and CISC represent two opposite design philosophies for a processor's instruction set architecture.

RISC uses a small set of simple, fixed-length instructions that mostly execute in a single clock cycle, a load-store model where only LOAD/STORE access memory, few addressing modes, a hardwired control unit, and a large register file; this makes RISC processors easy to pipeline efficiently, though programs tend to have more instructions.

CISC, in contrast, uses a large set of complex, variable-length instructions that can take multiple clock cycles, allows many instructions to access memory directly, supports many addressing modes, and typically uses micro-programmed control; this results in smaller program sizes but instructions that are harder to pipeline efficiently because of their variable length and complexity.

In short, RISC trades a larger instruction count for simpler, faster, and more predictable execution, while CISC trades hardware simplicity for more powerful, code-dense instructions.

Q3. What is parallel processing? Explain Flynn's classification of computer architectures.

Ans. Parallel processing is the simultaneous execution of multiple instructions, tasks, or data items using multiple functional units or processors, with the aim of improving the overall throughput of a computing system beyond what a strictly sequential processor can achieve.

Michael J. Flynn classified computer architectures into four categories based on the number of concurrent instruction streams and data streams: SISD (Single Instruction, Single Data) describes a conventional sequential uniprocessor; SIMD (Single Instruction, Multiple Data) describes a system where one instruction operates on many data items simultaneously, as in vector processors and GPUs; MISD (Multiple Instruction, Single Data), where multiple units process the same data stream with different instructions, is largely of theoretical interest; and MIMD (Multiple Instruction, Multiple Data) describes systems with multiple autonomous processors

executing different instructions on different data, which describes most modern multiprocessor and multicore systems.

Q4. Define instruction pipelining. Describe the stages of a typical five-stage pipeline.

Ans. Instruction pipelining is a technique that overlaps the execution of successive instructions by dividing instruction processing into a sequence of smaller stages, each handled by separate hardware, so that while one instruction is being decoded, another can be fetched, and so on — much like an assembly line.

A typical RISC processor uses a five-stage pipeline: IF (Instruction Fetch), where the instruction is fetched from memory using the program counter; ID (Instruction Decode), where the instruction is decoded and operand registers are read; EX (Execute), where the arithmetic/logic unit performs the required computation or address calculation; MEM (Memory Access), where data memory is read from or written to, if required; and WB (Write Back), where the result is written back into the destination register.

Once the pipeline is full, a new instruction completes every clock cycle even though each individual instruction still takes five cycles to pass through the whole pipeline, which is what gives pipelining its throughput advantage.

Q5. Explain the space-time diagram of a pipeline and derive the expression for speed-up due to pipelining.

Ans. A space-time diagram shows, for each clock cycle, which pipeline stage each instruction currently occupies; instructions are listed along one axis and clock cycles along the other, making the staggered, overlapped execution of instructions visually clear.

For a pipeline with k stages (each taking one clock cycle) executing n instructions: without pipelining, the total execution time is T(non-pipeline) = n × k cycles, since each instruction must fully complete before the next begins. With pipelining, the first instruction still takes k cycles, but every subsequent instruction finishes one cycle after the previous one, so the total time is T(pipeline) = k + (n − 1) cycles.

The speed-up due to pipelining is defined as S = T(non-pipeline) / T(pipeline) = (n × k) / (k + n − 1). As n becomes very large, S approaches k, the number of pipeline stages, which represents the ideal maximum speed-up; in practice, hazards and unequal stage delays keep the actual speed-up below this ideal value.

Q6. Explain the various techniques used to run a pipeline with minimum idling.

Ans. Pipeline idling (stalling) wastes clock cycles and reduces throughput, so several hardware and compiler-based techniques are used to minimise it.

These include: instruction scheduling/reordering by the compiler to separate dependent instructions; operand forwarding (bypassing), which supplies a result directly to an earlier stage instead of waiting for write-back; delayed branching, which fills the branch delay slot with useful work; branch prediction, which lets the processor guess a branch's outcome and keep fetching along the predicted path; providing multiple/parallel functional units to avoid structural hazards; out-of-order execution, which lets ready instructions bypass stalled ones; register renaming, which removes false name dependencies; and, in multi-stream designs, interleaving instructions from independent streams so a stall in one stream does not idle the whole pipeline.

Q7. Discuss why RISC architecture is well suited for pipelining.

Ans. RISC instruction sets were deliberately designed with pipelining in mind, and several of their defining features directly support smooth, efficient pipeline operation.

Fixed-length instructions let the Instruction Fetch stage always know exactly how many bytes to fetch without first decoding the instruction; a small number of uniform instruction formats keeps decoding fast and predictable; the load-store model means only a subset of instructions use the Memory Access stage meaningfully, simplifying pipeline control; near-uniform single-cycle execution keeps all pipeline stages balanced in duration; few addressing modes keep address computation in the Execute stage simple and fast; and a large register file reduces how often memory operands are needed, cutting down on stalls.

CISC instructions, being variable in length and often requiring multiple cycles, are comparatively difficult to pipeline efficiently — this was one of the key motivations behind the original move towards RISC design.

Q8. What are pipeline hazards? Explain the different types with their detection and minimization techniques.

Ans. A pipeline hazard is a situation that prevents the next instruction in the pipeline from executing in its designated cycle, potentially producing incorrect results if not handled, or forcing the pipeline to stall. There are three broad types.

Structural hazards occur when two instructions need the same hardware resource simultaneously; they are detected through resource-conflict checks during scheduling and minimised by duplicating hardware resources (such as separate instruction and data memories) or by stalling when duplication is not feasible.

Data hazards occur when an instruction depends on a result that a previous, still-executing instruction has not yet produced (most commonly a Read-After-Write hazard); they are detected by comparing source and destination register numbers of instructions in the pipeline, and minimised through operand forwarding, compiler-based instruction scheduling, register renaming, and, if unavoidable, inserting a pipeline stall (bubble).

Control hazards arise from branch and jump instructions, since the next instruction's address is uncertain until the branch is resolved; they are minimised through branch prediction, delayed branching, branch target buffers, and speculative execution with the ability to flush the pipeline if a prediction proves wrong.

Section C – Solved Numerical Problems

Q1. A processor has a 5-stage pipeline, and each stage takes one clock cycle of 10 ns. If 8 instructions are executed, find (a) the non-pipelined execution time, (b) the pipelined execution time, and (c) the speed-up due to pipelining.

Solution:

Given: k = 5 stages, n = 8 instructions, clock cycle = 10 ns.

(a) Non-pipelined time = n × k × cycle time = 8 × 5 × 10 = 400 ns.

(b) Pipelined time = (k + n − 1) × cycle time = (5 + 8 − 1) × 10 = 12 × 10 = 120 ns.

(c) Speed-up S = Non-pipelined time / Pipelined time = 400 / 120 = 3.33 (approximately).

Q2. A non-pipelined processor executes an instruction in 250 ns. It is redesigned into a 5-stage pipeline with equal stage delays. Find (a) the time per stage, (b) the pipeline cycle time, and (c) the time taken to execute 40 instructions on the pipelined processor.

Solution:

Given: total instruction time (non-pipelined) = 250 ns, k = 5 equal stages, n = 40 instructions.

(a) Time per stage = 250 / 5 = 50 ns.

(b) Pipeline cycle time = time of the slowest stage = 50 ns (since all stages are equal here).

(c) Pipelined execution time = (k + n − 1) × cycle time = (5 + 40 − 1) × 50 = 44 × 50 = 2200 ns.

Q3. For a 4-stage pipeline, plot (conceptually describe) the space-time occupancy and find the number of clock cycles needed to execute 6 instructions. Also calculate the speed-up.

Solution:

Given: k = 4 stages, n = 6 instructions.

Total clock cycles required (pipelined) = k + n − 1 = 4 + 6 − 1 = 9 cycles.

In the space-time diagram, instruction I1 would occupy stages S1–S4 in cycles 1–4, I2 would occupy S1–S4 in cycles 2–5, and so on, with each successive instruction starting one cycle after the previous one, until I6 occupies S1–S4 in cycles 6–9.

Non-pipelined cycles required = n × k = 6 × 4 = 24 cycles.

Speed-up S = (n × k) / (k + n − 1) = 24 / 9 = 2.67 (approximately).

Q4. A 5-stage pipelined processor runs at a clock frequency of 2 GHz. Out of 1000 instructions executed, 20% are branch instructions, each causing a stall of 2 clock cycles due to a control hazard. Assuming no other stalls, find the effective CPI and the total execution time.

Solution:

Given: clock frequency = 2 GHz → clock cycle time = 1 / (2×10^9) = 0.5 ns; total instructions n = 1000; branch instructions = 20% of 1000 = 200; stall per branch = 2 cycles.

Ideal CPI (no stalls) = 1 (typical for a RISC pipeline once it is full).

Total stall cycles = 200 × 2 = 400 cycles.

Total clock cycles = (ideal cycles for 1000 instructions) + stall cycles = 1000 + 400 = 1400 cycles (using the simplifying assumption of 1 cycle/instruction as the base rate).

Effective CPI = Total cycles / Total instructions = 1400 / 1000 = 1.4.

Total execution time = Total cycles × cycle time = 1400 × 0.5 ns = 700 ns.

Q5. In a program of 500 instructions, 10% are LOAD instructions that are immediately followed by an instruction depending on the loaded value, causing a 1-cycle stall (data hazard) each time forwarding cannot fully resolve it. Find the total number of stall cycles and the effective CPI, assuming an ideal CPI of 1.

Solution:

Given: total instructions n = 500; LOAD instructions causing a dependent stall = 10% of 500 = 50; stall per occurrence = 1 cycle.

Total stall cycles = 50 × 1 = 50 cycles.

Total clock cycles = ideal cycles + stall cycles = 500 + 50 = 550 cycles.

Effective CPI = Total cycles / Total instructions = 550 / 500 = 1.1.