Advanced Computer Architecture Lab 3: Pipelining the Processor

Question 1: Harvard Architecture

The new processor adopts the Harvard architecture, which separates instruction and data memory.

Advantages:
- Removes contention between instruction fetch and data access, eliminating a class of structural hazards.
- Enables parallelism since instructions and data can be accessed in the same cycle.
- Provides higher potential bandwidth with two independent memory ports.

Disadvantages:
- Increases hardware complexity and cost (two memories instead of one).
- Requires additional effort to maintain coherence between instruction and data memories.
- Less flexible if program demands are unbalanced between instruction and data usage.

Von Neumann alternative:
- A single memory simplifies design and reduces hardware cost.
- However, it introduces the Von Neumann bottleneck, where only one access (instruction or data) is possible per cycle, stalling the pipeline.

Conclusion: The Harvard architecture is the better choice for a pipelined design because it reduces structural hazards and improves performance.

Question 2: Structural and Data Hazards

Introducing pipelining brings new hazards compared to the simple state machine model.

Structural Hazards:
- Instruction vs. Data memory: avoided by Harvard separation.
- Register read/write conflicts: managed with careful staging.

Data Hazards:
- Read After Write (RAW): Occurs when an instruction depends on the result of a previous instruction. Resolved using forwarding from EXEC0/EXEC1 to DEC1. If forwarding is insufficient (e.g., load-use case), a stall is inserted.
- Write After Write (WAW): Avoided as writes occur in order in EXEC1.
- Write After Read (WAR): Also avoided due to strict pipeline order.

Resolution mechanisms:
- Forwarding paths ensure dependent instructions can use results before they are written

back.
- Stalling handles cases where data is unavailable until the following cycle.

Example: A load-use hazard:
LD r1, [r2]   ; result available only at EXEC1
ADD r3, r1, r4   ; must stall one cycle before execution

Question 3: Branches and Branch Prediction

Branches introduce control hazards.

Prediction scheme:
- Simple static scheme (predict "not taken") or a dynamic 1-bit predictor.

Branch handling:
- On fetch, prediction provides a target PC.
- On resolution in EXEC0, the actual outcome is known.
- If prediction was wrong, the pipeline flushes instructions in earlier stages and restarts at the correct PC.

Pipeline flushes:
- Misfetched instructions are invalidated.
- Causes bubbles (lost cycles) until new instructions are fetched.

Timing impact: If resolved at EXEC0, up to three stages may be flushed.

Question 4: Low-Level Simulator Pipeline Implementation

The pipeline simulator includes the following stages:
- F0 (FETCH0): Initiates instruction fetch.
- F1 (FETCH1): Samples instruction.
- D0 (DEC0): Decodes opcode, dst, src0, src1, immediate.
- D1 (DEC1): Prepares operands.
- E0 (EXEC0): Executes ALU and LD.
- E1 (EXEC1): Writes back results and handles ST.

Implementation notes:
- Each stage has its own register copy (e.g., dec0_inst, exec0_opcode).
- The simulator generates two traces:
  - Instruction trace: Matches ISS output from Lab 1.
  - Cycle trace: Shows pipelined, overlapped execution (shorter than Lab 2/4).
- Includes integrated DMA functionality.

Question 5: Low-Level Testing

Test programs:
- add.bin: Adds two signed numbers.
- sqrtq.bin: Computes quotient portion of a square root.

Expected outputs:
- inst_trace.txt: Matches ISS from Lab 1.
- cycle_trace.txt: Demonstrates overlapped pipelining.
- sramd: Matches memory output of Lab 1.

Question 6: Speedup Comparison

1. Speedup calculation:
   Speedup = (Execution time of Lab 2) / (Execution time of Lab 3)
   Example: If Lab 2 takes 60 cycles and Lab 3 takes 15 cycles, speedup = 4.

2. Is CPI = 1?
   - Not exactly. While the ideal goal is CPI=1, hazards, stalls, and flushes push average CPI above 1.

3. Next generation improvements:
   - Deeper pipelines and superscalar execution.
   - Better branch prediction (2-bit, BTB).
   - Out-of-order execution.
   - Use of caches instead of plain SRAM.

Question 7: DMA Testing in a Pipeline Environment

Test design:
- Construct a program with DMA transfers concurrent with CPU operations.
- Include overlaps between instruction fetch and data memory access.

Hazard coverage:
- Ensures cases where DMA and CPU both access memory.
- Tests load/store operations during DMA transfers.
- Covers pipeline stalls and hazard resolutions.

Verification:
- dma_pipe_cycle_trace.txt: Shows correct hazard management during DMA.
- dma_pipe_sramd_out.txt: Confirms correct data transfers.
- Annotated assembly highlights overlap points.

Conclusion

The pipelined SP processor significantly improves performance by adopting Harvard architecture, hazard resolution mechanisms, and branch prediction. Testing demonstrates functional correctness and notable speedup compared to the non-pipelined design. Future improvements can push CPI closer to 1 and potentially beyond using superscalar and out-of-order techniques.