EE309 - Microprocessors and Computer Architecture

Pipelined CPU Implementation

Team Members

23B1277 B Yaswanth Ram Kumar

23B1229 K Sri Charan Raj

Under the guidance of

Prof. V. Rajbabu

Department of Electrical Engineering

Indian Institute of Technology Bombay

May 3, 2025

Contents

1 Introduction

This report documents the design and implementation of a 5-stage pipelined processor developed as a course project for EE309 under the guidance of Prof. V. Rajbabu from the Department of Electrical Engineering, IIT Bombay. This project builds upon our previous work on a RISC Multi-Cycle CPU developed during the Digital Systems course in Autumn 2024. The current implementation extends the architecture to a fully pipelined 16-bit CPU featuring multiple independent execution pipelines.

2 Instruction Set Architecture

The processor utilizes a 16-bit instruction format and features eight general-purpose registers, each 16 bits wide. The design incorporates separate instruction memory (IMEM) and data memory (DMEM), allowing independent storage and access for programs and data. The DMEM operates as a queue: all LOAD instructions access data from the tail of the queue, while STORE instructions add data to the head.

2.1 Instruction Format

Instruction	Opcode	Operands	Description
LW	0000	Ra	Load word from DMEM tail to register Ra
SW	0001	Ra	Store word from register Ra to DMEM head
ADD	0010	Ra, Rb, Rc	Ra $\leftarrow$ Rb + Rc
SUB	0011	Ra, Rb, Rc	Ra $\leftarrow$ Rb - Rc
MUL	0100	Ra, Rb, Rc	Ra $\leftarrow$ Rb $\times$ Rc
ADDI	0101	Ra, Rb, 6-bit Imm	Ra $\leftarrow$ Rb + Immediate
SLL	0110	Ra, Rb, Rc	Ra $\leftarrow$ Rb $\ll$ (Rc[3:0])
JRI	0111	Ra, 9-bit Imm	PC $\leftarrow$ Ra + 2 $\times$ Immediate

Table 1: Processor instruction set with opcodes and operations

3 Pipeline Architecture

3.1 Pipeline Overview

The processor features 5 independent execution pipelines (`pl1` through `pl5`), each capable of processing instructions through all five stages of execution. Instructions are issued in a round-robin fashion across these pipelines:

- Instruction 1 $\rightarrow$ `pl1`
- Instruction 2 $\rightarrow$ `pl2`
- ...
- Instruction 6 $\rightarrow$ back to `pl1`, and so on

Each instruction progresses through five stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). The pipelines operate independently but identically, simply staggered in their start times.

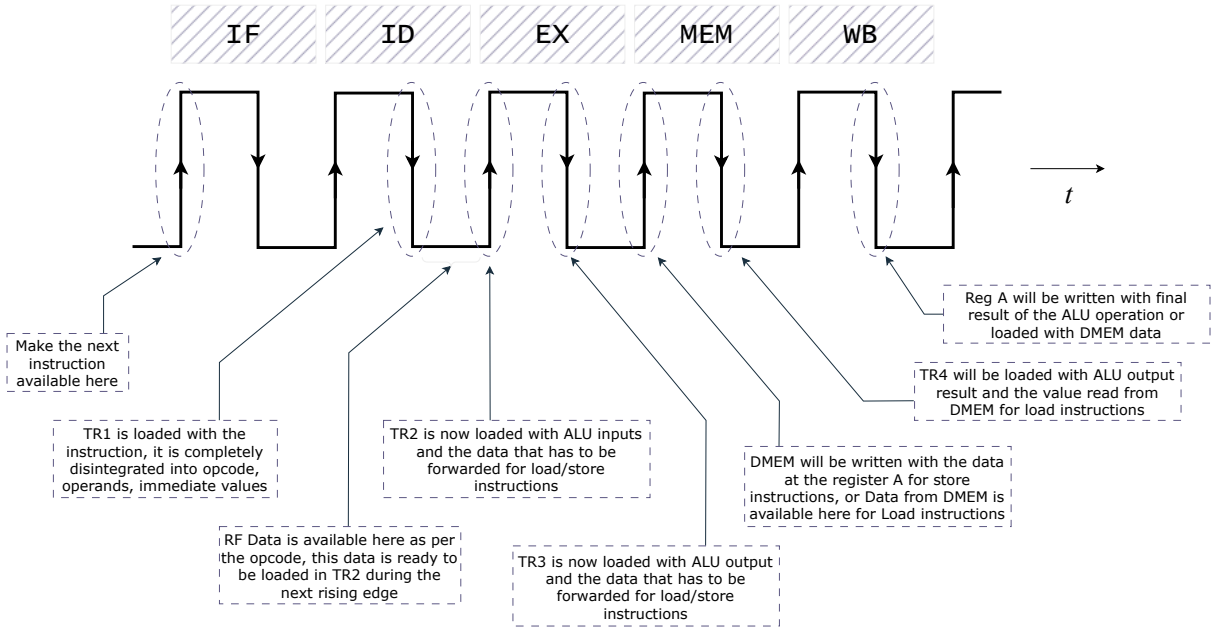

Figure 1: Edge wise Procedure

3.2 Pipeline Stages

3.2.1 Instruction Fetch (IF)

- **Rising edge**: The instruction appears at `imem_out` from the ring buffer (IMEM).
- **Falling edge**: The instruction is input into the `TR1` decoder.
- PC is updated at the falling edge for the pipe that just fetched.
  - For normal cases: `PC ← PC + 1`
  - For `JRI`: `PC ← regA + 2 × imm`

3.2.2 Instruction Decode (ID)

- **Rising edge**: State transitions from IF to ID.
- **Falling edge**:
  - Register file is read based on the opcode in `TR1` output.
  - Multiplexer select lines (`mux2x1_3bit` and `mux_alub_sel`) are determined.

3.2.3 Execute (EX)

- **Rising edge**: State transitions to EX and `TR2` is loaded.
- ALU operations are performed during this phase.
- **Falling edge**:
  - `TR3` is loaded with the ALU result.
  - For load/store instructions, read/write enables for `dmem` are set.

3.2.4 Memory Access (MEM)

- **Rising edge**: Memory access occurs for load/store instructions.
- **Falling edge**:
  - `TR4` is loaded with either memory data or forwarded ALU output.
  - `dmem` read/write enables are reset to `'0'`.

3.2.5 Write Back (WB)

- **Rising edge**:
  - Register file write enable is asserted.
  - Destination select line (`dest_mux_sel`) is set based on instruction type.
- **Falling edge**:
  - Register A is written back with the selected value.

4 Implementation Details

4.1 Instruction Lifecycle

Each instruction requires 5 clock cycles to complete, regardless of instruction type. Even store instructions that do not perform work in the WB stage still occupy the pipeline for the full 5 cycles to maintain consistent timing.

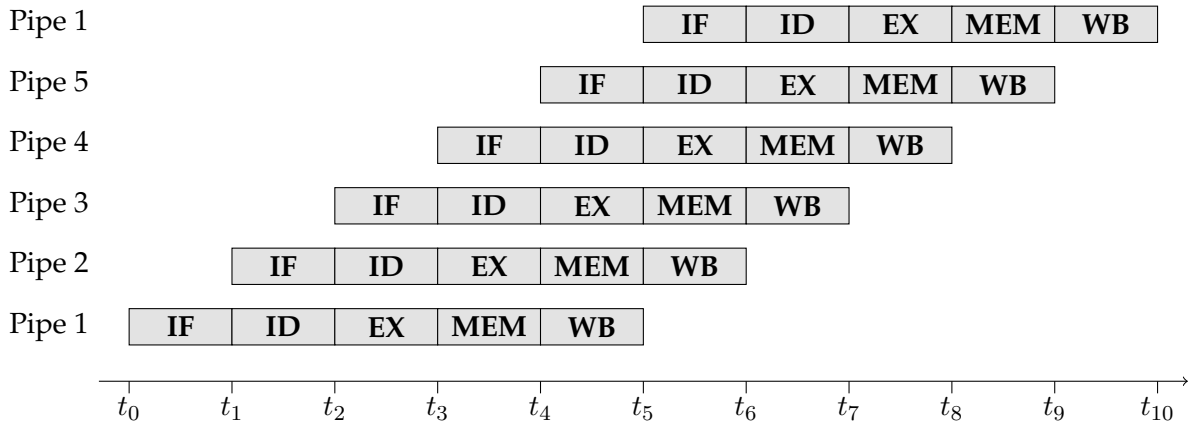

Figure 2: Pipeline execution timing diagram showing round-robin instruction issuing

4.2 Key Components

4.2.1 Ring Buffer (IMEM)

- Functions as the instruction memory.
- Operates at the rising edge of the clock.
- Pre-implemented and used as-is in the design.

4.2.2 Transition Registers (TR1-TR4)

- `TR1`: Serves as the instruction decoder and operates at the falling edge.
- `TR2`: Captures register values and control signals at the rising edge.
- `TR3`: Stores ALU results at the falling edge.
- `TR4`: Stores memory access results at the falling edge.

4.2.3 Register File

- Performs combinational reads based on decode output.
- For `store`, `load`, and `jri` instructions, `regA` is read via `rfa1`.
- All other instructions read from `regB` only.
- Register writes occur at the falling edge of the clock.

4.2.4 Datapath

- Contains the ALU and data routing logic.
- Pre-implemented and used as-is in the design.

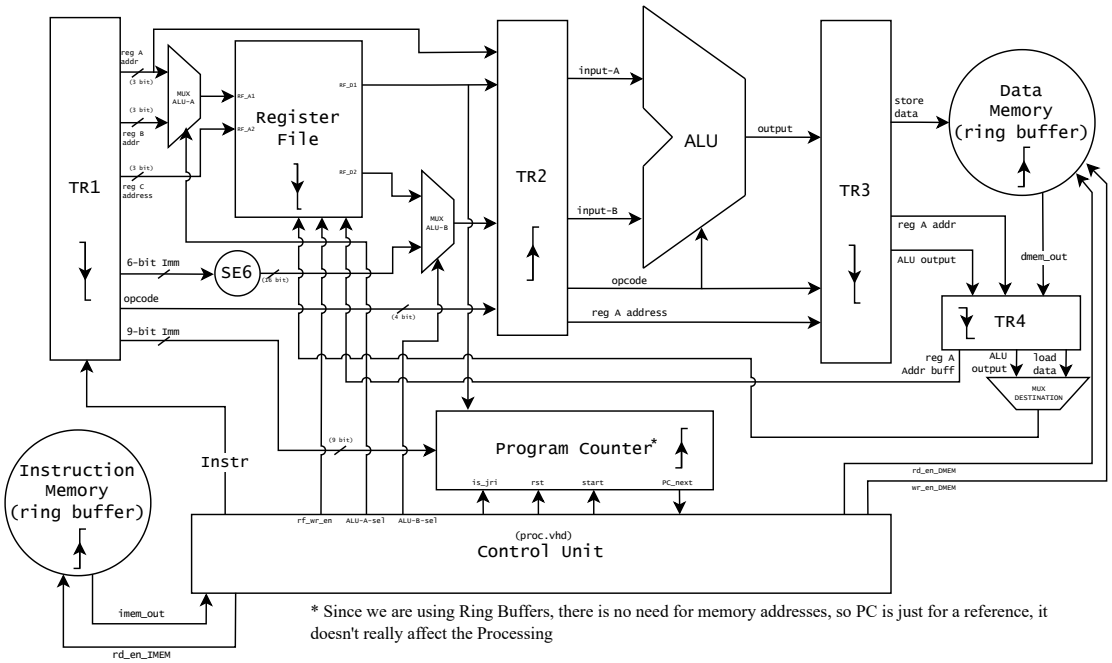

Figure 3: Pipelined Datapath

4.3 Control Logic

The control logic for all pipelines is identical, just staggered in time. To minimize redundancy, the design employs:

- Generic pipeline stage variables
- Looping constructs to handle similar logic across pipelines
- Unified control signal generation

4.4 Additional Implementation Notes

- No hazard detection or forwarding mechanisms are implemented in this design.
- All pipelines are initialized with `valid <= '0'` at reset, representing the NOP state.
- Each pipeline stage's control paths are activated only during the appropriate cycle.

5 Conclusion

The implemented 5-stage pipelined processor successfully extends our previous multi-cycle design with parallel execution capabilities. The round-robin instruction issuing across five independent pipelines allows for improved throughput while maintaining a clean, modular design. Each instruction progresses through all five stages (IF, ID, EX, MEM, WB) in a controlled manner, with appropriate timing for register accesses, ALU operations, and memory interactions.

Our performance analysis demonstrates significant improvement:

- **Baseline Multi-cycle Implementation:**
  Execution time = Clock period (10ns) × 5 stages × 5 instructions = 250ns
- **Pipelined Implementation:**
  Execution time = Clock period (10ns) × [5 stages + (instructions - 1)] = 90ns

This represents a **speedup of 2.78×**, clearly demonstrating the efficiency advantages of pipelining.

This implementation demonstrates our understanding of pipelined processor architectures and provides a foundation for further enhancements such as hazard detection, data forwarding, and branch prediction in future iterations.

6 Future Work

Potential enhancements to the current design could include:

- Implementation of hazard detection and data forwarding mechanisms
- Branch prediction to reduce control hazards
- Extension of the instruction set for more complex operations
- Performance analysis and optimization of critical paths

Thank You