Project Proposal: Pipelining and Hazard Handling in a CPU Simulator

Rishit Mittal
Roll No.: CS24BTECH11053

October 2025

1 Project Objective and Overview

The core objective is to **extend the existing in-house single-cycle processor simulator** to support a high-performance architecture based on a **five-stage pipeline** (IF, ID, EX, MEM, WB). This involves partitioning the current logic, introducing pipeline registers, and implementing various hazard detection and resolution mechanisms.

The goal is to move the simulator from a single-cycle model to a multi-cycle pipelined model to quantitatively demonstrate the impact of pipelining on performance, measured primarily through a reduction in the total cycle count for benchmark programs.

Course	CS2323
Student	Rishit Mittal
Roll No.	CS24BTECH11053

2 Scope of Work

The project scope includes fundamental modifications to the simulator's execution core to support multi-stage processing and comprehensive hazard control.

2.1 Core Implementation

- **Five-Stage Pipelining**: Implementation of the Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB) stages.
- **Pipeline Registers**: Introduction and management of data passing through pipeline registers between stages.

2.2 Configurable Pipelined Modes (Hazard Control)

The simulator must offer options to switch between different execution modes for comparative analysis:

1. **No Pipeline**: The existing single-cycle baseline.
2. **Pipelined (No Hazard Control)**: Basic five-stage pipeline without any stall or forwarding logic (expected to yield incorrect results with data hazards).
3. **Pipelined with Stalling**: Implements a Hazard Detection Unit to insert NOPs/bubbles for resolution, but **no forwarding**.
4. **Pipelined with Forwarding**: Implements both hazard detection and **data forwarding** (bypassing) to resolve most data hazards efficiently.
5. **Pipelined with Forwarding and Static Branch Prediction**: Includes a static prediction scheme (e.g., predict-not-taken) for control hazard mitigation.

6. **Pipelined with Forwarding and Dynamic Branch Prediction**: Incorporates a basic **1-bit dynamic branch predictor** for advanced control flow management.

3 Development Approach

The development will be executed in a phased approach, building complexity sequentially from the baseline simulator.

1. **Repository Study and Partitioning**: Thoroughly analyze the structure of the existing in-house simulator repository on Git. The single-cycle logic will be carefully partitioned and mapped onto the five required pipeline stages.
2. **Basic Pipelining Core**: Refactor the code to integrate pipeline registers and implement the clocking mechanism to support basic concurrent instruction execution (Mode 2).
3. **Hazard Resolution Extension**: Implement the specific control units for stalling (Mode 3), data forwarding paths (Mode 4), and branch prediction logic (Modes 5 and 6).
4. **Verification and Benchmarking**: Use a set of assembly codes designed to stress-test hazards. Monitor the correctness and quantitatively compare the cycle count for each of the six implemented modes.

4 Verification Methodology

Verification will be dual-faceted, focusing on both **functional correctness** and **performance assessment**.

4.1 Correctness Verification

- **Baseline Comparison**: The final register and memory state after running an assembly program in any pipelined mode (Modes 3 through 6) must exactly match the state produced by the trusted **No Pipeline (Mode 1)** single-cycle implementation.
- **Hazard Testing Suites**: Specific assembly code segments will be created to intentionally induce different types of hazards (RAW Data Hazards, Control Hazards) to ensure:
  - Mode 2 (No Hazard Control) produces an incorrect result (register/memory mismatch).
  - Modes 3, 4, 5, and 6 successfully resolve the hazards and produce the correct result.

4.2 Performance Verification

- **Cycle Count Tracking**: The simulator will be instrumented to accurately count the total number of clock cycles required for execution in each mode.
- **Quantitative Analysis**: Performance gain will be measured by comparing the cycle counts of the hazard-controlled modes (Modes 3-6) against the baseline (Mode 1) and the theoretically fastest case (ideal CPI = 1).

5 Deliverables, Team, and Timeline

5.1 Deliverables

- **Updated Git Repository**: The final, working source code containing all six pipeline configurations.
- **Simulator Utility**: A functional Python file or executable demonstrating the running of assembly code under all specified modes.
- **Performance Analysis Report**: A formal document analyzing the quantitative performance results (cycle count comparison) and design choices.
- **Optional GUI/Visualization Support**: If time permits, implementation of APIs or a simple script to visualize the pipeline flow.

5.2 Team Structure

This is done by me individually.
Name : **Rishit Mittal**.

5.3 Timeline (Approx. 35 Days)

The project is scheduled for completion within an accelerated period of approximately 35 days (5 weeks).

Table 1: Estimated Project Timeline (Days)

Phase	**Duration (Days)**	**Key Focus**
Analysis & Partitioning	7	Code structure study; logic segmentation.
Core Pipelining & Mode 2	10	Pipeline registers; basic pipelined mode.
Hazard Control (Modes 3, 4)	10	Stalling and Forwarding implementation.
Advanced Hazards & Testing	8	Branch Prediction (Modes 5, 6); verification and reporti
Total Estimated Days	**35**	