RISC -V 32 Bit Processor

Ben Feng & Leo Wang

Cognichip Hackathon: [Risc-V 32 Processor]

Team Name: Pro

Team Members (2-4 per team):
[Ben Feng, 2026, zf2179@nyu.edu, Member]
[Leo Wang, 2026, tw2567@nyu.edu, Captain]

Problem Domain:
Traditional CPU design relies heavily on manual RTL coding, iterative debugging, and extensive handwritten testbenches to verify correctness. This process is slow, error-prone, and highly dependent on human inspection of waveforms and control signals. Even for a relatively simple 5-stage in-order RISC-V 32 processor, ensuring correct hazard handling, control flow, datapath connectivity, and pipeline timing requires large amounts of repetitive coding and comprehensive test coverage. Writing sufficient directed and corner-case testbenches is time-consuming, and missing even a small scenario can result in undetected functional bugs. Under limited development time, the conventional hardware design and verification flow becomes a bottleneck, reducing productivity and increasing the risk of architectural or logical mistakes.

Proposed Innovation:

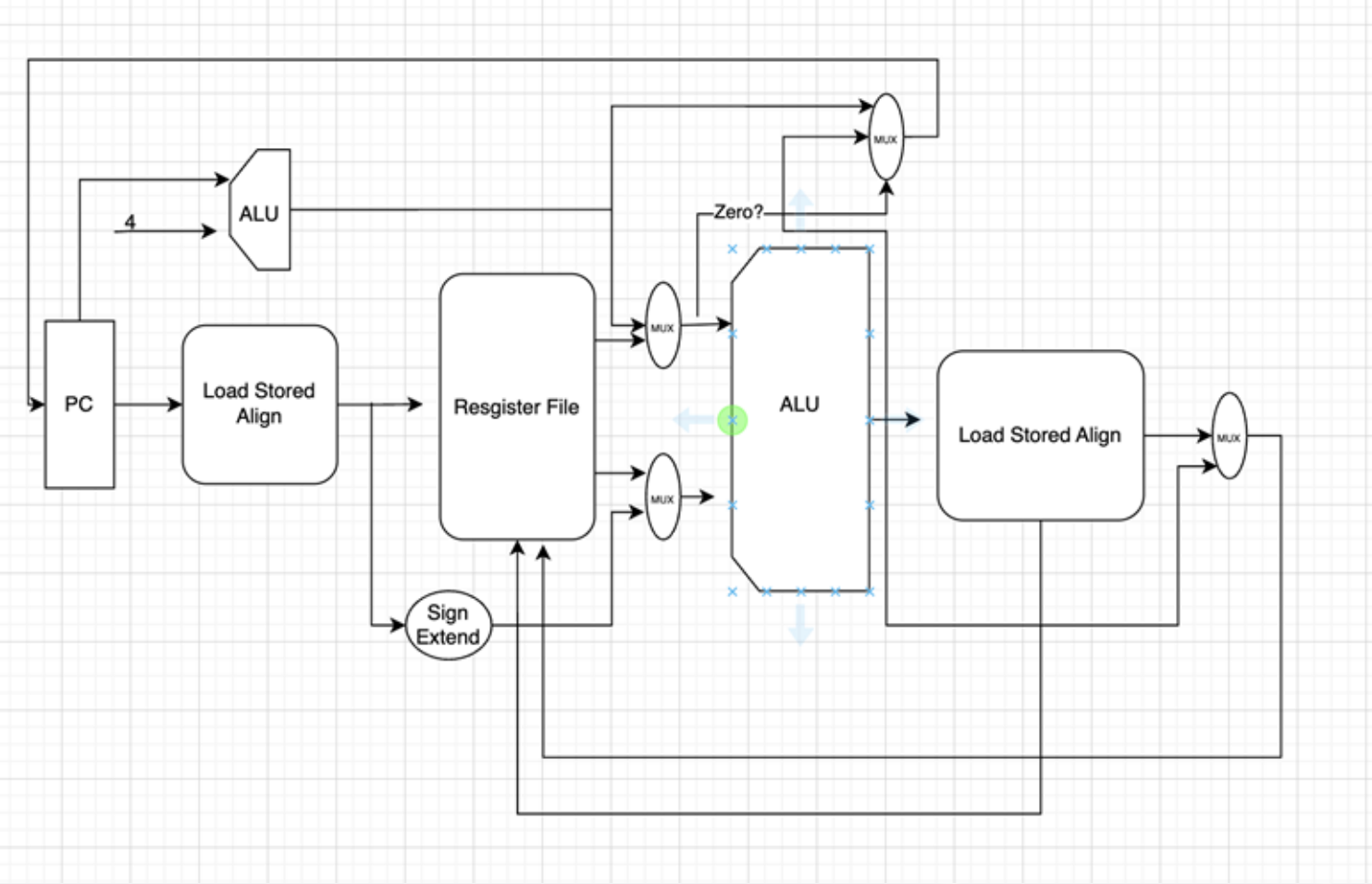
This project focuses on designing, from the ground up, a 5-stage in-order RISC-V 32 processor based on the classic IF–ID–EX–MEM–WB pipeline. The objective is to build a fully functional core that correctly implements instruction decoding, ALU operations, memory access, hazard detection, data forwarding, and branch control within limited development time. Rather than increasing architectural complexity, the emphasis is on structural correctness, clean modular design, and reliable execution across different instruction scenarios.

Beyond simply implementing the processor, the key innovation is integrating AI as a design and verification partner throughout the development flow. AI will be used to generate and refine RTL modules such as the ALU, control unit, forwarding unit, hazard detection logic, and pipeline registers, ensuring consistent interfaces and reducing manual coding overhead. It will also help explore alternative hazard-handling strategies and forwarding structures, comparing their impact on stall cycles and hardware cost.

In parallel, AI will automatically generate directed and randomized self-checking testbenches that stress RAW hazards, load-use conditions, branch behavior, and corner cases that are often missed in manual testing. Through iterative interaction, these AI-assisted designs will be implemented in SystemVerilog, and the top-level architecture will be reviewed to verify correct signal connectivity and pipeline data flow.

As a stretch goal, we plan to integrate a lightweight reference checking mechanism that compares architectural register states after execution, further strengthening verification. This approach demonstrates how AI can significantly accelerate CPU development while improving correctness and test coverage.

Archetecture



IF: Program Counter

ID: Load Stored Unit, Register File

EX: ALU

MEM: Load stored Unit

WB: Mux selection

Simulation results

14 tests were conducted to verify addition, subtraction, multiplication, and other operations, and all 14 tests were successfully passed.

Challenge & Lesson Learned

- AI may not be able to fully understand the request.
- The generated RTL and testbench contain errors and require manual review
- AI is a helpful assistant

Github Link

https://github.com/BenFeng666/OoO-Design-Project/tree/Congichip-Correct

Thank you