A black background with blue text

AI-generated content may be incorrect.

**Milestone 2**

**Computer architecture**

Youssef attalah 900221247

Youssef Nader 900226170

**1. Introduction**

This project involves the implementation of a single-cycle processor that supports all 42 base instructions from the RV32I RISC-V Instruction Set Architecture. The processor is designed and implemented using Verilog HDL and tested using waveform simulations.

The processor supports all 42 instructions categorized under R, I, S, B, U, and J types. These include arithmetic, logical, memory access, control transfer, and immediate instructions. This is milestone 3 which includes pipeline architecture with a single memory.

**3. Proposed Datapath Design**

The pipelined architecture is structured to process instructions over multiple stages with register-based isolation to enable parallelism. Instructions are issued every two cycles to accommodate a single-ported memory.

The datapath consists of the following major components:

* **Program Counter (PC)**: Holds the address of the current instruction. It is updated either sequentially (PC + 4) or by a branch/jump target depending on control signals.
* **Single Memory Module**: A unified memory block (SingleMem) that handles both instruction fetch and data access, serialized across clock cycles to avoid conflicts.
* **Single Memory Decoder**: Distinguishes between instruction and data access based on control logic and memory content.
* **Clock Divider**: Generates a slower, stable clock used to time pipeline stages and memory access across two cycles per instruction issue.
* **IF/ID, ID/EX, EX/MEM, MEM/WB Pipeline Registers**: Maintain intermediate values and control signals between pipeline stages to preserve instruction data across clock cycles.
* **Register File**: Contains 32 general-purpose registers. Supports simultaneous read of two registers and write to one register per instruction cycle.
* **ALU (Arithmetic Logic Unit)**: Executes arithmetic and logic operations. Controlled by the ALU control unit and supports flag outputs (Zero, Carry, Sign, Overflow) for conditional logic.
* **ALU Control Unit**: Maps instruction fields (funct3, funct7, ALUOp) to specific ALU operations.
* **Immediate Generator**: Decodes and sign-extends immediate values from various instruction types (I, S, B, U, J).
* **Control Unit**: Decodes the opcode and generates control signals for the entire datapath, such as ALU source selection, memory access, register write-back, and branching logic.
* **Branch Control Unit**: Evaluates conditional branch instructions using ALU flags and controls flushing of the pipeline when a branch is taken.
* **Hazard Detection Unit**: Monitors pipeline state and stalls the pipeline to resolve load-use hazards.
* **Forwarding Unit**: Enables data forwarding between EX, MEM, and WB stages to minimize stalls from RAW dependencies.
* **Multiplexers (MUXes)**: Used throughout the datapath to select inputs for ALU, PC update, memory access, and write-back values.

**Instruction Flow through the Pipeline**

* **Fetch**: PC fetches the instruction from memory; IF/ID stores the instruction and PC.
* **Decode**: Control signals are generated; register operands and immediate values are read into ID/EX.
* **Execute**: ALU performs operations; branch logic determines whether a flush and PC change are needed.
* **Memory**: If applicable, memory operations (load/store) are conducted via SingleMem and decoded by the memory decoder.
* **Write-back**: Register file is updated from either ALU or memory output, depending on instruction type.

**Program Execution Summary**The overall execution of a program on this pipelined RISC-V processor involves the sequential processing of instructions through fetch, decode, execute, memory access, and write-back stages. Each instruction advances through the pipeline with the help of dedicated pipeline registers, allowing multiple instructions to be in different stages simultaneously. Hazards are detected and resolved dynamically to ensure correctness, while forwarding and stalling mechanisms maintain efficient instruction throughput. The program continues executing until a halting instruction is encountered, effectively stopping the program counter.

This pipelined architecture significantly enhances throughput while maintaining correctness via hazard detection and resolution mechanisms.

**5. Issues faced and solutions**

**Debugging**

Debugging took a full day of trying different test cases

**Tracing**

Tracing was difficult across many modules