Arab Academy of Science and Technology

MIPS Processor Design

Date:

May 2024

Lecturer:

Dr. Marwa Abdelrazik

Table of Contents

Introduction

This project focuses on the design and implementation of a simple MIPS processor using VHDL, with an emphasis on pipeline architecture to improve performance. The pipeline technique allows the processor to execute multiple instructions simultaneously by breaking down the execution process into distinct stages. This approach enhances the instruction throughput and overall efficiency of the processor. The implemented processor supports basic operations including data movement, input/output, multiplication, branching, and instruction skipping. Hazard detection mechanisms are incorporated to handle potential conflicts during instruction execution in the pipeline.

Research Questions

- How does the implementation of a MIPS processor using VHDL impact overall system efficiency compared to other design approaches?
- What are the advantages and limitations of incorporating a pipeline architecture in the design of a MIPS processor?
- How do hazard detection mechanisms, such as data forwarding and branch prediction, contribute to improving the performance of the pipeline architecture?
- What are the key factors influencing the choice of components and their configurations in designing a MIPS processor with VHDL?
- How does the complexity of the instruction set influence the design and implementation of a MIPS processor?
- What are the key challenges encountered during the design, implementation, and testing phases of a MIPS processor project using VHDL?

Addressing Performance, Simplicity, and Efficiency: The MIPS Architecture's Solution

The MIPS (Microprocessor without Interlocked Pipeline Stages) architecture was designed to address the challenge of achieving high performance and efficiency in computer processors while maintaining simplicity and ease of implementation. MIPS aimed to streamline instruction execution and optimize processor performance by focusing on a smaller set of instructions with uniform formats and reduced complexity. This design philosophy aimed to simplify the processor architecture, enabling faster execution, reduced power consumption, and improved scalability. Additionally, the pipeline architecture of MIPS processors allowed for parallel execution of instructions, further enhancing performance by overlapping different stages of instruction processing.

Methodology

1. Implementing Processor Components

- **Register File**
- **Memory**
  - **Data Memory**
  - **Instruction Memory**
- **ALU (Arithmetic Logic Unit)**
- **ALU control**
- **Control Unit**
- **Multiplexers**
- **Pipeline Registers**
- **Adder**

2. Testing Individual Components

- Each component is tested individually using VHDL test benches to verify its functionality. Simulations are performed to ensure correct behavior under various conditions and edge cases.

3. Implementing the Data Path

- The data path integrates all components into a cohesive processor architecture. It is designed to handle the following pipeline stages: *Instruction Fetch (IF)*, *Instruction Decode (ID)*, *Execute (EX)*, *Memory Access (MEM)*, and *Write Back (WB)*.
- Proper connections between components are established to ensure data and control signals flow correctly through the pipeline stages.

4. Initializing Memory with Instructions

- The memory is initialized with a simple program consisting of various instructions to test the processor's capabilities. This program includes operations like *MOV*, *INP*, *OUT*, *MUL*, *BUN*, and *SKP*.

5. Testing the Data Path

- A comprehensive test bench is developed for the entire processor. The processor is simulated to verify the correct execution of instructions and proper operation of the pipeline.
- The test includes checking the proper functioning of instruction fetch, decode, execution, memory access, and write-back stages.

6. Implementing Hazard Detection

- Hazard detection mechanisms are implemented within the control unit or a dedicated hazard detection unit.

Processor Operations

OPCODE	Name	Description
1	MOV	Move data between registers or between a register and memory
2	INP	Input a value from the user and save it in a register
3	OUT	Output the data from a register to the screen
4	MUL	Multiply the content of memory with register content
5	BUN	Use the effective address as the address of the next instruction
6	SKP	Skip the next instruction if the memory content is positive

Design and Implementation

1. Registers

- The register file consists of 32 general-purpose registers. Each register can hold a 32-bit value.
- The register file supports read and write operations controlled by the control unit.

2. Memory

Instruction Memory

- A memory unit specifically designed for storing instructions. It allows read operations to fetch instructions for execution.

Data Memory

- A memory unit for storing data, supporting both read and write operations. This memory is used during the Memory Access (MEM) stage of the pipeline.

4. ALU

- The ALU performs essential arithmetic and logical operations. It supports operations like addition, subtraction, multiplication, and logical AND/OR.

5. Control Unit

- The control unit decodes instruction opcodes and generates control signals to manage the data path and ALU operations.
- It handles the sequencing of operations and the control of multiplexers and pipeline registers.

6. Multiplexers (MUX)

- Multiplexers are used to select data paths based on control signals. For example, selecting between immediate values and register values for ALU operations.

7. Sign Extender

- The sign extender takes immediate values from instructions and extends their sign to match the 32-bit width of the data path, ensuring correct arithmetic operations.

8. Pipeline Registers

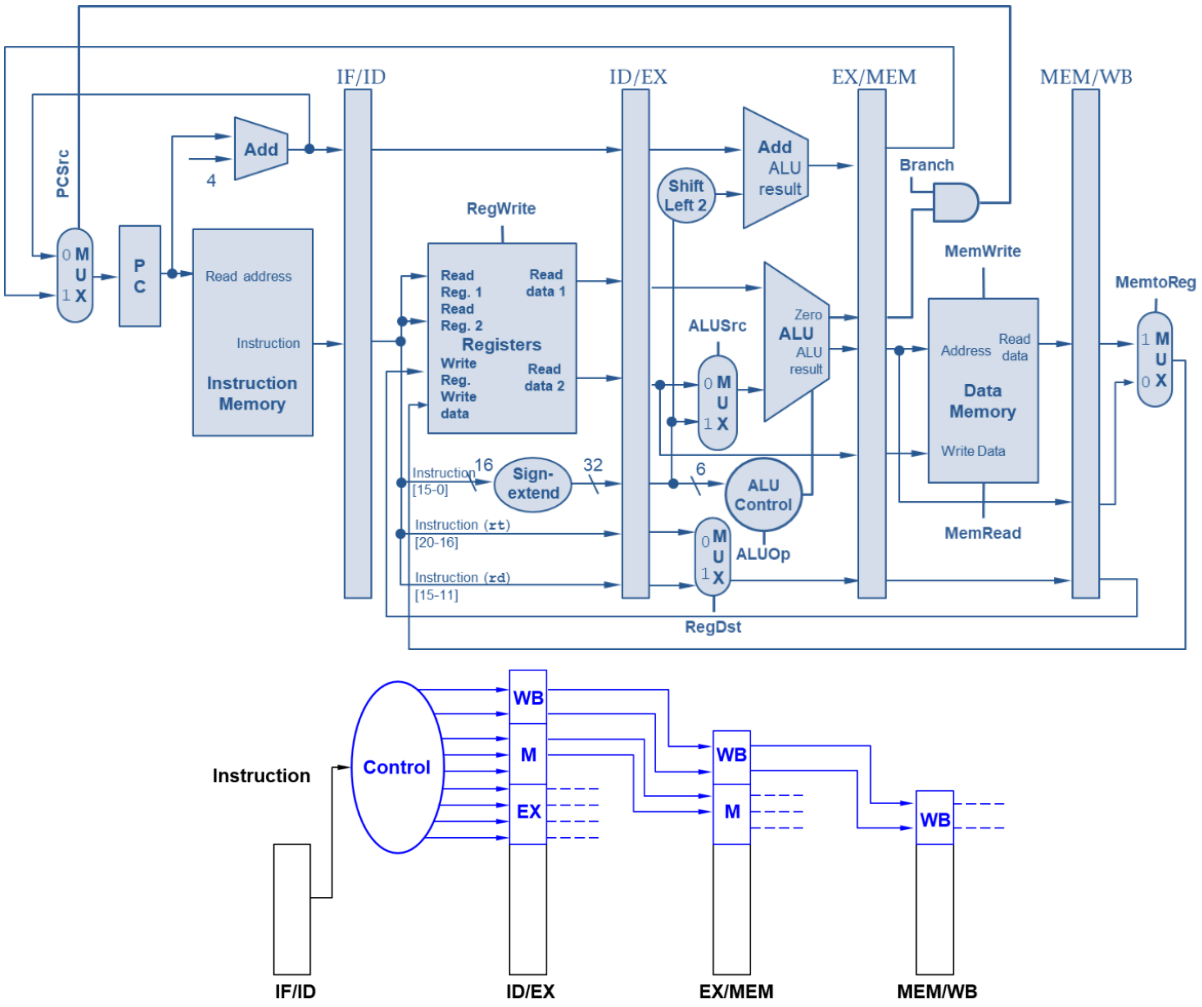
- Pipeline registers hold intermediate values between pipeline stages, ensuring data consistency and synchronization across stages.

9. Adder

- The adder is used for calculating the next program counter (PC) value, typically incrementing the current PC by 4 to point to the next instruction.

10. ALU Control:

- The ALU control unit generates control signals for the ALU based on the opcode of the instruction being executed.



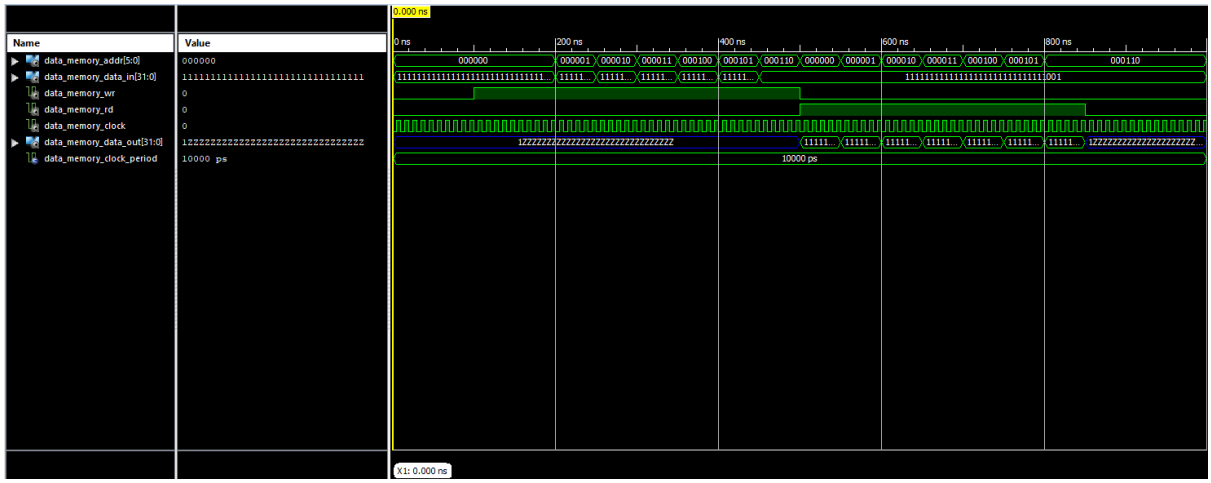
Testing and Results

1. Component Testing

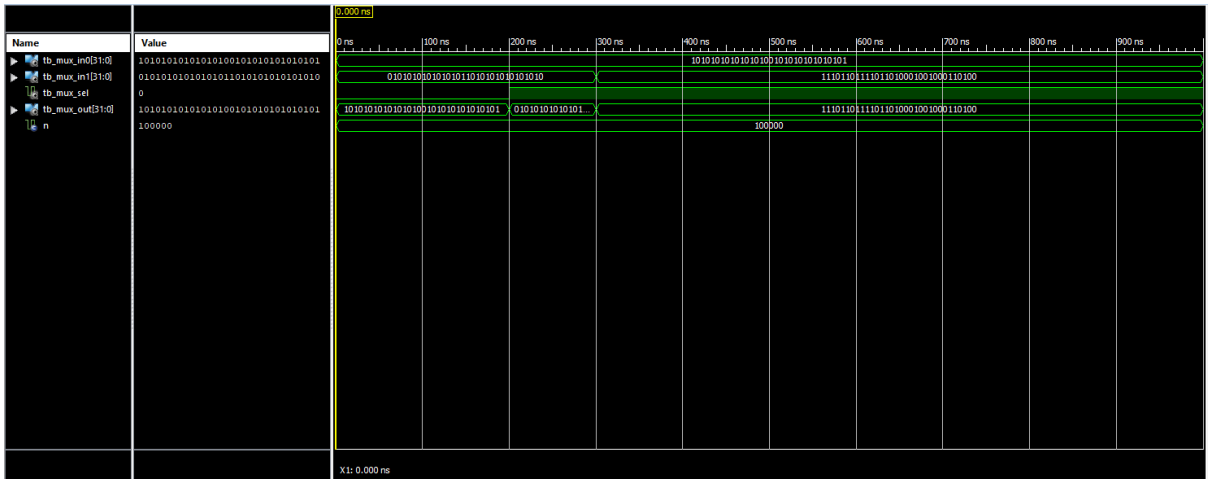
- Each component was tested using dedicated VHDL test benches. Simulations confirmed correct functionality for registers, instruction memory, data memory, ALU, ALU control, multiplexers, sign extender, adder, and the control unit.

  - Instruction Memory:

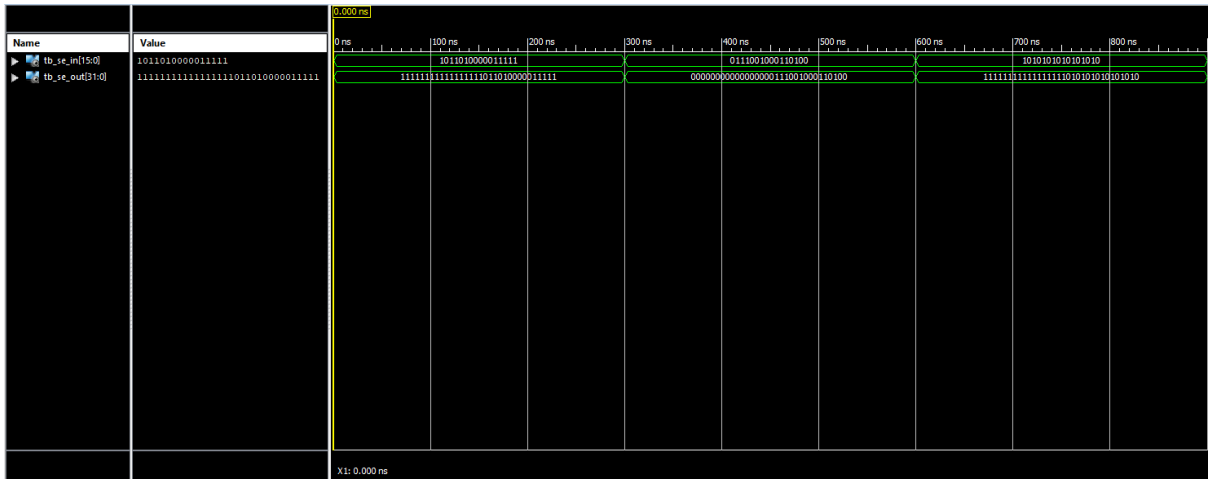
  - Data Memory:



- Multiplexers:

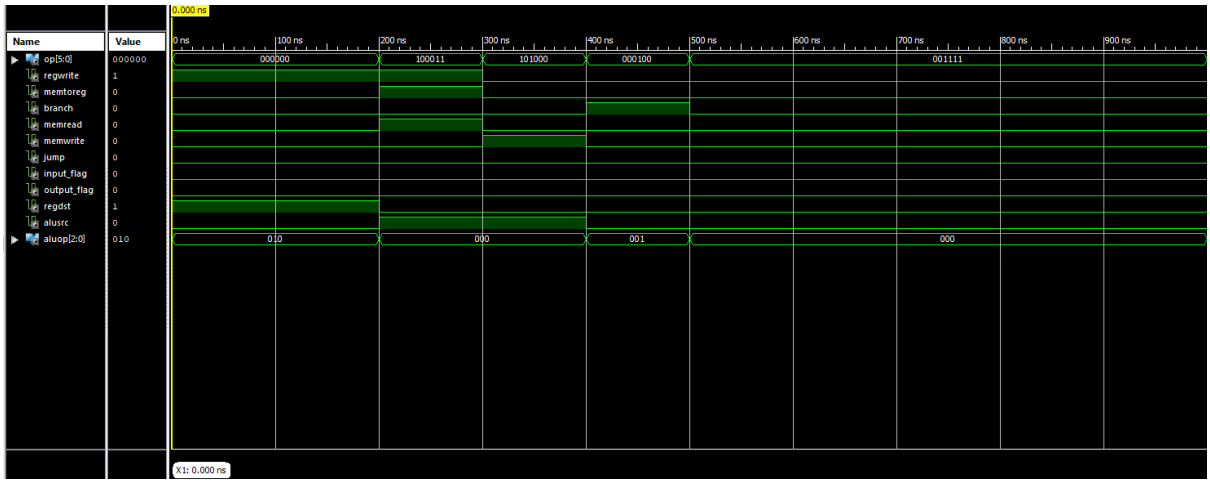


- Sign Extender:

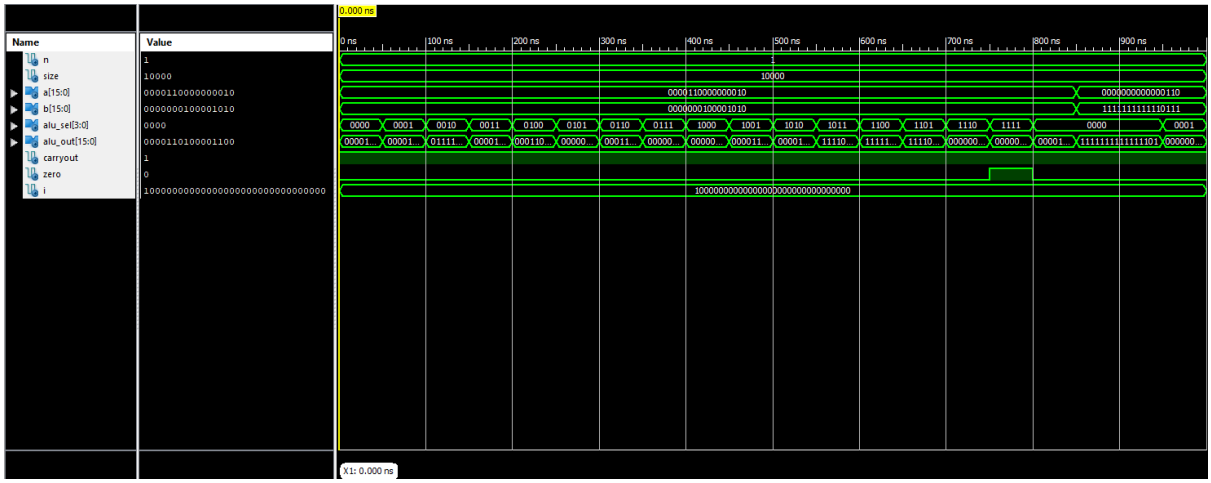


- Adder:

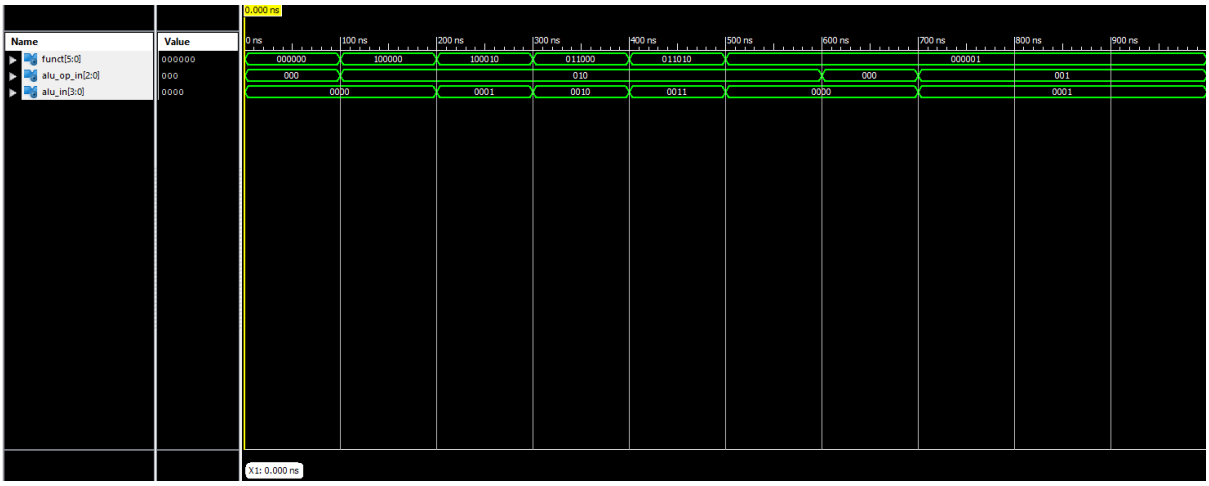
- Control Unit:



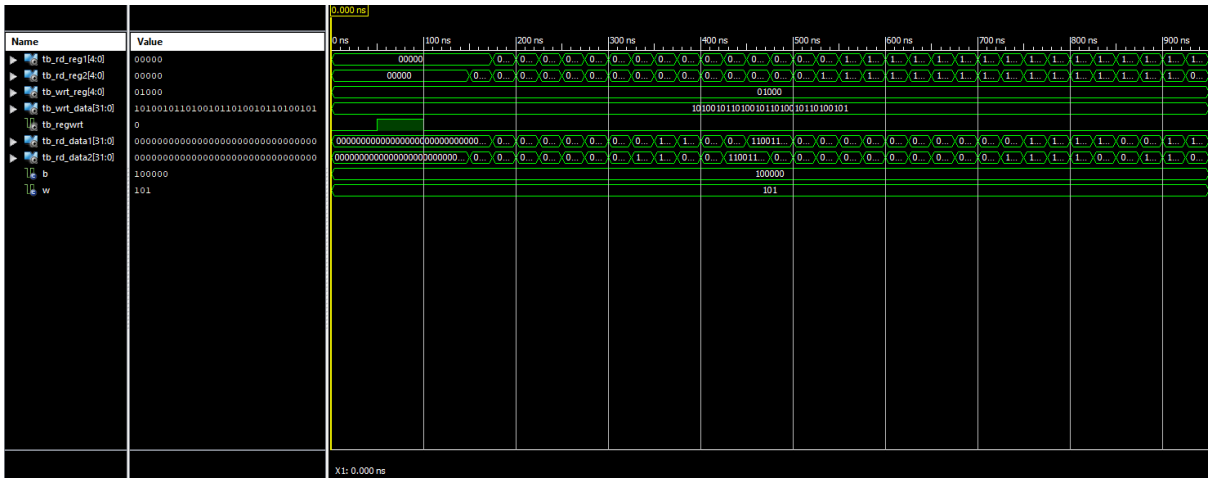
- ALU:



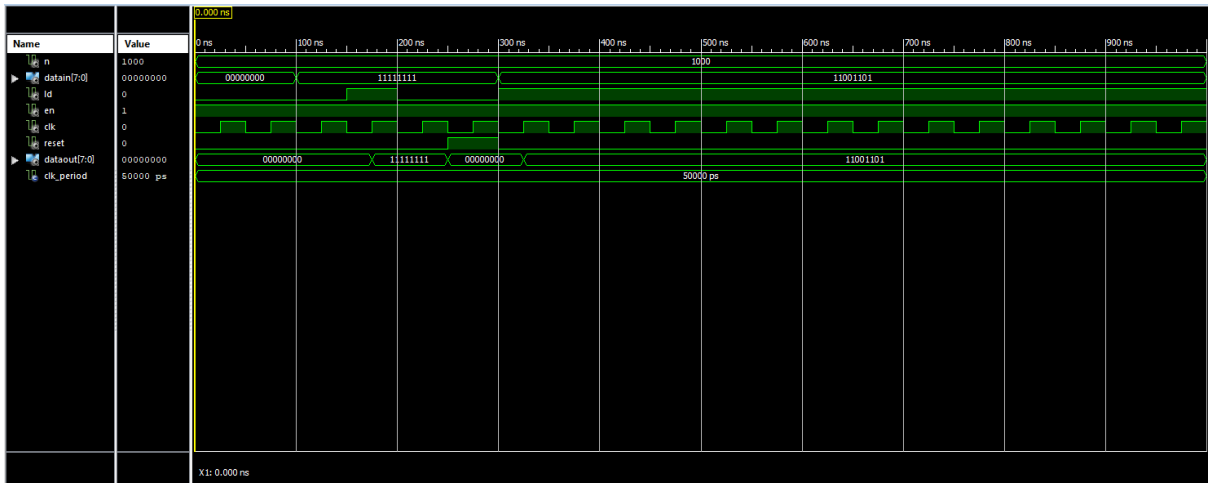
- ALU control:



- Register file:

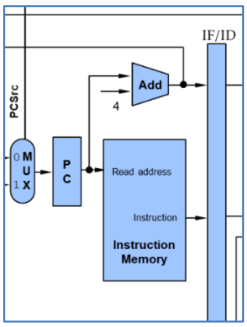


- Register:

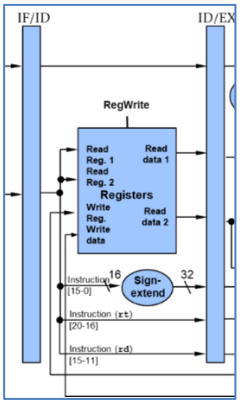


2. Pipeline Cycle Analysis

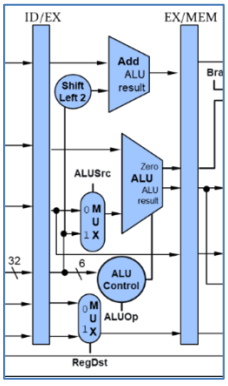
- The pipeline stages (IF, ID, EX, MEM, WB) were analyzed across multiple cycles to ensure correct instruction flow and data handling.
    - **Cycle 1 (Instruction Fetch - IF):**
        - The MOV instruction is fetched from the instruction memory.
        - The program counter (PC) is incremented to point to the next instruction.



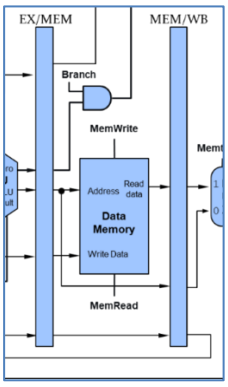
    - **Cycle 2 (Instruction Decode - ID):**
        - The fetched instruction is decoded to determine the opcode and the source and destination operands.
        - The necessary control signals are generated.



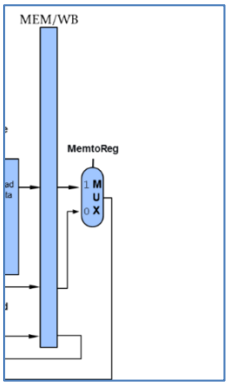
    - **Cycle 3 (Execute - EX):**
        - The ALU performs the required operation (e.g., addition, subtraction) based on the decoded instruction.
        - If the instruction involves an immediate value, the sign extender adjusts its width.



- **Cycle 4 (Memory Access - MEM):**
  - If the instruction involves memory access (e.g., load/store), the appropriate address is accessed in the data memory.



- **Cycle 5 (Write Back - WB):**
  - The result of the operation is written back to the destination register.
  - The pipeline registers ensure that the correct data is transferred between stages.



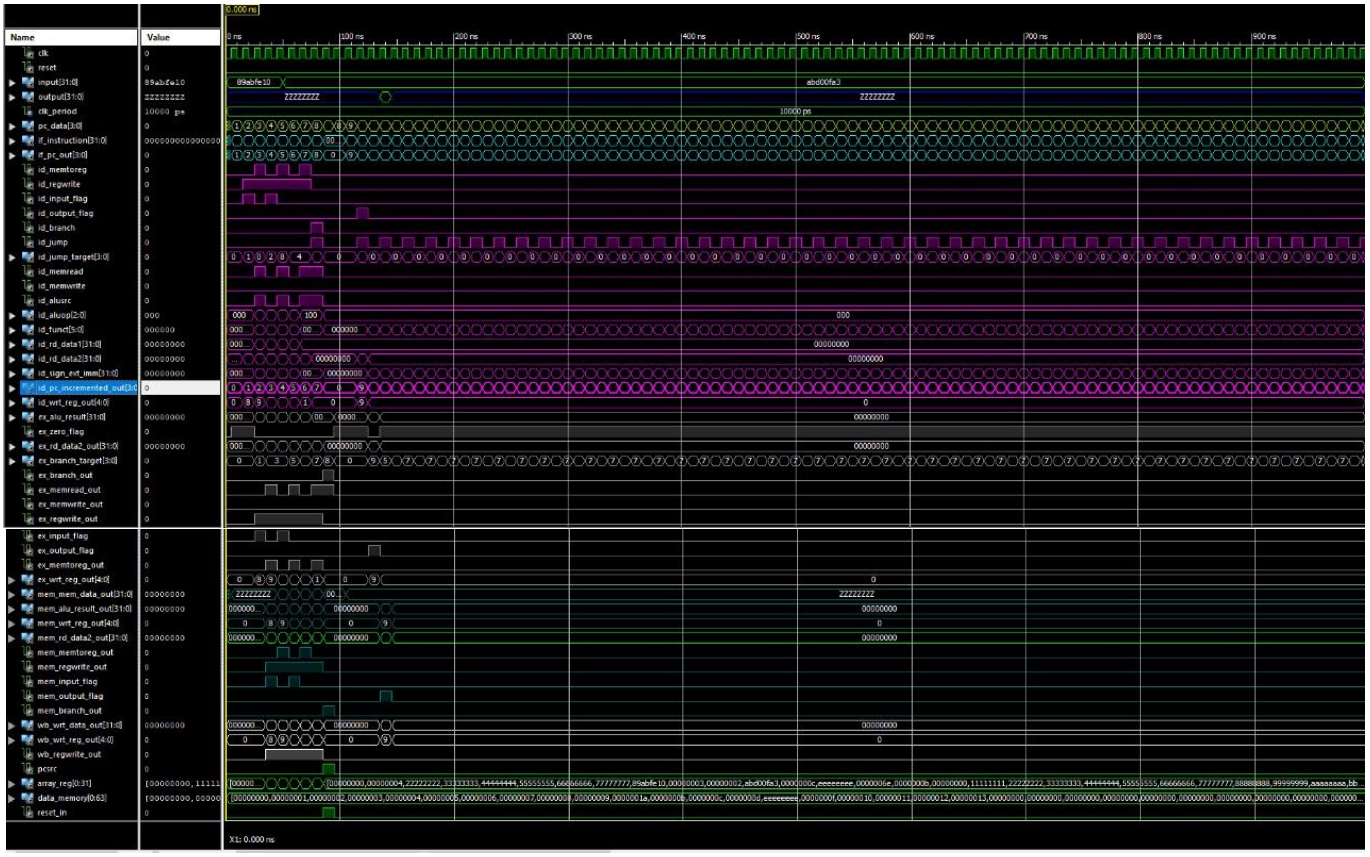
- Simulations confirmed that instructions were fetched, decoded, executed, accessed memory, and wrote back results in an interleaved manner, demonstrating the effectiveness of the pipeline architecture.

3. Hazard Detection

- Hazard detection mechanisms such as branch prediction were tested using specific test cases designed to trigger hazards.
    - **Branch Prediction:** Managing control hazards by predicting the outcome of branch instructions to minimize stalling.
- Simulation results showed that the following hazards were correctly detected and resolved, ensuring smooth pipeline operation.
    - **Control Hazards:** Managed by branch prediction techniques. The control unit predicts the outcome of branch instructions and fetches subsequent instructions accordingly. If the prediction is wrong, instructions in the pipeline are flushed and correct instructions are fetched.
    - **Structural Hazards:** Avoided by ensuring that no two pipeline stages compete for the same hardware resources. For example, separate memory units for instructions and data ensure that instruction fetch, and data memory access can occur simultaneously.

4. MIPS Processor Testing

- The complete processor was tested using a comprehensive test bench. A simple program loaded into instruction memory verified the execution of all supported instructions.
- Simulations confirmed correct instruction execution, proper handling of pipeline stages, and correct output results.



Conclusion

The MIPS processor was successfully designed and implemented using VHDL with a pipeline architecture. The processor efficiently performed basic arithmetic and logical operations while managing pipeline hazards through effective detection and resolution mechanisms. Comprehensive testing and simulation validated the processor's functionality and performance, demonstrating the successful implementation of a simple yet efficient MIPS processor.

Project Team Members

Ziad Ahmed Esmat - 211004746
Felopater Ashraf - 211004976
Fam Moheb - 211006142
Mohamed Hisham Amin - 211004487
Kareem Mohamed Fathy - 211004229
Tariq Ali Hassan - 211004633
Karim Hesham Fathy - 211004050