ABSTRACT

The project details the design and verification of a 32-bit single-cycle MIPS processor implemented using Verilog HDL, supporting a subset of the MIPS instruction set for educational and foundational exploration in processor design. The single-cycle architecture ensures simplicity by executing each instruction within a single clock cycle, covering essential operations like addition, subtraction, bitwise AND, OR, XOR, memory access (load and store), and control flow (branch and jump) to demonstrate core MIPS principles. The design integrates key components, including instruction memory, a register file, an ALU, data memory, and a control unit, interconnected for seamless instruction execution. Verilog enables efficient simulation and synthesis of the design, with verification carried out through comprehensive testbenches to ensure correctness and alignment with MIPS specifications. Simulation results validate accurate instruction execution, confirming the processor's functional reliability. While single-cycle architectures prioritize simplicity over efficiency, this work serves as a foundational platform for enhancements like pipelining or multi-cycle optimizations and demonstrates Verilog's effectiveness in digital system design and processor architecture development.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

[31:0] - Instruction bits for a 32-bit instruction.

I[25–21] - Source register 1 field in instructions.

I[20–16] - Source or destination register field.

I[15–11] - Destination register field for R-Type instructions.

I[15–0] - Immediate value or offset field for I-Type instructions.

clk - Clock signal.

pc - Program Counter signal.

current_pc - Current Program Counter value.

next_pc - Next Program Counter value.

LIST OF ABBREVIATIONS

ADC: Analog-to-Digital Converter

ALU: Arithmetic Logic Unit

CLK: Clock signal

DAC: Digital-to-Analog Converter

DFF: D Flip-Flop, used in sequential circuits

DRAM: Dynamic Random-Access Memory

EP: EDA Playground

FPGA: Field-Programmable Gate Array

GB: Gigabytes

GPIO: General-Purpose Input/Output

HDL: Hardware Description Language

IC: Integrated Circuit

ID: Instruction Decode

IF: Instruction Fetch

EX: Execution

ISA: Instruction Set Architecture

MEM: Memory unit or module

MUX: Multiplexer

PC: Program Counter

RISC: Reduced Instruction Set Computing

RTL: Register Transfer Level

SPICE: Simulation Program with Integrated Circuit Emphasis

SRAM: Static Random-Access Memory

VLSI: Very Large Scale Integration

WB: Write Back

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO MIPS PROCESSOR

Designing and verifying a 32-bit single-cycle MIPS processor are significant projects undertaken during an internship at Tessolve Semiconductors, a leading semiconductor services company. The project provides hands-on experience in digital system design and verification while aligning with industry standards and practices. The processor is based on the MIPS (Microprocessor without Interlocked Pipeline Stages) architecture, which embodies the principles of Reduced Instruction Set Computing (RISC). Known for its simplicity and efficiency, MIPS is widely used in academic and industrial applications, making it an excellent choice for this project.

Implemented using Verilog, the processor demonstrates the core functionality of MIPS architecture while providing a platform for understanding the intricacies of hardware design. The single-cycle architecture, chosen for its simplicity, ensures that each instruction is executed within a single clock cycle, making the design ideal for learning and exploration.

It operates on 32-bit instructions and data, supporting tasks that require larger data word sizes. By executing each instruction in a single clock cycle, the processor achieves straightforward control logic and timing, making it an ideal model for foundational learning. Implemented using Verilog, the design ensures accurate simulation, debugging, and synthesis, adhering to industry standards. Its modular design integrates key components like the ALU, control unit, register file, and memory, enabling scalability and potential enhancements such as pipelining or multi-cycle execution.

The processor finds applications in both educational and industrial domains. It serves as a hands-on tool for students and professionals to understand processor architectures and RISC principles while also providing a foundation for hardware

design and verification training. In the semiconductor industry, the design acts as a prototype for lightweight embedded systems and application-specific processors. Additionally, it supports research and development by enabling experimentation with advanced features. The processor's Verilog implementation further facilitates deployment on FPGA platforms for real-world applications, such as signal processing and low-power computing tasks.

1.2. TYPES OF MIPS PROCESSOR

MIPS processors are renowned for their efficient architecture and Reduced Instruction Set Computing (RISC) design, making them a preferred choice in VLSI design and embedded systems. These processors are categorized based on their execution methodologies and performance optimizations. MIPS processors come in several types, each optimized for different performance and design needs:

1. **Single-Cycle MIPS Processor:** This executes every instruction in a single clock cycle. While it simplifies the design and reduces control complexity, it limits performance due to the fixed clock cycle duration determined by the slowest instruction.
2. **Multi-Cycle MIPS Processor:** Instructions are broken into stages, with each stage executed in separate cycles. This design reduces hardware requirements by reusing components like the ALU and memory over multiple cycles.
3. **Pipelined MIPS Processor:** This employs a pipeline architecture to execute multiple instructions simultaneously, dividing execution into stages (e.g., fetch, decode, execute). It significantly improves throughput and overall performance.
4. **Superscalar MIPS Processor:** Enhancing parallelism, this type executes multiple instructions in the same clock cycle by employing multiple execution units.

5. **MIPS16:** A compact version of the MIPS architecture using 16-bit compressed instructions, aimed at improving code density and reducing memory usage. Ideal for embedded systems, IoT devices, and other applications requiring low power and cost efficiency.
6. **MIPS32:** It processes 32-bit data and supports smaller memory spaces, making it ideal for resource-constrained environments. MIPS32 is widely used in devices like routers, set-top boxes, and consumer electronics where energy efficiency and cost-effectiveness are essential.
7. **MIPS64:** Designed to handle 64-bit data and address larger memory spaces. It is optimized for high-performance computing and applications requiring extensive data handling, such as servers, advanced networking equipment, and data-intensive tasks.

CHAPTER 2

LITERATURE SURVEY

[1] The paper titled **"Design & Simulation of a 32-bit RISC based MIPS processor using Verilog" in *International Journal of Research in Engineering and Technology*, 10.15623/ijret.2016.0511030. by Bhardwaj, Priyavrat & Murugesan, Siddharth** outlines the design and simulation of a 32-bit RISC-based MIPS processor using Verilog HDL, showcasing a five-stage pipelined architecture for efficient instruction execution within a single clock cycle. Key strengths include modular design, effectively handling diverse instruction types, and robust simulation using tools like ModelSim and Xilinx. However, the design lacks focus on power optimization, scalability, and error-handling mechanisms, which are critical for modern applications. Future work could address these gaps by incorporating low-power design techniques, adaptive pipelining, and enhanced error correction. Additionally, extending support for advanced functionalities, such as floating-point operations and multicycle pipelines, could broaden its practical applications. Overall, the processor demonstrates significant potential for embedded systems.

[2] The paper titled **"Design and simulation of 32-Bit RISC architecture based on MIPS using VHDL" in *International Conference on Advanced Computing and Communication Systems*, pp. 1-6, doi: 10.1109/ICACCS.2015.7324067. by S. P. Ritpurkar, M. N. Thakare and G. D. Korde** presents a RISC processor based on the MIPS architecture, designed using VHDL for FPGA implementation. It covers the instruction set, processor architecture, and timing diagram, focusing on floating-point to fixed-point conversion. The processor uses a 32-bit architecture with various instruction formats like R-Type, I-Type, and J-Type. Key components include a memory unit, pipeline stages, a decoder unit, and an execution unit. Hazards such as data, structural, and control hazards are addressed using methods like forwarding and flushing. The

design, synthesis, and simulation were done using Xilinx ISE 13.1i. Its modular design allows for easier modifications and extensions while pipelining and parallel processing enhance overall performance. However, Limited by FPGA capacity for large-scale processor implementations and Floating-point to fixed-point conversion may introduce computational inaccuracies.

[3] The paper titled **"Design & Implementation Of 32-Bit Risc (MIPS) Processor" in** ***International Journal of Engineering Trends and Technology (IJETT)*****, V4(10):4466-4474 Oct 2013. ISSN:2231-5381. www.ijettjournal.org. published in the seventh sense research group. by SMarri Mounika , Aleti Shankar** describes the design and VHDL implementation of a MIPS single-cycle processor for FPGA. The processor was optimized for an 8-bit data width due to FPGA limitations, enabling the use of fewer memory blocks. It covers the design of combinational and sequential elements, including the ALU, register file, and memory units. The results show successful simulation and synthesis, with a 100ps instruction execution time. An advantage is efficient FPGA resource usage, while a drawback is the limitation of data width, reducing the processor's processing capacity.

[4] The paper titled **"Design and Implementation of RISC MIPS Processor on FPGA" in** ***International Journal For Science Technology And Engineering*****, 11(4), 1406-1410. by Vidyashree, H.** presents a 16-bit pipelined RISC MIPS processor, designed using Verilog and implemented on an FPGA, significantly improves performance by breaking down instructions into smaller stages, allowing concurrent execution. It utilizes a five-stage pipeline architecture—Instruction Fetch, Instruction Decode, Execute, Memory Access, and Write Back—resulting in faster processing and reduced power consumption. The key advantage of this design is its enhanced throughput, as multiple instructions can be processed simultaneously, reducing overall delay and improving efficiency. However, a notable drawback is

the complexity of pipeline management, which can lead to issues like data hazards or control hazards that require additional handling, potentially complicating the design. The future scope of this 16-bit pipelined RISC MIPS processor includes optimizing it for higher bit-width architectures and integrating advanced features like out-of-order execution and dynamic power management for enhanced performance.

[5] The paper titled **"Qualitative Analysis of 32 Bit MIPS Pipelined Processor" in *International Journal of Engineering Research and Technology* , 9(05) doi: 10.17577/IJERTV9IS050484 by Shobhit, Shrivastav., Shubham, Kumar., Sarthak, Gupta., Bharat, Bhushan** presents a 32-bit pipelined MIPS processor designed to reduce power consumption while maintaining high performance, comparing it with a 32-bit non-pipelined processor. The proposed pipelined processor, which uses a 5-stage pipeline architecture, shows a significant power reduction, consuming three times less power than the non-pipelined version. The main advantage of the design is its enhanced throughput and efficiency due to pipelining, which allows for parallel execution of instructions. However, a drawback is the occurrence of hazards like structural, data, and control hazards, which can affect performance. Future improvements could focus on optimizing hazard resolution techniques and further reducing power consumption through advanced power gating methods.

CHAPTER 3

EXISTING SYSTEM

The existing systems for implementing a 32-bit MIPS processor primarily focus on utilizing a Reduced Instruction Set Computing (RISC) architecture, known for its simplicity and efficiency. These processors are designed to maximize speed and minimize power consumption, making them ideal for embedded systems and applications requiring low-power operations. The general architecture of a 32-bit MIPS processor includes components like the program counter (PC), ALU (Arithmetic Logic Unit), instruction memory, data memory, register file, and control unit, each responsible for specific tasks in the execution of instructions.

Most implementations of the 32-bit MIPS processor are based on a single-cycle or multi-cycle design. In the single-cycle design, each instruction is completed in one clock cycle, simplifying the architecture but limiting performance due to the requirement of having long clock cycles. On the other hand, multi-cycle implementations break down instruction execution into multiple stages, with each stage taking a separate clock cycle, resulting in higher throughput. However, this can lead to complexities such as the introduction of hazards.

Recent trends in MIPS processor design are focused on pipelining, where instruction execution is divided into multiple stages—typically five: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). Pipelining significantly improves processor efficiency by enabling concurrent instruction processing, enhancing performance and throughput. However, pipelining introduces challenges such as structural, data, and control hazards, which must be addressed through techniques like forwarding and hazard detection.

Power consumption is another key consideration, with researchers focusing on optimizing designs to reduce energy usage without sacrificing performance. Techniques like power gating and clock gating are often applied to achieve this balance.

The 32-bit MIPS processor architecture has evolved significantly from single-cycle designs to multi-cycle and pipelined implementations. Each approach addresses trade-offs between simplicity, performance, and energy efficiency. With ongoing research focusing on advanced techniques for hazard resolution and power optimization, MIPS processors continue to be a versatile and reliable choice for modern computing applications.

CHAPTER 4

PROPOSED METHOD

4.1 SYSTEM SPECIFICATION

4.1.1 HARDWARE REQUIREMENTS

Table 4.1 : Requirement Specification

COMPONENT	QUANTITY	SPECIFICATION
Processor	1	Intel Core i5 or higher
RAM	1	8 GB minimum (16 GB recommended for smoother performance)
Storage	1	256 GB SSD (or higher) for faster data access

4.1.2 SOFTWARE REQUIREMENTS

- **Operating System**: Windows 10/11, macOS, or Linux (Ubuntu preferred for VLSI projects)
- **Web Browser**: Latest version of Google Chrome, Mozilla Firefox, or Microsoft Edge for accessing EDA Playground
- **EDA Playground**: No installation needed; accessible via a web browser
- **Verilog/SystemVerilog Simulator**: Available on EDA Playground (e.g., Synopsys VCS or ModelSim)
- **Code Editor**: Optional for offline editing (e.g., Visual Studio Code, Sublime Text, or Notepad++)

4.2 PROPOSED SYSTEM

The proposed system involves the design and implementation of a 32-bit single-cycle MIPS processor, tailored to execute all instructions within a single clock cycle. This processor will leverage the benefits of the MIPS architecture's simplicity and efficiency, ensuring a compact and high-performance design suitable for embedded and IoT applications. The Key Features include

1. **Single-Cycle Architecture**

In a single-cycle architecture, all instructions (arithmetic, memory, control) are executed in one clock cycle. The major characteristic of this approach is the simplicity and speed with which the processor executes its instructions. Here's an explanation of the process and its impact:

- **Instruction Fetch:** The processor fetches the instruction from memory using the Program Counter (PC), which stores the address of the next instruction to be executed.
- **Instruction Decode:** The fetched instruction is decoded by the Control Unit to generate the necessary control signals for the ALU and other components.
- **Execution:** The ALU performs the specified arithmetic or logic operation, while the necessary operands are retrieved from the Register File.
- **Memory Access and Write Back:** For memory operations (e.g., load or store), data is accessed in the Data Memory. Finally, the result of the operation is written back to the register file.

In a single-cycle design, every operation takes one cycle. The duration of a clock cycle is determined by the slowest instruction (e.g., memory access or a branch instruction), which may limit overall performance. This approach removes pipeline complexities and synchronization issues but sacrifices performance for simplicity.

2. **Reduced Instruction Set Computing (RISC)**

The MIPS architecture is based on RISC principles, where only a small, highly optimized set of instructions is used. RISC focuses on simplifying the processor design, reducing the complexity of control logic, and improving execution speed. By keeping the instruction set small, MIPS achieves:

- Faster Instruction Execution: The instructions are simple and consistent in terms of their execution time, resulting in reduced processing time per instruction.
- Reduced Control Logic: With fewer instructions, the design of the control unit becomes simpler and requires less hardware, saving power and increasing efficiency.
- Better Power Efficiency: Simplified design means less power consumption, which is critical for low-power embedded systems and IoT devices.

3. **Data Path Components**
    - Arithmetic Logic Unit (ALU): Performs arithmetic and logic operations.
    - Register File: Stores operands and results during execution.
    - Instruction Memory: Holds the program's instructions.
    - Data Memory: Used for temporary storage during program execution.
4. **Control Unit**
    - Generates precise control signals to orchestrate the operations of the data path, ensuring that instructions execute seamlessly.
5. **Multiplexer (MUX)**

A Multiplexer (MUX) is a crucial component in selecting inputs based on control signals. It allows the processor to decide between two or more sources for a particular data path. In the case of the MIPS processor:

- MUX in the ALU: A multiplexer is used to select between register values or immediate values based on the instruction type (e.g., add, subtract, or load).

- MUX for Branch/Jump Operations: It allows the PC to either continue sequentially or jump to a different address depending on the control signal.
- Data Path MUX: The MUX helps select between different inputs to the ALU (e.g., between two registers or an immediate value).

6. Sign Extension

Sign extension is used to extend the size of an immediate value in the instruction set from 16 bits to 32 bits. This is necessary because some instructions may involve immediate operands, and the ALU requires the full 32-bit value for computation. For example:

- In a 16-bit immediate instruction, the immediate value needs to be extended to 32 bits for operations like arithmetic and memory access.
- The sign extension process copies the most significant bit (sign bit) of the immediate field into the higher-order bits to preserve the number's sign during the extension.

EDA Playground Implementation and EP Waveform Formation

EDA Playground Implementation

EDA Playground is an online tool used for simulating hardware designs with Verilog/SystemVerilog. It allows developers to quickly prototype and test designs like the MIPS processor without needing a local simulation environment.

Design Steps

- Write the Verilog/SystemVerilog code for the data path, control unit, and ALU.
- Set up the test benches to simulate various instructions (e.g., load, store, add, subtract).
- Use the simulation tool (e.g., ModelSim or Synopsys VCS) on EDA Playground to run the design and test its functionality.

Waveform Formation

The waveform generated by EDA Playground helps visualize the behavior of the processor during simulation. It provides a graphical representation of the signals in the data path, control unit, and ALU. Users need to select EP waveform to visualize the graph.

- **Key signals** to watch:
    - The Program Counter (PC) during instruction fetch.
    - The Control signals during the decode stage.
    - The ALU output during execution.
    - The Data Memory signals during load/store instructions.
- Waveforms allow us to verify that the processor correctly fetches, decodes, executes, and writes back instructions, all within a single cycle.

4.3 SYSTEM ARCHITECTURE

The data path integrates essential components such as instruction memory, registers, the ALU (Arithmetic Logic Unit), and data memory, coordinated by multiplexers and control signals to ensure smooth data flow and operation as shown in Fig 4.1. Immediate values are handled through a sign-extension unit for proper processing in arithmetic operations.

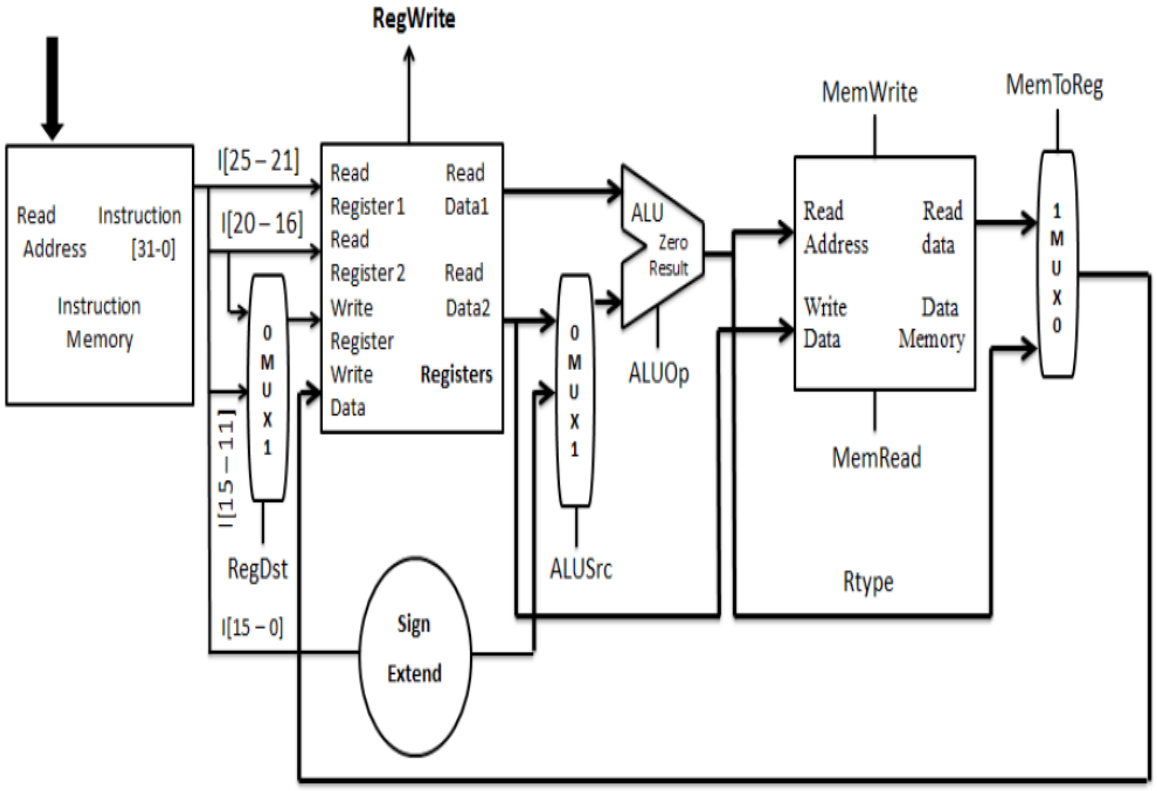

Figure 4.1: System Architecture of 32-bit MIPS Processor

By visualizing the interaction between components and control signals, this block diagram highlights the modular and systematic nature of the MIPS architecture, enabling efficient execution of instructions within a single clock cycle.

1. **Arithmetic Logic Unit (ALU):**

The ALU performs all arithmetic and logic operations (addition, subtraction, AND, OR, etc.).It is an essential component for executing the instructions that require computation.

- **Function**: Performs arithmetic and logical operations based on the **ALUOp** control signal.
- **Inputs**: First input is from the register file, and the second input is chosen by **ALUSrc**.
- **Outputs**: Result of the operation and a **Zero** signal (used for branching).

2. **Register File:**

The register file holds operands and results during execution. It is essentially a set of registers where the CPU stores temporary data. It is typically composed of two read ports and one write port for reading operands and writing results.

- **Function**: Stores data temporarily
- **Operation**:
    - Reads data from source registers based on instruction fields **rs** and **rt**.
    - Writes data to the destination register based on the **Write Register** signal controlled by **RegWrite**.

3. **Instruction Fields**

- **I[31–0]**: The fetched instruction is divided into specific fields:
    - **I[25–21]**: Source register 1 (rs).
    - **I[20–16]**: Source register 2 (rt) or destination register for immediate-type instructions.
    - **I[15–11]**: Destination register (rd) for R-type instructions.
    - **I[15–0]**: Immediate value or offset for I-type instructions.

4. Instruction Memory

This component stores the program's instructions. It is a read-only memory, where instructions are fetched sequentially unless a jump or branch instruction alters the flow.

- **Function**: Stores program instructions.
- **Operation**: The instruction is fetched using the program counter (PC) as the read address. Each instruction is 32 bits wide in this 32-bit MIPS processor.

5. Data Memory:

Used for temporary storage, especially when performing memory operations like load or store. The data memory operates during load and store instructions to move data between registers and memory.

- **Function**: Reads or writes data for memory-access instructions (e.g., lw, sw).
- **Operation**:
    - Controlled by **MemWrite** (write enable) and **MemRead** (read enable) signals.
    - The address comes from the ALU result.

6. MUX (Multiplexers)

- **MUX1 (RegDst)**: Selects between **rt** and **rd** as the destination register for write operations.
- **MUX2 (ALUSrc)**: Chooses between the second register value (from rt) or the sign-extended immediate value as the second input to the ALU.
- **MUX3 (MemToReg)**: Determines whether the data to be written back to the register comes from the ALU result or memory.

7. Sign Extend

- **Function**: Converts a 16-bit immediate value to a 32-bit value.
- **Purpose**: Ensures proper arithmetic operations with both positive and negative values.

8. Control Signals

Generated by the **Control Unit** to manage the data path elements and execute instructions properly.

- **RegWrite**: Enables register write.
- **MemRead** and **MemWrite**: Enable memory operations.
- **ALUSrc**: Select the second ALU input.
- **MemToReg**: Select the write-back data source.
- **RegDst**: Chooses the destination register.

9. Zero Signal

- Generated by the ALU.
- Indicates if the result of an operation (e.g., subtraction) is zero. Often used for branch decisions.

4.4 WORKFLOW

To simplify the design and implementation, the project is divided into manageable phases:

- **Phase 1: Instruction Fetch and Decode**
    - Design and implement the program counter (PC) to manage instruction addresses.
    - Develop instruction memory to store the instructions and a control unit to decode them.
- **Phase 2: Execution**
    - Build the ALU to execute operations specified by the instructions.
    - Create a register file to facilitate data storage and retrieval.
- **Phase 3: Memory Access and Write-Back**
    - Implement data memory access mechanisms for load and store instructions.
    - Finalize the write-back logic to update register values after execution.
- **Phase 4: Multiplexer and Sign Extend**
    - Multiplexers allows the processor to decide between two or more sources for a particular data path and also used to integrate overall data paths.
    - Sign extension is used to extend the size of an immediate value in the instruction set from 16 bits to 32 bits.
- **Phase 5: Debugging Process and Waveform Generation**
    - The EP waveforms generated in each phase are to be analyzed well and should go through many debugging processes.
    - After debugging, the resultant output is proved by the waveforms obtained by all data path components.

4.4.1 INSTRUCTION SET

The Instruction Set of the MIPS Processor in the simulation emulates essential MIPS operations like arithmetic (ADD, SUB), logical (AND, OR), comparisons (SLT), memory access (LW, SW), and branching (BEQ, BGTZ). These instructions serve as the foundation for executing programs on a MIPS processor by providing basic building blocks for computation and control flow.

The code used in the project models a basic MIPS instruction memory in Verilog. In code, program_counter module and tb_program_counter, the use of logic is a SystemVerilog feature. always_ff is also a SystemVerilog construct and replaces always for sequential logic. Basically, verilog and features of system verilog is used in the instruction set.

This implementation is designed to simulate the functionality of fetching instructions from memory for a simple MIPS processor. The design includes three key modules:

1. **InstructionMemory:** Simulates a memory array containing instructions.
2. **program_counter:** Tracks the current program counter and updates it to point to the next instruction.
3. **mips_top**: Integrates the InstructionMemory and program_counter modules to form the top-level structure of the processor.

Instruction Memory Module:

- The InstructionMemory module contains a memory array memory[0:19] capable of storing 20 instructions (32-bit each).
- The Instruction Format of each instruction and Instruction Field representation of each Type of Instruction is given in the Table 4.2 and Table 4.3.

Table 4.2 Instruction Format for Each Instruction

Name	Format	Example					Comments
		3 bits	3 bits	3 bits	3 bits	4 bits	
add	R	0	2	3	1	0	add $1,$2,$3
sub	R	0	2	3	1	1	sub $1,$2,$3
and	R	0	2	3	1	2	and $1,$2,$3
or	R	0	2	3	1	3	or $1,$2,$3
slt	R	0	2	3	1	4	slt $1,$2,$3
jr	R	0	7	0	0	8	jr $7
lw	I	4	2	1	7		lw $1, 7 ($2)
sw	I	5	2	1	7		sw $1, 7 ($2)
beq	I	6	1	2	7		beq $1,$2, 7
addi	I	7	2	1	7		addi $1,$2, 7
j	J	2	500				j 1000
jal	J	3	500				jal 1000
slti	I	1	2	1	7		slti $1,$2, 7

Table 4.3 Instruction Field Representation for each Instruction Type

Name	Fields					Comments
Field size	3 bits	3 bits	3 bits	3 bits	4 bits	All MIPS-L instructions 16 bits
R-format	op	rs	rt	rd	funct	Arithmetic instruction format
I-format	op	rs	rt	Address/immediate		Transfer, branch, immediate format
J-format	op	target address				Jump instruction format

- Each instruction is manually initialized during simulation using the initial block. These instructions include R-type, I-type, and J-type instructions, which are basic formats of MIPS instructions:
    - ***R-Type Instructions*:** Perform arithmetic or logical operations.
        - Format: opcode (6 bits) | rs (5 bits) | rt (5 bits) | rd (5 bits) | shamt (5 bits) | funct (6 bits)

        - Example: memory[0]=32'b000000_00001_00010_00011_00000_100000 corresponds to ADD $3, $1, $2 (Add the values in registers $1 and $2 and store the result in $3).
    - ***I-Type Instructions*:** Use immediate values for computations or memory access.
        - Format: opcode (6 bits) | rs (5 bits) | rt (5 bits) | immediate (16 bits)
        - Example: memory[5]=32'b001000_00001_00010_0000000000000101 corresponds to ADDI $2, $1, 5 (Add the immediate value 5 to the value in register $1 and store the result in $2).
    - ***J-Type Instructions*:** Perform unconditional jumps to specified addresses.
        - Format: opcode (6 bits) | address (26 bits)
        - Example: memory[11]=32'b000010_00000000000000000000001010 corresponds to J 10 (Jump to address 10).

Program Counter Module:

- Tracks the current instruction address using a 32-bit current_pc signal.
- On every clock cycle, the current_pc is updated with the next_pc value, which is computed as current_pc + 4 to fetch the next sequential instruction. The counter resets to 0 when the reset signal is high.

Top-Level Module (mips_top):

- Integrates the program_counter and InstructionMemory to simulate the fetch phase of instruction execution.
- On each clock cycle, the program counter fetches an instruction from memory based on the current address (pc), which is then output as instruction.

Testbenches:

Three test benches verify individual modules and the top-level integration:

1. **tb_instruction_memory:** Tests the fetching of instructions for various memory addresses.
2. **tb_program_counter:** Verifies program counter functionality, including reset behavior and sequential address updating.
3. **tb_mips_top:** Simulates the top-level integration, ensuring correct interaction between program_counter and InstructionMemory.

This implementation provides a foundational simulation of instruction memory in a MIPS processor. It models how instructions are fetched sequentially or through branching, which is critical for implementing a processor's control and data flow. This demonstration serves as a stepping stone for building more complex pipelines and understanding processor architectures. Through the provided testbenches, opcode correctness and memory access functionality are verified, ensuring reliable instruction fetch operations.

4.4.2 ARITHMETIC LOGIC UNIT (ALU)

The **Arithmetic Logic Unit (ALU)** is one of the core components of a 32-bit single-cycle MIPS processor. It performs arithmetic and logical operations as required by the MIPS instruction set.

Role of the ALU in the MIPS Processor

The ALU is responsible for executing operations on the data fetched from registers. It processes instructions related to arithmetic, logic, comparison, and address calculations for memory access.

For example:

- **Arithmetic Operations**: Add (add), Subtract (sub)
- **Logical Operations**: Bitwise AND (and), Bitwise OR (or), NOR (nor)
- **Comparison Operations**: Set on Less Than (slt)
- **Address Calculations**: Add the base address and offset for memory instructions (lw, sw)

Structure of the ALU

A typical ALU for a single-cycle MIPS processor has:

1. **Inputs**:
    - Two 32-bit operands (A and B).
    - A 4-bit control signal (ALUControl), which specifies the operation to be performed.
2. **Outputs**:
    - A 32-bit result (Result) representing the output of the operation.
    - A single-bit Zero flag, which is set to 1 if the Result is zero (used in branch instructions).

ALU Control Signal

The ALUControl signal determines which operation the ALU performs. The ALUControl is derived from the **ALUOp** signal and the **funct** field in the instruction.

- **ALUOp**: A 2-bit signal provided by the control unit, which guides the ALU.
- **funct**: A 6-bit field in R-type instructions that specifies the exact operation (e.g., add, sub, etc.).

Table 4.4 ALU Control Unit

ALU Control				
ALU op	**Function**	**ALUcnt**	**ALU Operation**	**Instruction**
11	xxxx	000	ADD	Addi,lw,sw
01	xxxx	001	SUB	BEQ
00	00	000	ADD	R-type: ADD
00	01	001	SUB	R-type: sub
00	02	010	AND	R-type: AND
00	03	011	OR	R-type: OR
00	04	100	slt	R-type: slt
10	xxxxxx	100	slt	i-type: slti

Process of the ALU

1. Fetch the Inputs: The inputs to the ALU (A and B) are fetched from the register file or generated as a sign-extended immediate value.
2. Determine the Operation: The Control Unit decodes the instruction and sets the ALUOp signal. For R-type instructions, the funct field in the instruction is used to specify the exact operation, passed to the ALUControl logic.

3. Perform the Operation:
    - Based on the ALUControl signal, the ALU performs one of the following:
        - Addition (add or lw/sw address calculation)
        - Subtraction (sub or beq comparison)
        - Logical AND, OR, or NOR
        - Comparison for slt
    - For example:
        - add: Result=A+B\text{Result} = A + BResult=A+B
        - sub: Result=A−B\text{Result} = A - BResult=A−B
        - slt: Result=1 if A<B, else 0\text{Result} = 1 \text{ if } A < B, \text{ else } 0Result=1 if A<B, else 0
4. Set the Zero Flag:
    - If the result of the operation is zero, the Zero output is set to 1. This is primarily used in branch instructions like beq to decide if the program counter (PC) should jump to a new location.
5. Pass the Result:
    - The ALU's 32-bit result is passed back to the register file (for R-type instructions) or to the data memory unit (for lw/sw instructions).

4.4.3 REGISTERS

Registers are small, fast storage locations within a processor that hold data temporarily during computation. These registers are critical for storing operands, intermediate results, and control data, allowing the processor to perform calculations without frequently accessing slower main memory. In most processors, registers are directly connected to the Arithmetic and Logic Unit (ALU), which performs arithmetic and logical operations on the data stored in them.

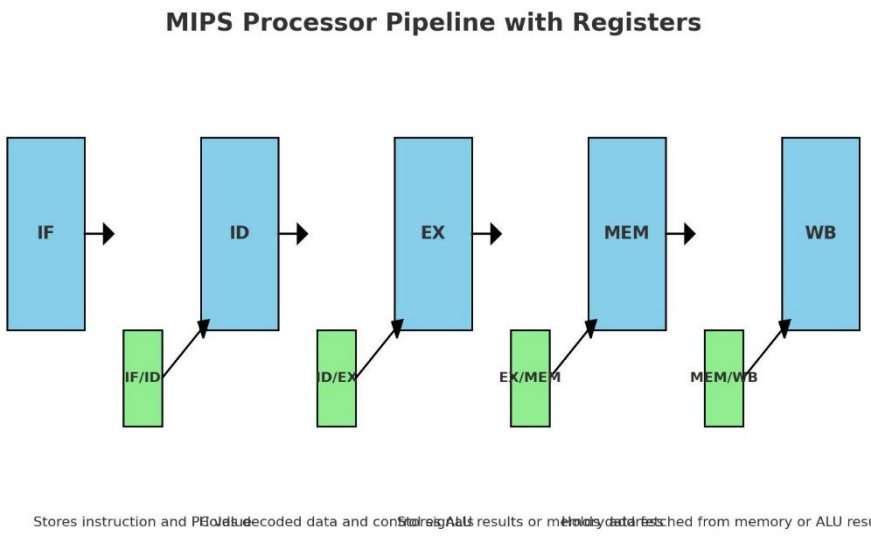

Figure 4.3: Specialized pipeline registers

In the context of the **MIPS processor**, registers are crucial for executing instructions efficiently, particularly with its **RISC (Reduced Instruction Set Computing)** architecture. MIPS uses a specific set of registers, including **general-purpose registers** and specialized **pipeline registers in Fig 4.2**. The **general-purpose registers** store operands and results during instruction execution, while the **pipeline registers** isolate and manage data as it progresses through the processor's five pipeline stages: Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MEM), and Write Back (WB).

The pipeline registers in MIPS include the **IF/ID register**, which holds the instruction and Program Counter (PC) from the Instruction Fetch stage; the **ID/EX register**, which stores the decoded instruction and operands for the Execution stage; the **EX/MEM register**, which stores ALU results or memory addresses for the Memory Access stage; and the **MEM/WB register**, which stores data to be written back to the register file after Memory Access. These pipeline registers allow for efficient concurrent processing of multiple instructions by isolating the stages, preventing conflicts, and ensuring that each stage has the necessary data. The use of these registers plays a key role in optimizing the throughput and overall performance of the MIPS processor.

4.4.4 MEMORY ACCESS

The memory unit is a fundamental component of the 32-bit single-cycle MIPS processor, responsible for storing instructions and data required during program execution. It provides a unified memory space that accommodates both instruction and data memory, utilizing a 32-bit address bus. This enables the processor to address up to 2^32 memory locations, corresponding to a total addressable memory capacity of 4 GB. The memory unit is designed to facilitate efficient data retrieval and storage while ensuring precise synchronization with the processor's operations.

1. Structure of the Memory Unit

- **Instruction Memory**: Stores the program's executable instructions. These instructions are fetched by the processor during each cycle.
- **Data Memory**: Holds variables and intermediate data. It is used for storing results of operations and temporary values required during program execution.
- **Unified Address Space**: The memory is accessed using a 32-bit address, allowing a total of 2^32locations, equivalent to 4 GB of addressable space.

2. Memory Access Operations

- **Load (lw)**: The memory unit retrieves data from the specified address and provides it to the processor for use in computations.
- **Store (sw)**: The memory unit saves data provided by the processor to the specified address.
- **Word Alignment**: All memory addresses are word-aligned (multiples of 4), ensuring efficient access to 32-bit words and avoiding partial-word issues.

3. Control Unit and Synchronization

- The control unit is responsible for managing how the processor interacts with the memory unit. It coordinates memory operations like loading and storing data.

- **Load Operation (lw)**: The control unit sends the address to the memory, asking for data. Once the memory returns the data, the control unit provides it to the processor.
- **Store Operation (sw)**: The control unit sends the data and the address to the memory, instructing it to store the data at that location.
- Since the memory operations must complete in one clock cycle in a single-cycle architecture, synchronization is essential.

4. Role of Cache Memory

In this architecture, where every operation must complete in one clock cycle, the inclusion of a cache helps improve performance without altering the simplicity of the design.

Key Roles of Cache Memory

- **Cache Integration**: A cache is a small, high-speed memory unit added to store frequently accessed data or instructions. In a single-cycle MIPS processor, it helps reduce the delay caused by accessing slower main memory.
- **Instruction Cache**: The instruction cache stores commonly used instructions from the program. This allows the processor to fetch instructions quickly during execution, reducing the time needed to access instruction memory for repetitive or frequently executed tasks.
- **Data Cache**: The data cache temporarily holds frequently accessed data values. For operations like lw (load word) and sw (store word), the processor can directly interact with the data cache instead of accessing the main memory, improving execution speed.

Performance Improvement: Caches significantly reduce memory latency by decreasing the number of times the processor needs to access the main memory. This enhancement allows the processor to maintain the single-cycle execution model while handling memory-intensive tasks more efficiently.

4.5 INTEGRATION OF COMPONENTS THROUGH MUX

In a 32-bit single-cycle MIPS processor, multiplexers (MUXes) play a vital role in integrating various components and controlling data flow. They enable the processor to dynamically select between multiple data sources based on control signals, ensuring efficient utilization of shared hardware resources. For example, a MUX is used to determine the second input to the Arithmetic Logic Unit (ALU). This input could either be a register value (for R-type instructions) or a sign-extended immediate value (for I-type instructions). The control signal ALUSrc, generated by the control unit, directs the MUX to select the appropriate input. Similarly, MUXes are employed to choose whether the result to be written back to the registers comes from the ALU or the data memory and to decide the program counter (PC) update path for branch or jump instructions.

The integration of MUXes ensures flexibility and modularity in the design. By routing the correct inputs to the ALU, register file, or memory based on the type of instruction, MUXes allow the processor to handle diverse instructions efficiently. This dynamic selection eliminates the need for duplicate pathways, simplifying the overall design and making it more scalable. Through effective MUX integration, the MIPS processor achieves a compact yet versatile architecture capable of executing various operations in a single clock cycle.

CHAPTER 5

RESULTS AND DISCUSSION

WORKING OF MIPS PROCESSOR

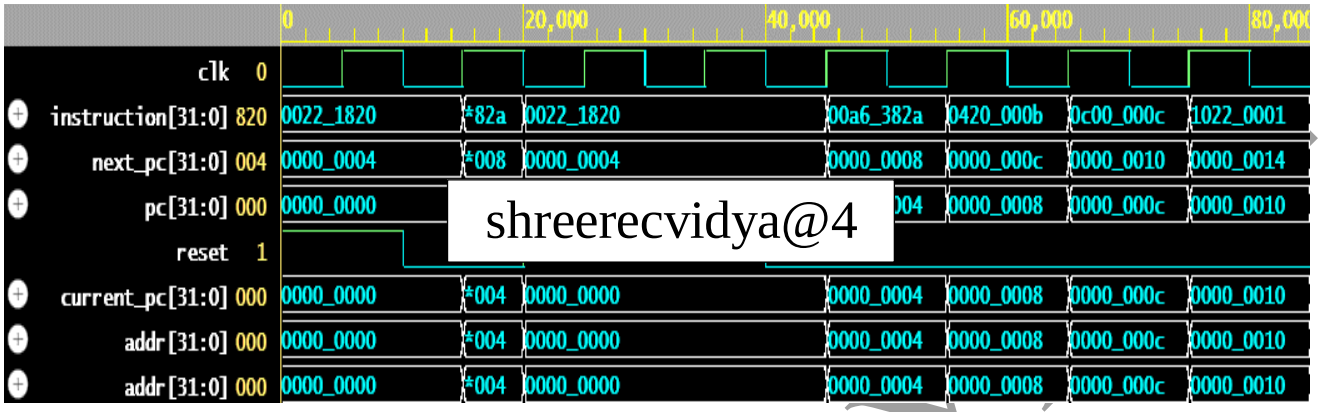

Figure 5.1 Simulated Output of Instruction Set

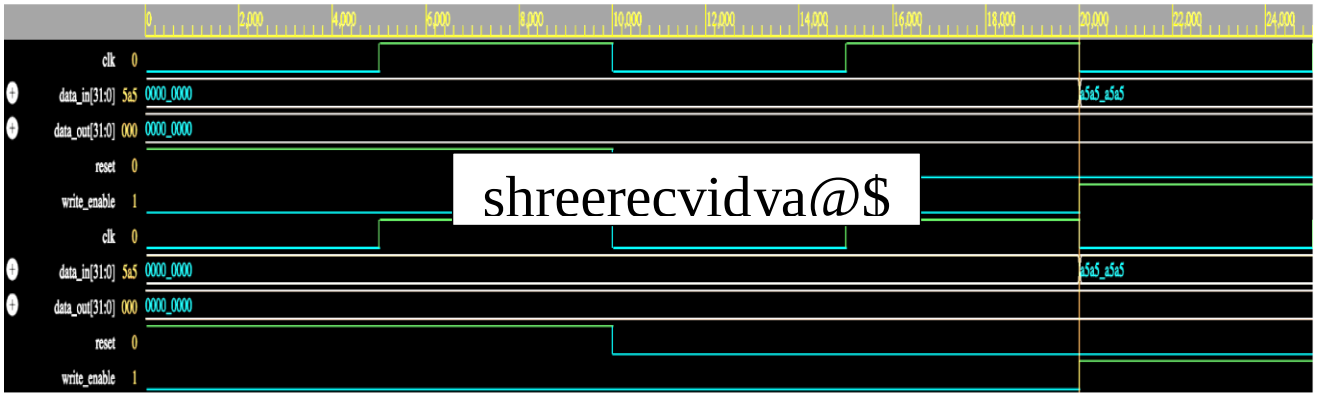

Figure 5.2 Simulated Output of Register

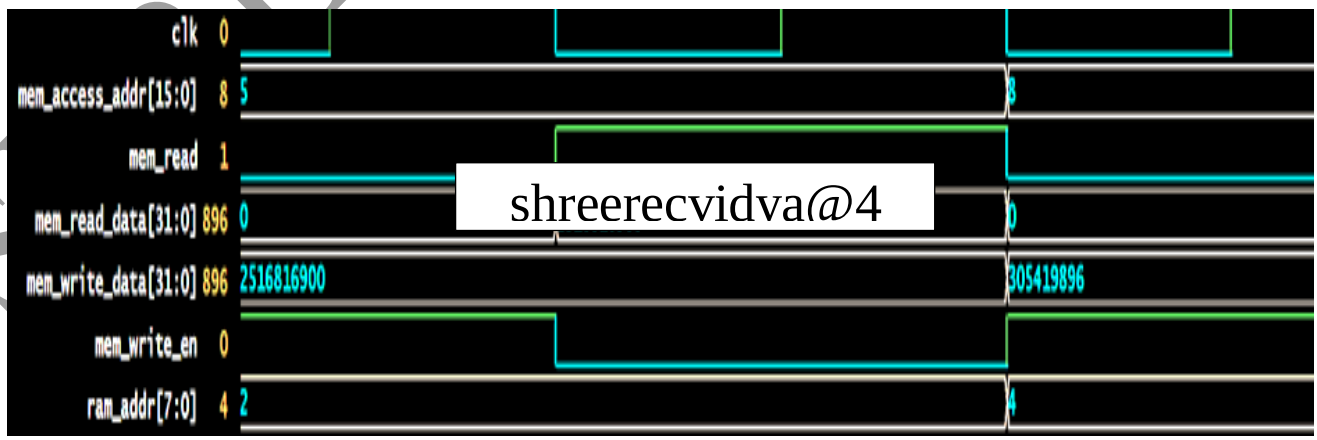

Figure 5.3 Simulated Output of Memory Access

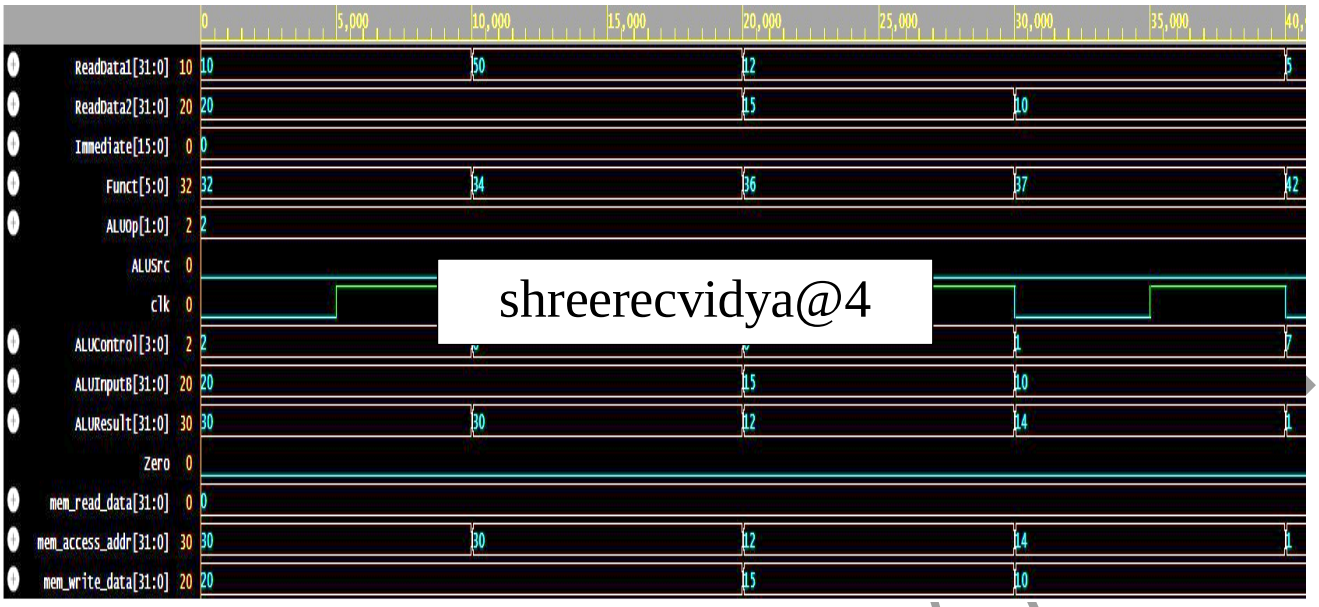

Figure 5.4 Simulated Output of ALU Unit

OVERALL INTEGRATED OUTPUT

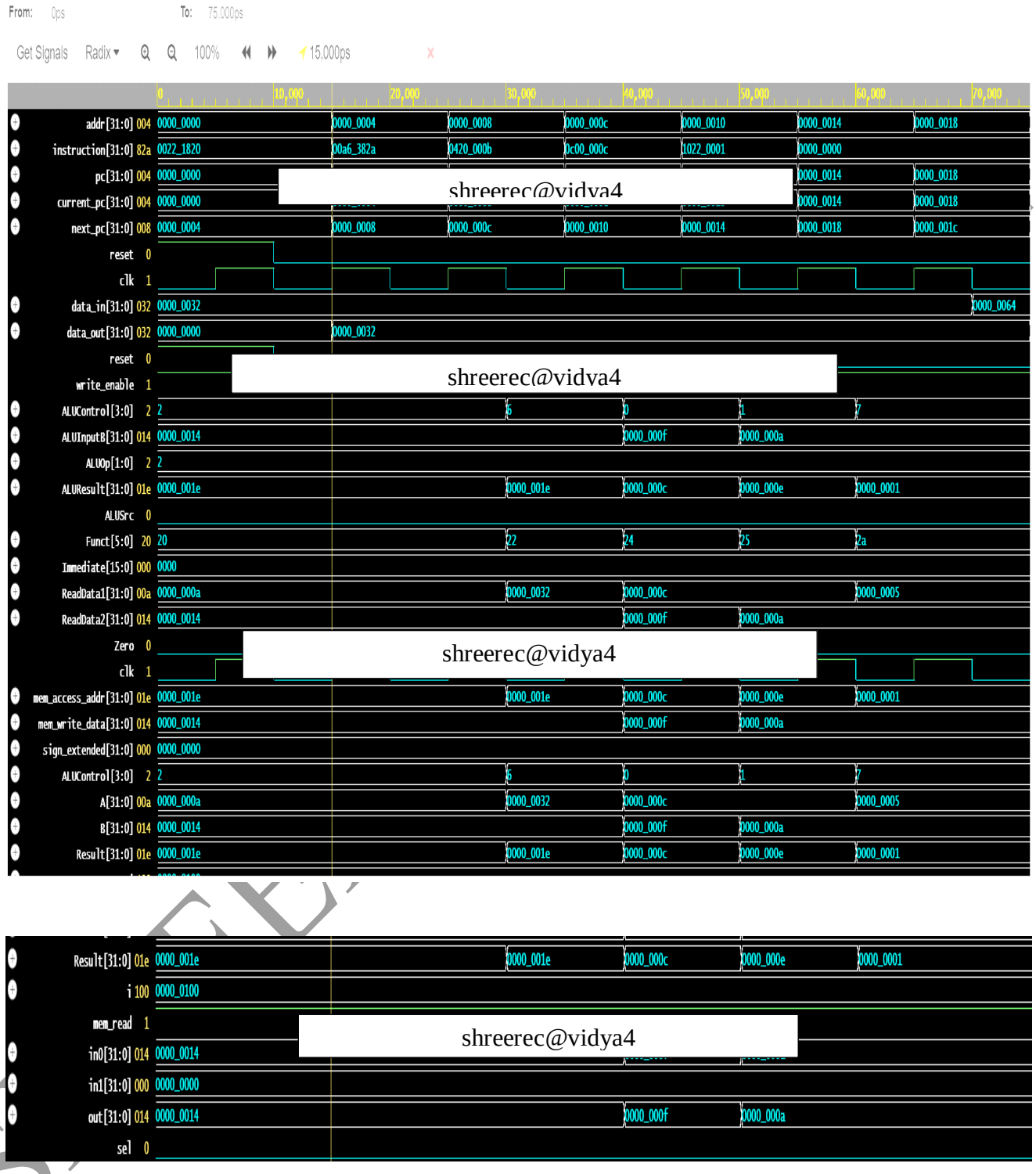

Figure 5.5 Simulated Output of MIPS Processor

OUTPUT OF 32-BIT MIPS PROCESSOR DISCUSSION

The waveform shown in Fig 5.1 simulation highlights the functionality of the MIPS instruction memory and program counter modules. The clock signal (clk) drives the sequential operations of the processor, toggling between 0 and 1. Each rising edge of the clock triggers the program counter (pc) to update, enabling continuous instruction fetching. The instruction[31:0] signal represents the fetched 32-bit instruction corresponding to the pc value, demonstrating accurate retrieval from the instruction memory. The next_pc[31:0] signal shows the next program counter value, incrementing by 4 to ensure word-aligned instruction fetching. The pc[31:0] signal represents the current program counter value, which is updated sequentially. The reset signal initializes the pc to 0, allowing the processor to restart fetching instructions when required. The addr[31:0] signal matches the pc value, verifying that the memory address for instruction fetching is accurate. Finally, the current_pc[31:0] output mirrors the pc, ensuring synchronization across the module. The simulation confirms the proper functionality of the MIPS instruction memory and program counter, enabling sequential instruction fetching and reset handling.

The waveform in Fig 5.2 shows the simulation of a register with signals clk, data_in, data_out, reset, and write_enable. Initially, from 0–20 ns, the reset signal is high, forcing data_out to 0 irrespective of the input, demonstrating the reset functionality. After reset goes low at 20 ns, the register begins normal operation. When write_enable is high, the register latches the value of data_in on the rising edge of clk, updating data_out accordingly. If write_enable is low, the register holds its previous value, maintaining stability in the output. This behavior verifies proper register functionality under varying input and control conditions.

The waveform in Fig 5.3 demonstrates memory read and write operations synchronized with the clock (clk). When mem_read is high, the memory fetches data from the address specified by mem_access_addr, and the value is reflected in mem_read_data (e.g., 2516816900 for address 8). During the write operation, when

mem_write_en is high, the value in mem_write_data (e.g., 305418896) is written to the specified address. The ram_addr signal shows the internal memory mapping, derived from mem_access_addr.

This waveform in Fig 5.4 represents the simulation of a MIPS processor's datapath, showcasing how signals flow through its components. The ReadData1 and ReadData2 signals represent data fetched from the register file, while the Immediate signal provides constant values for I-Type instructions. The ALUSrc control signal selects whether the ALU's second operand comes from ReadData2 (register) or Immediate. The Funct field specifies the operation for R-Type instructions, which is translated into specific control signals by ALUControl. The inputs to the ALU, ALUInput1 and ALUInput2, are processed to produce ALUResult, representing the computed output. The Zero flag indicates if the result is zero, often used for branching decisions. Memory-related signals (mem_read_addr, mem_access_addr, and mem_write_data) also show data flow to/from memory for load and store instructions. The timing of these signals aligns with the processor's clock (Clk), illustrating the single-cycle execution.

This waveform in Fig 5.5 illustrates the simulation of a MIPS processor, showcasing key components such as the program counter (PC), instruction memory, data memory, ALU, and control signals. The addr[31:0] and instruction[31:0] signals represent the PC address and fetched instruction, with current_pc incrementing sequentially and next_pc reflecting potential branch or jump logic. The ALU, guided by ALUControl[3:0], performs operations (e.g., addition, subtraction) on inputs (ALUInput1, ALUInput2), producing results in ALUResult. Instruction decoding signals (Funct, Immediate, ReadData1, ReadData2) specify the operation and data. Memory signals (data_in, data_out, mem_access_addr) handle data flow, with write_enable controlling updates. The Zero flag aids branch evaluations. Clock (clk) synchronizes operations, while reset initializes the processor. The waveform confirms the processor’s expected execution of fetch, decode, execute, and memory phases, demonstrating seamless component coordination.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

The completion of the 32-bit single-cycle MIPS processor project emphasizes the simplicity and efficiency of this architecture, showcasing its capability to execute instructions like arithmetic operations, branching, and memory access within a single clock cycle. By leveraging the reduced instruction set computing (RISC) paradigm, the design achieves streamlined control logic and optimized performance, making it well-suited for tasks requiring minimal latency. The integration of core components such as the ALU, Instruction set, Registers and Memory unit ensures robustness and accuracy in execution. This foundational design serves as a stepping stone for exploring more advanced processor architectures, addressing hardware inefficiencies, and expanding functionalities to meet diverse application demands.

The proposed future enhancements aim to extend the processor's functionality and adaptability across modern computing challenges. Transitioning to a **pipelined architecture** will introduce multiple stages—**fetch**, **decode**, **execute**, **memory access**, and **write-back**—to enable simultaneous instruction processing. This enhancement will significantly improve instruction throughput and resource utilization, making the processor more efficient for complex workloads.

For **IoT optimization**, integrating power-saving techniques such as **power and clock gating** will minimize energy consumption, making the processor ideal for battery-operated and resource-constrained environments. Furthermore, incorporating **hardware security features** like encryption and secure key storage will strengthen the processor's reliability in embedded systems, safeguarding against cyber vulnerabilities.

The future scope also includes **custom AI accelerators**, enabling the processor to handle AI/ML workloads effectively, particularly in edge devices and specialized applications. By adopting **advanced packaging techniques** like **3D integration** and **chiplet designs**, the processor can achieve better performance and a reduced footprint, enhancing its utility in high-performance and space-constrained scenarios. Additionally, leveraging **AI-driven design optimization** can streamline the development process, refining the processor's layout, power efficiency, and testing for superior performance.

REFERENCES

[1] Ahmad, Ahmadi., Reza, Faghih, Mirzaee. (2019). MIPS-Core Application Specific Instruction-Set Processor for IDEA Cryptography - Comparison between Single-Cycle and Multi-Cycle Architectures.

[2] Bhardwaj, Priyavrat & Murugesan, Siddharth. (2017). DESIGN & SIMULATION OF A 32-BIT RISC BASED MIPS PROCESSOR USING VERILOG. International Journal of Research in Engineering and Technology. 10.15623/ijret.2016.0511030.

[3] Design And Analysis Of 64 Bit MIPS Processor", International Journal of Emerging Technologies and Innovative Research (www.jetir.org), ISSN:2349-5162, Vol.5, Issue 5, pageno.1052-1057, May-2018. Available: http://www.jetir.org/papers/JETIR1805188.pdf.

[4] Gross, Thomas R., Hennessy, John L., Przybylski, Steven A., and Rowen, Christopher. (1988). Measurement and evaluation of the MIPS architecture and processor. ACM Trans. Comput. Syst. 6, 3 (Aug. 1988), 229–257. https://doi.org/10.1145/45059.45060.

[5] Hennessy, John, Jouppi, Norman, Przybylski, Steven, Rowen, Christopher, Gross, Thomas, Baskett, Forest, and Gill, John. (1982). MIPS: A microprocessor architecture. Proceedings of the 15th annual workshop on Microprogramming (MICRO 15). IEEE Press, 17–22.

[6] Marri Mounika, Aleti Shankar. "Design & Implementation Of 32-Bit Risc (MIPS) Processor". International Journal of Engineering Trends and Technology (IJETT). V4(10):4466-4474 Oct 2013. ISSN:2231-5381.www.ijettjournal.org. Published by Seventh Sense Research Group.

[7] P. M. B. G. R. (2015). "Improved RISC Processor Design Using MIPS Instruction Set Approach". International Journal on Recent and Innovation Trends in Computing and Communication, 3(5), pp. 2599–2604. doi: 10.17762/ijritcc.v3i5.4292.

[8] Ritpurkar, S. P., Thakare, M. N., and Korde, G. D. (2015). "Design and simulation of 32-Bit RISC architecture based on MIPS using VHDL". 2015 International Conference on Advanced Computing and Communication Systems, Coimbatore, India, pp. 1-6. doi: 10.1109/ICACCS.2015.7324067.

[9] Shobhit, Shrivastav., Shubham, Kumar., Sarthak, Gupta., Bharat, Bhushan. (2020). Qualitative Analysis of 32 Bit MIPS Pipelined Processor. International Journal of Engineering Research and Technology, 9(05). doi: 10.17577/IJERTV9IS050484.

[10] Vidyashree, H. (2023). Design and Implementation of RISC MIPS Processor on FPGA. International Journal For Science Technology And Engineering, 11(4), 1406-1410. Available from: 10.22214/ijraset.2023.50352.