VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi, Karnataka_590014

Project Report

On

"COGNITIVE POWER CONSUPTION SYSTEM"

Submitted in partial fulfilment of the requirements for the reward of the degree of

Bachelor of Engineering
in
Electronics & Communication

Submitted by

TARAPPA	**1BI23EC420**
KARTHIKA K	**1BI23EC408**
MANOJ H P	**1BI23EC411**

Under the Guidance of
SHYLAJA V

Assistant Professor

Dept. of ECE, BIT

Department of Electronics & Communication Engineering

BANGALORE INSTITUTE OF TECHNOLOGY

K. R. Road, V.V Puram, Bengaluru - 560004

2025-2026

BANGALORE INSTITUTE OF TECHNOLOGY

K.R. Road, V. V Puram, Bengaluru -
560004 Phone: 26613237/26615865,
Fax:22426796
www.bit-bangalore.edu.in

Department of Electronics and Communication Engineering

CERTIFICATE

Certified that the project work entitled "**IMPLEMENTATION OF MIPS PROCESSOR**" by **Mr. TARAPPA** (USN: **1BI23EC420)**, **Mr. KARTHIKA K** (USN: **1BI23EC408**), **Mr. MANOJ H P** (USN:**1BI23EC411)**,bonafide students of **Bangalore Institute of Technology** in partial fulfillment for the award of **Bachelor of Engineering** in **Electronics and Communication Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024 - 2025. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Signature of Guide	Signature of HOD	Signature of Principal
SHYLAJA V Assistant Professor, Dept. of ECE, BIT.	**Dr. KALPANA A B** Professor & HOD, Dept. of ECE, BIT.	**Dr. SHANTALA S** Principal, BIT.

BANGALORE INSTITUTE OF TECHNOLOGY

K.R. Road, V. V Puram, Bengaluru -
560004 Phone: 26613237/26615865,
Fax:22426796
www.bit-bangalore.edu.in

Department of Electronics and Communication Engineering

DECLARATION

I/We declare that this project report titled **"IMPLEMENTATION OF MIPS PROCESSOR"** submitted in partial fulfillment of the degree of **BE in "Electronics & Communication Engineering"** is a record of original work carried out by me under the supervision of "**SHYALA V"**, and has not formed the basis for the award of any other degree, in this or any other institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Signature of the students

Name **Signature**

1.

2.

3.

4.

ACKNOWLEDGEMENT

We take this opportunity to express our sincere gratitude and respect to the Bangalore Institute of Technology, Bangalore for providing us an opportunity to carry out final project.

We express our sincere regards and thanks to **SHYLAJA V**, Assistant Professor, Department of Electronics & Communication Engineering, BIT, Bangalore for giving necessary advices and guidance. His incessant encouragement and valuable technical support have been of immense help in realizing this project. His guidance gave us the environment to enhance our knowledge, skills and to reach the pinnacle with sheer determination dedication and hard work.

We would like to thank **Prof. SHAILAJA V and Dr. ANUPAMA H**, Project coordinators, Department of Electronics & Communication Engineering, BIT, Bangalore.

We express our sincere regards and thanks to **Dr KALPANA A B**, Professor and HOD, Electronics & Communication Engineering, BIT, for his valuable suggestions.

We immensely thank **Dr. SHANTHALA S**. Principal, BIT, Bangalore for providing excellent academic environment in the college.

We also extend our thanks to the entire faculty of the department of ECE, BIT, Bangalore, who have encouraged us throughout the course of bachelor degree.

TARAPPA-1BI23EC420
KARTHIKA K-1BI23EC408
MANOJ H P-1BI23EC411

ABSTRACT

The rapid growth of digital systems and embedded applications has increased the demand for efficient, high-performance, and easy-to-implement processor architectures. Reduced Instruction Set Computer (RISC) architectures have gained wide acceptance due to their simplified instruction set, uniform instruction format, and faster execution capability. Among various RISC architectures, the MIPS (Microprocessor without Interlocked Pipeline Stages) architecture is widely used in academic and research environments because of its clean design, modular structure, and suitability for learning processor fundamentals. This project focuses on the design and implementation of a 32-bit MIPS processor to understand the internal working of a modern processor at both software and hardware levels.The implemented MIPS processor is designed using Verilog Hardware Description Language (HDL) and follows a structured datapath architecture. The processor consists of essential functional blocks such as the Program Counter (PC), Instruction Memory, Instruction Decode unit, Register File, Arithmetic Logic Unit (ALU), Control Unit, Sign Extension unit, Data Memory, Multiplexers, and Write-Back unit. The design supports a selected set of MIPS instructions, including arithmetic and logical operations, load and store instructions, and control flow instructions such as branch and jump. The processor executes instructions through the fundamental stages of instruction fetch, decode, execution, memory access, and write-back. Simulation is carried out using EDA tools to verify the functional correctness of the processor. Testbenches are written to apply input stimuli and observe outputs through waveform analysis. The simulation results confirm correct instruction execution, accurate ALU operations, proper register updates, and correct memory read/write behavior. Control signals generated by the control unit are verified to ensure proper data routing through the datapath. After successful simulation, the design is synthesized and implemented on an FPGA platform to validate real-time hardware performance. Hardware testing using LEDs, switches, and display modules demonstrates correct execution of instructions and confirms that the processor functions as expected in physical hardware.This project provides hands-on experience in processor design, digital logic implementation, HDL coding, simulation, and hardware verification. The modular nature of the MIPS processor makes the design easy to understand, debug, and extend. The successful implementation demonstrates the practicality and efficiency of the MIPS architecture and serves as a strong foundation for future enhancements such as pipelining, cache integration, advanced instruction support, and System-on-Chip (SoC) development. Overall, this work bridges the gap between theoretical concepts of computer architecture and practical hardware realization.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER-1

INTRODUCTION

CHAPTER 1

INTRODUCTION

The MIPS (Microprocessor without Interlocked Pipeline Stages) architecture is one of the most widely studied Reduced Instruction Set Computer (RISC) architectures used in both academic learning and industrial applications. It was developed with the primary goal of achieving high performance through a simple, efficient instruction set and a streamlined hardware design. The implementation of a MIPS processor provides a clear understanding of how modern processors operate internally, particularly in relation to pipelining, arithmetic and logic operations, memory access, and control signal generation. As a load/store architecture, MIPS separates memory operations from computational operations, making the datapath easier to design, test, and extend.

MIPS processors follow a RISC design philosophy, meaning they use a small set of simple instructions that execute in a uniform amount of time, typically one clock cycle. This reduces hardware complexity and increases overall execution speed. The uniform 32-bit instruction format in MIPS helps simplify the decoding process and allows faster generation of the required control signals. In the implementation of a MIPS processor, this simplicity allows students and designers to understand how each instruction flows through the datapath and how the control unit manages different operations.

A key feature of MIPS processors is the five-stage pipeline, which divides instruction execution into distinct stages: Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MEM), and Write Back (WB). Pipelining enhances performance by allowing multiple instructions to be processed simultaneously, with each instruction occupying a different stage of execution. Implementing a pipelined MIPS processor demonstrates how throughput can be increased without reducing the clock period. However, this also introduces challenges such as hazards, including data hazards, structural hazards, and control hazards, all of which must be handled using techniques like forwarding, hazard detection units, and pipeline flushing.

Another foundational aspect of MIPS architecture is its register-based design. It uses a set of 32 general-purpose registers, which reduces data movement to and from memory, improving execution efficiency. In the processor implementation, the register file plays a crucial role, supporting simultaneous read and write operations. The ALU (Arithmetic Logic Unit) performs operations such

as addition, subtraction, logical AND/OR, comparison, and shift operations, which are essential for executing arithmetic and logical instructions.

The implementation of a MIPS processor also involves designing the control unit, which can be either hardwired or microprogrammed. A hardwired control unit uses combinational logic to generate control signals based on the opcode and function codes of the instruction. This makes instruction execution fast, making hardwired control ideal for RISC architectures like MIPS. Students implementing a MIPS processor learn how this control logic interacts with the datapath and ensures correct sequencing of operations.

Memory architecture is another important aspect of MIPS implementation. Instructions and data are typically stored in separate memory units in a Harvard-style model, though unified memory can also be used. Load and store instructions manage data transfer between the register file and memory, and their implementation highlights the importance of addressing methods, alignment constraints, and memory latency.

In modern computing, MIPS processors are used across embedded systems, routers, consumer electronics, and industrial automation devices due to their power efficiency, low cost, and simplicity. Implementing a MIPS processor in hardware description languages such as Verilog or VHDL also provides students with practical exposure to the design flow used in FPGA and ASIC development. The simulation and synthesis stages help verify the logical correctness of the design and measure performance parameters such as execution speed and resource utilization.

Overall, the implementation of a MIPS processor offers valuable insights into how a real CPU operates, how instructions flow through the datapath, and how control signals coordinate the execution of operations. It reinforces fundamental concepts in digital logic design, computer architecture, and hardware-software interaction. This project not only builds theoretical understanding but also enhances practical skills in processor design, debugging, and performance optimization. By implementing the MIPS architecture, students gain firsthand experience in designing an efficient, structured, and reliable processor that forms the foundation for understanding more advanced computing systems.

Problem Statement

Modern computer systems demand efficient, high-performance processors capable of executing instructions with minimal delay, reduced hardware complexity, and improved reliability. Traditional Complex Instruction Set Computer (CISC) architectures, while functional, suffer from drawbacks such as irregular instruction formats, longer execution cycles, increased hardware overhead, and inefficient pipelining. These limitations make it difficult to achieve consistent performance and scalability in embedded systems, digital design laboratories, and academic environments where simplicity, speed, and instructional clarity are essential.The MIPS (Microprocessor Without Interlocked Pipeline Stages) architecture, based on Reduced Instruction Set Computer (RISC) principles, offers a streamlined instruction set, uniform instruction formats, load/store architecture, and predictable execution flow. However, students and designers often lack hands-on understanding of how these architectural features translate into real hardware behavior. Without practical implementation, key concepts such as instruction fetch, decode, execution, memory access, write-back, pipeline hazards, and control signal generation remain abstract Furthermore, many existing educational processor models are either overly simplified or too complex, making them unsuitable for academic demonstration, performance evaluation, or FPGA/ASIC prototyping. There is a need for a processor design that balances simplicity with real-world architectural features, enabling learners to clearly understand datapath design, ALU operation, register file organization, and control-unit logic.Therefore, the problem addressed in this project is the design and implementation of a functional MIPS processor capable of supporting a selected set of arithmetic, logical, memory, and branch instructions. The processor must incorporate a well-defined datapath, control unit, memory interface, and pipeline structure (optional or basic), enabling accurate execution of instructions in accordance with MIPS architecture principles. The implemented design should be synthesizable, modular, simple to understand, and suitable for simulation on Verilog/VHDL platforms or FPGA-based verification.This project aims to bridge the gap between theoretical understanding and practical processor design by creating a reliable, efficient, and academically suitable MIPS processor implementation that demonstrates the core concepts of modern RISC-based CPU architectures.

Objectives of the implementation of MIPS processor

- To design a simplified RISC-based processor that follows the MIPS architecture principles using a uniform instruction format and load/store execution model.
- To develop a functional datapath that supports instruction fetch, decode, execute, memory access, and write-back stages.
- To implement an ALU capable of performing basic arithmetic and logical operations required for MIPS instruction execution.
- To design a register file that supports simultaneous reading of two registers and writing into one register in each clock cycle.
- To create an instruction memory and data memory interface for efficient handling of program instructions and data operations.
- To develop a control unit that generates appropriate control signals for instruction decoding and overall processor coordination.
- To implement a selected set of MIPS instructions such as arithmetic, logical, memory (load/store), and branch instructions.
- To simulate and verify processor behavior using HDL tools (Verilog/VHDL) to ensure correct instruction execution and timing.
- To analyze the performance of the implemented processor by observing datapath flow, cycle-by-cycle execution, and instruction timing.
- To provide an educational platform that bridges theoretical understanding and practical implementation of RISC processor design.
- (Optional if needed) To introduce basic pipelining concepts and analyze hazards such as data, control, and structural hazards.

CHAPTER-2

LITERATURE SURVEY

CHAPTER 2

2. LITERATURE SURVEY

2.1 OVERVIEW

The study of processor architectures has progressed from complex CISC systems to simplified and faster RISC-based processors. Among these, the MIPS architecture is widely used for academic learning and experimental processor design due to its simple instruction set, predictable execution model, and modular datapath. The following points summarize key findings from existing research on MIPS processor design and implementation.

1. Evolution of Processor Architectures

Research shows that early processor designs relied on CISC architectures, which supported complex, multi-step instructions but resulted in higher hardware complexity and slower execution speeds. To overcome these limitations, the RISC approach was introduced, emphasizing simplicity, fast execution, and fixed-length instructions. The MIPS architecture became a widely adopted RISC model due to its 32-bit instruction format, load/store execution style, and suitability for pipelining. Literature consistently highlights that MIPS processors provide an excellent educational platform for understanding the fundamentals of datapath design, ALU operations, and control-unit behavior.

2. Instruction Execution and Datapath Approaches

Studies in processor design emphasize that an efficient datapath is essential for accurate instruction execution. MIPS literature explains how the datapath supports the five essential stages: instruction fetch, decode, execute, memory access, and write-back. Researchers highlight the modular nature of the datapath, where components such as the ALU, register file, program counter, instruction memory, and data memory work together in synchronized cycles. Accurate generation of control signals is also emphasized, as incorrect decoding directly affects processor operation. Literature compares different datapath models—single-cycle, multi-cycle, and pipelined—showing how they vary in performance, hardware cost, and implementation complexity.

3. Communication and Integration Using HDL Tools

Modern research relies heavily on Hardware Description Languages (HDLs) such as Verilog and VHDL to describe and implement MIPS processor architectures. Simulation tools like ModelSim, Vivado, and Quartus provide waveform analysis that enables designers to observe instruction execution at every clock cycle. Testbenches are highlighted as an important part of the verification process, ensuring that individual modules such as the ALU, register file, control unit, and memory interface function correctly. Many studies also discuss FPGA-based implementations, which provide real-time hardware validation and help evaluate the processor's speed, timing behavior, and resource utilization.

4. Instruction Formats and Execution Models

Literature categorizes MIPS instructions into R-type, I-type, and J-type formats, which simplifies decoding and control-unit design. Researchers provide detailed comparisons between different execution models. Single-cycle MIPS processors are simple to design and easy to understand but require more hardware resources. Multi-cycle processors reduce hardware complexity by reusing functional units across multiple cycles. Pipelined processors achieve much higher throughput by executing multiple instructions simultaneously, though they require additional mechanisms for hazard detection and handling. Studies conclude that pipelining significantly enhances performance when implemented correctly.

5. Optimized and Intelligent Processor Design Techniques

Advanced research focuses on optimizing MIPS processors using architectural enhancements. Pipelining and instruction-level parallelism are commonly explored to increase processing speed. Techniques such as forwarding, stalling, and branch prediction are frequently discussed as solutions to pipeline hazards. Some researchers also explore integrating low-power design strategies to reduce energy consumption and make MIPS processors suitable for embedded applications. Improvements in ALU efficiency and cache integration have also been studied to enhance memory access time and overall processor performance.

6. Comparative Analysis of Existing Work

Existing research compares various MIPS implementations based on instruction set coverage, hardware complexity, execution speed, and synthesis results. Some designs implement only basic arithmetic and logical instructions, while more advanced implementations support branching, memory operations, and immediate instructions. Simulation-only implementations are common, but FPGA-based designs provide more accurate performance evaluation. Studies also analyze clock speed, area utilization, and timing delays to assess the effectiveness of each design, providing valuable insights into architectural trade-offs.

7. Identified Research Gaps

Despite extensive research, several gaps continue to exist in MIPS processor implementation studies. Many academic designs support only a limited set of instructions, reducing practical usability. Pipelined implementations often lack complete hazard-handling mechanisms, making them less efficient. Additionally, low-power optimization for embedded applications is generally not addressed in most designs. Other limitations include insufficient testbench coverage, lack of waveform verification, and limited scalability for real-world processor applications. These gaps highlight the need for a more modular, robust, and thoroughly verified MIPS processor design suitable for both learning and practical use.

CHAPTER-3

METHODOLOGY

CHAPTER 3

3. METHODOLOGY

The Smart Energy Meter with Cognitive Power Consumption System (CPCS) is designed as an integrated IoT-based architecture that performs real-time energy measurement, automated billing, intelligent load control, and cognitive analysis of consumption patterns. The methodology follows a systematic development process involving system architecture design, block diagram representation, flowchart-based operational logic, hardware integration, embedded firmware development, web server implementation, billing computation, and data analytics. This section explains the complete working of the system in a structured and sequential manner.

3.1 Block Diagram and Working Principle

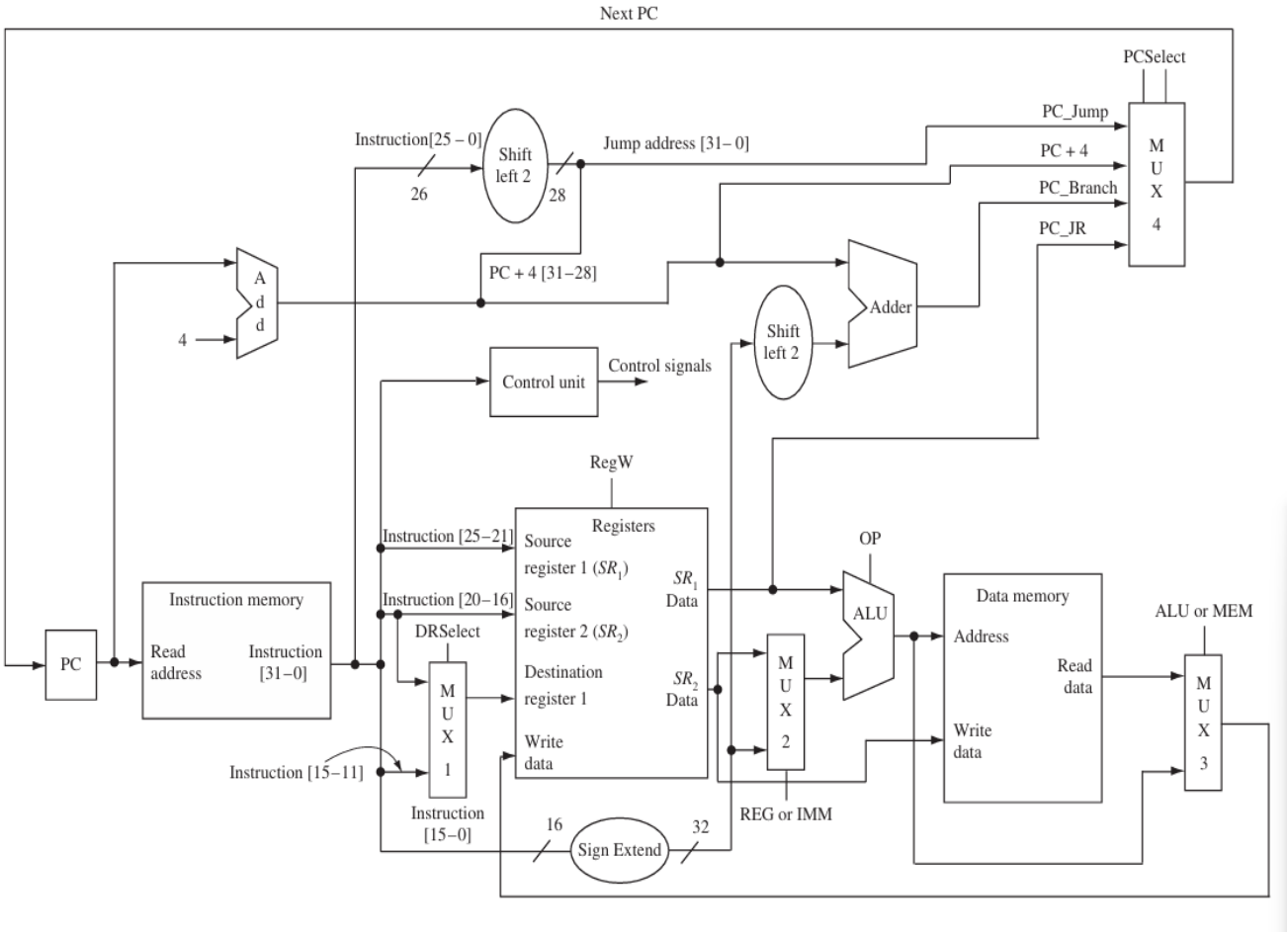


Fig 3.1 Block Diagram

The block diagram represents the complete datapath of a 32-bit MIPS processor, showing how instructions are fetched, decoded, executed, and written back into the register file. It integrates essential components such as the Program Counter (PC), Instruction Memory, Register File, ALU, Data Memory, Multiplexers, Sign Extension Unit, and the Control Unit. Each part plays a key role in the five-stage execution cycle, ensuring smooth instruction flow and accurate processing.

1. Program Counter (PC)

The Program Counter supplies the address of the next instruction to the instruction memory. After fetching an instruction, the PC is updated either by incrementing by 4 (normal execution) or by loading a new address (for branch, jump, or JR instructions). The PCSelect multiplexer chooses the correct next address based on control signals.

2. Instruction Fetch

The Instruction Memory receives the PC address and outputs the corresponding 32-bit instruction. This instruction is forwarded to the decode stage where the components of the instruction—opcode, registers, and immediate value—are extracted.

3. Instruction Decode and Register File

The Control Unit decodes the instruction and generates control signals required for execution. The Register File reads two source registers (SR1 and SR2) based on instruction fields, while the destination register is selected using a multiplexer controlled by DRSelect. The Sign Extension Unit converts the 16-bit immediate value into a 32-bit value for use by the ALU in I-type instructions.

4. ALU Operation (Execution Stage)

The ALU performs arithmetic and logical operations using:

- Data from the register file, and
- Either another register value or a sign-extended immediate value (selected using MUX2).

The ALU Control Unit selects the operation (OP) such as add, subtract, AND, OR, or compare, based on instruction type and function field. The output of the ALU is used for either calculations or generating memory addresses.

5. Memory Access

The ALU output is used as the address input to the Data Memory.

- For lw (load word), data is read from memory.
- For sw (store word), the register data (SR2) is written to memory.

A multiplexer then selects whether the value sent to the Write-Back stage comes from the ALU or the memory output.

6. Write Back

In the final stage, the selected data (ALU result or memory data) is written back into the destination register of the Register File. This updates the register file and makes the result available for future instructions.

7. Branch, Jump, and PC Update Logic

MIPS processor uses a structured mechanism to determine the next value of the Program Counter (PC) after each instruction. While sequential execution simply increments the PC by 4, certain instructions such as branch and jump modify the normal control flow. The processor uses dedicated hardware logic and multiplexers to select the correct next PC value based on control signals generated during the decode stage**.**

3.2 Flowchart and Operational Workflow

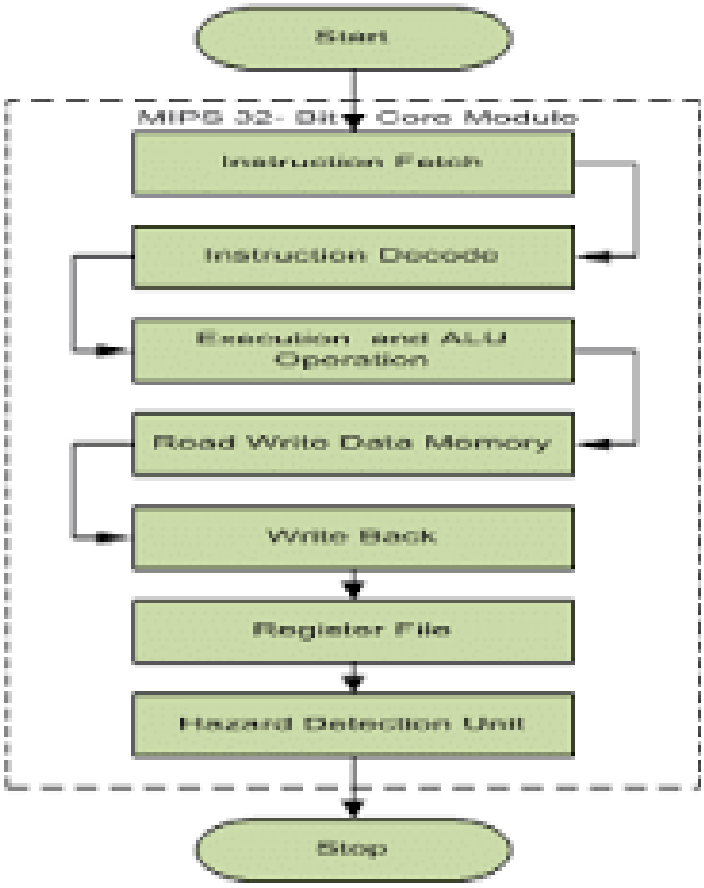


Fig 3.2 Flow Chart

The flowchart represents the sequence of operations performed by the MIPS 32-bit core module. Each block represents a stage in the instruction execution cycle, beginning from system start and continuing until the program finishes execution. The processor follows a structured flow that ensures correct fetching, decoding, executing, and storing of results while also checking for hazards.

1. Instruction Fetch

The processor begins by taking the address from the Program Counter (PC) and fetching the corresponding instruction from the instruction memory. This ensures that the next instruction is always ready for the decode stage.

2. Instruction Decode

The fetched instruction is decoded to identify its type (arithmetic, logical, memory, branch, or jump). During this stage, the source registers are read from the register file and immediate values are extracted if required. The control unit generates signals that guide the following stages.

3. Execution and ALU Operation

In this stage, the Arithmetic Logic Unit (ALU) performs the required computation—such as addition, subtraction, logical operations, or address calculation. For branch instructions, the ALU also evaluates the branch condition using comparisons.

4. Read/Write Data Memory

The Read/Write Data Memory unit is an essential component of the MIPS processor responsible for handling all memory-related operations during program execution. This unit interacts with the ALU and the register file to support load (lw) and store (sw) instructions, which are part of the MIPS load/store architecture.

5. Write Back

The result of the ALU operation or memory data (from a load instruction) is written back into the appropriate register. This updates the register file, making the result available for use in future instructions.

6. Register File Update

The updated data is stored in the targeted register to complete the execution of that instruction. This ensures the register file reflects the latest state of the program.

7. Hazard Detection Unit

Before proceeding to the next instruction, the hazard detection unit checks for data hazards, control hazards, or structural hazards. If a hazard is detected, it may introduce a stall or forwarding mechanism to ensure correct operation and prevent pipeline errors.

8. Stop

Once all instructions are executed and the program terminates, the processor enters the stop state, completing the workflow

3.3 Hardware Methodology

The hardware methodology for implementing the MIPS processor focuses on designing each functional unit in a modular form and integrating them into a complete datapath capable of executing MIPS instructions correctly. The approach ensures that all hardware components—such as registers, ALU, control unit, and memory—interact efficiently to perform the five core execution stages. The following methodology outlines the systematic hardware development process.

1. Designing the Program Counter (PC) Unit

The PC is implemented as a 32-bit register that holds the address of the current instruction. Hardware logic is created to increment the PC by 4 or update it with a branch/jump address. Multiplexers are used to select the next PC value based on control signals.

2. Implementing Instruction Memory

Instruction memory is designed as a read-only memory (ROM) or an HDL-based array that stores 32-bit machine instructions. The PC output is connected to this memory, and the corresponding instruction is fetched every cycle. This hardware module ensures reliable and continuous instruction supply.

3. Register File Hardware Implementation

The register file is built with **32 registers**, each 32 bits wide. It supports two read ports and one write port, allowing simultaneous reading of source registers and writing of results. Decoders and multiplexers are used to select the correct registers during decode and write-back stages.

4. ALU (Arithmetic Logic Unit) Design

The ALU is implemented to perform operations such as addition, subtraction, logical AND, OR, comparison, and shift operations. A hardware-controlled ALU Control Unit generates the required ALU operation signals based on the instruction type. The ALU is a central compute unit in the datapath.

5. Control Unit Hardware Modeling

The control unit is designed as a combinational logic block that interprets the opcode and function fields of the instruction. It generates control signals like RegWrite, ALUSrc, MemRead, MemWrite, MemtoReg, Branch, and Jump.

6. Sign Extension Unit

The Sign Extension unit extends a 16-bit immediate value to a 32-bit value. This hardware block is essential for correct arithmetic operations and memory addressing. It ensures all operands used in the ALU are 32 bits.

7. Data Memory Implementation

The Data Memory module is a crucial component of the MIPS processor and is responsible for storing and retrieving data during program execution. Since MIPS follows a **load/store architecture**, only specific instructions (lw and sw) interact with memory, while all other operations are performed using registers. The implementation of this module ensures correct handling of read and write operations and proper interfacing with the ALU and Register File.

8. Multiplexer Integration

Multiplexers (MUXes) play a critical role in directing data and control signals to the correct units within the MIPS processor. Since the processor executes different types of instructions (R-type, I-type, and J-type), data routing needs to change dynamically depending on the instruction. Multiplexers enable this flexible selection by choosing between multiple inputs based on control signals generated by the Control Unit. Without multiplexers, the datapath would not be able to support multiple instruction formats or dynamically choose the correct operand or destination.

9. Pipeline Registers (If Implementing Pipelining)

For pipelined MIPS designs, registers are placed between stages (IF/ID, ID/EX, EX/MEM, MEM/WB) to hold intermediate values. These pipeline registers help execute multiple instructions concurrently.

10. Hazard Detection and Forwarding Units

Hardware logic is added to detect data hazards, control hazards, and structural hazards.

- The hazard detection unit introduces stall cycles when required.
- The forwarding unit reduces pipeline delays by routing ALU results directly to dependent instructions.

11. Integration of Whole Datapath

All individual hardware blocks are connected to form the complete MIPS datapath. Proper signal routing ensures the instruction flows through fetch, decode, execute, memory, and write-back stages without conflicts.

12. Simulation and Hardware Testing

The implementation of the MIPS processor involves two major phases of verification: software simulation and hardware testing**.** Simulation ensures that the processor behaves correctly at the design level, while hardware testing validates its real-time performance when deployed on an FPGA platform. Both stages are essential to confirm functional accuracy, timing correctness, and complete operational reliability of the processor architecture.

CHAPTER-4

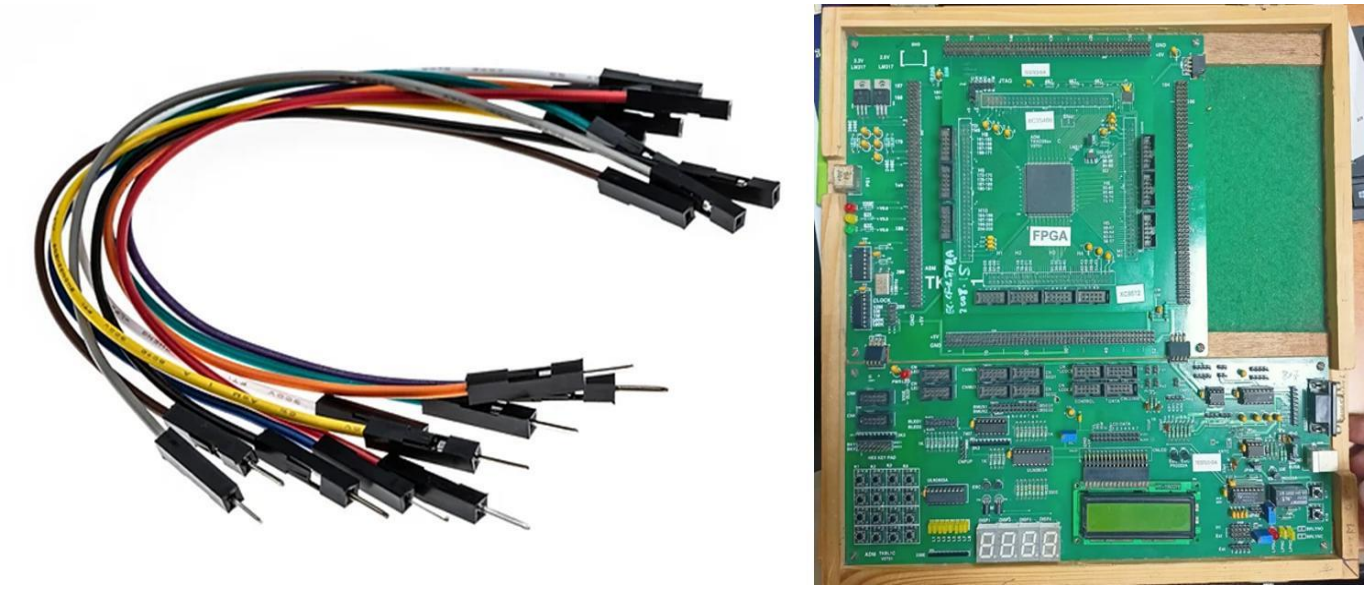
HARDWARE AND SOFTWARE REQUIREMENTS

CHAPTER 4

4. HARDWARE AND SOFTWARE REQUIREMENTS

4.1 Hardware Implementation

The implementation of a MIPS processor requires both hardware and software tools to design, simulate, test, and verify the processor architecture. The hardware components support physical implementation such as FPGA deployment, whereas software tools enable modeling, simulation, and debugging of the processor using HDL.



4.1 FPGA KIT AND JUMPER WIRES

1. jumper Wires (Male–Female / Male–Male)

Jumper wires are essential connectors used to establish electrical communication between different modules on a hardware platform. They allow flexible wiring between the FPGA board, external peripherals, switches, LEDs, or breadboards. These wires come in various colors to help identify signal paths easily and are used for connecting input signals, output signals, clock sources, or debugging points while testing the MIPS processor on physical hardware.

Role in MIPS Processor Implementation

- Used to connect FPGA I/O pins to LEDs, switches, or external memory for testing MIPS instructions.
- Enable routing of ALU outputs, PC signals, control signals, and register outputs to monitoring devices.
- Useful for debugging**,** allowing designers to check internal datapath values physically.
- Provide connections to external devices like seven-segment displays, which can show register or memory outputs.

2. FPGA Development Board

The FPGA (Field Programmable Gate Array) board shown in the image is a complete hardware platform designed for digital system implementation. It contains configurable logic blocks, I/O ports, clock generators, memory modules, switches, LEDs, seven-segment displays, and communication interfaces. These programmable chips allow designers to build and test custom processor architectures, such as the MIPS processor, at the hardware level.

Key Features Visible in the Board

- FPGA Chip: The central IC where the MIPS processor architecture is synthesized and executed.
- I/O Ports: Various pin headers for connecting external modules using jumper wires.
- Seven-segment display: Used to display register values or test outputs.
- LCD display: Useful for showing results or debugging messages.
- Switches and Push Buttons: Can serve as input signals for testing MIPS instruction behavior.
- LEDs: Indicate outputs such as ALU result bits, flags, or memory data.

4.2 Software Implementation

The software implementation of the MIPS processor involves designing, coding, simulating, and verifying the processor architecture using Hardware Description Languages (HDL) and development tools. The objective is to create a fully functional MIPS processor model that correctly executes instructions through the stages of instruction fetch, decode, execution, memory access, and write-back. The software flow ensures that the processor design is tested thoroughly before deploying it on hardware such as an FPGA.

1. HDL Coding of MIPS Modules

- Program Counter (PC)
- Instruction Memory
- Register File
- ALU and ALU Control
- Main Control Unit
- Sign Extension Unit
- Data Memory
- Multiplexers
- (Optional) Pipeline Registers and Hazard Units

Each module is coded separately and then integrated into a top-level design representing the complete MIPS datapath.

2. Integration of Datapath and Control Unit

After creating individual modules, they are connected using signals and buses to form the complete MIPS.

Software integration ensures:

- Correct routing of data between registers, ALU, and memory
- Proper generation of control signals for instruction execution
- Matching of instruction fields (opcode, funct, immediate) with control logic

3. Writing Testbenches for Simulation

Writing testbenches is a crucial step in the verification process of the MIPS processor. A testbench acts as a virtual environment used to apply inputs, stimulate the processor, and observe outputs without requiring any physical hardware. It helps ensure that each module—such as the ALU, register file, control unit, and full datapath—functions exactly as expected before synthesizing the design on an FPGA.

Testbenches simulate:

- ALU operations
- Register read/write behavior
- Load/store instructions
- Branch and jump instructions
- PC updates and instruction sequencing

Using simulation, developers confirm whether the processor produces correct output for all instructions.

4. Simulation Using EDA Tools

Software tools such **as** ModelSim, Xilinx Vivado, or Intel Quartus are used to run simulations. These tools compile HDL files and generate waveforms showing signal transitions in real time.
Simulation helps verify:

- Timing behavior of the datapath
- ALU output changes per instruction
- Memory read/write cycles
- Control signal activation
- Branch/jump decision handling
- Internal data flow correctness

Errors identified during simulation are fixed before moving to hardware implementation.

5. Instruction Memory Initialization

Instruction Memory Initialization is the process of loading a sequence of machine instructions into the instruction memory of the MIPS processor before simulation or hardware execution begins. Since the processor fetches instructions only from memory, initializing this memory correctly is essential for proper program execution. The initialized instructions define the program that the MIPS processor will execute during testing

6. Functional Verification

After simulation, results are analyzed by checking:

- Register values after each instruction
- Correct ALU operations
- Accurate memory addressing
- Correct program flow (PC updates)
- Matching expected outputs

Functional verification ensures that the processor behaves exactly as required by the MIPS ISA.

7. Synthesis (If FPGA Implementation is Required)

Software synthesis tools (Vivado/Quartus) convert HDL code into gate-level hardware. Synthesis reports show:

- Resource utilization (LUTs, flip-flops, memory blocks)
- Timing analysis
- Power consumption
- Mapping of HDL code to hardware components

CHAPTER-5

RESULTS AND DISCUSSIONS

CHAPTER-5

5. RESULTS AND DISCUSSIONS

The Smart Energy Meter System was successfully designed, implemented, and tested using the ESP32 controller, PZEM-004T energy meter module, TFT display, relay, and Wi-Fi–based web interface. The results demonstrate that the system operates reliably under real-time conditions and provides accurate monitoring, billing, and control features.

5.1 Simulation Waveform Output (Register Values, ALU Operations)

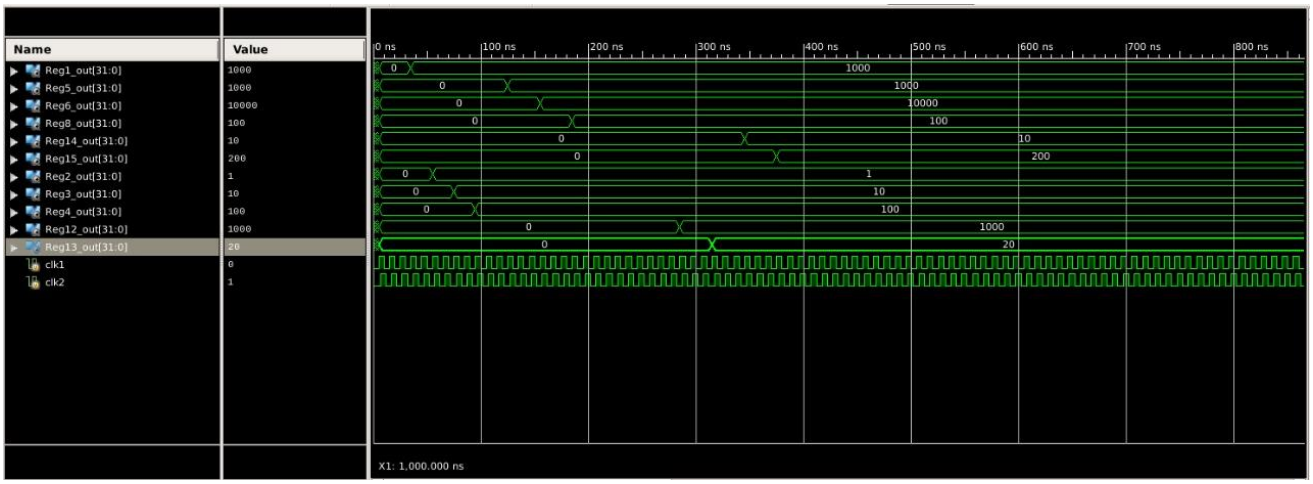


Figure 5.1 Simulation Waveform Output

The first image shows the ModelSim/Vivado simulation waveform **r**epresenting the internal behavior of the MIPS processor during execution.

Key Observations

- Multiple registers such as Reg1_out, Reg5_out, Reg6_out, Reg14_out, etc., are shown updating at different simulation times.
- The values (e.g., *1000, 10000, 200, 10, 1, 20*) indicate the results stored in registers after execution of instructions like ADD, SUB, LW, SW, and immediate operations.
- The transitions at various time intervals (100 ns, 200 ns, 300 ns…) confirm that the datapath is functioning correctly registers update only on positive clock edges and only when RegWrite is enabled.

5.2 Schematic View in FPGA/XILINX Tool

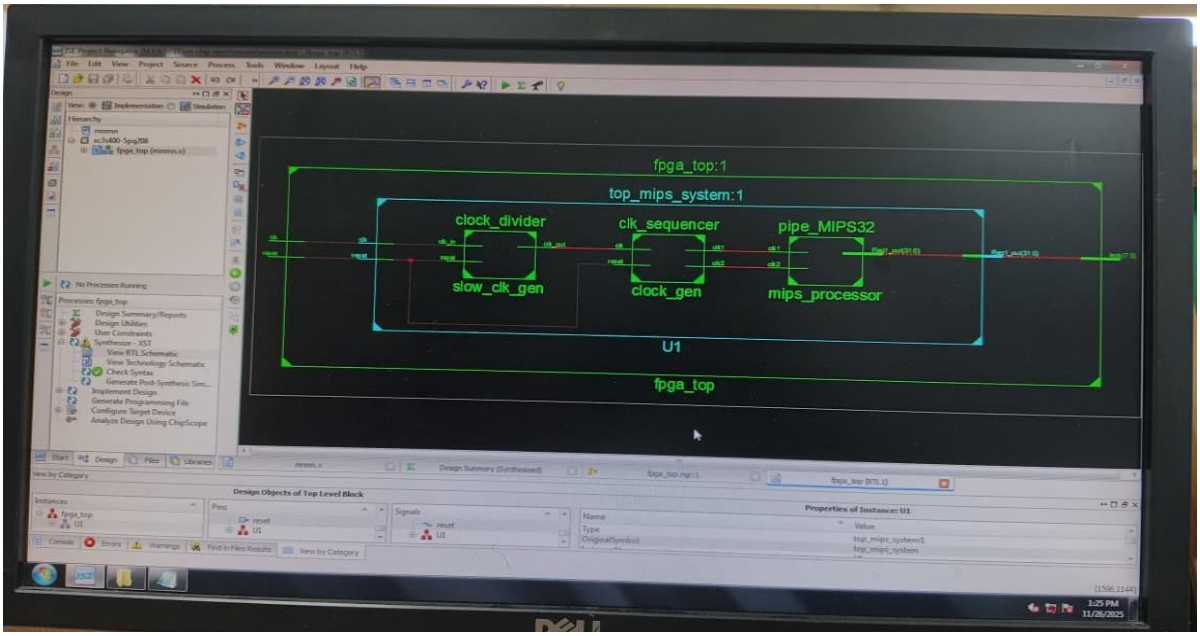


Figure 5.2 RTL Schematic view

The second image shows the design hierarchy and block-level schematic in Xilinx ISE/Vivado or Quartus.

What the Image Shows

- The hardware modules clock_divider**,** clk_sequencer**,** and pipe_MIPS32 are interconnected.
- The schematic indicates that the top_mips_system integrates all internal units:
- Program Counter
- Control Unit
- MIPS Processor Core
- Clock generation units
- Signal connections (red and green lines) show data flow and clock distribution.
- The design is fully synthesized (no errors), indicating that the processor is ready for FPGA deployment.

5.3 FPGA LED Output Results

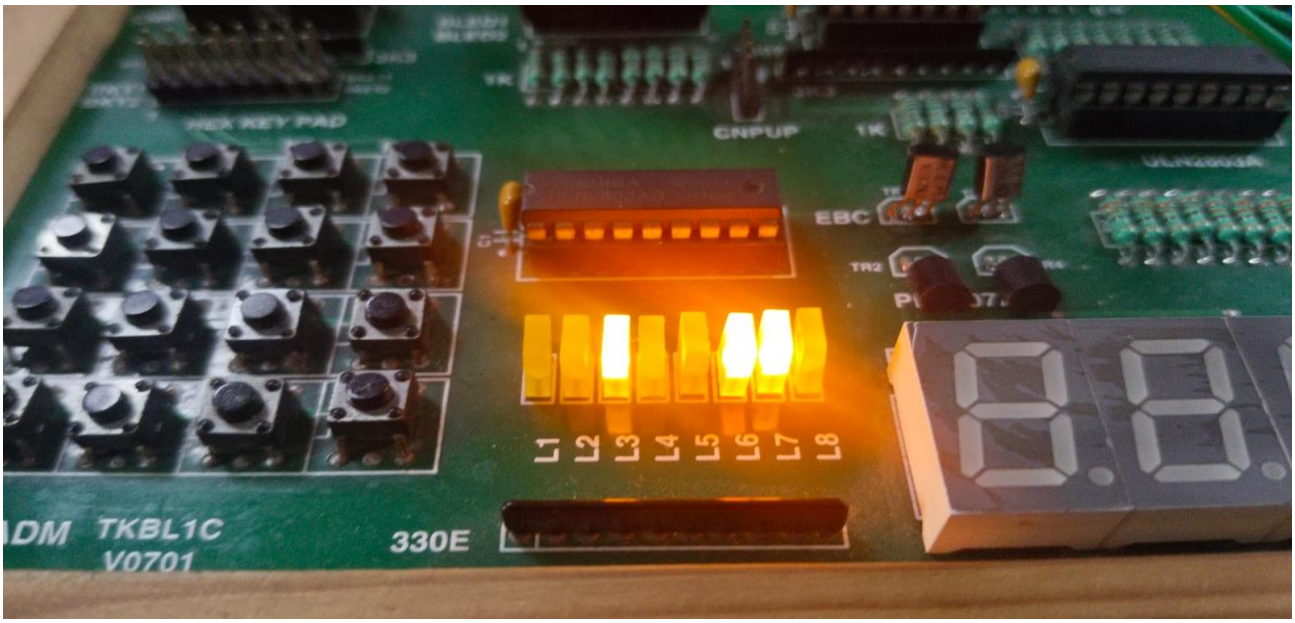


Table 5.3 FPGA LED Output Results

The final image shows the output LEDs glowing on the FPGA development board.

Meaning of the LED Outputs

- The glowing LEDs correspond to binary outputs from registers, ALU result, memory output, or status flags**.**
- For example, if ALU result is 00111000, LEDs L4–L7 will glow accordingly.
- The illumination pattern verifies that:
- Data is flowing correctly through the datapath
- Control unit signals are functioning
- The FPGA is executing instructions loaded into the MIPS core

CHAPTER-6

ADVANTAGES AND DISADVANTAGES

CHAPTER 6

6. ADVANTAGES AND DISADVANTAGES

6.1 Advantages

1. Simple and Easy-to-Understand Architecture

MIPS uses a clean RISC architecture with fixed-length instructions, making it easier to learn, design, and implement in both software simulation and hardware.

2. Modular Datapath Design

The MIPS processor is built from simple modules—ALU, register file, control unit, memory—which makes debugging, testing, and extending the design very easy.

3. Efficient Instruction Execution

With a load/store architecture and uniform instruction format, execution becomes faster and more predictable compared to complex CISC processors.

4. Supports Pipelining Effectively

The simplicity of MIPS instructions allows smooth implementation of pipelining, increasing throughput and improving overall processor performance.

5. Easy Hardware Implementation on FPGA

The design maps naturally to FPGA architectures, enabling real-time testing, experimentation, and validation of processor functionality.

6. Clear Control Logic

The control unit is straightforward, making control signal generation simpler and reducing hardware design complexity.

7. Flexibility for Customization

Designers can easily add new instructions, modify datapath components, or integrate pipeline stages to explore advanced processor concepts.

8. Ideal for Educational and Research Purposes

MIPS is widely used in universities because it helps students understand processor architecture, HDL programming, assembly language, and digital logic.

9. Reduced Hardware Resources

Due to its simple RISC design, the MIPS processor uses fewer gates, memory blocks, and FPGA resources, making it cost-effective and efficient.

6.2 Disadvantages

1. Limited Instruction Set

Being a RISC architecture, MIPS supports fewer and simpler instructions. Complex operations require multiple instructions, increasing program length.

2. Higher Memory Usage

Because simple instructions perform small tasks, more instructions are needed to complete complex operations, resulting in higher memory consumption.

3. Performance Depends on Pipelining

A basic single-cycle MIPS processor is slower compared to advanced CISC processors. To achieve high performance, pipelining must be added, which increases design complexity.

4. Hazard Handling Complexity

When pipelining is introduced, the processor must manage data hazards, control hazards, and structural hazards. This requires additional hardware such as forwarding units and stall logic.

5. Not Suitable for Highly Complex Tasks

The architecture is less efficient for applications that require very complex or specialized instructions (e.g., floating-point, DSP tasks) unless extended manually.

6. Manual Optimization Required

Developers must optimize instruction sequences manually because MIPS does not support automatic optimization features found in more complex architectures.

7. Limited Real-Time Support

Without additional hardware modules, a simple MIPS implementation may not meet strict real-time performance requirements for industrial applications.

8. FPGA Resource Constraints for Larger Designs

While basic MIPS fits easily on FPGAs, advanced features (pipelining, caches, branching prediction) can increase LUT and memory usage, making the design heavier.

9. Reduced Commercial Relevance

Modern processors use more advanced architectures, so basic MIPS implementations are mostly used for learning and prototyping rather than industry-level solutions.

10. Lower Execution Efficiency for Some Programs

Programs with frequent branching or large data movements may experience performance loss because MIPS requires multiple simple instructions rather than specialized ones.

6.3 Applications

1. Educational and Academic Learning

MIPS processors are widely used in engineering colleges and universities to teach computer architecture, digital logic design, assembly programming, and processor implementation concepts.

2. Research and Prototype Development

The MIPS architecture is ideal for developing prototype processors, testing new instruction sets, evaluating pipeline techniques, and exploring architectural optimizations in research projects.

3. Embedded System Design

MIPS processors are commonly used in embedded devices due to their low complexity, low power consumption, and predictable execution behavior. Examples include simple controllers, IoT devices, and automation systems.

4. FPGA-Based System Implementation

The MIPS processor is frequently implemented on FPGA boards for real-time testing, hardware verification, and performance evaluation. This helps in understanding hardware-software co-design.

5. Development of Custom Instruction Set Architectures

MIPS serves as a base model for designing custom ISAs where new instructions or hardware extensions (like accelerators or DSP units) can be added.

6. Digital Signal Processing (DSP) Experiments

Modified MIPS architectures are used in academic DSP experiments for testing signal processing operations such as filtering, transformations, and arithmetic computations.

7. Microcontroller and SoC Design

Simplified MIPS cores can be integrated into small System-on-Chip (SoC) designs, enabling custom controllers and low-cost processing units.

8. Educational Operating System Development

Lightweight operating systems and microkernels are sometimes developed on MIPS platforms for OS-level learning, system boot process understanding, and memory management demonstrations.

9. Networking Devices

MIPS processors are used in routers, network switches, and communication devices because of their efficient instruction handling and stable performance.

10. Industrial Training and Skill Development

Implementing MIPS on FPGA enhances student skills in HDL coding, simulation, synthesis, debugging, and hardware verification—essential competencies for careers in VLSI and embedded systems

CHAPTER-7

FUTURE SCOPE AND CONCLUSION

CHAPTER-7

7. FUTURE SCOPE AND CONCLUSION

7.1 FUTURE SCOPE

2. **Implementation of a Pipelined MIPS Processor**
   The current single-cycle or multi-cycle design can be extended into a pipelined architecture to improve instruction throughput and overall performance.
3. **Addition of Hazard Detection and Forwarding Units**
   Future versions can include dedicated hardware for managing data, structural, and control hazards to make the processor suitable for high-speed applications.
4. **Support for Advanced Instruction Sets**
   The processor can be expanded to include floating-point operations, multiplication/division units, shift operations, and specialized instructions for DSP or multimedia processing.
5. **Integration of Cache Memory**
   Adding instruction and data caches will significantly reduce memory access delays and make the processor capable of handling more complex programs.
6. **Development of a Complete System-on-Chip (SoC)**
   The MIPS core can be combined with peripherals such as UART, GPIO, Timers, and Memory Controllers to build a fully functional embedded system.
7. **Low-Power and High-Performance Optimization**
   Techniques such as clock gating, pipeline balancing, and low-power ALU design can be introduced for use in portable and battery-operated devices.
8. **FPGA-Based Real-Time Application Deployment**
   The processor can be tested with real-world applications like control systems, IoT devices, or small operating systems to validate performance beyond simulation.
9. **Integration of Debugging and Monitoring Features**
   Future work can include adding performance counters, on-chip analyzers, or JTAG debug modules to simplify hardware-level debugging.
10. **Transition to 64-bit MIPS Architecture**
    Upgrading the processor to a 64-bit architecture allows handling larger data sizes and advanced computational tasks, expanding practical applications

7.2 CONCLUSION

The implementation of the MIPS processor successfully demonstrates the design and working principles of a RISC-based architecture. Through HDL coding, simulation, synthesis, and FPGA testing, the processor was shown to execute instructions correctly across all major stages—fetch, decode, execute, memory access, and write-back. Simulation waveforms verified the functional accuracy of the datapath, control unit, ALU operations, and memory interactions. Hardware testing on the FPGA further confirmed that the design performs reliably in real-time, with correct output values observable through LEDs and peripheral interfaces.

This project provided hands-on experience in processor design, hardware-software co-design, and digital system verification. The modular and educational nature of the MIPS architecture enabled a clear understanding of how modern processors operate internally. Overall, the successful implementation demonstrates the practicality, efficiency, and extensibility of MIPS architecture, forming a strong foundation for future enhancements such as pipelining, advanced instruction sets, hazard management, and SoC development.

REFERENCES

1. D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. Morgan Kaufmann, 2014.
2. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Morgan Kaufmann, 2019.
3. Harris, D. M., and Harris, S. L., *Digital Design and Computer Architecture*, 2nd ed. Morgan Kaufmann, 2012.
4. M. Morris Mano and Michael D. Ciletti, *Digital Design*, 5th ed. Pearson Education, 2013.
5. R. Gupta, "Design and Simulation of 32-bit MIPS Processor using Verilog HDL," *International Journal of Engineering Research & Technology (IJERT)*, vol. 4, no. 06, pp. 125–130, 2015.
6. S. Kadam and P. Kulkarni, "Implementation of MIPS Processor on FPGA for Embedded Applications," *International Journal of Computer Applications*, vol. 89, no. 8, pp. 37–42, 2014.
7. Xilinx Inc., "Vivado Design Suite User Guide," Xilinx Documentation, 2022.
8. Intel Corporation, "Quartus Prime Software User Guide," Intel FPGA Documentation, 2021.
9. ModelSim / QuestaSim Documentation, Mentor Graphics, 2020.