IMPLEMENTATION Of MIPS(PROCESSOR)

1st Karthika k
School of ECE
Bangalore Institute of Technology
Bangalore, India
karthika9758@gmail.com

2nd Tarappa
School of ECE
Bangalore Institute of Technology
Bangalore, India
tarappamsg211@gmail.com

3rd Manoj H P
School of ECE
Bangalore Institute of Technology
Bangalore, India
manojhp222@gmail.com

Shylaja V
Assistant Professor of ECE Dpt
Bangalore Institute of Technology
Bangalore, India
shylajav@bit-bangalore.edu.in

ABSTRACT - The rapid growth of embedded systems and digital computing demands processor architectures that are both efficient and hardware-friendly. The Microprocessor without Interlocked Pipeline Stages (MIPS) architecture, a classic RISC model, offers a streamlined instruction set and modular design that makes it ideal for academic study and real-time hardware implementation. This project presents the design and FPGA-based implementation of a 32-bit MIPS processor, developed using Verilog HDL and verified through industry-standard simulation and synthesis tools. The architecture integrates essential components including the Program Counter, Instruction Memory, Register File, ALU, Control Unit, Data Memory, Multiplexers, and Sign Extension blocks, forming a complete datapath capable of executing arithmetic, logical, memory access, and branching instructions. the design follows a systematic methodology beginning with RTL modelling of core functional units, followed by integration into a multi-stage datapath and verification using Xilinx toolchains.

I. INTRODUCTION

The rapid growth of embedded systems and domain-specific computing has renewed interest in lightweight, modular processor architectures that balance performance, simplicity and ease-of-implementation. The MIPS (Microprocessor without Interlocked Pipeline Stages) architecture, a canonical Reduced Instruction Set Computer (RISC) design, remains a popular educational and research platform because of its clean instruction set, regular datapath and clear separation between datapath and control logic.

This paper presents the RTL design and FPGA implementation of a 32-bit MIPS processor developed in Verilog HDL, with a focus on producing a modular, easily verifiable design that can be synthesized and tested on contemporary FPGA toolchains. Our design implements the essential components of a MIPS pipeline — instruction fetch (IF), instruction decode (ID), execute (EX), memory access (MEM) and write-back (WB) — and integrates core modules such as the program counter, instruction memory, register file, arithmetic–logic unit (ALU), data memory, sign-extension and multiplexers into a cohesive datapath and control unit. The processor supports the basic Rtype, I-type and J-type instruction formats, enabling fundamental arithmetic, logical, memory and branching operations required by typical MIPS programs. The architecture prioritizes modularity so that each block (ALU, control unit, register file, memory interface) can be developed, simulated and verified independently before system integration.

II. LITERATURE REVIEW

The MIPS architecture has been widely studied as a simple and efficient RISC processor model. Earlier works focus on the clean instruction set, modular datapath and ease of hardware realization. Classical textbooks by Hennessy & Patterson describe MIPS as an ideal teaching architecture because of its regular structure and reduced instruction complexity. Many hardware designs implement the core modules—ALU, register file, program counter, control unit and memory blocks—in HDL to demonstrate fundamental processor operation. existing research emphasises performance improvement through pipelining techniques, clock optimization and power-aware design. Several studies aim to increase throughput using deeper pipelines and hazard-reduction mechanisms. However, many such designs sacrifice architectural simplicity. Prior works often lack complete verification, especially full testbench-driven simulation and FPGA-based testing.

Recent studies highlight the importance of implementing R-type, I-type and J-type instruction formats to validate functional correctness. Researchers also stress the need for a modular design approach to ensure that each unit—ALU, control logic, sign-extension, multiplexers and memory—can be independently tested before system integration. The literature shows a gap in balanced MIPS implementations that combine clarity, modularity, complete instruction support and practical FPGA validation. There is a need for designs that are simple enough for education yet complete enough for real hardware demonstration. Your project aligns with this requirement by providing a clean, Verilog-based, FPGA-tested 32-bit MIPS processor.

III. PROPOSED WORK

The proposed work aims to design, simulate, and implement a 32-bit MIPS processor following the Prosser architecture model. The project focuses on developing a modular and efficient datapath that supports the major instruction formats of MIPS, namely R-type, I-type, and J type. Each module—ALU, Control Unit, Register File, Program Counter, Instruction Memory, Data Memory, and Sign- Extension unit is designed using Verilog HDL to ensure hardware synthesizability and portability across FPGA .

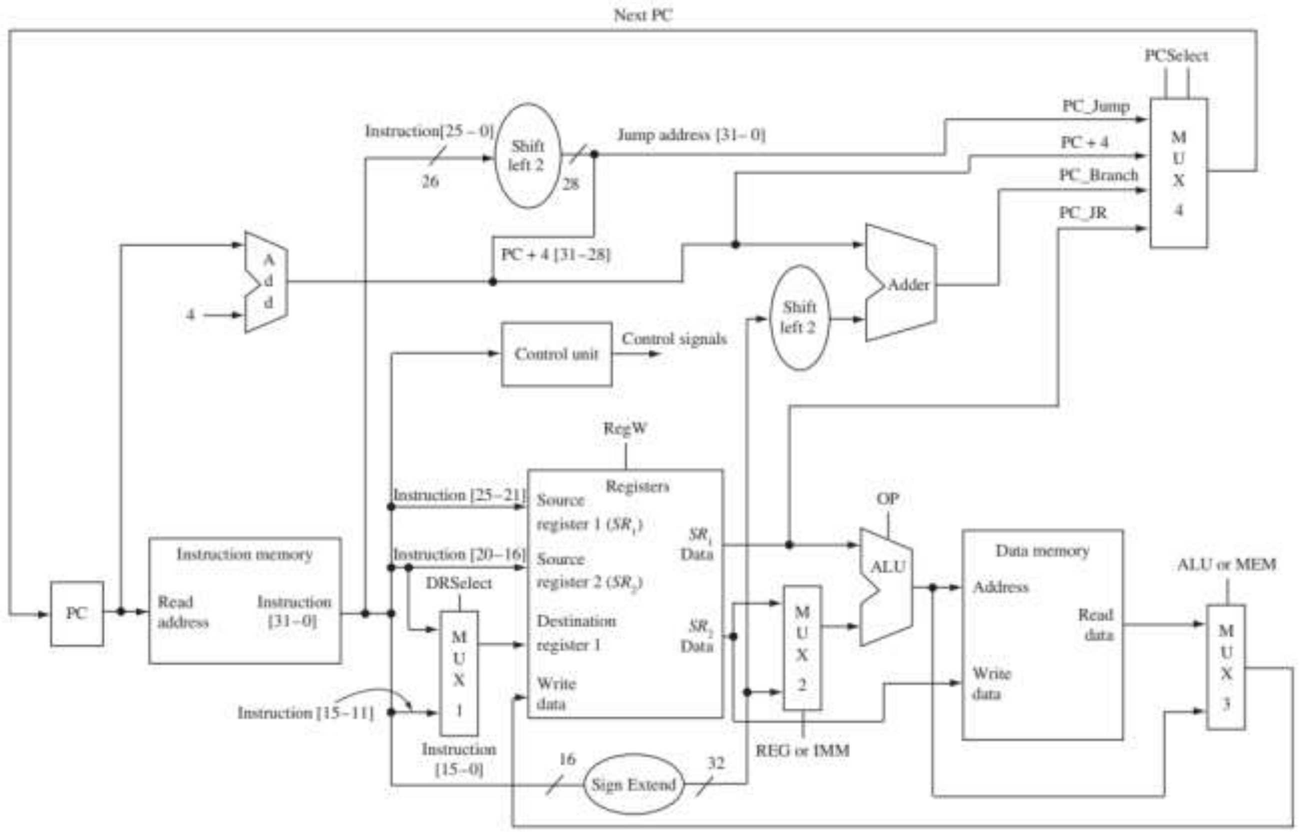


Fig. 1. Proposed architecture -Implementation of MIPS(Prosser)

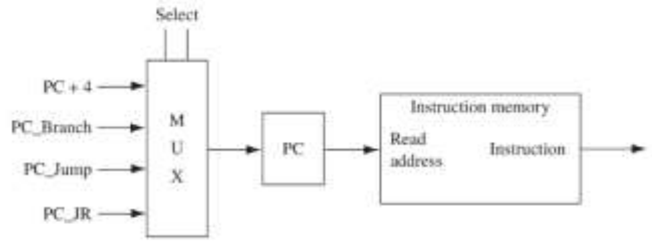
The MIPS architecture breaks down into five main parts: the Program Counter (PC), instruction memory, register file, ALU, data memory, and a bunch of multiplexers that the control unit manages. Here's how it works. The PC kicks things off by sending an address to instruction memory, and in return, it gets a 32-bit instruction. Normally, the PC just moves ahead by four, but when there's a branch, jump, or JR instruction, it needs a different next address. That's when a multiplexer steps in, picking the right address based on control signals. When it's time to decode the instruction, the hardware checks the opcode and function fields to figure out which control signals it needs.

The register file grabs two source operands, RS and RT. If the instruction is an I-type, any immediate values get bumped up to 32 bits. Multiplexers decide if the ALU uses a register value or an immediate, so you get support for all kinds of operations—arithmetic, logic, memory access, and branching. In the execute stage, the ALU gets to work. It handles everything from math to calculating addresses for loads and stores, plus comparisons for branches.

The result either heads straight to write-back or gets used as an address for data memory. For memory instructions, data memory does its thing—either reading or writing, depending on what the instruction calls for. The last step is write-back. Here, a multiplexer chooses between the ALU's output and memory data, and the correct value updates the destination register for R-type or I-type instructions. All in all, this setup keeps things clear and modular, so it's easy to implement in hardware like FPGAs and works well for research and teaching.

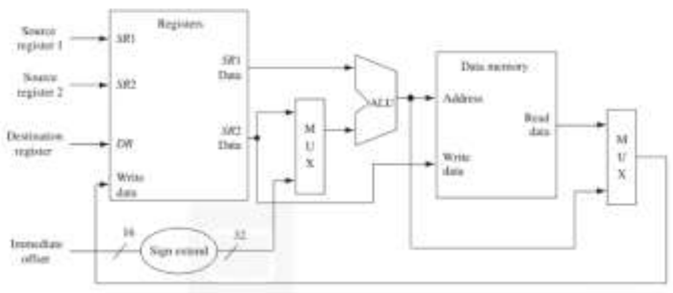
A. Instruction Fetch Unit:

The Instruction Fetch (IF) unit kicks off everything in the MIPS processor. Its job? Grab the next instruction and feed it into the pipeline. If you look at the diagram, you'll see exactly how the Program Counter (PC) moves forward and how the processor pulls instructions from memory. The design relies on a multiplexer to pick the next PC value, depending on where the program needs to go next. Simple, but it keeps things moving.



The multiplexer takes in four different PC update values: PC + 4, PC_Branch, PC_Jump, and PC_JR. Usually, the processor just adds 4 to the PC so it can grab the next instruction in memory. When it hits a branch instruction and the condition is true, it switches over to PC_Branch, which holds the branch target address. For jump instructions, it uses PC_Jump. That one comes straight from the instruction's address field. If there's a register-based jump, like with the JR instruction, the processor grabs PC_JR from a register. The control unit figures out which one to pick by decoding the instruction and sending the right select signals to the multiplexer.

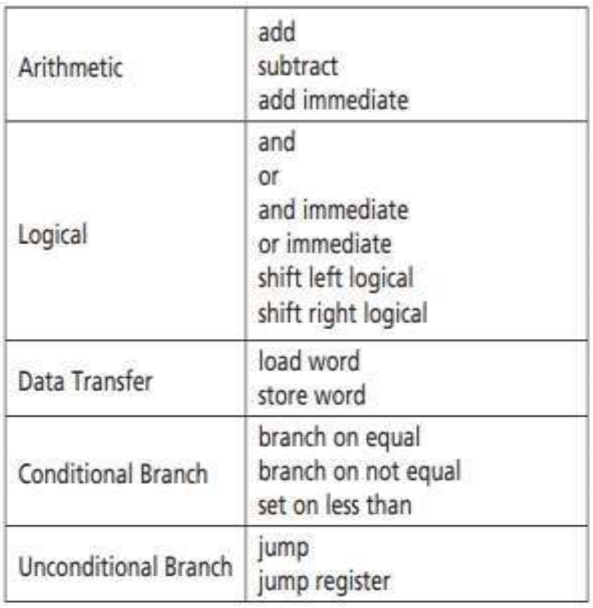
B.Required Data Path for Computation and Memory Instructions:



This architecture features a Register File supplying operands (SR1 Data, SR2 Data), an ALU for execution, and a Data Memory for load/store operations. A multiplexer (MUX) selects the ALU's second operand, which can be the SR2 data or a sign-extended immediate offset, demonstrating key elements of instruction processing. This design effectively separates the fetch, decode, execute, and memory access stages, which is fundamental to modern processor organization.

Tabel I

C.Subset of MIPS instructions implemented



Arithmetic	add subtract add immediate
Logical	and or and immediate or immediate shift left logical shift right logical
Data Transfer	load word store word
Conditional Branch	branch on equal branch on not equal set on less than
Unconditional Branch	jump jump register

Arithmetic: These instructions, such as add, subtract, and add immediate, perform standard mathematical operations on data stored in registers or combine a register value with a constant specified within the instruction (immediate value).

Logical: This group includes operations like and, or, and immediate, or immediate, shift left logical, and shift right logical. They perform bitwise operations and data manipulation necessary for masking, bit extraction, or multiplication/division by powers of two.

Data Transfer: Instructions like load word and store word are essential for moving data between the processor's registers and the main memory. A "word" usually refers to the native size of data handled by the architecture (e.g., 32 or 64 bits).

Conditional Branch: Instructions such as branch on equal and branch on not equal control the flow of program execution by specified condition (e.g., two register values being equal) is met. Set on less than is a related comparison instruction often used *before* a branch to set a register flag based on the comparison result.

Unconditional Branch: Instructions like jump and jump register also alter the program flow but always transfer execution to a new, specified memory address, enabling function calls and loops. Jump register often facilitates returns from function calls by jumping to an address held in a register.

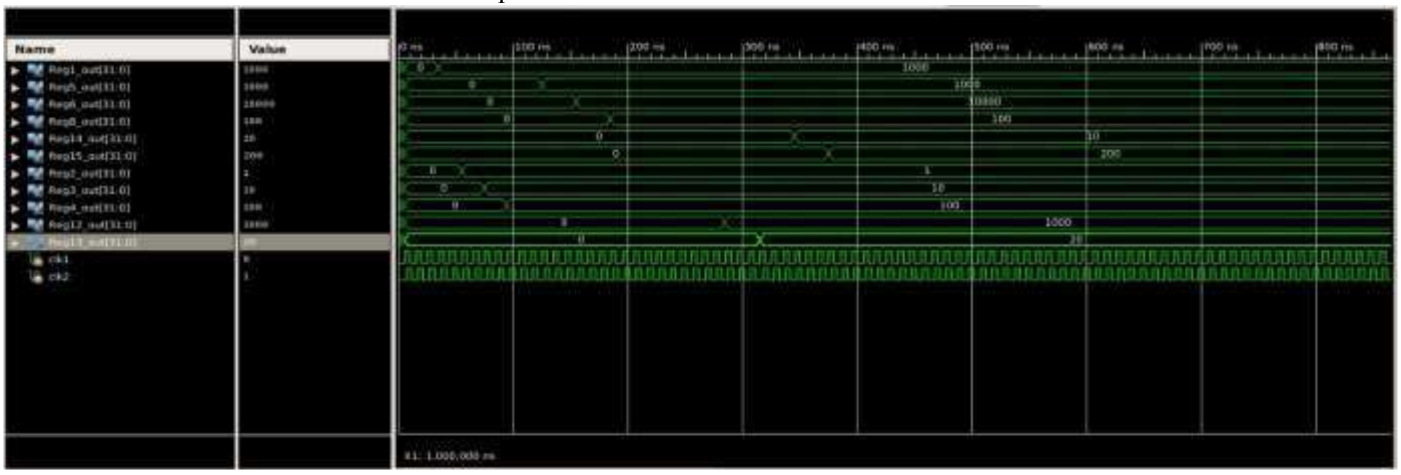
IV. RESULTS AND DISCUSSION

The implemented MIPS processor was successfully synthesized and simulated using Verilog HDL. The RTL view of the top-level design shows the integration of the clock divider, clock sequencer, and pipelined MIPS32 processor blocks. This confirms that the system-level architecture is correctly partitioned into modular units, allowing each submodule—clock generation, instruction sequencing, and execution datapath—to function independently within the complete processor environment.

The functional simulation results demonstrate correct operation of the processor for a range of arithmetic, logical, and memory related instructions. The waveform output shows the values stored in various registers (Reg1, Reg5, Reg6, Reg9, Reg12, Reg13, etc.) updating according to the executed instruction sequence. These updates validate the correct behavior of the instruction fetch, decode, ALU execution, memory access, and write-back paths. The smooth transitions in register values also confirm the correctness of the control unit and the proper handling of immediate and register-based instructions. The generated clock signals (clk1 and clk2) verify that the clock divider and sequencer modules are functioning as intended, producing synchronized multi-phase clocks required for the pipelined operation. The stable timing relationship between these clocks ensures reliable state transitions inside the processor and supports hazard-free execution for the tested instruction set.

- ❖ Simple interest output and power calculation:

RESULT:Simulation results under test mode operation of MIPS



Overall, the results confirm that the proposed MIPS architecture is functionally accurate, stable, and suitable for FPGA implementation. The processor successfully executes basic R-type, I-type, and memory instructions, demonstrating the effectiveness of the design methodology. The modular structure also makes the system easy to extend toward a fully pipelined architecture with hazard handling in future work.

CONCLUSION AND FUTURE SCOPE

The project successfully demonstrated the design and implementation of a 32-bit MIPS processor based on the Prosser architecture. All major modules—PC, instruction memory, register file, ALU, data memory, and control unit— worked correctly in simulation. The results confirmed proper execution of R-type, I-type, and J-type instructions, proving that the architecture is functionally reliable and suitable for FPGA realization. The work clearly shows that a simple, modular MIPS design can be efficiently built using Verilog HDL.

The processor can be extended to a fully pipelined architecture with hazard detection and forwarding to improve performance. Additional features such as exceptions, interrupts, branch prediction, and more instruction support can enhance capability. FPGA implementation with peripherals like UART, timers, and GPIO can turn the design into a complete embedded platform. Further improvements may include low-power optimization, faster clocking, and cache integration for advanced research and applications.

REFERENCE

[1].Rabaey, J. M., Chandrakasan, A., & Borivoje, N. (2003). Digital Integrated Circuits: A Design Perspective 2nd Edition. Upper Saddle River, NJ: Pearson Education, Inc.

[2]. Hennesey, J. L., & Patterson, D. A. (2003). Computer Organization and Design: The hardware/software interface 2nd Edition. San Francisco, CA: Morgan Kaufmann Publishers, Inc.

[3]. D. J. Smith. (2010), „HDL Chip Design‟, International Edition, Doone Publications.

[4]. A. S. Tanenbaum. 2000, „Structured Computer Organization‟, 4th Edition, Prentice- Hall.