UNITECH SELECTED PAPERS

INTERNATIONAL SCIENTIFIC CONFERENCE
20-22 November 2025, GABROVO

MODELING AND SIMULATION OF A SINGLE-CYCLE MIPS MICROARCHITECTURE AS AN EDUCATIONAL TOOL FOR COMPUTER ARCHITECTURE

Petar Minev*, Ilian Varbov, Valentina Kukenska, Matyo Dinev

Technical University of Gabrovo, 4 H. Dimitar Str., Gabrovo, Bulgaria
**Corresponding author: pminev@tugab.bg*

Abstract
This report details the design, modeling, and simulation of a Single-Cycle MIPS Microarchitecture using the VHDL hardware description language. The primary objective of this project is to create a robust and pedagogically effective model suitable for use as an educational tool in introductory Computer Architecture and Organization courses.
The work focuses on implementing a Reduced Instruction Set Computing (RISC) approach, specifically covering a critical subset of the MIPS Instruction Set Architecture (ISA), including R-type, I-type, and basic control flow instructions. The single-cycle design was chosen for its straightforward pipeline-less operation, which facilitates the clear understanding and visualization of the core execution phases: Instruction Fetch, Decode, Execute, Memory Access, and Write Back.
The document provides an in-depth analysis of the main functional blocks—the Program Counter (PC), the Register File, the Arithmetic Logic Unit (ALU), and the Control Unit—and demonstrates their seamless integration within the Datapath. The VHDL implementation is verified through extensive testbenches, generating precise timing diagrams that confirm functional correctness and serve as illustrative examples for students. The resulting model successfully bridges the gap between theoretical knowledge and practical hardware implementation, making complex processor concepts accessible to learners.
Keywords: MIPS Microarchitecture, Single-Cycle, VHDL, Educational Tool, Computer Architectures.

INTRODUCTION

The study of Computer Architecture and Organization is fundamental to understanding how modern computing systems operate [1]. At the core of any digital system lies the processor, a complex component whose inner workings are often challenging for students to grasp solely through abstract theory. This difficulty stems from the need to correlate high-level programming constructs with low-level hardware mechanisms [2].

To bridge this gap between theory and practical application, the development of functional, simplified processor models is paramount. The MIPS (Microprocessor without Interlocked Pipelined Stages) architecture serves as an ideal platform for such educational endeavors. Being a classic example of a Reduced Instruction Set Computing (RISC) architecture, MIPS offers a clean, straightforward instruction set and a clear architectural design, making it highly suitable for pedagogical purposes [2].

The primary objective of this report is to detail the design, modeling, and simulation of a Single-Cycle MIPS Microarchitecture using the VHDL hardware description language. By focusing on a single-cycle implementation, the project aims to create a transparent and verifiable model that clearly demonstrates the five fundamental stages of instruction execution: Instruction Fetch, Decode, Execute, Memory Access, and Write Back.

This document will cover the theoretical foundations of the MIPS Instruction Set

doi: www.doi.org/10.70456/.....

Architecture (ISA), the low-level design of the datapath and control unit, and the practical implementation and verification of the model through VHDL simulation. The goal is to provide a robust and comprehensible educational tool that significantly enhances the learning experience for future computer architects and engineers.

EXPOSITION

I. Theoretical Background and Pedagogical Focus

The MIPS (Microprocessor without Interlocked Pipelined Stages) architecture is widely adopted in academic settings as a prime example of a modern processor design [2]. This section outlines the essential MIPS concepts and justifies the choice of the Single-Cycle model for educational purposes.

1.1. The MIPS Instruction Set Architecture (ISA)

The MIPS processor is based on the Reduced Instruction Set Computing (RISC) philosophy. This approach simplifies the hardware design and instruction set, facilitating faster execution and lower complexity compared to Complex Instruction Set Computing (CISC) architectures [2].

The core principle of RISC is to use a small, uniform set of instructions, typically executed within a single clock cycle, promoting a higher Clock Per Instruction (CPI) rate [2]. The uniform instruction size and format allow for simplified decoding logic, which is crucial for the educational model presented in this report.

MIPS instructions are 32 bits long and are structured into three main formats, determined by the 6-bit OpCode field (bits 31–26) [5]:

R-Type (Register): Used for arithmetic, logical, and shift operations (add, sub, and). It specifies three register operands (rs, rt, rd) and uses the Funct field (bits 5–0) to determine the specific operation.

I-Type (Immediate): Used for instructions that involve a constant (immediate) value, such as memory access (lw, sw) and immediate arithmetic (addi). It includes two register operands (rs, rt) and a 16-bit immediate value.

J-Type (Jump): Used for unconditional jump instructions, providing a 26-bit target address.

The uniformity of these formats is key to simplifying the design of the Control Unit, making the decoding process highly transparent to the student.

The MIPS architecture utilizes 32 general-purpose registers, typically denoted as $r0 through $r31 [6]. Register $r0 is hardwired to the value zero and cannot be overwritten. The register set is essential for the instruction execution phase, providing fast, localized data storage for the ALU operations.

1.2. Microarchitectural Model Selection

The chosen microarchitecture directly impacts the complexity and speed of the processor. For an instructional project, clarity and ease of tracing are prioritized over maximum performance.

We have selected the Single-Cycle MIPS Microarchitecture for this model. In this design, every instruction is guaranteed to complete its execution within a single clock cycle.

Pedagogical Rationale:

- **Simplicity:** The single-cycle design removes the complexity of pipelining (hazards, stalls), allowing students to focus entirely on the flow of data through the datapath and the generation of control signals.
- **Traceability:** The execution of an instruction is straightforward to trace, making it easy to observe which functional blocks are active and how multiplexers are switched based on the instruction's OpCode.

The model is implemented using VHDL (Very High-Speed Integrated Circuit Hardware Description Language) at the

doi: www.doi.org/10.70456/.....

Register Transfer Level (RTL). RTL modeling describes the flow of data between hardware registers and how functional units transform that data. VHDL is chosen because it allows for a clear, structural, and behavioral description of the hardware components, making the design suitable for both simulation and eventual synthesis onto a Field-Programmable Gate Array (FPGA).

II. Single-Cycle MIPS Microarchitecture Design

The successful modeling of the MIPS processor hinges on the design of two primary functional components: the **Datapath** (where data is processed) and the **Control Unit** (which orchestrates the operations). This section details the design of these elements. Fig. 1 presents overall microarchitecture in adherence to the single-cycle principle.

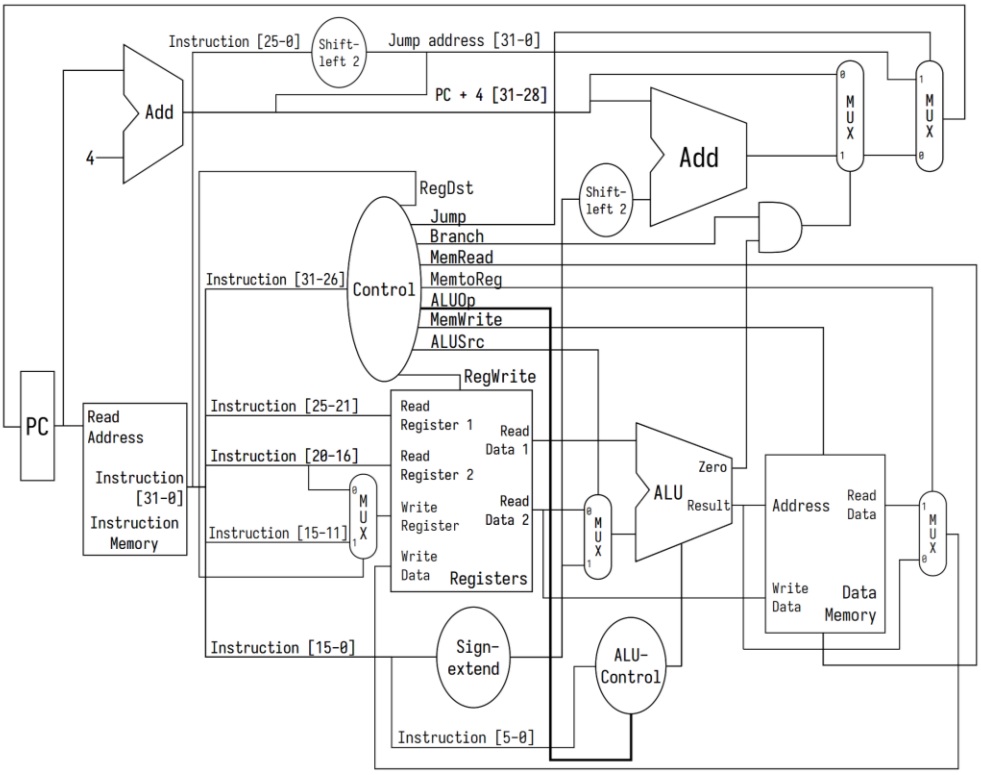

Fig. 1. *Compiled Datapath and required Control Unit*

2.1. The Single-Cycle Datapath Architecture

The datapath represents the collection of functional units - registers, ALUs, memories, and interconnecting buses - required for instruction execution. Since the architecture is single-cycle, the datapath must contain enough hardware to complete every instruction within one clock tick, although only the required units are active for any given instruction.

The instruction execution in the single-cycle model passes through five conceptual phases in sequence:

Instruction Fetch (IF): The instruction is read from the Instruction Memory based on the address stored in the Program Counter (PC). The PC is incremented by 4 for the next sequential instruction.

Instruction Decode (ID): The instruction is decoded, register operands are identified, and the Register File reads the necessary values from the registers specified by the rs and rt fields.

Execute (EX): The Arithmetic Logic Unit (ALU) performs the required operation (arithmetic, logical, or address calculation for memory access/branching).

doi: www.doi.org/10.70456/.....

Memory Access (MEM): If the instruction is a Load or Store, data is read from or written to the Data Memory.

Write Back (WB): The result (from the ALU or Data Memory) is written back to the Register File.

The following components form the datapath, modeled at the RTL using VHDL:

Program Counter (PC) and PC Adder: The PC holds the address of the current instruction. A dedicated Adder calculates the address of the next sequential instruction (PC + 4). Logic is included to handle updates from branch and jump instructions.

Instruction Memory: A component modeled as a large array of registers, responsible for storing and retrieving the 32-bit instruction word based on the address provided by the PC.

Register File: A crucial component implemented with two read ports and one write port. It receives two 5-bit register addresses (rs and rt) and outputs their contents simultaneously. The write operation is controlled by the RegWrite control signal.

Arithmetic Logic Unit (ALU): The ALU performs all computational tasks. It receives two 32-bit inputs and performs a function (e.g., ADD, SUB, AND, OR) determined by the 4-bit ALU Control signal, which is derived from the main Control Unit.

Data Memory: Modeled to support both read and write operations, controlled by the MemRead and MemWrite signals.

2.2. The Control Unit Design

The Control Unit is responsible for generating all the necessary control signals that direct the flow of data through the datapath and specify the operation of the functional units. Its input is primarily the 6-bit OpCode (bits 31–26) of the instruction being executed.

The Control Unit is typically implemented as combinational logic. For each instruction OpCode, the unit must assert a specific set of output signals. Table 1 presents main part of them.

Table 1. *Control signals*

Signal	***Function***
RegDst	Selects the destination register for the Write Back phase (Rt for I-Type, Rd for R-Type).
ALUSrc	Selects the second ALU operand (Read Data 2 from Register File or Sign-Extended Immediate Value).
MemtoReg	Selects the data source for the Register File write (ALU Result or Data Memory output).
RegWrite	Enables writing the result back into the Register File.
MemRead	Enables reading from Data Memory (lw instruction).
MemWrite	Enables writing to Data Memory (sw instruction).
Branch	Enables the Program Counter to branch if the ALU Zero flag is set (for beq).
Jump	Enables the Program Counter to take the unconditional jump address.

The detailed logic mapping between the instruction OpCode/Funct field and these control signals forms the core of the Control Unit's VHDL implementation.

III. VHDL Implementation and Verification

This section describes the practical realization of the Single-Cycle MIPS Microarchitecture using VHDL and details the methodology employed to verify the model's functional correctness through simulation, which is crucial for establishing its reliability as an educational tool.

3.1. Integration and RTL Modeling

The VHDL project utilizes an RTL modeling approach [3] [4] where the processor is composed by processes and signals interconnecting the processes, describing the components detailed in Section II (Control Unit, ALU, Register File, Memories).

Each functional block (e.g., ALU, Register File, PC) is modeled as a separate VHDL process. The signals connecting these components correspond directly to the data and control lines illustrated in the

doi: www.doi.org/10.70456/.....

Datapath Block Diagram.

Memory Implementation: Both the Instruction Memory and Data Memory are implemented as arrays within the VHDL code, often initialized with test instructions and data to ensure a controlled simulation environment.

3.2. Simulation Methodology and Test Programs

Verification is performed using a VHDL Testbench, which serves as the external environment interacting with the processor model.

The Testbench provides the necessary clock and reset signals (clk, rst) and monitors the internal and output signals of the MIPS processor (e.g., PC value, Register File contents, Data Memory access).

A sequence of MIPS instructions is pre-loaded into the Instruction Memory component of the model (fig. 2). The chosen instruction set covers the basic R-type, Load/Store, and Branch operations.

```
imem(4)  <= "00100000";
imem(5)  <= "01010000";
imem(6)  <= "00001001";
imem(7)  <= "00000001"; -- ADD R10, R8, R9
                        -- #R10 = R8 + R9
imem(8)  <= "00100010";
imem(9)  <= "01010000";
imem(10) <= "00001001";
imem(11) <= "00000001"; -- SUB R10, R8, R9
                        -- #R10 = R8 - R9
imem(12) <= "00001010";
imem(13) <= "00000000";
imem(14) <= "00001001";
imem(15) <= "00100001"; -- ADDI R9, R8, 10
                        -- #R9 = R8 + 10
imem(16) <= "00001001";
imem(17) <= "00000000";
imem(18) <= "00101000";
imem(19) <= "10101101"; -- SW R8, R9, 9
                        -- #MEM[R9 + 9] = R8

imem(20) <= "00001000";
imem(21) <= "00000000";
imem(22) <= "00101000";
imem(23) <= "10001101"; -- LW R8, R9, 8
                        -- #R8 = MEM[R9 + 8]
imem(24) <= "11111011";
imem(25) <= "11111111";
imem(26) <= "01001000";
imem(27) <= "00010101"; -- BNE R10, R8, -5
                        -- #if R10 == R8 then
                        -- # PC = PC + 4 - 5
imem(28) <= "00000010";
imem(29) <= "00000000";
imem(30) <= "00000000";
```

Fig. 2. *Preloaded test program in the instruction memory*

To demonstrate correct operation for educational purposes, test cases include programs designed to:

- Arithmetic: Verify R-type instruction ADD and I-type instruction ADDI by checking if the result is correctly written back to the destination register (fig. 3 and fig. 4).
- Data Transfer: Verify Load (LW) and Store (SW) operations by confirming that the ALU calculates the correct memory address and that the data is transferred between the Data Memory and the Register File (fig. 5 and fig. 6).
- Control Flow: Verify Branch Not Equal (BNE) by ensuring the Program Counter updates correctly to the branch target address only when the ALU's zero flag is not set (fig. 7).

3.3. Simulation Results and Educational Demonstration

The simulation results, typically presented as waveform diagrams, provide visual proof of the processor's functionality. For an educational model, these traces are vital for correlating the assembly code with hardware behavior.

The waveform in figure 3 clearly shows the correct assertion of control signals for an R-Type instruction. Specifically, the Control Unit asserts the *RegDst* and *RegWrite* signals, directing the result to the destination register specified by the *R10* field. The values read from the source registers (*R8* and *R9*) are routed to the ALU inputs (*A* and *B*), where the requested operation (addition) is performed. The result from the ALU (*aluRes*) is successfully written to the Register File during the rising edge of the next clock cycle, confirming the model's functional integrity for fundamental arithmetic operations.

doi: www.doi.org/10.70456/.....

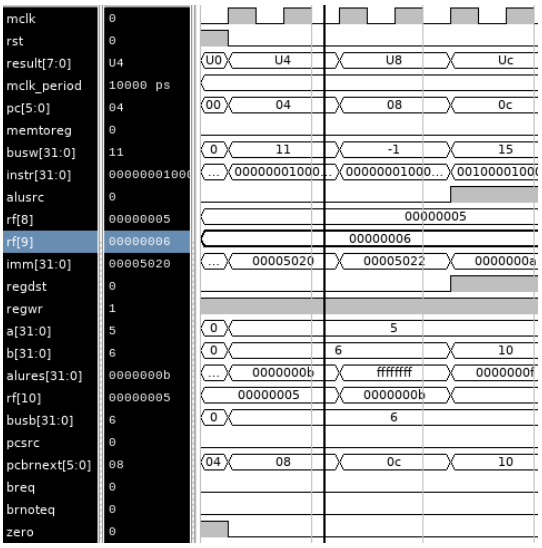

Fig. 3. *Simulation result of the model showing execution of the instruction ADD R10, R8, R9*

Figure 4 shows the tracing of the execution of the I-type arithmetic instruction *ADDI R9, R8, 10*. The ADDI instruction demonstrates the use of immediate values and the *ALUSrc* control signal. The simulation shows that the Control Unit asserts *ALUSrc = 1*. This directs the sign-extended 32-bit immediate value to the second input of the ALU, bypassing the second read port of the Register File. The ALU result is routed to the register specified by the Rt field, as is standard for I-Type arithmetic. The value read from the source register (*R8*) is correctly added to the sign-extended immediate value (*10*) by the ALU, and the resulting sum is written to the destination register (*R9*) upon the next clock edge. This verifies the correct implementation of the sign-extension unit and the routing logic.

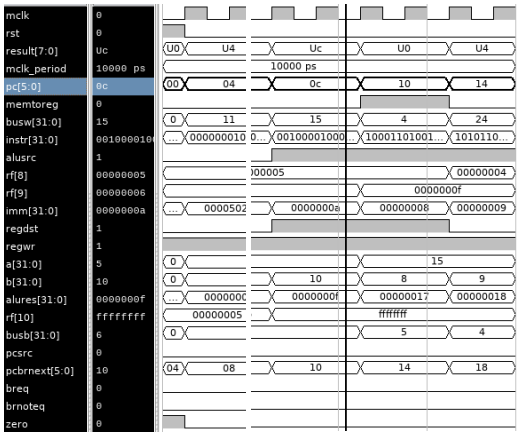

Fig. 4. *Simulation result of the model showing execution of the instruction ADDI R9, R8, 10*

Figure 5 presents the tracing of the execution of the store word instruction *SW R8, 9(R9)*. The simulation waveform demonstrates the memory write path and confirms the following: The Control Unit asserts *MemWrite = '1'*, indicating a write operation to the Data Memory. Critically, *RegWrite* is asserted to '0' since no data is being written back to the Register File; The ALU, instructed by the *aluCtrl* signal, calculates the effective memory address by summing the contents of the base register *(R9)* and the sign-extended 16-bit immediate offset *(9)*; The value to be stored, read from the Register File's second read port (Register *R8*), is routed directly to the input of the Data Memory; Upon the rising edge of the clock, the Data Memory writes the content of *R8* to the calculated effective address. This verifies the correct functionality of the address calculation logic and the memory write mechanism.

Figure 6 demonstrates the tracing of the execution of the load word instruction *LW R8, 8(R9)*. The LW instruction is the counterpart (or inverse operation) to SW and demonstrates the memory read path followed by a write-back to the register file. The simulation waveform confirms the following: The Control Unit asserts both *MemRead = '1'* (to read from Data Memory) and *RegWrite = '1'* (to update the Register File). Since the data comes from memory, *MemtoReg* must be set to select the Data Memory output; Like SW, the ALU calculates the effective memory address by summing the contents of the base register *(R9)* and the sign-extended 16-bit immediate offset (*8*); The calculated address is routed to the Data Memory's address input. The 32-bit data retrieved from the Data Memory is then routed back toward the Register File; The retrieved data is written back to the destination register (*R8*) upon the rising edge of the next clock cycle. This confirms the correct implementation of the load operation and the memory-to-register data path.

doi: www.doi.org/10.70456/.....

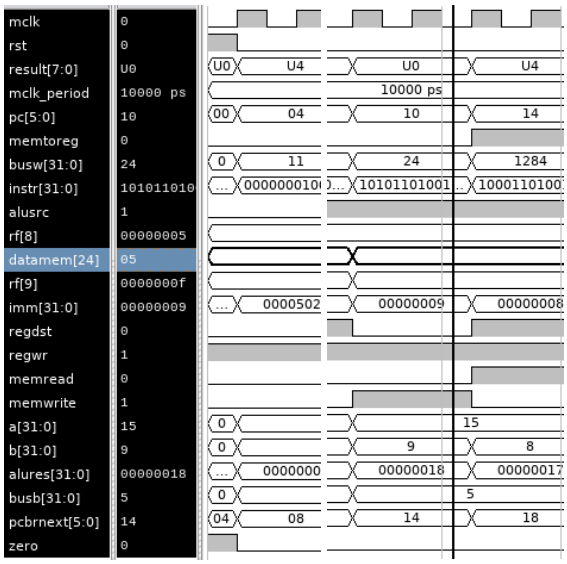

Fig. 5. *Simulation result of the model showing execution of the instruction SW R8, 9(R9)*

The tracing of the control flow instruction Branch Not Equal is presented in figure 7. This demonstration is key to understanding how the processor handles conditional control flow when the condition is not met. For the instruction *BNE R10, R8, -5*, the processor performs the following: The ALU performs a subtraction between the contents of source registers *R10* and *R8*. The *Zero* Flag is set to '*0*' if the values are not equal, satisfying the branch condition; The Control Unit asserts the signal *BrNotEq,* which is based on the BNE OpCode, indicating a conditional branch is active. The target address is calculated by adding the sign-extended, shifted offset (*-5 * 4 = -20* bytes) to the *PC + 4* value; The dedicated branch logic checks for the condition: (*BrNotEq AND NOT Zero*). If this condition is *TRUE* (meaning the registers are not equal), the logic selects the calculated branch target address; The simulation trace clearly shows the Program Counter (*PC*) being updated with the calculated target address (*PC + 4 - 20*), resulting in the next instruction being executed from a previous memory location, demonstrating a successful backward jump.

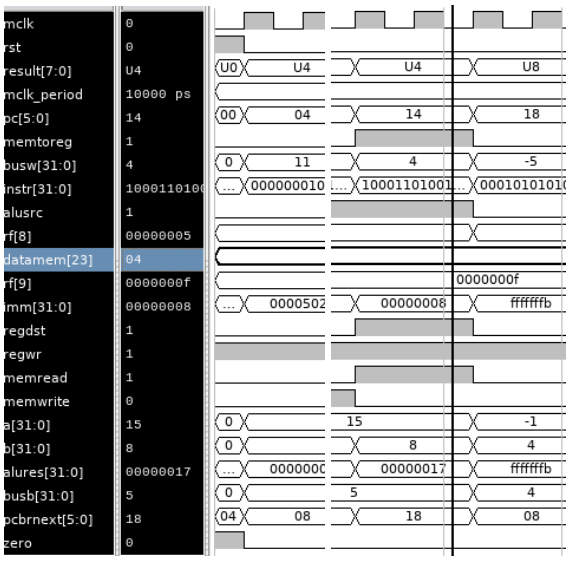

Fig. 6. *Simulation result of the model showing execution of the instruction LW R8, 10(R9)*

The successful simulation and verification of these core instructions confirm the functional correctness of the Single-Cycle MIPS model, validating its reliability as a hands-on tool for teaching Computer Architecture concepts.

The complete VHDL source code, including the Testbench necessary for simulation, is publicly available to support collaborative education and reproducibility. The project repository can be accessed at: *https://github.com/pminev-debug/mipsEdu*.

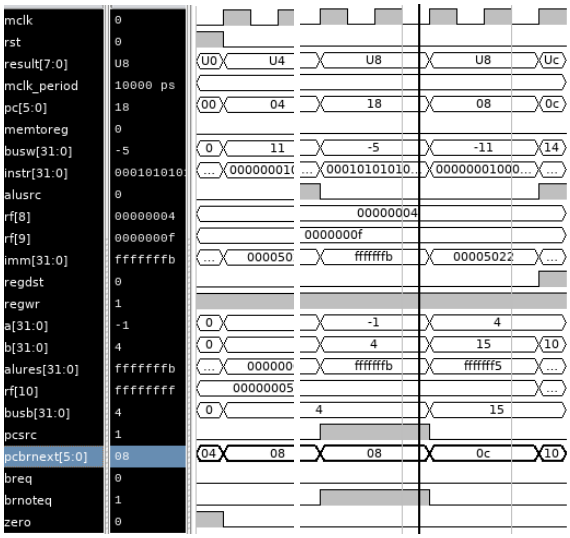

Fig. 7. *Simulation result of the model showing execution of the instruction BNE R10, R8, -5*

doi: www.doi.org/10.70456/.....

CONCLUSION

This report successfully presented the design, VHDL modeling, and verification of a Single-Cycle MIPS Microarchitecture specifically adapted as an educational tool for computer architecture courses. The primary objective of creating a simple, transparent, and functionally correct model was achieved.

The key accomplishments include:

- Selection and Justification: The choice of the MIPS ISA and the Single-Cycle model was rigorously justified on pedagogical grounds, prioritizing ease of understanding and traceability over performance complexity.
- RTL Design: The processor's Datapath and Control Unit were designed and modeled at the Register Transfer Level (RTL) using VHDL, ensuring that the model is suitable for both simulation and hardware synthesis.
- Verification: Comprehensive testing using a VHDL Testbench demonstrated the functional correctness of the model across core instruction types (R-type, Load/Store, and Branching). The simulation waveforms provide clear visual proof of data flow and control signal assertion during instruction execution, making them invaluable for instructional demonstrations.

The result is a reliable and modular hardware description that effectively bridges the gap between theoretical instruction set architecture and practical digital hardware implementation.

While the Single-Cycle model serves its purpose as an introductory tool, future work can expand the project to introduce concepts related to performance and optimization:

- Transition to Pipelined Architecture: The most critical extension would be to transform the single-cycle model into a Five-Stage Pipelined MIPS Processor. This would introduce the challenges of data hazards, control hazards, and require the implementation of solutions like forwarding units and branch prediction, representing the next level of complexity in computer architecture education.
- Performance Analysis: Implementing the design on a physical FPGA device would allow for real-world measurements of clock speed and latency, providing a practical dimension to the theoretical performance analysis.
- Graphical User Interface (GUI) Integration: Developing a front-end interface to visualize the state of the Register File, Data Memory, and the active path lights during execution would greatly enhance the model's utility as a teaching aid.

The authors are grateful to the Ministry of Education and Science of the Republic of Bulgaria for the support under contract NIP2025-15 in Technical University of Gabrovo.

REFERENCE

[1] Patterson D., J. Hennessy, Computer Organization and Design MIPS Edition: The Hardware/Software Interface, 6th Edition, 2020, Morgan Kaufmann.
[2] Patterson D., J. Hennessy, Computer Architecture, Fifth Edition: A Quantitative Approach, 2011, Morgan Kaufmann.
[3] Haskell R., D. Hanna, Introduction to Digital Design Using Digilent FPGA Boards, 2009, Rochester Hills.
[4] Chu P., FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version 1st Edition, 2008, Wiley-Interscience.
[5] Negru M., F. Oniga, S. Nedevschi, COMPUTER ARCHITECTURE Laboratory Guide, 2015, Cluj-Napoca.
[6] Wave Computing, MIPS® Architecture for Programmers Volume II-A: The MIPS32® Instruction Set Manual, 2016.

doi: www.doi.org/10.70456/.....