Design and Implementation of 32-bit RISC-V Processor using Verilog

Manjusha Rao P
Dept. of ECE, NMAM Institute of Technology, Nitte (Deemed to be University), Nitte, Karkala Udupi, Karnataka, India
358manjusha@gmail.com

Prabha Niranjan
Dept. of ECE, NMAM Institute of technology, Nitte (Deemed to be University), Nitte, Karkala, Udupi, Karnataka, India
prabhan@nitte.edu.in

Dileep Kumar M J
Dept. of ECE, NMAM Institute of technology, Nitte (Deemed to be University), Nitte, Karkala, Udupi, Karnataka, India
dileepmj@nitte.edu.in

2024 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER) | 979-8-3503-5059-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/DISCOVER62353.2024.10750638

Abstract— The design and implementation of a 32-bit single-cycle RISC-V processor in Verilog is a sophisticated and elaborate process that aims to create a functioning processor architecture that adheres to the RISC-V instruction set. To execute instructions in a single clock cycle, this research work requires the synthesis of components such as the program counter, register file, arithmetic logic unit (ALU), and memory modules. The Verilog-based implementation includes RISC-V instructions such as arithmetic, memory access, and control flow instructions. The design prioritizes simplicity and clarity, laying the groundwork for educational study and the eventual development of more advanced processing functionality. This emphasizes importance in understanding processor architecture and Verilog-based processor design.

Keywords— RISC-V, Instruction Set Architecture, RV32IM, Verilog, Simulation

I. INTRODUCTION

Over recent decades, the field of computer hardware has experienced rapid growth, particularly in relation to x86 and ARM CPUs, which have grown in popularity integral to the international semiconductor market. Central processors now pervade various domains from the cores of CPU to the controllers of embedded chips. Rooted in the concept of the tuning machine, a processor functions as a digital circuit executing a single operation specified by an instruction. By combining numerous simple instructions, processors can execute diverse algorithms. Processors are broadly categorized into two types: General-purpose processors and dedicated processors. Single-purpose processors, often referred to as microprocessors, offer versatility by supporting a wide array of applications through software programming. Examples include the central processing units (CPUs) found in mobile phones and desktop computers. On the other hand, single-purpose processors typically excel in speed, power efficiency, and area usage but are tailored to specific applications. Examples include graphic processing units (GPUs) and artificial intelligence (AI) accelerators. Instructions ar fetched by the microprocessor from external memory. Subsequently, the control unit interprets these instructions and orchestrates the routing of data within the data-path, directing operands to the ALU and selecting the appropriate function within the ALU to perform a specific operation. A standard microprocessor comprises three principal parts: the control unit, the arithmetic logic unit (ALU), and the data In Harvard architectures, data and instructions are kept in separate memory units, where distinct pathways exist for fetching instructions and data. Conversely, in the Von Neumann Architecture, both instructions and data share one storage unit.

This research aims to delve into the intricacies of modern CPU core architectures to enhance performance and streamline the compilation process, bridging the gap between software and hardware. This is achieved by implementing single cycle RISC-V processor using Verilog and simulating the functionality of the design.

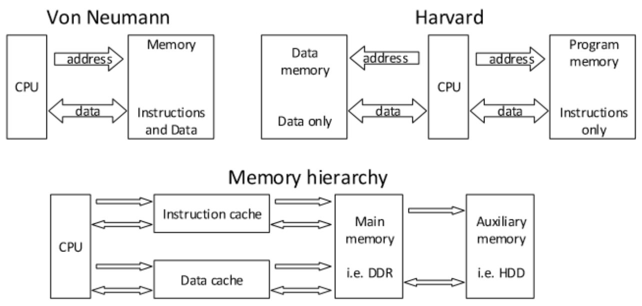


Fig. 1: Von Neumann Architecture, Harvard Architecture and Memory Hierarchy

II. LITERATURE REVIEW

This section provides background information on both CISC and RISC architectures, highlighting the advantages of RISC architecture, particularly in real-time applications. Additionally, it introduces the RISC-V architecture, a specific type of RISC architecture chosen for this research work. This section discusses the RISC-V specification, focusing on the RV32IM instruction set, which forms the basis of the research work. Furthermore, the section previews the related work section, where additional literature on RISC-V will be explored. This will include discussions on other researchers RISC-V products and the importance of RISC-V ISA enhancements [1].

RISC-V Instruction Set Architecture (ISA) has become a popular option for System-on-Chip (SoC) and Internet-of-things (IoT) technologies. Because of its open-source nature and strong safety measures, it may compete with the popular ARM technology. This work covers the design of a current hardware design technique-based, compact, open-source implementation of a RISC-V CPU. Thorough testing is done throughout, with the implementation intended for Field Programmable Gate Arrays (FPGAs) deployment [2].

The primary objective of this endeavor is to develop a RISC-V micro-processor that is accessible to beginners and sufficiently portable to be deployed on modest-scale FPGAs. While existing open-source RISC-V implementations exist, they often lack the intuitive clarity necessary for beginners to grasp. Consequently, this paper prioritizes simplicity by minimizing the utilization of unnecessary protocols and parts typically found Architectural standards and hardware elements, focusing solely on essential elements for a barebones implementation [3].

This work uses Verilog to design and implement a 16-bit RISC processor. Given the intricate computations and resource-intensive nature of IC chip design, employing an HDL offers a resource-efficient and time-saving approach to implementation. Drawing inspiration from the widely adopted MIPS architecture, the processor's design strategies borrow elements from MIPS in some circumstances. The resultant processor has a carefully planned and emulated 16-bit arithmetic and logical instruction set [4].

This simple instruction set provides valuable insights into the hardware requirements necessary for executing instructions effectively. Key components such as the Arithmetic Logic Unit (ALU), RAM, control unit, program counter, register file, and instruction register are seamlessly integrated within the proposed processor design. Through this endeavour, the paper aims to showcase the feasibility and functionality of a 16-bit RISC Processor implemented using VHDL [5].

This paper describes a design strategy specifically for implementing processors that follow the RISC-V ISA. The paper describes the development of three different processors with different designs and functionalities as an example of applying this methodology. This approach simplifies the RISC-V processor design process and allows for its realisation in a variety of architectural setups and feature sets [6].

This implementation of an Out-of-Order RISC-V core design is presented in this study. With the increasing popularity of open-standard computer architectures and related technologies, custom hardware development becomes a more attractive alternative to off-the-shelf components. The open Instruction Set Architecture (ISA) known as RISC-V has been quite popular in the last few years, finding use in anything from high-performance computing to low-cost microcontrollers. In this study, we offer a specially designed core that can run the DCT algorithm (Discrete Cosine Transform) [7].

This functionality opens up potential applications in image and video processing. By leveraging a custom FPGA RISC-V processor, we demonstrate the ability to efficiently execute the DCT algorithm, paving the way for enhanced capabilities in image and video processing tasks [8].

In response to the limitations of traditional CPU platforms in fully exploiting the parallelism inherent in Convolutional Neural Networks (CNNs), this paper proposes the adoption of the RISC-V architecture for experimental design. Based on the RISC-V architecture, a new convolutional neural network processor is designed. aiming to leverage CNN parallelism and enhance flexibility. The designed CNN processor utilizes a classic five-stage pipeline structure and incorporates instruction buffer memory and data buffer memory. Peripheral components such as FLASH, SRAM, and SDRAM are integrated to support various functionalities [9].

Additionally, custom instructions are introduced to optimize CNN operations. Given the frequent occurrence of convolution operations in CNNs, specialized instructions including vector store, vector load, vector addition, and convolution operation instructions are devised to accelerate convolution processes. By harnessing the RISC-V architecture, the designed CNN processor effectively exploits CNN parallelism while maintaining flexibility to execute custom instructions. This approach promises to enhance the performance and efficiency of CNN-based applications [10].

The utilization of a in the development of hardware embedded systems, pipeline design based on Reduced Instruction Set Computers (RISC) is highly crucial. The internal architecture of a 16-bit embedded hardware RISC processor is presented in this research, constructed on the Harvard data path-based architectural paradigm. There are five different steps in the CPU pipeline: fetch, decode, execute, memory, and write back. The first step is to examine the data flow path in the RISC processor, which is made up of two basic categories of digital devices: sequential state machines and combinational logic gates. Next, it is addressed how to synchronise read and write activities with the control signals produced by the control unit [11].

The VHDL programming language is used to model all of the hardware components in the engineered system. The Xilinx ISE Design Suite 14.7 tool is used to simulate the pipeline processor. This tool allows for the coupling of layers between the pipelined stages and provides a schematic view of the designed modules. utilising the XC3S200 chip, the CPU is implemented on a Spartan-3 FPGA. In conclusion, the paper details the design and simulation of a 16-bit MIPS RISC pipeline processor, which is implemented on a Spartan-3 FPGA. VHDL is used for modelling, while the Xilinx ISE Design Suite 14.7 tool is used for simulation [12].

RISC instruction sets typically feature a smaller number of instructions compared to CISC architectures. These instructions are designed to be simple and uniform, with consistent lengths and fields. This uniformity enables efficient instruction pipelining, allowing for the initiation of a new instruction every clock cycle, although individual instructions may still require multiple cycles to execute [13].

III. RISC-V AND ITS KEY FEATURES

The processor in this research work utilizes the RISC-V Instruction Set Architecture (ISA), which belongs to the family of RISC architectures. Introduced in 2011, RISC-V stands out as a free and open instruction set, offering high flexibility and cost-effectiveness. This characteristic makes it particularly advantageous for developing specialized microprocessors tailored to specific applications, thanks to its adaptability and affordability. Within the RISC-V framework, numerous extension instructions are available, further enhancing its versatility and applicability to diverse computing tasks. An illustration of RISC-V's versatility lies in its standard extensions, such as the "A" extension facilitating atomic memory operations, the "M" extension tailored for integer multiplication and division, and the "F" extension dedicated to floating-point computations.

These extensions empower users to tailor the instruction set according to their specific requirements, enhancing its utility across a broad spectrum of applications, including portable and wearable devices, as well as aerospace equipment RISC-V's open-source nature contributes significantly to its growing popularity, particularly in commercial contexts, where its appeal is pronounced. Nevertheless, a notable drawback currently affecting RISC-

V is its limited software compatibility. This challenge stems from the fact that most existing software applications are designed for architectures like ARM or x86, thus necessitating efforts to improve compatibility for broader adoption.

The key features of RISC V architecture include: RISC V architecture is designed to be modular, allowing for easy addition or removal of optional features based on specific application requirements. RISC V architecture is designed to be portable across different hardware platforms, enabling easy migration of software between systems. RISC V architecture follows a standardized instruction set, ensuring compatibility and interoperability between different RISC V-based processors and systems. RISC V architecture aims to provide a simple and elegant instruction set, making it easier to understand, implement, and optimize. By adopting the RISC V architecture, designers can benefit from improved performance, reduced power consumption, scalability, and customization options.

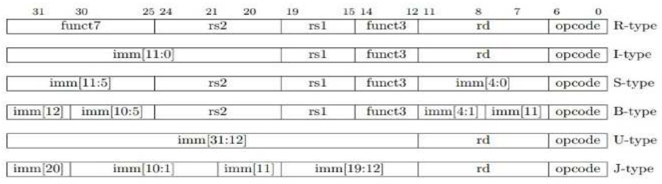

31 30 25 24 21 20 19 15 14 12 11 8 7 6 0
funct7 | rs2 | rs1 | funct3 | rd | opcode | R-type
imm[11:0] | rs1 | funct3 | rd | opcode | I-type
imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode | S-type
imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode | B-type
imm[31:12] | rd | opcode | U-type
imm[20] | imm[10:1] | imm[11] | imm[19:12] | rd | opcode | J-type

Fig. 2: RISC-V based instruction sets

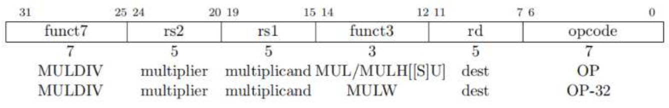

31 25 24 20 19 15 14 12 11 7 6 0
funct7 | rs2 | rs1 | funct3 | rd | opcode
7 5 5 3 5 7
MULDIV multiplier multiplicand MUL/MULH[[S][U] dest OP
MULDIV multiplier multiplicand MULW dest OP-32

Fig. 3: Multiplication Operations

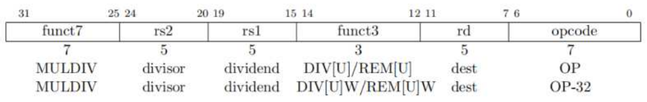

31 25 24 20 19 15 14 12 11 7 6 0
funct7 | rs2 | rs1 | funct3 | rd | opcode
7 5 5 3 5 7
MULDIV divisor dividend DIV[U]/REM[U] dest OP
MULDIV divisor dividend DIV[U]W/REM[U]W dest OP-32

Fig. 4: Division Operations

The "M" standard extension within the RISC-V architecture is specifically designated for integer multiplication and division operations. In these operations, two integer registers hold the values involved. For multiplication, the format of multiplication instructions, as depicted in Fig. 3, involves multiplying the contents of two registers, rs1 and rs2, both of which are XLEN-bit in size. The result is stored in the lower XLEN bits.

Similarly, division instructions follow a format illustrated in Fig. 4. Normal division involves dividing the XLEN-bit value stored in rs1 by the XLEN-bit value stored in rs2, with rounding towards zero. Additionally, the REM instruction is utilized to obtain the remainder resulting from the division operation.

The RISC-V architecture encompasses four base Instruction Set Architectures (ISAs): RV32I, RV64I, RV128I, and RV32E. Each base ISA features its own specifications regarding integer register width, address space size, and the number of integer registers. While RV32I and RV64I offer 32-bit and 64-bit address spaces respectively, RV32E stands out as a variant of RV32I with half the number of integer registers, initially tailored for small-scale microcontrollers.

Across all base ISAs, signed number values are represented using the Two's complement method. In the RV32I ISA, six instruction formats are employed: R-type, I-type, S-type, B-type, U-type, and J-type, each fixed at a length of 32 bits. Moreover, these instructions must be aligned on a 4-byte boundary within memory. Fig. 2.2 illustrates all the formats of RISC-V base instruction types.

To streamline the decoding process, the positions of source registers (rs1 and rs2) and destination registers (rd) remain consistent across all instruction formats. Additionally, immediate numbers undergo sign extension, with bit 31 of the instruction consistently serving as the sign bit for all immediate values.

IV. ARCHITECTURE OF RISC-V PROCESSOR

Within this System-on-Chip (SoC) design, a single Processing core RISC-V is present. In this context, a component qualifies as a core if it possesses an autonomous instruction fetch unit. This RISC-V core operates by continuously executing instructions stored in memory. Additionally, it performs data loading or storing operations between the memory and peripheral devices based on various instructions.

The RISC-V core incorporated in this design adopts a three-stage pipeline architecture, a configuration known for enhancing processor efficiency. These three stages operate as follows:

Instruction Fetch Stage: This initial stage focuses on retrieving instructions from memory. Instruction Decode and Data Retrieval Stage: In this phase, instructions are decoded, and relevant information is read from memory, whether from the register bank or RAM. Stages of Computation and Data Write-Back: The final stage involves executing computations based on the decoded instructions and subsequently writing back the resulting data. Importantly, each stage of the pipeline functions independently, ensuring smooth and efficient processing of instructions.

This block diagram represents a simplified RISC-V single-cycle processor, detailing the various components involved in executing an instruction and their interconnections. Each component in the diagram plays a crucial role in fetching, decoding, executing, and writing back instructions.

The Program Counter (PC) is the starting point, holding the address of the next instruction to be executed. The PC sends this address to the Instruction Memory, which retrieves the instruction stored at that location. This instruction is then broken down into its constituent fields, such as the opcode, source and destination register addresses, and immediate values. These fields are sent to various parts of the processor for further processing.

The Control Unit receives the opcode and generates control signals that dictate the operation of the processor. These signals include Branch, MemRead, MemtoReg, ALUOp, MemWrite, ALUSrc, and RegWrite. Each signal plays a specific role, such as enabling memory read/write operations, selecting ALU operations, or determining the source of data for the ALU. The Register File is a critical component that contains all the general-purpose registers. It receives the addresses of the source registers (rs1 and rs2) and the destination register (rd) from the instruction. The data from the source registers is read and sent to the ALU for computation. If the instruction requires an immediate value, the Immediate Generator (Imm Gen) extends the immediate field from the instruction to a 64-bit value suitable for processing.

The ALU (Arithmetic Logic Unit) performs arithmetic and logical operations based on the control signals from the Control Unit. It takes inputs from the register file or the Immediate Generator and produces a result. The result is either stored back in the register file, sent to memory, or used

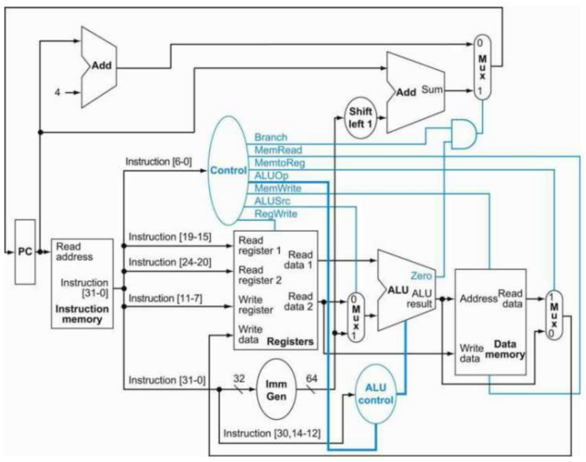


Fig. 5: RISC-V Core Architecture

to update the PC for branch instructions. The ALU also generates a Zero flag, which is used to determine the outcome of branch instructions.

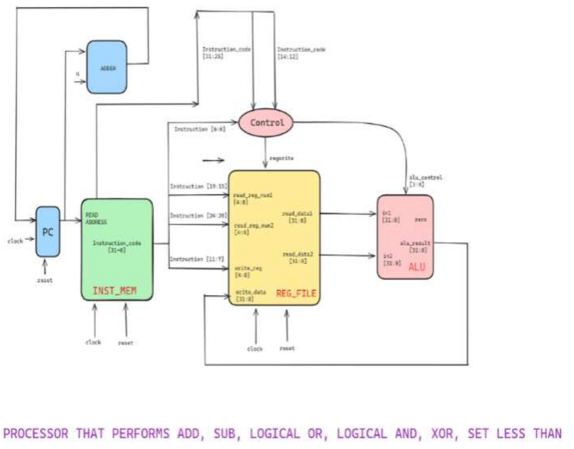
For memory operations, the Data Memory component is involved. When a load or store instruction is executed, the Data Memory reads from or writes data to the specified address. The address is calculated by the ALU, and the data to be written or read is routed through multiplexers controlled by the MemtoReg signal.

The Adders in the diagram have specific roles. One adder increments the PC by 4 to point to the next instruction sequentially. Another adder is used for calculating branch target addresses. The output of this adder, along with the Zero flag from the ALU, determines if a branch is taken. If so, the multiplexer selects the branch target address as the new PC value; otherwise, it selects the incremented PC.

Finally, multiplexers (MUX) are used extensively throughout the processor to select between different inputs based on control signals. For example, one multiplexer selects between the ALU result and the data read from memory to be written back to the register file. Another multiplexer selects between the immediate value and the second register value as the second input to the ALU.

V. DESIGN OF RISC-V PROCESSOR

The Fig. 6 illustrates a simplified RISC-V processor designed to perform various arithmetic and logical operations, including ADD, SUB, OR, AND, XOR, and SET LESS THAN. Central to this design is the Program Counter (PC), which holds the address of the current instruction and increments by 4 via an adder to move to the next instruction. Instructions are fetched from the Instruction Memory (INST_MEM) using the address from the PC and are then sent to both the Control Unit and Register File. The Control Unit decodes the instruction to generate necessary control signals that govern the processor's operations, including selecting ALU operations and determining register writes. The Register File consists of a set of registers used for temporary data storage, providing operands to the ALU and storing results if write-back is enabled by the control signals. The Arithmetic Logic Unit (ALU) executes the specified operations using operands from the Register File, producing a result and setting a zero flag if the result is zero. This flag can be utilized for branching operations, though branching is not depicted in this diagram.



ADDER
Control
regwrite
PC
clock
reset
INST_MEM
REG_FILE
ALU
alu_control
zero
alu_result
RISCv PROCESSOR THAT PERFORMS ADD, SUB, LOGICAL OR, LOGICAL AND, XOR, SET LESS THAN

Fig. 6: Implemented RISC-V processor

The adder is employed to increment the PC by 4, ensuring the processor proceeds to the next instruction. In essence, this RISC-V processor efficiently handles basic arithmetic and logical operations through a streamlined and well-coordinated flow of data and control signals, embodying the simplicity and power of RISC architecture.

Block diagram in Fig 7. illustrates the execution of a RISC-V `add` instruction, specifically `add t1, s0, s1`. This instruction adds the contents of registers `s0` and `s1` and stores the result in register `t1`. The diagram highlights the key components and data flow within a simplified processor architecture.

Starting with the Program Counter (PC), this component holds the address of the current instruction to be executed. The PC sends this address to the Instruction Memory (INST_MEM), which retrieves the instruction stored at that location. For our instruction `add t1, s0, s1`, the binary representation is fetched and passed to the Control Unit. The Control Unit decodes the instruction and generates the necessary control signals for the operation.

The fetched instruction is broken down into its fields: opcode, destination register (rd), function codes (funct3 and funct7), and source registers (rs1 and rs2). The Control Unit uses this information to determine the operation type, which in this case is addition, as indicated by the funct3 and funct7 fields combined with the opcode.

Next, the Register File (REG_FILE) is accessed to read the values of the source registers `s0` and `s1`. These registers are identified by their binary addresses in the instruction. In this example, the value of register `s0` (register 8) is 8, and the value of register `s1` (register 9) is 9. These values are then fed into the Arithmetic Logic Unit

(ALU), which is responsible for performing the addition operation.

The ALU performs the addition operation based on the control signals provided by the Control Unit. It takes the input values from `s0` and `s1`, adds them, and produces the result. In this case, 8 + 9 equals 17. This result is then ready to be written back into the destination register `t1` (register 6) in the Register File. The Register File updates register `t1` with the new value, 17. Finally, the PC is updated to point to the next instruction. An Adder component increments the current PC value by 4 (the size of one instruction in bytes) to get the address of the next instruction. This updated PC value is then fed back into the PC to continue the execution cycle with the next instruction in the sequence.

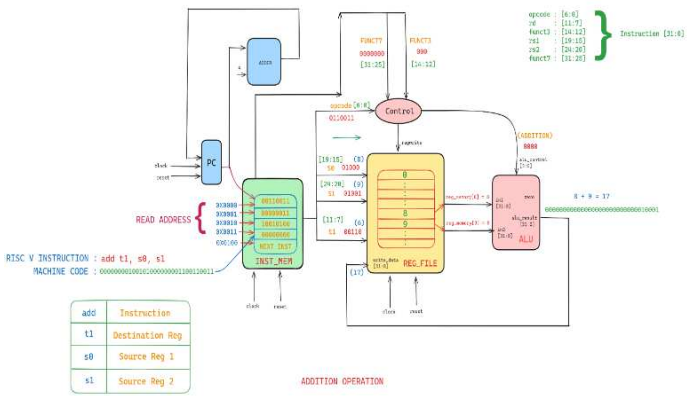


Fig. 7: RISC-V Processor performing Addition operation

VI. RESULTS AND DISCUSSION

The Fig. 8 represents a detailed timing diagram of a RISC-V processor executing various operations, including ADD, SUB, OR, AND, XOR, and SET LESS THAN (SLT). The clock signal oscillates regularly, providing a consistent timing reference. Initially, the reset signal is active, setting the Program Counter (PC) to 0 at time 0. Once reset becomes inactive, the PC increments by 4 every clock cycle, indicating the address of the next instruction in the instruction memory, corresponding to the ADDER's function. As each instruction is executed, the alu_control signal changes to reflect the specific operation the ALU will perform, such as ADD at time 50, SUB at 100, and so on. The ALU processes the inputs in1 and in2, sourced from the register file's read_data1 and read_data2, producing outputs like 17 for ADD and 2 for SUB. These results are indicated by the alu_result signal. The read_reg_num1 and read_reg_num2 signals specify which registers to read, while write_reg denotes where the result should be stored. The write_data signal shows the data to be written back into the register file.

The regwrite signal controls the writing process to the register file, ensuring that the ALU results are stored in the designated registers. For example, during the ADD operation at time 50, the result 17 is written back to the register specified by write_reg. This timing diagram effectively illustrates the sequential processing of instructions within the RISC-V processor, demonstrating the coordination between control signals and data flow to execute arithmetic and logical operations efficiently.

VII. CONCLUSION AND FUTURE SCOPE

This research work designed and implemented a 32-bit single-cycle RISC-V processor using Verilog, executing operations like addition, subtraction, logical OR, AND, XOR, and SLT. Rigorous testing confirmed its functionality, validating components such as the Program Counter, Instruction Memory, Register File, and ALU. The research work demonstrated a solid grasp of RISC-V architecture and Verilog, paving the way for future improvements. Planned enhancements include a pipelined architecture for better performance, expanded instruction support for versatility, optimization techniques to reduce power consumption, and interrupt handling for real-time applications. Future work of this project is to implement multi-cycle RISC-V processor with pipelined structure. This can be implemented by using NOP operation or by using the concept of data forwarding. These upgrades will make the processor more powerful and versatile.

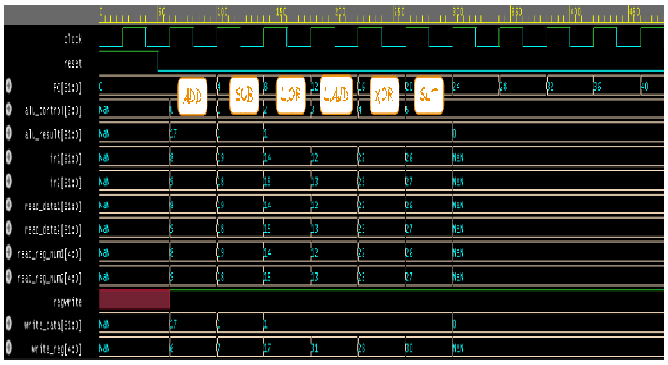


Fig. 8: Result of RISC-V Processor

REFERENCES

[1] Poli, L., Saha, S., Zhai, X., & Mcdonald-Maier, K. (2021). Design and Implementation of a RISC V Processor on FPGA. 2021 17th International Conference on Mobility, Sensing and Networking (MSN), 161-166.

[2] Thakor, K., Pal, A., & Shirodkar, M. (2017). Design of a 16-bit RISC Processor Using VHDL. International Journal of Engineering Research.

[3] Barriga, A. (2020). RISC-V processors design: a methodology for cores development. 2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS), 1-6.

[4] Harmina, T., Hofman, D., & Benjak, J. (2023). DCT Implementation on a Custom FPGA RISC-V Processor. 2023 International Symposium ELMAR, 163-167.

[5] Li, Z., Hu, W., & Chen, S. (2019). Design and Implementation of CNN Custom Processor Based on RISC-V Architecture. 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 1945-1950.

[6] Khairullah, S. (2022). Realization of a 16-bit MIPS RISC pipeline processor. 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), 1-6.

[7] Akshay Birari et al., "A RISC-V ISA Compatible Processor IP," 24th International Symposium on VLSI Design and Test, pp. 1-6, 2020.

[8] Aneesh Raveendran et al., "A RISC-V Instruction Set Processor Microarchitecture Design and Analysis," International Conference on VLSI Systems, Architectures, Technology and Applications, pp. 1-7, 2016.

[9] Aslesa Singh et al., "Design and Implementation of a 32-bit ISA RISC-V Processor Core using Virtex-7 and Virtex-UltraScale," IEEE, 5th International Conference on Computing Communication and Automation, pp. 126-130, 2020.

[10] Sudhir Dagar, and Geeta Nijhawan, "Area Efficient Moving Object Detection using Spatial and Temporal Method in FPGA," International Journal of Engineering Trends and Technology, vol. 70, no. 9, pp. 138-147, 2022.

[11] Jaewon Lee et al., “RISC-V FPGA Platform toward ROS-Based Robotics Application,” 30th International Conference on Field-Programmable Logic and Applications, pp. 370-370, 2020.

[12] RISC-V Specifications, 2021. Available: https://riscv.org/technical/specifications

[13] Pierre Maillard et al., “Test Methodology & Neutron Characterization of Xilinx 16nm Zynq® UltraScale+™ Multi-Processor System- on-Chip (MPSoC),” IEEE Radiation Effects Data Workshop (REDW), pp. 1-4, 2018.

[14] Stangherlin, Kleber, and Manoj Sachdev. "Design and implementation of a secure RISC-V microprocessor." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 30.11 (2022): 1705-1715.

[15] An, Hyogeun, et al. "Single Cycle 32-bit RISC-V ISA Implementation and Verification."

[16] R. Samanth, A. Amin and S. G. Nayak, "Design and Implementation of 32-bit Functional Unit for RISC architecture applications," 2020 5th International Conference on Devices, Circuits and Systems (ICDCS), Coimbatore, India, 2020, pp. 46-48, doi:10.1109/ICDCS48716.2020.243545.

[17] A. Singh, N. Franklin, N. Gaur and P. Bhulania, "Design and Implementation of a 32-bit ISA RISC-V Processor Core using Virtex-7 and Virtex- UltraScale," 2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA), Greater Noida, India, 2020, pp. 126-130, doi: 10.1109/ICCCA49541.2020.9250850.

[18] P. Mantovani, R. Margelli, D. Giri and L. P. Carloni, "HL5: A 32-bit RISC-V Processor Designed with High-Level Synthesis," 2020 IEEE Custom Integrated Circuits Conference (CICC), Boston, MA, USA, 2020, pp. 1-8, doi: 10.1109/CICC48029.2020.9075913.

[19] Gokulan, T., Akshay Muraleedharan, and Kuruvilla Varghese. "Design of a 32-bit, dual pipeline superscalar RISC-V processor on FPGA." 2020 23rd Euromicro Conference on Digital System Design (DSD). IEEE, 2020.

[20] V. Jain, A. Sharma and E. A. Bezerra, "Implementation and Extension of Bit Manipulation Instruction on RISC-V Architecture using FPGA," 2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT), Gwalior, India, 2020, pp. 167-172, doi: 10.1109/CSNT48778.2020.9115759.

[21] R. Höller, D. Haselberger, D. Ballek, P. Rössler, M. Krapfenbauer and M. Linauer, "Open-Source RISC-V Processor IP Cores for FPGAs — Overview and Evaluation," 2019 8th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 2019, pp. 1-6, doi: 10.1109/MECO.2019.8760205.

[22] Mangalwedhe, Sneha, Roopa Kulkarni, and S. Y. Kulkarni. "Low power 35 implementation of 32-bit RISC processor with pipelining." Proceeding of the Second International Conference on Microelectronics, Computing & Communication Systems (MCCS 2017). Springer Singapore, 2019.

[23] A. Kamaleldin, M. Ali, P. Amini Rad, M. Gottschalk and D. Göhringer, "Modular Memory System for RISC-V Based MPSoCs on Xilinx FPGAs," 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Singapore, 2019, pp. 68-73, doi: 10.1109/MCSoC.2019.00017.

[24] Toker, Onur. "A High-Level Synthesis Approach for a RISC-V RV32I-Based System on Chip and Its FPGA Implementation." Engineering Proceedings 58.1 (2023): 72.

[25] Ngu, David Teck Joung. Design and simulate RISC-V processor using verilog. Diss. UTAR, 2023.

[26] Jain, Vineet, Abhishek Sharma, and Eduardo Augusto Bezerra. "Implementation and extension of bit manipulation instruction on RISC-V architecture using FPGA." 2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT). IEEE, 2020.

[27] Jeemon, Jikku. "Pipelined 8-bit RISC processor design using Verilog HDL on FPGA." 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT). IEEE, 2016.

[28] Dennis, Don Kurian, et al. "Single cycle RISC-V micro architecture processor and its FPGA prototype." 2017 7th International Symposium on Embedded Computing and System Design (ISED). IEEE, 2017.