International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

# Design of RISC-V Processor Using Verilog

*Baggu Maneesha*[1], *Gudla Sriram*[2], *Ganteti Sai*[3], *Murapaka Siddhu*[4], *Smt. E. Jaya*[5]

[1,2,3,4] UG Student, [5]Guide, M. Tech. Associate Professor
Department of Electronics and Communication Engineering, Aditya Institute of Technology and Management

**ABSTRACT**

This project's primary objective is to create a 32-bit pipelined processor based on the open- source RV32I Version 2.0 ISA, RISC-V, with several modules. A processor known as a RISC (Reduced Instruction Set Computer) requires less hardware than a CISC (Complex Instruction Set Computer) in order to reduce the complexity of the instruction set and accelerate the execution time of each instruction. With the help of the required block diagrams, we also built this processor with five levels of pipelining, each of which has a detailed description of its operation. This project uses Verilog to develop and simulate a RISC-V. Because of its free and open instruction set architecture (ISA), the RISC-V processor design offers computer designers an alternative for both software and hardware creation. In addition, the five-stage pipeline procedures that the proposed RISC-V processor will employ will enhance the machine's overall performance. The pipeline register (IF/ID, ID/IEx, IEx/IMem, and IMem/IW), alu, aludec, maindec, imem, dmem, regfile, pc_mux, result_mux, and pipeline register are the first main modules to be implemented in the project. In addition, a hazard unit is incorporated into the design to lessen hazardous situations. Xilinx Vivado was used to simulate and validate these modules functioning.

**Keywords:** Xilinx Vivado, Hazard, 5-stage pipeline, Verilog and RISC-V processor.

## 1. Introduction

Transistor was invented in 1948 (23 December 1947 in Bell lab). IC was invented in 1958 (Fair Child Semiconductors) By Texas Instruments J Kilby. The first microprocessor was invented by INTEL.

### *1.1 Generations of microprocessors*

**First-generation** – From 1971 to 1972, the first generation of microprocessors appeared, including the INTEL 4004, Rockwell International PPS-4, and INTEL 8008 among others.

**Second generation** – From 1973 until 1978, the second generation of 8-bit microprocessors was developed. Processors such as the INTEL 8085, Motorola 6800, and 6801 were developed.

**Third generation** – The third generation saw the introduction of 16-bit processors such as the INTEL 8086/80186/80286, Motorola 68000 68010, and others. This generation employed the HMOS technology from 1979 to 1980.

**Fourth generation** – Between 1981 and 1995, the fourth generation existed. 32-bit CPUs based on HMOS manufacturing were created. This generation's prominent processors include the INTEL 80386 and Motorola 68020.

**Fifth-generation** –We've been in the fifth generation since 1995. There were 64-bit CPUs such as the PENTIUM, Celeron, twin, quad, and octa-core processors.

### *1.2 Types of microprocessors*

#### *1.2.1 Complex instruction set microprocessor*

These processors can download, upload, and recall data from memory, among other things. This microprocessor can also do sophisticated mathematical calculations in a single instruction, in addition to these functions. Example: IBM 370/168, VAX 11/780.

### *1.2.2 Reduced instruction set microprocessor*

These processors are built to do specific tasks. They are meant to use a reduced instruction set to reduce execution time. They are capable of carrying out minor tasks in response to particular commands. These processors are more efficient at completing commands. To implement a result at uniform execution time, they simply need one clock cycle. There are a lot of registers and a lot of transistors. The LOAD and STORE instructions are used to access the memory location. Example: Power PC 601, 604, 615, 620.

### *1.2.3 Superscalar microprocessor*

These processors are capable of handling multiple tasks at once. ALUs and multiplier-like arrays can both benefit from them. They feature several operating units and execute multiple orders to complete task.

### *1.2.4 Application-specific integrated circuit*

Application-specific integrated circuit (ASIC) processors excel in efficiency and performance due to their tailored design for specific tasks, offering customization advantages that optimize performance and reduce costs. Despite their complex development process, ASICs find applications across various industries, from finance to artificial intelligence, powering critical functions in niche markets where off-the-shelf solutions fall short. Their immutable design, once fabricated, underscores the importance of meticulous planning and optimization during the development phase. As technology advances, ASIC processors are poised to remain indispensable, driving innovation and efficiency in specialized computing domains.

### *1.2.5 Digital signal multiprocessor*

Signals such as analogue to digital and digital to analogue are converted using these processors. These processors' chips are found in a variety of devices, including RADAR SONAR home theatre systems.

## *1.3 RISC vs CISC*

Over the past decades, microprocessors and microcontrollers has been constructed around two types of architectures, Reduced Instruction Set computer and Complex Instruction Set Computer.

CISC instructions are variable in length and are encoded for doing more micro-operations per instruction. As a result, the complex architecture of CISC processors makes instructions take a longer time to execute. Since all instructions of a RISC processor have the same instruction length, the decoding process becomes easier compared to a CISC processor.

RISC is widely used due to its efficient architecture which can be used for low power and high- speed processing application. It supports very few addressing modes, LOAD and STORE instructions are the only instructions that are used to access the external memory. Hence RISC processors are mainly dependent on software and CISC processors are mainly dependent on hardware for executing complex tasks.

### *1.3.1 Reduced Instruction Set Architecture (RISC)*

The fundamental goal is to make hardware simpler by employing an instruction set that consists of only a few basic loading, evaluating, and storing stages. A store command does the same thing as a load command: it stores the data.

### *1.3.2 Complex Instruction Set Architecture (CISC)*

The basic notion is that a single instruction will handle all loading, evaluating, and storing operations, similar to how a multiplication command will handle loading, evaluating, and storing data, which is why it's complicated.

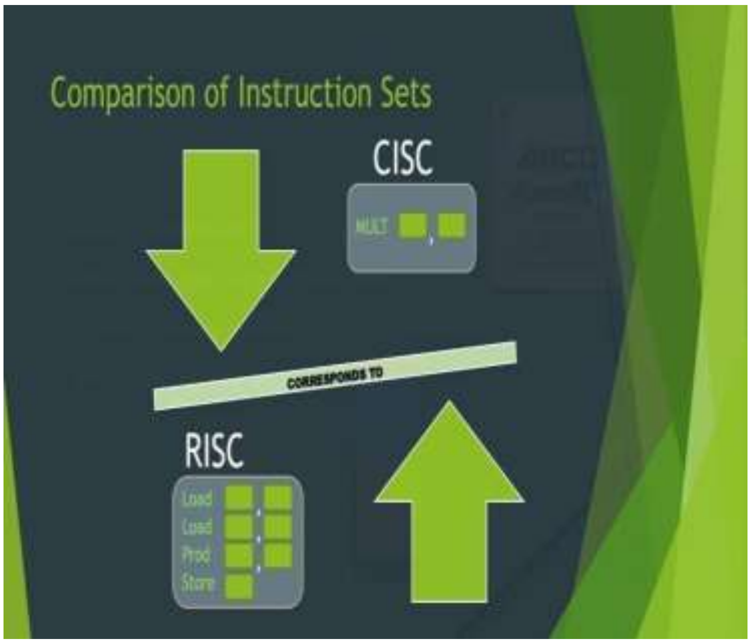


**Fig 1.3 RISC vs CISC**

## 2. Literature Review

### *2.1 Overview*

In this chapter, the implementation of difference architectures such as MIPS, RISC, and CISC by other authors will be discussed.

### *2.2 Previous Work*

The design of a 32-bit RISC processor based on MIPS using VHDL coding was presented by (S. P. Pitpurkar et al., 2015). They argued that RISC CPU had more benefits than CISC such as higher speed, simpler structure, and easier implementation. They used pipeline design to describe the system and achieve fewer clock cycles per instruction. They verified the design through extensive simulations. They used Xilinx 13.li ISE Simulator to design, synthesize and simulate the RISC processor based on MIPS. Their results showed that the design had a combinational delay of 0.758 ns and a maximum operating frequency of 1.350 Ghz.

Using Cadence, a software tool for electronic design automation, (Mohit N. Topiwala et al., 2014) designed and implemented a 32-bit processor based on MIPS. MIPS is a RISC architecture, which stands for reduced instruction set computer. RISC architectures aim to increase the speed of the processor by using a small set of simple and fast instructions. The authors stated that power consumption is a critical factor for embedded and portable applications. However, there is a trade- off between power, area, and delay in integrated circuits. For some applications, low power circuits are required, and the design engineers have to sacrifice more area and delay. Therefore, they suggested a power reduction technique by skipping pipeline stages that cause unnecessary switching activities. They designed Hazard detection unit and Data forwardingunit for efficient implementation of the pipeline.

They used Verilog-HDL to implement the design and Cadence RTL complier to synthesize it using typical libraries of TSMC 0.18 μm technology.

Using Verilog HDL, (Shofiqul Islam et al., 2006) developed a high speed- pipelined execution unit of 32-bit RISC processor. They arranged the block indifferent stages of pipeline so that the pipeline can operate at high frequency. The execution stage in a typical pipeline scheme consists of input data mux, operational block and output ALU mux. To increase the speed of the pipeline, they selected the data for the computational blocks in the execution stage one stage earlier in the data select stage. They also proposed a dependency resolver module to deal with a possible problem of consecutive data dependent instruction in the pipeline. This module handles both stalling and data forwarding. They synthesized the processor at 0.1-micron technology and achieved a working frequency of 714Mhz.

(Animesh Kulshreshtha et al., 2021) compared the behavioural models of 16- bit and 32-bit RISC processors and their different instruction sets. The 16-bit RISC processor was a non-pipeline CPU based on Harvard architecture, which had separate data memory and instruction memory. The 32- bit RISC processor was a pipelined CPU that followed the MIPS architecture. They aimed to study the differences between the models based on their instruction set and performance factors such as speed and power consumption. They used an optimized Multiplier algorithm to improve the data path. In general, the 32-bitprocessor consumed about 60% more power than the 16-bit processor because of its higher operating frequency. However, the 32-bit processor was 70% faster than the 16-bit processor. These results were expected because the 32-bit processor can store more computational values and the pipelined architecture of the processor reduces the length of each instruction cycle, which increases the operating frequency and decreases the combinational delay.

Based on their results, the maximum operating frequency for the 16-bit processor and the 32-bit processor was 78.654 MHz and 139.438 MHz, respectively. The maximum combinational delay 13.981 ns for the 16-bit processor and 7.028 ns for the 32-bit processor.

Using Verilog HDL coding, (Shawkat S. Khairullah, 2022) designed and implemented a 16-bit RISC processor with 5 pipeline stages that was simulated using Xilinx ISE Design Suite 14.7 tool. They synthesized the design on device Xilinx XC3S200FT256 FPGA chip. They showed the experimental and timing diagram results that indicated that the execution unit hardware design used 2%of Spartan – FPGAXC3S2000 area with a maximum speed of 56.8 MHz. They also showed that the data memory unit hardware design used 8% of the same FPGA area with a maximum speed of 67.32 MHz. Moreover, they showed that he instruction unit hardware design used 6% of the same FPGA area with a maximum speed of 106 MHz.

Using VHDL, (Sarah M. Al-sudany et al., 2021) designed and implemented a multicore RISC processor on FPGA. They used 32-bit MIPS processor with three main components: 32-bit data path, control unit, and hazard unit. They divided the single cycle MIPS system into five pipeline stages to create the pipeline MIPS processor. They also solved the data hazard, control hazard, and structural hazard in their design. They developed the MIPS using Xilinx ISE 14.7 design suite and successfully implemented it on Xilinx Virtex-6 XC6VLX240T-1FFG1156 FPGA. The multicore MIPS processor consumed3.422 watt of power and had an operating frequency of 136.444 MHz.

### *2.3 Implementation and design of various processor from previous works*

| Author | Description |
|---|---|
| Pitpurkar et al.,2015 | Implement a design of 32-bit RISC based MIPS processorusing VHDL coding<br>RISC has more advantages, such as faster speed, simplified structure, and easier to be implemented as compared to CISC<br>Xilinx 13.li ISE Simulator was used to the design, synthesis and simulation<br>Achieved combinational delay of 0.758ns and maximum operating frequency of 1.350 GHz |
| N. Topiwala etal., 2014 | Implemented a 32-bit MIPS based processor using Cadence<br>Power is the most important parameter for embedded and portable applications<br>Proposed a power reduction technique throught by- passingpipeline stages that cause unnecessary switching activities.<br>Hazard detection unit and Data forwarding unit were designed for efficient implementation of the pipeline Implemented using Verilog-HDL and synthesized using Cadence RTL complier using<br>typical libraries of TSMC 0.18 $\mu m$ technology |
| Shawkat S. Khairullah, 2022 | Presented hardware realization of 5 pipeline stages of a16-bit RISC processor<br>Processor is simulated using Xilinx ISE Design Suite 14.7tool<br>Synthesis process of the proposed system is realized on device Xilinx XC3S200FT256 FPGA chip<br>Execution unit uses 2% of Spartan – FPGA XC3S2000 area with maximum allowable speed of 56.8 MHz.<br>Data memory unit uses 8% of Spartan – FPGAXC3S2000 area with maximum allowable speed of 67.32 MHz Instruction unit uses 6% of the same FPGA ship area with maximum allowable speed of 106<br>MHz |
| Iqul Islam et al.,2006 | Designed a high speed-pipelined execution unit of 32- bitRISC processor<br>Data selection for the computational blocks in Execution stage is performed one stage ahead Dependency Resolver module was proposed to solve consecutive data dependent instruction in the pipeline Basically, the module handles both stalling as |

| | |
|---|---|
| | well as data forwarding |
| mesh Kulshreshthaet al., 2021 | Analyzed behavioural model of 16-bit and 32- bit RISC processor and their independent instruction sets<br>16-bit RISC processor was a non-pipeline Harvard architecture-based CPU<br>32-bit RISC was a pipelined processor from MIPS architecture<br>Total power consumption for 32-bit processor was about 60% more than 16-bit processor due to higher operating frequency<br>32-bit processor was 70% faster than 16-bit processor Maximum operating frequency for 16-bit processor and 32-bit processor is 78.654 MHz and 139.438 MHz Maximum combinational delay for 16-bit processor and 32-bit processor is 13.981 ns and<br>7.028 ns |
| M. Al-sudany etal., 2021 | Studied for multicore RISC processor implemented on FPGA<br>MIPS was developed using Xilinx ISE 14.7design suite and were implemented successfully on Xilinx Virtex-6 XC6VLX240T-1FFG1156 FPGA<br>32-bit MIPS processor was designed using VHDL with 3 main structures: 32-bit data path, control unit and hazard unit<br>Total power analysis of multicore MIPS processor is<br>3.422 watt, and the operating frequency is 136.444 MHz |

## 3. Methodology

### *3.1 Background Theories*

#### *3.1.1 Instruction Set Architecture*

Instruction Set Architecture (ISA) is a term that refers to the set of instructions that a computer processor can execute. It is a fundamental aspect of computer architecture that determines the capabilities and limitations of a processor.

An ISA is composed of a set of instructions that define the operations that a processor can perform, as well as the format and meaning of each instruction. Each instruction typically includes an opcode, which specifies the operation to be performed, and one of more operands, which specify the data on which the operation is to be performed. Figure xxx shows the example of opcode and operand for a MOV instruction.

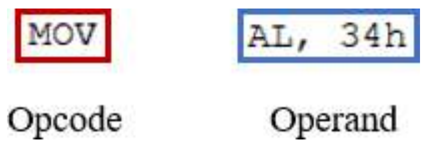


**Figure 3.1 Opcode and Operand of a MOV instruction**

From Figure 3.1, the opcode is the MOV instruction. The other parts are called the operands. Operands are manipulated by the opcode. In this example, the operands are the register named AL and the value 34 hex.

ISAs can be classified into 2 main categories: Reduces Instruction Set Computing (RISC) and Complex Instruction Set Computing (CISC). RISC ISAs have a smallerset of simple instructions can be executed quickly, whereas CISC ISAs have a larger set of more complex instructions that can perform multiple operations in a single instruction. The RISC approach is generally favored in modern processors because it allows for faster execution times and simpler processor designs. However, CISC architectures are still used in certain specialized applications where the additional complexity is justified by the increased functionality.

One example of a CISC ISA is the x86 architecture, which is used in many personal computers and servers. The x86 ISA includes a large set of instructions which are quite complex and can perform multiple operations in a single instruction. For example, MOV instruction from CISC ISA can transfer data between two memory locations or between a register and a memory location in a single instruction. Besides, the x86 ISA also includes a variety of specialized instructions for performing common tasks such as string manipulation, input/output operations, and floating-point arithmetic. x86 ISA is known for its complexity and backward compatibility. However, this complexity can make it difficult to optimize for performance or energy efficiency. Hence, it is more challenging to write software that runs efficiently on different processor with different implementations of the ISA.

One example of a RISC ISA is the ARM architecture, which is used in a wide range of device including smartphones, tables, and embedded systems. The ARM ISA includes a relatively small set of simple instructions which can be executed faster. For example, the MOV instruction from RISC ISA only copies data from one register to another. Besides, ARM ISA also includes a variety of specialized instruction such as loading and storing multiple registers at once, performing conditional instructions, and performing arithmetic operations on multiple data types.The ARM ISA is known for its simplicity and energy efficiency.

The simplicity of the ISA also makes it easier to optimize for performance and make it easier to write software on different processors with different implementations of the ISA.

### *3.1.2 5-Stage pipeline*

With traditional single cycle data path, instructions are executed in a single clock cycle. The next instruction will need to wait for the previous instruction to be complete before it can be executed. Moreover, the execution time for each instruction is different. There are instructions that takes longer time to execute than the other. This might lead to wasted clock cycle and reduced performance of the processor. However, this issue can be solved by using pipelining.

Pipelining is a technique used in computer architecture to increase the overall performance of a processor. It involves breaking down the execution of a task into smaller stages and allowing these stages to overlap in time. With this, multiple instructions can be executed simultaneously and improve the throughput of the system.

The most common used pipeline technique in modern processor design is 5-Stage pipeline technique. As suggested by its name, the instruction execution cycle was divided into 5 different stages. Each stage performs a specific operation on the input data and passes the result to the next stage. The output of the first stage become the input of the second stage, and so on. Besides, every stage operates on a different part of the data, which allow multiple instructions to be in different stages of execution at the same time. The 5 stages of a typical pipeline are: fetch, decode, execute, memory, and writeback. The details for each stage were discussed below.

1. Fetch

In this stage, the processor fetches the instruction from memory. Then, the instruction is loaded into the instruction cache, which is used to store the recently used instructions.

2. Decode

In this stage, the processor decodes the fetched instruction to determine the operation it needs to perform. The instruction is analyzed to determine the type of operation, registers, and the locations of any operands.

3. Execute

In this stage, the processor performs actual operation specified by the instruction. This involves arithmetic or logical operations, such as addition and subtraction. Besides, it also involves memory access or branching to a different part of the program.

4. Memory

In this stage, the processor access memory to read or write the data obtained from previous stage. However, this stage is optional. Some instruction such as “add” does not involve memory access. 5.Writeback

In this stage, the results of the operation are written back to the appropriate register in the registerfile.

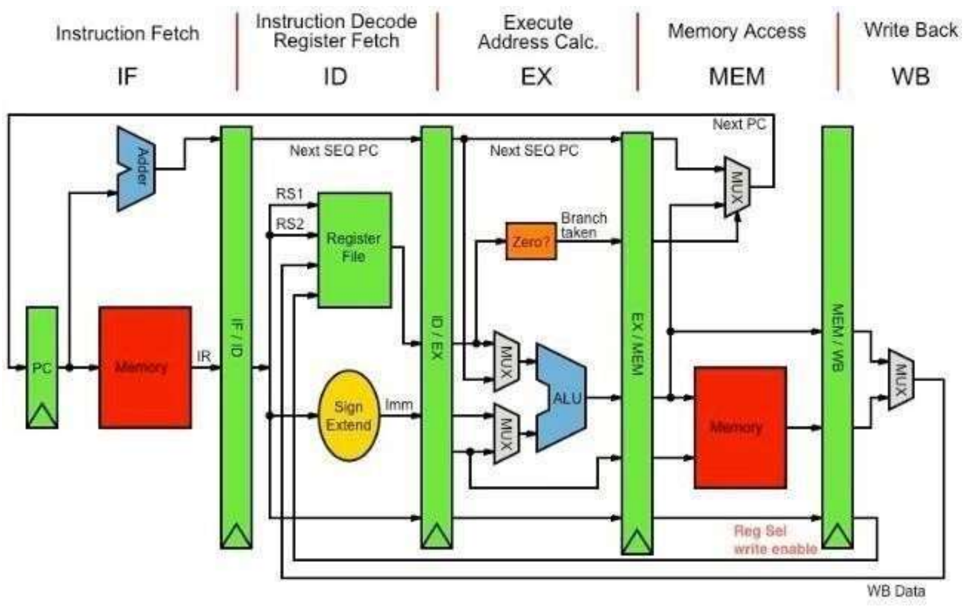


**Figure 3.2: 5 Stage pipeline**

### *3.1.3 Xilinx Vivado*

Xilinx Vivado is a sophisticated software suite designed to facilitate the development and implementation of FPGA (Field-Programmable Gate Array) and SoC (System on Chip) systems. It serves as a comprehensive tool for engineers and designers working on a wide range of digital circuitry projects, from simple logic designs to complex embedded systems. With its user-friendly interface and powerful features, Vivado enables users to efficiently design, simulate, implement, and verify their FPGA designs.

At the core of Vivado's capabilities is its support for multiple design entry methods. Users can choose to create their designs using Hardware Description Languages (HDL) such as Verilog or VHDL, allowing for low-level control and customization. Alternatively, Vivado offers a block diagram-based design entry tool called IP Integrator, which simplifies the process of integrating pre-designed IP cores and building complex systems through a graphical interface. This flexibility in design entry makes Vivado suitable for a wide range of users, from experienced RTL designers to those who prefer a more visual approach. Once the design entry phase is complete, Vivado

guides users through the subsequent steps of the FPGA design flow, including synthesis, implementation, and verification. During synthesis, the RTL code is transformed into a gate-level netlist, optimizing the design for the target FPGA device. The implementation phase involves floor planning, placing, and routing the synthesized logic onto the physical FPGA chip while meeting timing and resource constraints. Vivado provides advanced analysis tools to ensure that designs are functionally correct and meet performance specifications before proceeding to the next stage. One of Vivado's key strengths is its comprehensive set of features for design optimization and debugging. Engineers can leverage built-in tools for timing analysis, power estimation, and resource utilization to fine-tune their designs for optimal performance and efficiency. Additionally, Vivado offers robust debugging capabilities, including hardware debugging with integrated logic analyzers and waveform viewers, enabling users to identify and resolve issues quickly.

Beyond design implementation, Vivado also supports the generation of configuration bitstreams and programming of FPGA devices, allowing users to deploy their designs onto physical hardware for testing and deployment. Throughout the design process, Vivado provides extensive documentation and reporting features to help users track their progress, document design decisions, and collaborate effectively with team members.

Beyond design implementation, Vivado also supports the generation of configuration bitstreams and programming of FPGA devices, allowing users to deploy their designs onto physical hardware for testing and deployment. Throughout the design process, Vivado provides extensive documentation and reporting features to help users track their progress, document design decisions, and collaborate effectively with team members.

Overall, Xilinx Vivado stands as a comprehensive and powerful tool suite for FPGA design, offering a seamless workflow from design entry to deployment. Its intuitive interface, advanced features, and extensive documentation make it a preferred choice for engineers and designers working on cutting-edge digital circuitry projects.

### *3.2 Implementation of Design*

The processor consists of alu, aludec, maindec, imem, dmem, IF_ID, ID_IEx, IEx_IMem, Imem_IW, pc_mux, reg_file, Each component of the datapath will be discussed in this section.

### *3.2.1 Arithmetic Logic Unit (alu)*

Arithmetic Logic Unit (alu) is a fundamental component of a CPU. It is responsiblefor performing arithmetic and logical operations on binary data. It reads the data from pipeline register ID_IEx as an input and perform various arithmetic operationbased on the signals from aludec. The output is stored as ALUResults. In this RISC-V processor design, 7 instructions are implemented in the ALU.

Table 3.1: Base Integer Instructions for RV321

| **Category** | inputs | Description |
|---|---|---|
| **Arithmetic** | | |
| ADD | rd, rs1, rs2 | rd □ rs1 + rs2 |
| SUB | rd, rs1, rs2 | rd □ rs1 - rs2 |
| **Logical** | | |
| XOR | rd, rs1, rs2 | rd □ rs1 ^ rs2 |
| AND | rd, rs1, rs2 | rd □ rs1 & rs2 |
| OR | rd, rs1, rs2 | rd □ rs1 \| rs2 |
| **Shifts** | | |
| SHL | rd, rs1, rs2 | rd □ rs1 << rs2 |
| SHR | rd, rs1, rs2 | rd □ rs1 >> rs2 |

### *3.2.2 ALU Decoder (aludec)*

Arithmetic Logic Unit Decoder (aludec) is used to decode the instructions. It received the signal from the Main Decoder Unit (maindec) and determine the type of operation that had to be performed by the alu. It combined all 4 inputs from ALUOp, funct3, funct7b5 and opb5 to decode the instruction. funct7b5 is referring to instruction [31:25] funct3 is referring instruction [14:12], whereas opb5 is referring to instruction [0:6] in RISC-V instruction format. RTypeSub is obtained by performing AND operation on funct7b5 and opb5. This is used to differentiate the R-type ADD and SUB instruction.

### *3.2.3 Main Decoder (maindec)*

Main Decoder (maindec) is used to generate the control signals from the 7 bits opcode (instruction [6:0]) to determine the types of instruction. The control signals are RegWrite, MemWrite, Result, Branch, ALUOp,and Jump. Each of these control signals control the multiplexer for making decisions in the datapath to allow the data flow accordingly to the instructions.

### *3.2.4 Data Memory (dmem)*

In computer architecture, data memory is a component of a computer system that is responsible for storing and retrieving data. Data memory is typically used to store data that is actively being processed by alu. Usually there are 3 inputs in this module,which are write_enable, data_address, and write_data. The module takes the memory address from the results of alu (data_address) and data from register file (write_data). Write enable (write_enable) is used to control the write permission of the data to the data memory.

### *3.2.5 Instruction Memory (imem)*

In computer architecture, instruction memory is a component of a computer system that stores the instruction of a program. It is responsible for holding the sequence of instruction that the CPU fetches, decodes, and executes during the program execution. In this module, 32-bit instruction set is generated and stored in the ram array. The instruction to be fetch is based on the program counter fetch (PCF) input. As each instruction is 4 bytes, the value of PCF will be incremented by 4 to fetch.

### *3.2.6 Pipeline Register*

In computer architecture, pipeline register is a temporary storage element used un a processor’s pipeline to hold data between different stages of instruction execution process. It serves as a synchronization point between adjacent stages, allowing instruction to flow through the pipeline in a controlled manner.

In 5 stages pipeline,there are 4 pipeline registers namely IF_ID, ID_IEx, IEx_IMem, and IMem_IW. The registers are named for the two stages separated by that register. For example,the first pipeline register is IF_ID because it separates the instruction fetch and instruction decode stages.

#### *3.2.6.1 IF_ID*

IF_ID register as it names called, it separates the instruction fetch and instruction decode stages. It used to store data such as instruction fetch from instruction memory and ready to be released to decode stage on the next clock cycle. Besides, the current PC and next incremented PC address(PCPlus4F) is also saved in the IF_ID register in case it needed later for an instruction.

ID_IEx register as it names called, it separates the instruction decode and execute stages. It used to store information such as read data (RD1, RD2) from the register file and extended immediate value (ImmExt). Besides, it carries forward the data of PC and PCPlus4F from IF_ID register. Instruction[11:7] (rd), Instruction[19:15] (rs1), and Instruction[24:20] (rs2)will also be stored to ID_IEx register and send to Hazard Unit in execute.

#### *3.2.6.2 IEx_IMem*

IEx_IMem register as it names called, it separates the execute and memorystages. It is used to store the ALU results (ALUResult) and write data (Writedata). At the same time, Instruction [11:7](rd) and PCPlus4F are also carried forward from previous pipeline registers and stored in IEx_IMem register. The implementation of IEx_IMem Is done by Verilog code.

#### *3.2.6.3 IMem_IW*

IMem_IW register as it names called, it separates the memory and writeback stages. It used to store ALU results (ALUResult) and read data (ReadData)from data memory. Instruction [11:7] (rd) and PCPlus4F are also carried forward from previous pipeline registers and store in IMem_IW register. The implementation of IMem_IW done by Verilog code.

### *3.2.7 Write Data Selection MUX (result_mux)*

The ALU has the capability to carry out arithmetic operations like addition (A+B)or logical operation like equality comparison (A=B). Depending on the specific instructions being executed, the output of the ALU could be either a memory address or the result obtained from the ALU operation. To handle this situation, a MUX is needed to make decision between selecting thedata address or the ALU output value for writing back to the register file. This allows flexibility and efficient data handling in the processor's pipeline. The implementation of result_mux is done by Verilog code.

### *3.2.8 Program Counter Selection MUX (pc_mux)*

The Program Counter (PC) is a vital component used by the CPU to maintain the current instruction being executed. Under normal circumstances, the program counter increments by a fixed value, typically 4 (corresponding to a 32-bit instruction) for each clock cycle. This ensures that the program counter always points to the memory address of the next instruction to be executed. However, the program counter can be interrupted or modified by a jump signal (jump) from the control unit. When the predetermined conditions are met, the control unit instructs the program counter to deviate from its regular incrementation and instead updated its value to the jump.

### *3.2.9 Registerfile*

The register file (regfile) in a CPU plays a critical role in storing and manipulating data during program execution. It serves as a high-speed storage component that holds a set of registers, each capable of storing a fixed width if data. The Register File (regfile) in a RISC-V processor is responsible for storing the processor's general-purpose registers (GPRs) and providing read and write access to these registers. In a simple implementation, the regfile can be represented as an array of 32 32-bit registers, where each register holds a 32-bit value this implementation,rs1_addr, rs2_addr, and rd_addr are the input addresses for the first source register, second source register, and destination register, respectively. wd_data is the input data to be written into the register file, and wr_en is the write enable signal. The rs1_data and rs2_data outputs provide the data read from the first and second source registers.

### *3.2.10 Hazard Unit (hazard unit)*

The Hazard Unit is a component in a CPU's architecture that is responsible for detecting and handling hazards that can occur during the execution of instructions. Hazards refer to situations where the sequential execution of instructions may lead to incorrect or unintended behavior due to dependencies or conflicts between instructions. The hazard unit detects these hazards and takes appropriate actions to mitigate their effects. With this, the dependencies of write data begins from a pipeline register, rather than waiting for the WB stage to write the register file. Thus, the required data exists in time for later instructions, with the pipeline registers holding the data to be forwarded.

## 4. Results and Discussion

### *4.1 Overview*

In this chapter, each module that were discussed on previous chapter were simulated by using Xilinx Vivado software. Waveforms were generated from the simulations and the results were analyzed and discussed. Besides, the results were used to verify the functionality for each module of the design.

In the simulation, the functionality for each module were tested by running a set of test sequence with different instructions. Furthermore, the clock cycle used throughout the simulation is set to 100ps. The test sequence consists of a main code with label main, 4 branches with label around, end, and wrong, and end with a done label. In the main label, it consists of some arithmetic and logical instruction such as add, or, and xor with these instructions, alu, data memory, instruction memory, register file, and control unit module can be tested. Besides, the test was also designed to hit hazard conditions. The future depends on the computing speed of these RISC processors. IOT devices has already started emerging in this century. More and more RISC based processors will be designed and verified for the consumers. Thus, more flexible designs are needed like the one being projected in this report. We were writing the code and design 5-stage pipeline within these 5 stages fetch, decode, execution, memory, write back and synthesis the processor and find the power and time delay and efficiency of overall the processor.

### *4.2 Simulation and Analysis of Waveforms*

#### *4.2.1 PC MUX and analysis*

In a RISC-V processor, the program counter (PC) multiplexer is used to select the next PC value based on different conditions, such as a branch or jump instruction. In this module, PCMux, the pc_sel signal determines which PC value to select. When pc_sel is 2'b00, the next PC value is the next sequential instruction (pc_next). When pc_sel is 2'b01, the next PC value is for branch or jump instructions (pc_branch). When pc_sel is 2'b10, the next PC value is for jump instructions (pc_jump). If pc_sel does not match any of these conditions, the default behavior is to select the next sequential instruction (pc_next). This module can be used in the control unit of a RISC-V processor to control the flow of instructions by selecting the appropriate PC value based on the current instruction and control signals.

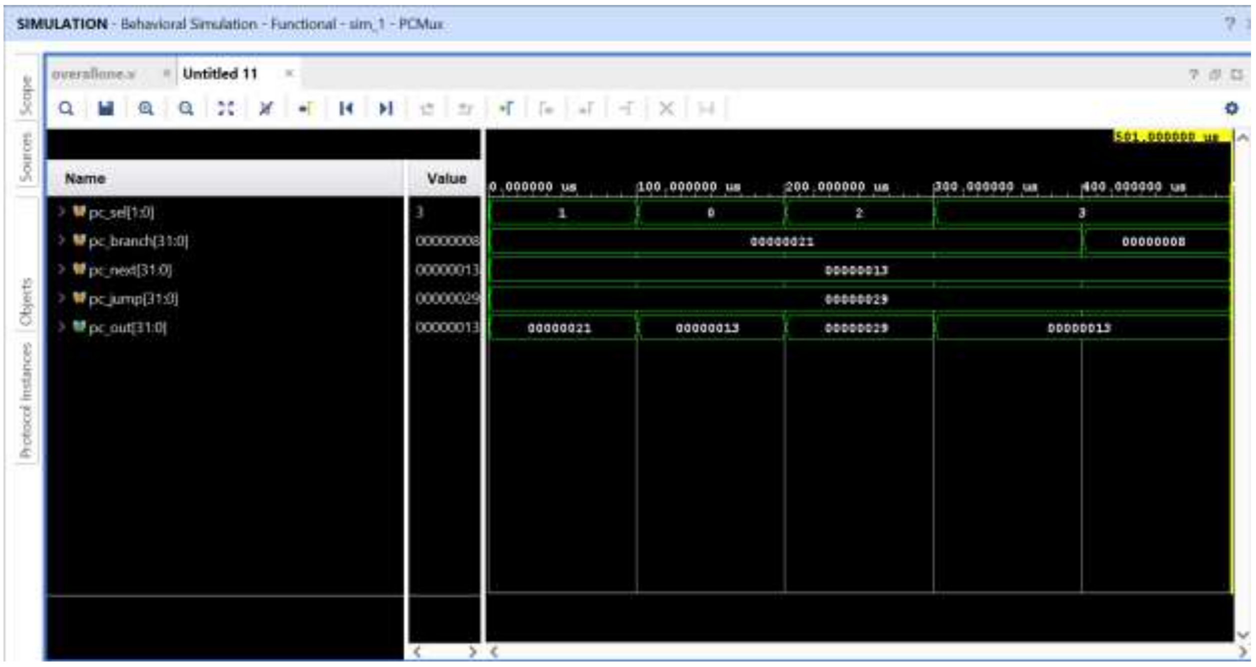


**Figure 4.2.1 Simulation waveform pc mux**

#### *4.2.2 Instruction Memory (imem) Waveform Analysis*

The instruction memory module is used to setup all the instruction flows of the test. From the simulation, there is a program counter fetch (PCF) input determines which instruction to fetch. Since each instruction had 4 bytes, the PCF value will increase by 4 to get the following instruction.

A part simulation waveform for imem is captured and shown in Figure 4.3. The functionality of the module is verified by viewing the changes of PCF value and the corresponding fetched instruction.

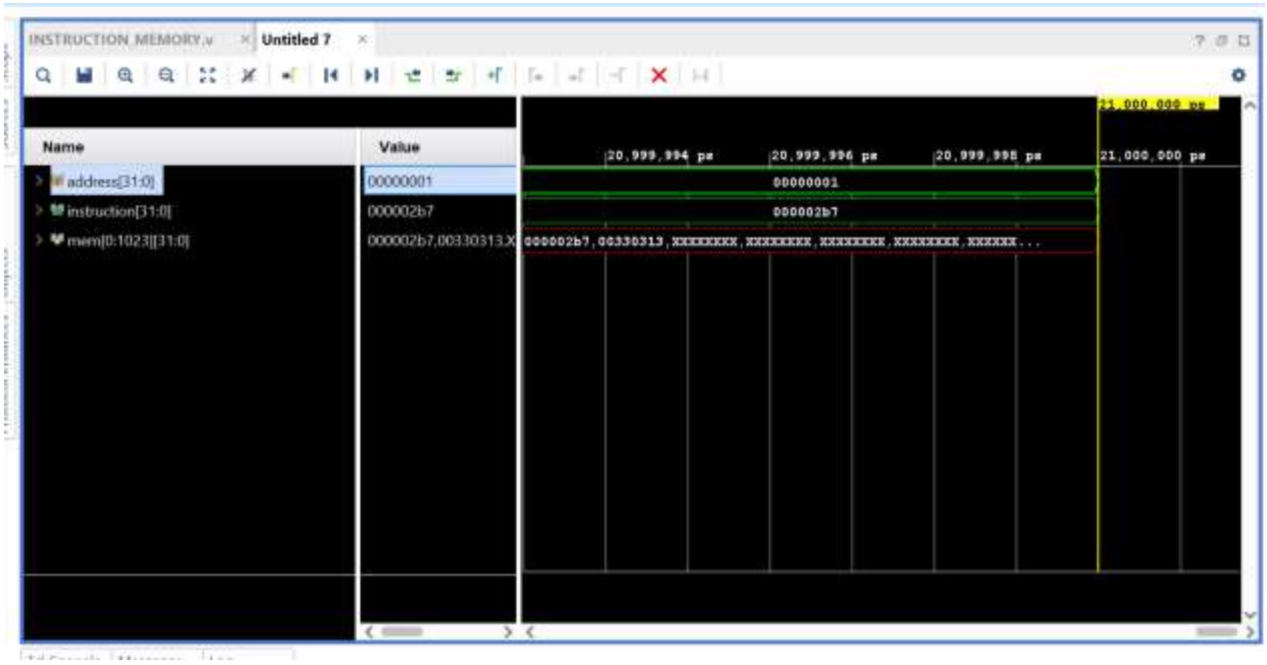


**Figure 4.2.2 Simulation waveform of instruction memory**

### *4.2.3 Data memory (dmem) Waveform Analysis*

The data memory (dmem) is used to store data in the datapath. The data is written to the module if write enable bit (MemWrite) is set. Else, the data is read from the data address (a) from the alu output through IEx/IMem pipeline register.

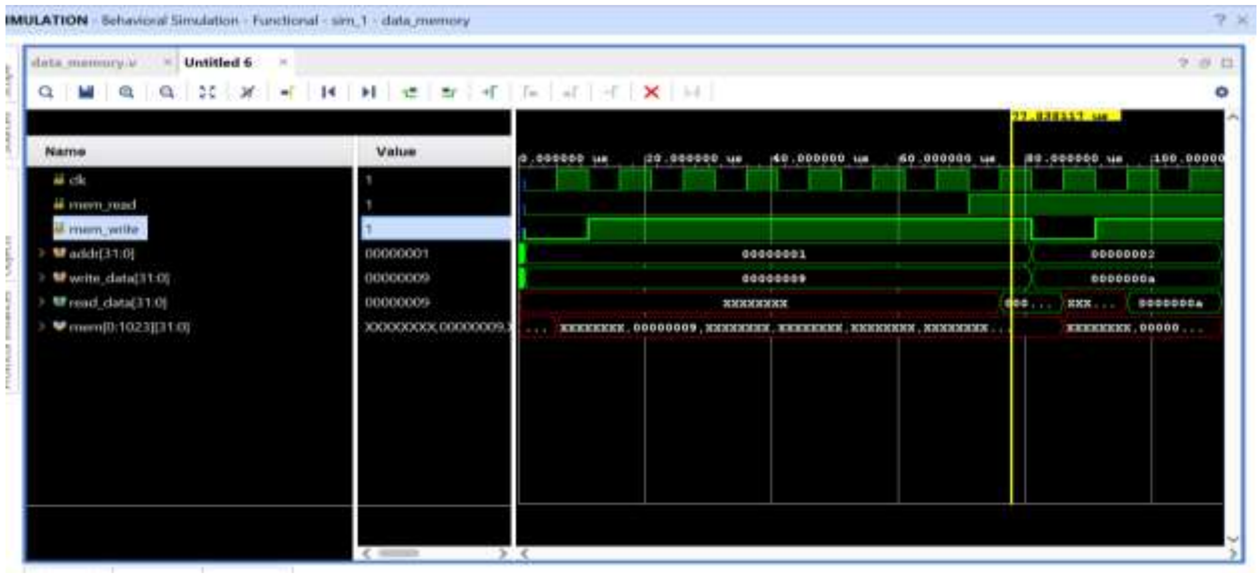


**Figure 4.2.3 Simulation waveform of dmem**

The focus of the simulation for dmem module instruction because it is the only instruction that have MemWrite set to 1. Instruction 0x0471aa23 is an example of sw instruction. It is notice that after 3 cycles after sw instruction is fetched, the data reached memory stage. In memory stage, it read the MemWrite from the control unit. If the value is set, the writed ata (wd) is written to data memory and output the value (rd) on the next cycle. Therefore, the functionality of dmem is verified.

### *4.2.4 Register File (regfile) Waveform Analysis*

The register file (regfile) serves as a temporary memory to store data. From the simulation, the inputs for the regfile include read address (a1 and a2), the write address of register (a3), the data to be written into the register (wd3), the data read from the register at the outputs (rd1 and rd2), and the control signal (we3) to enable write data into the register.

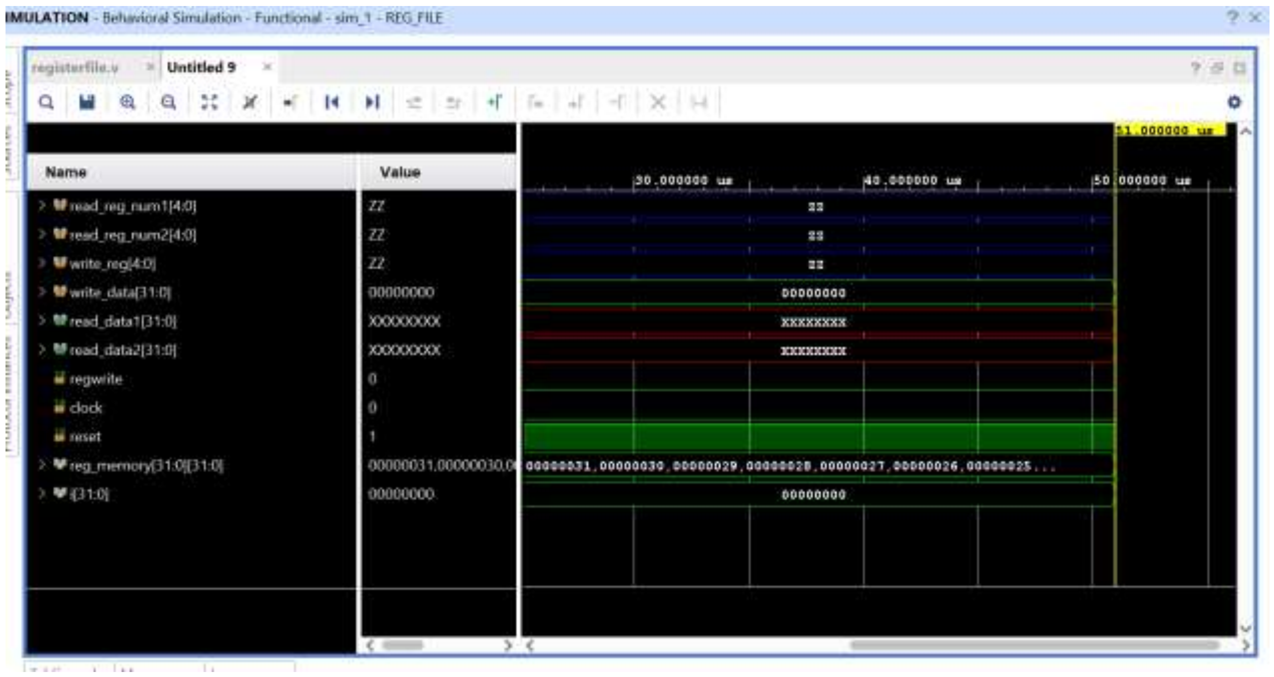


**Figure 4.2.4 Simulation waveform of regfile**

From Figure 4.5, it is noticed that a1 and a2 holds the addresses of rs1 and rs2. Besides, then we3 is high, the data (wd3) is written into the addressed holds by a3.Furthermore, rd1 and rd2 output the value from the addresses holds by a1 and a2 respectively. Therefore, the functionality of regfile module is verified.

### *4.2.5 Arithmetic Logic Unit (alu) Waveform Analysis*

The alu is an essential component of a CPU that handles arithmetic and logical operations on binary data. It receives input data from the ID_IEx stage and perform various arithmetic operations based on signals from the ALU decoder (ALU Control). The output is stored as ALU Results.

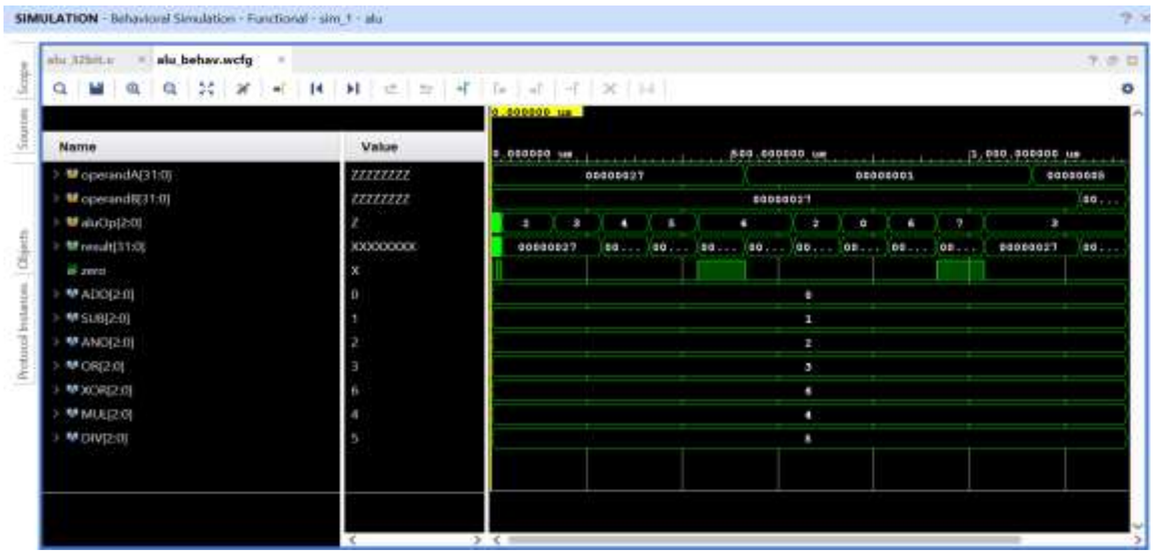


**Figure 4.2.5 Simulation waveform of alu**

operand_a and operand_b are the input operands to the ALU, alu_op is the 3-bit control signal specifying the operation to be performed, result is the output of the ALU, and zero is a flag indicating whether the result is zero. instruction OR, XOR, ADD, BEQ, and SLT is simulated. With different ALU Control, the alu operates different operations. For example, with ALU Control = 0011, alu will perform OR operation. The inputs are 0x0001 and 0x0005 respectively. The result of the operation is 0x0007 and stored in ALU Results. Therefore, the functionality of alu module is verified. The analysis of the simulated alu waveform.

### *4.2.6 Arithmetic Logic Unit Decoder (aludec) Waveform Analysis*

A RISC-V processor, the ALU decoder is responsible for interpreting the ALU operation code (ALUOp) from the instruction and generating control signals for the ALU to perform the desired operation. The ALU decoder takes the ALUOp as input and produces control signals for the ALU, such as add, sub, mul, div, shl, shr, etc.opcode is the input representing the opcode of the instruction.

Based on the opcode, the alu_control output is set to the corresponding control signals for the ALU operation. Each case in the case statement corresponds to a specific opcode and sets alu_control accordingly. The aludec decodes the instructions ALUOp fromthe control unit to determine the types of operation that has to be performed by the alu.A part simulation waveform for aludec is captured.

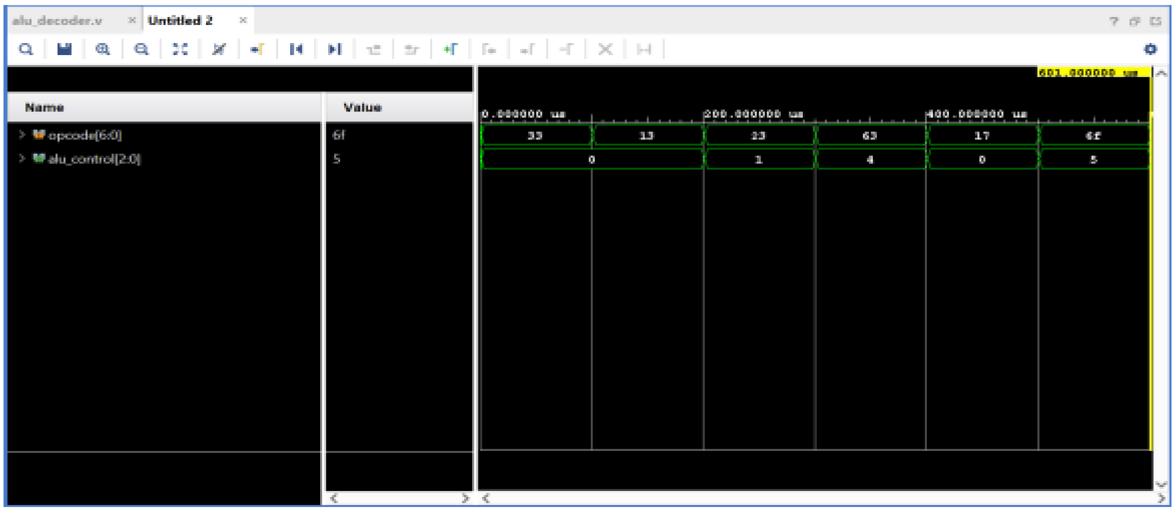


**Figure 4.2.6 Simulation waveform of aludec**

### *4.2.7 Main decoder (maindec) Waveform Analysis*

The maindec is used to generate control signals based on the 7 bits opcode (instruction [6:0]) of an instruction. Its purpose is to interpret the opcode and determine the type of instruction being executed.

In a RISC-V processor, the main decoder is responsible for decoding the instruction opcode and generating control signals for various parts of the processor, including the ALU, memory units, and register file. It takes the opcode of the instruction as input and produces control signals that determine the operation to be performed in each stage of the pipeline.

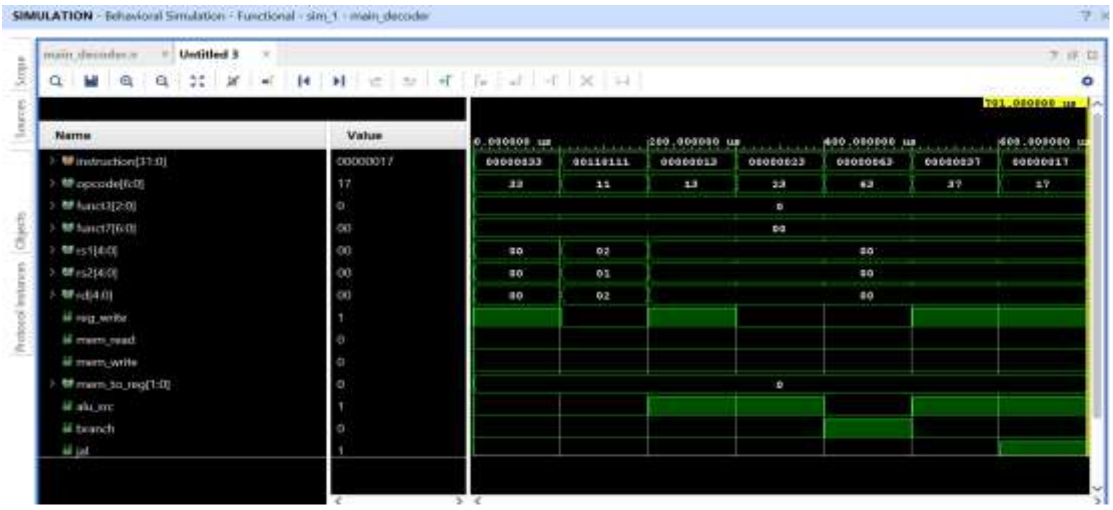
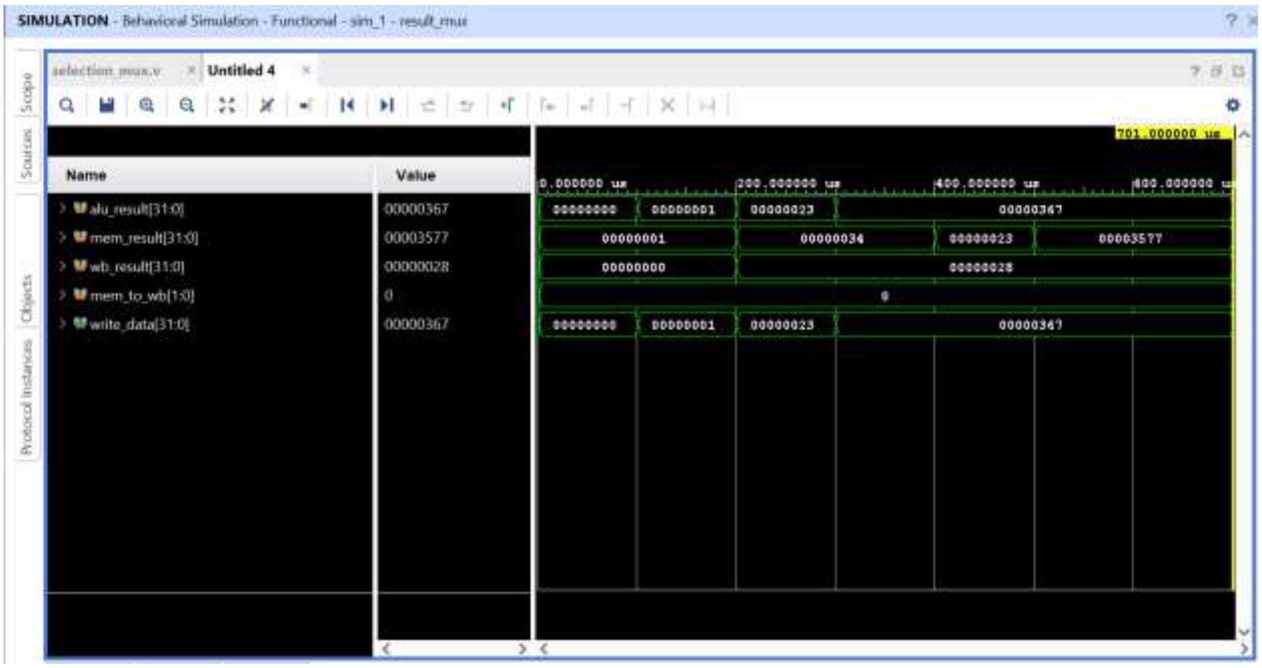


**Figure 4.2.7 Simulation output of maindec.**

opcode is the input representing the opcode of the instruction. Based on the opcode, the main decoder generates control signals for the ALU (alu_control), memory read and write (mem_read and mem_write), register file write (reg_write), destination register address (rd_addr), source register addresses (rs1_addr and rs2_addr), and immediate value (imm) for I-type instructions. Therefore, the functionality of maindec is verified.

### *4.2.8 Write Data Selection MUX (result_mux) Waveform Analysis*

In a RISC-V processor, the data selection result multiplexer (mux) is used to select the result from the ALU or the memory data, depending on the instruction being executed. alu_result is the output from the ALU, mem_data is the data read from memory, and mem_to_reg is a control signal indicating whether to select the data from memory (mem_data) or the ALU (alu_result). The result output is the selected data that will be written back to the register file. You would need to connect the alu_result, mem_data, and mem_to_reg signals from your processor to the inputs of this module, and the result output to the input of the register file write port.



| Name | Value |
|---|---|
| alu_result[31:0] | 00000367 |
| mem_result[31:0] | 00003577 |
| wb_result[31:0] | 00000028 |
| mem_to_wb[1:0] | 0 |
| write_data[31:0] | 00000367 |

```
alu_result[31:0]   00000000 | 00000001 | 00000023 | 00000367
mem_result[31:0]   00000001 | 00000034 | 00000023 | 00003577
wb_result[31:0]    00000000 | 00000028
mem_to_wb[1:0]     0
write_data[31:0]   00000000 | 00000001 | 00000023 | 00000367
```

**Figure 4.2.8 Simulation output of result_mux**

### *4.2.9 Pipeline Register*

The pipeline register is used to store information from previous stage and load them to next stage. This ensures data can be carried forward correctly and allow instructions to be executed through the pipeline.

#### *4.2.9.1 IF_ID*

IF_ ID register located in between fetch stage and decode stage. In this Verilog module, pc_in and instruction_in are the inputs from the datapath fetch stage, enable is a control signal indicating that the decoder stage is ready to accept the new instruction, pc_out and instruction_out are the outputs to the decoder stage, and clk and reset are the clock and reset signals, respectively.

The pc_reg and instruction_reg registers store the values of the program counter (PC) and instruction fetched from memory. The values are updated on the rising edge of the clock (clk) when enable is asserted. The stored values are then outputted to the decoder stage. It helps to store instructions (Instr) from the fetch stage and load them to decode stage on the next cycle. RISC-V processor, the pipeline registers between the datapath fetch and decoder stages is responsible for holding the instruction fetched from memory and passing it to the decoder stage. In this Verilog module, pc_in and instruction_in are the inputs from the datapath fetch stage, enableis a control signal indicating that the decoder stage is ready to accept the new instruction, pc_out and instruction_out are the outputs to the decoder stage, and clk and reset are the clock and reset signals, respectively.

The pc_reg and instruction_reg registers store the values of the program counter (PC) and instruction fetched from memory. The values are updated on the rising edge of the clock (clk) when enable is asserted. The stored values are then outputted to the decoder stage.

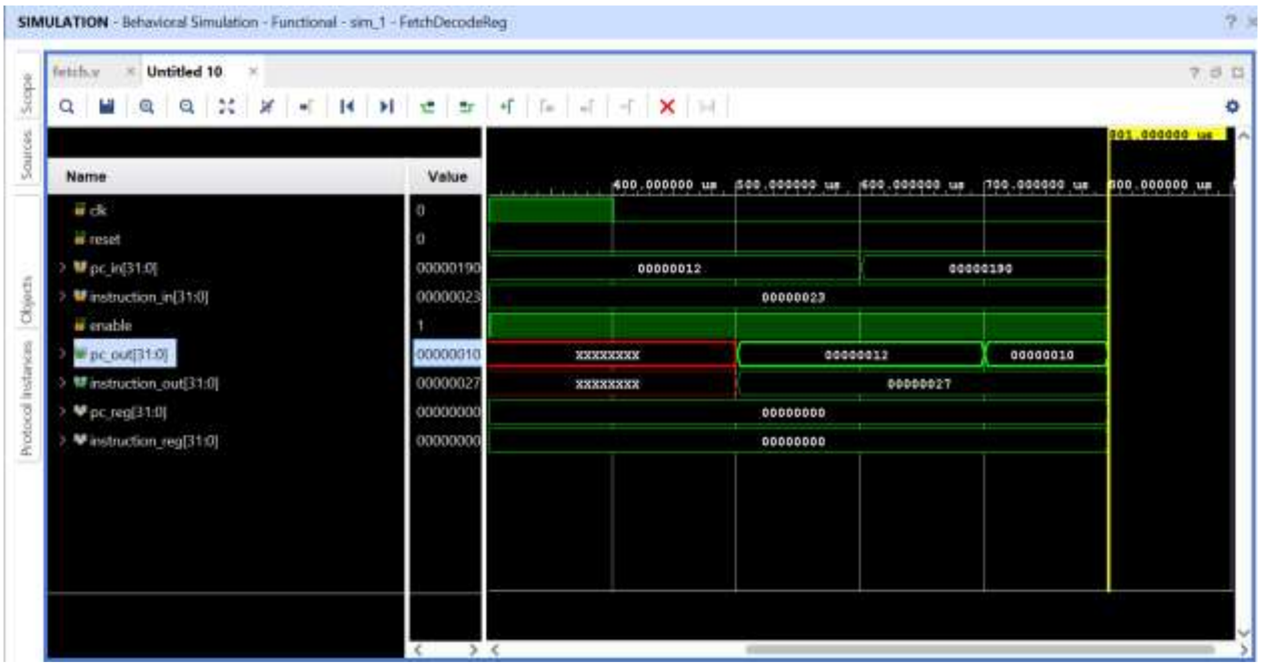


**Figure 4.2.9.2 Simulation waveform of IF/ID register**

*4.2.9.2 ID_IEx*

ID_IEx register located in between decode stage and execute stage. It helps to store read data (RD1, RD2) from the register file, extended immediate value (Imm Ext), Instruction [11:7] (rd), Instruction [19:15] (rs1), and Instruction [24:20] (rs2) from decode stage and load them to execute stage. Besides, PC and PCPlus4F from IF_ID register was also carried forward and stored to ID_IEx. in execute stage. It was noticed that the values from decode stage are loaded to the respective registers in execute stage in the next clock cycle. Therefore, the functionality of ID_IEx is verified.

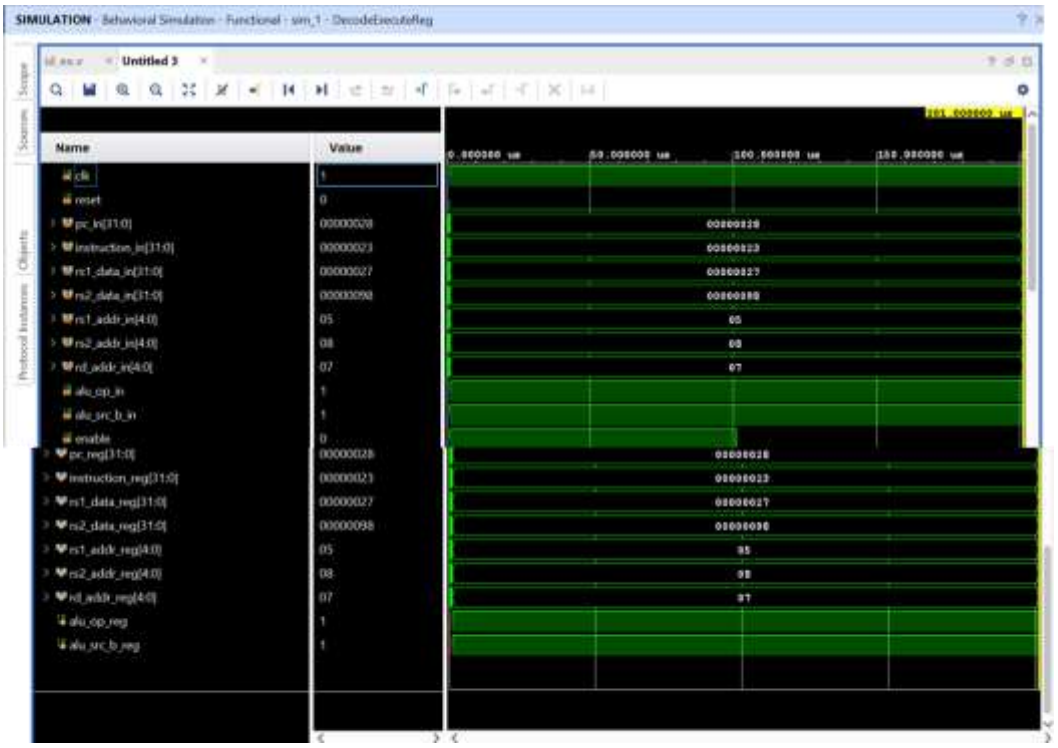
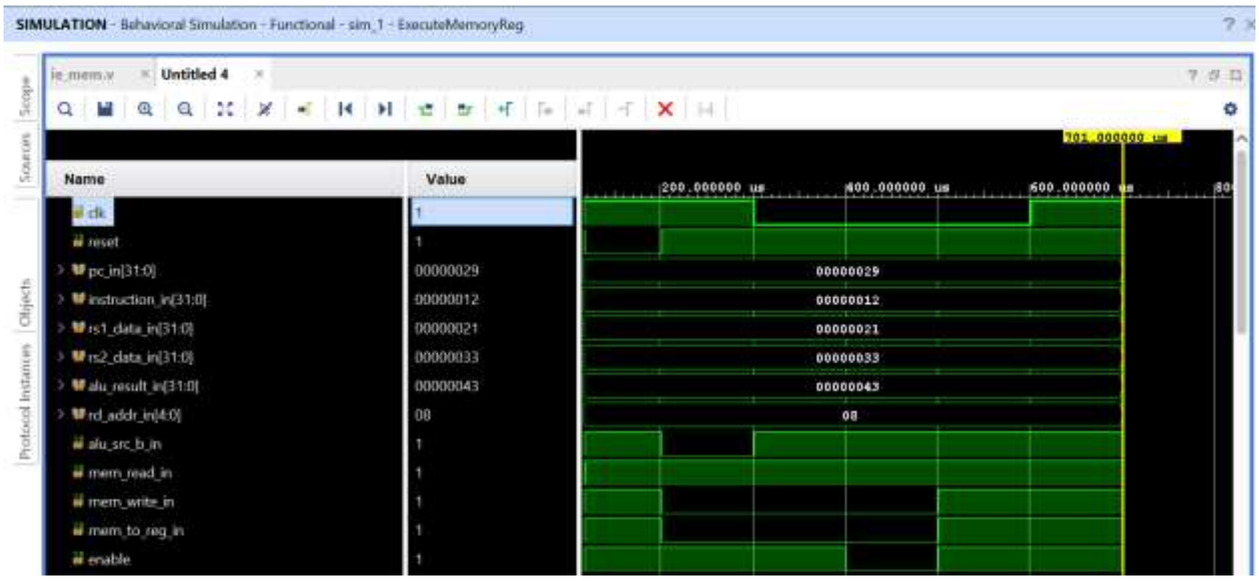


**Figure 4.2.9.2 Simulation waveform of ID/IEX register**

### *4.2.9.3 IEx_IMem*

IEx_IMem register located in between execute stage and memory stage. It helps to store the ALU results (ALUResults) and write data (WriteData) from execute stage and load them in memory stage. At the same time, In this module, Execute Memory Reg, we have added registers to store the values of the program counter (pc_reg), instruction (instruction_reg), data from source register 1 (rs1_data_reg), data from source register 2 (rs2_data_reg), ALU result (alu_result_reg), address of destination register (rd_addr_reg), ALU source B select signal (alu_src_b_reg), memory read signal (mem_read_reg), memory write signal (mem_write_reg), and memory to register signal (mem_to_reg_reg). These registers are updated on the rising edge of the clock (clk) when the enable signal is asserted.

The inputs to this module (pc_in, instruction_in, rs1_data_in, rs2_data_in, alu_result_in, rd_addr_in, alu_src_b_in, mem_read_in, mem_write_in, mem_to_reg_in) are connected from the outputs of the execute stage, and the outputs (pc_out, instruction_out, rs1_data_out, rs2_data_out, alu_result_out, rd_addr_out, alu_src_b_out, mem_read_out, mem_write_out, mem_to_reg_out) are connected to the inputs of the memory stage.



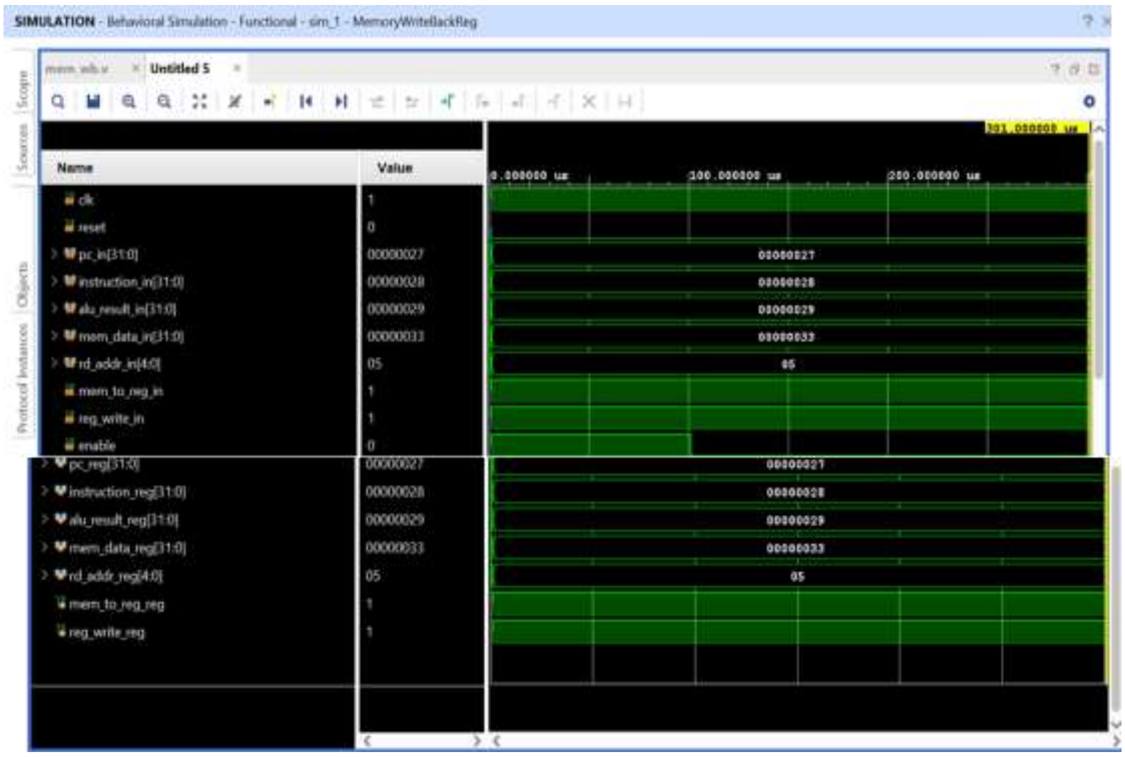
| Name | Value |
|---|---|
| clk | 1 |
| reset | 1 |
| pc_in[31:0] | 00000029 |
| instruction_in[31:0] | 00000012 |
| rs1_data_in[31:0] | 00000021 |
| rs2_data_in[31:0] | 00000033 |
| alu_result_in[31:0] | 00000043 |
| rd_addr_in[4:0] | 08 |
| alu_src_b_in | 1 |
| mem_read_in | 1 |
| mem_write_in | 1 |
| mem_to_reg_in | 1 |
| enable | 1 |

**Figure 4.2.9.3 Simulation waveform of IEX/IMem register**

### *4.2.9.4 IMem_IW*

IMem_IW is located in between memory stage and writeback stage. It helps to store the ALU results (ALUResults) from the alu and read data (ReadData) from data memory. In this module, MemoryWriteBackReg, we have added registers to store the values of the program counter (pc_reg), instruction (instruction_reg), ALU result (alu_result_reg), data read from memory (mem_data_reg), address of destination registers (rd_addr_reg), memory to register signal (mem_to_reg_reg), and register write signal (reg_write_reg). These registers are updated on the rising edge of the clock (clk) when the enable signal is asserted.

The inputs to this module (pc_in, instruction_in, alu_result_in, mem_data_in, rd_addr_in, mem_to_reg_in, reg_write_in) are connected from the outputs of the memory stage, and the outputs (pc_out, instruction_out, alu_result_out, mem_data_out, rd_addr_out, mem_to_reg_out, reg_write_out) are connected to the inputs of the write-back stage.



| Name | Value |
|---|---|
| clk | 1 |
| reset | 0 |
| pc_in[31:0] | 00000027 |
| instruction_in[31:0] | 00000028 |
| alu_result_in[31:0] | 00000029 |
| mem_data_in[31:0] | 00000033 |
| rd_addr_in[4:0] | 05 |
| mem_to_reg_in | 1 |
| reg_write_in | 1 |
| enable | 0 |
| pc_reg[31:0] | 00000027 |
| instruction_reg[31:0] | 00000028 |
| alu_result_reg[31:0] | 00000029 |
| mem_data_reg[31:0] | 00000033 |
| rd_addr_reg[4:0] | 05 |
| mem_to_reg_reg | 1 |
| reg_write_reg | 1 |

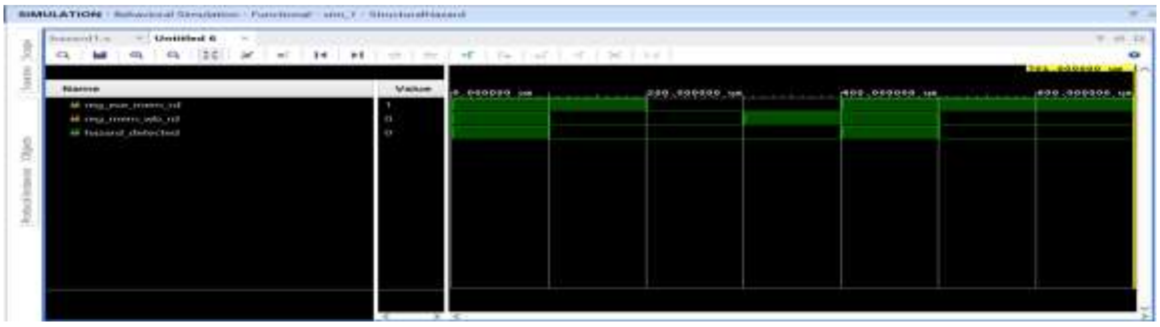
**Figure 4.2.9.4 Simulation waveform of IMem/IW register**

### *4.2.9.5 Hazard Unit (hazard unit)*

The hazard unit is used to detect situations where the sequential execution of instructions may lead to incorrect behavior due to data dependencies from the previous instruction. n a RISC-V processor, hazards refer to situations that can cause incorrect behavior or stalls in the pipeline. There are three main types of hazards: structural hazards, data hazards, and control hazards. Structural hazards occur when two instructions require the same hardware resource at the same time, leading to conflicts. For example, if one instruction needs to write to a register while another instruction is trying to read from the same register, a structural hazard occurs. This can be resolved by adding additional hardware resources or by inserting pipeline stalls.

Data hazards occur when an instruction depends on the result of a previous instruction that has not yet completed. This can happen in situations where the pipeline stages for different instructions overlap. For example, if one instruction writes to a register in the write-back stage while another instruction reads from the same register in the decode stage, a data hazard occurs. Data hazards are typically resolved using techniques such as forwarding (bypassing) or stalling the pipeline.

Control hazards occur when the flow of control is changed, such as with branch instructions. If a branch instruction is taken, the instructions following the branch may need to be flushed from the pipeline if they have already been fetched but are no longer needed. This can lead to performance penalties. Techniques such as branch prediction are used to mitigate the impact of control hazards by predicting whether a branch will be taken or not and fetching the correct instructions accordingly.


SIMULATION - Behavioral Simulation - Functional - sim_1 - StructuralHazard
Untitled 6
0.000000 us
200.000000 us
400.000000 us
600.000000 us

| Name | Value |
|---|---|
| reg_ex_mem_rd | 1 |
| reg_mem_wb_rd | 0 |
| hazard_detected | 0 |

**Figure 4.2.10.1 Structural hazard condition**

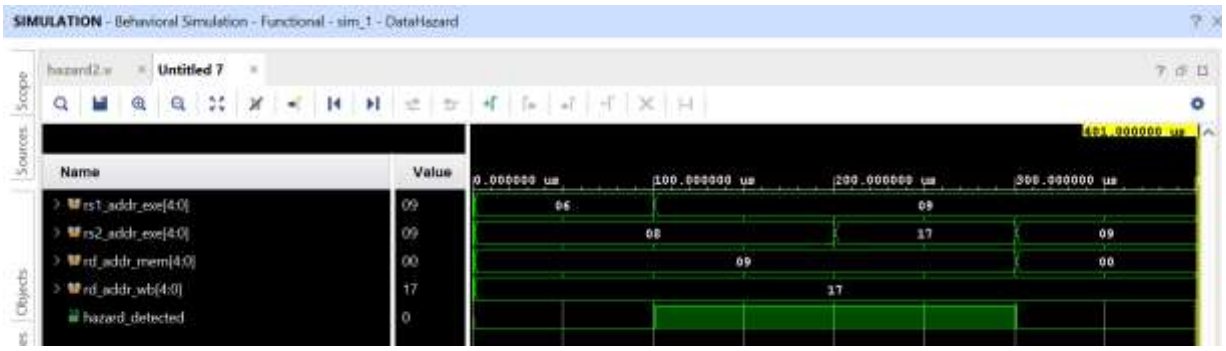


**Figure 4.2.10.3 data hazard condition**

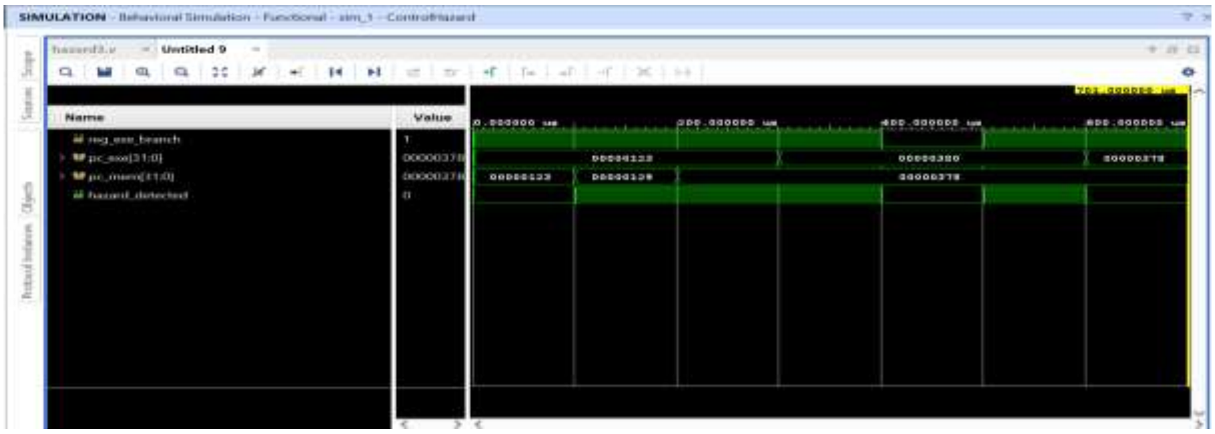


**Figure 4.2.10.3 control hazard condition**

### *4.3 Integrating RISC-V processor*

The RISC-V processor is developed by integrating all the verified modules together.The integration had 3 main units which namely controller unit, hazard unit, and datapath unit. The controller unit helped to send out control signals based on different instruction being executed. Besides, the hazard unit helped to detect hazard conditions during the execution and mitigated them. Moreover,the Datapath unit helped to carry information throughout the 5 pipeline stages in the Datapath. Then, the riscv_pip_27 module is integrated with the imem and dmem module to form the top design.

In a 5-stage pipelined RISC-V processor, the datapath unit, control unit, and hazard unit play crucial roles in ensuring correct operation and handling of hazards. Here's an overview of each:

**Datapath Unit:** The datapath unit is responsible for executing the operations specified by theinstructions in the pipeline. It consists of various components such as registers, ALU (ArithmeticLogic Unit), data memory, and multiplexers.

**Control Unit:** The control unit generates control signals that manage the operation of the datapath unit based on the current instruction. It decodes the instruction and generates signals to control the multiplexers, ALU operations, and other datapath elements.

**Hazard Unit:** The hazard unit detects and resolves hazards that occur due to pipeline dependencies. It identifies data hazards (when an instruction depends on the result of a previous instruction) and control hazards (when a branch instruction changes the flow of instructions).

The Verilog code for the datapath unit includes modules for registers, ALU, data memory, and multiplexers. Each module is instantiated and interconnected according to the RISC- V architecture.

The control unit is typically implemented as a finite state machine (FSM) in Verilog. It takes the opcode of the instruction as input and generates control signals for the datapath based on the current state of the FSM. The hazard unit detects hazards by comparing the current instruction with the instructions in the pipeline. It stalls the pipeline or inserts bubbles to resolve hazards.

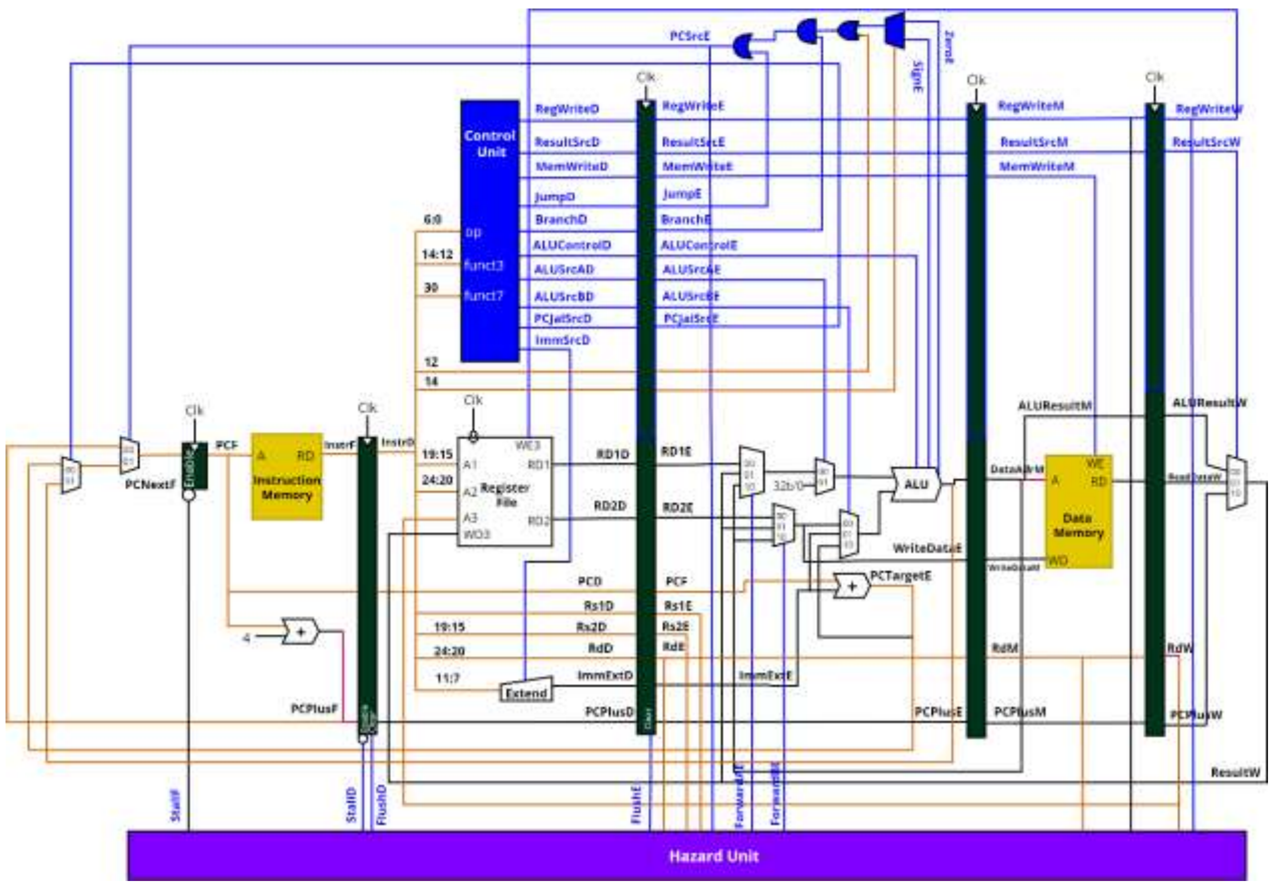


Figure 4.3: Schematic diagram of top design

## 5 Conclusion and references

### 5.1 Conclusion and Future work

In this project, a 32-bit 5 stage pipeline RISC-V processor is designed and implemented using Verilog coding. The top design consists of 3 main modules that are riscv_pip_27, imem and dmem. riscv_pip_27 module is an integrated module forming by control_unit, hazard_unit, and datapath_unit module. The control_unit module is used to send out control signals based on different instruction being executed. Then, the hazard_unit module is used to detect different hazard conditions during execution and mitigate them. Next, the datapath_unit module isused to carry information in the datapath through pipeline registers. The imem module is used to store the instruction of a program whereas the dmem is responsible for storing and retrieving data. Besides, several important modules such as alu, aludec, maindec, regfile, result_mux, pc_mux, pipeline register (IF/ID, ID/IEx, IEx/IMem, and IMem/IW designed and integrated in the processor.

**Table 5.1: Table of modules and their functionality**

| Module | Functionality |
|---|---|
| alu | Performing arithmetic and logical operations |
| aludec | Decode the instructions and receive signal from maindec to determine the type of operations that had the be performed |
| maindec | Generate control signal from the Opcode to determine types of instruction |
| regfile | Storing and manipulating data during program execution |
| result_mux | Select some inputs and give write_data as output |
| IF/ID | Store data from fetch stage and load them in decode stage |
| ID/IEx | Store data from decode stage and load them in execute stage |

| | |
|---|---|
| IEx/IMem | Store data from execute stage and load them in memory stage |
| IMem/IW | Store data from memory stage and load them in writeback stage |

The functionality of these modules were and verified analyzing the waveform generate by using Xilinx vivado software. Besides, and also design the overall top design and verify theoverall functionality of the 32-bit 5 stage pipeline RISC-V processor.

### *5.2 Reference*

1. D. Bhandarkar and D.W. Clark, "Performance from Architecture: Comparing a RISC and a CISC with Similar Hardware Organization,"Proceedings of the 4th Int'l. Conference on ASPLOS, Santa Clara, California, April 8-11, 1991.
2. Kulshreshtha, A., Moudgil, A., Chaurasia, A. and Bhushan, B., 2021, March. Analysis of 16-Bit and 32-Bit RISC Processors. In 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS) (Vol. 1, pp. 1318-1324). IEEE.
3. Khairullah, S.S., 2022, June. Realization of a 16-bit MIPS RISC pipeline processor. In 2022 International Congress on Human-Computer.
4. M. N. Topiwala and N. Saraswathi, "Implementation of a 32-bit MIPS based RISC processor using Cadence," 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, 2014.
5. Islam, S., Chattopadhyay, D., Das, M.K., Neelima, V. and Sarkar, R., 2006, September.Design of High-Speed-Pipelined Execution Unit of 32-bit RISC Processor. In 2006 Annual IEEE India Conference (pp. 1-5). IEEE.
6. S. P. 6. Ritpurkar, M. N. Thakare and G. D. Korde, "Design and simulation of 32-Bit RISC architecture based on MIPS using VHDL," 2015 International Conference on Advanced Computing and Communication Systems, 2015.Interaction, Optimization and Robotic Applications (HORA) (pp. 1-6). IEEE.
7. Al-sudany, S.M., Al-Araji, A.S. and Saeed, B.M., 2021. FPGA-Based Multi-Core MIPS Processor Design. IRAQI JOURNAL OF COMPUTERS, COMMUNICATION, CONTROL & SYSTEMS ENGINEERING, 21(2).
8. Wang, W., Han, J., Cheng, X. and Zeng, X., 2021. An energy-efficient cryptoextension design for RISC-V. Microelectronics Journal, 115, p.105165.