NEW HORIZON COLLEGE OF ENGINEERING

Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC

Accredited by NAAC with 'A' Grade, Accredited by NBA

DESIGN AND IMPLEMENTATION OF 32-BIT MIPS RISC PROCESSOR WITH FLEXIBLE 5-STAGE PIPELINING AND DYNAMIC THERMAL CONTROL

A MAJOR PROJECT REPORT

Submitted by

LOKESH M	**1NH19EC065**
TEJAS C GHORPADE	**1NH19EC119**
MANOJ KUMAR S	**1NH19EC073**
NISHANTH H	**1NH19EC085**

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

NEW HORIZON
COLLEGE OF ENGINEERING
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

Certified that the Major project work entitled DESIGN AND IMPLEMENTATION OF 32-BIT MIPS RISC PROCESSOR WITH FLEXIBLE 5-STAGE PIPELINING AND DYNAMIC THERMAL carried out by **Mr.LOKESH M USN: 1NH19EC065, Mr.TEJAS C GHORPADE USN: 1NH19EC119, Mr.NISHANTH USN:1NH19EC085, Mr.MANOJ KUMAR S USN: 1NH19EC073** bona fide students of New Horizon College of Engineering in partial fulfilment for the award of Bachelor of Engineering in Electronics and Communication Engineering of the Visveswaraya Technological University, Belagavi during the year 2022-2023. It is certified that all corrections/suggestions indicated for Assessment have been incorporated in the Report deposited in the departmental library. The Final-Report has been approved as it satisfies the academic requirements in respect of Major project work prescribed for the said Degree.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

We have great pleasure in expressing gratitude to **Dr. Mohan Manghnani**, Chairman, New Horizon Educational Institutions, for providing necessary infrastructure and creating good environment.

We take this opportunity to express our profound gratitude to **Dr. Manjunatha,** Principal, New Horizon College of Engineering, for his constant support and encouragement.

We would like to thank **Dr. R J Anandhi,** Professor and Dean-Academics, NHCE, for his valuable guidance.

We would also like to thank **Dr. Aravinda K** Professor and HOD, Department of Computer Science and Engineering, for her constant support.

We also express our gratitude to **Mr. Avinash N J**, our project guide, for constantly monitoring the development of the project and setting up precise deadlines. His / Her valuable suggestions were the motivating factors in completing the work.
Finally, a note of thanks to all the teaching and non-teaching staff of Dept of Electronics and Communication Engineering for their cooperation extended to us, and our parents and friends, who helped us directly or indirectly in the course of the project work.

LOKESH M (1NH19EC065)

TEJAS C GHORPADE (1NH19EC119)

NISHANTH H (1NH19EC085)

MANOJ KUMAR S (1NH19EC073)

ABSTRACT

The main aim of this design is designing a 5-stage flexible pipelined 32-bit RISC-V processor using system Verilog including it Dynamic thermal management technique. This Design is be based on MIPS instruction set architecture (ISA) in which stages of pipeline include Instruction fetch, Instruction Decode, Execute, Memory access, Write back. The RISC [Reduced Instruction set Computed] compared to CISC [Complex instruction set computer] executes instructions in one clock and instructions are always uniformly lengthened and fixed instruction format-based opcode. Hence RISC is preferable more used for less complex tasks and low power designs. Based on the literature the proposed design brings significant improvements in simulation speeds due to flexible pipeline technique and low power consumption is achieved. The Primary desire of this paper is to simulate and synthesize the design by perform basic ALU operations and provide desired outputs.

Keywords: RISC, CISC, MIPS, ALU, ISA.

CONTENTS

CHAPTER 1

1.1 INTRODUCTION

RISC [Reduced Instruction Set Computers] are extensively used in all type of computational tasks especially in the area of basic scientific computing such as DSP and DIP etc. There are a variety of processor types on the market, including some of the processors that come with Hardware Description Language (HDL) like VHDL (Very High-speed Integrated Circuit HDL) and Verilog-HDL is used to write a specific type of processor. RISC (Reduced Instruction Set Computer) is an efficient computer architecture which is used for high-speed, low-power applications [1]. The processor uses the reduced instruction set (RISC) to allow pipelined execution of instruction, increasing performance and throughput performance. We included Dynamic thermal management unit using dynamic frequency scaling for thermal stability of processor. The processor can work at a high clock frequency and thus yields higher speed. More recently Verilog is used as an input for synthesis programs which will generate a gate-level description for the circuit. The RISC-V architecture comprises of load and store operation. This means it has instruction called LOAD for accessing data from the memory and STORE instruction for write the data back to memory [14]. The architecture is called Hardwired architecture and it will increase the overall performance of RISC processor. Our design is based on synchronous technique which is fast and reliable technique in the present industry processor.

CHAPTER 2

2.1 LITERATURE SURVEY:

"Design of 32-bit Processor for Embedded System" designed and developed by Hyun Woo Oh, Kwon Neung Cho and Seung Eun Lee, they built a processor for the embedded system compatible with MIPS ISA. The Processor has the Data Forwarding unit and Stall unit for better performance. The processor was tested with Dhrystone MIPS Benchmark and got result which was measured at 27.71 at 50MHZ operation

"Dynamic Thermal Management for High Performance Processor" designed and developed by David Brooks Department of Electrical Engineering Princeton University dbrooks. They did research On Clock Frequency scaling and resulting in finding it trades a linear Performance loss for a linear power savings.

"Design and Analysis of 16- Bit RISC Processor using Low Power", designed and developed by P. Brundavani, Dr. D. Vishnu Vardhan, P. Mahesh3, R. Suresh. The 16-bit RISC Processor using low power pipeline is designed and simulated using Xilinx 14.3 version software and power analysis was done using X Analyzer.

"ASIC Design of MIPS RISC Processor for High Performance", by Agneti Ashok, V.Ravi this design included hazard detection unit and inter locked pipeline stages and ALU forwarding unit, the main aim of this design to remove hazards in the data path and provide accurate results from pipeline registers . It also made sure processor didn't got to the high impedance or unknown state and it resulted in high performance

"Design and Implementation of 32-Bit RISC Processor with Five stage pipeline" by Neha Dwivedhi and Pradeep Chhawcharia, the paper included a pipeline processor with single cycle architecture which executes single instructions one after the other simultaneously and improving throughout speed of output. All modules were coded in Verilog HD

"Design of Low Power Pipelined RISC Processor" designed by Indu.M and Arun Kumar.M, this paper presented the design and implementation of a 32-bit RISC processor with low power pipeline. The low power technique was proposed in front end process only. The clock gating technique was used to minimize power consumption core RISC. Dynamic power consumption was monitored by using software Altera powerplay analyzer.

CHAPTER 3

3.1 PROPOSED SYSTEM:

Our System mainly focused on flexible pipelining and dynamic thermal control which in turn provides low power consumption and faster performance speed of the processor. Main features include 32-Bit MIPS (Microprocessor without Interlocked Pipeline Stages) Architecture based Processor, Synchronous Pipeline Implementation. Supports 48 Instructions. RISC architecture used follows single-cycle instruction execution. Low power and high-performance pipelining technique is being used.[1]

It majorly Contains three addressing modes:

Immediate mode: In this addressing mode the operand is explicitly specified.

Register-Register mode: this mode defines the transfer of data between two registers.

Memory Mode: This mode represents the data I/O between external program and ram memory.[5]

3.11 Pipeline technique:

Pipeline can be defined as processing a set of data elements in series order having connections where one output of one processed element is input to next new element. This technique comes under Harvard Architecture which is known for its speed and performance.[4]

1. Instruction fetch (IF): The processor reads the next Instruction to be executed.

2. Instruction decode (ID): Processor works out what this Instruction is.

3. Instruction execution (EX): The processor executes Instruction on operand values.

4. Memory Access (MEM): The Processor will access the input and output memory.

5. Write Back (WB): Processor writes the result of the operation in the register/memory.

3.12 MIPS: [Microprocessor without Interlocked Pipeline Stages] architecture:

Five-stage of execution pipeline: fetch, decode, execute, memory-access, write-result. Regular instruction set which means all instructions are 32-bit. Three-operand arithmetical and logical instructions.32 general-purpose registers of 32-bits each are present. No status register or instruction side-effects. No complex instructions (like stack management instruction, string operations, etc. [6]

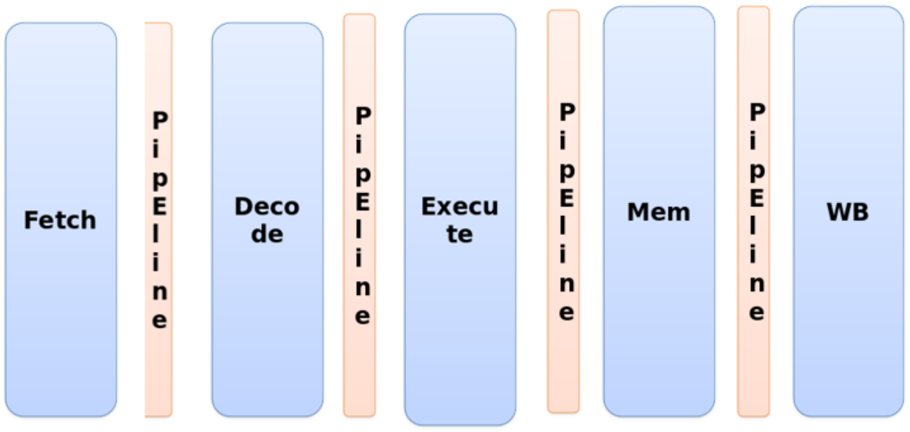


Fig.1. Block Diagram of pipeline technique

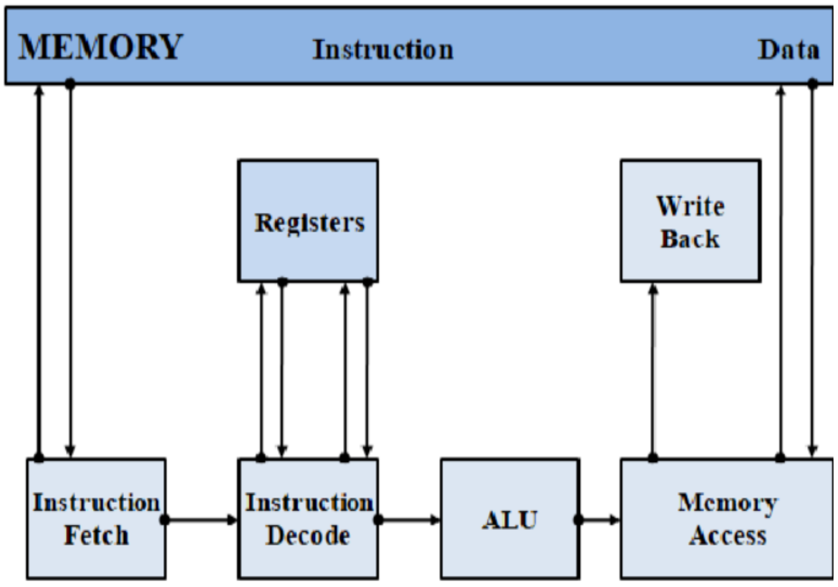


Fig.2. Block Diagram of MIPS Architecture

memory, data memory and register memory. In the IF stage the 32-bit Instruction is loaded to the instruction memory by using Program counter (PC). After instruction is fetched the PC will be incremented by one and next stage is pipelined.[4]

The next stage includes decode stage in which 32-bit instruction is decoded based upon Opcode which is 6-bit. The Opcode is sent to ID stage with appropriate control signals will generated.

Following stage include a ALU unit which will perform all the necessary operations as per the decoded information. In MEM stage the operations like load and store will access the memory required. The Last Stage include WB which write operation back to memory [9]

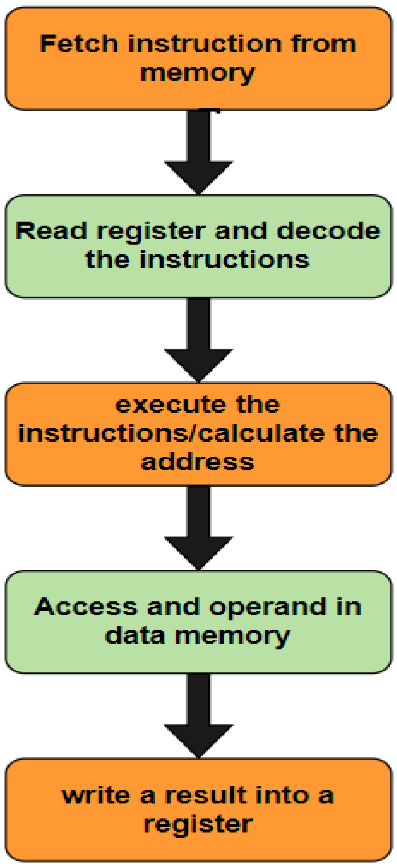

Fetch instruction from memory
Read register and decode the instructions
execute the instructions/calculate the address
Access and operand in data memory
write a result into a register

Fig.3. Flowchart of Instruction flow.

3.13 Flexible Pipelining:

The pipeline is made such a way that it can flexibly to switch between 4 stage and 5 stage with based on instruction type. Thus, by reducing the data dependency the processing time is reduced. This feature was included to enhance the overall speed of the processor based with less simulation time.

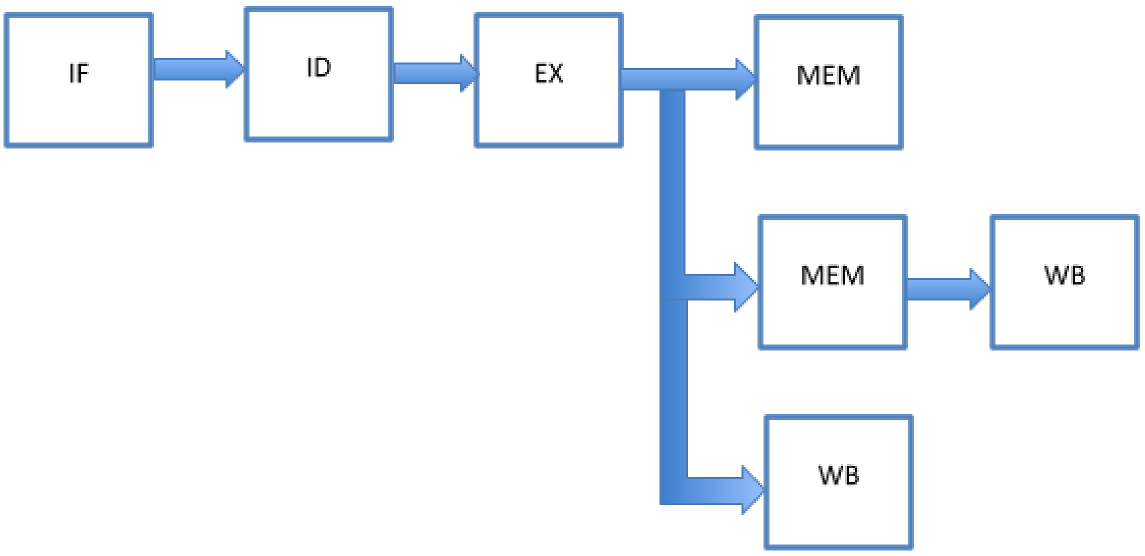


Fig.4. Block Diagram for Flexible Pipeline

3.14 Dynamic Thermal Control:

This feature includes clock frequency scaling which essentially trades a linear performance loss for a linear power savings which over improves the performance and power consumption. The technology used for clock frequency scaling is Transmeta's Long Run technology it performs both clock and voltage scaling dynamically.[2][10]

3.15 Conceptual Design using Logisim:

The Design for 32-RISC is designed in Logisim software for basic simulation by using program memory and RAM memory. This brings further understanding for Processor implantation.

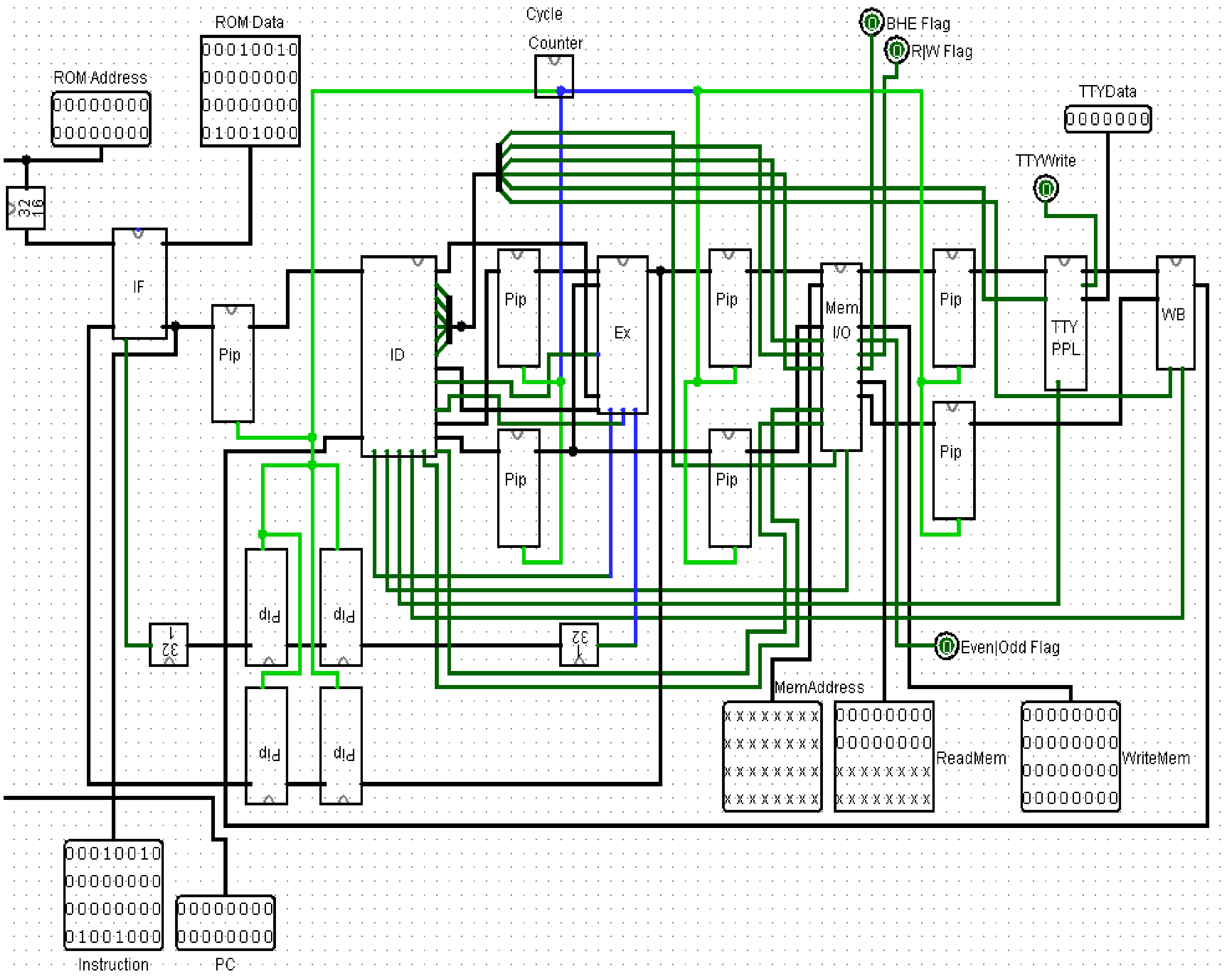


Fig.5. Design Created in Logisim

CHAPTER 4

4.1 SOFTWARE SPECIFICATION:

4.11 EDA Playground:

The EDA playground is website which gives students and engineers immediate hands-on exposure for simulation and synthesis for System Verilog, Verilog, C++ and Other HDLs. with a simple click we can run the code and see the console output in real time. For simulation it provides EP Wave which is a browser-based wave viewer. The main feature of this website it provides selecting paid services of Cadence Synopsis and Mentor Graphics for free of cost. We can get immediate results for our design. We used Cadence Xcelium 20.9 for simulation of the design

4.12 Yosys Open Synthesis Suite:

yosys is an open-source framework for RTL synthesis. This software at present has extensive verilog-2005 support. It provides all the basic set of synthesis algorithms for various designs. Yosys is a software which is controlled using synthesis scripts.[11]

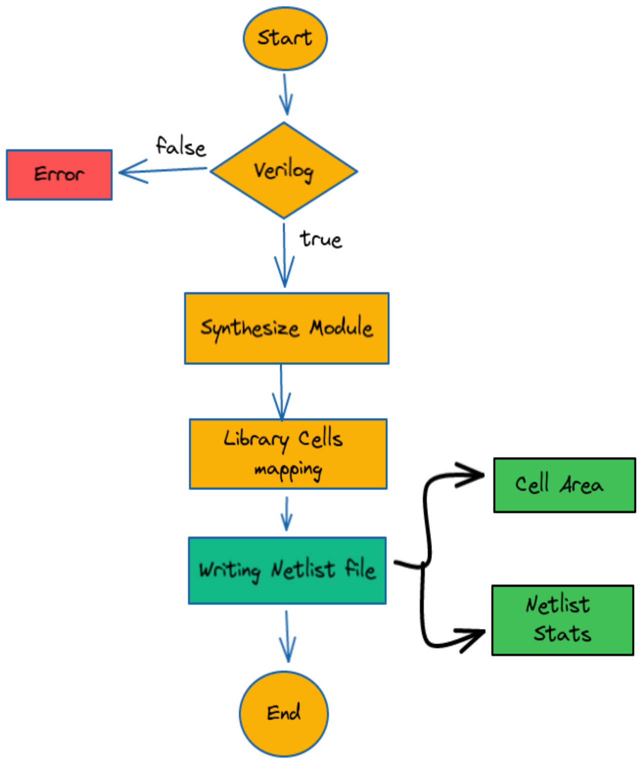


Fig.6. Flowchart for yosys synthesis flow

4.13 Logisim:

Logisim is an open-source software tool used for designing and simulating digital logic circuits. It is developed by Carl Burch and is available for Windows, macOS, and Linux operating systems. Logisim provides a graphical user interface that allows users to drag and drop components, such as logic gates, flip-flops, and multiplexers, onto a design canvas and connect them with wires to create complex digital circuits. The software also features a built-in simulator that allows users to test their circuits and see the output based on the input provided. Additionally, Logisim supports hierarchical design, allowing users to create sub-circuits and reuse them in larger designs. Overall, Logisim is a useful tool for learning and experimenting with digital circuits and logic design.

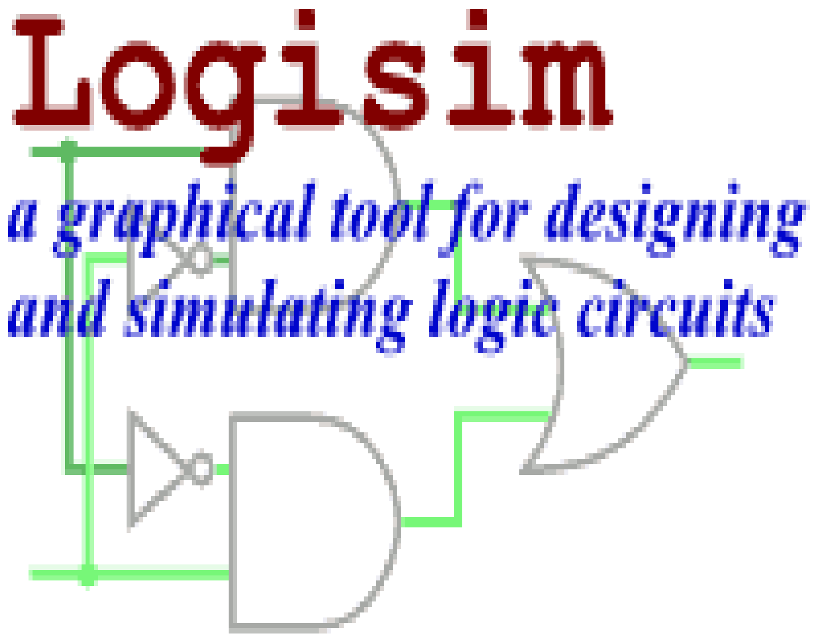


Fig:7 Logisim

Logisim has a user-friendly graphical user interface that allows users to create, edit, and simulate digital circuits easily. The software provides an extensive library of digital components such as gates, flip-flops, and multiplexers that users can use to create their circuits.

4.2 PROCESSOR DESIGN IN LOGISIM

4.21 Designing Processor in Logisim:

Designing a processor in Logisim involves a multi-step process that requires a solid understanding of digital logic circuits and computer architecture. Here are some general steps that may be involved in designing a processor in Logisim:

1. Define the instruction set architecture (ISA) of the processor, including the instruction formats, registers, and supported operations.
2. Implement each instruction as a combinational logic circuit or a sequence of combinational and sequential logic circuits.
3. Assemble the individual instruction circuits into a control unit that can decode the instruction and generate the appropriate control signals to execute the instruction.
4. Implement the data path of the processor, which includes the registers, arithmetic logic unit (ALU), and memory interface.
5. Connect the control unit and data path circuits to create the complete processor design.
6. Test the processor design using various inputs and verifying that the outputs match the expected results.
7. Optimize the processor design by reducing the number of gates used and increasing its speed and efficiency.

Designing a processor in Logisim requires a high level of technical expertise and may take a significant amount of time and effort. However, Logisim provides a powerful set of tools that can assist with the design process and facilitate the creation of complex digital circuits.

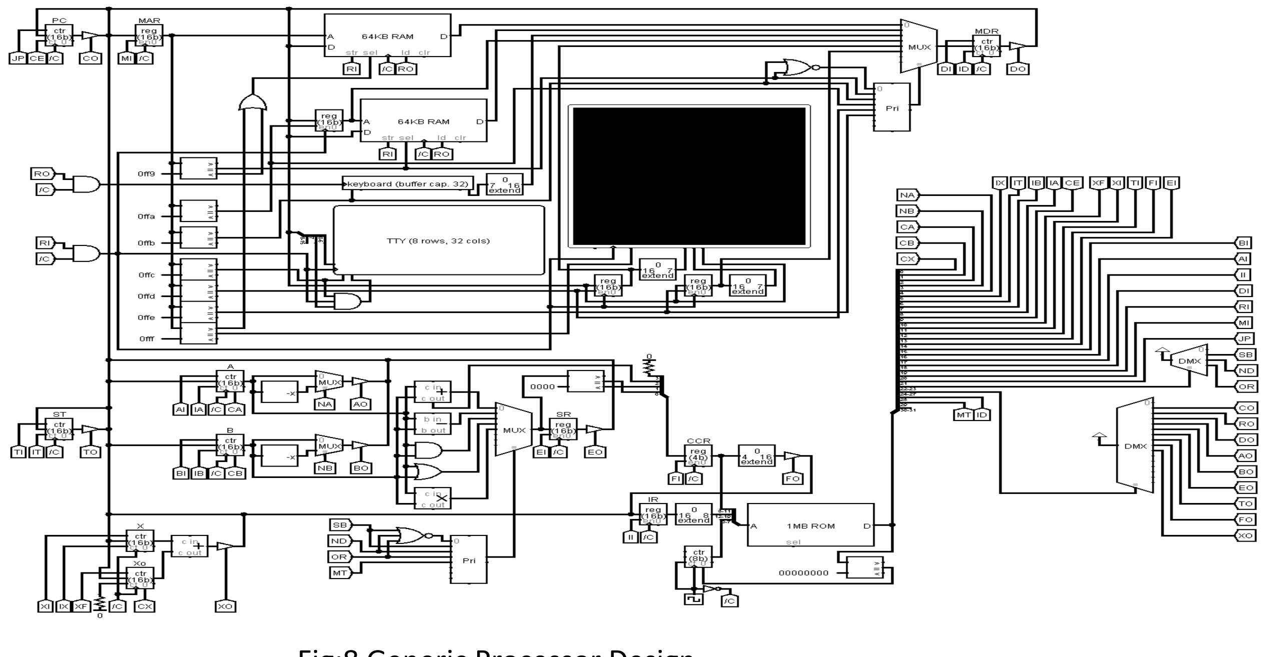


Fig:8 Generic Processor Design

4.22 Processor Instruction set Architecture:

The Instruction Set Architecture (ISA) of a 32-bit RISC processor with a 5-stage pipeline typically includes the following characteristics:

1. Fixed instruction length: The ISA uses a fixed-length instruction format, usually 32 bits, which simplifies the instruction decoding process.

2. Load/store architecture: The processor only operates on data that is stored in registers, and all memory accesses are performed using explicit load and store instructions.

3. Register file: The processor has a register file that contains a fixed number of general-purpose registers, usually 32, which can be accessed by the instructions.

4. Arithmetic and logic operations: The ISA supports a variety of arithmetic and logic operations, such as add, subtract, multiply, divide, AND, OR, and XOR, which are performed by the Arithmetic Logic Unit (ALU).

5. Branching and control flow: The ISA includes instructions for branching and control flow, such as conditional and unconditional jumps, subroutine calls, and returns.

6. Pipelining: The processor is divided into five stages, namely, Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM), and Write Back (WB). Each stage is responsible for performing a specific operation, and multiple instructions can be executed simultaneously in different stages of the pipeline, improving the processor's overall performance.

7. Reduced instruction set: The ISA is a Reduced Instruction Set Computing (RISC) architecture, which means that it has a smaller instruction set than traditional Complex Instruction Set Computing (CISC) architectures, resulting in faster execution times and simpler hardware design.

Overall, the ISA of a 32-bit RISC processor with a 5-stage pipeline is designed to provide high performance, low power consumption, and efficient use of hardware resources.

4.23 Processor Instruction Implementation:

Implementing each instruction in a 32-bit RISC processor design in Logisim involves breaking down each instruction into its constituent parts, including the opcode, source and destination registers, immediate values, and memory addresses. Here are the general steps involved in implementing an instruction in Logisim:

1. Determine the opcode of the instruction: The opcode determines the type of operation that the instruction performs, such as add, subtract, load, store, jump, or branch. The opcode is usually located in the first few bits of the instruction.

2. Determine the instruction format: The instruction format specifies the layout of the instruction, including the positions of the opcode, source and destination registers, immediate values, and memory addresses. The instruction format may be fixed or variable length, depending on the architecture.

3. Decode the instruction: The decoder circuit uses the opcode and instruction format to determine the specific operation that the instruction performs and the operands that it uses.

4. Calculate the effective address: If the instruction involves a memory access, such as a load or store operation, the effective address must be calculated by adding the base address and any offset or displacement specified in the instruction.

5. Perform the operation: The ALU performs the arithmetic or logic operation specified in the instruction, using the source registers, immediate values, or memory contents as operands.

6. Write back the result: If the instruction involves a register write-back, the ALU result is written back to the destination register.

7. Update the program counter: If the instruction involves a branch or jump, the program counter is updated to point to the target instruction.

8. Implement any pipeline registers: In a pipelined design, the results of each stage must be stored in pipeline registers before being passed to the next stage.

Overall, implementing each instruction in a 32-bit RISC processor design in Logisim requires a combination of combinational and sequential logic circuits, including decoders, multiplexers, adders, ALUs, registers, and pipeline registers. The exact implementation will depend on the specific instruction set architecture and design choices made by the designer.

4.24 Assembling the Instruction Circuits into a Control unit:

Assembling the individual instruction circuits into a control unit in Logisim involves creating a combinational logic circuit that can decode the instruction and generate the appropriate control signals to execute the instruction. Here are the general steps involved in assembling the instruction circuits into a control unit:

1. Identify the instruction fields: The control unit must decode the instruction and extract the relevant fields, such as the opcode, source and destination registers, immediate values, and memory addresses. These fields are usually located at fixed positions within the instruction.

2. Create the opcode decoder: The opcode decoder is a combinational logic circuit that determines the specific operation that the instruction performs based on its opcode. The decoder uses the opcode to generate control signals that activate the appropriate functional units in the processor, such as the ALU, memory interface, or control flow unit.

3. Create the register file decoder: The register file decoder is a combinational logic circuit that determines which registers are used as sources and destinations for the instruction. The decoder uses the source and destination register fields to generate control signals that select the appropriate registers from the register file.

4. Create the immediate value decoder: If the instruction involves an immediate value, the immediate value decoder is a combinational logic circuit that extracts the value from the instruction and generates a control signal that passes the value to the functional unit that requires it.

5. Create the memory address decoder: If the instruction involves a memory access, such as a load or store operation, the memory address decoder is a combinational logic circuit that calculates the effective address and generates a control signal that passes the address to the memory interface.

6. Generate the control signals: Once the relevant fields have been extracted and decoded, the control unit generates the appropriate control signals that activate the functional units and direct the data flow in the processor. The control signals are usually generated as a set of binary values that are passed to the functional units as input.

7. Implement the pipeline registers: In a pipelined design, the control signals must be stored in pipeline registers between each stage of the pipeline to ensure that the correct signals are available at the correct time.

Overall, assembling the individual instruction circuits into a control unit in Logisim requires a solid understanding of digital logic circuits and computer architecture, as well as careful attention to detail to ensure that the control signals are generated correctly for each instruction.

4.25 Implement the data path of the processor:

Implementing the data path of a processor in Logisim involves creating the functional units that perform the arithmetic, logic, and memory operations required by the instruction set architecture. Here are the general steps involved in implementing the data path of a processor:

1. Create the register file: The register file is a set of registers that hold the operands and results of the arithmetic and logic operations. The register file is typically implemented as a bank of flip-flops with read and write ports. The read ports are used to select the source operands for the ALU, while the write ports are used to update the register values.

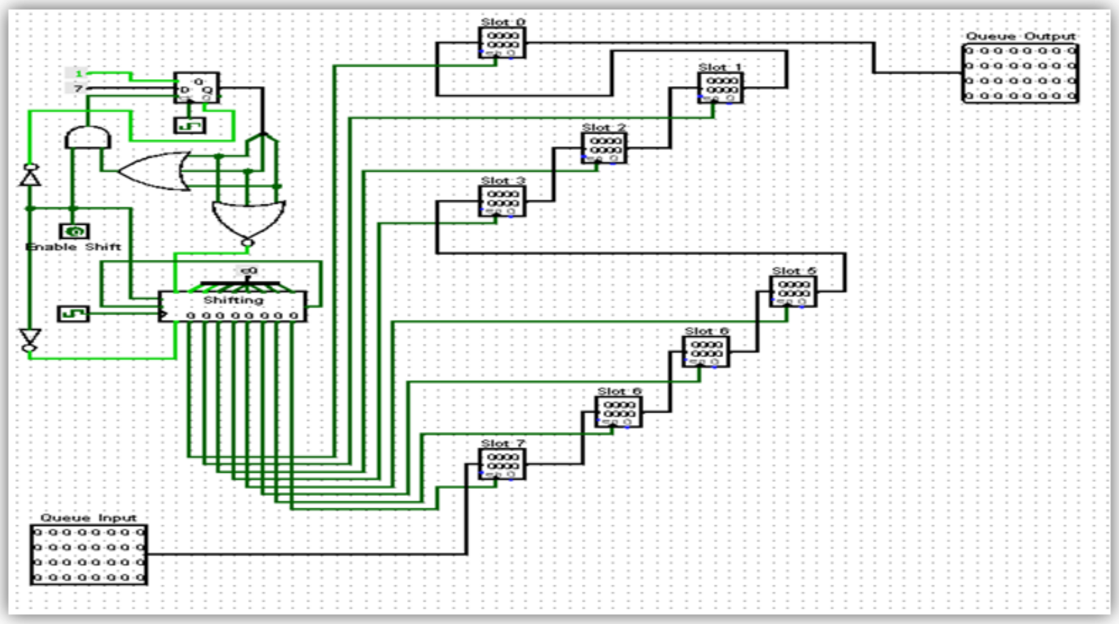


Fig:9 Register Design

2. Create the ALU: The ALU is a combinational logic circuit that performs the arithmetic and logic operations specified by the instruction set architecture. The ALU typically includes adders, subtractors, and logic gates, as well as control inputs that select the operation to be performed.

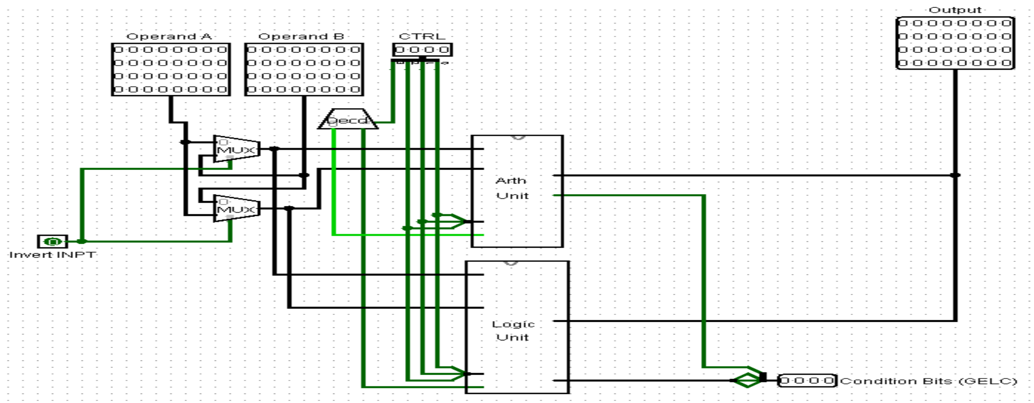


Fig:10 ALU Design

3. Create the memory interface: The memory interface is a set of circuits that enable the processor to access the main memory. The memory interface includes an address bus, a data bus, and control signals that enable the processor to read or write data to and from the memory.

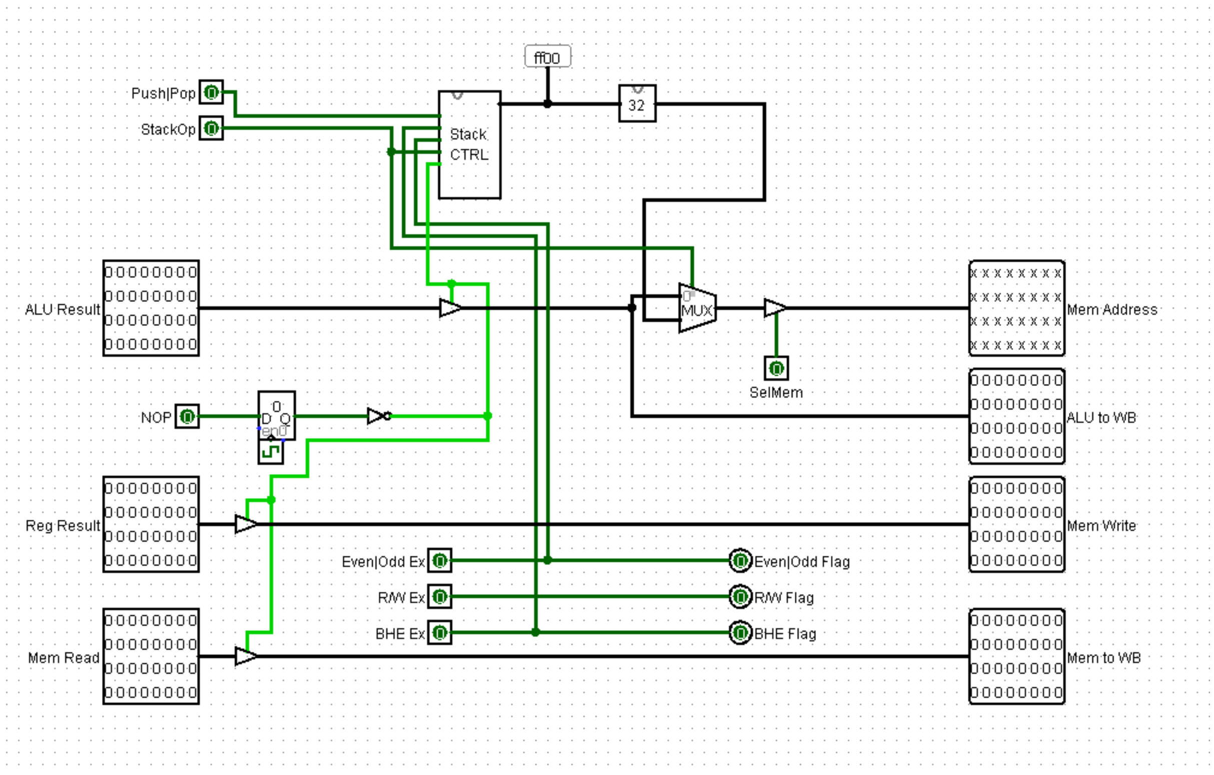


Fig:11 Memory Interface

4. Create the multiplexers: Multiplexers are combinational logic circuits that select between two or more input signals based on a control signal. Multiplexers are used in the data path to select the source operands for the ALU and to select between the ALU result and the memory data during a load operation.

5. Implement the pipeline registers: In a pipelined design, the results of each stage must be stored in pipeline registers before being passed to the next stage. Pipeline registers are typically implemented as banks of flip-flops that store the results of each stage until they are needed by the next stage.

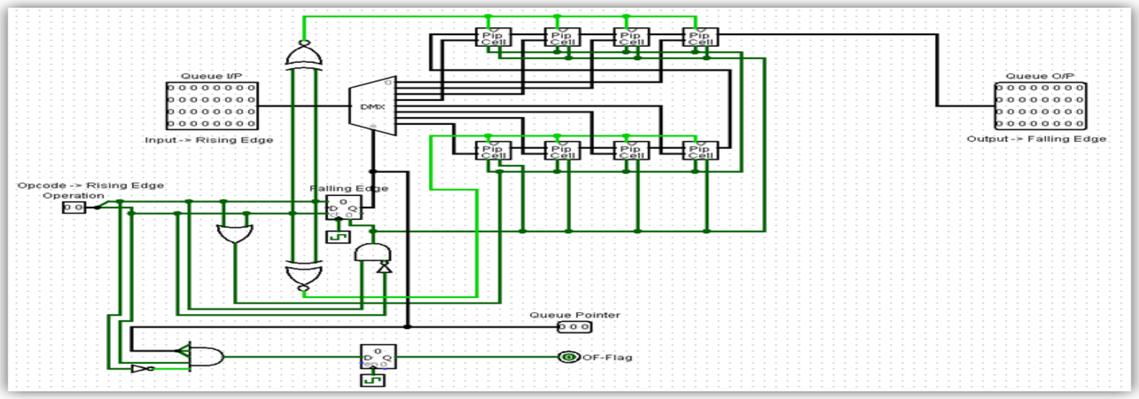


Fig:12 Pipeline Registers

6. Connect the functional units: The functional units are connected together using the appropriate data paths and control signals. The register file is connected to the ALU and memory interface using multiplexers, while the ALU is connected to the memory interface using the address bus and data bus. Control signals are used to activate the appropriate functional units and direct the data flow through the processor.

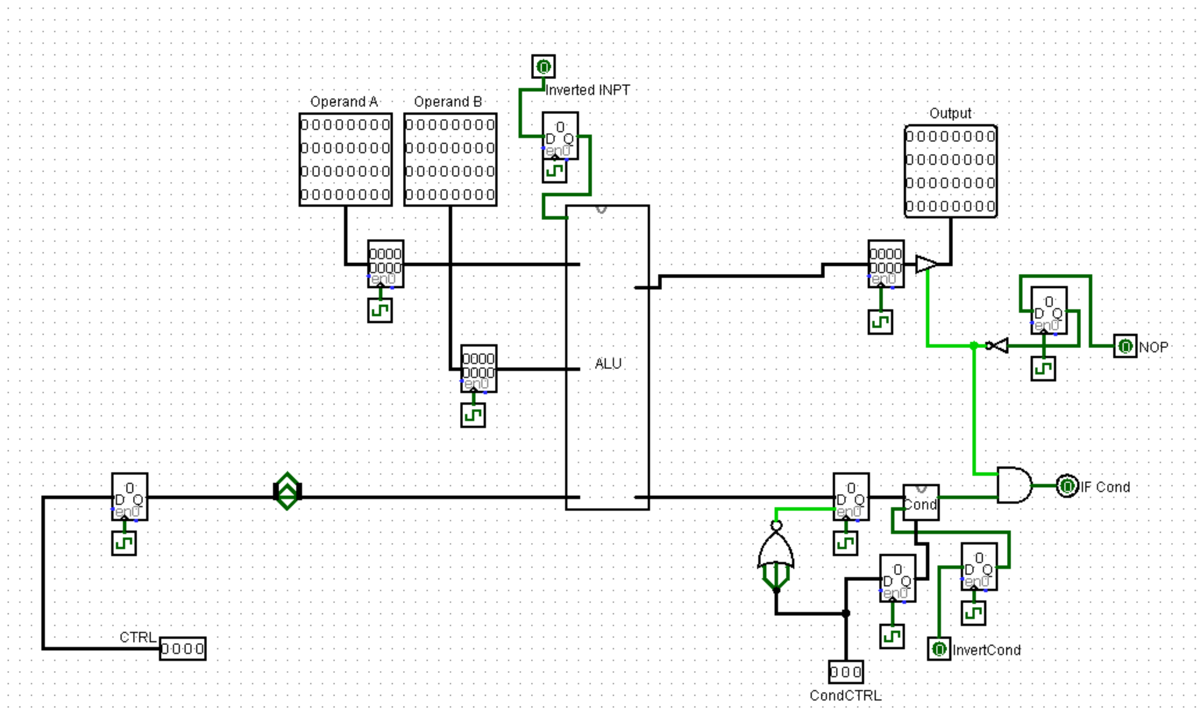


Fig:13 Functional unit Connections

Overall, implementing the data path of a processor in Logisim requires a solid understanding of digital logic circuits and computer architecture, as well as careful attention to detail to ensure that the functional units are connected together correctly and that the data flows through the processor as intended.

4.26 Program and RAM Memory :

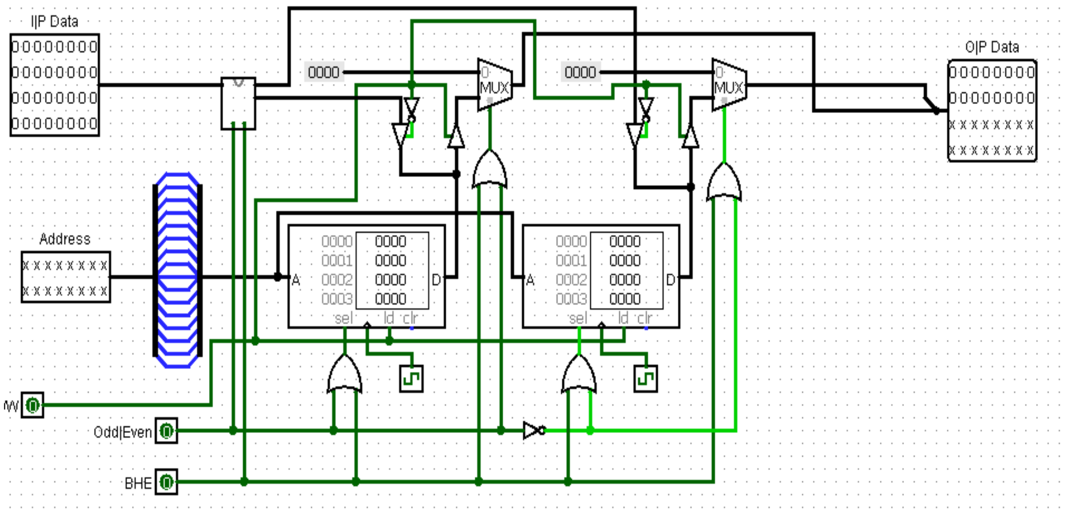


Fig:14 Memory Design

Creating the program and RAM memory for a 32-bit RISC processor with a 5-stage pipeline in Logisim involves the following steps:

1. Open Logisim: Open Logisim on your computer and create a new blank circuit.

2. Add the RAM memory: From the left-hand side menu, select the "Memory" category and drag the "RAM" component onto the canvas. Double-click on the RAM component to open its configuration dialog. Set the memory size to the desired value (e.g., 256 words of 32 bits each) and select the appropriate data width and address width.

3. Add the program memory: From the left-hand side menu, select the "Memory" category and drag the "ROM" component onto the canvas. Double-click on the ROM component to open its configuration dialog. Set the memory size to the desired value (e.g., 256 words of 32 bits each) and select the appropriate data width and address width. Enter the program instructions into the ROM using the "Edit Contents" button in the configuration dialog.

4. Connect the memories to the processor: Add the processor circuit to the canvas and connect the output port of the program memory to the instruction input port of the processor. Connect the input and output ports of the RAM memory to the appropriate input and output ports of the processor.

5. Save the circuit: Save the circuit by selecting "Save As" from the "File" menu and choosing a file name and location.

6. Test the circuit: To test the circuit, simulate the execution of a sequence of instructions using the "Simulate" mode in Logisim. Apply a sequence of instructions to the input port of the processor and monitor the output ports to ensure that the correct results are generated.Overall, creating the program and RAM memory for a 32-bit RISC processor with a 5-stage pipeline in Logisim requires careful attention to detail to ensure that the memories are configured correctly and connected to the processor in the right way. The design must also be thoroughly tested to ensure that it operates correctly for a wide range of instructions and input data

4.27 Other Important Tools:

1 .Program Counter and Instruction Display:

In Logisim, the Program Counter (PC) and RAM address can be displayed using the HEX digit Display component. This component is used to display hexadecimal values on the screen.

To display the value of the PC or RAM address in hexadecimal format, you can follow these steps:

1. Add the HEX digit Display component: From the left-hand side menu, select the "Base" category and drag the "HEX digit Display" component onto the canvas.

2. Configure the HEX digit Display: Double-click on the HEX digit Display component to open its configuration dialog. Set the number of digits to the desired value (e.g., 8 digits for a 32-bit value) and set the radix to "hexadecimal."

3. Connect the input port: Connect the input port of the HEX digit Display component to the output port of the PC or RAM address. The PC output port can be found on the control unit circuit, while the RAM address output port can be found on the RAM memory circuit.

4. Test the display: To test the display, simulate the execution of a program using the "Simulate" mode in Logisim. Monitor the HEX digit Display component to ensure that the correct value is displayed when the PC or RAM address changes.

Overall, using the HEX digit Display component to display the value of the PC or RAM address in Logisim is a simple and effective way to monitor the operation of a processor. By displaying the value in hexadecimal format, it is easy to visualize the memory address and to debug any issues that arise during program execution.

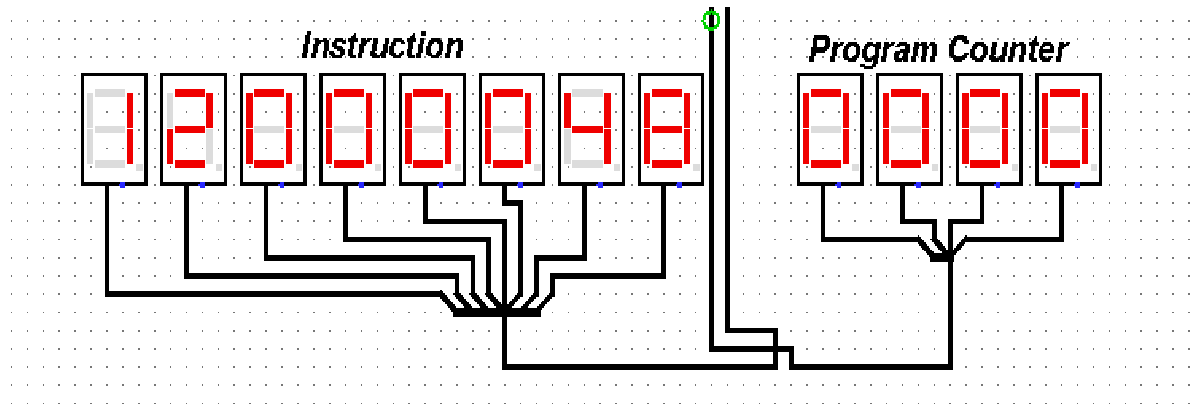


Fig:15 Instruction and Program Counter Indicators

2. Terminal Display :

In Logisim, the Terminal Display component is used to display ASCII characters on the screen. This component is useful for debugging programs that involve character input or output.

To add a Terminal Display component to your Logisim circuit, you can follow these steps:

1. Add the Terminal Display component: From the left-hand side menu, select the "I/O" category and drag the "Terminal Display" component onto the canvas.

2. Configure the Terminal Display: Double-click on the Terminal Display component to open its configuration dialog. Set the number of columns and rows to the desired values (e.g., 80 columns by 25 rows). You can also set the font size and background color.

3. Connect the input port: Connect the input port of the Terminal Display component to the output port of the component that generates the ASCII character data. For example, you can connect it to the output port of a UART (universal asynchronous receiver-transmitter) component that communicates with a serial device such as a keyboard or display.

4. Test the display: To test the Terminal Display, simulate the execution of a program that generates ASCII characters. Monitor the Terminal Display to ensure that the correct characters are displayed when the program runs.

Overall, the Terminal Display component is a useful tool for debugging programs that involve character input or output. By displaying the characters in a human-readable format, it is easy to understand the behavior of the program and to identify any issues that arise.

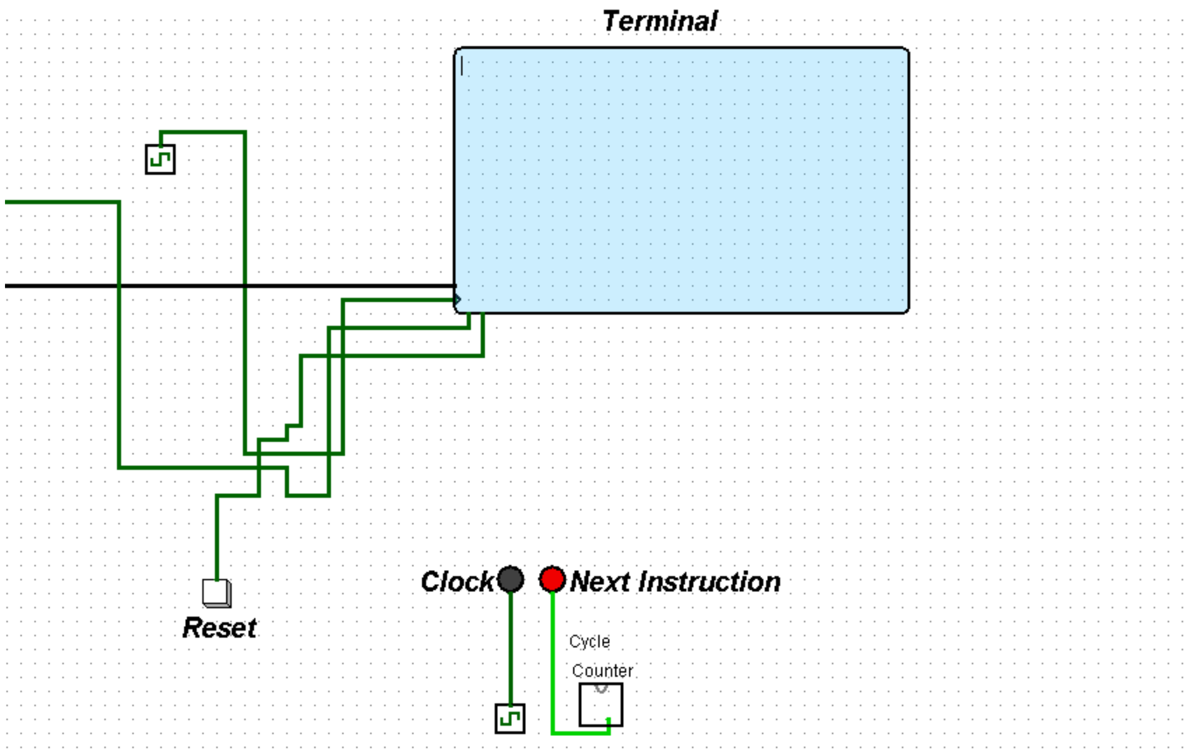


Fig:16 Terminal Display

4.28 Instruction :

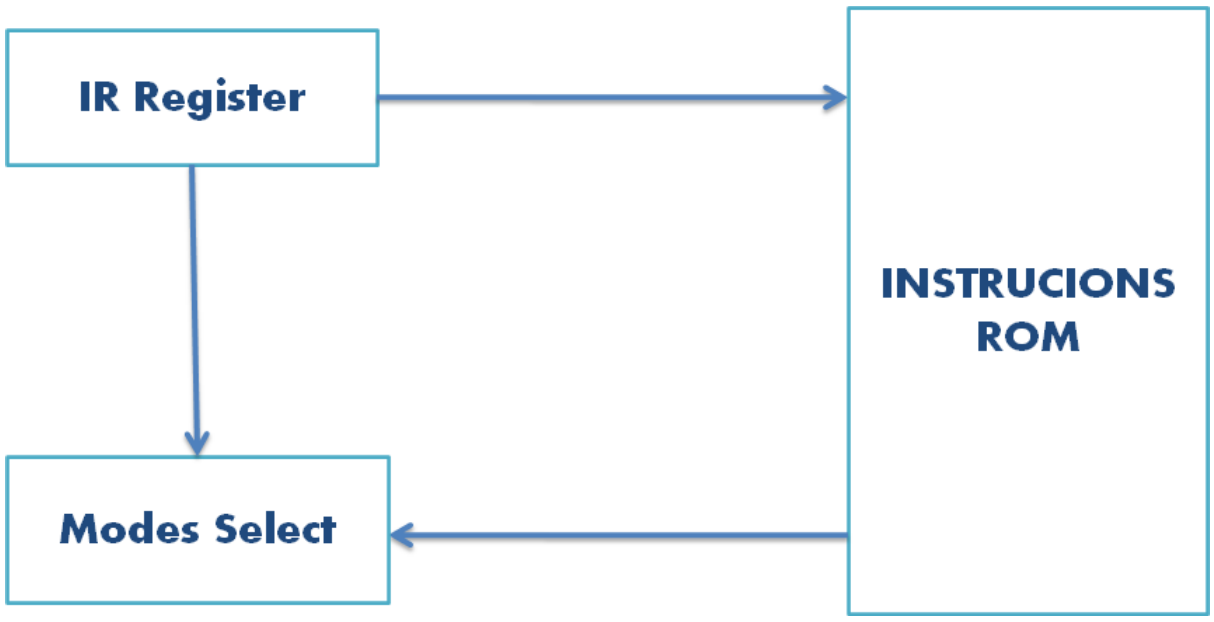


Fig: 17 : IR Register Connections

In a 32-bit RISC processor with a 5-stage pipeline, the Instruction Register (IR) is a circuit element that holds the current instruction being executed. The IR is located in the "Fetch" stage of the pipeline and is responsible for receiving the instruction from memory and making it available to the next stage of the pipeline.

The IR is typically a 32-bit register that is connected to the output of the program counter (PC) and the input of the instruction decoder. The PC provides the address of the instruction to be fetched, and the IR stores the actual instruction opcode and any associated operands.

During the fetch stage, the PC sends the address of the next instruction to be executed to the memory interface. The memory interface reads the instruction from memory and sends it to the IR. Once the instruction is in the IR, it is available to the instruction decoder, which interprets the opcode and generates the appropriate control signals to execute the instruction.

The IR is a critical component of the processor, as it ensures that the correct instruction is being executed at each stage of the pipeline. By holding the instruction in a register, the processor can quickly access and decode the opcode, without having to repeatedly access memory for each instruction. This improves the efficiency and performance of the processor.

Overall, the Instruction Register is an essential element of the 5-stage pipeline in a 32-bit RISC processor, enabling the efficient execution of instructions and improving the overall performance of the processor.

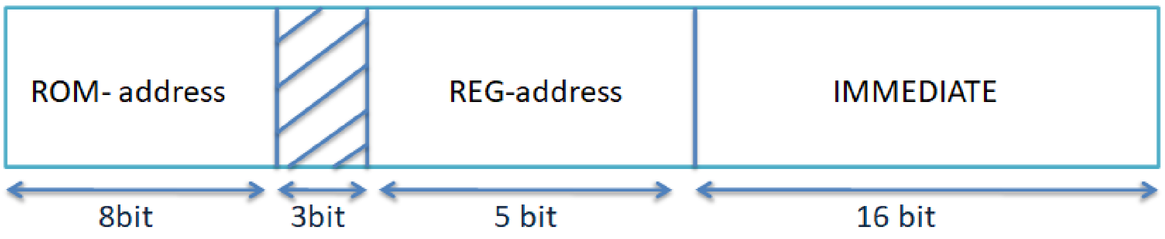
ROM Controls:

HEX Code	Instruction	32 bit HEX Value
00	NOP	00008000
01	MOV reg,immediate	8c005000
02	ADD d.reg,s1.reg,s2.reg	20007000
03	ADC d.reg,s1.reg,s2.reg	22007000
04	SUB d.reg,s1.reg,s2.reg	24007000
05	SUW d.reg,s1.reg,s2.reg	26007000
06	MUL d.reg,s1.reg,s2.reg	28007000
07	DIV d.reg,s1.reg,s2.reg	2a007000
08	TRSA d.reg,s1.reg	2c007000
09	TRSB d.reg,s2.reg	2e007000
0a	AND d.reg,s1.reg,s2.reg	30007000
0b	OR d.reg,s1.reg,s2.reg	32007000
0c	NAND d.reg,s1.reg,s2.reg	34007000
0d	NOR d.reg,s1.reg,s2.reg	36007000
0e	XOR d.reg,s1.reg,s2.reg	38007000
0f	XNOR d.reg,s1.reg,s2.reg	3a007000
10	NOT d.reg,s1.reg	3c007000
11	CMP reg1,reg2	3e002000
12	IPNT immediate	4c022000
13	RPNT reg	2c022000
14	DSTOWE address,reg	4c102000
15	DSTOWO address,reg	4c142000
16	DSTODW address,reg	4c302000
17	DLODWE reg,address	4c195000

18	DLODWO reg,address	4c1d5000
19	DLODDW reg,address	4c395000
1a	JG address	8c400000
1b	JE address	8c800000
1c	JL address	8cc00000
1d	JC address	8d000000
1e	JNG address	8c400800
1f	JNE address	8c800800
20	JNL address	8cc00800
21	JNC address	8d000800
22	JMP address	8d400000
23	IDSTOWE address	2c102000
24	IDSTOWO address	2c142000
25	IDSTODW address	2c302000
26	IDLODWE address	2c187000
27	IDLODWO address	2c1c7000
28	IDLODDW address	2c387000
29	PUSHWE reg	4c102200
2a	PUSHWO reg	4c142200
2b	PUSHDW reg	4c302200
2c	POPWE reg	4c195600
2d	POPWO reg	4c1d5600
2e	POPDW reg	4c395600
2f	INC reg,immediate	40007000
30	DEC reg,immediate	44007100

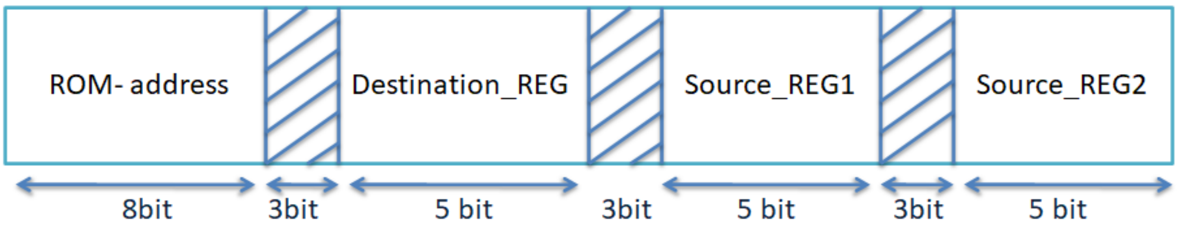
4.29 Addressing Modes:

1. Immediate addressing: In immediate addressing, the operand is encoded directly in the instruction itself. For example, an instruction may include an immediate value to be added to a register.



Ex : MOV reg,immediate -8c005000

2. Register addressing: In register addressing, the operand is stored in a register. The instruction provides the register number rather than the actual operand.



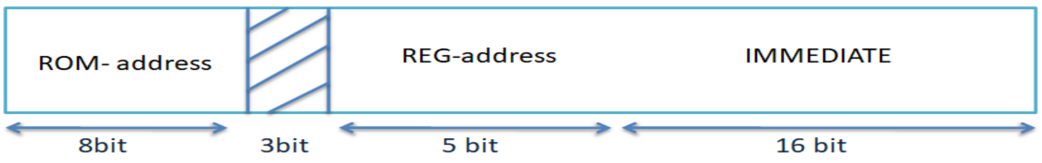
Ex:ADD d.reg,s1.reg,s2.reg -20007000

3. Indirect addressing: In indirect addressing, the instruction contains a memory address that points to another memory location containing the actual operand. The processor reads the operand from the second memory location.

Ex : IDSTOWE address - 2c102000

4. Memory Mode : Instructions used for pushing and popping data from Memory

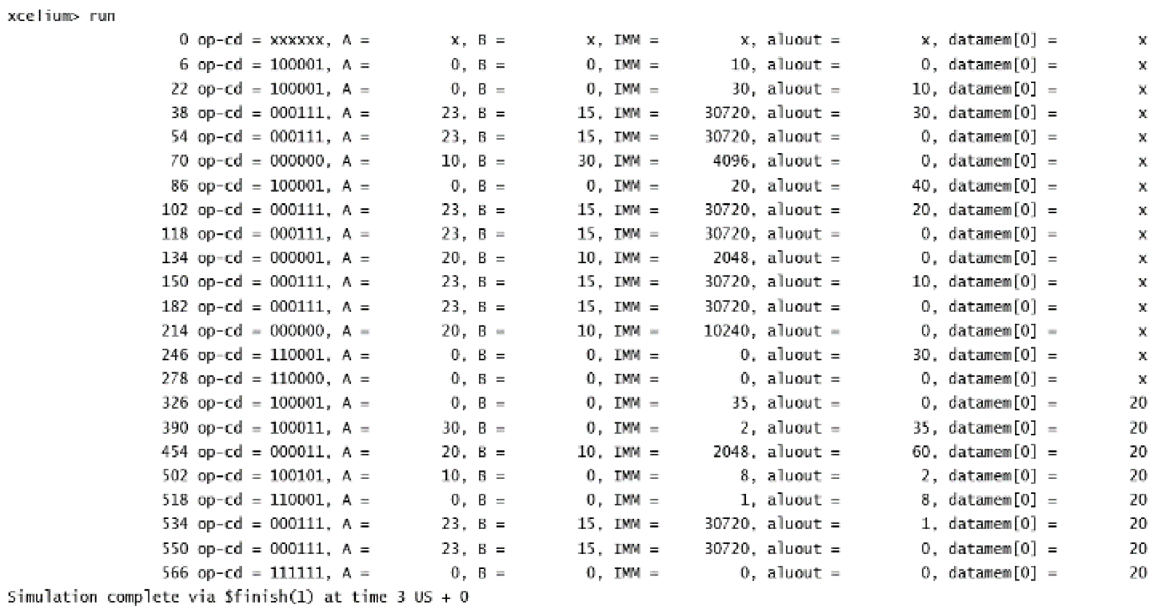
PUSHWE reg	-4c102400
PUSHWO reg	-4c142400
PUSHDW reg	-4c302400
POPWE reg	-4c195600
POPWO reg	-4c1d5600
POPDW reg	-4c395600



CHAPTER 5

5.1 SIMULATION RESULT:

The design was simulated using Cadence Xcelium 20.9 and with a total simulation time of 3 microsecond, it performed totally 13 operation which included all the basic operation. The Dynamic frequency scaling was tested with different instruction and desired output was found.



```
xcelium> run
                    0 op-cd = xxxxxx, A =          x, B =          x, IMM =          x, aluout =          x, datamem[0] =          x
                    6 op-cd = 100001, A =          0, B =          0, IMM =         10, aluout =          0, datamem[0] =          x
                   22 op-cd = 100001, A =          0, B =          0, IMM =         30, aluout =         10, datamem[0] =          x
                   38 op-cd = 000111, A =         23, B =         15, IMM =      30720, aluout =         30, datamem[0] =          x
                   54 op-cd = 000111, A =         23, B =         15, IMM =      30720, aluout =          0, datamem[0] =          x
                   70 op-cd = 000000, A =         10, B =         30, IMM =       4096, aluout =          0, datamem[0] =          x
                   86 op-cd = 100001, A =          0, B =          0, IMM =         20, aluout =         40, datamem[0] =          x
                  102 op-cd = 000111, A =         23, B =         15, IMM =      30720, aluout =         20, datamem[0] =          x
                  118 op-cd = 000111, A =         23, B =         15, IMM =      30720, aluout =          0, datamem[0] =          x
                  134 op-cd = 000001, A =         20, B =         10, IMM =       2048, aluout =          0, datamem[0] =          x
                  150 op-cd = 000111, A =         23, B =         15, IMM =      30720, aluout =         10, datamem[0] =          x
                  182 op-cd = 000111, A =         23, B =         15, IMM =      30720, aluout =          0, datamem[0] =          x
                  214 op-cd = 000000, A =         20, B =         10, IMM =      10240, aluout =          0, datamem[0] =          x
                  246 op-cd = 110001, A =          0, B =          0, IMM =          0, aluout =         30, datamem[0] =          x
                  278 op-cd = 110000, A =          0, B =          0, IMM =          0, aluout =          0, datamem[0] =          x
                  326 op-cd = 100001, A =          0, B =          0, IMM =         35, aluout =          0, datamem[0] =         20
                  390 op-cd = 100011, A =         30, B =          0, IMM =          2, aluout =         35, datamem[0] =         20
                  454 op-cd = 000011, A =         20, B =         10, IMM =       2048, aluout =         60, datamem[0] =         20
                  502 op-cd = 100101, A =         10, B =          0, IMM =          8, aluout =          2, datamem[0] =         20
                  518 op-cd = 110001, A =          0, B =          0, IMM =          1, aluout =          8, datamem[0] =         20
                  534 op-cd = 000111, A =         23, B =         15, IMM =      30720, aluout =          1, datamem[0] =         20
                  550 op-cd = 000111, A =         23, B =         15, IMM =      30720, aluout =          0, datamem[0] =         20
                  566 op-cd = 111111, A =          0, B =          0, IMM =          0, aluout =          0, datamem[0] =         20
Simulation complete via $finish(1) at time 3 US + 0
```

Fig.7. Register values after simulation

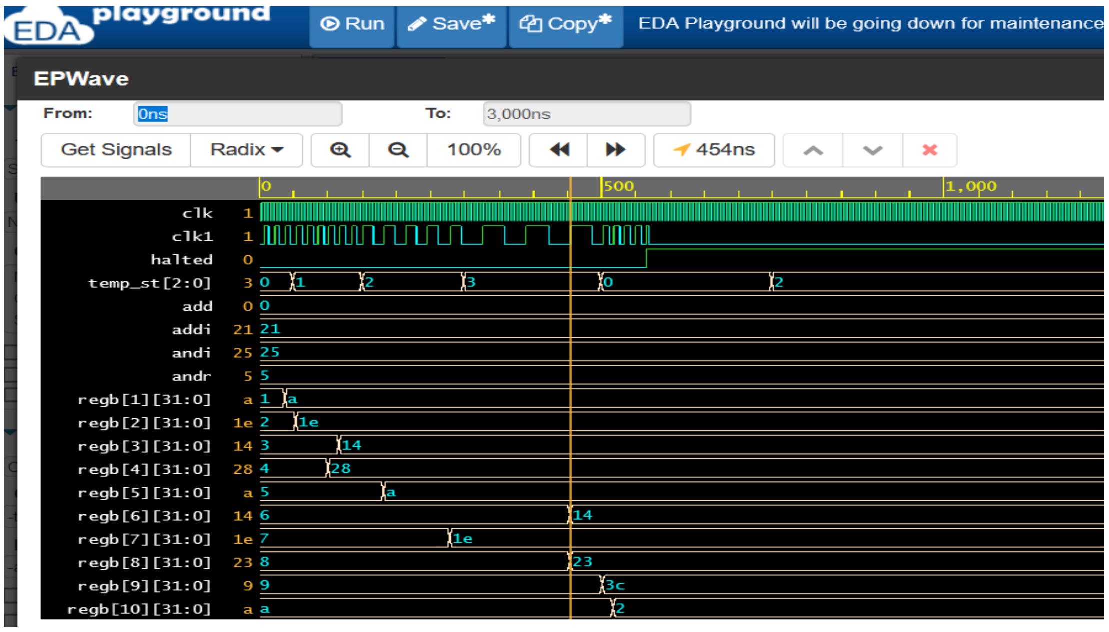
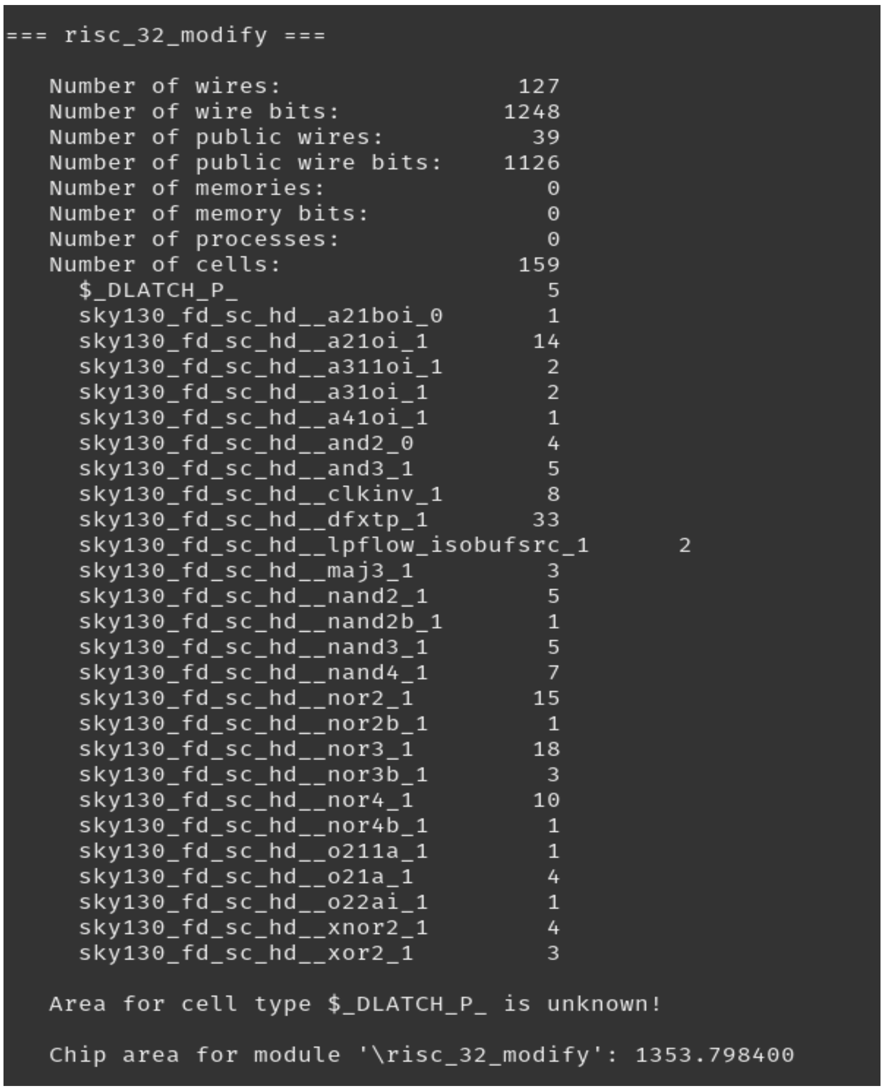


Fig.8. Simulation Output

5.2 Synthesis Using Yosys:

Yosys synthesis tool was used and the design was mapped with Sky water 130nm Library with high density standard cell.

Sky Water Open Source PDK: The Sky Water Open Source PDK is an Open Source PDK which is created by collaboration between Sky water Tech foundry and Google. This PDK is fully open access Process Design Kit which provides all the required resources. Using This PDK we can design any processor from RTL to GDS. This Design is manufacturable at Sky water tech foundry. There are totally 7 standard cell libraries of various configuration..Libraries: sky130_fd_sc_hs(highspeed),sky130_fd_sc_hd(highdensity)sky130_



```
=== risc_32_modify ===

   Number of wires:                127
   Number of wire bits:           1248
   Number of public wires:          39
   Number of public wire bits:    1126
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                159
     $_DLATCH_P_                     5
     sky130_fd_sc_hd__a21boi_0       1
     sky130_fd_sc_hd__a21oi_1       14
     sky130_fd_sc_hd__a311oi_1       2
     sky130_fd_sc_hd__a31oi_1        2
     sky130_fd_sc_hd__a41oi_1        1
     sky130_fd_sc_hd__and2_0         4
     sky130_fd_sc_hd__and3_1         5
     sky130_fd_sc_hd__clkinv_1       8
     sky130_fd_sc_hd__dfxtp_1       33
     sky130_fd_sc_hd__lpflow_isobufsrc_1      2
     sky130_fd_sc_hd__maj3_1         3
     sky130_fd_sc_hd__nand2_1        5
     sky130_fd_sc_hd__nand2b_1       1
     sky130_fd_sc_hd__nand3_1        5
     sky130_fd_sc_hd__nand4_1        7
     sky130_fd_sc_hd__nor2_1        15
     sky130_fd_sc_hd__nor2b_1        1
     sky130_fd_sc_hd__nor3_1        18
     sky130_fd_sc_hd__nor3b_1        3
     sky130_fd_sc_hd__nor4_1        10
     sky130_fd_sc_hd__nor4b_1        1
     sky130_fd_sc_hd__o211a_1        1
     sky130_fd_sc_hd__o21a_1         4
     sky130_fd_sc_hd__o22ai_1        1
     sky130_fd_sc_hd__xnor2_1        4
     sky130_fd_sc_hd__xor2_1         3

   Area for cell type $_DLATCH_P_ is unknown!

   Chip area for module '\risc_32_modify': 1353.798400
```

Fig.8. Synthesis Output

5.3 Complete Design in Logisim :

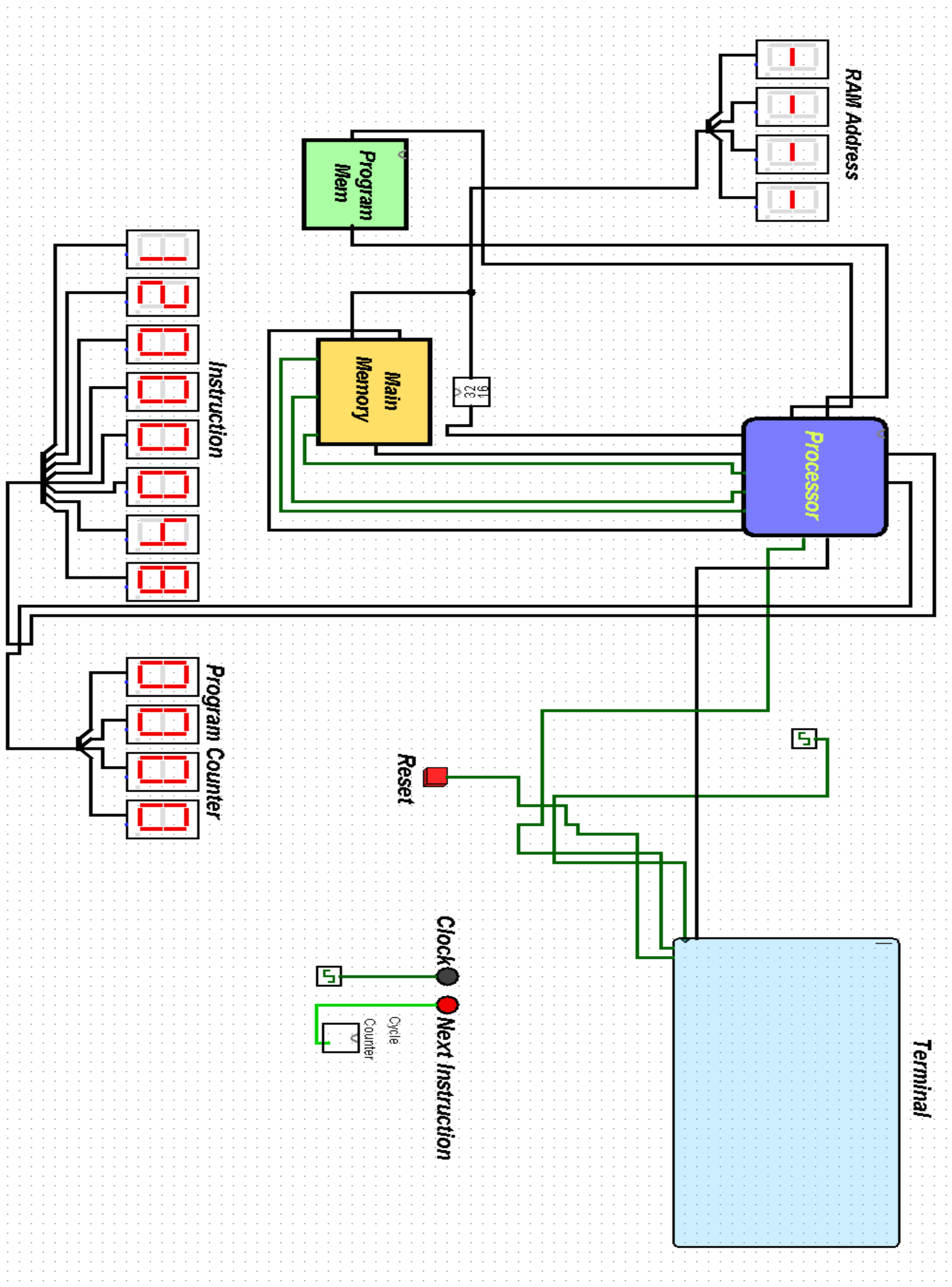
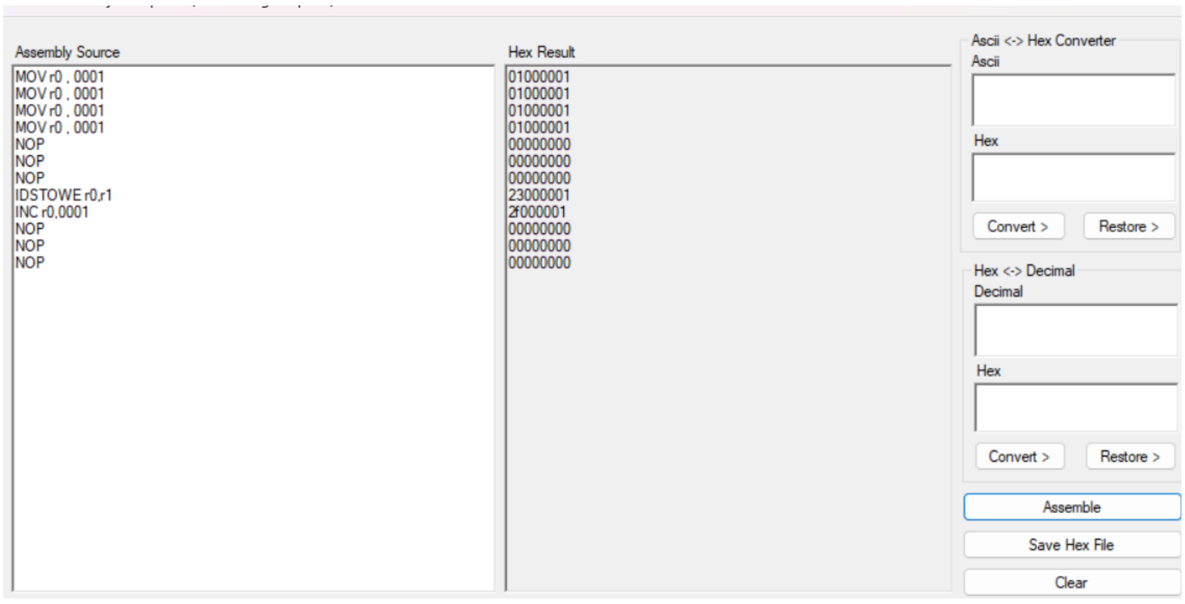


Fig:18 Complete Design

5.4 Simulation In Logisim

Creating HEX image using Assembly Interpreter:

- Using the Instructions write program in assembly level language
- Written program will be converted to 32 bit HEX codes image
- This HEX code image need to be uploaded into program memory for execution


Assembly Source

```
MOV r0 , 0001
MOV r0 , 0001
MOV r0 , 0001
MOV r0 , 0001
NOP
NOP
NOP
IDSTOWE r0,r1
INC r0,0001
NOP
NOP
NOP
```

Hex Result

```
01000001
01000001
01000001
01000001
00000000
00000000
00000000
23000001
2f000001
00000000
00000000
00000000
```

Fig:19 Assembly Interpreter

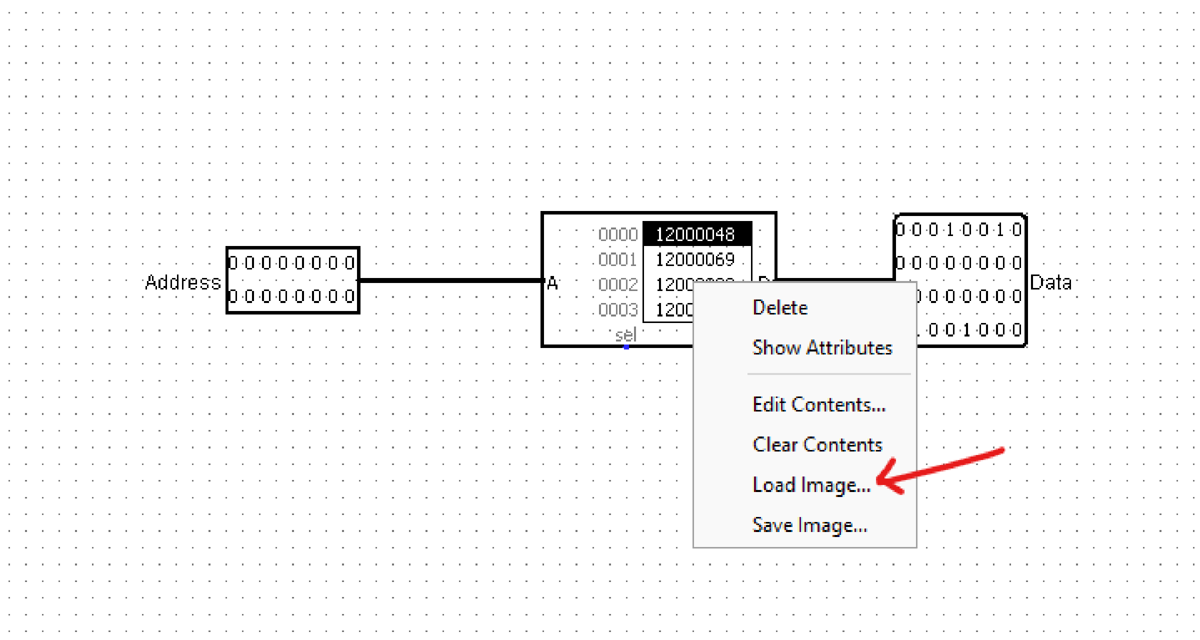
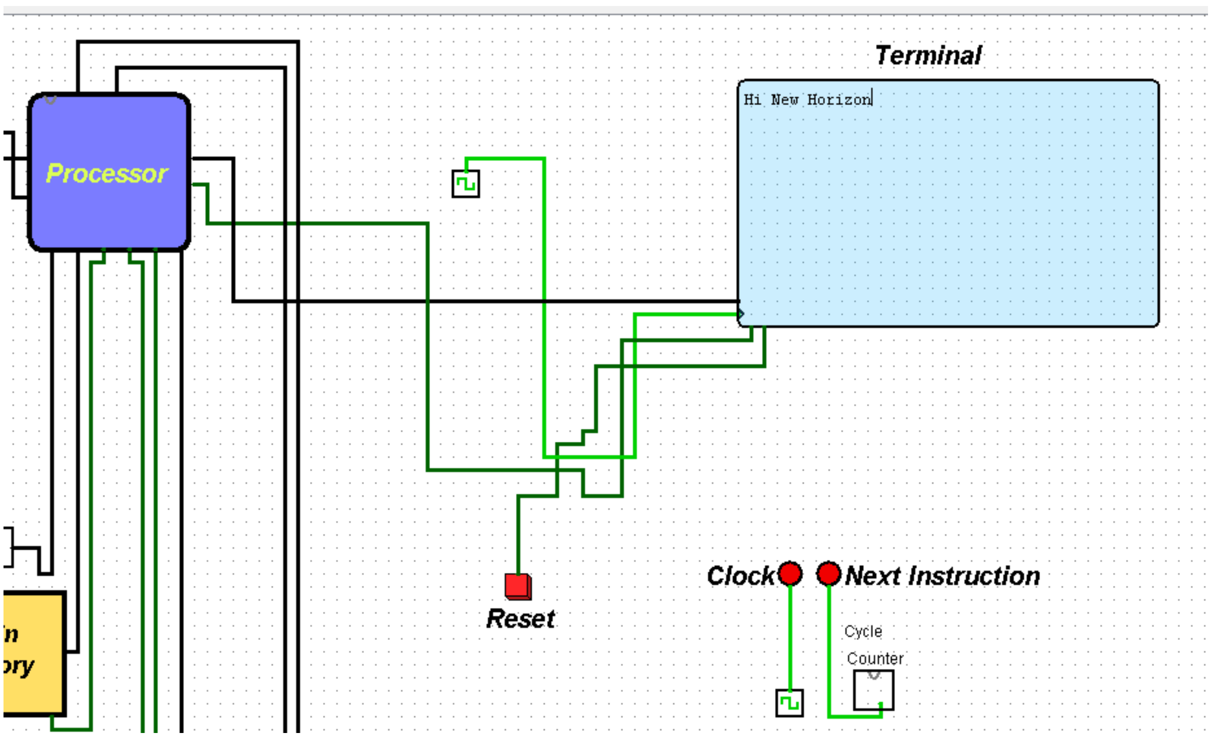


Fig:20 Image loading in memory

After loading the image file click on Simulation

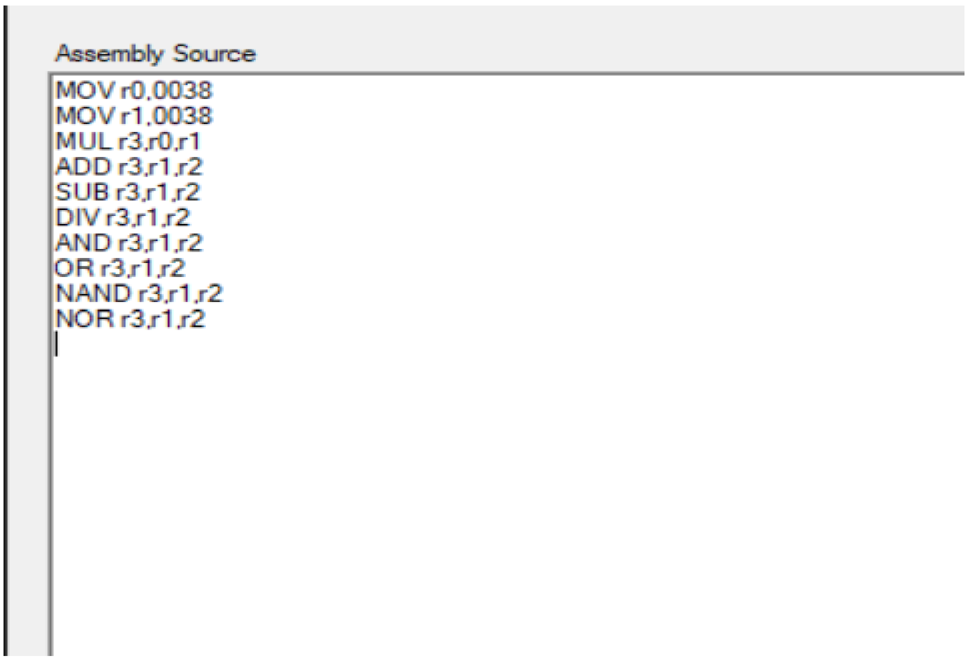
Example : Displaying Hi New Horizon in terminal :



```
Hi New Horizon
```

Fig:21 Output

Performing basic operations :



```
MOV r0,0038
MOV r1,0038
MUL r3,r0,r1
ADD r3,r1,r2
SUB r3,r1,r2
DIV r3,r1,r2
AND r3,r1,r2
OR r3,r1,r2
NAND r3,r1,r2
NOR r3,r1,r2
```

Fig:22 Example Assembly Code

CHAPTER 6

6.1 CONCLUSION

The design for 32-RISC MIPS processor with Flexible Pipeline and Dynamic thermal control was successfully simulated with UVM testbench and desired output with total simulation time of 3 microsecond and after simulation total cell area was found to be 1353. 7984 micrometer.The future scope of this design will be to design it for more number instruction and improving the frequency scaling for better results. The design can be implemented for 64-bit RISC architecture.

In conclusion, the design and implementation of a 32-bit RISC processor with a 5-stage pipeline in Logisim software was successfully completed. The project involved designing and implementing various components of the processor such as the control unit, data path, instruction register, and memory interfaces.

The design was based on the RISC architecture, which is known for its simplicity and efficiency. The processor was able to execute a variety of instructions, including arithmetic, logical, and memory instructions, using different addressing modes.

The project also involved the creation of a program and data memory and uploading of a hex file containing machine code instructions. The processor was able to run the program and display the output on a terminal display module.

Overall, the project provided a comprehensive understanding of the design and implementation of a 32-bit RISC processor with a 5-stage pipeline. It also provided hands-on experience in using Logisim software for designing and simulating digital circuits.

CHAPTER 7

REFERENCES

[1] Hyun Woo Oh, Kwon Neung Cho and Seung Eun Lee" Design of 32-bit Processor for Embedded Systems"- 2020 International SoC Design Conference

[2] David Brooks "Dynamic Thermal Management for High-Performance Microprocessors"-2021|IEEE

[3] Saroj Ashish, Kumar Ramprakash" A Review on Analysis of 32-bit and 64-Bit RISC Processors"

[4] Sneha Mangalwedhe and Roopa Kulkarni" Low Power implentation of 32-Bit RISC Processor with Pipeling"2021|Research Gate.

[5] Neha Dwivedhi, Preadeep Chhwacharia" Design and Implementation of 32-bit RISC with Five Stage Pipeline" 201732-RISC Processor"-IJERA-2017

[6] Indu M, Arun Kumar M" Design of Low Power pipelined RISC Processor" IJAREEIE, vol.2, issue 8,2013

[7] G Rajesh Babu, M Bhanu Prakash" Design of 32-Bit Asynchronous RISC-V Processor using Verilog"2020 JETIR

[8] Pritesh Kumar Yadav and Prasanna Kumar Misra" Power Aware Study of 32-bit 5-stage pipeline RISC CPU using 180nm CMOS technology

[9] Wael M EL Medany, Khalid A Al Kooheji" Design and Implementation of a 32-Bit RISC processor on Xilinx FPGA"2012

[10] Maciej Frankiewicz, Piotr Kocanda" Design of RISC Microcontroller with Dynamic Thermal Management Unit for Temperature-controller Oscillator" MIXDES 2014

[11] Clifford Wolf, Johannes Kepler University "yosys-A Free Verilog Synthesis Suite"

[12] Archana Rani and Dr.Naresh Grover" Novel Design of 32-bit Asynchronous (RISC) Microprocessor & its Implementation on FPGA"- I.J. Information Engineering and Electronic Business, 2018.

[13] Shaik Afroz "Implementation of RISC-Based Architecture for Low Power Applications"-ISOR-2013

[14] Mr.Shivkumar V, Mrs.Swomya Sunkara" Design and Verification of 32-bit Pipelined RISC Architecture using Verilog modeling and UVM test bench Methodology"

[15] M.Kishore and MD. Shabeena "FPGA based implementation of 32-Bit RISC Processor" --IJERA-2013