ವಿಶ್ವೇಶ್ವರಯ್ಯ ತಾಂತ್ರಿಕ ವಿಶ್ವವಿದ್ಯಾಲಯ, ಬೆಳಗಾವಿ

VISVESVARAYA TECHNOLOGICAL UNIVERSITY - BELAGAVI

A
Project Report
on

"Simulation of RISC-V Single Cycle Microarchitecture"

Submitted in the partial fulfillment for the award of
Bachelor of Engineering
in
Electronics and Communication Engineering

<u>Submitted by</u>

PRIYANKA C H	**2GO21EC033**
YOGESH F R	**2GO21EC052**
DARSHAN S M	**2GO21EC056**
SANJEEV B K	**2GO22EC406**

<u>Under the Guidance of</u>

Prof. PRASHANTHI H J M.Tech
Assistant Professor

GOVERNMENT ENGINEERING COLLEGE
HAVERI- 581 110

DEPARTMENT
OF
ELECTRONICS & COMMUNICATION ENGINEERING

2024-25

GOVERNMENT ENGINEERING COLLEGE

HAVERI - 581 110

(Affiliated to Visvesvaraya Technological University)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Certificate

*This is to certify that the project work entitled **"SIMULATION OF RISC-V SINGLE CYCLE MICROARCHITECTURE** carried out by **PRIYANKA C H 2GO21EC033, YOGESH F R 2GO21EC052, DARSHAN S M 2GO21EC056, SANJEEV B K 2GO22EC406** are bonafide students of Government Engineering College, Haveri in partial fulfillment for the award of Bachelor of Engineering in Electronics and Communication Engineering of the Visvesvaraya Technological University, Belagavi during the year 2024-2025. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Degree.*

…………………………	…………………...……….
Project Guide	Project Coordinator
Prof. PRASHANTHI H J M.Tech	Dr. BASAVARAJ RABAKAVI M.Tech, Ph.D
Assistant Professor	Assistant Professor
…………………...……….	…………………...……….
Head of the Department	Principal
Dr. JAYAPRAKASHA H M.Tech, Ph.D	Dr. MUNIRAJU M M.E, Ph.D
Assistant Professor	

Examiners:

1. ______________________________ _ _ _ _ _ _ _ _ _ _ _ _ _ _

2. ______________________________

DATE :

PLACE: Haveri

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JNANA SANGAMA", BELAGAVI-590 018

DECLARATION

We, ***PRIYANKA C H 2GO21EC033, YOGESH F R 2GO21EC052DARSHAN S M 2GO21EC056 and SANJEEV B K 2GO22EC406*** *are the students of the 8th semester Bachelor of Engineering in Electronics and Communication Engineering, Government Engineering College, Haveri hereby declare that the project report titled* ***"SIMULATION OF RISC-V SINGLE CYCLE MICROARCHITECTURE"*** *has been prepared and presented by us and submitted in partial fulfillment of the course requirements for the award of degree in* ***Bachelor of Engineering*** *in* ***Electronics and Communication Engineering*** *of* ***Visvesvaraya Technological University, Belagavi*** *during the year* ***2024-2025****. Further, the content of the report has not been submitted previously for the award of any degree or diploma to any other university.*

Place: Haveri

Project Associates
2GO21EC033
2GO21EC052
2GO21EC056
2GO22EC406

ACKNOWLEDGEMENT

We express our sincere gratitude to our guide **Prof. Prashanthi H J**, Dept. of Electronics & Communication Engineering, GEC, Haveri, for his/her stimulating guidance, continuous encouragement, and supervision throughout the project work.

We would like to place on record our deep sense of gratitude to **Dr. Jayapraksha H**, Head of the Department. of Electronics & Communication Engineering, GEC, Haveri for his generous guidance, help and useful suggestions.

We would like to thank **Dr. Basavaraj Rabakavi,** Project Coordinator, and all the staff members of the Department of Electronics and Communication Engineering for their insightful comments and constructive suggestions to improve the quality of this project work.

We are extremely thankful to **Dr. Muniraju M,** Principal, GEC, Haveri, for providing us with infrastructural facilities to work on the project, without which this work would not have been possible.

We are extremely happy to acknowledge and express our sincere gratitude to our parents, friends who gave us constant support and encouragement, and well wishes for their help and cooperation and solutions to problems during the Project Work.

Names of Students	**USN**
PRIYANKA C H	2GO21EC033
YOGESH F R	2GO21EC052
DARSHAN S M	2GO21EC056
SANJEEV B K	2GO22EC406

ABSTRACT

The Single-Cycle RISC-V processor is a simplified processor architecture designed to execute each instruction in a single clock cycle. The processor supports a reduced instruction set, including arithmetic, logical, memory access, and control flow instructions, enabling efficient computation with minimal hardware complexity. Key components such as the Program Counter (PC), Instruction Memory, Register File, Arithmetic Logic Unit (ALU), Data Memory, and Control Unit are integrated into a unified data path.

The processor fetches an instruction from memory, decodes it, executes the operation, and writes back the result within one clock cycle. Control signals generated by the Control Unit ensure seamless coordination between components. The implementation is written in Verilog, and Vivado is used for simulation, debugging, and synthesis, ensuring functionality and hardware compatibility. This abstraction simplifies the understanding of processor design by focusing on essential operations, laying the groundwork for further exploration of advanced architectural concepts.

List of Figures

CONTENTS

CHAPTER-1
INTRODUCTION

RISC-V, an open-source instruction set architecture (ISA), has been making waves in the world of computer architecture. “RISC-V” stands for Reduced Instruction Set Computing (RISC) and the “V” represents the fifth version of the RISC architecture.Unlike proprietary architectures such as ARM and x86, RISC-V is an open standard, allowing anyone to implement it without the need for licensing fees. This openness has led to a surge in interest and adoption across various industries, making RISC-V a key player in the evolving landscape of computing. At its core, an instruction set architecture defines the interface between software and hardware, dictating how a processor executes instructions. RISC-V follows the principles of RISC, emphasizing simplicity and efficiency in instruction execution. This simplicity facilitates easier chip design, reduces complexity, and allows for more straightforward optimization of hardware and software interactions. This stands in contrast to Complex Instruction Set Computing (CISC) architectures, which have more elaborate and versatile instructions, often resulting in more complex hardware designs.

The open nature of RISC-V is one of its most significant strengths. The ISA is maintained by the RISC-V Foundation, a non-profit organization that oversees its development and evolution. The RISC-V Foundation owns, maintains, and publishes the RISC-V Instruction Set Architecture (ISA), an open standard for processor design. The RISC-V Foundation was founded in 2015 and comprises more than 200 members from various sectors of the industry and academia. A single-cycle RISC-V processor is a simplified implementation of the RISC-V instruction set architecture (ISA) that executes one instruction per clock cycle. It is designed for educational and small-scale projects, making it ideal for understanding the fundamentals of processor design.

The RISC-V ISA, known for its open-source and modular nature, uses fixed 32-bit instruction formats, such as R-type, I-type, and S-type, to simplify decoding and execution. In a single-cycle processor, the fetch, decode, execute, memory access, and write-back stages are completed in one clock cycle. While this design is straightforward and highlights the core principles of instruction execution, its limitation lies in inefficiency for complex instructions, as all operations must fit within the duration of the longest instruction.

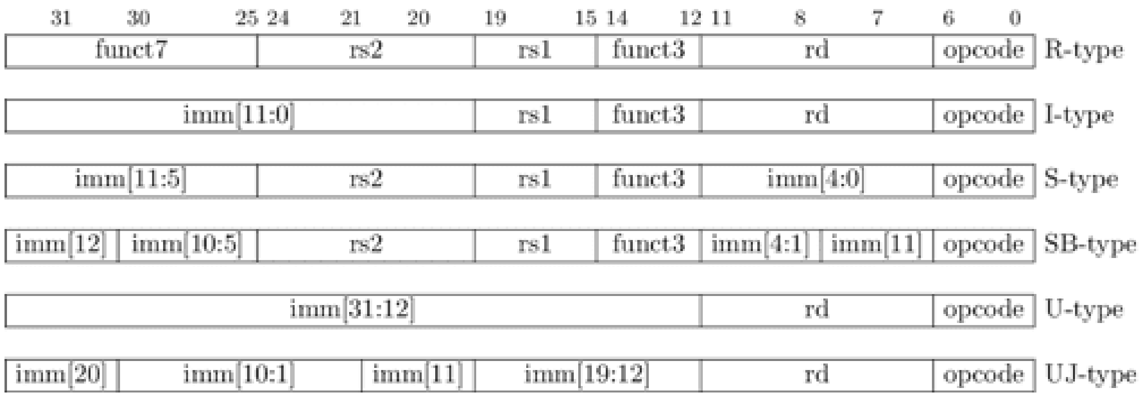
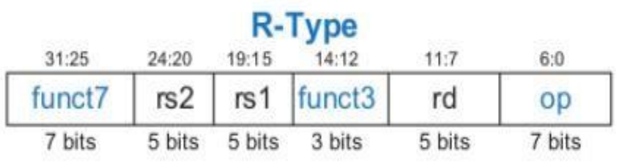


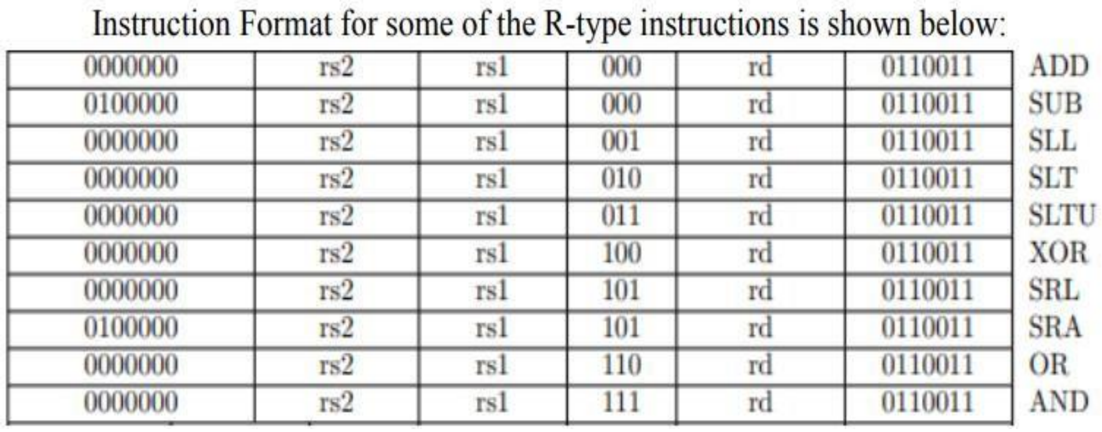
Fig1.1: 32-bit RISC-V instruction formats

- ***opcode* (7 bits):** Partially specifies one of the 6 types of *instruction formats*.
- ***funct7* (7 bits) and *funct3* (3 bits):** These two fields extend the *opcode* field to specify the operation to be performed.
- ***rs1* (5 bits) and *rs2* (5 bits):** Specify, by index, the first and second operand registers respectively (i.e., source registers).
- ***rd* (5 bits):** Specifies, by index, the destination register to which the computation result will be directed.

1.1 R-Type Instruction

The name R-type is short for register-type. R-type instructions use three registers as operands: two as sources, and one as a destination. The figure below shows the R-type machine instruction format. The 32-bit instruction has six fields: op, rs1, rs2, rd, funct3, and funct7. Each field is five or seven bits, as indicated. The operation the instruction performs is encoded in the three fields highlighted in blue: op (also called opcode or operation code) and funct3 and funct7 (also called the function). All R-type instructions have an opcode of 33. The specific R-type operation is determined by the function fields. The operands are encoded in the three fields: rs1, rs2, and rd. The first two registers, rs1, and rs2 are the source registers; rd is the destinationreg.




Instruction Format for some of the R-type instructions is shown below:

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Fig 1.2: Instruction Format for R-Type

1.2 PROBLEM STATEMENT

Design a single-cycle RISC processor capable of executing basic instructions, including arithmetic, logical, data movement, and control flow, within one clock cycle. The processor should feature a simple data path, including an ALU, register file, memory interface, and control unit, ensuring all operations complete in a single cycle.

The design must prioritize simplicity, avoid pipelining, and ensure correct data flow with appropriate control signals. Expected outcomes include a functional VHDL/Verilog implementation, verified through simulation, and analysis of performance metrics like clock frequency and resource usage.

1.3 OBJECTIVES

1. Design of Single-Cycle RISC V Processor microarchitecture.

2. Simulation of R type instruction format

CHAPTER 2

LITERATURE SURVEY

1.Title: ”Design of a single-cycle RISC microarchitecture**”**

Volume: 5(59) **Issue:**2022

Authors: Michal Štepanovský, Pavel Tvrdík.

Description: Project focuses on entire asserting charecterstics of the Single Cycle RISC Architecture and design Flow and execution asserting ideas about single cycle execution.

2.Title: ” An Overview of RISC Architecturture”

Volume: 5(6) **Issue**:2020

Authors: Samuel O. Aletan

Description: This Project focus on the second generation of RISC processors are utilizing superpipelining or superscalar to achieve execution speed that was once attainable only on mainframes and supercomputers. Future RISC processors will use both superscalar and superpipelining techniques to achieve speed beyond the 50 MIPS that is attainable on some of today’s processors.

3.Title: ” Design of 16-bit RISC Processor**”**

Volume: 5(59) **Issue:**2022

Authors: Supraj Gaonkar , Anitha M.k

Description: In this paper the behavioural design and functional characteristics of 16-bit RISC processor is proposed, which utilizes minimum functional units without compromising in performance. The design is based on Harvard architecture having separate data memory and instruction memory. The instruction word length is 24-bit wide. The processor supports 16 instructions with three addressing modes. It has 16 general purpose registers. Each register can store 16-bit data. The processor has 16- bit ALU capable of performing 11 arithmetical logical

4.Title: ” Hardware Modelling Of A 32-Bit, Single Cycle Risc Processor Using Vhdl**”**

Volume: 5(8) **Issue:**2017

Authors: Safaa S. Omran, Hadeel S. Mahmood,

Description: In this research, the VHDL (Very high speed IC Hardware Description Language) hardware modelling of the complete design of a 32-bit MIPS (Microprocessor without Interlocked Pipeline Stages), single cycle RISC (Reduced Instruction Set Computer) processor is presented. First the work defines the MIPS ISA (Instruction Set Architecture) and then describes how to divide the processor’s complete design into two parts: the datapath unit,control unit

CHAPTER-3

SOFTWARE DESCRIPTION

3. PROGRAMMING LANGUAGE

3.1 Verilog HDL

In the semiconductor and electronic design industry, Verilog is a hardware description language (HDL) used to model electronic systems. Verilog HDL, not to be confused with VHDL (a competing language), is most commonly used in the design, verification, and implementation of digital logic chips at the register- transfer level of abstraction. It is also used in the verification of analog and mixed-signal circuits.

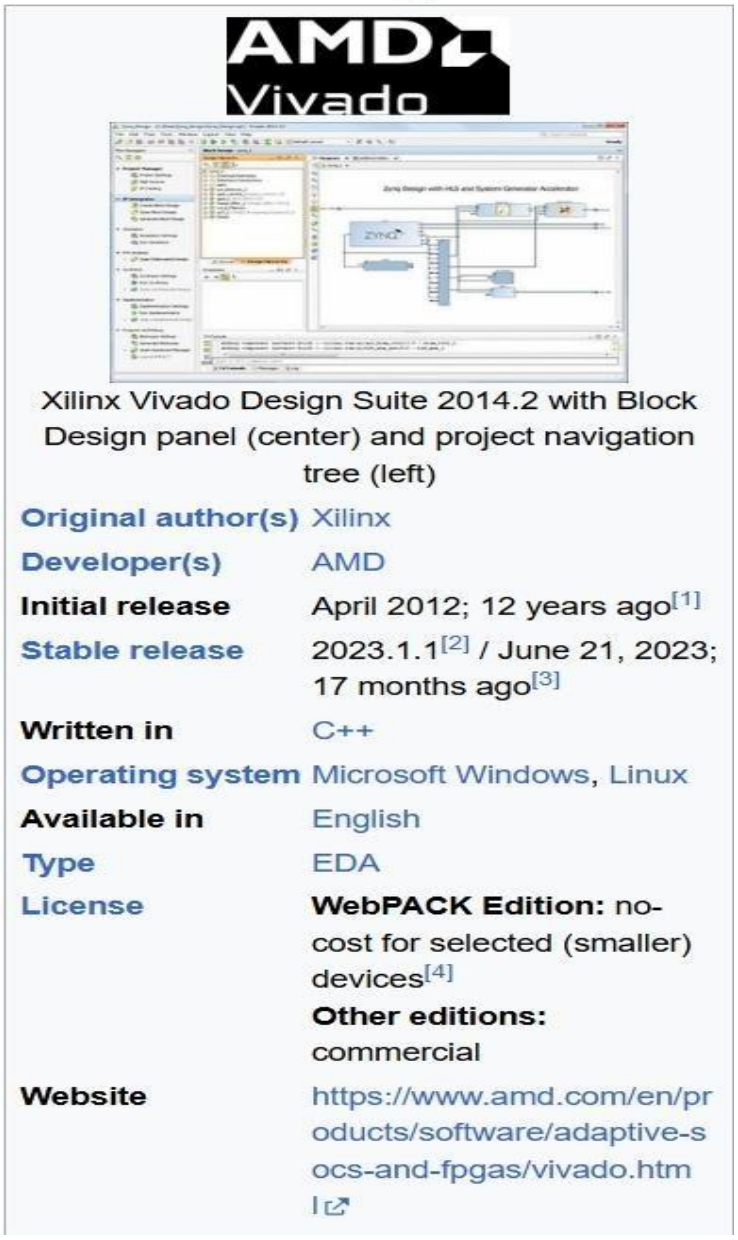
Hardware description languages such as Verilog differ from software programming languages because they include ways of describing the propagation of time and signal dependencies (sensitivity). There are two assignment operators, a blocking assignment (=), and a non-blocking (<=) assignment. The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variables.

A Verilog design consists of a hierarchy of modules. Modules encapsulate design hierarchy, and communicate with other modules through a set of declared input, output, and bidirectional ports. Internally, a module can contain any combination of the following: net/variable declarations (wire, reg, integer, the blocks themselves are executed concurrently, making Verilog a data flow language).

3.2 Xilinx Vivado

Vivado Design Suite is a software suite for synthesis and analysis of hardware description language (HDL) designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis. Vivado represents a ground-up rewrite and re-thinking of the entire design flow (compared to ISE). Like the later versions of ISE, Vivado includes the in-built logic simulator. Vivado also introduces high-level synthesis, with a toolchain that converts C code into programmable logic.

Replacing the 15 year old ISE with Vivado Design Suite took 1000 man years and cost US$200 million.



Xilinx Vivado Design Suite 2014.2 with Block Design panel (center) and project navigation tree (left)

Original author(s)	Xilinx
Developer(s)	AMD
Initial release	April 2012; 12 years ago[1]
Stable release	2023.1.1[2] / June 21, 2023; 17 months ago[3]
Written in	C++
Operating system	Microsoft Windows, Linux
Available in	English
Type	EDA
License	**WebPACK Edition:** no-cost for selected (smaller) devices[4] **Other editions:** commercial
Website	https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html

Fig 3.1 : AMD Vivado Suite

ISIM may be used to perform the following types of simulations:

- Logical verification, to ensure the module produces expected results
- Behavioural verification, to verify logical and timing issues
- Post-place & route simulation.

Vivado was introduced in April 2012,and is an integrated design environment (IDE) with system-to-IC level tools built on a shared scalable data model and a common debug environment. Vivado includes electronic system level (ESL) design tools for synthesizing and verifying C-based algorithmic IP; standards based packaging of both algorithmic and RTL IP for reuse; standards based IP stitching and systems integration of all types of system building blocks; and the verification of blocks and systems.A free version WebPACK Edition of Vivado provides designers with a limited version of the design environment.

The Vivado High-Level Synthesis compiler enables C, C++ and SystemC programs to be directly targeted into Xilinx devices without the need to manually create RTL. Vivado HLS is widely reviewed to increase developer productivity, and is confirmed to support C++ classes, templates, functions and operator overloading. Vivado 2014.1 introduced support for automatically converting OpenCL kernels to IP for Xilinx devices. OpenCL kernels are programs that execute across various CPU, GPU and FPGA platforms.

The Vivado Simulator is a component of the Vivado Design Suite. It is a compiled-language simulator that supports mixed-language, Tcl scripts, encrypted IP and enhanced verification.

The Vivado IP Integrator allows engineers to quickly integrate and configure IP from the large Xilinx IP library. The Integrator is also tuned for MathWorks Simulink designs built with Xilinx's System Generator and Vivado High-Level Synthesis.

The Vivado Tcl Store is a scripting system for developing add-ons to Vivado, and can be used to add and modify Vivado's capabilities. Tcl is the scripting language on which Vivado itself is based. All of Vivado's underlying functions can be invoked and controlled via Tcl scripts.

It is one of most popular software tools used to synthesize Verilog code. This tool includes many steps. To make user feel comfortable with the tool the steps are given below: -

Step 1: Launch Vivado

- Double-click the Vivado icon on your desktop or navigate to the installation directory and run vivado.exe.
- Select the desired workspace and click "OK".

Step 2: Create a New Project

- In the Vivado IDE, click "File" > "Project" > "New Project".
- Select "RTL Project" and click "Next".

- Enter the project name, location, and select the desired device family (e.g., Artix-7, Kintex-7, etc.).
- Click "Finish" to create the project.

Step 3: Add Verilog Files

- In the Vivado IDE, click "File" > "Add Sources".
- Select "Add or create design sources" and click "Next".
- Navigate to the location of your Verilog file (.v) and select it.
- Click "Finish" to add the Verilog file to the project.

Step 4: Set the Top Module

- In the Vivado IDE, click "Flow Navigator" > "Simulation" > "Set Top Module".
- Select the top-level Verilog module from the dropdown list.
- Click "OK" to set the top module.

Step 5: Compile the Design

- In the Vivado IDE, click "Flow Navigator" > "Synthesis" > "Run Synthesis".
- Wait for the synthesis process to complete.

Step 6: Simulate the Design

- In the Vivado IDE, click "Flow Navigator" > "Simulation" > "Run Simulation".
- Select the desired simulator (e.g., Vivado Simulator, ModelSim, etc.).
- Click "OK" to start the simulation.

Step 7: View Simulation Results

- In the Vivado IDE, click "Window" > "Simulation" > "Waveform".
- The waveform viewer will display the simulation results.

CHAPTER-4
METHODOLOGY

4.1 BLOCK DIAGRAM OF RISC V MICROARCHITECTURE

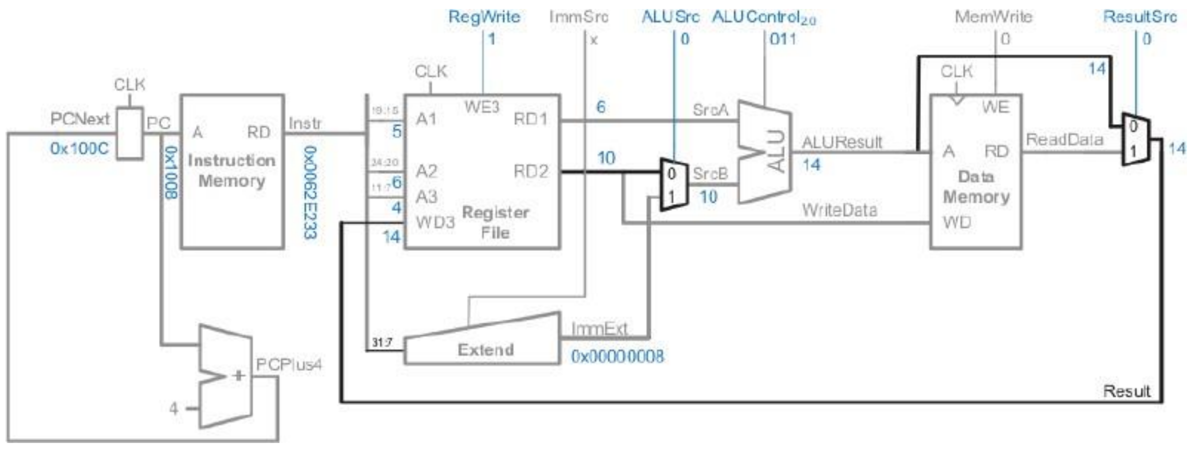


Fig 4.1 : BLOCK DIAGRAM OF RISC V MICROARCHITECTURE

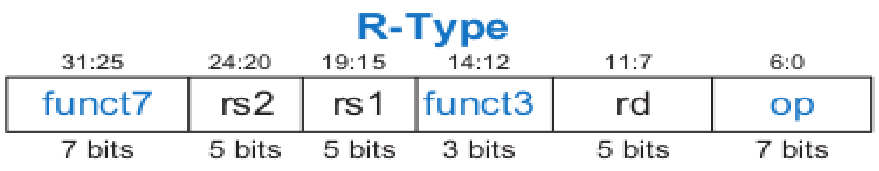
In this section we will extend the data path to handle R-type instructions add, sub, and, or, and slt. All of these instructions read two registers from the register file, perform some ALU operation on them, and write the result back to a third register in the register file. They differ only in the specific ALU operation. Hence, they can all be handled with the same hardware, using different ALUControl signals.

Figure below shows the enhanced datapath handling R-type instructions. The register file reads two registers. The ALU performs an operation on these two registers. In previous labs we saw that the ALU always received its SrcB operand from the sign-extended immediate (ImmExt). Now, we add a multiplexer to choose SrcB from either the register file RD2 port or ImmExt. The multiplexer is controlled by a new signal, ALUSrc. ALUSrc is 0 for R-type instructions to choose SrcB from the register file; it is 1 for Load and store to choose ImmExt.

This principle of enhancing the datapath's capabilities by adding a multiplexer to choose inputs from several possibilities is extremely useful. Indeed, we will apply it twice more to complete the handling of R-type instructions. Previously, the register file always got its written data from the data memory. However, R-type instructions write the ALUResult to the register file. Therefore, we add another multiplexer to choose between ReadData and ALUResult. We call its output Result. This multiplexer is controlled by another new signal, ResultSrc. ResultSrc is 0 for R-type instructions to choose Result from the ALUResult; it is 1 for Load to choose ReadData. We don't care about the value of ResultSrc for store, because store does not write to the register file.

The name R-type is short for register-type. R-type instructions use three registers as operands: two as sources, and one as a destination. Figure below shows the R-type machine instruction format. The 32-bit instruction has six fields: op, rs1, rs2, rd, funct3, and funct7. Each field is five or seven bits, as indicated. The operation the instruction performs is encoded in the three fields highlighted in blue: op (also called opcode or operation code) and funct3 and funct7 (also called the function). All R-type instructions have an opcode of 33.

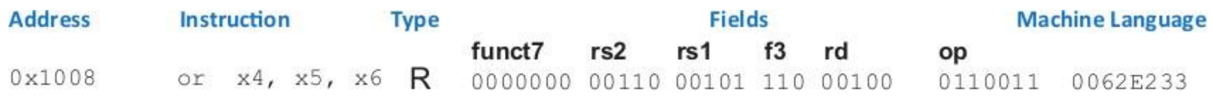
The specific R-type operation is determined by the funct field. The operands are encoded in the three fields: rs1, rs2, and rd. The first two registers, rs1 and rs2, are the source registers; rd is the destination register.



R-Type

31:25	24:20	19:15	14:12	11:7	6:0
funct7	rs2	rs1	funct3	rd	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Figure below shows an example for R-Type instruction represented in assembly code and machine code.



Address	Instruction	Type	Fields funct7	rs2	rs1	f3	rd	op	Machine Language
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233

Fig.4.2 : R-type instruction for OR-operation

4.2 Arithmetic Logic Unit (ALU)

Design an ALU that can perform a subset of the ALU operations of a full Processor ALU.

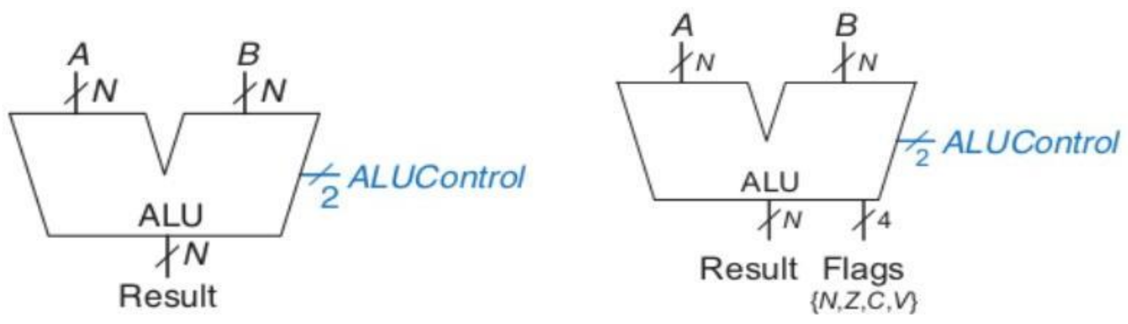
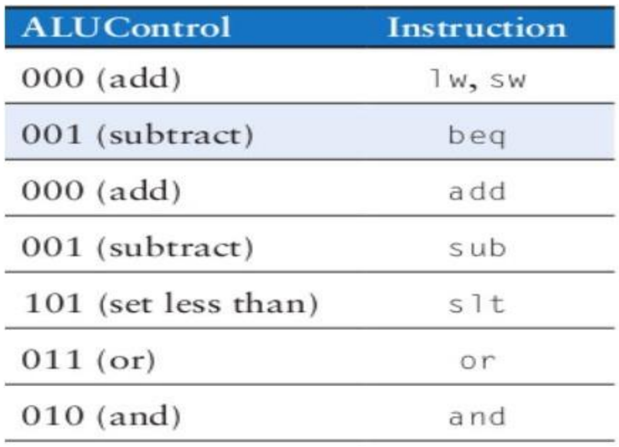


Fig 4.3 : Basic ALU Block Diagram

In this exercise, we will develop an ALU that will take two 2-inputs, A and B and is able to execute the following instructions:



ALUControl	Instruction
000 (add)	lw, sw
001 (subtract)	beq
000 (add)	add
001 (subtract)	sub
101 (set less than)	slt
011 (or)	or
010 (and)	and

The ALU will generate a 32-bit output that we will call ‘Result’ and an additional 1-bit flag ‘Zero’ that will be set to ‘logic-1’ if all the bits of ‘Result’ are 0. The different operations will be selected by a 3-bit control signal called ‘ALUControl’ according to the following table.

For example, when the ‘ALUControl’ input is ‘011’, the function Result = A or B should be calculated. It is easy to see that there are many values of ‘ALUControl’ for which no operation has been defined. It is not very important what the circuit does when ‘ALUControl’ has these values, since the ‘Result’ will simply be ignored in these cases. You can use this to your advantage to simplify the circuit. Right now, the described operations may look random, but once we learn more about the Instruction set architecture, these choices will make more sense.

The first order of business should be drawing a block diagram. The following is one approach to analyzing what is needed and come up with a block diagram. You are free to follow this example or come up with your own ideas. It is just important that you think about how the circuit should be implemented. Let us first examine the different commands. You should see that we have two types of instructions. The three instructions add, sub, and slt are arithmetic operations, whereas the two remaining and, or are logical operations. Therefore, we have two separate groups of operations. Now let us look at the figure above and determine for which values of ALUControl we perform an operation from which group. It should be pretty clear that when ALUControl[1] is logic-1 we select a logic operation and when ALUControl[1] is logic-0 we have an arithmetic operation. This means that the output of either group can be selected by a 4-input multiplexer that is controlled by ALUControl[1:0]. Figure below shows this distribution.

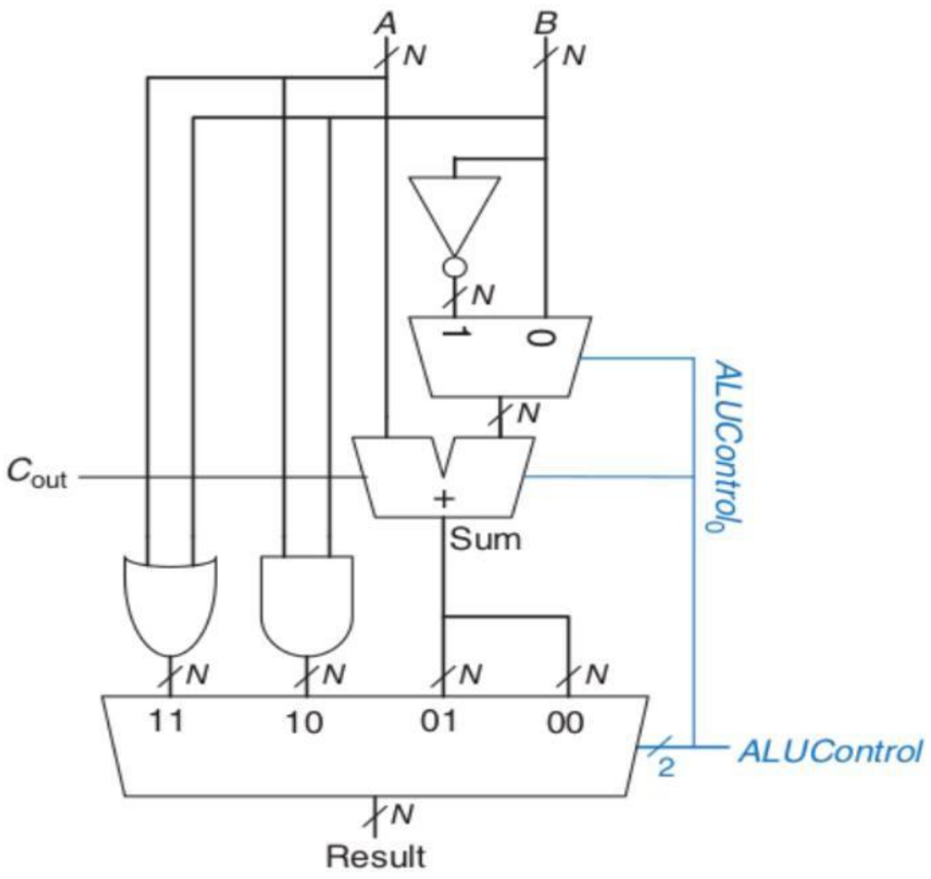


Fig 4.4 : Different ALU Functions

4.3 Control Unit

RISC-V uses 32-bit instructions. RISC-V consists of defining the following instruction formats: R-type, I-type, S-Type, B-Type, U-type, and J-type. R-type instructions operate on three registers. I-type, S-type and B-type instructions operate on two registers and a 12-bit immediate. U-type and J-type (jump) instructions operate on one 20-bit immediate.

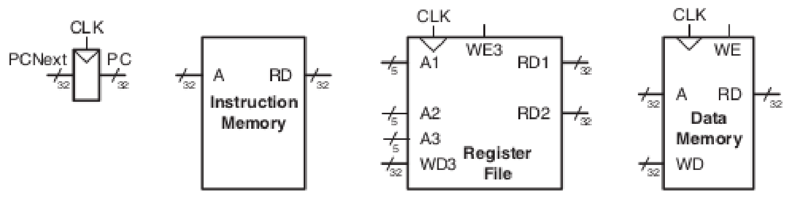


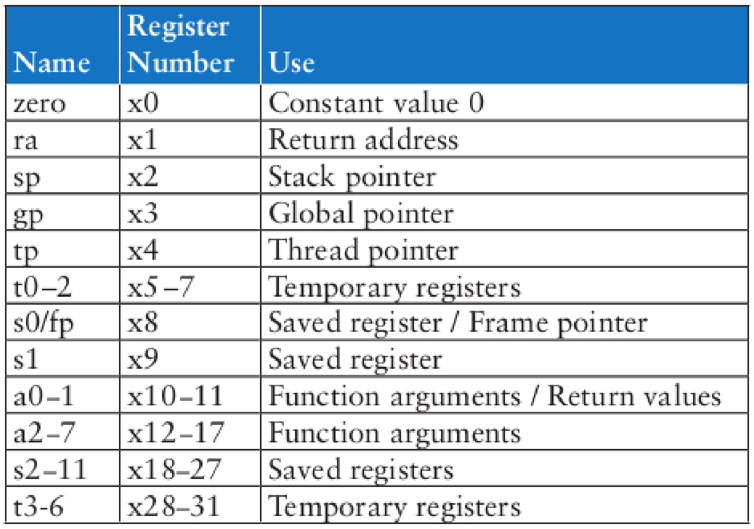
Fig.4.5 : Block diagram of RISC-V microarchitecture

A good way to design a complex system is to start with hardware containing the state elements. These elements include the memories and the architectural state (the program counter and registers). Then, add blocks of combinational logic between the state elements to compute the new state based on the current state. The instruction is read from part of memory; load and store instructions then read or write data from another part of memory. Hence, it is often convenient to partition the overall memory into two smaller memories, one containing instructions and the other containing data. Figure above shows a block diagram with the four state elements: the program counter, register file, and instruction and data memories.

In above Figure heavy lines are used to indicate 32-bit data busses. Medium lines are used to indicate narrower busses, such as the 5-bit address busses on the register file. Narrow blue lines are used to indicate control signals, such as the register file write enable. We will use this convention throughout the chapter to avoid cluttering diagrams with bus widths. Also, state elements usually have a reset input to put them into a known state at start-up. Again, to save clutter, this reset is not shown.

The program counter is an ordinary 32-bit register. Its output, PC, points to the current instruction. Its input, PCNext, indicates the address of the next instruction.the instruction memory has a single read port. It takes a 32-bit instruction address input, A, and reads the 32-bit data (i.e., instruction) from that address onto the read data output, RD.

The 32-element × 32-bit register file has two read ports and one write port. The read ports take 5-bit address inputs, A1 and A2, each specifying one of 2 = 32 registers as source 5 operands. They read the 32-bit register values onto read data outputs RD1 and RD2, respectively. The write port takes a 5-bit address input, A3; a 32-bit write data input, WD; a write enable input, WE3; and a clock. If the write enable is 1, the register file writes the data into the specified register on the rising edge of the clock.



Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0–2	x5 –7	Temporary registers
s0/fp	x8	Saved register / Frame pointer
s1	x9	Saved register
a0–1	x10–11	Function arguments / Return values
a2–7	x12–17	Function arguments
s2–11	x18–27	Saved registers
t3-6	x28–31	Temporary registers

Fig.4.6 : 32 bit register format in RISC-V Microarchitecture.

The data memory has a single read/write port. If the write enable, WE, is 1, it writes data WD into address A on the rising edge of the clock. If the write enable is 0, it reads address A onto RD.

The instruction memory, register file, and data memory are all read combinationally. In other words, if the address changes, the new data appears at RD after some propagation delay; no clock is involved. They are written only on the rising edge of the clock. In this fashion, the state of the system is changed only at the clock edge. The address, data, and write enable must be set up sometime before the clock edge and must remain stable until a hold time after the clock edge. Because the state elements change their state only on the rising edge of the clock, they are synchronous sequential circuits.

The microprocessor is built of clocked state elements and combinational logic, so it too is a synchronous sequential circuit. Indeed, the processor can be viewed as a giant finite state machine, or as a collection of simpler interacting state machines.

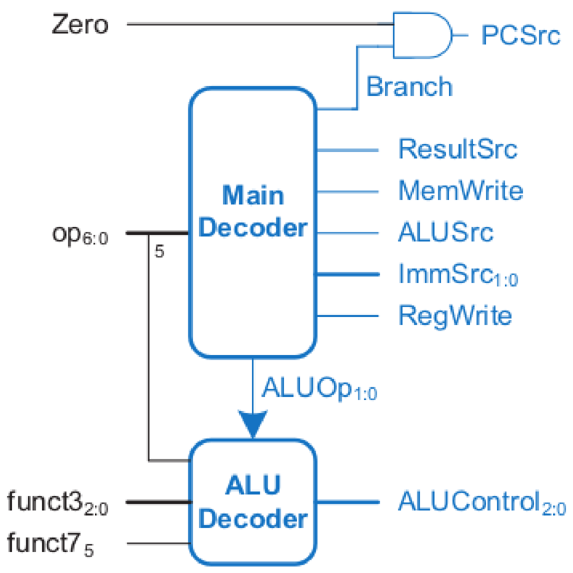


Fig. 4.7 : Block diagram of Decoders.

The control unit computes the control signals based on the opcode and funct fields of the instruction, Instr[31:25], Instr[14:12] and Instr[6:0]. Most of the control information comes from the opcode, but for further operations function fields are used. Thus, we will simplify our design by factoring the control unit into two blocks of combinational logic, as shown in Figure Above.

4.4 MAIN Decoder

The table below is a truth table for the main decoder that summarizes the control signals as a function of the opcode. All R-type instructions use the same main decoder values; they differ only in the ALU decoder output. Recall that, for instructions that do not write to the register file (e.g., S-type and B-type), the ResultSrc control signals is don't care (X); the address and data to the register write port do not matter because RegWrite is not asserted. The logic for the decoder can be designed using your favorite techniques for combinational logic design.

Instruction	Op	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
lw	0000011	1	00	1	0	1	0	00
sw	0100011	0	01	1	1	x	0	00
R-type	0110011	1	xx	0	0	0	0	10
beq	1100011	0	10	0	0	x	1	01

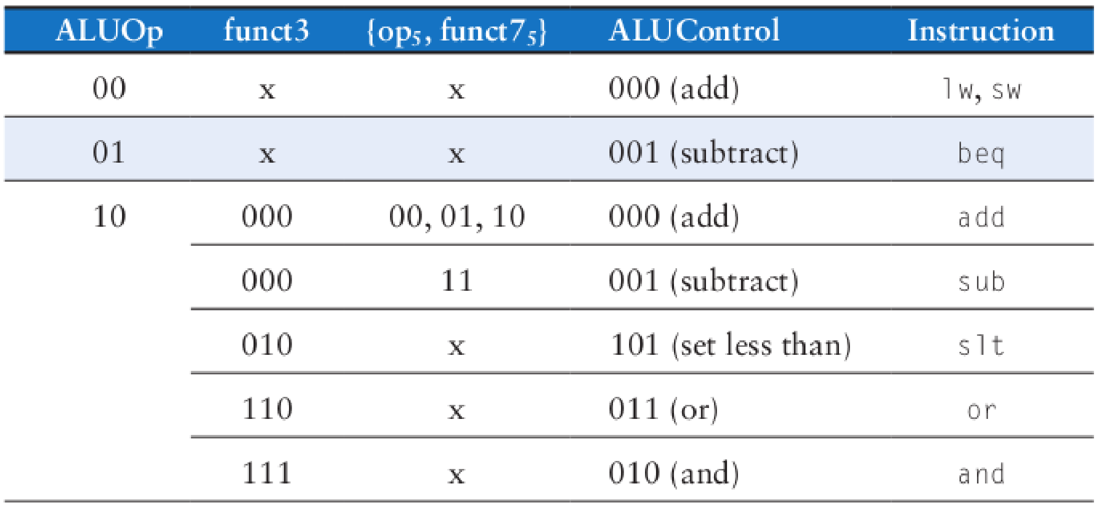
Fig.4.8 : Truth table for Main Decoder.

4.5 ALU Decoder

The main decoder computes most of the outputs from the opcode. It also determines a 2-bit ALUOp signal. The ALU decoder uses this ALUOp signal in conjunction with the funct field and opcode bit to compute ALUControl. The meaning of the ALUOp signal is given in Table below

ALUOp	**Meaning**
00	add
01	subtract
10	look at funct fields and opcode bit
11	N/A

The logic of ALUControl was covered in the above chapter. When ALUOp is 00 or 01, the ALU should add or subtract, respectively. When ALUOp is 10, the decoder examines the function fields and operand bit to determine the ALUControl. The control signals for each instruction were described as we built the datapath.



ALUOp	funct3	$\{op_5, funct7_5\}$	ALUControl	Instruction
00	x	x	000 (add)	`lw`, `sw`
01	x	x	001 (subtract)	`beq`
10	000	00, 01, 10	000 (add)	`add`
	000	11	001 (subtract)	`sub`
	010	x	101 (set less than)	`slt`
	110	x	011 (or)	`or`
	111	x	010 (and)	`and`

Fig. 4.9 : Truth table for the ALU decoder.

4.6 Flow diagram of RISC V architecture

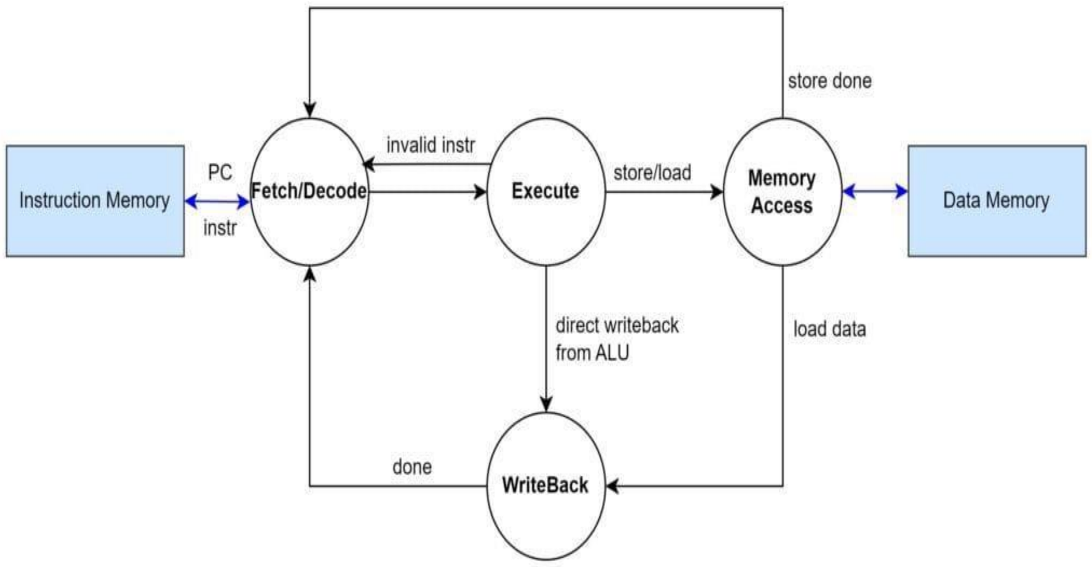


Fig. 4.10: Flow diagram of RISC V architecture

- In this project the instruction is written in form of machine code and that code is written in the instruction memory.
- From that memory the instruction will be fetched.
- That instruction is decoded in the decoder block.
- That decoded instruction is executed in the ALU block.
- The output of the ALU block is then stored in data memory.

4.7 Flow chart of R type Instruction execution

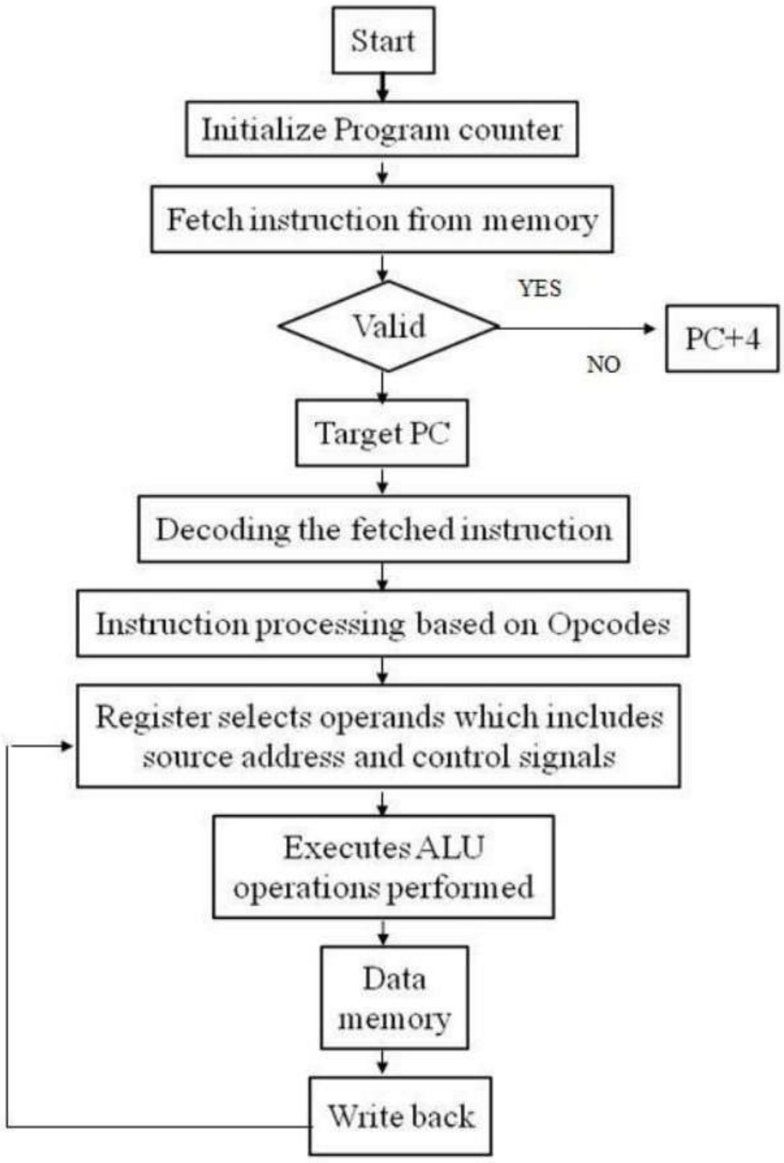

Start
Initialize Program counter
Fetch instruction from memory
Valid
YES
NO
PC+4
Target PC
Decoding the fetched instruction
Instruction processing based on Opcodes
Register selects operands which includes source address and control signals
Executes ALU operations performed
Data memory
Write back

Fig.4.11 : Flow chart of R Type instruction execution.

4.8 APPLICATIONS

- Educational Purposes
- Embedded Systems
- Prototyping and Research
- Custom Hardware Development
- Low-Power Applications
- Control Systems

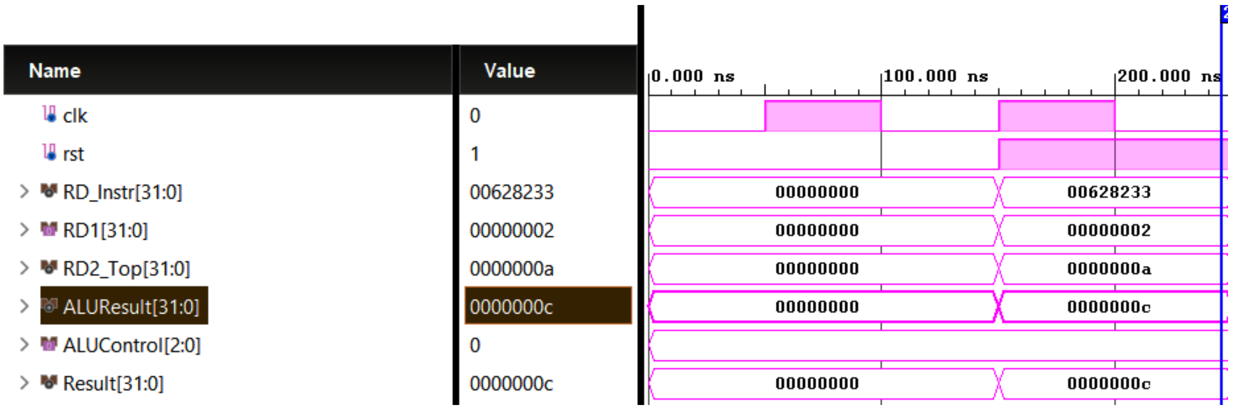
4.9 Advantages

- Simplicity
- Predictable Performance
- Low Latency
- No Control Hazards
- Ease of Understanding
- No Data Forwarding or Hazard Resolution
- Minimal Hardware Complexity

CHAPTER-5

RESULT AND DISCUSSION

5.1 RESULT OF ADD OPERATION

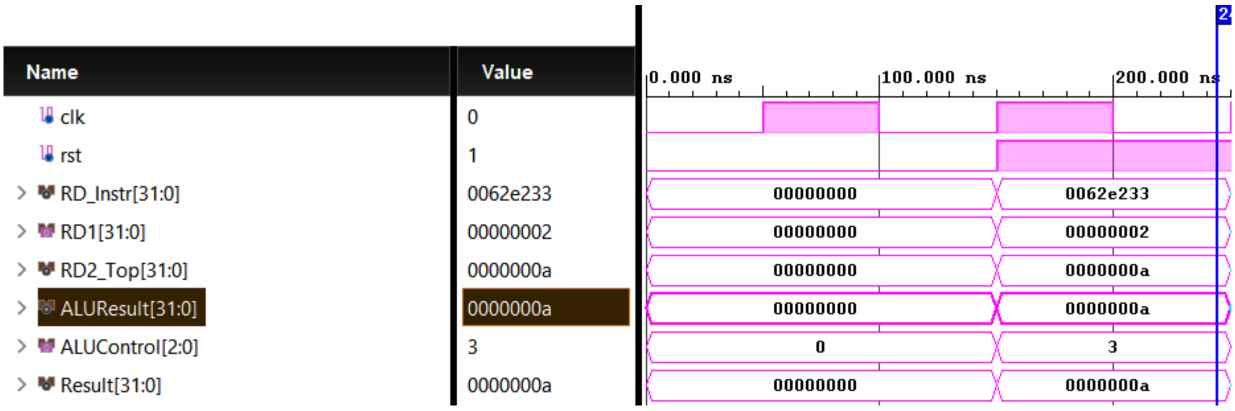


Name	Value		
clk	0		
rst	1		
RD_Instr[31:0]	00628233	00000000	00628233
RD1[31:0]	00000002	00000000	00000002
RD2_Top[31:0]	0000000a	00000000	0000000a
ALUResult[31:0]	0000000c	00000000	0000000c
ALUControl[2:0]	0		
Result[31:0]	0000000c	00000000	0000000c

Fig.5.1 : Result of ADD-Operation

In this project it used the R type instruction in that we execute the add Operation. Machine code for the add operation is 00628233. And the machine code will be decoded and get the output in result. The RD1 consist of the rs1 which is 2 and 0010 in binary form and RD2 consist of rs2 which is A means decimal 10 in number and 1010 in binary form. After the add operation it get result as C which is 1100 as a result.

5.2 RESULT OF OR OPERATION

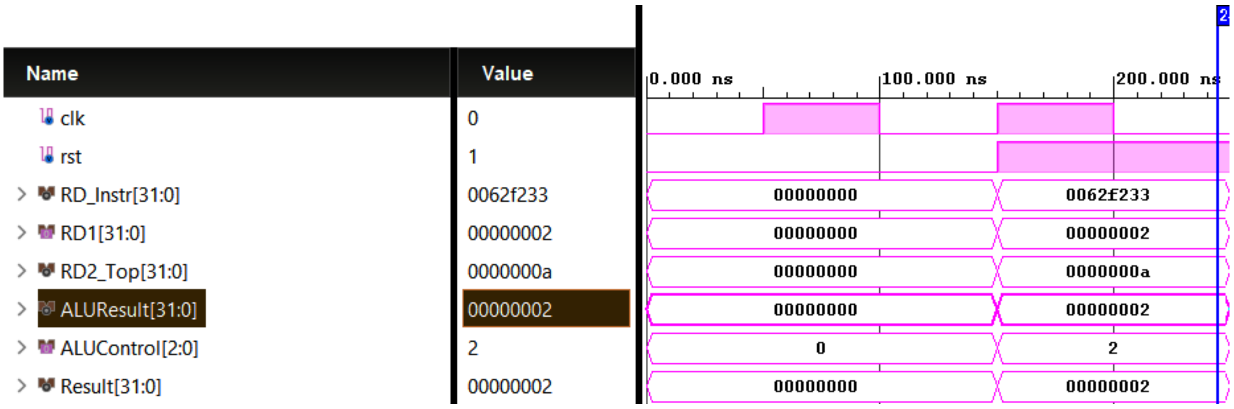


Name	Value		
clk	0		
rst	1		
RD_Instr[31:0]	0062e233	00000000	0062e233
RD1[31:0]	00000002	00000000	00000002
RD2_Top[31:0]	0000000a	00000000	0000000a
ALUResult[31:0]	0000000a	00000000	0000000a
ALUControl[2:0]	3	0	3
Result[31:0]	0000000a	00000000	0000000a

Fig.5.2 : Result of OR-Operation

In this Project it used R type instruction in that it execute the OR Operation. Machine code for the add operation is 0062E233. And the machine code will be decoded and get the output in result. The RD1 consist of the rs1 which is 2 and 0010 in binary form and RD2 consist of rs2 which is A means decimal 10 in number and 1010 in binary form. After the OR operation it get result as A which is 1010 as a result.

5.3 RESULT OF AND OPERATION

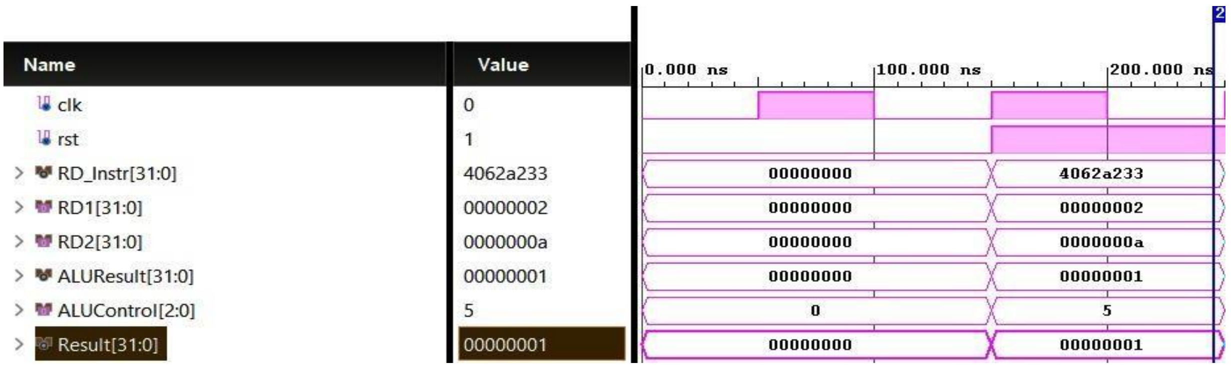


Name	Value
clk	0
rst	1
RD_Instr[31:0]	0062f233
RD1[31:0]	00000002
RD2_Top[31:0]	0000000a
ALUResult[31:0]	00000002
ALUControl[2:0]	2
Result[31:0]	00000002

Fig.5.3 : Result of AND-Operation

This Project used the R type instruction in that it execute the AND Operation. Machine code for the add operation is 0062F233. And the machine code will be decoded and get the output in result. The RD1 consist of the rs1 which is 2 and 0010 in binary form and RD2 consist of rs2 which is A means decimal 10 in number and 1010 in binary form. After the AND operation we get result as 2 which is 0010 as a result.

5.4 RESULT OF SLT OPERATION



Name	Value
clk	0
rst	1
RD_Instr[31:0]	4062a233
RD1[31:0]	00000002
RD2[31:0]	0000000a
ALUResult[31:0]	00000001
ALUControl[2:0]	5
Result[31:0]	00000001

Fig.5.4 : Result of SLT-Operation

This Project used the R type instruction in that it execute the select less than(SLT) Operation. Machine code for this operation is 0062A233. And the machine code will be decoded and get the output in result. The RD1 consist of the rs1 which is 2 and 0010 in binary form and RD2 consist of rs2 which is A means decimal 10 in number and 1010 in binary form. here the RS1 is less than RS2 so there is 1 as the output of SLT operation.

CHAPTER-6

CONCLUSION AND FUTURE SCOPE

The design and implementation of a single-cycle RISC processor using Verilog and Vivado successfully demonstrated the fundamental principles of processor architecture. By supporting a reduced instruction set and completing each instruction in a single clock cycle, the processor highlights the simplicity and efficiency of RISC-based designs. Through modular development of key components, including the Program Counter, ALU, Register File, Instruction Memory, and Control Unit, the processor's functionality was thoroughly verified via simulation in Vivado.

It serves as an excellent learning tool for understanding the fundamentals of processor operation. This implementation lays a strong foundation for exploring advanced concepts like pipelining and multi-cycle processors, and it also showcases the utility of HDLs like Verilog and tools like Vivado in digital design and FPGA prototyping.

REFERENCE

[1].Mamun B, Shabiul I. and Sulaiman S,"A Single Clock Cycle MIPS RISC Processor Design using VHDL 2019"

[2]. Hamblen J." Synthesis, Simulation, and Hardware Emulation to Prototype a Pipelined RISC Computer System 2021"

[3]. Zainalabedin N,"VHDL for Modeling and Design of Processing Units"

[4]. Takanori M, Satoshi A and Masaaki I, "A Multithread Processor Architecture Based on the Continuation"

[5]. Kasuga-Koen, Kasuga, Fukuoka, " The Innovative Architecture for Future Generation High-Performance Processors and Systems"

[6]. Virendra S. and Michiko I,"Instruction Based SelfTesting of Delay Faults in Pipelined Processors", IEEE Transaction on VLSI systems, vol. 14, no.11,pp.1203-1215.

[7]. Patterson A. and Hennessy J,"Computer Organization & Design", Morgan Kaufmann Publishers, 2022

[8]. Peter J Ashenden,"Digital Design, An embedded systems approach using Verilog", Morgan Kaufmann Publishers,2020