
Manipal Center of Excellence
MULTI CYCLE PIPELINED
RISCv BASED PROCESSOR
mirafra
TECHNOLOGIES
BATCH-9
Team - 10

Date	Document Version	Remarks	Drafted by
10-September-2024	Version 1.0	Design Overview	Team 10
11-September-2024	Version 2.0	Updated microarchitecture	Team10

Table of Contents

Chapter 1- Design Overview

1.1 Introduction to RISC V processor

The RISC-V (RV32I) instruction set uses 32-bit instructions in little-endian format. It has 32 general-purpose registers, with register 0 always set to zero, and a 32-bit program counter that increments by one for word-aligned instructions. RISC-V is efficient, easy to pipeline, and requires minimal hardware and power. Techniques like loop unrolling and compiler scheduling help improve performance. The RV32I supports six instruction formats: R-type, U-type, I-type, B-type, J-type, and S-type.

1.2 Project Overview

- **Design a 5-Stage Pipelined Processor:**
  Implement a RISC-V processor with five pipeline stages: Fetch, Decode, Execute, Memory Access, and Write Back.
- **ALU Operations:**
  Implement the ALU to handle the following operations:
  - ADD
  - SUB
  - AND
  - OR
  - XOR
  - NOP
- **RISC-V Instructions:**
  Implement RISC-V instructions for each ALU operation (ADD, SUB, AND, OR ,XOR, and NOP).

- **Write a Testbench:**
  Create a testbench to verify the correct functionality of the processor by simulating all ALU operations.

1.3 INSTRUCTION SET

ALU INSTRUCTIONS (ADD, SUB, AND, OR, XOR)

Here R0 –R32 represent the registers in the register set.

Bit position	31- 25	24- 20	19 - 15	14 - 12	11 - 7	6 - 0	
OPERATION	FUNC7	RS2	RS1	FUNC3	RD	OPCODE	STATUS
ADD	0000000	R0-R32	R0 -R32	000	R0 -R32	0110011	RD = RS1 + RS2
SUB	0100000	R0-R32	R0 -R32	000	R0 -R32	0110011	RD = RS1 - RS2
AND	0000000	R0-R32	R0 -R32	110	R0 -R32	0110011	RD = RS1 & RS2
OR	0000000	R0-R32	R0 -R32	111	R0 -R32	0110011	RD = RS1 | RS2
XOR	0000000	R0-R32	R0 -R32	100	R0 -R32	0110011	RD = RS1 ^ RS2

ADDI INSTRUCTION

Here R0 is the register which is hardwired to 32'd0 and it used to load an immediate value into the register set directly

Bit position	31 - 25	24 -20	19 - 15	14 - 12	11 - 7	6 - 0	
OPERATION	FUNC7	IMM VALUE	RS1	FUNC3	RD	OPCODE	STATUS
ADDI	0000000	VALUE	R0(00000)	000	R0 -R32	1110011	RD = RS1(R0) + VALUE

HALT INSTRUCTION

Bit position	31 - 25	24 - 20	19 - 15	14 - 12	11 - 7	6 - 0	
OPERATION	FUNC7	RS2	RS1	FUNC3	RD	OPCODE	STATUS
ADDI	0000000	00000	00000	000	00000	1111111	PC STOPS

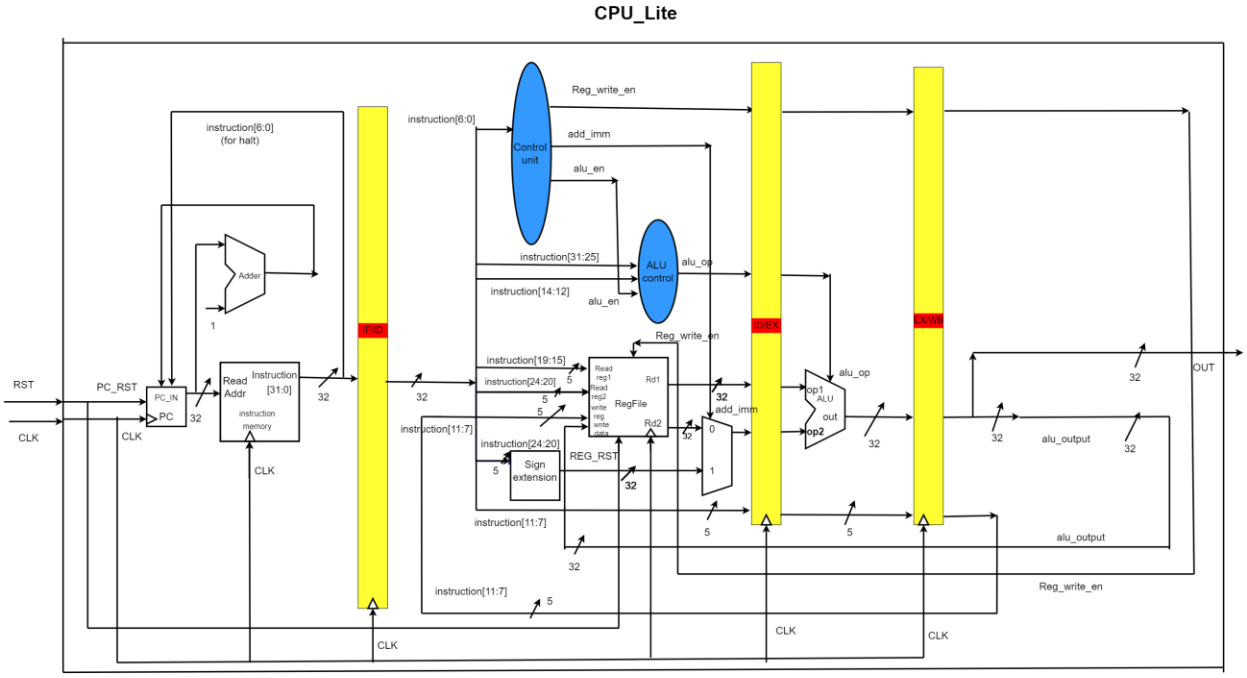
NO OPERATION INSTRUCTION

This instruction is used whenever we are executing an instruction that is dependent on the previous instruction

Bit position	31 - 25	24 - 20	19 - 15	14 - 12	11 - 7	6 - 0	
OPERATION	FUNC7	RS2	RS1	FUNC3	RD	OPCODE	STATUS
NOP	0000000	00000	00000	000	00000	0000000	The CPU remains idle

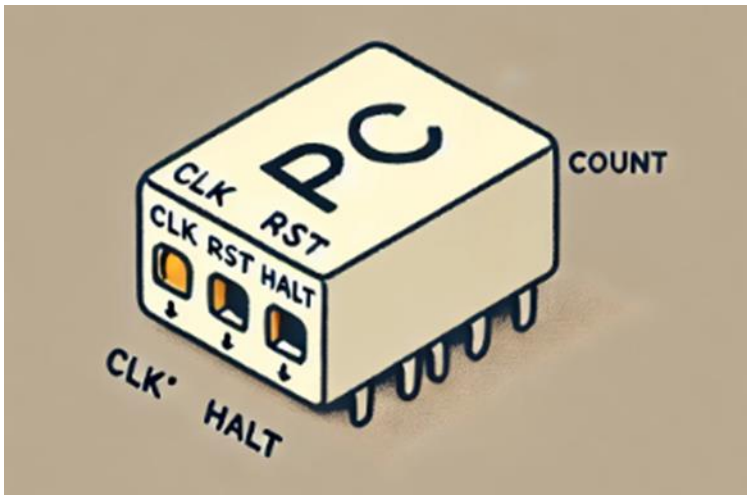
1.4 INSTRUCTION SET

1.3 Microarchitecture



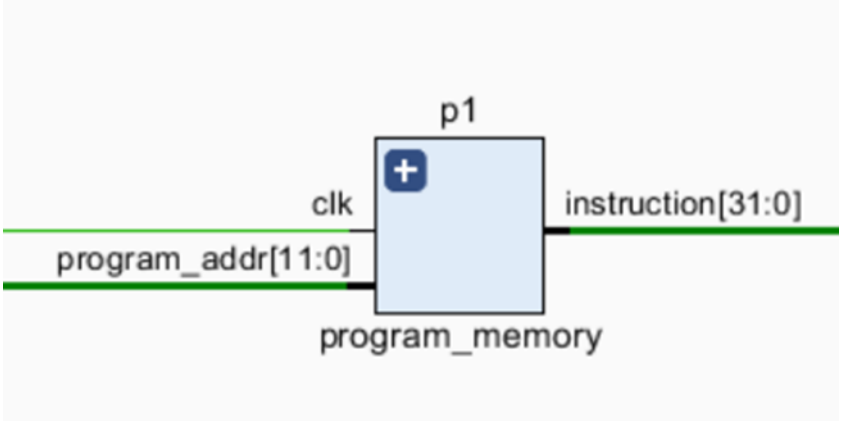
1.4.1 Different components of pipelined RISC V processor

Program Counter



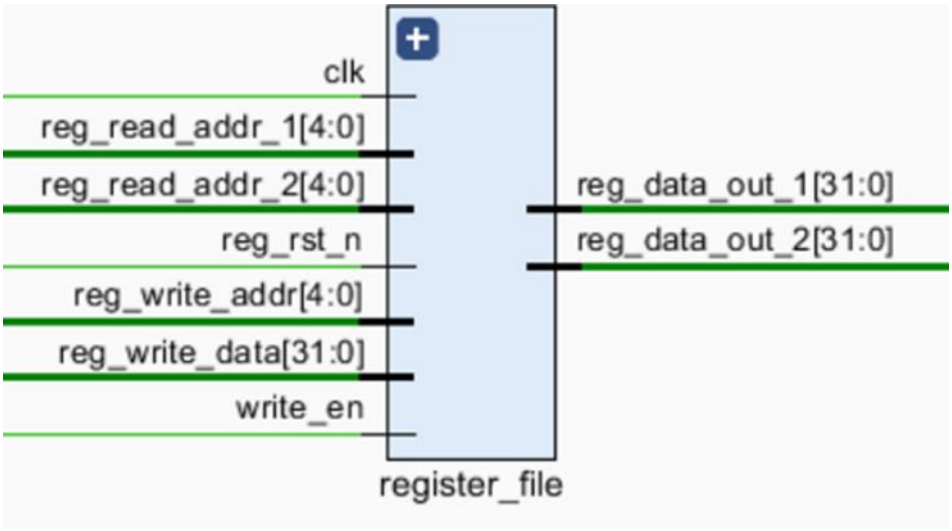
The program counter, also known as the instruction pointer or simply PC, is a 32-bit special register in a CPU that tracks where the next instruction is in memory. It tells the CPU which instruction to fetch and execute next. The PC is automatically loaded with zero when the CPU is reset.

Program Memory



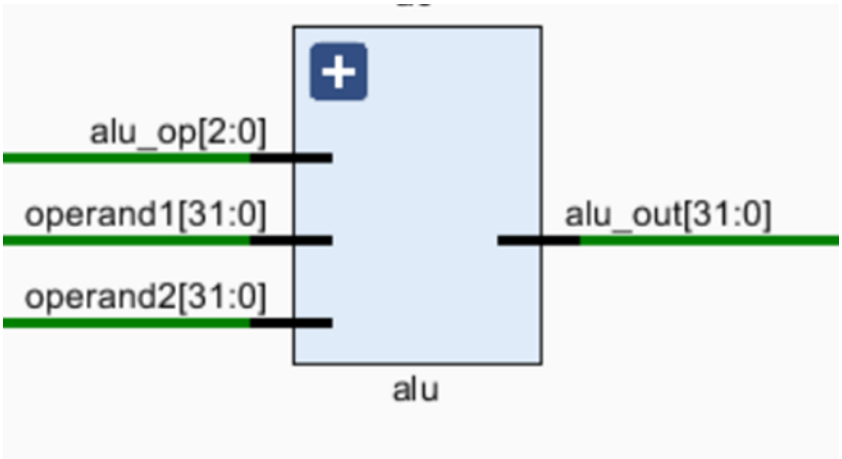
Program Memory holds the program instructions and keeps them unchanged during normal operation. It's a type of non-volatile memory, meaning it retains data even when the power is off. When the CPU needs an instruction, it looks up the address in Program Memory and retrieves the instruction directly

Register File



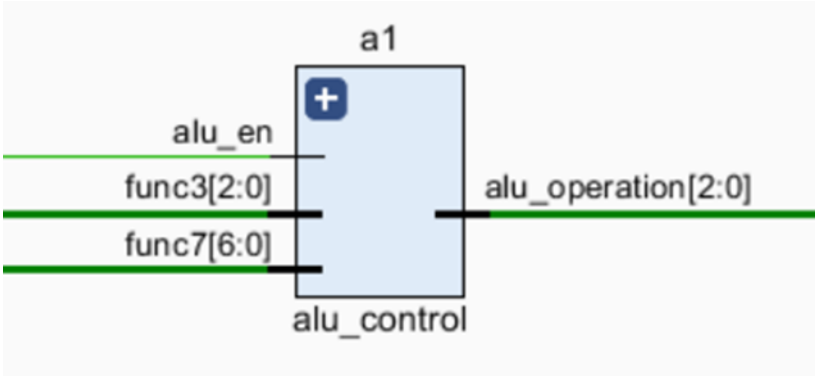
The 32x32 register file is a vital component containing 32 registers, each 32 bits in width, used for storing operands and operation results. It allows the processor to efficiently perform read and write operations. The register file uses 5-bit addresses to access Rs1 and Rs2 and operates in synchronization with a clock signal. While data can be read from any register, writing data to a register occurs in alignment with the clock signal.

Arithmetic Logic Unit (ALU)



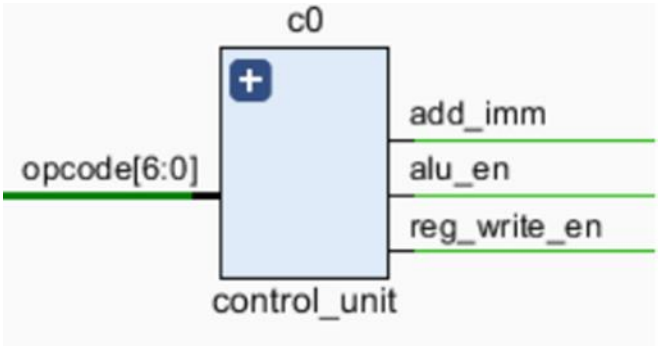
Performs arithmetic and logical operations on data from the register file. The ALU operations are controlled by the ALU control unit and include addition, subtraction, AND, OR, etc.

ALU Control



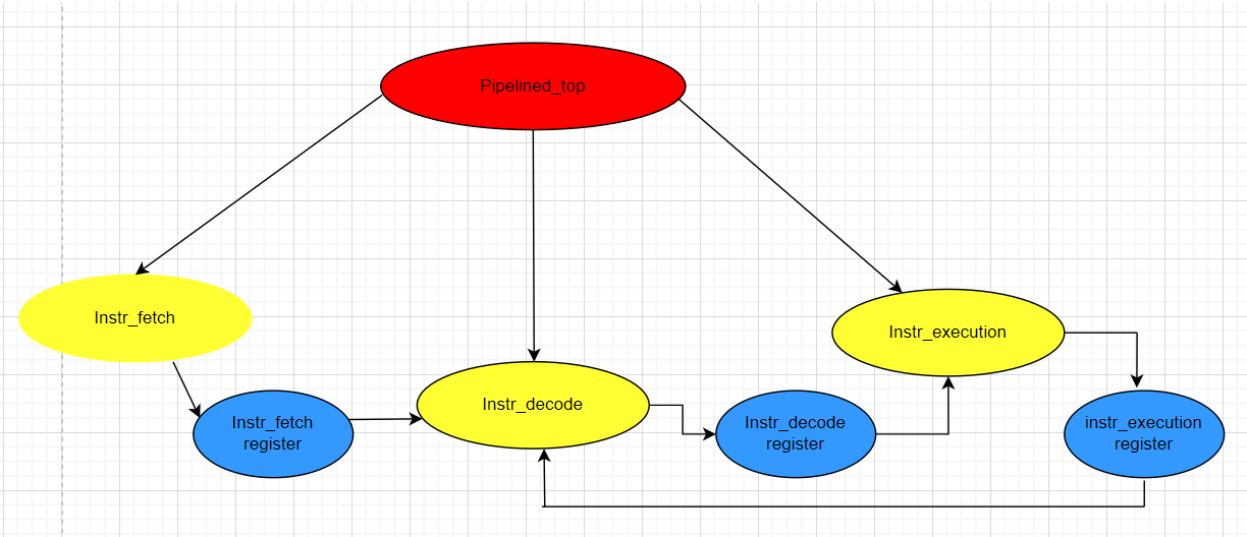
Generates control signals for the ALU, specifying which operation to perform based on the instruction opcode.

Control Unit



The control unit is a crucial component of the processor, responsible for managing the select lines of multiplexers (mux). By doing so, it determines the DataPath configuration required for executing different instructions.

1.5 RTL HEIRARCHY



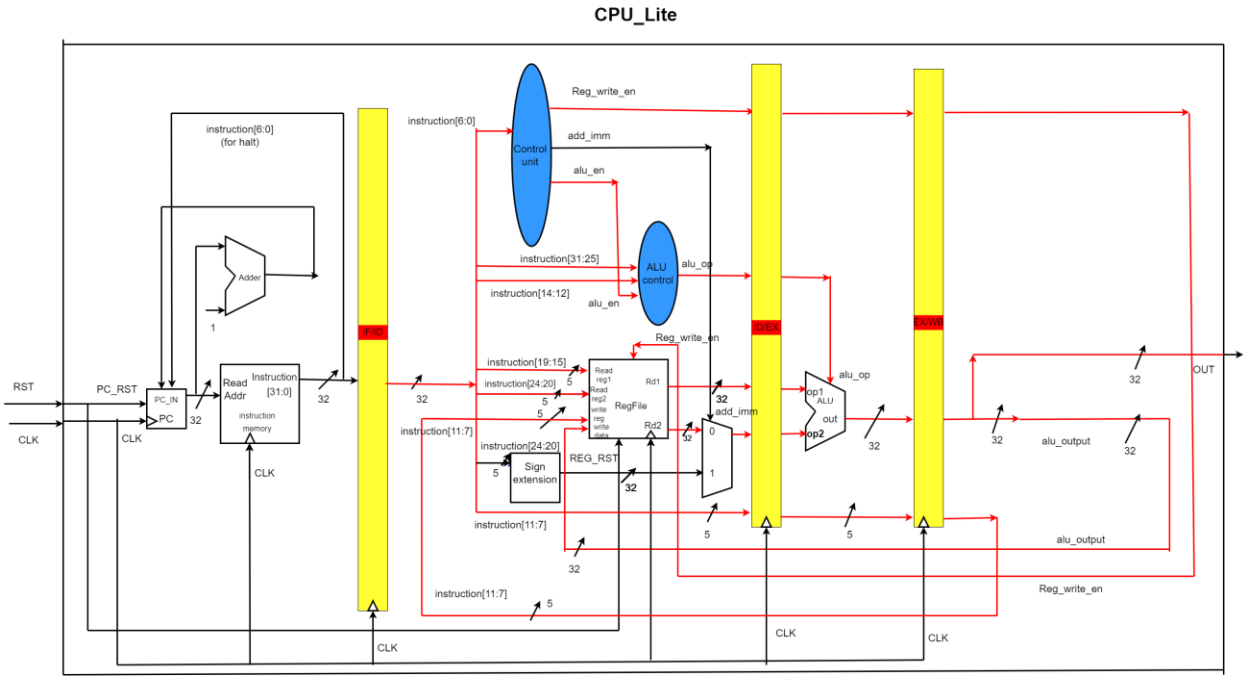
The design is divided into three stages: **Instruction Fetch (Instr_fetch)**, **Instruction Decode (Instr_decode)**, and **Instruction Execution (Instr_execution)**. To ensure that all these stages work in a **pipelined** manner, each stage is provided with a separate clock. This allows all stages to operate simultaneously on different instructions.

- **Instr_fetch**: Retrieves the next instruction from memory.
- **Instr_decode**: Decodes the fetched instruction to determine the necessary control signals and operands.
- **Instr_execution**: Executes the instruction, using the decoded signals and operands.
- In addition to these stages, you have depicted **registers** between each stage:
- **Instr_fetch register**: Stores the fetched instruction to be used in the decode stage.
- **Instr_decode register**: Holds the decoded instruction signals for the execution stage.
- **Instr_execution register**: Keeps the results after execution for further use or output.

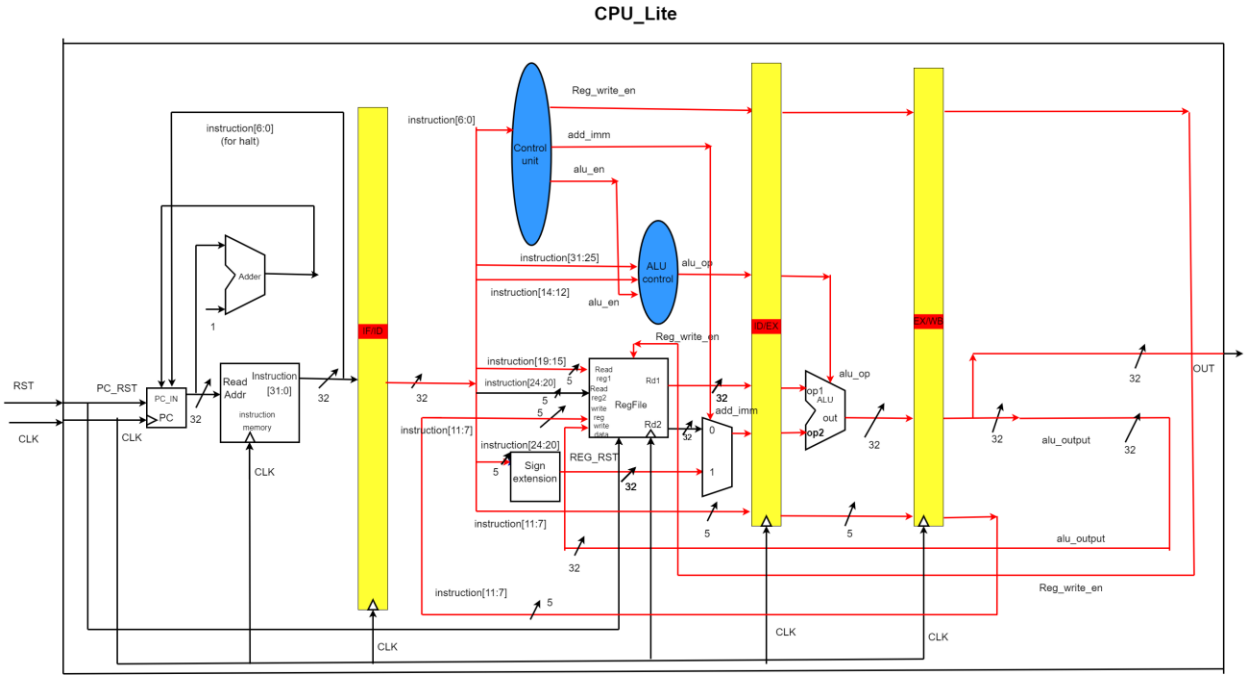
By providing each stage with a dedicated clock, the design ensures that the pipeline flows smoothly and that each stage completes its operation in a synchronized manner, leading to higher throughput and overall performance.

1.6 Datapath for different instructions:

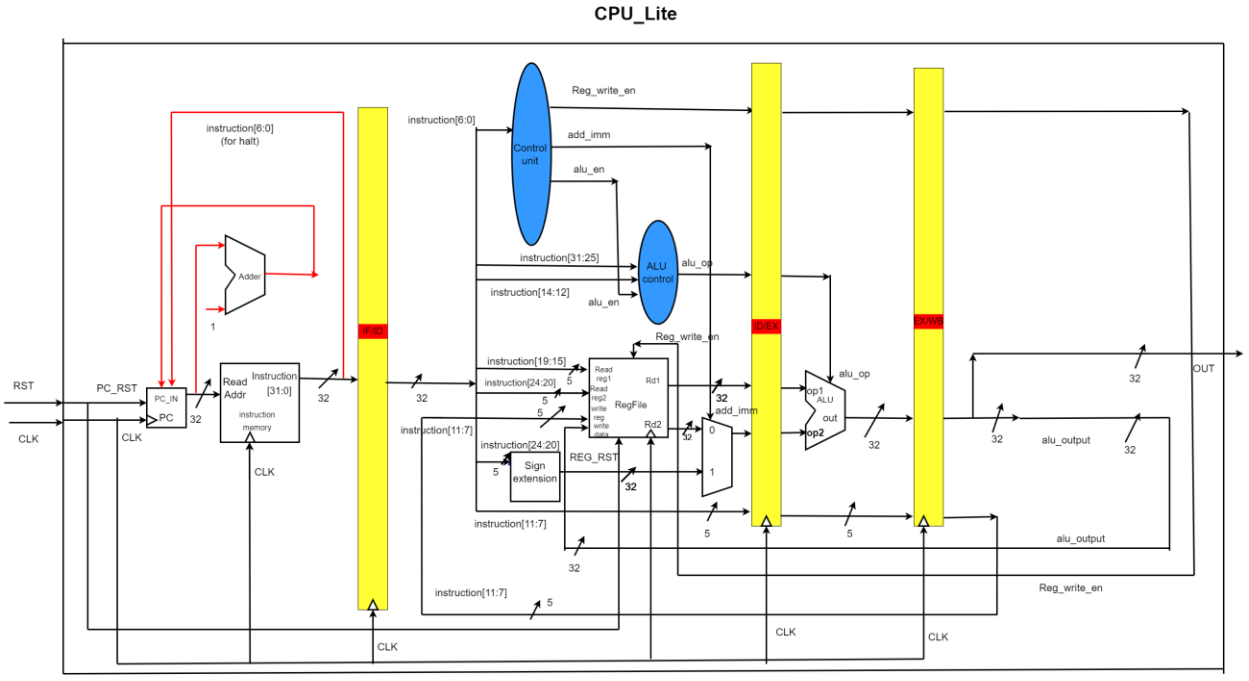
1.6.1 ALU instruction*:*



1.6.2 ADDI instruction:



1.6.3 Halt instruction:



1.7 Control unit truth table:

INPUTS	OUPUTS		
OPCODE	ADD_IMM	REG_WRITE_EN	ALU_EN
7'b0110011	0	1	1
7'b1110011	1	1	1
Default	0	0	0

1.8 ALU Control unit truth table:

INPUTS		OUTPUT
FUNC7	FUNC3	ALU_OPERATION
7'b0000000	000	3'd0
7'b0100000	000	3'd1
7'b0000000	110	3'd2
7'b0000000	111	3'd3
7'b0000000	100	3'd4

1.9 ALU OPERATION:

INPUT(OPCODE)	OPERATION
3'd0	ADDITION
3'd1	SUBTRACTION
3'd2	BITWISE AND
3'd3	BITWISE OR
3'd4	BITWISE XOR

Chapter 2- Simulation Waveform

2.1 ADD Immediate Operation

Example: programmemory[0] <= 32'b0000000_00110_00000_000_00010_1110011;

Operation: R2=R0+6

The content of register R0 is added with immediate value 6 and stored in register R2. Register R0 is hardwired to 0. Using add immediate operation we load the value 6 into register R2.

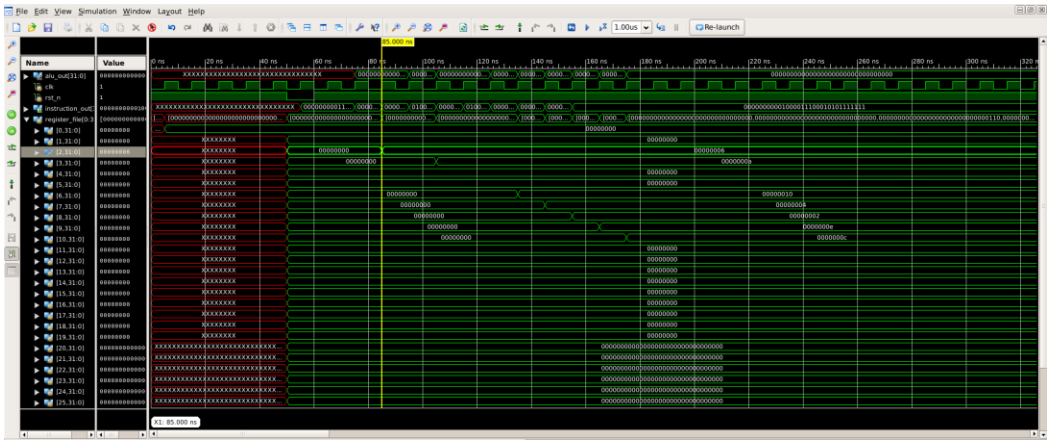


Fig1: ADD immediate value to Register 2

Example: programmemory[1] <= 32'b0000000_01010_00000_000_00011_1110011

Operation: R3=R0+10

The content of register R0 is added with immediate value 10 and stored in register R3. Register R0 is hardwired to 0. So, using add immediate operation we load the value 10 into register R3

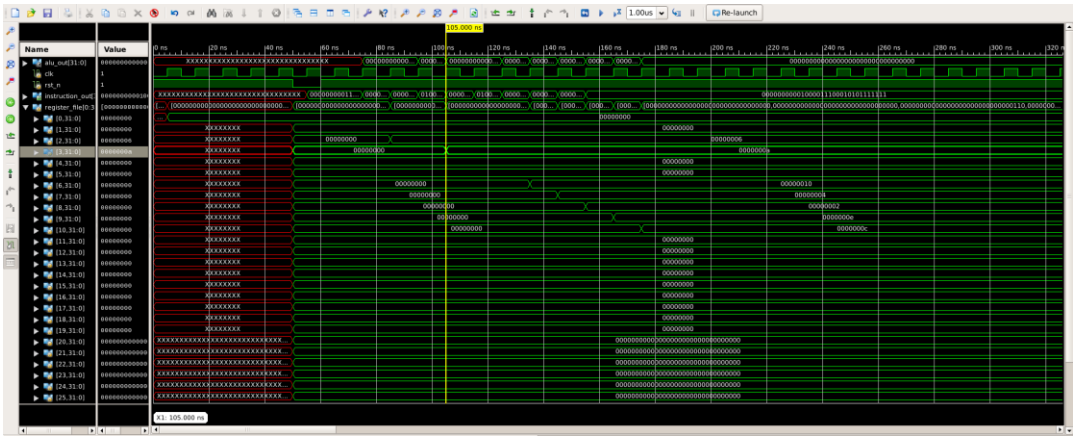


Fig2: ADD immediate value to Register 3

2.2 NOP Operation

Example: programmemory[2] <= 32'b0000000_00010_00011_000_00110_0000000;

This operation is done to prevent data hazards in the CPU

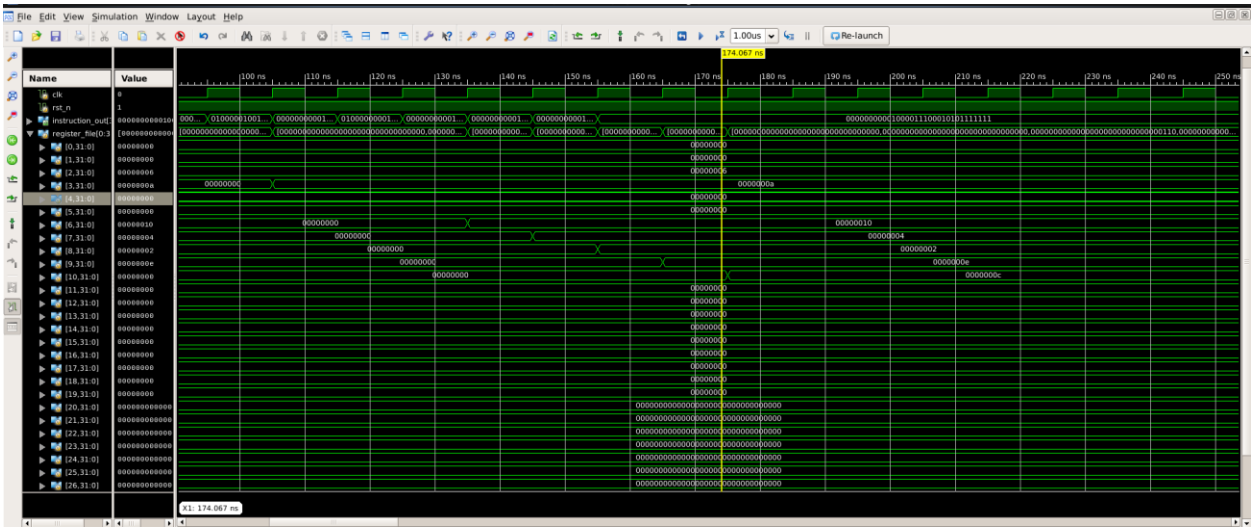


Fig3: NOP operation

2.3 ADD OPERATION

Example: programmemory[4] <= 32'b0000000_00010_00011_000_00110_0110011

Operation: R6=R3+R2

The content of Register R3 and R2 is added and stored in Register R6.

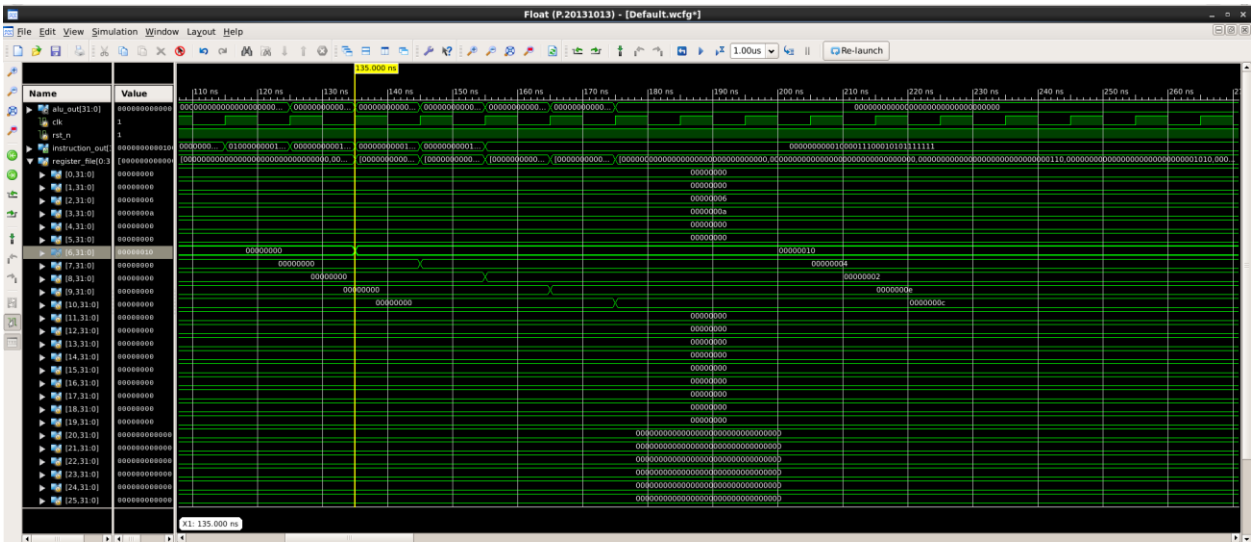


Fig4: ADD operation

2.4 SUB OPERATION

Example: programmemory[5] <= 32'b0100000_00010_00011_000_00111_0110011

Operation: R7=R3-R2

The content of Register R3 and R2 is subtracted and stored in Register R7.

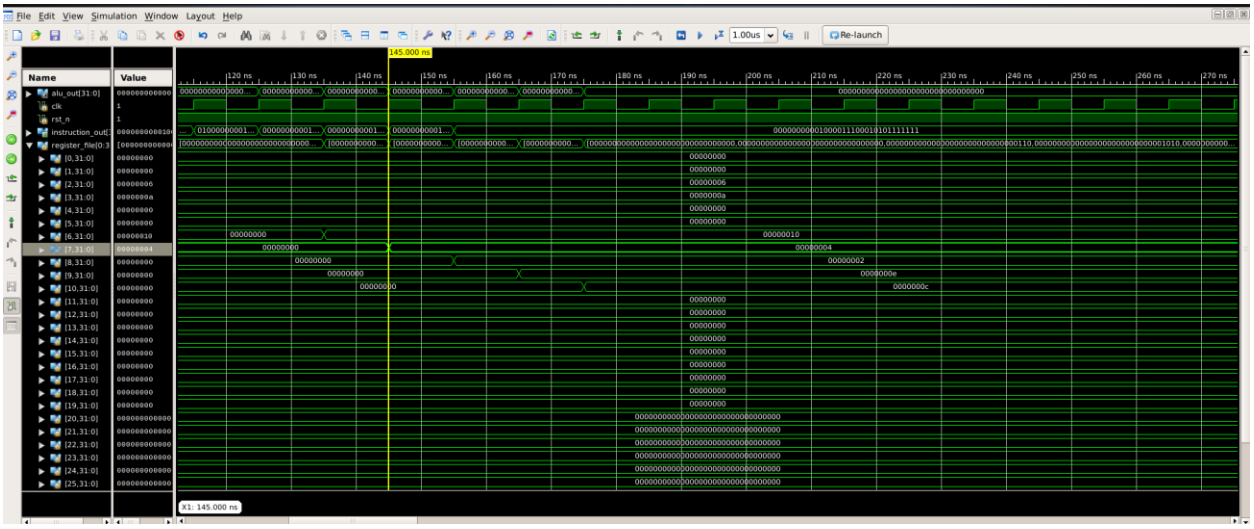


Fig5: SUB operation

2.3 AND OPERATION

Example: programmemory[6] <= 32'b0000000_00010_00011_110_01000_0110011

Operation: R8=R3&R2

The content of Register R3 and R2 is bitwise ANDed and stored in Register R8.

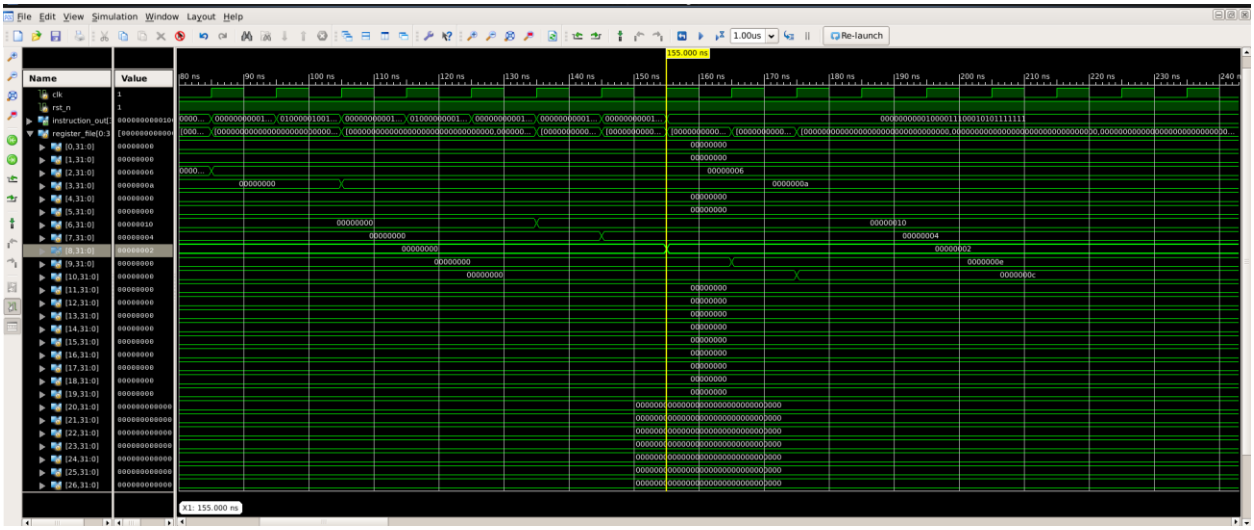


Fig6:AND operation

2.4 OR OPERATION

Example: programmemory[7] <= 32'b0000000_00010_00011_111_01001_0110011

Operation: R9=R3|R2

The content of Register R3 and R2 is bitwise ORed and stored in Register R9.

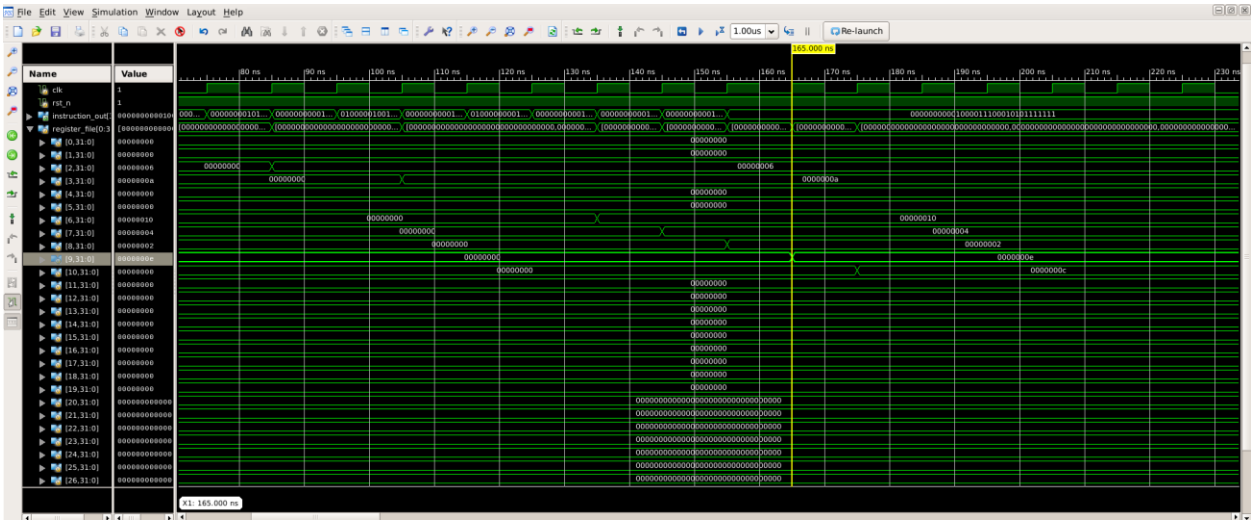


Fig7: OR operation

2.5 XOR OPERATION

Example: programmemory[8] <= 32'b0000000_00010_00011_100_01010_0110011

Operation: R10=R3^R2

The content of Register R3 and R2 is bitwise XORed and stored in Register R10.

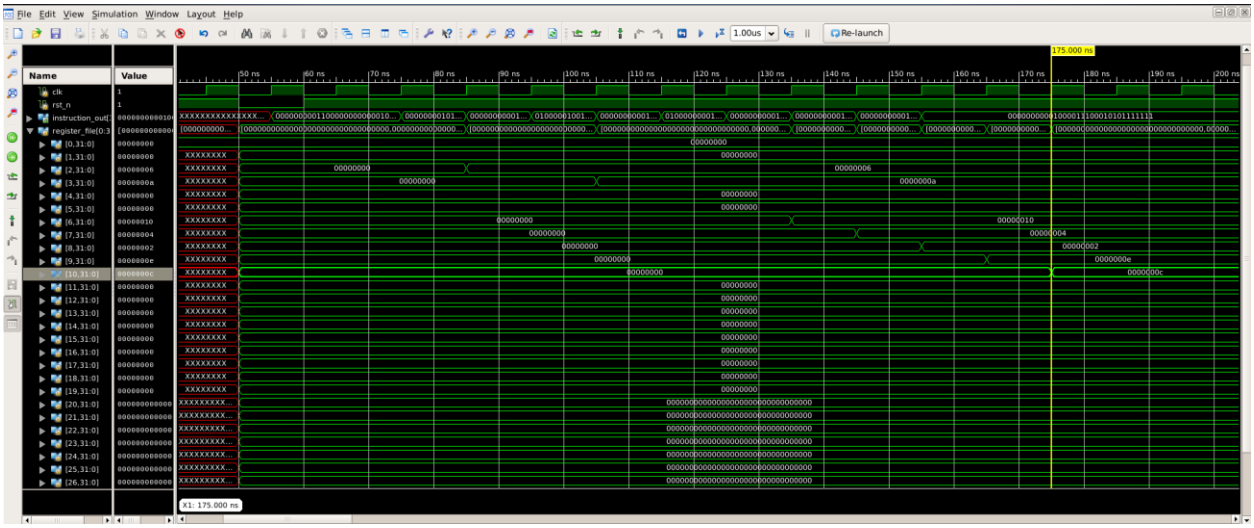


Fig8:XOR operation

2.6 HALT OPERATION

Example: programmemory[9] <= 32'b0000000_00010_00011_100_01010_1111111

Using this instruction the PC is halted, and the CPU goes into halt state.

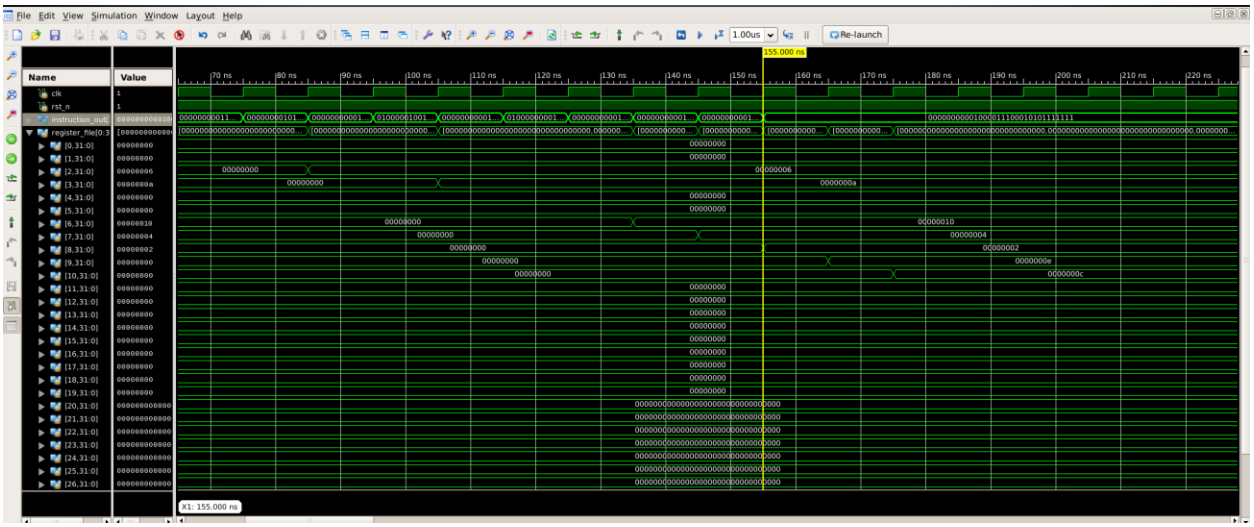


Fig9: HALT operation