

EE 224 - Multicycle RISC CPU



Design and Implementation of a Mutlicycle RISC CPU

3rd December, 2024

Name	Roll No.
Athav Vantan B.	23B1306
Kaarmukilan B.	23B1215
Himanshi Agrawal	23B1214
Dishika Nawal	23B1307

Course Instructor: Prof. Virendra singh

IITB-CPU Design Report

Problem Statement

Design a computing system, IITB-CPU, whose instruction set architecture is provided. Use VHDL as HDL to implement. IITB-CPU is a 16-bit very simple computer developed for teaching purposes. It is an 8-register, 16-bit computer system, i.e., it can process 16 bits at a time. It should use point-to-point communication infrastructure.

The IITB-CPU is based on the Little Computer Architecture. It is an 8-register, 16-bit computer system with 8 general-purpose registers (**R0** to **R7**). The program counter (PC) always points to the next instruction. All addresses are short word addresses (i.e., address 0 corresponds to the first two bytes of main memory, address 1 corresponds to the second two bytes, and so on). The architecture uses a condition code register with two flags:

- **Carry flag (C):** Indicates carry from the most significant bit.
- **Zero flag (Z):** Indicates if the result of an operation is zero.

The IITB-CPU supports three machine-code instruction formats (**R**, **I**, and **J** types) and a total of 14 instructions. Below is the description of the components used in the IITB-CPU.

Components

Control Unit

- Handles the state transition process and determines which enable and control pins are active at each state.
- Takes inputs: **Opcode**, **Zero Flag**, and **Carry Flag**.
- Outputs all control signals, including MUX control signals, enable pins, ALU signals, and Sign Extender signals (essentially **Opcode** due to its design).

Instruction Register

- Stores the instruction obtained from the Instruction Memory.

Multiplexers (4x1 and 2x1)

- Used to decide which signals pass through specific wires.
- Two types are implemented, taking 16-bit inputs and 3-bit inputs.

Instruction Memory Unit

- Stores the program instructions.
- Instructions are fetched during the execution cycle.

Data Memory Unit

- Stores data either from compiling or during execution.

Register File

- Contains 7 general-purpose registers (R0 to R7) for fast access to data.

Program Counter (Inside Register File)

- Holds the address of the next instruction to be executed.
- Facilitates navigation through instructions.

Temporary Registers (T3 and T4)

- Temporarily store data after the completion of specific operations.

Arithmetic Logic Unit (ALU)

- Takes three inputs: two 16-bit operands and an Opcode.
- Produces three outputs: a 16-bit result, Zero Flag, and Carry Flag.
- Performs six operations: ADD, SUB, MULTIPLY, AND, OR, and IMPLICATION.

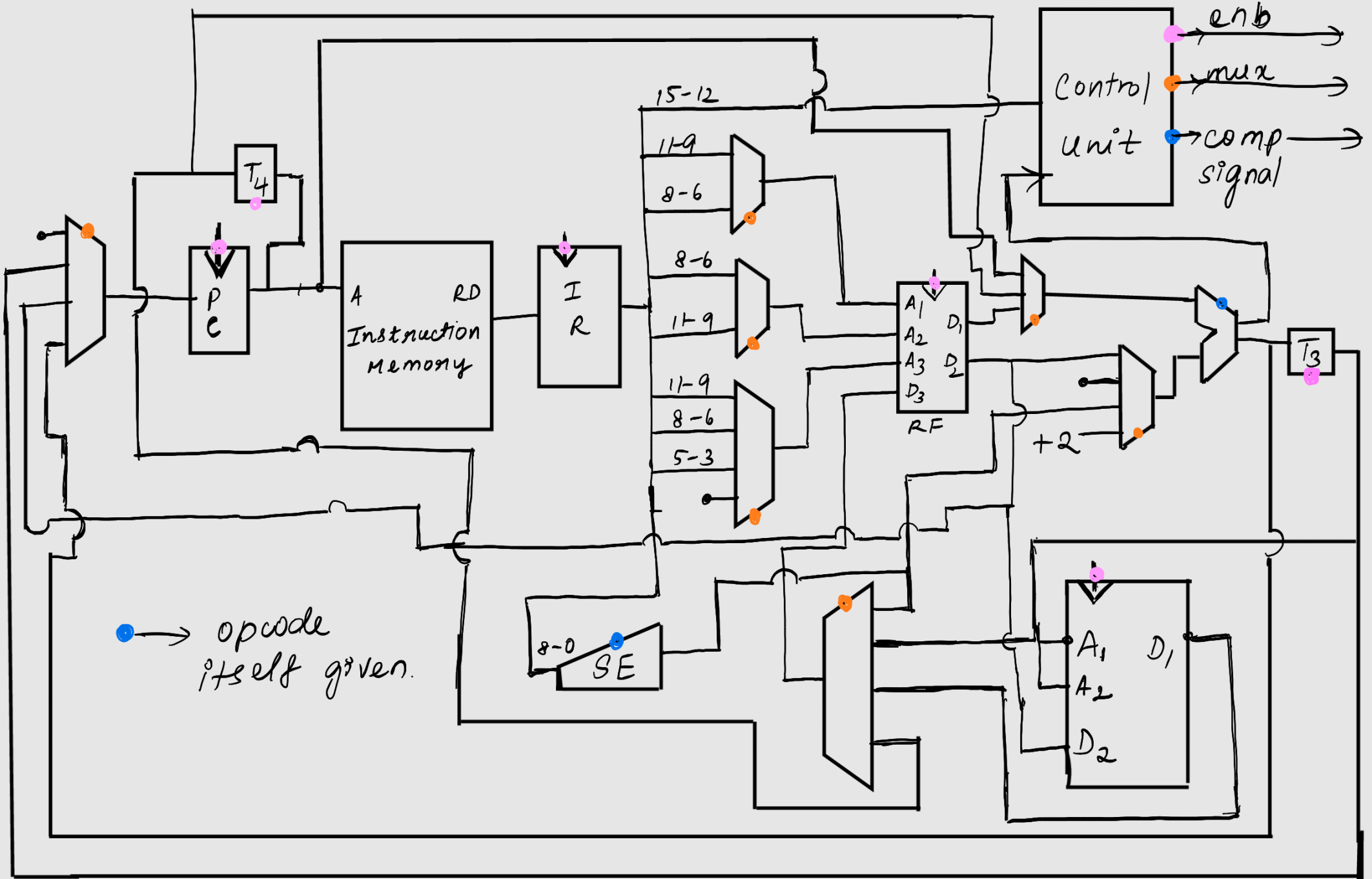
Sign Extender Unit

- Takes two inputs: a 9-bit value and Opcode.
- Outputs a 16-bit result.
- Performs multiple functions:
 - Signed padding
 - Unsigned padding
 - Padding zeros on the right
 - Padding zeros on the left
 - Left shift by one bit
- The added functionality of left shift acts as a multiplier by 2, and zero padding has proven useful for specific instructions.

Data Path

- The data path shows how all components are interconnected.

Datapath: Note: mux orders may vary in VHDL code.



States and State Transition

Each state has been designed in such a way that it takes one clock cycle to complete. The complete state description and transition diagram is shown below.

R-type instructions (ADD, SUB, MUL, AND, OR, IMP)

S₀:

PC \longrightarrow Mem-R	
Mem-R \longrightarrow IR	IR-en
PC \longrightarrow T ₄	T ₄ -en
PC \longrightarrow ALU-A	PC-en
+2 \longrightarrow ALU-B	PC-en
ALU-C \longrightarrow PC	

S₁:

IR(11-9) \longrightarrow RF-A ₁	
IR(8-6) \longrightarrow RF-A ₂	ALU-mux signals
RF-D ₁ \longrightarrow ALU-A	RF-mux
RF-D ₂ \longrightarrow ALU-B	T ₃ -en
ALU-C \longrightarrow T ₃	

S₂:

IR(5-3) \longrightarrow RF-A ₃	
T ₃ \longrightarrow RF-D ₃	RF-mux

Itype Instructions (ADDI, LW, SW, BEQ, JLR)

ADDI

S_0 → S_3

$IR(11-9) \rightarrow RF_A_1$ $IR(8-0) \rightarrow SE$ $SE \rightarrow ALU_B$ $RF_D_1 \rightarrow ALU_A$ $ALU_C \rightarrow T_3$	SE - demux ALU - input muxes
---	--

S_4 :

$IR(8-6) \rightarrow RF_A_3$ $T_3 \rightarrow RF_D_3$	RF - mux
--	------------

LW

S_0 → S_5

$IR(8-6) \rightarrow RF_A_1$ $IR(8-0) \rightarrow SE$ $SE \rightarrow ALU_B$ $RF_D_1 \rightarrow ALU_A$ $ALU_C \rightarrow T_3$	ALU - mux signals RF - mux T_3 - en SE - demux
--	--

S₆:

$T_3 \longrightarrow DMF_A$ $DMF \longrightarrow RF_P_3$ $IR(11-9) \longrightarrow RF_A_3$	DMF_en RF_mux
---	----------------------------

SW

(S₀)

S₇:

$IR(8-6) \longrightarrow RF_A_1$ $IR(8-0) \longrightarrow SE$ $SE \longleftrightarrow ALU_B$ $RF_D_1 \longrightarrow ALU_A$ $ALU_C \longrightarrow T_3$	ALU_mux signals RF_mux T_3-en $SE-demux$
--	--

S₈:

$IR(11-9) \longrightarrow RF_A_2$ $RF_P_2 \longrightarrow DMF_V$ $T_3 \longrightarrow DMF_A$	DMF_enable
--	---------------

BEQ: (S0):

S0:	<p>IR(11-9) \longrightarrow RF-A₁</p> <p>IR(8-6) \longrightarrow RF-A₂</p> <p>RF-D₁ \longrightarrow ALU-A</p> <p>RF-D₂ \longrightarrow ALU-B</p> <p>zero flag \longrightarrow control unit</p>	<p>ALU-mux signals</p> <p>RF-mux</p>
-----	---	--------------------------------------

S10:	<p>T₄ \longrightarrow ALU-A</p> <p>IR(8-0) \longrightarrow SE</p> <p>SE \longrightarrow ALU-B</p> <p>ALU-C \longrightarrow T₃</p>	<p>ALU-mux signals</p> <p>SE-demux</p> <p>T₃-en</p>
------	---	--

S11:	<p>T₃ \longrightarrow PC</p>	<p>PC-mux signal</p> <p>PC-en</p>
------	--	-----------------------------------

JLR:

S_0

S_{12} :

$T_4 \longrightarrow RF-D_3$ $IR(1-9) \longrightarrow RF-A_3$	$RF-en$ $RF-mux$
--	---------------------

S_{13} :

$IR(8-6) \longrightarrow RF-A_2$ $RF-D_2 \longrightarrow PC$	$PC-en$ $PC-mux$
---	---------------------

J-type (LHI, LLJ, JAL, J)

LHI:

S_0

S_{14} :

$IR(8-0) \longrightarrow SE$ $SE \longrightarrow RF_P_3$ $IR(11-9) \longrightarrow RF_A_3$	RF_en RF_mux $SE\ de\ mux$
--	--

left padding

LLJ:

S_0

S_{13} :

$IR(8-0) \longrightarrow SE$ $SE \longrightarrow RF_P_3$ $IR(11-9) \longrightarrow RF_A_3$	RF_en RF_mux $SE\ de\ mux$
--	--

right padding

JAL:

S_0

S_{16} :

$T_4 \longrightarrow RF_P_3$ $IR(11-9) \longrightarrow RF_A_3$	RF_en RF_muxes
---	-------------------------

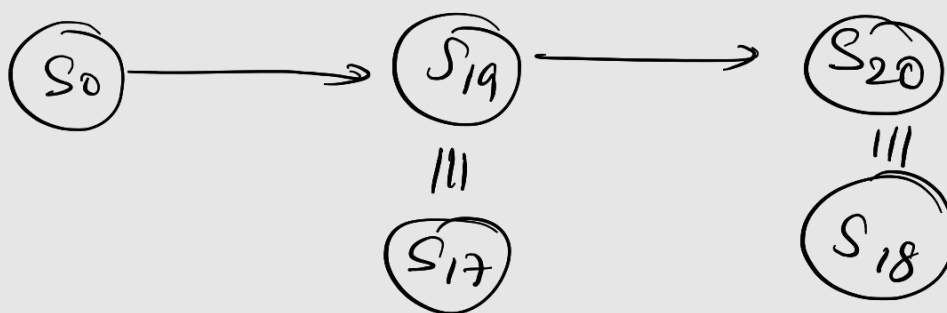
S_{17} :

$T_4 \longrightarrow ALU-A$ $IR(8-0) \longrightarrow SE$ $SE \longrightarrow ALU-B$ $ALU-C \longrightarrow T_3$	ALU_mux $signals$ $SE - demux$ $T_3 - en$
--	---

S_{18} :

$T_3 \longrightarrow PC$	PC_mux $signal$ $PC - en$
--------------------------	------------------------------------

J :

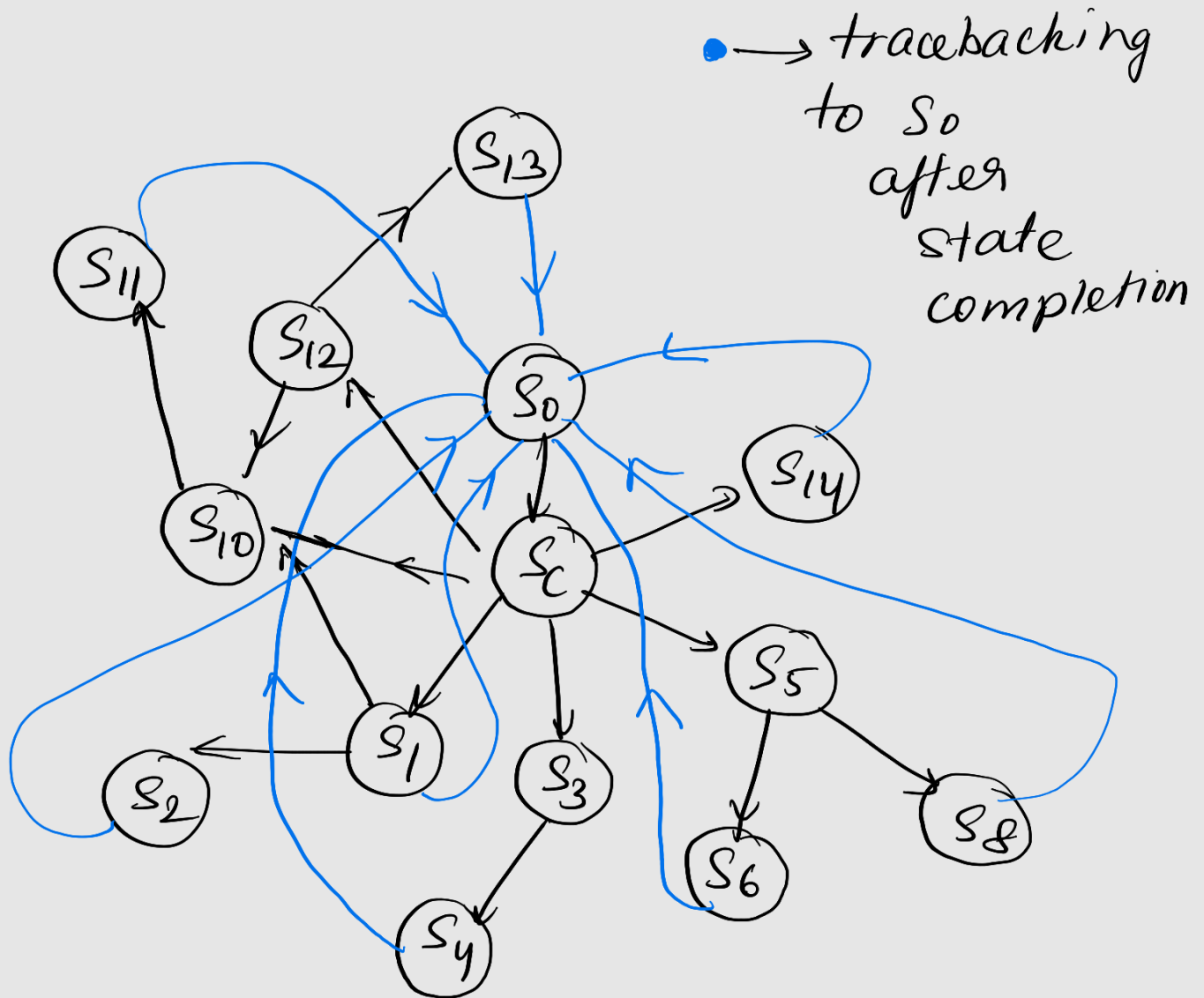


"J-instruction is exactly JAL without state S_{16} "

State Transition diagram:

S_c — state for instruction to be read

↳ give number '100' in simulation.



→ $S_7 \sim S_5$

→ $S_9 \sim S_1$

→ $S_{15} \sim S_{14}$

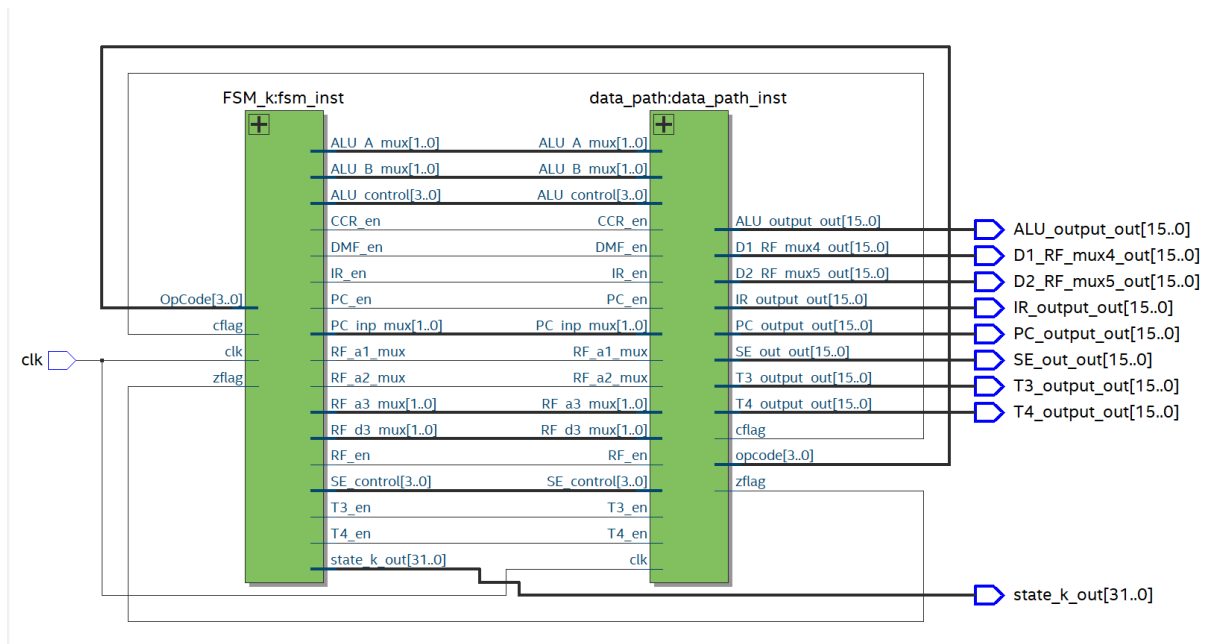
→ $S_{16} \sim S_{12}$

→ $S_{17} \sim S_{10}$

→ $S_{18} \sim S_{11}$

RTL Viewer

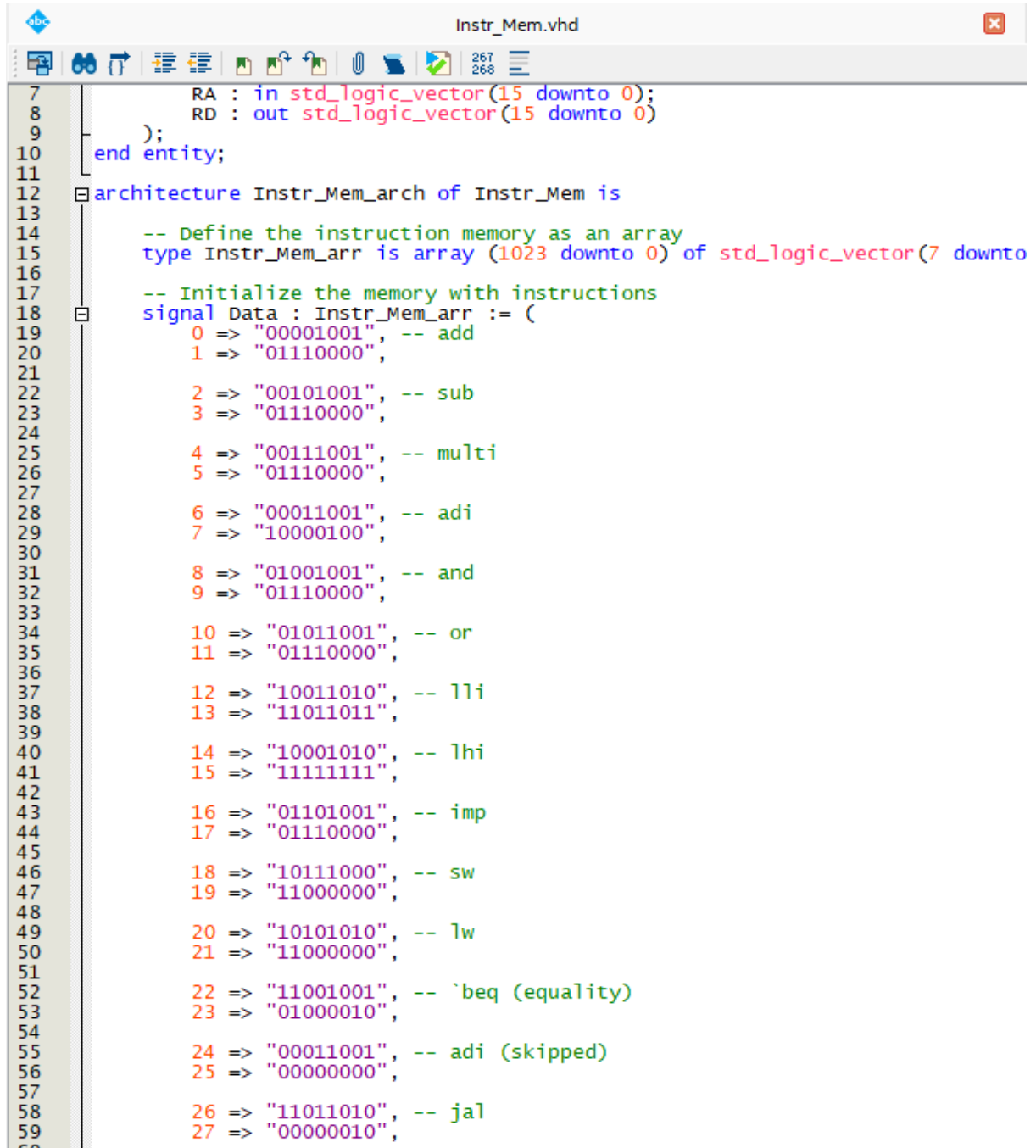
The RTL viewer of the complete project, with necessary signals pulled out for testing, is shown below.



[Insert the RTL Viewer image here]

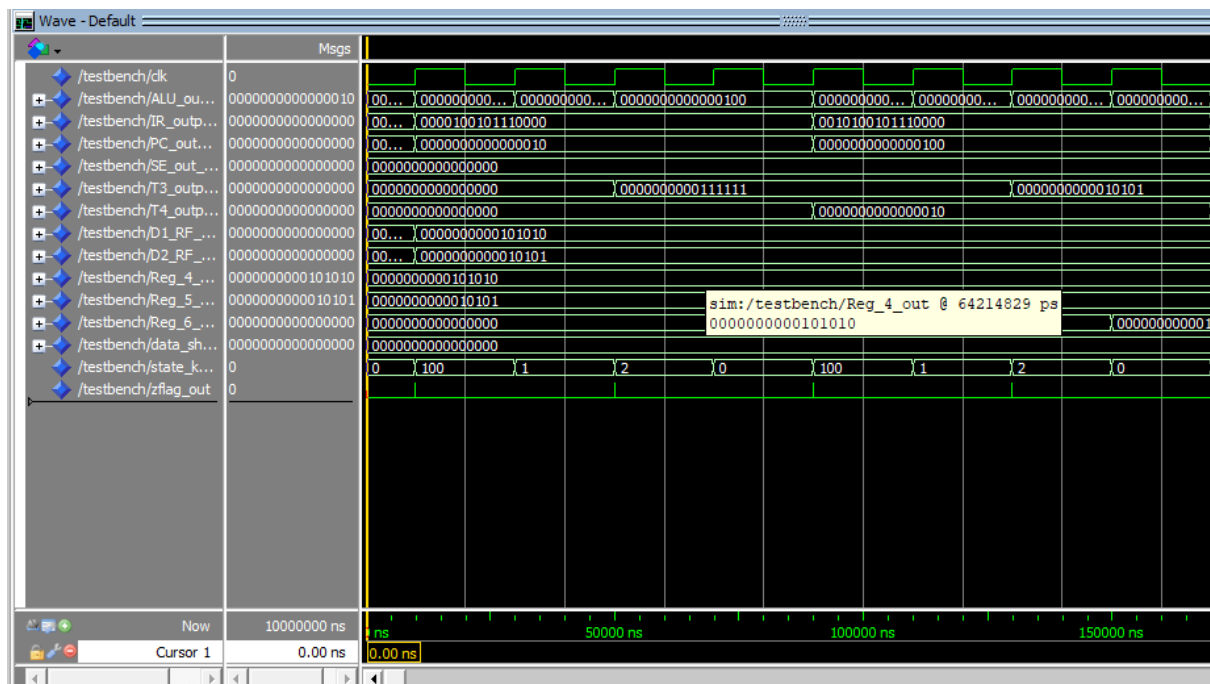
Testing and Simulation

We gave a large instruction set with all 15 instructions jumbled. Below is a screenshot of the instructions manually coded in the instruction memory file, and a testbench was written to test it.



```
7      RA : in std_logic_vector(15 downto 0);
8      RD : out std_logic_vector(15 downto 0)
9    );
10  end entity;
11
12  architecture Instr_Mem_arch of Instr_Mem is
13
14    -- Define the instruction memory as an array
15    type Instr_Mem_arr is array (1023 downto 0) of std_logic_vector(7 downto 0)
16
17    -- Initialize the memory with instructions
18    signal Data : Instr_Mem_arr := (
19      0 => "00001001", -- add
20      1 => "01110000",
21
22      2 => "00101001", -- sub
23      3 => "01110000",
24
25      4 => "00111001", -- multi
26      5 => "01110000",
27
28      6 => "00011001", -- adi
29      7 => "10000100",
30
31      8 => "01001001", -- and
32      9 => "01110000",
33
34      10 => "01011001", -- or
35      11 => "01110000",
36
37      12 => "10011010", -- lli
38      13 => "11011011",
39
40      14 => "10001010", -- lhi
41      15 => "11111111",
42
43      16 => "01101001", -- imp
44      17 => "01110000",
45
46      18 => "10111000", -- sw
47      19 => "11000000",
48
49      20 => "10101010", -- lw
50      21 => "11000000",
51
52      22 => "11001001", -- `beq (equality)
53      23 => "01000010",
54
55      24 => "00011001", -- adi (skipped)
56      25 => "00000000",
57
58      26 => "11011010", -- jal
59      27 => "00000010",
60    )
```

All tests were verified, and hence the CPU's functioning is ensured. The testbench simulation could not be added here due to the large instruction set, but a sample of two or three instructions is shown below.



Acknowledgement

We acknowledge Professor Virendra Singh for teaching us and supporting us to complete this project. We also thank all the Teaching Assistants who helped during the course.

This project has greatly enhanced our learning and deepened our understanding of the concepts. We thank the professor who provided such an opportunity.