IITB-RISC-23

EE309: Microprocessors (Team-3)

By

K Ashvanth

(22B1289)

Chinmay Moorjani

(22B1212)

Nimay Upen Shah

(22B1232)

Sravan K Suresh

(22B3936)

Under the guidance of

Prof. Virendra Singh

Department of Electrical Engineering,

Indian Institute of Technology, Bombay

April-May, 2024

May 11, 2024

Contents

1 Introduction

The IITB-RISC-23 is a 16-bit computer system with 8 registers. It follows the standard 6 stage pipelines (Instruction Fetch, Instruction Decode, Register Read, Execute, Memory Access, and Write Back).
The architecture we have designed is optimized for performance, includes hazard mitigation techniques, for which we have implemented a forwarding mechanism.

Instruction Set Architecture

The IITB-RISC-23 is an 8-register, 16-bit computer system. It has 8 general purpose registers (R0 to R7).
Register R0 is always stores Program Counter. All addresses are byte addresses and instructions.
It always fetches two bytes for instruction and data.
This architecture uses condition code register which has two flags Carry flag (C) and Zero flag (Z). There are three machine-code instruction formats (R, I, and J type) and a total of 14 instructions.

Instructions Encoding:

ADA:	00_01	RA	RB	RC	0	00
ADC:	00_01	RA	RB	RC	0	10
ADZ:	00_01	RA	RB	RC	0	01
AWC:	00_01	RA	RB	RC	0	11
ACA:	00_01	RA	RB	RC	1	00
ACC:	00_01	RA	RB	RC	1	10
ACZ:	00_01	RA	RB	RC	1	01
ACW:	00_01	RA	RB	RC	1	11
ADI:	00_00	RA	RB	6 bit Immediate		
NDU:	00_10	RA	RB	RC	0	00
NDC:	00_10	RA	RB	RC	0	10
NDZ:	00_10	RA	RB	RC	0	01
NCU:	00_10	RA	RB	RC	1	00
NCC:	00_10	RA	RB	RC	1	10
NCZ:	00_10	RA	RB	RC	1	01
LLI:	00_11	RA	9 bit Immediate			
LW:	01_00	RA	RB	6 bit Immediate		
SW:	01_01	RA	RB	6 bit Immediate		
LM:	01_10	RA	0 + 8 bits corresponding to Reg R0 to R7 (left to right)			
SM:	01_11	RA	0 + 8 bits corresponding to Reg R0 to R7 (left to right)			
BEQ:	10_00	RA	RB	6 bit Immediate		
BLT	10_01	RA	RB	6 bit Immediate		
BLE	10_01	RA	RB	6 bit Immediate		

2 Work Distribution

Chinmay - code for IITB_CPU_Pipeline.vhdl, Testbench, assistance in pen-paper design and memory components
Nimay - Datapath, all pen-paper designs
Sravan - Level-1 flowcharts, Control signals, some sub-components
K Ashvanth - Control signals, Datapath

Link to the GitHub repository of this project: Click Here

3 Datapath

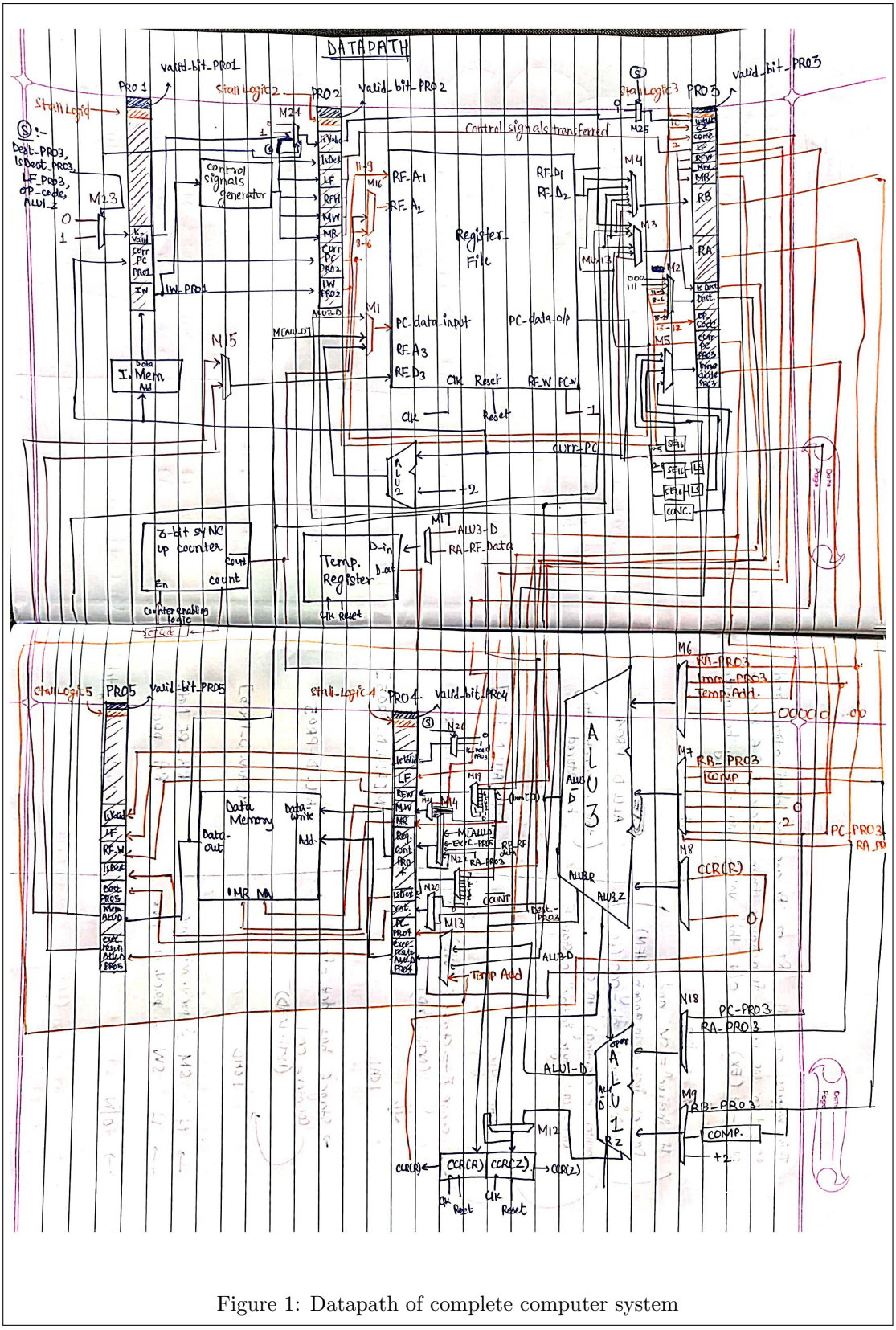

Figure 1: Datapath of complete computer system

4 MUX Tables

Condition	Decision
Destination is R_0 (non-load type instr.)	select ALU3_D
BEQ instr. with a taken branch	select ALU3_D
BLT instr. with a taken branch	select ALU3_D
BLE instr. with a taken branch	IR 12_15
JAL or JLR or JRI instr.	select ALU3_D
Destination is R_0 (load type instr.)	Memory Data at ALU3_D address
others	select ALU2_D

Table 1: **MUX-1 (Choose PC)**

Condition	Decision
ADD or NAND type instr.	destination bits 5-3 (register R_C)
ADI instr.	destination bits 8-6 (register R_B)
LLI or LW or JAL or JLR instr.	destination bits 9-11 (register R_A)
LM instr.	Counter Complement Output
others	000

Table 2: **MUX-2 (Decide Destination)**

Condition	Decision
source(1) of instr. ≡ dest. of instr. in EX stage (Non-Load)	MUX-13
source(1) of instr. ≡ dest. of instr. in MEM stage (Non-Load)	select ALU_D_PRO4
source(1) of instr. ≡ dest. of non-load type instr. in WB stage	select ALU_D_PRO5
source(1) of instr. ≡ dest. of load type instr. in WB stage	M[ALU_D_PRO5]
otherwise	RA_RF_Data

Table 3: **MUX-3 (RA_Forwarding)**

Condition	Decision
source(2) of instr. ≡ dest. of instr in EX stage (non-load)	MUX-13
source(2) of instr. ≡ dest. of instr. in MEM stage (non-load)	select ALU_D_PRO4
source(2) of instr. ≡ dest. of non-load type instr. in WB stage	select ALU_D_PRO5
source(2) of instr. ≡ dest. of load type instr. in WB stage	M[ALU_D_PRO5]
otherwise	RB_RF_Data

Table 4: **MUX-4 (RB_Forwarding)**

Condition	Decision
ADI or LW or SW instr.	sign extend 6-bit Imm.
BEQ or BLT or BLE instr.	compute signed Imm.×2 from 6-bit Imm.
LLI instr.	convert to 16 bit
LM or JAL or JLR instr.	compute signed Imm.×2 from 9-bit Imm.
others	0000000000000000

Table 5: **MUX-5 (Choice of Immediate)**

Condition	Decision
ADD or ADI or NAND instr.	select R_A from PRO_3 pipeline reg.
LW or LLI or SW or BEQ or BLT or BLE or JAL or JRI instr.	Imm. from PRO_3 pipeline reg.
LM or SM instr.	Temporary Address from the Up Counter
others	0000000000000000

Table 6: **MUX-6 (ALU3_A Input Decide)**

Condition	Decision
ADA or ADC or ADZ or AWC or NAND or LW or SW or JLR instr.	select R_B from PRO_3 pipeline reg.
ACA or ACC or ACZ or ACW instr.	select R_B complement from PRO_3 pipeline reg.
ADI instr.	Imm. from PRO_3 pipeline reg.
LLI instr.	0000000000000000 (perform +0)
LM or SM instr.	0000000000000010 (perform +2)
BEQ or BLT or BLE or JAL instr.	select current PC from PRO_3 pipeline reg.
others	select R_A from PRO_3 pipeline reg.

Table 7: **MUX-7 (ALU3_B Input Decide)**

Condition	Decision
AWC or ACW instr.	Carry from CCR
others	0

Table 8: **MUX-8 (ALU3_C Input Decide)**

Condition	Decision
NCU or NCC or NCZ instr.	select R_B complement from PRO_3 pipeline reg.
NDU or NDC or NDZ instr.	select R_B from PRO_3 pipeline reg.
JAL or JLI instr.	0000000000000010 (perform +2)
others	select R_B from PRO_3 pipeline reg.

Table 9: **MUX-9 (ALU1_B Input Decide)**

Condition	Decision
ADD or ADI or LW instr.	Z flag from ALU3
NAND instr.	Z flag from ALU1
others	latch

Table 10: **MUX-12 (Choose Zero)**

Condition	Decision
LM or SM instr.	Temporary Address from Up Counter
NAND or JAL or JLR instr.	select ALU1_D
others	select ALU3_D

Table 11: **MUX-13 (Choose Result)**

Condition	Decision
Instr ≡ `SM` and Dest. of instr in Stage-5 (`MEM`) is R7, counter $o/p = 000$ (counter starting) in a NON-LOAD type instr	Select ALU_D_PRO4
Instr ≡ `SM` and Dest. of instr in Stage-5 (`MEM`) is R7, counter $o/p = 000$ (counter starting) in a LOAD type instr	select M[ALU_D_PRO4]
Instr ≡ `SM` and Dest. of instr in Stage-6 (`WB`) is R7, counter $o/p = 000$ (counter starting) in a NON-LOAD type instr	select ALU_D_PRO5
Instr ≡ `SM` and Dest. of instr in Stage-6 (`WB`) is R7, counter $o/p = 000$ (counter starting) in a LOAD type instr	select M[ALU_D_PRO5]
`SM with the previous conditions FALSE`	RB_RF_Data
`SW with the previous conditions FALSE`	RA_RF_Data
Otherwise	Latch

Table 12: **MUX-14 (Choose Register Content)**

Condition	Decision
not Load_Forwarded and Is_Destination	result from ALU_D in $PRO5$
Load_Forwarded and Is_Destination	result from Mem_ALU_D in $PRO5$
others	result from ALU_D in $PRO5$

Table 13: **MUX-15 (Choose Write-Back Data)**

Condition	Decision
SM instr.	Counter Complement Output
others	destination bits 8-6 (corresponding to R_B)

Table 14: **MUX-16 (Choose RF_A2)**

Condition	Decision
LM or SM instr.	select ALU3_D
others	RA_RF_Data

Table 15: **MUX-17 (Choose Temporary Address)**

Condition	Decision
JAL or JLR instr.	select Current PC from $PRO3$ pipeline reg.
others	select R_A from PRO_3 pipeline reg.

Table 16: **MUX-18 (Choose ALU1_A_input)**

Condition	Decision
Counter Output = 000	select Instr. Word(7)
Counter Output = 001	select Instr. Word(6)
Counter Output = 010	select Instr. Word(5)
Counter Output = 011	select Instr. Word(4)
Counter Output = 100	select Instr. Word(3)
Counter Output = 101	select Instr. Word(2)
Counter Output = 110	select Instr. Word(1)
Counter Output = 111	select Instr. Word(0)

Table 17: **MUX-19 (Choose nth bit of Imm. for LM/SM)**

5 Simulation Results

The following simulations were obtained when the Instruction Memory was hard-coded with different instructions to test the functioning of our CPU by loading the corresponding Testbench which was built separately.
The following are the simulation results:

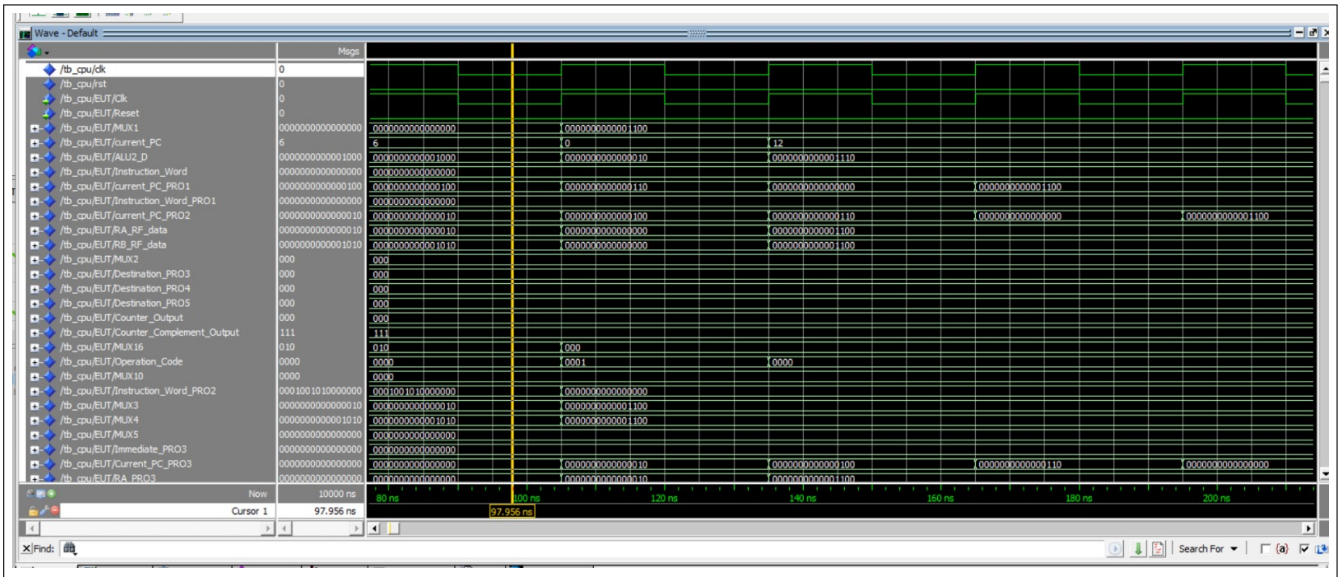

Figure 2: Testing ADA Instr.

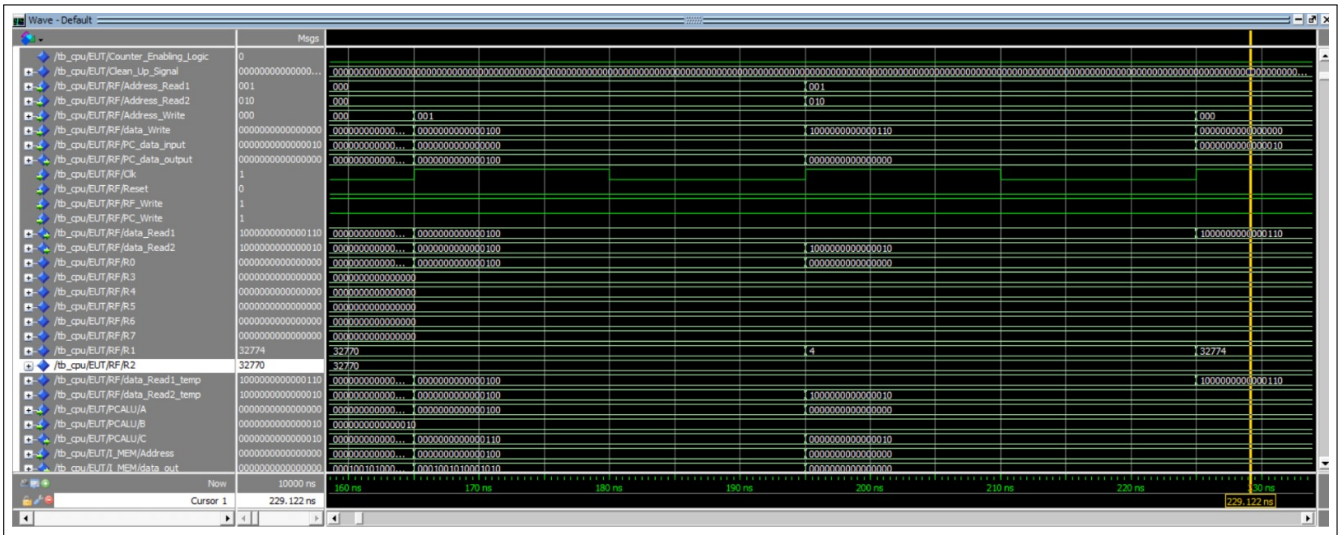

Figure 3: Testing ADC Instr.

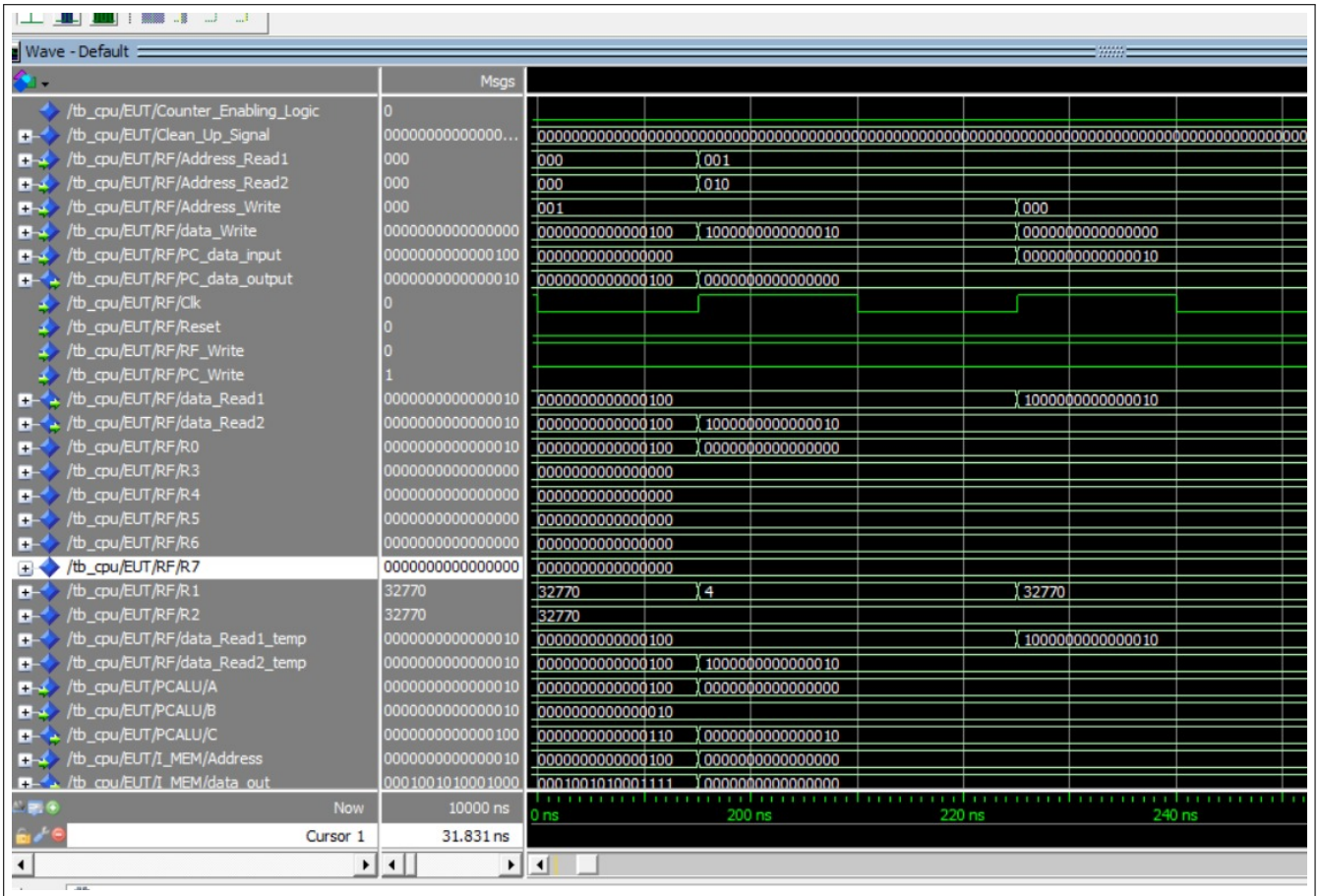

Figure 4: Testing ACC Instr.

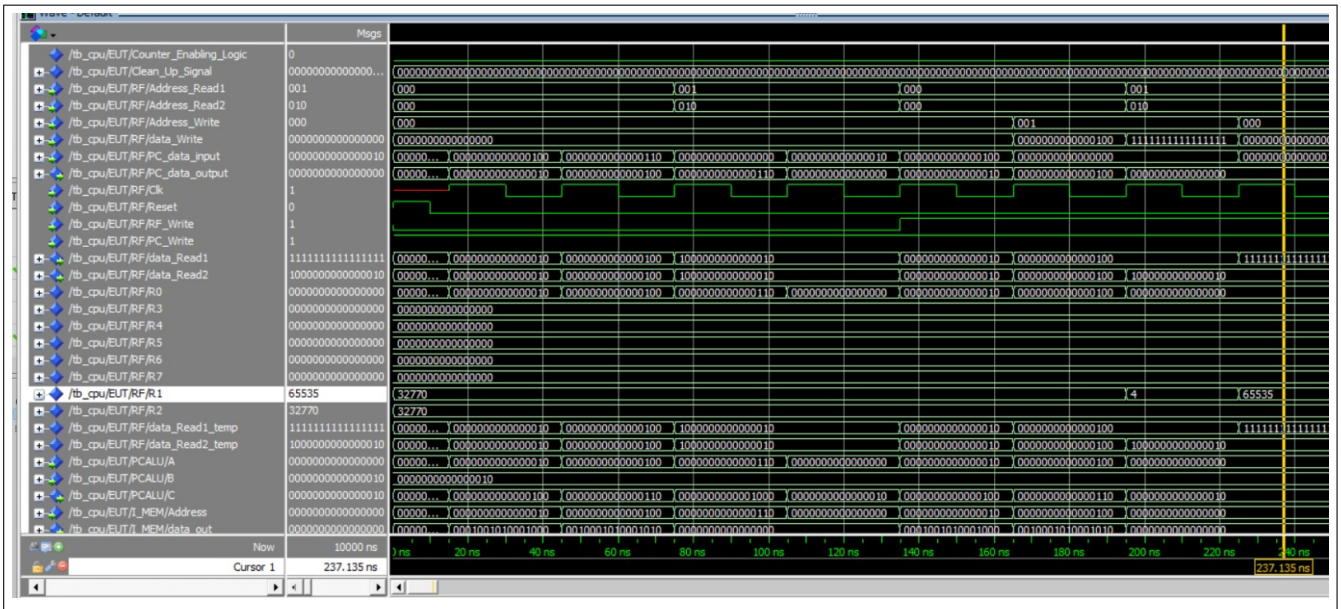

Figure 5: Testing NDC Instr.

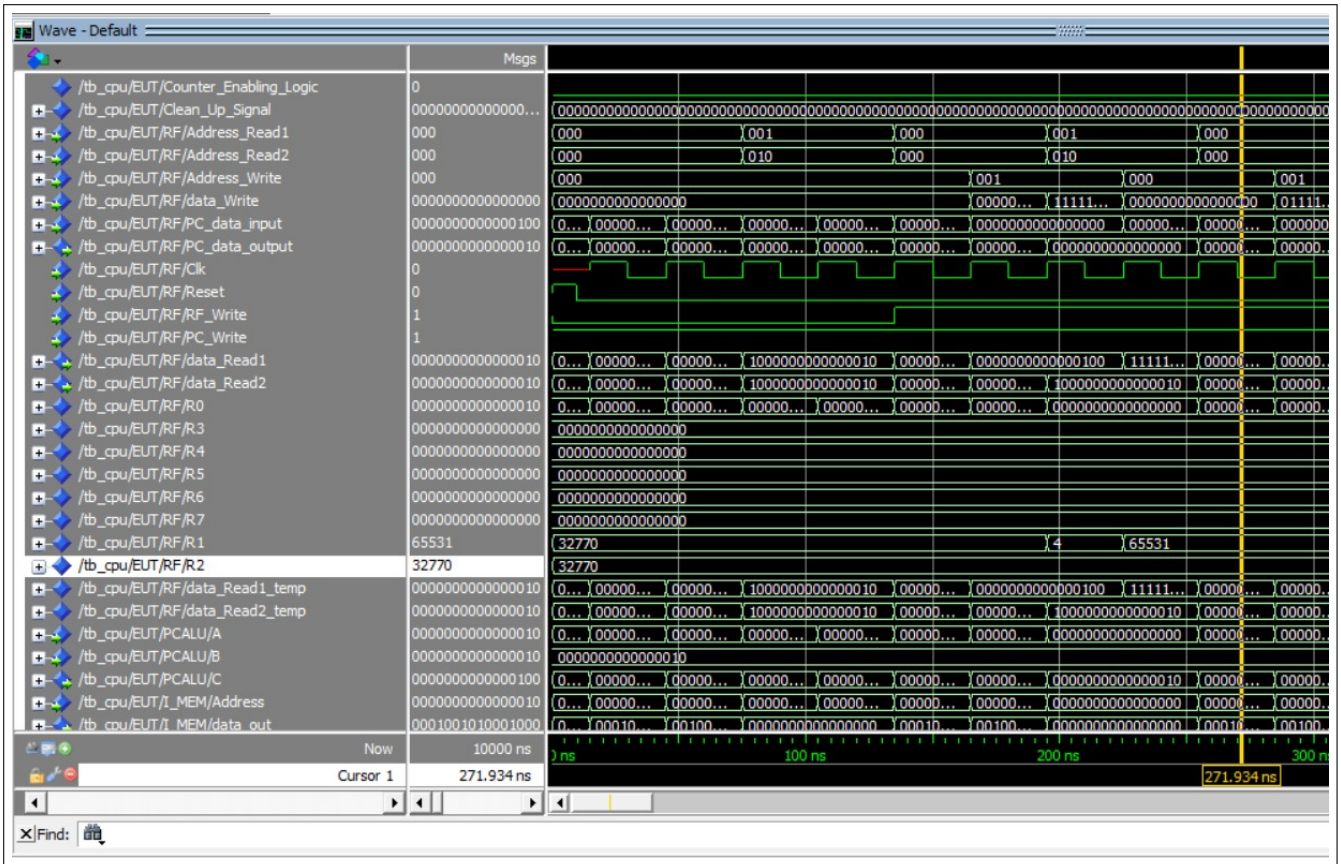

Figure 6: Testing NCC Instr.

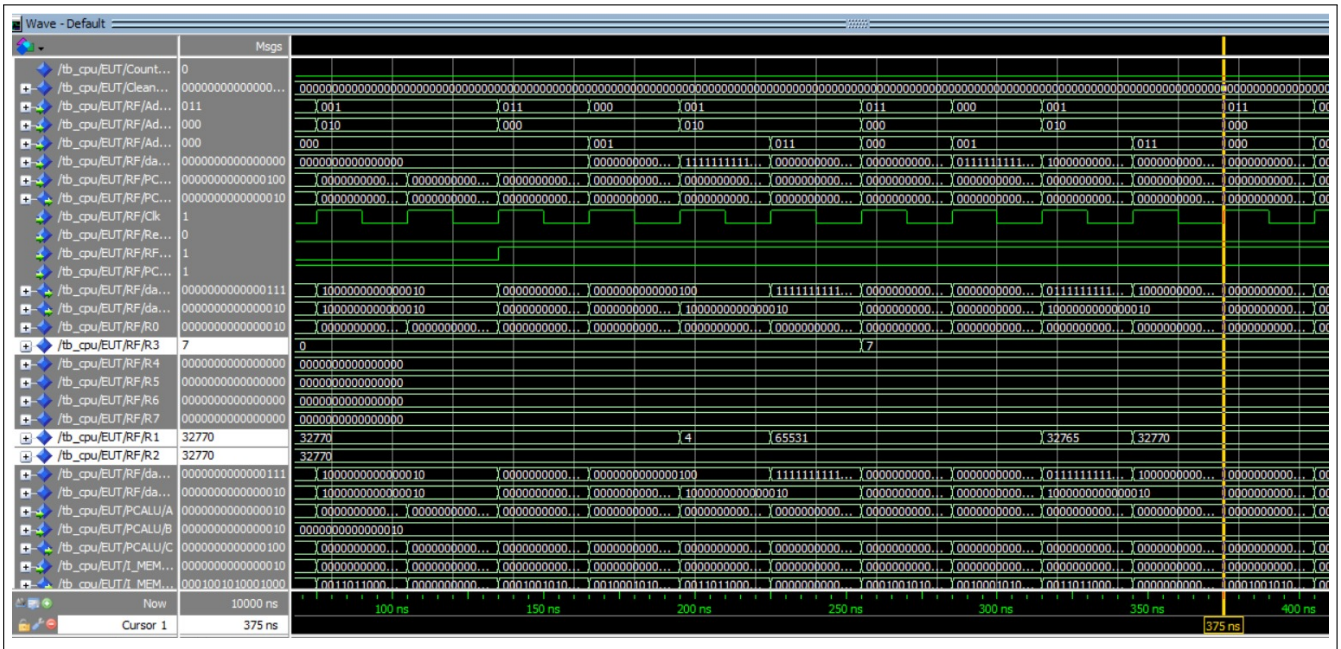

Figure 7: Testing LLI Instr.

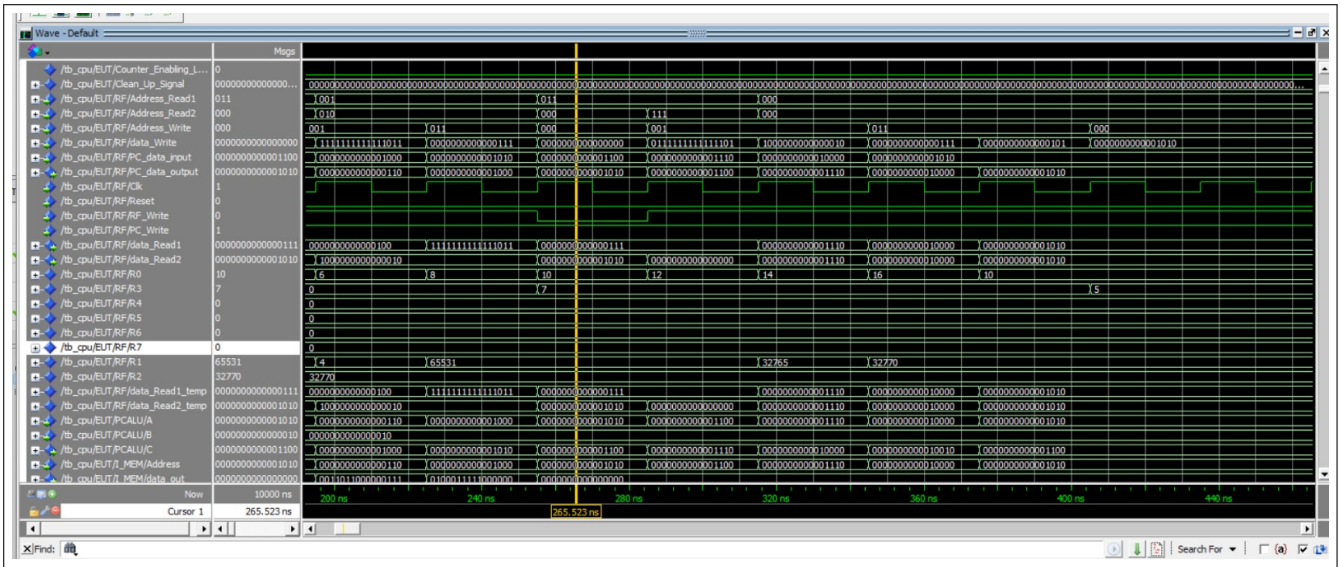

Figure 8: Testing LW Instr.

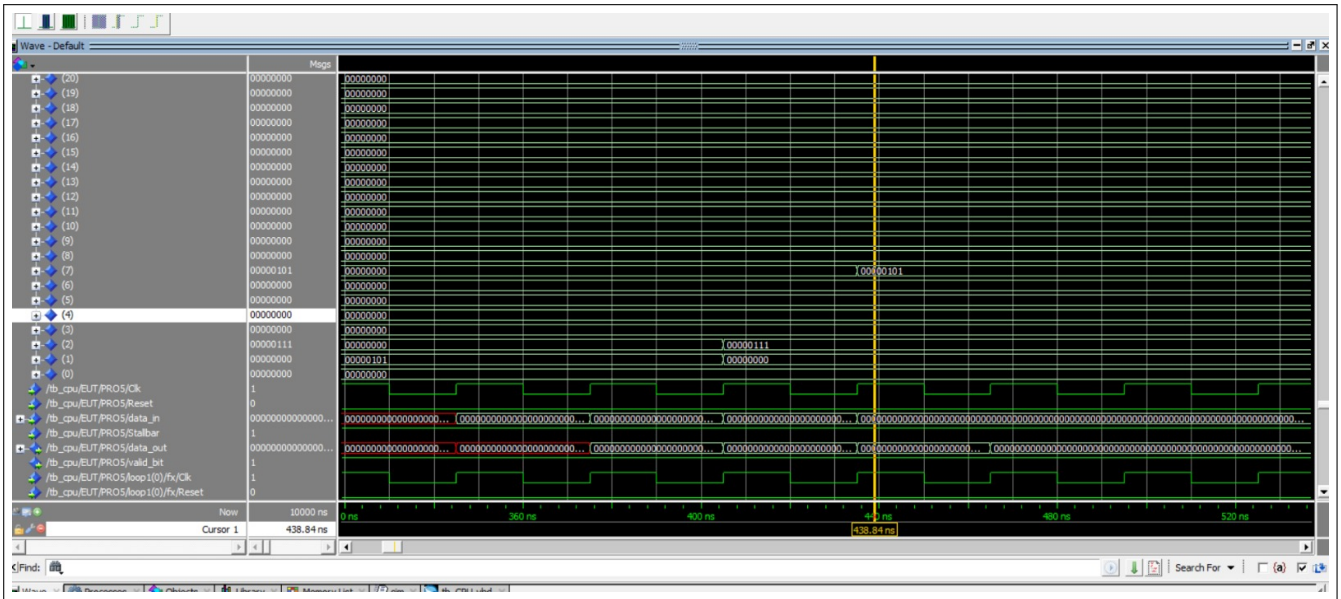

Figure 9: Testing SW Instr.

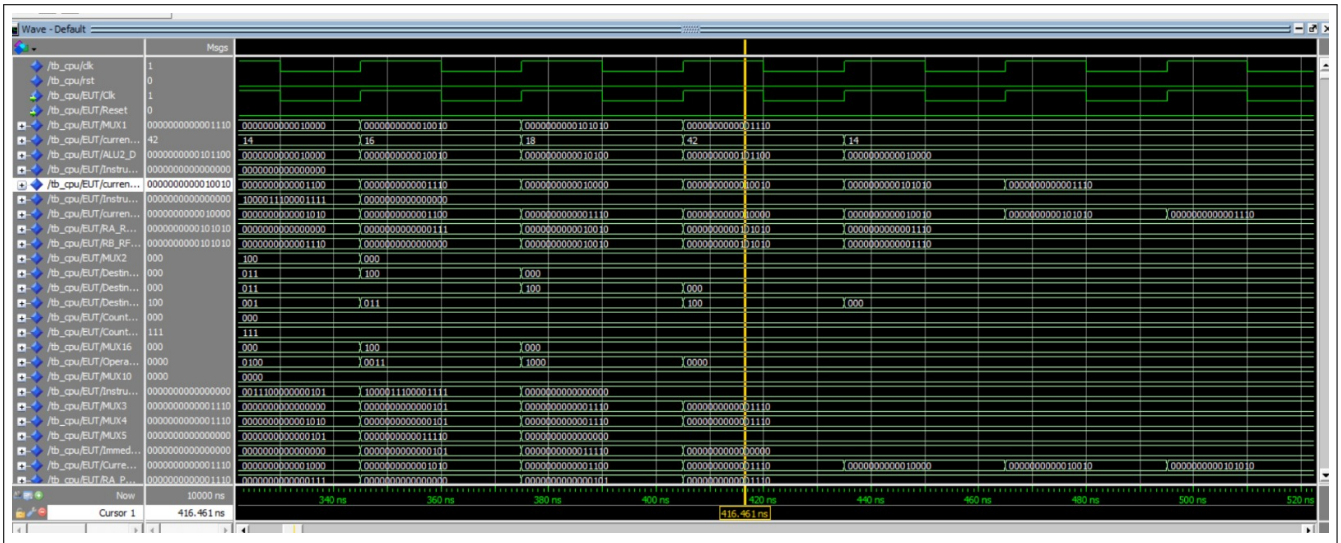

Figure 10: Testing BEQ Instr.

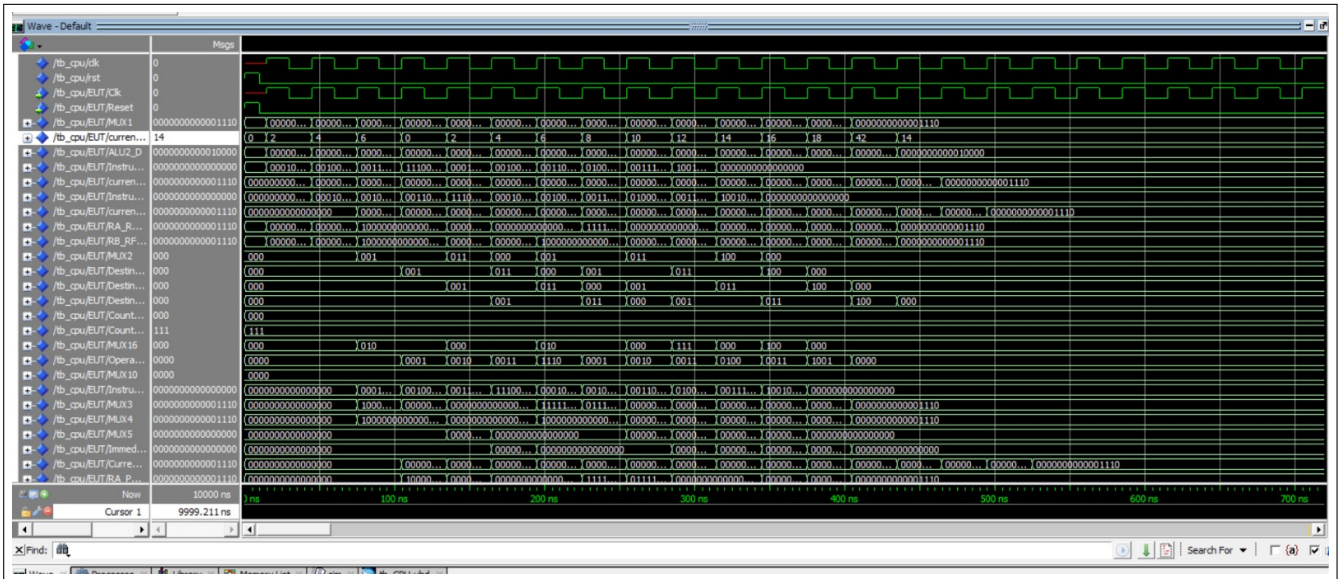

Figure 11: Testing BLT Instr.

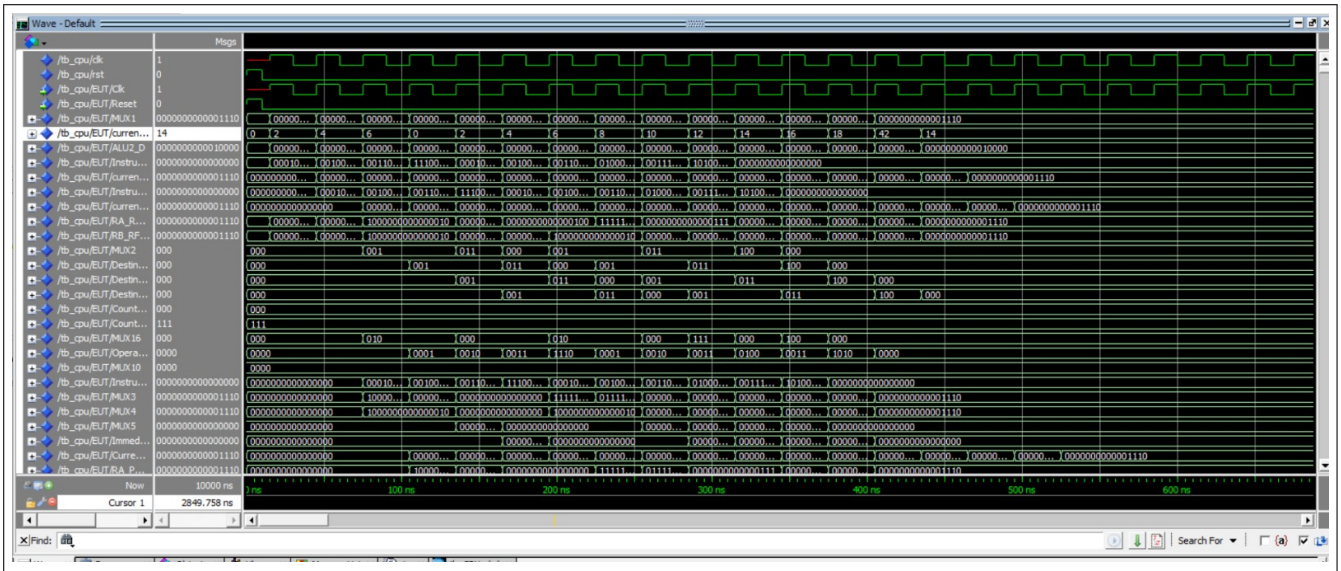

Figure 12: Testing BLE Instr.

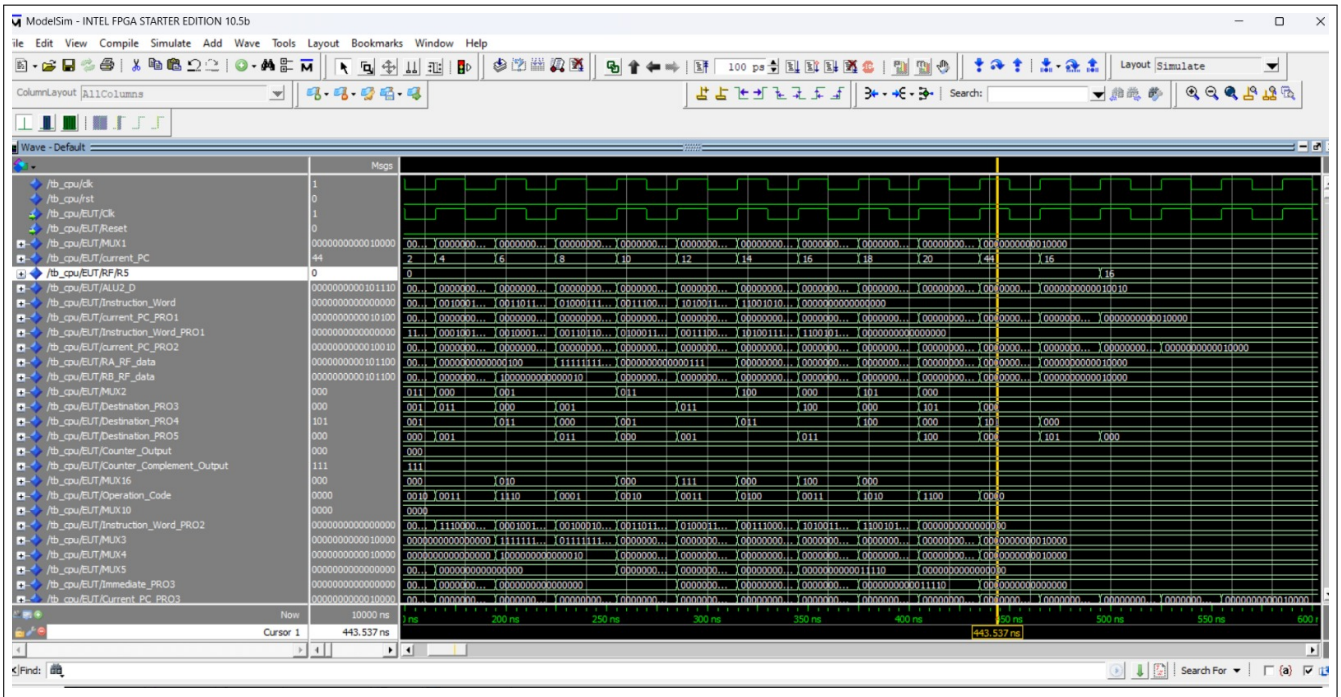

Figure 13: Testing JAL Instr.

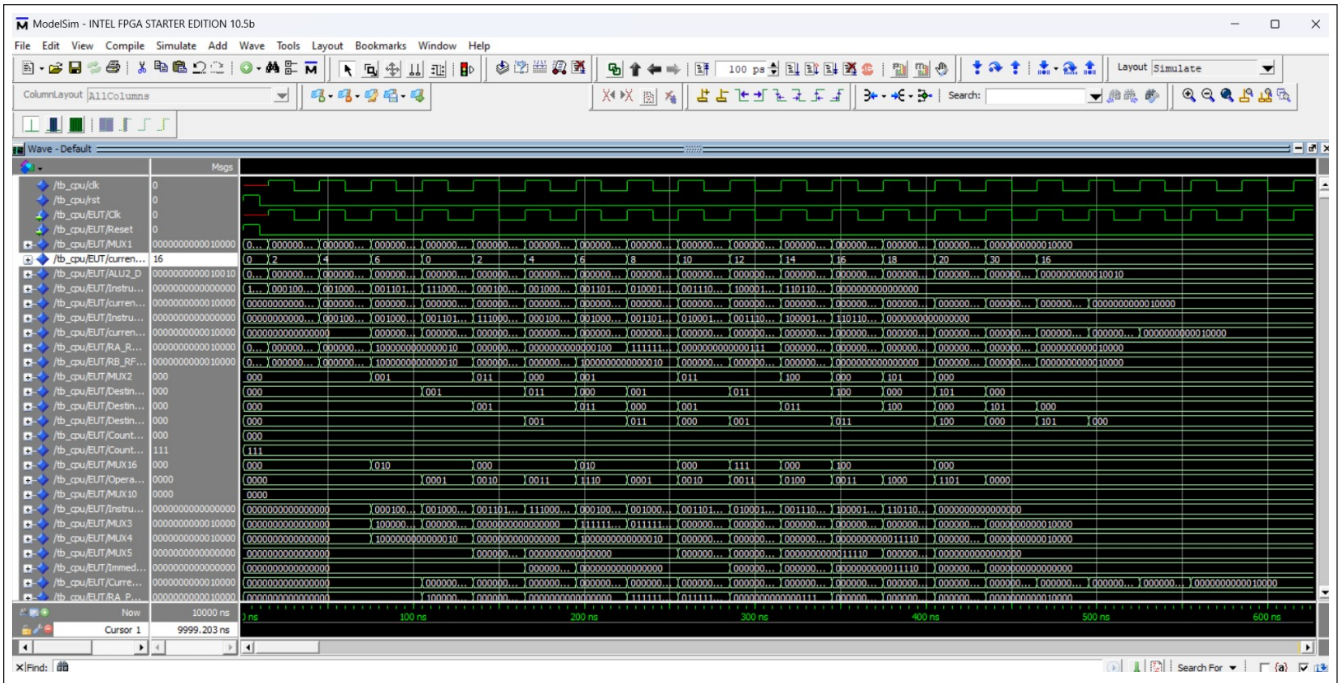

Figure 14: Testing JLR Instr.

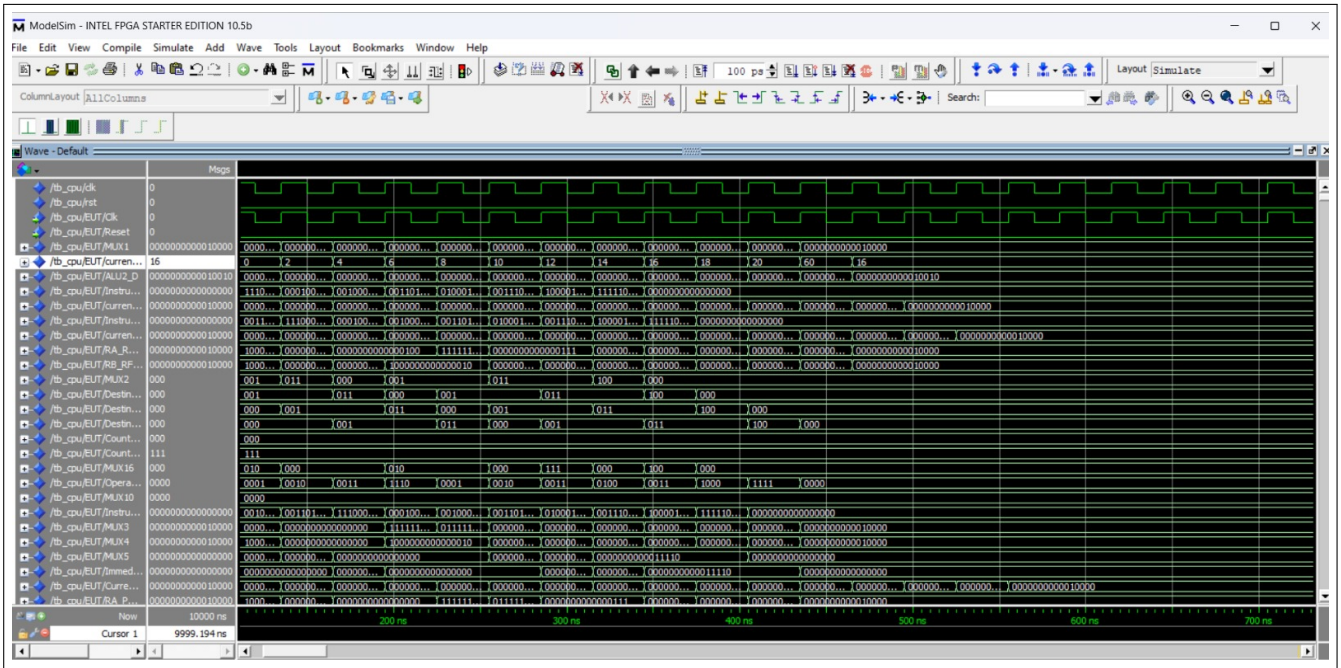

Figure 15: Testing JRI Instr.

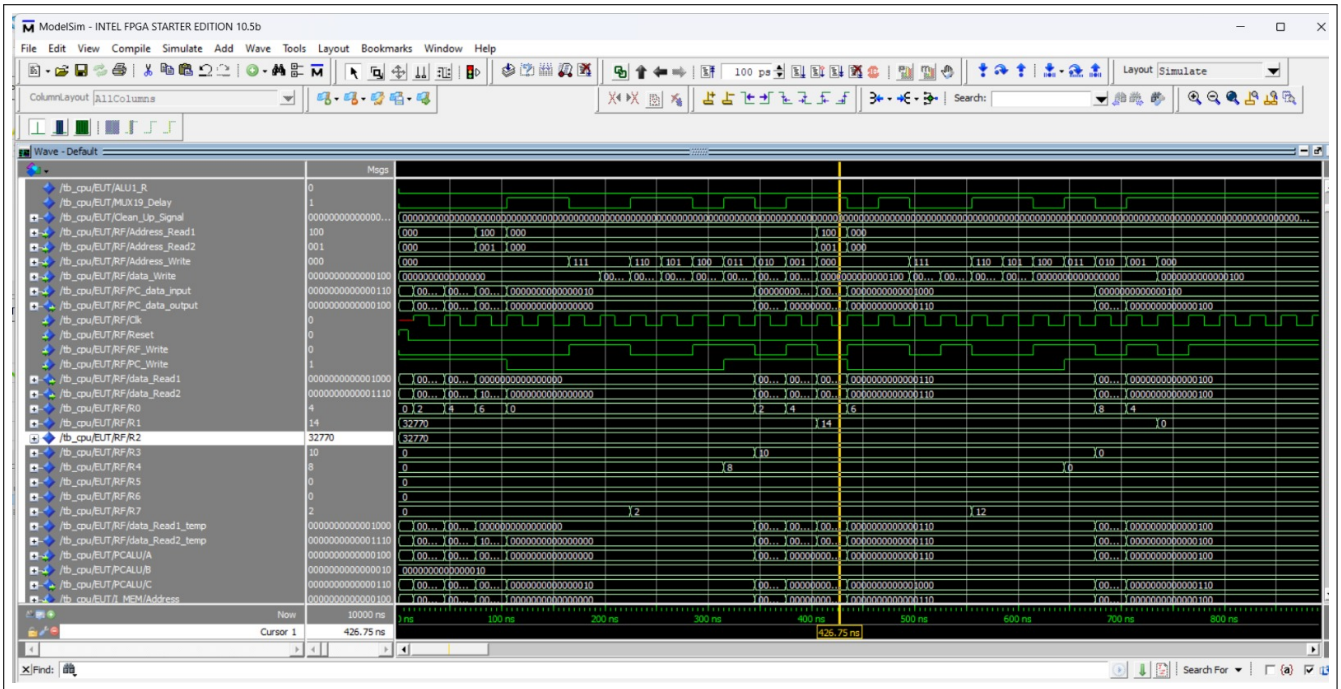

Figure 16: Testing LM Instr.

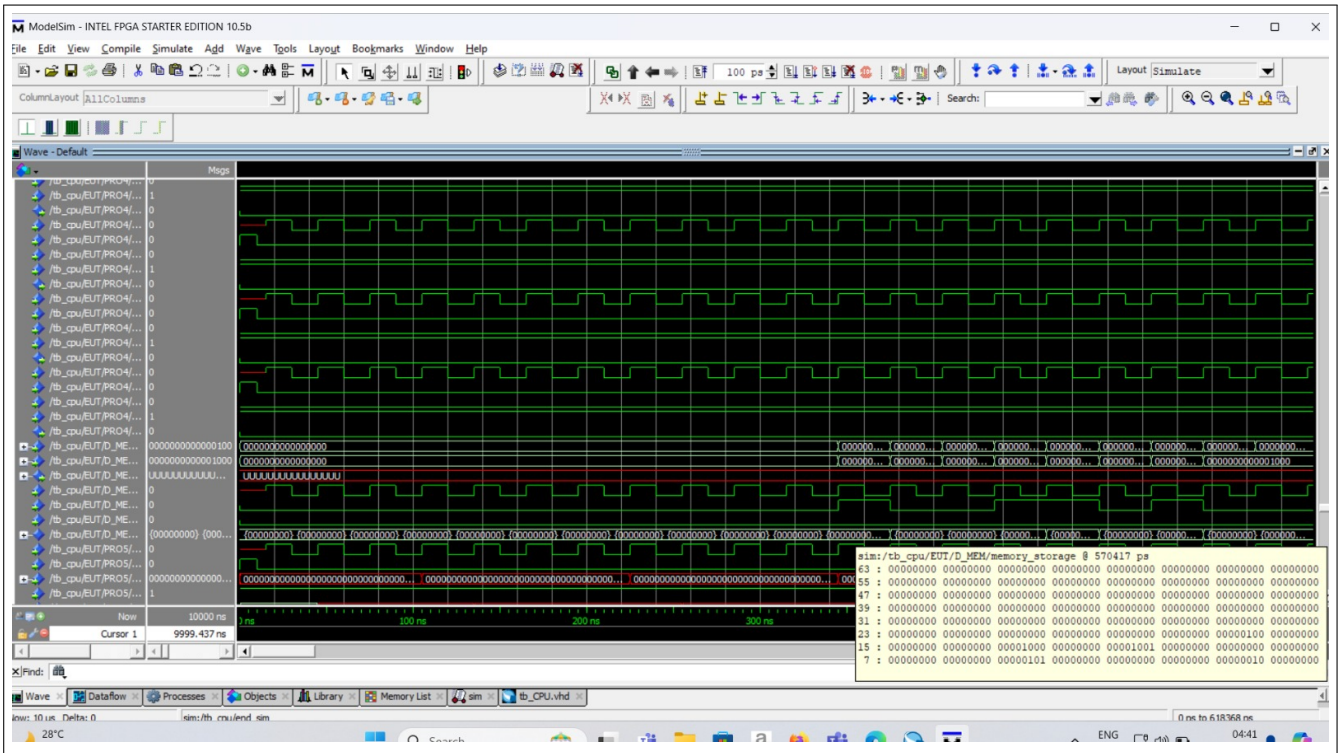

Figure 17: Testing SM Instr.

6 Design and Summary

We have built a 16-bit, six-stage (Instruction fetch, instruction decode, register read, execute, memory access, and write back) pipelined processor that has 8 registers. We have implemented forwarding mechanism using the processes `R_A Forwarding` and `R_B Forwarding` so as to mitigate hazard changes. The CPU mitigates most of the common hazards, including wrong branch predictions and immediate dependency of LOAD instructions etc. so as to bring the CPI as close to 1 as possible. We also implemented a 3-bit synchronous counter, for instructions like LM,SM so that we can search all the registers from R0 to R7, and with the same we implemented a **Stalling_Logic** for all the pipelined registers, which stalls the system for 8 cycles while executing LM or SM and then continues ahead with the next instruction. The counter is in synchronization with the CPU Clock. We also implemented a **Valid_Bit** based resetting logic, to kill the instructions by disabling their control signals- RF-Write and Memory-Write. With a total of 5 pipeline registers that store a variety of information- the Instruction Word, the Current PC, the Control Signals and other data which needs to be transferred from one stage to another. We have the following components in the CPU:

- Instruction Memory
- Data Memory
- 3 ALUs, one of them being a 3-input, rest being 2-input
- 5 Pipeline Registers
- A Control Signal Generator Block
- 26 MUXes with some of them being 2x1, 4x1 and 8x1
- Register File
- 3-Bit Synchronous Up Counter
- Temporary Register
- 2 D- Flip Flops for storing carry and zero
- Overall CPU Clock and Reset Signals