EE-224: Digital Design

IITB-CPU

Instructor: Prof. Virendra Singh

Team ID - 7

Aman Rishal CH – 22b3914

Aman Milind Moon – 22b1216

Chinmay Tripurwar – 22b3902

Swarup Dasharath Patil – 22b3953

Introduction

The IITB-CPU, crafted as a 16-bit computer system, is a foundational architecture designed for educational purposes, particularly for teaching Little Computer Architecture. Featuring eight general-purpose registers (R0 to R7), with Register R7 exclusively assigned to storing the Program Counter (PC), this compact CPU employs a memory addressing scheme where each address corresponds to a single byte. The architecture supports three distinct instruction formats: R, I, and J types, each tailored for specific tasks in executing machine code. In total, the IITB-CPU accommodates 14 instructions, making it a versatile and practical platform for educational exploration.

Major Components Used

1. Memory
   The memory is organized as an array with 512 bytes, and each memory location holds 8 bits of data.
2. Register File
   The register file consists of eight 16-bit registers of which register R7 is used as a program counter.
3. 5 Temporary registers
   The system employs four 16-bit temporary registers and one 3-bit register for quick and versatile data storage during computational tasks.
4. Instruction Register
   The Instruction Register, a 16-bit storage unit, holds instructions in the r, i, or j format
5. ALU
   The ALU (Arithmetic Logic Unit) performs operations like addition, subtraction, multiplication, logical OR, logical AND, and implication on two 16-bit numbers within the computer system.
6. Sign Extender
   The sign extender extends either a 6-bit or 9-bit number to a 16-bit number based on the control signal, preserving the sign bit for accurate representation in the computer system.
7. Shifter
   The shifter performs either left shifts or right shifts on binary data, governed by the control signal, within the computer system.

States

Description:

State 1: Read from memory and increment PC

State 2: Read Instruction (R Type)

State 3: Execute Specific ALU operation

State 4: Update Result

State 5: Read Instruction (LHI/LLI)

State 6: Read Instruction (ADI)

State 7: Read Instruction (LW/SW)

State 8: Read from memory

State 9: Write to memory

State 10: Branch to PC + Imm*2

State 11: PC to Reg. A

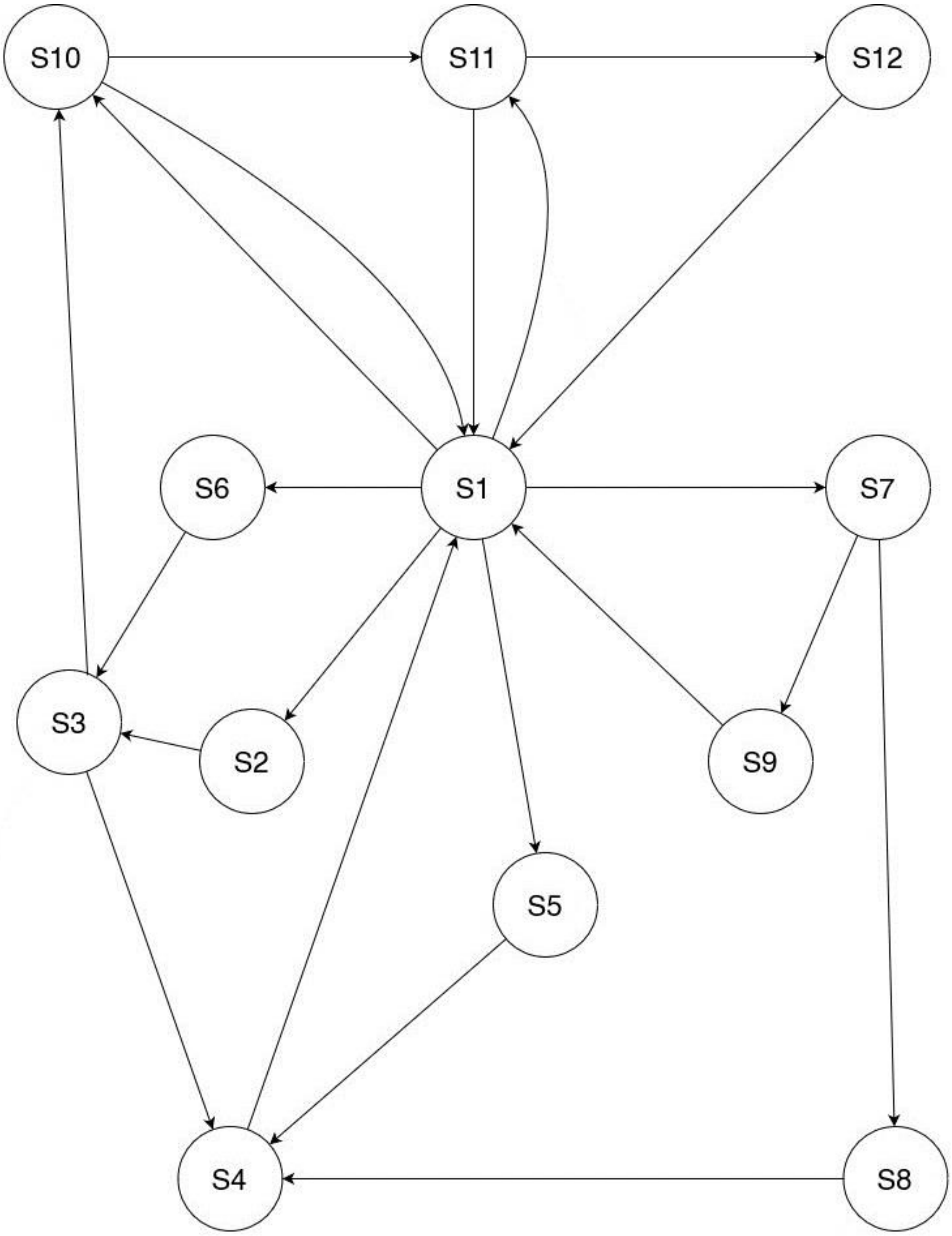
State 12: Branch to Reg. B

State Flowcharts

States	Data Transfer	Control Signal
S1	111 → RF_A1 RF_D1 → MEM_ADD MEM_OUT → IR RF_D1 → ALU_A +2 → ALU_B ALU_C → RF_D3 111 → RF_A3 RF_D1 → T5	M_RD IR_W ALU_CTRL RF_WR T5_WR
S2	IR_11_9 → RF_A1 IR_8_6 → RF_A2 RF_D1 → T1 RF_D2 → T2 IR_5_3 → T4	T1_WR T2_WR T4_WR
S3	T1 → ALU_A	ALU_CTRL

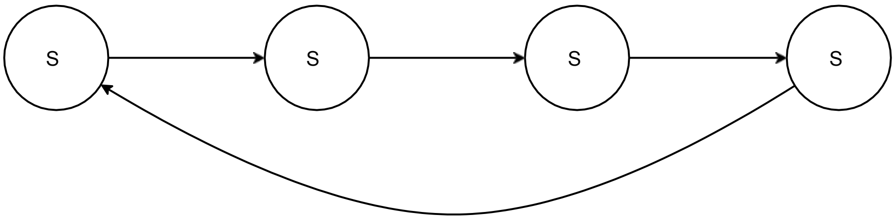
	T2 → ALU_B ALU_C → T3 ALU_Z → Zin	T3_WR
S4	T3 → RF_D3 T4 → RF_A3	RF_WR
S5	IR_8_0 → SHIFTER_IN SHIFTER_OUT → T3 IR_11_9 → T4	SHIFT_SIGNAL T3_WR T4_WR
S6	IR_11_9 → RF_A1 RF_D1 → T1 IR_8_6 → T4 IR_5_0 → SE_IN_6 SE_OUT → T2	T1_WR T2_WR SE_SIGNAL T4_WR
S7	IR_11_9 → T4 IR_5_0 → SE_IN_6 SE_OUT → ALU_A IR_8_6 → RF_A2 RF_D2 → ALU_B ALU_C → T3 IR_11_9 → RF_A1 RF_D1 → T1	T4_WR SE_SIGNAL ALU_CTRL T3_WR T1_WR
S8	T3 → MEM_ADD MEM_OUT → T3	M_RD
S9	T1 → MEM_IN T3 → MEM_ADD	M_WR
S10	T5 → ALU_A IR_5_0 → SE_IN_6 IR_8_0 → SE_IN_9 SE_OUT → SHIFTER_IN SHIFTER_OUT → ALU_B ALU_C → RF_D3 111 → RF_A3	SE_SIGNAL SHIFTER_SIGNAL ALU_CTRL RF_WR
S11	TR_11_9 → RF_A3 T5 → RF_D3	RF_WR
S12	IR_8_6 → RF_A2 RF_D2 → RF_D3 111 → RF_A3	RF_WR

FSM

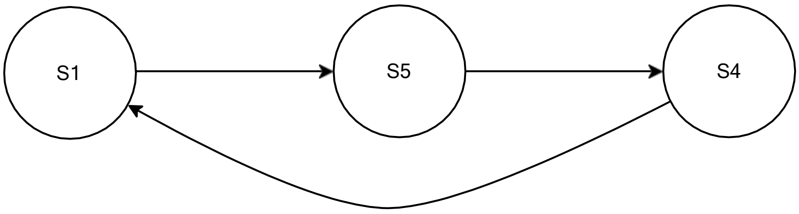


STATE FLOW DIAGRAM

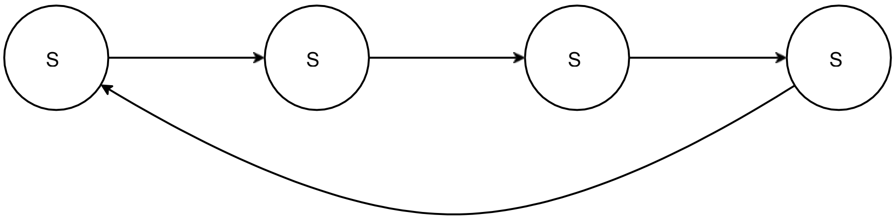
ALU



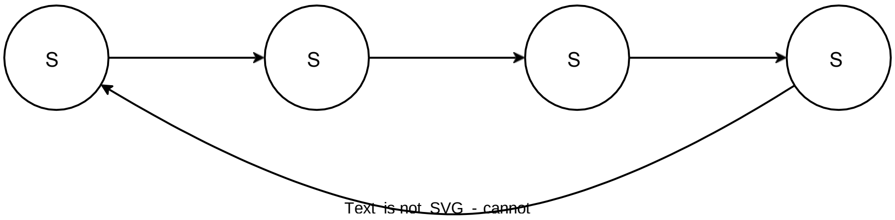
J- Type



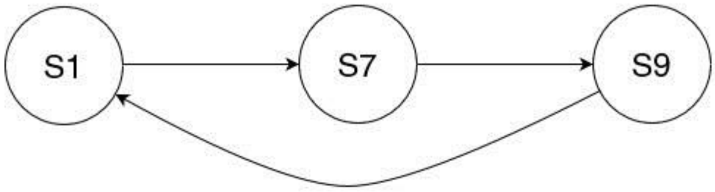
ADI



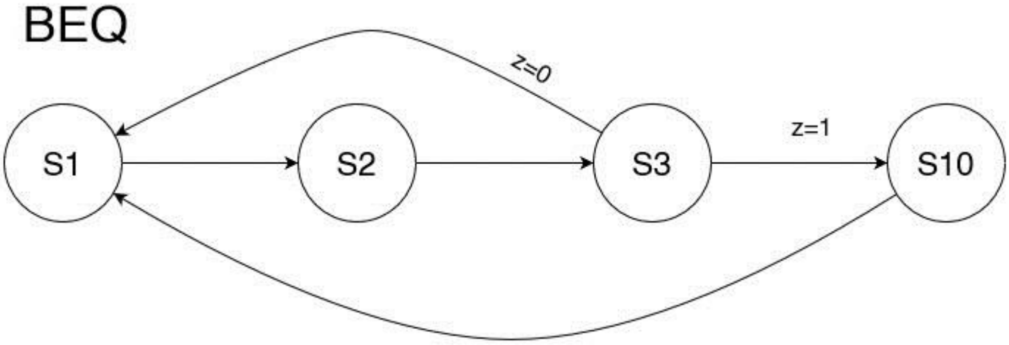
LW



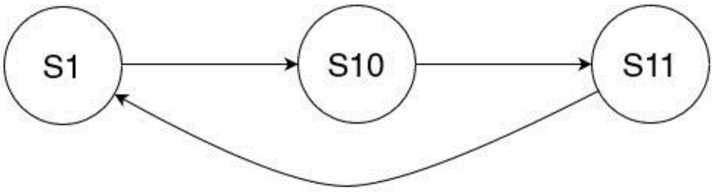
SW



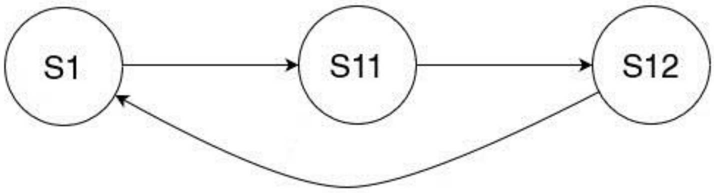
BEQ



JAL

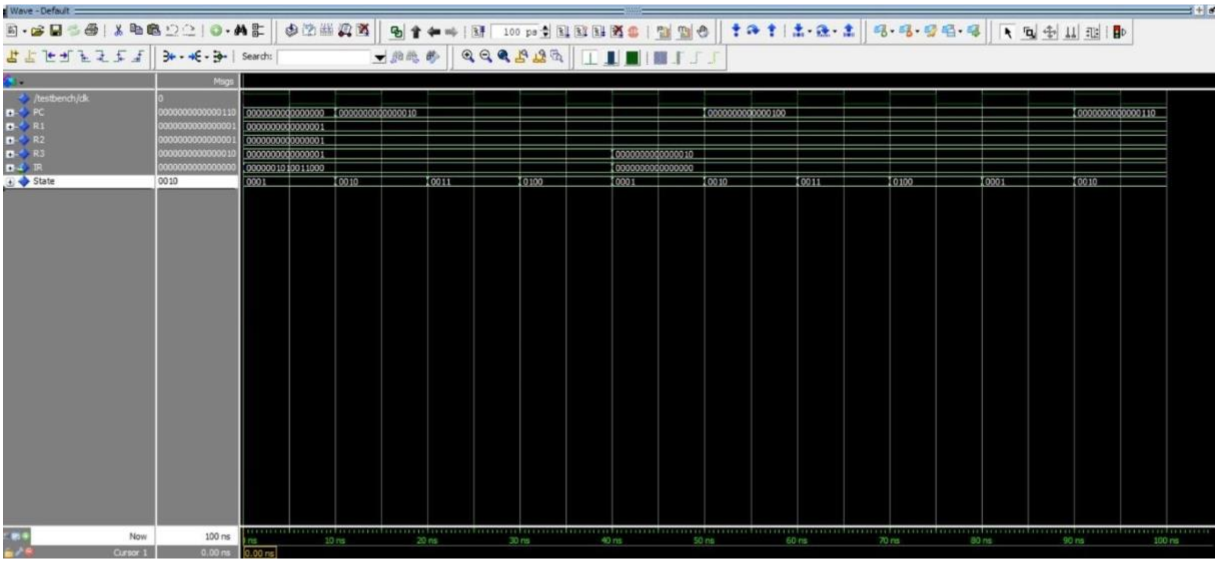


JLR

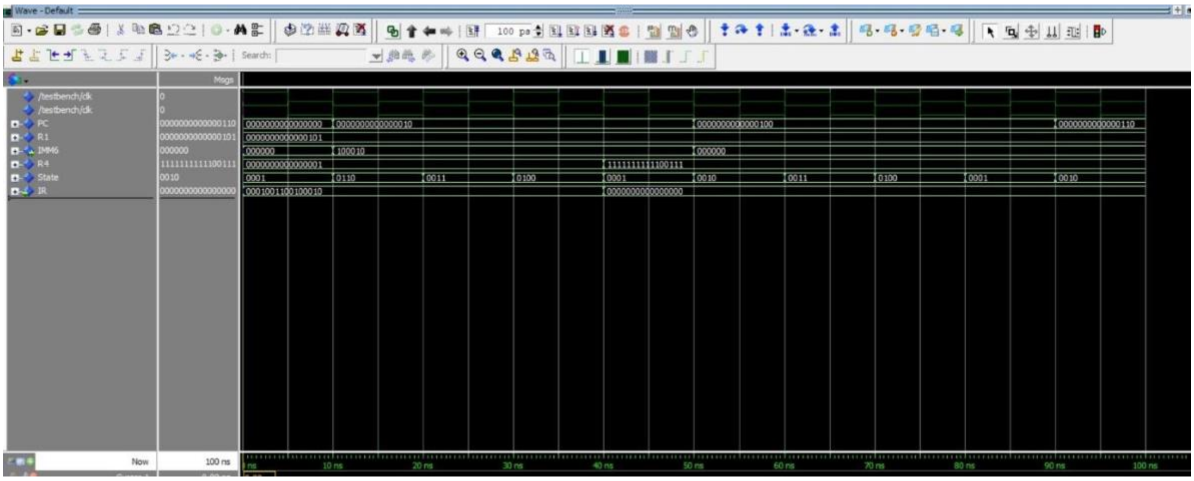


RTL SIMULATIONS

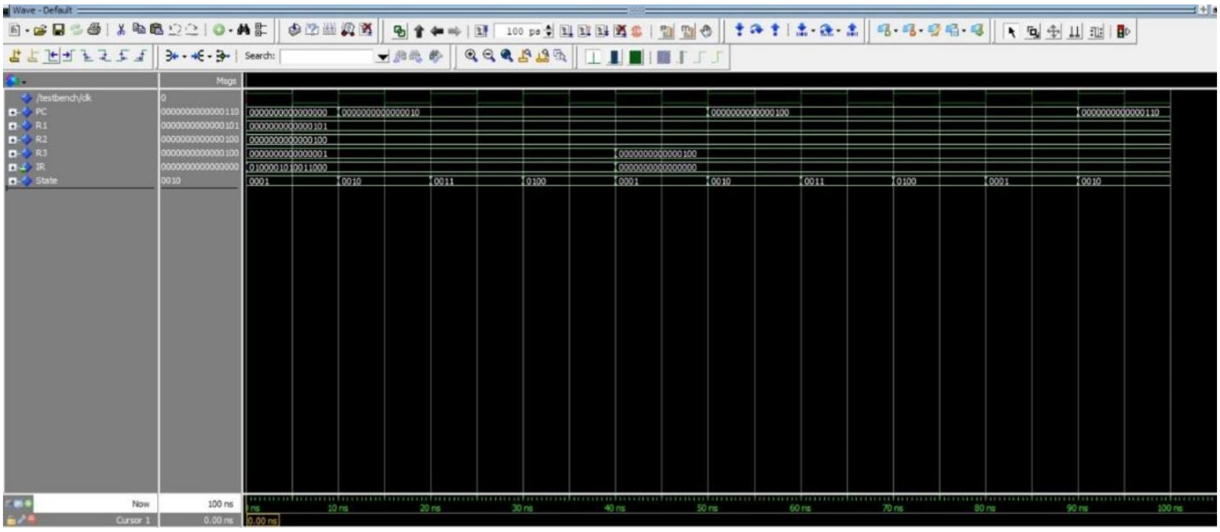
ADD R1 R2 R3



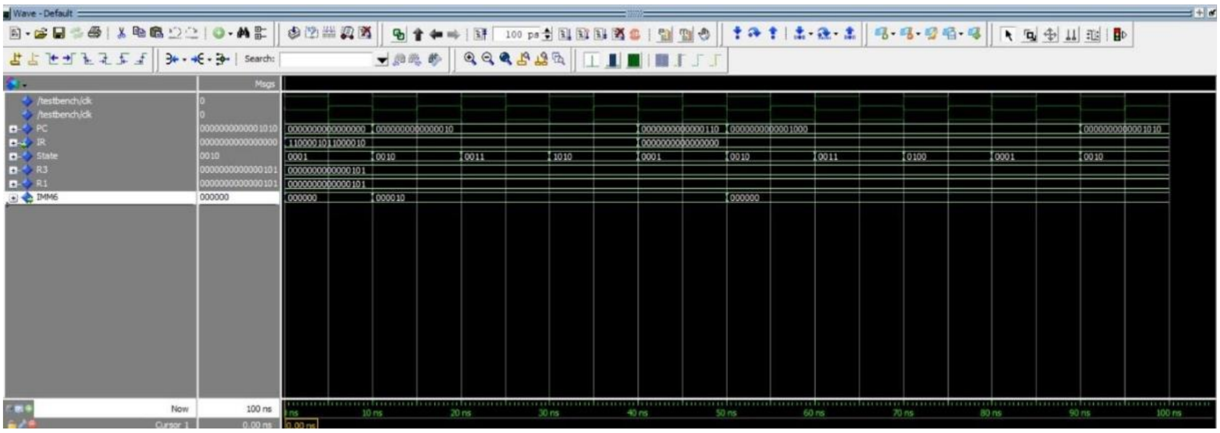
ADI R1 R4 100010



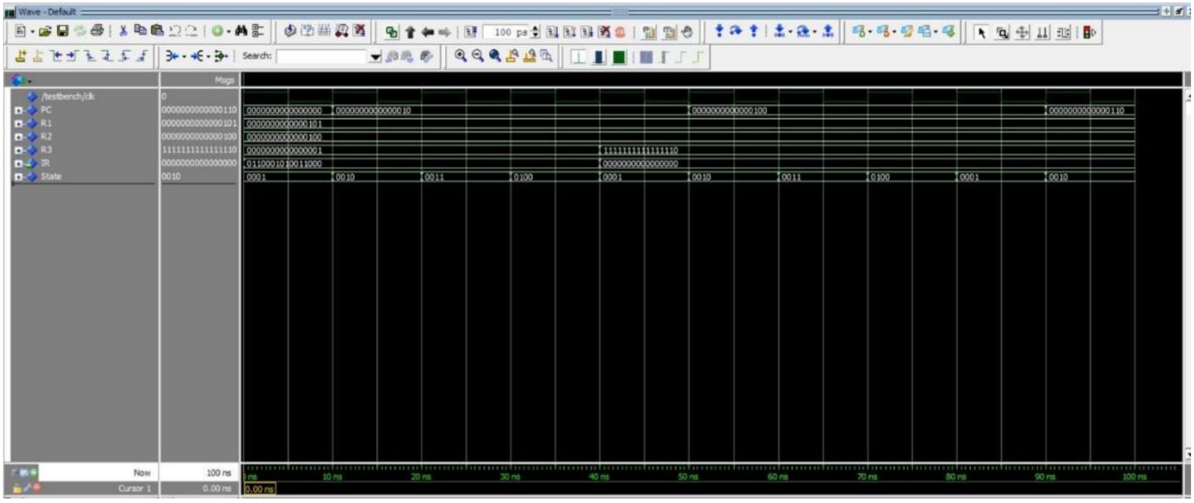
AND R1 R2 R3



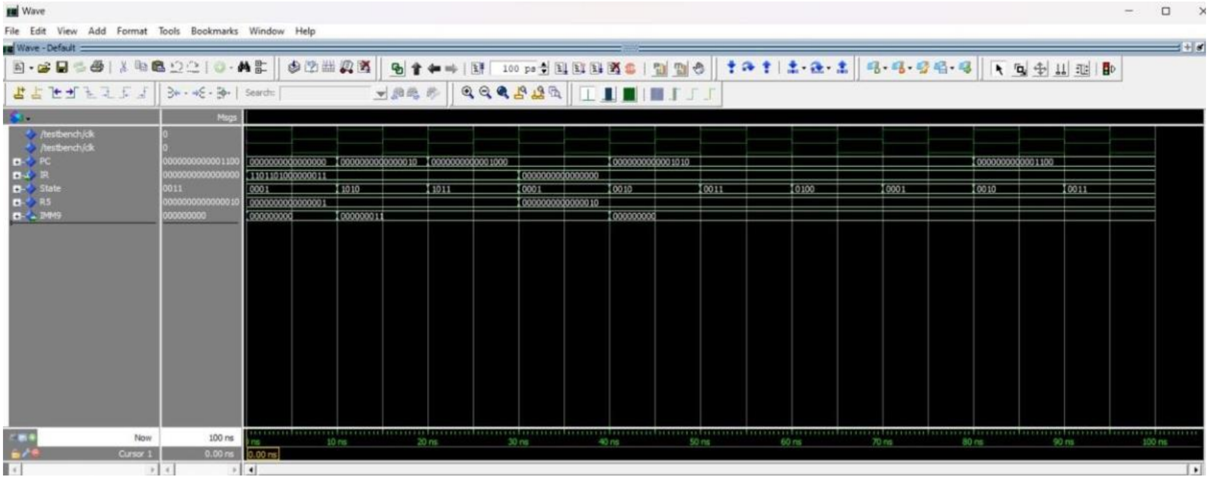
BEQR1 R3000010



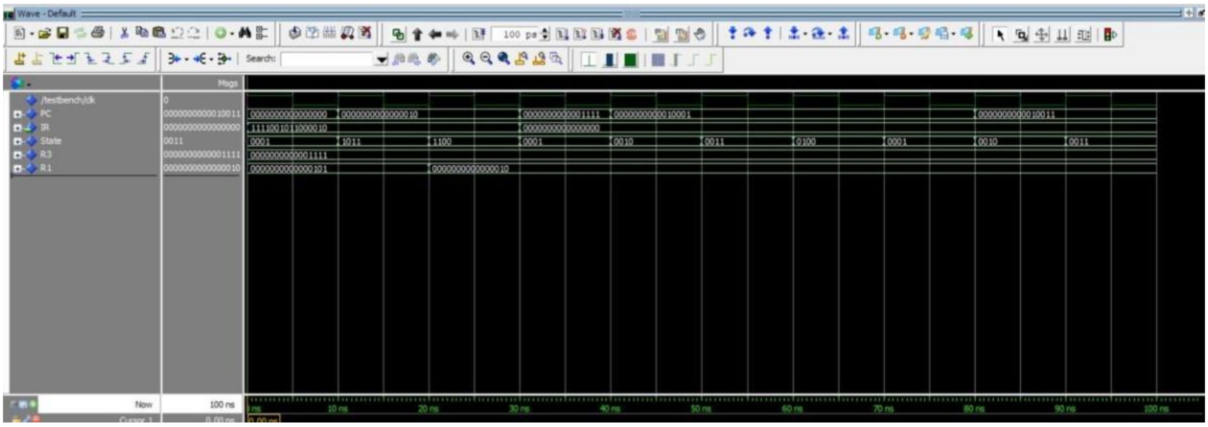
IMP R1 R2 R3



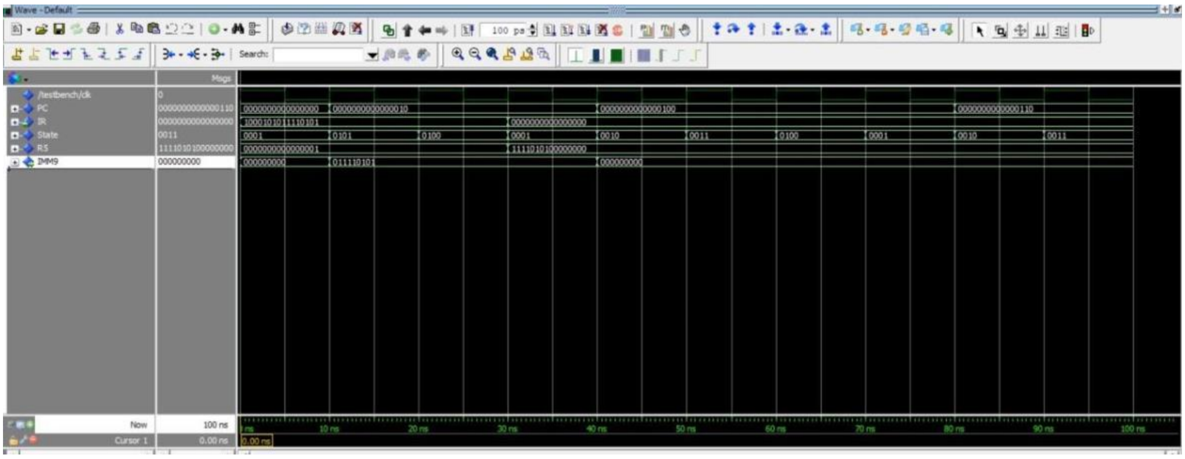
JAL R5 000000011



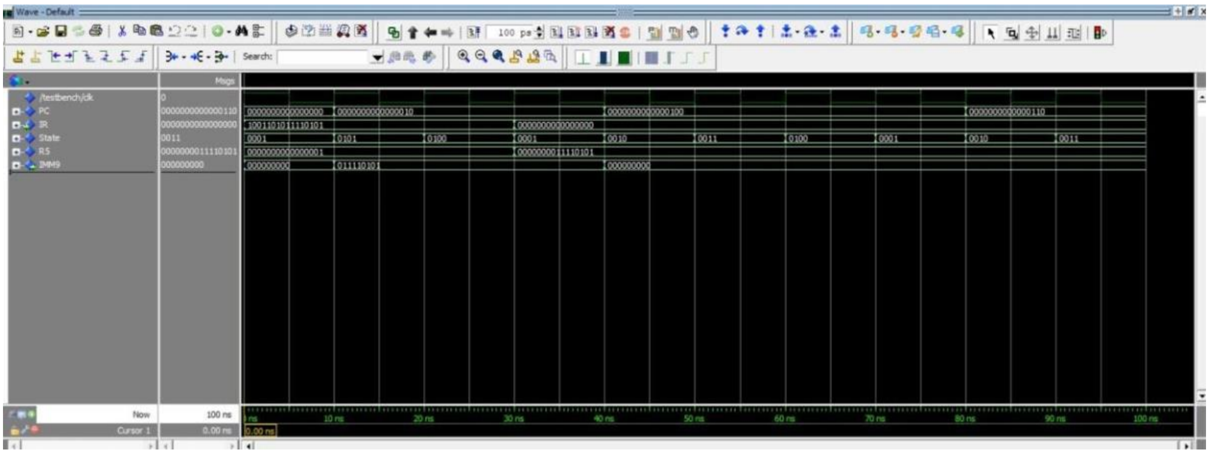
JLR R1 R3 000000



LHI R5 11110101

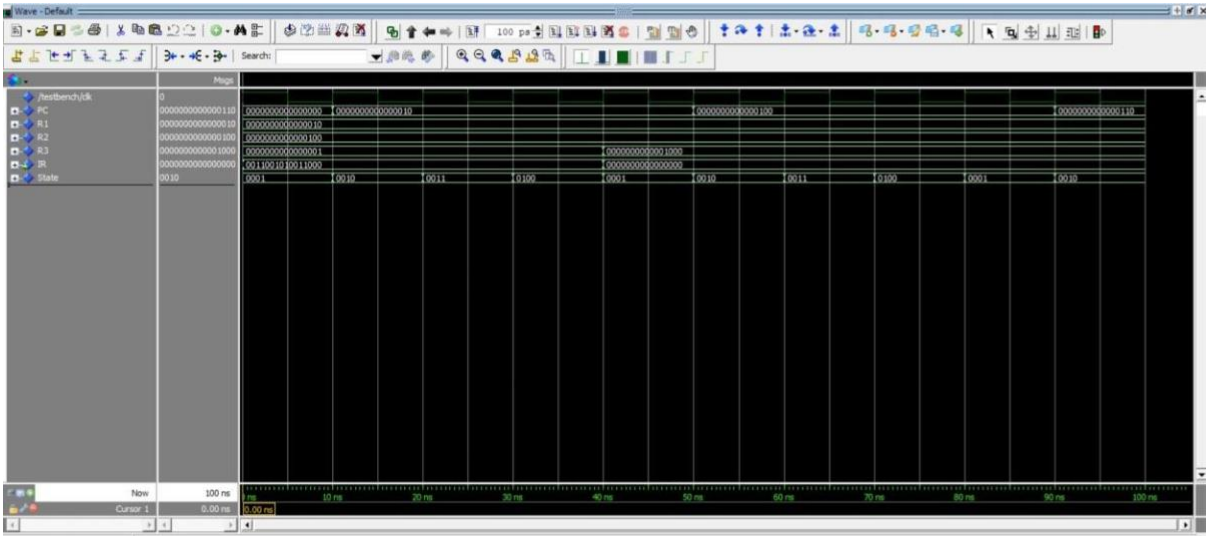


LLI R5 11110101



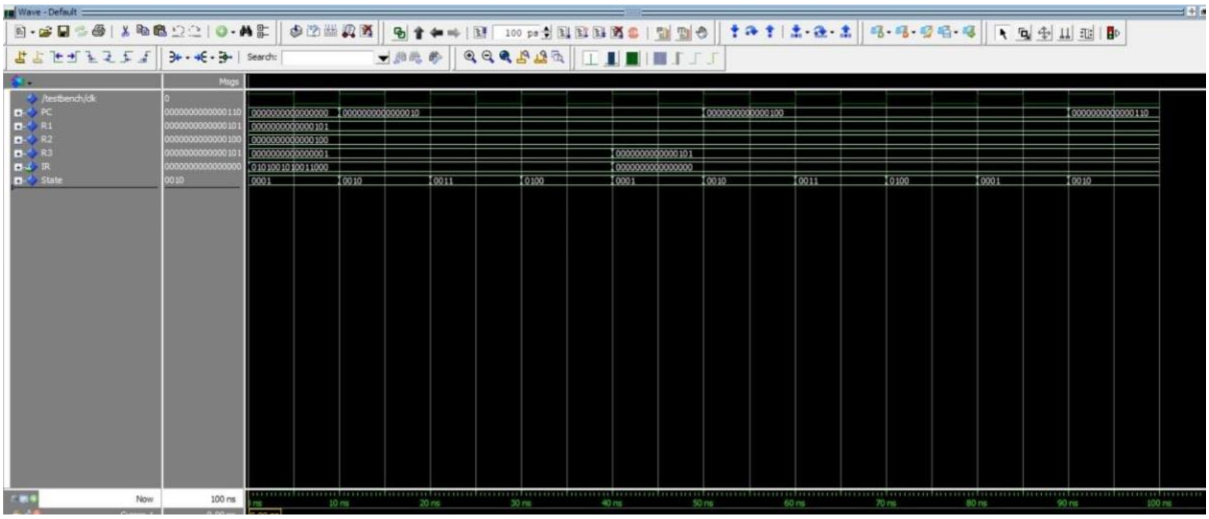
```
/testbench/clk   0
PC               0000000000000110   0000000000000000  0000000000000010  0000000000000100  0000000000000110
IR               0000000000000000   1001101011110101  0000000000000000
State            0011               0001  0101  0100  0001  0010  0011  0100  0001  0010  0011
R5               0000000011110101   0000000000000001  0000000011110101
IMM9             000000000          000000000  011110101  000000000
```

MUL R1 R2 R3



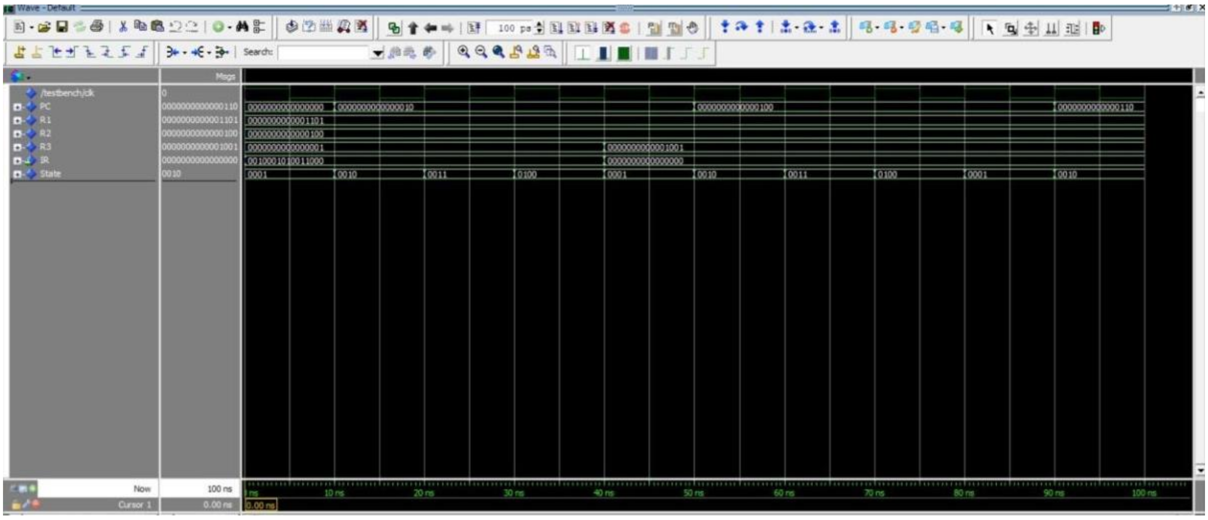
```
/testbench/clk   0
PC               0000000000000110   0000000000000000  0000000000000010  0000000000000100  0000000000000110
R1               0000000000000010   0000000000000010
R2               0000000000000100   0000000000000100
R3               0000000000001000   0000000000000001  0000000000001000
IR               0000000000000000   0011001010011000  0000000000000000
State            0010               0001  0010  0011  0100  0001  0010  0011  0100  0001  0010
```

ORA R1 R2 R3

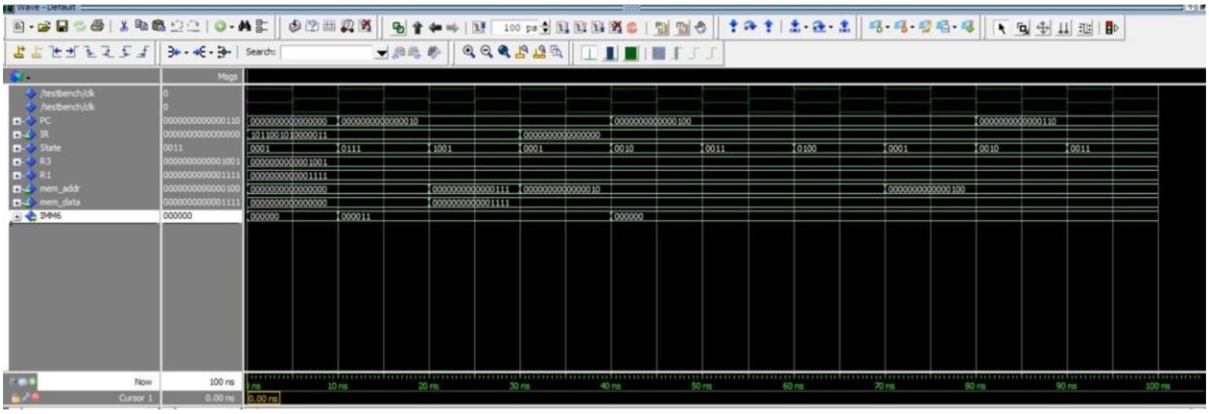


```
/testbench/clk   0
PC               0000000000000110   0000000000000000  0000000000000010  0000000000000100  0000000000000110
R1               0000000000000101   0000000000000101
R2               0000000000000100   0000000000000100
R3               0000000000000101   0000000000000001  0000000000000101
IR               0000000000000000   0101001010011000  0000000000000000
State            0010               0001  0010  0011  0100  0001  0010  0011  0100  0001  0010
```

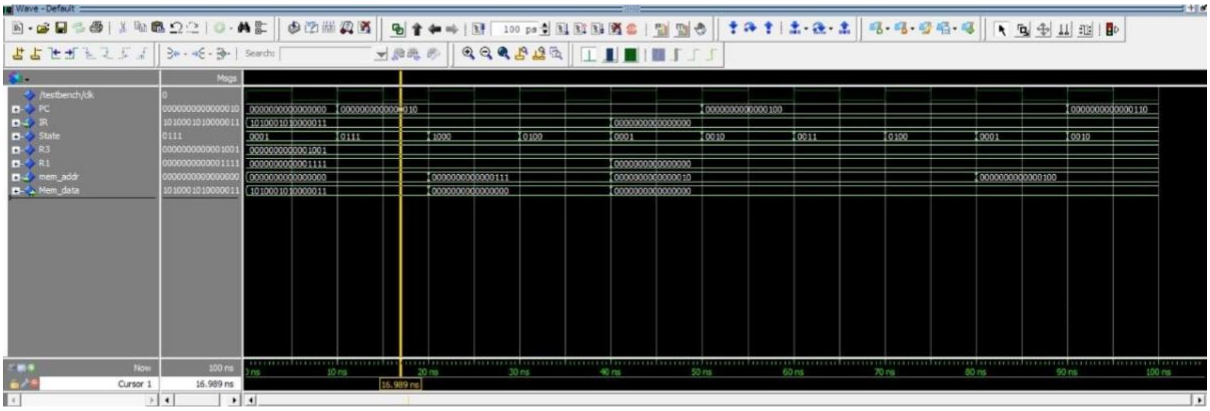
SUB R1 R2 R3



SW R1 R3 000011



LW R1 R3 000011



Work Distribution

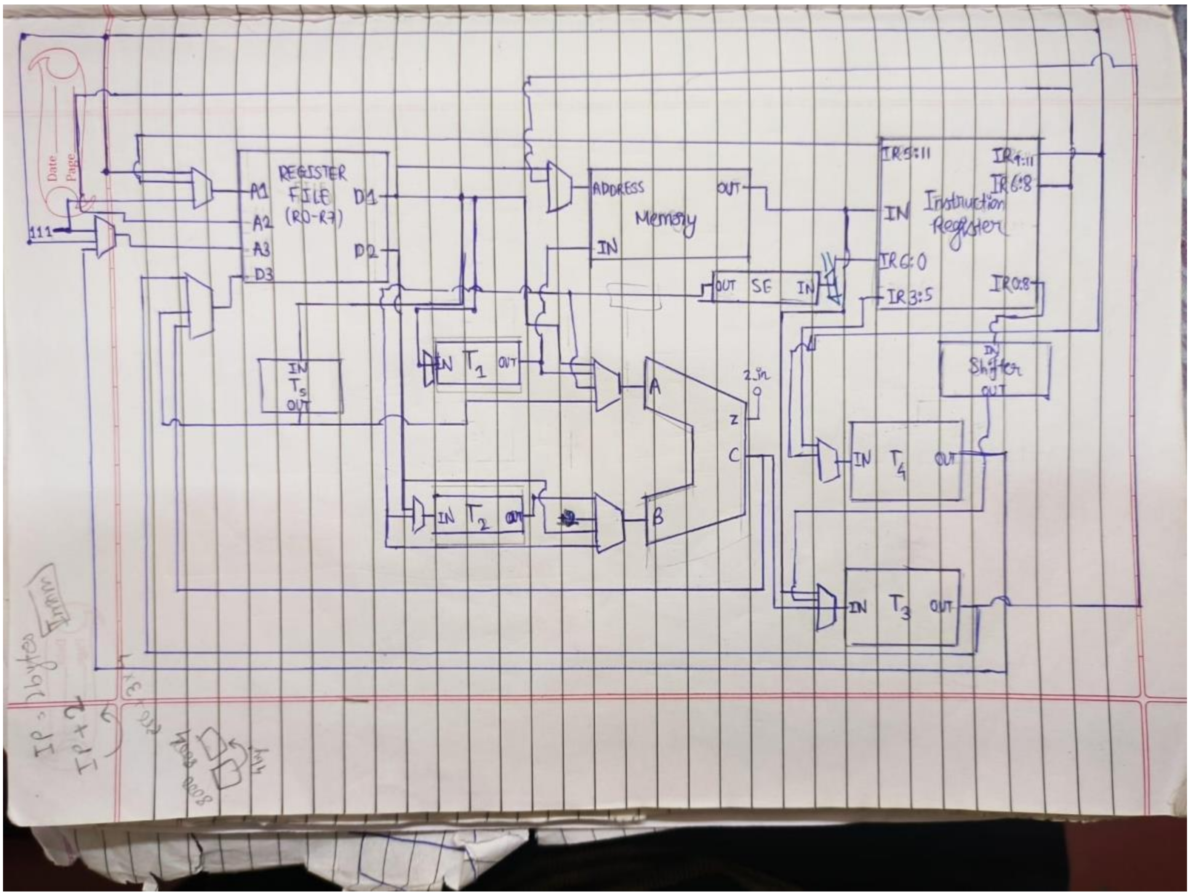
Aman M - ALU, Registers, Register file, and additional feature. Pin planning for board implementation, added the output signals, tested the board implementation.

Aman R - Initial design of FSM and its VHDL implementation, Shifter, and testing & debugging, created state for reset , solved the issues related to PC. Debugged the latches warning, added the output signals, created a top level entity with outputs for board implementation.

Chinmay - Initial flowcharts of each instruction, Instruction register, SE, Memory, and testing & debugging,final fsm debugged the latches warning. Halt satellite debugging.

Swarup - Initial design of DataPath and its VHDL implementation, Final report and testing & debugging, created state for halt, Combined DataPath And FSM, added the output signals, made changes to memory,final debugging for board implementation with halt.

Complete DataPath

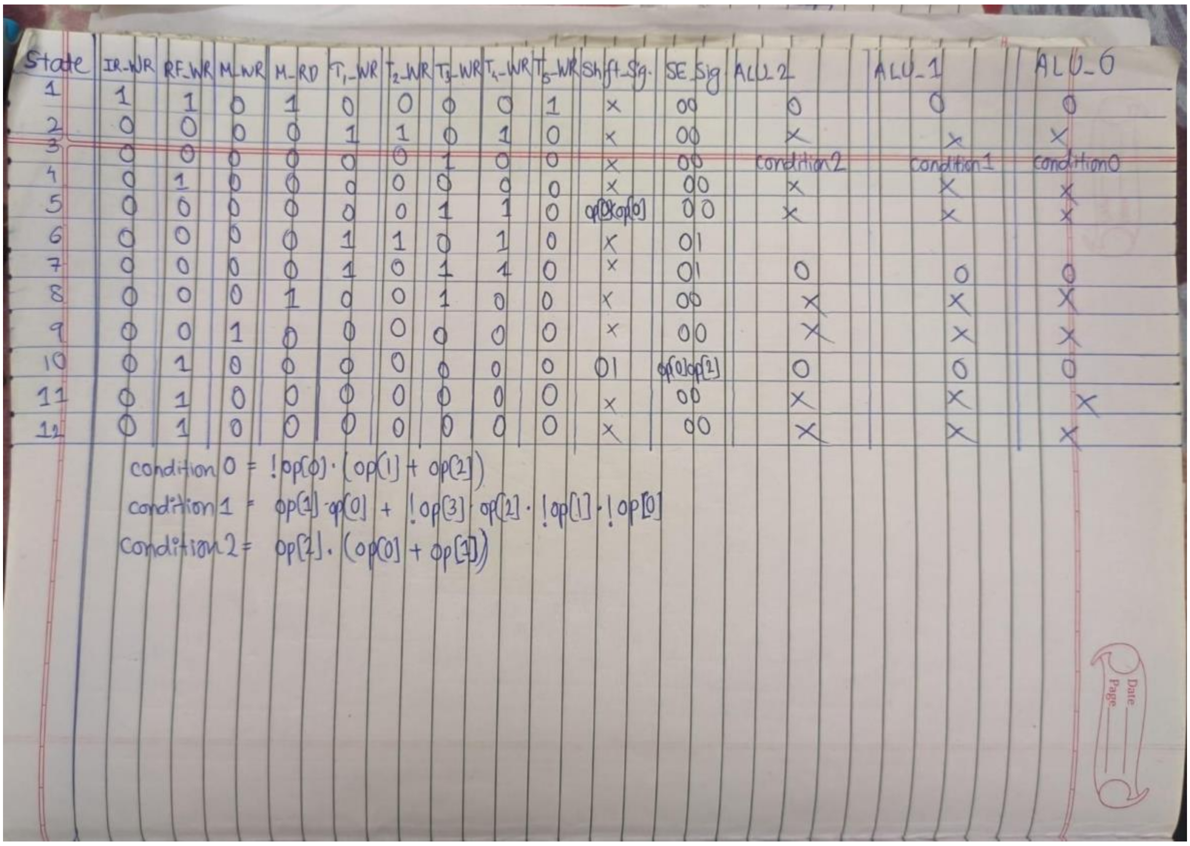


Control Signals

To control the operations of various components in a given state we have used controlsignals:

- Memory read enable(M_RD)
- Memory write enable(M_WR)
- Instruction register write enable(IR_WR)
- Temporary register1 write enable(T1_WR)
- Temporary register2 write enable(T2_WR)
- Temporary register3 write enable(T3_WR)
- Temporary register4 write enable(T4_WR)
- Temporary register5 write enable(T5_WR)
- Sign extender signals (SE_signal(1) and SE_signal(0))
- Shifter signal (shift_signal(1) and shift_signal(0))
- ALU control signal (ALU_CTRL(2), ALU_CTRL(1) and ALU_CTRL(0))

Most of the signal are like on-off switch but certain control signals depend on the instruction being executed so those signals depend on the opcode.



State	IR-WR	RF-WR	M-WR	M-RD	T_1-WR	T_2-WR	T_3-WR	T_4-WR	T_5-WR	Shift-Sig.	SE Sig	ALU 2	ALU_1	ALU_0
1	1	1	0	1	0	0	0	0	1	X	00	0	0	0
2	0	0	0	0	1	1	0	1	0	X	00	X	X	X
3	0	0	0	0	0	0	1	0	0	X	00	condition2	condition1	condition0
4	0	1	0	0	0	0	0	0	0	X	00	X	X	X
5	0	0	0	0	0	0	1	1	0	op[1]op[0]	00	X	X	X
6	0	0	0	0	1	1	0	1	0	X	01			
7	0	0	0	0	1	0	1	1	0	X	01	0	0	0
8	0	0	0	1	0	0	1	0	0	X	00	X	X	X
9	0	0	1	0	0	0	0	0	0	X	00	X	X	X
10	0	1	0	0	0	0	0	0	0	01	op[0]op[2]	0	0	0
11	0	1	0	0	0	0	0	0	0	X	00	X	X	X
12	0	1	0	0	0	0	0	0	0	X	00	X	X	X

condition 0 = !op(0)·(op(1) + op(2))

condition 1 = op(1)·op(0) + !op(3)·op(2)·!op(1)·!op[0]

condition 2 = op[2]·(op(0) + op[1])

Significant Changes to Architecture for FPGA Implementation

- **Top-Level Entity:**
  - A new top-level entity named 'Board' has been created. This entity is designed to interface with switches, buttons, and a clock as inputs and outputs through LEDs.

- **Reset Button:**
  The introduction of a reset button (Button 3) has been implemented. This button is configured to reset the Finite State Machine (FSM), datapath, and other components, except for memory.

- **PC and R7 Handling:**
  - An issue with the Program Counter (PC) and Register 7 (R7) has been addressed. The solution includes the integration of two 2x1 multiplexers to effectively manage the output of R7.

- **Datapath Update:**
  - The datapath has been updated to include the FSM as a component, enhancing the system's modular structure and operational efficiency.
  - Additional output signals such as reg_dp_out and mem_dp_out have been incorporated. These signals store output data and connect with the top-level Board entity for display purposes.

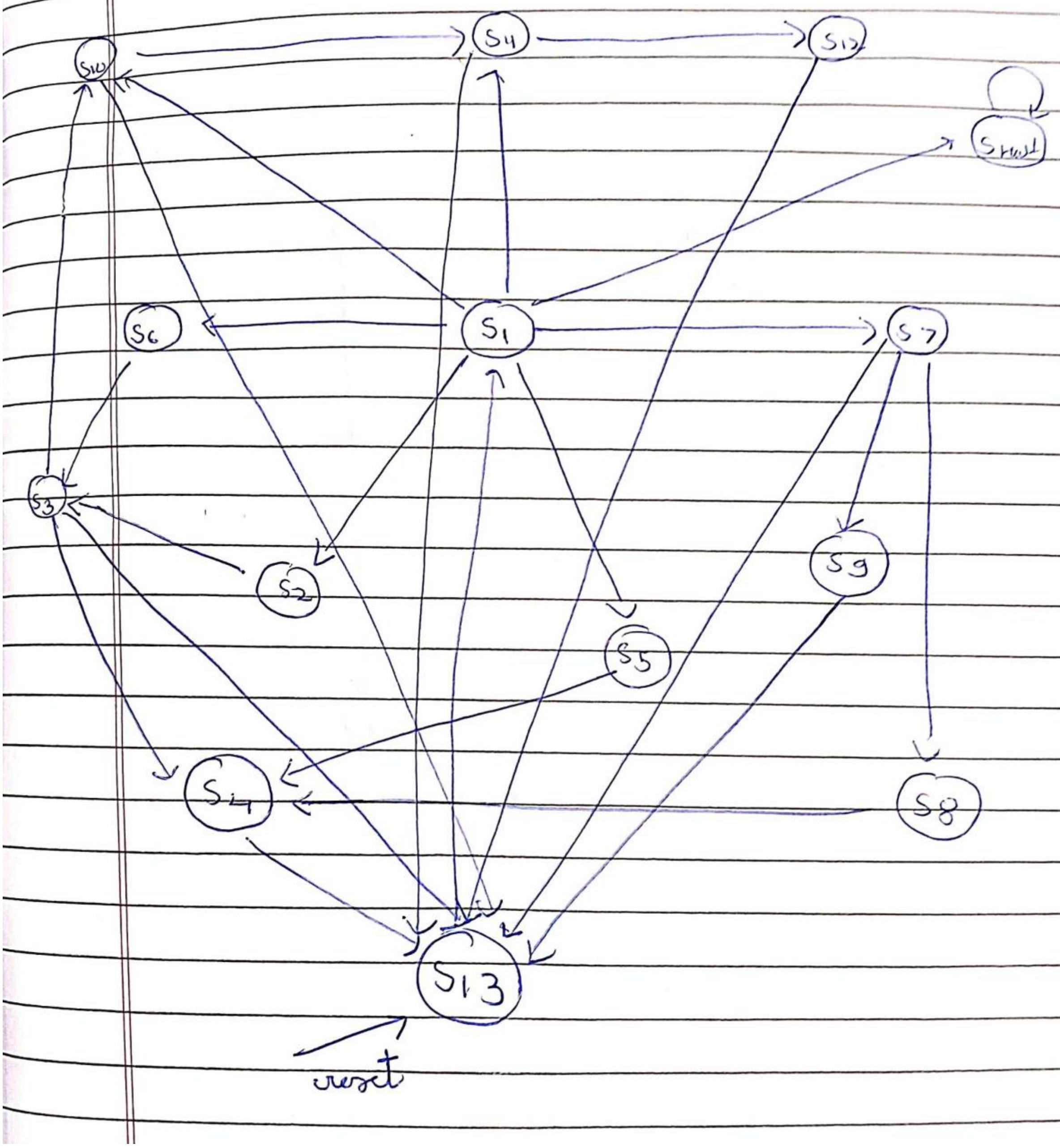
- **FSM State Update:**
  - A new state, termed state 13, has been added to the FSM. This state functions as a reset state, facilitating the storage of data from memory and registers into output signals.

- **Memory Changes**:
  - A significant modification has been made to the memory configuration. The memory has been transitioned from byte addressable to word addressable, aligning with common practices in digital system design.

- **Initialization within Process:**
  -To address warnings related to latches, the design approach was adjusted to initialize signals within the process. This ensures that all signals have defined values, reducing the risk of latch creation, which can lead to unintended behavior in synthesized hardware.

- **FSM and Datapath Integration:**
  -The Finite State Machine (FSM) was merged with the datapath to eliminate unnecessary warnings. This integration streamlined the design, allowing for more efficient communication between the control and data handling segments of the system, and reduced the complexity that could lead to synthesis and implementation issues.

  - **Modified FSM**


S10
S4
S12
Shalt
S6
S1
S7
S3
S9
S2
S5
S14
S8
S13
reset