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

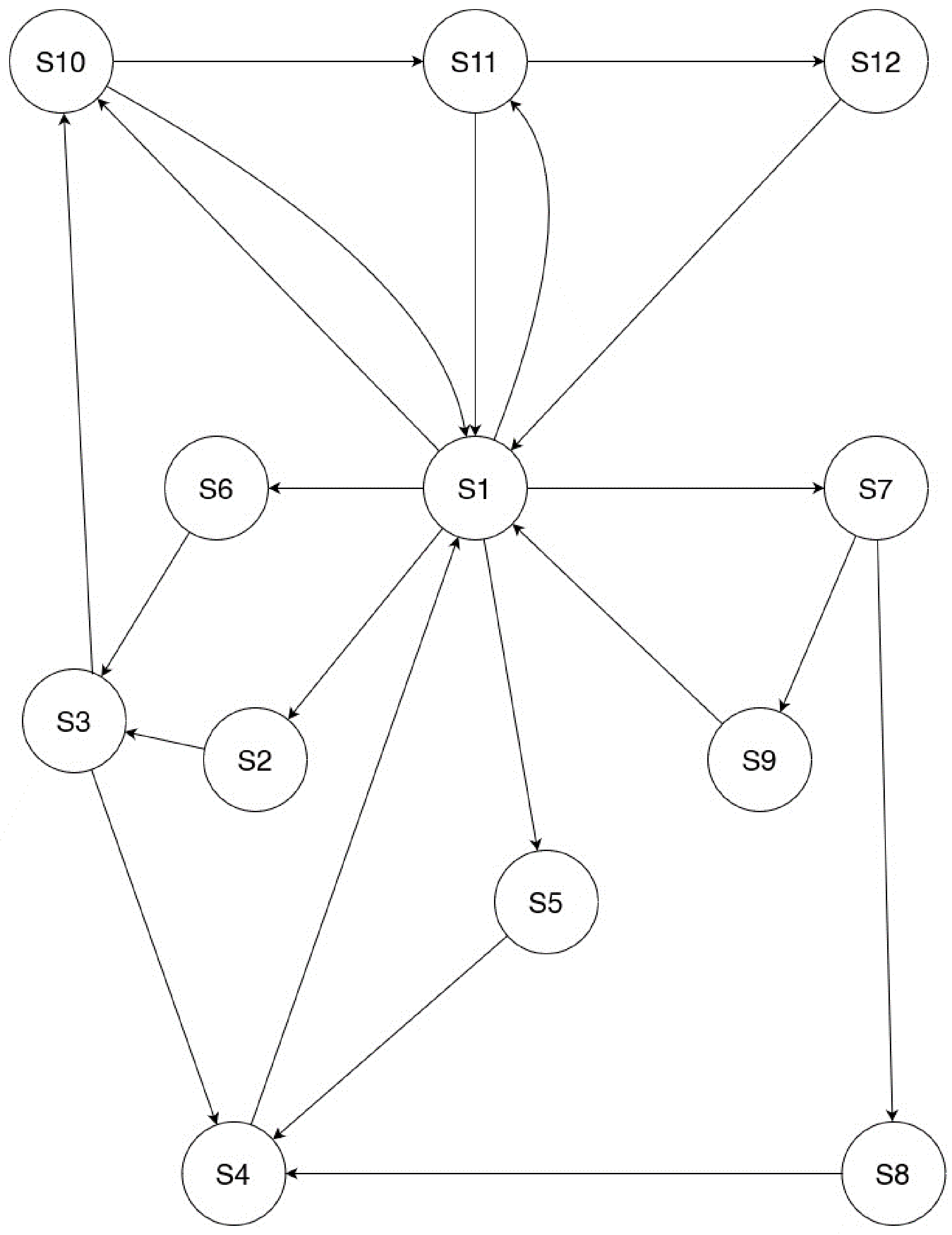
State 12: Branch to Reg. B

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

$$\begin{tikzcd}
S1 \arrow[r] & S2 \arrow[r] & S3 \arrow[r] & S4 \arrow[lll, bend left]
\end{tikzcd}$$

J- Type

$$\begin{tikzcd}
S1 \arrow[r] & S5 \arrow[r] & S4 \arrow[ll, bend left]
\end{tikzcd}$$

ADI

$$\begin{tikzcd}
S1 \arrow[r] & S6 \arrow[r] & S3 \arrow[r] & S4 \arrow[lll, bend left]
\end{tikzcd}$$

LW

$$\begin{tikzcd}
S1 \arrow[r] & S7 \arrow[r] & S8 \arrow[r] & S4 \arrow[lll, bend left]
\end{tikzcd}$$

Text is not SVG - cannot display

SW

$$\begin{tikzcd}
S1 \arrow[r] & S7 \arrow[r] & S9 \arrow[ll, bend left]
\end{tikzcd}$$

BEQ

$$\begin{tikzcd}
S1 \arrow[r] & S2 \arrow[r] & S3 \arrow[ll, bend right, "z=0"'] \arrow[r, "z=1"] & S10 \arrow[lll, bend left]
\end{tikzcd}$$

JAL

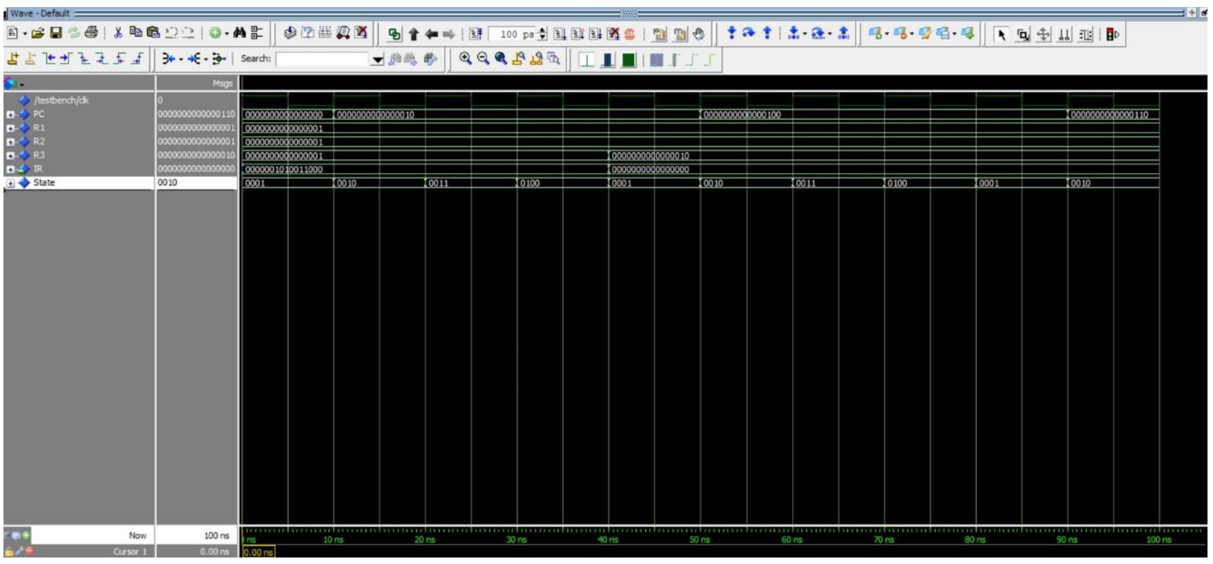
$$\begin{tikzcd}
S1 \arrow[r] & S10 \arrow[r] & S11 \arrow[ll, bend left]
\end{tikzcd}$$

JLR

$$\begin{tikzcd}
S1 \arrow[r] & S11 \arrow[r] & S12 \arrow[ll, bend left]
\end{tikzcd}$$

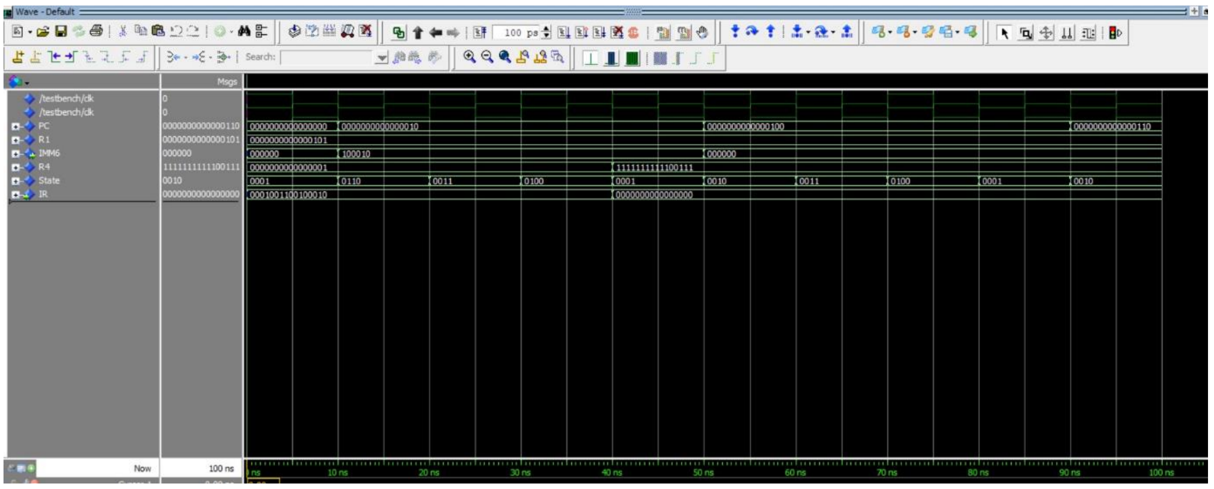
RTL SIMULATIONS

ADD R1 R2 R3



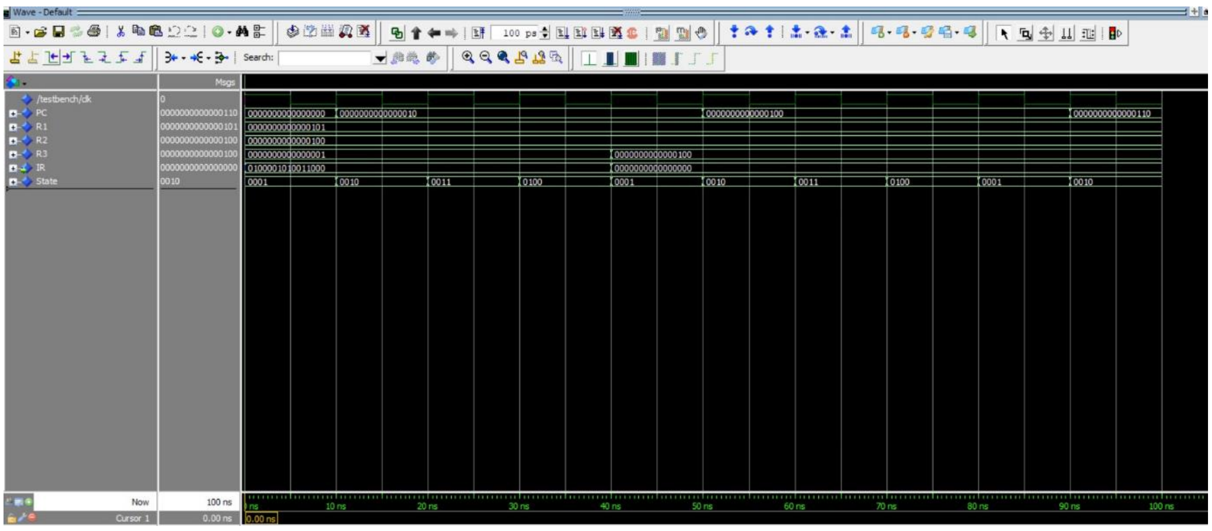
```
Wave - Default
                    Msgs
/testbench/clk      0
PC                  0000000000000110
R1                  0000000000000001
R2                  0000000000000001
R3                  0000000000000010
IR                  0000000000000000
State               0010
Now                 100 ns
Cursor 1            0.00 ns
```

ADI R1 R4 100010



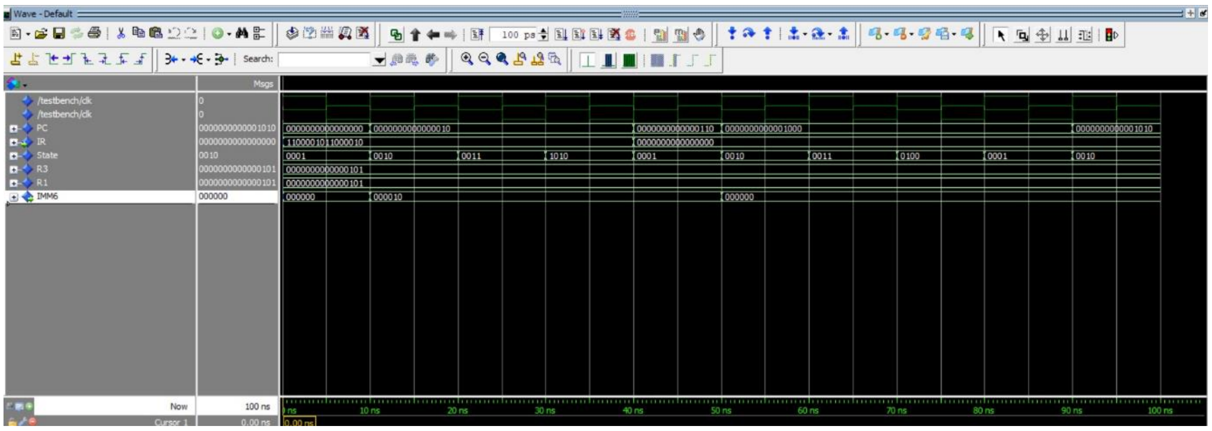
```
Wave - Default
                    Msgs
/testbench/clk      0
/testbench/clk      0
PC                  0000000000000110
R1                  0000000000000101
IMM6                000000
R4                  1111111111100111
State               0010
IR                  0000000000000000
Now                 100 ns
```

AND R1 R2 R3



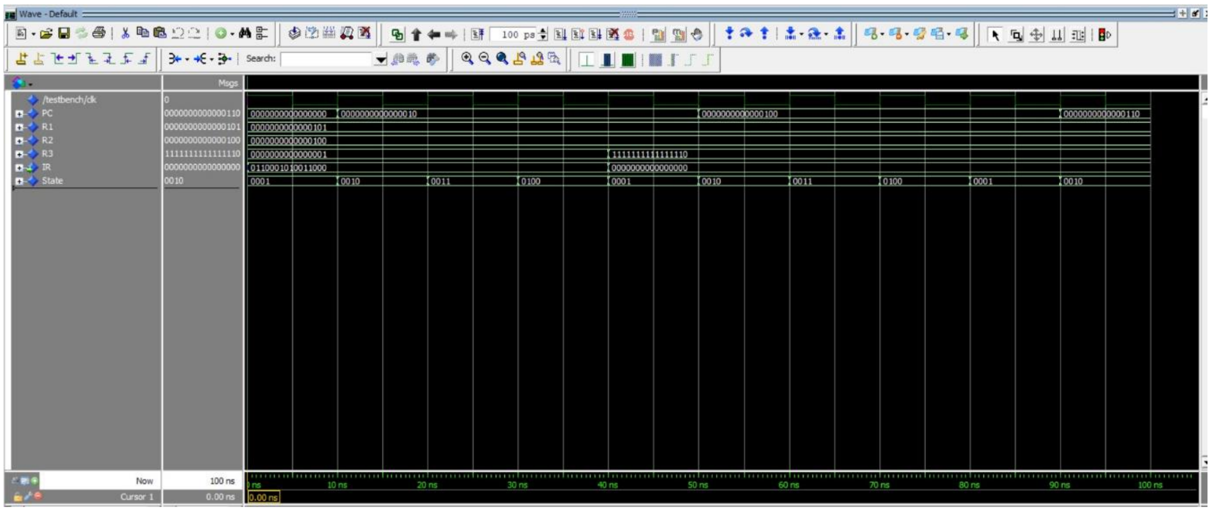
```
Wave - Default
                     Msgs
/testbench/clk       0
PC                   0000000000000110   0000000000000000 | 0000000000000010 | 0000000000000100 | 0000000000000110
R1                   0000000000000101   0000000000000101
R2                   0000000000000100   0000000000000100
R3                   0000000000000100   0000000000000001 | 0000000000000100
IR                   0000000000000000   0100001010011000 | 0000000000000000
State                0010               0001 | 0010 | 0011 | 0100 | 0001 | 0010 | 0011 | 0100 | 0001 | 0010
Now                  100 ns
Cursor 1             0.00 ns            0.00 ns
```

BEQR1 R3000010



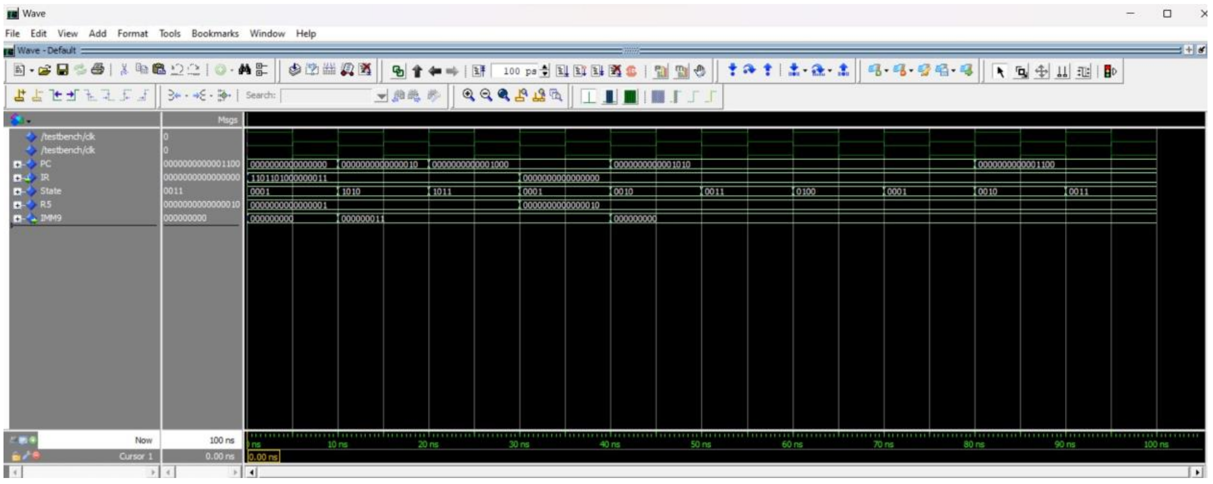
```
Wave - Default
                     Msgs
/testbench/clk       0
/testbench/clk       0
PC                   0000000000001010   0000000000000000 | 0000000000000010 | 0000000000000110 | 0000000000001000 | 0000000000001010
IR                   0000000000000000   1100001011000010 | 0000000000000000
State                0010               0001 | 0010 | 0011 | 1010 | 0001 | 0010 | 0011 | 0100 | 0001 | 0010
R3                   0000000000000101   0000000000000101
R1                   0000000000000101   0000000000000101
IMM6                 000000             000000 | 000010 | 000000
Now                  100 ns
Cursor 1             0.00 ns            0.00 ns
```

IMP R1 R2 R3



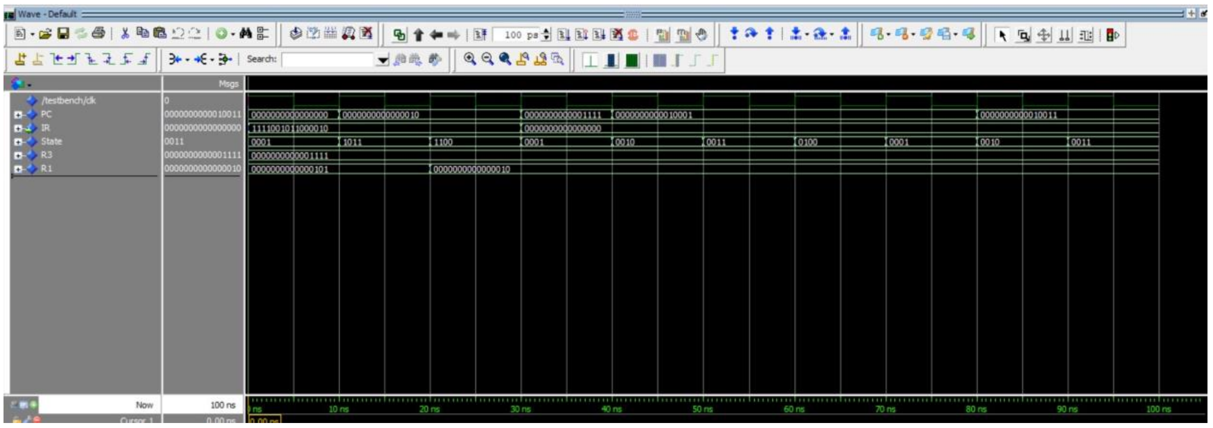
```
Wave - Default
                     Msgs
/testbench/clk       0
PC                   0000000000000110   0000000000000000 | 0000000000000010 | 0000000000000100 | 0000000000000110
R1                   0000000000000101   0000000000000101
R2                   0000000000000100   0000000000000100
R3                   1111111111111110   0000000000000001 | 1111111111111110
IR                   0000000000000000   0110001010011000 | 0000000000000000
State                0010               0001 | 0010 | 0011 | 0100 | 0001 | 0010 | 0011 | 0100 | 0001 | 0010
Now                  100 ns
Cursor 1             0.00 ns            0.00 ns
```

JAL R5 000000011



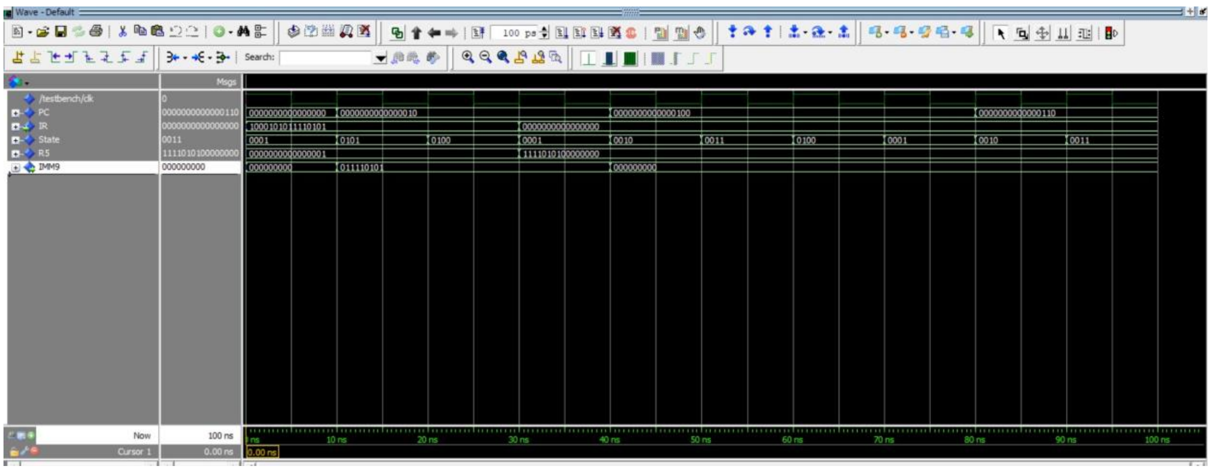
```
/testbench/clk   0
/testbench/clk   0
PC      0000000000001100   0000000000000000 | 0000000000000010 | 0000000000001000 | 0000000000001010 | 0000000000001100
IR      0000000000000000   1101101000000011 | 0000000000000000
State   0011               0001 | 1010 | 1011 | 0001 | 0010 | 0011 | 0100 | 0001 | 0010 | 0011
R5      0000000000000010   0000000000000001 | 0000000000000010
IMM9    000000000          000000000 | 000000011 | 000000000
Now 100 ns
Cursor 1 0.00 ns
```

JLR R1 R3 000000



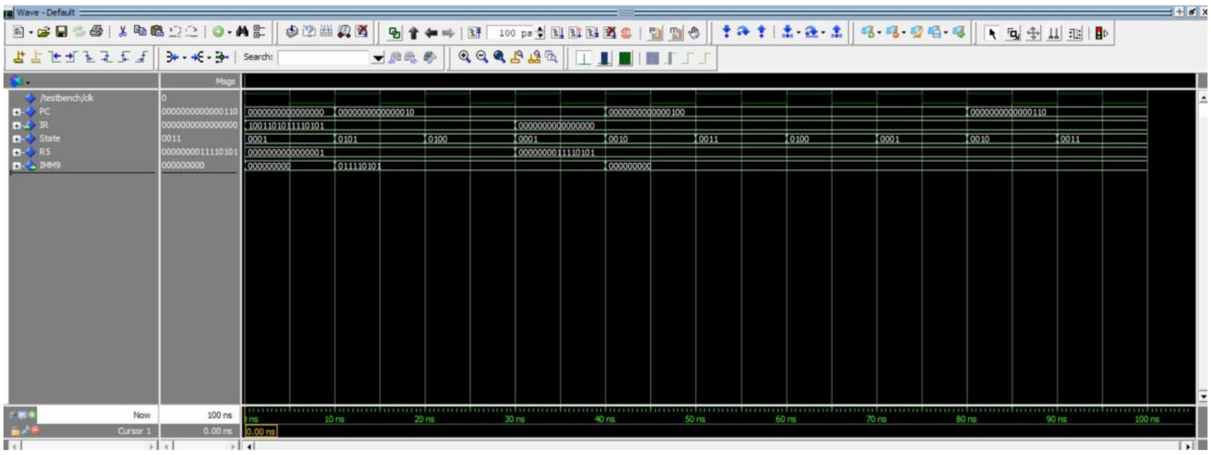
```
/testbench/clk   0
PC      0000000000010011   0000000000000000 | 0000000000000010 | 0000000000001111 | 0000000000010001 | 0000000000010011
IR      0000000000000000   1111001011000010 | 0000000000000000
State   0011               0001 | 1011 | 1100 | 0001 | 0010 | 0011 | 0100 | 0001 | 0010 | 0011
R3      0000000000001111   0000000000001111
R1      0000000000000010   0000000000000101 | 0000000000000010
Now 100 ns
```

LHI R5 11110101



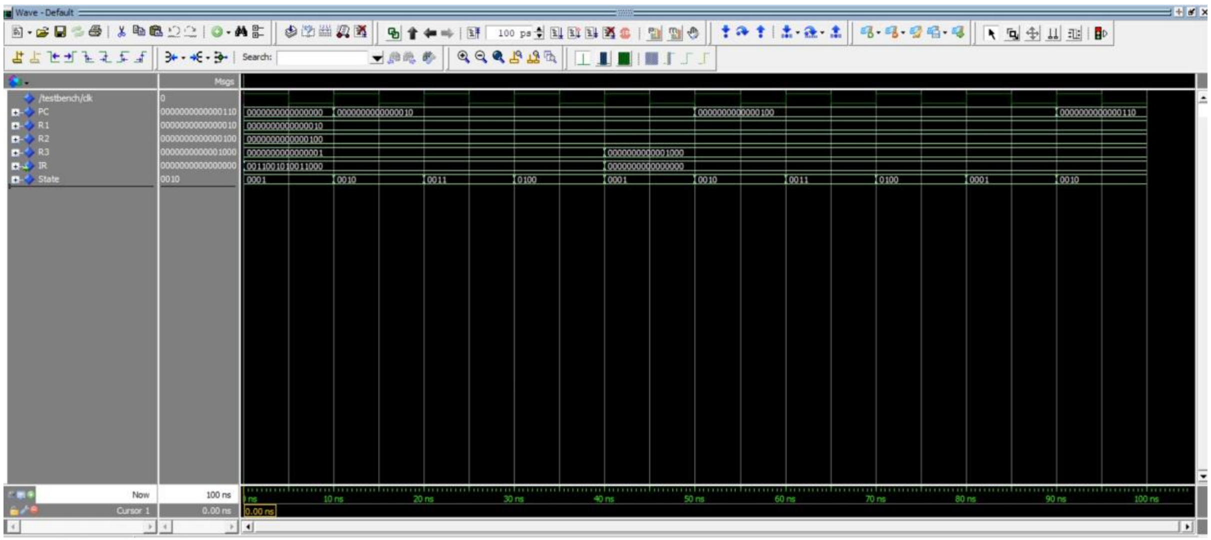
```
/testbench/clk   0
PC      0000000000000110   0000000000000000 | 0000000000000010 | 0000000000000100 | 0000000000000110
IR      0000000000000000   1000101011110101 | 0000000000000000
State   0011               0001 | 0101 | 0100 | 0001 | 0010 | 0011 | 0100 | 0001 | 0010 | 0011
R5      1111010100000000   0000000000000001 | 1111010100000000
IMM9    000000000          000000000 | 011110101 | 000000000
Now 100 ns
Cursor 1 0.00 ns
```

LLI R5 11110101



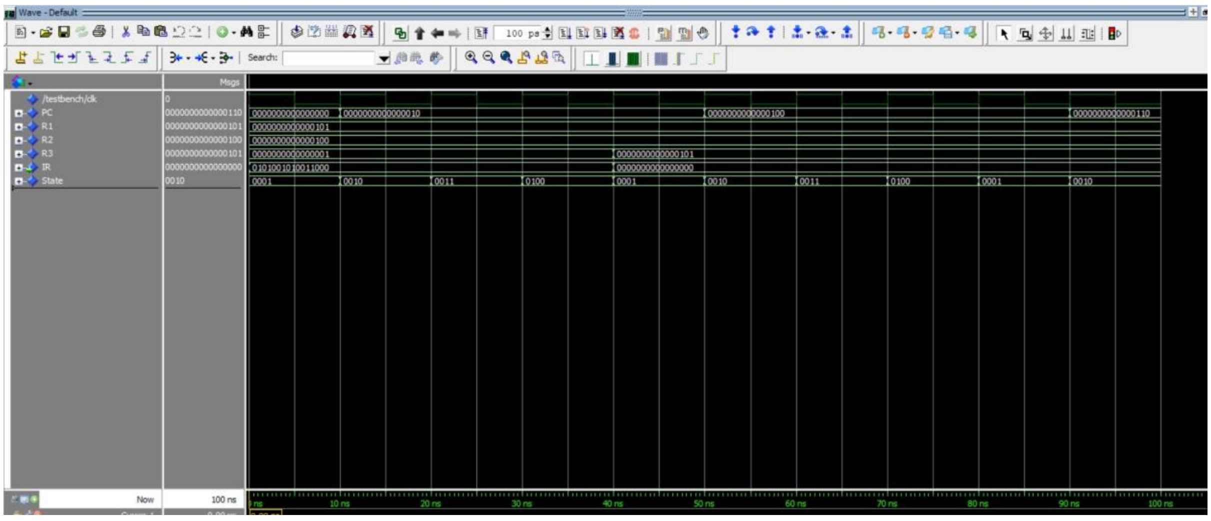
```
Wave - Default
                  Msgs
/testbench/clk    0
PC                0000000000000110   0000000000000000  0000000000000010  0000000000000100  0000000000000110
IR                0000000000000000   1001101011110101  0000000000000000
State             0011               0001  0101  0100  0001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

Now       100 ns
Cursor 1  0.00 ns
0 ns  10 ns  20 ns  30 ns  40 ns  50 ns  60 ns  70 ns  80 ns  90 ns  100 ns
```

MUL R1 R2 R3



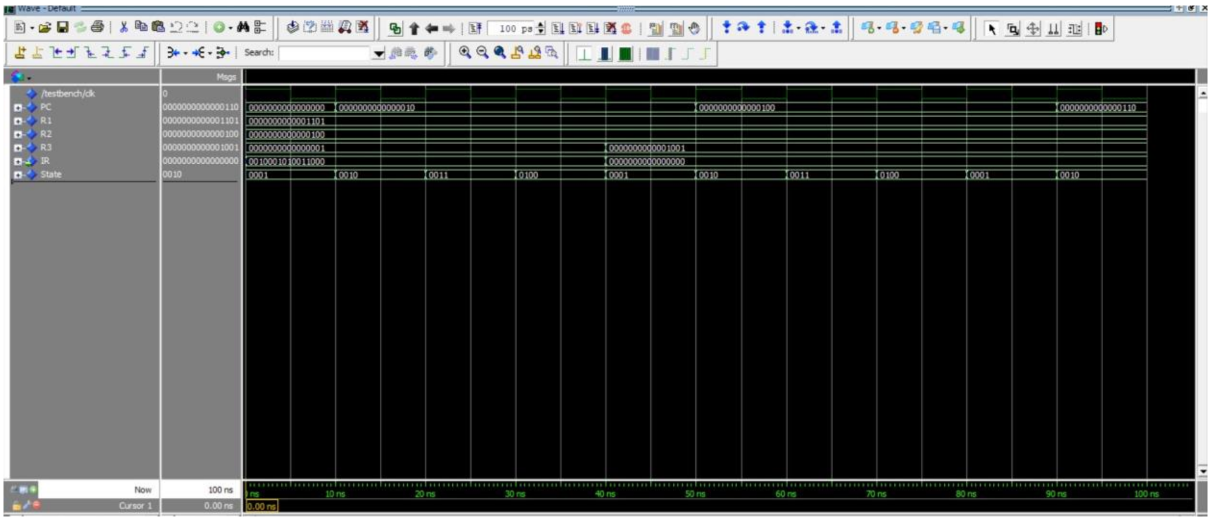
```
Wave - Default
                  Msgs
/testbench/clk    0
PC                0000000000000110   0000000000000000  0000000000000010  0000000000000100  0000000000000110
R1                0000000000000010   0000000000000010
R2                0000000000000100   0000000000000100
R3                0000000000001000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

Now       100 ns
Cursor 1  0.00 ns
0 ns  10 ns  20 ns  30 ns  40 ns  50 ns  60 ns  70 ns  80 ns  90 ns  100 ns
```

ORA R1 R2 R3



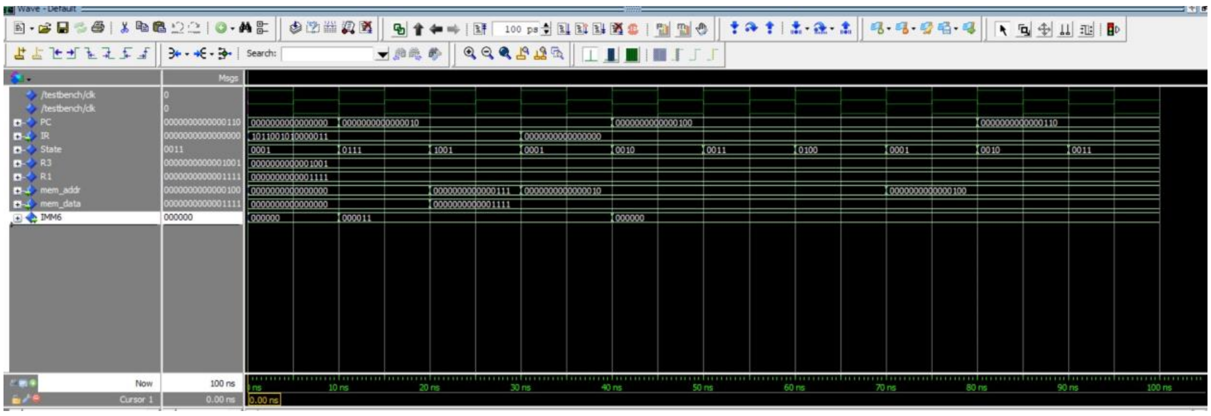
```
Wave - Default
                  Msgs
/testbench/clk    0
PC                0000000000000110   0000000000000000  0000000000000010  0000000000000100  0000000000000110
R1                0000000000000101   0000000000000101
R2                0000000000000100   0000000000000100
R3                0000000000000101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

Now       100 ns
0 ns  10 ns  20 ns  30 ns  40 ns  50 ns  60 ns  70 ns  80 ns  90 ns  100 ns
```

SUB R1 R2 R3



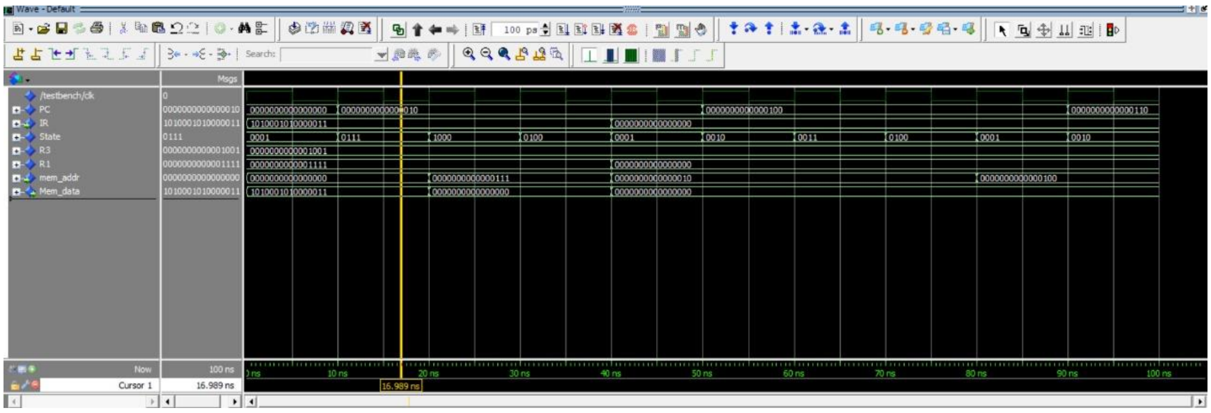
```
/testbench/clk   0
PC               0000000000000110
R1               0000000000001101
R2               0000000000000100
R3               0000000000001001
IR               0000000000000000
State            0010
```

SW R1 R3 000011



```
/testbench/clk   0
/testbench/clk   0
PC               0000000000000110
IR               0000000000000000
State            0011
R3               0000000000001001
R1               0000000000001111
mem_addr         0000000000000100
mem_data         0000000000001111
IMM6             000000
```

LW R1 R3 000011



```
/testbench/clk   0
PC               0000000000000010
IR               1010001010000011
State            0111
R3               0000000000001001
R1               0000000000001111
mem_addr         0000000000000000
Mem_data         1010001010000011
```

Work Distribution

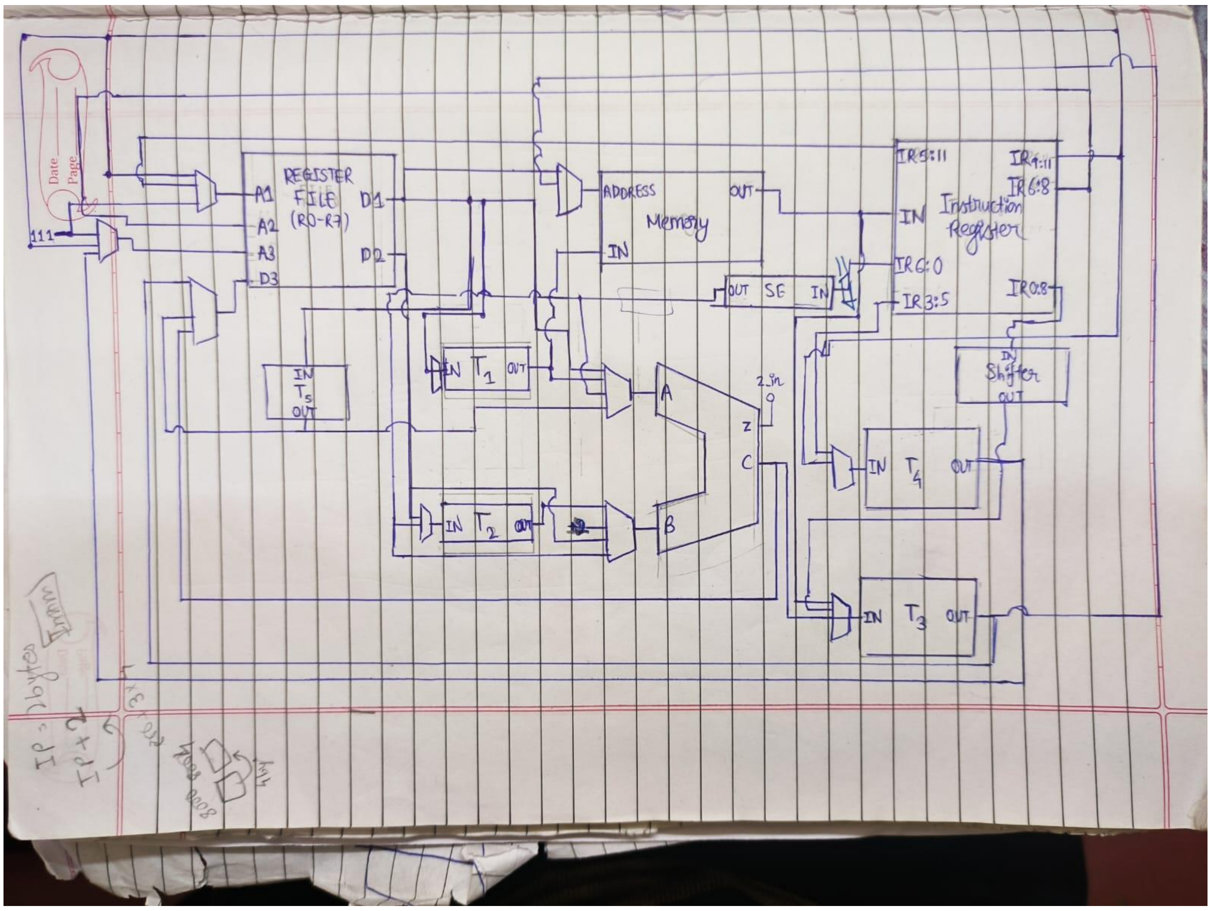
Aman M - ALU , Registers , Register file and additional feature.

Aman R - Initial design of FSM and its VHDL implementation , Shifter and testing & debugging

Chinmay - Initial flowcharts of each instruction, Instruction register, SE , Memory and testing & debugging

Swarup - Initial design of DataPath and its VHDL implementation ,Final report and testing & debugging

Complete DataPath

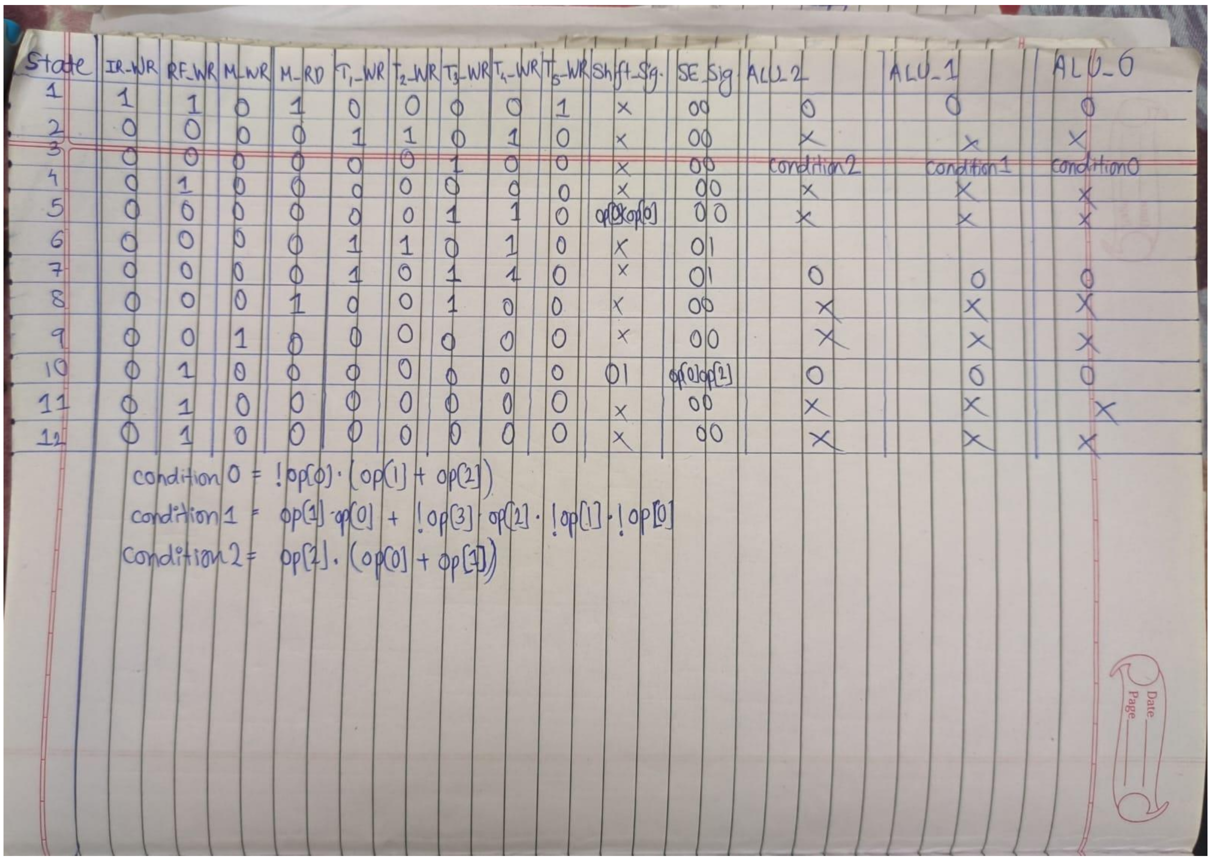


Control Signals

In order to control the operations of various components in a given state we have used control signals:

- 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[3]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

condition0 = !op(0)·(op(1] + op(2])

condition1 = op(1]·op(0] + !op(3]·op(2]·!op(1]·!op[0]

condition2 = op[2].(op(0] + op[1])