IITB-CPU

TEAM-ID - 8

NAME	ROLL NUMBER
Aman Verma	22B3929
Angad Singh	22B1211
Praveen	22B3931
Hardik Khariwal	22B3954

WORK DISTRIBUTION

1. AMAN VERMA

- Define states and did state minimisation
- Designed ALU and defined all necessary functions required • Designed state output code and combined it with state transition code
- Final debugging of main CPU.vhd
- Tested all the internal signals for 4 instructions

Wrote the Output Process to get the required Output

2. ANGAD SINGH

- Designed states, FSM and Datapath
- Designed sign extenders and temporary registers
- Designed state transition code for all states
- Debug and test first 2 functions of ALU
- Tested all the internal signals for 4 instructions

Modified the PC Counter and store to retrieve later

Testing of Code on Board

3. PRAVEEN

- Designed states, FSM and Datapath
- Designed register file including program counter register
- Figures out all decoder logic for state transition
- Debug and test all the components (except ALU)
- Tested all the internal signals for 4 instructions

Modified the reset Logic

Testing of Code on Board

4. HARDIK KHARIWAL

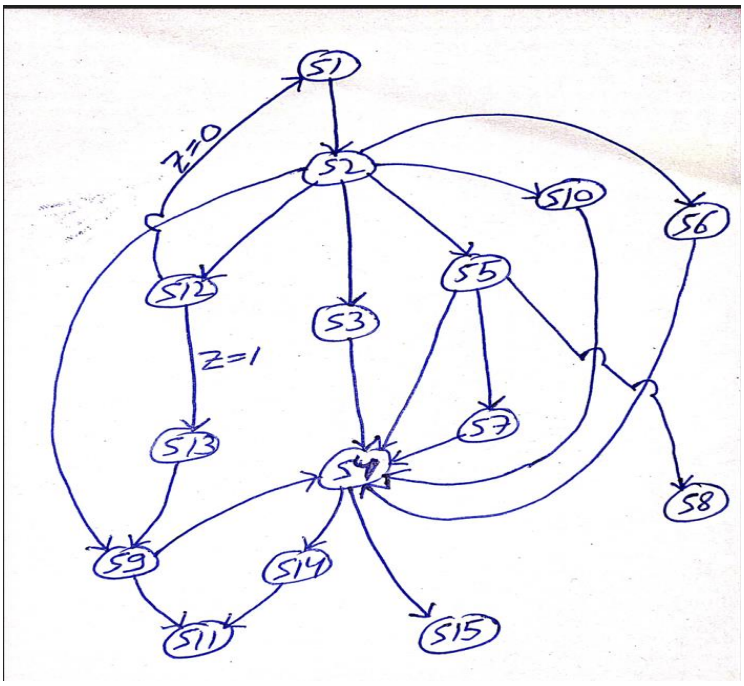
- Did state minimisation
- Designed code for memory
- Figures out control pins signal for all the states
- Debug and test last 4 functions of ALU
- Tested of all internal signals for last 2 instructions
- Prepared final report

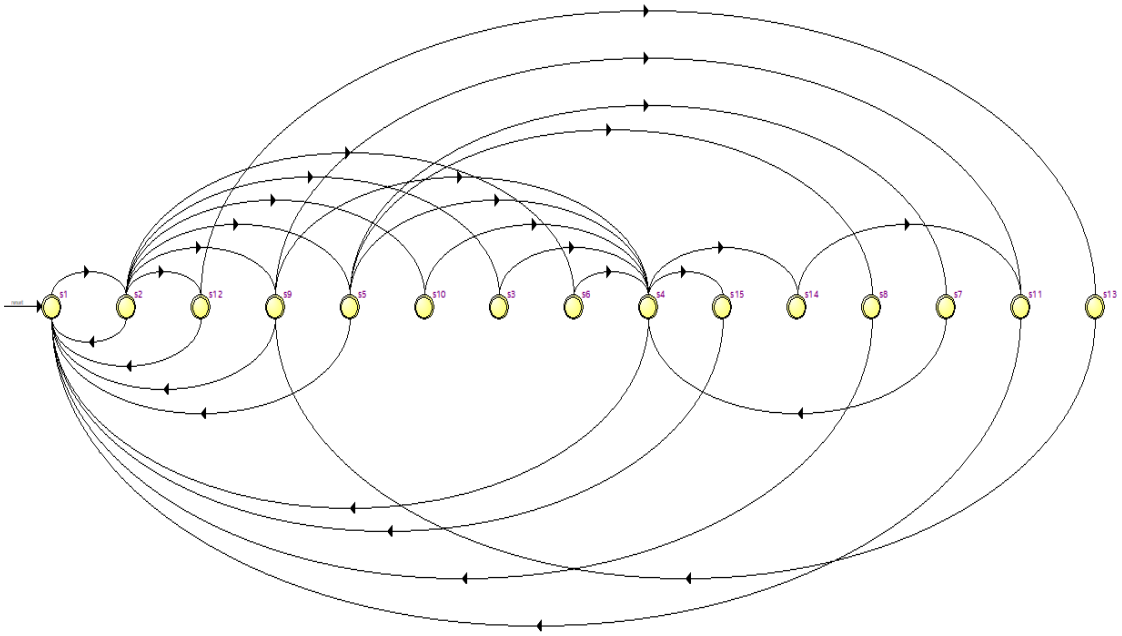
Testing of Code on Board

Changes Made for FPGA Implementation

1. Modified the PC to be first stored in a temp register to use later and whenever in the same instruction PC is called then this temp is used
2. Changed the Reset logic separate for Memory and Register File whenever Reset is one then the register file is wiped
   And reset_memory is used to reset the values in the Memory
3. Made a Final: Process to choose the output to show as per the input of the user and mapped it with the FPGA Board Pins
   Used the 8 pins to select what to show and which address to show
   Also reduced the size of Memory to 31 as lesser number of pins available
   The first 3 pins are used to select Register File or Memory or Clock or Carry or Zero value and higher 8 bits or lower 8 bits

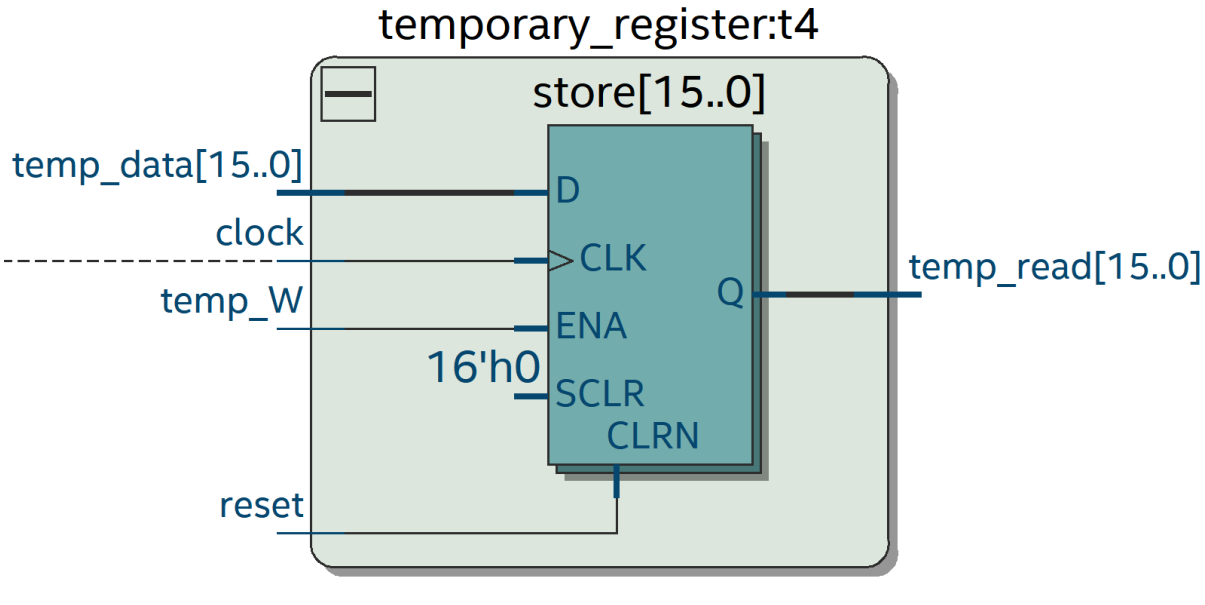
FSM


S1
Z=0
S2
S10
S6
S12
S5
S3
Z=1
S13
S7
S4
S8
S9
S14
S11
S15

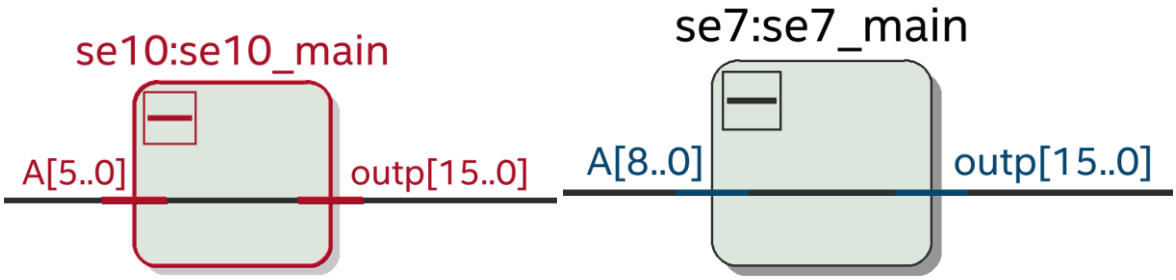

S1
S2
S12
S9
S5
S10
S3
S6
S4
S15
S14
S8
S7
S11
S13

COMPONENTS

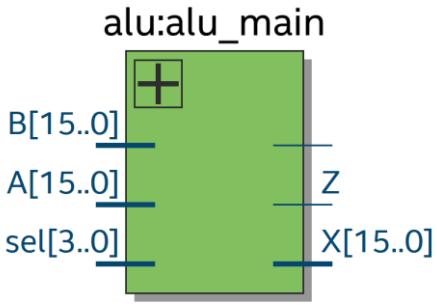
TEMPORARY REGISTER



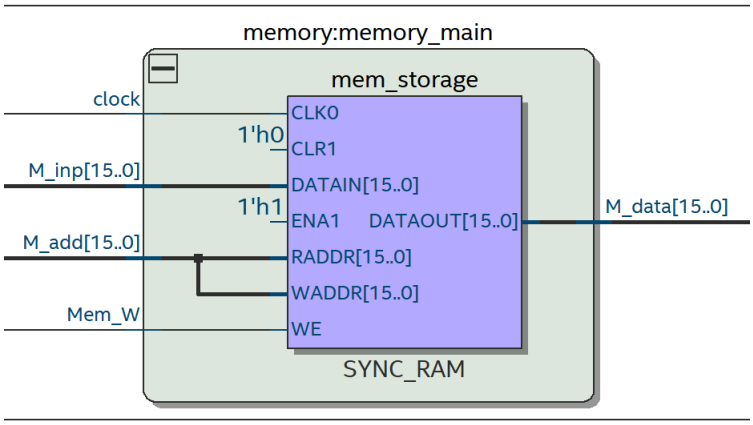
SIGN EXTENDER



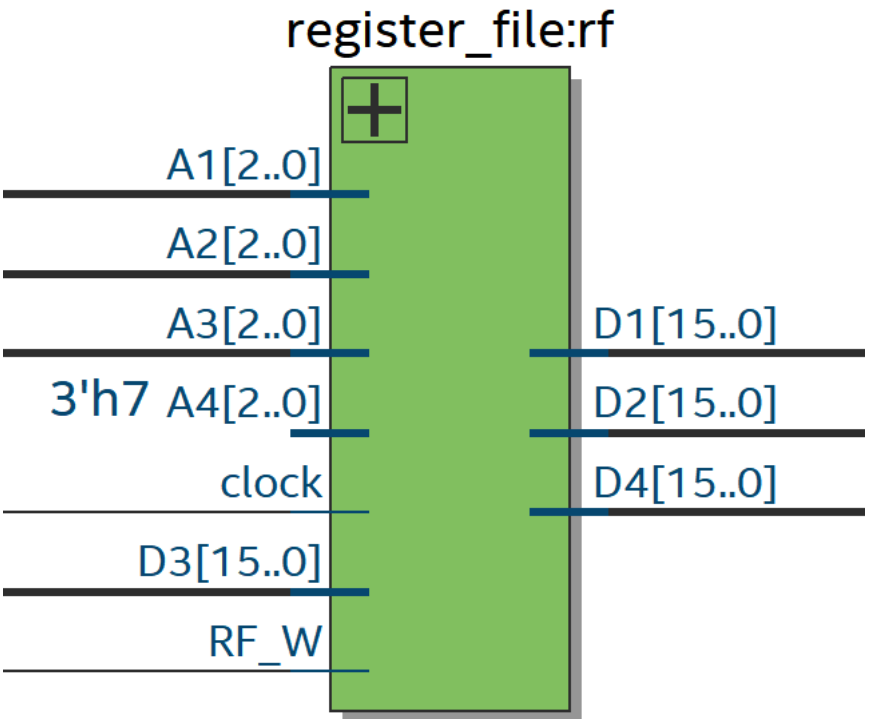
ALU



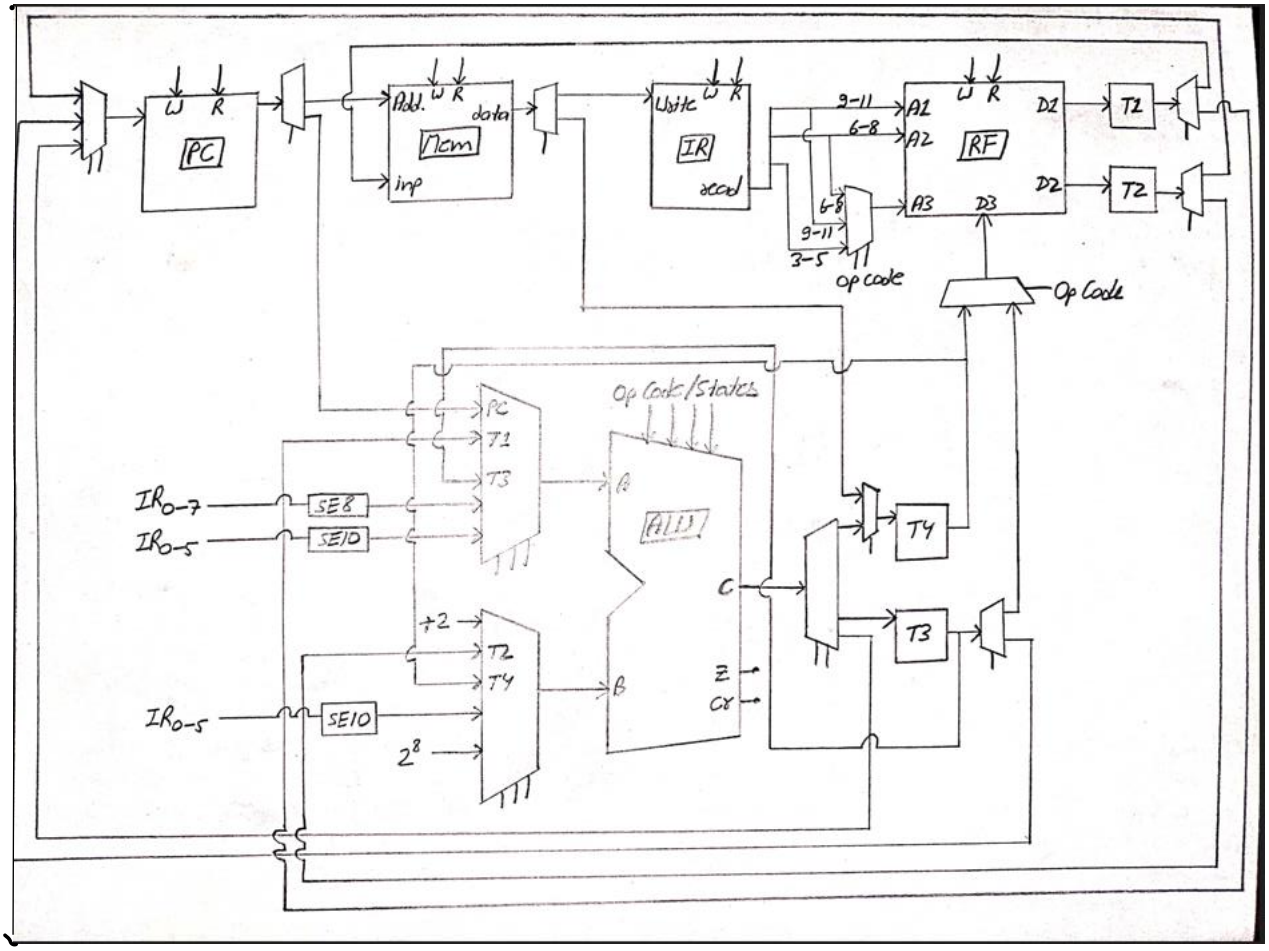
MEMORY



REGISTER FILE



DATAPATH


W
R
PC
Add.
data
Mem
inp
Write
IR
addr
9-11
6-8
3-5
A1
A2
A3
RF
D1
D2
T2
op code
Op Code
Op Code/States
PC
T1
T3
ALU
A
B
C
Z
CY
T4
T3
+2
T2
T4
2^8
IR0-7
IR0-5
SE8
SE10

STATE DESCRIPTION

S1

- **MEMORY FETCH AND PROGRAM COUNTER UPDATE AND STATE TRANSITION.**

```
when s1=>

        A4<="111";--Point at R7.
        M_add<=D4;
        IR_in<=M_data;
        --Fetch the Memory Data

        IR_w<='1';
        alu_A<=D4;
        alu_B<="0000000000000001";
                        sel<="0000";
-- Update R7
        D3<=alu_C;
        A3<="111";
                                                --Always go to state 2 always
                                                state_next<=s2;
```

S2

- **REGISTER FILE WRITING , DATA FETCH , REGISTER WRITING AND STATE TRANSITION.**

```
when s2=>
        RF_w<='1';
        --The A3 and D3 is ready RF_w writes it in R7
        A1<=IR_op(11 downto 9);
        A2<=IR_op(8 downto 6);--signal
        --Fetch data in T1 and T2
        T1_in<=D1;
        T2_in<=D2;
        T1_W<='1';
        T2_W<='1';
                                        -- On the basis of op code in each of the corresponding code
                                    case IR_op(15 downto 12) is
                                         when "1101" =>  state_next <= s9;
                                         when "1000" =>  state_next <= s6;
                                         when "1001" =>  state_next <= s10;
                                         when "1110" =>  state_next <= s1;
                                         when "0111" =>  state_next <= s1;
                                         when "1011" => state_next <= s5;
                                         when "0001" => state_next <= s5;
                                         when "1010" => state_next <= s5;
                                         when "1111" => state_next <= s9;
                                         when "1100" => state_next <= s12;
                                         when others => state_next <= s3;
                                     end case;
```

S3

- **ALU ARITHMETIC OPERATION REGISTER WRITING AND STATE TRANSITION.**

```
when s3=>


      alu_A<=T1_op;
       alu_B<=T2_op;
       sel<=IR_op(15 downto 12);
       T3_in<=alu_C;
       T3_W<='1';

       -- Use the ALU to perform the operation involved
                                       state_next<=s4;
                                       --Always go to s4 to update the register involved
```

S4

- **STATE TRANSITION AND RESULT UPDATION (REGISTER FILE WRITING).**

```
                if (IR_op( 15 downto 12)="1111") then
                        A3<=IR_op(11 downto 9);
                        D3<=T3_op;

                elsif (IR_op( 15 downto 12)="1101") then
                        A3<=IR_op(11 downto 9);
                        D3<=T3_op;


                elsif (IR_op( 15 downto 12)="0001") then
                        A3<=IR_op(8 downto 6);
                        D3<=T3_op;


                elsif (IR_op( 15 downto 12)="1000") then
                        A3<=IR_op(11 downto 9);
                        D3<=T3_op;


                elsif (IR_op( 15 downto 12)="1001") then
                        A3<=IR_op(11 downto 9);
                       D3<=T3_op;

                elsif (IR_op( 15 downto 12)="1010") then
                        A3<=IR_op(11 downto 9);
                        D3<=T4_op;
                else
                    A3<=IR_op(5 downto 3);
                    D3<=T3_op;
                end if;
                --Use the OP_code to get the Address to be updated

 Rf_W<='1';
 --Turn Rf_w is 1
                                --Now based on op-code decide the next state

                            case IR_op(15 downto 12) is
                                when "1111" => state_next <= s15;
                                when "1101" => state_next <= s14;
                                when others => state_next <= s1;
                              end case;
```

S5

- **DATA LOADING , REGISTER WRITING , ALU OPERATION AND STATE TRANSITION.**

```
when s5=>

       if (IR_op(15 downto 12)="0001") then
       alu_A<=T1_op;
       elsif (IR_op(15 downto 12)="1010") then
       alu_A<=T2_op;
       elsif (IR_op(15 downto 12)="1011") then
       alu_A<=T2_op;
       else
         NUll;
       end if;
       -- To load Data from regA or regB based on the instruction
       se10_in<=IR_op(5 downto 0);
       alu_B<=se10_out;
       sel<="0000";
       T3_in<=alu_C;
       T3_w<='1';
       -- Add it with sign extended IMM
                                    --move to the address as per IR_op
                                   case IR_op(15 downto 12) is
                                        when "0001" => state_next <= s4;
                                        when "1010" => state_next <= s7;
                                        when "1011" => state_next <= s8;
                                        when others => state_next <= s1;
                                        end case;
```

S6

- **SIGN EXTENDING , SHIFTING DUE TO INTSTRUCTION(REGISTER WRITING) AND STATE TRANSITION.**

```
when s6=>

       se7_in<='0'&IR_op(7 downto 0);
       alu_A<=se7_out;
       alu_B<="0000000100000000";
       sel<="0011";
       T3_in<=alu_C;
       T3_w<='1';

       --To shift we by 8 bits multiply by 2^8 using the select 0011 to Multiply
                                    --move to s4 always from this state
                                    state_next<=s4;
```

S7

- **REGISTER WRITING AND STATE TRANSITION.**

```
when s7=>

        M_add<=T3_op;
        T4_in<=M_data;
        T4_w<='1';
         --To load the data from the given regB+IMM*2

                                        --move to next state s4 always
                                        state_next<=s4;
```

S8

- **MEMORY WRITING AND STATE TRANSITION.**

```
when s8=>
        M_add<=T3_op;
        M_inp<=T1_op;
        Mem_W<='1';
        --To store to the address at T3_op
        --and for input use T1_op which has the data of regA
                                        --move to s1 always
                                        state_next<=s1;
```

S9

- **ALU OPERATION , REGISTER WRITING AND STATE TRANSITION.**

```
when s9=>

       A4<="111";
       alu_A<=D4;
       alu_B<="0000000000000001";
       sel<="0010";
       T3_in<=alu_C;
       T3_w<='1';
       -- Do PC-1 to take the updated PC one step back
                                -- On the basis of Op-code move to required state
                                  case IR_op(15 downto 12) is
                                   when "1111" => state_next <= s4;
                                   when "1101" => state_next <= s4;

                                   when "1100" => state_next <= s11;
                                   when others => state_next <= s1;
                                  end case;
```

S10

- **REGISTER WRITING , SIGN EXTENDING AND STATE TRANSITION.**

```
when s10=>
       se7_in<='0'&IR_op( 7 downto 0);
       T3_in<=se7_out;
       T3_w<='1';

       --to sign extend and store in a register to be used in next state
                                --move to s4 always from this state
                                state_next<=s4;
```

S11

- **ALU OPERATION , REGISTER FILE WRITING AND STATE TRANSITION.**

```
when s11=>
        alu_A<=T3_op;
        alu_B<=t4_op;
        sel<="0000";
        A3<="111";
        D3<=alu_C;
        RF_w<='1';
        -- To do PC+IMM*2
                                        --Go back to S1
                                        state_next<=s1;
```

S12

- **ALU OPERATION , ZERO FLAG CHECK AND STATE TRANSITION.**

```
when s12=>

        alu_A<=T1_op;
        alu_B<=T2_op;
        sel<="0010";
        -- For BEQ to subtract T1 and T2
                                --Based on op_code and Zero Flag='1' move to the state needed
                                case IR_op(15 downto 12) is
                                      when "1100" =>
                                         if Zero='1' then
                                           state_next<=s13;
                                         else
                                           state_next<=s1;
                                         end if;
                                         when others => state_next <= s1;
                                      end case;
```

S13

- **ALU OPERATION , SIGN EXTENDING , REGISTER WRITING AND STATE TRANSITION.**

```
when s13=>
        se10_in<=IR_op(5 downto 0);
        alu_A<=se10_out;

        alu_B<="0000000000000010";
        sel<="0011";
        --To do IMM*2 use the select as multiply and store in T3
        T4_in<=alu_C;
        T4_w<='1';
                                    --move to s9 always from s13
                                    state_next<=s9;
```

S14

- **ALU OPERATION , SIGN EXTENDING , REGISTER WRITING AND STATE TRANSITION.**

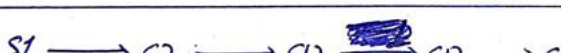
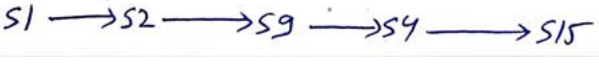
```
when s14=>

        se7_in<=IR_op(8 downto 0);
        alu_A<=se7_out;
        alu_B<="0000000000000010";
        sel<="0011";

        T4_in<=alu_C;
        T4_w<='1';
        --  To do IMM*2 here IMM is 9 bits
                                        --move to s11 always from s14
                                        state_next<=s11;
```

S15

- **REGISTER FILE WRITING AND STATE TRANSITION.**

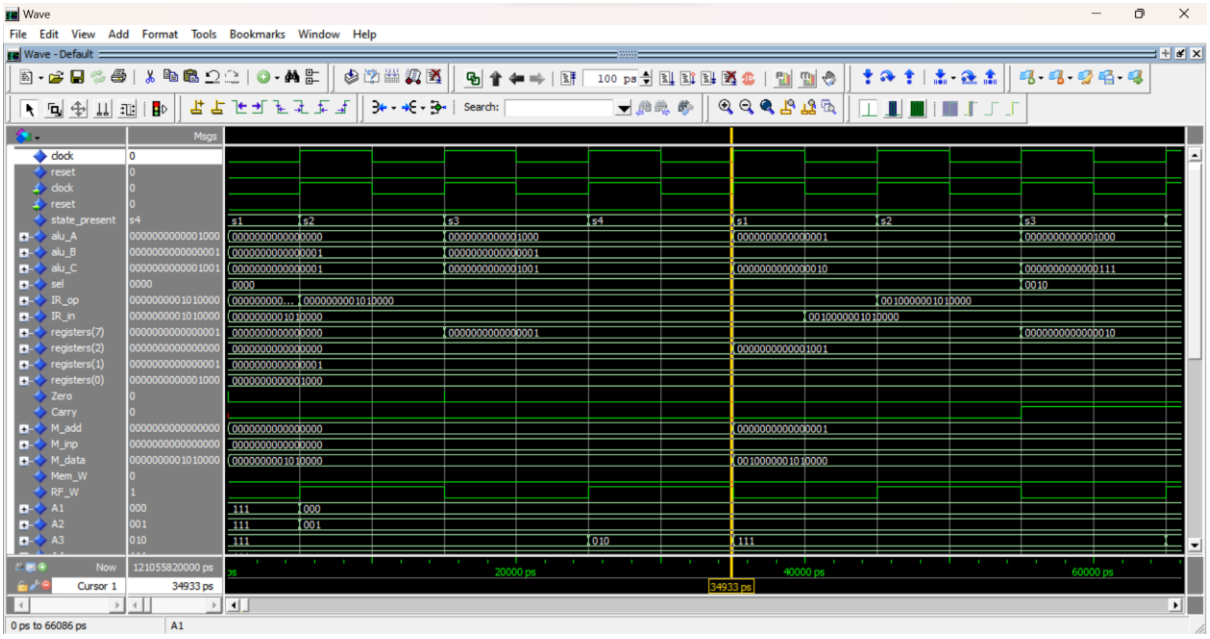
```
when s15=>

        A3<="111";
        D3<=T2_op;
        RF_w<='1';
        --TO branch to the data at JLR
                                                --move to s1 always.
                                                state_next<=s1;
```

INSTRUCTIONS

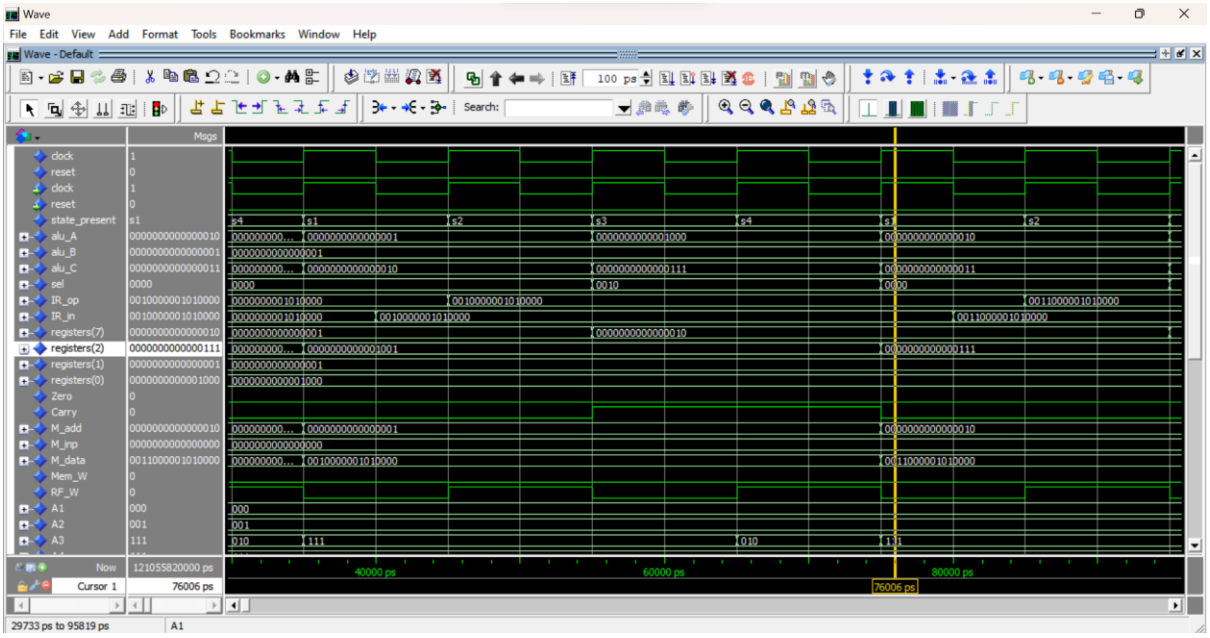
ADD , AND SUB , ORA MUL, SMP	
ADI	
LHI	
LLI	
SW	
LW	
BEQ	
JAL	
JLR	

SIMULATIONS

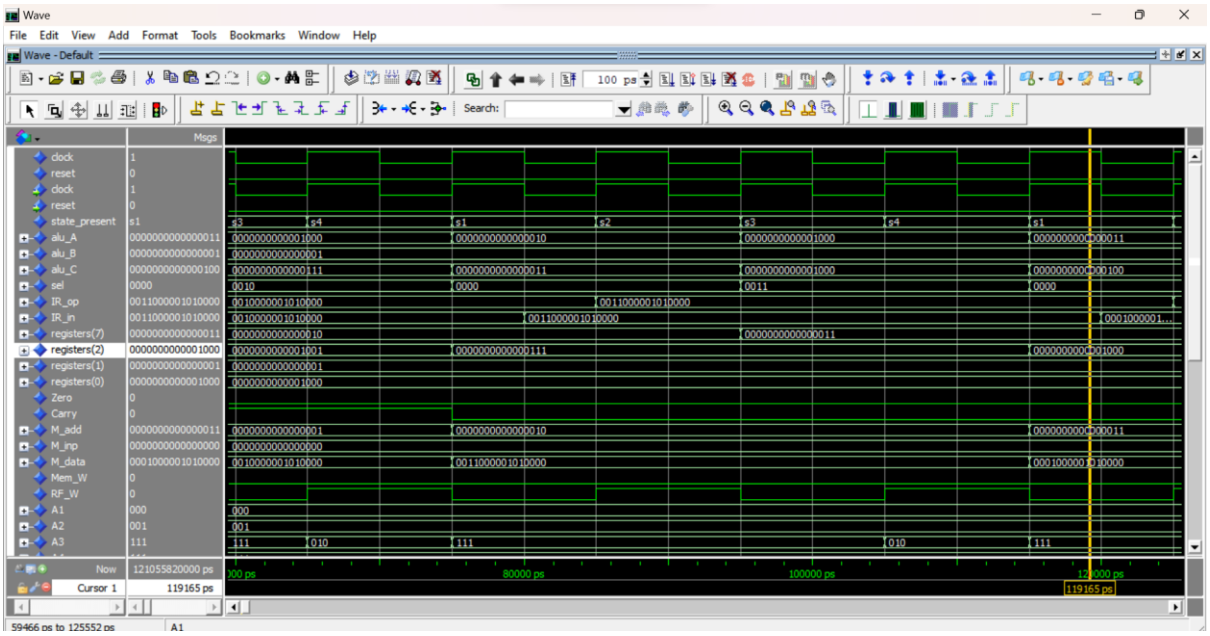
ADD: 00_00 RA RB RC 0 00



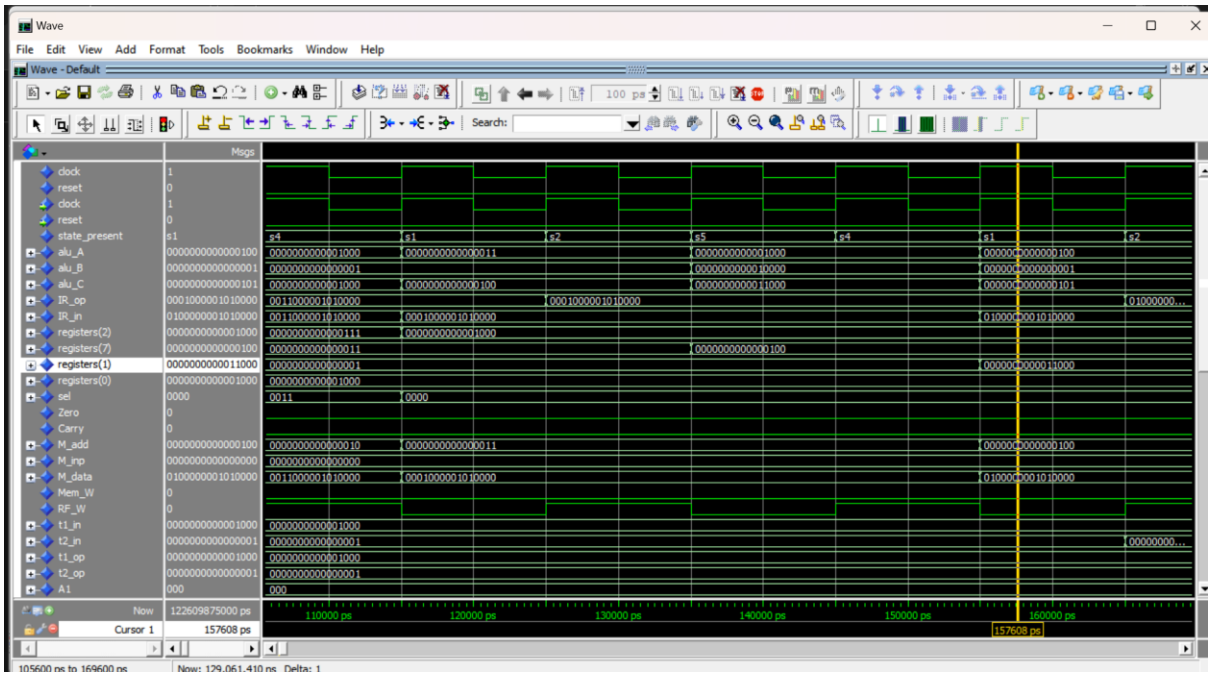
SUB: 00_10 RA RB RC 0 00



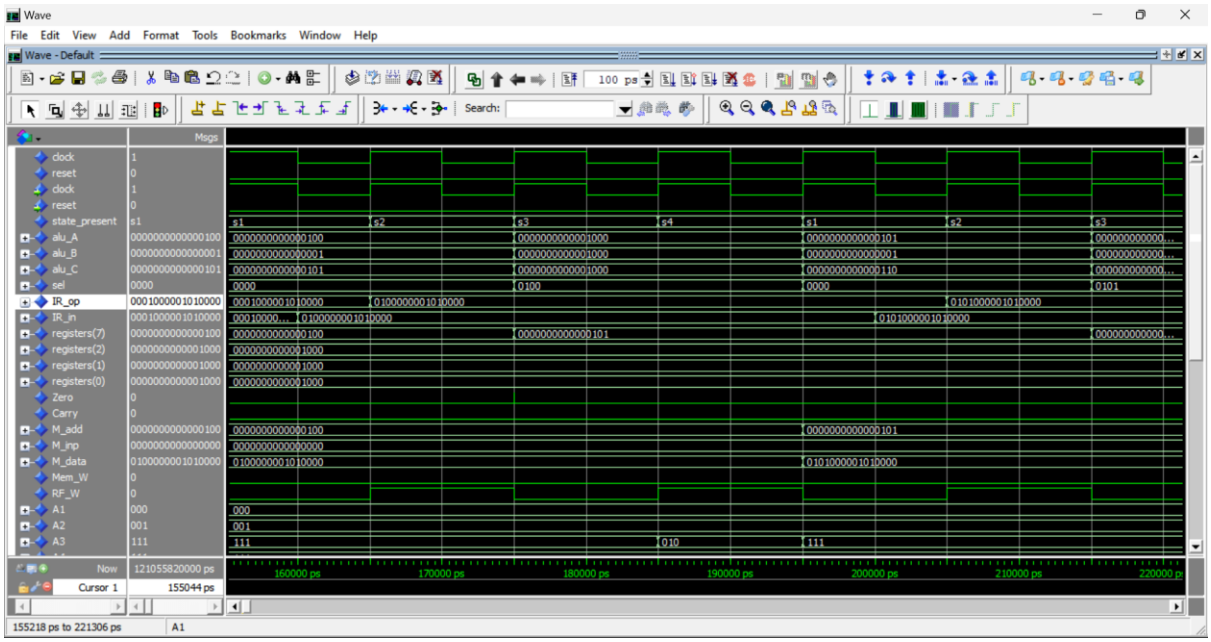
MUL: 00_11 RA RB RC 0 00



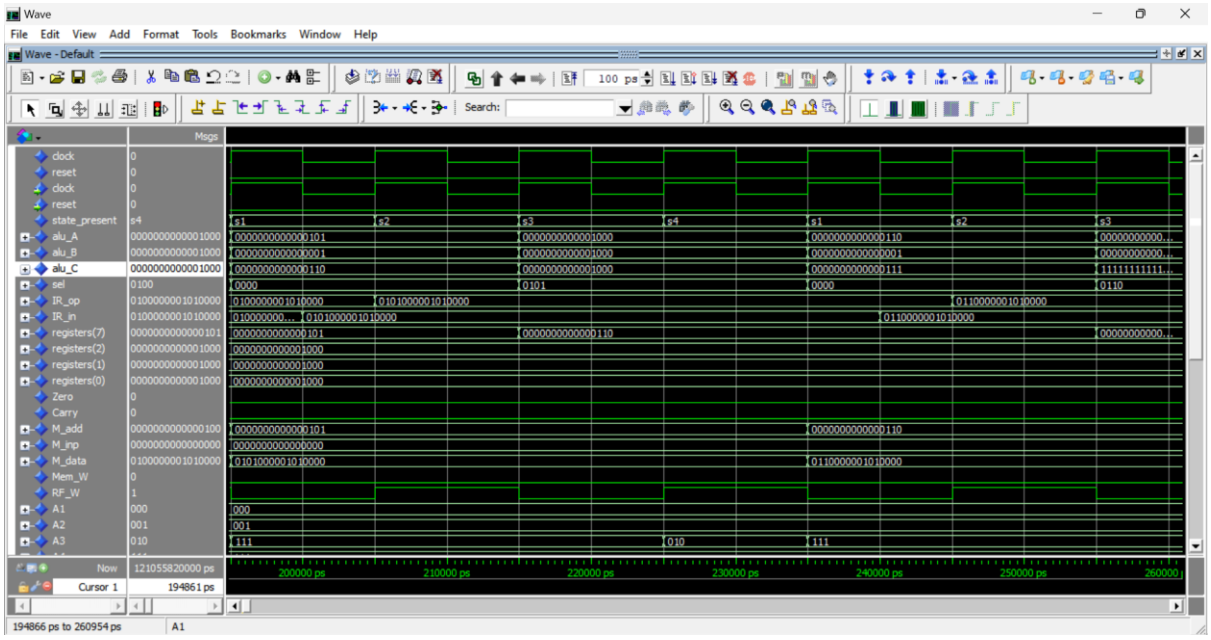
ADI: 00_01 RA RB 6 bit Immediate



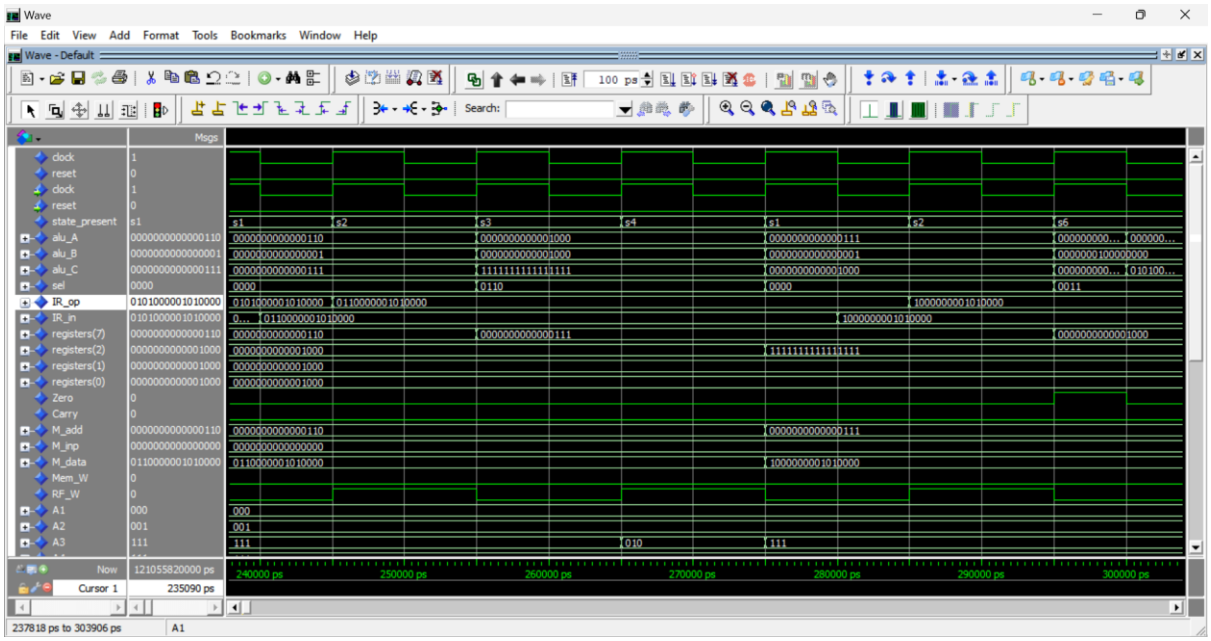
AND: 01_00 RA RB RC 0 00



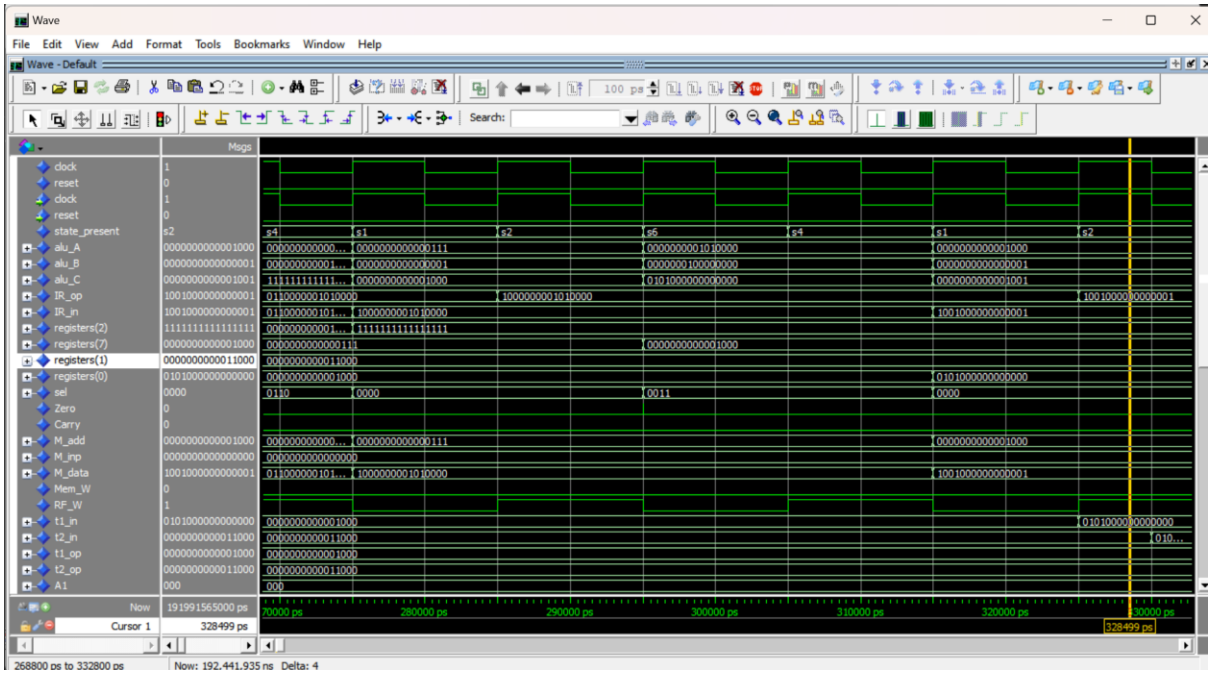
ORA: 01_01 RA RB RC 0 00



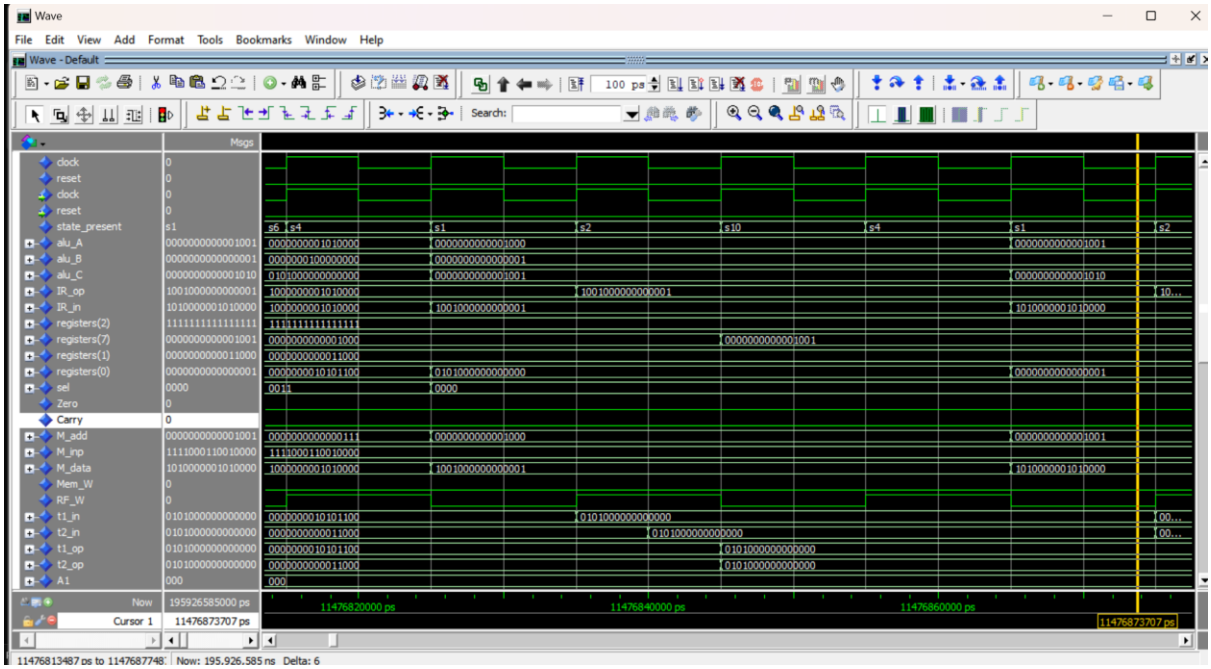
IMP: 01_10 RA RB RC 0 00



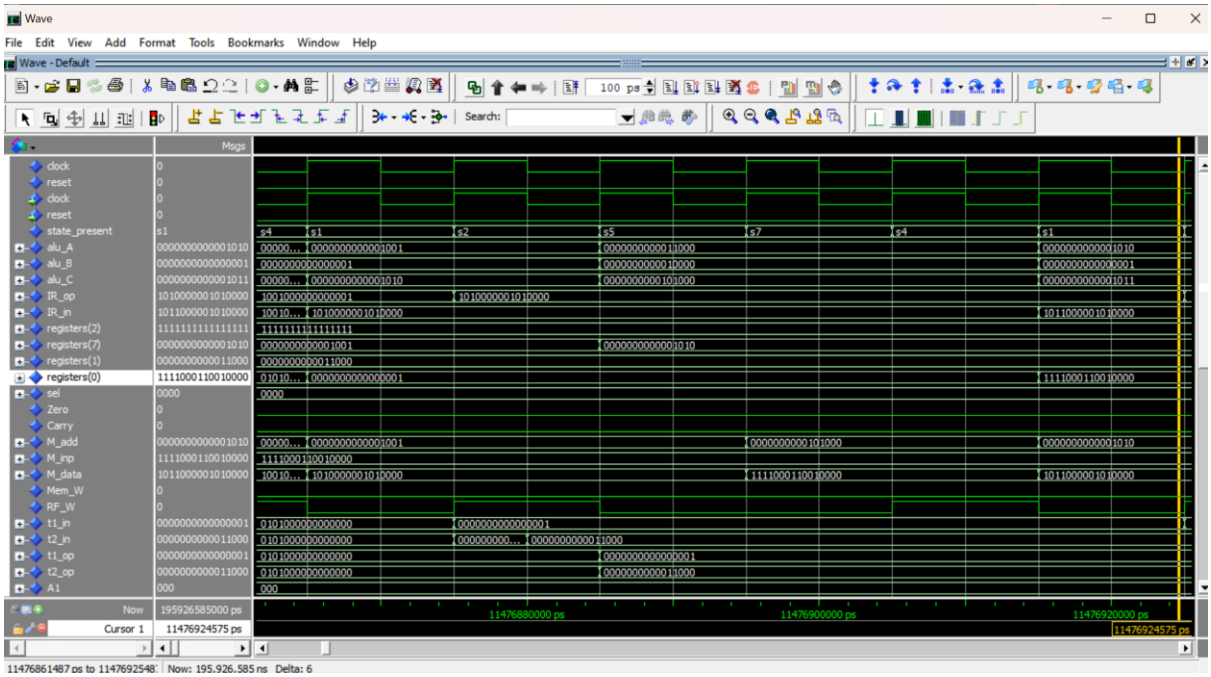
LHI: 10_00 RA 0 + 8 bit Immediate



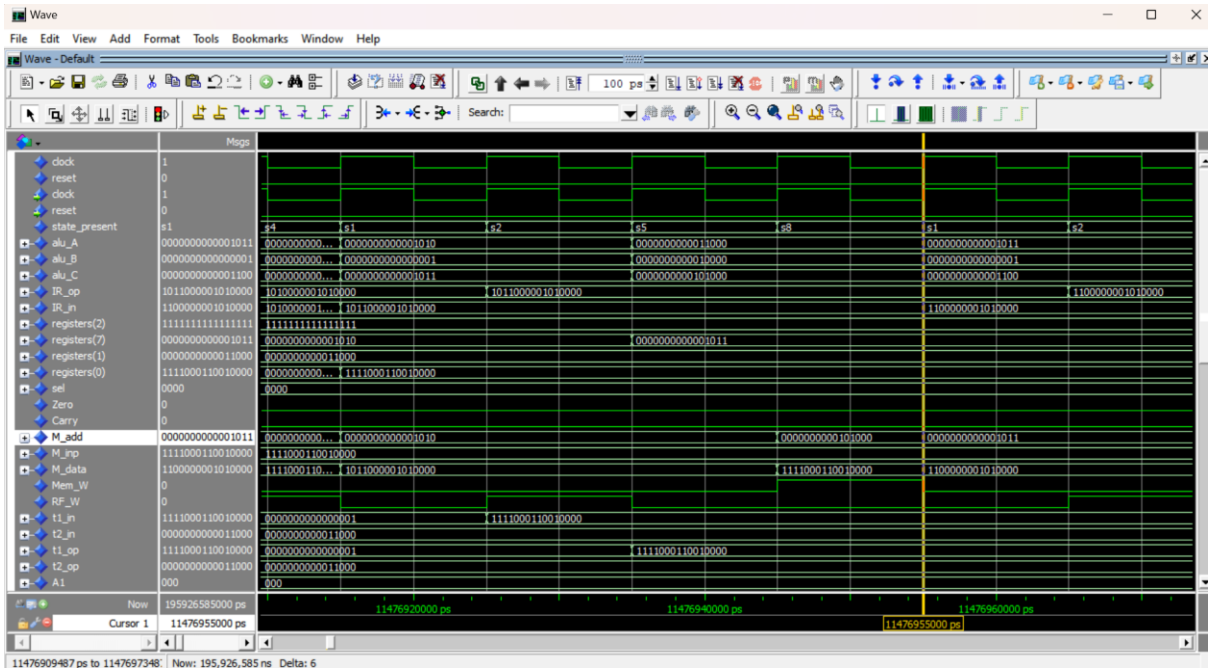
LLI: 10_01 RA 0 + 8 bit Immediate



LW: 10_10 RA RB 6 bit Immediate

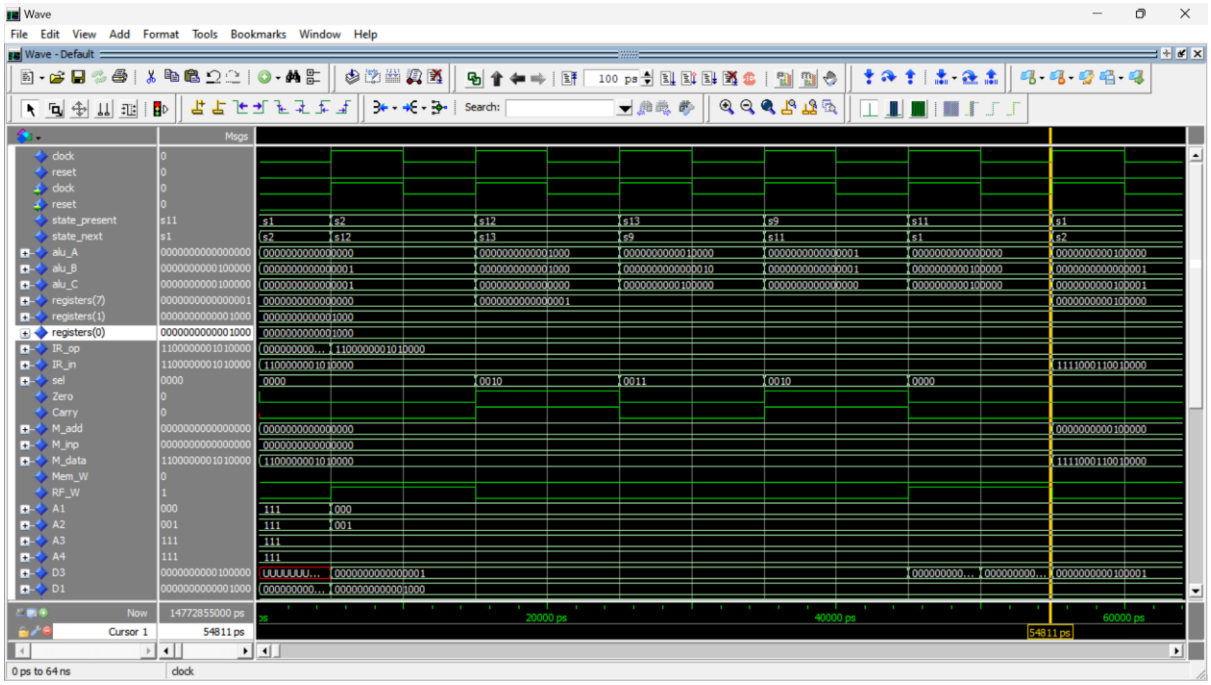


SW: 10_11 RA RB 6 bit Immediate

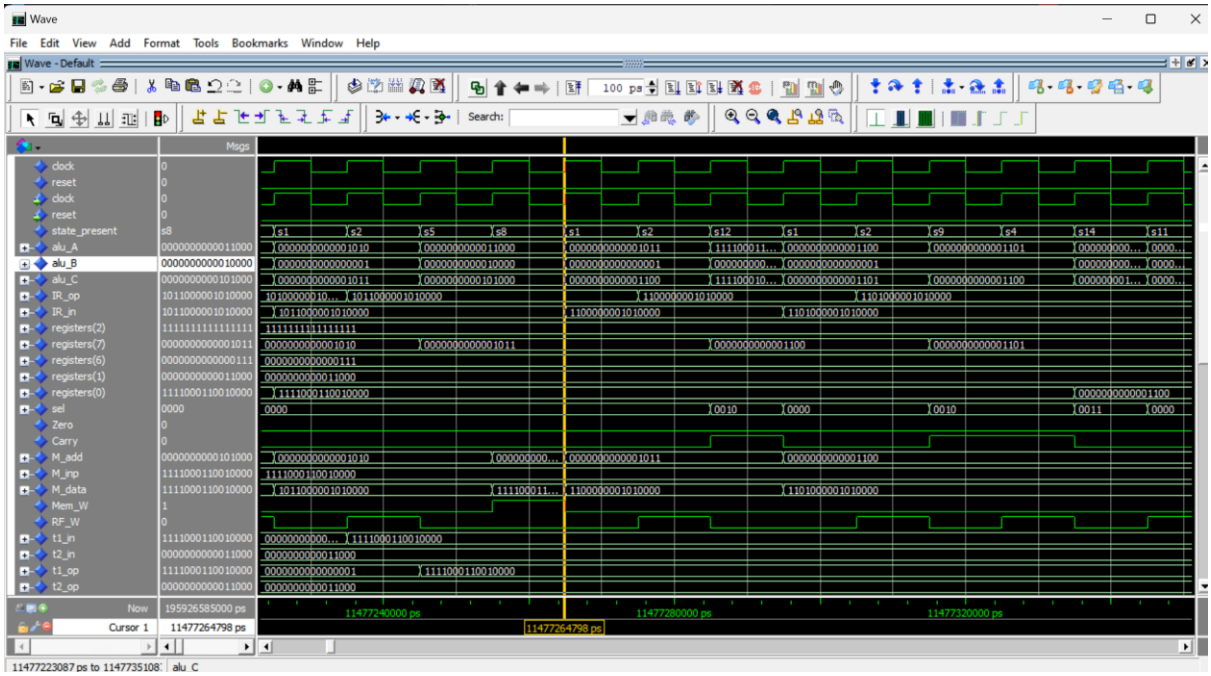


BEQ: 11_00 RA RB 6 bit Immediate

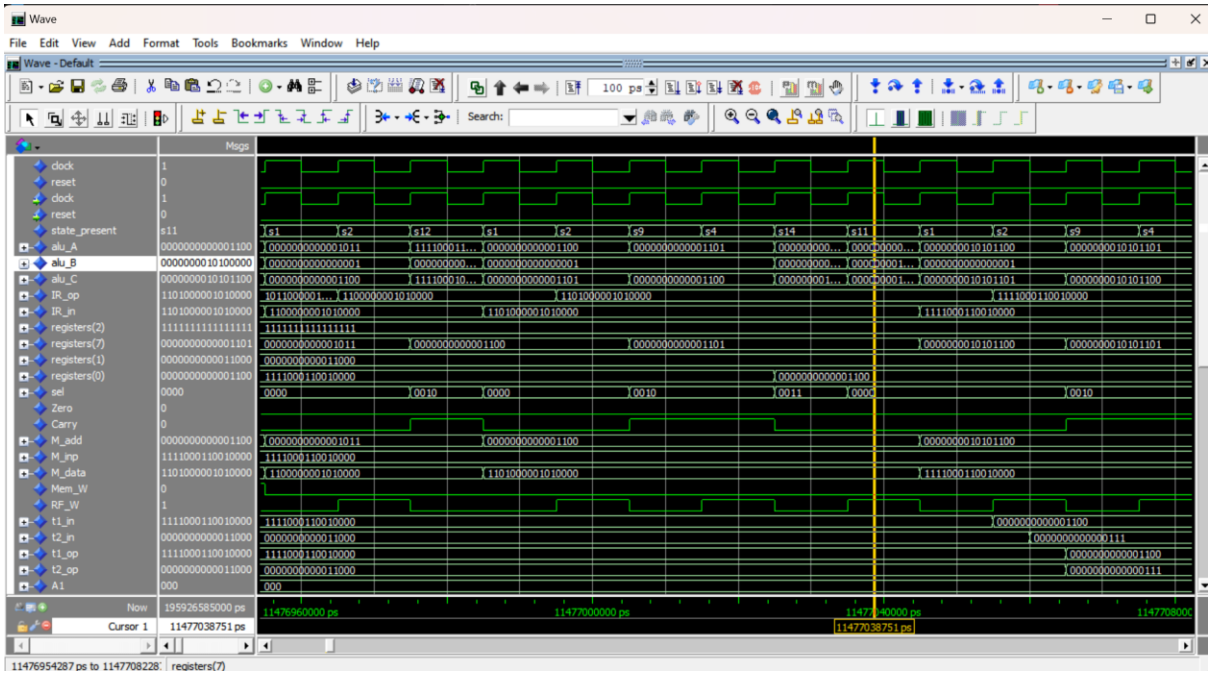
BEQ if Ra and Rb Equal



BEQ if Ra and Rb not Equal



JAL: 11_01 RA 9 bit Immediate offset



JLR: 11_11 RA RB 000_000