MILESTONE 1 : SINGLE CYCLE

Name	ID
Islam Mohamed Sayed Ali	*21P0151*
Kareem Ahmed Mohamed Shawky	*21P0075*
Ahmed Mostafa Gomaa Mahmoud	*21P0407*

CONTROL SIGNALS

Instructions	RegDst	Regwrite	Extd	ALUsrc	memRead	memwrite	memtoreg	stw	J	branch	Jmem
R-type	1	1	0	0	0	0	0	0	0	0	0
ADDI	0	1	1	1	0	0	0	0	0	0	0
ANDI	0	1	0	1	0	0	0	0	0	0	0
lw	0	1	1	1	1	0	1	1	0	0	0
sw	0	0	1	1	0	1	0	0	0	0	0
J	0	0	0	0	0	0	0	0	1	0	0
beq	0	0	1	0	0	0	0	0	0	1	0
Jmem	0	0	1	0	1	1	0	0	0	0	1
stw	0	1	1	0	0	1	0	1	0	0	0

*we used 0 to represent don't care values

REGISTER DECLARATION

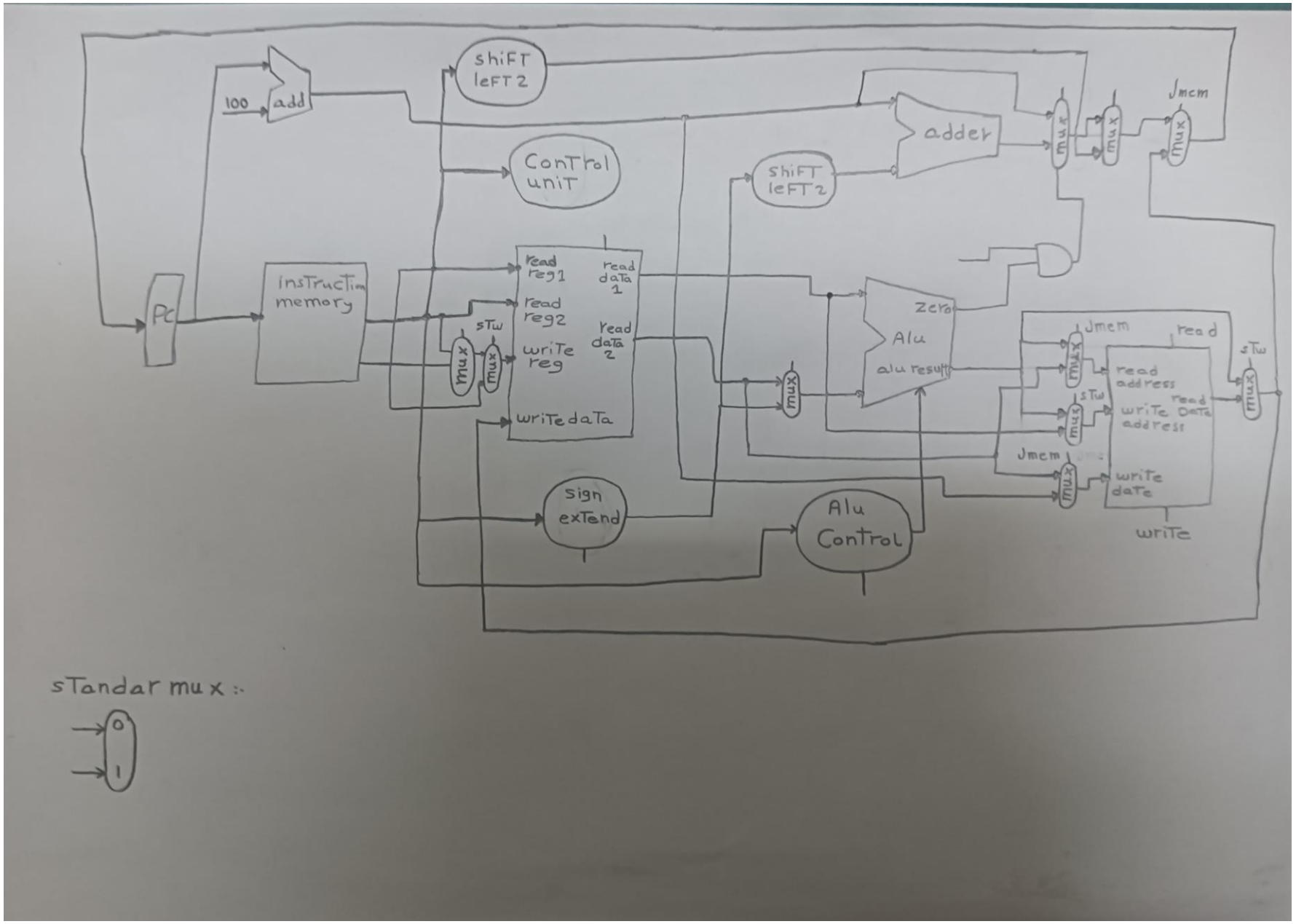
rs		rt		rd	
000	Add	001	Add	010	Add
001	Sub	111	Sub	110	Sub
010	And	100	And	101	And
011	Or	000	Or	010	Or
100	Slt	111	Slt	001	Slt
101	Addi	100	Addi		
111	Andi	001	Andi		
000	Stw	011	Stw		
110	Lw	010	Lw		
010	Sw	011	Sw		
100	Beq	101	Beq		
101	Jmem	101	Jmem		

INSTRUCTION VALUES

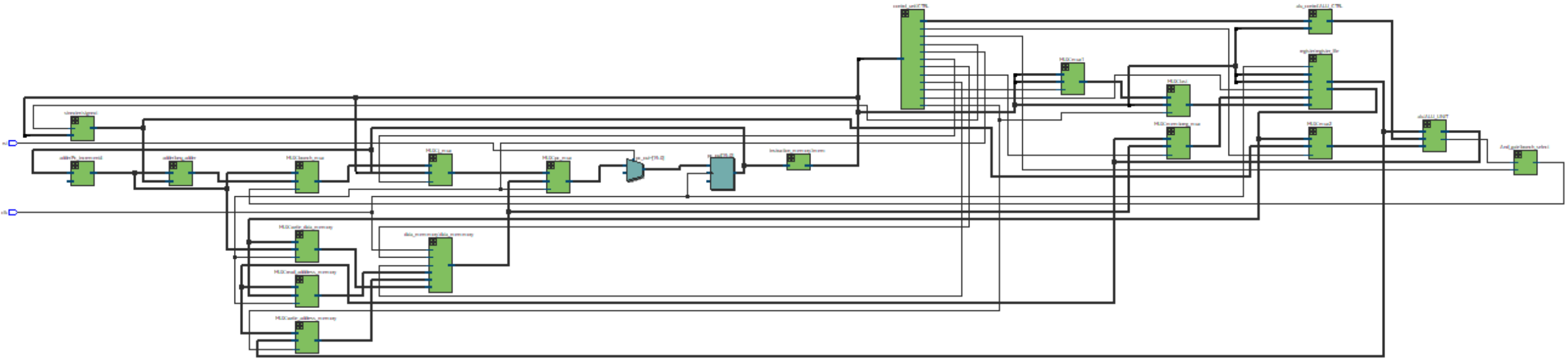
R-type	OP(4)	Rs(3)	Rt(3)	Rd(3)	Shamt(3)	Func(4)
Add	0000	000	001	010	000*	0001
Sub	0000	001	111	110	000*	0010
And	0000	010	100	101	000*	0011
Or	0000	011	100	010	000*	0100
Slt	0000	100	111	001	000*	0101
I-type	**OP(4)**	**Rs(3)**	**Rt(3)**	**Immediate(10)**		
Addi	0001	101	100	00_0000_1010		
Andi	0010	111	001	00_0000_0111		
Stw	0011	000	011	00_0000_1011		
Lw	0100	110	010	00_0000_0011		
Sw	0101	010	011	00_0000_0101		
Beq	0110	100	101	00_0000_0001		
J-type	**OP(4)**	**Rs(3)**	**Address(13)**			
Jump	0111	000*	0_0000_0000_0000			
New item	**OP(4)**	**Rs(3)**	**Rt(3)**	**Immediate(10)**		
Jmem	1000	011	001	00_0000_1110		

Single Cycle

-MIPS

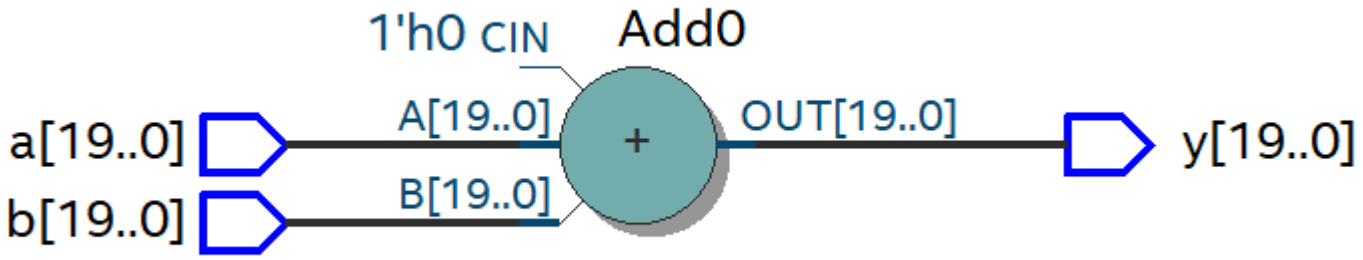

100
add
shift left 2
Control unit
shift left 2
adder
mux
mux
Jmem
mux
Pc
instruction memory
read reg1
read reg2
write reg
write data
read data 1
read data 2
sTw
mux
mux
zero
Alu
alu result
mux
Jmem
read
read address
write address
read Data
write data
sTw
mux
sTw
mux
Jmem
mux
mux
write
sign extend
Alu Control
sTandar mux :-
0
1

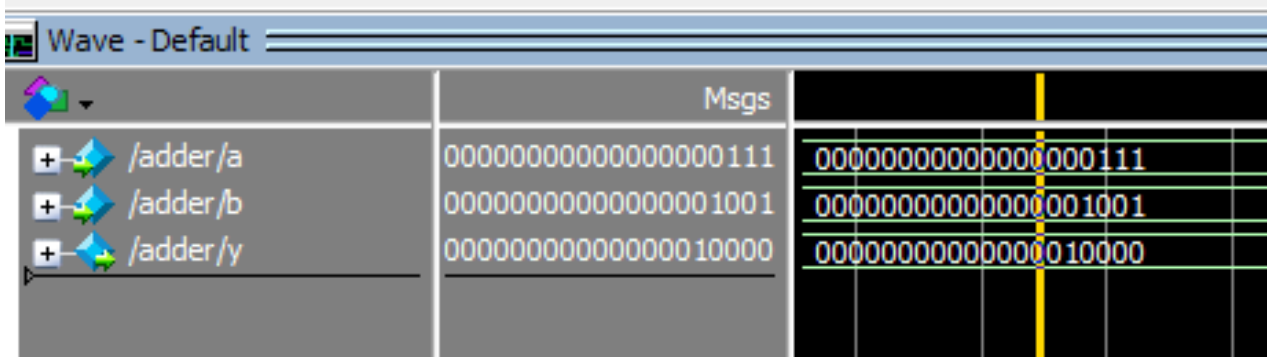
RTL SCHEMATIC ANALYSIS



1- ADDER

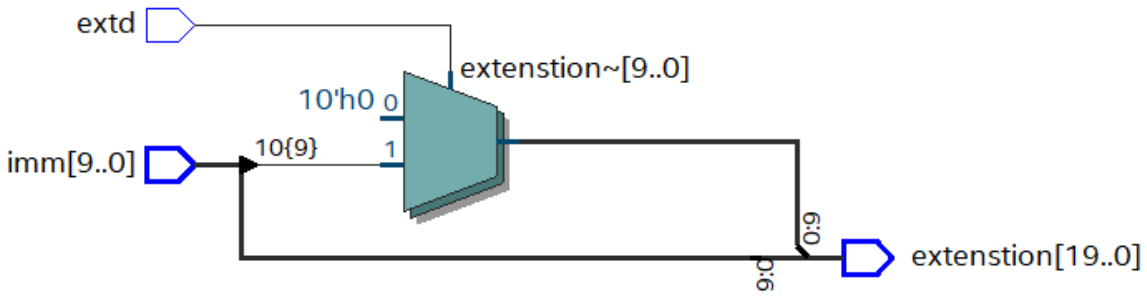
```
1   module adder ( a , b , y) ;
2
3   input [19:0] a ;
4   input [19:0] b ;
5   output reg [19:0] y ;
6
7   always @(*)
8   begin
9
10  y <= a + b ;
11
12  end
13  endmodule
14
15
```

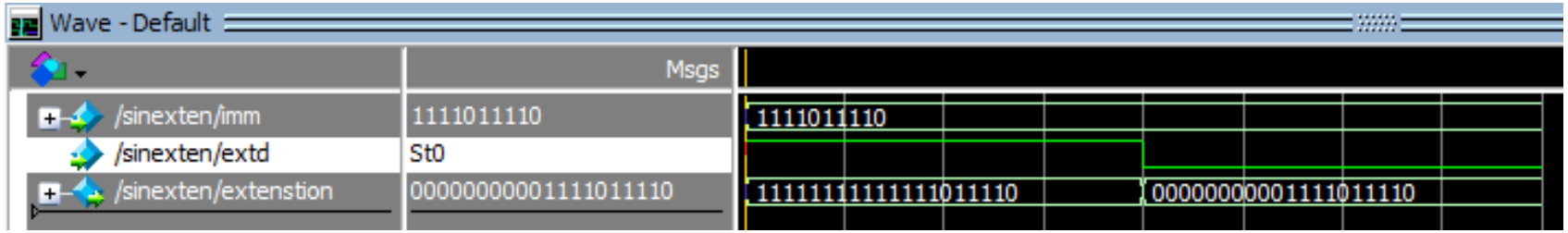




2- SIGN EXTENSION

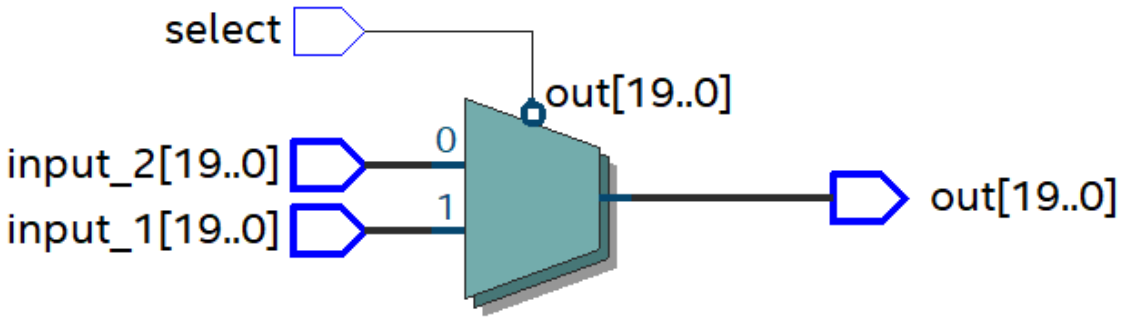
```
Ln#
1   module sinexten (extd , imm , extenstion);
2
3   input [9:0] imm ; // 10 bit immediate
4   input extd ;
5   output reg [19:0] extenstion ; // 19 bit extension value
6
7   always @ (*) begin
8
9   if (extd ==1)
10
11  begin
12
13  extenstion <= { {10{ imm[9]}}, imm } ;
14
15  end
16  else
17
18  begin
19  extenstion <= {  10 'b 0000000000 , imm } ;
20  // () is a replicator brackets to replicate most significant bit of immediate
21
22  end
23  end
24  endmodule
25
```

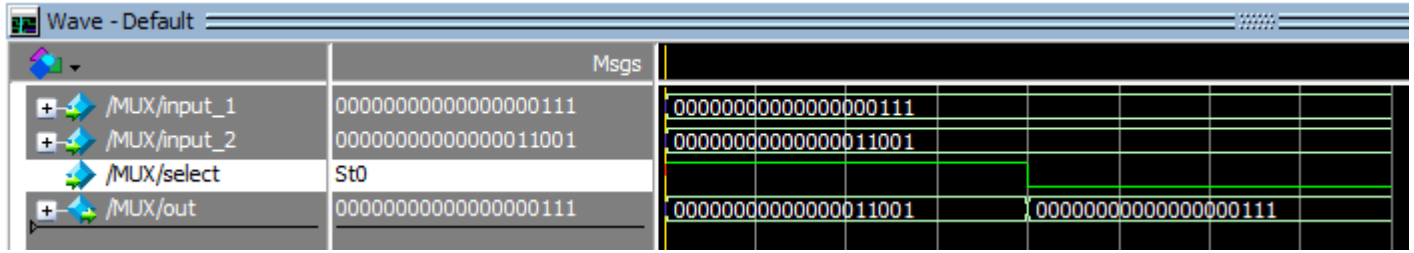




3- MUX

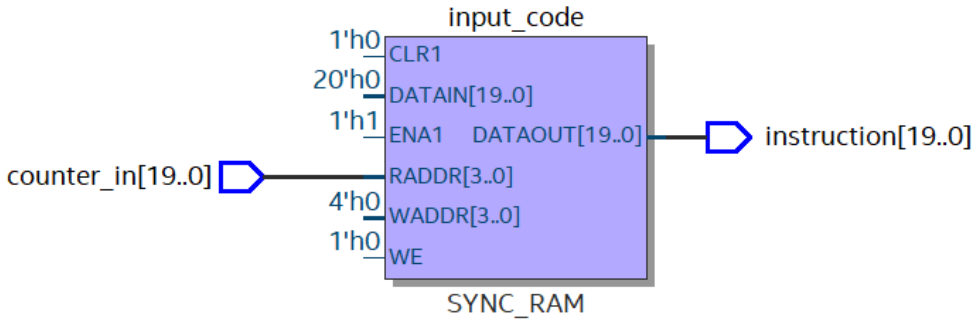
```
Ln#
1   module MUX (  input_1 , input_2 , select ,  out ) ;
2     input [19:0] input_1 ;
3   input [19:0] input_2 ;
4   input select ;
5     output reg [19:0] out ;
6
7    always @(*) begin
8   if (select == 0 )
9   begin
10  out = input_1 ;
11  end
12  else if ( select == 1 )
13  begin
14
15  out = input_2 ;
16
17  end
18  end
19  endmodule
20
```

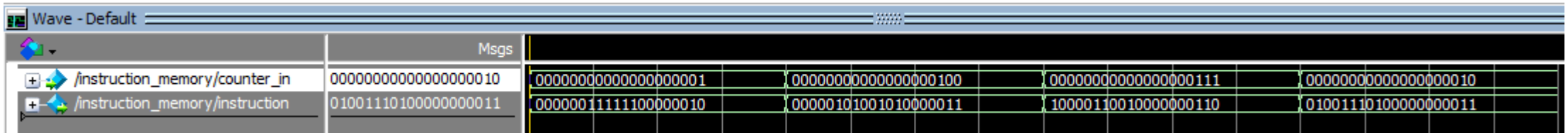




4-INSTRUCTION MEMORY

```
1  module instruction_memory (counter_in ,instruction ) ;
2    input [19:0] counter_in ;
3    output reg [ 19 : 0 ] instruction ;
4    reg [ 19 : 0 ] input_code [ 0 : 14 ] ;
5    initial
6    begin
7
8      input_code [0]  =  20'b 0001_101__100_00_0000_1010; //addi imm value 10
9      input_code [1]  =  20'b 0000_001_111_110_000_0010 ; //sub
10     input_code [2]  =  20'b 0100_111_010_00_0000_0011 ; //lw imm value 3
11     input_code [3]  =  20'b 0000_100_111_001_000_0101 ; //slt
12     input_code [4]  =  20'b 0000_010_100_101_000_0011 ; //and
13     input_code [5]  =  20'b 0000_000_001_010_000_0001 ; //add
14     input_code [6]  =  20'b 0011_000_011_00_0000_1010 ; //stw imm value 10
15     input_code [7]  =  20'b 1000_011_001_00_0000_0110  ; //jmem imm value 6
16     input_code [10]  =  20'b 0000_011_100_010_000_0100 ; //or
17     input_code [11]  =  20'b 0101_010_011_00_0000_0101 ; //sw imm value 5
18     input_code [12] =  20'b 0010_111_001_00_0000_0111 ; //andi imm value 7
19     input_code [13] =  20'b 0110_100_101_00_0000_0001 ; //beq  imm value 1
20     input_code [14] =  20'b 0111_0000_0000_0000_0000  ; //jump to zero
21
22
23   end
24   always @ ( * )
25   begin
26     instruction <= input_code [ counter_in ] ;
27   end
28 endmodule
```



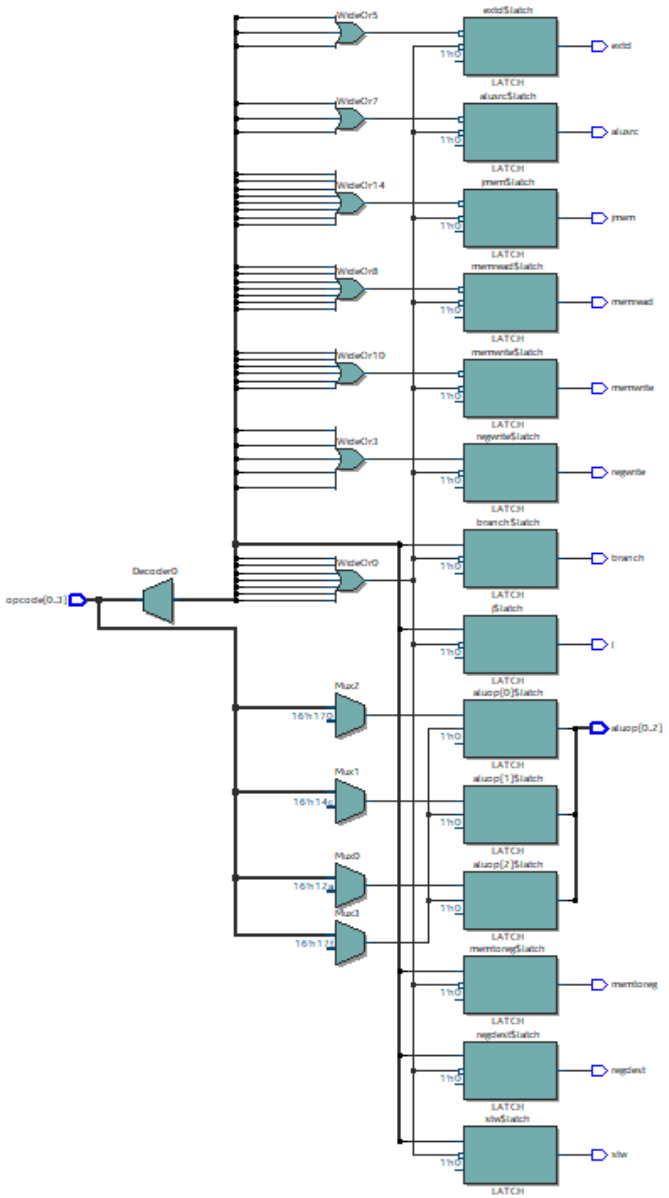


5-CONTROL UNIT

```
module control_unit(opcode,regdest,regwrite,extd,alusrc,memread,memwrite,memtoreg,j,branch,jmem,aluop,stw);

input [0:3] opcode ;
output reg regdest,regwrite,extd,alusrc,memread,memwrite,memtoreg,j,branch,stw;
output reg jmem; //new control signal used for the new operation
output reg [0:2] aluop;

always @(*)
 begin
  case (opcode)
//in case of r type instructions (add,sub,and,or,slt)
   4'b0000:
   begin
        regdest=1;
        regwrite=1;
        extd=0;
        alusrc=0;
        memread=0;
        memwrite=0;
        memtoreg=0;
        j=0;
        branch=0;
        jmem=0;
        stw=0;
        aluop= 3'b 000;
   end

// ADDi case
   4'b0001:
   begin
        regdest=0;
        regwrite=1;
        extd=1;
        alusrc=1;
        memread=0;
        memwrite=0;
        memtoreg=0;
        j=0;
        branch=0;
        jmem=0;
        stw=0;
        aluop= 3'b 001;
   end

   // ANDi case
    4'b0010:
    begin
         regdest=0;
         regwrite=1;
         extd=0;
         alusrc=1;
         memread=0;
         memwrite=0;
         memtoreg=0;
         j=0;
         branch=0;
         jmem=0;
         stw=0;
         aluop= 3'b 010;
    end

   // stw case
    4'b0011:
    begin
         regdest=0;
         regwrite=1;
         extd=1;
         alusrc=1;
         memread=0;
         memwrite=1;
         memtoreg=0;
         j=0;
         branch=0;
         jmem=0;
         aluop= 3'b 011;
         stw=1 ;
    end

   // lw case
    4'b0100:
    begin
         regdest=0;
         regwrite=1;
         extd=1;
         alusrc=1;
         memread=1;
         memwrite=0;
         memtoreg=1;
         j=0;
         branch=0;
         jmem=0;
         stw=0;
         aluop= 3'b 100;
    end
```

5-CONTROL UNIT

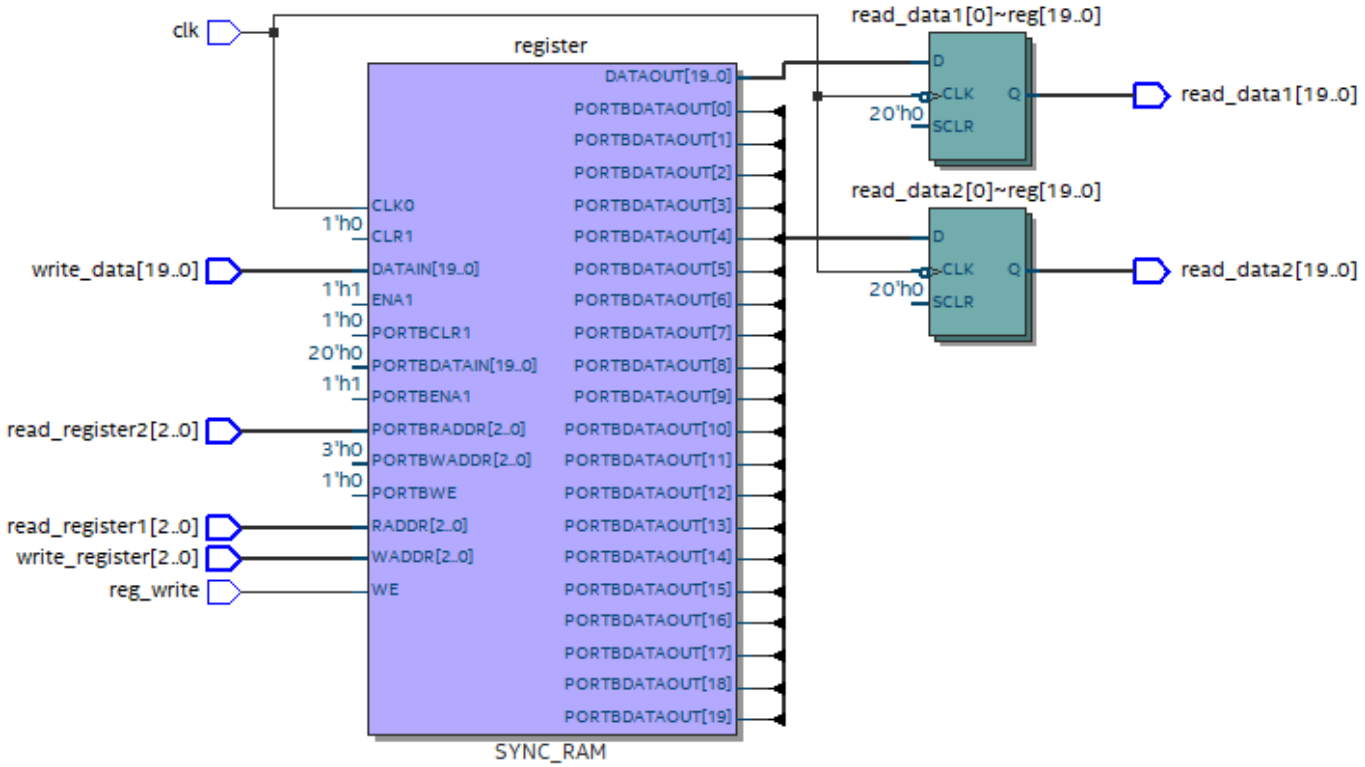
```
// sw case
  4'b0101:
  begin
        regdest=0;
        regwrite=0;
        extd=1;
        alusrc=1;
        memread=0;
        memwrite=1;
        memtoreg=0;
        j=0;
        branch=0;
        jmem=0;
        stw=0;
        aluop= 3'b 101;
  end

 // beq case
  4'b0110:
  begin
        regdest=0;
        regwrite=0;
        extd=1;
        alusrc=0;
        memread=0;
        memwrite=0;
        memtoreg=0;
        j=0;
        branch=1;
        jmem=0;
        stw=0;
        aluop= 3'b 110;
  end

 // j case
  4'b0111:
  begin
        regdest=0;
        regwrite=0;
        extd=0;
        alusrc=0;
        memread=0;
        memwrite=0;
        memtoreg=0;
        j=1;
        branch=0;
        jmem=0;
        stw=0;
        //we won't use the ALU in the jump case
  end

 // jmem case
  4'b1000:
  begin
        regdest=0;
        regwrite=0;
        extd=1;
        alusrc=1;
        memread=1;
        memwrite=1;
        memtoreg=0;
        j=0;
        branch=0;
        jmem=1;
        stw=0;
        aluop= 3'b 111;
  end
 endcase
end
endmodule
```



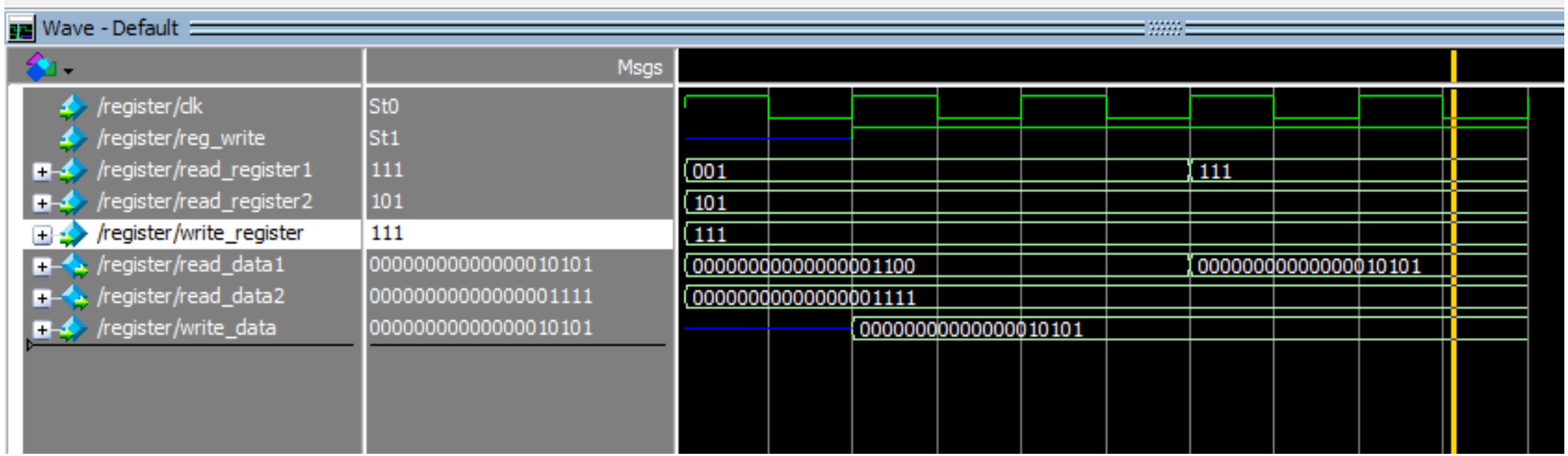
5-CONTROL UNIT

6- REGISTER FILE

```
module register ( clk , reg_write , read_register1 , read_register2 , write_register , read_data1 , read_data2 , write_data ) ;
  input clk , reg_write ;
  input [ 2 : 0 ] read_register1 ;
  input [ 2 : 0 ] read_register2 ;
  input [ 2 : 0 ] write_register ;
  output reg [ 19 : 0 ] read_data1 ;
  output reg [19 : 0 ] read_data2 ;
  input [ 19 : 0 ] write_data ;
  reg [ 19 : 0 ] register [ 0 : 7 ] ;
  initial
  begin
    register [0] = 20'b 0000_0000_0000_0000_0100 ;//4
    register [1] = 20'b 0000_0000_0000_0000_1100 ;//12
    register [2] = 20'b 0000_0000_0000_0000_0110 ; //6
    register [3] = 20'b 0000_0000_0000_0000_0011 ; //3
    register [4] = 20'b 0000_0000_0000_0000_1000 ; //8
    register [5] = 20'b 0000_0000_0000_0000_1111 ; //15
    register [6] = 20'b 0000_0000_0000_0000_0101 ;//5
    register [7] = 20'b 0000_0000_0000_0000_1001 ;//9
  end
  always @ ( posedge clk )
  begin
    if ( reg_write == 1 )
      begin
      register [ write_register ] <= write_data ;
      end
    end
  always @ (*)
  begin
    read_data1 <= register [ read_register1 ] ;
    read_data2 <= register [ read_register2 ] ;
    end
endmodule
```



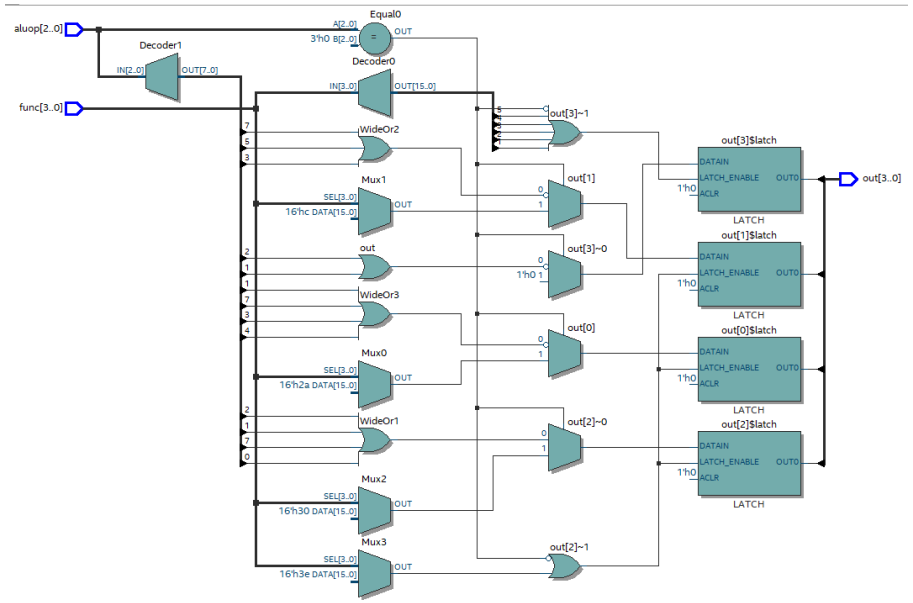
6- REGISTER FILE

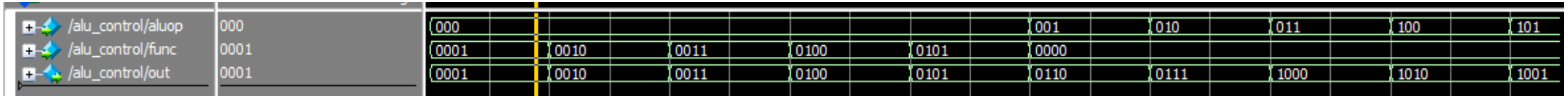


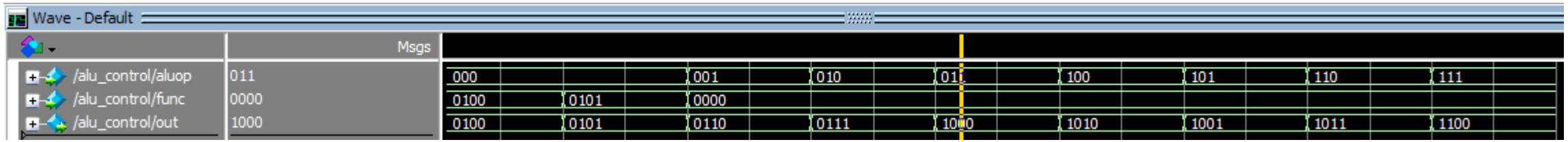
7- ALU CONTROL

```
Ln#
1   module alu_control ( aluop , func , out );
2
3   input [2:0] aluop ;
4   input [3:0] func ;
5   output reg [3:0] out ; // goes to the aluop of the alu
6
7   always @ (*) begin
8
9   if (aluop == 3'b 000) // for the r-type instructions
10  begin
11     case (func )
12
13         4'b 0001 : out = 4'b 0001 ; // add
14         4'b 0010 : out = 4'b 0010 ; // sub
15         4'b 0011 : out = 4'b 0011 ; // and
16         4'b 0100 : out = 4'b 0100 ; // or
17         4'b 0101 : out = 4'b 0101 ; // slt
18
19      endcase
```

```
23  else begin
24
25     case (aluop)
26          3'b 001 : out = 4'b 0110 ;  // addi
27          3'b 010 : out = 4'b 0111 ;  // andi
28          3'b 011 : out = 4'b 1000 ;  // stw case
29          3'b 101 : out = 4'b 1001 ;  //store
30          3'b 100 : out = 4'b 1010 ; // load
31          3'b 110 : out = 4'b 1011 ; //beq
32          3'b 111 : out = 4'b 1100 ; // jmem
33  default:
34     out = 4'b 1111 ;
35
36  endcase
37  end
38  end
39  endmodule
40
```

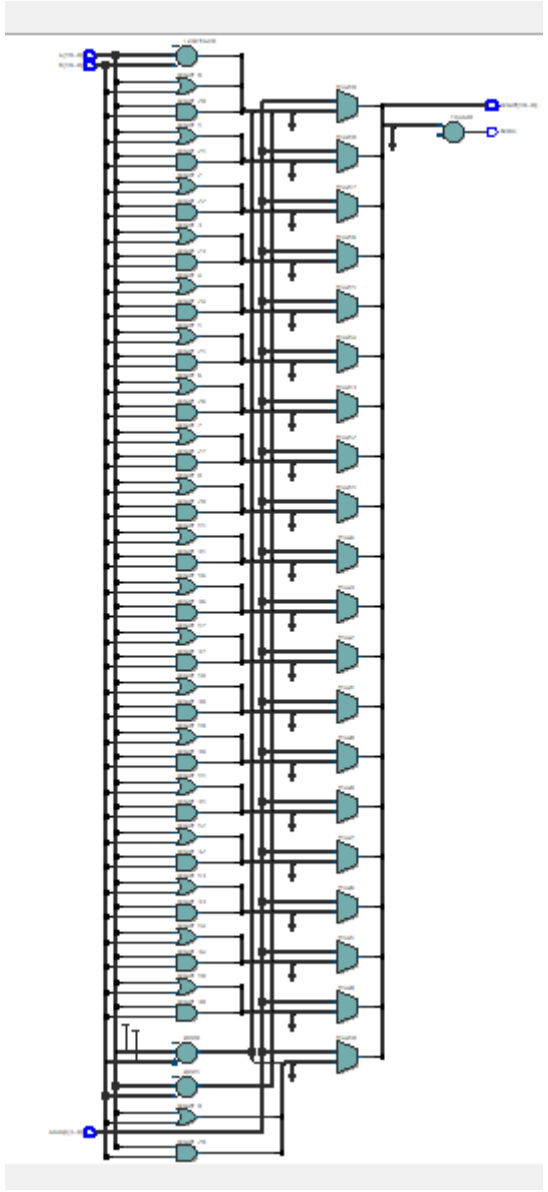






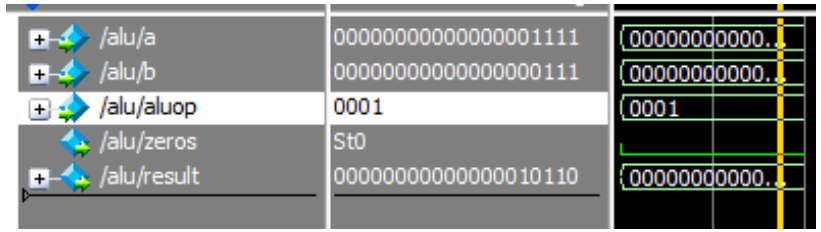
8-ALU

```
module alu ( a , b , aluop , result , zeros );

input [19:0] a ; // input 1
input [19:0] b ;  //input 2
input [3:0] aluop ;  // choose the operation done by the alu coming from the alu control block
output zeros ; // this output flags a 1 when the result of the operation is zero
output reg [19:0] result ; //output of alu of the operation chosen by aluop

always@(*) begin

   case (aluop)

     4'b 0001 : result = a + b ; // add

     4'b 0010 : result = a - b ; // sub

     4'b 0011 : result = a & b ; // and

     4'b 0100 : result = a | b ; // or

     4'b 0101 : result = (a < b) ? 1 : 0 ; // slt

     4'b 0110 : result = a + b ; // addi , alusrc=1 , add sign extensioned immediate to input 1 (a)

     4'b 0111 : result = a & b ; // andi , alusrc=1 , will AND sign extensioned immediate to input 1 (a)

     4'b 1000 : result = a + b ; //stw case at alusrc = 1 the sign extension immediate is added to input 1 (a)

     4'b 1001 : result = a + b ; //store at alusrc = 1 the sign extension immediate is added to input 1 (a)

     4'b 1010 : result = a + b ; //store at alusrc = 1 the sign extension immediate is added to input 1 (a)

    4'b 1011 : result = a - b ; // in beq if the result equals zero it will flags the zeros output

    4'b 1100 : result = a + b ; // jmem  here we will add the rs value to the sign extension value


   default :

    result = 0 ; //if the aluop is not one of these values we will set the result to be equal zero to enable the zero flag

  endcase

end

assign zeros = (result == 0 ) ; // flags an output 1 when result = 0

endmodule
```

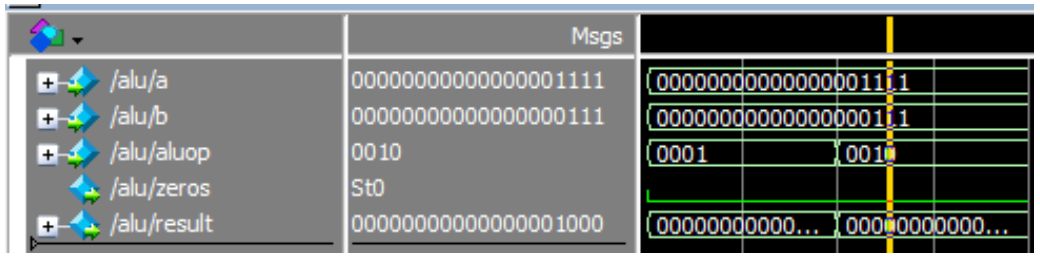


8-ALU

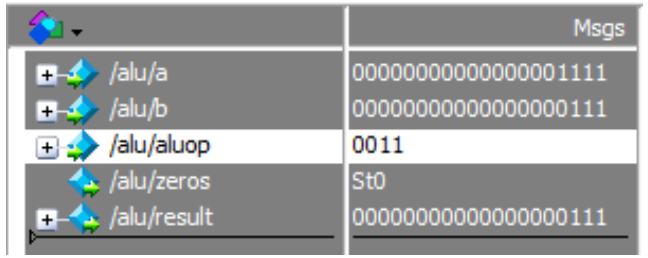
1)ADD



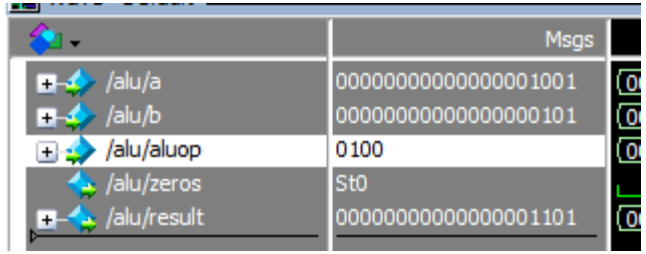
2)SUB



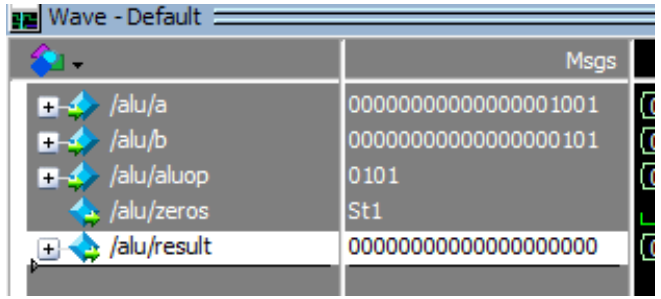
3)AND



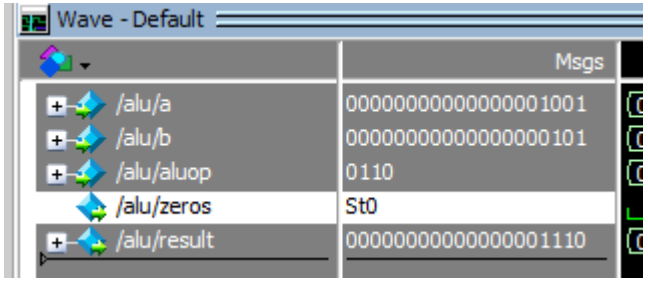
4)OR



5) SLT

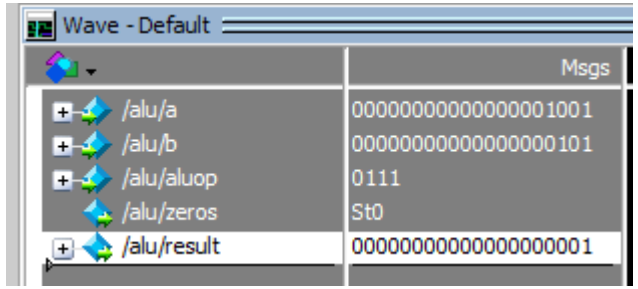


6)ADDI

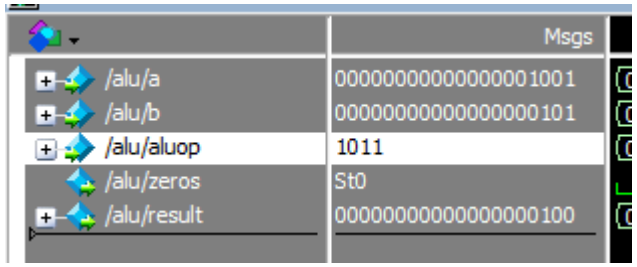


8-ALU

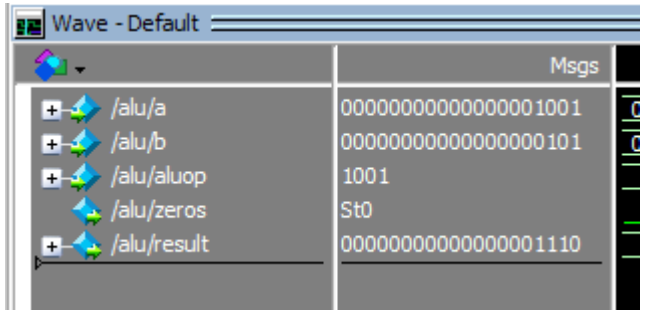
7)ANDI



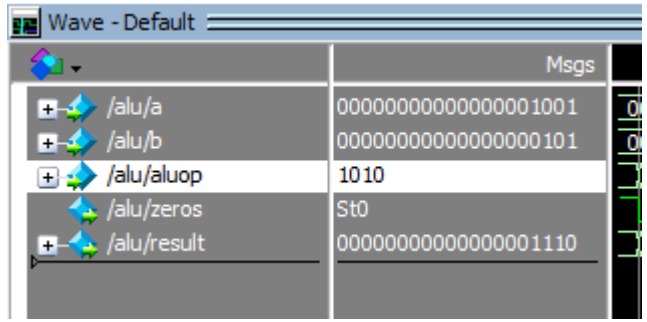
8) BEQ



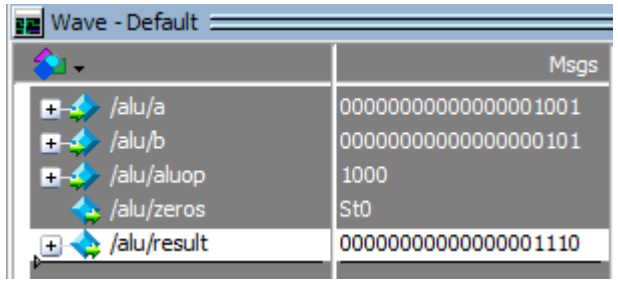
9)STW



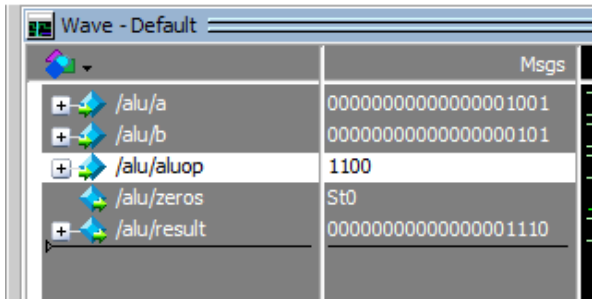
10)LW/SW



11) STW

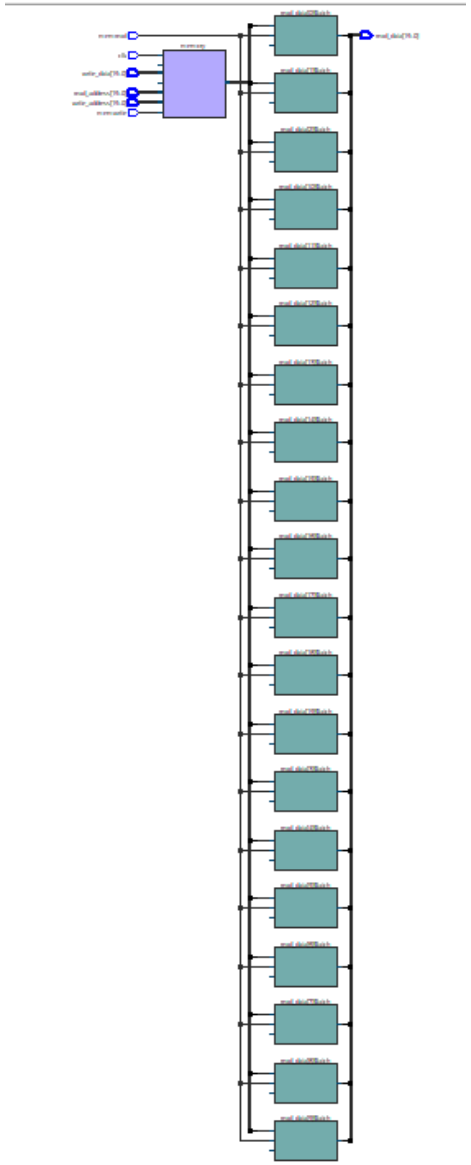


12)JMEM



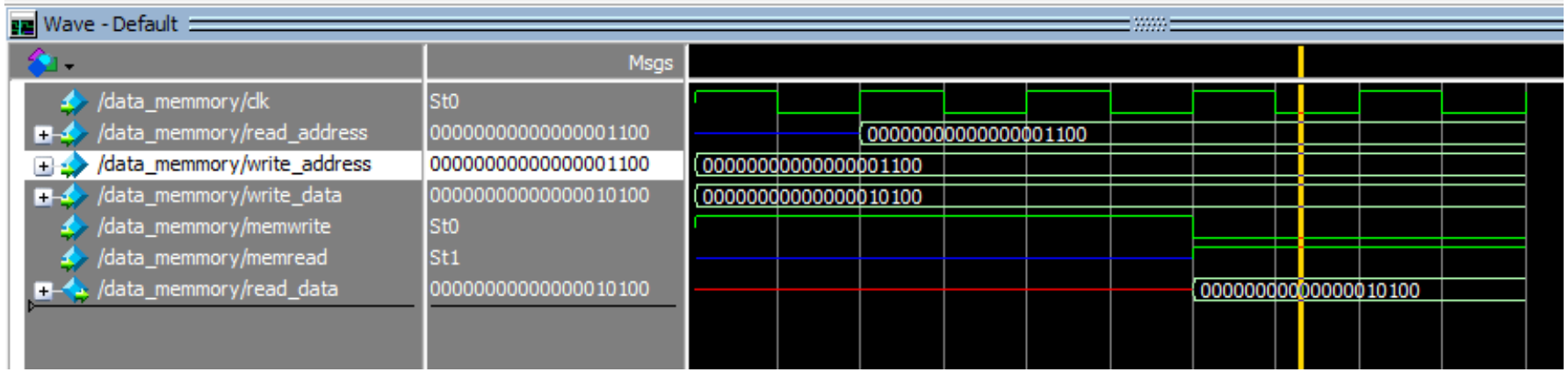
9-DATA MEMORY

```
Ln#
module data_memmory(clk,read_address,write_address,write_data, memwrite,memread,read_data);

 input clk;
  input [19:0] read_address;
  input [19:0] write_address;
  input [19:0] write_data;
  input memwrite;
  input memread;
  output reg [19:0] read_data ;


  reg [19:0] memory [0:1000];//idefining the memory as ana array that conatains 1 Mb of elements



always @(*)
  begin
    if (memread)
    begin
         read_data <= memory[read_address];
memory[12] = 20'b0000_0000_0000_0000_0001;
memory[0] = 20'b0000_0000_0000_0000_1010;
    end
end

         always @(posedge clk)
  begin
        if(memwrite) begin
          memory[write_address] <= write_data;
        end
  end
endmodule
```



9-DATA MEMORY

- Writing the value of 10100 in reg[12]



The reading the value of reg[12]

10- TOP MODULE

1-Initialization of the wires and regs

```
module major_task ( input clk , input rst );

//wires


reg [19:0] pc_out; //wires out of the pc

wire [19:0] instruction_wire; //wires out of the imem

wire [2:0] mux_writereg; //mux for the data to be written in the reg file

wire [19:0] read_data1; //read data 1 from regs file

wire [19:0] read_data2; //read data 2 from regs file

wire [19:0] sign_extension ; // sign extension output value

wire [3:0] alu_operation ; // the output of the alu control

wire [19:0] mux_alusrc ; //output of the mux entering the alu

wire [19:0] alu_output ; // gives the result of the operation done in the alu

wire zero_flag ; // gives an output of 1 if the result of the alu is equal zero

wire [ 19 : 0 ] write_data ; // write data in data memory

wire [ 19 : 0 ] read_data ;  // read data feom memory
```

```
wire [ 19 : 0 ] data_write_reg ; // output of mux to write register ( memtoreg )

wire [ 19 : 0 ] adder1_output ; // increment pc with 1

wire [ 19 : 0 ] adder2_output ; // output ( PC + 1 ) + ( 1 * sign extention )

wire and_output ;  // selection of branch

wire [ 19 : 0 ] output_mux_branch ;  // slsection between PC + 4 and branch

wire [ 19 : 0 ] read_memory_address ;  //  selection between rt and alu output

wire [ 19 : 0 ] instruction_fetch ;  // slection between j and branch

wire [ 19 : 0 ] wire_write_address ; // selection between rs , alu output ;

wire [ 19 : 0 ] jmem_mux_output ;  // selection between read data2 and adder1_output

wire [ 2 : 0 ] out_last ; // slection between rs and rd and rt

wire [ 19 : 0 ] pc_in2 ; //last value of the pc

    //ctrl wires

wire regdest1 , regwrite1 , extd1 , alusrc1 , memread1 , memwrite1 , memtoreg1 , j1 , branch1 , jmem1 , stw1  ;

wire [2:0] aluop1 ;
```

2-PC Counter

```
reg [ 19 : 0 ] a ;
initial
begin
  a = 20'b 0000_0000_0000_0000_0001 ;

end

//pc counter
 always @ (posedge clk )
  begin
    if (rst)
      begin
        pc_out <= 20'b 0000_0000_0000_0000_0000 ;
      end
   else
     begin
        pc_out <= pc_in2;
     end
 end
```

3-Wiring of :

(Imem, Regdest Mux, Register File, CTRL Unit , Sign Extension , ALU Unit)

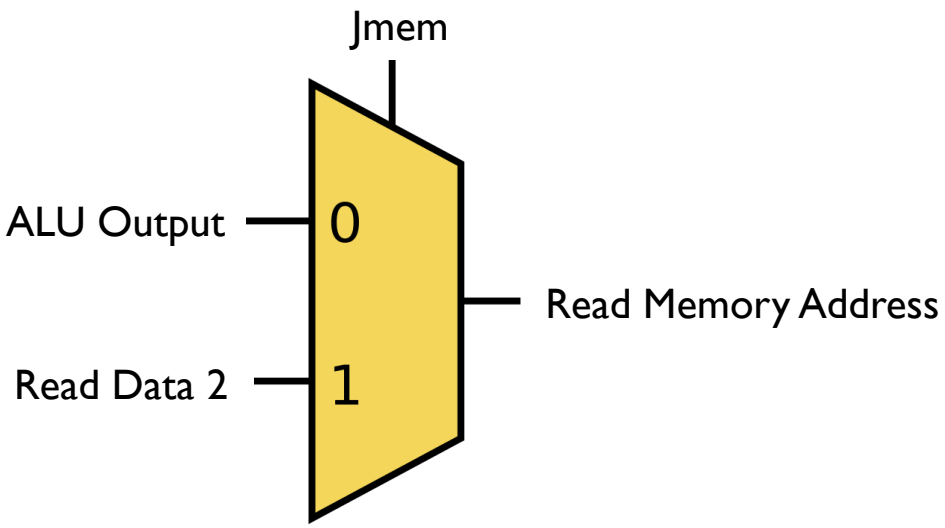
```
instruction_memory imem ( pc_out , instruction_wire ); //instruction memory

MUX mux1( instruction_wire [12:10] , instruction_wire [9:7] , regdest1 , mux_writereg ); //mux for write register

register register_file( clk , regwrite1 , instruction_wire [15:13] , instruction_wire [12:10] , out_last , read_data1 , read_data2 , data_write_reg );

control_unit CTRL ( instruction_wire [19:16] , regdest1 , regwrite1 , extd1 , alusrc1 , memread1 , memwrite1 , memtoreg1 , j1 , branch1 , jmem1 , aluop1 , stw1 );

sinexten signext ( extd1 , instruction_wire [9:0] , sign_extension );

alu_control ALU_CTRL ( aluop1 , instruction_wire [3:0] , alu_operation );

MUX mux2 ( read_data2 , sign_extension , alusrc1 , mux_alusrc ); //data entering the alu

alu ALU_UNIT ( read_data1 , mux_alusrc , alu_operation ,alu_output , zero_flag ) ; //alu excution
```

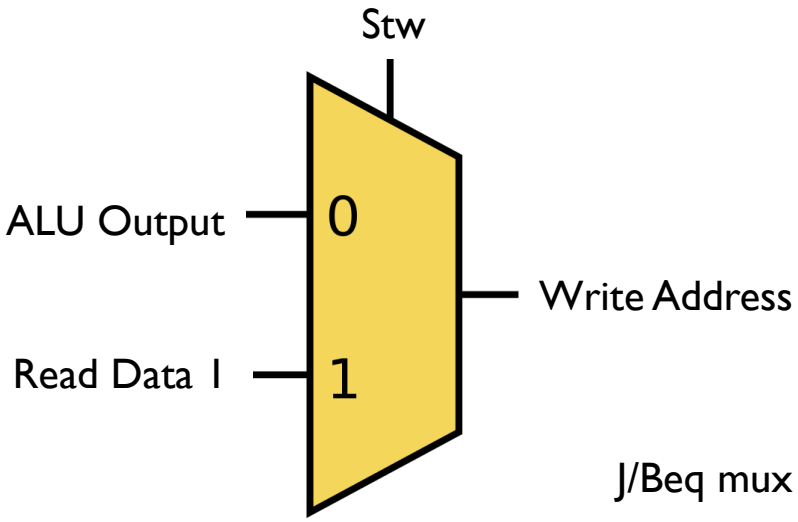
3-Wiring of :

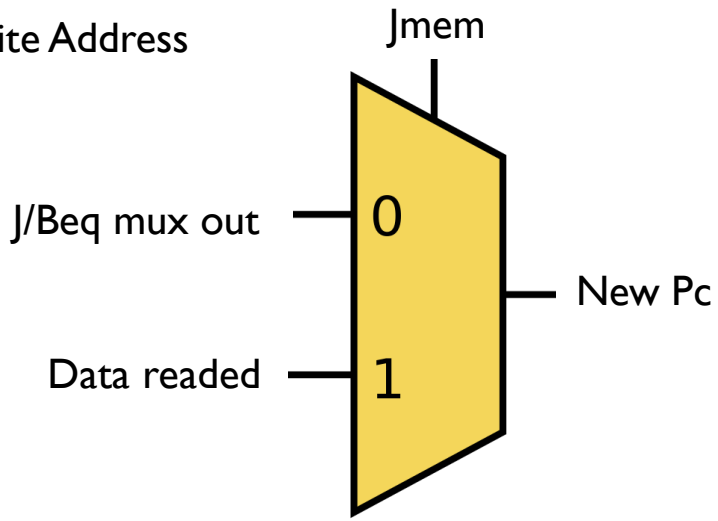
(5 New MUXs , Dmem, Memtoreg MUX ,Adder For PC ,Adder Of The branch , Branch AND Gate, MUX Selecting The Branch , MUX Between J &BEQ)

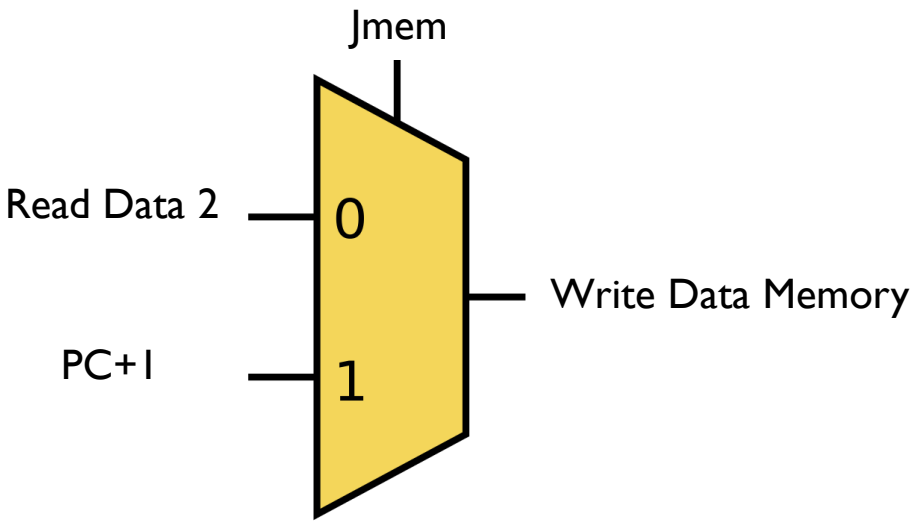
```
MUX read_adddress_memory (  alu_output ,  read_data2 , jmeml , read_memory_address ) ;//data to be readed from the data memory

MUX write_data_memory ( read_data2 , adder1_output , jmeml , write_data ) ;//data to be written in the data memory

MUX write_address_memory ( alu_output , read_data1 , stwl , wire_write_address ) ;

MUX last ( mux_writereg , instruction_wire [ 15 : 13 ] , stwl , out_last ) ;

data_memmory Dmem ( clk , read_memory_address  ,  wire_write_address , write_data , memwritel , memreadl ,read_data ) ;

MUX memtoreg_mux ( alu_output , read_data , memtoregl , data_write_reg ) ;

adder Pc_increment4 ( pc_out , a , adder1_output ) ;//pc+1

adder beq_adder ( adder1_output , sign_extension ,adder2_output ) ;

And_gate branch_select ( zero_flag , branchl , and_output ) ;

MUX branch_mux ( adder1_output , adder2_output , and_output , output_mux_branch ) ;

MUX j_mux ( output_mux_branch , {pc_out[19:16], instruction_wire [ 15 : 0 ]} , jl , instruction_fetch  ) ;

MUX pc_mux ( instruction_fetch , read_data , jmeml , pc_in2 ) ;


endmodule
```

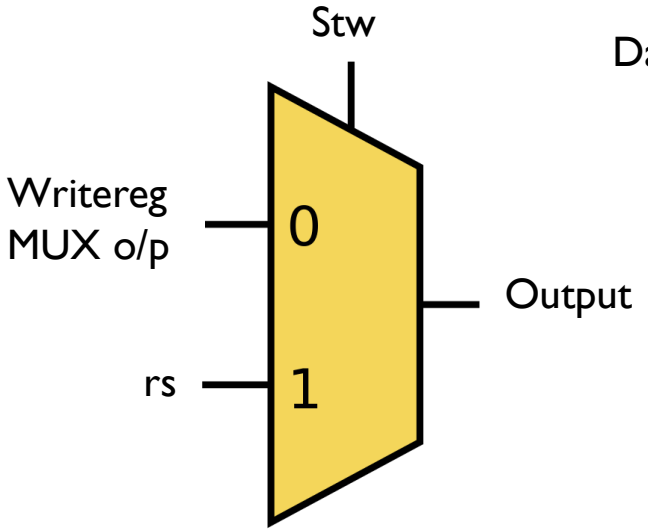
5-NEW MUXs


Jmem
ALU Output
0
Read Memory Address
Read Data 2
1


Stw
ALU Output
0
Write Address
Read Data 1
1


Jmem
J/Beq mux out
0
New Pc
Data readed
1


Jmem
Read Data 2
0
Write Data Memory
PC+1
1


Stw
Writereg
MUX o/p
0
Output
rs
1

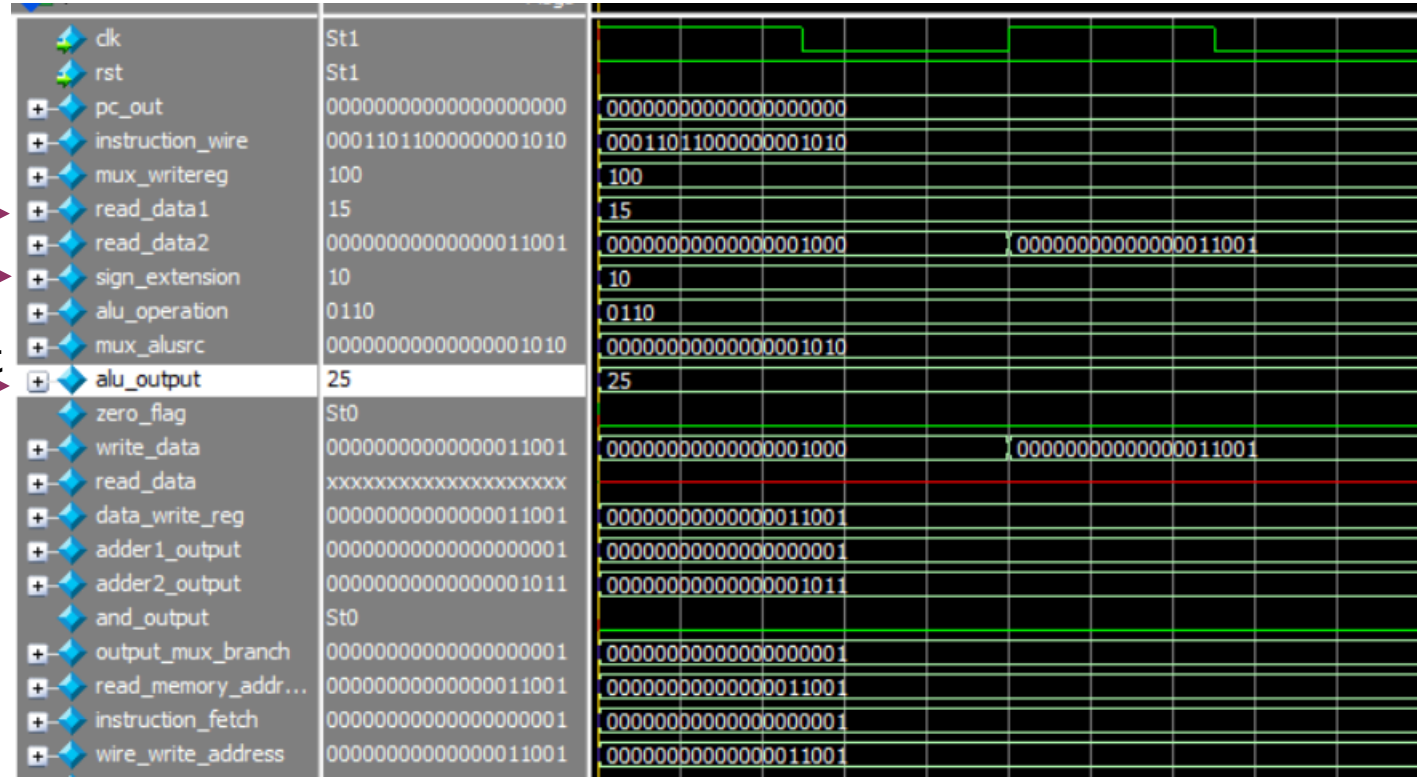
6- Simulations

I-ADDi

rs

imm

output



Result of R[rs]+imm=25

R[rt]=25 , rt=$t4

2-Sub

rs

rt

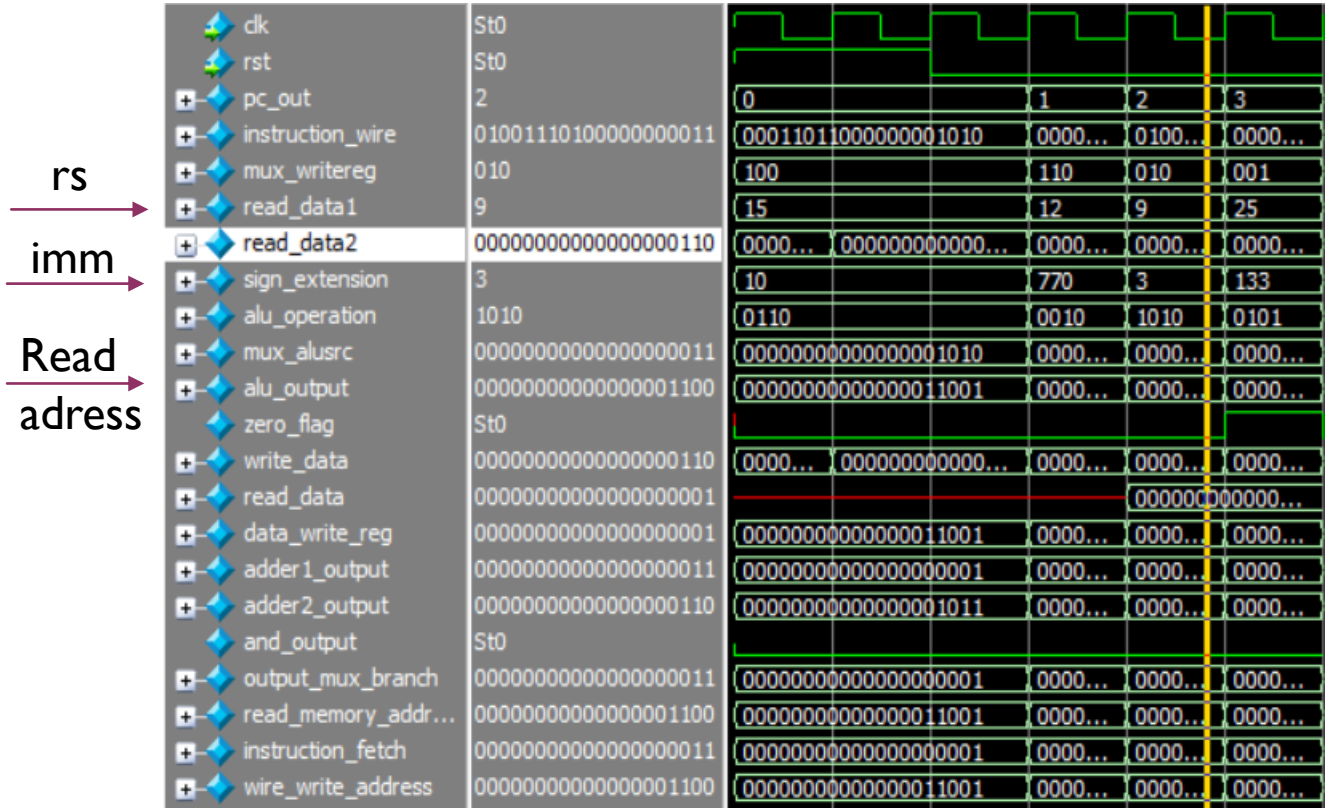
output

clk	St1	
rst	St0	
pc_out	00000000000000000001	00000000000000000001
instruction_wire	00000011111100000010	00000011111100000010
mux_writereg	110	110
read_data1	12	12
read_data2	9	9
sign_extension	00000000001100000010	00000000001100000010
alu_operation	0010	0010
mux_alusrc	00000000000000001001	00000000000000001001
alu_output	3	3
zero_flag	St0	
write_data	00000000000000001001	00000000000000001001
read_data	xxxxxxxxxxxxxxxxxxxx	
data_write_reg	00000000000000000011	00000000000000000011
adder1_output	00000000000000000010	00000000000000000010
adder2_output	00000000001100000100	00000000001100000100
and_output	St0	
output_mux_branch	00000000000000000010	00000000000000000010
read_memory_addr...	00000000000000000011	00000000000000000011
instruction_fetch	00000000000000000010	00000000000000000010
wire_write_address	00000000000000000011	00000000000000000011

Result of R[rs]-R[rt]=3

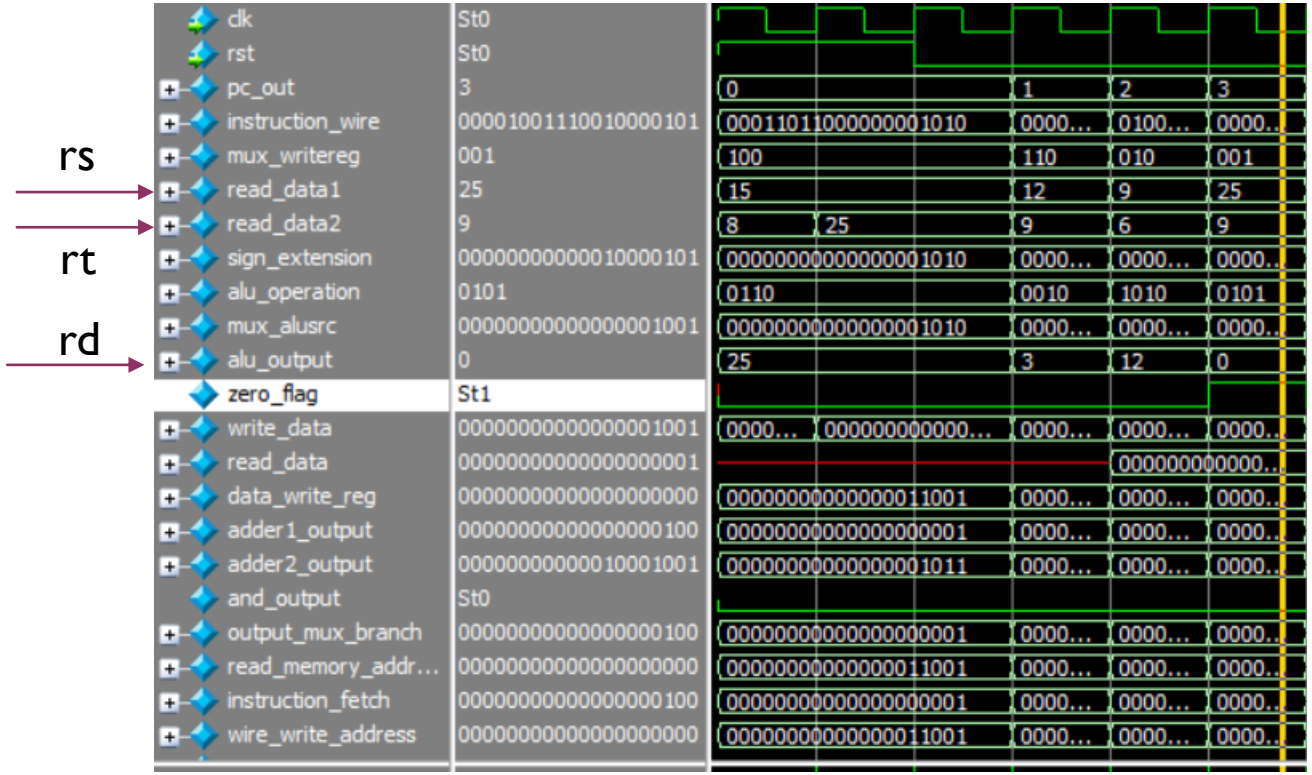
R[rd]=3 , rt=$t6

3-lw



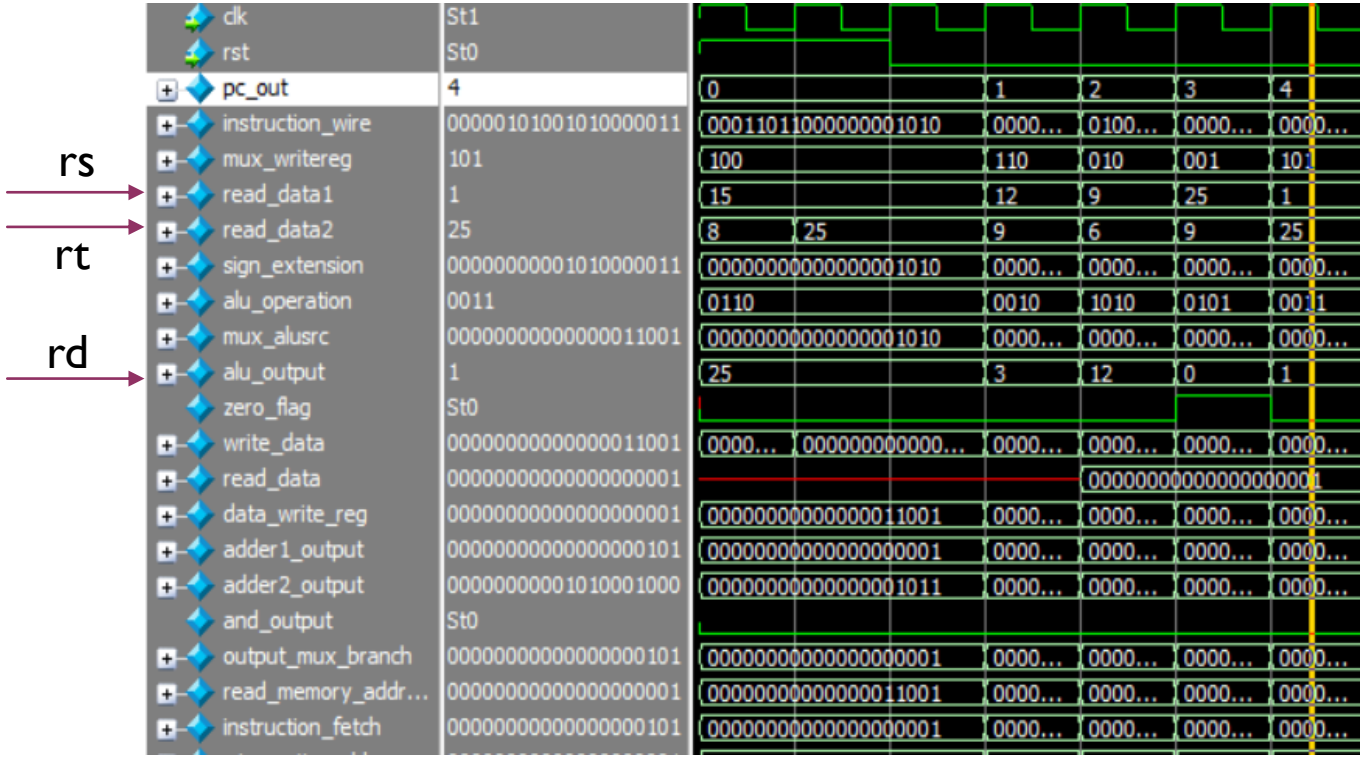
Lw the value in MEM[rs+offset] in rt.
rs + offset = 12
MEM[12] = 1
R[rt] = 1 , rt = $t2

4-slt



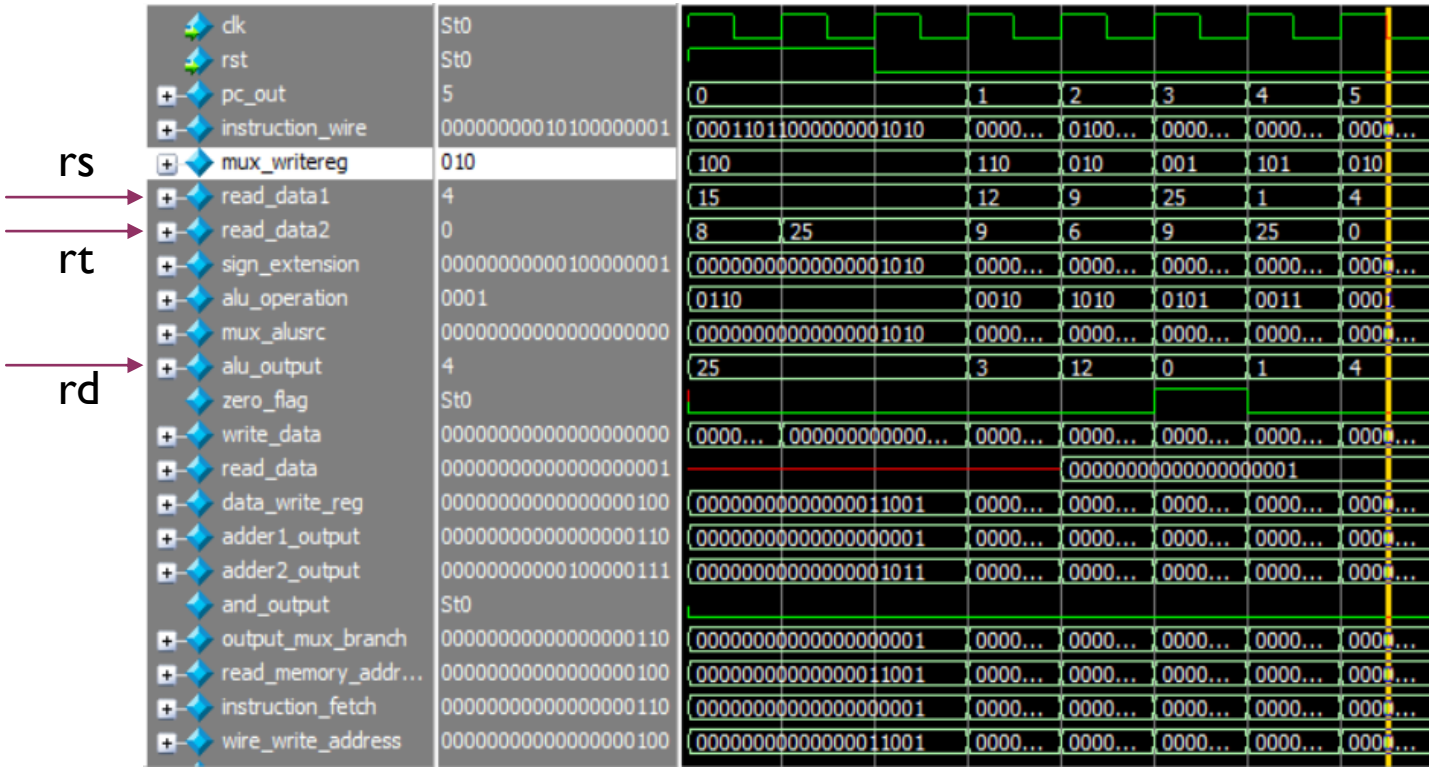
Set R[rd] = 1 if R[rs] < R[rt].
R[rs]=25
R[rt]=9
Condition not true so set the value of R[rd] to 0
rd = $t1

5-AND



R[rs] & R[rt]= R[rd]
25 & 1 =1
R[rd] = 1
rd = $t5

6-ADD



R[rs] + R[rt]= R[rd]

4 + 0 = 4

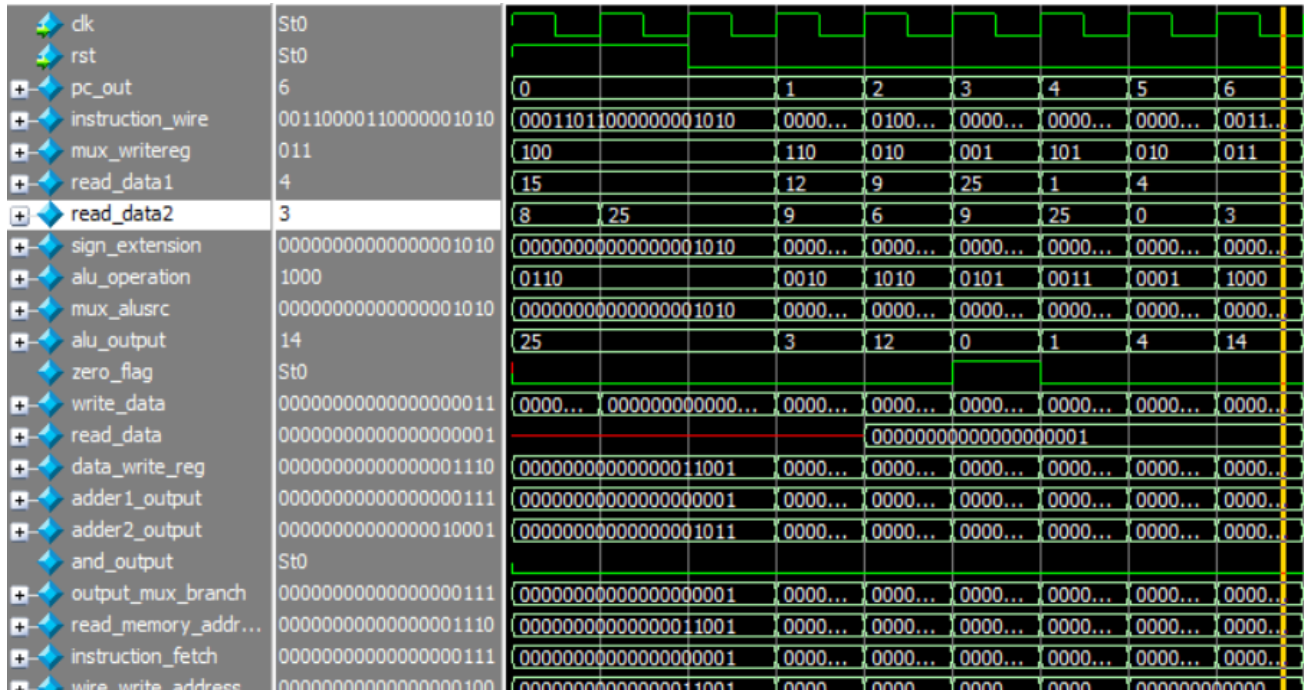
R[rd] = 4

rd = $t2

7-Stw

rs

rt



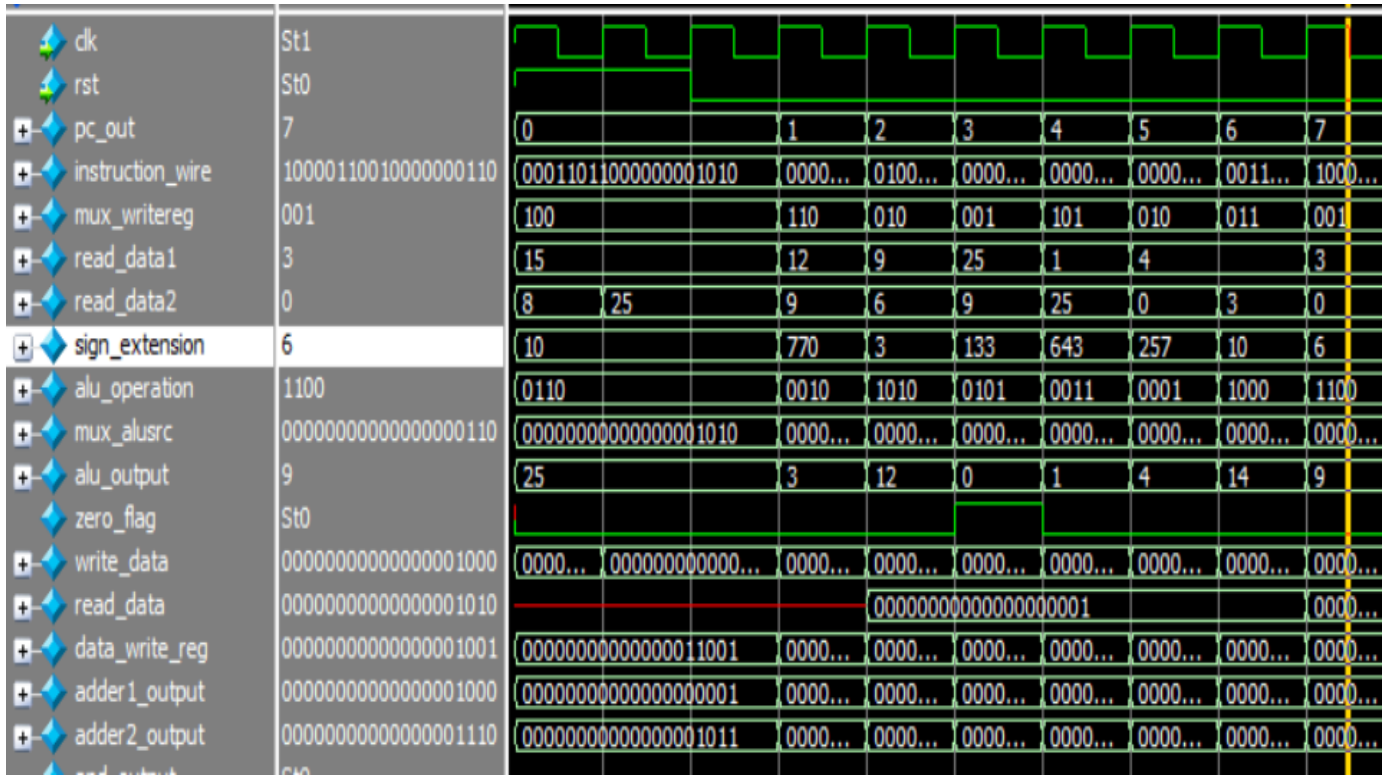
imm

Memory[R[rs]] = R[rt]
R[rs]=4 , rs=$t0
rt=3
Memory[4]=3 << Dmem

4	3

R[rs] = R[rs] + imm
Imm=10
R[rs]=14

8-Jmem



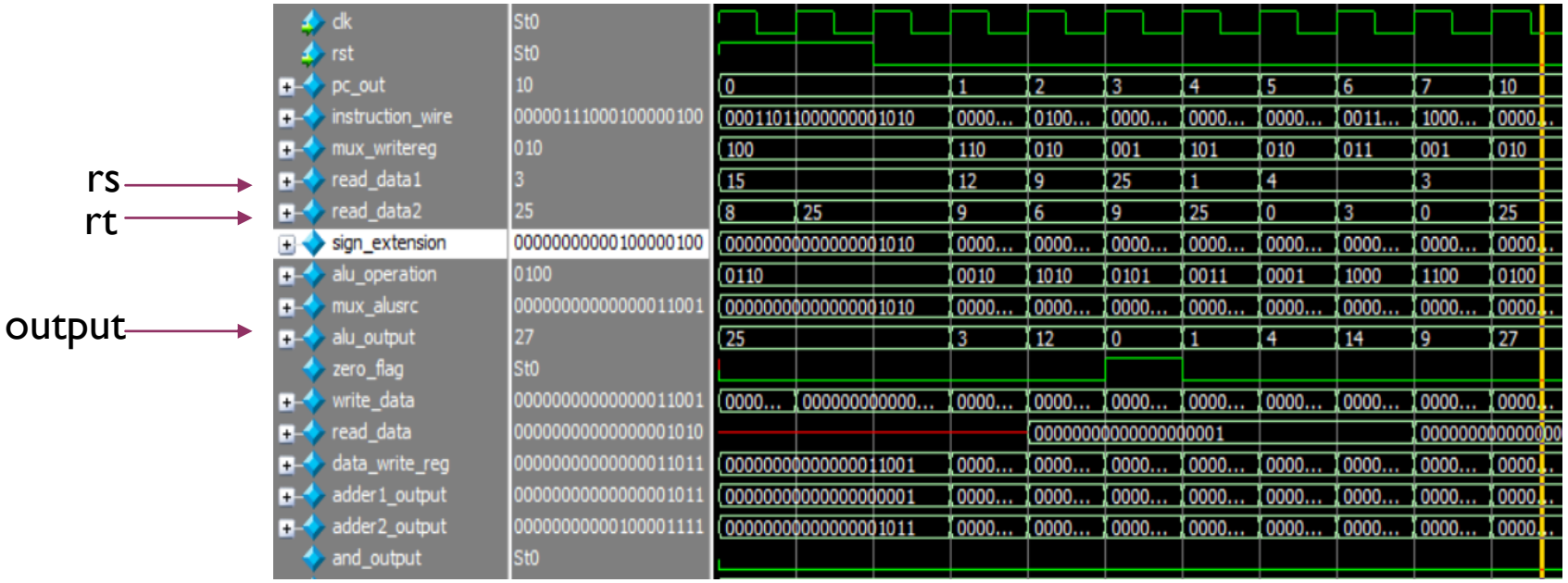
Memory [R[rs] + offset] = PC + 1
Offset=6
R[rs]=3
Memory [9] = 8

PC = Memory[R[rt]]
R[rt]=0 , rt=$t1

PC = Memory[0] =10

0	10
1	x
2	x
3	x
4	3
5	x
6	x
7	x
8	x
9	8

9-OR

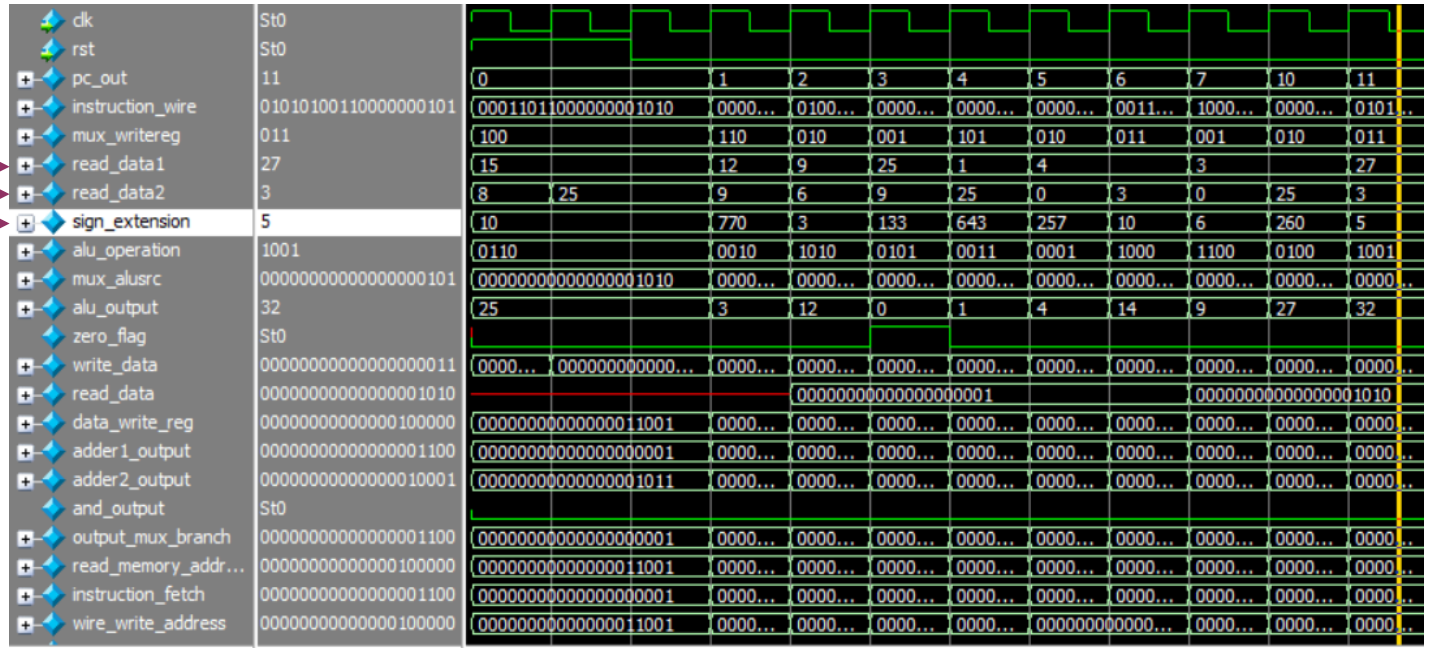


R[rs] | R[rt] = R[rd]
R[rs]=3
R[rt] = 25
R[rd] = 27

rd = $t2

10-sw

rs
rt
imm



Sw in the address MEM[R[rs]+offset] R[rt].

rs + offset = 32

MEM[32] = 3

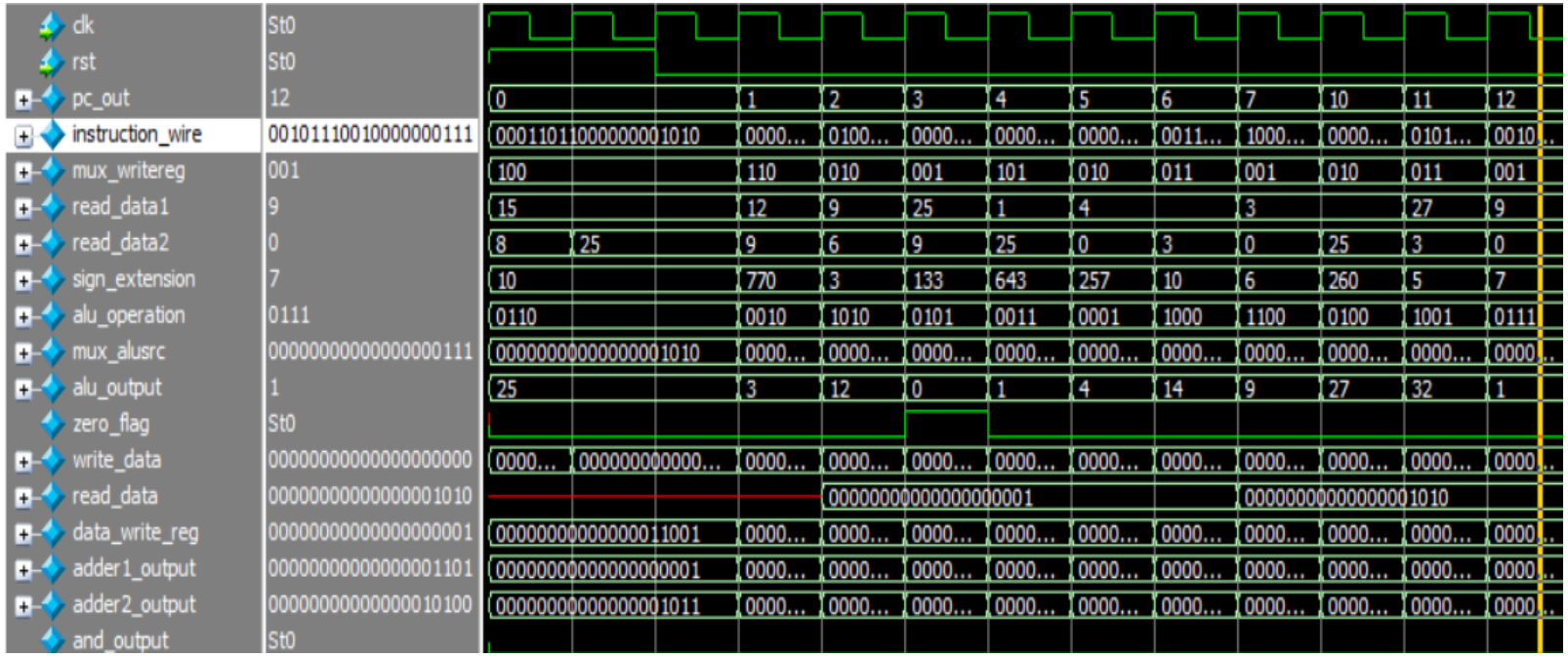
31	x
32	3
33	x

11-ANDi

rs

imm

out



R[rs] & Imm= R[rt]

Imm = 7

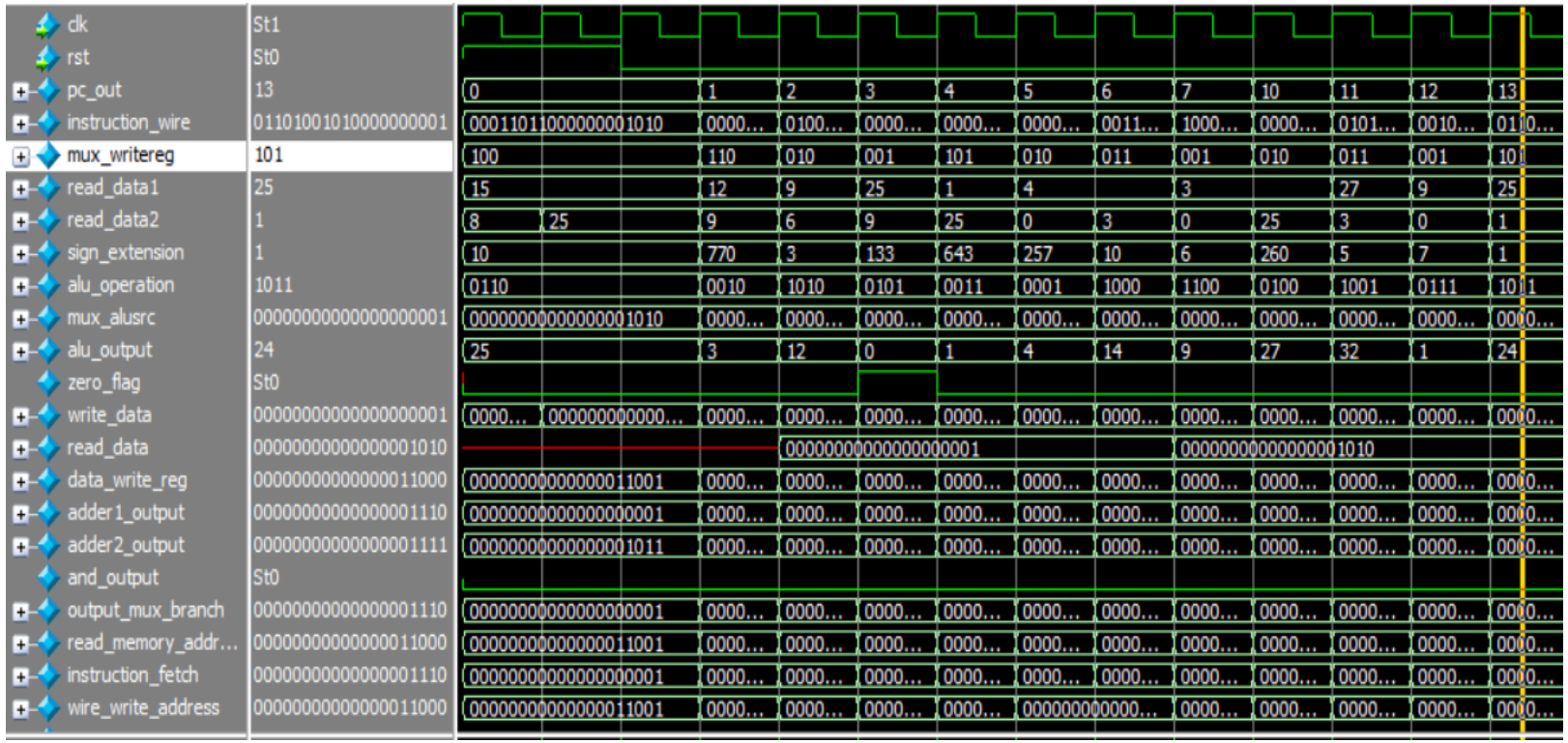
R [rs] = 9

9 & 7 =1

R[rt] = 1

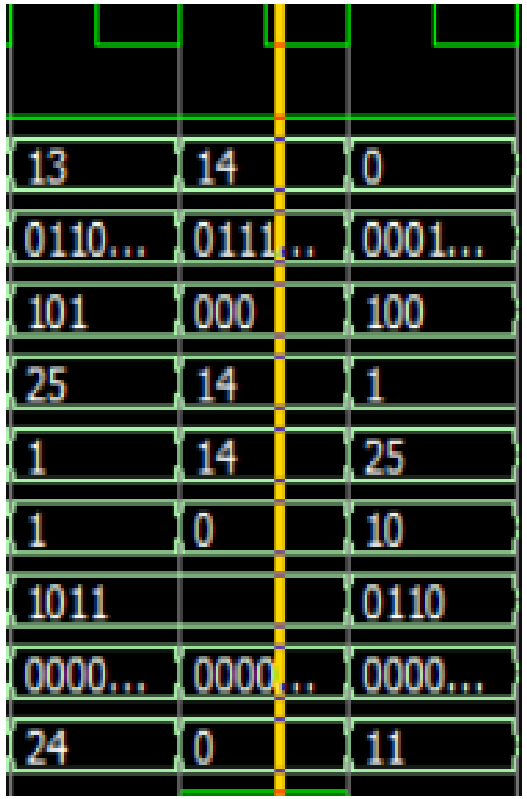
rt = $t1

12-Beq



Branch to 1*offset+pc+1
when R[rs] = R[rt]
Imm = 7
R [rs] = 9
9 & 7 =1
R[rt] = 1
rt = $t1

13-Jump



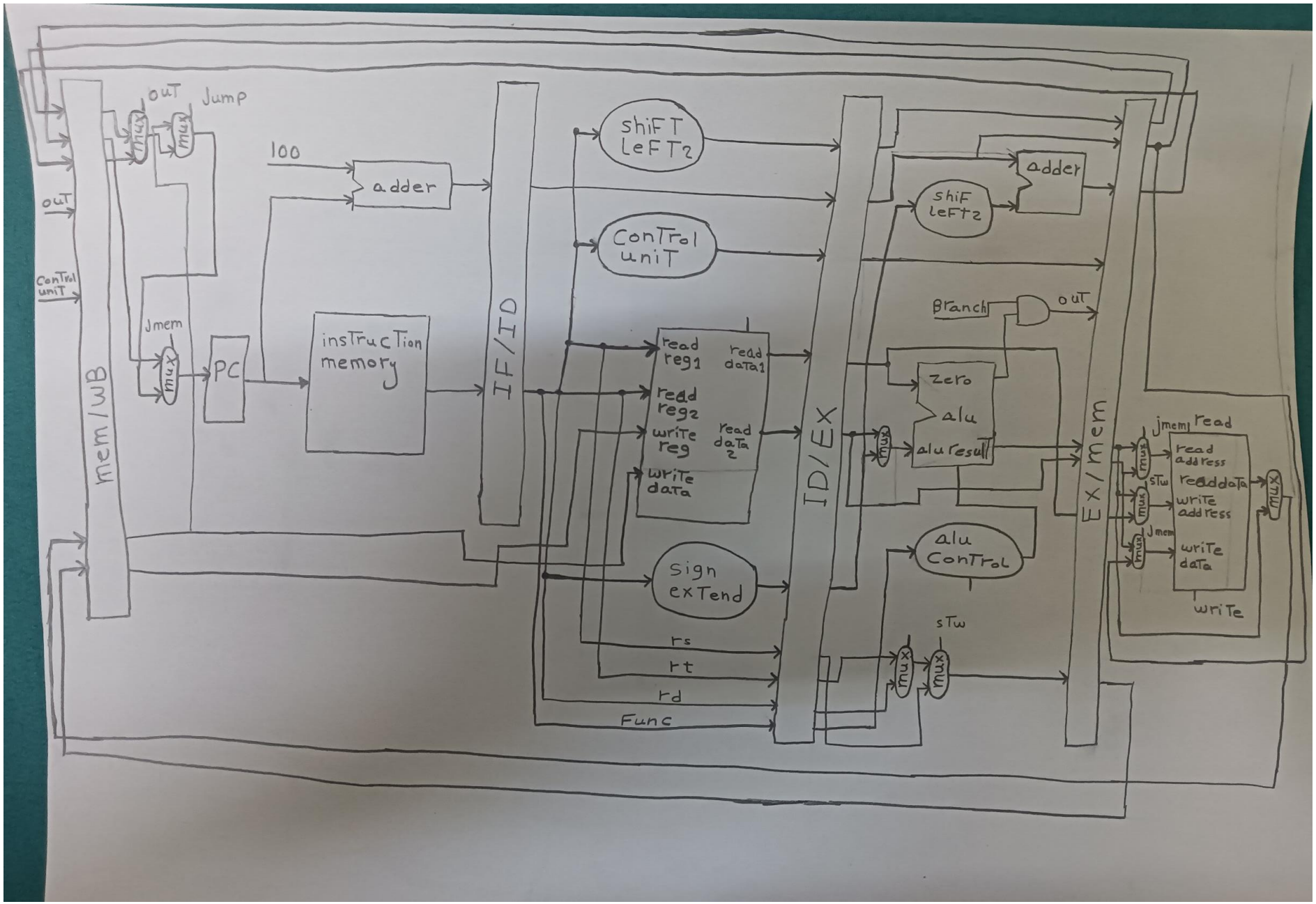
PC

Jump to the address 0.

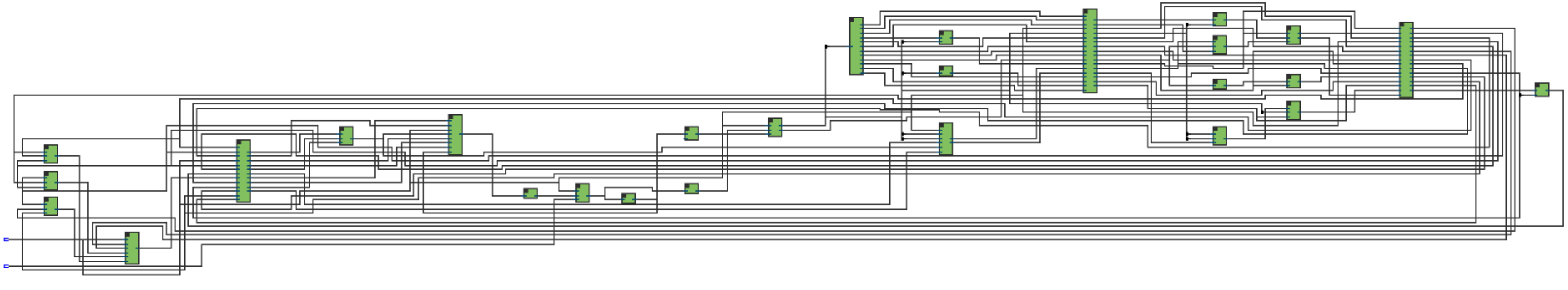
As shown aside the pc value of the jump is 14 after clock cycle it will jump to the address specified in the instruction code.

pipeline

-MIPS

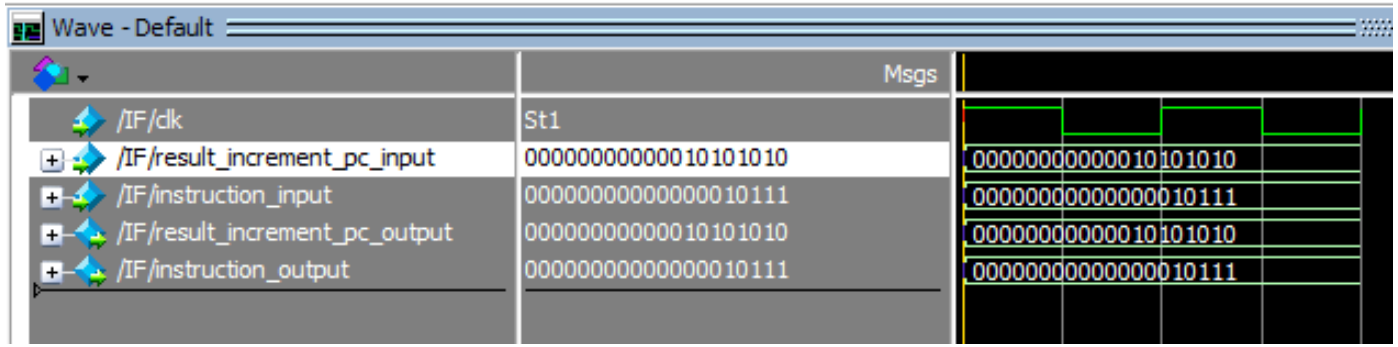

out
Jump
mux
mux
100
adder
out
Control unit
Jmem
mux
PC
instruction memory
mem/WB
IF/ID
shift left 2
Control unit
read reg1
read reg2
write reg
write data
read data1
read data 2
Sign extend
rs
rt
rd
Func
ID/EX
adder
shif left 2
Branch
out
zero
alu
alu result
alu Control
sTw
mux
mux
EX/mem
jmem
read
read address
readdata
write address
write data
sTw
Jmem
write
mux

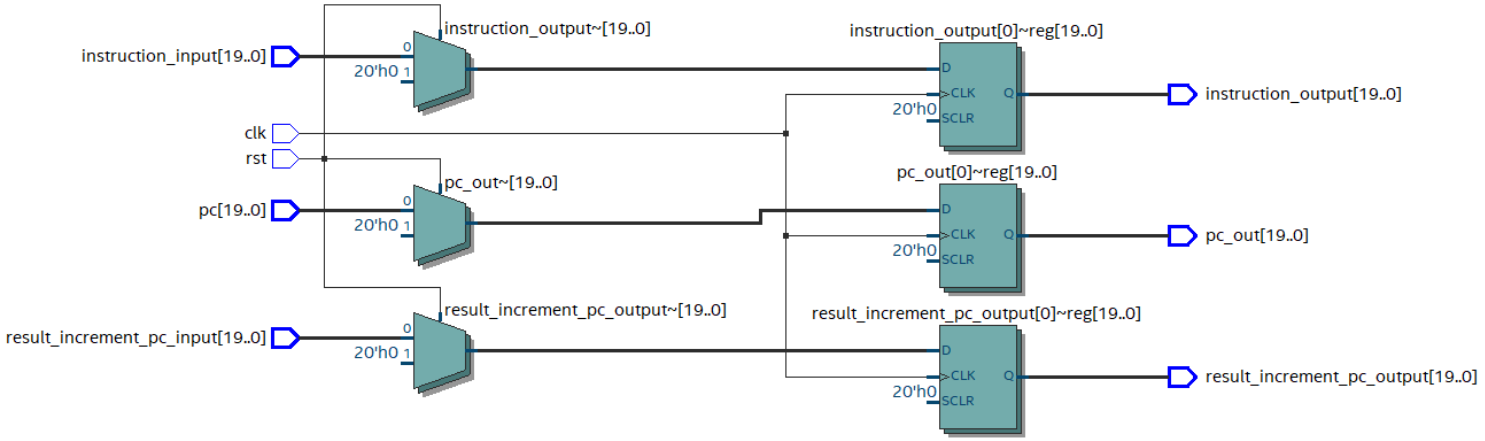
RTL SCHEMATIC ANALYSIS



I- IF/ID

```
1  module IF ( clk , result_increment_pc_input , instruction_input ,
2   result_increment_pc_output , instruction_output ) ;
3
4  input clk ;
5
6  input [ 19 : 0 ]  result_increment_pc_input ;
7  input [ 19 : 0 ]  instruction_input ;
8  output reg [ 19 : 0 ] result_increment_pc_output ;
9  output reg [ 19 : 0 ] instruction_output ;
10
11 always @( posedge clk )
12 begin
13   result_increment_pc_output <= result_increment_pc_input ;
14   instruction_output <= instruction_input ;
15 end
16 endmodule
17
18
```





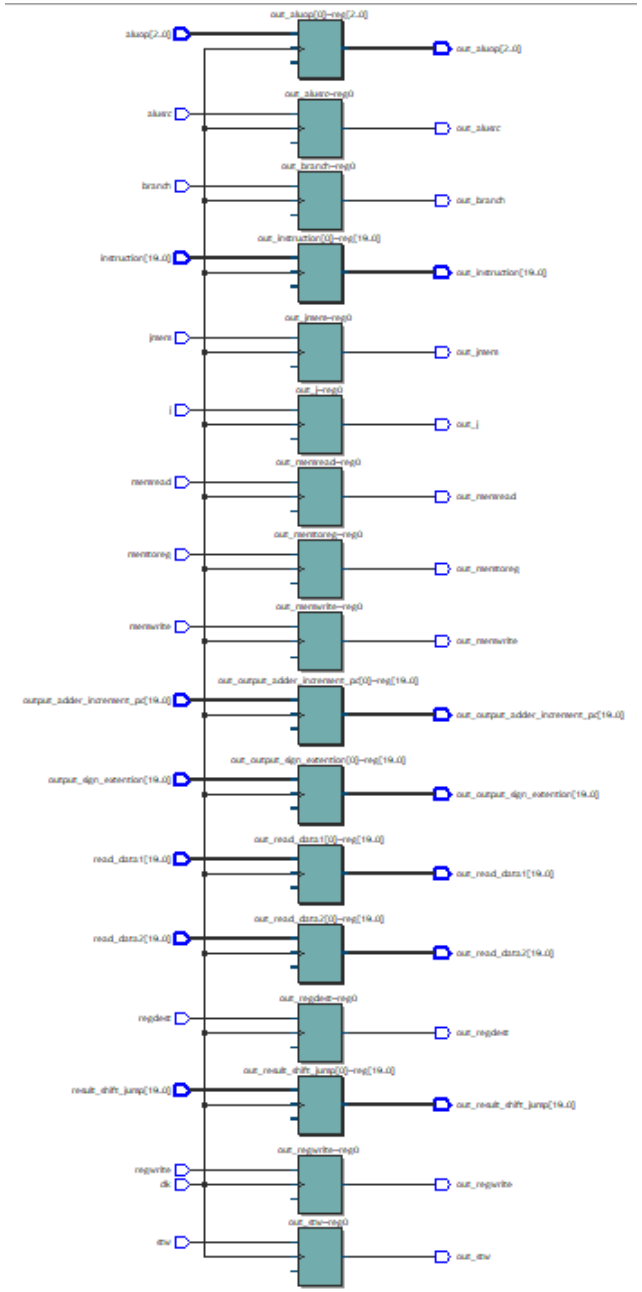
2- ID/EX

```
module ID ( clk , regdest , regwrite , alusrc , memread , memwrite , memtoreg ,
 j , branch , jmem , aluop , stw , result_shift_jump , output_adder_increment_pc ,
 read_data1 , read_data2 , output_sign_extention , instruction ,  out_regdest , out_regwrite ,
 out_alusrc , out_memread , out_memwrite , out_memtoreg , out_j , out_branch , out_jmem , out_aluop ,
 out_stw , out_result_shift_jump , out_output_adder_increment_pc , out_read_data1 , out_read_data2 ,
 out_output_sign_extention , out_instruction ) ;

  input clk , regdest , regwrite , alusrc , memread , memwrite , memtoreg , j , branch , jmem  , stw  ;
  input [ 2 : 0 ] aluop ;
  input [ 19 : 0 ] result_shift_jump ;
  input [ 19 : 0 ] output_adder_increment_pc ;
  input [ 19 : 0 ] read_data1 ;
  input [ 19 : 0 ] read_data2 ;
  input [ 19 : 0 ] output_sign_extention ;
  input [ 19 : 0 ] instruction ;


  output reg  out_regdest , out_regwrite , out_alusrc , out_memread ,
out_memwrite , out_memtoreg , out_j , out_branch , out_jmem  , out_stw  ;
  output reg [ 2 : 0 ] out_aluop ;
  output reg [ 19 : 0 ] out_result_shift_jump ;
  output reg [ 19 : 0 ] out_output_adder_increment_pc ;
  output reg [ 19 : 0 ] out_read_data1 ;
  output reg [ 19 : 0 ] out_read_data2 ;
  output reg [ 19 : 0 ] out_output_sign_extention ;
  output reg [ 19 : 0 ] out_instruction ;
```

```

  always @ ( posedge clk )
  begin
     out_regdest <= regdest ;
     out_regwrite <= regwrite ;
     out_alusrc <= alusrc ;
     out_memread <= memread ;
     out_memwrite <= memwrite ;
     out_memtoreg <= memtoreg ;
     out_j <= j ;
     out_branch <= branch ;
     out_jmem  <= jmem ;
     out_stw <= stw ;
     out_aluop <= aluop ;
     out_result_shift_jump <= result_shift_jump ;
     out_output_adder_increment_pc <= output_adder_increment_pc ;
     out_read_data1 <= read_data1 ;
     out_read_data2 <= read_data2 ;
     out_output_sign_extention <= output_sign_extention ;
     out_instruction <= instruction ;
   end
endmodule
```

2- ID/EX

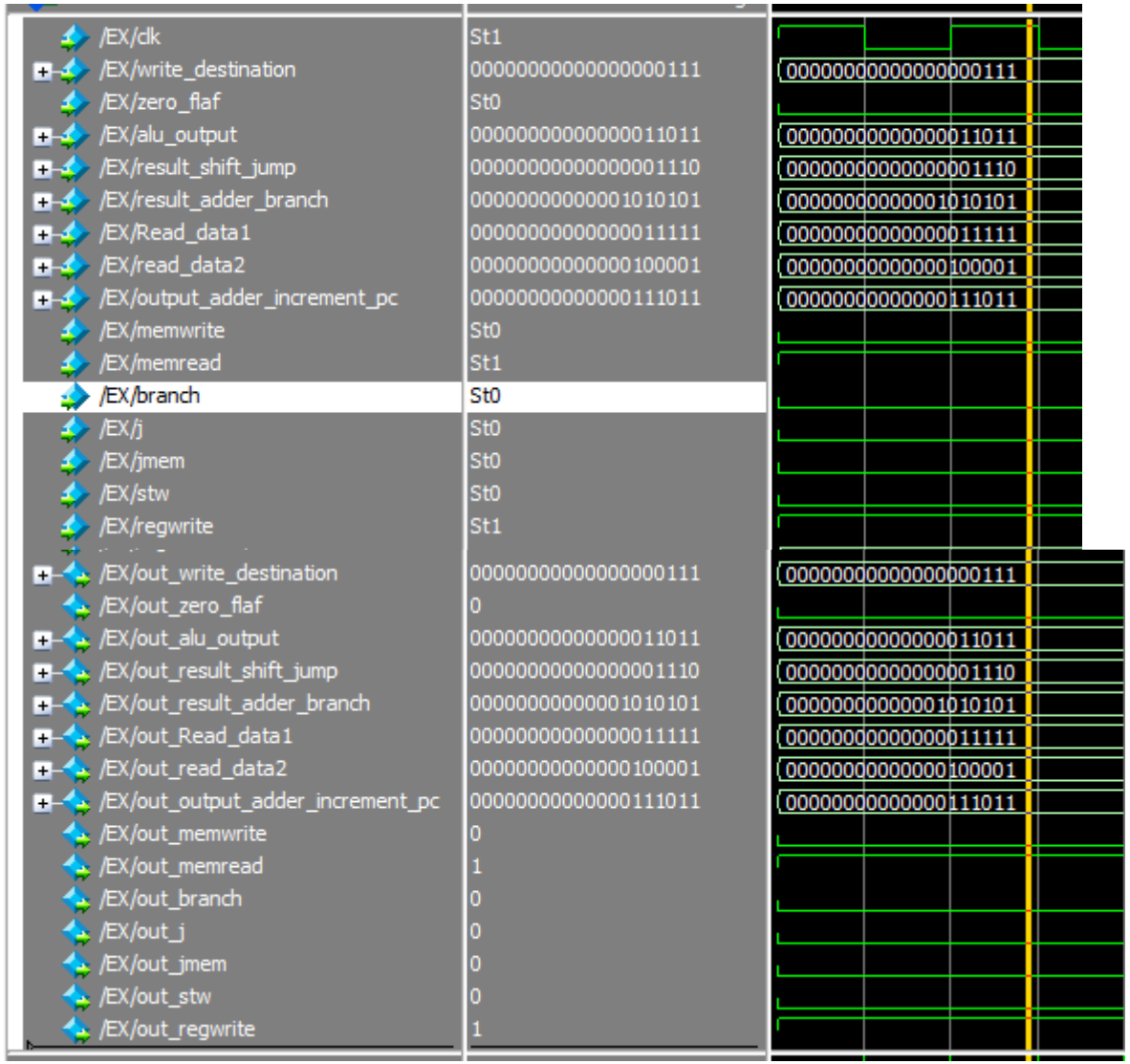
/ID/clk	St1	
/ID/regdest	St1	
/ID/regwrite	St0	
/ID/alusrc	St1	
/ID/memread	St1	
/ID/memwrite	St0	
/ID/memtoreg	St0	
/ID/j	St0	
/ID/branch	St0	
/ID/jmem	St0	
/ID/stw	St0	
/ID/aluop	101	101
/ID/result_shift_jump	00000000000000000001	00000000000000000001
/ID/output_adder_increment_pc	00000000000000000111	00000000000000000111
/ID/read_data1	00000000000000001010	00000000000000001010
/ID/read_data2	00000000000000001111	00000000000000001111
/ID/output_sign_extention	00000001010101010101	00000001010101010101
/ID/instruction	00000010101101010100	00000010101101010100
/ID/out_regdest	1	
/ID/out_regwrite	0	
/ID/out_alusrc	1	
/ID/out_memread	1	
/ID/out_memwrite	0	
/ID/out_memtoreg	0	
/ID/out_j	0	
/ID/out_branch	0	
/ID/out_jmem	0	
/ID/out_stw	0	
/ID/out_aluop	101	101
/ID/out_result_shift_jump	00000000000000000001	00000000000000000001
/ID/out_output_adder_increme...	00000000000000000111	00000000000000000111
/ID/out_read_data1	00000000000000001010	00000000000000001010
/ID/out_read_data2	00000000000000001111	00000000000000001111
/ID/out_output_sign_extention	00000001010101010101	00000001010101010101
/ID/out_instruction	00000010101101010100	00000010101101010100



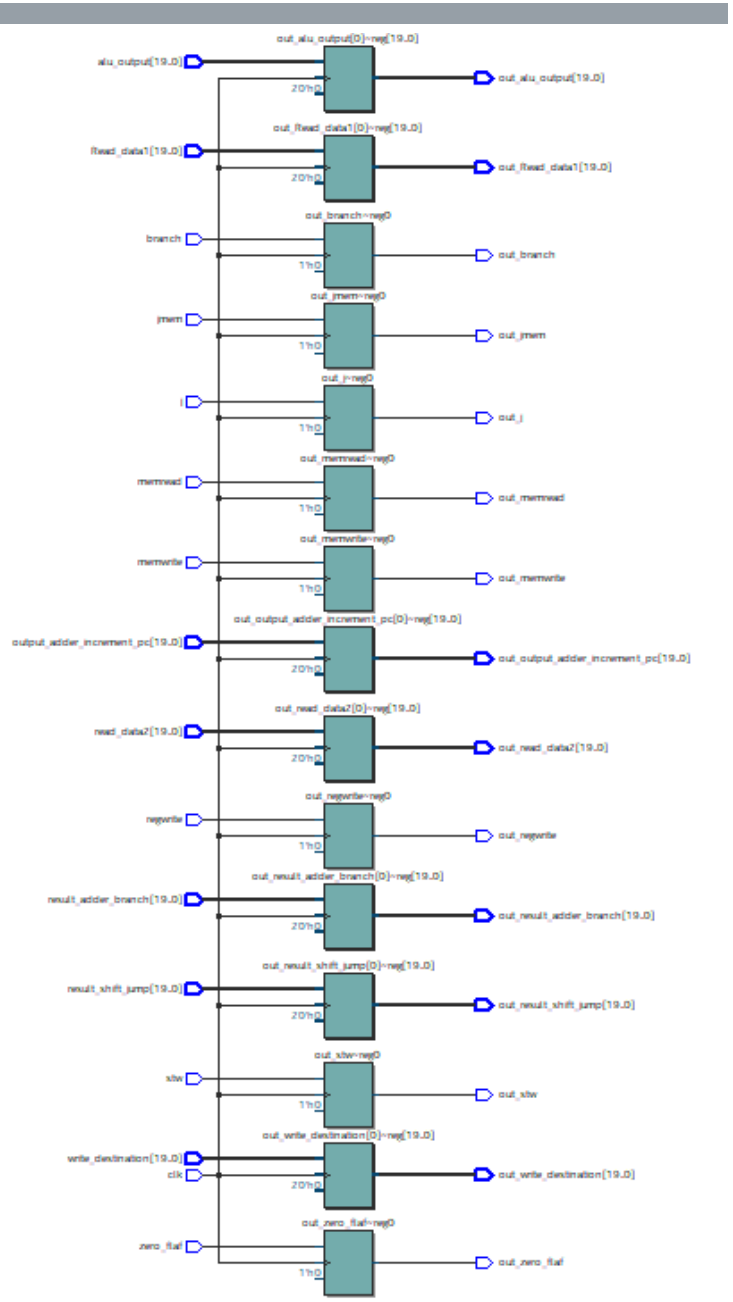
3-EX/MEM

```
module EX ( clk , write_destination , zero_flaf , alu_output ,
result_shift_jump , result_adder_branch , Read_data1 , read_data2 ,
output_adder_increment_pc , memwrite , memread , branch , j , jmem , stw ,
regwrite , out_write_destination , out_zero_flaf , out_alu_output ,
out_result_shift_jump , out_result_adder_branch , out_Read_data1 , out_read_data2 ,
 out_output_adder_increment_pc , out_memwrite , out_memread , out_branch , out_j ,
out_jmem , out_stw , out_regwrite );

input clk ;
input [ 19 : 0 ] write_destination ;
input zero_flaf ;
input [ 19 : 0 ] alu_output ;
input [ 19 : 0 ]  result_shift_jump ;
input [ 19 : 0 ] result_adder_branch ;
input [ 19 : 0 ] Read_data1 ;
input [ 19 : 0 ] read_data2 ;
input [ 19 : 0 ] output_adder_increment_pc ;
input memwrite , memread , branch , j , jmem , stw , regwrite ;

output reg [ 19 : 0 ] out_write_destination ;
output reg out_zero_flaf ;
output reg [ 19 : 0 ] out_alu_output ;
output reg [ 19 : 0 ] out_result_shift_jump ;
output reg [ 19 : 0 ] out_result_adder_branch ;
output reg [ 19 : 0 ] out_Read_data1 ;
output reg [ 19 : 0 ] out_read_data2 ;
output reg [ 19 : 0 ]  out_output_adder_increment_pc ;
output reg out_memwrite , out_memread , out_branch , out_j , out_jmem , out_stw , out_regwrite  ;


always @ ( posedge clk )
begin

  out_write_destination <= write_destination ;
  out_zero_flaf <= zero_flaf ;
  out_alu_output <= alu_output ;
  out_result_shift_jump <= result_shift_jump ;
  out_result_adder_branch <= result_adder_branch ;
  out_Read_data1 <= Read_data1 ;
  out_read_data2 <= read_data2 ;
  out_output_adder_increment_pc <= output_adder_increment_pc ;
  out_memwrite <= memwrite ;
  out_memread <= memread ;
  out_branch <= branch ;
  out_j <= j ;
  out_jmem <= jmem ;
  out_stw <= stw ;
  out_regwrite <= regwrite ;

end


endmodule
```

3-EX/MEM



/EX/clk	St1
/EX/write_destination	00000000000000000111
/EX/zero_flaf	St0
/EX/alu_output	00000000000000011011
/EX/result_shift_jump	00000000000000001110
/EX/result_adder_branch	00000000000001010101
/EX/Read_data1	00000000000000011111
/EX/read_data2	00000000000000100001
/EX/output_adder_increment_pc	00000000000000111011
/EX/memwrite	St0
/EX/memread	St1
/EX/branch	St0
/EX/j	St0
/EX/jmem	St0
/EX/stw	St0
/EX/regwrite	St1
/EX/out_write_destination	00000000000000000111
/EX/out_zero_flaf	0
/EX/out_alu_output	00000000000000011011
/EX/out_result_shift_jump	00000000000000001110
/EX/out_result_adder_branch	00000000000001010101
/EX/out_Read_data1	00000000000000011111
/EX/out_read_data2	00000000000000100001
/EX/out_output_adder_increment_pc	00000000000000111011
/EX/out_memwrite	0
/EX/out_memread	1
/EX/out_branch	0
/EX/out_j	0
/EX/out_jmem	0
/EX/out_stw	0
/EX/out_regwrite	1

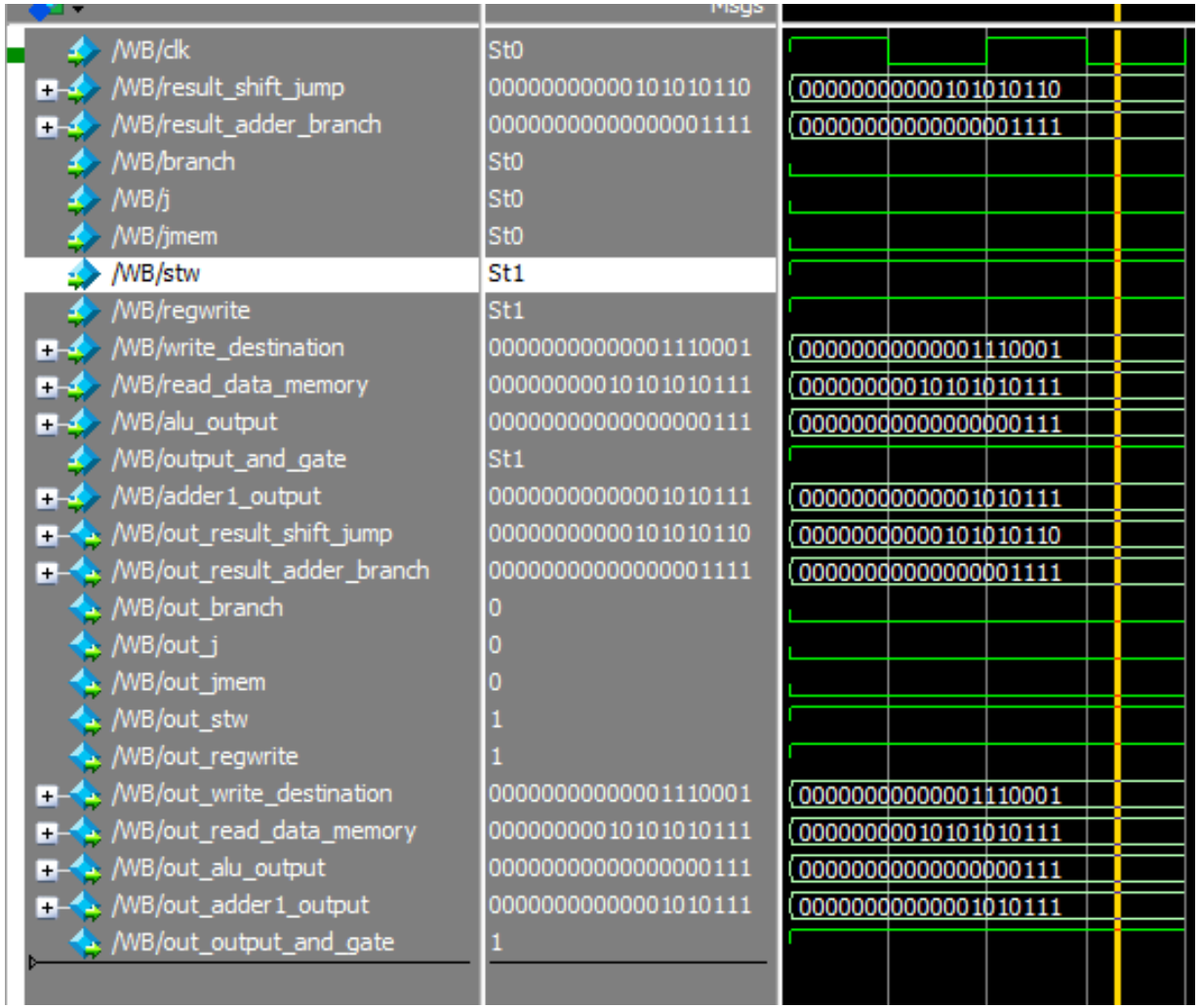


4-MEM/WB

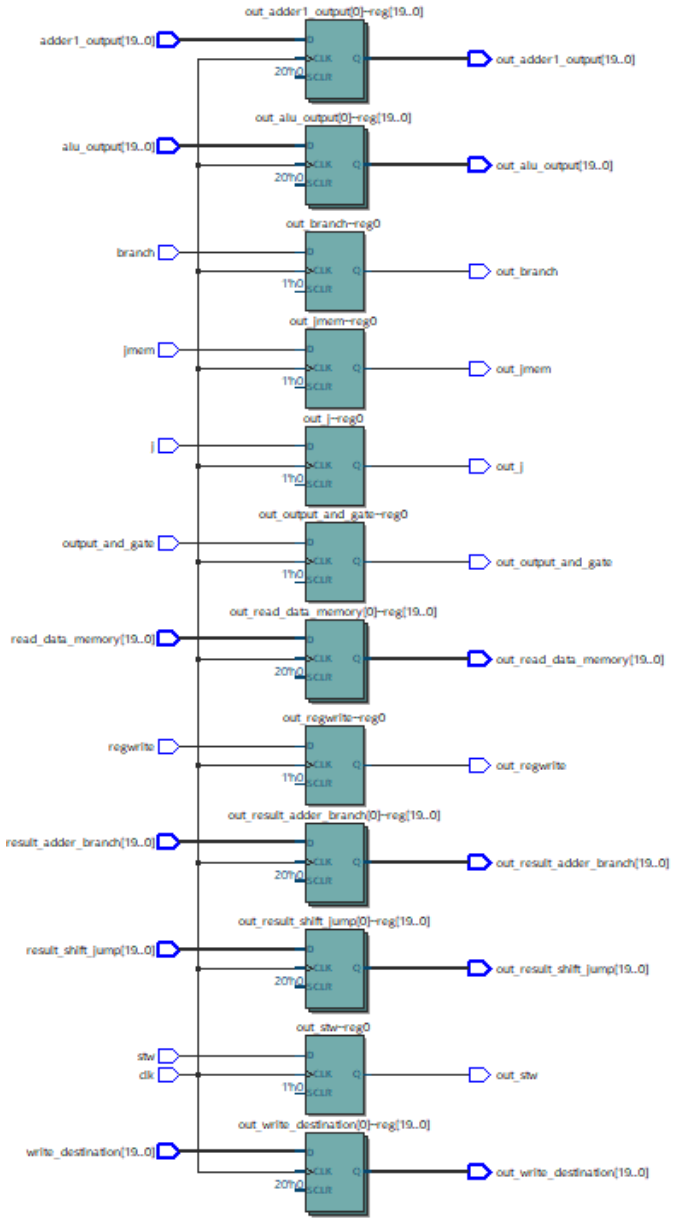
```
module WB ( clk ,result_shift_jump , result_adder_branch ,
 branch , j , jmem , stw ,
 regwrite , write_destination , read_data_memory , alu_output ,
 output_and_gate , out_result_shift_jump ,
out_result_adder_branch ,  out_branch , out_j , out_jmem ,
 out_stw , out_regwrite ,
out_write_destination , out_read_data_memory , out_alu_output ,
 out_output_and_gate , adder1_output , out_adder1_output   ) ;

 input clk ;
 input [ 19 : 0 ] result_shift_jump ;
 input [ 19 : 0 ]  result_adder_branch ;
 input branch , j , jmem , stw , regwrite ;
 input [ 19 : 0 ] write_destination ;
 input [ 19 : 0 ]  read_data_memory ;
 input [ 19 : 0 ]  alu_output ;
 input output_and_gate ;
 input [ 19 : 0 ] adder1_output ;


 output reg [ 19 : 0 ] out_result_shift_jump ;
 output reg [ 19 : 0 ]  out_result_adder_branch ;
 output reg out_branch , out_j , out_jmem , out_stw , out_regwrite ;
 output reg [ 19 : 0 ] out_write_destination ;
 output reg [ 19 : 0 ]  out_read_data_memory ;
 output reg [ 19 : 0 ]  out_alu_output ;
 output reg [ 19 : 0 ] out_adder1_output ;
 output reg out_output_and_gate ;
```

```


 always @ ( posedge clk )
 begin

   out_result_shift_jump <= result_shift_jump ;
   out_result_adder_branch <= result_adder_branch ;
   out_branch <= branch ;
   out_j <= j ;
   out_jmem <= jmem ;
   out_stw <= stw ;
   out_regwrite <= regwrite ;
   out_write_destination <= write_destination ;
   out_read_data_memory <= read_data_memory ;
   out_alu_output <= alu_output ;
   out_output_and_gate <= output_and_gate ;
   out_adder1_output <= adder1_output ;

 end

endmodule
```

4-MEM/WB



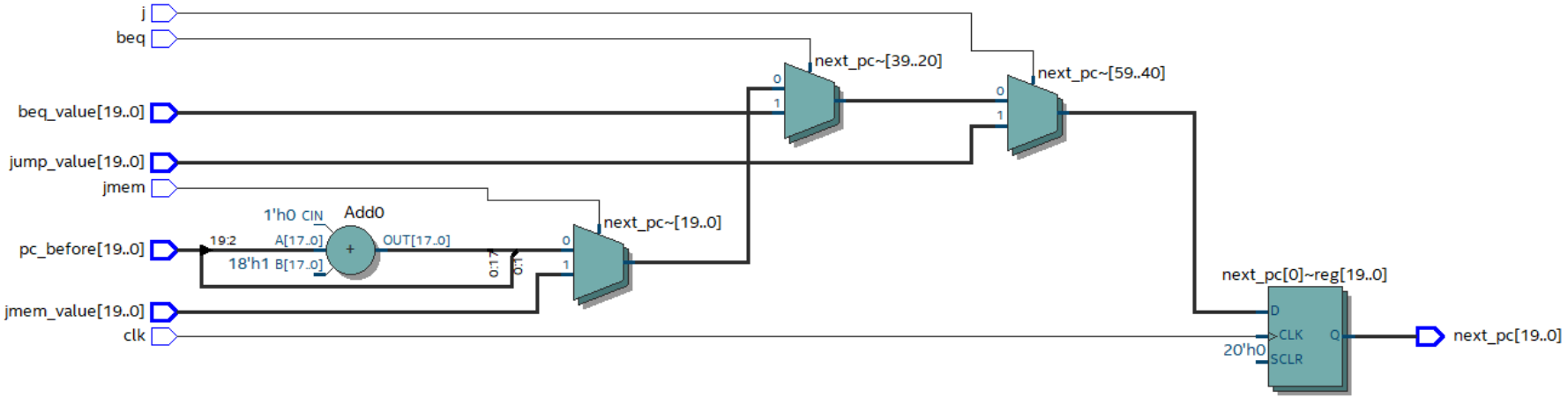
/WB/clk	St0
/WB/result_shift_jump	00000000000101010110
/WB/result_adder_branch	00000000000000001111
/WB/branch	St0
/WB/j	St0
/WB/jmem	St0
/WB/stw	St1
/WB/regwrite	St1
/WB/write_destination	00000000000001110001
/WB/read_data_memory	00000000010101010111
/WB/alu_output	00000000000000000111
/WB/output_and_gate	St1
/WB/adder1_output	00000000000001010111
/WB/out_result_shift_jump	00000000000101010110
/WB/out_result_adder_branch	00000000000000001111
/WB/out_branch	0
/WB/out_j	0
/WB/out_jmem	0
/WB/out_stw	1
/WB/out_regwrite	1
/WB/out_write_destination	00000000000001110001
/WB/out_read_data_memory	00000000010101010111
/WB/out_alu_output	00000000000000000111
/WB/out_adder1_output	00000000000001010111
/WB/out_output_and_gate	1



5-IF CONDITION

D:/FOR ME/Modelsim/project/if_condition.v - Default

```
module if_condition ( clk , j , beq , jmem , jump_value , beq_value , jmem_value , pc_before ,next_pc );

input clk , j , beq , jmem ;
input [19:0] pc_before ;
input [19:0] jump_value ;
input [19:0] beq_value ;
input [19:0] jmem_value ;
output reg [19:0] next_pc ;

 always @ ( posedge clk )
  begin
if ( j == 1'b 1)
begin
  next_pc <= jump_value ;
end
else if ( beq == 1'b 1)
begin
next_pc <= beq_value ;
end
else if ( jmem == 1'b 1)
begin
next_pc <= jmem_value ;
end
else
begin
next_pc <=  pc_before + 3'b 100 ;
end
end

endmodule
```



6- HAZARD UNIT

Condition for Data Hazard

Memory Stage

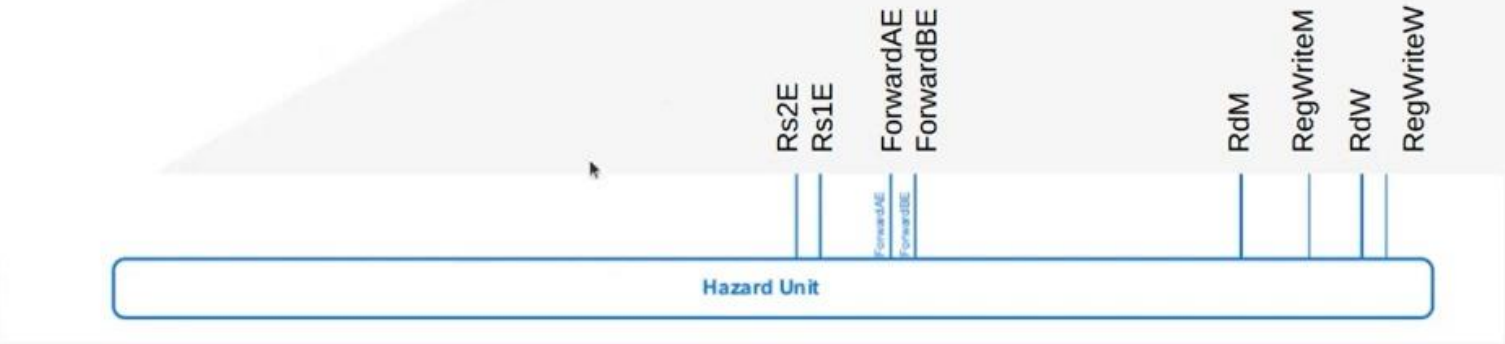
if (RegWriteM and (RdM != 0) and (RdM == Rs1E))
    ForwardAE = 10

if (RegWriteM and (RdM != 0) and (RdM == Rs2E))
    ForwardBE = 10

WriteBack Stage

if (RegWriteW and (RdW != 0) and (RdW == Rs1E))
    ForwardAE = 01

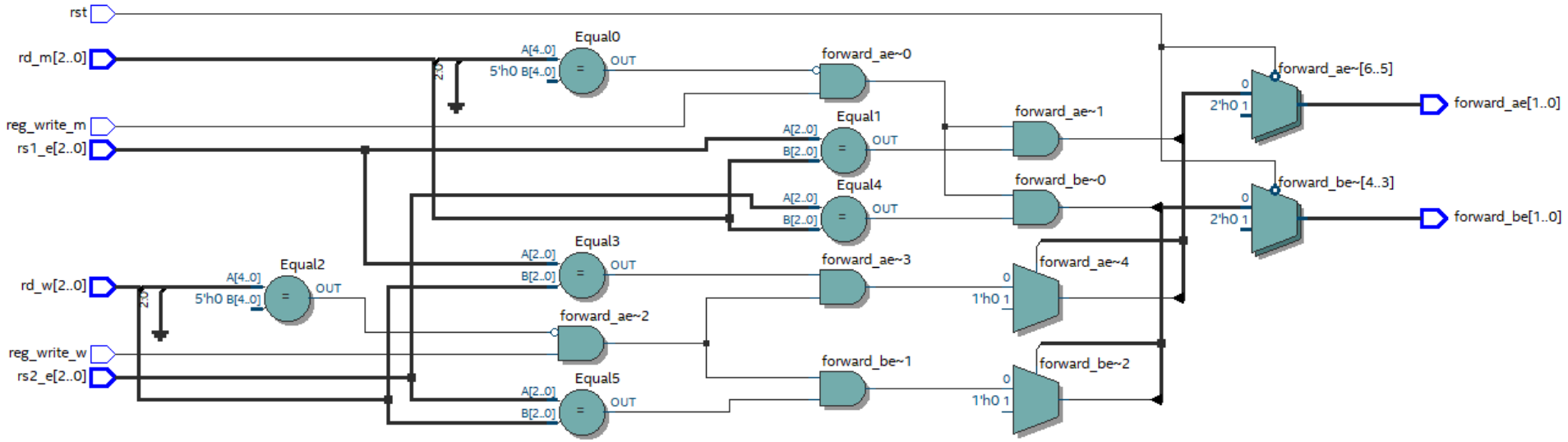
if (RegWriteW and (RdW != 0) and (RdW == Rs2E))
    ForwardBE = 01



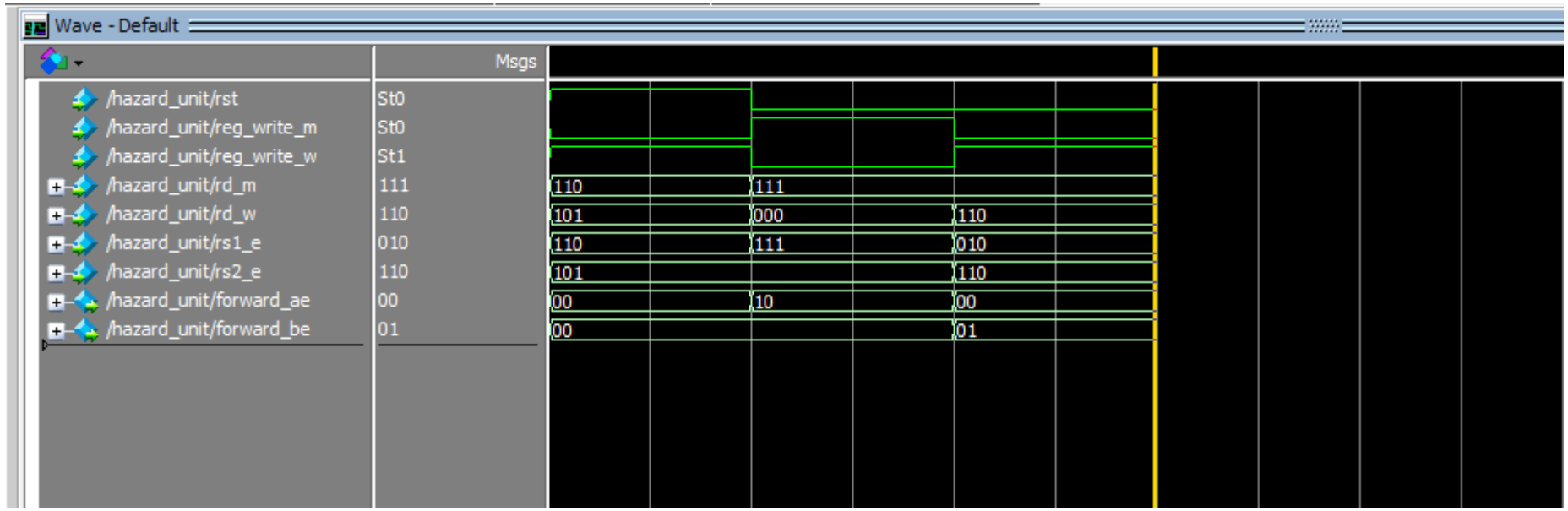
6- HAZARD UNIT

C:/Users/Dell G15/Desktop/hazard_unit.v (/hazard_unit) - Default

```
module hazard_unit (rst, reg_write_m, reg_write_w, rd_m, rd_w, rs1_e, rs2_e, forward_ae, forward_be);

  input rst, reg_write_m, reg_write_w;
  input [2:0] rd_m, rd_w, rs1_e, rs2_e;
  output reg [1:0] forward_ae, forward_be;

  always @(*)
  begin
    forward_ae = (rst == 1'b1) ? 2'b00 :
                 ((reg_write_m == 1'b1) && (rd_m !== 5'h00) && (rd_m == rs1_e)) ? 2'b10 :
                 ((reg_write_w == 1'b1) && (rd_w !== 5'h00) && (rd_w == rs1_e)) ? 2'b01 :
                 2'b00;

    forward_be = (rst == 1'b1) ? 2'b00 :
                 ((reg_write_m == 1'b1) && (rd_m !== 5'h00) && (rd_m == rs2_e)) ? 2'b10 :
                 ((reg_write_w == 1'b1) && (rd_w == 5'h00) && (rd_w == rs2_e)) ? 2'b01 :
                 2'b00;
  end
endmodule
```



6- HAZARD UNIT



7- TOP MODULE

D:/FOR ME/Modelsim/project/pipeline.v - Default

```
module pipeline ( input clk , input rst) ;

wire [19:0] pc_in; //wires into the pc from jumb and branch

wire [19:0] pc_out; //wires out of the pc

wire [19 : 0 ] new_pc ; // used to shoft pc value twice to get pc+4 to help in calc

wire [ 19 : 0 ] adder1_output ; // increment pc with 4

wire [19:0] instruction_wire; //wires out of the imem

wire [19 :0] adder1_output_out1 ; //the output of if/id register of adder1

wire [19:0] instruction_wire_out1 ; //the output of if/id register of imem

wire [19:0] read_data1; //read data 1 from regs file

wire [19:0] read_data2; //read data 2 from regs file

wire [19:0] sign_extension ; // sign extension output value

wire [19:0] alusrc_mux_output;

wire [2:0] stw_mux_output;

wire [19:0]alu_result;

wire zeros ;

wire [2 :0] regdst_mux_output ;

wire [19:0] shift2_output ;

wire [19:0] branch_result ;

wire [ 17 : 0 ] output_shift1 ;  // shift for address [ instruction memory ] ( jump )

wire [17:0] output_shift1_out2 ; // shift for address [ instruction memory ] ( jump ) after being stored in reg

wire [19:0] adder1_output_out2; //the output of if/id register of adder1

wire [19:0] read_data1_out2 ; //read data 1 from regs file

wire [19:0] read_data2_out2 ;//read data 1 from regs file

wire [19:0] sign_extension_out2;// sign extension output value

wire [19:0] instruction_wire_out2 ; //the output of id/ex register of imem

wire [3:0] alu_operation ; // the output of the alu control
```

D:/FOR ME/Modelsim/project/pipeline.v - Default

```
wire [ 2 : 0 ] stw_mux_output_out3 ;

wire zeros_out3 ;

wire [ 19 : 0 ] alu_result_out3 ;

wire [ 17 : 0 ] output_shift1_out3 ;

wire [ 19 : 0 ] branch_result_out3 ;

wire [ 19 : 0 ] read_data1_out3 ;

wire [ 19 : 0 ] read_data2_out3 ;

wire [ 19 : 0 ] adder1_output_out3 ;

wire memwrite1_out3 ;

wire memread1_out3 ;

wire branch1_out3 ;

wire j1_out3 ;

wire jmem1_out3 ;

wire stw1_out3 ;

wire regwrite1_out3 ;

wire out_and ;

wire [ 19 : 0 ] output_read_address_mem ;

wire [ 19 : 0 ] output_write_address_mem ;

wire [ 19 : 0 ] output_write_data_mem ;

wire [ 19 : 0 ] output_read_data_memory ;

wire [ 17 : 0 ] output_shift1_out4 ;

wire [ 19 : 0 ] branch_result_out4 ;

wire  branch1_out4 ;

wire j1_out4 ;

wire jmem1_out4 ;

wire stw1_out4 ;
```

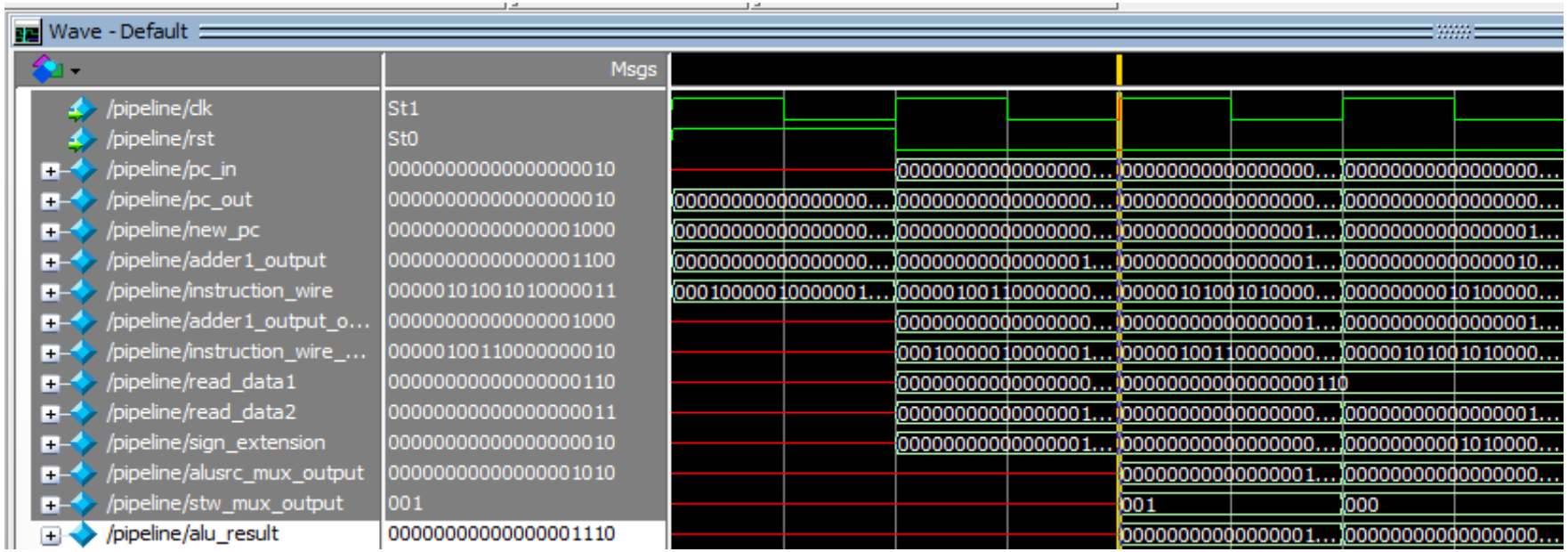
D:/FOR ME/Modelsim/project/pipeline.v - Default

```
wire regwrite1_out4 ;

wire [ 2 : 0 ] stw_mux_output_out4 ;

wire [ 19 : 0 ] output_read_data_memory_out4 ;

wire [ 19 : 0 ] alu_result_out4 ;

wire out_and_out4 ;

wire [ 19 : 0 ] out_write_register ;

wire [ 19 : 0 ] pc_in2 ;

wire [ 19 : 0 ] adder1_output_out4 ;


wire regdest1 , regwrite1 , extd1 , alusrc1 , memread1 , memwrite1 , memtoreg1 , j1 , branch1 , jmem1 , stw1  ;

wire  regdest1_out2 , regwrite1_out2  , alusrc1_out2 , memread1_out2 ,
 memwrite1_out2 , memtoreg1_out2 , j1_out2 , branch1_out2 , jmem1_out2 , stw1_out2 ;

wire [2:0] aluop1 ;

wire [2:0] aluop1_out2 ;

reg [ 19 : 0 ] a ;

initial
begin
  a = 20'b 0000_0000_0000_0000_0100 ;

end


counter_pip PC ( clk , rst , pc_in , pc_out ); //pc

shift_left shift3 ( pc_out , new_pc ) ;

adder ADD1 ( new_pc , a , adder1_output  ) ;

instruction_memory_pipeline imem ( pc_out , instruction_wire );

IF IFID (clk , adder1_output , instruction_wire , adder1_output_out1 , instruction_wire_out1 )  ;

register reg_file ( clk , regwrite1_out4 , instruction_wire_out1 [15:13] , instruction_wire_out1 [12:10] ,
stw_mux_output_out4 , read_data1 , read_data2 , out_write_register ) ;

control_unit CTRL ( instruction_wire_out1 [19:16] , regdest1 , regwrite1 , extd1 , alusrc1 , memread1 ,
 memwrite1 , memtoreg1 , j1 , branch1 , jmem1 , aluop1 , stw1 );

sinexten signext ( extd1 , instruction_wire_out1 [9:0] , sign_extension );
```

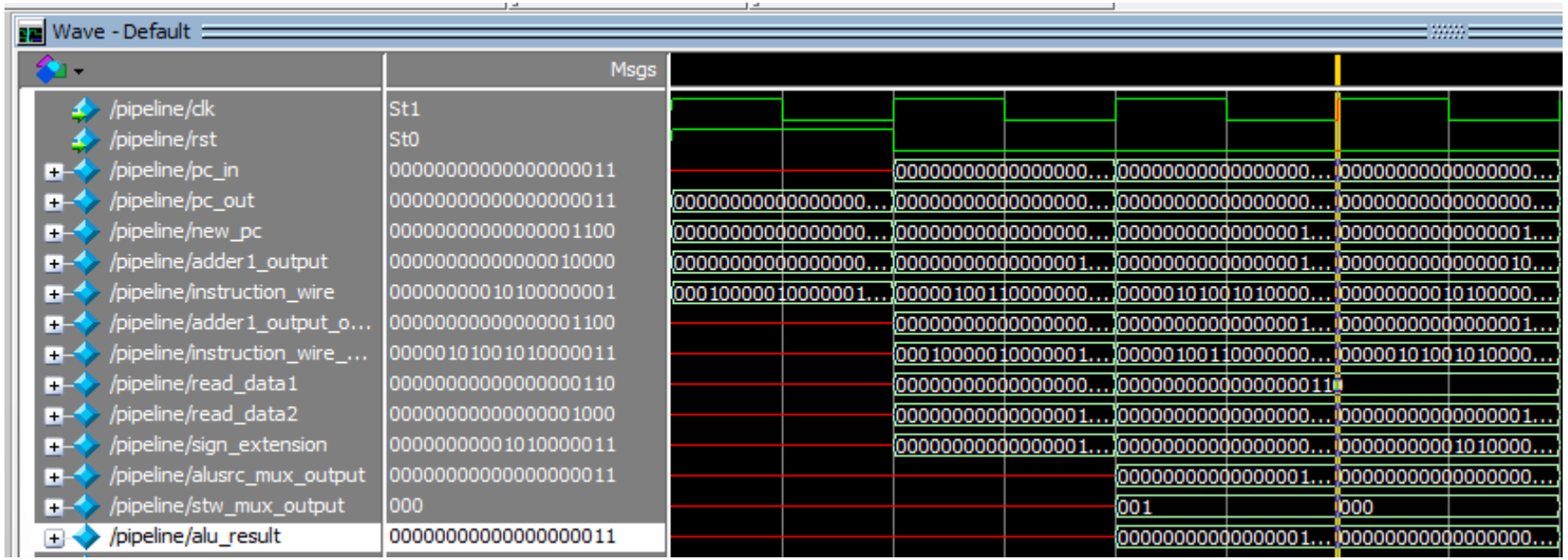
D:/FOR ME/Modelsim/project/pipeline.v - Default

```
 shift_left shift1 ( instruction_wire_out1 [ 15 : 0 ]  , output_shift1 ) ;

ID IDEX ( clk , regdest1 , regwrite1 , alusrc1 , memread1 ,
 memwrite1 , memtoreg1 , j1 , branch1 , jmem1 , aluop1 , stw1 , output_shift1 ,
 adder1_output_out1 , read_data1 , read_data2 , sign_extension , instruction_wire_out1 ,
 regdest1_out2 , regwrite1_out2 , alusrc1_out2 , memread1_out2 ,
 memwrite1_out2 , memtoreg1_out2 , j1_out2 , branch1_out2 , jmem1_out2 , aluop1_out2 , stw1_out2 , output_shift1_out2 ,
 adder1_output_out2 , read_data1_out2 , read_data2_out2 , sign_extension_out2 , instruction_wire_out2  );

alu_control alu_CTRL ( aluop1_out2 , instruction_wire_out2 [3:0] , alu_operation );

MUX alusrc_mux ( read_data2_out2 , sign_extension_out2 , alusrc1_out2 , alusrc_mux_output );

alu ALU ( read_data1_out2 , alusrc_mux_output , alu_operation , alu_result , zeros );

MUX regdst_mux ( instruction_wire_out2 [12:10] , instruction_wire_out2 [9:7] , regdest1_out2 , regdst_mux_output ) ;

MUX stw_mux ( regdst_mux_output , instruction_wire_out1 [15:13] , stw1_out2 , stw_mux_output );

shift_left shift2 ( sign_extension_out2 , shift2_output );

adder adder2 ( shift2_output , adder1_output_out2 , branch_result );

EX EXWB ( clk , stw_mux_output , zeros , alu_result ,
output_shift1_out2 , branch_result , read_data1_out2 , read_data2_out2 ,
adder1_output_out2 , memwrite1_out2 , memread1_out2 , branch1_out2 , j1_out2 , jmem1_out2 , stw1_out2 ,
regwrite1_out2 , stw_mux_output_out3 , zeros_out3 , alu_result_out3 ,
output_shift1_out3 , branch_result_out3 , read_data1_out3 , read_data2_out3 ,
adder1_output_out3 , memwrite1_out3 , memread1_out3 , branch1_out3 , j1_out3 ,
jmem1_out3 , stw1_out3 , regwrite1_out3 );

And_gate branch_select ( zeros_out3 , branch_result_out3 , out_and ) ;

MUX read_address_mem ( alu_result_out3 , read_data2_out3 , jmem1_out3 , output_read_address_mem ) ;

MUX write_address_mem ( alu_result_out3 , read_data1_out3 , stw1_out3 , output_write_address_mem ) ;

MUX write_data_mem ( read_data2_out3 , adder1_output_out3 , jmem1_out3 , output_write_data_mem ) ;

data_memmory data_mem ( clk , output_read_address_mem  ,  output_write_address_mem ,  output_write_data_mem  ,  memwrite1_out3  , memread1_out3 , output_read_data_memory  );

WB wb ( clk , output_shift1_out3 , branch_result_out3 ,  branch1_out3 , j1_out3 , jmem1_out3 , stw1_out3 ,
regwrite1_out3 , stw_mux_output_out3 , output_read_data_memory , alu_result_out3 , out_and , output_shift1_out4 ,
branch_result_out4 ,  branch1_out4 , j1_out4 , jmem1_out4 , stw1_out4 , regwrite1_out4 ,
stw_mux_output_out4 , output_read_data_memory_out4 , alu_result_out4 , out_and_out4 , adder1_output_out3 , adder1_output_out4  ) ;

MUX read_memory ( alu_result_out4  ,  output_read_data_memory_out4  , stw1_out4 , out_write_register ) ;

if_condition pc_if ( clk , j1_out4 , branch1_out4 , jmem1_out4 , output_shift1_out4 , branch_result_out4 , out_write_register , new_pc , pc_in2 ) ;

shift_right pc ( pc_in2 , pc_in ) ;
```

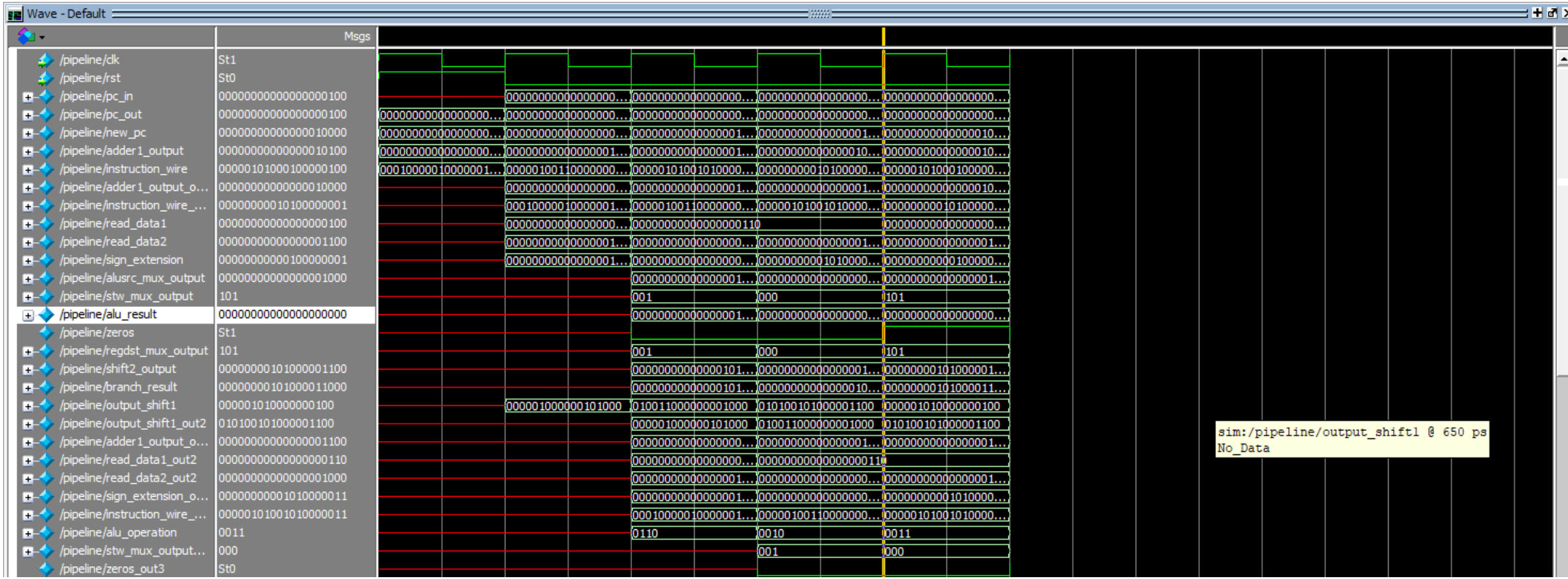
8- ADDI


Wave - Default
Msgs
/pipeline/clk
/pipeline/rst
/pipeline/pc_in
/pipeline/pc_out
/pipeline/new_pc
/pipeline/adder1_output
/pipeline/instruction_wire
/pipeline/adder1_output_o...
/pipeline/instruction_wire_...
/pipeline/read_data1
/pipeline/read_data2
/pipeline/sign_extension
/pipeline/alusrc_mux_output
/pipeline/stw_mux_output
/pipeline/alu_result
St1
St0
00000000000000000010
00000000000000000010
00000000000000001000
00000000000000001100
00000101001010000011
00000000000000001000
00000100110000000010
00000000000000000110
00000000000000000011
00000000000000000010
00000000000000001010
001
00000000000000001110

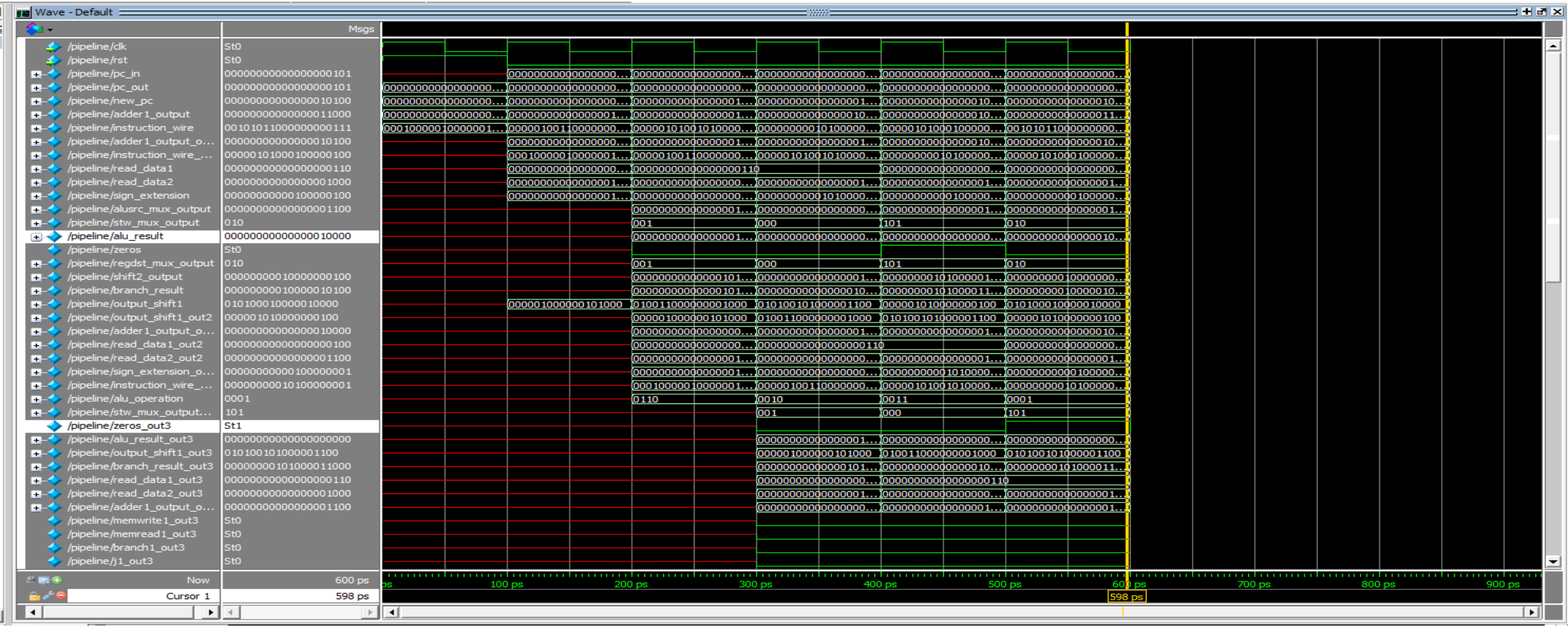
9- SUB


Wave - Default
Msgs
/pipeline/clk
/pipeline/rst
/pipeline/pc_in
/pipeline/pc_out
/pipeline/new_pc
/pipeline/adder1_output
/pipeline/instruction_wire
/pipeline/adder1_output_o...
/pipeline/instruction_wire_...
/pipeline/read_data1
/pipeline/read_data2
/pipeline/sign_extension
/pipeline/alusrc_mux_output
/pipeline/stw_mux_output
/pipeline/alu_result
St1
St0
00000000000000000011
00000000000000000011
00000000000000001100
00000000000000010000
00000000010100000001
00000000000000001100
00000101001010000011
00000000000000000110
00000000000000001000
00000000001010000011
00000000000000000011
000
00000000000000000011
001
000

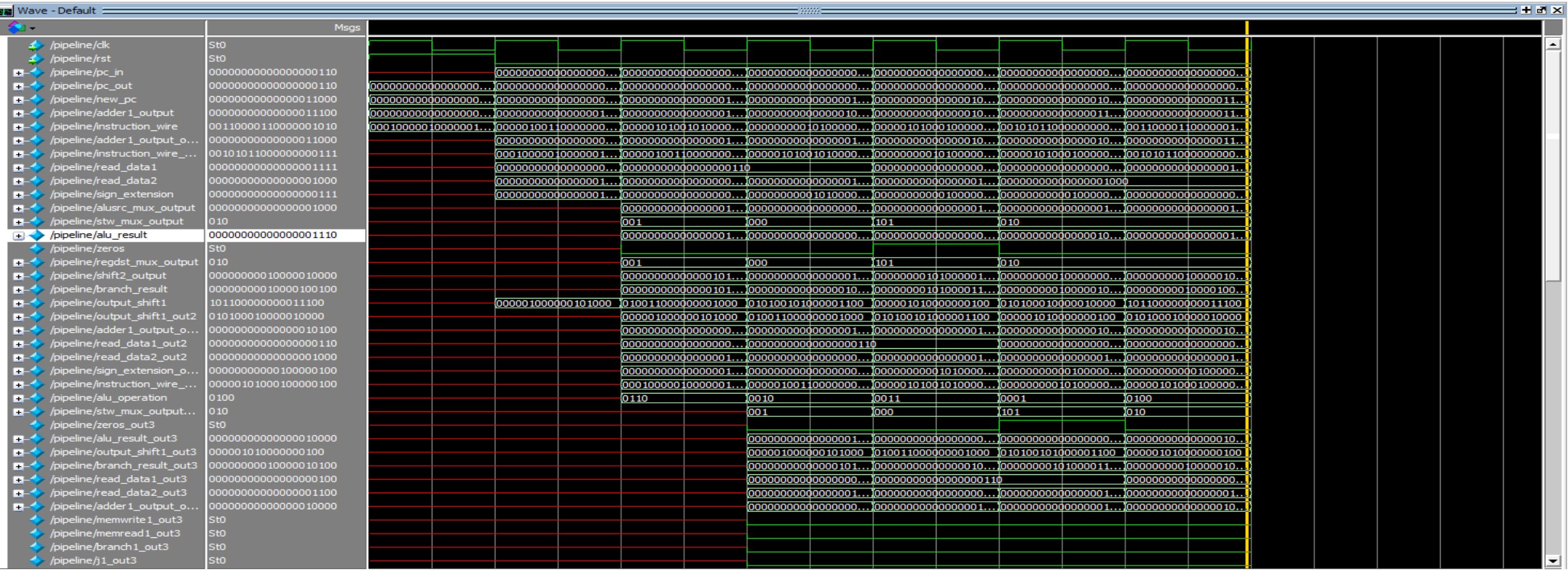
10- AND


Wave - Default
Msgs
/pipeline/clk St1
/pipeline/rst St0
/pipeline/pc_in 00000000000000000100
/pipeline/pc_out 00000000000000000100
/pipeline/new_pc 00000000000000010000
/pipeline/adder1_output 00000000000000010100
/pipeline/instruction_wire 00000101000100000100
/pipeline/adder1_output_o... 00000000000000010000
/pipeline/instruction_wire_... 00000000010100000001
/pipeline/read_data1 00000000000000000100
/pipeline/read_data2 00000000000000001100
/pipeline/sign_extension 00000000000100000001
/pipeline/alusrc_mux_output 00000000000000001000
/pipeline/stw_mux_output 101
/pipeline/alu_result 00000000000000000000
/pipeline/zeros St1
/pipeline/regdst_mux_output 101
/pipeline/shift2_output 00000000101000001100
/pipeline/branch_result 00000000101000011000
/pipeline/output_shift1 00000101000000100
/pipeline/output_shift1_out2 01010010100000 1100
/pipeline/adder1_output_o... 00000000000000001100
/pipeline/read_data1_out2 00000000000000000110
/pipeline/read_data2_out2 00000000000000001000
/pipeline/sign_extension_o... 00000000001010000011
/pipeline/instruction_wire_... 00000101001010000011
/pipeline/alu_operation 0011
/pipeline/stw_mux_output... 000
/pipeline/zeros_out3 St0
sim:/pipeline/output_shift1 @ 650 ps
No_Data

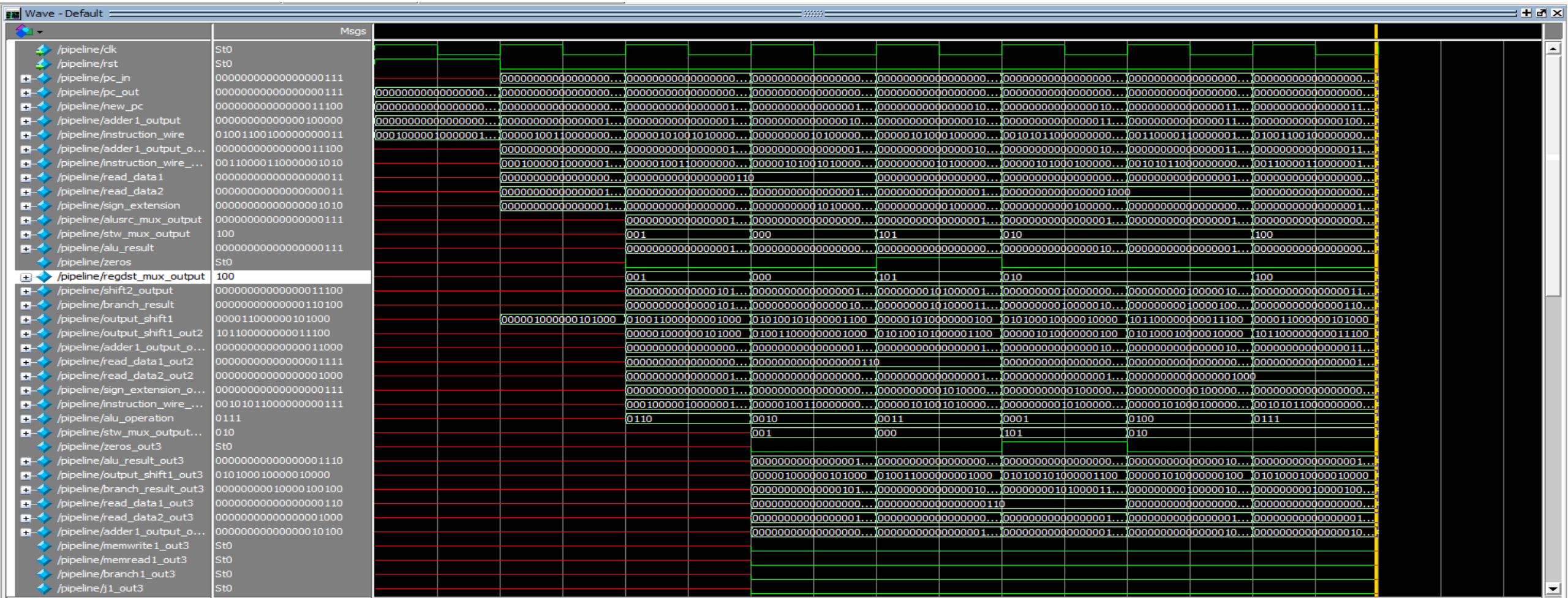
I I- ADD


Wave - Default
Msgs
/pipeline/clk
/pipeline/rst
/pipeline/pc_in
/pipeline/pc_out
/pipeline/new_pc
/pipeline/adder1_output
/pipeline/instruction_wire
/pipeline/adder1_output_o...
/pipeline/instruction_wire_...
/pipeline/read_data1
/pipeline/read_data2
/pipeline/sign_extension
/pipeline/alusrc_mux_output
/pipeline/stw_mux_output
/pipeline/alu_result
/pipeline/zeros
/pipeline/regdst_mux_output
/pipeline/shift2_output
/pipeline/branch_result
/pipeline/output_shift1
/pipeline/output_shift1_out2
/pipeline/adder1_output_o...
/pipeline/read_data1_out2
/pipeline/read_data2_out2
/pipeline/sign_extension_o...
/pipeline/instruction_wire_...
/pipeline/alu_operation
/pipeline/stw_mux_output...
/pipeline/zeros_out3
/pipeline/alu_result_out3
/pipeline/output_shift1_out3
/pipeline/branch_result_out3
/pipeline/read_data1_out3
/pipeline/read_data2_out3
/pipeline/adder1_output_o...
/pipeline/memwrite1_out3
/pipeline/memread1_out3
/pipeline/branch1_out3
/pipeline/j1_out3
Now
600 ps
Cursor 1
598 ps
100 ps
200 ps
300 ps
400 ps
500 ps
700 ps
800 ps
900 ps

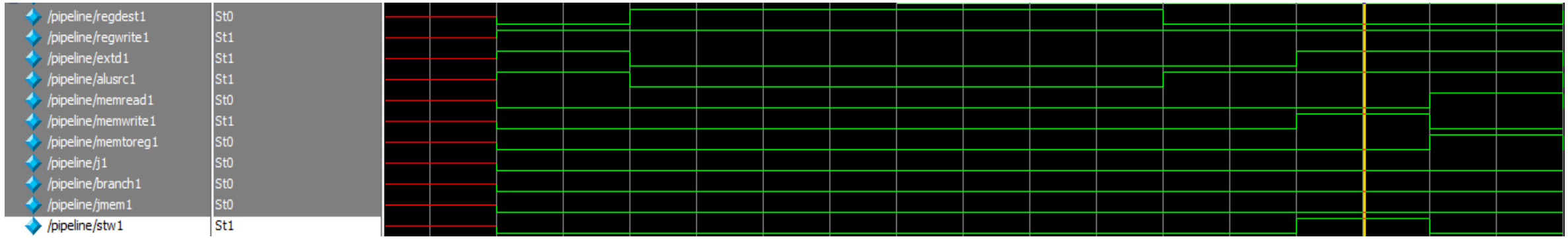
I2- OR

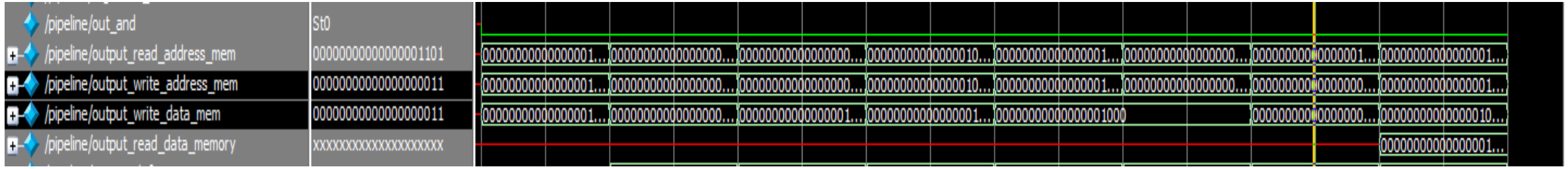

Wave - Default
Msgs
/pipeline/clk
/pipeline/rst
/pipeline/pc_in
/pipeline/pc_out
/pipeline/new_pc
/pipeline/adder1_output
/pipeline/instruction_wire
/pipeline/adder1_output_o...
/pipeline/instruction_wire_...
/pipeline/read_data1
/pipeline/read_data2
/pipeline/sign_extension
/pipeline/alusrc_mux_output
/pipeline/stw_mux_output
/pipeline/alu_result
/pipeline/zeros
/pipeline/regdst_mux_output
/pipeline/shift2_output
/pipeline/branch_result
/pipeline/output_shift1
/pipeline/output_shift1_out2
/pipeline/adder1_output_o...
/pipeline/read_data1_out2
/pipeline/read_data2_out2
/pipeline/sign_extension_o...
/pipeline/instruction_wire_...
/pipeline/alu_operation
/pipeline/stw_mux_output...
/pipeline/zeros_out3
/pipeline/alu_result_out3
/pipeline/output_shift1_out3
/pipeline/branch_result_out3
/pipeline/read_data1_out3
/pipeline/read_data2_out3
/pipeline/adder1_output_o...
/pipeline/memwrite1_out3
/pipeline/memread1_out3
/pipeline/branch1_out3
/pipeline/j1_out3

I3- ANDI

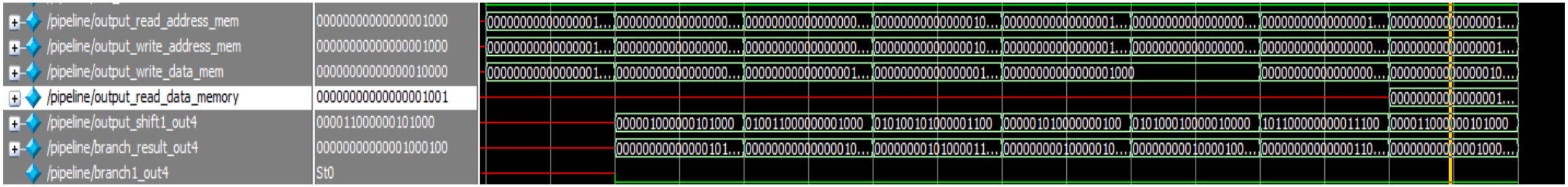

Wave - Default
Msgs
/pipeline/clk St0
/pipeline/rst St0
/pipeline/pc_in 00000000000000000111
/pipeline/pc_out 00000000000000000111
/pipeline/new_pc 00000000000000011100
/pipeline/adder1_output 00000000000000100000
/pipeline/instruction_wire 01001100100000000011
/pipeline/adder1_output_o... 00000000000000011100
/pipeline/instruction_wire_... 00110000110000001010
/pipeline/read_data1 00000000000000000011
/pipeline/read_data2 00000000000000000011
/pipeline/sign_extension 00000000000000001010
/pipeline/alusrc_mux_output 00000000000000000111
/pipeline/stw_mux_output 100
/pipeline/alu_result 00000000000000000111
/pipeline/zeros St0
/pipeline/regdst_mux_output 100
/pipeline/shift2_output 00000000000000011100
/pipeline/branch_result 00000000000000110100
/pipeline/output_shift1 00001100000001010 00
/pipeline/output_shift1_out2 101100000000011100
/pipeline/adder1_output_o... 00000000000000011000
/pipeline/read_data1_out2 00000000000000001111
/pipeline/read_data2_out2 00000000000000001000
/pipeline/sign_extension_o... 00000000000000000111
/pipeline/instruction_wire_... 00101011000000000111
/pipeline/alu_operation 0111
/pipeline/stw_mux_output... 010
/pipeline/zeros_out3 St0
/pipeline/alu_result_out3 00000000000000001110
/pipeline/output_shift1_out3 01010001000010000
/pipeline/branch_result_out3 00000000010000100100
/pipeline/read_data1_out3 00000000000000000110
/pipeline/read_data2_out3 00000000000000001000
/pipeline/adder1_output_o... 00000000000000010100
/pipeline/memwrite1_out3 St0
/pipeline/memread1_out3 St0
/pipeline/branch1_out3 St0
/pipeline/j1_out3 St0

14- STW


/pipeline/regdest1 St0
/pipeline/regwrite1 St1
/pipeline/extd1 St1
/pipeline/alusrc1 St1
/pipeline/memread1 St0
/pipeline/memwrite1 St1
/pipeline/memtoreg1 St0
/pipeline/j1 St0
/pipeline/branch1 St0
/pipeline/jmem1 St0
/pipeline/stw1 St1


/pipeline/out_and St0
/pipeline/output_read_address_mem 00000000000000001101
/pipeline/output_write_address_mem 00000000000000000011
/pipeline/output_write_data_mem 00000000000000000011
/pipeline/output_read_data_memory xxxxxxxxxxxxxxxxxxxx

15- IW



D:/FOR ME/Modelsim/project/register.v (/pipeline/reg_file) - Default

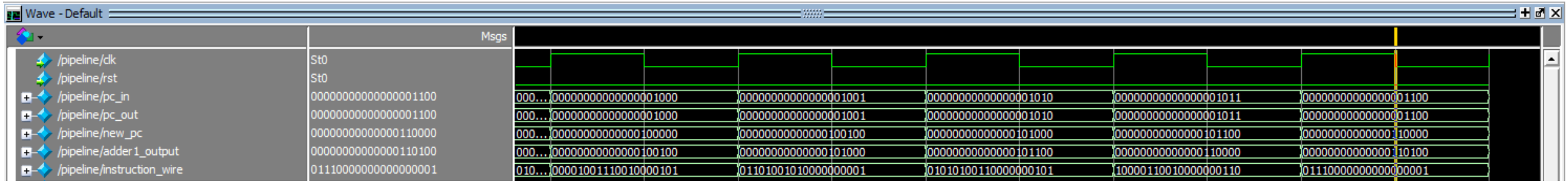
Ln#

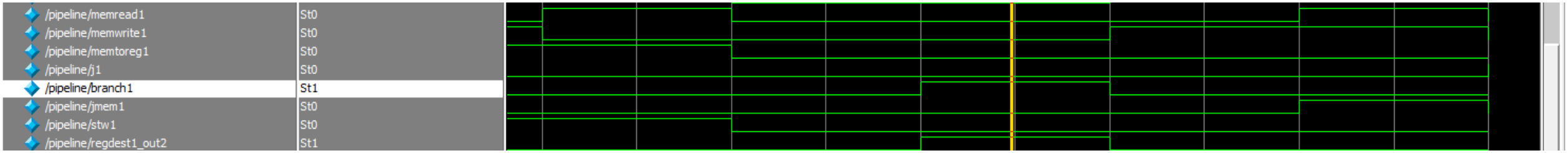
```
1   module register ( clk , reg_write , read_register1 , read_register2 , write_register , read_data1 , read_data2 , write_data ) ;
2     input clk , reg_write ;
3     input [ 2 : 0 ] read_register1 ;
4     input [ 2 : 0 ] read_register2 ;
5     input [ 2 : 0 ] write_register ;
6     output reg [ 19 : 0 ] read_data1 ;
7     output reg [19 : 0 ] read_data2 ;
8     input [ 19 : 0 ] write_data ;
9     reg [ 19 : 0 ] register [ 0 : 7 ] ;
10    initial
11    begin
```

/pipeline/reg_file/write_data
00000000000000001001

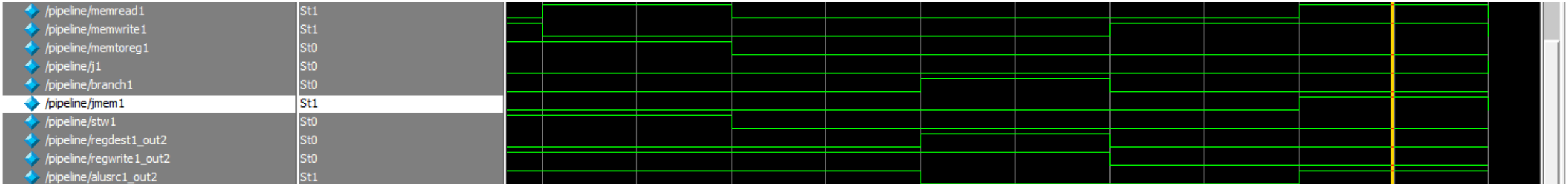
16- BRANCH

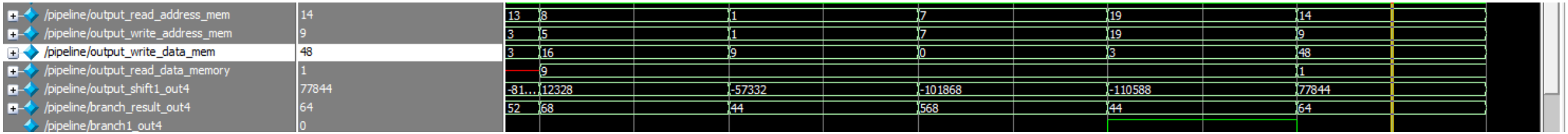
NOT TAKEN

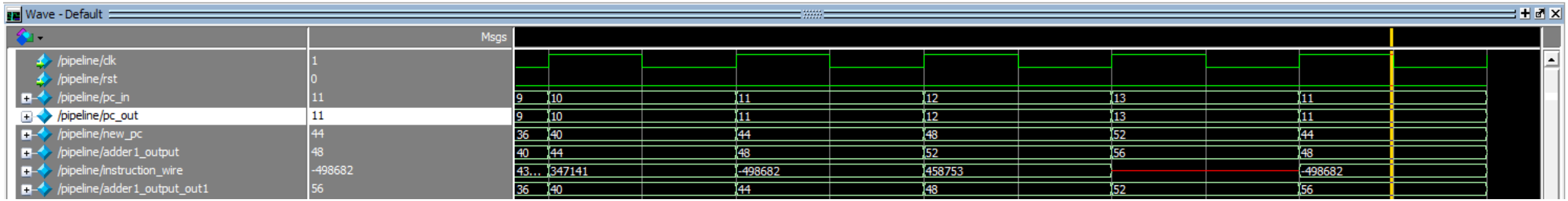




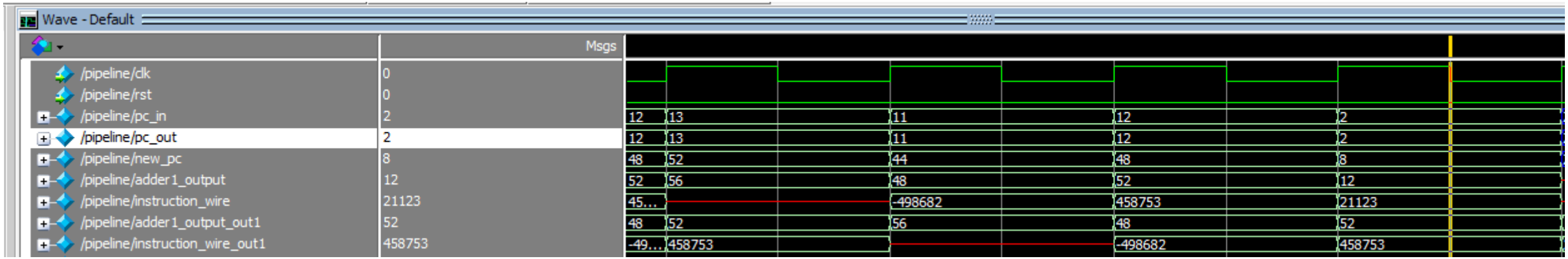
16- JMEM


/pipeline/memread1 St1
/pipeline/memwrite1 St1
/pipeline/memtoreg1 St0
/pipeline/j1 St0
/pipeline/branch1 St0
/pipeline/jmem1 St1
/pipeline/stw1 St0
/pipeline/regdest1_out2 St0
/pipeline/regwrite1_out2 St0
/pipeline/alusrc1_out2 St1


/pipeline/output_read_address_mem 14 13 8 1 7 19 14
/pipeline/output_write_address_mem 9 3 5 1 7 19 9
/pipeline/output_write_data_mem 48 3 16 9 0 3 48
/pipeline/output_read_data_memory 1 9 1
/pipeline/output_shift1_out4 77844 -81... 12328 -57332 -101868 -110588 77844
/pipeline/branch_result_out4 64 52 68 44 568 44 64
/pipeline/branch1_out4 0


Wave - Default
Msgs
/pipeline/clk 1
/pipeline/rst 0
/pipeline/pc_in 11 9 10 11 12 13 11
/pipeline/pc_out 11 9 10 11 12 13 11
/pipeline/new_pc 44 36 40 44 48 52 44
/pipeline/adder1_output 48 40 44 48 52 56 48
/pipeline/instruction_wire -498682 43... 347141 -498682 458753 -498682
/pipeline/adder1_output_out1 56 36 40 44 48 52 56

16- JUMP


Wave - Default
Msgs
/pipeline/clk
/pipeline/rst
/pipeline/pc_in
/pipeline/pc_out
/pipeline/new_pc
/pipeline/adder1_output
/pipeline/instruction_wire
/pipeline/adder1_output_out1
/pipeline/instruction_wire_out1
0
0
2
2
8
12
21123
52
458753
12 13 11 12 2
12 13 11 12 2
48 52 44 48 8
52 56 48 52 12
45... -498682 458753 21123
48 52 56 48 52
-49... 458753 -498682 458753

REFRENCES

- Z.Wang , “CDA3101: Computer Organization II” . https://www.cs.fsu.edu/~zwang/cda3101.html (last accessed 14/5/2023) .

- Elsevier, Interface (the MIPS edition) *5th edition, 2013 (last accessed 4/5/2023)*