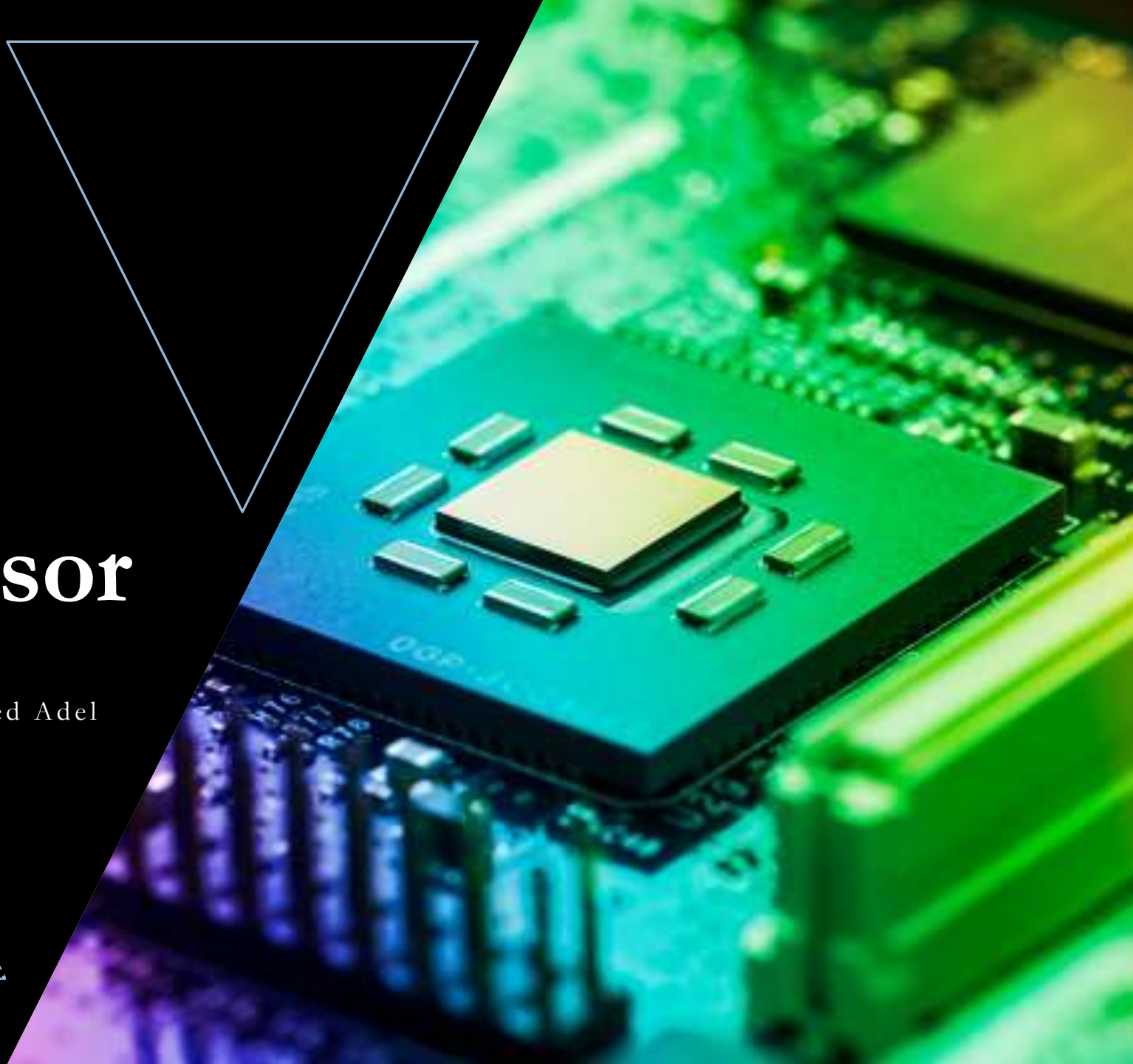
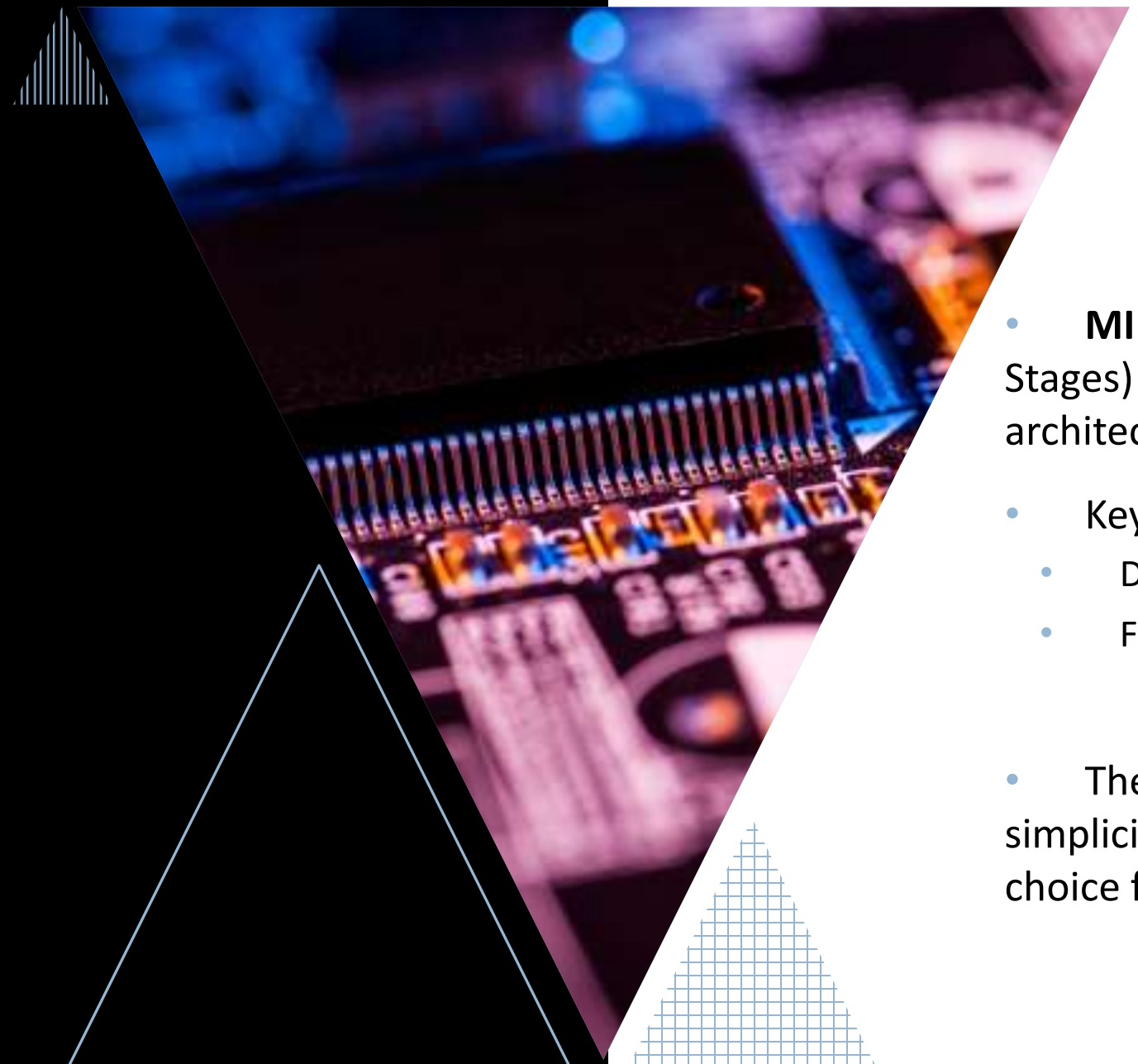
MIPS Processor

Hazem Alaa – Nouran Hamdy – Mohamed Adel

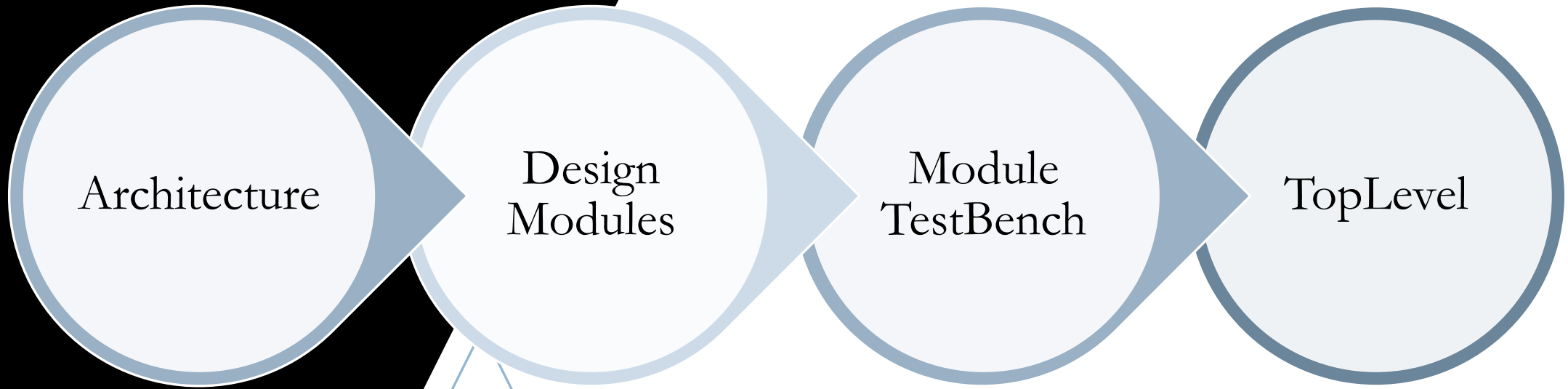


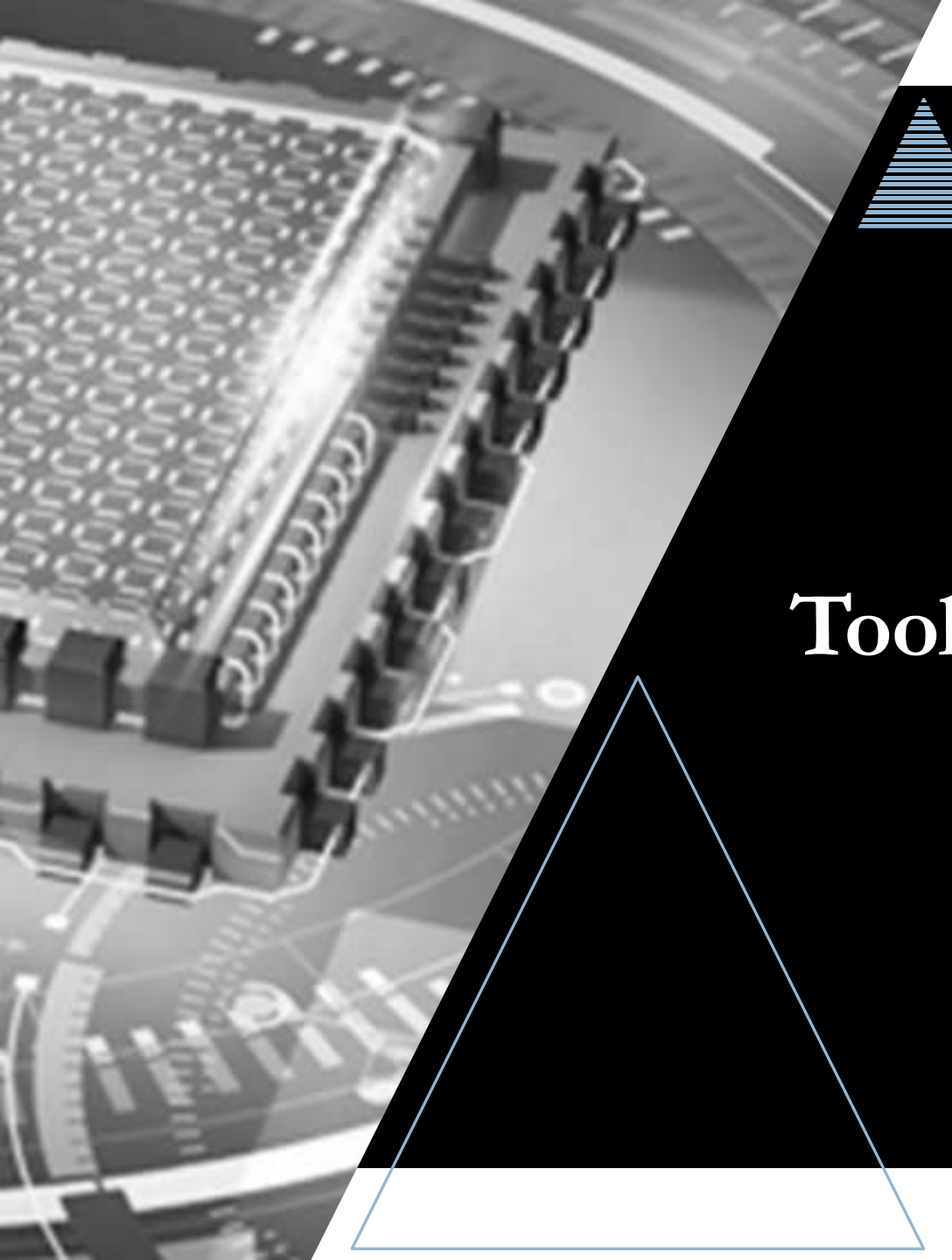


Introduction to MIPS

- **MIPS** (Microprocessor without Interlocked Pipeline Stages) is a reduced instruction set computing (RISC) architecture.
- Key Features:
  - Data Transfer between memory and processor.
  - Fixed-length instructions (32-bits).
- The MIPS architecture has been successful due to its simplicity, performance, and efficiency, making it a popular choice for a wide range of applications.

Design Flow



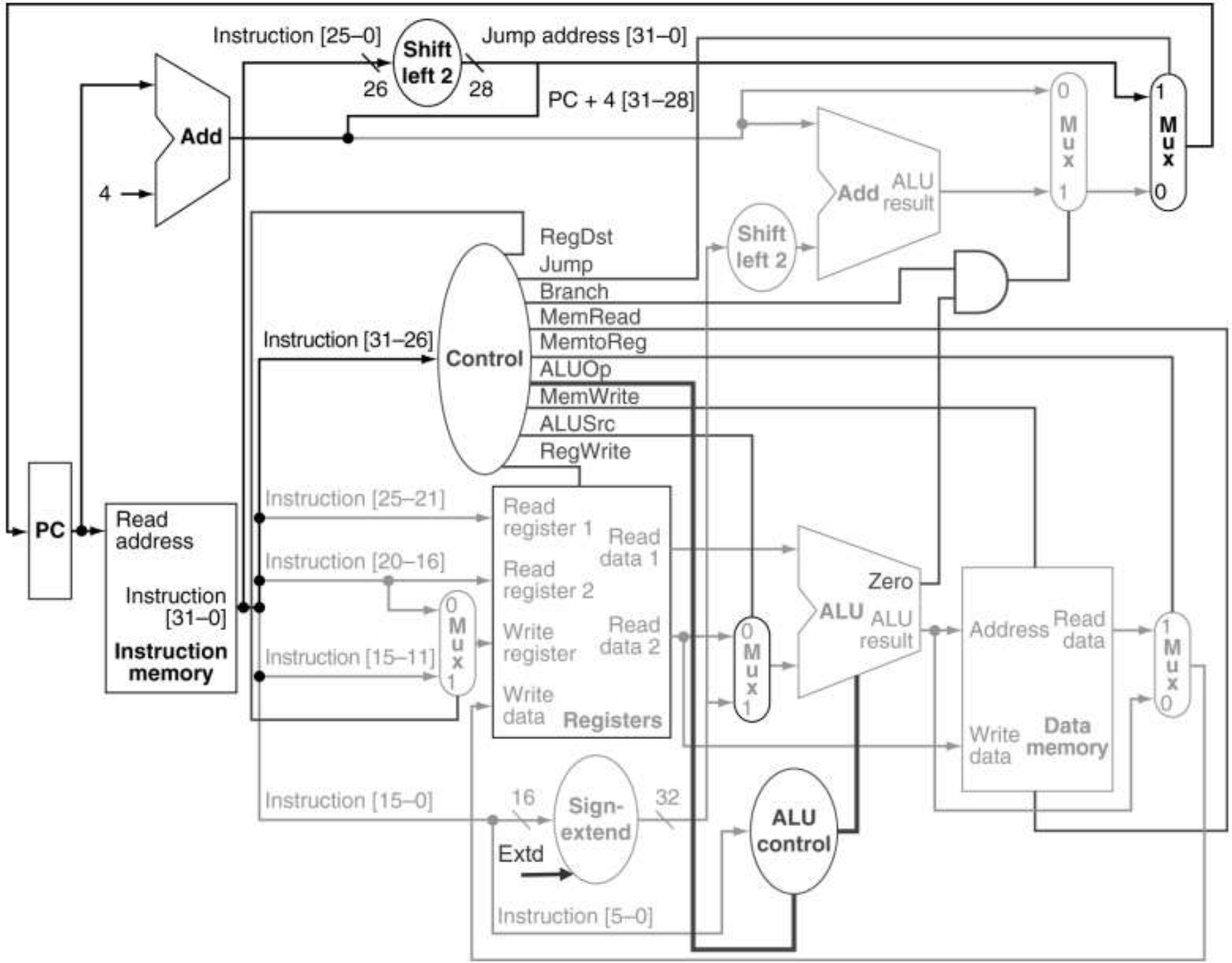


Tools Used

ModelSim

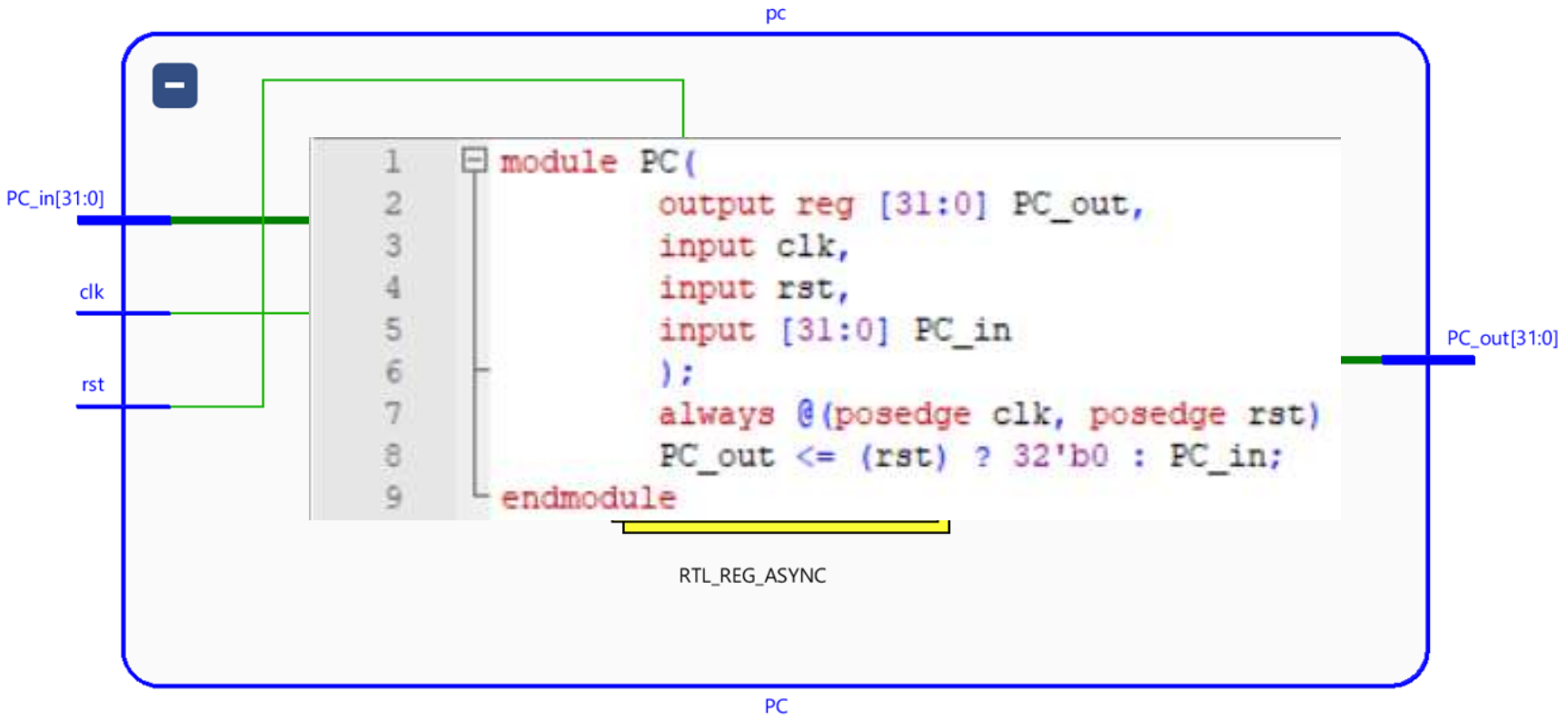
XilinxISE

Vivado2018.3

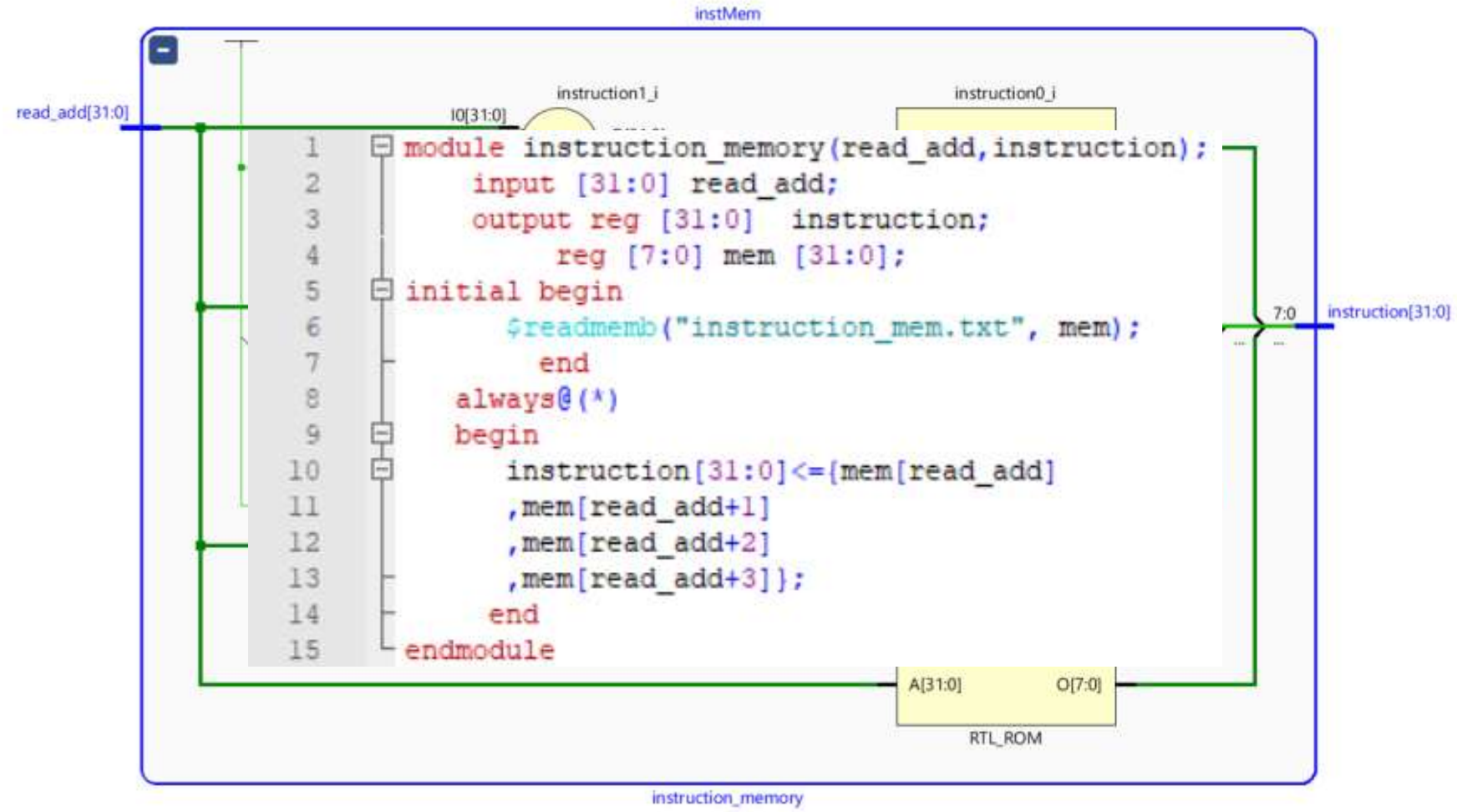

Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
Add
4
0
M
u
x
1
1
M
u
x
0
Add
ALU result
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
Instruction [15–0]
16
Sign-extend
32
Extd
ALU control
Instruction [5–0]

Single Cycle Modules

PC


pc
PC_in[31:0]
clk
rst
PC_out[31:0]
1 module PC(
2 output reg [31:0] PC_out,
3 input clk,
4 input rst,
5 input [31:0] PC_in
6);
7 always @(posedge clk, posedge rst)
8 PC_out <= (rst) ? 32'b0 : PC_in;
9 endmodule
RTL_REG_ASYNC
PC

Instruction Memory

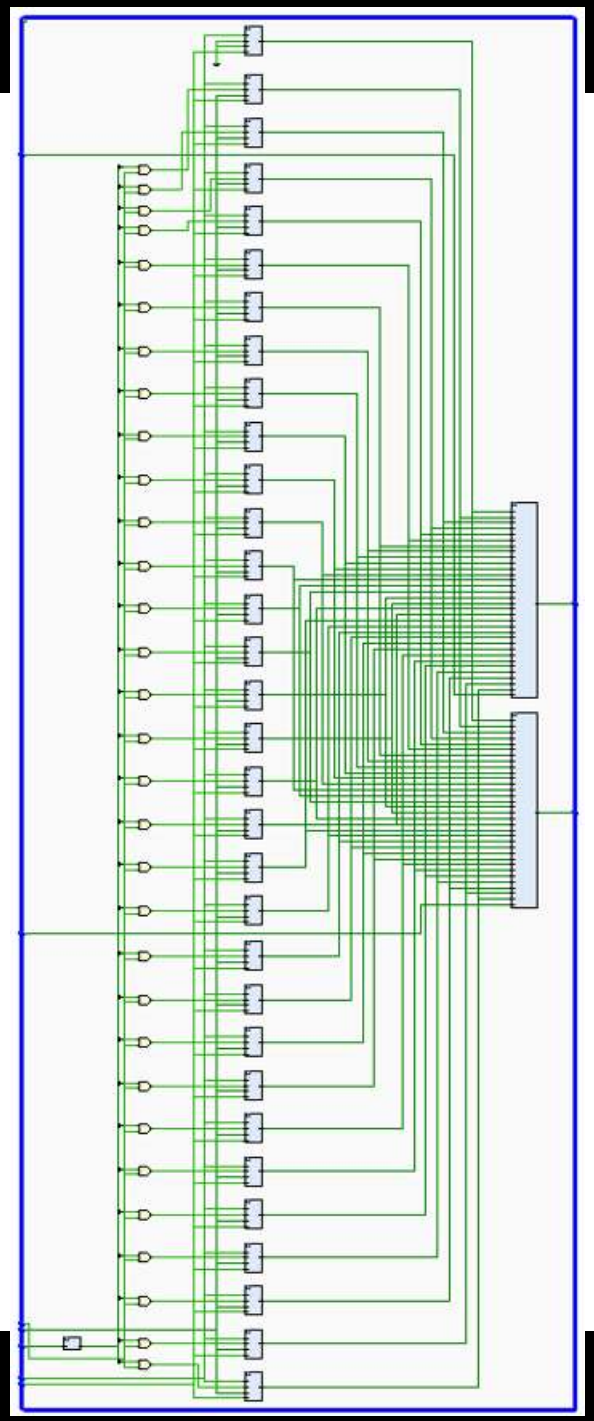



```
module Control(
        output reg RegDst, Jump, Branch, MemRead,
        input [5:0] opcode,
        output reg [1:0] ALUOp );

        always @(opcode) begin
                                RegDst <= 1'b0;
                                Jump <= 1'b0;       //
                                Branch <= 1'b0;
                                MemRead <= 1'b0;
                                MemtoReg <= 1'b0;
                                MemWrite <= 1'b0;
                                ALUSrc <= 1'b0;
                                RegWrite <= 1'b0;
                                ALUOp <= 2'bXX;

        case(opcode)

                6'b000000: //r-controls
                begin
                        RegDst <= 1'b1;
                        Jump <= 1'b0;
                        Branch <= 1'b0;
//                      MemRead <= 1'bz;
                        MemtoReg <= 1'b0;
                        MemWrite <= 1'b0;
                        ALUSrc <= 1'b0;
                        RegWrite <= 1'b1;
                        ALUOp <= 2'b10;
                end
```

```
                6'b001000: //addi-controls                    //
                begin
                        RegDst <= 1'b0;
                        Jump <= 1'b0;                        //
                        Branch <= 1'b0;                      //
                        MemRead <= 1'bz;
                        MemtoReg <= 1'b0;
                        MemWrite <= 1'b0;
                        ALUSrc <= 1'b1;
                        RegWrite <= 1'b1;
                        ALUOp <= 2'b11;
                end

                6'b100011: //lw-controls
                begin
                        RegDst <= 1'b0;
                        Jump <= 1'b0;
                        Branch <= 1'b0;
                        MemRead <= 1'b1;
                        MemtoReg <= 1'b1;
                        MemWrite <= 1'b0;
                        ALUSrc <= 1'b1;
                        RegWrite <= 1'b1;
                        ALUOp <= 2'b00;
                end

                6'b101011: //sw-controls
                begin
                        RegDst <= 1'bz;
                        Jump <= 1'b0;
                        Branch <= 1'b0;
                        MemRead <= 1'bz;
                        MemtoReg <= 1'bz;
                        MemWrite <= 1'b1;
                        ALUSrc <= 1'b1;
                        RegWrite <= 1'b0;
                        ALUOp <= 2'b00;
                end

                6'b000100: //beq-controls
                begin
                        RegDst <= 1'b1;                      //
                        Jump <= 1'b0;
                        Branch <= 1'b1;
                        MemRead <= 1'b0;
                        MemtoReg <= 1'b1;
                        MemWrite <= 1'b0;
                        ALUSrc <= 1'b0;
                        RegWrite <= 1'b0;
                        ALUOp <= 2'b01;
                end

                6'b000010: //j-controls
                        Jump <= 1'b1;
```

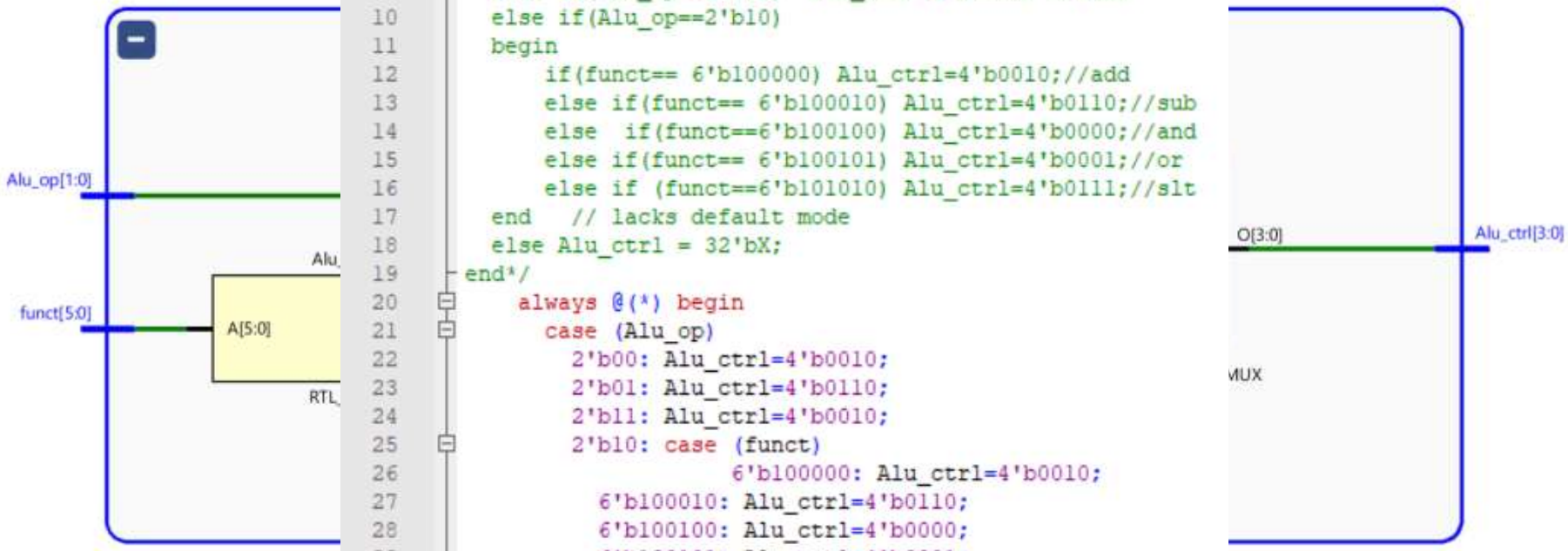
```
                default:
                begin
                        RegDst <= 1'b0;
                        Jump <= 1'b0;
                        Branch <= 1'b0;
                        MemRead <= 1'b0;
                        MemtoReg <= 1'b0;
                        MemWrite <= 1'b0;
                        ALUSrc <= 1'b0;
                        RegWrite <= 1'b0;
                        ALUOp <= 2'bzz;
                end
                endcase
                        end

endmodule
```

Register File



```
1   module RegisterFile(
2       input [4:0] ReadReg1,
3       input [4:0] ReadReg2,
4       input [4:0] WriteReg,
5       input [31:0] WriteData,
6       input RegWrite_ctrl,
7       input clk, rst,
8       output [31:0] ReadData1,
9       output [31:0] ReadData2
10      );
11      wire [31:0] decoderToAnd, andToEnable,
12      regToMux0,regToMux1,regToMux2,regToMux3,
13      regToMux11,regToMux12,regToMux13,regToMu
14      regToMux23,regToMux24,regToMux25,regToMu

16      assign andToEnable[0] = 0;
17      assign andToEnable[1] = decoderToAnd[1] & RegWrite_ctrl;
18      assign andToEnable[2] = decoderToAnd[2] & RegWrite_ctrl;
19      assign andToEnable[3] = decoderToAnd[3] & RegWrite_ctrl;
20      assign andToEnable[4] = decoderToAnd[4] & RegWrite_ctrl;
21      assign andToEnable[5] = decoderToAnd[5] & RegWrite_ctrl;
22      assign andToEnable[6] = decoderToAnd[6] & RegWrite_ctrl;
23      assign andToEnable[7] = decoderToAnd[7] & RegWrite_ctrl;

50      Decoder32_5 Decoder
51      (.decoder_in (WriteReg),
52       .decoder_out(decoderToAnd));
53
54      register32 reg0
55      (.reg_in(0),
56       .enable(andToEnable[0]),
57       .clk(clk), .rst(rst),
58       .reg_out(regToMux0)
59      );
60      register32 reg1
61      (.reg_in(WriteData),
62       .enable(andToEnable[1]),
63       .clk(clk), .rst(rst),
64       .reg_out(regToMux1)
65      );
66      register32 reg2
67      (.reg_in(WriteData),
68       .enable(andToEnable[2]),
69       .clk(clk), .rst(rst),
70       .reg_out(regToMux2)
71      );

247     MUX1024_5 MUX1
248     (.MUXin0(regToMux0),
249      .MUXin1(regToMux1),
250      .MUXin2(regToMux2),
251      .MUXin3(regToMux3),

283     MUX1024_5 MUX2
284     (.MUXin0(regToMux0),
285      .MUXin1(regToMux1),
286      .MUXin2(regToMux2),
287      .MUXin3(regToMux3),
```

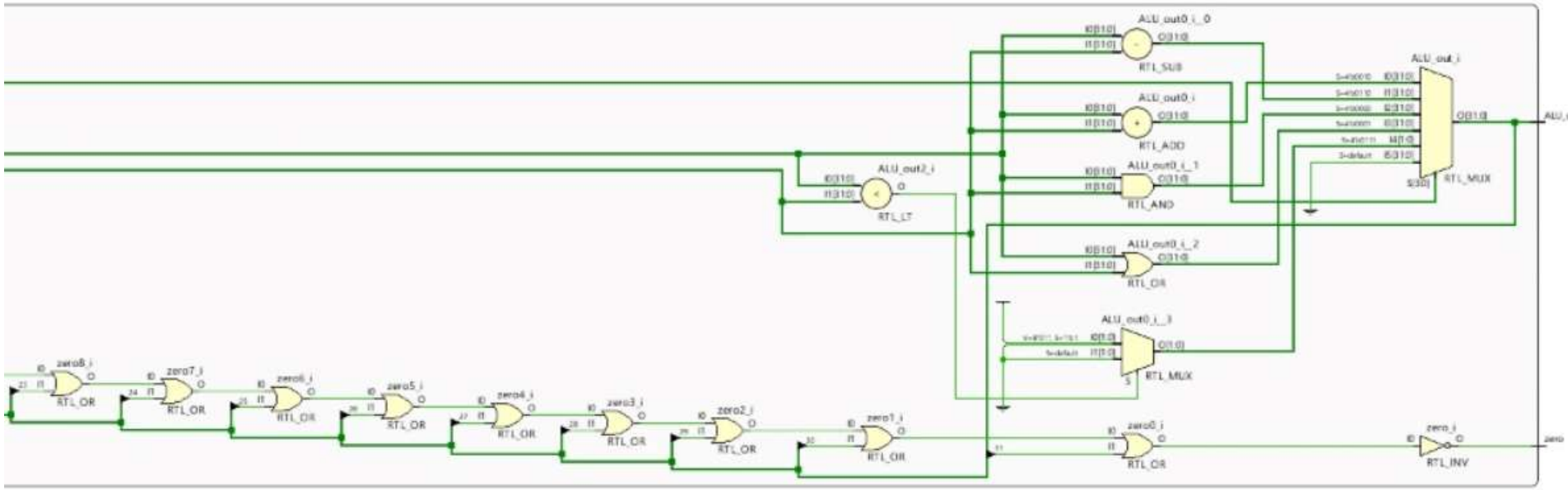
ALU Control Unit




```
1  module Alu_ctrl(Alu_op,funct,Alu_ctrl);
2    input [1:0]Alu_op;
3    input [5:0]funct;
4    output reg[3:0]Alu_ctrl;
5
6    /*always@(*) begin
7    if (Alu_op==2'b00)  Alu_ctrl=4'b0010;  //lw,sw-->add
8    else if(Alu_op==2'b01) Alu_ctrl=4'b0110; //beq-->sub
9    else if(Alu_op==2'b11)  Alu_ctrl=4'b0010; //addi
10   else if(Alu_op==2'b10)
11   begin
12       if(funct== 6'b100000) Alu_ctrl=4'b0010;//add
13       else if(funct== 6'b100010) Alu_ctrl=4'b0110;//sub
14       else  if(funct==6'b100100) Alu_ctrl=4'b0000;//and
15       else if(funct== 6'b100101) Alu_ctrl=4'b0001;//or
16       else if (funct==6'b101010) Alu_ctrl=4'b0111;//slt
17   end   // lacks default mode
18   else Alu_ctrl = 32'bX;
19 end*/
20     always @(*) begin
21       case (Alu_op)
22         2'b00: Alu_ctrl=4'b0010;
23         2'b01: Alu_ctrl=4'b0110;
24         2'b11: Alu_ctrl=4'b0010;
25         2'b10: case (funct)
26                     6'b100000: Alu_ctrl=4'b0010;
27           6'b100010: Alu_ctrl=4'b0110;
28           6'b100100: Alu_ctrl=4'b0000;
29           6'b100101: Alu_ctrl=4'b0001;
30           6'b101010: Alu_ctrl=4'b0111;
31           default: Alu_ctrl=4'bXXXX;
32         endcase
33         default: Alu_ctrl=4'bXXXX;
34       endcase
35     end
36
37 endmodule
```

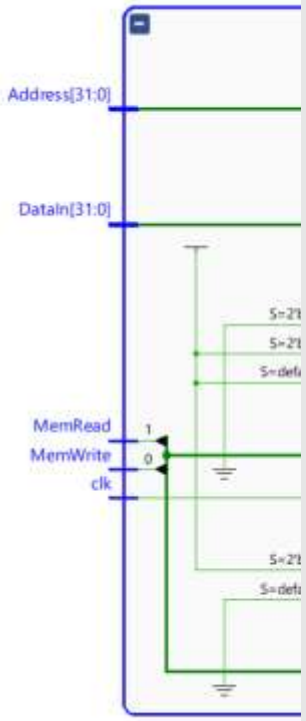
ALU

```
1   module ArithLogicUnit(
2       input wire [31:0] Input1,
3       input wire [31:0] Input2,
```

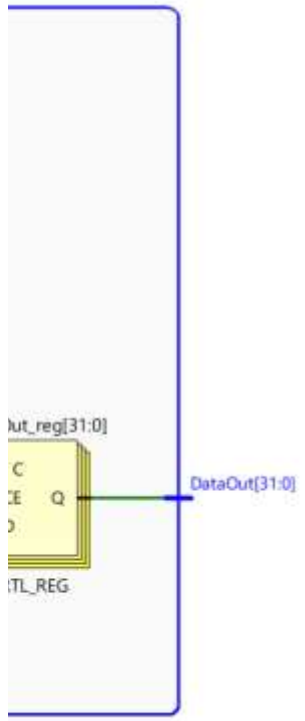



```
24
25
26  endmodule
```

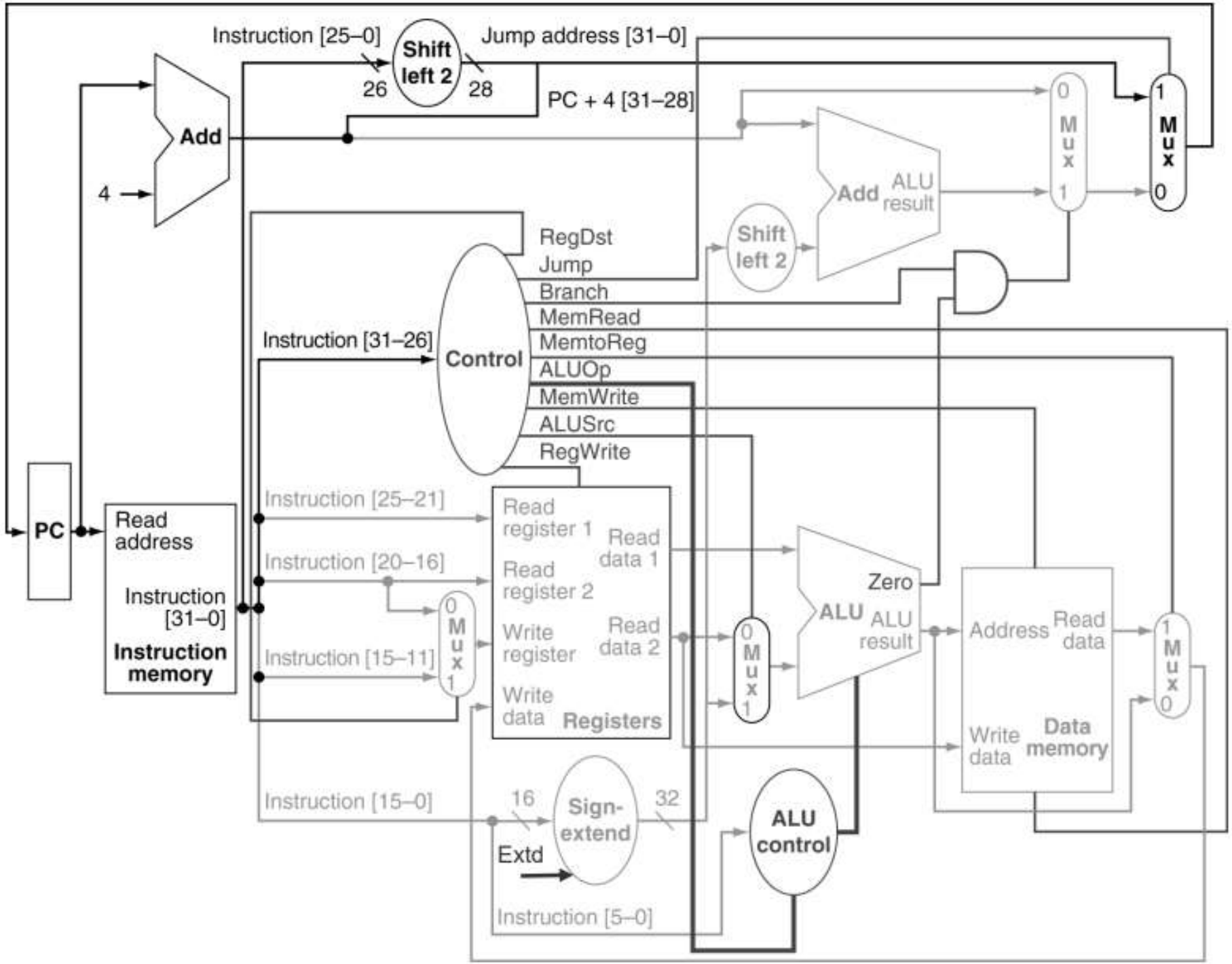
Data Memory




```
1   module DataMemory(
2       input [31:0] DataIn, Address,
3       input MemRead, MemWrite, clk,
4       output reg [31:0] DataOut
5       );
6       reg [31:0] mem [63:0];
7       always @ (negedge clk)
8       case ({MemRead, MemWrite})
9       2'b10  : DataOut <= mem[Address >> 2];
10      2'b01  : mem [Address >> 2] <= DataIn;
11      default:begin
12                mem [Address >> 2] <= mem[Address >> 2];
13                DataOut <= 32'hXXXX_XXXX;
14              end
15      endcase
16
17
18  endmodule
```



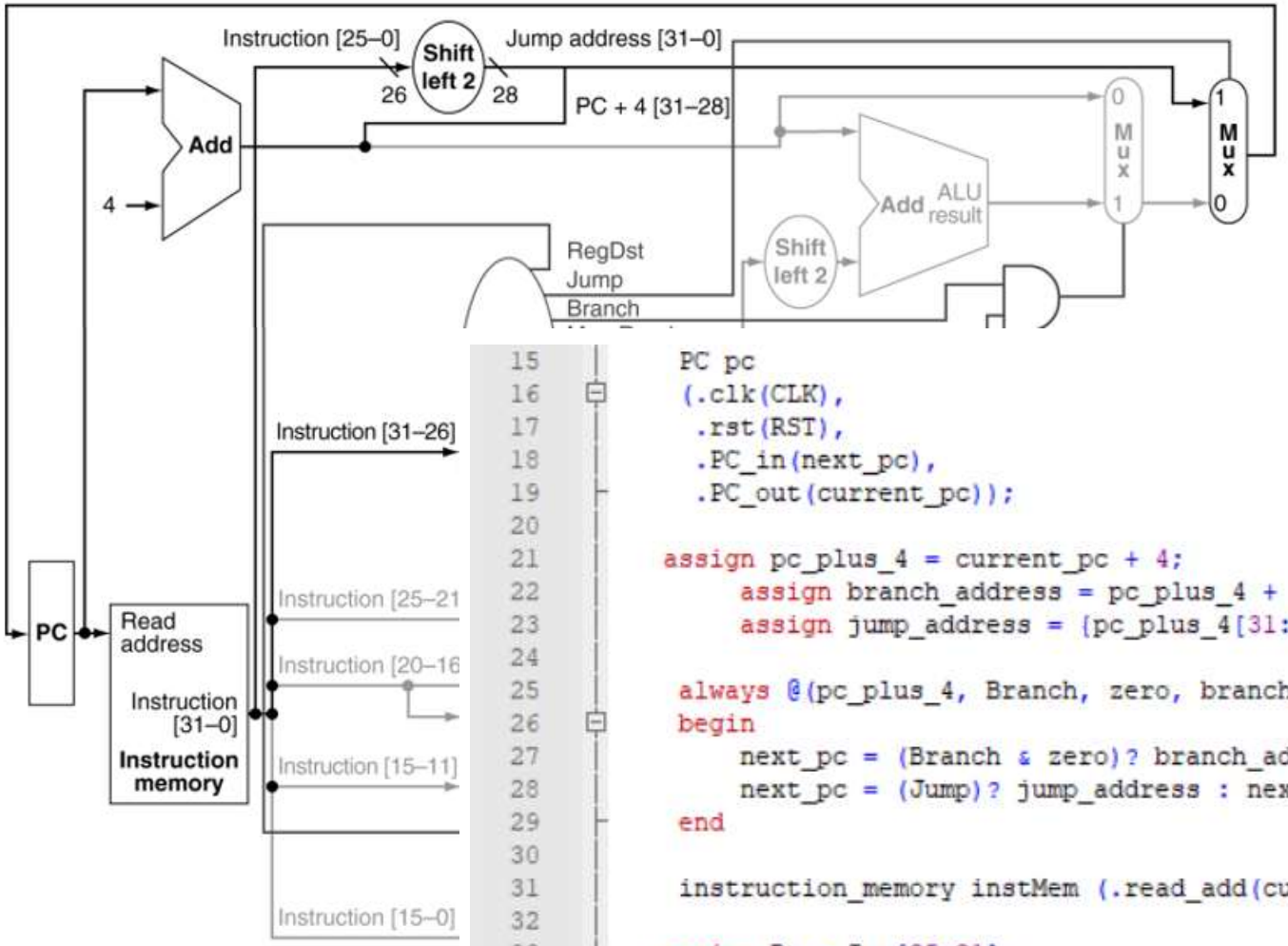
Top Level Module


Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
Add
4
0
M
u
x
1
1
M
u
x
0
Add
ALU result
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
Instruction [15–0]
16
Sign-extend
32
Extd
ALU control
Instruction [5–0]

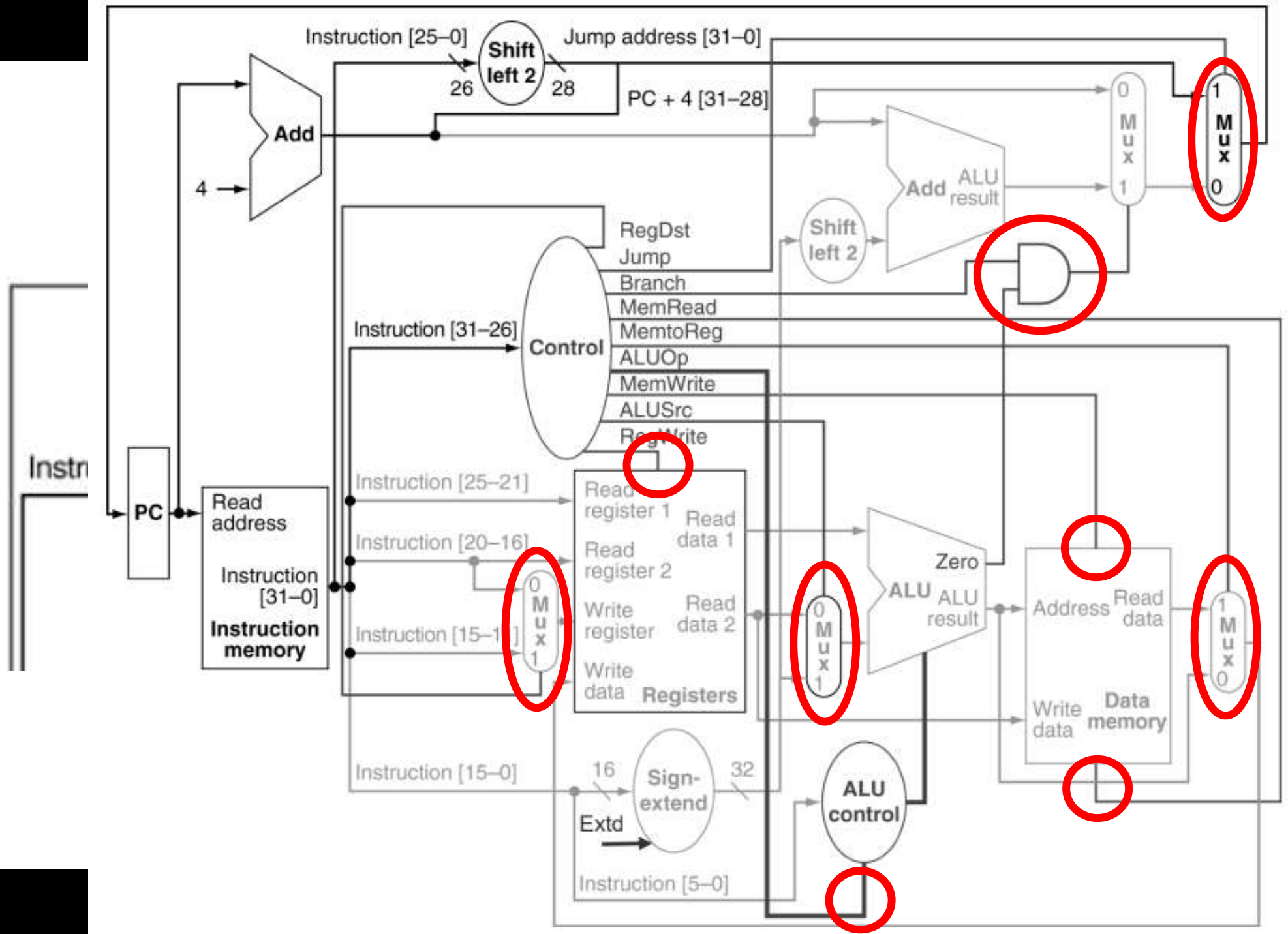
Inputs, Wires, Outputs Initialization

```
1   module MIPS_topmodule(input CLK, RST);
2
3       wire [31:0] current_pc, pc_plus_4, branch_address, jump_address, Ins;
4       reg [31:0] ImmExt, next_pc, alu_in2, WriteData;
5       wire [31:0] instruction, alu_in1, ReadData2, alu_result, ReadData_Mem;
6       wire [25:0] dirJump_address;
7       wire [15:0] Imm;
8       wire Branch, zero, RegWrite, ALUSrc, MemRead, MemWrite, MemtoReg, RegDst, Jump;
9       wire [5:0] funct, CUopcode;
10      wire [4:0] Rs, Rt, Rd;
11      reg [4:0] WriteReg;
12      wire [3:0] ALUopcode;
13      wire [1:0] ALUop;
14
```

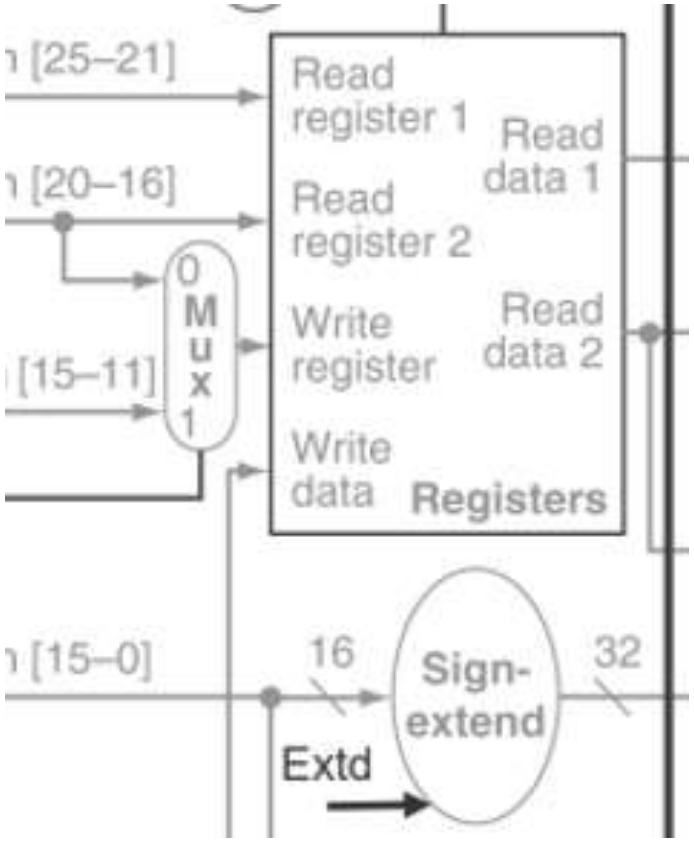
PC
Branch & Jump
Instruction Memory




```
15      PC pc
16      (.clk(CLK),
17       .rst(RST),
18       .PC_in(next_pc),
19       .PC_out(current_pc));
20
21    assign pc_plus_4 = current_pc + 4;
22        assign branch_address = pc_plus_4 + (ImmExt << 2);
23        assign jump_address = {pc_plus_4[31:28], dirJump_address, 2'b00};
24
25     always @(pc_plus_4, Branch, zero, branch_address, jump_address, next_pc, Jump)
26     begin
27        next_pc = (Branch & zero)? branch_address : pc_plus_4;
28        next_pc = (Jump)? jump_address : next_pc;
29     end
30
31     instruction_memory instMem (.read_add(current_pc), .instruction(Ins));
32
33    assign Rs = Ins[25:21];
34        assign Rt = Ins[20:16];
35        assign Rd = Ins[15:11];
36        assign Imm = Ins[15:0];
37    assign funct = Ins[5:0];
38    assign CUopcode = Ins[31:26];
39        assign dirJump_address = Ins[25:0];
```

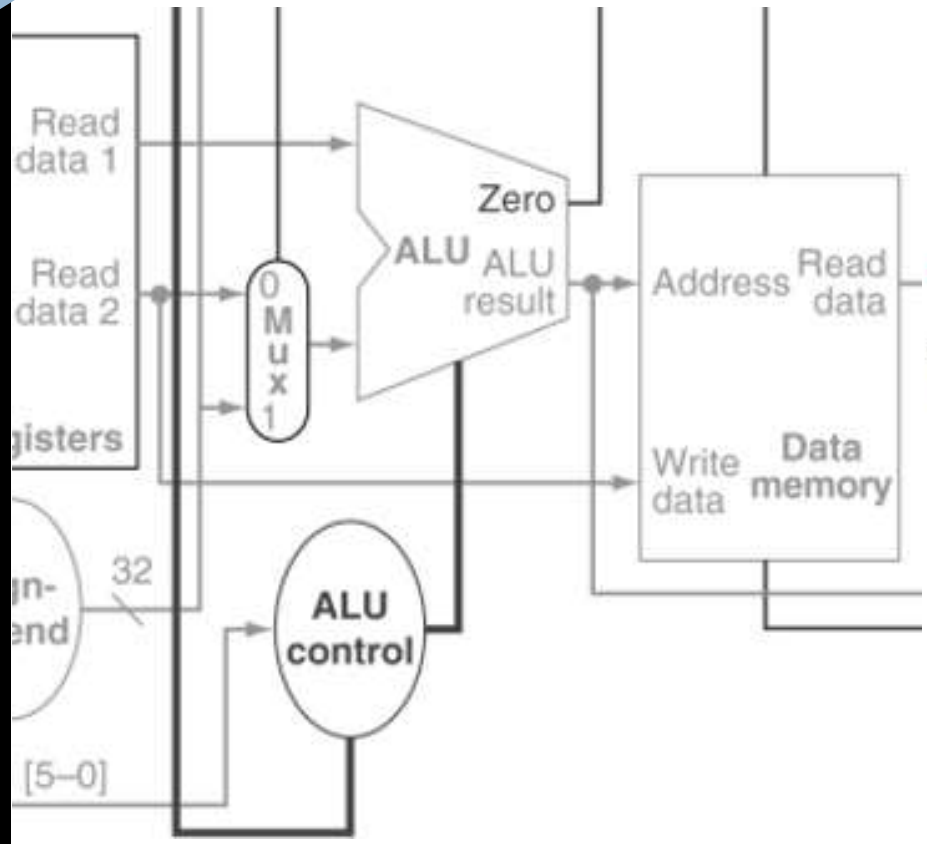

Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
Add
4
0
M
u
x
1
1
M
u
x
0
Add
ALU result
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
Extd
ALU
Zero
ALU result
ALU control
Address
Read data
Write data
Data memory

Register File & Sign Extender




```
40      always @(Rd, Rt, RegDst) WriteReg = (RegDst)? Rd : Rt;
41
42      RegisterFile RegFile
43      (.ReadReg1(Rs),
44       .ReadReg2(Rt),
45       .WriteReg(WriteReg),
46       .WriteData(WriteData),
47       .RegWrite_ctrl(RegWrite),
48       .clk(CLK),
49       .rst(RST),
50       .ReadData1(alu_in1),
51       .ReadData2(ReadData2));
52
53      always @(Imm, Ins[15], ALUSrc, ReadData2) begin
54          ImmExt = (Ins[15]) ? {16'hffff, Imm} : {16'b0, Imm};
55          alu_in2 = (ALUSrc) ? ImmExt : ReadData2;
56      end
```

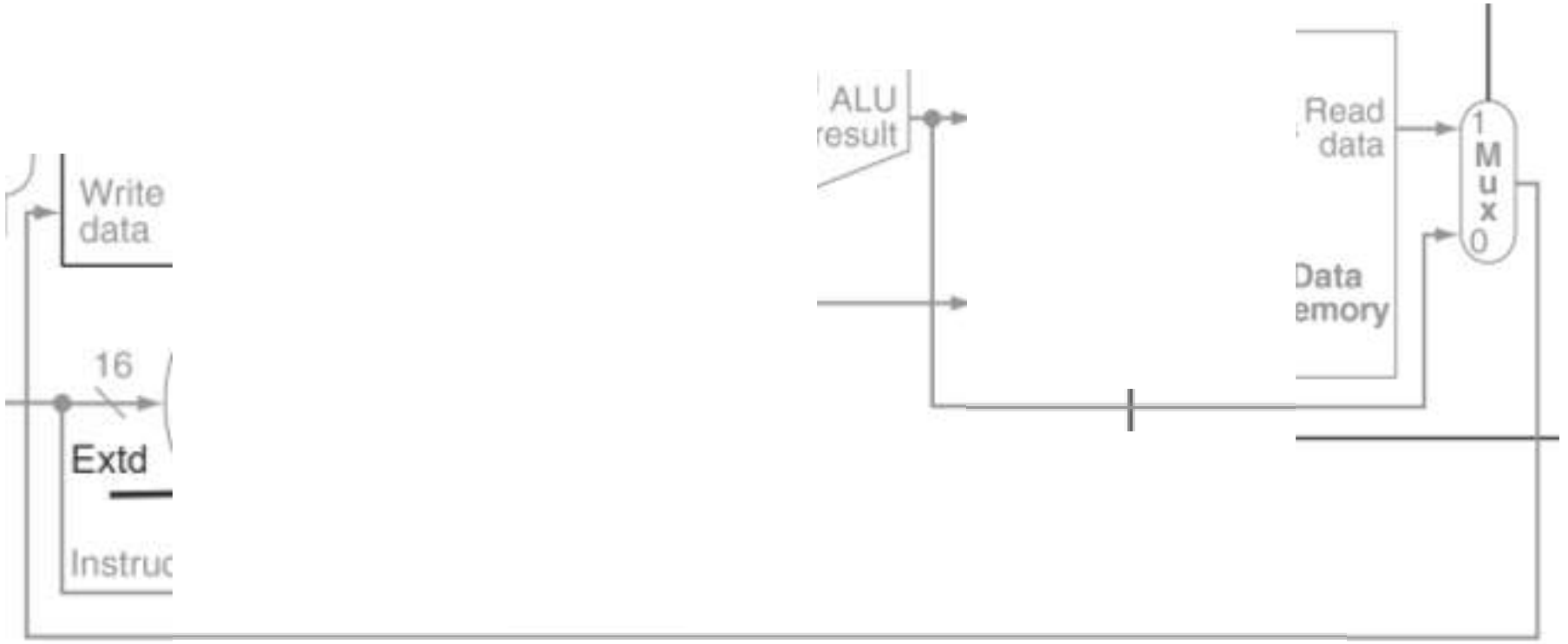
ALU & Data Memory



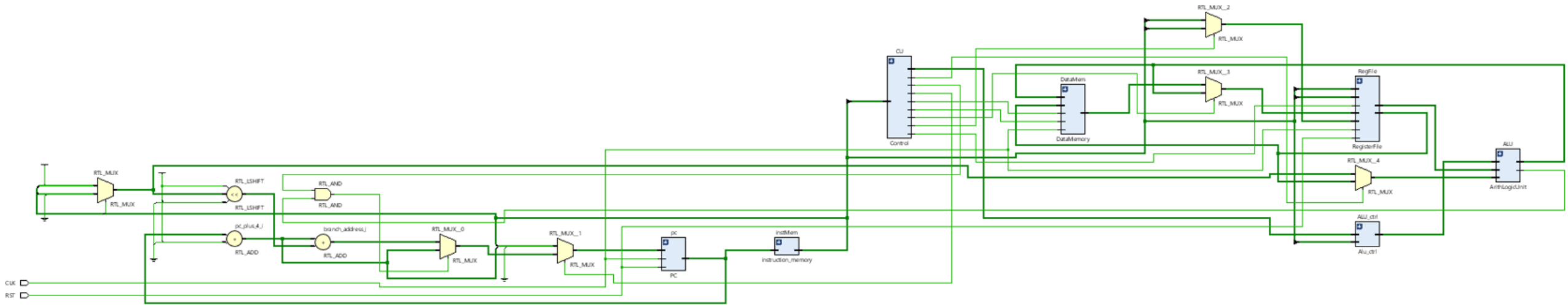

```
    alu_in2 = (ALUSrc) ? ImmExt : ReadData2;
end

Alu_ctrl ALU_ctrl
(.Alu_op(ALUop),
 .funct(funct),
 .Alu_ctrl(ALUopcode));

    ArithLogicUnit ALU(alu_in1, alu_in2, ALUopcode, alu_result, zero);
    DataMemory DataMem(ReadData2, alu_result, MemRead, MemWrite, CLK, ReadData_Mem);
```

WriteBack

```
66      always @(alu_result, ReadData_Mem, MemtoReg)
67          WriteData = (MemtoReg) ? ReadData_Mem : alu_result;
```



RTL Schematic


RTL_MUX
RTL_MUX
RTL_LSHIFT
RTL_LSHIFT
RTL_AND
RTL_AND
pc_plus4_i
RTL_ADD
branch_address_i
RTL_ADD
RTL_MUX__0
RTL_MUX
RTL_MUX__1
RTL_MUX
pc
PC
instMem
instruction_memory
CLK
RST
CU
Control
DataMem
DataMemory
RTL_MUX__2
RTL_MUX
RTL_MUX__3
RTL_MUX
RegFile
RegisterFile
RTL_MUX__4
RTL_MUX
ALU
ArithLogicUnit
ALU_ctrl
Alu_ctrl

Single Cycle Testing

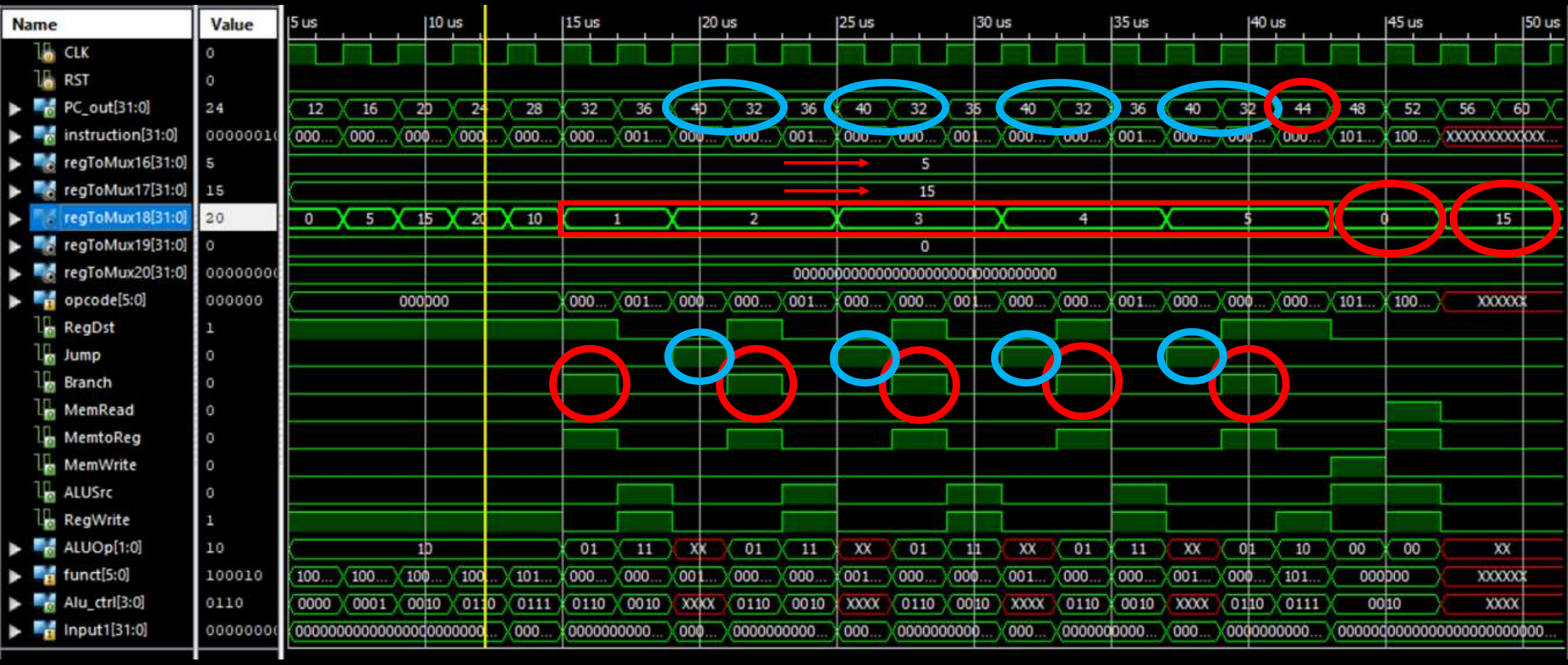
- Top module Testbench
- Code to be Tested
- Waveform

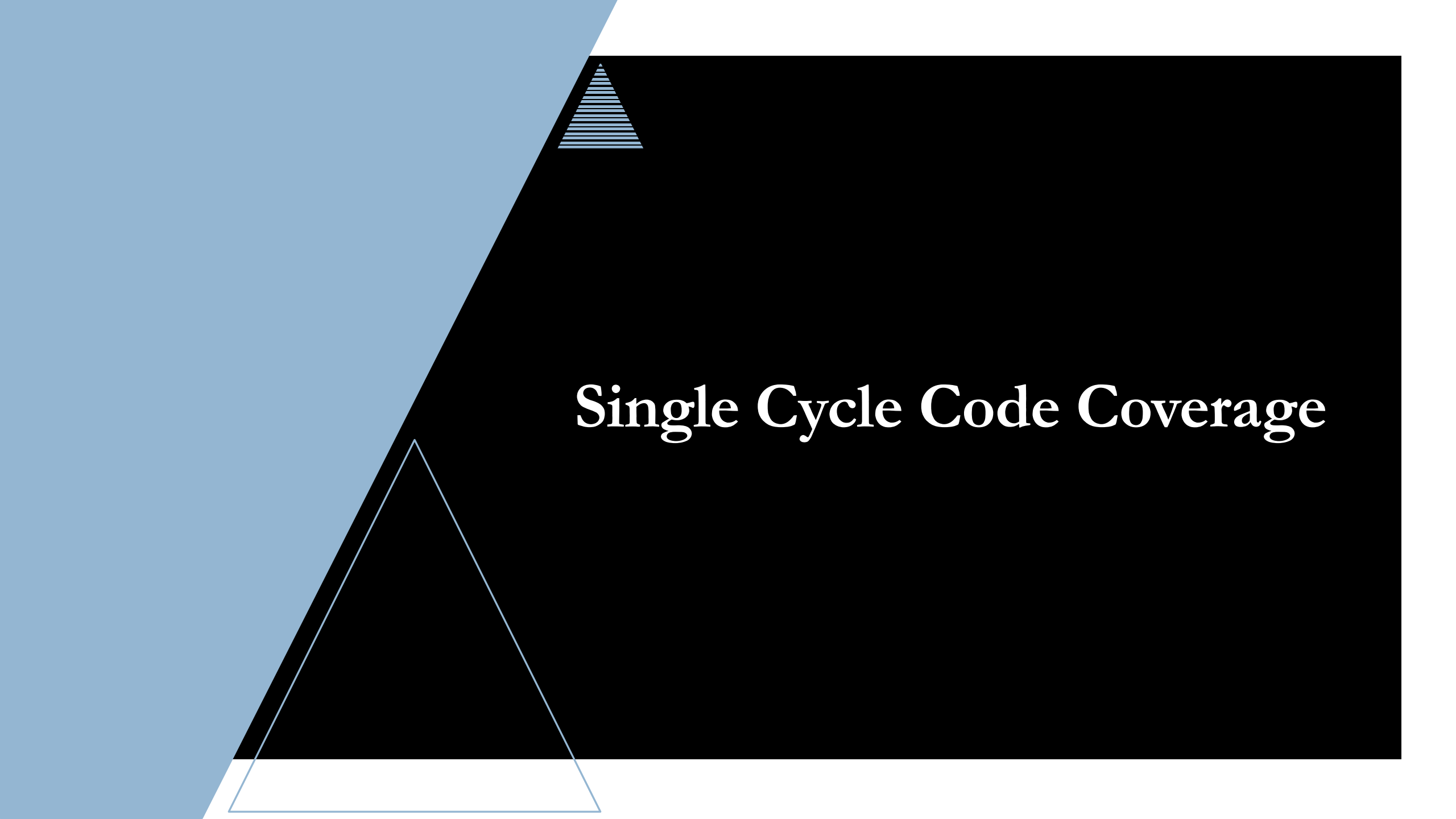
Top Module Testbench

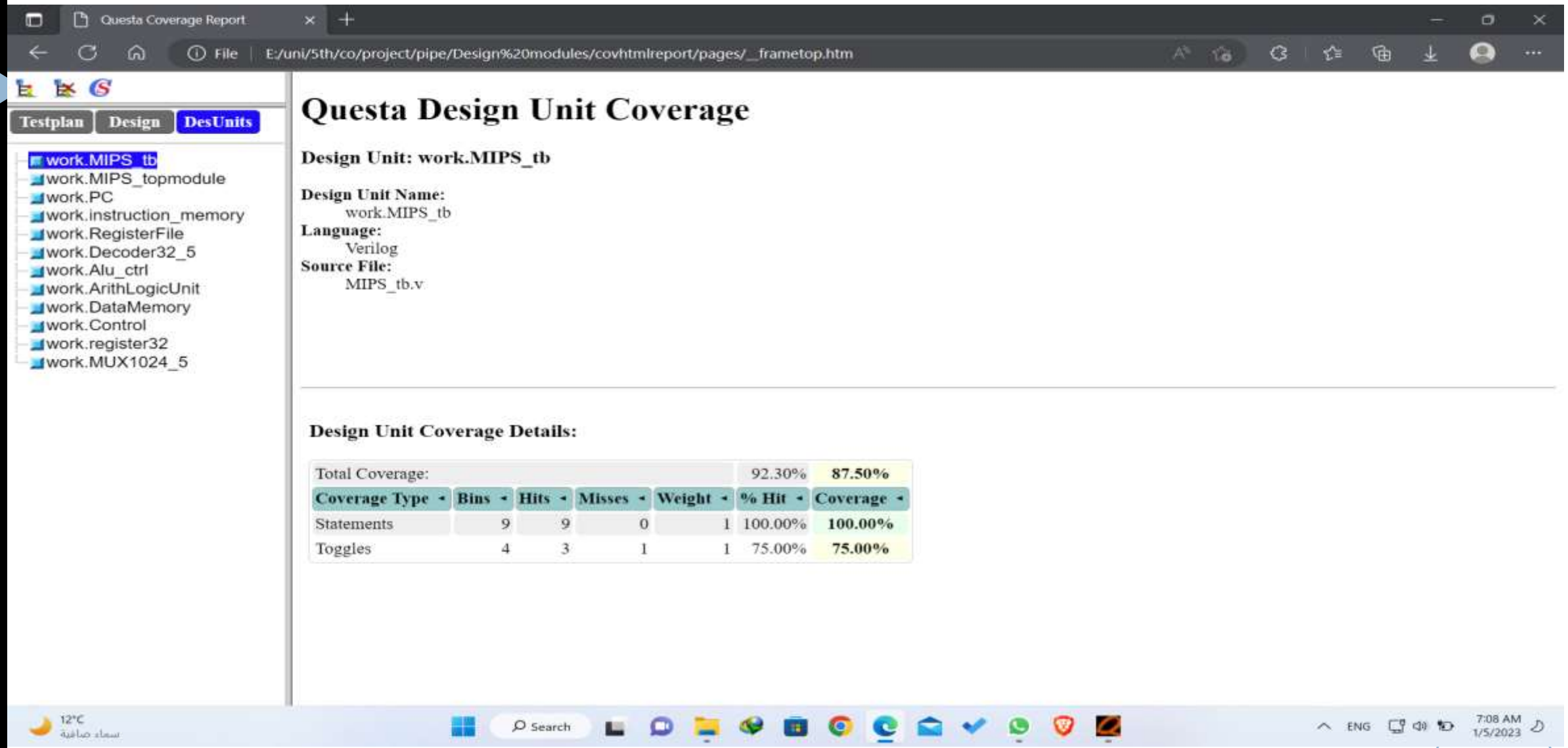
```
`timescale 1ns / 1ps

module MIPS_tb;

	// Inputs
	reg CLK;
	reg RST;

	// Instantiate the Unit Under Test (UUT)
	MIPS_topmodule uut (
		.CLK(CLK),
		.RST(RST)
	);
	always #1000 CLK = ~CLK;
	initial begin
		// Initialize Inputs
		CLK = 0;
		RST = 1;

		// Wait 100 ns for global reset to finish
		#100;
      RST = 0;
		#500;
		// Add stimulus here

	end

endmodule
```

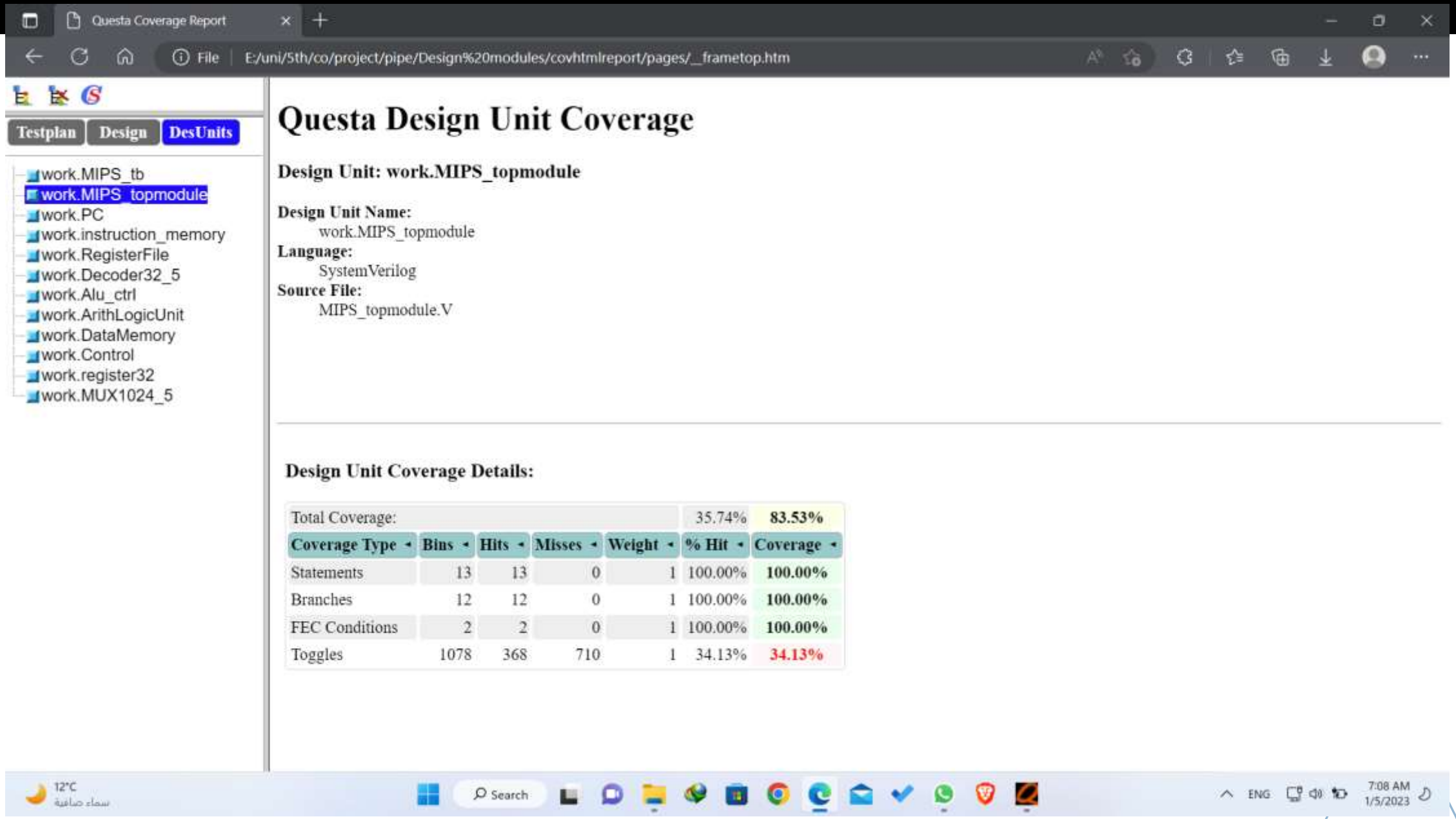
MIPS Code to be Tested

```
0                nop
4                addi $s0, $zero, 0x5      #s0 = 5
8                addi $s1, $zero, 0xF      #s1 = 15
12               and $s2, $s1, $s0         #s2 = 5
16               or  $s2, $s1, $zero       #s2 = 15
20               add $s2, $s1, $s0         #s2 = 20
24               sub $s2, $s1, $s0         #s2 = 10
28               slt $s2, $zero, $s1       #s2 = 1
32     Loop:     beq $s2, $s0, Exit
36               addi $s2, $s2, 0x1
40               j   Loop
44     Exit:     slt $s2, $s2, $s0         #s2 = 0
48               sw  $s1, 0x0($zero)
52               lw  $s2, 0x0($zero)       #s2 = 15
```


Name
Value
CLK
RST
PC_out[31:0]
instruction[31:0]
regToMux16[31:0]
regToMux17[31:0]
regToMux18[31:0]
regToMux19[31:0]
regToMux20[31:0]
opcode[5:0]
RegDst
Jump
Branch
MemRead
MemtoReg
MemWrite
ALUSrc
RegWrite
ALUOp[1:0]
funct[5:0]
Alu_ctrl[3:0]
Input1[31:0]
5 us
10 us
15 us
20 us
25 us
30 us
35 us
40 us
45 us
50 us


Single Cycle Code Coverage


Questa Coverage Report
File
E:/uni/5th/co/project/pipe/Design%20modules/covhtmlreport/pages/_frametop.htm
Testplan
Design
DesUnits
work.MIPS_tb
work.MIPS_topmodule
work.PC
work.instruction_memory
work.RegisterFile
work.Decoder32_5
work.Alu_ctrl
work.ArithLogicUnit
work.DataMemory
work.Control
work.register32
work.MUX1024_5
Questa Design Unit Coverage
Design Unit: work.MIPS_tb
Design Unit Name:
work.MIPS_tb
Language:
Verilog
Source File:
MIPS_tb.v
Design Unit Coverage Details:
Total Coverage: 92.30% 87.50%
Coverage Type | Bins | Hits | Misses | Weight | % Hit | Coverage
Statements 9 9 0 1 100.00% 100.00%
Toggles 4 3 1 1 75.00% 75.00%
12°C
سماء صافية
Search
ENG
7:08 AM
1/5/2023

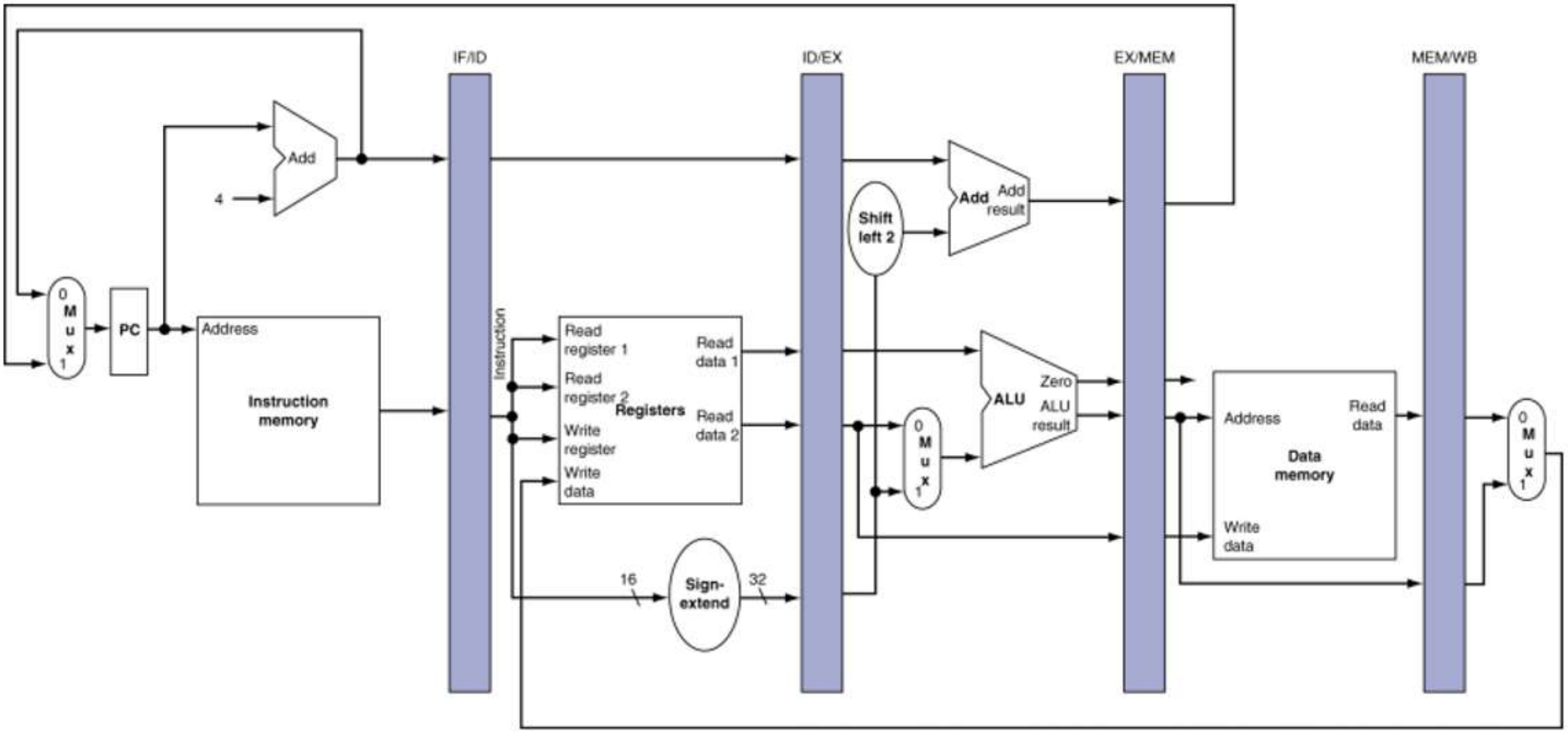

Questa Coverage Report
File
E:/uni/5th/co/project/pipe/Design%20modules/covhtmlreport/pages/__frametop.htm
Testplan
Design
DesUnits
work.MIPS_tb
work.MIPS_topmodule
work.PC
work.instruction_memory
work.RegisterFile
work.Decoder32_5
work.Alu_ctrl
work.ArithLogicUnit
work.DataMemory
work.Control
work.register32
work.MUX1024_5
Questa Design Unit Coverage
Design Unit: work.MIPS_topmodule
Design Unit Name:
work.MIPS_topmodule
Language:
SystemVerilog
Source File:
MIPS_topmodule.V
Design Unit Coverage Details:
Total Coverage: 35.74% 83.53%
Coverage Type | Bins | Hits | Misses | Weight | % Hit | Coverage
Statements 13 13 0 1 100.00% 100.00%
Branches 12 12 0 1 100.00% 100.00%
FEC Conditions 2 2 0 1 100.00% 100.00%
Toggles 1078 368 710 1 34.13% 34.13%
12°C
Search
ENG
7:08 AM
1/5/2023

Main Issues for Single Cycle

- **Limited Clock Speed**
- **Poor Utilization of Hardware**
- **Limited Instruction-level Parallelism**
- **High Power Consumption**

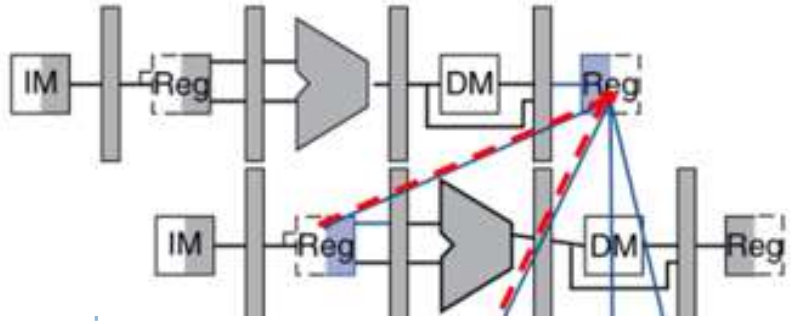
Which leads to the concept of **PIPELINING.**

Pipelining – Main Idea

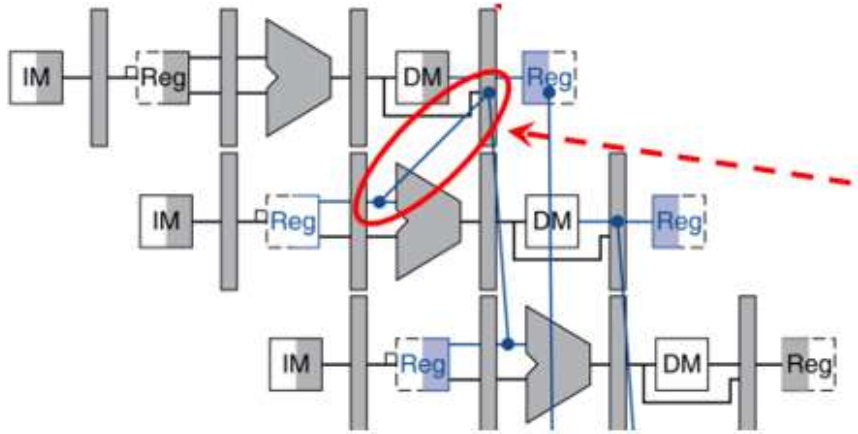


3 Main Hazards Solved

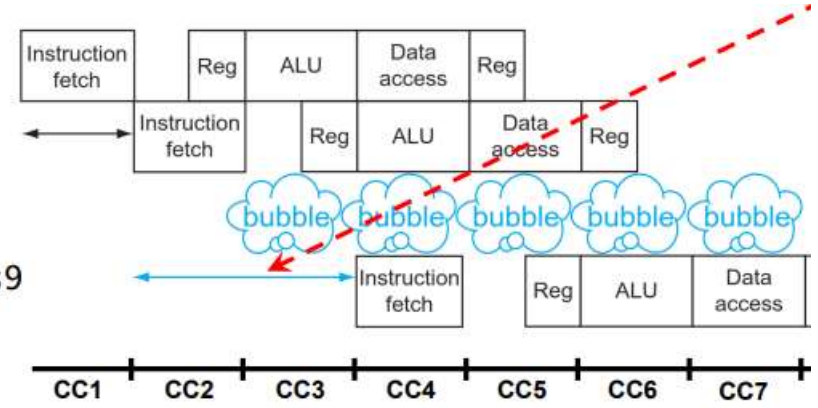
```
sub$s2, $t1, $t3
and$s4, $s2, $t5
```

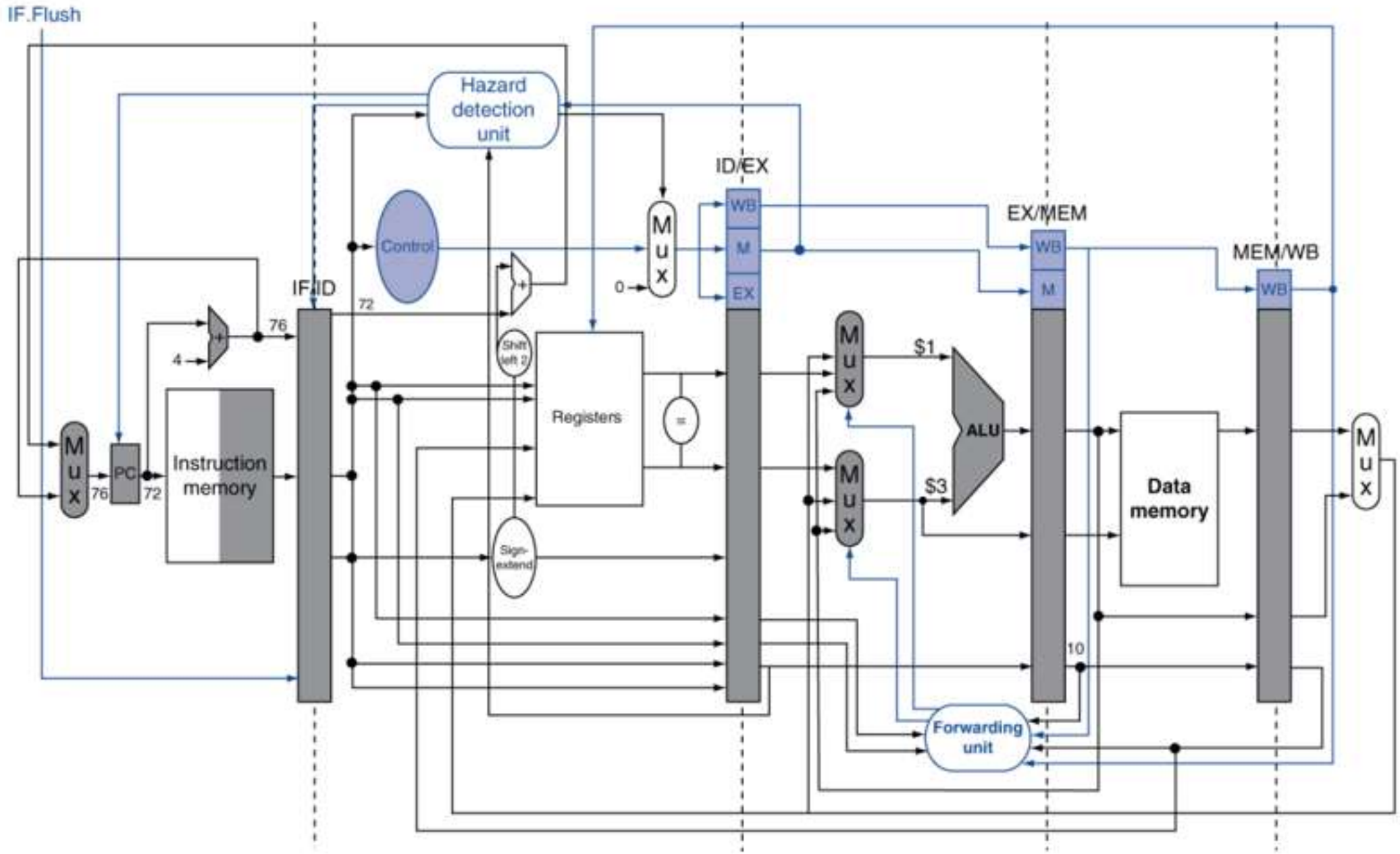



```
lw $s2, 20($s1)
and $s4, $s2, $t5
or $s3, $s2, $t3
```




```
Add $s4, $s5, $s6
beq $s1, $s2, L1
L1: or $s7, $s8, $s9
```




IF.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
IF/ID
72
76
4
0
Shift left 2
Registers
=
Sign-extend
Instruction memory
PC
76
72
$1
$3
ALU
Data memory
10
Forwarding unit
Mux

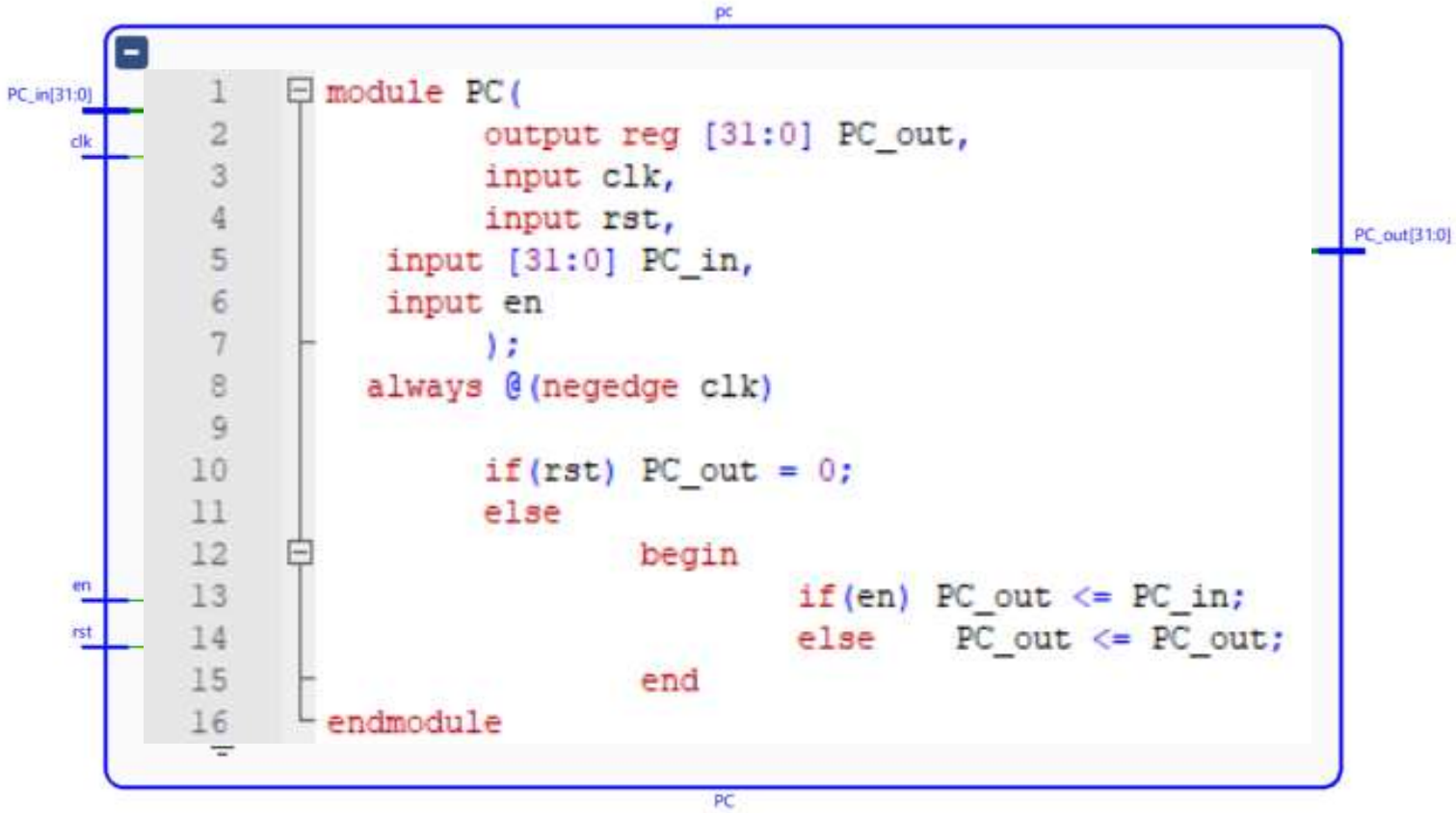
Pipelined Processor

RTL Schematic & Verilog Code for:

- Modules
- TopLevel


5 Pipeline Registers

PC


pc
PC_in[31:0]
clk
en
rst
PC_out[31:0]
PC
1 module PC(
2 output reg [31:0] PC_out,
3 input clk,
4 input rst,
5 input [31:0] PC_in,
6 input en
7);
8 always @(negedge clk)
9
10 if(rst) PC_out = 0;
11 else
12 begin
13 if(en) PC_out <= PC_in;
14 else PC_out <= PC_out;
15 end
16 endmodule

IF/ID

IFID

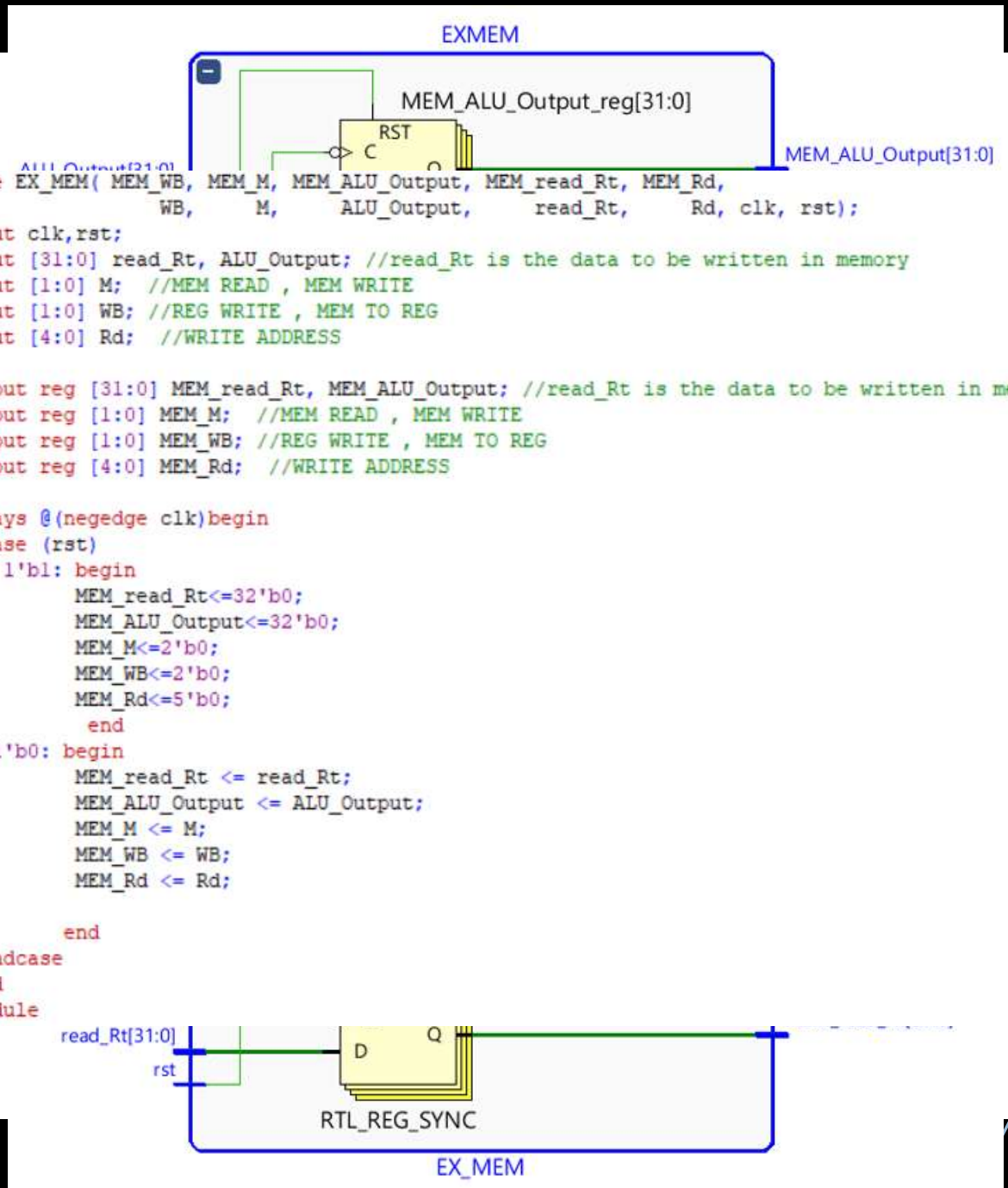
```
1  module IF_ID(ID_PC,ID_instruction, instruction, newPC, clk, rst, en);
2
3  input clk,rst,en;
4  input [31:0] instruction, newPC;
5  output reg [31:0] ID_instruction;
6  output reg [31:0] ID_PC;
7
8          always @(negedge clk) begin
9          if(rst) begin
10             ID_PC<=32'b0;
11             ID_instruction<=32'b0;
12                          end
13         else begin
14                          if(en) begin
15           ID_PC<=newPC;
16           ID_instruction<=instruction;
17          end
18                          else begin
19           ID_PC<=ID_PC;
20           ID_instruction<=ID_instruction;
21          end
22                end
23   end
24 endmodule
```

IF_ID

ID/EX


```
module ID_EX(EX_WB, EX_M, EX_EX, EX_read_Rs, EX_read_Rt, EX_Rs, EX_Rt, EX_Rd, EX_sign_extended32,
             WB,    M,     EX,    read_Rs,    read_Rt,    Rs,    Rt,    Rd,    sign_extended32, clk,rst);

input clk,rst;
input [31:0] sign_extended32, read_Rs, read_Rt;
input [2:0] WB;    //Regdst used in the EX stage as selction line between Rt , Rd ,, Reg write , Mem to reg
input [1:0] M;     //Mem read , Mem write
input [2:0] EX;    //ALU_src for mux selection , Aluop to the Alucontrol unit
input [4:0] Rs, Rt, Rd;  //Rt to be duplicated one copy goes to the Mux with Rd and the other copy to the forwarding unit with Rs

output reg [31:0] EX_sign_extended32, EX_read_Rs, EX_read_Rt;
output reg [2:0] EX_WB;    //Reg dst used in the EX stage as selction line between Rt , Rd ,, Reg write , Mem to reg
output reg [1:0] EX_M;     //Mem read , Mem write
output reg [2:0] EX_EX;    //ALU_src for mux selection , Aluop to the Alucontrol unit
output reg [4:0] EX_Rs, EX_Rt, EX_Rd;

always @(negedge clk) begin
case (rst)
1'b0 : begin
       EX_WB<=WB;
       EX_M<=M;
       EX_EX<=EX;
       EX_read_Rs<=read_Rs;
       EX_read_Rt<=read_Rt;
       EX_sign_extended32<=sign_extended32;
       EX_Rs<=Rs;
       EX_Rt<=Rt;
       EX_Rd<=Rd;
      end
1'b1 : begin
       EX_WB<=3'b0;
       EX_M<=2'b0;
       EX_EX<=3'b0;
       EX_read_Rs<=32'b0;
       EX_read_Rt<=32'b0;
       EX_sign_extended32<=32'b0;
       EX_Rs<=5'b0;
       EX_Rt<=5'b0;
       EX_Rd<=5'b0;
      end
endcase
end
endmodule
```

EX/MEM




```
module EX_MEM( MEM_WB, MEM_M, MEM_ALU_Output, MEM_read_Rt, MEM_Rd,
                    WB,     M,     ALU_Output,     read_Rt,     Rd, clk, rst);
  input clk,rst;
  input [31:0] read_Rt, ALU_Output; //read_Rt is the data to be written in memory
  input [1:0] M;  //MEM READ , MEM WRITE
  input [1:0] WB; //REG WRITE , MEM TO REG
  input [4:0] Rd;  //WRITE ADDRESS

  output reg [31:0] MEM_read_Rt, MEM_ALU_Output; //read_Rt is the data to be written in memory
  output reg [1:0] MEM_M;  //MEM READ , MEM WRITE
  output reg [1:0] MEM_WB; //REG WRITE , MEM TO REG
  output reg [4:0] MEM_Rd;  //WRITE ADDRESS

  always @(negedge clk)begin
    case (rst)
      1'b1: begin
            MEM_read_Rt<=32'b0;
            MEM_ALU_Output<=32'b0;
            MEM_M<=2'b0;
            MEM_WB<=2'b0;
            MEM_Rd<=5'b0;
             end
     1'b0: begin
            MEM_read_Rt <= read_Rt;
            MEM_ALU_Output <= ALU_Output;
            MEM_M <= M;
            MEM_WB <= WB;
            MEM_Rd <= Rd;

           end
    endcase
   end
endmodule
```

MEM/WB

MEMWB

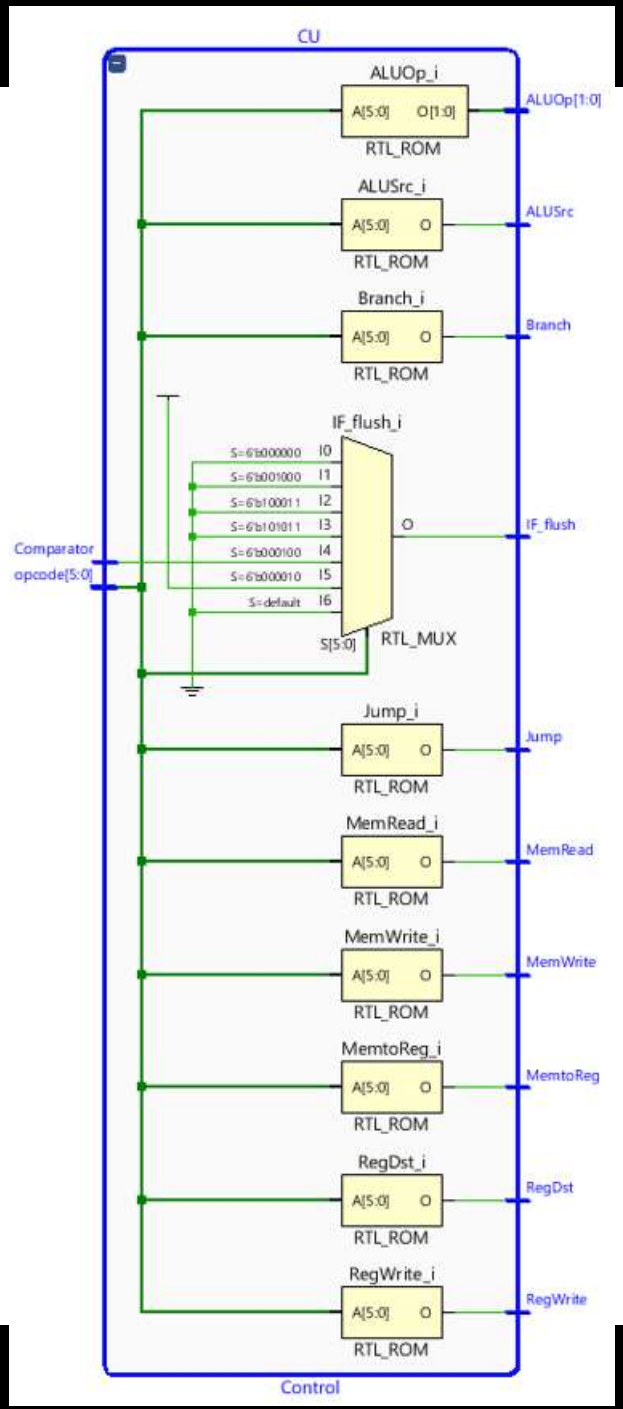
```
module MEM_WB( WB_WB, WB_memory_Output, WB_ALU_Output, WB_Rd
             ,   WB,    memory_Output,    ALU_Output,    Rd, clk,rst);

  input clk,rst;
  input [31:0] ALU_Output, memory_Output;
  input [4:0] Rd;
  input [1:0] WB;

  output reg [31:0] WB_ALU_Output, WB_memory_Output;
  output reg [4:0] WB_Rd;
  output reg [1:0] WB_WB;

 always @(negedge clk)begin
   case (rst)
   1'b1  : begin
                          WB_ALU_Output<=0;
             WB_memory_Output<=0;
             WB_Rd<=0;
             WB_WB<=0;
           end

   1'b0 : begin
                  WB_ALU_Output<=ALU_Output;
            WB_memory_Output<=memory_Output;
            WB_Rd<=Rd;
            WB_WB<=WB;
          end
   endcase
 end
endmodule
```

MEM_WB

Edited & Added Modules

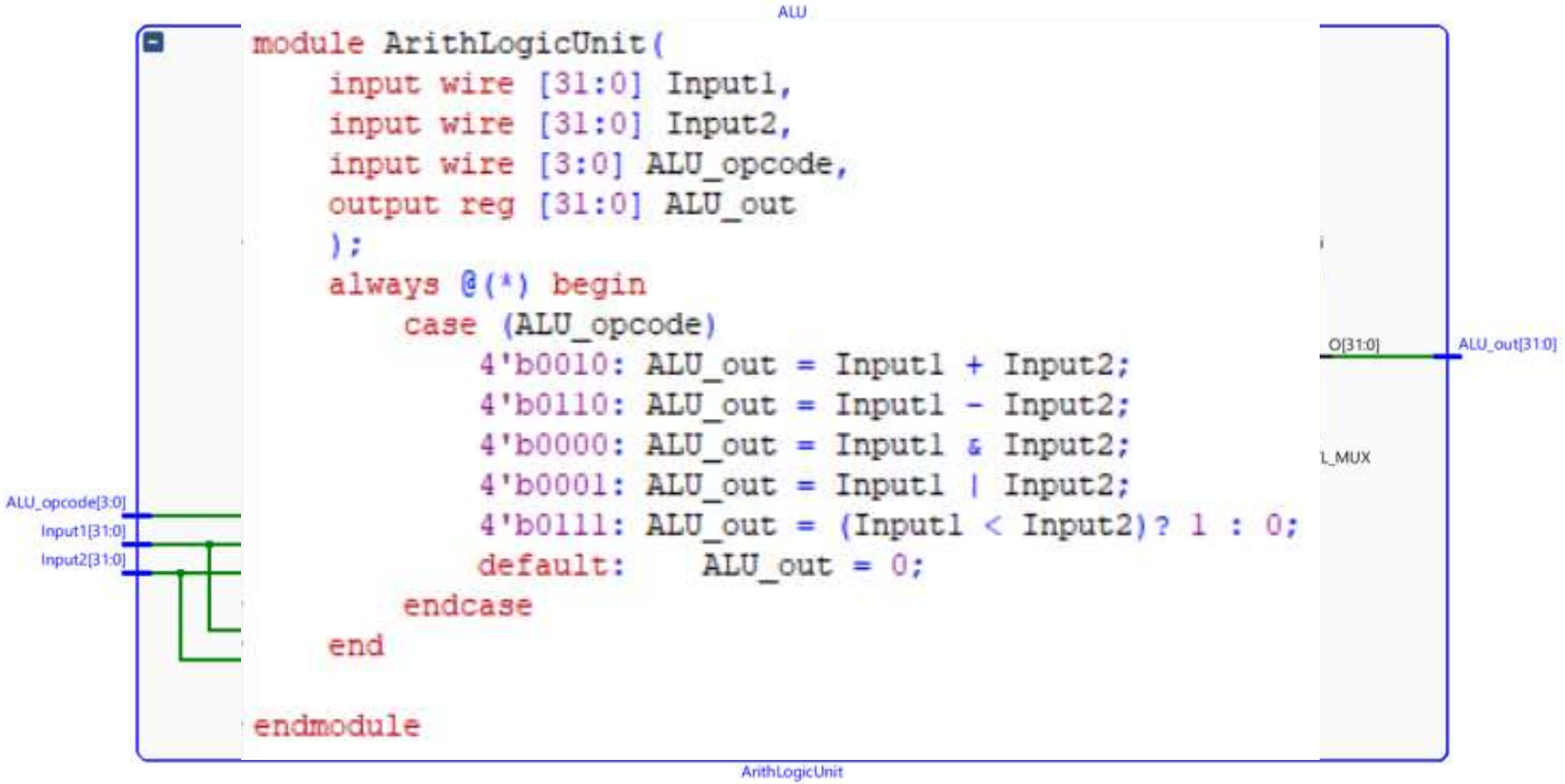
- Control Unit
- ALU
- Forwarding Unit
- Hazard Detection Unit

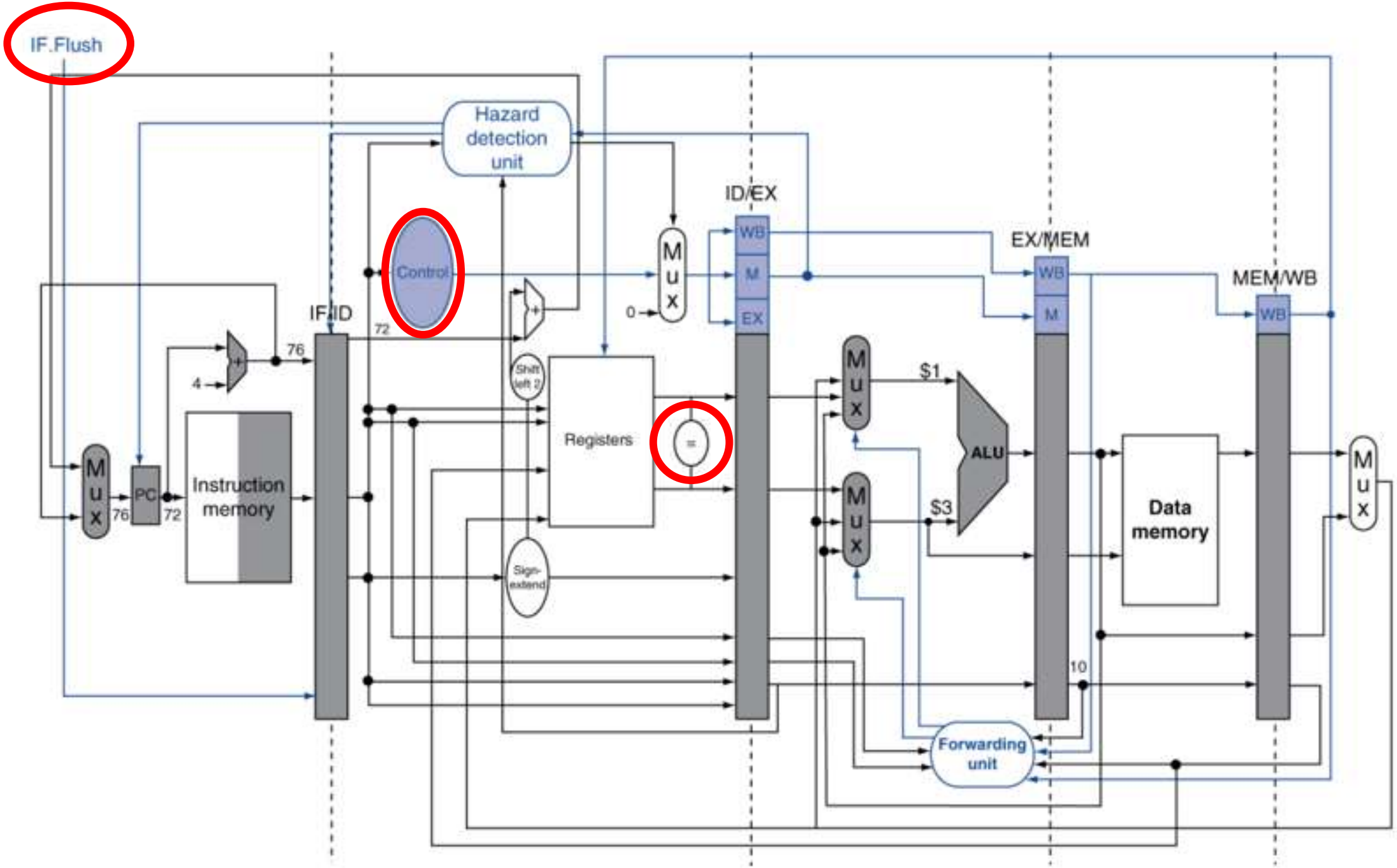
Control Unit




```
6'b000100: //beq-controls
begin
        RegDst <= 1'b1;
        Branch <= 1'b1;
        MemtoReg <= 1'b1;
        ALUOp <= 2'b01;
        IF_flush <= (Comparator)? 1'b1 : 1'b0;
end

6'b000010:
        begin           //j-controls
        Jump <= 1'b1;
        IF_flush <= 1'b1;
        end
```

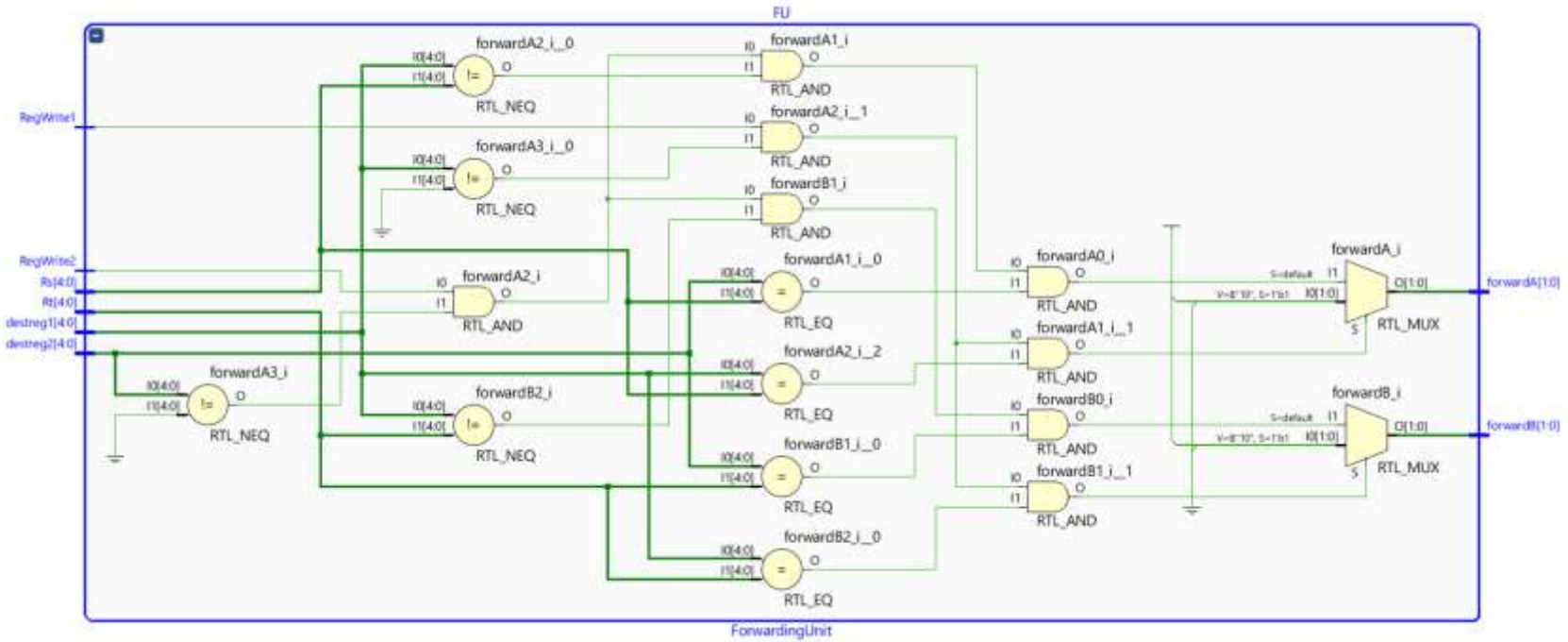
ALU


ALU
module ArithLogicUnit(
input wire [31:0] Input1,
input wire [31:0] Input2,
input wire [3:0] ALU_opcode,
output reg [31:0] ALU_out
);
always @(*) begin
case (ALU_opcode)
4'b0010: ALU_out = Input1 + Input2;
4'b0110: ALU_out = Input1 - Input2;
4'b0000: ALU_out = Input1 & Input2;
4'b0001: ALU_out = Input1 | Input2;
4'b0111: ALU_out = (Input1 < Input2)? 1 : 0;
default: ALU_out = 0;
endcase
end
endmodule
ALU_opcode[3:0]
Input1[31:0]
Input2[31:0]
O[31:0]
ALU_out[31:0]
L_MUX
ArithLogicUnit


IF.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
IF/ID
72
76
4
0
Shift left 2
Registers
=
Sign-extend
PC
Instruction memory
ALU
$1
$3
Data memory
10
Forwarding unit
Mux

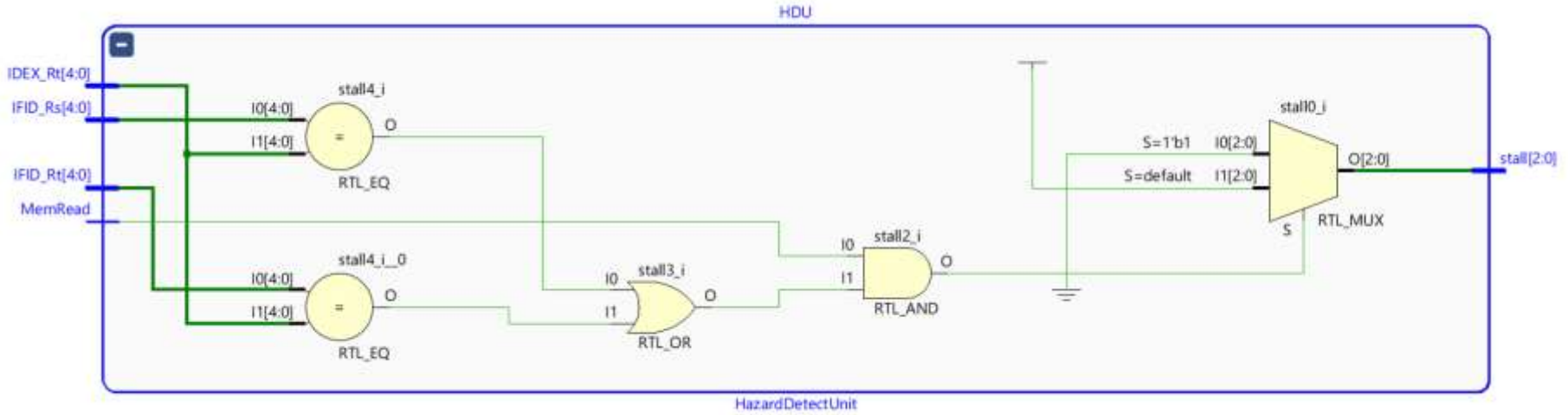
Forwarding Unit

```
module ForwardingUnit(forwardA, forwardB, Rs, Rt, destreg1, destreg2, RegWrite1, RegWrite2);
// Rs = ID_EX.Rs; Rt = ID_EX.Rt;
//destreg1 = EX_MEM.destination_reg; destreg2 = MEM_WB.destination_reg;
//RegWrite1 = EX_MEM.control signals.RegWrite;
```

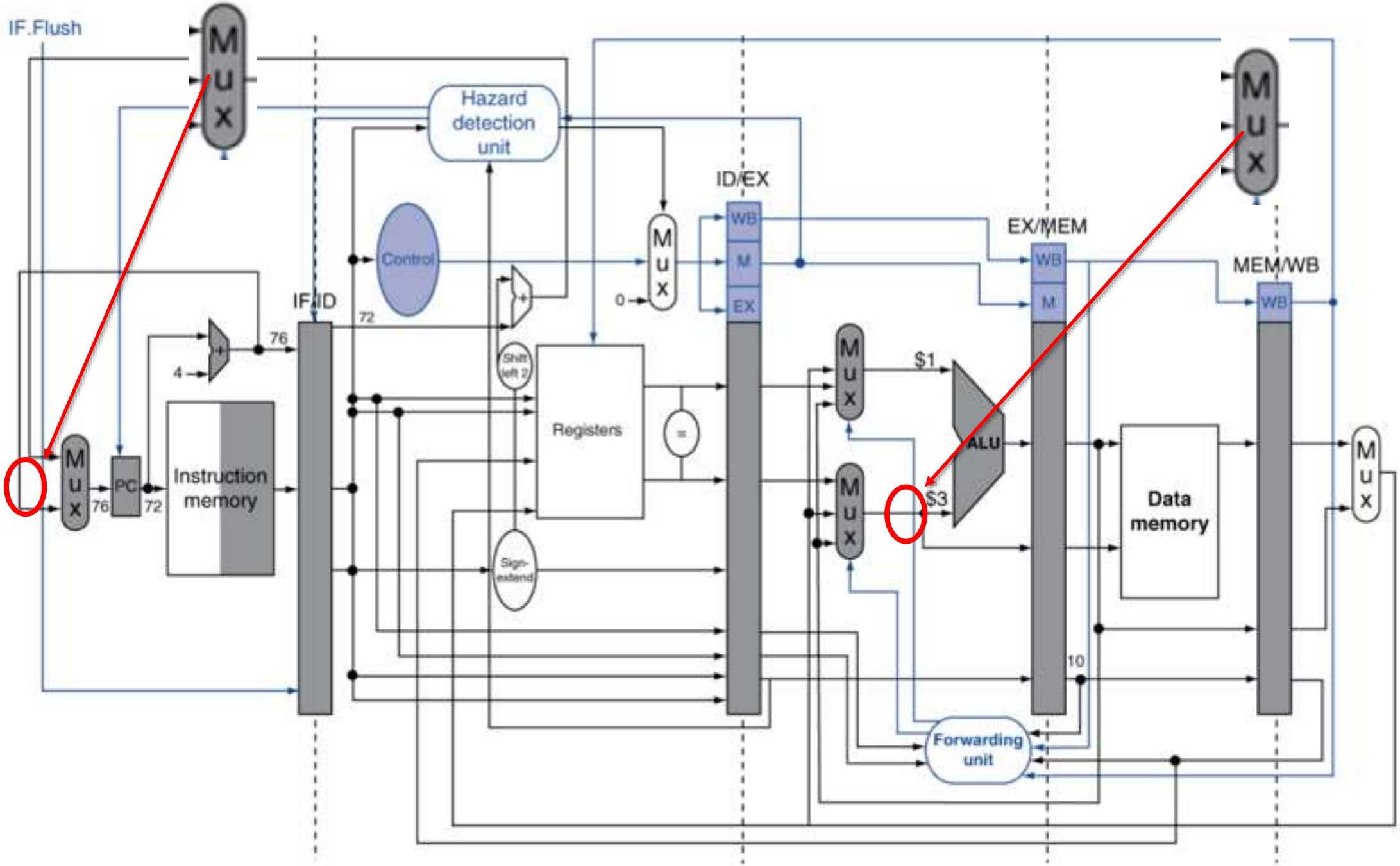



```
            forwardB <= 2'b00;
        end

endmodule
```

Hazard Detection Unit



Top Level Module

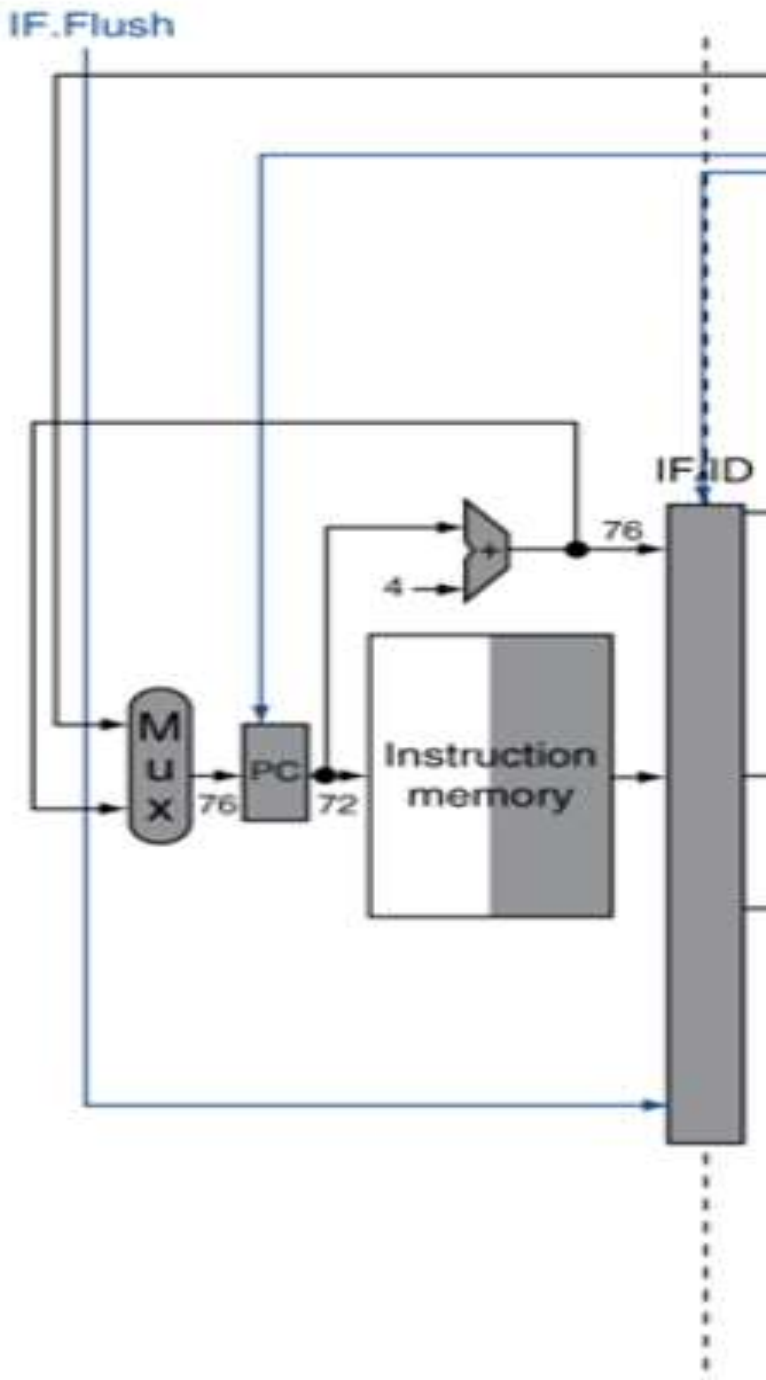

IF.Flush
M u x
Hazard detection unit
Control
ID/EX
WB
M
EX
EX/MEM
MEM/WB
IF/ID
72
76
4
+
0
Shift left 2
Registers
=
Sign-extend
PC
Instruction memory
$1
$3
ALU
Data memory
10
Forwarding unit

Inputs, Wires, Outputs Initialization

```
1  module MIPS_pipelined(input CLK, RST);
2
3      wire [31:0] current_pc, pc_plus_4, pc_plus_4_out, branch_address, jump_address, Ins, IFID_inst;
4      reg [31:0] ImmExt, next_pc, WriteData, MUX_A, MUX_B,MUX_C, WB_data;
5      wire [31:0] instruction, ReadData1, ReadData2, alu_result, ReadData_Mem;
6      wire [25:0] dirJump_address;
7      wire [15:0] Imm;
8      wire  IF_flush, RegWrite, ALUSrc, MemRead, MemWrite, MemtoReg, RegDst, Jump, BranchFlag, EN, hazard_to_MUX;
9      reg comparator;
10     wire [5:0] CUopcode;
11     wire [4:0] Rs, Rt1, Rt2, Rd, destreg2;
12     wire [3:0] ALUopcode;
13     wire [1:0] ALUop;
14     wire [63:0] IF_IDoutput;
15     wire [125:0] IDEX_out;
16     wire [9:0] control_signals;
17     wire [9:0] IDEXcontrol_signals;
18     wire [4:0] IDEX_Rs, IDEX_Rt1, IDEX_Rt2, IDEX_Rd;
19     reg [4:0] destreg1;
20     wire [31:0]  EXMEM_aluOut, IDEXread_Rs, IDEXread_Rt, regtoDataMem, IDEXsign_extended32;
21     wire [4:0]  EXMEM_destreg1;
22     wire [3:0] EXMEMcontrol_signals;
23     wire [31:0] MEMWBmemory_data, MEMWBalu_out;
24     wire [1:0] MEMWBcontrol_signals, forwardA, forwardB;
25     wire [70:0] MEMWB_out;
26     wire [2:0] stall_signal;
27     reg [9:0] MUXtoIDEX;
28     wire [72:0] EXMEM_out;
29     wire [1:0] EXMEM_M;  //MEM READ , MEM WRITE
30     wire [1:0] EXMEM_WB; //REG WRITE , MEM TO REG
31     wire [2:0] IDEX_WB;   //Reg dst used in the EX stage as selction line between Rt , Rd ,, Reg write , Mem to reg
32     wire [1:0] IDEX_M;    //Mem read , Mem write
33     wire [2:0] IDEX_EX;   //ALU_src for mux selection , Aluop to the Alucontrol unit
34     wire [1:0] MEMWB_WB;
```

IF_Stage (Instruction Fetch)

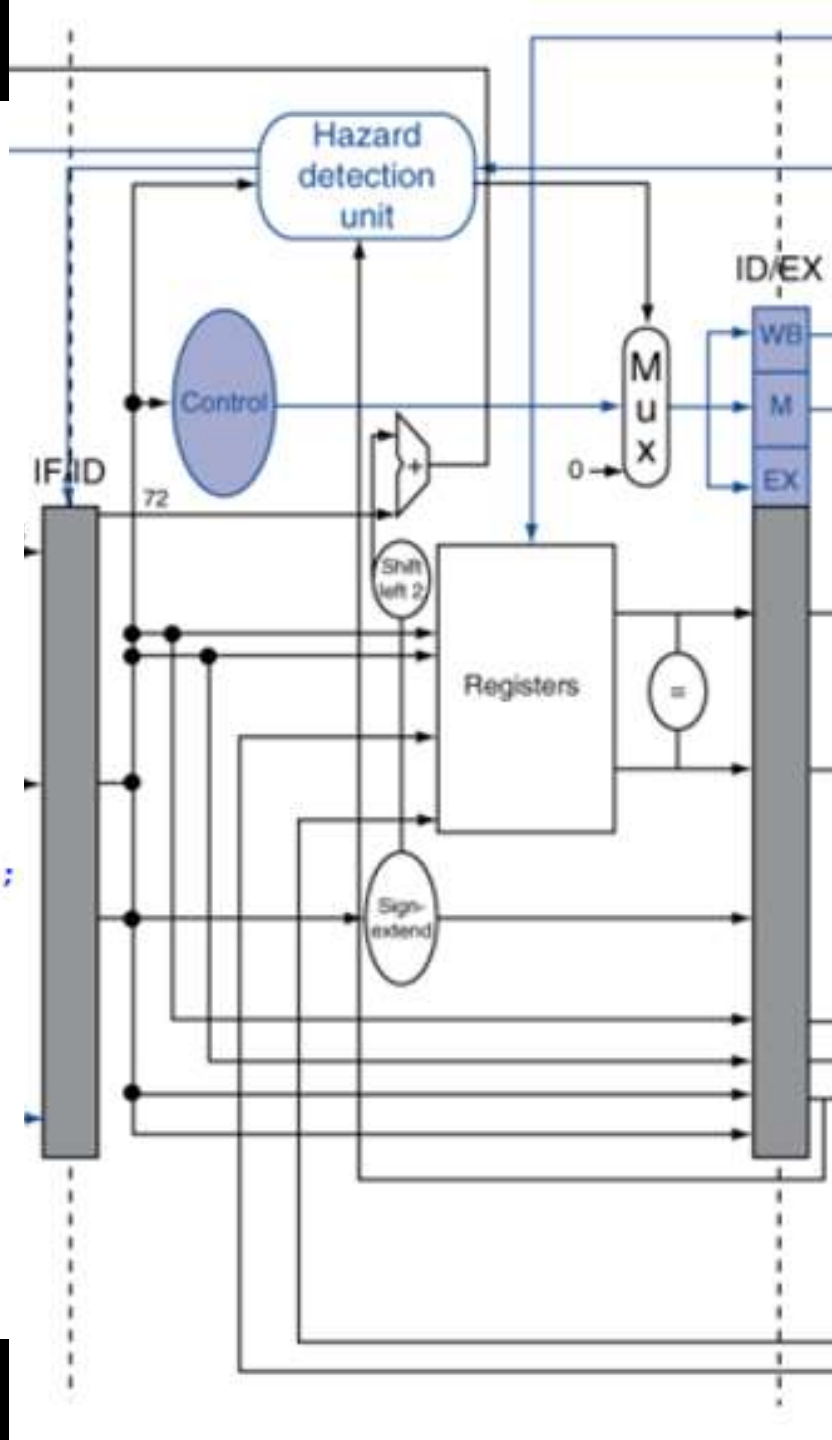
```
PC pc
(.clk(CLK),
 .rst(RST),
 .PC_in(next_pc),
 .PC_out(current_pc),
 .en(stall_signal[1]));

always @(pc_plus_4, branch_address, Jump, jump_address, IF_flush) begin
  next_pc = (IF_flush)? branch_address : pc_plus_4;
  next_pc = (Jump)? jump_address : next_pc;
end
assign pc_plus_4 = current_pc + 4;

instruction_memory instMem (.read_add(current_pc), .instruction(Ins));
```



ID_Stage (Instruction Decode)

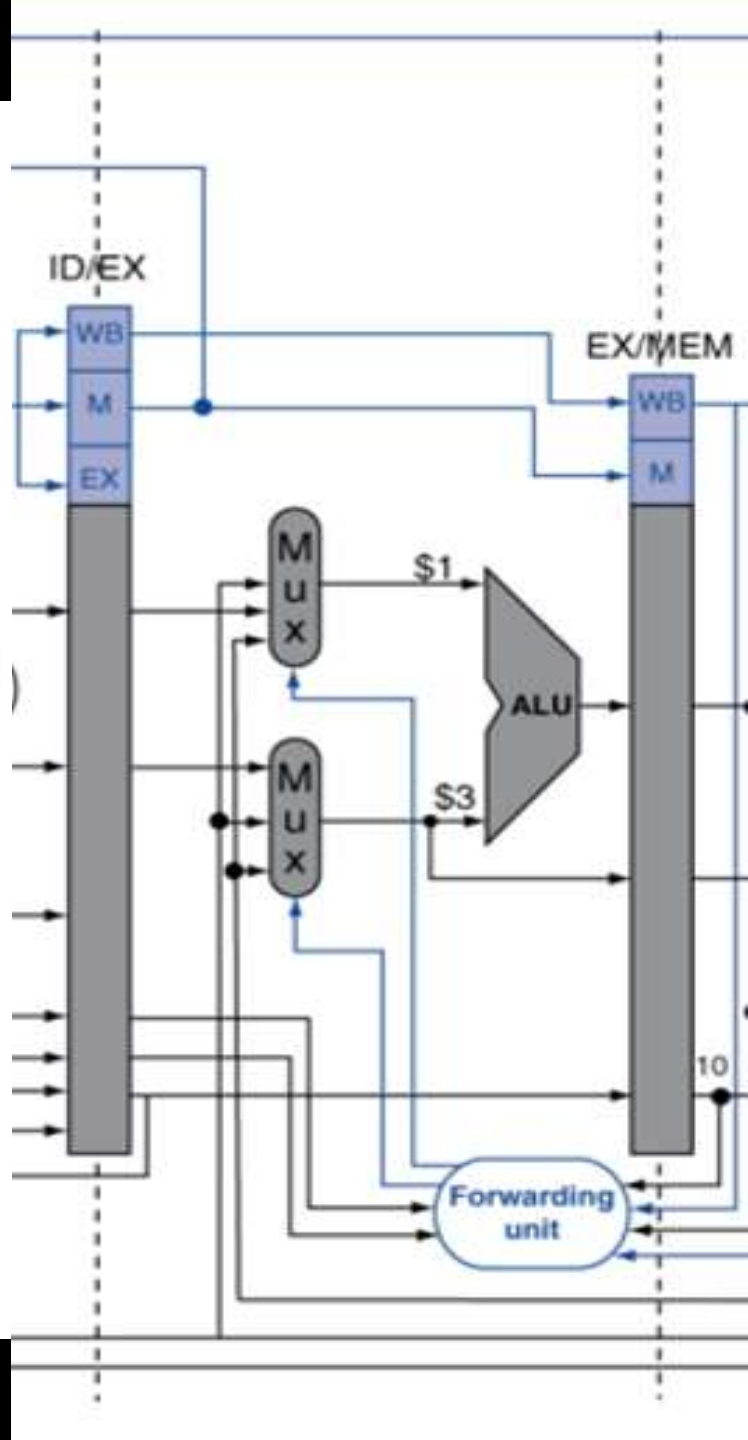
```
RegisterFile RegFile
(.ReadReg1(Rs),
 .ReadReg2(Rt1),
 .WriteReg(destreg2),
 .WriteData(WB_data),
 .RegWrite_ctrl(MEMWB_WB[1]),
 .clk(CLK),
 .rst(RST),
 .ReadData1(ReadData1),
 .ReadData2(ReadData2));

Control CU
(.RegDst(RegDst),
 .ALUSrc(ALUSrc),
 .MemtoReg(MemtoReg),
 .RegWrite(RegWrite),
 .MemRead(MemRead),
 .MemWrite(MemWrite),
 .Branch(Branch),
 .Jump(Jump),
 .ALUOp(ALUop),
 .opcode(CUopcode),
 .Comparator(comparator),
 .IF_flush(IF_flush));
```

```
assign control_signals[9] = RegDst;
assign control_signals[8] = Jump;
assign control_signals[7] = Branch;
assign control_signals[6] = MemRead;
assign control_signals[5] = MemtoReg;
assign control_signals[4] = MemWrite;
assign control_signals[3] = RegWrite;
assign control_signals[2] = ALUSrc;
assign control_signals[1:0] = ALUop;
assign hazard_to_MUX = stall_signal[0];

always @(hazard_to_MUX, control_signals)
MUXtoIDEX = (hazard_to_MUX)? control_signals: 10'b0;

HazardDetectUnit HDU
(.MemRead(IDEX_M[1]),
 .IFID_Rs(Rs),
 .IFID_Rt(Rt1),
 .IDEX_Rt(IDEX_Rt1),
 .stall(stall_signal));
```



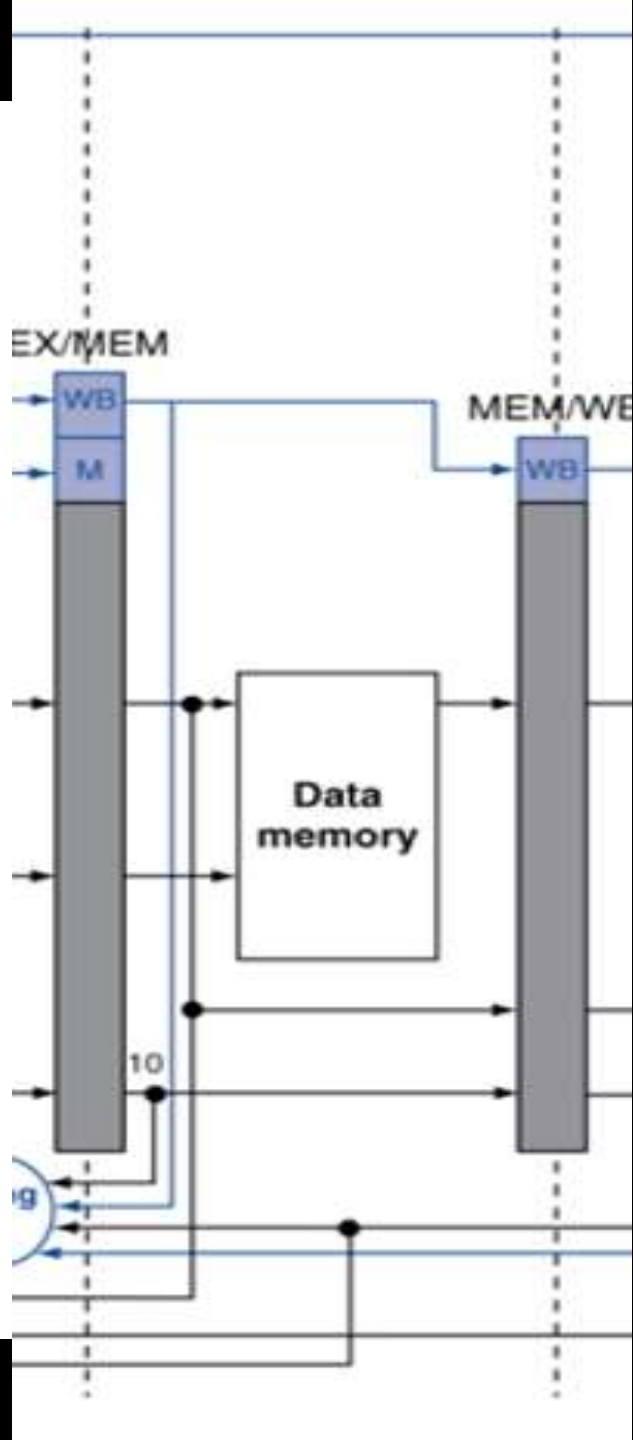
EX_Stage (Execute)

```
  ID_EX IDEX
always @(IDE    ArithLogicUnit ALU
destreg1 = (    (MUX_A, MUX_C, ALUopcode, alu_result);  ls[9] = RegDst
                Alu_ctrl ALU_ctrl
always @(for    (.Alu_op(IDEX_EX [1:0]),
     case (f     .funct(IDEXsign_extended32 [5:0]),
         2'b     .Alu_ctrl(ALUopcode));
         2'b
         2'b
        defa
     endcase    ForwardingUnit FU
     end        (.RegWrite1(EXMEM_WB[1]),
                 .RegWrite2(MEMWB_WB[1]),
always @(for                                    ], IDEXsign_extended32) begin
case (forwar     .Rs(IDEX_Rs),
    2'b00 :      .Rt(IDEX_Rt1),
    2'b01 :      .destreg1(EXMEM_destreg1),
    2'b10 :      .destreg2(destreg2),
   default:           .forwardA(forwardA),
endcase               .forwardB(forwardB));
MUX_C = (IDE
end
   .EX_Rd(IDEX_Rd));
```



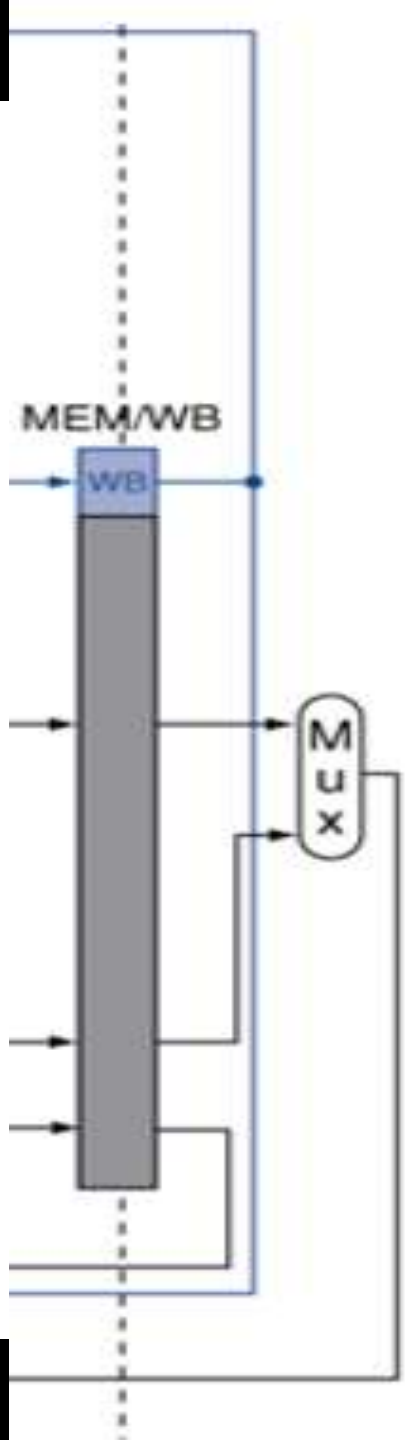
MEM_Stage (Memory Access)

```
EX_MEM EXMEM
(.clk(CLK),
 .rst(RST),
 .ALU_Output(alu_result),
 .read_Rt(MUX_B),
 .Rd(destregl),
 .WB(IDEX_WB[1:0]),
 .M(IDEX_M),
 .MEM_read_Rt(regtoDataMem),
 .MEM_ALU_Output(EXMEM_aluOut),
 .MEM_M(EXMEM_M),
 .MEM_WB(EXMEM_WB),
 .MEM_Rd(EXMEM_destregl));

 DataMemory DataMem(regtoDataMem, EXMEM_aluOut, EXMEM_M[1], EXMEM_M[0], CLK, ReadData_Mem);
```

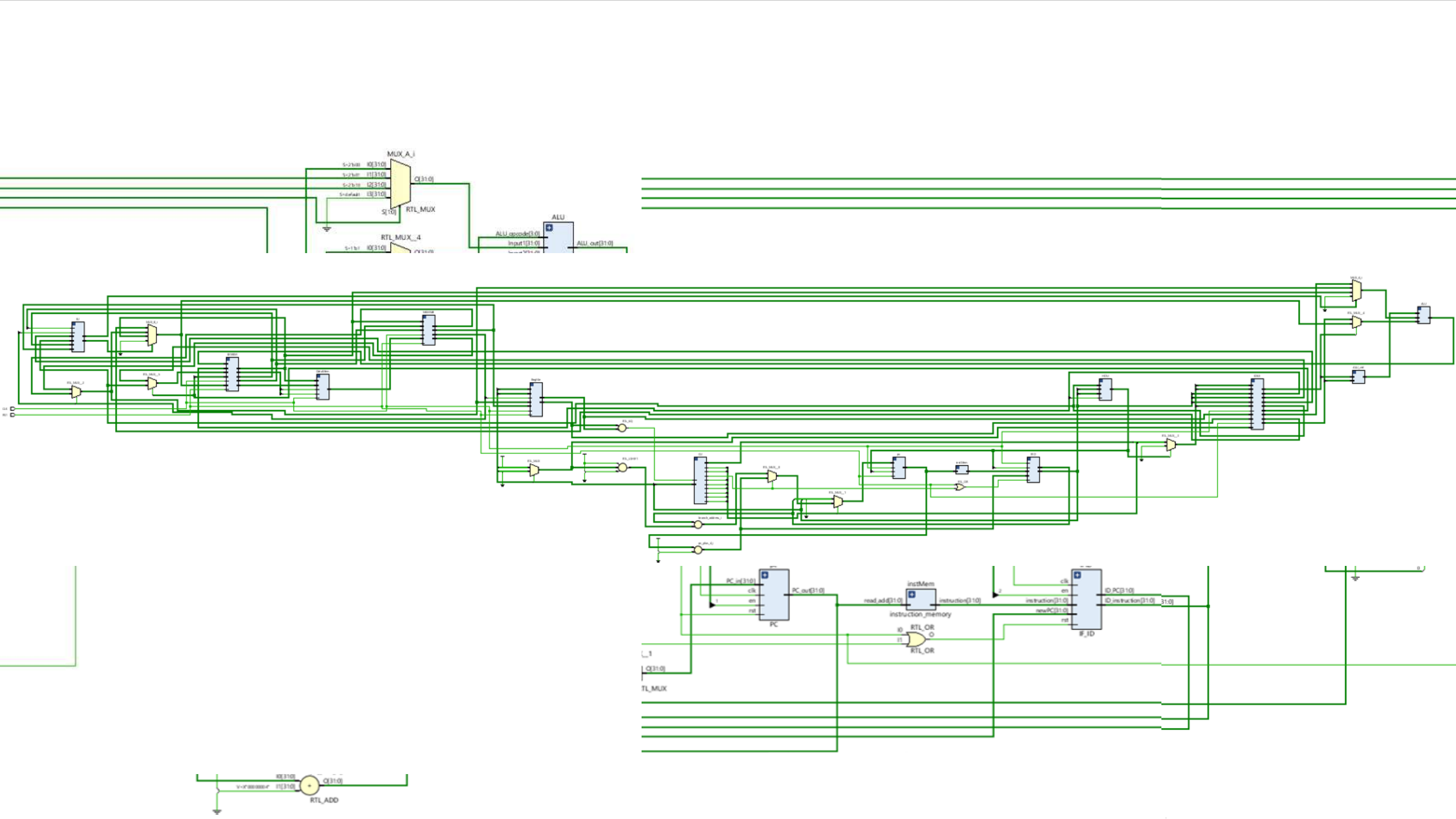


WB_Stage (Write Back)

```
MEM_WB MEMWB
(.WB(EXMEM_WB),
 .memory_Output(ReadData_Mem),
 .ALU_Output(EXMEM_aluOut),
 .Rd(EXMEM_destreg1),
 .clk(CLK),
 .rst(RST),
 .WB_WB(MEMWB_WB),
 .WB_ALU_Output(MEMWBalu_out),
 .WB_memory_Output(MEMWBmemory_data),
 .WB_Rd(destreg2)
 );
 always @(MEMWBmemory_data, MEMWBalu_out, MEMWB_WB[0])
 WB_data = (MEMWB_WB[0])? MEMWBmemory_data : MEMWBalu_out;

endmodule
```



RTL Schematic


MUX_A_i
I0[31:0]
I1[31:0]
I2[31:0]
I3[31:0]
O[31:0]
S[1:0]
RTL_MUX
RTL_MUX_4
ALU
ALU_opcode[3:0]
input1[31:0]
ALU_out[31:0]
PC_in[31:0]
clk
en
rst
PC_out[31:0]
PC
instMem
read_add[31:0]
instruction[31:0]
instruction_memory
RTL_OR
I0
I1
O
RTL_OR
instruction[31:0]
newPC[31:0]
IF_ID
ID_PC[31:0]
ID_instruction[31:0]
31:0]
TL_MUX
I0[31:0]
I1[31:0]
O[31:0]
RTL_ADD

Pipelined Testing

- Top module Testbench
- Code to be Tested
- Waveform

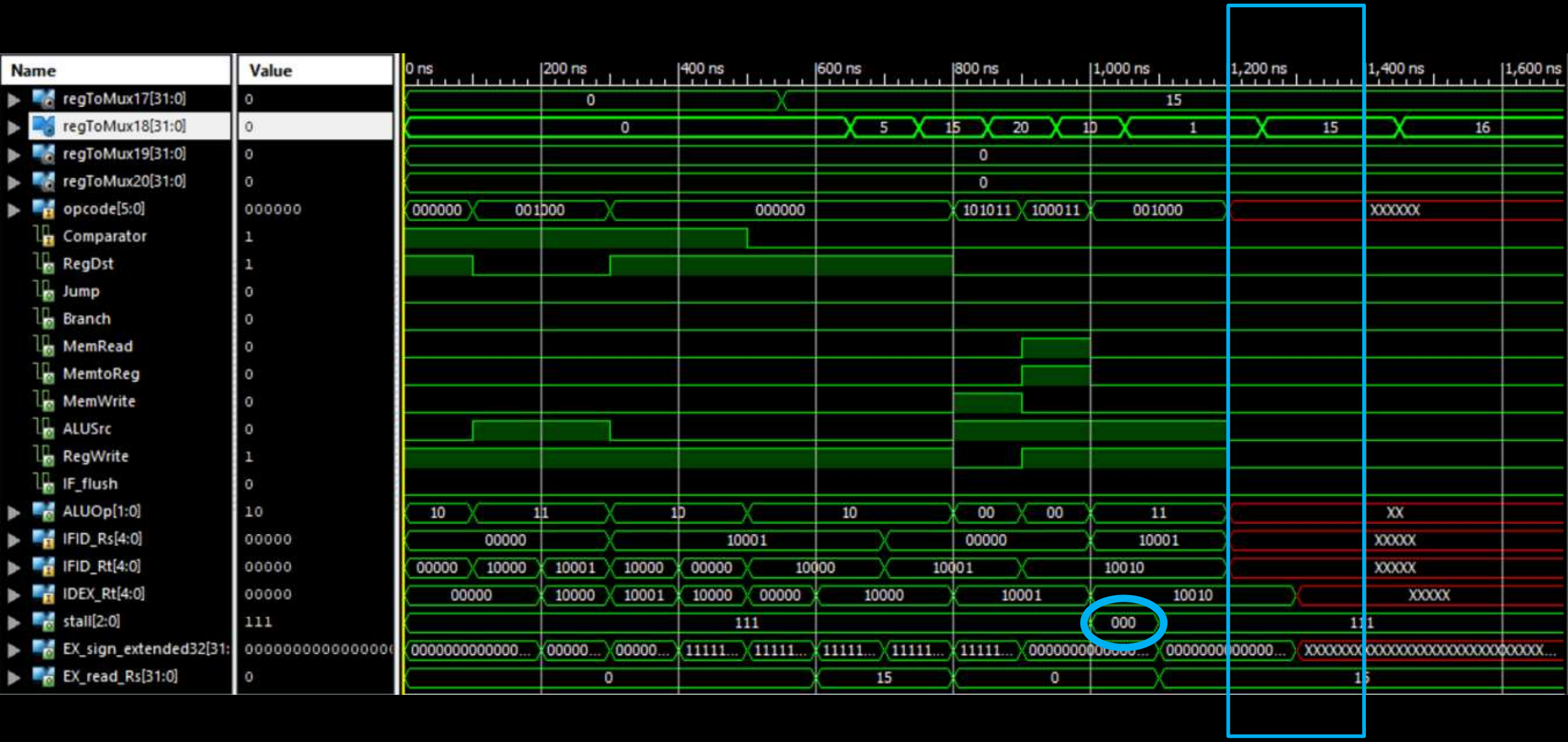
Top Module Testbench

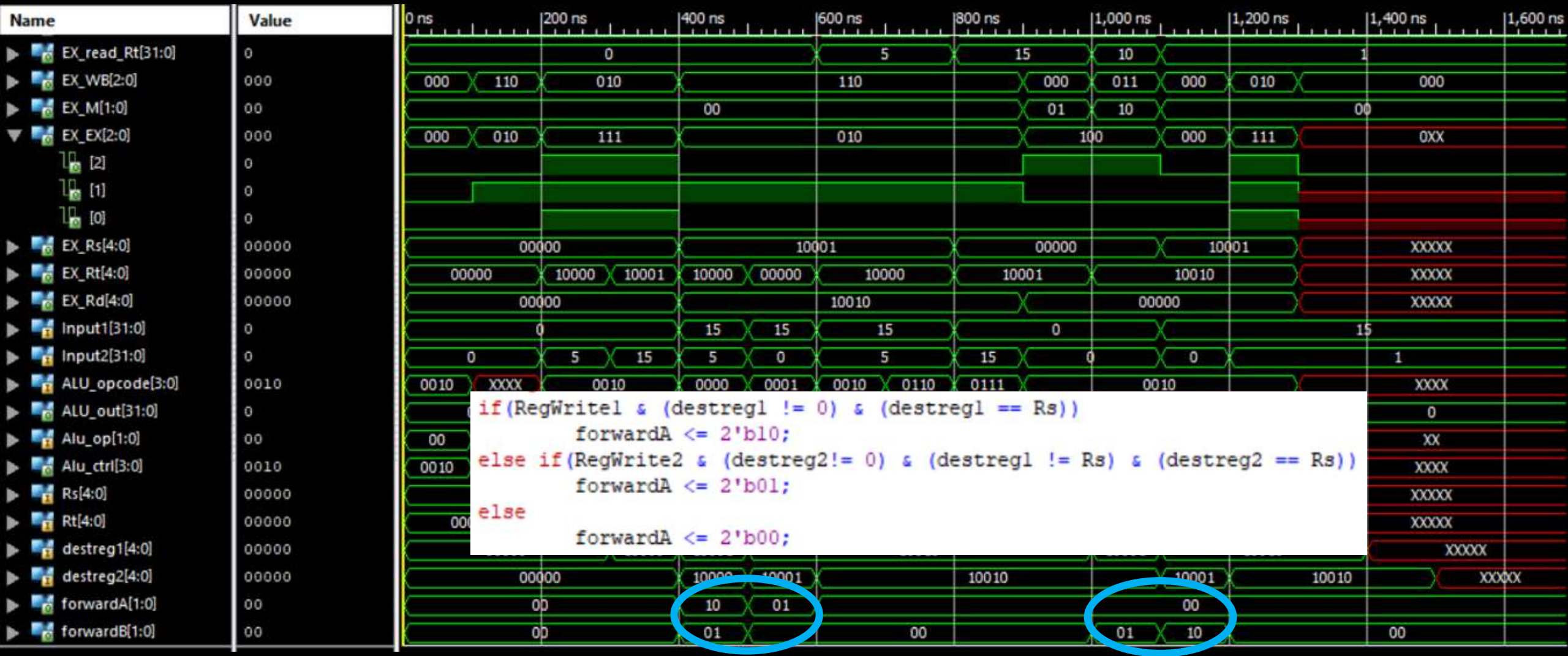
```
`timescale 1ns / 1ps


module MIPS_pip_tb;

	// Inputs
	reg CLK;
	reg RST;

	// Instantiate the Unit Under Test (UUT)
	MIPS_pipelined uut (
		.CLK(CLK),
		.RST(RST)
	);

	always #1000 CLK = ~CLK;
	initial begin
		// Initialize Inputs
		CLK = 0;
		RST = 1;

		// Wait 100 ns for global reset to finish
		#100;
      RST = 0;
		#500;
		// Add stimulus here

	end

endmodule
```

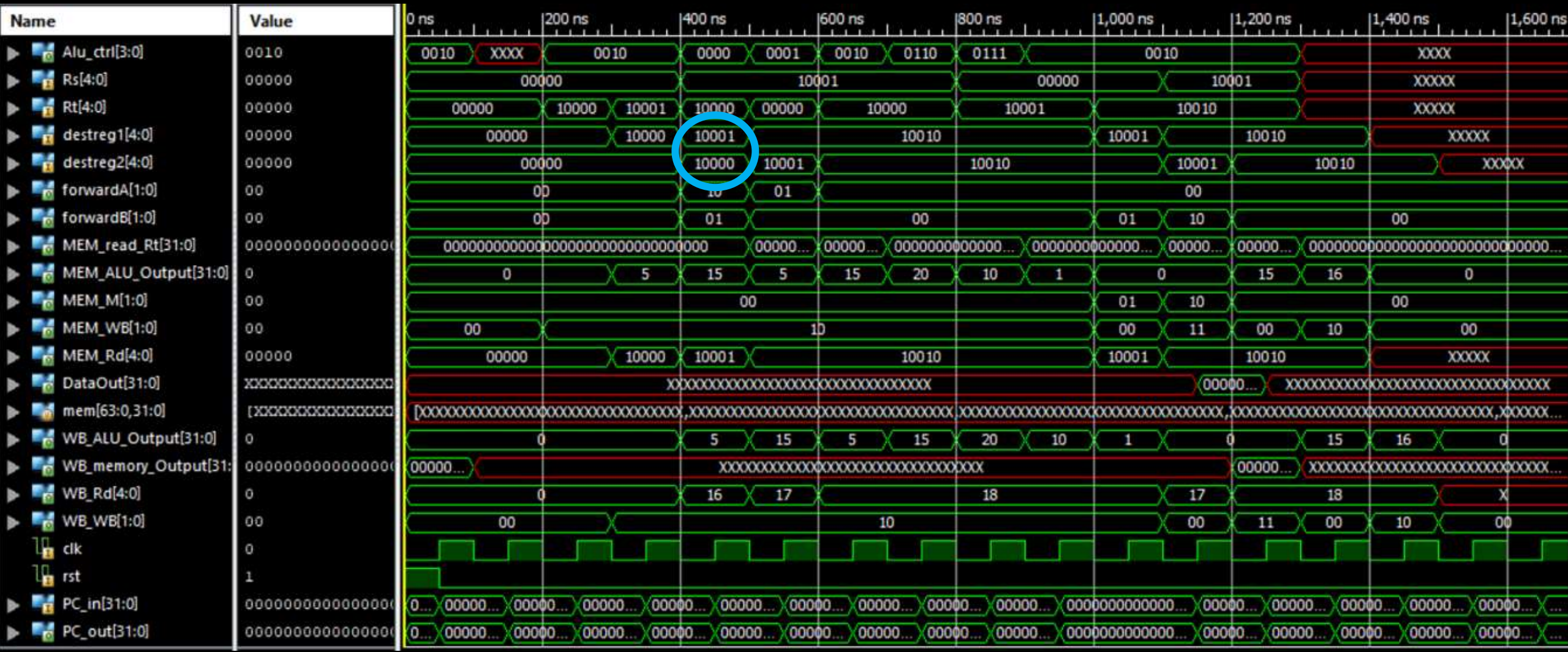
MIPS Code to be Tested

```
0     nop
4     addi $s0, $zero, 0x5      #s0 = 5
8     addi $s1, $zero, 0xF      #s1 = 15
12    and $s2, $s1, $s0         #s2 = 5
16    or  $s2, $s1, $zero       #s2 = 15
20    add $s2, $s1, $s0         #s2 = 20
24    sub $s2, $s1, $s0         #s2 = 10
28    slt $s2, $zero, $s1       #s2 = 1
32    sw  $s1, 0x0($zero)
36    lw  $s2, 0x0($zero)       #s2 = 15
40    addi $s2, $s1, 0x1        #s2 = 16
```


Name
Value
CLK
RST
en
PC_out[31:0]
instruction[31:0]
instruction[31:0]
ID_instruction[31:0]
ID_PC[31:0]
regToMux16[31:0]
regToMux17[31:0]
regToMux18[31:0]
regToMux19[31:0]
regToMux20[31:0]
opcode[5:0]
Comparator
RegDst
Jump
Branch
MemRead
MemtoReg
MemWrite
ALUSrc
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns
1,600 ns


Name
Value
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns
1,600 ns
regToMux17[31:0]
0
regToMux18[31:0]
0
regToMux19[31:0]
0
regToMux20[31:0]
0
opcode[5:0]
000000
Comparator
1
RegDst
1
Jump
0
Branch
0
MemRead
0
MemtoReg
0
MemWrite
0
ALUSrc
0
RegWrite
1
IF_flush
0
ALUOp[1:0]
10
IFID_Rs[4:0]
00000
IFID_Rt[4:0]
00000
IDEX_Rt[4:0]
00000
stall[2:0]
111
EX_sign_extended32[31:
EX_read_Rs[31:0]
0

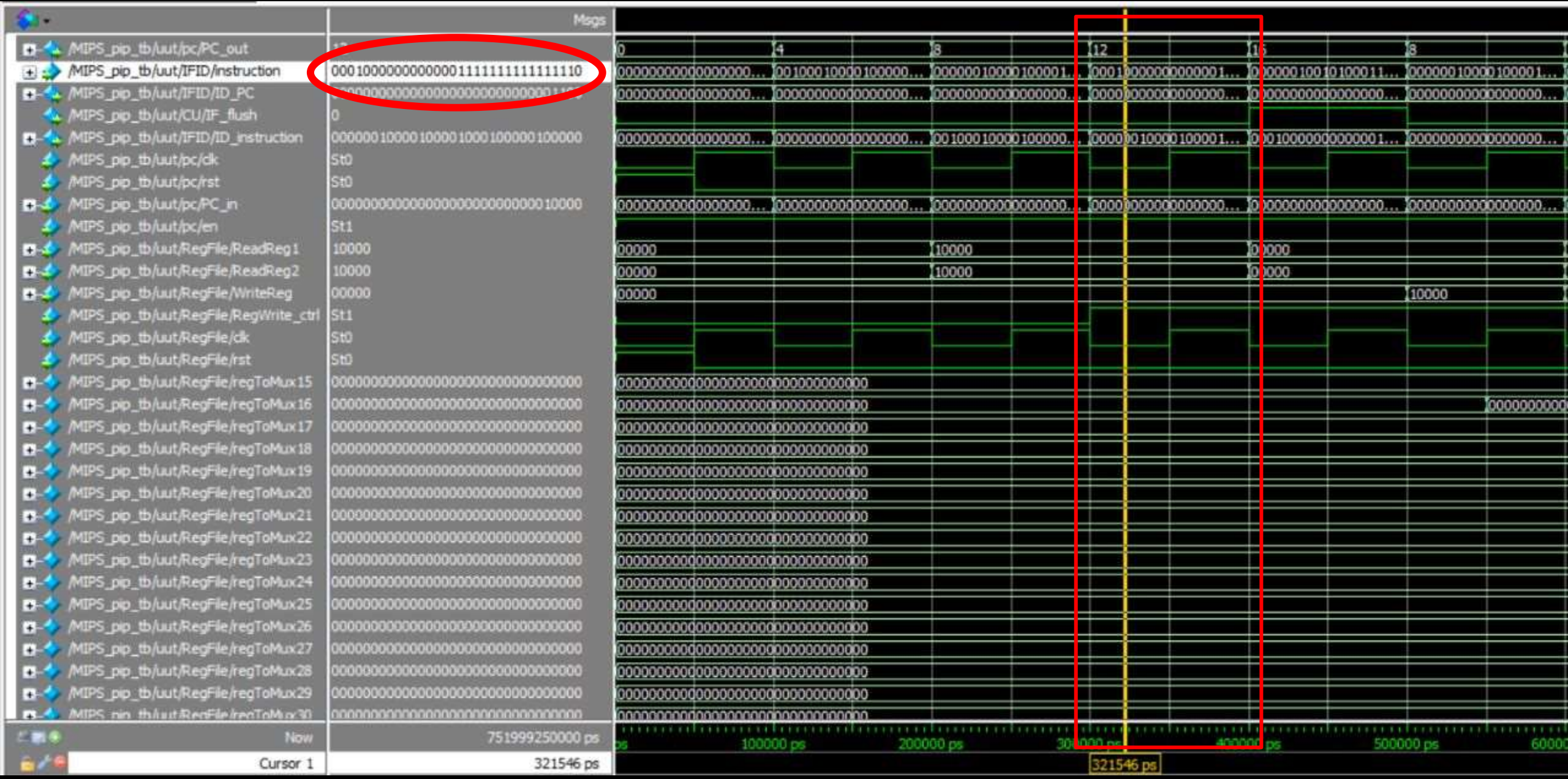

Name
Value
EX_read_Rt[31:0]
EX_WB[2:0]
EX_M[1:0]
EX_EX[2:0]
[2]
[1]
[0]
EX_Rs[4:0]
EX_Rt[4:0]
EX_Rd[4:0]
Input1[31:0]
Input2[31:0]
ALU_opcode[3:0]
ALU_out[31:0]
Alu_op[1:0]
Alu_ctrl[3:0]
Rs[4:0]
Rt[4:0]
destreg1[4:0]
destreg2[4:0]
forwardA[1:0]
forwardB[1:0]
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns
1,600 ns
if(RegWrite1 & (destreg1 != 0) & (destreg1 == Rs))
forwardA <= 2'b10;
else if(RegWrite2 & (destreg2!= 0) & (destreg1 != Rs) & (destreg2 == Rs))
forwardA <= 2'b01;
else
forwardA <= 2'b00;

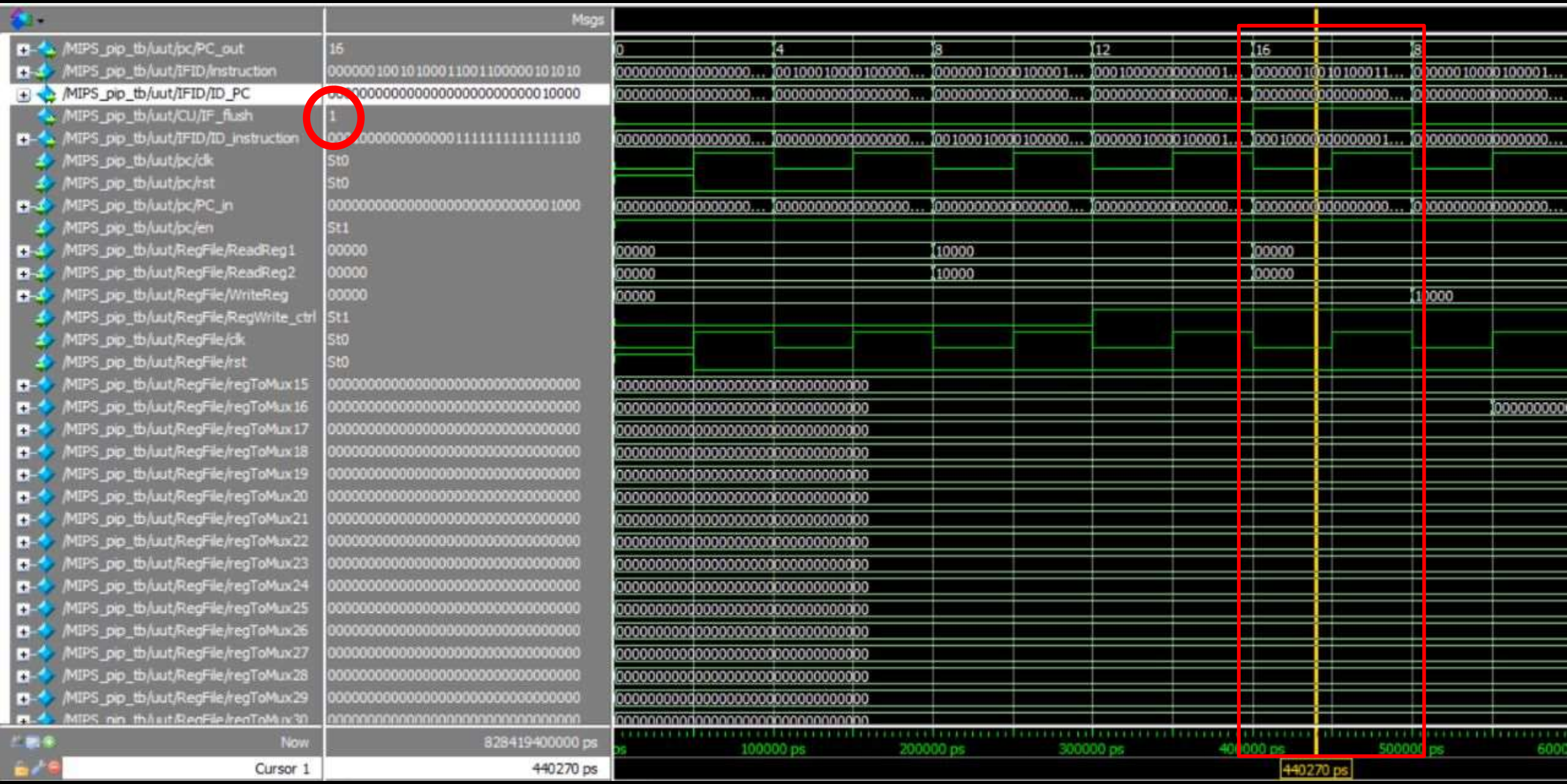

Name
Value
Alu_ctrl[3:0]
Rs[4:0]
Rt[4:0]
destreg1[4:0]
destreg2[4:0]
forwardA[1:0]
forwardB[1:0]
MEM_read_Rt[31:0]
MEM_ALU_Output[31:0]
MEM_M[1:0]
MEM_WB[1:0]
MEM_Rd[4:0]
DataOut[31:0]
mem[63:0,31:0]
WB_ALU_Output[31:0]
WB_memory_Output[31:
WB_Rd[4:0]
WB_WB[1:0]
clk
rst
PC_in[31:0]
PC_out[31:0]
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns
1,600 ns

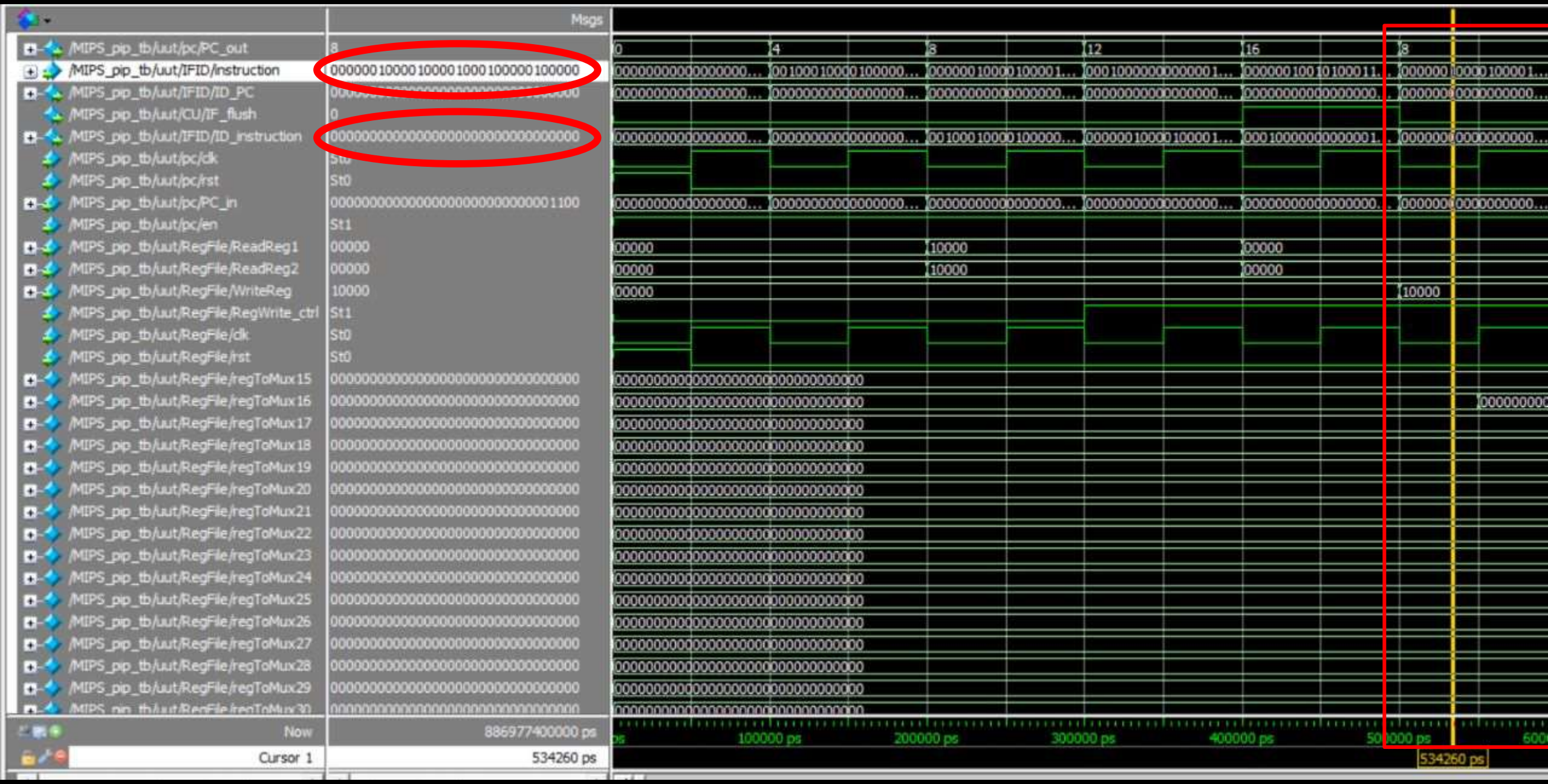

Name
Value
38 us
40 us
42 us
44 us
46 us
48 us
50 us
52 us
[21,31:0]
[20,31:0]
[19,31:0]
[18,31:0]
[17,31:0]
[16,31:0]
[15,31:0]
[14,31:0]
[13,31:0]
[12,31:0]
[11,31:0]
[10,31:0]
[9,31:0]
[8,31:0]
[7,31:0]
[6,31:0]
[5,31:0]
[4,31:0]
[3,31:0]
[2,31:0]
[1,31:0]
[0,31:0]
15
X
15

MIPS Code to be Tested

```
12          beq $s2, $s1, 0xFFFE
```

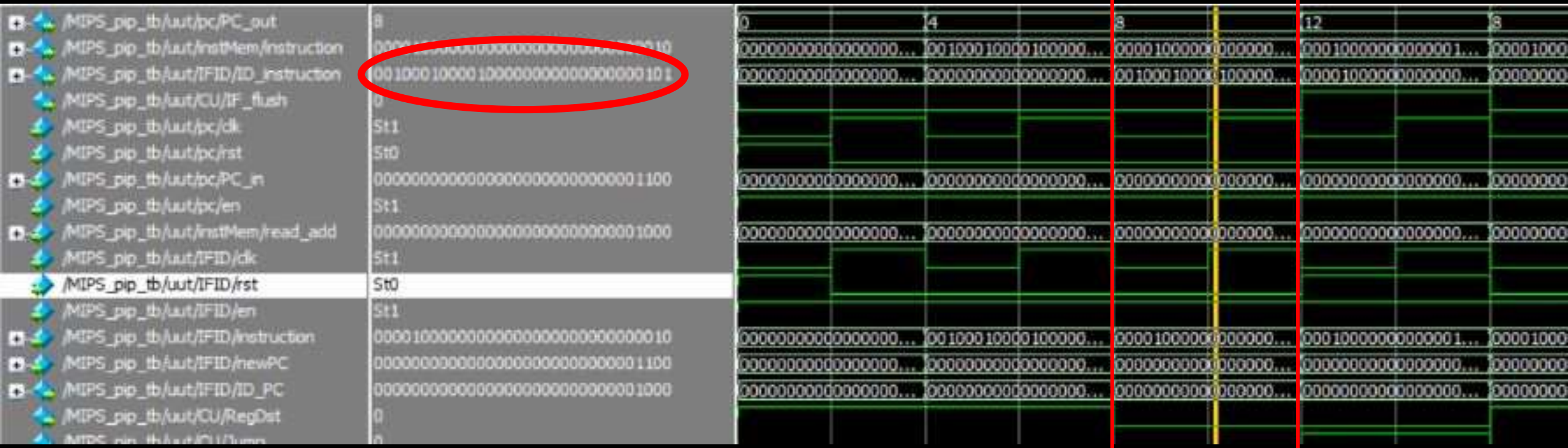

Msgs
/MIPS_pip_tb/uut/pc/PC_out
/MIPS_pip_tb/uut/IFID/instruction
/MIPS_pip_tb/uut/IFID/ID_PC
/MIPS_pip_tb/uut/CU/IF_flush
/MIPS_pip_tb/uut/IFID/ID_instruction
/MIPS_pip_tb/uut/pc/clk
/MIPS_pip_tb/uut/pc/rst
/MIPS_pip_tb/uut/pc/PC_in
/MIPS_pip_tb/uut/pc/en
/MIPS_pip_tb/uut/RegFile/ReadReg1
/MIPS_pip_tb/uut/RegFile/ReadReg2
/MIPS_pip_tb/uut/RegFile/WriteReg
/MIPS_pip_tb/uut/RegFile/RegWrite_ctrl
/MIPS_pip_tb/uut/RegFile/clk
/MIPS_pip_tb/uut/RegFile/rst
/MIPS_pip_tb/uut/RegFile/regToMux15
/MIPS_pip_tb/uut/RegFile/regToMux16
/MIPS_pip_tb/uut/RegFile/regToMux17
/MIPS_pip_tb/uut/RegFile/regToMux18
/MIPS_pip_tb/uut/RegFile/regToMux19
/MIPS_pip_tb/uut/RegFile/regToMux20
/MIPS_pip_tb/uut/RegFile/regToMux21
/MIPS_pip_tb/uut/RegFile/regToMux22
/MIPS_pip_tb/uut/RegFile/regToMux23
/MIPS_pip_tb/uut/RegFile/regToMux24
/MIPS_pip_tb/uut/RegFile/regToMux25
/MIPS_pip_tb/uut/RegFile/regToMux26
/MIPS_pip_tb/uut/RegFile/regToMux27
/MIPS_pip_tb/uut/RegFile/regToMux28
/MIPS_pip_tb/uut/RegFile/regToMux29
000100000000000011111111111111110
000000 10000 10000 1000 100000 100000
St0
St0
St1
10000
10000
00000
St1
St0
St0
Now
751999250000 ps
Cursor 1
321546 ps
100000 ps
200000 ps
500000 ps
321546 ps

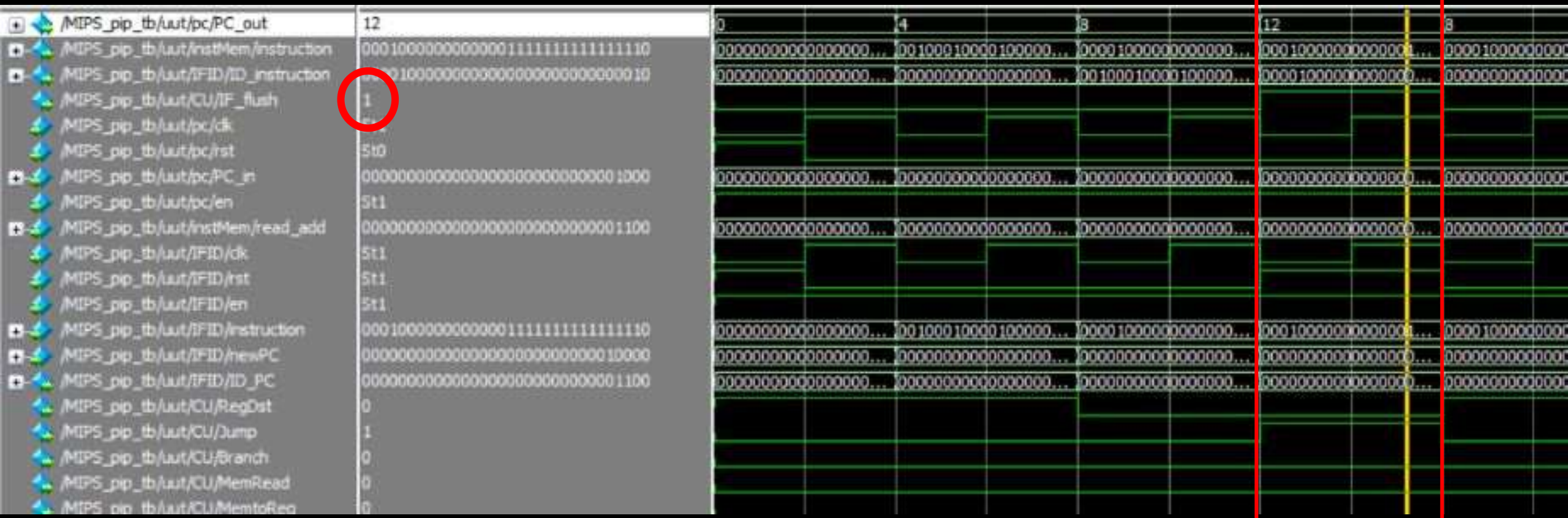

Msgs
/MIPS_pip_tb/uut/pc/PC_out
/MIPS_pip_tb/uut/IFID/Instruction
/MIPS_pip_tb/uut/IFID/ID_PC
/MIPS_pip_tb/uut/CU/IF_flush
/MIPS_pip_tb/uut/IFID/ID_instruction
/MIPS_pip_tb/uut/pc/clk
/MIPS_pip_tb/uut/pc/rst
/MIPS_pip_tb/uut/pc/PC_in
/MIPS_pip_tb/uut/pc/en
/MIPS_pip_tb/uut/RegFile/ReadReg1
/MIPS_pip_tb/uut/RegFile/ReadReg2
/MIPS_pip_tb/uut/RegFile/WriteReg
/MIPS_pip_tb/uut/RegFile/RegWrite_ctrl
/MIPS_pip_tb/uut/RegFile/clk
/MIPS_pip_tb/uut/RegFile/rst
/MIPS_pip_tb/uut/RegFile/regToMux15
/MIPS_pip_tb/uut/RegFile/regToMux16
/MIPS_pip_tb/uut/RegFile/regToMux17
/MIPS_pip_tb/uut/RegFile/regToMux18
/MIPS_pip_tb/uut/RegFile/regToMux19
/MIPS_pip_tb/uut/RegFile/regToMux20
/MIPS_pip_tb/uut/RegFile/regToMux21
/MIPS_pip_tb/uut/RegFile/regToMux22
/MIPS_pip_tb/uut/RegFile/regToMux23
/MIPS_pip_tb/uut/RegFile/regToMux24
/MIPS_pip_tb/uut/RegFile/regToMux25
/MIPS_pip_tb/uut/RegFile/regToMux26
/MIPS_pip_tb/uut/RegFile/regToMux27
/MIPS_pip_tb/uut/RegFile/regToMux28
/MIPS_pip_tb/uut/RegFile/regToMux29
16
1
St0
St1
00000
Now
828419400000 ps
Cursor 1
440270 ps
100000 ps
200000 ps
300000 ps
500000 ps
440270 ps

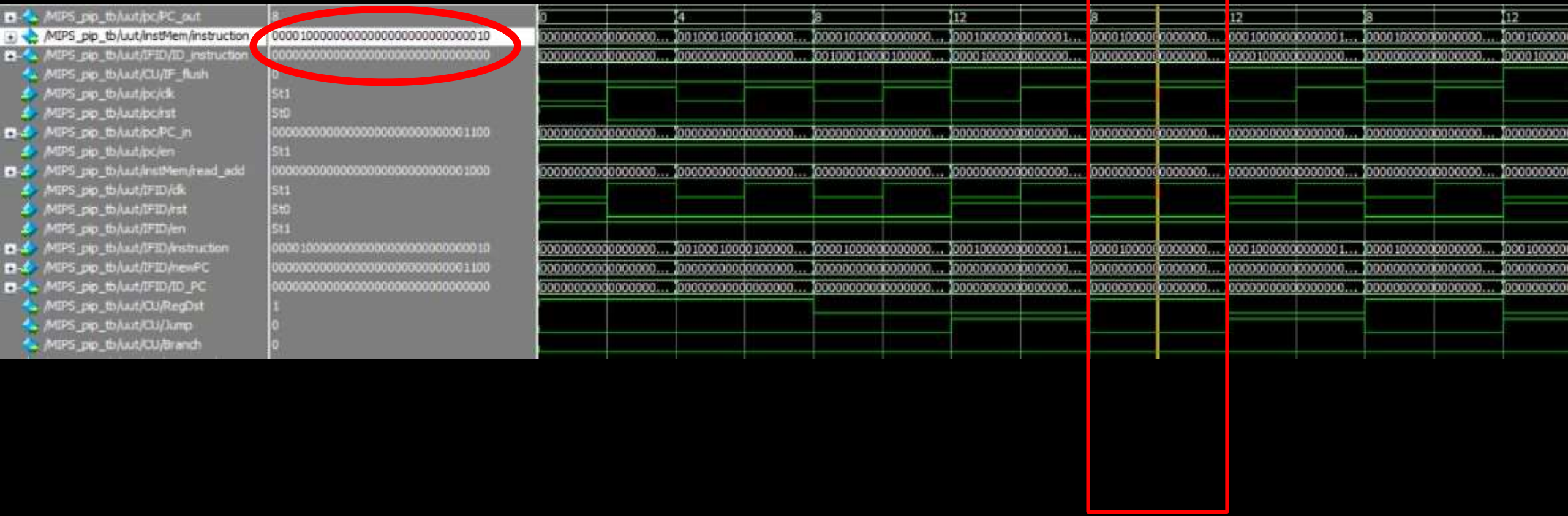

Msgs
/MIPS_pip_tb/uut/pc/PC_out
/MIPS_pip_tb/uut/IFID/instruction
/MIPS_pip_tb/uut/IFID/ID_PC
/MIPS_pip_tb/uut/CU/IF_flush
/MIPS_pip_tb/uut/IFID/ID_instruction
/MIPS_pip_tb/uut/pc/clk
/MIPS_pip_tb/uut/pc/rst
/MIPS_pip_tb/uut/pc/PC_in
/MIPS_pip_tb/uut/pc/en
/MIPS_pip_tb/uut/RegFile/ReadReg1
/MIPS_pip_tb/uut/RegFile/ReadReg2
/MIPS_pip_tb/uut/RegFile/WriteReg
/MIPS_pip_tb/uut/RegFile/RegWrite_ctrl
/MIPS_pip_tb/uut/RegFile/clk
/MIPS_pip_tb/uut/RegFile/rst
/MIPS_pip_tb/uut/RegFile/regToMux15
/MIPS_pip_tb/uut/RegFile/regToMux16
/MIPS_pip_tb/uut/RegFile/regToMux17
/MIPS_pip_tb/uut/RegFile/regToMux18
/MIPS_pip_tb/uut/RegFile/regToMux19
/MIPS_pip_tb/uut/RegFile/regToMux20
/MIPS_pip_tb/uut/RegFile/regToMux21
/MIPS_pip_tb/uut/RegFile/regToMux22
/MIPS_pip_tb/uut/RegFile/regToMux23
/MIPS_pip_tb/uut/RegFile/regToMux24
/MIPS_pip_tb/uut/RegFile/regToMux25
/MIPS_pip_tb/uut/RegFile/regToMux26
/MIPS_pip_tb/uut/RegFile/regToMux27
/MIPS_pip_tb/uut/RegFile/regToMux28
/MIPS_pip_tb/uut/RegFile/regToMux29
8
000000 10000 10000 1000 100000 100000
St0
St1
00000
10000
Now
886977400000 ps
Cursor 1
534260 ps
100000 ps
200000 ps
300000 ps
400000 ps
534260 ps

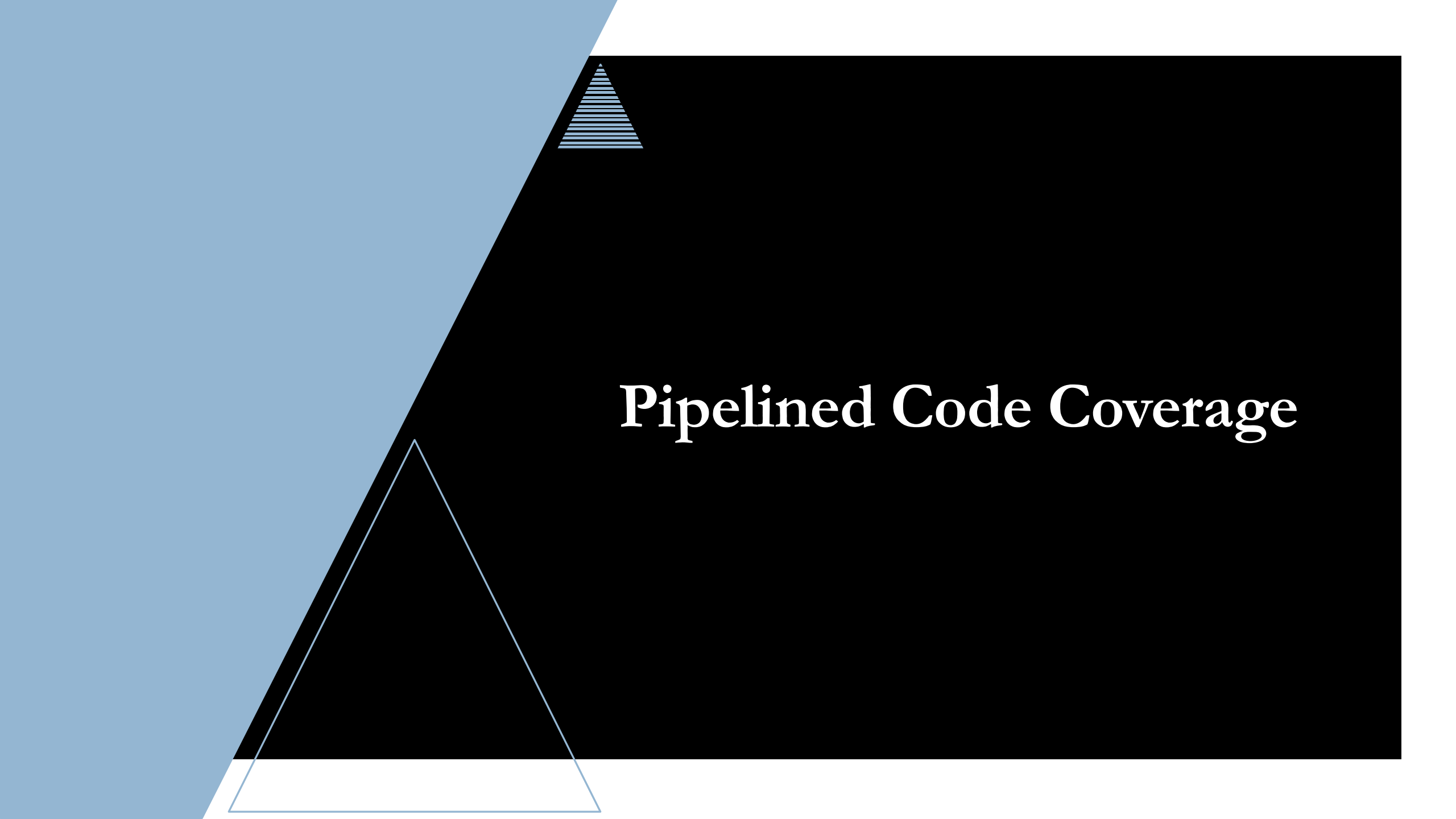
MIPS Code to be Tested

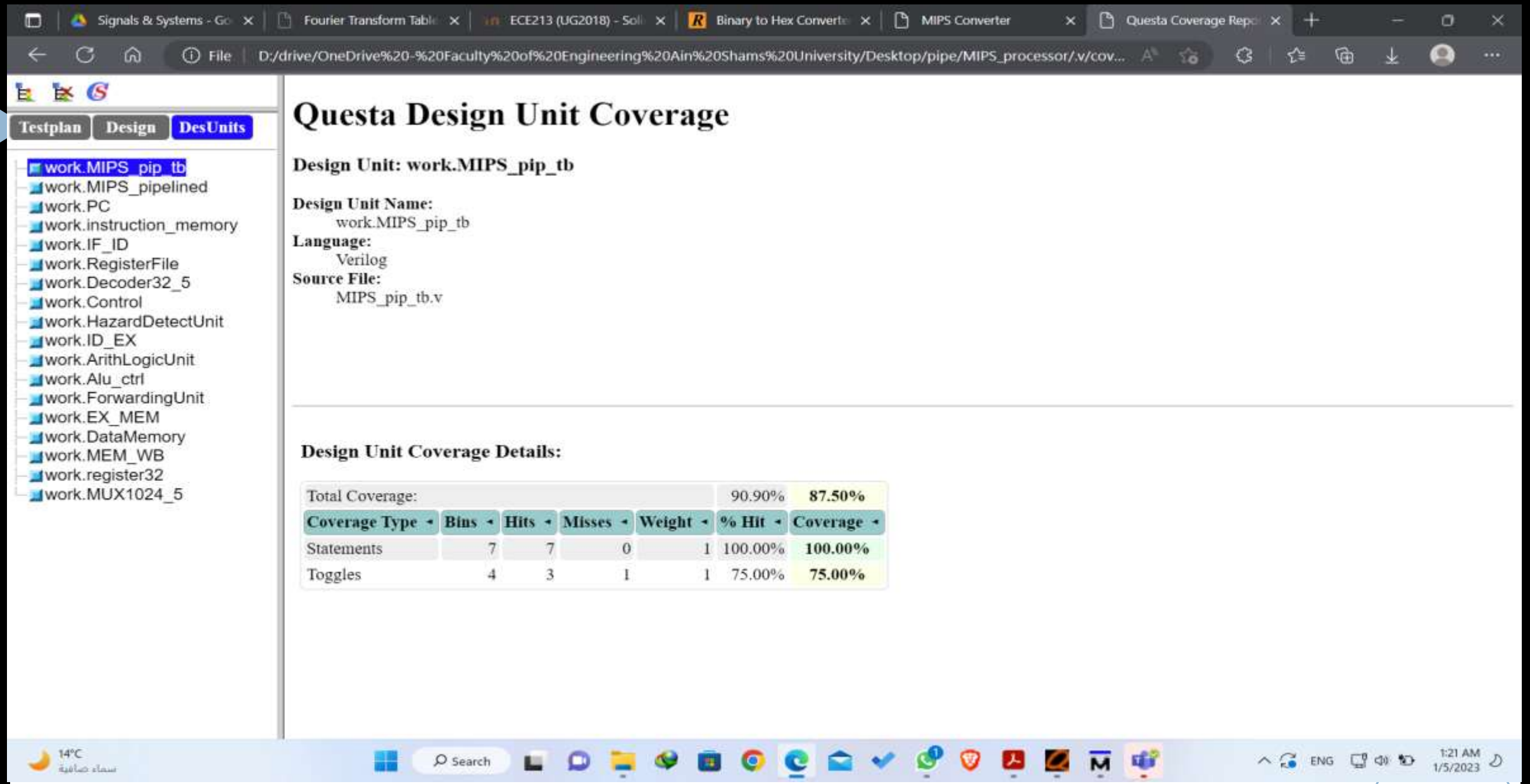
```
8          j 0x2
```


/MIPS_pip_tb/uut/pc/PC_out
/MIPS_pip_tb/uut/instMem/instruction
/MIPS_pip_tb/uut/IFID/ID_instruction
/MIPS_pip_tb/uut/CU/IF_flush
/MIPS_pip_tb/uut/pc/clk
/MIPS_pip_tb/uut/pc/rst
/MIPS_pip_tb/uut/pc/PC_in
/MIPS_pip_tb/uut/pc/en
/MIPS_pip_tb/uut/instMem/read_add
/MIPS_pip_tb/uut/IFID/clk
/MIPS_pip_tb/uut/IFID/rst
/MIPS_pip_tb/uut/IFID/en
/MIPS_pip_tb/uut/IFID/instruction
/MIPS_pip_tb/uut/IFID/newPC
/MIPS_pip_tb/uut/IFID/ID_PC
/MIPS_pip_tb/uut/CU/RegDst
St1
St0
0
4
8
12


/MIPS_pip_tb/uut/pc/PC_out
12
/MIPS_pip_tb/uut/instMem/instruction
/MIPS_pip_tb/uut/IFID/ID_instruction
/MIPS_pip_tb/uut/CU/IF_flush
1
/MIPS_pip_tb/uut/pc/clk
/MIPS_pip_tb/uut/pc/rst
St0
/MIPS_pip_tb/uut/pc/PC_in
/MIPS_pip_tb/uut/pc/en
St1
/MIPS_pip_tb/uut/instMem/read_add
/MIPS_pip_tb/uut/IFID/clk
St1
/MIPS_pip_tb/uut/IFID/rst
St1
/MIPS_pip_tb/uut/IFID/en
St1
/MIPS_pip_tb/uut/IFID/instruction
/MIPS_pip_tb/uut/IFID/newPC
/MIPS_pip_tb/uut/IFID/ID_PC
/MIPS_pip_tb/uut/CU/RegDst
0
/MIPS_pip_tb/uut/CU/Jump
1
/MIPS_pip_tb/uut/CU/Branch
0
/MIPS_pip_tb/uut/CU/MemRead
0
0
4
8
12
8


/MIPS_pip_tb/uut/pc/PC_out
/MIPS_pip_tb/uut/instMem/instruction
/MIPS_pip_tb/uut/IFID/ID_instruction
/MIPS_pip_tb/uut/CU/IF_flush
/MIPS_pip_tb/uut/pc/clk
/MIPS_pip_tb/uut/pc/rst
/MIPS_pip_tb/uut/pc/PC_in
/MIPS_pip_tb/uut/pc/en
/MIPS_pip_tb/uut/instMem/read_add
/MIPS_pip_tb/uut/IFID/clk
/MIPS_pip_tb/uut/IFID/rst
/MIPS_pip_tb/uut/IFID/en
/MIPS_pip_tb/uut/IFID/instruction
/MIPS_pip_tb/uut/IFID/newPC
/MIPS_pip_tb/uut/IFID/ID_PC
/MIPS_pip_tb/uut/CU/RegDst
/MIPS_pip_tb/uut/CU/Jump
/MIPS_pip_tb/uut/CU/Branch
8
St1
St0
St1
St0
St1
1
0
0


Pipelined Code Coverage


Signals & Systems - Go
Fourier Transform Table
ECE213 (UG2018) - Soli
Binary to Hex Converte
MIPS Converter
Questa Coverage Repo
File
D:/drive/OneDrive%20-%20Faculty%20of%20Engineering%20Ain%20Shams%20University/Desktop/pipe/MIPS_processor/.v/cov...
Testplan
Design
DesUnits
work.MIPS_pip_tb
work.MIPS_pipelined
work.PC
work.instruction_memory
work.IF_ID
work.RegisterFile
work.Decoder32_5
work.Control
work.HazardDetectUnit
work.ID_EX
work.ArithLogicUnit
work.Alu_ctrl
work.ForwardingUnit
work.EX_MEM
work.DataMemory
work.MEM_WB
work.register32
work.MUX1024_5
Questa Design Unit Coverage
Design Unit: work.MIPS_pip_tb
Design Unit Name:
work.MIPS_pip_tb
Language:
Verilog
Source File:
MIPS_pip_tb.v
Design Unit Coverage Details:
Total Coverage: 90.90% 87.50%
Coverage Type | Bins | Hits | Misses | Weight | % Hit | Coverage
Statements 7 7 0 1 100.00% 100.00%
Toggles 4 3 1 1 75.00% 75.00%
14°C
Search
ENG
1:21 AM
1/5/2023

Testplan | Design | DesUnits

- work.MIPS_pip_tb
- work.MIPS_pipelined
- work.PC
- work.instruction_memory
- work.IF_ID
- work.RegisterFile
- work.Decoder32_5
- work.Control
- work.HazardDetectUnit
- work.ID_EX
- work.ArithLogicUnit
- work.Alu_ctrl
- work.ForwardingUnit
- work.EX_MEM
- work.DataMemory
- work.MEM_WB
- work.register32
- work.MUX1024_5

Questa Design Unit Coverage

Design Unit: work.MIPS_pipelined

Design Unit Name:
work.MIPS_pipelined
Language:
Verilog
Source File:
MIPS_piplined.v

Design Unit Coverage Details:

Total Coverage:					22.79%	65.66%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	27	25	2	1	92.59%	**92.59%**
Branches	22	18	4	1	81.81%	**81.81%**
FEC Expressions	3	2	1	1	66.66%	**66.66%**
Toggles	2694	581	2113	1	21.56%	**21.56%**