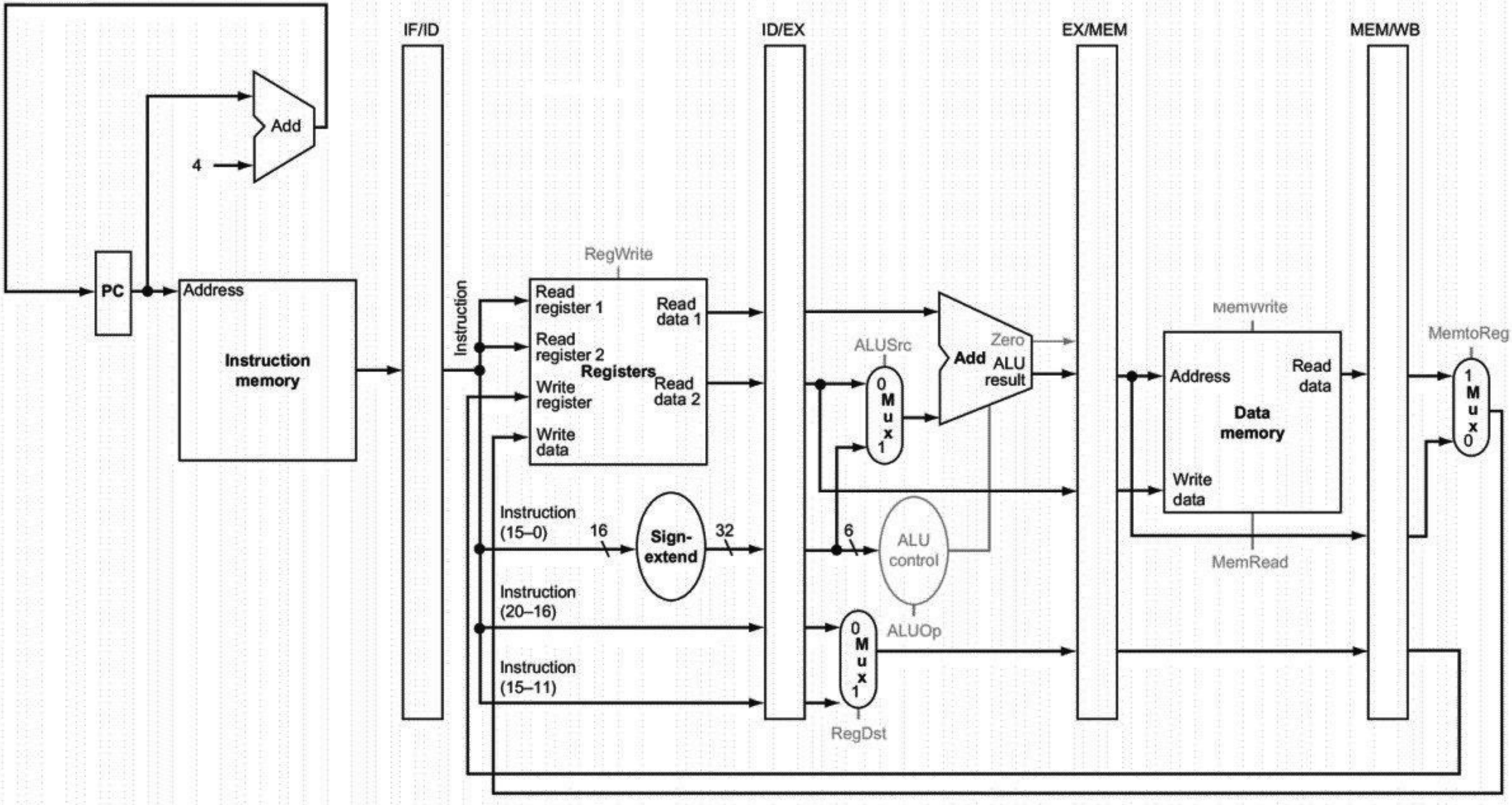

Design & Simulation of 32-bit five stage pipelined processor

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------


module tb_five;


reg clk=0, reset;


Pipelined_5stage UUT(clk, reset);

        initial
                begin
                reset=0; #20;
                reset=1; #400;
                end

        always

                begin
                        clk=~clk; #10;
                end


endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------


module ID_EX (datain1, datain2, datain_sign, IRin1, IRin2, dataout1, dataout2, dataout_sign, IRout1, IRout2, clk, reset);

input [31:0] datain1, datain2, datain_sign;
input [4:0] IRin1, IRin2;
input clk,reset;
output reg [31:0] dataout1, dataout2, dataout_sign;
output reg [4:0] IRout1, IRout2;

        always @ (posedge clk, negedge reset)
                begin
                if (reset==0)
                {dataout1, dataout2, dataout_sign,IRout1, IRout2}<=0;
                else
                {dataout1, dataout2, dataout_sign,IRout1, IRout2}<= {datain1, datain2, datain_sign, IRin1, IRin2};
                end
endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------


module MEM_WB (datain1, datain2, datain3, dataout1, dataout2, dataout3, clk, reset);

input [31:0] datain1, datain2;
input [4:0] datain3;
input reset, clk;
output reg [31:0] dataout1, dataout2;
output reg [4:0] dataout3;

        always @ (posedge clk, negedge reset)
                begin
                if (reset==0)
                {dataout1, dataout2, dataout3}<= 0;
                else
                {dataout1, dataout2, dataout3}<= {datain1, datain2,datain3};
                end


endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------


module sign_extend(data_in, data_out);

input [15:0] data_in;
output [31:0] data_out;


 assign data_out = data_in[15] ? {16'b1111_1111_1111_1111, data_in} : {16'b0000_0000_0000_0000,data_in};

endmodule


module tb;

reg [15:0] data_in;
wire [31:0] data_out;

sign_extend UUT (data_in, data_out);


        initial
        begin

        data_in=16'h1000; #10;
        data_in=16'h0100; #10;
        data_in=$random; #10;
        data_in=$random; #10;
        data_in=$random; #10;
        data_in=$random; #10;
        data_in=$random; #10;
        data_in=$random; #10;

        end

endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------


module IF_ID (data_in, data_out, clk, reset);

input [31:0] data_in;
input clk, reset;
output reg [31:0] data_out;


        always @ (posedge clk, negedge reset)
                begin
                if (reset==0)
                data_out<=0;
                else
                data_out<=data_in;
                end


endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------


module control_unit (IR, RegDst, ALUSrc, Memto_Reg, Reg_Write, Mem_Read, Mem_Write, Branch, ALUop1, ALUop0);
input [5:0] IR;
output reg RegDst, ALUSrc, Memto_Reg, Reg_Write, Mem_Read, Mem_Write, Branch, ALUop1, ALUop0;


        always @ (*)
        begin
        case (IR)
         0: {RegDst, ALUSrc, Memto_Reg, Reg_Write, Mem_Read, Mem_Write, Branch, ALUop1, ALUop0} =9'b100_100_010;
        35: {RegDst, ALUSrc, Memto_Reg, Reg_Write, Mem_Read, Mem_Write, Branch, ALUop1, ALUop0} =9'b011_110_000;
        43: {RegDst, ALUSrc, Memto_Reg, Reg_Write, Mem_Read, Mem_Write, Branch, ALUop1, ALUop0} =9'b010_001_000;
        4 : {RegDst, ALUSrc, Memto_Reg, Reg_Write, Mem_Read, Mem_Write, Branch, ALUop1, ALUop0} =9'b000_000_101;
        default: {RegDst, ALUSrc, Memto_Reg, Reg_Write, Mem_Read, Mem_Write, Branch, ALUop1, ALUop0} = 0;
        endcase
        end
endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------

module Instruction_Memory(output reg [31:0] data_out, input [4:0] address);


always@(address)begin
        case(address)

                /*5'h00: #11 data_out=32'b000000_00010_00011_00001_0000000_0000;                //ld 7  Pr#3a
                5'h01: #11 data_out=32'b000000_00101_00110_00100_0000000_0010;          //st 800
                5'h02: #11 data_out=32'b000000_01000_01001_00111_0000000_0100;          //add 8
                5'h03: #11 data_out=2_904_069;          //st 801
                5'h04: #11 data_out=9_535_498; */


                5'h00: #11 data_out=32'b000000_00010_00011_00001_00000100000;           //ld 7  Pr#3a
                5'h01: #11 data_out=32'b000000_000010011_000100000_00100010;            //st 800
                5'h02: #11 data_out=32'b000000_00001001000011100000100100;              //add 8
                5'h03: #11 data_out=32'b00000000111001110101000000100101;               //st 801
                5'h04: #11 data_out=32'b00000001010001001000000000100111;

                default: #7 data_out=32'h0000_0000;
        endcase
end
endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------


module EX_MEM (ALUin, datain, muxin, ALUout, dataout, mout, clk, reset);

input [31:0] ALUin, datain;
input [4:0] muxin;
input clk, reset;
output reg [31:0] ALUout, dataout;
output reg [4:0] mout;

        always @ (posedge clk, negedge reset)
                begin
                if (reset==0)
                {ALUout, dataout, mout}<=0;
                else
                {ALUout, dataout, mout}<={ALUin, datain, muxin};
                end
endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------

module Mux_32bits (A, B, sel, Y);

input [31:0] A, B;
input sel;
output [31:0] Y;

        assign Y = sel ? B : A;


endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------

module Pipelined_5stage (clk, reset);


input clk, reset;


wire [31:0] data_out, IF_out, sign_out, read_data1, read_data2, dataout1, dataout2, dataout_sign;
wire Reg_Write, Zero, ALUop1, ALUop0, Mem_Read, Mem_Write, RegDst, ALUSrc, Memto_Reg;
wire [4:0] IRout1, IRout2, Y, dataout33, mout;
wire [31:0] Y_mux, Result, ALUout, dataout, ReadData, dataout11,dataout22 ;
wire [3:0] ALUctl;
wire [31:0] f_data;
wire [15:0] out;

 PC Aa(out, clk, reset);

 Instruction_Memory Ab (data_out, out[4:0]);

 IF_ID Ac (data_out, IF_out, clk, reset);

 Register_file Ad (read_data1, read_data2, IF_out[25:21], IF_out[20:16], dataout33, f_data, Reg_Write, clk, reset);

 sign_extend Ae(IF_out[15:0], sign_out);

 ID_EX Af (read_data1, read_data2, sign_out, IF_out[20:16], IF_out[15:11], dataout1, dataout2, dataout_sign, IRout1, IRout
2, clk, reset);

 Mux_32bits Ag (dataout2, dataout_sign, ALUSrc, Y_mux);

 Mux_5bits Ah (IRout1, IRout2, RegDst, Y);

 ALU_control Ai ({ALUop1, ALUop0, dataout_sign[3:0]}, ALUctl);

 ALU_32bits Aj (Result, Zero, Overflow, CarryOut, dataout1, Y_mux, ALUctl );

 EX_MEM Ak(Result, dataout2, Y, ALUout, dataout, mout, clk, reset);

 DataMemory Al (ReadData, ALUout[15:0], dataout, Mem_Read, Mem_Write, ALUout);
```

```
 MEM_WB Am(ReadData, ALUout, mout, dataout11, dataout22, dataout33, clk, reset);
 Mux_32bits An(dataout11, dataout22, Memto_Reg, f_data);
 control_unit Ap (IF_out[31:26], RegDst, ALUSrc, Memto_Reg, Reg_Write, Mem_Read, Mem_Write, Branch, ALUop1, ALUop0);
endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------

module ALU_control (IR, ALUctl);

input [5:0] IR;
//input ALUop0, ALUop1;
output reg [3:0] ALUctl;

        always @ (*)
        begin
                case (IR)
                6'b100_000: ALUctl = 4'b0010; //ADD
                6'b100_010: ALUctl = 4'b0110; // SUB
                6'b100_100: ALUctl = 4'b0000; // AND
                6'b100_101: ALUctl = 4'b0001; //OR
                6'b101_010: ALUctl = 4'b1100; //NOR
                endcase
        end

endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------


module Register_file (read_data1, read_data2, addr_read1, addr_read2, addr_write, write_data, we, clk, reset);

input [4:0] addr_read1, addr_read2, addr_write;
input clk, reset,we;
input [31:0] write_data;
output [31:0] read_data1, read_data2;

reg [31:0] reg_bank [31:0];
integer i;

assign read_data1 = reg_bank[addr_read1];
assign read_data2 = reg_bank[addr_read2];

        always @ (posedge clk, negedge reset)
        begin
        if (reset ==0 )
                begin
                reg_bank[0]=0;
                        for (i=1; i<32; i=i+1)
                        reg_bank[i]=i+4'b0101;
                end
        else
                begin
                        if (we)
                        reg_bank[addr_write]<= write_data;
                        else
                        reg_bank[addr_write] <= reg_bank[addr_write];
                end
        end
endmodule




module R_file_tb ;

reg [4:0] addr_read1, addr_read2, addr_write;
```

```
reg clk=0, reset,we;
reg [31:0] write_data;
wire [31:0] read_data1, read_data2;

Register_file UUT (read_data1, read_data2, addr_read1, addr_read2, addr_write, write_data, we, clk, reset);

        always
        begin
        clk=~clk; #10;
        end


        initial
        begin
        reset=0; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=1; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=2; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=3; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=4; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=5; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=6; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=7; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=8; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=9; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=10; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=11; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=12; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=13; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=14; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=15; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=16; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=17; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=18; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=19; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=20; we=1; write_data=$random; #20;
        reset=1; addr_read1=0; addr_read2=0; addr_write=21; we=1; write_data=$random; #20;
        reset=1; addr_read1=1; addr_read2=2; addr_write=1; we=0; write_data=$random; #20;
        reset=1; addr_read1=3; addr_read2=4; addr_write=1; we=0; write_data=$random; #20;
        reset=1; addr_read1=5; addr_read2=6; addr_write=1; we=0; write_data=$random; #20;
        end

endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------

// mux 4 to 1

module mux_4to1(Y, sel, A, B, C, D);

                input A, B, C, D;
                input [0:1] sel;
                output reg Y;


                always @ (A,B,C,D,sel)

                        begin

                                case (sel)
                                        0: Y = A;
                                        1: Y = B;
                                        2: Y = C;
                                        3: Y = D;
                                        default: $display("Check your design");
                                endcase
                        end

endmodule


// mux 2 to 1


module mux_2to1(Y, sel, A, B);

        input A, B, sel;
        output Y;

        assign Y = (sel) ? B : A;


endmodule
```

```
// adder 1 bit

module adder_1bit(result, Cout, A, B, Cin);

        input A, B, Cin;
        output result, Cout;
        reg [1:0] temp;

        assign {Cout,result} = A + B + Cin;

endmodule


// ALU 1 bit

module ALU_1bit(Result, CarryOut, A, Ainvert, B, Binvert, CarryIn, op_sel);

                input A, Ainvert, B, Binvert, CarryIn;
                input [0:1] op_sel;
                output Result, CarryOut;
                wire y1, y2, add_out, o_out, a_out, ab, bb;

                mux_2to1 M1 (y1, Ainvert, A, ab);
                mux_2to1 M2 (y2, Binvert, B, bb);

                not N1 (bb, B);
                not N2 (ab, A);

                and A1 (a_out, y1, y2);

                or O1 (o_out, y1, y2);

                adder_1bit Add1 (add_out, CarryOut, y1, y2, CarryIn);

                mux_4to1 MM(Result, op_sel, a_out, o_out, add_out, B);


endmodule

module ALU_32bits (Result, Zero, Overflow, CarryOut, A, B, ops );

        output [31:0] Result;
        output Zero, Overflow, CarryOut;
```

```
        input [31:0] A, B;
        input [3:0] ops;
        wire [32:0] Carry;
        genvar i;


// module ALU_1bit(Result, CarryOut, A, Ainvert, B, Binvert, CarryIn, op_sel);6t

        generate

                for (i=0; i<32; i=i+1)

                ALU_1bit ALU (Result[i], Carry[i+1], A[i], ops[3], B[i], ops[2], Carry[i], ops[1:0]);

        endgenerate
        assign Carry[0] = ops[2];
        assign CarryOut = Carry[32];
        assign Overflow = Carry[31] ^ Carry [32];
        assign Zero = ~(| Result);



endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------

module PC(out, clk, reset);

input clk, reset;
output reg [15:0] out=0;


        always @ (posedge clk, negedge reset)
                begin
                if (reset ==0)
                out<=0;
                else
                out<=out+3'b001;
                end

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
// Company: SIU Edwardsville
// Engineer: Steve Muren
//
// Module Name:    DataMemory
// Project Name: ECE 483 Lab #7 Summer 2017
//
// Description: Data Memory module for implementation of
//              the 5-Stage Pipeline Processor.
//
//
// Revision 0.01 - File Created
// Additional Comments:
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------
//
//////////////////////////////////////////////////////////////////////////////////

module DataMemory(
    output reg[31:0] ReadData,
    input [15:0] address,
    input [31:0] WriteData,
    input MemRead,
    input MemWrite,
        input [31:0]datain
    );

    reg [31:0]  RAM  [63:0];

    always@  (address)
        begin
          if (MemWrite)
                RAM[address] <= WriteData;
          else if  (MemRead)
                ReadData <= RAM[address];
          else
                ReadData <= datain;
        end
endmodule
```

```
//----------------------------------------
// COPYRIGHT @ 2018 VIPIN JASORIA
// ALL CODES FOR REFERENCE ONLY
// TOP MODULE MIPS 5 STAGE PROCESSOR
//----------------------------------------

module Mux_5bits (A, B, sel, Y);

input [4:0] A, B;
input sel;
output [4:0] Y;

        assign Y = sel ? B : A;


endmodule
```