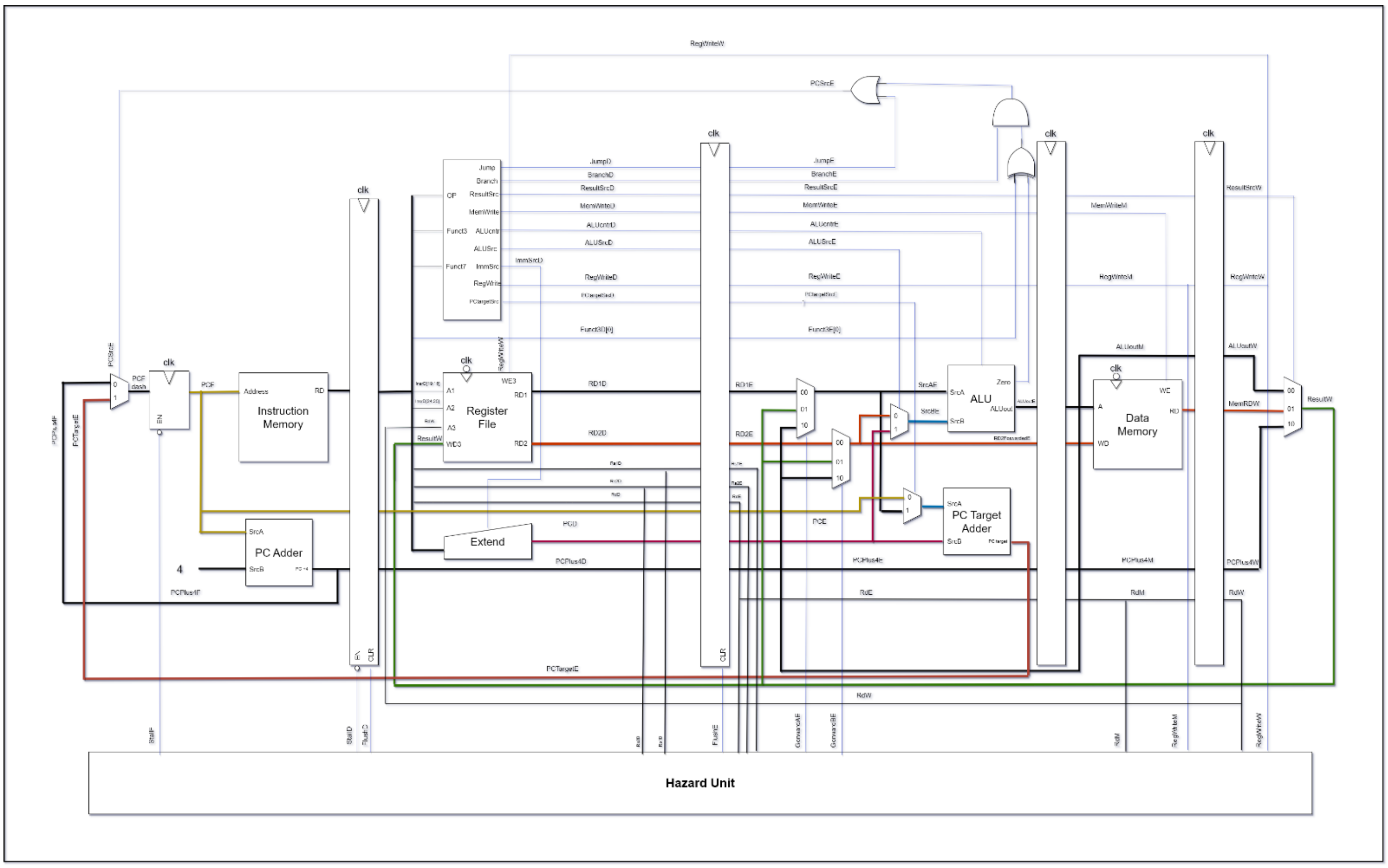
RISC-V Pipeline Hazard Architecture

Name: Khaled Mahoud Taher Fathallah Mohamed Elkorany

Track: Digital IC Design New Capital

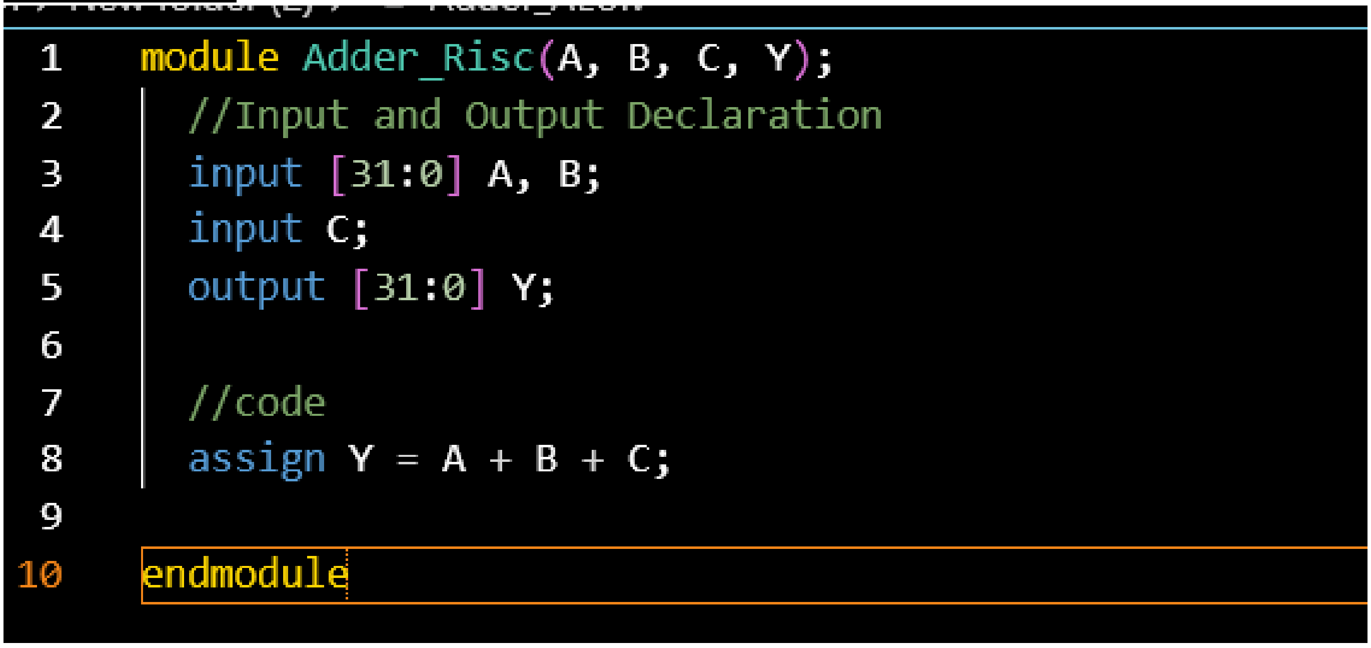
Number: 12

Block Diagram Of The Architecture



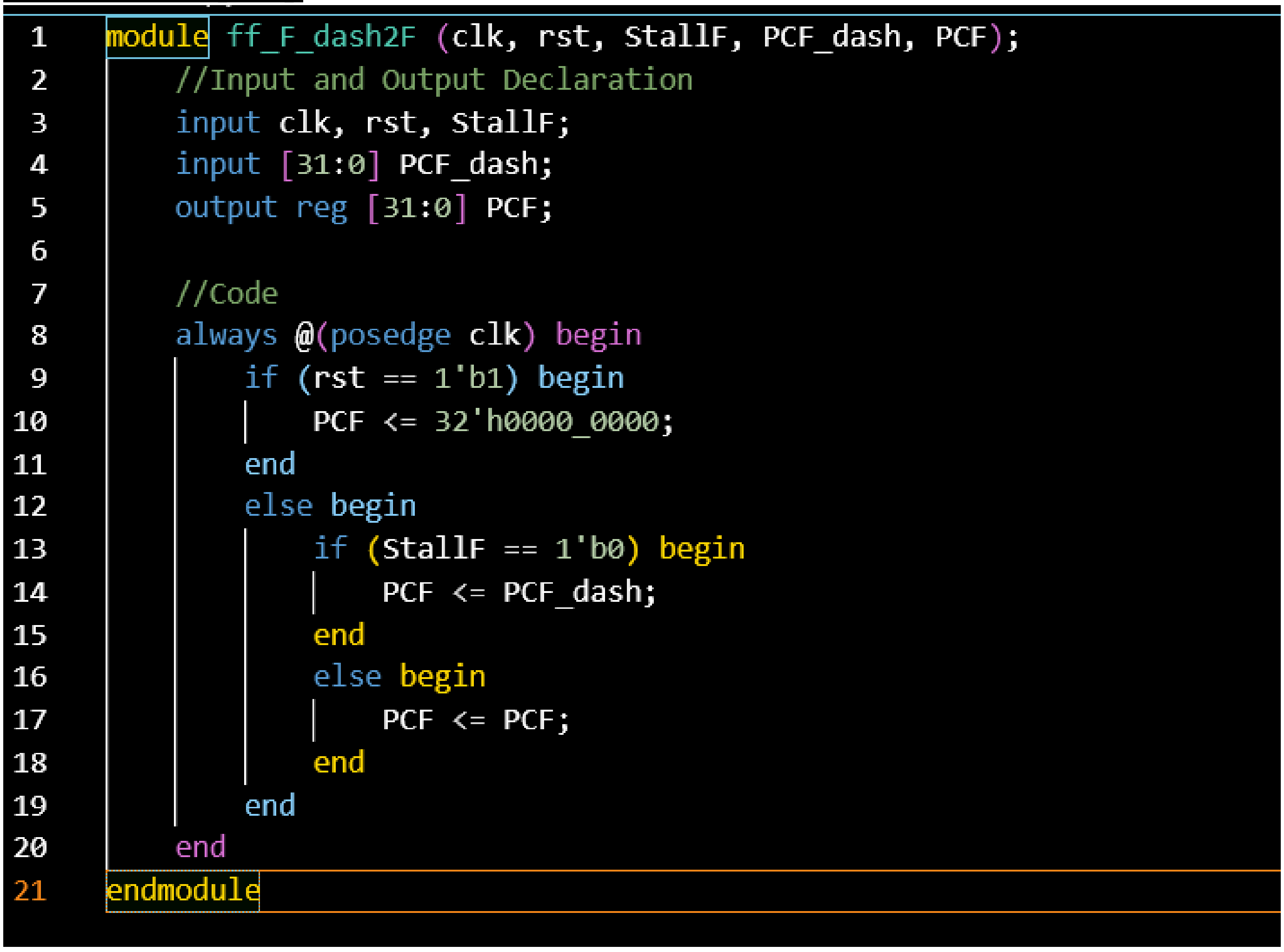
Verilog Code

Adder Module



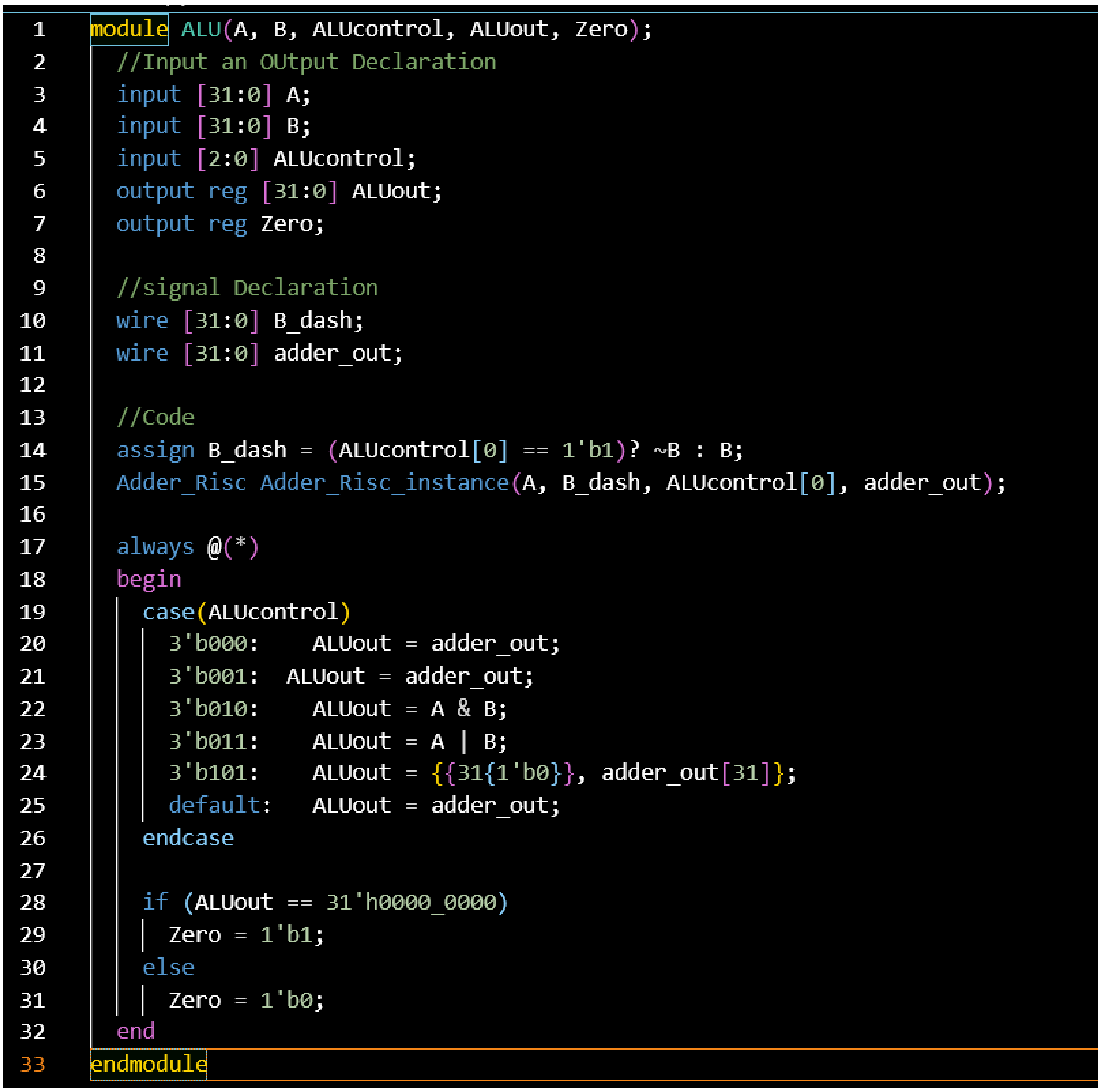
```verilog
module Adder_Risc(A, B, C, Y);
    //Input and Output Declaration
    input [31:0] A, B;
    input C;
    output [31:0] Y;

    //code
    assign Y = A + B + C;

endmodule
```

F_dash to F flip flop



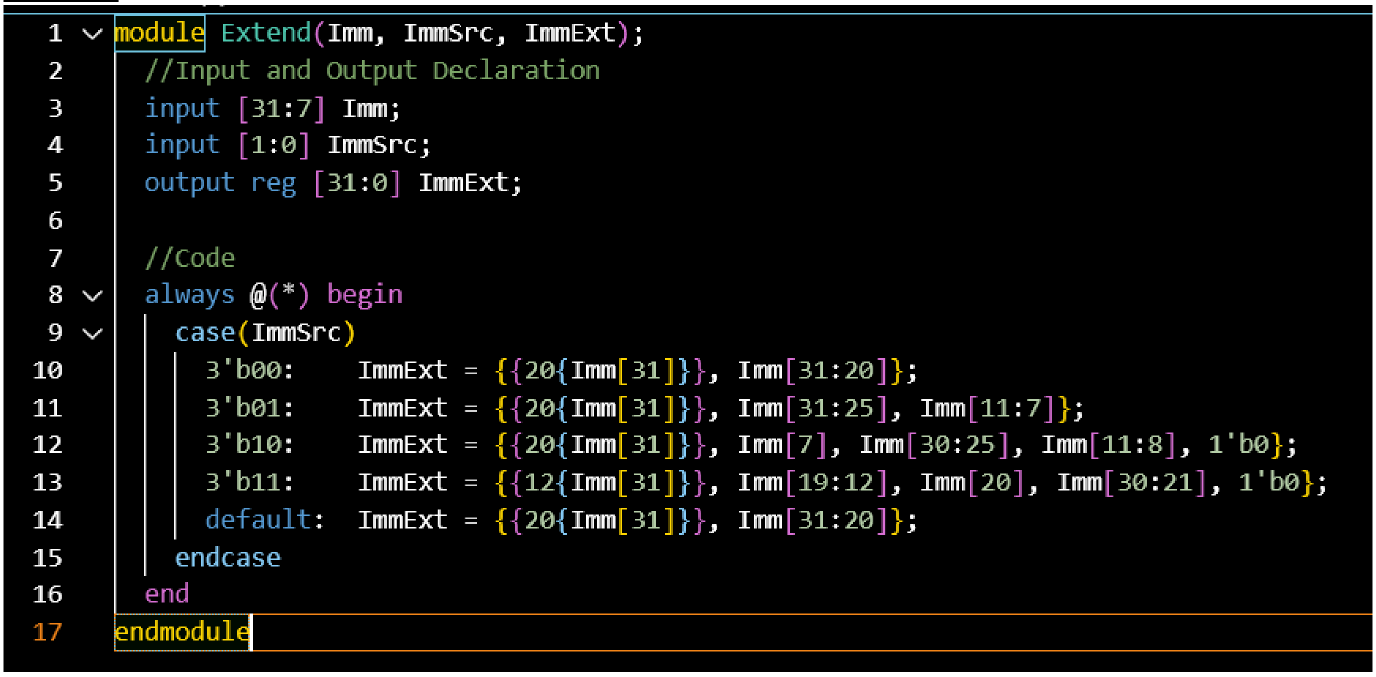
```verilog
module ff_F_dash2F (clk, rst, StallF, PCF_dash, PCF);
    //Input and Output Declaration
    input clk, rst, StallF;
    input [31:0] PCF_dash;
    output reg [31:0] PCF;

    //Code
    always @(posedge clk) begin
        if (rst == 1'b1) begin
            PCF <= 32'h0000_0000;
        end
        else begin
            if (StallF == 1'b0) begin
                PCF <= PCF_dash;
            end
            else begin
                PCF <= PCF;
            end
        end
    end
endmodule
```

ALU Module



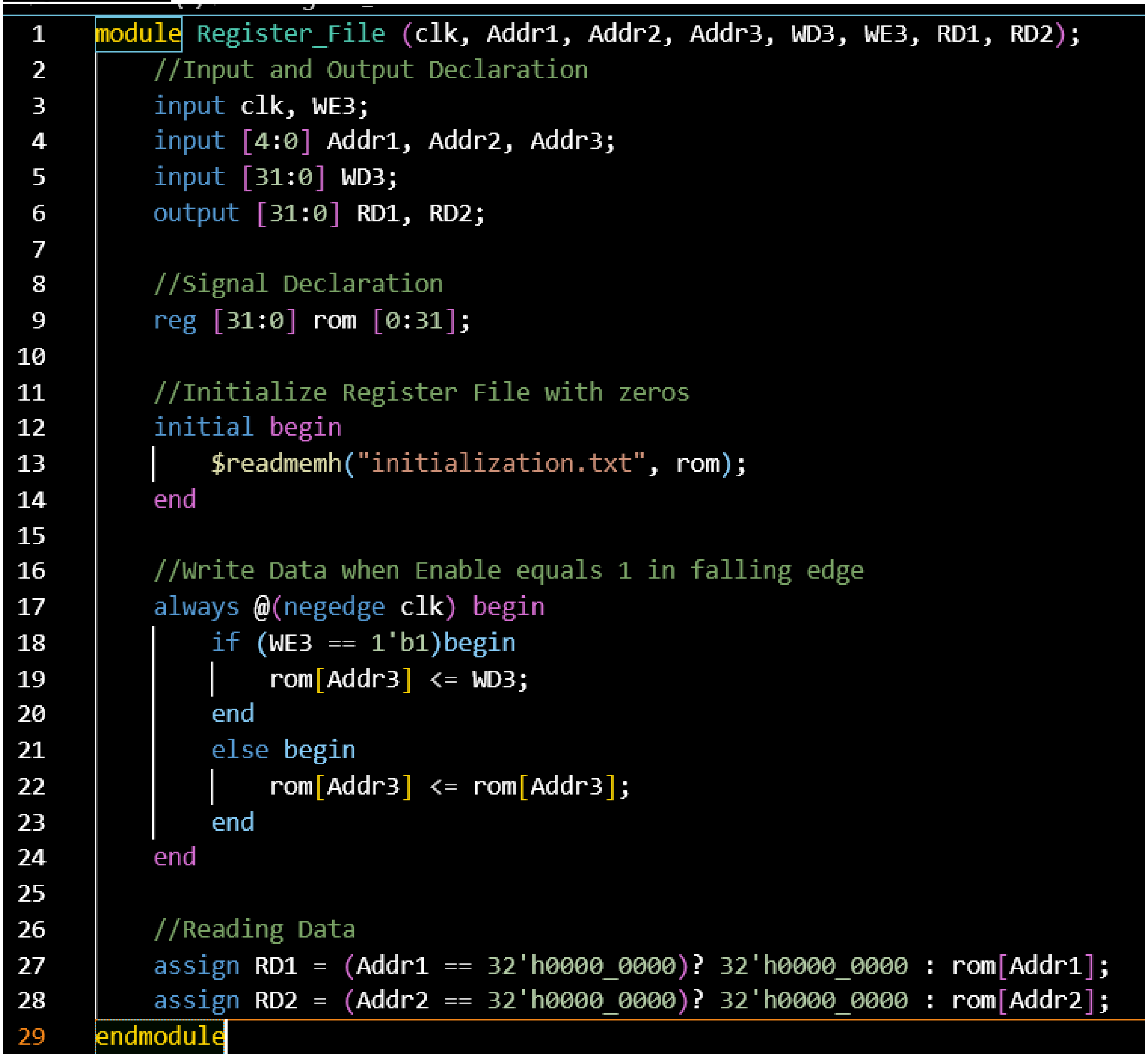
```verilog
module ALU(A, B, ALUcontrol, ALUout, Zero);
  //Input an OUtput Declaration
  input [31:0] A;
  input [31:0] B;
  input [2:0] ALUcontrol;
  output reg [31:0] ALUout;
  output reg Zero;

  //signal Declaration
  wire [31:0] B_dash;
  wire [31:0] adder_out;

  //Code
  assign B_dash = (ALUcontrol[0] == 1'b1)? ~B : B;
  Adder_Risc Adder_Risc_instance(A, B_dash, ALUcontrol[0], adder_out);

  always @(*)
  begin
    case(ALUcontrol)
      3'b000:    ALUout = adder_out;
      3'b001:  ALUout = adder_out;
      3'b010:    ALUout = A & B;
      3'b011:    ALUout = A | B;
      3'b101:    ALUout = {{31{1'b0}}, adder_out[31]};
      default:   ALUout = adder_out;
    endcase

    if (ALUout == 31'h0000_0000)
      Zero = 1'b1;
    else
      Zero = 1'b0;
  end
endmodule
```

Extend



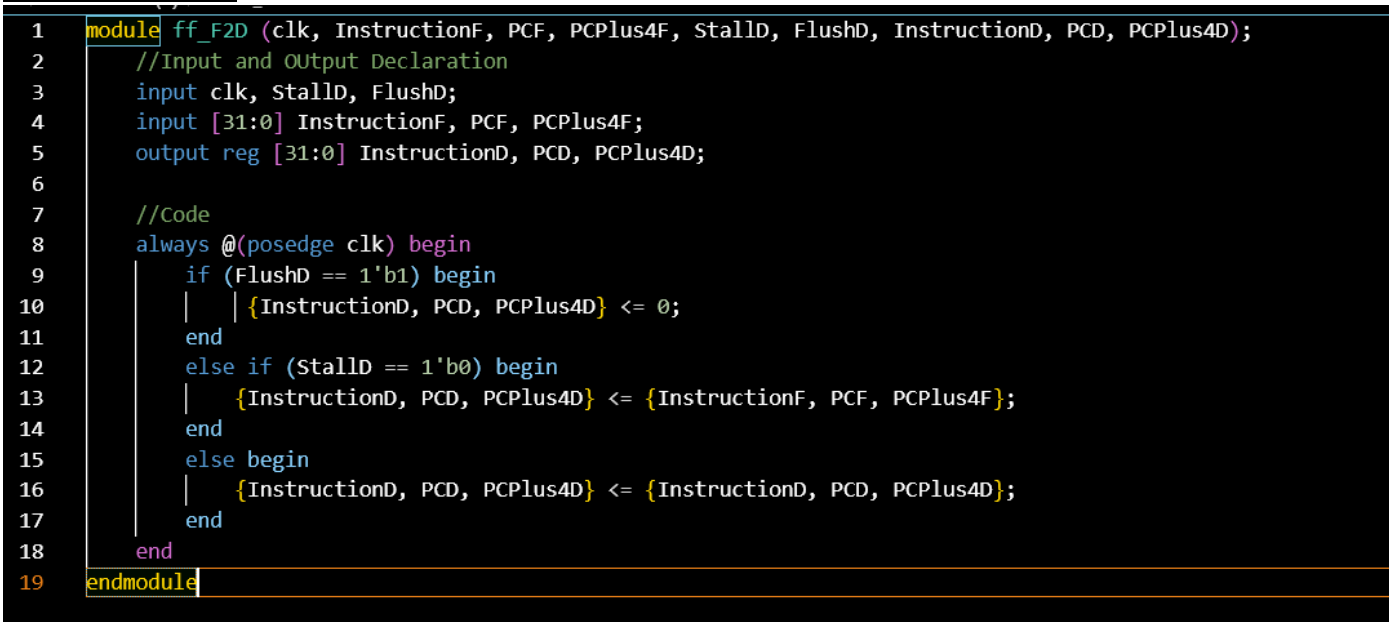
```verilog
module Extend(Imm, ImmSrc, ImmExt);
 //Input and Output Declaration
 input [31:7] Imm;
 input [1:0] ImmSrc;
 output reg [31:0] ImmExt;

 //Code
 always @(*) begin
   case(ImmSrc)
     3'b00:    ImmExt = {{20{Imm[31]}}, Imm[31:20]};
     3'b01:    ImmExt = {{20{Imm[31]}}, Imm[31:25], Imm[11:7]};
     3'b10:    ImmExt = {{20{Imm[31]}}, Imm[7], Imm[30:25], Imm[11:8], 1'b0};
     3'b11:    ImmExt = {{12{Imm[31]}}, Imm[19:12], Imm[20], Imm[30:21], 1'b0};
     default:  ImmExt = {{20{Imm[31]}}, Imm[31:20]};
   endcase
 end
endmodule
```

Register File



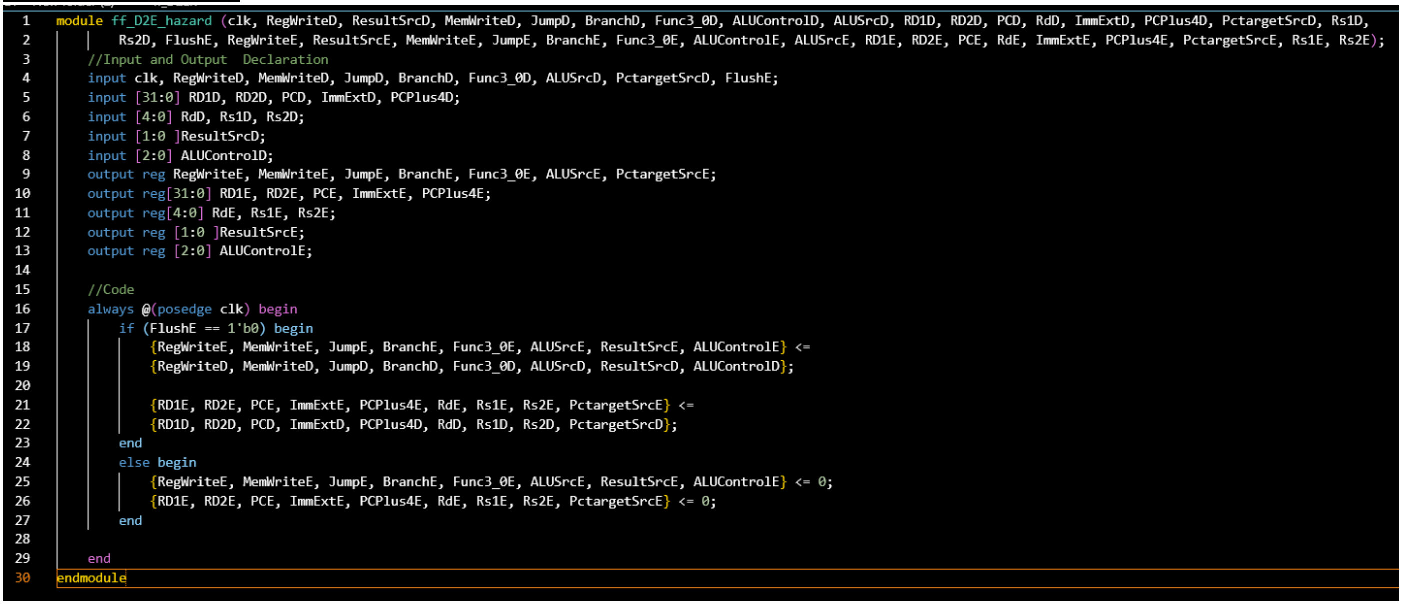
```verilog
module Register_File (clk, Addr1, Addr2, Addr3, WD3, WE3, RD1, RD2);
    //Input and Output Declaration
    input clk, WE3;
    input [4:0] Addr1, Addr2, Addr3;
    input [31:0] WD3;
    output [31:0] RD1, RD2;

    //Signal Declaration
    reg [31:0] rom [0:31];

    //Initialize Register File with zeros
    initial begin
        $readmemh("initialization.txt", rom);
    end

    //Write Data when Enable equals 1 in falling edge
    always @(negedge clk) begin
        if (WE3 == 1'b1)begin
            rom[Addr3] <= WD3;
        end
        else begin
            rom[Addr3] <= rom[Addr3];
        end
    end

    //Reading Data
    assign RD1 = (Addr1 == 32'h0000_0000)? 32'h0000_0000 : rom[Addr1];
    assign RD2 = (Addr2 == 32'h0000_0000)? 32'h0000_0000 : rom[Addr2];
endmodule
```

F to D flip flop



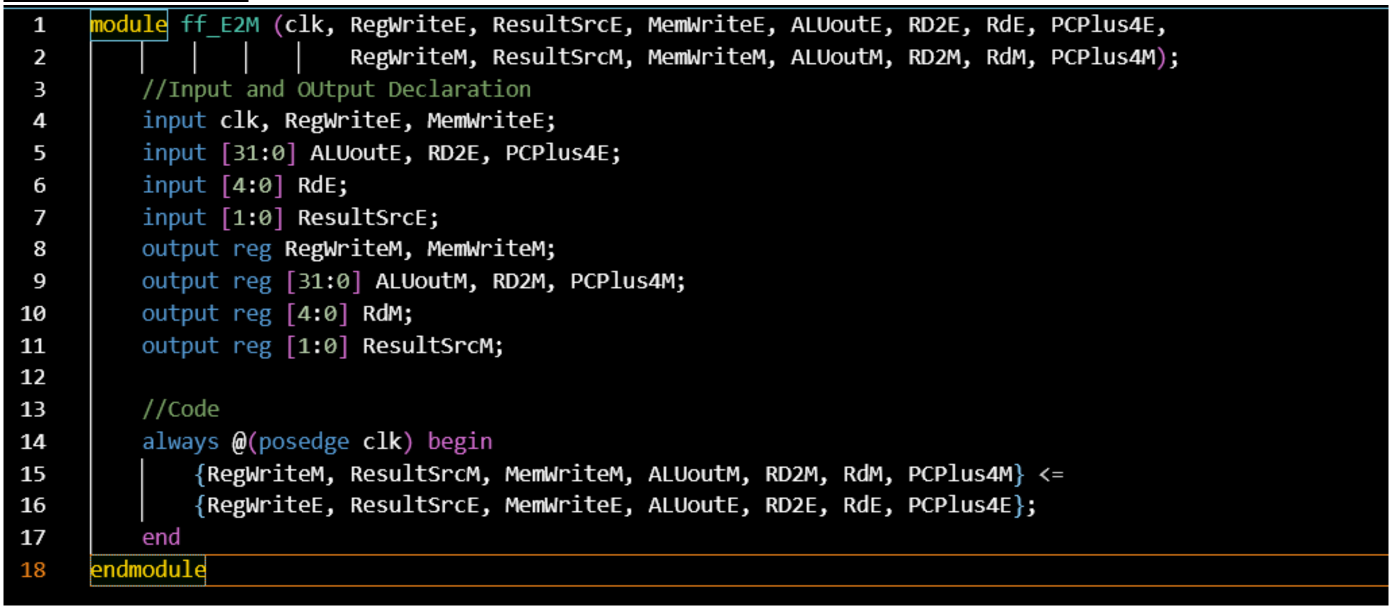
```verilog
module ff_F2D (clk, InstructionF, PCF, PCPlus4F, StallD, FlushD, InstructionD, PCD, PCPlus4D);
    //Input and Output Declaration
    input clk, StallD, FlushD;
    input [31:0] InstructionF, PCF, PCPlus4F;
    output reg [31:0] InstructionD, PCD, PCPlus4D;

    //Code
    always @(posedge clk) begin
        if (FlushD == 1'b1) begin
                {InstructionD, PCD, PCPlus4D} <= 0;
        end
        else if (StallD == 1'b0) begin
            {InstructionD, PCD, PCPlus4D} <= {InstructionF, PCF, PCPlus4F};
        end
        else begin
            {InstructionD, PCD, PCPlus4D} <= {InstructionD, PCD, PCPlus4D};
        end
    end
endmodule
```

D to E flip flop



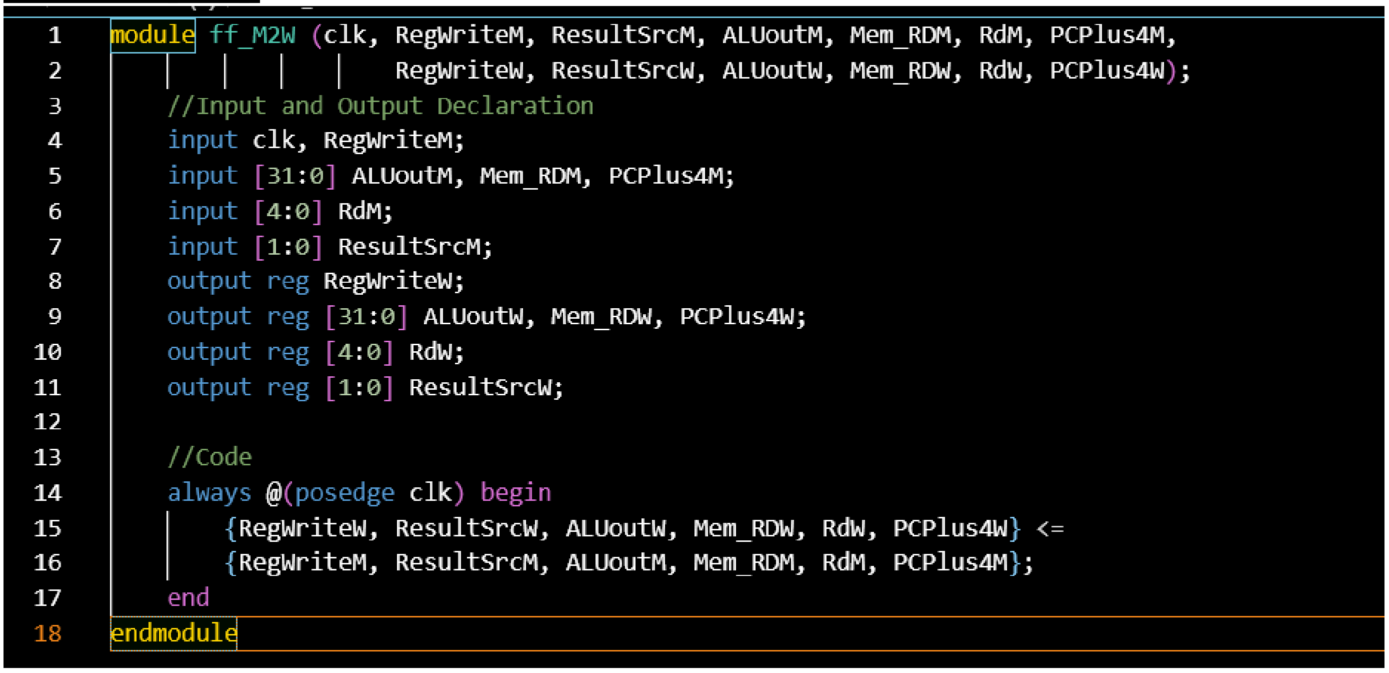
```verilog
module ff_D2E_hazard (clk, RegWriteD, ResultSrcD, MemWriteD, JumpD, BranchD, Func3_0D, ALUControlD, ALUSrcD, RD1D, RD2D, PCD, RdD, ImmExtD, PCPlus4D, PctargetSrcD, Rs1D,
        Rs2D, FlushE, RegWriteE, ResultSrcE, MemWriteE, JumpE, BranchE, Func3_0E, ALUControlE, ALUSrcE, RD1E, RD2E, PCE, RdE, ImmExtE, PCPlus4E, PctargetSrcE, Rs1E, Rs2E);
    //Input and Output  Declaration
    input clk, RegWriteD, MemWriteD, JumpD, BranchD, Func3_0D, ALUSrcD, PctargetSrcD, FlushE;
    input [31:0] RD1D, RD2D, PCD, ImmExtD, PCPlus4D;
    input [4:0] RdD, Rs1D, Rs2D;
    input [1:0 ]ResultSrcD;
    input [2:0] ALUControlD;
    output reg RegWriteE, MemWriteE, JumpE, BranchE, Func3_0E, ALUSrcE, PctargetSrcE;
    output reg[31:0] RD1E, RD2E, PCE, ImmExtE, PCPlus4E;
    output reg[4:0] RdE, Rs1E, Rs2E;
    output reg [1:0 ]ResultSrcE;
    output reg [2:0] ALUControlE;

    //Code
    always @(posedge clk) begin
        if (FlushE == 1'b0) begin
            {RegWriteE, MemWriteE, JumpE, BranchE, Func3_0E, ALUSrcE, ResultSrcE, ALUControlE} <=
            {RegWriteD, MemWriteD, JumpD, BranchD, Func3_0D, ALUSrcD, ResultSrcD, ALUControlD};

            {RD1E, RD2E, PCE, ImmExtE, PCPlus4E, RdE, Rs1E, Rs2E, PctargetSrcE} <=
            {RD1D, RD2D, PCD, ImmExtD, PCPlus4D, RdD, Rs1D, Rs2D, PctargetSrcD};
        end
        else begin
            {RegWriteE, MemWriteE, JumpE, BranchE, Func3_0E, ALUSrcE, ResultSrcE, ALUControlE} <= 0;
            {RD1E, RD2E, PCE, ImmExtE, PCPlus4E, RdE, Rs1E, Rs2E, PctargetSrcE} <= 0;
        end

    end
endmodule
```

E to M flip flop



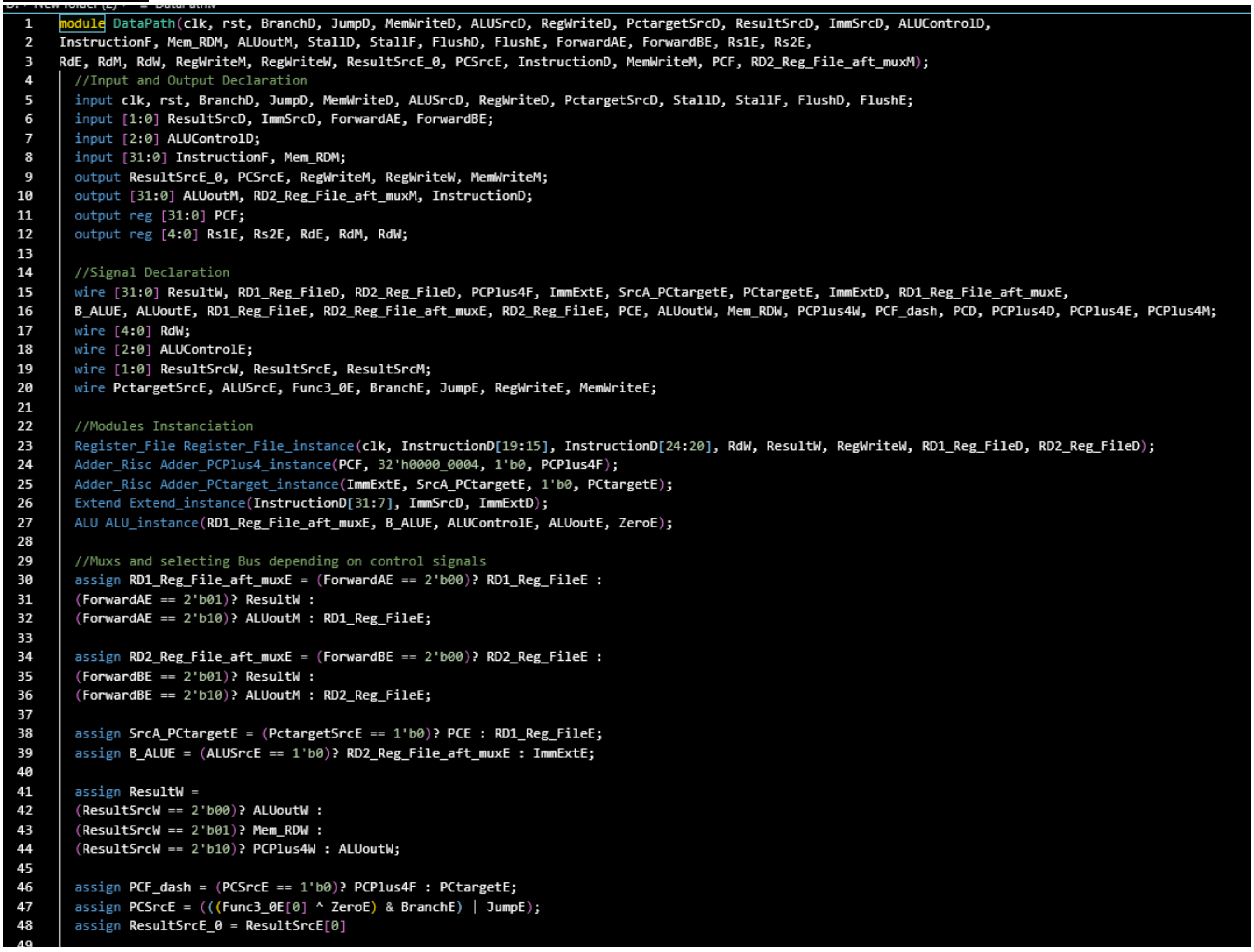
```verilog
module ff_E2M (clk, RegWriteE, ResultSrcE, MemWriteE, ALUoutE, RD2E, RdE, PCPlus4E,
                RegWriteM, ResultSrcM, MemWriteM, ALUoutM, RD2M, RdM, PCPlus4M);
    //Input and Output Declaration
    input clk, RegWriteE, MemWriteE;
    input [31:0] ALUoutE, RD2E, PCPlus4E;
    input [4:0] RdE;
    input [1:0] ResultSrcE;
    output reg RegWriteM, MemWriteM;
    output reg [31:0] ALUoutM, RD2M, PCPlus4M;
    output reg [4:0] RdM;
    output reg [1:0] ResultSrcM;

    //Code
    always @(posedge clk) begin
        {RegWriteM, ResultSrcM, MemWriteM, ALUoutM, RD2M, RdM, PCPlus4M} <=
        {RegWriteE, ResultSrcE, MemWriteE, ALUoutE, RD2E, RdE, PCPlus4E};
    end
endmodule
```

M to W flip flop

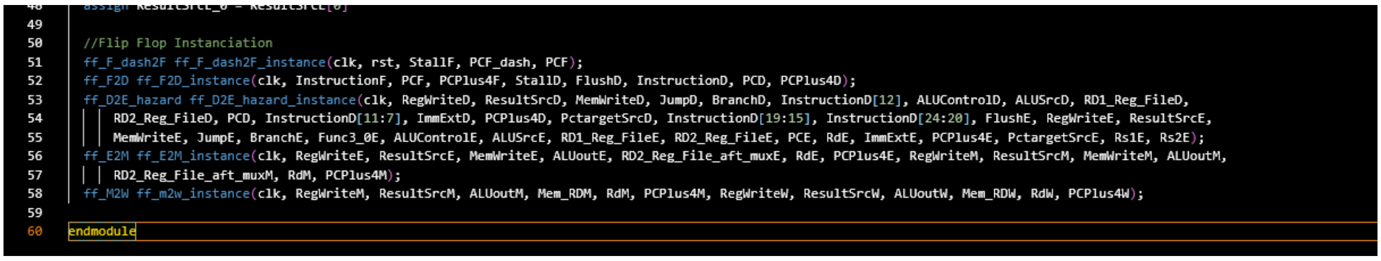


```verilog
module ff_M2W (clk, RegWriteM, ResultSrcM, ALUoutM, Mem_RDM, RdM, PCPlus4M,
                RegWriteW, ResultSrcW, ALUoutW, Mem_RDW, RdW, PCPlus4W);
    //Input and Output Declaration
    input clk, RegWriteM;
    input [31:0] ALUoutM, Mem_RDM, PCPlus4M;
    input [4:0] RdM;
    input [1:0] ResultSrcM;
    output reg RegWriteW;
    output reg [31:0] ALUoutW, Mem_RDW, PCPlus4W;
    output reg [4:0] RdW;
    output reg [1:0] ResultSrcW;

    //Code
    always @(posedge clk) begin
        {RegWriteW, ResultSrcW, ALUoutW, Mem_RDW, RdW, PCPlus4W} <=
        {RegWriteM, ResultSrcM, ALUoutM, Mem_RDM, RdM, PCPlus4M};
    end
endmodule
```

Data Path

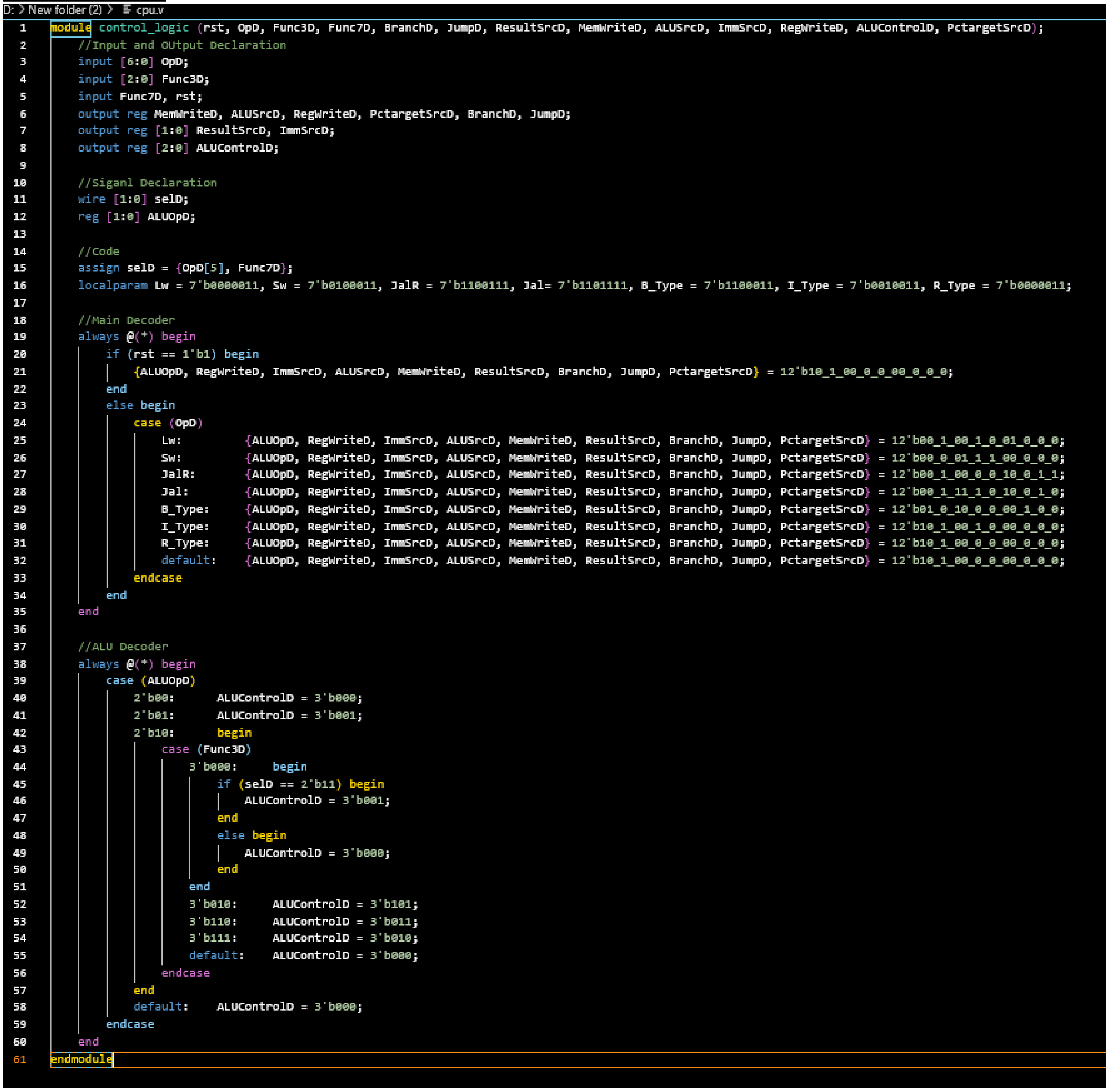


```verilog
module DataPath(clk, rst, BranchD, JumpD, MemWriteD, ALUSrcD, RegWriteD, PctargetSrcD, ResultSrcD, ImmSrcD, ALUControlD,
InstructionF, Mem_RDM, ALUoutM, StallD, StallF, FlushD, FlushE, ForwardAE, ForwardBE, Rs1E, Rs2E,
RdE, RdM, RdW, RegWriteM, RegWriteW, ResultSrcE_0, PCSrcE, InstructionD, MemWriteM, PCF, RD2_Reg_File_aft_muxM);
  //Input and Output Declaration
  input clk, rst, BranchD, JumpD, MemWriteD, ALUSrcD, RegWriteD, PctargetSrcD, StallD, StallF, FlushD, FlushE;
  input [1:0] ResultSrcD, ImmSrcD, ForwardAE, ForwardBE;
  input [2:0] ALUControlD;
  input [31:0] InstructionF, Mem_RDM;
  output ResultSrcE_0, PCSrcE, RegWriteM, RegWriteW, MemWriteM;
  output [31:0] ALUoutM, RD2_Reg_File_aft_muxM, InstructionD;
  output reg [31:0] PCF;
  output reg [4:0] Rs1E, Rs2E, RdE, RdM, RdW;

  //Signal Declaration
  wire [31:0] ResultW, RD1_Reg_FileD, RD2_Reg_FileD, PCPlus4F, ImmExtE, SrcA_PCtargetE, PCtargetE, ImmExtD, RD1_Reg_File_aft_muxE,
  B_ALUE, ALUoutE, RD1_Reg_FileE, RD2_Reg_File_aft_muxE, RD2_Reg_FileE, PCE, ALUoutW, Mem_RDW, PCPlus4W, PCF_dash, PCD, PCPlus4D, PCPlus4E, PCPlus4M;
  wire [4:0] RdW;
  wire [2:0] ALUControlE;
  wire [1:0] ResultSrcW, ResultSrcE, ResultSrcM;
  wire PctargetSrcE, ALUSrcE, Func3_0E, BranchE, JumpE, RegWriteE, MemWriteE;

  //Modules Instanciation
  Register_File Register_File_instance(clk, InstructionD[19:15], InstructionD[24:20], RdW, ResultW, RegWriteW, RD1_Reg_FileD, RD2_Reg_FileD);
  Adder_Risc Adder_PCPlus4_instance(PCF, 32'h0000_0004, 1'b0, PCPlus4F);
  Adder_Risc Adder_PCtarget_instance(ImmExtE, SrcA_PCtargetE, 1'b0, PCtargetE);
  Extend Extend_instance(InstructionD[31:7], ImmSrcD, ImmExtD);
  ALU ALU_instance(RD1_Reg_File_aft_muxE, B_ALUE, ALUControlE, ALUoutE, ZeroE);

  //Muxs and selecting Bus depending on control signals
  assign RD1_Reg_File_aft_muxE = (ForwardAE == 2'b00)? RD1_Reg_FileE :
  (ForwardAE == 2'b01)? ResultW :
  (ForwardAE == 2'b10)? ALUoutM : RD1_Reg_FileE;

  assign RD2_Reg_File_aft_muxE = (ForwardBE == 2'b00)? RD2_Reg_FileE :
  (ForwardBE == 2'b01)? ResultW :
  (ForwardBE == 2'b10)? ALUoutM : RD2_Reg_FileE;

  assign SrcA_PCtargetE = (PctargetSrcE == 1'b0)? PCE : RD1_Reg_FileE;
  assign B_ALUE = (ALUSrcE == 1'b0)? RD2_Reg_File_aft_muxE : ImmExtE;

  assign ResultW =
  (ResultSrcW == 2'b00)? ALUoutW :
  (ResultSrcW == 2'b01)? Mem_RDW :
  (ResultSrcW == 2'b10)? PCPlus4W : ALUoutW;

  assign PCF_dash = (PCSrcE == 1'b0)? PCPlus4F : PCtargetE;
  assign PCSrcE = (((Func3_0E[0] ^ ZeroE) & BranchE) | JumpE);
  assign ResultSrcE_0 = ResultSrcE[0]
```



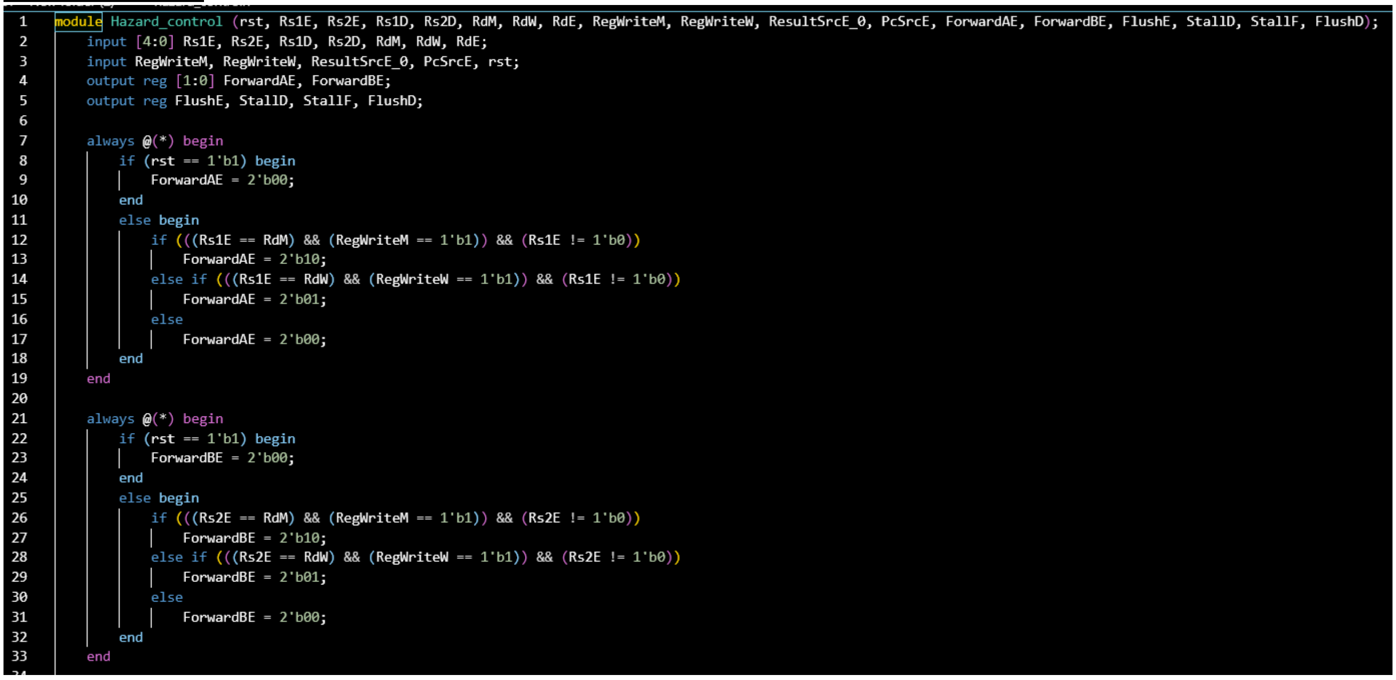
```verilog

  //Flip Flop Instanciation
  ff_F_dash2F ff_F_dash2F_instance(clk, rst, StallF, PCF_dash, PCF);
  ff_F2D ff_F2D_instance(clk, InstructionF, PCF, PCPlus4F, StallD, FlushD, InstructionD, PCD, PCPlus4D);
  ff_D2E_hazard ff_D2E_hazard_instance(clk, RegWriteD, ResultSrcD, MemWriteD, JumpD, BranchD, InstructionD[12], ALUControlD, ALUSrcD, RD1_Reg_FileD,
      RD2_Reg_FileD, PCD, InstructionD[11:7], ImmExtD, PCPlus4D, PctargetSrcD, InstructionD[19:15], InstructionD[24:20], FlushE, RegWriteE, ResultSrcE,
      MemWriteE, JumpE, BranchE, Func3_0E, ALUControlE, ALUSrcE, RD1_Reg_FileE, RD2_Reg_FileE, PCE, RdE, ImmExtE, PCPlus4E, PctargetSrcE, Rs1E, Rs2E);
  ff_E2M ff_E2M_instance(clk, RegWriteE, ResultSrcE, MemWriteE, ALUoutE, RD2_Reg_File_aft_muxE, RdE, PCPlus4E, RegWriteM, ResultSrcM, MemWriteM, ALUoutM,
      RD2_Reg_File_aft_muxM, RdM, PCPlus4M);
  ff_M2W ff_m2w_instance(clk, RegWriteM, ResultSrcM, ALUoutM, Mem_RDM, RdM, PCPlus4M, RegWriteW, ResultSrcW, ALUoutW, Mem_RDW, RdW, PCPlus4W);

endmodule
```

Control Unit

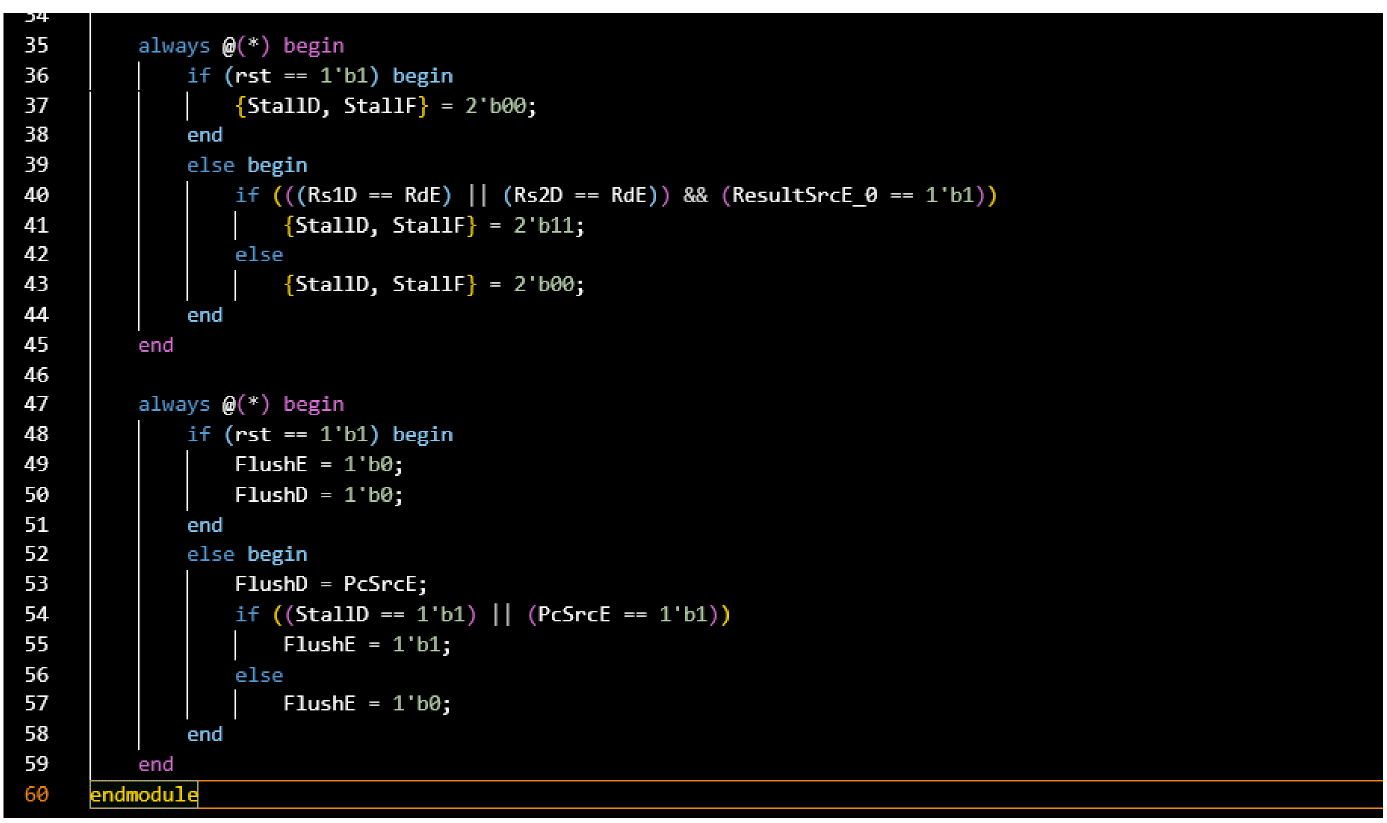

D: > New folder (2) > cpu.v

```verilog
module control_logic (rst, OpD, Func3D, Func7D, BranchD, JumpD, ResultSrcD, MemWriteD, ALUSrcD, ImmSrcD, RegWriteD, ALUControlD, PctargetSrcD);
    //Input and OUtput Declaration
    input [6:0] OpD;
    input [2:0] Func3D;
    input Func7D, rst;
    output reg MemWriteD, ALUSrcD, RegWriteD, PctargetSrcD, BranchD, JumpD;
    output reg [1:0] ResultSrcD, ImmSrcD;
    output reg [2:0] ALUControlD;

    //Siganl Declaration
    wire [1:0] selD;
    reg [1:0] ALUOpD;

    //Code
    assign selD = {OpD[5], Func7D};
    localparam Lw = 7'b0000011, Sw = 7'b0100011, JalR = 7'b1100111, Jal= 7'b1101111, B_Type = 7'b1100011, I_Type = 7'b0010011, R_Type = 7'b0000011;

    //Main Decoder
    always @(*) begin
        if (rst == 1'b1) begin
            {ALUOpD, RegWriteD, ImmSrcD, ALUSrcD, MemWriteD, ResultSrcD, BranchD, JumpD, PctargetSrcD} = 12'b10_1_00_0_0_00_0_0_0;
        end
        else begin
            case (OpD)
                Lw:         {ALUOpD, RegWriteD, ImmSrcD, ALUSrcD, MemWriteD, ResultSrcD, BranchD, JumpD, PctargetSrcD} = 12'b00_1_00_1_0_01_0_0_0;
                Sw:         {ALUOpD, RegWriteD, ImmSrcD, ALUSrcD, MemWriteD, ResultSrcD, BranchD, JumpD, PctargetSrcD} = 12'b00_0_01_1_1_00_0_0_0;
                JalR:       {ALUOpD, RegWriteD, ImmSrcD, ALUSrcD, MemWriteD, ResultSrcD, BranchD, JumpD, PctargetSrcD} = 12'b00_1_00_0_0_10_0_1_1;
                Jal:        {ALUOpD, RegWriteD, ImmSrcD, ALUSrcD, MemWriteD, ResultSrcD, BranchD, JumpD, PctargetSrcD} = 12'b00_1_11_1_0_10_0_1_0;
                B_Type:     {ALUOpD, RegWriteD, ImmSrcD, ALUSrcD, MemWriteD, ResultSrcD, BranchD, JumpD, PctargetSrcD} = 12'b01_0_10_0_0_00_1_0_0;
                I_Type:     {ALUOpD, RegWriteD, ImmSrcD, ALUSrcD, MemWriteD, ResultSrcD, BranchD, JumpD, PctargetSrcD} = 12'b10_1_00_1_0_00_0_0_0;
                R_Type:     {ALUOpD, RegWriteD, ImmSrcD, ALUSrcD, MemWriteD, ResultSrcD, BranchD, JumpD, PctargetSrcD} = 12'b10_1_00_0_0_00_0_0_0;
                default:    {ALUOpD, RegWriteD, ImmSrcD, ALUSrcD, MemWriteD, ResultSrcD, BranchD, JumpD, PctargetSrcD} = 12'b10_1_00_0_0_00_0_0_0;
            endcase
        end
    end

    //ALU Decoder
    always @(*) begin
        case (ALUOpD)
            2'b00:      ALUControlD = 3'b000;
            2'b01:      ALUControlD = 3'b001;
            2'b10:      begin
                case (Func3D)
                    3'b000:     begin
                        if (selD == 2'b11) begin
                            ALUControlD = 3'b001;
                        end
                        else begin
                            ALUControlD = 3'b000;
                        end
                    end
                    3'b010:     ALUControlD = 3'b101;
                    3'b110:     ALUControlD = 3'b011;
                    3'b111:     ALUControlD = 3'b010;
                    default:    ALUControlD = 3'b000;
                endcase
            end
            default:    ALUControlD = 3'b000;
        endcase
    end
endmodule
```

Hazard Unit

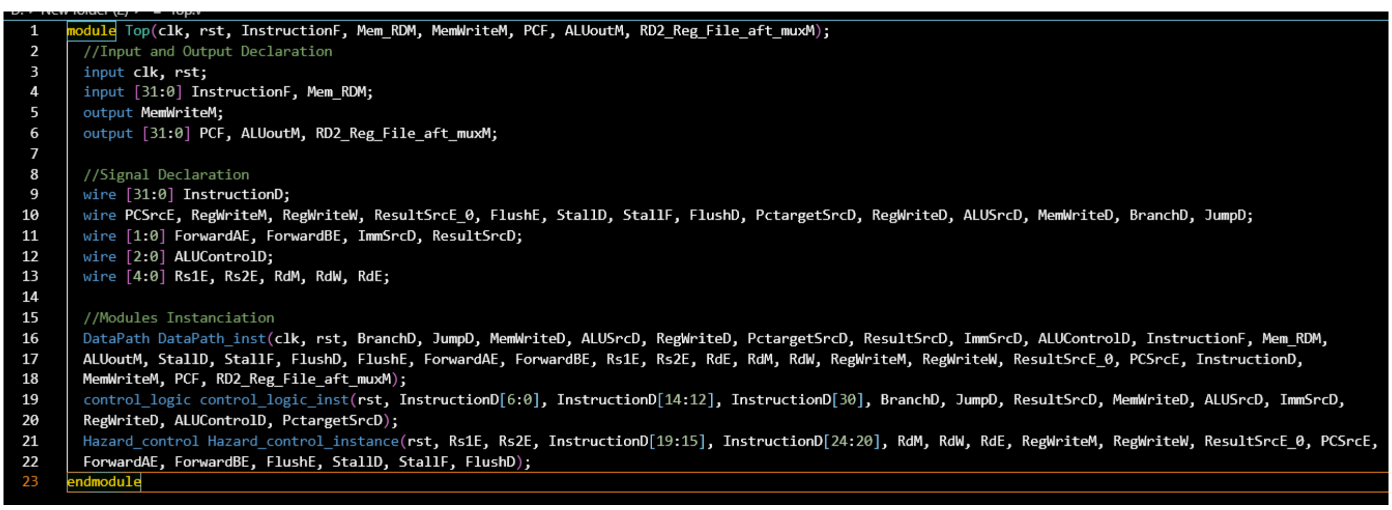


```verilog
module Hazard_control (rst, Rs1E, Rs2E, Rs1D, Rs2D, RdM, RdW, RdE, RegWriteM, RegWriteW, ResultSrcE_0, PcSrcE, ForwardAE, ForwardBE, FlushE, StallD, StallF, FlushD);
    input [4:0] Rs1E, Rs2E, Rs1D, Rs2D, RdM, RdW, RdE;
    input RegWriteM, RegWriteW, ResultSrcE_0, PcSrcE, rst;
    output reg [1:0] ForwardAE, ForwardBE;
    output reg FlushE, StallD, StallF, FlushD;

    always @(*) begin
        if (rst == 1'b1) begin
            ForwardAE = 2'b00;
        end
        else begin
            if (((Rs1E == RdM) && (RegWriteM == 1'b1)) && (Rs1E != 1'b0))
                ForwardAE = 2'b10;
            else if (((Rs1E == RdW) && (RegWriteW == 1'b1)) && (Rs1E != 1'b0))
                ForwardAE = 2'b01;
            else
                ForwardAE = 2'b00;
        end
    end

    always @(*) begin
        if (rst == 1'b1) begin
            ForwardBE = 2'b00;
        end
        else begin
            if (((Rs2E == RdM) && (RegWriteM == 1'b1)) && (Rs2E != 1'b0))
                ForwardBE = 2'b10;
            else if (((Rs2E == RdW) && (RegWriteW == 1'b1)) && (Rs2E != 1'b0))
                ForwardBE = 2'b01;
            else
                ForwardBE = 2'b00;
        end
    end
```



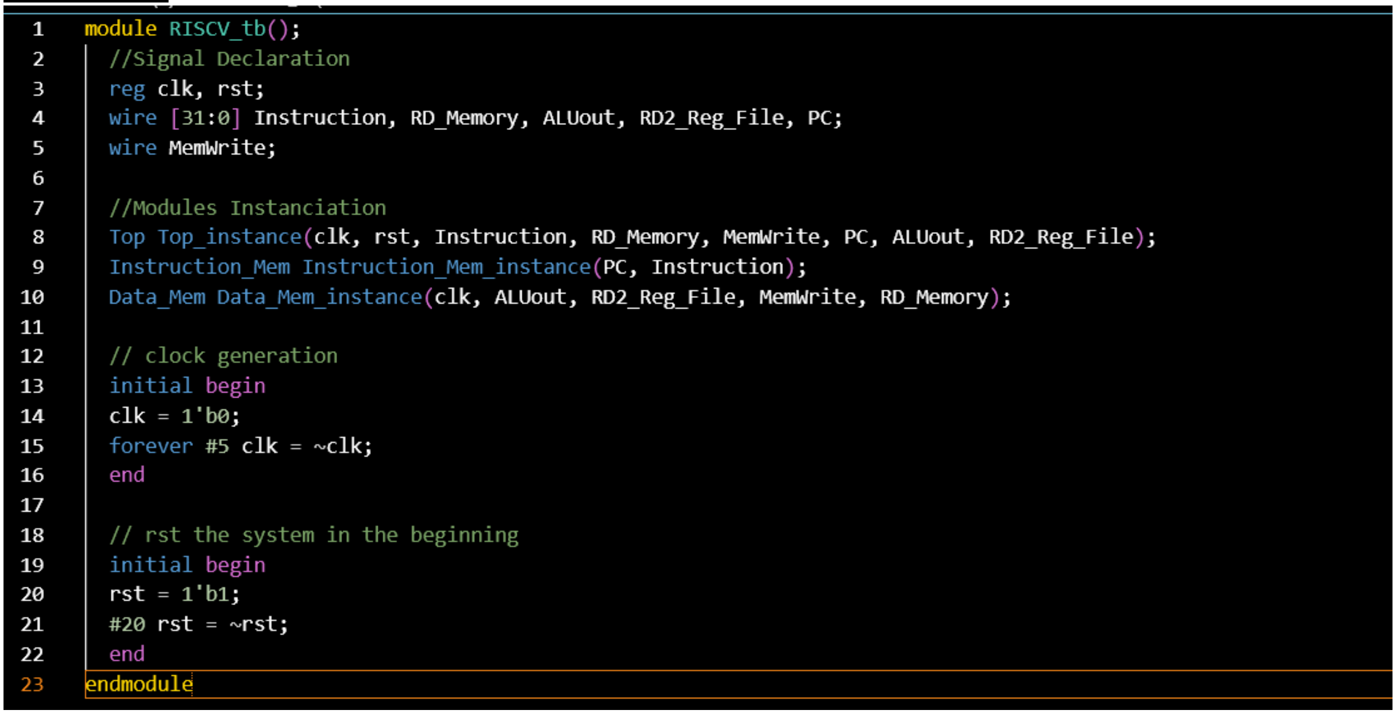
```verilog

    always @(*) begin
        if (rst == 1'b1) begin
            {StallD, StallF} = 2'b00;
        end
        else begin
            if (((Rs1D == RdE) || (Rs2D == RdE)) && (ResultSrcE_0 == 1'b1))
                {StallD, StallF} = 2'b11;
            else
                {StallD, StallF} = 2'b00;
        end
    end

    always @(*) begin
        if (rst == 1'b1) begin
            FlushE = 1'b0;
            FlushD = 1'b0;
        end
        else begin
            FlushD = PcSrcE;
            if ((StallD == 1'b1) || (PcSrcE == 1'b1))
                FlushE = 1'b1;
            else
                FlushE = 1'b0;
        end
    end
endmodule
```

Top Module



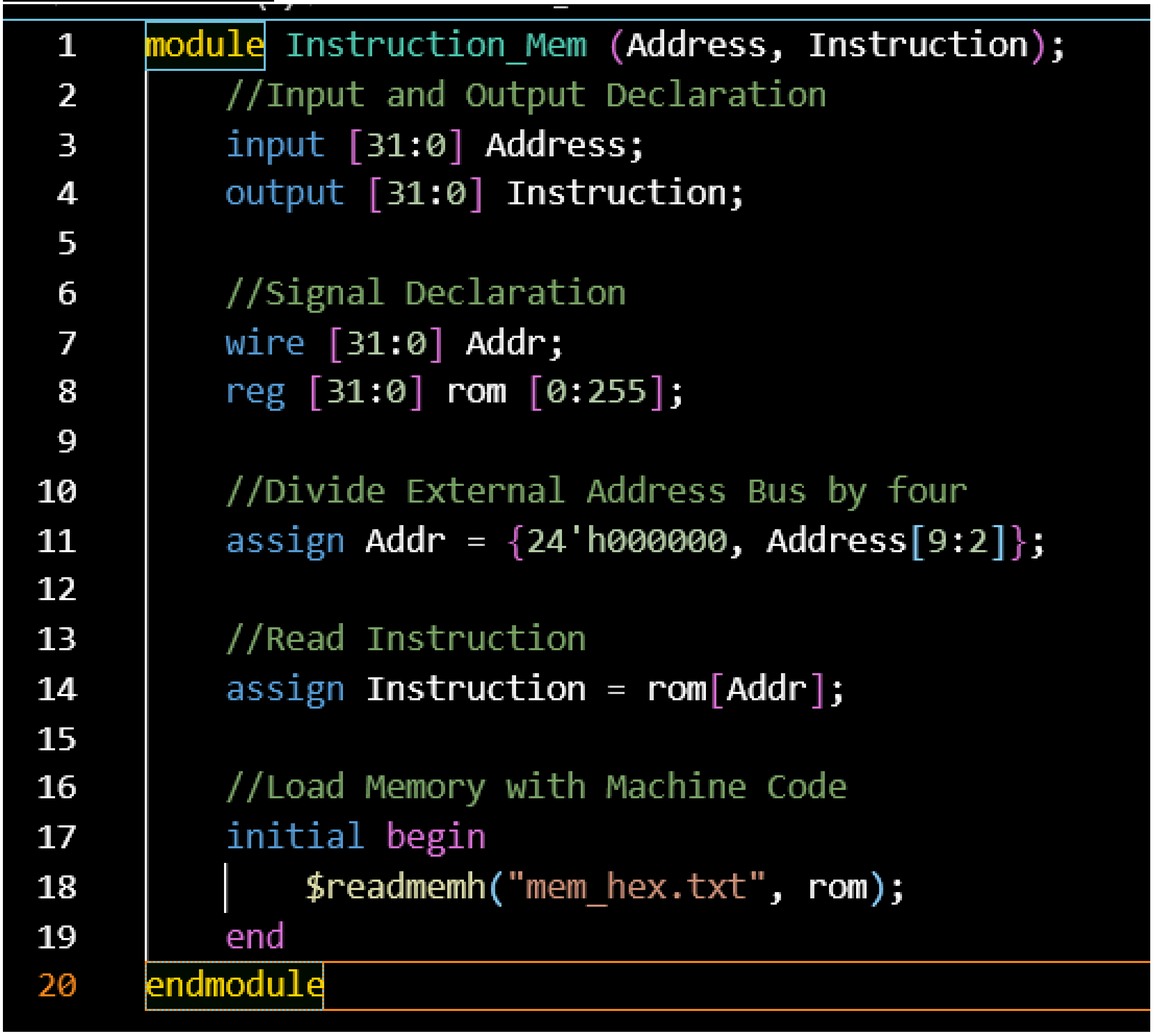
```
module Top(clk, rst, InstructionF, Mem_RDM, MemWriteM, PCF, ALUoutM, RD2_Reg_File_aft_muxM);
  //Input and Output Declaration
  input clk, rst;
  input [31:0] InstructionF, Mem_RDM;
  output MemWriteM;
  output [31:0] PCF, ALUoutM, RD2_Reg_File_aft_muxM;

  //Signal Declaration
  wire [31:0] InstructionD;
  wire PCSrcE, RegWriteM, RegWriteW, ResultSrcE_0, FlushE, StallD, StallF, FlushD, PctargetSrcD, RegWriteD, ALUSrcD, MemWriteD, BranchD, JumpD;
  wire [1:0] ForwardAE, ForwardBE, ImmSrcD, ResultSrcD;
  wire [2:0] ALUControlD;
  wire [4:0] Rs1E, Rs2E, RdM, RdW, RdE;

  //Modules Instanciation
  DataPath DataPath_inst(clk, rst, BranchD, JumpD, MemWriteD, ALUSrcD, RegWriteD, PctargetSrcD, ResultSrcD, ImmSrcD, ALUControlD, InstructionF, Mem_RDM,
  ALUoutM, StallD, StallF, FlushD, FlushE, ForwardAE, ForwardBE, Rs1E, Rs2E, RdE, RdM, RdW, RegWriteM, RegWriteW, ResultSrcE_0, PCSrcE, InstructionD,
  MemWriteM, PCF, RD2_Reg_File_aft_muxM);
  control_logic control_logic_inst(rst, InstructionD[6:0], InstructionD[14:12], InstructionD[30], BranchD, JumpD, ResultSrcD, MemWriteD, ALUSrcD, ImmSrcD,
  RegWriteD, ALUControlD, PctargetSrcD);
  Hazard_control Hazard_control_instance(rst, Rs1E, Rs2E, InstructionD[19:15], InstructionD[24:20], RdM, RdW, RdE, RegWriteM, RegWriteW, ResultSrcE_0, PCSrcE,
  ForwardAE, ForwardBE, FlushE, StallD, StallF, FlushD);
endmodule
```

TestBench



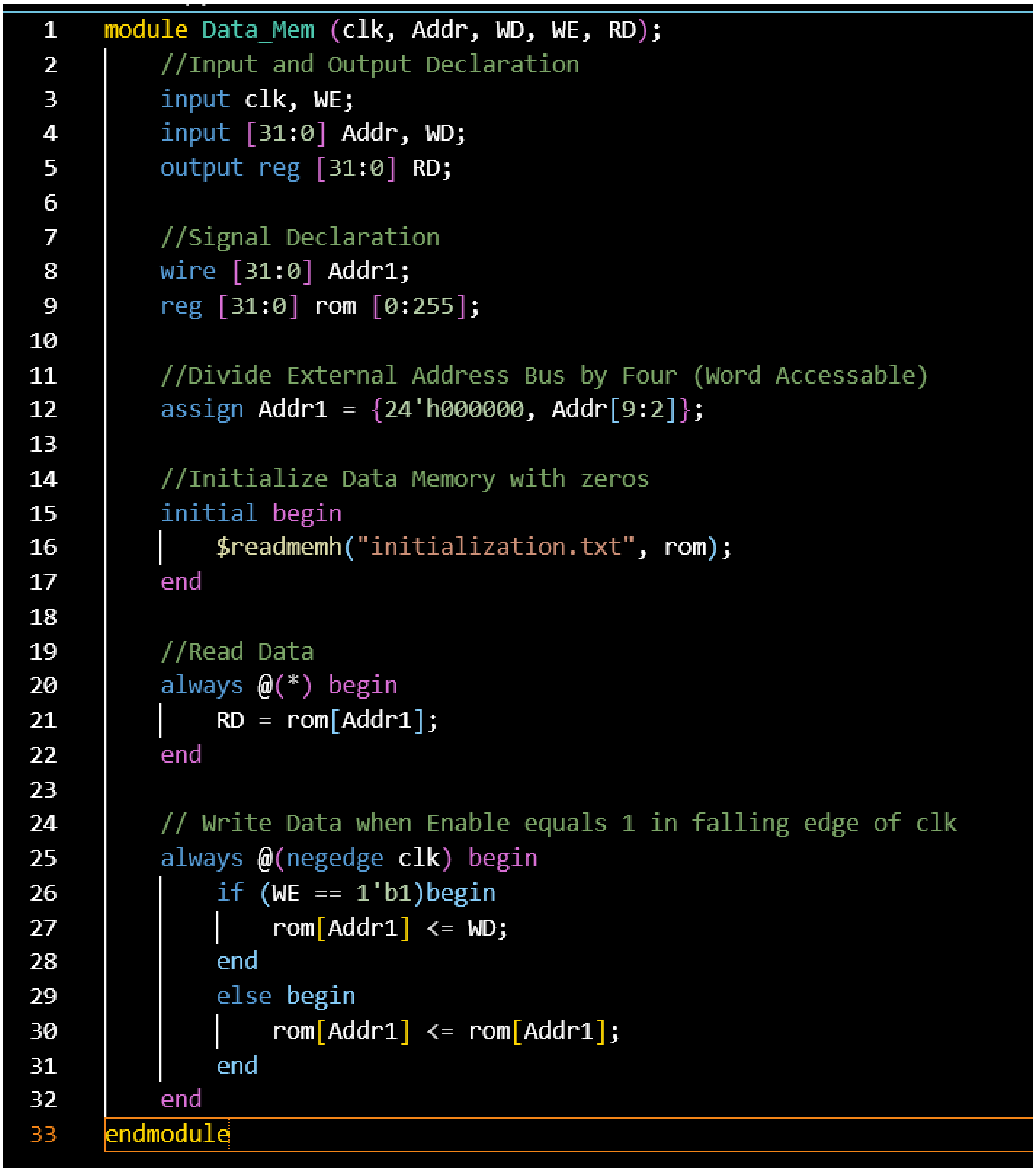
```
module RISCV_tb();
  //Signal Declaration
  reg clk, rst;
  wire [31:0] Instruction, RD_Memory, ALUout, RD2_Reg_File, PC;
  wire MemWrite;

  //Modules Instanciation
  Top Top_instance(clk, rst, Instruction, RD_Memory, MemWrite, PC, ALUout, RD2_Reg_File);
  Instruction_Mem Instruction_Mem_instance(PC, Instruction);
  Data_Mem Data_Mem_instance(clk, ALUout, RD2_Reg_File, MemWrite, RD_Memory);

  // clock generation
  initial begin
  clk = 1'b0;
  forever #5 clk = ~clk;
  end

  // rst the system in the beginning
  initial begin
  rst = 1'b1;
  #20 rst = ~rst;
  end
endmodule
```

Instruction Memory



```
module Instruction_Mem (Address, Instruction);
    //Input and Output Declaration
    input [31:0] Address;
    output [31:0] Instruction;

    //Signal Declaration
    wire [31:0] Addr;
    reg [31:0] rom [0:255];

    //Divide External Address Bus by four
    assign Addr = {24'h000000, Address[9:2]};

    //Read Instruction
    assign Instruction = rom[Addr];

    //Load Memory with Machine Code
    initial begin
        $readmemh("mem_hex.txt", rom);
    end
endmodule
```

Data Memory



```verilog
module Data_Mem (clk, Addr, WD, WE, RD);
    //Input and Output Declaration
    input clk, WE;
    input [31:0] Addr, WD;
    output reg [31:0] RD;

    //Signal Declaration
    wire [31:0] Addr1;
    reg [31:0] rom [0:255];

    //Divide External Address Bus by Four (Word Accessable)
    assign Addr1 = {24'h000000, Addr[9:2]};

    //Initialize Data Memory with zeros
    initial begin
        $readmemh("initialization.txt", rom);
    end

    //Read Data
    always @(*) begin
        RD = rom[Addr1];
    end

    // Write Data when Enable equals 1 in falling edge of clk
    always @(negedge clk) begin
        if (WE == 1'b1)begin
            rom[Addr1] <= WD;
        end
        else begin
            rom[Addr1] <= rom[Addr1];
        end
    end
endmodule
```

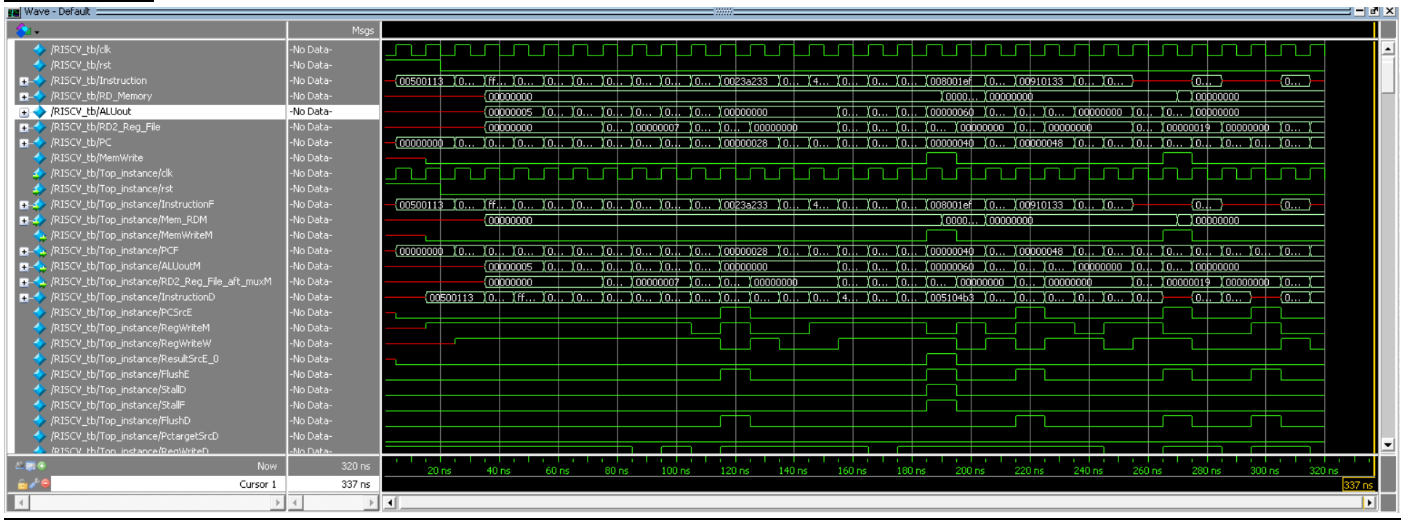
Assembly Code

```
main: addi x2, x0, 5    # x2 = 5 0 00500113
addi x3, x0, 12         # x3 = 12 4 00C00193
addi x7, x3, -9         # x7 = (12 - 9) = 3 8 FF718393
or x4, x7, x2           # x4 = (3 OR 5) = 7 C 0023E233
and x5, x3, x4          # x5 = (12 AND 7) = 4 10 0041F2B3
add x5, x5, x4          # x5 = 4 + 7 = 11 14 004282B3
beq x5, x7, end         # shouldn't be taken 18 02728863
#slt x4, x3, x4         # x4 = (12 < 7) = 0 1C 0041A233
beq x4, x0, around      # should be taken 20 00020463
addi x5, x0, 0          # shouldn't execute 24 00000293
around: #slt x4, x7, x2  # x4 = (3 < 5) = 1 28 0023A233
add x7, x4, x5          # x7 = (1 + 11) = 12 2C 005203B3
sub x7, x7, x2          # x7 = (12 - 5) = 7 30 402383B3
sw x7, 84(x3)           # [96] = 7 34 0471AA23
lw x2, 96(x0)           # x2 = [96] = 7 38 06002103
add x9, x2, x5          # x9 = (7 + 11) = 18 3C 005104B3
jal x3, end             # jump to end, x3 = 0x44 40 008001EF
addi x2, x0, 1          # shouldn't execute 44 00100113
end: add x2, x2, x9     # x2 = (7 + 18) = 25 48
sw x2, 0x20(x3)         # [100] = 25 0221A023
done: beq x2, x2, done  # infinite loop 50 00210063
```

Machine Code

0x00500113	0x00C00193	0xFF718393	0x0023E233	0x0041F2B3
0x004282B3	0x02728463	0x00020463	0x00000293	0x005203B3
0x402383B3	0x0471AA23	0x06002103	0x005104B3	0x008001EF
0x00100113	0x00910133	0x0221A023	0x00210063	

Output


```
Wave - Default
                                                  Msgs
/RISCV_tb/clk                                     -No Data-
/RISCV_tb/rst                                     -No Data-
/RISCV_tb/Instruction                             -No Data-
/RISCV_tb/RD_Memory                               -No Data-
/RISCV_tb/ALUout                                  -No Data-
/RISCV_tb/RD2_Reg_File                            -No Data-
/RISCV_tb/PC                                      -No Data-
/RISCV_tb/MemWrite                                -No Data-
/RISCV_tb/Top_instance/clk                        -No Data-
/RISCV_tb/Top_instance/rst                        -No Data-
/RISCV_tb/Top_instance/InstructionF               -No Data-
/RISCV_tb/Top_instance/Mem_RDM                    -No Data-
/RISCV_tb/Top_instance/MemWriteM                  -No Data-
/RISCV_tb/Top_instance/PCF                        -No Data-
/RISCV_tb/Top_instance/ALUoutM                    -No Data-
/RISCV_tb/Top_instance/RD2_Reg_File_aft_muxM      -No Data-
/RISCV_tb/Top_instance/InstructionD               -No Data-
/RISCV_tb/Top_instance/PCSrcE                     -No Data-
/RISCV_tb/Top_instance/RegWriteM                  -No Data-
/RISCV_tb/Top_instance/RegWriteW                  -No Data-
/RISCV_tb/Top_instance/ResultSrcE_0               -No Data-
/RISCV_tb/Top_instance/FlushE                     -No Data-
/RISCV_tb/Top_instance/StallD                     -No Data-
/RISCV_tb/Top_instance/StallF                     -No Data-
/RISCV_tb/Top_instance/FlushD                     -No Data-
/RISCV_tb/Top_instance/PctargetSrcD               -No Data-
/RISCV_tb/Top_instance/RegWriteD                  -No Data-
                                        Now       320 ns
                                   Cursor 1       337 ns
```