```
module mips_pipeline(clk, reset);
input clk, reset;

    // ********************************************************************
    //                             Signal Declarations
    // ********************************************************************

    // IF Signal Declarations

    wire [31:0] IF_instr, IF_pc, IF_pc_next, IF_pc4;

    // ID Signal Declarations

    reg [31:0] ID_instr, ID_pc4;  // pipeline register values from EX

    wire [5:0] ID_op, ID_funct;
    wire [4:0] ID_rs, ID_rt, ID_rd;
    wire [15:0] ID_immed;
    wire [25:0] ID_jump_offset;
    wire [31:0] ID_extend, ID_rd1, ID_rd2, ID_jump_in, ID_jump_offset_shifted;
    wire [31:0] stall_mux_out;

    assign ID_op = ID_instr[31:26];
    assign ID_rs = ID_instr[25:21];
    assign ID_rt = ID_instr[20:16];
    assign ID_rd = ID_instr[15:11];
    assign ID_immed = ID_instr[15:0];
    assign ID_jump_offset = ID_instr[25:0];

    wire ID_RegWrite, ID_Branch, ID_RegDst, ID_MemtoReg,  // ID Control signals
         ID_MemRead, ID_MemWrite, ID_ALUSrc, ID_Jump;
    wire [1:0] ID_ALUOp;

    // EX Signals

    reg  [31:0] EX_pc4, EX_extend, EX_rd1, EX_rd2;
    wire [31:0]  EX_offset, EX_btgt, EX_alub, EX_ALUOut;
    reg  [4:0]  EX_rt, EX_rd, EX_rs;
    wire [4:0]  EX_RegRd;
    wire [5:0] EX_funct;

    reg  EX_RegWrite, EX_Branch, EX_RegDst, EX_MemtoReg,  // EX Control Signals
         EX_MemRead, EX_MemWrite, EX_ALUSrc;

    reg  Stall = 0;
    reg  [31:0] add_increment = 32'd4;

    wire EX_Zero;

    reg  [1:0] EX_ALUOp;
    wire [2:0] EX_Operation;

   // MEM Signals

    wire MEM_PCSrc;

    reg  MEM_RegWrite, MEM_Branch, MEM_MemtoReg,
         MEM_MemRead, MEM_MemWrite, MEM_Zero;

    reg  [31:0] MEM_btgt, MEM_ALUOut, MEM_rd2;
    wire [31:0] MEM_memout;
    reg  [5:0] MEM_RegRd;

    // WB Signals

    reg WB_RegWrite, WB_MemtoReg;  // WB Control Signals

    reg  [31:0] WB_memout, WB_ALUOut;
    wire [31:0] WB_wd;
```

```
    reg  [4:0] WB_RegRd;

    // Forwarding Unit
    reg  [1:0] ForwardA, ForwardB;
    wire [31:0] EX_fw_a_out, EX_fw_b_out;


    // ********************************************************************
    //                             IF Stage
    // ********************************************************************

    // IF Hardware

    reg32               IF_PC(clk, reset, IF_pc_next, IF_pc);

    add32              IF_PCADD(IF_pc, add_increment, IF_pc4);

    mux2 #(32)  IF_PCMUX(MEM_PCSrc, IF_pc4, MEM_btgt, ID_jump_in);

    rom32              IMEM(IF_pc, IF_instr);

    // ********************************************************************
    //                            Lab 4 Code
    // ********************************************************************

    assign ID_jump_offset_shifted = ID_jump_offset << 2;

    mux2 #(32) JMPMUX(ID_Jump, ID_jump_in, ID_jump_offset_shifted, IF_pc_next);

    // ********************************************************************
    //                          END Lab 4 Code
    // ********************************************************************


    always @(posedge clk)                   // IF/ID Pipeline Register
    begin
        if (reset)
        begin
            ID_instr <= 0;
            ID_pc4   <= 0;
        end
        else if(Stall)
        begin
          ID_instr <= ID_instr;
          ID_pc4   <= ID_pc4;
        end
        else begin
            ID_instr <= IF_instr;
            ID_pc4   <= IF_pc4;
        end
    end

    // ********************************************************************
    //                             ID Stage
    // ********************************************************************

    reg_file    RFILE(clk, WB_RegWrite, ID_rs, ID_rt, WB_RegRd, ID_rd1, ID_rd2,
WB_wd);

    // sign-extender
    assign ID_extend = { {16{ID_immed[15]}}, ID_immed };

    control_pipeline CTL(.opcode(ID_op), .RegDst(ID_RegDst),
                         .ALUSrc(ID_ALUSrc), .MemtoReg(ID_MemtoReg),
                         .RegWrite(ID_RegWrite), .MemRead(ID_MemRead),
                         .MemWrite(ID_MemWrite), .Branch(ID_Branch),
                         .Jump(ID_Jump), .ALUOp(ID_ALUOp));
    always @(posedge clk)                   // ID/EX Pipeline Register
    begin
        if (reset)
        begin
            EX_RegDst   <= 0;
                  EX_ALUOp    <= 0;
            EX_ALUSrc   <= 0;
            EX_Branch   <= 0;
            EX_MemRead  <= 0;
            EX_MemWrite <= 0;
            EX_RegWrite <= 0;
            EX_MemtoReg <= 0;

            EX_pc4      <= 0;
            EX_rd1      <= 0;
            EX_rd2      <= 0;
            EX_extend   <= 0;
            EX_rt       <= 0;
            EX_rd       <= 0;
            EX_rs       <= 0;
        end
        else if (Stall)
        begin

            EX_rd       <= EX_rd;
            EX_rs       <= EX_rs;
            EX_rt       <= EX_rt;
            EX_RegWrite <= ID_RegWrite;
        end
        else begin
            EX_RegDst   <= ID_RegDst;
            EX_ALUOp    <= ID_ALUOp;
            EX_ALUSrc   <= ID_ALUSrc;
            EX_Branch   <= ID_Branch;
            EX_MemRead  <= ID_MemRead;
            EX_MemWrite <= ID_MemWrite;
            EX_RegWrite <= ID_RegWrite;
            EX_MemtoReg <= ID_MemtoReg;

            EX_pc4      <= ID_pc4;
            EX_rd1      <= ID_rd1;
            EX_rd2      <= ID_rd2;
            EX_extend   <= ID_extend;
            EX_rt       <= ID_rt;
            EX_rd       <= ID_rd;
            EX_rs       <= ID_rs;
        end
    end

    // ********************************************************************
    //                             EX Stage
    // ********************************************************************

    assign EX_offset = EX_extend << 2;

    assign EX_funct = EX_extend[5:0];

    add32               EX_BRADD(EX_pc4, EX_offset, EX_btgt);

    mux4 FWAMUX(ForwardA, EX_rd1, WB_wd, MEM_ALUOut, 32'd0, EX_fw_a_out);
    mux4 FWBMUX(ForwardB, EX_rd2, WB_wd, MEM_ALUOut, 32'd0, EX_fw_b_out);

    mux2 #(32)  ALUMUX(EX_ALUSrc, EX_fw_b_out, EX_extend, EX_alub);

    alu                 EX_ALU(EX_Operation, EX_fw_a_out, EX_alub, EX_ALUOut, EX
_Zero);

    mux2 #(5)   EX_RFMUX(EX_RegDst, EX_rt, EX_rd, EX_RegRd);
```

```
    alu_ctl     EX_ALUCTL(EX_ALUOp, EX_funct, EX_Operation);

    always @(posedge clk)                   // EX/MEM Pipeline Register
        begin
        if (reset)
        begin
            MEM_Branch   <= 0;
            MEM_MemRead  <= 0;
            MEM_MemWrite <= 0;
            MEM_RegWrite <= 0;
            MEM_MemtoReg <= 0;
            MEM_Zero     <= 0;

            MEM_btgt     <= 0;
            MEM_ALUOut   <= 0;
            MEM_rd2      <= 0;
            MEM_RegRd    <= 0;
        end
        else begin
            MEM_Branch   <= EX_Branch;
            MEM_MemRead  <= EX_MemRead;
            MEM_MemWrite <= EX_MemWrite;
            MEM_RegWrite <= EX_RegWrite;
            MEM_MemtoReg <= EX_MemtoReg;
            MEM_Zero     <= EX_Zero;

            MEM_btgt     <= EX_btgt;
            MEM_ALUOut   <= EX_ALUOut;
            MEM_rd2      <= EX_rd2;
            MEM_RegRd    <= EX_RegRd;
        end
    end

    // ********************************************************************
    //                             MEM Stage
    // ********************************************************************

    mem32               MEM_DMEM(clk, MEM_MemRead, MEM_MemWrite, MEM_ALUOut, MEM
_rd2, MEM_memout);

    and                 MEM_BR_AND(MEM_PCSrc, MEM_Branch, MEM_Zero);

    always @(posedge clk)               // MEM/WB Pipeline Register
    begin
        if (reset)
        begin
            WB_RegWrite <= 0;
            WB_MemtoReg <= 0;
            WB_ALUOut   <= 0;
            WB_memout   <= 0;
            WB_RegRd    <= 0;
        end
        else begin
            WB_RegWrite <= MEM_RegWrite;
            WB_MemtoReg <= MEM_MemtoReg;
            WB_ALUOut   <= MEM_ALUOut;
            WB_memout   <= MEM_memout;
            WB_RegRd    <= MEM_RegRd;
        end
    end

    // ********************************************************************
    //                             WB Stage
    // ********************************************************************
    mux2 #(32)  WB_WRMUX(WB_MemtoReg, WB_ALUOut, WB_memout, WB_wd);


    // ********************************************************************
    //                         Forwarding Unit
    // *********************************************************************
    always @(*)
    begin
      ForwardA = 2'b00;
      ForwardB = 2'b00;

      // EX Hazard
      if(MEM_RegWrite && (MEM_RegRd != 0) && (MEM_RegRd == EX_rs))
        ForwardA = 2'b10;

      if(MEM_RegWrite && (MEM_RegRd != 0) && (MEM_RegRd == EX_rt))
        ForwardB = 2'b10;

      //MEM Hazard
      if(WB_RegWrite && (WB_RegRd != 0) && !(MEM_RegWrite && (MEM_RegRd !=0)
      && (MEM_RegRd != EX_rs) && (WB_RegRd == EX_rs)))
        ForwardA = 2'b01;

      if(WB_RegWrite && (WB_RegRd != 0) && !(MEM_RegWrite && (MEM_RegRd !=0)
      && (MEM_RegRd != EX_rt) && (WB_RegRd == EX_rt)))
        ForwardB = 2'b01;
    end

    // *********************************************************************
    //                           Stalling Unit
    // *********************************************************************
    always @(posedge clk)
    begin
    if((EX_MemRead == 1'b1) && ((EX_rt == ID_rs) || (EX_rt == ID_rt)))
          begin
            Stall       <= 1;
            EX_ALUOp    <= 0;
            EX_ALUSrc   <= 0;
            EX_Branch   <= 0;
            EX_MemRead  <= 0;
            EX_MemWrite <= 0;
            EX_RegWrite <= 0;
            EX_MemtoReg <= 0;
            add_increment = 32'd0;
          end
        else
          begin
            Stall        <= 0;
            add_increment = 32'd4;
          end
      end
endmodule
```

```
//----------------------------------------------------------------------------
// Title         : MIPS Single-Cycle Control Unit
// Project       : ECE 313 - Computer Organization
//----------------------------------------------------------------------------
// File          : control_single.v
// Author        : John Nestor  <nestorj@lafayette.edu>
// Organization  : Lafayette College
// 
// Created       : November 2002
// Last modified : 7 January 2005
//----------------------------------------------------------------------------
// Description :
//   Control unit for the MIPS pipelined  processor implementation described
//   Section 6.3 of "Computer Organization and Design, 3rd ed."
//   by David Patterson & John Hennessey, Morgan Kaufmann, 2004 (COD3e).
//
//   It implements the function specified in Figure 6.25 on p. 401 of COD3e.
//
//----------------------------------------------------------------------------

module control_pipeline(opcode, RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, Mem
Write, Branch, Jump, ALUOp);
    input [5:0] opcode;
    output RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, Jump;
    output [1:0] ALUOp;
    reg    RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, Jump;
    reg    [1:0] ALUOp;

    parameter R_FORMAT = 6'd0;
    parameter LW = 6'd35;
    parameter SW = 6'd43;
    parameter BEQ = 6'd4;
    parameter J = 6'd2;

    always @(opcode)
    begin
        case (opcode)
          R_FORMAT :
          begin
              RegDst=1'b1; ALUSrc=1'b0; MemtoReg=1'b0; RegWrite=1'b1; MemRead=1'
b0;
              MemWrite=1'b0; Branch=1'b0; ALUOp = 2'b10; Jump = 1'b0;
          end
          LW :
          begin
              RegDst=1'b0; ALUSrc=1'b1; MemtoReg=1'b1; RegWrite=1'b1; MemRead=1'
b1;
              MemWrite=1'b0; Branch=1'b0; ALUOp = 2'b00; Jump = 1'b0;
          end
          SW :
          begin
              RegDst=1'bx; ALUSrc=1'b1; MemtoReg=1'bx; RegWrite=1'b0; MemRead=1'
b0;
              MemWrite=1'b1; Branch=1'b0; ALUOp = 2'b00; Jump = 1'b0;
          end
          BEQ :
          begin
              RegDst=1'bx; ALUSrc=1'b0; MemtoReg=1'bx; RegWrite=1'b0; MemRead=1'
b0;
              MemWrite=1'b0; Branch=1'b1; ALUOp = 2'b01; Jump = 1'b0;
          end
          J   :
          begin
              RegDst=1'bx; ALUSrc=1'b0; MemtoReg=1'bx; RegWrite=1'b0; MemRead=1'
b0;
              MemWrite=1'b0; Branch=1'b0; ALUOp = 2'bxx; Jump = 1'b1;
          end

          default
```

```
          begin
              $display("control_single unimplemented opcode %d", opcode);
              RegDst=1'b0; ALUSrc=1'b0; MemtoReg=1'b0; RegWrite=1'b0; MemRead=1'
b0;
              MemWrite=1'b0; Branch=1'b0; ALUOp = 2'b00; Jump = 1'b0;
          end
        endcase
    end
endmodule
```

```
//    A simple 32-bit by 32-word read-only memory model
//    ECE 313 Fall 2002


//  11/21/02 - modified to access up to 64 words JN

module rom32(address, data_out);
  input  [31:0] address;
  output [31:0] data_out;
  reg    [31:0] data_out;

  parameter BASE_ADDRESS = 25'd0; // address that applies to this memory

  wire [5:0] mem_offset;
  wire address_select;

  assign mem_offset = address[7:2];  // drop 2 LSBs to get word offset

  assign address_select = (address[31:8] == BASE_ADDRESS);  // address decoding

  always @(address_select or mem_offset)

  begin
    if ((address % 4) != 0) $display($time, " rom32 error: unaligned address %d", address)
;
    if (address_select == 1)
    begin
      case (mem_offset)

          // ADD test:
          /*
          5'd0 : data_out = { 6'd0,  5'd0, 5'd0, 5'd2, 5'd0, 6'd32 };//add $2, $
0, $0;
          5'd1 : data_out = { 6'd0,  5'd2, 5'd2, 5'd3, 5'd0, 6'd32 };//add $3, $
2, $2;
          5'd2 : data_out = { 6'd0,  5'd3, 5'd3, 5'd4, 5'd0, 6'd32 };//add $4, $
3, $3;
          */


          5'd0 : data_out = { 6'd35, 5'd0, 5'd1, 16'd4 };              // lw $1,
 4($0)    r1=1
          5'd1 : data_out = { 6'd35, 5'd0, 5'd2, 16'd4 };              // lw $2,
 4($0)    r2=1
          5'd2 : data_out = { 6'd0,  5'd1, 5'd2, 5'd3, 5'd0, 6'd32 };  // add $3
, $1, $2  r3=r1+r2
          5'd3 : data_out = { 6'd0,  5'd3, 5'd3, 5'd4, 5'd0, 6'd32 };  // add $4
, $3, $3  r3=r1+r2
          5'd4 : data_out = { 6'd2,  26'd0 };                          // j 0; r
estart  loop
          5'd5 : data_out = { 32'd0 };                                 // nop



         // 5'd3 : data_out = { 6'd0,  5'd0, 5'd0, 5'd4, 5'd0, 6'd32 };  //add $
2, $0, $0;
         // 5'd4 : data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };  //add $
2, $0, $0;
         // 5'd5 : data_out = { 6'd0,  5'd0, 5'd0, 5'd6, 5'd0, 6'd32 };  //add $
2, $0, $0;
         // 5'd6 : data_out = { 6'd0,  5'd0, 5'd0, 5'd7, 5'd0, 6'd32 };  //add $
2, $0, $0;
         // 5'd7 : data_out = { 6'd0,  5'd0, 5'd0, 5'd8, 5'd0, 6'd32 };  //add $
2, $0, $0;


          //5'd0 : data_out = { 6'd0,  5'd0, 5'd0, 5'd2, 5'd0, 6'd32 };  // add
$2, $0, $0  r2=0+0
          //5'd0 : data_out = { 6'd35, 5'd0, 5'd2, 16'd4 };              // lw $
```

```
2, 4($0)    r2=1
          //5'd1 : data_out = { 6'd0,  5'd0, 5'd2, 5'd3, 5'd0, 6'd32 };  // add
$3, $0, $2  r3=0+r2
          //5'd2 : data_out = { 32'd0 };                                 // nop
          //5'd3 : data_out = { 32'd0 };                                 // nop
          //5'd4 : data_out = { 32'd0 };                                 // nop
          //5'd5 : data_out = { 6'd2,  26'd0 };                          // j 0;
 restart  loop
          //5'd6 : data_out = { 32'd0 };                                 // nop
          /*
          5'd0 : data_out = { 6'd35, 5'd0, 5'd2, 16'd4 };              // lw $2,
 4($0)    r2=1
          5'd1 : data_out = { 6'd35, 5'd0, 5'd3, 16'd8 };              // lw $3,
 8($0)    r3=2
          5'd2 : data_out = { 6'd35, 5'd0, 5'd4, 16'd20 };             // lw $4,
 20($0)  r4=5
          //5'd3 : data_out = { 6'd2, 26'd0 }; //j 0; restart  loop

          5'd3 : data_out = { 6'd0, 5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };  // add $5,
 $0, $0  r5=0
          5'd4 : data_out = 0; // no-op to stall until add is done
          5'd5 : data_out = 0; // no-op to stall until add is done
          5'd6 : data_out = 0; // no-op to stall until add is done
          5'd7 : data_out = { 6'd0, 5'd5, 5'd2, 5'd5, 5'd0, 6'd32 };  // add $5,
 $5, $1  r5 = r6 + 1
          5'd8 : data_out = 0; // no-op to stall until add is done
          5'd9 : data_out = 0; // no-op to stall until add is done
          5'd10 : data_out = 0; // no-op to stall until add is done


          5'd11 : data_out = { 6'd0, 5'd4, 5'd5, 5'd6, 5'd0, 6'd42 };  // slt $6
, $4, $5  is $5 > 54?
          5'd12 : data_out = 0; // no-op to stall until slt is done
          5'd13 : data_out = 0; // no-op to stall until slt is done
          5'd14 : data_out = 0; // no-op to stall until slt is done

          5'd15 : data_out = { 6'd4, 5'd6, 5'd0, -16'd9 };             // beq $6
, $zero, -9  if not, go back 2
          5'd16 : data_out = 32'b0; // no-op after branch
          5'd17 : data_out = 32'b0; // no-op after branch
          5'd18 : data_out = 32'b0; // no-op after branch
          5'd19 : data_out = { 6'd43, 5'd0, 5'd5, 16'd0 };             // MEM[0]
 = $5
          5'd20 : data_out = { 6'd4, 5'd0, 5'd0, -16'd18 };            // beq $0
, $0, -18 restart loop at word 3
          // add more cases here as desired
          */
          default data_out = 32'hxxxx;

      endcase

      $display($time, " reading data: rom32[%h] => %h", address, data_out);

    end
  end
endmodule
```

```
//-----------------------------------------------------------------------------
// Title         : Register File (32 32-bit registers)
// Project       : ECE 313 - Computer Organization
//-----------------------------------------------------------------------------
// File          : reg_file.v
// Author        : John Nestor  <nestorj@lafayette.edu>
// Organization  : Lafayette College
//
// Created       : October 2002
// Last modified : 7 January 2005
//-----------------------------------------------------------------------------
// Description :
//   Behavioral model of the register file  used in the implementations of the M
IPS
//   processor subset described in Ch. 5-6 of "Computer Organization and Design,
 3rd ed."
//   by David Patterson & John Hennessey, Morgan Kaufmann, 2004 (COD3e).
//
//   It implements the function specified in Fig 5-7 on p. 295 of COD3e.
//
//-----------------------------------------------------------------------------

module reg_file(clk, RegWrite, RN1, RN2, WN, RD1, RD2, WD);
  input clk;
  input RegWrite;
  input [4:0] RN1, RN2, WN;
  input [31:0] WD;
  output [31:0] RD1, RD2;

  reg [31:0] RD1, RD2;
  reg [31:0] file_array [31:1];

  always @(RN1 or file_array[RN1])
  begin
    if (RN1 == 0) RD1 = 32'd0;
    else RD1 = file_array[RN1];
    $display($time, " reg_file[%d] => %d (Port 1)", RN1, RD1);
  end

  always @(RN2 or file_array[RN2])
  begin
    if (RN2 == 0) RD2 = 32'd0;
    else RD2 = file_array[RN2];
    $display($time, " reg_file[%d] => %d (Port 2)", RN2, RD2);
  end

  always @(posedge clk)
    if (RegWrite && (WN != 0))
    begin
      file_array[WN] <= WD;
      $display($time, " reg_file[%d] <= %d (Write)", WN, WD);
    end
endmodule
```

```
//4−to−1 MUX://
module mux4(sel, a, b, c, d, y);
  input [31:0] a,b,c,d;
  input [1:0] sel;
  output [31:0] y;
  reg [31:0] y;

  always@(*)
  case (sel)
    2’d0:  y=a;
    2’d1:  y=b;
    2’d2:  y=c;
    2’d3:  y=d;
  default: $display("Error in selection");
  endcase
endmodule
```