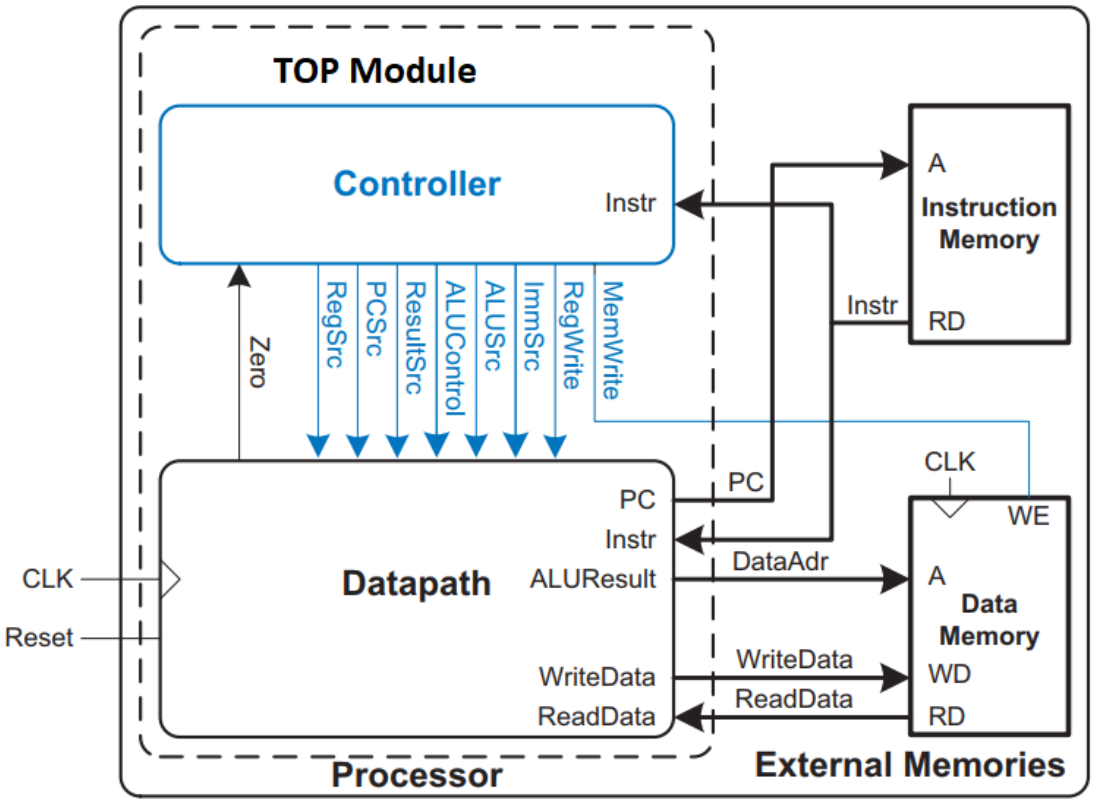
Abdallah Tarek Abdelnaby

Digital_design

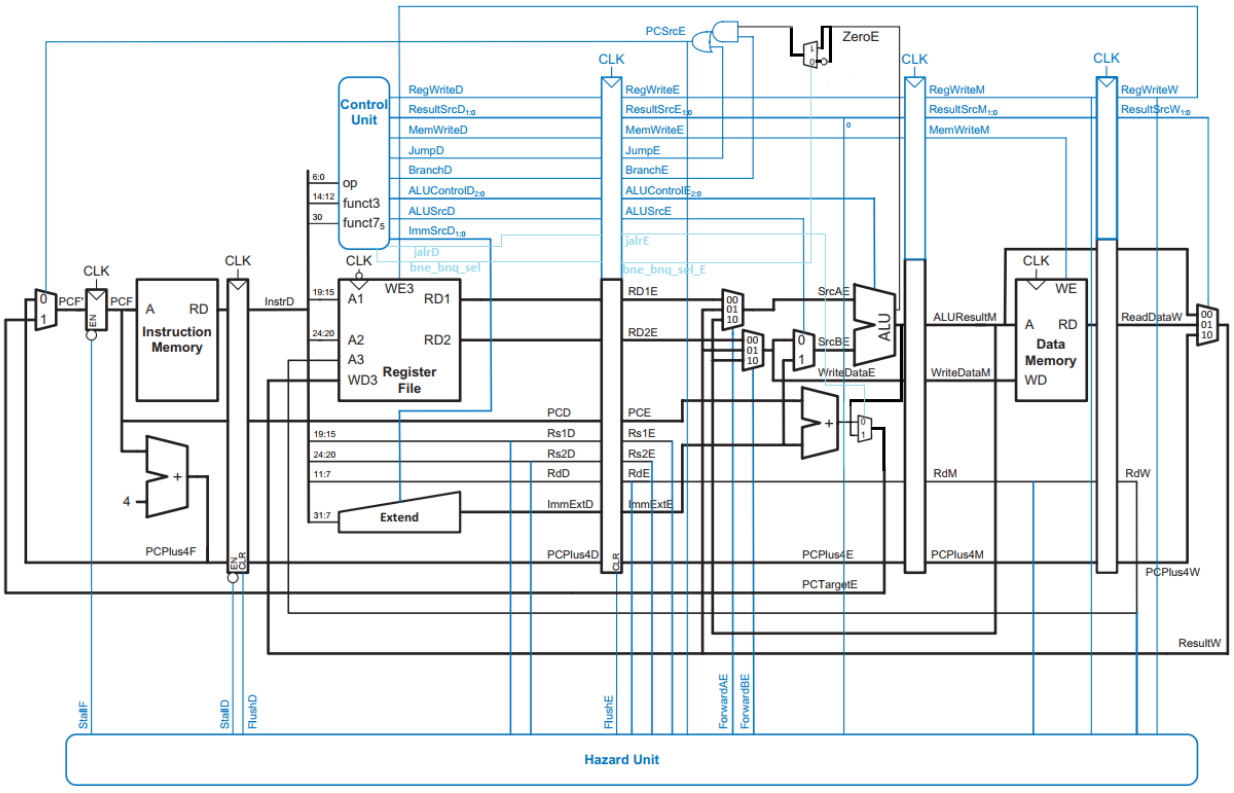
Project 1.1: RISC-V Pipelined Processor

Reviewer: eng.sara

The main block diagrams:



Test_Beanch



The building blocks of the design:

- **Memories**
  - Data_memory: 32bit word addressable RAM

```
// 32bit word data memory
module ram (address,data_in,data_out,clk,we);

parameter length = 256;
parameter width = 32;

input clk,we;

input [width-1:0]address;
input [width-1:0]data_in;
output [width-1:0]data_out;
reg [width-1:0] mem [length-1:0];

wire [$clog2(length)-1:0]trunc_add;

function [$clog2(length)-1:0] address_trunc;
input [width-1:0]add;

address_trunc = add;
endfunction

assign trunc_add = address_trunc (address);
assign data_out = mem [trunc_add];

always@(posedge clk)
begin
    if (we)
        mem [trunc_add] = data_in;
    else
        mem [trunc_add] = mem [trunc_add];
end
endmodule
```

  - Instruction_memory: 32bit Byte addressable ROM

```
module instruction_memory (address,data_out,clk);

parameter length = 256;
parameter width = 32;

input clk;
input [width-1:0] address;
output [width-1:0] data_out;
reg [width-1:0] mem [length-1:0];

assign data_out = mem [address>>2];

//$readmemh("inst.mem", mem);
endmodule
```

- **Control unit**

```
module control_unit (instr,result_sel,mem_write,alu_sel,imm_sel,reg_write,alu_control,jalr_sel,bne_beq_sel,jump,branch);

localparam lw     = 7'b0000011;
localparam sw     = 7'b0100011;
localparam R_type = 7'b0110011;
localparam I_type = 7'b0010011;
localparam jal    = 7'b1101111;
localparam beq    = 7'b1100011;
localparam jalr   = 7'b1100111;

input [31:0] instr;

output reg_write,alu_sel,mem_write,jalr_sel,branch,jump;
output reg bne_beq_sel;
output reg [2:0] alu_control;
output [1:0] result_sel,imm_sel;

wire [6:0] op_code = instr [6:0];
wire [2:0] func3 = instr [14:12];
wire func7_5 = instr [30];
wire [1:0] alu_operation;
reg [11:0] op;
assign reg_write = op [11];
assign imm_sel = op [10:9];
assign alu_sel = op [8];
assign mem_write = op [7];
assign result_sel = op [6:5];
assign branch = op [4];
assign jump = op [3];
assign alu_operation = op [2:1];
assign jalr_sel = op [0];
always@(op_code)   //main decoder
begin
    case (op_code)
//              reg_write  imm_sel  alu_sel   mem_write  result_sel   branch   jump   alu_operation   jalr_sel
    lw: op =     {1'b1,     2'b00,    1'b1,     1'b0,      2'b01,      1'b0,   1'b0,      2'b00,        1'b0};
    sw: op =     {1'b0,     2'b01,    1'b1,     1'b1,      2'b00,      1'b0,   1'b0,      2'b00,        1'b0};
    beq: op =    {1'b0,     2'b10,    1'b0,     1'b0,      2'b00,      1'b1,   1'b0,      2'b01,        1'b0};
    jal: op =    {1'b1,     2'b11,    1'b0,     1'b0,      2'b10,      1'b0,   1'b1,      2'b00,        1'b0};
    I_type: op = {1'b1,     2'b00,    1'b1,     1'b0,      2'b00,      1'b0,   1'b0,      2'b10,        1'b0};
    R_type: op = {1'b1,     2'b00,    1'b0,     1'b0,      2'b00,      1'b0,   1'b0,      2'b10,        1'b0};
    jalr: op =   {1'b1,     2'b00,    1'b1,     1'b0,      2'b10,      1'b0,   1'b1,      2'b00,        1'b1};
    default : op = 0;
    endcase
end
always@(*)   //alu decoder
begin
    if (func3 == 3'b001)
        bne_beq_sel = 1'b0;
    else
        bne_beq_sel = 1'b1;
    //alu_control = alu_control;
    if (alu_operation == 2'b0)
        alu_control = 3'b000;   //add //lw,sw
    else if (alu_operation == 2'b01)
        alu_control = 3'b001;   //sub  //beq
    else     //alu_operation = 2'b10     //R_type
    begin
        case (func3)
        3'b000:
            begin
                if ({op_code[5],func7_5} == 2'b11)
                    alu_control = 3'b001;     ///sub
                else
                    alu_control = 3'b000;     ///add
            end
        3'b010: alu_control = 3'b101;       //set less than ///slt
        3'b110: alu_control = 3'b011;       ///or
        3'b111: alu_control = 3'b010;       ///and
        default :   alu_control = 3'b000;
        endcase
    end
end
endmodule
```

- **Datapath components:**
    - Regester_file:

```
module reg_file (a1,a2,a3,wd3,rd1,rd2,clk,we3);

input clk,we3;
input [4:0] a1,a2,a3;
input [31:0] wd3;
output [31:0] rd1,rd2;

reg [31:0] mem [31:0];

assign rd1 = mem [a1];
assign rd2 = mem [a2];
assign mem[0] = 0;

always@(posedge clk)
    if (we3)
        mem [a3] <= wd3;
    else
        mem [a3] <= mem [a3];
endmodule
```

    - 2*1 MUX & 3*1 MUX:

```
module mux2 (mux_out,in0,in1,sel);
    output reg [31:0] mux_out;
    input [31:0] in0,in1;
    input sel;

    assign mux_out = sel ? in1 : in0;
endmodule

module mux3 (mux_out,in0,in1,in2,sel);
    output reg [31:0] mux_out;
    input [31:0] in0,in1,in2;
    input [1:0] sel;

    always@(*)
        case (sel)
        2'b00:  mux_out = in0;
        2'b01:  mux_out = in1;
        2'b10:  mux_out = in2;
        2'b11:  mux_out = 32'd0;
        endcase
endmodule
```

- ALU:

```
module ALU (zero,ALUout,a,b,ALUControl);

    output reg zero;
    output reg signed [31:0] ALUout;
    input [2:0] ALUControl;
    input [31:0] a,b;

    always @(*)
    begin
        case (ALUControl)
        3'b010 : ALUout = a & b;   //bitwise and
        3'b011 : ALUout = a | b;   //bitwise or
        3'b000 : ALUout = a + b;   //addition
        3'b001 : ALUout = a - b;   //subtraction
        3'b101 : ALUout = (a<b)? 1:0;   //compare
        //5 : ALUout = ~(a | b);
        default : ALUout = 0;
        endcase

        if (ALUout == 0)
            zero = 1;
        else
            zero = 0;
    end
endmodule
```

- Adder:

```
module full_adder_behave (f_sum,a,b);

    output reg [31:0] f_sum;
    input [31:0] a,b;

    always @ (*)
        f_sum = a + b ;
endmodule
```

- Sign_extend:

```
module sign_extend (in,out,sel);

input [31:7] in;
input [1:0] sel;
output reg [31:0] out;

always@(*)
    if (sel == 2'b00)    // I_type
        out = {{20{in[31]}}, in[31:20]};
    else if (sel == 2'b01)  // S_type
        out = {{20{in[31]}}, in[31:25], in[11:7]};
    else if (sel == 2'b10)       // B_type
        out = {{20{in[31]}}, in[7], in[30:25], in[11:8], 1'b0};
    else       // sel = 2'b11     // J_type
        out = {{12{in[31]}}, in[19:12], in[20], in[30:21], 1'b0};
endmodule
```

- PC_flip_flop:

```
module d_flip_flop #(parameter n = 32)(in,out,clk,reset,en);
input [n-1:0] in;
input clk,reset,en;
output reg [n-1:0] out;

always@(posedge clk)
    if (!en)
    begin
        if (reset)
            out <= 32'd0;
        else
            out <= in;
    end
    else
        out <= out;
endmodule
```

- **Datapath:**

```
module datapath (instr,instrD,read_data,clk,reset_F,reset_D,reset_E,reset_M,reset_W,pc_sel,en_F,en_D,reg_write,alu_sel,alu_control,result_sel,imm_sel,pc_out
                 ,alu_result_out,write_dataM,jalr_sel,bne_beq_sel,jump,branch,mem_write,mem_writeM);

input [31:0] instr;
input [31:0] read_data;
input clk,reset_F,reset_D,reset_E,reset_M,reset_W,pc_sel,en_F,en_D;

input reg_write,alu_sel,jalr_sel,bne_beq_sel,jump,branch,mem_write;
input [2:0] alu_control;
input [1:0] result_sel,imm_sel;

output mem_writeM;
output [31:0] pc_out;
output [31:0] alu_result_out;
output [31:0] write_dataM;
output [31:0] instrD;

wire [31:0] pcF;

wire [31:0] pcD,pc_plus4D,rd1,rd2,immexD;
wire [4:0] rs1D,rs2D,rdD;
assign rs1D = instrD [19:15];
assign rs2D = instrD [24:20];
assign rdD = instrD [11:7];

wire reg_writeE,alu_selE,jalr_selE,bne_beq_selE,mem_writeE,jumpE,branchE;
wire [2:0] alu_controlE;
wire [1:0] result_selE;
wire [31:0] pcE,pc_plus4E,rd1E,rd2E,immexE;
wire [4:0] rs1E,rs2E,rdE;
wire [31:0] write_dataE;

wire reg_writeM;
wire [1:0] result_selM;
wire [31:0] pc_plus4M,alu_resultM;
wire [4:0] rdM;

wire reg_writeW;
wire [1:0] result_selW;
wire [31:0] pc_plus4W,alu_resultW,read_dataW;
wire [4:0] rdW;
```

1

```
wire [1:0] forwardAE,forwardBE;
wire stallF,stallD,flushD,flushE;

wire [31:0] sourceB,sourceA;
wire zero,zero_flag;

//mux2 (mux_out,in0,in1,sel)
wire pc_sel_real;
wire [31:0] pc_next,pc_target,pc_plus4;
wire [31:0] pc_or_reg;    //for jalr selection
mux2 pc_mux(.mux_out (pc_next),
            .in0 (pc_plus4),
            .in1 (pc_or_reg),
            .sel (pc_sel_real));

//full_adder_behave (f_sum,a,b)
full_adder_behave add_plus_4(.f_sum (pc_plus4),
                             .a (32'd4),
                             .b (pcF));

//reg_file (a1,a2,a3,wd3,rd1,rd2,clk,we3)
wire [31:0] result;
reg_file reg_file1(.a1 (instrD[19:15]),
                   .a2 (instrD[24:20]),
                   .a3 (rdW),
                   .wd3 (result),
                   .rd1 (rd1),
                   .rd2 (rd2),
                   .clk (clk),
                   .we3 (reg_writeW));

//sign_extend (in,out,sel)
sign_extend extend(.in (instrD[31:7]),
                   .out (immexD),
                   .sel (imm_sel));

//mux3 (mux_out,in0,in1,in2,sel)
mux3 source_forwardingA(.mux_out (sourceA),
                        .in0 (rd1E),
                        .in1 (result),
                        .in2 (alu_resultM),
                        .sel (forwardAE));
```

2

```
//mux3 (mux_out,in0,in1,in2,sel)
mux3 source_forwardingB(.mux_out (write_dataE),
                        .in0 (rd2E),
                        .in1 (result),
                        .in2 (alu_resultM),
                        .sel (forwardBE));

//full_adder_behave (f_sum,a,b)
full_adder_behave add_imm(.f_sum (pc_target),
                          .a (immexE),
                          .b (pcE));

//mux2 (mux_out,in0,in1,sel)
mux2 reg_out_mux(.mux_out (sourceB),
                 .in0 (write_dataE),
                 .in1 (immexE),
                 .sel (alu_selE));

wire [31:0] alu_res;
//ALU (zero,ALUout,a,b,ALUControl)
ALU alu1(.zero (zero),
         .ALUout (alu_res),
         .a (sourceA),
         .b (sourceB),
         .ALUControl (alu_controlE));
assign zero_flag = bne_beq_selE ? zero : ~zero;
assign pc_sel_real = pc_sel ? (jumpE | (zero_flag & branchE)) : 1'b0;

//mux2 (mux_out,in0,in1,sel)
mux2 jalr_mux(.mux_out (pc_or_reg),
              .in0 (pc_target),
              .in1 (alu_res),
              .sel (jalr_selE));

//mux3 (mux_out,in0,in1,in2,sel)
mux3 result_mux(.mux_out (result),
                .in0 (alu_resultW),
                .in1 (read_dataW),
                .in2 (pc_plus4W),
                .sel (result_selW));
```

3

```
127   wire en_F_real = en_F ? stallF : 1'b0;
128   //d_flip_flop #(parameter n = 32)(in,out,clk,reset,en);
129   d_flip_flop #(32) featch(.in(pc_next),
130                            .out(pcF),
131                            .clk(clk),
132                            .reset(reset_F),
133                            .en(en_F_real));
134
135   wire reset_D_real = reset_D ? 1'b1 : flushD;
136   wire en_D_real = en_D ? stallD : 1'b0;
137   //d_flip_flop #(parameter n = 32)(in,out,clk,reset,en);
138   wire [95:0] decode_in = {instr,pcF,pc_plus4};
139   wire [95:0] decode_out;
140   assign {instrD,pcD,pc_plus4D} = decode_out;
141   d_flip_flop #(96) decode(.in(decode_in),
142                            .out(decode_out),
143                            .clk(clk),
144                            .reset(reset_D_real),
145                            .en(en_D_real));
146
147   //d_flip_flop #(parameter n = 32)(in,out,clk,reset,en);
148   wire reset_E_real = reset_E ? 1'b1 : flushE;
149   wire [186:0] excute_in = {reg_write,alu_sel,jalr_sel,bne_beq_sel,alu_control,result_sel,mem_write,rd1,rd2,pcD,rs1D,rs2D,rdD,immexD,pc_plus4D,jump,branch};
150   wire [186:0] excute_out;
151   assign {reg_writeE,alu_selE,jalr_selE,bne_beq_selE,alu_controlE,result_selE,mem_writeE,rd1E,rd2E,pcE,rs1E,rs2E,rdE,immexE,pc_plus4E,jumpE,branchE} = excute_out;
152   d_flip_flop #(187) excute(.in(excute_in),
153                             .out(excute_out),
154                             .clk(clk),
155                             .reset(reset_E_real),
156                             .en(1'b0));
157
158   //d_flip_flop #(parameter n = 32)(in,out,clk,reset,en);
159   wire [104:0] mem_in = {reg_writeE,result_selE,mem_writeE,alu_res,write_dataE,rdE,pc_plus4E};
160   wire [104:0] mem_out;
161   assign {reg_writeM,result_selM,mem_writeM,alu_resultM,write_dataM,rdM,pc_plus4M} = mem_out;
162   d_flip_flop #(105) memory(.in(mem_in),
163                             .out(mem_out),
164                             .clk(clk),
165                             .reset(reset_M),
166                             .en(1'b0));
168   //d_flip_flop #(parameter n = 32)(in,out,clk,reset,en);
169   wire [103:0] write_back_in = {reg_writeM,result_selM,alu_resultM,read_data,rdM,pc_plus4M};
170   wire [103:0] write_back_out;
171
172   assign {reg_writeW,result_selW,alu_resultW,read_dataW,rdW,pc_plus4W} = write_back_out;
173   d_flip_flop #(104) write_back(.in(write_back_in),
174                                 .out(write_back_out),
175                                 .clk(clk),
176                                 .reset(reset_W),
177                                 .en(1'b0));
178
179   //hazerd_unit (rs1D,rs2D,rdE,rs1E,rs2E,pc_sel,result_selE,rdM,reg_writeM,rdW,reg_writeW,forwardAE,forwardBE,stallF,stallD,flushD,flushE);
180   hazerd_unit u0(.rs1D(rs1D),
181                  .rs2D(rs2D),
182                  .rdE(rdE),
183                  .rs1E(rs1E),
184                  .rs2E(rs2E),
185                  .pc_sel(pc_sel_real),
186                  .result_selE(result_selE[0]),
187                  .rdM(rdM),
188                  .reg_writeM(reg_writeM),
189                  .rdW(rdW),
190                  .reg_writeW(reg_writeW),
191                  .forwardAE(forwardAE),
192                  .forwardBE(forwardBE),
193                  .stallF(stallF),
194                  .stallD(stallD),
195                  .flushD(flushD),
196                  .flushE(flushE));
197
198   //output assignment
199   assign pc_out = pcF;
200   assign alu_result_out = alu_resultM;
201
202
203   endmodule
```

4

5

- **Hazard_unit:**

```
module hazerd_unit (rs1D,rs2D,rdE,rs1E,rs2E,pc_sel,result_selE,rdM,reg_writeM,rdW
                    ,reg_writeW,forwardAE,forwardBE,stallF,stallD,flushD,flushE);

input pc_sel,result_selE,reg_writeM,reg_writeW;
input [4:0] rdE,rdM,rdW,rs1D,rs2D,rs1E,rs2E;

output reg stallF,stallD,flushD,flushE;
output reg [1:0] forwardAE,forwardBE;

always@(*)
begin
    //------------------forward for data Hazerd----------------------------
    if (((rs1E == rdM) & reg_writeM) & (rs1E != 0))
        forwardAE = 2'b10;
    else if (((rs1E == rdW) & reg_writeW) & (rs1E != 0))
        forwardAE = 2'b01;
    else
        forwardAE = 2'b00;

    if (((rs2E == rdM) & reg_writeM) & (rs2E != 0))
        forwardBE = 2'b10;
    else if (((rs2E == rdW) & reg_writeW) & (rs2E != 0))
        forwardBE = 2'b01;
    else
        forwardBE = 2'b00;
    //------------------------------------------------------------------------
end

wire lwStall;
assign lwStall = result_selE/*[0]*/ & ((rs1D == rdE) | (rs2D == rdE));
always@(lwStall or pc_sel)
begin
    stallF = 1'b0;
    stallD = 1'b0;
    flushE = 1'b0;
    flushD = 1'b0;
    //------------------stall for load Hazerd----------------------------
    if (lwStall == 1'b1)
        begin
            stallF = lwStall;
            stallD = lwStall;
        end
    //------------------------------------------------------------------------

    //------------------flush for controls Hazerd----------------------------
    if ((lwStall | pc_sel)== 1'b1)
    begin
        flushD = pc_sel;
        flushE = lwStall | pc_sel;
    end
    //------------------------------------------------------------------------
end
endmodule
```

- **Top_module:**

```
module top_module (clk,reset_F,reset_D,reset_E,reset_M,reset_W,pc_sel
                   ,en_F,en_D,instr,read_data,pc_out,write_en
                   ,write_data,alu_result);

input clk,reset_F,reset_D,reset_E,reset_M,reset_W,pc_sel,en_F,en_D;
input [31:0] instr,read_data;

output [31:0] pc_out,write_data,alu_result;
output write_en;

//control_unit (instr,zero,result_sel,mem_write,alu_sel,imm_sel,reg_write
//              ,alu_control,jalr_sel,bne_beq_sel);
wire alu_sel,reg_write,jalr_sel,bne_beq_sel,jump,branch,mem_write;
wire [1:0] result_sel,imm_sel;
wire [2:0] alu_control;
wire [31:0] instrD;
control_unit c1(.instr(instrD),
                .result_sel(result_sel),
                .mem_write(mem_write),
                .alu_sel(alu_sel),
                .imm_sel(imm_sel),
                .reg_write(reg_write),
                .alu_control(alu_control),
                .jalr_sel(jalr_sel),
                .bne_beq_sel(bne_beq_sel),
                .jump(jump),
                .branch(branch));

//datapath (instr,instrD,read_data,clk,reset_F,reset_D,reset_E,reset_M
//          ,reset_W,pc_sel,en_F,en_D,reg_write,alu_sel,alu_control,result_sel
//          ,imm_sel,zero,pc_out,alu_result_out,write_dataM,jalr_sel,bne_beq_sel
//          ,jump,branch,mem_write,mem_writeM);
datapath d1(.instr(instr),
            .instrD(instrD),
            .read_data(read_data),
            .clk(clk),
            .reset_F(reset_F),
            .reset_D(reset_D),
            .reset_E(reset_E),
            .reset_M(reset_M),
            .reset_W(reset_W),
            .pc_sel(pc_sel),
            .en_F(en_F),
            .en_D(en_D),
            .reg_write(reg_write),
            .alu_sel(alu_sel),
            .alu_control(alu_control),
            .result_sel(result_sel),
            .imm_sel(imm_sel),
            .pc_out(pc_out),
            .alu_result_out(alu_result),
            .write_dataM(write_data),
            .jalr_sel(jalr_sel),
            .bne_beq_sel(bne_beq_sel),
            .jump(jump),
            .branch(branch),
            .mem_write(mem_write),
            .mem_writeM(write_en));
endmodule
```

Test_bench:

```
module top_tb ();

localparam t = 20;

reg clk,reset_F,reset_D,reset_E,reset_M,reset_W,pc_sel,en_F,en_D;
wire write_en;
wire [31:0] instr,pc_out,write_data;
wire [31:0] alu_result;
wire [31:0] read_data;
reg [31:0] address;

//top_module (clk,reset_F,reset_D,reset_E,reset_M,reset_W,pc_sel,en_F,en_D,
//            instr,read_data,pc_out,write_en,write_data,alu_result);
top_module t1(.clk(clk),
              .reset_F(reset_F),
              .reset_D(reset_D),
              .reset_E(reset_E),
              .reset_M(reset_M),
              .reset_W(reset_W),
              .pc_sel(pc_sel),
              .en_F(en_F),
              .en_D(en_D),
              .instr(instr),
              .read_data(read_data),
              .pc_out(pc_out),
              .write_en(write_en),
              .write_data(write_data),
              .alu_result(alu_result));

//instruction_memory (address,data_out,clk);
instruction_memory instruct(.address (pc_out),
                            .data_out (instr),
                            .clk (clk));

assign address = alu_result;

ram data_mem(.address(address),
             .data_in(write_data),
             .data_out(read_data),
             .clk(clk),
             .we(write_en));
```

```
initial
begin
    clk = 0;
    forever #(t/2) clk = ~clk;
end

initial
begin
    reset_F = 1'b1;
    reset_D = 1'b1;
    reset_E = 1'b1;
    reset_M = 1'b1;
    reset_W = 1'b1;
    pc_sel = 1'b0;
    {en_F, en_D} = {2{1'b0}};
    #t
    reset_F = 1'b0;     //to free the Featch reset
    reset_D = 1'b0;     //to free the Decode reset
    #t
    reset_E = 1'b0;#t   //to free the Execute reset
    reset_M = 1'b0;#t   //to free the Memory reset
    reset_W = 1'b0;     //to free the Write_back reset
    pc_sel = 1'b1;      //to free pc_sel
    {en_F, en_D} = {2{1'b1}};   //to free the Featch and Decode enable
    #t

    #(t*50)    ///wait for the program to finish
```

```
    address = 32'h60;#t
    if (read_data == 32'h7)
    begin
        $display ("success in add 0x60");
        address = 32'h84;#t
        if (read_data == 32'h19)
        begin
            $display ("success in add 0x64");
            address = 32'h2;#t
            if (read_data == 32'h7)
            begin
                $display ("success in add 0x2");
                address = 32'hf;#t
                if (read_data != 32'h44)
                begin
                    $display ("success jalr jumping");
                    address = 32'h14;#t
                    if (read_data == 32'h88)
                        begin
                            $display ("success in add 0x14");
                            address = 32'hle;#t
                            if (read_data == 32'hc)
                            begin
                                $display ("success in add 0xle");
                                address = 32'h1f;#t
                                if (read_data == 32'hc)
                                begin
                                    $display ("success in add 0x1f")
                                    address = 32'hla;#t
                                    if (read_data == 32'h7)
                                    begin
                                        $display ("success in add 0x
                                        address = 32'h1b;#t
                                        if (read_data == 32'h7)
                                            $display ("success in ad
                                        else
                                            $display ("failure in ad
                                    end
                                    else
                                        $display ("failure in add 0x
                                end
                                else
                                    $display ("failure in add 0x1f")
                            end
                            else
                                $display ("failure in add 0xle");
                        end
                        else
                            $display ("failure in add 0x64");
                    end
                    else
                        $display ("failure jalr jumping");
                end
                else
                    $display ("failure in add 0x2");
            end
            else
                $display ("failure in add 0x64");
        end
        else
            $display ("failure in 0x60");
    $stop;
nd
ndmodule
```

The program loaded in the instruction memory:

main: addi x2, x0, 5 # x2 = 5 (0) 0x00500113

addi x3, x0, 12 # x3 = 12 (4) 0x00C00193

addi x7, x3, -9 # x7 = (12 - 9) = 3 (8) 0xFF718393

or x4, x7, x2 # x4 = (3 OR 5) = 7 (C) 0x0023E233

and x5, x3, x4 # x5 = (12 AND 7) = 4 (10) 0x0041F2B3

add x5, x5, x4 # x5 = 4 + 7 = 11 (14) 0x004282B3

beq x5, x7, end # shouldn't be taken (18) 0x02728863

slt x4, x3, x4 # x4 = (12 < 7) = 0 (1C) 0x0041A233

beq x4, x0, around # should be taken (20) 0x00020463

addi x5, x0, 0 # shouldn't execute (24) 0x00000293

around: slt x4, x7, x2 # x4 = (3 < 5) = 1 (28) 0x0023A233

add x7, x4, x5 # x7 = (1 + 11) = 12 (2C) 0x005203B3

add x30, x7, x0 # x30 = 12 =! 3 (30) #test forwarding 0x00038F33

add x31, x0, x7 # x31 = 12 =! 3 (34) #test forwarding   0x00700FB3

sw x30, 30(x0) # [30] = 12 =! 3 (38) #test forwarding   0x01E02F23

sw x31, 31(x0) # [31] = 12 =! 3 (3c) #test forwarding   0x01F02FA3

sub x7, x7, x2 # x7 = (12 - 5) = 7 (40) 0x402383B3

sw x7, 84(x3) # [96] = 7 (44) 0x0471AA23

lw x2, 96(x0) # x2 = [96] = 7 (48) 0x06002103

add x26, x2, x0 # x26 = 7 =! 5 (4c) #test stall 0x00010D33

add x27, x0, x2 # x27 = 7 =! 5 (50) #test stall 0x00200DB3

sw x26, 26(x0) # [26] = 7 =! 5 (54) #test stall 0x01A02D23

sw x27, 27(x0) # [27] = 7 =! 5 (58) #test stall 0x01B02DA3

add x9, x2, x5 # x9 = (7 + 11) = 18 (5c) 0x005104B3

jal x3, end # jump to end, x3 = 0x64 (60) 0x008001EF

addi x2, x0, 1 # shouldn't execute (64) 0x00100113

end: add x2, x2, x9 # x2 = (7 + 18) = 25 (68) 0x00910133

sw x2, 0x20(x3) # [132] = 25 (6c) 0x0221A023

bne x7,x5,test_bne # should be taken (70) 0x00539463

done: beq x2, x2, done # infinite loop (74) 0x00210063

test_bne: sw x7, 2(x0) # [2] = 7 (78) 0x00702123

addi x11, x0, 8 # x11 = 8 (7c) 0x00800593

sw x11, 15(x0) # [15] = 8 (80) 0x00B027A3

my_place: jalr x7, x11, my_place #(84) 0x060583E7

sw x3, 15(x0) # [15] != 0x64 (88) 0x003027A3

test_jlr: sw x7, 20(x0) # [20] = 0x88 (8c) 0x00702A23

bne x7,x5,done # should be taken (90) 0xFE5394E3

The machine code loaded in the instruction memory:

0x00500113

0x00C00193

0xFF718393

0x0023E233

0x0041F2B3

0x004282B3

0x04728863

0x0041A233

0x00020463

0x00000293

0x0023A233

0x005203B3

0x00038F33

0x00700FB3

0x01E02F23

0x01F02FA3

0x402383B3

0x0471AA23

0x06002103

0x00010D33

0x00200DB3

0x01A02D23

0x01B02DA3

0x005104B3

0x008001EF

0x00100113

0x00910133

0x0221A023

0x00539463

0x00210063

0x00702123

0x00800593

0x00B027A3

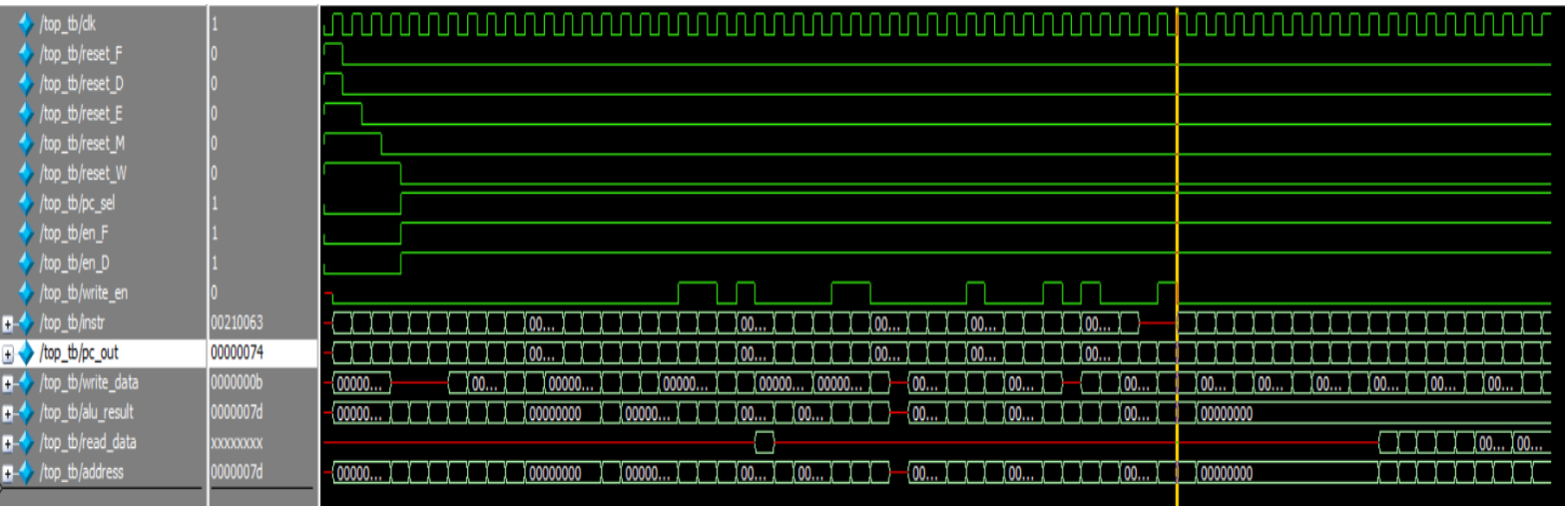
0x084583E7

0x003027A3

0x00702A23

0xFE5392E3

The result from test bench:



```
# GetModuleFileName: The specified module could not be found.
#
#
# success in add 0x60
# success in add 0x64
# success in add 0x2
# success jalr jumping
# success in add 0x14
# success in add 0x1e
# success in add 0x1f
# success in add 0x1a
# success in add 0x1b
# ** Note: $stop    : F:/ITI/RISC-V/pipline/top_tb.v(132)
#    Time: 1280 ps  Iteration: 0  Instance: /top_tb
```

Reg_file

0000001f	0000000c
0000001e	0000000c
0000001d	xxxxxxxx
0000001c	xxxxxxxx
0000001b	00000007
0000001a	00000007
00000019	xxxxxxxx
00000018	xxxxxxxx
00000017	xxxxxxxx
00000016	xxxxxxxx
00000015	xxxxxxxx
00000014	xxxxxxxx
00000013	xxxxxxxx
00000012	xxxxxxxx
00000011	xxxxxxxx
00000010	xxxxxxxx
0000000f	xxxxxxxx
0000000e	xxxxxxxx
0000000d	xxxxxxxx
0000000c	xxxxxxxx
0000000b	00000008
0000000a	xxxxxxxx
00000009	00000012
00000008	xxxxxxxx
00000007	00000088
00000006	xxxxxxxx
00000005	0000000b
00000004	00000001
00000003	00000064
00000002	00000019
00000001	xxxxxxxx
00000000	00000000

Comments:

- It stored in the address 0x60 (0d96) 0x7 which is the value of 0x7 register in the register file which verify the following instructions (as 0x7 can't have 0x7 value] unless the success of the instructions) :

  Sw, add, sub, beq, slt, addi, or, and.

- It stored in the address 0x84 (0d132) 0x19 (0d25) which is the value of 0x2 register in the register file which verify the following instructions (as 0x2 can't have 0x19 (0d25) value unless the success of the instructions) :

  lw, jal.

- It stored in the address 0x2 (0d2) 0x7 (0d7) which is the value of 0x7 register in the register file which verify the following instructions (as this instruction won't be executed if it wasn't for bne (bnq jumped is infinite loop)):

  bne.

- It didn't store in the address 0x3 (0d3) 0x64 (0d100) which is the value of 0x3 register in the register file which verify the jumping part of the following instructions (as this instruction jumps the sw instruction successfully):

  jalr.

- It didn't store in the address 0x14 (0d20) 0x88 (0d136) which is the value of 0x7 register in the register file which verify the storing of the return (pc+4) part of the following instructions (as this instruction stores the next instruction address (pc+4) in the 0x7 (destination register) successfully):

  jalr.

- It stored in the address 0x1e (0d30) 0xc (0d12) which is the value of 0x30 register in the register file which verify the successful forwarding from the memory stage of an instruction to the execution of dependent instruction.

- It stored in the address 0x1f (0d31) 0xc (0d12) which is the value of 0x31 register in the register file which verify the successful forwarding from the write back stage of an instruction to the execution of dependent instruction.

- It stored in the address 0x1a (0d26) 0x7 (0d7) which is the value of 0x26 register in the register file which verify the successful stalling from lw instruction to a dependent instruction till the write back stage of lw and forwarding to the execution stage to the dependent instruction.

- It stored in the address 0x1b (0d27) 0x7 (0d7) which is the value of 0x27 register in the register file which verify the successful stalling from lw instruction to a dependent instruction till the write back stage of lw and read it directly from the register file in the decode stage of the dependent instruction.

- Every B or J type instruction successfully executed proves the control hazard solution is successful.

Note:

The pc_sel modification is going to be added later in the next patch.