**EE-475L: Computer Architecture**

**Lab Report**

**Submitted by**

2020ee162

**Submitted to**

Sir Ali Imran

**Department of Electrical Engineering**

**University of Engineering and Technology, Lahore, Pakistan.**

**RTL Code**

Register File and Memories:

|  |
| --- |
| module Regfile (      input logic rst,      clk,      write\_en,      input logic [4:0] rs1\_in,      rs2\_in,      rd,      input logic [31:0] write\_data,      output logic [31:0] rs1\_out,      rs2\_out  );    logic [31:0] mem[0:31];    logic [31:0] result;    assign result = mem[12];    logic valid\_add1, valid\_add2, valid\_write\_en;    //validations    assign valid\_add1 = |rs1\_in;    assign valid\_add2 = |rs2\_in;    assign valid\_write\_en = |rd & write\_en;    assign rs1\_out = valid\_add1 ? mem[rs1\_in] : '0;    assign rs2\_out = valid\_add2 ? mem[rs2\_in] : '0;    always\_ff @(negedge clk)      if (rst) begin        mem = '{default: '0};      end else if (write\_en) mem[rd] <= write\_data;  endmodule |

**Listing 1. Register File**

|  |
| --- |
| module Instrmem (      input logic [31:0] addr\_i,      output logic [31:0] instruction\_o  );      logic [31:0] instrmem[0:31];      assign instruction\_o=instrmem[addr\_i[6:2]];  endmodule |

**Listing 2. Instruction Memory**

|  |
| --- |
| module data\_mem (      input clk,      input rst,      mem\_wr,      input logic [31:0] addr,      data\_wr,      input logic [3:0] mask,      input logic [2:0] load\_ctrl,      output logic [31:0] mem\_data  );    logic [31:0] mem\_data\_read;    logic [31:0] mem[0:31];    assign mem\_data\_read = mem\_wr ? '0 : mem[addr[6:2]];    always\_ff @(posedge clk) begin      if (rst) mem <= '{default: '0};      else if (mem\_wr) begin        $display("mask=%b datawr=%h", mask, data\_wr);        if (mask[0]) begin          mem[addr[31:2]][7:0] = data\_wr[7:0];        end        if (mask[1]) mem[addr[31:2]][15:8] = data\_wr[15:8];        if (mask[2]) begin          $display("mask2=%b", mask[3]);          mem[addr[31:2]][23:16] = data\_wr[23:16];        end        if (mask[3]) mem[addr[31:2]][31:24] = data\_wr[31:24];      end    end    always\_comb begin      case (load\_ctrl)        3'b000: begin          case (addr[1:0])            2'b00:   mem\_data = {{24{mem\_data\_read[7]}}, mem\_data\_read[7:0]};            2'b01:   mem\_data = {{24{mem\_data\_read[15]}}, mem\_data\_read[15:8]};            2'b10:   mem\_data = {{24{mem\_data\_read[23]}}, mem\_data\_read[23:16]};            2'b11:   mem\_data = {{24{mem\_data\_read[31]}}, mem\_data\_read[31:24]};            default: mem\_data = 'x;          endcase        end        3'b001: begin          case (addr[1])            0: mem\_data = {{16{mem\_data\_read[15]}}, mem\_data\_read[15:0]};            1: mem\_data = {{16{mem\_data\_read[31]}}, mem\_data\_read[31:16]};            default: mem\_data = 'x;          endcase        end        3'b010: begin          mem\_data = mem\_data\_read;        end        3'b011: begin          case (addr[1:0])            2'b00:   mem\_data = {{24{1'b0}}, mem\_data\_read[7:0]};            2'b01:   mem\_data = {{24{1'b0}}, mem\_data\_read[15:8]};            2'b10:   mem\_data = {{24{1'b0}}, mem\_data\_read[23:16]};            2'b11:   mem\_data = {{24{1'b0}}, mem\_data\_read[31:24]};            default: mem\_data = 'x;          endcase        end        3'b100: begin          case (addr[1])            1'b0: mem\_data = {{16{1'b0}}, mem\_data\_read[15:0]};            1'b1: mem\_data = {{16{1'b0}}, mem\_data\_read[31:16]};            default: mem\_data = 'x;          endcase        end        default: mem\_data = 'x;      endcase    end  endmodule |

**Listing 3. Data Memory**

**ALU:**

|  |
| --- |
| module mux11x1 (      input  logic [31:0] i1,      i2,      i3,      i4,      i5,      i6,      i7,      i8,      i9,      i10,      i11,      input  logic [ 3:0] s,      output logic [31:0] y  );    always\_comb begin      case (s)        4'd0:  y = i1;        4'd1:  y = i2;        4'd2:  y = i3;        4'd3:  y = i4;        4'd4:  y = i5;        4'd5:  y = i6;        4'd6:  y = i7;        4'd7:  y = i8;        4'd8:  y = i9;        4'd9:  y = i10;        4'd10: y = i11;        default: y = 32'bX;      endcase    end  endmodule  module ALU (      input  logic [31:0] a\_in,      b\_in,      input  logic [ 3:0] ALUctrl,      output logic [31:0] result\_o  );    logic [31:0]        and\_res, or\_res, xor\_res, add\_sub\_res, SLT\_res, SLTU\_res, t, SLL\_res, SRL\_res, SRA\_res;    logic [31:0] mux1\_o;    logic C\_out, N, V, W, C;  //Flags    assign N = add\_sub\_res[31];    assign C = C\_out;    assign V = (add\_sub\_res[31] ^ a\_in[31]) & (~(a\_in[31] ^ b\_in[31] ^ ALUctrl[0]));    assign W = add\_sub\_res[31] ^ V;    assign t = ~b\_in;    assign mux1\_o = ALUctrl[0] ? t : b\_in;    // Operations    assign {C\_out, add\_sub\_res} = a\_in + mux1\_o + ALUctrl[0];    assign and\_res = a\_in & b\_in;    assign or\_res = a\_in | b\_in;    assign xor\_res = a\_in ^ b\_in;    assign SLT\_res = {30'd0, W};    assign SLTU\_res = {30'd0, ~C};    assign SRA\_res = a\_in >>> b\_in;    assign SRL\_res = a\_in >> b\_in;    assign SLL\_res = a\_in << b\_in;    mux11x1 ALU\_mux (        .i1 (add\_sub\_res),        .i2 (add\_sub\_res),        .i3 (SLL\_res),        .i4 (SLT\_res),        .i5 (xor\_res),        .i6 (SLTU\_res),        .i7 (SRL\_res),        .i8 (SRA\_res),        .i9 (or\_res),        .i10(and\_res),        .i11(b\_in),        .s  (ALUctrl),        .y  (result\_o)    );  endmodule |

**Listing 4. ALU**

**Controllers:**

|  |
| --- |
| module Branch\_block (      input logic [31:0] op\_a,      op\_b,      input logic [2:0] func3,      output logic branch\_taken  );    logic [31:0] branch\_sub;    logic overflow, not\_zero, neg, carry;    assign branch\_sub = op\_a - op\_b;    assign not\_zero = |branch\_sub;    assign neg = branch\_sub[31];    assign {carry, overflow} = (op\_a ^ op\_b) && (op\_a ^ branch\_sub[31]);    always\_comb begin      case (func3)        //beq        3'b000:  branch\_taken = ~not\_zero;        //bneq        3'b001:  branch\_taken = not\_zero;        //blt        3'b100:  branch\_taken = branch\_sub[31];        //bge        3'b101:  branch\_taken = ~branch\_sub[31];        //bltu        3'b110:  branch\_taken = ~carry;        //bgeu        3'b111:  branch\_taken = carry;        default: branch\_taken = 1'bx;      endcase    end  endmodule |

**Listing 5. Branch Controller**

|  |
| --- |
| module LS\_controller (      input  logic [ 2:0] func3,      input  logic [ 1:0] address,      input  logic [31:0] rdata2,      output logic [31:0] wdata\_mem,      output logic [ 2:0] load\_ctrl,      output logic [ 3:0] mask  );    localparam b = 3'b000;    localparam h = 3'b001;    localparam w = 3'b010;    localparam bu = 3'b100;    localparam hu = 3'b101;    always\_comb begin      case (func3)        b: begin          case (address)            2'b00: begin              mask = 4'b0001;              wdata\_mem = {{24{1'b0}}, {rdata2[7:0]}};            end            2'b01: begin              mask = 4'b0010;              wdata\_mem = {{16{1'b0}}, {rdata2[7:0]}, {8{1'b0}}};            end            2'b10: begin              mask = 4'b0100;              wdata\_mem = {{8{1'b0}}, {rdata2[7:0]}, {16{1'b0}}};            end            2'b11: begin              mask = 4'b1000;              wdata\_mem = {{rdata2[7:0]}, {24{1'b0}}};            end            default: begin              mask = 'x;              wdata\_mem = 'x;            end          endcase          load\_ctrl = 3'b000;        end        h: begin          case (address[1])            0: begin              mask = 4'b0011;              wdata\_mem = {{16{1'b0}}, {rdata2[15:0]}};            end            1: begin              mask = 4'b1100;              wdata\_mem = {{rdata2[15:0]}, {16{1'b0}}};            end            default: begin              mask = 'x;              wdata\_mem = 'x;            end          endcase          load\_ctrl = 3'b001;        end        w: begin          mask = 4'b1111;          wdata\_mem = rdata2;          load\_ctrl = 3'b010;        end        bu: begin          mask = 'x;          wdata\_mem = rdata2;          load\_ctrl = 3'b011;        end        hu: begin          mask = 'x;          wdata\_mem = rdata2;          load\_ctrl = 3'b100;        end        default: begin          mask = 'x;          wdata\_mem = rdata2;          load\_ctrl = 3'b100;        end      endcase    end  endmodule |

**Listing 6. Load Store Controller**

Main Controller:

|  |
| --- |
| `include "LS\_controller.sv"  module Controller (      input clk,      input rst,      input logic [31:0] instruction,      input logic [1:0] mem\_col,      input logic b\_taken,      output logic [31:0] wdata\_mem,      output logic [3:0] ALUctrl,      output logic [2:0] load\_ctrl,      output logic [3:0] mask,      output logic mem\_wr,      output logic A\_sel,      B\_sel,      reg\_wr,      PC\_sel,      output logic [1:0] wb\_sel,      input logic [31:0] rdata2  );    localparam R\_type = 5'b01100;    localparam I\_type = 5'b00100;    localparam Load\_type = 5'b00000;    localparam S\_type = 5'b01000;    localparam B\_type = 5'b11000;    localparam J\_type = 5'b11011;    localparam Jalr\_type = 5'b11001;    localparam lui\_type = 5'b01101;    localparam auipc\_type = 5'b00101;    logic [6:0] opcode;    logic func7;    logic [2:0] func3;    assign func3  = instruction[14:12];    assign opcode = instruction[6:0];    assign func7  = instruction[30];    LS\_controller LS\_controller\_instance (        .func3(func3),        .address(mem\_col),        .rdata2(rdata2),        .wdata\_mem(wdata\_mem),        .load\_ctrl(load\_ctrl),        .mask(mask)    );    always\_comb begin      case (instruction[6:2])        R\_type: begin          casex ({            func7, func3          })            4'b0000: ALUctrl = 4'd0;  //ADD            4'b1000: ALUctrl = 4'd1;  //Sub            4'bX001: ALUctrl = 4'd2;  //SLL            4'bX010: ALUctrl = 4'd3;  //SLT            4'bX100: ALUctrl = 4'd4;  //XOR            4'bX011: ALUctrl = 4'd5;  //SLTU            4'b0101: ALUctrl = 4'd6;  //SRL            4'b1101: ALUctrl = 4'd7;  //SRA            4'bX110: ALUctrl = 4'd8;  //OR            4'bX111: ALUctrl = 4'd9;  //AND            default: ALUctrl = 4'bXXXX;          endcase          A\_sel  = 1;          PC\_sel = 0;          mem\_wr = 0;          B\_sel  = 0;          wb\_sel = 2'b01;          reg\_wr = 1;        end        I\_type: begin          casex ({            func7, func3          })            4'bX000: ALUctrl = 4'd0;  //ADD            4'bX001: ALUctrl = 4'd2;  //SLL            4'bX010: ALUctrl = 4'd3;  //SLT            4'bX100: ALUctrl = 4'd4;  //XOR            4'bX011: ALUctrl = 4'd5;  //SLTU            4'b0101: ALUctrl = 4'd6;  //SRL            4'b1101: ALUctrl = 4'd7;  //SRA            4'bX110: ALUctrl = 4'd8;  //OR            4'bX111: ALUctrl = 4'd9;  //AND            default: begin              ALUctrl = 4'bXXXX;              $display("run %b%b", func7, func3);            end          endcase          mem\_wr = 0;          A\_sel  = 1;          PC\_sel = 0;          B\_sel  = 1;          wb\_sel = 2'b01;          reg\_wr = 1;        end        Load\_type: begin          mem\_wr  = 0;          A\_sel   = 1;          PC\_sel  = 0;          B\_sel   = 1;          wb\_sel  = 2'b10;          reg\_wr  = 1;          ALUctrl = 4'd0;        end        S\_type: begin          mem\_wr  = 1;          A\_sel   = 1;          PC\_sel  = 0;          B\_sel   = 1;          wb\_sel  = 'x;          reg\_wr  = 0;          ALUctrl = 4'd0;        end        B\_type: begin          mem\_wr  = 0;          A\_sel   = 0;          B\_sel   = 1;          wb\_sel  = 'x;          reg\_wr  = 0;          ALUctrl = 4'd0;          case (b\_taken)            0: PC\_sel = 0;            1: PC\_sel = 1;            default: PC\_sel = 'x;          endcase        end        J\_type: begin          mem\_wr  = 0;          A\_sel   = 0;          B\_sel   = 1;          wb\_sel  = 2'b00;          reg\_wr  = 1;          ALUctrl = 4'd0;          PC\_sel  = 1'b1;        end        Jalr\_type: begin          mem\_wr  = 0;          A\_sel   = 1;          B\_sel   = 1;          wb\_sel  = 2'b00;          reg\_wr  = 1'b1;          ALUctrl = 4'd0;          PC\_sel  = 1'b1;        end        lui\_type: begin          mem\_wr  = 0;          A\_sel   = 1'bx;          B\_sel   = 1;          wb\_sel  = 2'b01;          reg\_wr  = 1'b1;          ALUctrl = 4'd10;          PC\_sel  = 1'b0;        end        auipc\_type: begin          mem\_wr  = 0;          A\_sel   = 0;          B\_sel   = 1;          wb\_sel  = 2'b01;          reg\_wr  = 1;          ALUctrl = 4'd0;          PC\_sel  = 1'b0;        end        default: begin          mem\_wr = 'x;          B\_sel  = 'x;          wb\_sel = 'x;          reg\_wr = 'x;        end      endcase    end  endmodule |

**Listing 7. Main Controller**

Datapath:

|  |
| --- |
| `include "Instrmem.sv"  `include "Regfile.sv"  `include "ALU.sv"  `include "data\_mem.sv"  `include "Branch\_block.sv"  module data\_path (      input logic clk,      rst,      reg\_wr,      A\_sel,      B\_sel,      mem\_wr,      PC\_sel,      input logic [1:0] wb\_sel,      input logic [3:0] mask,      input logic [2:0] load\_ctrl,      input logic [3:0] ALUctrl,      input logic [31:0] wdata\_mem,      output logic [31:0] instruction,      output logic [1:0] mem\_col,      output logic b\_taken,      output logic [31:0] rdata2  );    localparam I\_type = 5'b00100;    localparam Load\_type = 5'b00000;    localparam B\_type = 5'b11000;    localparam S\_type = 5'b01000;    localparam J\_type = 5'b11011;    localparam Jalr\_type = 5'b11001;    localparam lui\_type = 5'b01101;    localparam auipc\_type = 5'b00101;    logic [31:0] instruction\_addr;    logic [31:0] wdata, ALUresult, ReadData, rdata1;    logic [31:0] PC, PC\_mux\_o;    logic [4:0] raddr1, raddr2, waddr;    logic [31:0] rd2;    logic RegWrite;    logic [2:0] func3;    logic [31:0] ALU\_o;    logic [31:0] mem\_data;    logic [31:0] imm, ALU\_op\_b, ALU\_op\_a;    assign mem\_col = ALU\_o[1:0];    assign func3   = instruction[14:12];    assign raddr1  = instruction[19:15];    assign raddr2  = instruction[24:20];    assign waddr   = instruction[11:7];    //PC counter    initial begin      $readmemh("instructions.txt", Instrmem\_instance.instrmem);      $readmemh("registervalues.txt", Regfile\_instance.mem);    end    assign PC\_mux\_o = PC\_sel ? ALU\_o : PC + 4;    always\_ff @(posedge clk) begin      if (rst) PC <= 32'd0;      else PC <= PC\_mux\_o;    end    assign ALU\_op\_a = A\_sel ? rdata1 : PC;    assign ALU\_op\_b = B\_sel ? imm : rdata2;    always\_comb begin      case (wb\_sel)        2'b00:   wdata = PC + 4;        2'b01:   wdata = ALU\_o;        2'b10:   wdata = mem\_data;        default: wdata = 'bx;      endcase    end    Regfile Regfile\_instance (        .rst(1'b0),        .clk(clk),        .write\_en(reg\_wr),        .rs1\_in(raddr1),        .rs2\_in(raddr2),        .rd(waddr),        .write\_data(wdata),        .rs1\_out(rdata1),        .rs2\_out(rdata2)    );    ALU ALU\_instance (        .a\_in(ALU\_op\_a),        .b\_in(ALU\_op\_b),        .ALUctrl(ALUctrl),        .result\_o(ALU\_o)    );    Instrmem Instrmem\_instance (        .addr\_i(PC),        .instruction\_o(instruction)    );    data\_mem data\_mem\_instance (        .clk(clk),        .rst(rst),        .mem\_wr(mem\_wr),        .addr(ALU\_o),        .data\_wr(wdata\_mem),        .mask(mask),        .load\_ctrl(load\_ctrl),        .mem\_data(mem\_data)    );    //Immidiate generation    always\_comb begin      casex (instruction[6:2])        Load\_type, I\_type: imm = {{20{instruction[31]}}, instruction[31:20]};  //load,I        Jalr\_type: imm = {{20{instruction[31]}}, instruction[31:20]};        S\_type: imm = {{20{instruction[31]}}, instruction[31:25], instruction[11:7]};  //save        J\_type:        imm = {{12{instruction[31]}}, instruction[19:12], instruction[20], instruction[30:21], 1'b0};        B\_type:        imm = {{20{instruction[31]}}, instruction[7], instruction[30:25], instruction[11:8], 1'b0};        lui\_type, auipc\_type: imm = {{instruction[31:12]}, {12{1'b0}}};        default: begin          imm = 'x;        end      endcase    end    Branch\_block Branch\_block\_instance (        .op\_a(rdata1),        .op\_b(rdata2),        .func3(func3),        .branch\_taken(b\_taken)    );  endmodule |

**Listing 8. Datapath**

RISC-V:

|  |
| --- |
| `include "data\_path.sv"  `include "Controller.sv"  module RISC\_V (      input logic clk,      input logic rst  );    logic [31:0] instruction, wdata\_mem, rdata2;    logic reg\_wr, A\_sel, PC\_sel, B\_sel, br\_taken, mem\_wr;    logic [2:0] load\_ctrl;    logic [1:0] wb\_sel;    logic [1:0] mem\_col;    logic [3:0] ALUctrl;    logic [3:0] mask;    Controller Controller\_instance (        .clk(clk),        .rst(rst),        .instruction(instruction),        .mem\_col(mem\_col),        .b\_taken(br\_taken),        .wdata\_mem(wdata\_mem),        .ALUctrl(ALUctrl),        .load\_ctrl(load\_ctrl),        .mask(mask),        .mem\_wr(mem\_wr),        .A\_sel(A\_sel),        .B\_sel(B\_sel),        .wb\_sel(wb\_sel),        .reg\_wr(reg\_wr),        .PC\_sel(PC\_sel),        .rdata2(rdata2)    );    data\_path data\_path\_instance (        .clk(clk),        .rst(rst),        .reg\_wr(reg\_wr),        .A\_sel(A\_sel),        .B\_sel(B\_sel),        .mem\_wr(mem\_wr),        .PC\_sel(PC\_sel),        .wb\_sel(wb\_sel),        .mask(mask),        .load\_ctrl(load\_ctrl),        .ALUctrl(ALUctrl),        .wdata\_mem(wdata\_mem),        .instruction(instruction),        .mem\_col(mem\_col),        .b\_taken(br\_taken),        .rdata2(rdata2)    );  endmodule |

**Listing 9. RISC-V**

**Testbench**

Following testbench has been created for our design simulation.

|  |
| --- |
| `include "RISC\_V.sv"  module RISC\_V\_tb;    logic rst, clk;    RISC\_V RISC\_R\_instance (        .clk(clk),        .rst(rst)    );    //clock generation    localparam CLK\_PERIOD = 2;    initial begin      clk = 0;      forever begin        #(CLK\_PERIOD / 2);        clk = ~clk;      end    end    //Testbench    initial begin      rst = 1;      @(posedge clk);      rst = 0;      repeat (100) @(posedge clk);      $finish;    end    //Monitor values at posedge    always @(posedge clk) begin      $display("PC=%d factorial=%d num=%d", RISC\_R\_instance.data\_path\_instance.PC,               RISC\_R\_instance.data\_path\_instance.Regfile\_instance.mem[10],               RISC\_R\_instance.data\_path\_instance.Regfile\_instance.mem[11]);    end    initial begin      $dumpfile("RISC\_R\_dump.vcd");      $dumpvars;    end  endmodule |

**Listing 10. Testbench code**

|  |
| --- |
| #factorial.s  li a1, 4 # number =4  li a0, 1 # factorial  li t0, 0 #product  li t1, 2 #i  loop:  bgt t1, a1, end  addi t2 , t1 , 0 #t2=j  innerloop\_start:  beq t2, x0, innerloop\_end  add t0,t0,a0  addi t2,t2,-1  j innerloop\_start  innerloop\_end:  addi a0,t0,0  li t0,0  addi t1,t1,1  j loop  end:  j end |

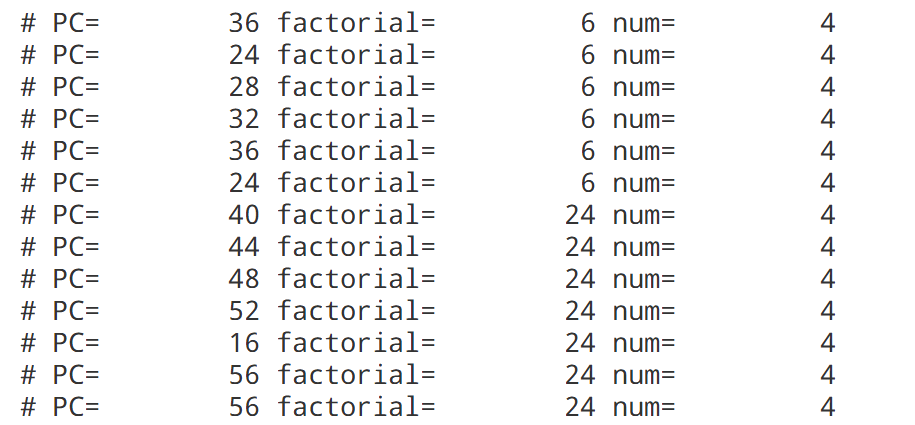
**Assembly code for Factorial**

|  |
| --- |
| **//Instructions.txt**  **00400593**  **00100513**  **00000293**  **00200313**  **0265c463**  **00030393**  **00038863**  **00a282b3**  **fff38393**  **ff5ff06f**  **00028513**  **00000293**  **00130313**  **fddff06f**  **0000006f** |

**Machine code for factorial**

**Results**

For the above testbench we get the following output.



**Figure 1. Monitor Results**