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;
```

```verilog
        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
```

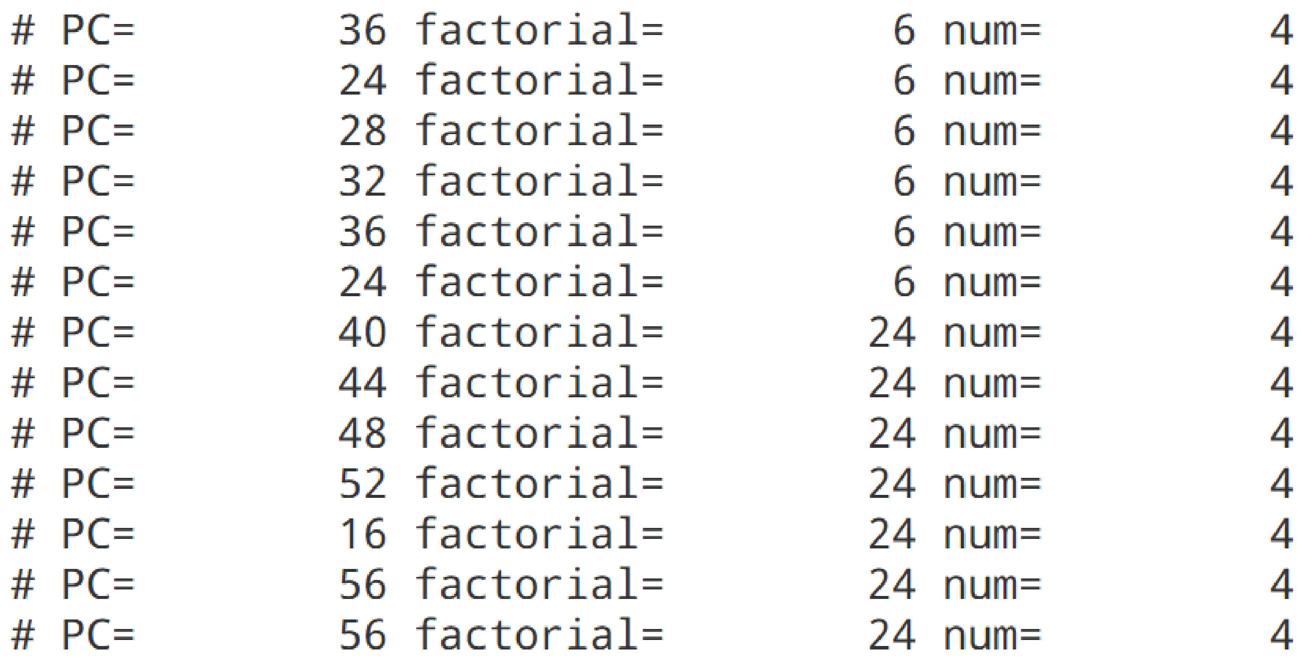
Assembly code for Factorial

```
//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.



```
# PC=          36 factorial=          6 num=          4
# PC=          24 factorial=          6 num=          4
# PC=          28 factorial=          6 num=          4
# PC=          32 factorial=          6 num=          4
# PC=          36 factorial=          6 num=          4
# PC=          24 factorial=          6 num=          4
# PC=          40 factorial=         24 num=          4
# PC=          44 factorial=         24 num=          4
# PC=          48 factorial=         24 num=          4
# PC=          52 factorial=         24 num=          4
# PC=          16 factorial=         24 num=          4
# PC=          56 factorial=         24 num=          4
# PC=          56 factorial=         24 num=          4
```

Figure 1. Monitor Results