PROJECT REPORT- RISC V

Nimrah Jawed (nj04707) and Umme Salma us04315)

Nimrah Jawed (nj04707)
Umme Salma (us04315)

Contents

Nimrah Jawed (nj04707)
Umme Salma (us04315)

Task 1

We executed task 1 in the following steps:

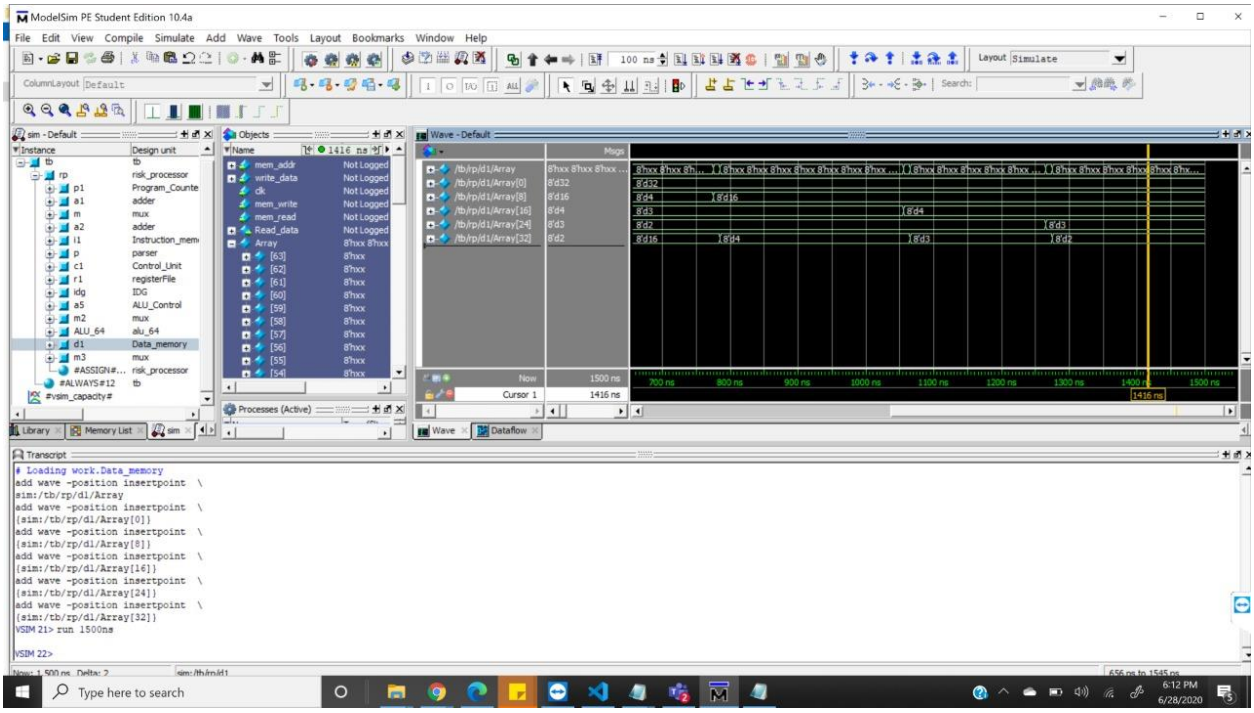
- Edited our bubble sort code such that the shift left happens 3 times. This is because the elements in our data memory are stored as 64 bits in 8 memory locations.
- We removed the memory addresses from load and store instructions and kept a 0 there instead.
- After we dumped the instructions, we changed the func3 of lw and sw to ld and sd's func3.
- We then updated our instruction memory
- We then added bne and slli opcodes and updated our ALU control and ALU_64bit to accomodate these
- To verify out result, we added 5 elements in out data memory in the following sequence: 4,16,3,2,32

The array was sorted correctly in ascending order as shown in the figure below.

This task took us the greatest amount of time. This is because even though the codes were logically written correctly, we missed a begin and end statement while updating the array when the memory write signal is high. This was causing the array to overwrite a few elements as the first location as the memory write was only applicable for the first line. While the regwrite signal was low throughout, this took us a very long time to notice.

Another challenge in this task was that even though our modules were individually checked in labs and tested, we missed an always block in one of the files. This affected our working and it took us the longest time to debug, this is because everytime the waves would run correctly for 600ns, and seemed logical to us. The logic of the control unit and alu_64 was tested individually and that worked also. we tested each files again to verify the result. This is when we noticed that an always block is missing.

Nimrah Jawed (nj04707)
Umme Salma (us04315)



```
module adder(
input [63:0] a,
input [63:0] b,
output reg [63:0] out);

always@(*)
begin
        out = a+b;
end

endmodule
```

```
module mux(
        input[63:0] a, [63:0]b,
        input sel,
        output reg[63:0] data_out
);
always@(*)
begin
case(sel)
        1'b0: assign data_out=a;
        1'b1: assign data_out=b;
endcase
end
endmodule
```

```
module alu_64(
        input [63:0] a,[63:0] b, [3:0] ALUOp,
        output reg [63:0]Result,
        output reg ZERO
);
always @ (*)
begin
  case({ALUOp})
        4'b0111: //slli
        begin
                Result= a<<b;
        end
        4'b0101://bne
        begin
                Result= a-b;
                if(Result==0)
                        begin
                        ZERO=0;
                        end
                else
                        ZERO=1;
        end
        4'b1010:
        begin
                Result = (a<b)?1:0;
                ZERO = Result;
        end
   4'b0000:
        begin
         Result = a&b;
        end
```

```
module ALU_Control(
input [1:0] ALUOp,
input [3:0] Funct,
output reg [3:0] Operation
);
always @(*)
begin
        case(ALUOp)
                2'b00:
                begin
                if (Funct==4'b0001) //slli
                begin
                        Operation=4'b0111;
                end
                else
                begin
                        Operation=4'b0010;
                end
                end

                2'b01:
                begin
                if (Funct==4'b0001)//bne
                begin
                        Operation=4'b0101;
                end
                else if (Funct==4'b0100)//blt
                begin
                        Operation=4'b1010;
                end
                else if (Funct==4'b0000) //beq
```

```
    4'b0001   :
          begin
           Result = a|b;
          end

    4'b0010   :
          begin
           Result = a+b;
          end

          4'b0110://beq
          begin
          Result = a-b;
          ZERO = Result?0:1;
          end

          default  : Result = ~(a|b);
   endcase


end
endmodule
```

```
                    begin
                              Operation=4'b0110;
                    end

                    end

                    2'b10:
                    begin
                    if (Funct==4'b0000)
                    begin
                              Operation=4'b0010;
                    end
                    else if (Funct==4'b1000)
                    begin
                              Operation=4'b0110;
                    end
                    else if (Funct==4'b0111)
                    begin
                              Operation=4'b0000;
                    end
                    else if (Funct==4'b0110)
                    begin
                              Operation=4'b0001;
                    end
                    end
          endcase
end
endmodule
```

```
module Program_Counter(
input clk, reset,
input [63:0] Pc_In,
output reg [63:0] PC_Out
);

always @(posedge clk or posedge reset)
begin
          if (reset)
                    PC_Out<=0;
          else
                    PC_Out<=Pc_In;


end
endmodule
```

```
module parser (
          input [31:0] instruction,
          output reg [6:0] opcode,
          output reg [4:0] rd,
          output reg [2:0] func3,
          output reg [4:0] rs1,
          output reg [4:0] rs2,
          output reg [6:0] func7
);
always@(instruction)
begin
assign opcode=instruction[6:0];
assign rd =instruction[11:7];
assign func3=instruction[14:12];
assign rs1 =instruction[19:15];
assign rs2=instruction[24:20];
assign func7=instruction[31:25];
end
endmodule
```

```
module Instruction_memory(
```

```
module Data_memory(
```

```
input [63:0] Inst_Adress,
output reg [31:0] Instruction
);
reg [7:0]Array[95:0];
initial
   begin
      Array[0] = 8'b00010011;
      Array[1] = 8'b00001011;
      Array[2] = 8'b00000000;
      Array[3] = 8'b00000000;
      Array[4] = 8'b00010011;
      Array[5] = 8'b00000101;
      Array[6] = 8'b01010000;
      Array[7] = 8'b00000000;
      Array[8] = 8'b00010011;
      Array[9] = 8'b00000001;
      Array[10] = 8'b00000000;
      Array[11] = 8'b00000000;
      Array[12] = 8'b10010011;
      Array[13] = 8'b00000001;
      Array[14] = 8'b00000000;
      Array[15] = 8'b00000000;
      Array[16] = 8'b10110011;
      Array[17] = 8'b00001011;
      Array[18] = 8'b01100000;
      Array[19] = 8'b00000001;
      Array[20] = 8'b01100011;
      Array[21] = 8'b00000110;
      Array[22] = 8'b10101011;
      Array[23] = 8'b00000100;
      Array[24] = 8'b00000011;
      Array[25] = 8'b00110011;
      Array[26] = 8'b00000001;
      Array[27] = 8'b00000000;
      Array[28] = 8'b00000011;
      Array[29] = 8'b10110010;
      Array[30] = 8'b00000001;
      Array[31] = 8'b00000000;
      Array[32] = 8'b01100011;
      Array[33] = 8'b01001110;
      Array[34] = 8'b01000011;
      Array[35] = 8'b00000000;
      Array[36] = 8'b10010011;
      Array[37] = 8'b10001011;
      Array[38] = 8'b00011011;
      Array[39] = 8'b00000000;
      Array[40] = 8'b00010011;
      Array[41] = 8'b00010001;
```

```
input [63:0]mem_addr,
input [63:0] write_data,
input clk,
input mem_write,
input mem_read,
output reg[63:0] Read_data
);

reg [7:0] Array [63:0];

initial
begin
Array[0]=8'b00000100;
Array[1]=8'b00000000;
Array[2]=8'b00000000;
Array[3]=8'b00000000;
Array[4]=8'b00000000;
Array[5]=8'b00000000;
Array[6]=8'b00000000;
Array[7]=8'b00000000;

Array[8]=8'b00010000;
Array[9]=8'b00000000;
Array[10]=8'b00000000;
Array[11]=8'b00000000;
Array[12]=8'b00000000;
Array[13]=8'b00000000;
Array[14]=8'b00000000;
Array[15]=8'b00000000;

Array[16]=8'b00000011;
Array[17]=8'b00000000;
Array[18]=8'b00000000;
Array[19]=8'b00000000;
Array[20]=8'b00000000;
Array[21]=8'b00000000;
Array[22]=8'b00000000;
Array[23]=8'b00000000;

Array[24]=8'b00000010;
Array[25]=8'b00000000;
Array[26]=8'b00000000;
Array[27]=8'b00000000;
Array[28]=8'b00000000;
Array[29]=8'b00000000;
Array[30]=8'b00000000;
Array[31]=8'b00000000;
```

```
    Array[42] = 8'b00111011;
    Array[43] = 8'b00000000;
    Array[44] = 8'b10010011;
    Array[45] = 8'b10010001;
    Array[46] = 8'b00111011;
    Array[47] = 8'b00000000;
    Array[48] = 8'b11100011;
    Array[49] = 8'b10010100;
    Array[50] = 8'b10101011;
    Array[51] = 8'b11111110;
    Array[52] = 8'b00010011;
    Array[53] = 8'b00001011;
    Array[54] = 8'b00011011;
    Array[55] = 8'b00000000;
    Array[56] = 8'b11100011;
    Array[57] = 8'b10001100;
    Array[58] = 8'b10101011;
    Array[59] = 8'b11111100;
    Array[60] = 8'b10110011;
    Array[61] = 8'b00000010;
    Array[62] = 8'b01100000;
    Array[63] = 8'b00000000;
    Array[64] = 8'b00100011;
    Array[65] = 8'b00110000;
    Array[66] = 8'b01000001;
    Array[67] = 8'b00000000;
    Array[68] = 8'b00100011;
    Array[69] = 8'b10110000;
    Array[70] = 8'b01010001;
    Array[71] = 8'b00000000;
    Array[72] = 8'b10010011;
    Array[73] = 8'b10001011;
    Array[74] = 8'b00011011;
    Array[75] = 8'b00000000;
    Array[76] = 8'b00010011;
    Array[77] = 8'b00010001;
    Array[78] = 8'b00111011;
    Array[79] = 8'b00000000;
    Array[80] = 8'b10010011;
    Array[81] = 8'b10010001;
    Array[82] = 8'b00111011;
    Array[83] = 8'b00000000;
    Array[84] = 8'b11100011;
    Array[85] = 8'b10010010;
    Array[86] = 8'b10101011;
    Array[87] = 8'b11111100;
    Array[88] = 8'b00010011;
    Array[89] = 8'b00001011;
```

```
    Array[32]=8'b00100000;
    Array[33]=8'b00000000;
    Array[34]=8'b00000000;
    Array[35]=8'b00000000;
    Array[36]=8'b00000000;
    Array[37]=8'b00000000;
    Array[38]=8'b00000000;
    Array[39]=8'b00000000;



end

always @(*)
begin
        if (mem_read)
        begin

        Read_data={Array[mem_addr+7],Array[
mem_addr+6],
Array[mem_addr+5],Array[mem_addr+4],Array
[mem_addr+3],Array[mem_addr+2],
Array[mem_addr+1],Array[mem_addr]};
        end

end

always @(posedge clk)
begin
        if (mem_write)
        begin

        Array[mem_addr]=write_data[7:0];

        Array[mem_addr+1]=write_data[15:8];

        Array[mem_addr+2]=write_data[23:16]
;

        Array[mem_addr+3]=write_data[31:24]
;

        Array[mem_addr+4]=write_data[39:32]
;

        Array[mem_addr+5]=write_data[47:40]
;
```

```
      Array[90] = 8'b00011011;
      Array[91] = 8'b00000000;
      Array[92] = 8'b11100011;
      Array[93] = 8'b10001010;
      Array[94] = 8'b10101011;
      Array[95] = 8'b11111010;
   end
always @ (Inst_Adress)
begin

          Instruction={Array[Inst_Adress+3],Array[Inst_
Adress+2], Array[Inst_Adress+1],Array[Inst_Adress]};
end
endmodule
```

```
          Array[mem_addr+6]=write_data[55:48]
;

          Array[mem_addr+7]=write_data[63:56]
;
          end
end

endmodule
```

```
module risk_processor(
input clk, reset
);
wire [63:0] b;
assign b= 16'h0000000000000004;

wire [63:0] PC_Out;
wire [63:0] adder1_out;
wire [63:0] adder2_out;
wire [63:0] mux1_out;
wire [31:0] Instruction_m;
wire[6:0] opcode;
wire[4:0] rd;
wire[2:0] func3;
wire[4:0] rs1;
wire [4:0] rs2;
wire[6:0] func7;
wire Branch, MemRead, MemtoReg, MemWrite,
ALUSrc, RegWrite;
wire [1:0] Aluop;
wire [63:0] imm_data;
wire [3:0] Operation;
wire [63:0] mux2_out;
wire [63:0] ReadData1;
wire [63:0] ReadData2;
wire [63:0] Result;
wire Zero;
wire[63:0] Read_data;
wire [63:0] mux3_out;

Program_Counter p1(.clk(clk), .reset(reset),
.Pc_In(mux1_out),.PC_Out(PC_Out));
adder a1(.a(PC_Out),.b(b),.out(adder1_out));
```

```
module tb();
reg clk;
reg reset;

risk_processor rp(.clk(clk), .reset(reset));
initial
begin
clk= 1'b0;
reset=1'b1;
#7 reset =1'b0;
end
always

#5 clk=~clk;

endmodule
```

```
mux m(.a(adder1_out),.b(adder2_out),.sel(Branch &
Zero),.data_out(mux1_out));
adder a2(.a(PC_Out),.b(imm_data <<
1),.out(adder2_out));
Instruction_memory
i1(.Inst_Adress(PC_Out),.Instruction(Instruction_m));
parser
p(.instruction(Instruction_m),.opcode(opcode),.rd(rd)
,.func3(func3),.rs1(rs1),.rs2(rs2),.func7(func7));
Control_Unit c1( .Opcode(opcode), .Branch(Branch),
.MemRead(MemRead), .MemtoReg(MemtoReg),
.MemWrite(MemWrite),
.ALUSrc(ALUSrc),.RegWrite(RegWrite),.ALUOp(Aluop))
;
registerFile r1 (.Rs1(rs1), .Rs2(rs2),  .Rd(rd),
.WriteData(mux3_out), .RegWrite(RegWrite),.clk(clk),
.reset(reset),.ReadData1(ReadData1),
.ReadData2(ReadData2));
IDG
idg(.instruction(Instruction_m),.imm_data(imm_data)
);
ALU_Control a5(.ALUOp(Aluop),.Funct({1'b0,
Instruction_m[14:12]}),.Operation(Operation));
mux
m2(.a(ReadData2),.b(imm_data),.sel(ALUSrc),.data_o
ut(mux2_out));
alu_64
ALU_64(.a(ReadData1),.b(mux2_out),.ALUOp(Operati
on),.Result(Result),.ZERO(Zero));
Data_memory d1(.mem_addr
(Result),.write_data(ReadData2),.clk(clk),.mem_write(
MemWrite),.mem_read(MemRead),.Read_data(Read
_data));
mux
m3(.a(Result),.b(Read_data),.sel(MemtoReg),.data_o
ut(mux3_out));
endmodule
```

```
module IDG(
input [31:0] instruction,
output reg [63:0] imm_data
);
always @(*)
begin
        case(instruction[6:5])
                2'b00:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31:20]};
```

```
                end
                2'b01:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31:25],instruction[1
1:7]};

                end
                2'b10:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31],instruction[7],in
struction[30:25],instruction[11:8]};
                end
                2'b11:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31],instruction[7],in
struction[30:25],instruction[11:8]};
                end
        endcase
end
endmodule
```

```
module Control_Unit(
input [6:0] Opcode,
output reg Branch, MemRead, MemtoReg,
MemWrite, ALUSrc, RegWrite,
output reg [1:0] ALUOp
);


always @(*)
begin
        case(Opcode)
                7'b0110011:
                begin
                ALUSrc=1'b0;
                MemtoReg=1'b0;
                RegWrite=1'b1;
                MemRead=1'b0;
                MemWrite=1'b0;
                Branch=1'b0;
                ALUOp=2'b10;
                end

                7'b0000011:
                begin
                ALUSrc=1'b1;
```

```
module registerFile(
input [4:0] Rs1, [4:0] Rs2, [4:0] Rd,
input [63:0] WriteData,
input RegWrite,
input clk, reset,
output reg [63:0] ReadData1, reg [63:0]
ReadData2
);

reg [63:0] Array[31:0];
initial
begin
  Array[0]=64'd0;
  Array[1]=64'd1;
  Array[2]=64'd2;
  Array[3]=64'd3;
  Array[4]=64'd4;
  Array[5]=64'd5;
  Array[6]=64'd6;
  Array[7]=64'd7;
  Array[8]=64'd8;
  Array[9]=64'd9;
  Array[10]=64'd10;
  Array[11]=64'd11;
  Array[12]=64'd12;
```

```
                MemtoReg=1'b1;
                RegWrite=1'b1;
                MemRead=1'b1;
                MemWrite=1'b0;
                Branch=1'b0;
                ALUOp=2'b00;
                end

                7'b0100011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'bx;
                RegWrite=1'b0;
                MemRead=1'b0;
                MemWrite=1'b1;
                Branch=1'b0;
                ALUOp=2'b00;
                end

                7'b1100011:
                begin
                ALUSrc=1'b0;
                MemtoReg=1'bx;
                RegWrite=1'b0;
                MemRead=1'b0;
                MemWrite= 1'b0;
                Branch=1'b1;
                ALUOp=2'b01;
                end

                7'b0010011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'b0;
                RegWrite=1'b1;
                MemRead=1'b1;
                MemWrite= 1'b0;
                Branch=1'b0;
                ALUOp=2'b00;
                end


        endcase
end
endmodule
```

```
  Array[13]=64'd13;
  Array[14]=64'd14;
  Array[15]=64'd15;
  Array[16]=64'd16;
  Array[17]=64'd17;
  Array[18]=64'd18;
  Array[19]=64'd19;
  Array[20]=64'd20;
  Array[21]=64'd21;
  Array[22]=64'd22;
  Array[23]=64'd23;
  Array[24]=64'd24;
  Array[25]=64'd25;
  Array[26]=64'd26;
  Array[27]=64'd27;
  Array[28]=64'd28;
  Array[29]=64'd29;
  Array[30]=64'd30;
  Array[31]=64'd31;
 end

always@(posedge clk)
begin
        if (RegWrite==1)
        begin
                Array[Rd]=WriteData;
        end

end


always @(Rs1, Rs2 , reset , Array , clk)
begin
        if (reset)
        begin
                ReadData1<=64'b0;
                ReadData2<=64'b0;
        end
        else
        begin
                ReadData1<=Array[Rs1];
                ReadData2<=Array[Rs2];
        end
end
endmodule
```

Nimrah Jawed (nj04707)
Umme Salma (us04315)

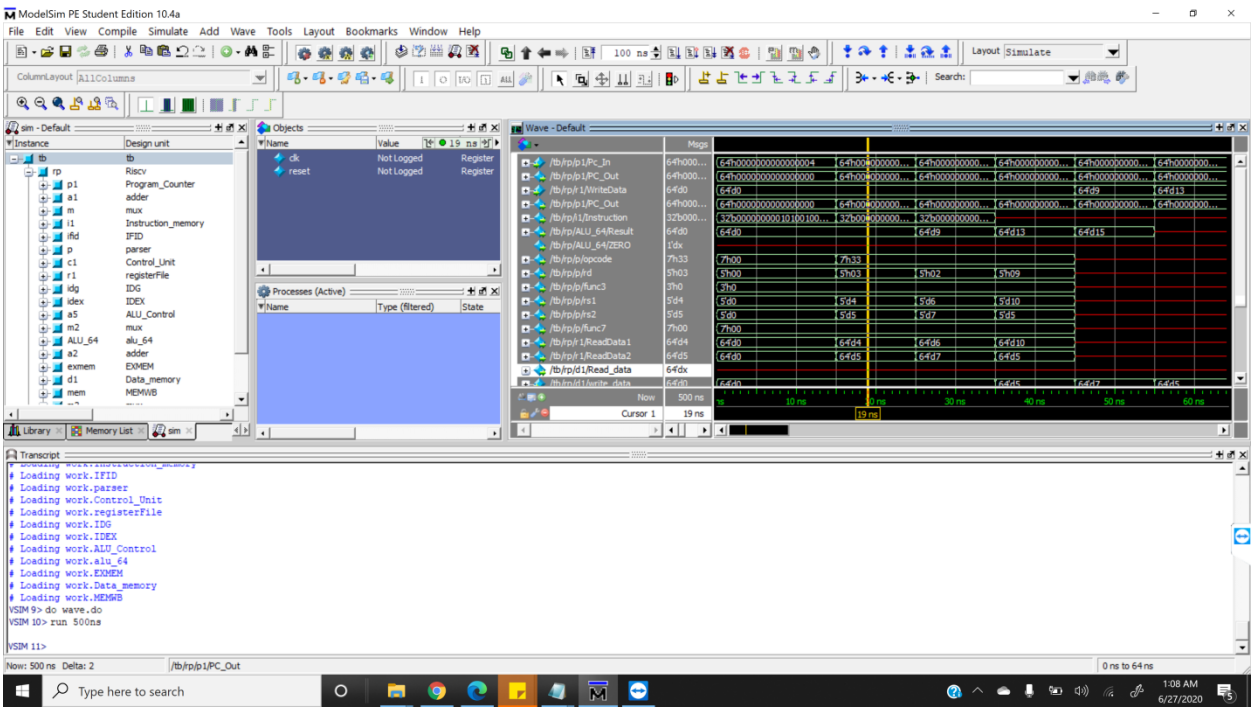
Task 2

To implement the 5 stage pipeline processor, we seperated the datapath in 5 phases in 4 modules. The modules were IF_ID, EX, MEM and WB. Each module had a clock and reset signal which is used to forward the values of each signal of the current clock cycle. This allows overlapped execution of instructions. The values are saved in registers so they retain the value to be carried forward to the next stages. We have ensured that reading takes place in the first half of the clock cycle and writing takes place in the second half of the clock cycle.

We verified this stage by giving independent add instructions which does not require any forwarding.

We gave the following instructions:

1. add x3,x4,x5
2. add x2, x6, x7
3. add x9,x10,x5



module Riscv(input clk, reset	module tb(); reg clk;

```
);
wire [63:0] b;
assign b= 16'h0000000000000004;
wire [63:0] PC_Out;
wire [63:0] PC_Outidex;
wire [63:0] adder1_out;
wire [63:0] adder2_out;
wire [63:0] adder_out_ex;
wire [63:0] mux1_out;
wire [31:0] Instruction_m;
wire [31:0] Instruction_if;
wire[6:0] opcode;
wire[4:0] rd_parser;
wire[4:0] rd_id;
wire[4:0] rd_mem;
wire[4:0] rd_ex;
wire[2:0] func3;
wire[4:0] rs1;
wire [4:0] rs2;
wire[4:0] rs1_id;
wire [4:0] rs2_id;
wire[6:0] func7;
wire Branch, MemRead, MemtoReg, MemWrite,
ALUSrc, RegWrite;
wire [1:0] Aluop;
wire Branch_id, MemRead_id, MemtoReg_id,
MemWrite_id, ALUSrc_id, RegWrite_id;
wire Branch_ex, MemRead_ex, MemtoReg_ex,
MemWrite_ex, RegWrite_ex;
wire MemtoReg_mem, RegWrite_mem;
wire [1:0] Aluop_id;
wire [63:0] imm_data;
wire [63:0] imm_dataOut;
wire [3:0] Operation;
wire [63:0] mux2_out;
wire [63:0] ReadData1;
wire [63:0] ReadData2;
wire[63:0] ReadData1Out;
wire [63:0] ReadData2Out;
wire [63:0] ReadData2_ex;
wire [63:0] Result;
wire [63:0]ALU_Result_out;
wire [63:0] ALU_Resultout_mem;
wire Zero;
wire zero_ex;
wire[63:0] Read_data;
wire [63:0] ReadDataout;
wire [63:0] mux3_out;
```

```
reg reset;

Riscv rp(.clk(clk), .reset(reset));
initial
begin
clk= 1'b0;
reset=1'b1;
#7 reset =1'b0;
end
always

#5 clk=~clk;

endmodule
```

```
wire [63:0] if_pc_out;
wire [3:0] funct;
Program_Counter p1(.clk(clk), .reset(reset),
.Pc_In(mux1_out),.PC_Out(PC_Out));
adder a1(.a(PC_Out),.b(b),.out(adder1_out));
mux
m(.a(adder1_out),.b(adder_out_ex),.sel(Branch_ex
& zero_ex),.data_out(mux1_out));
//adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));
//adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));
Instruction_memory
i1(.Inst_Adress(PC_Out),.Instruction(Instruction_m));
IFID ifid(.PC_in(PC_Out),
.instruction_in(Instruction_m),
.clk(clk),
.reset(reset),
.PC_out(if_pc_out),
.Instruction_out(Instruction_if));
Parser
p(.instruction(Instruction_if),.opcode(opcode),.rd(rd
_parser),.func3(func3),.rs1(rs1),.rs2(rs2),.func7(func
7));
Control_Unit c1( .Opcode(opcode), .Branch(Branch),
.MemRead(MemRead), .MemtoReg(MemtoReg),
.MemWrite(MemWrite),.ALUSrc(ALUSrc),.RegWrite(
RegWrite),.ALUOp(Aluop));
registerFile r1 (.Rs1(rs1), .Rs2(rs2),  .Rd(rd_mem),
.WriteData(mux3_out),
.RegWrite(RegWrite_mem),.clk(clk),
.reset(reset),.ReadData1(ReadData1),
.ReadData2(ReadData2));
IDG
idg(.instruction(Instruction_if),.imm_data(imm_data
));
IDEX idex(.clk(clk),
.reset(reset),.PC_inidex(if_pc_out),
.ReadData1In(ReadData1),.ReadData2In(ReadData2)
,
.imm_data(imm_data),.rs1(rs1),.rs2(rs2),.rd(rd_pars
er),
.inst({Instruction_if[30],
Instruction_if[14:12]}),.MemtoReg(MemtoReg),
.RegWrite(RegWrite),.branch(Branch),
.MemRead(MemRead),.MemWrite(MemWrite),
.ALUSrc(ALUSrc),.ALUOp(Aluop),
.PC_Outidex(PC_Outidex),
```

```
.ReadData1Out(ReadData1Out),
.ReadData2Out(ReadData2Out),
.imm_dataOut(imm_dataOut),
.funct(funct),.rdOut(rd_id),
.rs1Out(rs1_id),
.rs2Out(rs2_id),
.MemtoRegOut(MemtoReg_id),
.RegWriteOut(RegWrite_id),
.branchOut(Branch_id),
.MemReadOut(MemRead_id),
.MemWriteOut(MemWrite_id),
.ALUSrcOut(ALUSrc_id),
.ALUOpOut(Aluop_id)
);
ALU_Control
a5(.ALUOp(Aluop_id),.Funct(funct),.Operation(Operatio));
mux
m2(.a(ReadData2Out),.b(imm_dataOut),.sel(ALUSrc_id),.data_out(mux2_out));
alu_64
ALU_64(.a(ReadData1Out),.b(mux2_out),.ALUOp(Operation),.Result(Result),.ZERO(Zero));

adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));
EXMEM exmem(.clk(clk),
.reset(reset),
.adder_in(adder2_out),
.ZERO_in(Zero),
.ALU_Result_in(Result),
.ReadData2In(ReadData2Out),
.rd(rd_id),.MemtoReg(MemtoReg_id),
.RegWrite(RegWrite_id),
.branch(Branch_id),.MemRead(MemRead_id),.MemWrite(MemWrite_id),

.adder_out(adder_out_ex),.ALU_Result_out(ALU_Result_out), .ReadData2out(ReadData2_ex),
.rdOut(rd_ex),
.zero(zero_ex),.MemtoRegOut(MemtoReg_ex),.RegWriteOut(RegWrite_ex),.branchOut(Branch_ex),
.MemReadOut(MemRead_ex),.MemWriteOut(MemWrite_ex));
Data_memory d1(.mem_addr
(ALU_Result_out),.write_data(ReadData2_ex),.clk(clk),.mem_write(MemWrite_ex),.mem_read(MemRead_ex),.Read_data(Read_data));
```

```
MEMWB mem(.clk(clk),.reset(reset),
.ReadDatain(Read_data),
.ALU_Resultin(ALU_Result_out),
.MemtoReg(MemtoReg_ex),
.RegWrite(RegWrite_ex),
.rd(rd_ex),

.ReadDataout(ReadDataout),
.ALU_Resultout(ALU_Resultout_mem),
.rdOut(rd_mem),.MemtoRegOut(MemtoReg_mem),
.RegWriteOut(RegWrite_mem));
mux
m3(.a(ALU_Resultout_mem),.b(ReadDataout),.sel(M
emtoReg_mem),.data_out(mux3_out));
endmodule
```

```
module adder(
input [63:0] a,
input [63:0] b,
output reg [63:0] out);

always@(*)
begin
        out = a+b;
end

endmodule
```

```
module mux(
        input[63:0] a, [63:0]b,
        input sel,
        output reg[63:0] data_out
);
always@(*)
begin
case(sel)
        1'b0: assign data_out=a;
        1'b1: assign data_out=b;
endcase
end
endmodule
```

```
module alu_64(
        input [63:0] a,[63:0] b, [3:0] ALUOp,
        output reg [63:0]Result,
        output reg ZERO

);
always @ (*)
begin
  case({ALUOp})

        4'b0111: //slli
        begin
                Result= a<<b;

        end
        4'b0101://bne
        begin
                Result= a-b;
                if(Result==0)
                        begin
```

```
module ALU_Control(
input [1:0] ALUOp,
input [3:0] Funct,
output reg [3:0] Operation
);

always @(*)
begin
        case(ALUOp)
                2'b00:
                begin
                if (Funct==4'b0001) //slli
                begin
                        Operation=4'b0111;
                end
                else
                begin
                        Operation=4'b0010;
                end
                end
```

```
                        ZERO=0;
                        end
                else
                        ZERO=1;
        end
        4'b1010:
        begin
                Result = (a<b)?1:0;
                ZERO = Result;
        end

    4'b0000:
        begin
          Result = a&b;
        end
    4'b0001    :
        begin
          Result = a|b;
        end

    4'b0010    :
        begin
          Result = a+b;
        end

        4'b0110://beq
        begin
        Result = a-b;
        ZERO = Result?0:1;
        end

        default  : Result = ~(a|b);
   endcase


end
endmodule
```

```
                2'b01:
                begin
                if (Funct==4'b0001)//bne
                begin
                        Operation=4'b0101;
                end
                else if (Funct==4'b0100)//blt
                begin
                        Operation=4'b1010;
                end
                else if (Funct==4'b0000) //beq
                begin
                        Operation=4'b0110;
                end

                end

                2'b10:
                begin
                if (Funct==4'b0000)
                begin
                        Operation=4'b0010;
                end
                else if (Funct==4'b1000)
                begin
                        Operation=4'b0110;
                end
                else if (Funct==4'b0111)
                begin
                        Operation=4'b0000;
                end
                else if (Funct==4'b0110)
                begin
                        Operation=4'b0001;
                end
                end
          endcase

end
endmodule
```

```
module Control_Unit(
input [6:0] Opcode,
output reg Branch, MemRead, MemtoReg,
MemWrite, ALUSrc, RegWrite,
output reg [1:0] ALUOp
);
```

```
module registerFile(
input [4:0] Rs1, [4:0] Rs2, [4:0] Rd,
input [63:0] WriteData,
input RegWrite,
input clk, reset,
output reg [63:0] ReadData1, reg [63:0]
ReadData2
```

```
always @(*)
begin
        case(Opcode)
                7'b0110011:
                begin
                ALUSrc=1'b0;
                MemtoReg=1'b0;
                RegWrite=1'b1;
                MemRead=1'b0;
                MemWrite=1'b0;
                Branch=1'b0;
                ALUOp=2'b10;
                end

                7'b0000011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'b1;
                RegWrite=1'b1;
                MemRead=1'b1;
                MemWrite=1'b0;
                Branch=1'b0;
                ALUOp=2'b00;
                end

                7'b0100011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'bx;
                RegWrite=1'b0;
                MemRead=1'b0;
                MemWrite=1'b1;
                Branch=1'b0;
                ALUOp=2'b00;
                end

                7'b1100011:
                begin
                ALUSrc=1'b0;
                MemtoReg=1'bx;
                RegWrite=1'b0;
                MemRead=1'b0;
                MemWrite= 1'b0;
                Branch=1'b1;
                ALUOp=2'b01;
                end
```

```
);

reg [63:0] Array[31:0];
initial
begin
  Array[0]=64'd0;
  Array[1]=64'd1;
  Array[2]=64'd2;
  Array[3]=64'd3;
  Array[4]=64'd4;
  Array[5]=64'd5;
  Array[6]=64'd6;
  Array[7]=64'd7;
  Array[8]=64'd8;
  Array[9]=64'd9;
  Array[10]=64'd10;
  Array[11]=64'd11;
  Array[12]=64'd12;
  Array[13]=64'd13;
  Array[14]=64'd14;
  Array[15]=64'd15;
  Array[16]=64'd16;
  Array[17]=64'd17;
  Array[18]=64'd18;
  Array[19]=64'd19;
  Array[20]=64'd20;
  Array[21]=64'd21;
  Array[22]=64'd22;
  Array[23]=64'd23;
  Array[24]=64'd24;
  Array[25]=64'd25;
  Array[26]=64'd26;
  Array[27]=64'd27;
  Array[28]=64'd28;
  Array[29]=64'd29;
  Array[30]=64'd30;
  Array[31]=64'd31;
 end

always@(posedge clk)
begin
        if (RegWrite==1)
        begin
                Array[Rd]=WriteData;
        end

end
```

```
                7'b0010011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'b0;
                RegWrite=1'b1;
                MemRead=1'b1;
                MemWrite= 1'b0;
                Branch=1'b0;
                ALUOp=2'b00;
                end



        endcase
end
endmodule
```

```

always @(Rs1, Rs2 , reset , Array , clk)
begin
        if (reset)
        begin
                ReadData1<=64'b0;
                ReadData2<=64'b0;
        end
        else
        begin
                ReadData1<=Array[Rs1];
                ReadData2<=Array[Rs2];
        end
end
endmodule
```

```
module Data_memory(
input [63:0]mem_addr,
input [63:0] write_data,
input clk,
input mem_write,
input mem_read,
output reg[63:0] Read_data
);
reg [7:0] Array [63:0];
initial
begin
Array[0]=8'b00000100;
Array[1]=8'b00000000;
Array[2]=8'b00000000;
Array[3]=8'b00000000;
Array[4]=8'b00000000;
Array[5]=8'b00000000;
Array[6]=8'b00000000;
Array[7]=8'b00000000;
Array[8]=8'b00010000;
Array[9]=8'b00000000;
Array[10]=8'b00000000;
Array[11]=8'b00000000;
Array[12]=8'b00000000;
Array[13]=8'b00000000;
Array[14]=8'b00000000;
Array[15]=8'b00000000;
Array[16]=8'b00000011;
Array[17]=8'b00000000;
Array[18]=8'b00000000;
Array[19]=8'b00000000;
Array[20]=8'b00000000;
Array[21]=8'b00000000;
```

```
module Instruction_memory(
input [63:0] Inst_Adress,
output reg [31:0] Instruction
);

reg [7:0]Array[11:0];
 initial
   begin


      Array[0] =8'b10110011;
      Array[1] =8'b00000001;
      Array[2] =8'b01010010;
      Array[3] =8'b00000000;

      Array[4] =8'b00110011;
      Array[5] =8'b00000001;
      Array[6] =8'b01110011;
      Array[7] =8'b00000000;

      Array[8] =8'b10110011;
      Array[9] =8'b00000100;
      Array[10] =8'b01010101;
      Array[11] =8'b00000000;

   end



always @ (Inst_Adress)
begin
```

```
Array[22]=8'b00000000;
Array[23]=8'b00000000;
Array[24]=8'b00000010;
Array[25]=8'b00000000;
Array[26]=8'b00000000;
Array[27]=8'b00000000;
Array[28]=8'b00000000;
Array[29]=8'b00000000;
Array[30]=8'b00000000;
Array[31]=8'b00000000;
Array[32]=8'b00100000;
Array[33]=8'b00000000;
Array[34]=8'b00000000;
Array[35]=8'b00000000;
Array[36]=8'b00000000;
Array[37]=8'b00000000;
Array[38]=8'b00000000;
Array[39]=8'b00000000;
end
always @(*)
begin
        if (mem_read)
        begin
        Read_data={Array[mem_addr+7],Array[me
m_addr+6],
Array[mem_addr+5],Array[mem_addr+4],Array[me
m_addr+3],Array[mem_addr+2],
Array[mem_addr+1],Array[mem_addr]};
        end
end
always @(posedge clk)
begin
        if (mem_write)
        begin
                Array[mem_addr]=write_data[7:0];

        Array[mem_addr+1]=write_data[15:8];

        Array[mem_addr+2]=write_data[23:16];

        Array[mem_addr+3]=write_data[31:24];

        Array[mem_addr+4]=write_data[39:32];

        Array[mem_addr+5]=write_data[47:40];

        Array[mem_addr+6]=write_data[55:48];
```

```
        Instruction={Array[Inst_Adress+3],Array[
Inst_Adress+2],
Array[Inst_Adress+1],Array[Inst_Adress]};
end
endmodule
```

```
        Array[mem_addr+7]=write_data[63:56];
        end
end
endmodule
```

```
module EXMEM(
input clk,
input reset,
input [63:0] adder_in,
input ZERO_in,
input [63:0] ALU_Result_in,
input [63:0] ReadData2In,
input [4:0] rd,
input MemtoReg,RegWrite, branch,
MemRead,MemWrite,
output reg [63:0] adder_out,
output reg [63:0] ALU_Result_out,
output reg [63:0] ReadData2out,
output reg [4:0] rdOut,
output reg zero,
output reg MemtoRegOut, RegWriteOut, branchOut,
MemReadOut, MemWriteOut
);

always @(posedge clk)
begin
   if (reset==1'b0)
   begin
   adder_out=  adder_in;
   ALU_Result_out= ALU_Result_in;
   ReadData2out= ReadData2In;
   rdOut=rd;
   zero=ZERO_in;
   MemtoRegOut=MemtoReg;
   RegWriteOut=RegWrite;
   branchOut=branch;
   MemReadOut= MemRead;
   MemWriteOut=MemWrite;

   end
end
always@(reset)
begin
   if (reset==1'b1)
   begin
   adder_out=  64'b0;
   ALU_Result_out= 64'b0;
   ReadData2out= 64'b0;
```

```
module IDEX(
input clk,
input reset,
input [63:0] PC_inidex,
input [63:0] ReadData1In,
input [63:0] ReadData2In,
input [63:0] imm_data,
input [4:0] rs1,
input [4:0] rs2,
input [4:0] rd,
input [3:0] inst,
input MemtoReg,
input RegWrite,
input branch,
input MemRead,
input MemWrite,
input ALUSrc,
input [1:0] ALUOp,
output reg [63:0] PC_Outidex,
output reg [63:0] ReadData1Out,
output reg [63:0] ReadData2Out,
output reg [63:0] imm_dataOut,
output reg [3:0] funct,
output reg [4:0] rdOut,
output reg [4:0] rs1Out,
output reg [4:0] rs2Out,
output reg MemtoRegOut,
RegWriteOut,branchOut,MemReadOut,MemWri
teOut,ALUSrcOut,
output reg [1:0] ALUOpOut
);

always @(posedge clk)
begin
        if (reset==1'b0)
                begin
                        PC_Outidex=PC_inidex;

        ReadData1Out=ReadData1In;

        ReadData2Out=ReadData2In;

        imm_dataOut=imm_data;
```

```
   rdOut=5'b0;
   zero=1'b0;
   MemtoRegOut=1'b0;
   RegWriteOut=1'b0;
   branchOut=1'b0;
   MemReadOut= 1'b0;
   MemWriteOut=1'b0;
   end
end
endmodule
```

```
                        funct=inst;
                        rdOut=rd;
                        rs1Out=rs1;
                        rs2Out=rs2;

        MemtoRegOut=MemtoReg;
                        RegWriteOut=RegWrite;
                        branchOut=branch;
                        MemReadOut=
MemRead;

        MemWriteOut=MemWrite;
                        ALUSrcOut=ALUSrc;
                        ALUOpOut=ALUOp;
                end
end
always@(reset)
begin
        if(reset==1'b1)
                begin
                        PC_Outidex=64'b0;
                        ReadData1Out=64'b0;
                        ReadData2Out=64'b0;
                        imm_dataOut=64'b0;
                        funct=4'b0;
                        rdOut=5'b0;
                        rs1Out=5'b0;
                        rs2Out= 5'b0;
                        MemtoRegOut=1'b0;
                        RegWriteOut=1'b0;
                        branchOut=1'b0;
                        MemReadOut= 1'b0;
                        MemWriteOut=1'b0;
                        ALUSrcOut=1'b0;
                        ALUOpOut=2'b0;



                end
end
endmodule
```

```
module IDG(
input [31:0] instruction,
output reg [63:0] imm_data
);
always @(*)
begin
        case(instruction[6:5])
                2'b00:
```

```
module IFID(
input [63:0] PC_in,
input [31:0] instruction_in,
input clk,
input reset,
output reg [63:0] PC_out,
output reg [31:0] Instruction_out
);
```

```
                    begin
                    imm_data=
{{52{instruction[31]}},instruction[31:20]};
                    end
                    2'b01:
                    begin
                    imm_data=
{{52{instruction[31]}},instruction[31:25],instruction[
11:7]};

                    end
                    2'b10:
                    begin
                    imm_data=
{{52{instruction[31]}},instruction[31],instruction[7],i
nstruction[30:25],instruction[11:8]};
                    end
                    2'b11:
                    begin
                    imm_data=
{{52{instruction[31]}},instruction[31],instruction[7],i
nstruction[30:25],instruction[11:8]};
                    end
          endcase
end
endmodule
```

```
always@(posedge clk)
begin
        if (reset==1'b0)
                begin
                PC_out=PC_in;
                Instruction_out=instruction_in;
                end
end
always@(reset)
begin
        if (reset==1'b1)
                begin
                PC_out=64'b0;
                Instruction_out=32'b0;
                end
end

endmodule
```

```
module MEMWB(
  input clk,
  input reset,
  input [63:0] ReadDatain,
  input [63:0] ALU_Resultin,
  input MemtoReg, RegWrite,
  input [4:0] rd,
  output reg [63:0] ReadDataout,
  output reg [63:0] ALU_Resultout,
  output reg [4:0] rdOut,
  output reg MemtoRegOut, RegWriteOut
);

always @(posedge clk)
begin
  if (reset==1'b0)
  begin
  ReadDataout=ReadDatain;
  ALU_Resultout=ALU_Resultin;
  rdOut=rd;
  MemtoRegOut=MemtoReg;
  RegWriteOut=RegWrite;
```

```
module Program_Counter(
input clk, reset,
input [63:0] Pc_In,
output reg [63:0] PC_Out
);

always @(posedge clk or posedge reset)
begin
        if (reset)
                PC_Out<=0;
        else
                PC_Out<=Pc_In;


end
endmodule
```

```
   end
end
always@(reset)
begin
   if (reset==1'b1)
   begin
   ReadDataout=64'b0;
   ALU_Resultout=64'b0;
   rdOut=5'b0;
   MemtoRegOut=1'b0;
   RegWriteOut=1'b0;

   end
end
endmodule
```

```
module parser (
          input [31:0] instruction,
          output reg [6:0] opcode,
          output reg [4:0] rd,
          output reg [2:0] func3,
          output reg [4:0] rs1,
          output reg [4:0] rs2,
          output reg [6:0] func7

);
always@(instruction)
begin
assign opcode=instruction[6:0];
assign rd =instruction[11:7];
assign func3=instruction[14:12];
assign rs1 =instruction[19:15];
assign rs2=instruction[24:20];
assign func7=instruction[31:25];
end

endmodule
```

Nimrah Jawed (nj04707)
Umme Salma (us04315)

Task 3

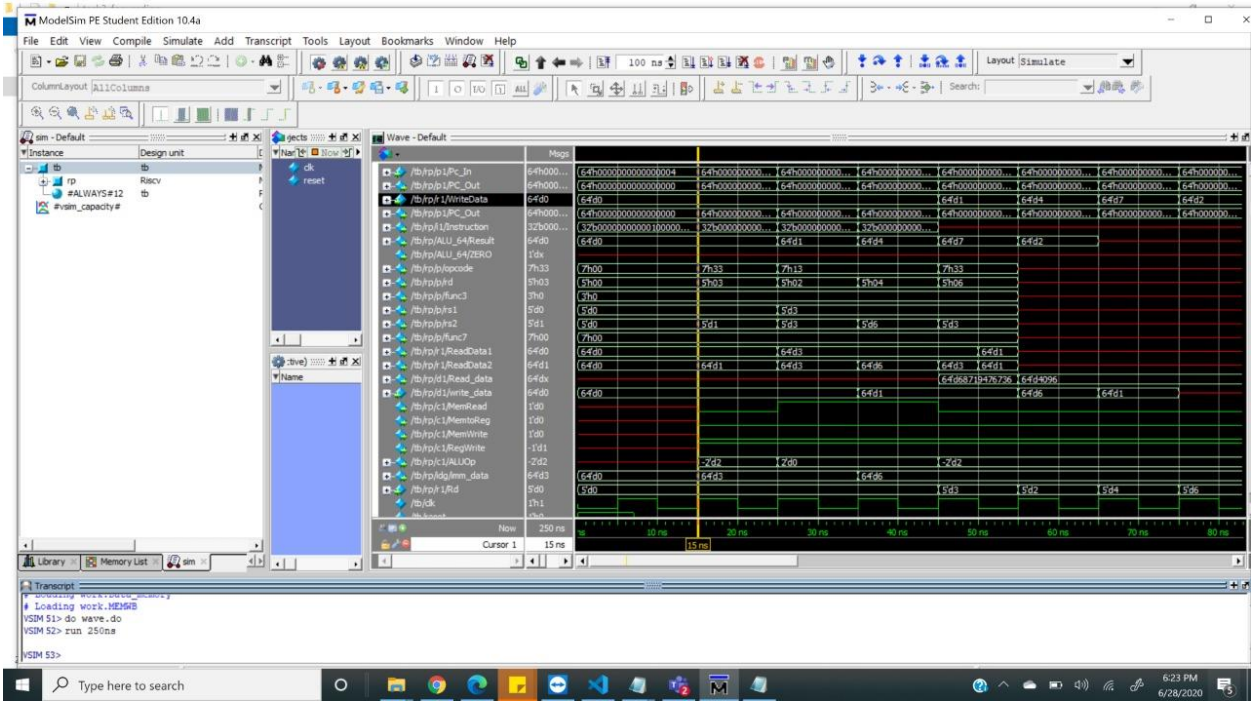
Forwarding:

To implement circuitry that accomodates dependant instructions, we have further implemented Forwarding. We have done this by adding a forwarding unit module between the IDEX and EXMEM stage.

To verify we have run the following instructions. The Destination register of the first instruction is passed as the source register of the next ones. The final values must be 1 4 7 2 and it was , as expected.

1. Add x3, x0, x1 (where x1 has a value of 1)
2. Addi x2, x3, 3
3. Addi x4, x3, 6
4. Add x6, x3, x3

A challenge we encounted in this task was that after every two instructions, the 3rd instruction produced a wrong result. This is when we realized that we should alter the register file in such a way that it reads on half a clock cycle and writes on the next half.



```
module adder(
input [63:0] a,
input [63:0] b,
output reg [63:0] out);

always@(*)
begin
        out = a+b;
end

endmodule
```

```
module mux(
        input[63:0] a, [63:0]b,
        input sel,
        output reg[63:0] data_out
);
always@(*)
begin
case(sel)
        1'b0: assign data_out=a;
        1'b1: assign data_out=b;
endcase
end
endmodule
```

```
module alu_64(
        input [63:0] a,[63:0] b, [3:0] ALUOp,
        output reg [63:0]Result,
        output reg ZERO

);
always @ (*)
begin
  case({ALUOp})

        4'b0111: //slli
        begin
                Result= a<<b;

        end
        4'b0101://bne
        begin
                Result= a-b;
                if(Result==0)
                        begin
                        ZERO=0;
                        end
                else
                        ZERO=1;
        end
        4'b1010:
        begin
                Result = (a<b)?1:0;
                ZERO = Result;
        end

   4'b0000:
        begin
         Result = a&b;
        end
```

```
module ALU_Control(
input [1:0] ALUOp,
input [3:0] Funct,
output reg [3:0] Operation
);

always @(*)
begin
        case(ALUOp)
                2'b00:
                begin
                if (Funct==4'b0001) //slli
                begin
                        Operation=4'b0111;
                end
                else
                begin
                        Operation=4'b0010;
                end
                end

                2'b01:
                begin
                if (Funct==4'b0001)//bne
                begin
                        Operation=4'b0101;
                end
                else if (Funct==4'b0100)//blt
                begin
                        Operation=4'b1010;
                end
                else if (Funct==4'b0000) //beq
                begin
                        Operation=4'b0110;
                end
```

```
    4'b0001    :
          begin
            Result = a|b;
          end

    4'b0010    :
          begin
            Result = a+b;
          end

          4'b0110://beq
          begin
          Result = a-b;
          ZERO = Result?0:1;
          end

          default  : Result = ~(a|b);
   endcase


end
endmodule
```

```
                    end

                    2'b10:
                    begin
                    if (Funct==4'b0000)
                    begin
                              Operation=4'b0010;
                    end
                    else if (Funct==4'b1000)
                    begin
                              Operation=4'b0110;
                    end
                    else if (Funct==4'b0111)
                    begin
                              Operation=4'b0000;
                    end
                    else if (Funct==4'b0110)
                    begin
                              Operation=4'b0001;
                    end
                    end
          endcase

end
endmodule
```

```
module Control_Unit(
input [6:0] Opcode,
output reg Branch, MemRead, MemtoReg,
MemWrite, ALUSrc, RegWrite,
output reg [1:0] ALUOp
);


always @(*)
begin
          case(Opcode)
                    7'b0110011:
                    begin
                    ALUSrc=1'b0;
                    MemtoReg=1'b0;
                    RegWrite=1'b1;
                    MemRead=1'b0;
                    MemWrite=1'b0;
                    Branch=1'b0;
                    ALUOp=2'b10;
                    end
```

```
module Data_memory(
input [63:0]mem_addr,
input [63:0] write_data,
input clk,
input mem_write,
input mem_read,
output reg[63:0] Read_data
);

reg [7:0] Array [63:0];

initial
begin
Array[0]=8'b00000100;
Array[1]=8'b00000000;
Array[2]=8'b00000000;
Array[3]=8'b00000000;
Array[4]=8'b00000000;
Array[5]=8'b00000000;
Array[6]=8'b00000000;
Array[7]=8'b00000000;
Array[8]=8'b00010000;
```

```
                7'b0000011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'b1;
                RegWrite=1'b1;
                MemRead=1'b1;
                MemWrite=1'b0;
                Branch=1'b0;
                ALUOp=2'b00;
                end

                7'b0100011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'bx;
                RegWrite=1'b0;
                MemRead=1'b0;
                MemWrite=1'b1;
                Branch=1'b0;
                ALUOp=2'b00;
                end

                7'b1100011:
                begin
                ALUSrc=1'b0;
                MemtoReg=1'bx;
                RegWrite=1'b0;
                MemRead=1'b0;
                MemWrite= 1'b0;
                Branch=1'b1;
                ALUOp=2'b01;
                end

                7'b0010011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'b0;
                RegWrite=1'b1;
                MemRead=1'b1;
                MemWrite= 1'b0;
                Branch=1'b0;
                ALUOp=2'b00;
                end


        endcase
end
endmodule
```

```
Array[9]=8'b00000000;
Array[10]=8'b00000000;
Array[11]=8'b00000000;
Array[12]=8'b00000000;
Array[13]=8'b00000000;
Array[14]=8'b00000000;
Array[15]=8'b00000000;
Array[16]=8'b00000011;
Array[17]=8'b00000000;
Array[18]=8'b00000000;
Array[19]=8'b00000000;
Array[20]=8'b00000000;
Array[21]=8'b00000000;
Array[22]=8'b00000000;
Array[23]=8'b00000000;
Array[24]=8'b00000010;
Array[25]=8'b00000000;
Array[26]=8'b00000000;
Array[27]=8'b00000000;
Array[28]=8'b00000000;
Array[29]=8'b00000000;
Array[30]=8'b00000000;
Array[31]=8'b00000000;
Array[32]=8'b00100000;
Array[33]=8'b00000000;
Array[34]=8'b00000000;
Array[35]=8'b00000000;
Array[36]=8'b00000000;
Array[37]=8'b00000000;
Array[38]=8'b00000000;
Array[39]=8'b00000000;
end
always @(*)
begin
        if (mem_read)
        begin

        Read_data={Array[mem_addr+7],Array[mem_
addr+6],
Array[mem_addr+5],Array[mem_addr+4],Array[mem_
addr+3],Array[mem_addr+2],
Array[mem_addr+1],Array[mem_addr]};
        end
end
always @(posedge clk)
begin
        if (mem_write)
        begin
```

```
                                    Array[mem_addr]=write_data[7:0];

                Array[mem_addr+1]=write_data[15:8];

                Array[mem_addr+2]=write_data[23:16];

                Array[mem_addr+3]=write_data[31:24];

                Array[mem_addr+4]=write_data[39:32];

                Array[mem_addr+5]=write_data[47:40];

                Array[mem_addr+6]=write_data[55:48];

                Array[mem_addr+7]=write_data[63:56];
                end
end
endmodule
```

```
module EXMEM(
input clk,
input reset,
input [63:0] adder_in,
input ZERO_in,
input [63:0] ALU_Result_in,
input [63:0] ReadData2In,
input [4:0] rd,
input MemtoReg,RegWrite, branch,
MemRead,MemWrite,
output reg [63:0] adder_out,
output reg [63:0] ALU_Result_out,
output reg [63:0] ReadData2out,
output reg [4:0] rdOut,
output reg zero,
output reg MemtoRegOut, RegWriteOut,
branchOut, MemReadOut, MemWriteOut
);

always @(posedge clk)
begin
   if (reset==1'b0)
   begin
   adder_out=  adder_in;
   ALU_Result_out= ALU_Result_in;
   ReadData2out= ReadData2In;
   rdOut=rd;
   zero=ZERO_in;
   MemtoRegOut=MemtoReg;
   RegWriteOut=RegWrite;
```

```
module forwarding_unit(

   input[4:0] rs1_idex,
   input [4:0] rs2_idex,
   input regwrite_memwb,
   input regwrite_exmem,
   input [4:0] rd_memwb,
   input [4:0] rd_exmem,

   output reg [1:0] Forward_A,
   output reg [1:0] Forward_B

);



always @(*)
begin

if ((regwrite_exmem==1'b1) && (rd_exmem!=1'b0)&&
(rd_exmem==rs1_idex))
   Forward_A=2'b10;
else if ((regwrite_memwb==1'b1)&&
(rd_memwb!=1'b0)&& (rd_memwb==rs1_idex))
   Forward_A=2'b01;
else
          Forward_A=2'b00;

if ((regwrite_exmem==1'b1)&& (rd_exmem!=1'b0)&&
(rd_exmem==rs2_idex))
```

```
   branchOut=branch;
   MemReadOut= MemRead;
   MemWriteOut=MemWrite;
   end
end
always@(reset)
begin
   if (reset==1'b1)
   begin
   adder_out=  64'b0;
   ALU_Result_out= 64'b0;
   ReadData2out= 64'b0;
   rdOut=5'b0;
   zero=1'b0;
   MemtoRegOut=1'b0;
   RegWriteOut=1'b0;
   branchOut=1'b0;
   MemReadOut= 1'b0;
   MemWriteOut=1'b0;
   end
end
endmodule
```

```
   Forward_B=2'b10;

else if ((regwrite_memwb==1'b1)&&
(rd_memwb!=1'b0)&& (rd_memwb==rs2_idex))
   Forward_B=2'b01;
else
          Forward_B=2'b00;



end
endmodule
```

```
module IDEX(
input clk,
input reset,
input [63:0] PC_inidex,
input [63:0] ReadData1In,
input [63:0] ReadData2In,
input [63:0] imm_data,
input [4:0] rs1,
input [4:0] rs2,
input [4:0] rd,
input [3:0] inst,
input MemtoReg,
input RegWrite,
input branch,
input MemRead,
input MemWrite,
input ALUSrc,
input [1:0] ALUOp,
output reg [63:0] PC_Outidex,
output reg [63:0] ReadData1Out,
output reg [63:0] ReadData2Out,
output reg [63:0] imm_dataOut,
output reg [3:0] funct,
output reg [4:0] rdOut,
output reg [4:0] rs1Out,
output reg [4:0] rs2Out,
```

```
module Instruction_memory(
input [63:0] Inst_Adress,
output reg [31:0] Instruction
);

reg [7:0]Array[15:0];
 initial
   begin


      Array[0] =8'b10110011;
      Array[1] =8'b00000001;
      Array[2] =8'b00010000;
      Array[3] =8'b00000000;

      Array[4] =8'b00010011;
      Array[5] =8'b10000001;
      Array[6] =8'b00110001;
      Array[7] =8'b00000000;

      Array[8] =8'b00010011;
      Array[9] =8'b10000010;
      Array[10] =8'b01100001;
      Array[11] =8'b00000000;

          Array[12] =8'b00110011;
```

```
output reg MemtoRegOut,
RegWriteOut,branchOut,MemReadOut,Mem
WriteOut,ALUSrcOut,
output reg [1:0] ALUOpOut
);

always @(posedge clk)
begin
        if (reset==1'b0)
                begin

        PC_Outidex=PC_inidex;

        ReadData1Out=ReadData1In;

        ReadData2Out=ReadData2In;

        imm_dataOut=imm_data;
                        funct=inst;
                        rdOut=rd;
                        rs1Out=rs1;
                        rs2Out=rs2;

        MemtoRegOut=MemtoReg;

        RegWriteOut=RegWrite;
                        branchOut=branch;
                        MemReadOut=
MemRead;

        MemWriteOut=MemWrite;
                        ALUSrcOut=ALUSrc;
                        ALUOpOut=ALUOp;
                end
end
always@(reset)
begin
        if(reset==1'b1)
                begin
                        PC_Outidex=64'b0;
                        ReadData1Out=64'b0;
                        ReadData2Out=64'b0;
                        imm_dataOut=64'b0;
                        funct=4'b0;
                        rdOut=5'b0;
                        rs1Out=5'b0;
                        rs2Out= 5'b0;
                        MemtoRegOut=1'b0;
        Array[13] =8'b10000011;
        Array[14] =8'b00110001;
        Array[15] =8'b00000000;


   end



always @ (Inst_Adress)
begin

        Instruction={Array[Inst_Adress+3],Array[Inst_A
dress+2], Array[Inst_Adress+1],Array[Inst_Adress]};
end
endmodule
```

```
                        RegWriteOut=1'b0;
                        branchOut=1'b0;
                        MemReadOut= 1'b0;
                        MemWriteOut=1'b0;
                        ALUSrcOut=1'b0;
                        ALUOpOut=2'b0;



                end
end
endmodule
```

```
module IDG(
input [31:0] instruction,
output reg [63:0] imm_data
);
always @(*)
begin
        case(instruction[6:5])
                2'b00:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31:20]};
                end
                2'b01:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31:25],instru
ction[11:7]};

                end
                2'b10:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31],instructio
n[7],instruction[30:25],instruction[11:8]};
                end
                2'b11:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31],instructio
n[7],instruction[30:25],instruction[11:8]};
                end
        endcase
end
endmodule
```

```
module IFID(
input [63:0] PC_in,
input [31:0] instruction_in,
input clk,
input reset,
output reg [63:0] PC_out,
output reg [31:0] Instruction_out
);

always@(posedge clk)
begin
        if (reset==1'b0)
                begin
                PC_out=PC_in;
                Instruction_out=instruction_in;
                end
end
always@(reset)
begin
        if (reset==1'b1)
                begin
                PC_out=64'b0;
                Instruction_out=32'b0;
                end
end

endmodule
```

```
module MEMWB(
  input clk,
  input reset,
```

```
   input [63:0] ReadDatain,
   input [63:0] ALU_Resultin,
   input MemtoReg, RegWrite,
   input [4:0] rd,
   output reg [63:0] ReadDataout,
   output reg [63:0] ALU_Resultout,
   output reg [4:0] rdOut,
   output reg MemtoRegOut, RegWriteOut
);

always @(posedge clk)
begin
   if (reset==1'b0)
   begin
   ReadDataout=ReadDatain;
   ALU_Resultout=ALU_Resultin;
   rdOut=rd;
   MemtoRegOut=MemtoReg;
   RegWriteOut=RegWrite;
   end
end
always@(reset)
begin
   if (reset==1'b1)
   begin
   ReadDataout=64'b0;
   ALU_Resultout=64'b0;
   rdOut=5'b0;
   MemtoRegOut=1'b0;
   RegWriteOut=1'b0;

   end
end
endmodule
```

```
module mux_3(
   input [1:0] sel,
   input [63:0] a,
   input [63:0] b,
   input [63:0] c,
   output reg [63:0] three_muxout
);
always @(*)
begin
   case(sel)
   2'b00: three_muxout=a;
   2'b01: three_muxout=b;
   2'b10: three_muxout=c;
   endcase
```

```
module parser (
        input [31:0] instruction,
        output reg [6:0] opcode,
        output reg [4:0] rd,
        output reg [2:0] func3,
        output reg [4:0] rs1,
        output reg [4:0] rs2,
        output reg [6:0] func7

);
always@(instruction)
begin
assign opcode=instruction[6:0];
assign rd =instruction[11:7];
```

```
end
endmodule
```

```
assign func3=instruction[14:12];
assign rs1 =instruction[19:15];
assign rs2=instruction[24:20];
assign func7=instruction[31:25];
end

endmodule
```

```
module Program_Counter(
input clk, reset,
input [63:0] Pc_In,
output reg [63:0] PC_Out
);

always @(posedge clk or posedge reset)
begin
        if (reset)
                PC_Out<=0;
        else
                PC_Out<=Pc_In;


end
endmodule
```

```
module tb();
reg clk;
reg reset;

Riscv rp(.clk(clk), .reset(reset));
initial
begin
clk= 1'b0;
reset=1'b1;
#7 reset =1'b0;
end
always

#5 clk=~clk;

endmodule
```

```
module registerFile(
input [4:0] Rs1, [4:0] Rs2, [4:0] Rd,
input [63:0] WriteData,
input RegWrite,
input clk, reset,
output reg [63:0] ReadData1, reg [63:0]
ReadData2
);

reg [63:0] Array[31:0];
initial
begin
  Array[0]<=64'd0;
  Array[1]<=64'd1;
  Array[2]<=64'd2;
  Array[3]<=64'd3;
  Array[4]<=64'd4;
  Array[5]<=64'd5;
  Array[6]<=64'd6;
  Array[7]<=64'd7;
  Array[8]<=64'd8;
  Array[9]<=64'd9;
  Array[10]<=64'd10;
  Array[11]<=64'd11;
  Array[12]<=64'd12;
```

```
module Riscv(
input clk, reset
);
wire [63:0] b;
assign b= 16'h0000000000000004;

wire [63:0] PC_Out;

wire [63:0] PC_Outidex;
wire [63:0] adder1_out;
wire [63:0] adder2_out;
wire [63:0] adder_out_ex;
wire [63:0] mux1_out;
wire [31:0] Instruction_m;
wire [31:0] Instruction_if;
wire[6:0] opcode;
wire[4:0] rd_parser;
wire[4:0] rd_id;
wire[4:0] rd_mem;
wire[4:0] rd_ex;
wire[2:0] func3;
wire[4:0] rs1;
wire [4:0] rs2;
wire[4:0] rs1_id;
wire [4:0] rs2_id;
```

```
  Array[13]<=64'd13;
  Array[14]<=64'd14;
  Array[15]<=64'd15;
  Array[16]<=64'd16;
  Array[17]<=64'd17;
  Array[18]<=64'd18;
  Array[19]<=64'd19;
  Array[20]<=64'd20;
  Array[21]<=64'd21;
  Array[22]<=64'd22;
  Array[23]<=64'd23;
  Array[24]<=64'd24;
  Array[25]<=64'd25;
  Array[26]<=64'd26;
  Array[27]<=64'd27;
  Array[28]<=64'd28;
  Array[29]<=64'd29;
  Array[30]<=64'd30;
  Array[31]<=64'd31;
 end

always@(clk)
begin
        if (RegWrite==1)
        begin
                Array[Rd]<=WriteData;
        end

end



always @(Rs1, Rs2 , reset , Array , clk, negedge
clk)
begin
        if (reset)
        begin
                ReadData1<=64'b0;
                ReadData2<=64'b0;
        end
        else
        begin
                ReadData1<=Array[Rs1];
                ReadData2<=Array[Rs2];
        end
end
endmodule
```

```
wire[6:0] func7;
wire Branch, MemRead, MemtoReg, MemWrite,
ALUSrc, RegWrite;
wire [1:0] Aluop;
wire Branch_id, MemRead_id, MemtoReg_id,
MemWrite_id, ALUSrc_id, RegWrite_id;
wire Branch_ex, MemRead_ex, MemtoReg_ex,
MemWrite_ex, RegWrite_ex;
wire MemtoReg_mem, RegWrite_mem;
wire [1:0] Aluop_id;
wire [63:0] imm_data;
wire [63:0] imm_dataOut;
wire [3:0] Operation;
wire [63:0] mux2_out;
wire [63:0] ReadData1;
wire [63:0] ReadData2;
wire[63:0] ReadData1Out;
wire [63:0] ReadData2Out;
wire [63:0] ReadData2_ex;
wire [63:0] Result;
wire [63:0]ALU_Result_out;
wire [63:0] ALU_Resultout_mem;
wire Zero;
wire zero_ex;
wire[63:0] Read_data;
wire [63:0] ReadDataout;
wire [63:0] mux3_out;
wire [63:0] if_pc_out;
wire [3:0] funct;
wire [1:0] Forward_A_in;
wire [1:0] Forward_B_in;
wire [63:0] forward_b_muxout;
wire [63:0] alu_64_a;
Program_Counter p1(.clk(clk), .reset(reset),
.Pc_In(mux1_out),.PC_Out(PC_Out));
adder a1(.a(PC_Out),.b(b),.out(adder1_out));
mux
m(.a(adder1_out),.b(adder_out_ex),.sel(Branch_ex &
zero_ex),.data_out(mux1_out));
//adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));

//adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));

Instruction_memory
i1(.Inst_Adress(PC_Out),.Instruction(Instruction_m));
```

```
IFID ifid(.PC_in(PC_Out),
.instruction_in(Instruction_m),
.clk(clk),
.reset(reset),
.PC_out(if_pc_out),
.Instruction_out(Instruction_if)
);

parser
p(.instruction(Instruction_if),.opcode(opcode),.rd(rd_p
arser),.func3(func3),.rs1(rs1),.rs2(rs2),.func7(func7));

Control_Unit c1( .Opcode(opcode), .Branch(Branch),
.MemRead(MemRead), .MemtoReg(MemtoReg),
.MemWrite(MemWrite),
.ALUSrc(ALUSrc),.RegWrite(RegWrite),.ALUOp(Aluop));

registerFile r1 (.Rs1(rs1), .Rs2(rs2),  .Rd(rd_mem),
.WriteData(mux3_out),
.RegWrite(RegWrite_mem),.clk(clk),
.reset(reset),.ReadData1(ReadData1),
.ReadData2(ReadData2));

IDG
idg(.instruction(Instruction_if),.imm_data(imm_data));

IDEX idex(
.clk(clk),
.reset(reset),
.PC_inidex(if_pc_out),
.ReadData1In(ReadData1),
.ReadData2In(ReadData2),
.imm_data(imm_data),
.rs1(rs1),
.rs2(rs2),
.rd(rd_parser),
.inst({Instruction_if[30], Instruction_if[14:12]}),
.MemtoReg(MemtoReg),
.RegWrite(RegWrite),
.branch(Branch),
.MemRead(MemRead),
.MemWrite(MemWrite),
.ALUSrc(ALUSrc),
.ALUOp(Aluop),

.PC_Outidex(PC_Outidex),
.ReadData1Out(ReadData1Out),
.ReadData2Out(ReadData2Out),
```

```
.imm_dataOut(imm_dataOut),
.funct(funct),
.rdOut(rd_id),
.rs1Out(rs1_id),
.rs2Out(rs2_id),
.MemtoRegOut(MemtoReg_id),
.RegWriteOut(RegWrite_id),
.branchOut(Branch_id),
.MemReadOut(MemRead_id),
.MemWriteOut(MemWrite_id),
.ALUSrcOut(ALUSrc_id),
.ALUOpOut(Aluop_id)
);

ALU_Control
a5(.ALUOp(Aluop_id),.Funct(funct),.Operation(Operati
on));

mux
m2(.a(forward_b_muxout),.b(imm_dataOut),.sel(ALUS
rc_id),.data_out(mux2_out));

alu_64
ALU_64(.a(alu_64_a),.b(mux2_out),.ALUOp(Operation)
,.Result(Result),.ZERO(Zero));

adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));
forwarding_unit forward(

   .rs1_idex(rs1_id),
   .rs2_idex(rs2_id),
   .regwrite_memwb(RegWrite_mem),
   .regwrite_exmem(RegWrite_ex),
   .rd_memwb(rd_mem),
   .rd_exmem(rd_ex),

   .Forward_A(Forward_A_in),
   .Forward_B(Forward_B_in)

);

mux_3 firstmux(
   .sel(Forward_A_in),
   .a(ReadData1Out),
   .b(mux3_out),
   .c(ALU_Result_out),
   .three_muxout(alu_64_a)
```

```
);

mux_3 secondmux(
   .sel(Forward_B_in),
   .a(ReadData2Out),
   .b(mux3_out),
   .c(ALU_Result_out),
   .three_muxout(forward_b_muxout)
);


EXMEM exmem(
.clk(clk),
.reset(reset),
.adder_in(adder2_out),
.ZERO_in(Zero),
.ALU_Result_in(Result),
.ReadData2In(forward_b_muxout),
.rd(rd_id),
.MemtoReg(MemtoReg_id),
.RegWrite(RegWrite_id),
.branch(Branch_id),
.MemRead(MemRead_id),
.MemWrite(MemWrite_id),


.adder_out(adder_out_ex),
 .ALU_Result_out(ALU_Result_out),
 .ReadData2out(ReadData2_ex),
 .rdOut(rd_ex),
.zero(zero_ex),
.MemtoRegOut(MemtoReg_ex),
.RegWriteOut(RegWrite_ex),

.branchOut(Branch_ex),
.MemReadOut(MemRead_ex),.MemWriteOut(MemWr
ite_ex)
);


Data_memory d1(.mem_addr
(ALU_Result_out),.write_data(ReadData2_ex),.clk(clk),.
mem_write(MemWrite_ex),.mem_read(MemRead_ex)
,.Read_data(Read_data));

MEMWB mem(
```

	.clk(clk), .reset(reset), .ReadDatain(Read_data), .ALU_Resultin(ALU_Result_out), .MemtoReg(MemtoReg_ex), .RegWrite(RegWrite_ex), .rd(rd_ex), .ReadDataout(ReadDataout), .ALU_Resultout(ALU_Resultout_mem), .rdOut(rd_mem), .MemtoRegOut(MemtoReg_mem), .RegWriteOut(RegWrite_mem)); mux m3(.a(ALU_Resultout_mem),.b(ReadDataout),.sel(Me mtoReg_mem),.data_out(mux3_out)); endmodule

Nimrah Jawed (nj04707)
Umme Salma (us04315)

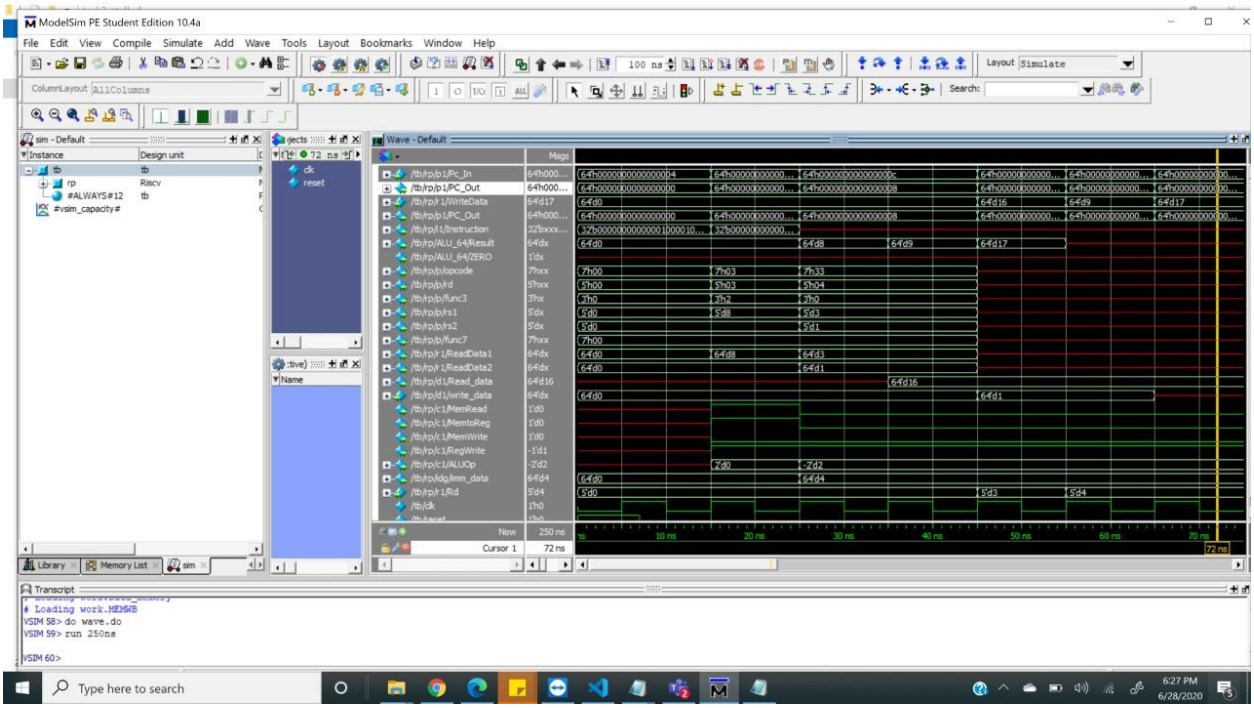
Stalling:

To ensure that the data hazards are taken care of, we have implemented stalling. This basically causes delay in execution of a instruction to resolve a hazard. This stalls the instructions in the fetch stage . This prevents the instruction in that stage from being overwritten by the upcoming instructions.

we have verified this by the following instruction and the result is the same as the expected value. It has been correctly executed.

1. lw x3,0(x8) where x8 has a value of 16
2. add x4,x3,x1

We did not face any challenge at this stage. The book really helped us understanding the process.



module adder(input [63:0] a,	module mux(input[63:0] a, [63:0]b,

```
input [63:0] b,
output reg [63:0] out);

always@(*)
begin
        out = a+b;
end

endmodule
```

```
        input sel,
        output reg[63:0] data_out
);
always@(*)
begin
case(sel)
        1'b0: assign data_out=a;
        1'b1: assign data_out=b;
endcase
end
endmodule
```

```
module alu_64(
        input [63:0] a,[63:0] b, [3:0]
ALUOp,
        output reg [63:0]Result,
        output reg ZERO

);
always @ (*)
begin
  case({ALUOp})

        4'b0111: //slli
        begin
            Result= a<<b;

        end
        4'b0101://bne
        begin
            Result= a-b;
            if(Result==0)
                    begin
                    ZERO=0;
                    end
            else
                    ZERO=1;
        end
        4'b1010:
        begin
            Result = (a<b)?1:0;
            ZERO = Result;
        end

   4'b0000:
        begin
          Result = a&b;
        end
   4'b0001   :
```

```
module ALU_Control(
input [1:0] ALUOp,
input [3:0] Funct,
output reg [3:0] Operation
);

always @(*)
begin
        case(ALUOp)
                2'b00:
                begin
                if (Funct==4'b0001) //slli
                begin
                        Operation=4'b0111;
                end
                else
                begin
                        Operation=4'b0010;
                end
                end

                2'b01:
                begin
                if (Funct==4'b0001)//bne
                begin
                        Operation=4'b0101;
                end
                else if (Funct==4'b0100)//blt
                begin
                        Operation=4'b1010;
                end
                else if (Funct==4'b0000) //beq
                begin
                        Operation=4'b0110;
                end

                end
```

```
            begin
              Result = a|b;
            end

    4'b0010   :
            begin
              Result = a+b;
            end

            4'b0110://beq
            begin
            Result = a-b;
            ZERO = Result?0:1;
            end

            default  : Result = ~(a|b);
   endcase


end
endmodule
```

```
                2'b10:
                begin
                if (Funct==4'b0000)
                begin
                        Operation=4'b0010;
                end
                else if (Funct==4'b1000)
                begin
                        Operation=4'b0110;
                end
                else if (Funct==4'b0111)
                begin
                        Operation=4'b0000;
                end
                else if (Funct==4'b0110)
                begin
                        Operation=4'b0001;
                end
                end
        endcase

end
endmodule
```

```
module Control_Unit(
input [6:0] Opcode,
output reg Branch, MemRead, MemtoReg,
MemWrite, ALUSrc, RegWrite,
output reg [1:0] ALUOp
);


always @(*)
begin
        case(Opcode)
                7'b0110011:
                begin
                ALUSrc=1'b0;
                MemtoReg=1'b0;
                RegWrite=1'b1;
                MemRead=1'b0;
                MemWrite=1'b0;
                Branch=1'b0;
                ALUOp=2'b10;
                end

                7'b0000011:
                begin
```

```
module Data_memory(
input [63:0]mem_addr,
input [63:0] write_data,
input clk,
input mem_write,
input mem_read,
output reg[63:0] Read_data
);

reg [7:0] Array [63:0];

initial
begin
Array[0]=8'b00000100;
Array[1]=8'b00000000;
Array[2]=8'b00000000;
Array[3]=8'b00000000;
Array[4]=8'b00000000;
Array[5]=8'b00000000;
Array[6]=8'b00000000;
Array[7]=8'b00000000;

Array[8]=8'b00010000;
Array[9]=8'b00000000;
```

```
                ALUSrc=1'b1;
                MemtoReg=1'b1;
                RegWrite=1'b1;
                MemRead=1'b1;
                MemWrite=1'b0;
                Branch=1'b0;
                ALUOp=2'b00;
                end

                7'b0100011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'bx;
                RegWrite=1'b0;
                MemRead=1'b0;
                MemWrite=1'b1;
                Branch=1'b0;
                ALUOp=2'b00;
                end

                7'b1100011:
                begin
                ALUSrc=1'b0;
                MemtoReg=1'bx;
                RegWrite=1'b0;
                MemRead=1'b0;
                MemWrite= 1'b0;
                Branch=1'b1;
                ALUOp=2'b01;
                end

                7'b0010011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'b0;
                RegWrite=1'b1;
                MemRead=1'b1;
                MemWrite= 1'b0;
                Branch=1'b0;
                ALUOp=2'b00;
                end


        endcase
end
endmodule
```

```
Array[10]=8'b00000000;
Array[11]=8'b00000000;
Array[12]=8'b00000000;
Array[13]=8'b00000000;
Array[14]=8'b00000000;
Array[15]=8'b00000000;

Array[16]=8'b00000011;
Array[17]=8'b00000000;
Array[18]=8'b00000000;
Array[19]=8'b00000000;
Array[20]=8'b00000000;
Array[21]=8'b00000000;
Array[22]=8'b00000000;
Array[23]=8'b00000000;

Array[24]=8'b00000010;
Array[25]=8'b00000000;
Array[26]=8'b00000000;
Array[27]=8'b00000000;
Array[28]=8'b00000000;
Array[29]=8'b00000000;
Array[30]=8'b00000000;
Array[31]=8'b00000000;

Array[32]=8'b00100000;
Array[33]=8'b00000000;
Array[34]=8'b00000000;
Array[35]=8'b00000000;
Array[36]=8'b00000000;
Array[37]=8'b00000000;
Array[38]=8'b00000000;
Array[39]=8'b00000000;




end

always @(*)
begin
        if (mem_read)
        begin

        Read_data={Array[mem_addr+7],Array[mem_a
ddr+6],
Array[mem_addr+5],Array[mem_addr+4],Array[mem_a
ddr+3],Array[mem_addr+2],
Array[mem_addr+1],Array[mem_addr]};
```

```
        end

end

always @(posedge clk)
begin
        if (mem_write)
        begin
                Array[mem_addr]=write_data[7:0];
                Array[mem_addr+1]=write_data[15:8];

        Array[mem_addr+2]=write_data[23:16];

        Array[mem_addr+3]=write_data[31:24];

        Array[mem_addr+4]=write_data[39:32];

        Array[mem_addr+5]=write_data[47:40];

        Array[mem_addr+6]=write_data[55:48];

        Array[mem_addr+7]=write_data[63:56];
        end
end

endmodule
```

```
module EXMEM(
input clk,
input reset,
input [63:0] adder_in,
input ZERO_in,
input [63:0] ALU_Result_in,
input [63:0] ReadData2In,
input [4:0] rd,
input MemtoReg,RegWrite, branch,
MemRead,MemWrite,

output reg [63:0] adder_out,

output reg [63:0] ALU_Result_out,
output reg [63:0] ReadData2out,
output reg [4:0] rdOut,
output reg zero,
output reg MemtoRegOut, RegWriteOut,
branchOut, MemReadOut, MemWriteOut
);

always @(posedge clk)
```

```
module forwarding_unit(

   input[4:0] rs1_idex,
   input [4:0] rs2_idex,
   input regwrite_memwb,
   input regwrite_exmem,
   input [4:0] rd_memwb,
   input [4:0] rd_exmem,

   output reg [1:0] Forward_A,
   output reg [1:0] Forward_B

);



always @(*)
begin

if ((regwrite_exmem==1'b1) && (rd_exmem!=1'b0)&&
(rd_exmem==rs1_idex))
   Forward_A=2'b10;
```

```
begin
   if (reset==1'b0)
   begin
   adder_out=  adder_in;
   ALU_Result_out= ALU_Result_in;
   ReadData2out= ReadData2In;
   rdOut=rd;
   zero=ZERO_in;
   MemtoRegOut=MemtoReg;
   RegWriteOut=RegWrite;
   branchOut=branch;
   MemReadOut= MemRead;
   MemWriteOut=MemWrite;

   end
end
always@(reset)
begin
   if (reset==1'b1)
   begin
   adder_out=  64'b0;
   ALU_Result_out= 64'b0;
   ReadData2out= 64'b0;
   rdOut=5'b0;
   zero=1'b0;
   MemtoRegOut=1'b0;
   RegWriteOut=1'b0;
   branchOut=1'b0;
   MemReadOut= 1'b0;
   MemWriteOut=1'b0;
   end
end
endmodule
```

```
else if ((regwrite_memwb==1'b1)&&
(rd_memwb!=1'b0)&& (rd_memwb==rs1_idex))
   Forward_A=2'b01;
else
         Forward_A=2'b00;

if ((regwrite_exmem==1'b1)&& (rd_exmem!=1'b0)&&
(rd_exmem==rs2_idex))
   Forward_B=2'b10;

else if ((regwrite_memwb==1'b1)&&
(rd_memwb!=1'b0)&& (rd_memwb==rs2_idex))
   Forward_B=2'b01;
else
         Forward_B=2'b00;



end
endmodule
```

```
module hazard_unit (
input MemRead_idex,
input [4:0] rd_idex,
input [4:0] Rs1_ifid,
input [4:0] Rs2_ifid,
output reg PcWrite,
output reg Ifid_write,
output reg mux_sel
);
always@(*)
begin



if (MemRead_idex && ((rd_idex==Rs1_ifid) ||
(rd_idex==Rs2_ifid)))
```

```
module IDG(
input [31:0] instruction,
output reg [63:0] imm_data
);
always @(*)
begin
         case(instruction[6:5])
                  2'b00:
                  begin
                  imm_data=
{{52{instruction[31]}},instruction[31:20]};
                  end
                  2'b01:
                  begin
```

```
   begin
   PcWrite=1'b0;
   Ifid_write=1'b0;
   mux_sel=1'b1;
   end
else
begin
   PcWrite=1'b1;
   Ifid_write=1'b1;
   mux_sel=1'b0;

end
end
endmodule
```

```
                imm_data=
{{52{instruction[31]}},instruction[31:25],instruction[11:
7]};

                end
                2'b10:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31],instruction[7],instr
uction[30:25],instruction[11:8]};
                end
                2'b11:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31],instruction[7],instr
uction[30:25],instruction[11:8]};
                end
          endcase
end
endmodule
```

```
module IDEX(
input clk,
input reset,
input [63:0] PC_inidex,
input [63:0] ReadData1In,
input [63:0] ReadData2In,
input [63:0] imm_data,
input [4:0] rs1,
input [4:0] rs2,
input [4:0] rd,
input [3:0] inst,
input MemtoReg,
input RegWrite,
input branch,
input MemRead,
input MemWrite,
input ALUSrc,
input [1:0] ALUOp,
output reg [63:0] PC_Outidex,
output reg [63:0] ReadData1Out,
output reg [63:0] ReadData2Out,
output reg [63:0] imm_dataOut,
output reg [3:0] funct,
output reg [4:0] rdOut,
output reg [4:0] rs1Out,
output reg [4:0] rs2Out,
```

```
module registerFile(
input [4:0] Rs1, [4:0] Rs2, [4:0] Rd,
input [63:0] WriteData,
input RegWrite,
input clk, reset,
output reg [63:0] ReadData1, reg [63:0] ReadData2
);

reg [63:0] Array[31:0];
initial
begin
  Array[0]<=64'd0;
  Array[1]<=64'd1;
  Array[2]<=64'd2;
  Array[3]<=64'd3;
  Array[4]<=64'd4;
  Array[5]<=64'd5;
  Array[6]<=64'd6;
  Array[7]<=64'd7;
  Array[8]<=64'd8;
  Array[9]<=64'd9;
  Array[10]<=64'd10;
  Array[11]<=64'd11;
  Array[12]<=64'd12;
  Array[13]<=64'd13;
  Array[14]<=64'd14;
  Array[15]<=64'd15;
  Array[16]<=64'd16;
```

```
output reg MemtoRegOut,
RegWriteOut,branchOut,MemReadOut,Mem
WriteOut,ALUSrcOut,
output reg [1:0] ALUOpOut
);

always @(posedge clk)
begin
        if (reset==1'b0)
                begin

        PC_Outidex=PC_inidex;

        ReadData1Out=ReadData1In;

        ReadData2Out=ReadData2In;

        imm_dataOut=imm_data;
                        funct=inst;
                        rdOut=rd;
                        rs1Out=rs1;
                        rs2Out=rs2;

        MemtoRegOut=MemtoReg;

        RegWriteOut=RegWrite;
                        branchOut=branch;
                        MemReadOut=
MemRead;

        MemWriteOut=MemWrite;
                        ALUSrcOut=ALUSrc;
                        ALUOpOut=ALUOp;
                end
end
always@(reset)
begin
        if(reset==1'b1)
                begin
                        PC_Outidex=64'b0;

        ReadData1Out=64'b0;

        ReadData2Out=64'b0;
                        imm_dataOut=64'b0;
                        funct=4'b0;
                        rdOut=5'b0;
                        rs1Out=5'b0;
  Array[17]<=64'd17;
  Array[18]<=64'd18;
  Array[19]<=64'd19;
  Array[20]<=64'd20;
  Array[21]<=64'd21;
  Array[22]<=64'd22;
  Array[23]<=64'd23;
  Array[24]<=64'd24;
  Array[25]<=64'd25;
  Array[26]<=64'd26;
  Array[27]<=64'd27;
  Array[28]<=64'd28;
  Array[29]<=64'd29;
  Array[30]<=64'd30;
  Array[31]<=64'd31;
 end

always@(negedge clk)
begin
        if (RegWrite==1)
        begin
                Array[Rd]<=WriteData;
        end

end


always @(Rs1, Rs2 , reset , Array , clk, posedge clk)
begin
        if (reset)
        begin
                ReadData1<=64'b0;
                ReadData2<=64'b0;
        end
        else
        begin
                ReadData1<=Array[Rs1];
                ReadData2<=Array[Rs2];
        end
end
endmodule
```

```
                        rs2Out= 5'b0;
                        MemtoRegOut=1'b0;
                        RegWriteOut=1'b0;
                        branchOut=1'b0;
                        MemReadOut= 1'b0;
                        MemWriteOut=1'b0;
                        ALUSrcOut=1'b0;
                        ALUOpOut=2'b0;



                end
end
endmodule
```

```
module IFID(
input [63:0] PC_in,
input [31:0] instruction_in,
input clk,
input ifid_write,
input reset,
output reg [63:0] PC_out,
output reg [31:0] Instruction_out
);

always@(posedge clk)
begin
        if (ifid_write==1'b1)
        begin
        if (reset==1'b0)
                begin
                PC_out=PC_in;

        Instruction_out=instruction_in;
                end
        end
end

always@(reset)
begin
        if (reset==1'b1)
                begin
                PC_out=64'b0;
                Instruction_out=32'b0;
                end
end

endmodule
```

```
module Instruction_memory(
input [63:0] Inst_Adress,
output reg [31:0] Instruction
);

reg [7:0]Array[7:0];
 initial
   begin



      Array[0] =8'b10000011;
      Array[1] =8'b00100001;
      Array[2] =8'b00000100;
      Array[3] =8'b00000000;

      Array[4] =8'b00110011;
      Array[5] =8'b10000010;
      Array[6] =8'b00010001;
      Array[7] =8'b00000000;


   end



always @ (Inst_Adress)
begin

        Instruction={Array[Inst_Adress+3],Array[Inst_A
dress+2], Array[Inst_Adress+1],Array[Inst_Adress]};
end
endmodule
```

```
module MEMWB(
  input clk,
```

```
module parser (
        input [31:0] instruction,
```

```
   input reset,
   input [63:0] ReadDatain,
   input [63:0] ALU_Resultin,
   input MemtoReg, RegWrite,
   input [4:0] rd,
   output reg [63:0] ReadDataout,
   output reg [63:0] ALU_Resultout,
   output reg [4:0] rdOut,
   output reg MemtoRegOut, RegWriteOut
);

always @(posedge clk)
begin
   if (reset==1'b0)
   begin
   ReadDataout=ReadDatain;
   ALU_Resultout=ALU_Resultin;
   rdOut=rd;
   MemtoRegOut=MemtoReg;
   RegWriteOut=RegWrite;
   end
end
always@(reset)
begin
   if (reset==1'b1)
   begin
   ReadDataout=64'b0;
   ALU_Resultout=64'b0;
   rdOut=5'b0;
   MemtoRegOut=1'b0;
   RegWriteOut=1'b0;

   end
end
endmodule
```

```
          output reg [6:0] opcode,
          output reg [4:0] rd,
          output reg [2:0] func3,
          output reg [4:0] rs1,
          output reg [4:0] rs2,
          output reg [6:0] func7

);
always@(instruction)
begin
assign opcode=instruction[6:0];
assign rd =instruction[11:7];
assign func3=instruction[14:12];
assign rs1 =instruction[19:15];
assign rs2=instruction[24:20];
assign func7=instruction[31:25];
end

endmodule
```

```
module mux_3(
   input [1:0] sel,
   input [63:0] a,
   input [63:0] b,
   input [63:0] c,
   output reg [63:0] three_muxout
);
always @(*)
begin
   case(sel)
   2'b00: three_muxout=a;
   2'b01: three_muxout=b;
   2'b10: three_muxout=c;
```

```
module tb();
reg clk;
reg reset;

Riscv rp(.clk(clk), .reset(reset));
initial
begin
clk= 1'b0;
reset=1'b1;
#7 reset =1'b0;
end
always
```

endcase end endmodule	#5 clk=~clk; endmodule

module Program_Counter(
input clk, reset,
input [63:0] Pc_In,
input PCWrite,
output reg [63:0] PC_Out
);
 	
always @(posedge clk or posedge reset)
begin
if (reset)
PC_Out<=0;
else
if (PCWrite==1'b1)
begin
PC_Out<=Pc_In;
end
 	
end
endmodule
	module Riscv(
	input clk, reset
	);
	wire [63:0] b;
	assign b= 16'h0000000000000004;
	wire [63:0] PC_Out;
	wire [63:0] PC_Outidex;
	wire [63:0] adder1_out;
	wire [63:0] adder2_out;
	wire [63:0] adder_out_ex;
	wire [63:0] mux1_out;
	wire [31:0] Instruction_m;
	wire [31:0] Instruction_if;
	wire[6:0] opcode;
	wire[4:0] rd_parser;
	wire[4:0] rd_id;
	wire[4:0] rd_mem;
	wire[4:0] rd_ex;
	wire[2:0] func3;
	wire[4:0] rs1;
	wire [4:0] rs2;
	wire[4:0] rs1_id;
	wire [4:0] rs2_id;
	wire[6:0] func7;
	wire Branch, MemRead, MemtoReg, MemWrite,
	ALUSrc, RegWrite;
	wire Branch_mux, MemRead_mux, MemtoReg_mux,
	MemWrite_mux, ALUSrc_mux, RegWrite_mux;
	wire [1:0] Aluop_mux;
	wire [1:0] Aluop;
	wire Branch_id, MemRead_id, MemtoReg_id,
	MemWrite_id, ALUSrc_id, RegWrite_id;
	wire Branch_ex, MemRead_ex, MemtoReg_ex,
	MemWrite_ex, RegWrite_ex;
	wire MemtoReg_mem, RegWrite_mem;
	wire [1:0] Aluop_id;
	wire [63:0] imm_data;
	wire [63:0] imm_dataOut;
	wire [3:0] Operation;
	wire [63:0] mux2_out;
	wire [63:0] ReadData1;
	wire [63:0] ReadData2;
	wire[63:0] ReadData1Out;
	wire [63:0] ReadData2Out;
	wire [63:0] ReadData2_ex;

```
wire [63:0] Result;
wire [63:0]ALU_Result_out;
wire [63:0] ALU_Resultout_mem;
wire Zero;
wire zero_ex;
wire[63:0] Read_data;
wire [63:0] ReadDataout;
wire [63:0] mux3_out;
wire [63:0] if_pc_out;
wire [3:0] funct;
wire [1:0] Forward_A_in;
wire [1:0] Forward_B_in;
wire [63:0] forward_b_muxout;
wire [63:0] alu_64_a;
wire PcWrite_ctrl;
wire ifid_write_ctrl;
wire mux_sel_ctrl;
wire [63:0] control_idex_out;
Program_Counter p1(.clk(clk),
.reset(reset),.PCWrite(PcWrite_ctrl),
.Pc_In(mux1_out),.PC_Out(PC_Out));
adder a1(.a(PC_Out),.b(b),.out(adder1_out));
mux m(.a(adder1_out),.b(adder_out_ex),.sel(Branch_ex
& zero_ex),.data_out(mux1_out));
//adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));

//adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));

Instruction_memory
i1(.Inst_Adress(PC_Out),.Instruction(Instruction_m));

IFID ifid(.PC_in(PC_Out),
.instruction_in(Instruction_m),
.clk(clk),
.ifid_write(ifid_write_ctrl),
.reset(reset),
.PC_out(if_pc_out),
.Instruction_out(Instruction_if)
);
parser
p(.instruction(Instruction_if),.opcode(opcode),.rd(rd_pa
rser),.func3(func3),.rs1(rs1),.rs2(rs2),.func7(func7));

Control_Unit c1( .Opcode(opcode), .Branch(Branch),
.MemRead(MemRead), .MemtoReg(MemtoReg),
```

```
.MemWrite(MemWrite),
.ALUSrc(ALUSrc),.RegWrite(RegWrite),.ALUOp(Aluop));
hazard_unit hu(
.MemRead_idex(MemRead_id),
.rd_idex(rd_id),
.Rs1_ifid(rs1),
.Rs2_ifid(rs2),
.PcWrite(PcWrite_ctrl),
.Ifid_write(ifid_write_ctrl),
.mux_sel(mux_sel_ctrl)
);
mux
mux_control(.a({56'd0,{RegWrite},{MemtoReg},{MemR
ead},{MemWrite},{Branch},{ALUSrc},{Aluop}}),
.b(64'd0),
.sel(mux_sel_ctrl),
.data_out(control_idex_out)
);
registerFile r1 (.Rs1(rs1), .Rs2(rs2),  .Rd(rd_mem),
.WriteData(mux3_out),
.RegWrite(RegWrite_mem),.clk(clk),
.reset(reset),.ReadData1(ReadData1),
.ReadData2(ReadData2));
IDG
idg(.instruction(Instruction_if),.imm_data(imm_data));

IDEX idex(
.clk(clk),
.reset(reset),
.PC_inidex(if_pc_out),
.ReadData1In(ReadData1),
.ReadData2In(ReadData2),
.imm_data(imm_data),
.rs1(rs1),
.rs2(rs2),
.rd(rd_parser),
.inst({Instruction_if[30], Instruction_if[14:12]}),
.MemtoReg(control_idex_out[6]),
.RegWrite(control_idex_out[7]),
.branch(control_idex_out[3]),
.MemRead(control_idex_out[5]),
.MemWrite(control_idex_out[4]),
.ALUSrc(control_idex_out[2]),
.ALUOp(control_idex_out[1:0]),
.PC_Outidex(PC_Outidex),
.ReadData1Out(ReadData1Out),
.ReadData2Out(ReadData2Out),
.imm_dataOut(imm_dataOut),
```

```
.funct(funct),
.rdOut(rd_id),
.rs1Out(rs1_id),
.rs2Out(rs2_id),
.MemtoRegOut(MemtoReg_id),
.RegWriteOut(RegWrite_id),
.branchOut(Branch_id),
.MemReadOut(MemRead_id),
.MemWriteOut(MemWrite_id),
.ALUSrcOut(ALUSrc_id),
.ALUOpOut(Aluop_id)
);
ALU_Control
a5(.ALUOp(Aluop_id),.Funct(funct),.Operation(Operatio
n));
alu_64
ALU_64(.a(alu_64_a),.b(forward_b_muxout),.ALUOp(O
peration),.Result(Result),.ZERO(Zero));
adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));
forwarding_unit forward(
   .rs1_idex(rs1_id),
   .rs2_idex(rs2_id),
   .regwrite_memwb(RegWrite_mem),
   .regwrite_exmem(RegWrite_ex),
   .rd_memwb(rd_mem),
   .rd_exmem(rd_ex),
   .Forward_A(Forward_A_in),
   .Forward_B(Forward_B_in));
mux_3 firstmux(
   .sel(Forward_A_in),
   .a(ReadData1Out),
   .b(mux3_out),
   .c(ALU_Result_out),
   .three_muxout(alu_64_a)
);
mux_3 secondmux(
   .sel(Forward_B_in),
   .a(ReadData2Out),
   .b(mux3_out),
   .c(ALU_Result_out),
   .three_muxout(forward_b_muxout));
EXMEM exmem(
.clk(clk),
.reset(reset),
.adder_in(adder2_out),
.ZERO_in(Zero),
.ALU_Result_in(Result),
```

```
.ReadData2In(forward_b_muxout),
.rd(rd_id),
.MemtoReg(MemtoReg_id),
.RegWrite(RegWrite_id),
.branch(Branch_id),
.MemRead(MemRead_id),
.MemWrite(MemWrite_id),
.adder_out(adder_out_ex),
 .ALU_Result_out(ALU_Result_out),
 .ReadData2out(ReadData2_ex),
 .rdOut(rd_ex),
.zero(zero_ex),
.MemtoRegOut(MemtoReg_ex),
.RegWriteOut(RegWrite_ex),
.branchOut(Branch_ex),
.MemReadOut(MemRead_ex),.MemWriteOut(MemWri
te_ex)
);
Data_memory d1(.mem_addr
(ALU_Result_out),.write_data(ReadData2_ex),.clk(clk),.
mem_write(MemWrite_ex),.mem_read(MemRead_ex),
.Read_data(Read_data));

MEMWB mem(.clk(clk),
.reset(reset),
.ReadDatain(Read_data),
.ALU_Resultin(ALU_Result_out),
.MemtoReg(MemtoReg_ex), .RegWrite(RegWrite_ex),
.rd(rd_ex),.ReadDataout(ReadDataout),
.ALU_Resultout(ALU_Resultout_mem),
.rdOut(rd_mem),
.MemtoRegOut(MemtoReg_mem),
.RegWriteOut(RegWrite_mem));
mux
m3(.a(ALU_Resultout_mem),.b(ReadDataout),.sel(Mem
toReg_mem),.data_out(mux3_out));
endmodule
```

Flushing:

The and gate with inputs branch (from ex) and zero(from ex) will give an output ALUSrc. We are then sending this to IFID, IDEX and EXMEM. If this is 0, then these modules will assign the input values. If this signal is high, it will flush all contents of IFID IDEX and EXMEM and outputs zeros. Even though the logic seems correct, we were unable to debug exactly where the issue lies. We traced the main main signals and they seemed to have glitches. Especially the ZERO signal. Our processor , at this stage doesn't take any branches and executes the code sequentially.

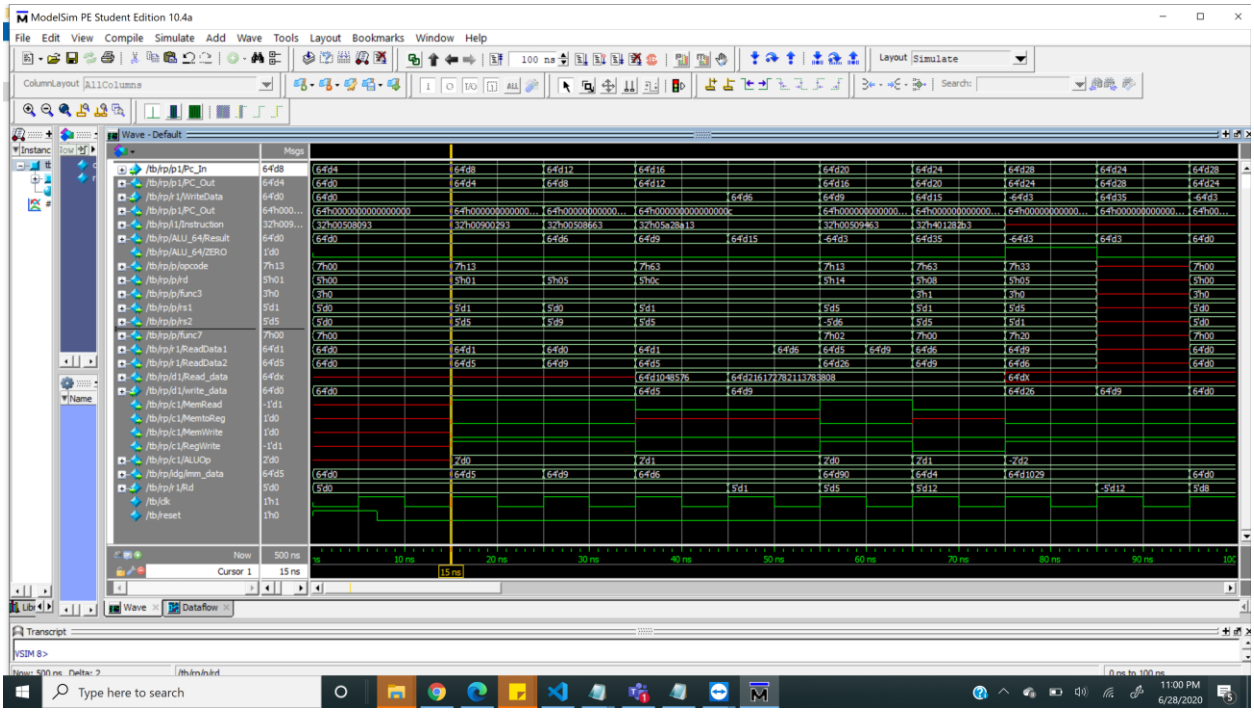
To verify this we executed the following code:

1. addi x1,x1,5
2. addi x5,x0,9
3. beq x1,x5,if
4. addi x20,x5,90
5. bne x1,x5,exit
6. if:
7. sub x5, x5,x1
8. exit:

In this case, the results were not same as expected.

As we can see in the following simulation, the program counter increases sequentially, without taking any branches.

Nimrah Jawed (nj04707)
Umme Salma (us04315)




```
module adder(
input [63:0] a,
input [63:0] b,
output reg [63:0] out);

always@(*)
begin
        out = a+b;
end

endmodule
```

```
module mux(
        input[63:0] a, [63:0]b,
        input sel,
        output reg[63:0] data_out
);
always@(*)
begin
case(sel)
        1'b0: assign data_out=a;
        1'b1: assign data_out=b;
endcase
end
endmodule
```

```
module alu_64(
        input [63:0] a,[63:0] b, [3:0]
ALUOp,
        output reg [63:0]Result,
        output reg ZERO

);
always @ (*)
begin
  case({ALUOp})

        4'b0111: //slli
        begin
                Result= a<<b;
```

```
module ALU_Control(
input [1:0] ALUOp,
input [3:0] Funct,
output reg [3:0] Operation
);

always @(*)
begin
        case(ALUOp)
                2'b00:
                begin
                if (Funct==4'b0001) //slli
                begin
                        Operation=4'b0111;
```

```
                ZERO=1'b0;

        end
        4'b0101://bne
        begin
                Result= a-b;
                if(Result==0)
                        begin
                        ZERO=1'b0;
                        end
                else
                        ZERO=1'b1;
        end
        4'b1010:
        begin
                Result = (a<b)?1:0;
                ZERO = Result;
        end

    4'b0000:
        begin
          Result = a&b;
          ZERO=1'b0;
        end
    4'b0001   :
        begin
          Result = a|b;
          ZERO=1'b0;
        end

    4'b0010   :
        begin
          Result = a+b;
          ZERO=1'b0;
        end

        4'b0110://beq
        begin
        Result = a-b;
        ZERO = Result?0:1;
        end

        default  : Result = ~(a|b);
   endcase


end
endmodule
```

```
                end
                else
                begin
                        Operation=4'b0010;
                end
                end

                2'b01:
                begin
                if (Funct==4'b0001)//bne
                begin
                        Operation=4'b0101;
                end
                else if (Funct==4'b0100)//blt
                begin
                        Operation=4'b1010;
                end
                else if (Funct==4'b0000) //beq
                begin
                        Operation=4'b0110;
                end

                end

                2'b10:
                begin
                if (Funct==4'b0000)
                begin
                        Operation=4'b0010;
                end
                else if (Funct==4'b1000)
                begin
                        Operation=4'b0110;
                end
                else if (Funct==4'b0111)
                begin
                        Operation=4'b0000;
                end
                else if (Funct==4'b0110)
                begin
                        Operation=4'b0001;
                end
                end
        endcase

end
endmodule
```

```
module Control_Unit(
input [6:0] Opcode,
output reg Branch, MemRead,
MemtoReg, MemWrite, ALUSrc,
RegWrite,
output reg [1:0] ALUOp
);


always @(*)
begin
        case(Opcode)
                7'b0110011:
                begin
                ALUSrc=1'b0;
                MemtoReg=1'b0;
                RegWrite=1'b1;
                MemRead=1'b0;
                MemWrite=1'b0;
                Branch=1'b0;
                ALUOp=2'b10;
                end

                7'b0000011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'b1;
                RegWrite=1'b1;
                MemRead=1'b1;
                MemWrite=1'b0;
                Branch=1'b0;
                ALUOp=2'b00;
                end

                7'b0100011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'bx;
                RegWrite=1'b0;
                MemRead=1'b0;
                MemWrite=1'b1;
                Branch=1'b0;
                ALUOp=2'b00;
                end

                7'b1100011:
                begin
                ALUSrc=1'b0;
```

```
module Data_memory(
input [63:0]mem_addr,
input [63:0] write_data,
input clk,
input mem_write,
input mem_read,
output reg[63:0] Read_data
);

reg [7:0] Array [63:0];

initial
begin
Array[0]=8'b00000100;
Array[1]=8'b00000000;
Array[2]=8'b00000000;
Array[3]=8'b00000000;
Array[4]=8'b00000000;
Array[5]=8'b00000000;
Array[6]=8'b00000000;
Array[7]=8'b00000000;

Array[8]=8'b00010000;
Array[9]=8'b00000000;
Array[10]=8'b00000000;
Array[11]=8'b00000000;
Array[12]=8'b00000000;
Array[13]=8'b00000000;
Array[14]=8'b00000000;
Array[15]=8'b00000000;

Array[16]=8'b00000011;
Array[17]=8'b00000000;
Array[18]=8'b00000000;
Array[19]=8'b00000000;
Array[20]=8'b00000000;
Array[21]=8'b00000000;
Array[22]=8'b00000000;
Array[23]=8'b00000000;

Array[24]=8'b00000010;
Array[25]=8'b00000000;
Array[26]=8'b00000000;
Array[27]=8'b00000000;
Array[28]=8'b00000000;
Array[29]=8'b00000000;
Array[30]=8'b00000000;
Array[31]=8'b00000000;
```

```
                MemtoReg=1'bx;
                RegWrite=1'b0;
                MemRead=1'b0;
                MemWrite= 1'b0;
                Branch=1'b1;
                ALUOp=2'b01;
                end

                7'b0010011:
                begin
                ALUSrc=1'b1;
                MemtoReg=1'b0;
                RegWrite=1'b1;
                MemRead=1'b1;
                MemWrite= 1'b0;
                Branch=1'b0;
                ALUOp=2'b00;
                end


        endcase
end
endmodule
```

```
Array[32]=8'b00100000;
Array[33]=8'b00000000;
Array[34]=8'b00000000;
Array[35]=8'b00000000;
Array[36]=8'b00000000;
Array[37]=8'b00000000;
Array[38]=8'b00000000;
Array[39]=8'b00000000;




end

always @(*)
begin
        if (mem_read)
        begin

        Read_data={Array[mem_addr+7],Array[mem_addr+6
],
Array[mem_addr+5],Array[mem_addr+4],Array[mem_addr+3
],Array[mem_addr+2],
Array[mem_addr+1],Array[mem_addr]};
        end

end

always @(posedge clk)
begin
        if (mem_write)
        begin
                Array[mem_addr]=write_data[7:0];
                Array[mem_addr+1]=write_data[15:8];
                Array[mem_addr+2]=write_data[23:16];
                Array[mem_addr+3]=write_data[31:24];
                Array[mem_addr+4]=write_data[39:32];
                Array[mem_addr+5]=write_data[47:40];
                Array[mem_addr+6]=write_data[55:48];
                Array[mem_addr+7]=write_data[63:56];
        end
end

endmodule
```

```
module EXMEM(
input clk,
input reset,
input [63:0] adder_in,
```

```
module IDEX(
input clk,
input reset,
input [63:0] PC_inidex,
```

```
input ZERO_in,
input PCSrc,
input [63:0] ALU_Result_in,
input [63:0] ReadData2In,
input [4:0] rd,
input MemtoReg,RegWrite, branch,
MemRead,MemWrite,

output reg [63:0] adder_out,

output reg [63:0] ALU_Result_out,
output reg [63:0] ReadData2out,
output reg [4:0] rdOut,
output reg zero,
output reg MemtoRegOut,
RegWriteOut, branchOut,
MemReadOut, MemWriteOut
);

always @(posedge clk)
begin
if (PCSrc==1'b0)
begin
   if (reset==1'b0)
   begin
   adder_out=  adder_in;
   ALU_Result_out= ALU_Result_in;
   ReadData2out= ReadData2In;
   rdOut=rd;
   zero=ZERO_in;
   MemtoRegOut=MemtoReg;
   RegWriteOut=RegWrite;
   branchOut=branch;
   MemReadOut= MemRead;
   MemWriteOut=MemWrite;
   end
end
else
begin
   adder_out=  64'b0;
   ALU_Result_out= 64'b0;
   ReadData2out= 64'b0;
   rdOut=5'b0;
   zero=1'b0;
   MemtoRegOut=1'b0;
   RegWriteOut=1'b0;
   branchOut=1'b0;
   MemReadOut= 1'b0;
```

```
input [63:0] ReadData1In,
input [63:0] ReadData2In,
input [63:0] imm_data,
input [4:0] rs1,
input [4:0] rs2,
input [4:0] rd,
input [3:0] inst,
input PCSrc,
input MemtoReg,
input RegWrite,
input branch,
input MemRead,
input MemWrite,
input ALUSrc,
input [1:0] ALUOp,
output reg [63:0] PC_Outidex,
output reg [63:0] ReadData1Out,
output reg [63:0] ReadData2Out,
output reg [63:0] imm_dataOut,
output reg [3:0] funct,
output reg [4:0] rdOut,
output reg [4:0] rs1Out,
output reg [4:0] rs2Out,
output reg MemtoRegOut,
RegWriteOut,branchOut,MemReadOut,MemWriteOut,ALUSrc
Out,
output reg [1:0] ALUOpOut
);
always @(posedge clk)
begin
if (PCSrc==1'b0)
begin
        if (reset==1'b0)
                begin
                        PC_Outidex=PC_inidex;
                        ReadData1Out=ReadData1In;
                        ReadData2Out=ReadData2In;
                        imm_dataOut=imm_data;
                        funct=inst;
                        rdOut=rd;
                        rs1Out=rs1;
                        rs2Out=rs2;
                        MemtoRegOut=MemtoReg;
                        RegWriteOut=RegWrite;
                        branchOut=branch;
                        MemReadOut= MemRead;
                        MemWriteOut=MemWrite;
                        ALUSrcOut=ALUSrc;
```

```
   MemWriteOut=1'b0;
end

end
always@(reset)
begin
   if (reset==1'b1)
   begin
   adder_out=  64'b0;
   ALU_Result_out= 64'b0;
   ReadData2out= 64'b0;
   rdOut=5'b0;
   zero=1'b0;
   MemtoRegOut=1'b0;
   RegWriteOut=1'b0;
   branchOut=1'b0;
   MemReadOut= 1'b0;
   MemWriteOut=1'b0;
   end
end
endmodule
```

```
                ALUOpOut=ALUOp;
            end
end
else
        begin
                PC_Outidex=64'b0;
                ReadData1Out=64'b0;
                ReadData2Out=64'b0;
                imm_dataOut=64'b0;
                funct=4'b0;
                rdOut=5'b0;
                rs1Out=5'b0;
                rs2Out= 5'b0;
                MemtoRegOut=1'b0;
                RegWriteOut=1'b0;
                branchOut=1'b0;
                MemReadOut= 1'b0;
                MemWriteOut=1'b0;
                ALUSrcOut=1'b0;
                ALUOpOut=2'b0;
        end
end
always@(reset)
begin
        if(reset==1'b1)
                begin
                PC_Outidex=64'b0;
                ReadData1Out=64'b0;
                ReadData2Out=64'b0;
                imm_dataOut=64'b0;
                funct=4'b0;
                rdOut=5'b0;
                rs1Out=5'b0;
                rs2Out= 5'b0;
                MemtoRegOut=1'b0;
                RegWriteOut=1'b0;
                branchOut=1'b0;
                MemReadOut= 1'b0;
                MemWriteOut=1'b0;
                ALUSrcOut=1'b0;
                ALUOpOut=2'b0;
                end
end
endmodule
```

```
module forwarding_unit(

   input[4:0] rs1_idex,
   input [4:0] rs2_idex,
```

```
module hazard_unit (
input MemRead_idex,
input [4:0] rd_idex,
input [4:0] Rs1_ifid,
```

```
   input regwrite_memwb,
   input regwrite_exmem,
   input [4:0] rd_memwb,
   input [4:0] rd_exmem,

   output reg [1:0] Forward_A,
   output reg [1:0] Forward_B

);

always @(*)
begin

if ((regwrite_exmem==1'b1) &&
(rd_exmem!=1'b0)&&
(rd_exmem==rs1_idex))
   Forward_A=2'b10;
else if ((regwrite_memwb==1'b1)&&
(rd_memwb!=1'b0)&&
(rd_memwb==rs1_idex))
   Forward_A=2'b01;
else
          Forward_A=2'b00;

if ((regwrite_exmem==1'b1)&&
(rd_exmem!=1'b0)&&
(rd_exmem==rs2_idex))
   Forward_B=2'b10;

else if ((regwrite_memwb==1'b1)&&
(rd_memwb!=1'b0)&&
(rd_memwb==rs2_idex))
   Forward_B=2'b01;
else
          Forward_B=2'b00;
end
endmodule
```

```
input [4:0] Rs2_ifid,
output reg PcWrite,
output reg Ifid_write,
output reg mux_sel
);
always@(*)
begin


if (MemRead_idex && ((rd_idex==Rs1_ifid) ||
(rd_idex==Rs2_ifid)))
   begin
   PcWrite=1'b0;
   Ifid_write=1'b0;
   mux_sel=1'b1;
   end
else
begin
   PcWrite=1'b1;
   Ifid_write=1'b1;
   mux_sel=1'b0;

end
end
endmodule
```

```
module IDG(
input [31:0] instruction,
output reg [63:0] imm_data
);
always @(*)
begin
          case(instruction[6:5])
                    2'b00:
                    begin
```

```
module IFID(
input [63:0] PC_in,
input [31:0] instruction_in,
input clk,
input PCSrc,
input ifid_write,
input reset,
output reg [63:0] PC_out,
output reg [31:0] Instruction_out
);
```

```
                imm_data=
{{52{instruction[31]}},instruction[31:20]
};
                end
                2'b01:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31:25]
,instruction[11:7]};

                end
                2'b10:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31],ins
truction[7],instruction[30:25],instructio
n[11:8]};
                end
                2'b11:
                begin
                imm_data=
{{52{instruction[31]}},instruction[31],ins
truction[7],instruction[30:25],instructio
n[11:8]};
                end
        endcase
end
endmodule
```

```
always@(posedge clk)
begin
if (PCSrc==1'b0)
begin
        if (ifid_write==1'b1)
        begin
        if (reset==1'b0)
                begin
                PC_out=PC_in;
                Instruction_out=instruction_in;
                end
        end
end
else
begin
                PC_out=64'b0;
                Instruction_out=32'b0;
end
end

always@(reset)
begin
        if (reset==1'b1)
                begin
                PC_out=64'b0;
                Instruction_out=32'b0;
                end
end

endmodule
```

```
module Instruction_memory(
input [63:0] Inst_Adress,
output reg [31:0] Instruction
);

reg [7:0]Array[23:0];
initial
  begin
    Array[0] = 8'b10010011;
    Array[1] = 8'b10000000;
    Array[2] = 8'b01010000;
    Array[3] = 8'b00000000;

    Array[4] = 8'b10010011;
    Array[5] = 8'b00000010;
    Array[6] = 8'b10010000;
    Array[7] = 8'b00000000;
```

```
module MEMWB(
  input clk,
  input reset,
  input [63:0] ReadDatain,
  input [63:0] ALU_Resultin,
  input MemtoReg, RegWrite,
  input [4:0] rd,
  output reg [63:0] ReadDataout,
  output reg [63:0] ALU_Resultout,
  output reg [4:0] rdOut,
  output reg MemtoRegOut, RegWriteOut
);

always @(posedge clk)
begin
  if (reset==1'b0)
  begin
  ReadDataout=ReadDatain;
```

```
      Array[8] = 8'b01100011;
      Array[9] = 8'b10000110;
      Array[10] = 8'b01010000;
      Array[11] = 8'b00000000;

      Array[12] = 8'b00010011;
      Array[13] = 8'b10001010;
      Array[14] = 8'b10100010;
      Array[15] = 8'b00000101;

      Array[16] = 8'b01100011;
      Array[17] = 8'b10010100;
      Array[18] = 8'b01010000;
      Array[19] = 8'b00000000;

      Array[20] = 8'b10110011;
      Array[21] = 8'b10000010;
      Array[22] = 8'b00010010;
      Array[23] = 8'b01000000;
   end
always @ (Inst_Adress)
begin

          Instruction={Array[Inst_Adress+
3],Array[Inst_Adress+2],
Array[Inst_Adress+1],Array[Inst_Adress
]};
end
endmodule
```

```
   ALU_Resultout=ALU_Resultin;
   rdOut=rd;
   MemtoRegOut=MemtoReg;
   RegWriteOut=RegWrite;
   end
end
always@(reset)
begin
   if (reset==1'b1)
   begin
   ReadDataout=64'b0;
   ALU_Resultout=64'b0;
   rdOut=5'b0;
   MemtoRegOut=1'b0;
   RegWriteOut=1'b0;

   end
end
endmodule
```

```
module mux_3(
   input [1:0] sel,
   input [63:0] a,
   input [63:0] b,
   input [63:0] c,
   output reg [63:0] three_muxout
);
always @(*)
begin
   case(sel)
   2'b00: three_muxout=a;
   2'b01: three_muxout=b;
   2'b10: three_muxout=c;
   endcase
end
endmodule
```

```
module parser (
          input [31:0] instruction,
          output reg [6:0] opcode,
          output reg [4:0] rd,
          output reg [2:0] func3,
          output reg [4:0] rs1,
          output reg [4:0] rs2,
          output reg [6:0] func7
);
always@(instruction)
begin
assign opcode=instruction[6:0];
assign rd =instruction[11:7];
assign func3=instruction[14:12];
assign rs1 =instruction[19:15];
assign rs2=instruction[24:20];
assign func7=instruction[31:25];
end
endmodule
```

```
module Program_Counter(
input clk, reset,
input [63:0] Pc_In,
input PCWrite,
output reg [63:0] PC_Out
);

always @(posedge clk or posedge
reset)
begin
        if (reset)
                PC_Out<=0;
        else
                if (PCWrite==1'b1)
                begin
                PC_Out<=Pc_In;
                end

end
endmodule
```

```
module tb();
reg clk;
reg reset;

Riscv rp(.clk(clk), .reset(reset));
initial
begin
clk= 1'b0;
reset=1'b1;
#7 reset =1'b0;
end
always

#5 clk=~clk;

endmodule
```

```
module registerFile(
input [4:0] Rs1, [4:0] Rs2, [4:0] Rd,
input [63:0] WriteData,
input RegWrite,
input clk, reset,
output reg [63:0] ReadData1, reg [63:0]
ReadData2
);

reg [63:0] Array[31:0];
initial
begin
  Array[0]<=64'd0;
  Array[1]<=64'd1;
  Array[2]<=64'd2;
  Array[3]<=64'd3;
  Array[4]<=64'd4;
  Array[5]<=64'd5;
  Array[6]<=64'd6;
  Array[7]<=64'd7;
  Array[8]<=64'd8;
  Array[9]<=64'd9;
  Array[10]<=64'd10;
  Array[11]<=64'd11;
  Array[12]<=64'd12;
  Array[13]<=64'd13;
  Array[14]<=64'd14;
  Array[15]<=64'd15;
```

```
module Riscv(
input clk, reset
);
wire [63:0] b;
assign b= 16'h0000000000000004;

wire [63:0] PC_Out;

wire [63:0] PC_Outidex;
wire [63:0] adder1_out;
wire [63:0] adder2_out;
wire [63:0] adder_out_ex;
wire [63:0] mux1_out;
wire [31:0] Instruction_m;
wire [31:0] Instruction_if;
wire[6:0] opcode;
wire[4:0] rd_parser;
wire[4:0] rd_id;
wire[4:0] rd_mem;
wire[4:0] rd_ex;
wire[2:0] func3;
wire[4:0] rs1;
wire [4:0] rs2;
wire[4:0] rs1_id;
wire [4:0] rs2_id;
wire[6:0] func7;
wire Branch, MemRead, MemtoReg, MemWrite, ALUSrc,
RegWrite;
```

```
  Array[16]<=64'd16;
  Array[17]<=64'd17;
  Array[18]<=64'd18;
  Array[19]<=64'd19;
  Array[20]<=64'd20;
  Array[21]<=64'd21;
  Array[22]<=64'd22;
  Array[23]<=64'd23;
  Array[24]<=64'd24;
  Array[25]<=64'd25;
  Array[26]<=64'd26;
  Array[27]<=64'd27;
  Array[28]<=64'd28;
  Array[29]<=64'd29;
  Array[30]<=64'd30;
  Array[31]<=64'd31;
 end

always@(negedge clk)
begin
        if (RegWrite==1)
        begin
                Array[Rd]<=WriteData;
        end

end


always @(Rs1, Rs2 , reset , Array , clk,
posedge clk)
begin
        if (reset)
        begin
                ReadData1<=64'b0;
                ReadData2<=64'b0;
        end
        else
        begin

        ReadData1<=Array[Rs1];

        ReadData2<=Array[Rs2];
        end
end
endmodule
```

```
wire Branch_mux, MemRead_mux, MemtoReg_mux,
MemWrite_mux, ALUSrc_mux, RegWrite_mux;
wire [1:0] Aluop_mux;
wire [1:0] Aluop;
wire Branch_id, MemRead_id, MemtoReg_id, MemWrite_id,
ALUSrc_id, RegWrite_id;
wire Branch_ex, MemRead_ex, MemtoReg_ex,
MemWrite_ex, RegWrite_ex;
wire MemtoReg_mem, RegWrite_mem;
wire [1:0] Aluop_id;
wire [63:0] imm_data;
wire [63:0] imm_dataOut;
wire [3:0] Operation;
wire [63:0] mux2_out;
wire [63:0] ReadData1;
wire [63:0] ReadData2;
wire[63:0] ReadData1Out;
wire [63:0] ReadData2Out;
wire [63:0] ReadData2_ex;
wire [63:0] Result;
wire [63:0]ALU_Result_out;
wire [63:0] ALU_Resultout_mem;
wire Zero;
wire zero_ex;
wire[63:0] Read_data;
wire [63:0] ReadDataout;
wire [63:0] mux3_out;
wire [63:0] if_pc_out;
wire [3:0] funct;
wire [1:0] Forward_A_in;
wire [1:0] Forward_B_in;
wire [63:0] forward_b_muxout;
wire [63:0] alu_64_a;
wire PcWrite_ctrl;
wire ifid_write_ctrl;
wire mux_sel_ctrl;
wire PCSrc;

wire [63:0] control_idex_out;
Program_Counter p1(.clk(clk),
.reset(reset),.PCWrite(PcWrite_ctrl),
.Pc_In(mux1_out),.PC_Out(PC_Out));
adder a1(.a(PC_Out),.b(b),.out(adder1_out));
assign PCSrc=Branch_ex & zero_ex;
mux
m(.a(adder1_out),.b(adder_out_ex),.sel(PCSrc),.data_out(mu
x1_out));
```

```
//adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));

//adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));

Instruction_memory
i1(.Inst_Adress(PC_Out),.Instruction(Instruction_m));

IFID ifid(.PC_in(PC_Out),
.instruction_in(Instruction_m),
.clk(clk),
.PCSrc(PCSrc),
.ifid_write(ifid_write_ctrl),
.reset(reset),
.PC_out(if_pc_out),
.Instruction_out(Instruction_if)
);

parser
p(.instruction(Instruction_if),.opcode(opcode),.rd(rd_parser),.
func3(func3),.rs1(rs1),.rs2(rs2),.func7(func7));

Control_Unit c1( .Opcode(opcode), .Branch(Branch),
.MemRead(MemRead), .MemtoReg(MemtoReg),
.MemWrite(MemWrite),
.ALUSrc(ALUSrc),.RegWrite(RegWrite),.ALUOp(Aluop));

hazard_unit hu(
.MemRead_idex(MemRead_id),
.rd_idex(rd_id),
.Rs1_ifid(rs1),
.Rs2_ifid(rs2),
.PcWrite(PcWrite_ctrl),
.Ifid_write(ifid_write_ctrl),
.mux_sel(mux_sel_ctrl)
);

mux
mux_control(.a({56'd0,{RegWrite},{MemtoReg},{MemRead},{
MemWrite},{Branch},{ALUSrc},{Aluop}}),
.b(64'd0),
.sel(mux_sel_ctrl),
.data_out(control_idex_out)
);

registerFile r1 (.Rs1(rs1), .Rs2(rs2),  .Rd(rd_mem),
.WriteData(mux3_out), .RegWrite(RegWrite_mem),.clk(clk),
```

```
.reset(reset),.ReadData1(ReadData1),
.ReadData2(ReadData2));

IDG idg(.instruction(Instruction_if),.imm_data(imm_data));

IDEX idex(
.clk(clk),
.reset(reset),
.PCSrc(PCSrc),
.PC_inidex(if_pc_out),
.ReadData1In(ReadData1),
.ReadData2In(ReadData2),
.imm_data(imm_data),
.rs1(rs1),
.rs2(rs2),
.rd(rd_parser),
.inst({Instruction_if[30], Instruction_if[14:12]}),
.MemtoReg(control_idex_out[6]),
.RegWrite(control_idex_out[7]),
.branch(control_idex_out[3]),
.MemRead(control_idex_out[5]),
.MemWrite(control_idex_out[4]),
.ALUSrc(control_idex_out[2]),
.ALUOp(control_idex_out[1:0]),
.PC_Outidex(PC_Outidex),
.ReadData1Out(ReadData1Out),
.ReadData2Out(ReadData2Out),
.imm_dataOut(imm_dataOut),
.funct(funct),
.rdOut(rd_id),
.rs1Out(rs1_id),
.rs2Out(rs2_id),
.MemtoRegOut(MemtoReg_id),
.RegWriteOut(RegWrite_id),
.branchOut(Branch_id),
.MemReadOut(MemRead_id),
.MemWriteOut(MemWrite_id),
.ALUSrcOut(ALUSrc_id),
.ALUOpOut(Aluop_id)
);

ALU_Control
a5(.ALUOp(Aluop_id),.Funct(funct),.Operation(Operation));


alu_64
ALU_64(.a(alu_64_a),.b(forward_b_muxout),.ALUOp(Operati
on),.Result(Result),.ZERO(Zero));
```

```
adder a2(.a(PC_Outidex),.b(imm_dataOut <<
1),.out(adder2_out));
forwarding_unit forward(
   .rs1_idex(rs1_id),
   .rs2_idex(rs2_id),
   .regwrite_memwb(RegWrite_mem),
   .regwrite_exmem(RegWrite_ex),
   .rd_memwb(rd_mem),
   .rd_exmem(rd_ex),
   .Forward_A(Forward_A_in),
   .Forward_B(Forward_B_in)

);

mux_3 firstmux(
   .sel(Forward_A_in),
   .a(ReadData1Out),
   .b(mux3_out),
   .c(ALU_Result_out),
   .three_muxout(alu_64_a)
);

mux_3 secondmux(
   .sel(Forward_B_in),
   .a(ReadData2Out),
   .b(mux3_out),
   .c(ALU_Result_out),
   .three_muxout(forward_b_muxout)
);
EXMEM exmem(
.clk(clk),
.PCSrc(PCSrc),
.reset(reset),
.adder_in(adder2_out),
.ZERO_in(Zero),
.ALU_Result_in(Result),
.ReadData2In(forward_b_muxout),
.rd(rd_id),
.MemtoReg(MemtoReg_id),
.RegWrite(RegWrite_id),
.branch(Branch_id),
.MemRead(MemRead_id),
.MemWrite(MemWrite_id),
.adder_out(adder_out_ex),
 .ALU_Result_out(ALU_Result_out),
 .ReadData2out(ReadData2_ex),
 .rdOut(rd_ex),
```

	.zero(zero_ex), .MemtoRegOut(MemtoReg_ex), .RegWriteOut(RegWrite_ex), .branchOut(Branch_ex), .MemReadOut(MemRead_ex),.MemWriteOut(MemWrite_ex)); Data_memory d1(.mem_addr (ALU_Result_out),.write_data(ReadData2_ex),.clk(clk),.mem_ write(MemWrite_ex),.mem_read(MemRead_ex),.Read_data(Read_data)); MEMWB mem(.clk(clk), .reset(reset), .ReadDatain(Read_data), .ALU_Resultin(ALU_Result_out), .MemtoReg(MemtoReg_ex), .RegWrite(RegWrite_ex), .rd(rd_ex), .ReadDataout(ReadDataout), .ALU_Resultout(ALU_Resultout_mem), .rdOut(rd_mem), .MemtoRegOut(MemtoReg_mem), .RegWriteOut(RegWrite_mem)); mux m3(.a(ALU_Resultout_mem),.b(ReadDataout),.sel(MemtoReg _mem),.data_out(mux3_out)); endmodule