# PROJECT REPORT- RISC V

Nimrah Jawed (nj04707) and Umme Salma us04315)

# **Contents**

Task 1	2
Task 2	12
Task 3	25
Forwarding:	25
Stalling:	40
Flushing	

#### Task 1

We executed task 1 in the following steps:

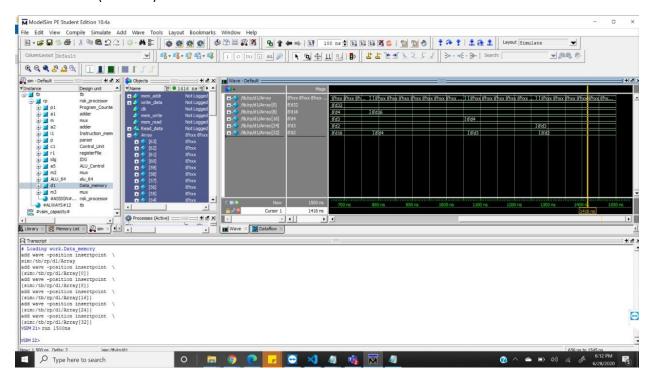
- Edited our bubble sort code such that the shift left happens 3 times. This is because the elements in our data memory are stored as 64 bits in 8 memory locations.
- We removed the memory addresses from load and store instructions and kept a 0 there instead.
- After we dumped the instructions, we changed 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 accommodate 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(
                                                       module mux(
input [63:0] a,
                                                               input[63:0] a, [63:0]b,
input [63:0] b,
                                                               input sel,
output reg [63:0] out);
                                                               output reg[63:0] data_out
                                                       );
                                                       always@(*)
always@(*)
begin
                                                       begin
       out = a+b;
                                                       case(sel)
end
                                                               1'b0: assign data out=a;
                                                               1'b1: assign data_out=b;
endmodule
                                                       endcase
                                                       end
                                                       endmodule
module alu 64(
                                                       module ALU Control(
       input [63:0] a,[63:0] b, [3:0] ALUOp,
                                                       input [1:0] ALUOp,
       output reg [63:0]Result,
                                                       input [3:0] Funct,
       output reg ZERO
                                                       output reg [3:0] Operation
);
                                                       );
always @ (*)
                                                       always @(*)
begin
                                                       begin
  case({ALUOp})
                                                               case(ALUOp)
       4'b0111: //slli
                                                                       2'b00:
       begin
                                                                       begin
               Result= a<<b;
                                                                       if (Funct==4'b0001) //slli
        end
                                                                       begin
        4'b0101://bne
                                                                               Operation=4'b0111;
       begin
                                                                       end
               Result= a-b;
                                                                       else
               if(Result==0)
                                                                       begin
                                                                               Operation=4'b0010;
                       begin
                       ZERO=0;
                                                                       end
                                                                       end
                       end
               else
                                                                       2'b01:
                       ZERO=1;
       end
                                                                       begin
       4'b1010:
                                                                       if (Funct==4'b0001)//bne
       begin
                                                                       begin
               Result = (a < b)?1:0;
                                                                               Operation=4'b0101;
               ZERO = Result;
                                                                       end
                                                                       else if (Funct==4'b0100)//blt
        end
   4'b0000:
                                                                       begin
                                                                               Operation=4'b1010;
       begin
        Result = a&b;
                                                                       end
                                                                       else if (Funct==4'b0000) //beq
       end
```

```
4'b0001 :
                                                                       begin
       begin
                                                                               Operation=4'b0110;
         Result = a|b;
                                                                       end
        end
                                                                       end
   4'b0010 :
                                                                       2'b10:
       begin
         Result = a+b;
                                                                       begin
        end
                                                                       if (Funct==4'b0000)
                                                                       begin
        4'b0110://beq
                                                                               Operation=4'b0010;
        begin
                                                                       end
        Result = a-b;
                                                                       else if (Funct==4'b1000)
       ZERO = Result?0:1;
                                                                       begin
                                                                               Operation=4'b0110;
        end
                                                                       end
       default : Result = ^{\sim}(a|b);
                                                                       else if (Funct==4'b0111)
  endcase
                                                                       begin
                                                                               Operation=4'b0000;
                                                                       end
end
                                                                       else if (Funct==4'b0110)
endmodule
                                                                       begin
                                                                               Operation=4'b0001;
                                                                       end
                                                                       end
                                                               endcase
                                                       end
                                                       endmodule
module Program Counter(
                                                       module parser (
input clk, reset,
                                                               input [31:0] instruction,
                                                               output reg [6:0] opcode,
input [63:0] Pc In,
output reg [63:0] PC Out
                                                               output reg [4:0] rd,
);
                                                               output reg [2:0] func3,
                                                               output reg [4:0] rs1,
always @(posedge clk or posedge reset)
                                                               output reg [4:0] rs2,
                                                               output reg [6:0] func7
begin
       if (reset)
               PC_Out<=0;
                                                       always@(instruction)
                                                       begin
        else
               PC Out<=Pc In;
                                                       assign opcode=instruction[6:0];
                                                       assign rd =instruction[11:7];
                                                       assign func3=instruction[14:12];
end
                                                       assign rs1 =instruction[19:15];
endmodule
                                                       assign rs2=instruction[24:20];
                                                       assign func7=instruction[31:25];
                                                       end
                                                       endmodule
module Instruction memory(
                                                       module Data memory(
```

```
input [63:0] Inst Adress,
                                                      input [63:0]mem addr,
output reg [31:0] Instruction
                                                      input [63:0] write data,
                                                      input clk,
);
reg [7:0]Array[95:0];
                                                      input mem_write,
initial
                                                      input mem read,
                                                      output reg[63:0] Read data
  begin
    Array[0] = 8'b00010011;
                                                      );
    Array[1] = 8'b00001011;
    Array[2] = 8'b000000000;
                                                      reg [7:0] Array [63:0];
    Array[3] = 8'b000000000;
    Array[4] = 8'b00010011;
                                                      initial
    Array[5] = 8'b00000101;
                                                      begin
    Array[6] = 8'b01010000;
                                                      Array[0]=8'b00000100;
    Array[7] = 8'b000000000;
                                                      Array[1]=8'b00000000;
    Array[8] = 8'b00010011;
                                                      Array[2]=8'b00000000;
    Array[9] = 8'b00000001;
                                                      Array[3]=8'b00000000;
    Array[10] = 8'b000000000;
                                                      Array[4]=8'b00000000;
    Array[11] = 8'b00000000;
                                                      Array[5]=8'b00000000;
    Array[12] = 8'b10010011;
                                                      Array[6]=8'b00000000;
    Array[13] = 8'b00000001;
                                                      Array[7]=8'b00000000;
    Array[14] = 8'b000000000;
    Array[15] = 8'b000000000;
                                                      Array[8]=8'b00010000;
    Array[16] = 8'b10110011;
                                                      Array[9]=8'b00000000;
    Array[17] = 8'b00001011;
                                                      Array[10]=8'b00000000;
    Array[18] = 8'b01100000;
                                                      Array[11]=8'b00000000;
    Array[19] = 8'b00000001;
                                                      Array[12]=8'b00000000;
    Array[20] = 8'b01100011;
                                                      Array[13]=8'b00000000;
    Array[21] = 8'b00000110;
                                                      Array[14]=8'b00000000;
    Array[22] = 8'b10101011;
                                                      Array[15]=8'b00000000;
    Array[23] = 8'b00000100;
    Array[24] = 8'b00000011;
                                                      Array[16]=8'b00000011;
                                                      Array[17]=8'b00000000;
    Array[25] = 8'b00110011;
    Array[26] = 8'b00000001;
                                                      Array[18]=8'b00000000;
    Array[27] = 8'b000000000;
                                                      Array[19]=8'b00000000;
    Array[28] = 8'b00000011;
                                                      Array[20]=8'b00000000;
    Array[29] = 8'b10110010;
                                                      Array[21]=8'b00000000;
    Array[30] = 8'b00000001;
                                                      Array[22]=8'b00000000;
    Array[31] = 8'b000000000;
                                                      Array[23]=8'b00000000;
    Array[32] = 8'b01100011;
    Array[33] = 8'b01001110;
                                                      Array[24]=8'b00000010;
    Array[34] = 8'b01000011;
                                                      Array[25]=8'b00000000;
    Array[35] = 8'b000000000;
                                                      Array[26]=8'b00000000;
                                                      Array[27]=8'b00000000;
    Array[36] = 8'b10010011;
    Array[37] = 8'b10001011;
                                                      Array[28]=8'b00000000;
    Array[38] = 8'b00011011;
                                                      Array[29]=8'b000000000:
    Array[39] = 8'b000000000;
                                                      Array[30]=8'b00000000;
    Array[40] = 8'b00010011;
                                                      Array[31]=8'b00000000;
    Array[41] = 8'b00010001;
```

```
Array[42] = 8'b00111011;
                                                 Array[32]=8'b00100000;
Array[43] = 8'b000000000;
                                                 Array[33]=8'b00000000;
Array[44] = 8'b10010011;
                                                 Array[34]=8'b00000000;
                                                 Array[35]=8'b00000000;
Array[45] = 8'b10010001;
Array[46] = 8'b00111011;
                                                 Array[36]=8'b00000000;
Array[47] = 8'b000000000;
                                                 Array[37]=8'b00000000;
Array[48] = 8'b11100011;
                                                 Array[38]=8'b00000000;
Array[49] = 8'b10010100;
                                                 Array[39]=8'b00000000;
Array[50] = 8'b10101011;
Array[51] = 8'b11111110;
Array[52] = 8'b00010011;
Array[53] = 8'b00001011;
                                                 end
Array[54] = 8'b00011011;
Array[55] = 8'b000000000;
                                                 always @(*)
Array[56] = 8'b11100011;
                                                 begin
                                                         if (mem_read)
Array[57] = 8'b10001100;
Array[58] = 8'b10101011;
                                                         begin
Array[59] = 8'b11111100;
Array[60] = 8'b10110011;
                                                         Read data={Array[mem addr+7],Array[
Array[61] = 8'b00000010;
                                                 mem addr+6],
Array[62] = 8'b01100000;
                                                 Array[mem addr+5], Array[mem addr+4], Array
                                                 [mem addr+3], Array[mem addr+2],
Array[63] = 8'b000000000;
Array[64] = 8'b00100011;
                                                 Array[mem addr+1],Array[mem addr]};
Array[65] = 8'b00110000;
Array[66] = 8'b01000001;
Array[67] = 8'b000000000;
                                                 end
Array[68] = 8'b00100011;
Array[69] = 8'b10110000;
                                                 always @(posedge clk)
Array[70] = 8'b01010001;
                                                 begin
Array[71] = 8'b000000000;
                                                         if (mem write)
Array[72] = 8'b10010011;
                                                         begin
Array[73] = 8'b10001011;
                                                        Array[mem addr]=write data[7:0];
Array[74] = 8'b00011011;
Array[75] = 8'b000000000;
Array[76] = 8'b00010011;
                                                        Array[mem addr+1]=write data[15:8];
Array[77] = 8'b00010001;
Array[78] = 8'b00111011;
                                                        Array[mem addr+2]=write data[23:16]
Array[79] = 8'b000000000;
Array[80] = 8'b10010011;
Array[81] = 8'b10010001;
                                                        Array[mem_addr+3]=write_data[31:24]
Array[82] = 8'b00111011;
Array[83] = 8'b000000000;
Array[84] = 8'b11100011;
                                                        Array[mem addr+4]=write data[39:32]
Array[85] = 8'b10010010;
Array[86] = 8'b10101011;
                                                        Array[mem addr+5]=write data[47:40]
Array[87] = 8'b11111100;
Array[88] = 8'b00010011;
Array[89] = 8'b00001011;
```

```
Array[90] = 8'b00011011;
    Array[91] = 8'b000000000;
                                                               Array[mem addr+6]=write data[55:48]
    Array[92] = 8'b11100011;
    Array[93] = 8'b10001010;
                                                               Array[mem addr+7]=write data[63:56]
    Array[94] = 8'b10101011;
    Array[95] = 8'b11111010;
                                                               end
  end
always @ (Inst Adress)
                                                       end
begin
                                                       endmodule
        Instruction={Array[Inst Adress+3],Array[Inst
Adress+2], Array[Inst Adress+1], Array[Inst Adress]};
end
endmodule
                                                       module tb();
module risk processor(
input clk, reset
                                                       reg clk;
                                                       reg reset;
wire [63:0] b;
assign b= 16'h00000000000000004;
                                                       risk processor rp(.clk(clk), .reset(reset));
                                                       initial
wire [63:0] PC Out;
                                                       begin
wire [63:0] adder1 out;
                                                       clk= 1'b0;
wire [63:0] adder2 out;
                                                       reset=1'b1;
wire [63:0] mux1_out;
                                                       #7 reset =1'b0;
wire [31:0] Instruction m;
                                                       end
wire[6:0] opcode;
                                                       always
wire[4:0] rd;
wire[2:0] func3;
                                                       #5 clk=~clk;
wire[4:0] rs1;
                                                       endmodule
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));
```

```
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(
                                                      module registerFile(
input [6:0] Opcode,
                                                      input [4:0] Rs1, [4:0] Rs2, [4:0] Rd,
output reg Branch, MemRead, MemtoReg,
                                                      input [63:0] WriteData,
MemWrite, ALUSrc, RegWrite,
                                                      input RegWrite,
output reg [1:0] ALUOp
                                                      input clk, reset,
                                                      output reg [63:0] ReadData1, reg [63:0]
);
                                                      ReadData2
                                                      );
always @(*)
begin
                                                      reg [63:0] Array[31:0];
       case(Opcode)
                                                      initial
               7'b0110011:
                                                      begin
                                                        Array[0]=64'd0;
               begin
               ALUSrc=1'b0;
                                                        Array[1]=64'd1;
               MemtoReg=1'b0;
                                                        Array[2]=64'd2;
               RegWrite=1'b1;
                                                        Array[3]=64'd3;
               MemRead=1'b0;
                                                        Array[4]=64'd4;
               MemWrite=1'b0;
                                                        Array[5]=64'd5;
               Branch=1'b0;
                                                        Array[6]=64'd6;
               ALUOp=2'b10;
                                                        Array[7]=64'd7;
               end
                                                        Array[8]=64'd8;
                                                        Array[9]=64'd9;
               7'b0000011:
                                                        Array[10]=64'd10;
               begin
                                                        Array[11]=64'd11;
               ALUSrc=1'b1;
                                                        Array[12]=64'd12;
```

```
MemtoReg=1'b1;
                                                      Array[13]=64'd13;
              RegWrite=1'b1;
                                                     Array[14]=64'd14;
              MemRead=1'b1;
                                                     Array[15]=64'd15;
              MemWrite=1'b0;
                                                     Array[16]=64'd16;
              Branch=1'b0;
                                                     Array[17]=64'd17;
              ALUOp=2'b00;
                                                     Array[18]=64'd18;
              end
                                                     Array[19]=64'd19;
                                                     Array[20]=64'd20;
              7'b0100011:
                                                     Array[21]=64'd21;
              begin
                                                     Array[22]=64'd22;
              ALUSrc=1'b1;
                                                     Array[23]=64'd23;
              MemtoReg=1'bx;
                                                     Array[24]=64'd24;
              RegWrite=1'b0;
                                                     Array[25]=64'd25;
              MemRead=1'b0;
                                                     Array[26]=64'd26;
              MemWrite=1'b1;
                                                     Array[27]=64'd27;
              Branch=1'b0;
                                                     Array[28]=64'd28;
              ALUOp=2'b00;
                                                     Array[29]=64'd29;
              end
                                                     Array[30]=64'd30;
                                                     Array[31]=64'd31;
              7'b1100011:
                                                    end
              begin
              ALUSrc=1'b0;
                                                    always@(posedge clk)
              MemtoReg=1'bx;
                                                    begin
              RegWrite=1'b0;
                                                           if (RegWrite==1)
              MemRead=1'b0;
                                                           begin
              MemWrite= 1'b0:
                                                                   Array[Rd]=WriteData;
              Branch=1'b1;
                                                           end
              ALUOp=2'b01;
              end
                                                    end
              7'b0010011:
                                                    always @(Rs1, Rs2, reset, Array, clk)
              begin
              ALUSrc=1'b1;
                                                    begin
              MemtoReg=1'b0;
                                                           if (reset)
              RegWrite=1'b1;
                                                           begin
              MemRead=1'b1;
                                                                   ReadData1<=64'b0;
              MemWrite= 1'b0:
                                                                   ReadData2<=64'b0;
              Branch=1'b0;
                                                           end
              ALUOp=2'b00;
                                                           else
              end
                                                           begin
                                                                   ReadData1<=Array[Rs1];</pre>
                                                                   ReadData2<=Array[Rs2];</pre>
       endcase
                                                           end
end
                                                    end
endmodule
                                                    endmodule
```

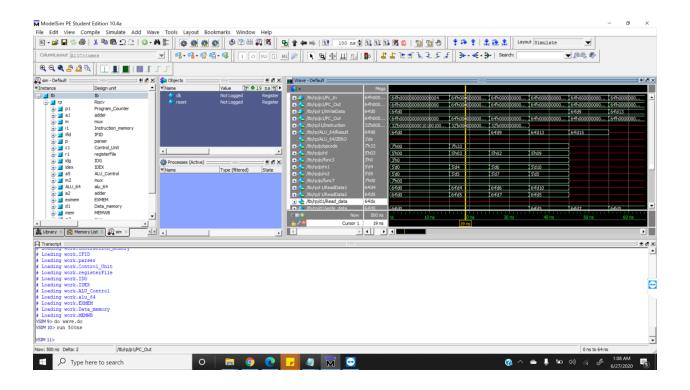
# Task 2

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

```
);
                                                      reg reset;
wire [63:0] b;
assign b= 16'h00000000000000004;
                                                      Riscv rp(.clk(clk), .reset(reset));
wire [63:0] PC_Out;
                                                      initial
wire [63:0] PC Outidex;
                                                     begin
wire [63:0] adder1 out;
                                                     clk= 1'b0;
wire [63:0] adder2_out;
                                                      reset=1'b1;
wire [63:0] adder out ex;
                                                     #7 reset =1'b0;
wire [63:0] mux1 out;
                                                     end
wire [31:0] Instruction_m;
                                                     always
wire [31:0] Instruction if;
wire[6:0] opcode;
                                                     #5 clk=~clk;
wire[4:0] rd parser;
wire[4:0] rd id;
                                                     endmodule
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;
```

```
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(Opera
tio));
mux
m2(.a(ReadData2Out),.b(imm_dataOut),.sel(ALUSrc
id),.data out(mux2 out));
alu 64
ALU 64(.a(ReadData1Out),.b(mux2 out),.ALUOp(Op
eration),.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),.Mem
Write(MemWrite_id),
.adder_out(adder_out_ex),.ALU_Result_out(ALU_Re
sult out), .ReadData2out(ReadData2 ex),
.rdOut(rd ex),
.zero(zero ex),.MemtoRegOut(MemtoReg ex),.Reg
WriteOut(RegWrite ex),.branchOut(Branch ex),
.MemReadOut(MemRead ex),.MemWriteOut(Mem
Write ex));
Data memory d1(.mem addr
(ALU Result out),.write data(ReadData2 ex),.clk(cl
k),.mem_write(MemWrite_ex),.mem_read(MemRea
d 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));
m3(.a(ALU Resultout mem),.b(ReadDataout),.sel(M
emtoReg_mem),.data_out(mux3_out));
endmodule
module adder(
                                                    module mux(
input [63:0] a,
                                                            input[63:0] a, [63:0]b,
input [63:0] b,
                                                            input sel,
output reg [63:0] out);
                                                            output reg[63:0] data out
                                                    );
always@(*)
                                                    always@(*)
begin
                                                    begin
       out = a+b;
                                                    case(sel)
                                                            1'b0: assign data_out=a;
end
                                                            1'b1: assign data out=b;
endmodule
                                                    endcase
                                                    end
                                                    endmodule
module alu_64(
                                                    module ALU_Control(
                                                    input [1:0] ALUOp,
       input [63:0] a,[63:0] b, [3:0] ALUOp,
       output reg [63:0]Result,
                                                    input [3:0] Funct,
       output reg ZERO
                                                    output reg [3:0] Operation
                                                    );
);
always @ (*)
                                                    always @(*)
begin
                                                    begin
  case({ALUOp})
                                                            case(ALUOp)
                                                                    2'b00:
       4'b0111: //slli
                                                                    begin
       begin
                                                                    if (Funct==4'b0001) //slli
               Result= a<<b;
                                                                    begin
                                                                           Operation=4'b0111;
       end
                                                                    end
       4'b0101://bne
                                                                    else
       begin
                                                                    begin
               Result= a-b;
                                                                           Operation=4'b0010;
               if(Result==0)
                                                                    end
                       begin
                                                                    end
```

ZERO=0;		
end	2'b01:	
else	begin	
ZERO=1;	if (Funct==4'b0001)//bne	
end end	begin	
4'b1010:	Operation=4'b0101;	
	•	
begin	end else if (Funct==4'b0100)//blt	
Result = (a <b)?1:0;< td=""><td>, , , , , , , , , , , , , , , , , , , ,</td></b)?1:0;<>	, , , , , , , , , , , , , , , , , , , ,	
ZERO = Result;	begin	
end	Operation=4'b1010;	
411.0000	end	
4'b0000:	else if (Funct==4'b0000) //beq	
begin	begin	
Result = a&b	Operation=4'b0110;	
end	end	
4'b0001 :		
begin	end	
Result = a b;		
end	2'b10:	
	begin	
4'b0010 :	if (Funct==4'b0000)	
begin	begin	
Result = a+b;	Operation=4'b0010;	
end	end	
	else if (Funct==4'b1000)	
4'b0110://beq	begin	
begin	Operation=4'b0110;	
Result = a-b;	end	
ZERO = Result?0:1;	else if (Funct==4'b0111)	
end	begin	
Cita	Operation=4'b0000;	
default : Result = ~(a b);	end	
endcase	else if (Funct==4'b0110)	
Eliacase	,	
	begin	
and	Operation=4'b0001;	
end	end	
endmodule	end	
	endcase	
	end	
	endmodule	
module Control_Unit(	module registerFile(	
input [6:0] Opcode,	input [4:0] Rs1, [4:0] Rs2, [4:0] Rd,	
output reg Branch, MemRead, MemtoReg,	input [63:0] WriteData,	
MemWrite, ALUSrc, RegWrite,	input RegWrite,	
output reg [1:0] ALUOp	input clk, reset,	
);	output reg [63:0] ReadData1, reg [63:0]	
	ReadData2	
	l	

```
);
always @(*)
                                                   reg [63:0] Array[31:0];
begin
       case(Opcode)
                                                   initial
              7'b0110011:
                                                   begin
              begin
                                                     Array[0]=64'd0;
              ALUSrc=1'b0;
                                                     Array[1]=64'd1;
                                                     Array[2]=64'd2;
              MemtoReg=1'b0;
                                                     Array[3]=64'd3;
              RegWrite=1'b1;
              MemRead=1'b0;
                                                     Array[4]=64'd4;
              MemWrite=1'b0;
                                                     Array[5]=64'd5;
              Branch=1'b0;
                                                     Array[6]=64'd6;
              ALUOp=2'b10;
                                                     Array[7]=64'd7;
              end
                                                     Array[8]=64'd8;
                                                     Array[9]=64'd9;
              7'b0000011:
                                                     Array[10]=64'd10;
              begin
                                                     Array[11]=64'd11;
              ALUSrc=1'b1;
                                                     Array[12]=64'd12;
              MemtoReg=1'b1;
                                                     Array[13]=64'd13;
              RegWrite=1'b1;
                                                     Array[14]=64'd14;
              MemRead=1'b1;
                                                     Array[15]=64'd15;
              MemWrite=1'b0;
                                                     Array[16]=64'd16;
              Branch=1'b0;
                                                     Array[17]=64'd17;
              ALUOp=2'b00;
                                                     Array[18]=64'd18;
              end
                                                     Array[19]=64'd19;
                                                     Array[20]=64'd20;
              7'b0100011:
                                                     Array[21]=64'd21;
              begin
                                                     Array[22]=64'd22;
              ALUSrc=1'b1;
                                                     Array[23]=64'd23;
              MemtoReg=1'bx;
                                                     Array[24]=64'd24;
              RegWrite=1'b0;
                                                     Array[25]=64'd25;
              MemRead=1'b0;
                                                     Array[26]=64'd26;
              MemWrite=1'b1;
                                                     Array[27]=64'd27;
              Branch=1'b0;
                                                     Array[28]=64'd28;
              ALUOp=2'b00;
                                                     Array[29]=64'd29;
              end
                                                     Array[30]=64'd30;
                                                     Array[31]=64'd31;
              7'b1100011:
                                                    end
              begin
              ALUSrc=1'b0;
                                                   always@(posedge clk)
              MemtoReg=1'bx;
                                                   begin
              RegWrite=1'b0;
                                                           if (RegWrite==1)
              MemRead=1'b0;
                                                           begin
              MemWrite= 1'b0;
                                                                  Array[Rd]=WriteData;
              Branch=1'b1;
                                                           end
              ALUOp=2'b01;
              end
                                                   end
```

```
7'b0010011:
               begin
                                                    always @(Rs1, Rs2, reset, Array, clk)
               ALUSrc=1'b1;
                                                    begin
               MemtoReg=1'b0;
                                                            if (reset)
               RegWrite=1'b1;
                                                            begin
               MemRead=1'b1;
                                                                   ReadData1<=64'b0;
               MemWrite= 1'b0;
                                                                   ReadData2<=64'b0;
               Branch=1'b0;
                                                            end
               ALUOp=2'b00;
                                                            else
               end
                                                            begin
                                                                   ReadData1<=Array[Rs1];
                                                                   ReadData2<=Array[Rs2];
       endcase
                                                            end
end
                                                    end
endmodule
                                                    endmodule
module Data memory(
                                                    module Instruction memory(
input [63:0]mem addr,
                                                    input [63:0] Inst Adress,
input [63:0] write data,
                                                    output reg [31:0] Instruction
input clk,
                                                    );
input mem write,
input mem read,
                                                    reg [7:0]Array[11:0];
output reg[63:0] Read_data
                                                     initial
                                                      begin
);
reg [7:0] Array [63:0];
initial
begin
                                                        Array[0] =8'b10110011;
Array[0]=8'b00000100;
                                                        Array[1] =8'b00000001;
                                                        Array[2] =8'b01010010;
Array[1]=8'b00000000;
                                                        Array[3] =8'b00000000;
Array[2]=8'b00000000;
Array[3]=8'b00000000;
Array[4]=8'b00000000;
                                                        Array[4] =8'b00110011;
Array[5]=8'b00000000;
                                                        Array[5] = 8'b00000001;
Array[6]=8'b00000000;
                                                        Array[6] =8'b01110011;
Array[7]=8'b00000000;
                                                        Array[7] =8'b00000000;
Array[8]=8'b00010000;
Array[9]=8'b00000000;
                                                        Array[8] =8'b10110011;
Array[10]=8'b00000000;
                                                        Array[9] =8'b00000100;
Array[11]=8'b00000000;
                                                        Array[10] =8'b01010101;
Array[12]=8'b00000000;
                                                        Array[11] =8'b00000000;
Array[13]=8'b00000000;
Array[14]=8'b00000000;
                                                      end
Array[15]=8'b00000000;
Array[16]=8'b00000011;
Array[17]=8'b00000000;
Array[18]=8'b00000000;
                                                    always @ (Inst_Adress)
Array[19]=8'b00000000;
                                                    begin
Array[20]=8'b00000000;
Array[21]=8'b00000000;
```

```
Array[22]=8'b00000000;
                                                          Instruction={Array[Inst Adress+3],Array[
Array[23]=8'b00000000;
                                                  Inst Adress+2],
Array[24]=8'b00000010;
                                                  Array[Inst Adress+1],Array[Inst Adress]};
Array[25]=8'b00000000;
                                                  end
                                                  endmodule
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];
```

```
Array[mem addr+7]=write data[63:56];
       end
end
endmodule
module EXMEM(
                                                    module IDEX(
input clk,
                                                    input clk,
input reset,
                                                    input reset,
input [63:0] adder in,
                                                    input [63:0] PC inidex,
                                                    input [63:0] ReadData1In,
input ZERO_in,
input [63:0] ALU_Result_in,
                                                    input [63:0] ReadData2In,
input [63:0] ReadData2In,
                                                    input [63:0] imm data,
input [4:0] rd,
                                                    input [4:0] rs1,
input MemtoReg,RegWrite, branch,
                                                    input [4:0] rs2,
MemRead, MemWrite,
                                                    input [4:0] rd,
output reg [63:0] adder out,
                                                    input [3:0] inst,
output reg [63:0] ALU Result out,
                                                    input MemtoReg,
output reg [63:0] ReadData2out,
                                                    input RegWrite,
output reg [4:0] rdOut,
                                                    input branch,
output reg zero,
                                                    input MemRead,
output reg MemtoRegOut, RegWriteOut, branchOut,
                                                    input MemWrite,
MemReadOut, MemWriteOut
                                                    input ALUSrc,
                                                    input [1:0] ALUOp,
);
                                                    output reg [63:0] PC_Outidex,
                                                    output reg [63:0] ReadData1Out,
always @(posedge clk)
begin
                                                    output reg [63:0] ReadData2Out,
  if (reset==1'b0)
                                                    output reg [63:0] imm_dataOut,
  begin
                                                    output reg [3:0] funct,
  adder out= adder in;
                                                    output reg [4:0] rdOut,
  ALU Result out= ALU Result in;
                                                    output reg [4:0] rs1Out,
  ReadData2out= ReadData2In;
                                                    output reg [4:0] rs2Out,
  rdOut=rd;
                                                    output reg MemtoRegOut,
  zero=ZERO_in;
                                                    RegWriteOut,branchOut,MemReadOut,MemWri
  MemtoRegOut=MemtoReg;
                                                    teOut,ALUSrcOut,
  RegWriteOut=RegWrite;
                                                    output reg [1:0] ALUOpOut
  branchOut=branch;
                                                    );
  MemReadOut= MemRead;
  MemWriteOut=MemWrite;
                                                    always @(posedge clk)
                                                    begin
  end
                                                           if (reset==1'b0)
                                                                   begin
end
always@(reset)
                                                                           PC Outidex=PC inidex;
begin
  if (reset==1'b1)
                                                           ReadData1Out=ReadData1In;
  begin
  adder out= 64'b0;
                                                           ReadData2Out=ReadData2In;
 ALU_Result_out= 64'b0;
  ReadData2out= 64'b0;
                                                           imm dataOut=imm data;
```

```
rdOut=5'b0;
                                                                        funct=inst;
 zero=1'b0;
                                                                        rdOut=rd;
 MemtoRegOut=1'b0;
                                                                        rs1Out=rs1;
 RegWriteOut=1'b0;
                                                                        rs2Out=rs2;
 branchOut=1'b0;
 MemReadOut= 1'b0;
                                                         MemtoRegOut=MemtoReg;
 MemWriteOut=1'b0;
                                                                        RegWriteOut=RegWrite;
 end
                                                                        branchOut=branch;
end
                                                                        MemReadOut=
endmodule
                                                  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(
                                                  module IFID(
input [31:0] instruction,
                                                  input [63:0] PC in,
output reg [63:0] imm_data
                                                  input [31:0] instruction_in,
                                                  input clk,
);
always @(*)
                                                  input reset,
                                                  output reg [63:0] PC_out,
begin
       case(instruction[6:5])
                                                  output reg [31:0] Instruction_out
              2'b00:
```

```
begin
               imm_data=
                                                     always@(posedge clk)
{{52{instruction[31]}},instruction[31:20]};
                                                     begin
               end
                                                             if (reset==1'b0)
               2'b01:
                                                                     begin
               begin
                                                                     PC_out=PC_in;
               imm_data=
                                                                     Instruction_out=instruction_in;
{{52{instruction[31]}},instruction[31:25],instruction[
                                                                     end
11:7]};
                                                     end
                                                     always@(reset)
               end
                                                     begin
               2'b10:
                                                             if (reset==1'b1)
               begin
                                                                     begin
               imm data=
                                                                     PC out=64'b0;
{{52{instruction[31]}},instruction[31],instruction[7],i
                                                                     Instruction_out=32'b0;
nstruction[30:25],instruction[11:8]};
                                                                     end
               end
                                                     end
               2'b11:
                                                     endmodule
               begin
               imm data=
{{52{instruction[31]}},instruction[31],instruction[7],i
nstruction[30:25],instruction[11:8]};
               end
       endcase
end
endmodule
module MEMWB(
                                                     module Program_Counter(
  input clk,
                                                     input clk, reset,
  input reset,
                                                     input [63:0] Pc In,
  input [63:0] ReadDatain,
                                                     output reg [63:0] PC Out
  input [63:0] ALU Resultin,
                                                     );
  input MemtoReg, RegWrite,
  input [4:0] rd,
                                                     always @(posedge clk or posedge reset)
  output reg [63:0] ReadDataout,
                                                     begin
  output reg [63:0] ALU_Resultout,
                                                             if (reset)
  output reg [4:0] rdOut,
                                                                     PC_Out<=0;
  output reg MemtoRegOut, RegWriteOut
                                                             else
);
                                                                     PC_Out<=Pc_In;
always @(posedge clk)
begin
                                                     end
  if (reset==1'b0)
                                                     endmodule
  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 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
```

### Task 3

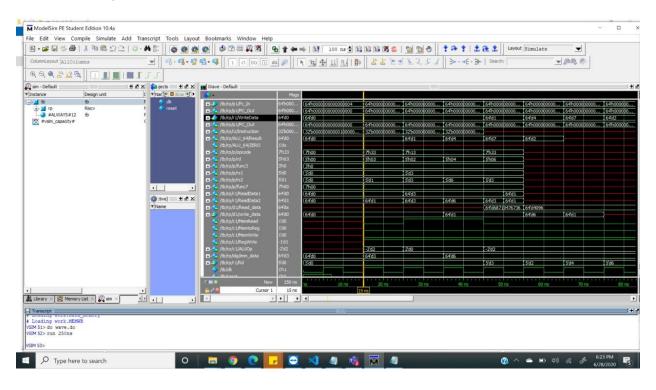
## Forwarding:

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 3<sup>rd</sup> 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(
                                                module mux(
input [63:0] a,
                                                        input[63:0] a, [63:0]b,
input [63:0] b,
                                                        input sel,
output reg [63:0] out);
                                                        output reg[63:0] data_out
                                                );
always@(*)
                                                always@(*)
                                                begin
begin
                                                case(sel)
       out = a+b;
end
                                                        1'b0: assign data out=a;
                                                        1'b1: assign data out=b;
endmodule
                                                endcase
                                                end
                                                endmodule
module alu 64(
                                                module ALU Control(
       input [63:0] a,[63:0] b, [3:0] ALUOp,
                                                input [1:0] ALUOp,
       output reg [63:0]Result,
                                                input [3:0] Funct,
       output reg ZERO
                                                output reg [3:0] Operation
);
always @ (*)
                                                always @(*)
begin
                                                begin
  case({ALUOp})
                                                        case(ALUOp)
                                                                2'b00:
       4'b0111: //slli
                                                                begin
                                                                if (Funct==4'b0001) //slli
       begin
               Result= a<<b;
                                                                begin
                                                                       Operation=4'b0111;
       end
                                                                end
       4'b0101://bne
                                                                else
       begin
                                                                begin
               Result= a-b;
                                                                       Operation=4'b0010;
               if(Result==0)
                                                                end
                       begin
                                                                end
                       ZERO=0;
                       end
                                                                2'b01:
               else
                                                                begin
                                                                if (Funct==4'b0001)//bne
                       ZERO=1;
       end
                                                                begin
       4'b1010:
                                                                       Operation=4'b0101;
       begin
                                                                end
               Result = (a < b)?1:0;
                                                                else if (Funct==4'b0100)//blt
               ZERO = Result;
                                                                begin
        end
                                                                       Operation=4'b1010;
                                                                end
   4'b0000:
                                                                else if (Funct==4'b0000) //beq
       begin
                                                                begin
         Result = a&b;
                                                                       Operation=4'b0110;
       end
                                                                end
```

```
4'b0001 :
       begin
                                                              end
        Result = a|b;
       end
                                                              2'b10:
                                                              begin
   4'b0010 :
                                                              if (Funct==4'b0000)
       begin
                                                              begin
        Result = a+b;
                                                                     Operation=4'b0010;
       end
                                                              end
                                                              else if (Funct==4'b1000)
       4'b0110://beq
                                                              begin
                                                                     Operation=4'b0110;
       begin
       Result = a-b;
                                                              end
                                                              else if (Funct==4'b0111)
       ZERO = Result?0:1;
       end
                                                              begin
                                                                     Operation=4'b0000;
       default : Result = ^{\sim}(a|b);
                                                              end
  endcase
                                                              else if (Funct==4'b0110)
                                                              begin
                                                                     Operation=4'b0001;
end
                                                              end
endmodule
                                                              end
                                                      endcase
                                              end
                                              endmodule
module Control Unit(
                                              module Data memory(
input [6:0] Opcode,
                                              input [63:0]mem addr,
output reg Branch, MemRead, MemtoReg,
                                              input [63:0] write data,
MemWrite, ALUSrc, RegWrite,
                                              input clk,
output reg [1:0] ALUOp
                                              input mem write,
                                              input mem read,
);
                                              output reg[63:0] Read_data
                                              );
always @(*)
                                              reg [7:0] Array [63:0];
begin
       case(Opcode)
               7'b0110011:
                                              initial
               begin
                                              begin
               ALUSrc=1'b0;
                                              Array[0]=8'b00000100;
               MemtoReg=1'b0;
                                              Array[1]=8'b00000000;
               RegWrite=1'b1;
                                              Array[2]=8'b00000000;
               MemRead=1'b0;
                                              Array[3]=8'b00000000;
                                              Array[4]=8'b00000000;
               MemWrite=1'b0;
                                              Array[5]=8'b00000000;
               Branch=1'b0;
               ALUOp=2'b10;
                                              Array[6]=8'b00000000;
               end
                                              Array[7]=8'b00000000;
                                              Array[8]=8'b00010000;
```

```
7'b0000011:
                                             Array[9]=8'b00000000;
              begin
                                             Array[10]=8'b00000000;
                                             Array[11]=8'b00000000;
              ALUSrc=1'b1;
                                            Array[12]=8'b00000000;
              MemtoReg=1'b1;
              RegWrite=1'b1;
                                             Array[13]=8'b00000000;
              MemRead=1'b1;
                                             Array[14]=8'b00000000;
                                             Array[15]=8'b00000000;
              MemWrite=1'b0;
              Branch=1'b0;
                                             Array[16]=8'b00000011;
              ALUOp=2'b00;
                                             Array[17]=8'b00000000;
              end
                                             Array[18]=8'b00000000;
                                             Array[19]=8'b00000000;
              7'b0100011:
                                             Array[20]=8'b00000000;
              begin
                                             Array[21]=8'b00000000;
              ALUSrc=1'b1;
                                             Array[22]=8'b00000000;
                                            Array[23]=8'b00000000;
              MemtoReg=1'bx;
              RegWrite=1'b0;
                                             Array[24]=8'b00000010;
              MemRead=1'b0;
                                             Array[25]=8'b00000000;
              MemWrite=1'b1;
                                             Array[26]=8'b00000000;
              Branch=1'b0;
                                             Array[27]=8'b00000000;
              ALUOp=2'b00;
                                             Array[28]=8'b00000000;
              end
                                             Array[29]=8'b00000000;
                                             Array[30]=8'b00000000;
              7'b1100011:
                                             Array[31]=8'b00000000;
              begin
                                             Array[32]=8'b00100000;
              ALUSrc=1'b0;
                                             Array[33]=8'b00000000;
              MemtoReg=1'bx;
                                            Array[34]=8'b00000000;
              RegWrite=1'b0;
                                            Array[35]=8'b00000000;
              MemRead=1'b0;
                                            Array[36]=8'b00000000;
              MemWrite= 1'b0;
                                             Array[37]=8'b00000000;
              Branch=1'b1;
                                             Array[38]=8'b00000000;
              ALUOp=2'b01;
                                             Array[39]=8'b00000000;
              end
                                             end
                                             always @(*)
              7'b0010011:
                                             begin
              begin
                                                    if (mem read)
              ALUSrc=1'b1;
                                                    begin
              MemtoReg=1'b0;
              RegWrite=1'b1;
                                                    Read_data={Array[mem_addr+7],Array[mem_
              MemRead=1'b1;
                                            addr+6],
              MemWrite= 1'b0;
                                             Array[mem addr+5],Array[mem addr+4],Array[mem
              Branch=1'b0;
                                            addr+3], Array[mem addr+2],
              ALUOp=2'b00;
                                             Array[mem addr+1],Array[mem addr]};
                                                    end
              end
                                             end
                                             always @(posedge clk)
       endcase
                                             begin
end
                                                    if (mem_write)
endmodule
                                                    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(
                                             module forwarding unit(
input clk,
input reset,
                                               input[4:0] rs1 idex,
input [63:0] adder_in,
                                               input [4:0] rs2_idex,
input ZERO in,
                                               input regwrite memwb,
input [63:0] ALU_Result_in,
                                               input regwrite_exmem,
input [63:0] ReadData2In,
                                               input [4:0] rd memwb,
input [4:0] rd,
                                               input [4:0] rd_exmem,
input MemtoReg,RegWrite, branch,
MemRead, MemWrite,
                                               output reg [1:0] Forward A,
output reg [63:0] adder out,
                                               output reg [1:0] Forward B
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 @(*)
);
                                             begin
always @(posedge clk)
                                             if ((regwrite_exmem==1'b1) && (rd_exmem!=1'b0)&&
                                             (rd exmem==rs1 idex))
begin
  if (reset==1'b0)
                                               Forward A=2'b10;
  begin
                                             else if ((regwrite memwb==1'b1)&&
                                             (rd memwb!=1'b0)&& (rd memwb==rs1 idex))
  adder out= adder in;
 ALU_Result_out= ALU_Result_in;
                                               Forward_A=2'b01;
  ReadData2out= ReadData2In;
                                             else
  rdOut=rd;
                                                     Forward_A=2'b00;
 zero=ZERO in;
  MemtoRegOut=MemtoReg;
                                             if ((regwrite_exmem==1'b1)&& (rd_exmem!=1'b0)&&
  RegWriteOut=RegWrite;
                                             (rd exmem==rs2 idex))
```

```
branchOut=branch;
                                                 Forward B=2'b10;
  MemReadOut= MemRead;
  MemWriteOut=MemWrite;
                                               else if ((regwrite memwb==1'b1)&&
  end
                                               (rd_memwb!=1'b0)&& (rd_memwb==rs2_idex))
                                                 Forward B=2'b01;
end
always@(reset)
                                              else
begin
                                                      Forward_B=2'b00;
  if (reset==1'b1)
  begin
  adder out= 64'b0;
  ALU Result out= 64'b0;
                                               end
  ReadData2out= 64'b0;
                                               endmodule
  rdOut=5'b0;
  zero=1'b0;
  MemtoRegOut=1'b0;
  RegWriteOut=1'b0;
  branchOut=1'b0;
  MemReadOut= 1'b0;
  MemWriteOut=1'b0;
  end
end
endmodule
module IDEX(
                                               module Instruction memory(
input clk,
                                              input [63:0] Inst_Adress,
                                              output reg [31:0] Instruction
input reset,
input [63:0] PC_inidex,
                                              );
input [63:0] ReadData1In,
input [63:0] ReadData2In,
                                               reg [7:0]Array[15:0];
input [63:0] imm data,
                                               initial
input [4:0] rs1,
                                                 begin
input [4:0] rs2,
input [4:0] rd,
input [3:0] inst,
                                                   Array[0] =8'b10110011;
input MemtoReg,
                                                   Array[1] =8'b00000001;
input RegWrite,
                                                   Array[2] =8'b00010000;
input branch,
                                                   Array[3] =8'b00000000;
input MemRead,
input MemWrite,
                                                   Array[4] = 8'b00010011;
input ALUSrc,
                                                   Array[5] =8'b10000001;
input [1:0] ALUOp,
                                                   Array[6] =8'b00110001;
output reg [63:0] PC Outidex,
                                                   Array[7] =8'b00000000;
output reg [63:0] ReadData1Out,
output reg [63:0] ReadData2Out,
                                                   Array[8] =8'b00010011;
output reg [63:0] imm_dataOut,
                                                   Array[9] =8'b10000010;
                                                   Array[10] =8'b01100001;
output reg [3:0] funct,
output reg [4:0] rdOut,
                                                   Array[11] =8'b00000000;
output reg [4:0] rs1Out,
output reg [4:0] rs2Out,
                                                      Array[12] =8'b00110011;
```

```
output reg MemtoRegOut,
                                                Array[13] =8'b10000011;
RegWriteOut,branchOut,MemReadOut,Mem
                                                Array[14] =8'b00110001;
WriteOut, ALUSrcOut,
                                                Array[15] =8'b00000000;
output reg [1:0] ALUOpOut
);
                                              end
always @(posedge clk)
begin
      if (reset==1'b0)
              begin
                                            always @ (Inst Adress)
                                            begin
       PC Outidex=PC inidex;
                                                    Instruction={Array[Inst_Adress+3],Array[Inst_A
                                            dress+2], Array[Inst_Adress+1], Array[Inst_Adress]};
       ReadData1Out=ReadData1In;
       ReadData2Out=ReadData2In;
                                            endmodule
      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;
```

```
RegWriteOut=1'b0;
                       branchOut=1'b0;
                       MemReadOut= 1'b0;
                       MemWriteOut=1'b0;
                       ALUSrcOut=1'b0;
                       ALUOpOut=2'b0;
               end
end
endmodule
module IDG(
                                                module IFID(
input [31:0] instruction,
                                                input [63:0] PC_in,
output reg [63:0] imm_data
                                                input [31:0] instruction in,
                                                input clk,
);
always @(*)
                                                input reset,
begin
                                                output reg [63:0] PC out,
       case(instruction[6:5])
                                                output reg [31:0] Instruction_out
               2'b00:
                                                );
               begin
               imm data=
                                                always@(posedge clk)
{{52{instruction[31]}},instruction[31:20]};
                                                begin
                                                        if (reset==1'b0)
               end
               2'b01:
                                                               begin
                                                               PC out=PC in;
               begin
               imm_data=
                                                               Instruction_out=instruction_in;
{{52{instruction[31]}},instruction[31:25],instru
                                                               end
ction[11:7]};
                                                end
                                                always@(reset)
                                                begin
               end
                                                       if (reset==1'b1)
               2'b10:
               begin
                                                               begin
               imm_data=
                                                               PC_out=64'b0;
{{52{instruction[31]}},instruction[31],instructio
                                                               Instruction_out=32'b0;
n[7],instruction[30:25],instruction[11:8]};
                                                               end
               end
                                                end
               2'b11:
               begin
                                                endmodule
               imm data=
{{52{instruction[31]}},instruction[31],instructio
n[7],instruction[30:25],instruction[11:8]};
               end
       endcase
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(
                                               module parser (
  input [1:0] sel,
                                                       input [31:0] instruction,
  input [63:0] a,
                                                       output reg [6:0] opcode,
  input [63:0] b,
                                                       output reg [4:0] rd,
  input [63:0] c,
                                                       output reg [2:0] func3,
  output reg [63:0] three_muxout
                                                       output reg [4:0] rs1,
);
                                                       output reg [4:0] rs2,
always @(*)
                                                       output reg [6:0] func7
begin
  case(sel)
                                               );
                                               always@(instruction)
  2'b00: three_muxout=a;
  2'b01: three_muxout=b;
                                               begin
  2'b10: three_muxout=c;
                                               assign opcode=instruction[6:0];
                                               assign rd =instruction[11:7];
  endcase
```

```
end
                                                 assign func3=instruction[14:12];
endmodule
                                                 assign rs1 =instruction[19:15];
                                                 assign rs2=instruction[24:20];
                                                 assign func7=instruction[31:25];
                                                 end
                                                 endmodule
module Program Counter(
                                                 module tb();
input clk, reset,
                                                 reg clk;
input [63:0] Pc In,
                                                 reg reset;
output reg [63:0] PC_Out
                                                 Riscv rp(.clk(clk), .reset(reset));
);
                                                 initial
always @(posedge clk or posedge reset)
                                                 begin
begin
                                                 clk= 1'b0;
       if (reset)
                                                 reset=1'b1;
                PC Out<=0;
                                                 #7 reset =1'b0;
        else
                                                 end
                PC Out<=Pc In;
                                                 always
                                                 #5 clk=~clk:
end
endmodule
                                                 endmodule
                                                 module Riscv(
module registerFile(
input [4:0] Rs1, [4:0] Rs2, [4:0] Rd,
                                                 input clk, reset
input [63:0] WriteData,
                                                 );
input RegWrite,
                                                 wire [63:0] b;
                                                 assign b= 16'h00000000000000004;
input clk, reset,
output reg [63:0] ReadData1, reg [63:0]
ReadData2
                                                 wire [63:0] PC Out;
);
                                                 wire [63:0] PC Outidex;
reg [63:0] Array[31:0];
                                                 wire [63:0] adder1 out;
initial
                                                 wire [63:0] adder2 out;
begin
                                                 wire [63:0] adder out ex;
 Array[0]<=64'd0;
                                                 wire [63:0] mux1 out;
 Array[1]<=64'd1;
                                                 wire [31:0] Instruction m;
 Array[2]<=64'd2;
                                                 wire [31:0] Instruction if;
 Array[3]<=64'd3;
                                                 wire[6:0] opcode;
 Array[4] <= 64'd4;
                                                 wire[4:0] rd_parser;
 Array[5]<=64'd5;
                                                 wire[4:0] rd id;
 Array[6]<=64'd6;
                                                 wire[4:0] rd_mem;
 Array[7] <= 64'd7;
                                                 wire[4:0] rd ex;
 Array[8]<=64'd8;
                                                 wire[2:0] func3;
 Array[9]<=64'd9;
                                                 wire[4:0] rs1;
                                                 wire [4:0] rs2;
 Array[10]<=64'd10;
 Array[11]<=64'd11;
                                                 wire[4:0] rs1_id;
 Array[12]<=64'd12;
                                                 wire [4:0] rs2 id;
```

```
Array[13]<=64'd13;
                                              wire[6:0] func7;
 Array[14]<=64'd14;
                                              wire Branch, MemRead, MemtoReg, MemWrite,
                                              ALUSrc, RegWrite;
 Array[15]<=64'd15;
 Array[16]<=64'd16;
                                              wire [1:0] Aluop;
 Array[17]<=64'd17;
                                              wire Branch id, MemRead id, MemtoReg id,
                                              MemWrite id, ALUSrc id, RegWrite id;
 Array[18]<=64'd18;
 Array[19]<=64'd19;
                                              wire Branch_ex, MemRead_ex, MemtoReg_ex,
 Array[20]<=64'd20;
                                              MemWrite ex, RegWrite ex;
 Array[21]<=64'd21;
                                              wire MemtoReg mem, RegWrite mem;
 Array[22]<=64'd22;
                                              wire [1:0] Aluop id;
 Array[23]<=64'd23;
                                              wire [63:0] imm data;
 Array[24]<=64'd24;
                                              wire [63:0] imm dataOut;
 Array[25]<=64'd25;
                                              wire [3:0] Operation;
 Array[26]<=64'd26;
                                              wire [63:0] mux2 out;
                                              wire [63:0] ReadData1;
 Array[27]<=64'd27;
 Array[28]<=64'd28;
                                              wire [63:0] ReadData2;
 Array[29]<=64'd29;
                                              wire[63:0] ReadData1Out;
 Array[30]<=64'd30;
                                              wire [63:0] ReadData2Out;
 Array[31]<=64'd31;
                                              wire [63:0] ReadData2 ex;
end
                                              wire [63:0] Result;
                                              wire [63:0]ALU Result out;
always@(clk)
                                              wire [63:0] ALU Resultout mem;
begin
                                              wire Zero;
       if (RegWrite==1)
                                              wire zero ex;
                                              wire[63:0] Read_data;
       begin
               Array[Rd]<=WriteData;
                                              wire [63:0] ReadDataout;
       end
                                              wire [63:0] mux3_out;
                                              wire [63:0] if pc out;
end
                                              wire [3:0] funct;
                                              wire [1:0] Forward A in;
                                              wire [1:0] Forward B in;
                                              wire [63:0] forward b muxout;
always @(Rs1, Rs2, reset, Array, clk, negedge
clk)
                                              wire [63:0] alu 64 a;
                                              Program Counter p1(.clk(clk), .reset(reset),
begin
       if (reset)
                                              .Pc In(mux1 out),.PC Out(PC Out));
       begin
                                              adder a1(.a(PC_Out),.b(b),.out(adder1_out));
               ReadData1<=64'b0:
               ReadData2<=64'b0;
                                              m(.a(adder1_out),.b(adder_out_ex),.sel(Branch_ex &
                                              zero ex),.data out(mux1 out));
       end
       else
                                              //adder a2(.a(PC_Outidex),.b(imm_dataOut <<
       begin
                                              1),.out(adder2 out));
               ReadData1<=Array[Rs1];
                                              //adder a2(.a(PC Outidex),.b(imm dataOut <<
               ReadData2<=Array[Rs2];
                                              1),.out(adder2 out));
       end
end
endmodule
                                              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
```

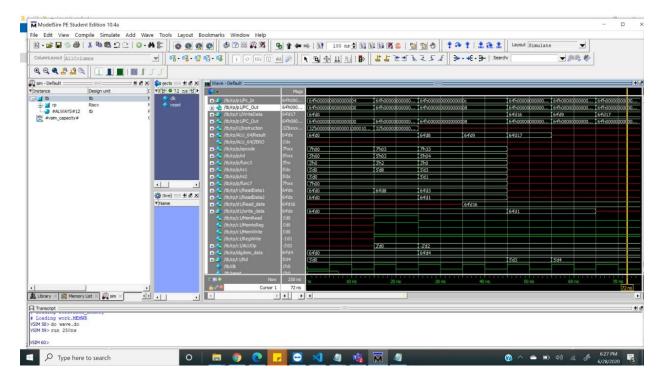
## Stalling:

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

```
input [63:0] b,
                                                       input sel,
output reg [63:0] out);
                                                       output reg[63:0] data_out
                                               );
always@(*)
                                               always@(*)
                                               begin
begin
       out = a+b;
                                               case(sel)
                                                       1'b0: assign data_out=a;
end
                                                       1'b1: assign data out=b;
endmodule
                                               endcase
                                               end
                                               endmodule
                                               module ALU_Control(
module alu_64(
           input [63:0] a,[63:0] b, [3:0]
                                               input [1:0] ALUOp,
ALUOp,
                                               input [3:0] Funct,
           output reg [63:0]Result,
                                               output reg [3:0] Operation
           output reg ZERO
                                               );
                                               always @(*)
);
always @ (*)
                                               begin
begin
                                                       case(ALUOp)
  case({ALUOp})
                                                              2'b00:
                                                              begin
           4'b0111: //slli
                                                              if (Funct==4'b0001) //slli
           begin
                                                               begin
               Result= a<<b;
                                                                      Operation=4'b0111;
                                                              end
           end
                                                              else
           4'b0101://bne
                                                               begin
           begin
                                                                      Operation=4'b0010;
               Result= a-b;
                                                              end
               if(Result==0)
                                                              end
                       begin
                       ZERO=0;
                                                              2'b01:
                       end
                                                               begin
               else
                                                              if (Funct==4'b0001)//bne
                       ZERO=1;
                                                              begin
           end
                                                                      Operation=4'b0101;
           4'b1010:
                                                              end
           begin
                                                              else if (Funct==4'b0100)//blt
               Result = (a < b)?1:0;
                                                               begin
               ZERO = Result;
                                                                      Operation=4'b1010;
           end
                                                              else if (Funct==4'b0000) //beq
   4'b0000:
                                                              begin
                                                                      Operation=4'b0110;
           begin
            Result = a&b;
                                                              end
           end
   4'b0001 :
                                                              end
```

```
begin
            Result = a|b;
                                                            2'b10:
                                                             begin
           end
                                                            if (Funct==4'b0000)
   4'b0010 :
                                                             begin
                                                                    Operation=4'b0010;
          begin
            Result = a+b;
                                                            end
                                                            else if (Funct==4'b1000)
           end
                                                             begin
          4'b0110://beq
                                                                    Operation=4'b0110;
          begin
                                                            end
                                                            else if (Funct==4'b0111)
          Result = a-b;
          ZERO = Result?0:1;
                                                             begin
          end
                                                                    Operation=4'b0000;
                                                            end
          default : Result = ^{(a|b)};
                                                            else if (Funct==4'b0110)
  endcase
                                                             begin
                                                                    Operation=4'b0001;
                                                            end
end
                                                            end
endmodule
                                                     endcase
                                             end
                                             endmodule
module Control Unit(
                                             module Data memory(
input [6:0] Opcode,
                                             input [63:0]mem addr,
output reg Branch, MemRead, MemtoReg,
                                             input [63:0] write_data,
MemWrite, ALUSrc, RegWrite,
                                             input clk,
output reg [1:0] ALUOp
                                             input mem write,
                                             input mem read,
);
                                             output reg[63:0] Read data
                                             );
always @(*)
begin
                                             reg [7:0] Array [63:0];
       case(Opcode)
               7'b0110011:
                                             initial
               begin
                                             begin
               ALUSrc=1'b0;
                                             Array[0]=8'b00000100;
               MemtoReg=1'b0;
                                             Array[1]=8'b00000000;
               RegWrite=1'b1;
                                             Array[2]=8'b00000000;
               MemRead=1'b0;
                                             Array[3]=8'b00000000;
               MemWrite=1'b0;
                                             Array[4]=8'b00000000;
               Branch=1'b0;
                                             Array[5]=8'b00000000;
               ALUOp=2'b10;
                                             Array[6]=8'b00000000;
               end
                                             Array[7]=8'b00000000;
               7'b0000011:
                                             Array[8]=8'b00010000;
                                             Array[9]=8'b00000000;
               begin
```

```
ALUSrc=1'b1;
                                           Array[10]=8'b00000000;
              MemtoReg=1'b1;
                                           Array[11]=8'b00000000;
              RegWrite=1'b1;
                                           Array[12]=8'b000000000:
              MemRead=1'b1;
                                           Array[13]=8'b00000000;
              MemWrite=1'b0;
                                           Array[14]=8'b00000000;
              Branch=1'b0;
                                           Array[15]=8'b00000000;
              ALUOp=2'b00;
                                           Array[16]=8'b00000011;
              end
                                           Array[17]=8'b00000000;
              7'b0100011:
                                           Array[18]=8'b00000000;
              begin
                                           Array[19]=8'b00000000;
                                           Array[20]=8'b000000000;
              ALUSrc=1'b1;
              MemtoReg=1'bx;
                                           Array[21]=8'b00000000;
              RegWrite=1'b0;
                                           Array[22]=8'b00000000;
              MemRead=1'b0;
                                           Array[23]=8'b00000000;
              MemWrite=1'b1;
              Branch=1'b0;
                                           Array[24]=8'b00000010;
              ALUOp=2'b00;
                                           Array[25]=8'b00000000;
              end
                                           Array[26]=8'b00000000;
                                           Array[27]=8'b00000000;
              7'b1100011:
                                           Array[28]=8'b00000000;
              begin
                                           Array[29]=8'b00000000;
              ALUSrc=1'b0;
                                           Array[30]=8'b00000000;
              MemtoReg=1'bx;
                                           Array[31]=8'b00000000;
              RegWrite=1'b0;
              MemRead=1'b0:
                                           Array[32]=8'b00100000;
                                           Array[33]=8'b00000000;
              MemWrite= 1'b0;
              Branch=1'b1;
                                           Array[34]=8'b00000000;
              ALUOp=2'b01;
                                           Array[35]=8'b00000000;
              end
                                           Array[36]=8'b00000000;
                                           Array[37]=8'b00000000;
              7'b0010011:
                                           Array[38]=8'b00000000;
              begin
                                           Array[39]=8'b00000000;
              ALUSrc=1'b1;
              MemtoReg=1'b0;
              RegWrite=1'b1;
              MemRead=1'b1:
                                           end
              MemWrite= 1'b0;
              Branch=1'b0;
                                           always @(*)
              ALUOp=2'b00;
                                           begin
              end
                                                   if (mem read)
                                                   begin
                                                   Read data={Array[mem addr+7],Array[mem a
       endcase
end
                                           ddr+61,
endmodule
                                           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 forwarding_unit(
module EXMEM(
input clk,
input reset,
                                               input[4:0] rs1 idex,
input [63:0] adder in,
                                               input [4:0] rs2 idex,
input ZERO in,
                                               input regwrite memwb,
input [63:0] ALU_Result_in,
                                               input regwrite exmem,
input [63:0] ReadData2In,
                                               input [4:0] rd_memwb,
input [4:0] rd,
                                               input [4:0] rd_exmem,
input MemtoReg,RegWrite, branch,
MemRead, MemWrite,
                                               output reg [1:0] Forward_A,
                                               output reg [1:0] Forward_B
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,
                                             always @(*)
output reg MemtoRegOut, RegWriteOut,
                                             begin
branchOut, MemReadOut, MemWriteOut
                                             if ((regwrite_exmem==1'b1) && (rd_exmem!=1'b0)&&
);
                                             (rd_exmem==rs1_idex))
                                               Forward A=2'b10;
always @(posedge clk)
```

```
else if ((regwrite memwb==1'b1)&&
begin
  if (reset==1'b0)
                                            (rd_memwb!=1'b0)&& (rd_memwb==rs1_idex))
                                              Forward A=2'b01;
  begin
  adder_out= adder_in;
                                            else
  ALU Result out= ALU Result in;
                                                    Forward A=2'b00;
  ReadData2out= ReadData2In;
                                            if ((regwrite_exmem==1'b1)&& (rd_exmem!=1'b0)&&
  rdOut=rd;
                                            (rd exmem==rs2 idex))
  zero=ZERO in;
                                              Forward B=2'b10;
  MemtoRegOut=MemtoReg;
  RegWriteOut=RegWrite;
  branchOut=branch;
                                            else if ((regwrite memwb==1'b1)&&
                                            (rd memwb!=1'b0)&& (rd_memwb==rs2_idex))
  MemReadOut= MemRead;
  MemWriteOut=MemWrite;
                                              Forward B=2'b01;
                                            else
                                                    Forward_B=2'b00;
  end
end
always@(reset)
begin
  if (reset==1'b1)
                                            end
                                            endmodule
  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
module hazard_unit (
                                            module IDG(
input MemRead idex,
                                            input [31:0] instruction,
input [4:0] rd_idex,
                                            output reg [63:0] imm_data
input [4:0] Rs1 ifid,
input [4:0] Rs2 ifid,
                                            always @(*)
output reg PcWrite,
                                            begin
output reg Ifid write,
                                                    case(instruction[6:5])
                                                           2'b00:
output reg mux sel
                                                           begin
);
always@(*)
                                                           imm_data=
                                            {{52{instruction[31]}},instruction[31:20]};
begin
                                                           end
                                                           2'b01:
if (MemRead_idex && ((rd_idex==Rs1_ifid) ||
                                                           begin
(rd idex==Rs2 ifid)))
```

```
begin
                                                                imm data=
  PcWrite=1'b0;
                                                {{52{instruction[31]}},instruction[31:25],instruction[11:
  Ifid write=1'b0;
                                                7]};
  mux_sel=1'b1;
  end
                                                                end
                                                                2'b10:
else
                                                                begin
begin
  PcWrite=1'b1;
                                                                imm data=
                                                {{52{instruction[31]}},instruction[31],instruction[7],instr
  Ifid write=1'b1;
                                                uction[30:25],instruction[11:8]};
  mux sel=1'b0;
                                                                end
                                                                2'b11:
end
end
                                                                begin
endmodule
                                                                imm data=
                                                {{52{instruction[31]}},instruction[31],instruction[7],instr
                                                uction[30:25],instruction[11:8]};
                                                                end
                                                        endcase
                                                end
                                                endmodule
                                                module registerFile(
module IDEX(
input clk,
                                                input [4:0] Rs1, [4:0] Rs2, [4:0] Rd,
input reset,
                                                input [63:0] WriteData,
input [63:0] PC_inidex,
                                                input RegWrite,
input [63:0] ReadData1In,
                                                input clk, reset,
input [63:0] ReadData2In,
                                                output reg [63:0] ReadData1, reg [63:0] ReadData2
input [63:0] imm_data,
                                                );
input [4:0] rs1,
input [4:0] rs2,
                                                reg [63:0] Array[31:0];
input [4:0] rd,
                                                initial
input [3:0] inst,
                                                begin
                                                  Array[0] <= 64'd0;
input MemtoReg,
input RegWrite,
                                                  Array[1]<=64'd1;
input branch,
                                                  Array[2]<=64'd2;
input MemRead,
                                                  Array[3]<=64'd3;
input MemWrite,
                                                  Array[4]<=64'd4;
input ALUSrc,
                                                  Array[5]<=64'd5;
input [1:0] ALUOp,
                                                  Array[6]<=64'd6;
output reg [63:0] PC Outidex,
                                                  Array[7]<=64'd7;
output reg [63:0] ReadData1Out,
                                                  Array[8]<=64'd8;
output reg [63:0] ReadData2Out,
                                                  Array[9]<=64'd9;
output reg [63:0] imm dataOut,
                                                  Array[10]<=64'd10;
output reg [3:0] funct,
                                                  Array[11]<=64'd11;
output reg [4:0] rdOut,
                                                  Array[12]<=64'd12;
output reg [4:0] rs1Out,
                                                  Array[13]<=64'd13;
output reg [4:0] rs2Out,
                                                  Array[14]<=64'd14;
                                                  Array[15]<=64'd15;
                                                  Array[16]<=64'd16;
```

```
output reg MemtoRegOut,
                                             Array[17]<=64'd17;
RegWriteOut,branchOut,MemReadOut,Mem
                                             Array[18]<=64'd18;
WriteOut, ALUSrcOut,
                                             Array[19]<=64'd19;
output reg [1:0] ALUOpOut
                                             Array[20]<=64'd20;
);
                                             Array[21]<=64'd21;
                                             Array[22]<=64'd22;
                                             Array[23]<=64'd23;
always @(posedge clk)
begin
                                             Array[24]<=64'd24;
       if (reset==1'b0)
                                             Array[25]<=64'd25;
              begin
                                             Array[26]<=64'd26;
                                             Array[27]<=64'd27;
                                             Array[28]<=64'd28;
       PC Outidex=PC inidex;
                                             Array[29]<=64'd29;
       ReadData1Out=ReadData1In;
                                             Array[30]<=64'd30;
                                             Array[31]<=64'd31;
       ReadData2Out=ReadData2In;
                                            end
       imm_dataOut=imm_data;
                                            always@(negedge clk)
                     funct=inst;
                                            begin
                      rdOut=rd;
                                                   if (RegWrite==1)
                      rs1Out=rs1;
                                                   begin
                      rs2Out=rs2;
                                                           Array[Rd]<=WriteData;
                                                   end
       MemtoRegOut=MemtoReg;
                                            end
       RegWriteOut=RegWrite;
                      branchOut=branch;
                      MemReadOut=
                                            always @(Rs1, Rs2, reset, Array, clk, posedge clk)
MemRead;
                                            begin
                                                   if (reset)
       MemWriteOut=MemWrite;
                                                   begin
                      ALUSrcOut=ALUSrc;
                                                           ReadData1<=64'b0;
                      ALUOpOut=ALUOp;
                                                           ReadData2<=64'b0;
                                                   end
              end
end
                                                   else
always@(reset)
                                                   begin
begin
                                                           ReadData1<=Array[Rs1];
       if(reset==1'b1)
                                                           ReadData2<=Array[Rs2];
              begin
                                                   end
                      PC_Outidex=64'b0;
                                            end
                                            endmodule
       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 IFID(
                                              module Instruction memory(
input [63:0] PC_in,
                                              input [63:0] Inst_Adress,
input [31:0] instruction in,
                                              output reg [31:0] Instruction
input clk,
                                              );
input ifid write,
                                              reg [7:0]Array[7:0];
input reset,
output reg [63:0] PC out,
                                               initial
output reg [31:0] Instruction out
                                                begin
);
always@(posedge clk)
                                                   Array[0] = 8'b10000011;
begin
                                                   Array[1] =8'b00100001;
       if (ifid_write==1'b1)
       begin
                                                   Array[2] =8'b00000100;
       if (reset==1'b0)
                                                   Array[3] =8'b00000000;
               begin
                                                   Array[4] =8'b00110011;
               PC out=PC in;
                                                   Array[5] =8'b10000010;
       Instruction_out=instruction_in;
                                                   Array[6] =8'b00010001;
               end
                                                   Array[7] =8'b00000000;
       end
end
                                                end
always@(reset)
begin
       if (reset==1'b1)
               begin
                                              always @ (Inst_Adress)
               PC out=64'b0;
                                              begin
               Instruction out=32'b0;
                                                      Instruction={Array[Inst_Adress+3],Array[Inst_A
               end
                                              dress+2], Array[Inst Adress+1], Array[Inst Adress]};
end
                                              end
endmodule
                                              endmodule
module MEMWB(
                                              module parser (
                                                      input [31:0] instruction,
  input clk,
```

```
output reg [6:0] opcode,
  input reset,
  input [63:0] ReadDatain,
                                                       output reg [4:0] rd,
  input [63:0] ALU Resultin,
                                                       output reg [2:0] func3,
  input MemtoReg, RegWrite,
                                                       output reg [4:0] rs1,
  input [4:0] rd,
                                                       output reg [4:0] rs2,
  output reg [63:0] ReadDataout,
                                                       output reg [6:0] func7
  output reg [63:0] ALU_Resultout,
  output reg [4:0] rdOut,
                                               );
  output reg MemtoRegOut, RegWriteOut
                                               always@(instruction)
);
                                               begin
                                               assign opcode=instruction[6:0];
always @(posedge clk)
                                               assign rd =instruction[11:7];
begin
                                               assign func3=instruction[14:12];
  if (reset==1'b0)
                                               assign rs1 =instruction[19:15];
  begin
                                               assign rs2=instruction[24:20];
  ReadDataout=ReadDatain;
                                               assign func7=instruction[31:25];
  ALU Resultout=ALU Resultin;
                                               end
  rdOut=rd;
  MemtoRegOut=MemtoReg;
                                               endmodule
  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(
                                               module tb();
  input [1:0] sel,
                                               reg clk;
  input [63:0] a,
                                               reg reset;
  input [63:0] b,
  input [63:0] c,
                                               Riscv rp(.clk(clk), .reset(reset));
  output reg [63:0] three muxout
                                               initial
                                               begin
);
always @(*)
                                               clk= 1'b0;
begin
                                               reset=1'b1;
  case(sel)
                                               #7 reset =1'b0;
  2'b00: three_muxout=a;
                                               end
  2'b01: three_muxout=b;
                                               always
  2'b10: three muxout=c;
```

endcase	#5 clk=~clk;		
end	#3 CIK- CIK,		
endmodule	endmodule		
module Program_Counter(	module Riscv(		
input clk, reset,	input clk, reset		
input [63:0] Pc_In,	);		
input PCWrite,	wire [63:0] b;		
output reg [63:0] PC_Out	assign b= 16'h0000000000000004;		
);	wire [63:0] PC_Out;		
<i>"</i>	wire [63:0] PC_Outidex;		
always @(posedge clk or posedge reset)			
begin	wire [63:0] adder1_out; wire [63:0] adder2_out;		
	wire [63:0] adder_out_ex;		
if (reset) PC Out<=0;	wire [63:0] mux1_out;		
else if (PCWrite==1'b1)	wire [31:0] Instruction_m; wire [31:0] Instruction_if;		
begin	wire[6:0] opcode;		
PC_Out<=Pc_In; end	wire[4:0] rd_parser; wire[4:0] rd_id;		
ena	wire[4:0] rd_id, wire[4:0] rd_mem;		
end	wire[4:0] rd_mem, wire[4:0] rd_ex;		
endmodule	wire[4:0] ru_ex, wire[2:0] func3;		
enamodale			
	wire[4:0] rs1;		
	wire [4:0] rs2;		
	wire[4:0] rs1_id;		
	wire [4:0] rs2_id;		
	wire Branch Mambaad MamtaBag MamWrite		
	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] 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));
m3(.a(ALU Resultout mem),.b(ReadDataout),.sel(Mem
toReg_mem),.data_out(mux3_out));
endmodule
```

Nimrah Jawed (nj04707) Umme Salma (us04315)

## Flushing:

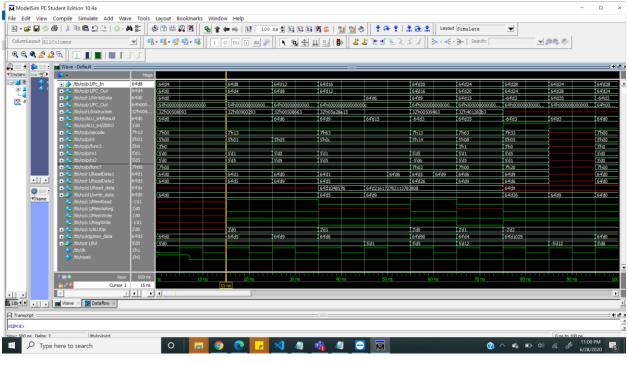
The and gate with inputs branch (from ex) and zero(from ex) will give an output ALUSrc. We are then sending this to IFID, IDEX and EXMEM. If this is 0, then these modules will assign the input values. If this signal is high, it will flush all contents of IFID IDEX and EXMEM and outputs zeros. Even though 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.



```
module adder(
                                         module mux(
input [63:0] a,
                                                 input[63:0] a, [63:0]b,
input [63:0] b,
                                                 input sel,
output reg [63:0] out);
                                                 output reg[63:0] data_out
                                         );
always@(*)
                                         always@(*)
begin
                                         begin
       out = a+b;
                                         case(sel)
end
                                                 1'b0: assign data out=a;
                                                 1'b1: assign data out=b;
endmodule
                                         endcase
                                         end
                                         endmodule
                                         module ALU Control(
module alu_64(
       input [63:0] a,[63:0] b, [3:0]
                                         input [1:0] ALUOp,
ALUOp,
                                         input [3:0] Funct,
       output reg [63:0]Result,
                                         output reg [3:0] Operation
       output reg ZERO
                                         );
                                         always @(*)
always @ (*)
                                         begin
begin
                                                 case(ALUOp)
                                                         2'b00:
  case({ALUOp})
                                                         begin
       4'b0111: //slli
                                                         if (Funct==4'b0001) //slli
       begin
                                                         begin
               Result= a<<b;
                                                                 Operation=4'b0111;
```

```
ZERO=1'b0;
                                                       end
                                                       else
       end
                                                       begin
       4'b0101://bne
                                                               Operation=4'b0010;
       begin
                                                       end
               Result= a-b;
                                                       end
               if(Result==0)
                                                       2'b01:
                       begin
                      ZERO=1'b0;
                                                       begin
                                                       if (Funct==4'b0001)//bne
                       end
               else
                                                       begin
                                                               Operation=4'b0101;
                       ZERO=1'b1;
       end
                                                       end
       4'b1010:
                                                       else if (Funct==4'b0100)//blt
       begin
                                                       begin
               Result = (a < b)?1:0;
                                                               Operation=4'b1010;
               ZERO = Result;
                                                       end
       end
                                                       else if (Funct==4'b0000) //beq
                                                       begin
   4'b0000:
                                                               Operation=4'b0110;
       begin
                                                       end
        Result = a&b;
        ZERO=1'b0;
                                                       end
       end
   4'b0001 :
                                                       2'b10:
       begin
                                                       begin
        Result = a|b;
                                                       if (Funct==4'b0000)
        ZERO=1'b0;
                                                       begin
       end
                                                               Operation=4'b0010;
                                                       end
   4'b0010 :
                                                       else if (Funct==4'b1000)
       begin
                                                       begin
        Result = a+b;
                                                               Operation=4'b0110;
        ZERO=1'b0;
                                                       end
       end
                                                       else if (Funct==4'b0111)
                                                       begin
       4'b0110://beq
                                                               Operation=4'b0000;
       begin
                                                       end
       Result = a-b;
                                                       else if (Funct==4'b0110)
       ZERO = Result?0:1;
                                                       begin
                                                               Operation=4'b0001;
       end
                                                       end
       default : Result = ^{(a|b)};
                                                       end
  endcase
                                                endcase
                                        end
                                        endmodule
end
endmodule
```

```
module Control Unit(
                                       module Data memory(
input [6:0] Opcode,
                                       input [63:0]mem_addr,
output reg Branch, MemRead,
                                       input [63:0] write data,
MemtoReg, MemWrite, ALUSrc,
                                       input clk,
RegWrite,
                                       input mem write,
output reg [1:0] ALUOp
                                       input mem read,
                                       output reg[63:0] Read_data
);
                                       );
                                       reg [7:0] Array [63:0];
always @(*)
begin
       case(Opcode)
                                       initial
              7'b0110011:
                                       begin
              begin
                                       Array[0]=8'b00000100;
              ALUSrc=1'b0;
                                       Array[1]=8'b00000000;
              MemtoReg=1'b0;
                                       Array[2]=8'b00000000;
              RegWrite=1'b1;
                                       Array[3]=8'b00000000;
              MemRead=1'b0;
                                       Array[4]=8'b00000000;
                                       Array[5]=8'b00000000;
              MemWrite=1'b0;
              Branch=1'b0;
                                       Array[6]=8'b00000000;
              ALUOp=2'b10;
                                       Array[7]=8'b00000000;
              end
                                       Array[8]=8'b00010000;
              7'b0000011:
                                       Array[9]=8'b00000000;
              begin
                                       Array[10]=8'b00000000;
              ALUSrc=1'b1;
                                       Array[11]=8'b00000000;
                                       Array[12]=8'b00000000;
              MemtoReg=1'b1;
              RegWrite=1'b1;
                                       Array[13]=8'b00000000;
              MemRead=1'b1;
                                       Array[14]=8'b00000000;
              MemWrite=1'b0;
                                       Array[15]=8'b00000000;
              Branch=1'b0;
              ALUOp=2'b00;
                                       Array[16]=8'b00000011;
              end
                                       Array[17]=8'b00000000;
                                       Array[18]=8'b00000000;
              7'b0100011:
                                       Array[19]=8'b00000000;
              begin
                                       Array[20]=8'b00000000;
              ALUSrc=1'b1;
                                       Array[21]=8'b00000000;
              MemtoReg=1'bx;
                                       Array[22]=8'b00000000;
              RegWrite=1'b0;
                                       Array[23]=8'b00000000;
              MemRead=1'b0;
              MemWrite=1'b1;
                                       Array[24]=8'b00000010;
              Branch=1'b0;
                                       Array[25]=8'b00000000;
              ALUOp=2'b00;
                                       Array[26]=8'b00000000;
              end
                                       Array[27]=8'b00000000;
                                       Array[28]=8'b00000000;
              7'b1100011:
                                       Array[29]=8'b00000000;
              begin
                                       Array[30]=8'b00000000;
              ALUSrc=1'b0;
                                       Array[31]=8'b00000000;
```

```
MemtoReg=1'bx;
              RegWrite=1'b0;
                                      Array[32]=8'b00100000;
              MemRead=1'b0;
                                      Array[33]=8'b00000000;
              MemWrite= 1'b0;
                                      Array[34]=8'b00000000;
              Branch=1'b1;
                                      Array[35]=8'b00000000;
              ALUOp=2'b01;
                                      Array[36]=8'b00000000;
              end
                                      Array[37]=8'b00000000;
                                      Array[38]=8'b00000000;
              7'b0010011:
                                      Array[39]=8'b00000000;
              begin
              ALUSrc=1'b1;
              MemtoReg=1'b0;
              RegWrite=1'b1;
                                      end
              MemRead=1'b1;
              MemWrite= 1'b0;
                                      always @(*)
                                      begin
              Branch=1'b0;
              ALUOp=2'b00;
                                              if (mem_read)
              end
                                              begin
                                              Read data={Array[mem addr+7],Array[mem addr+6
       endcase
                                      ],
end
                                      Array[mem addr+5],Array[mem addr+4],Array[mem addr+3
endmodule
                                      ],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(
                                      module IDEX(
input clk,
                                      input clk,
input reset,
                                      input reset,
input [63:0] adder in,
                                      input [63:0] PC_inidex,
```

input ZERO_in, input PCSrc, input [63:0] ReadData2In, input [63:0] ReadData2In, input [63:0] ReadData2In, input [4:0] rd, input MemtoReg,RegWrite, branch, MemRead,MemWrite,  output reg [63:0] adder_out, input MemtoReg [63:0] ReadData2out, output reg [63:0] ReadData2out, output reg [4:0] rdOut, output reg MemtoRegOut, RegWriteOut, branchOut, MemReadOut, MemWriteOut );  always @(posedge clk) begin if (PCSrc==1'b0)  input [63:0] ReadData2In, input [4:0] rs1, input [4:0] rs2, input [4:0] rd, input [4:0] rd, input [4:0] rd, input [4:0] rd, input PCSrc, input [4:0] rd, input PCSrc, input PCSrc, input PCSrc, input RegWrite, input PCSrc, input MemtoReg, input RegWrite, input MemtoReg, input MemtoReg, input MemRead, input MemWrite, input MemWrite, input MemWrite, input MemWrite, input MemWrite, input MemWrite, input MemRead, input MemWrite, input MemWrite, input MemWrite, input MemWrite, input MemRead, input PCSrc, input Reg G3:0] ReadData1Out, output reg [63:0] ReadData1Out, output reg [63:0] ReadData2Out, output reg [63:0] Imm_dataOut, output reg [63:0] Imm_dataOut, output reg [4:0] rs1Out, output reg [4:0] rs1Out, output reg [4:0] rs2Out, output reg MemtoRegOut,				
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, input MemRead, MemUrite,  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  input [63:0] imm_data, input [4:0] rs1, input [4:0] rs2, input [4:0] rd, input [3:0] inst, input MemtoReg, input MemtoReg, input MemRead, input MemRead, input MemWrite, 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 [63:0] imm_dataOut, output reg [4:0] rs1out, output reg [4:0] rs1out, output reg [4:0] rs2Out,				
input [63:0] ReadData2In, input [4:0] rd, input MemtoReg,RegWrite, branch, MemRead,MemWrite,  output reg [63:0] adder_out, input MemtoReg, input MemtoReg, input MemtoReg, input MemtoReg, input MemtoReg, input MemtoReg, input RegWrite, output reg [63:0] ALU_Result_out, output reg [63:0] ReadData2out, output reg [63:0] ReadData2out, output reg 2ero, output reg MemtoRegOut, RegWriteOut, branchOut, MemReadOut, MemWriteOut );  output reg [63:0] ReadData2out, output reg [63:0] PC_Outidex, output reg [63:0] ReadData1Out, output reg [63:0] ReadData2Out, output reg [63:0] imm_dataOut, output reg [63:0] funct, output reg [4:0] rs2, input [4:0] rs1, input [4:0] rs1, input [4:0] rs2, input				
input [4:0] rd, input MemtoReg,RegWrite, branch, MemRead,MemWrite,  output reg [63:0] adder_out,  input MemtoReg, input MemtoReg, input PCSrc, input MemtoReg, input RegWrite,  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 );  output reg [63:0] ReadData2out, output reg [63:0] ReadData2out, 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 [63:0] imm_dataOut, output reg [4:0] rdOut, if (PCSrc==1'b0) begin  input [4:0] rs2, input [4:0] rs2, input [4:0] rs2, input [4:0] rs1, input [4:0] rs2, input [4:0]				
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  input [4:0] rd, input [3:0] inst, input PCSrc, input MemtoReg, input MemtoReg, input MemRead, input MemRead, input MemWrite, input MemWrite, input ALUSrc, input [1:0] ALUOp, output reg [63:0] PC_Outidex, output reg [63:0] ReadData1Out, output reg [63:0] imm_dataOut, output reg [63:0] imm_dataOut, output reg [4:0] rs1Out, output reg [4:0] rs2Out,				
MemRead,MemWrite,  output reg [63:0] adder_out,  input MemtoReg, input RegWrite, input RegWrite, input MemRead, output reg [63:0] ReadData2out, output reg [4:0] rdOut, output reg MemtoRegOut, RegWriteOut, branchOut, MemReadOut, MemWriteOut );  output reg [63:0] ReadData2Out, output reg [63:0] PC_Outidex, output reg [63:0] ReadData1Out, output reg [63:0] ReadData2Out, output reg [63:0] ReadData2Out, output reg [63:0] imm_dataOut, output reg [3:0] funct, output reg [4:0] rdOut, if (PCSrc==1'b0) begin  input [3:0] inst, input MemtoReg, input MemRead, input MemRead, input MemRead, input MemRead, input MemRead, input MemRead, input MemWrite, input MemRead, input MemRead input Me				
input PCSrc, input MemtoReg, input RegWrite, output reg [63:0] ALU_Result_out, output reg [63:0] ReadData2out, output reg [4:0] rdOut, output reg MemtoRegOut, RegWriteOut, branchOut, MemReadOut, MemWriteOut ); output reg [63:0] ReadData2Out, output reg [63:0] ReadData2Out, output reg [63:0] ReadData2Out, output reg [63:0] ReadData1Out, output reg [63:0] imm_dataOut, output reg [63:0] imm_dataOut, output reg [4:0] rdOut, if (PCSrc==1'b0) begin input PCSrc, input MemtoReg, input RegWrite, input MemtoReg, input RegWrite, input NemtoReg input RegWrite, input NemtoReg input RegWrite, input RegWrite inpu				
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 );  output reg [63:0] ReadData2Out, input MemRead, input MemRead, input MemWrite, input MemWrite, input MemRead, input MemRead input				
input RegWrite, 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 ); output reg [63:0] ReadData1Out, output reg [63:0] ReadData1Out, output reg [63:0] ReadData2Out, output reg [63:0] imm_dataOut, output reg [3:0] rdOut, output reg [4:0] rs1Out, begin input RegWrite, input RegWrite, input Branch, input MemRead, input MemVrite, input MemRead, input MemRead, input MemRead, input MemRead, input MemRead, input MemRead, input MemVrite, input MemVrite, input MemVrite, input MemVrite, input MemVrite, input MemVrite, input MemVrite input Me				
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 ); output reg [63:0] ReadData1Out, 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] rs1Out, begin input MemRead, input MemVite, in				
output reg [63:0] ReadData2out, output reg [4:0] rdOut, input MemWrite, output reg zero, output reg MemtoRegOut, RegWriteOut, branchOut, MemReadOut, MemWriteOut ); output reg [63:0] PC_Outidex, output reg [63:0] ReadData1Out, output reg [63:0] ReadData2Out, output reg [63:0] imm_dataOut, output reg [63:0] funct, output reg [4:0] rs1Out, begin input MemRead, input MemVite, input MemVite, input ALUSrc, input [1:0] ALUOp, output reg [63:0] PC_Outidex, output reg [63:0] ReadData1Out, output reg [63:0] imm_dataOut, output reg [4:0] rdout, output reg [4:0] rs1Out, output reg [4:0] rs2Out,				
output reg [4:0] rdOut, output reg zero, output reg MemtoRegOut, RegWriteOut, branchOut, ); output reg [63:0] PC_Outidex, output reg [63:0] ReadData1Out, output reg [63:0] ReadData2Out, output reg [63:0] imm_dataOut, output reg [63:0] funct, output reg [4:0] rdOut, if (PCSrc==1'b0) begin input MemWrite, input ALUSrc, input [1:0] ALUOp, output reg [63:0] PC_Outidex, output reg [63:0] ReadData1Out, output reg [63:0] imm_dataOut, output reg [4:0] rdOut, output reg [4:0] rs1Out, output reg [4:0] rs2Out,				
output reg zero, output reg MemtoRegOut, RegWriteOut, branchOut, MemReadOut, MemWriteOut ); 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] rs1Out, begin if (PCSrc==1'b0) begin output reg [4:0] rs2Out,				
output reg zero, output reg MemtoRegOut, RegWriteOut, branchOut, MemReadOut, MemWriteOut ); 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] rs1Out, begin if (PCSrc==1'b0) begin output reg [4:0] rs2Out,				
output reg MemtoRegOut, RegWriteOut, branchOut, MemReadOut, MemWriteOut ); 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, if (PCSrc==1'b0) begin output reg [4:0] rs1Out, begin output reg [4:0] rs2Out,				
RegWriteOut, branchOut, MemReadOut, MemWriteOut ); 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, if (PCSrc==1'b0) begin output reg [4:0] rs1Out, begin output reg [4:0] rs2Out,	•			
MemReadOut, MemWriteOutoutput reg [63:0] ReadData1Out,);output reg [63:0] ReadData2Out,output reg [63:0] imm_dataOut,always @(posedge clk)output reg [3:0] funct,beginoutput reg [4:0] rdOut,if (PCSrc==1'b0)output reg [4:0] rs1Out,beginoutput reg [4:0] rs2Out,				
output reg [63:0] ReadData2Out, output reg [63:0] imm_dataOut, always @(posedge clk) begin if (PCSrc==1'b0) begin output reg [4:0] rs1Out, output reg [4:0] rs2Out,				
output reg [63:0] imm_dataOut, always @(posedge clk) begin if (PCSrc==1'b0) begin output reg [4:0] rdOut, output reg [4:0] rs1Out, output reg [4:0] rs2Out,				
always @(posedge clk)  begin  if (PCSrc==1'b0)  begin  output reg [3:0] funct,  output reg [4:0] rdOut,  output reg [4:0] rs1Out,  output reg [4:0] rs2Out,				
begin output reg [4:0] rdOut, if (PCSrc==1'b0) output reg [4:0] rs1Out, begin output reg [4:0] rs2Out,				
if (PCSrc==1'b0) output reg [4:0] rs1Out, begin output reg [4:0] rs2Out,				
begin output reg [4:0] rs2Out,				
It (leset1 bb)   Output leg Memitonegout,				
begin RegWriteOut,branchOut,MemReadOut,MemWriteOut,A	RegWriteOut,branchOut,MemReadOut,MemWriteOut,ALUSrc			
	Out,			
	·			
	output reg [1:0] ALUOpOut			
	);			
	always @(posedge clk)			
zero=ZERO_in; begin				
MemtoRegOut=MemtoReg; if (PCSrc==1'b0)				
RegWriteOut=RegWrite; begin				
branchOut=branch; if (reset==1'b0)				
MemReadOut= MemRead; begin				
MemWriteOut=MemWrite; PC_Outidex=PC_inidex;				
end ReadData1Out=ReadData1In;				
end ReadData2Out=ReadData2In;				
else imm_dataOut=imm_data;				
begin funct=inst;				
adder_out= 64'b0; rdOut=rd;				
ALU_Result_out= 64'b0; rs1Out=rs1;				
ReadData2out= 64'b0; rs2Out=rs2;				
rdOut=5'b0; MemtoRegOut=MemtoReg;				
zero=1'b0; RegWriteOut=RegWrite;				
MemtoRegOut=1'b0; branchOut=branch;				
RegWriteOut=1'b0; MemReadOut= MemRead;				
branchOut=1'b0; MemWriteOut=MemWrite;				
MemReadOut= 1'b0; ALUSrcOut=ALUSrc;				

MemWriteOut=1'b0;			ALUOpOut=ALUOp;
end		end	, 1.20 ο ρου ( - / 1.20 ο ρ ,
	end	Cilu	
end	else		
always@(reset)		begin	
begin	'	ocgiii	PC_Outidex=64'b0;
if (reset==1'b1)			ReadData1Out=64'b0;
begin			ReadData1Out=64'b0;
adder_out= 64'b0;			imm dataOut=64'b0;
ALU Result out= 64'b0;			funct=4'b0;
ReadData2out= 64'b0;			rdOut=5'b0;
			•
rdOut=5'b0;			rs10ut=5'b0;
zero=1'b0;			rs2Out= 5'b0;
MemtoRegOut=1'b0;			MemtoRegOut=1'b0;
RegWriteOut=1'b0;			RegWriteOut=1'b0;
branchOut=1'b0;			branchOut=1'b0;
MemReadOut= 1'b0;			MemReadOut= 1'b0;
MemWriteOut=1'b0;			MemWriteOut=1'b0;
end			ALUSrcOut=1'b0;
end		l	ALUOpOut=2'b0;
endmodule		end	
	end	M 1 \	
	always@	(reset)	
	begin	:f/	
	!	if(reset==1'b1)	
		begin	DC Continue CAllege
			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		
	endmod		
module forwarding_unit(		hazard_unit (	
		emRead_idex,	
input[4:0] rs1_idex,	-	0] rd_idex,	
input [4:0] rs2_idex,	input [4:	0] Rs1_ifid,	

```
input regwrite memwb,
                                        input [4:0] Rs2 ifid,
  input regwrite_exmem,
                                        output reg PcWrite,
  input [4:0] rd memwb,
                                        output reg Ifid write,
  input [4:0] rd_exmem,
                                        output reg mux_sel
                                        );
  output reg [1:0] Forward_A,
                                        always@(*)
  output reg [1:0] Forward_B
                                        begin
);
                                        if (MemRead idex && ((rd idex==Rs1 ifid) ||
always @(*)
                                        (rd idex==Rs2 ifid)))
begin
                                          begin
                                          PcWrite=1'b0;
if ((regwrite exmem==1'b1) &&
                                          Ifid write=1'b0;
(rd exmem!=1'b0)&&
                                          mux_sel=1'b1;
(rd_exmem==rs1_idex))
                                          end
  Forward A=2'b10;
                                        else
else if ((regwrite_memwb==1'b1)&&
                                        begin
(rd memwb!=1'b0)&&
                                          PcWrite=1'b1;
(rd memwb==rs1 idex))
                                          Ifid write=1'b1;
  Forward A=2'b01;
                                          mux sel=1'b0;
else
       Forward A=2'b00;
                                        end
                                        end
if ((regwrite_exmem==1'b1)&&
                                        endmodule
(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 IDG(
                                        module IFID(
input [31:0] instruction,
                                        input [63:0] PC in,
output reg [63:0] imm data
                                        input [31:0] instruction in,
);
                                        input clk,
always @(*)
                                        input PCSrc,
                                        input ifid_write,
begin
       case(instruction[6:5])
                                        input reset,
               2'b00:
                                        output reg [63:0] PC_out,
               begin
                                        output reg [31:0] Instruction_out
                                        );
```

```
imm data=
                                         always@(posedge clk)
{{52{instruction[31]}},instruction[31:20]
                                         begin
                                         if (PCSrc==1'b0)
};
               end
                                         begin
               2'b01:
                                                 if (ifid write==1'b1)
               begin
                                                 begin
               imm_data=
                                                 if (reset==1'b0)
{{52{instruction[31]}},instruction[31:25]
                                                         begin
,instruction[11:7]};
                                                         PC out=PC in;
                                                         Instruction out=instruction in;
               end
                                                         end
               2'b10:
                                                 end
               begin
                                         end
               imm data=
                                         else
{{52{instruction[31]}},instruction[31],ins
                                         begin
truction[7],instruction[30:25],instructio
                                                         PC_out=64'b0;
n[11:8]};
                                                         Instruction_out=32'b0;
               end
                                         end
               2'b11:
                                         end
               begin
               imm data=
                                         always@(reset)
{{52{instruction[31]}},instruction[31],ins
                                         begin
truction[7],instruction[30:25],instructio
                                                 if (reset==1'b1)
n[11:8]};
                                                         begin
               end
                                                         PC_out=64'b0;
       endcase
                                                         Instruction out=32'b0;
end
                                                         end
endmodule
                                         end
                                         endmodule
                                         module MEMWB(
module Instruction memory(
input [63:0] Inst Adress,
                                            input clk,
output reg [31:0] Instruction
                                            input reset,
);
                                            input [63:0] ReadDatain,
                                            input [63:0] ALU_Resultin,
reg [7:0]Array[23:0];
                                            input MemtoReg, RegWrite,
initial
                                            input [4:0] rd,
                                            output reg [63:0] ReadDataout,
  begin
    Array[0] = 8'b10010011;
                                            output reg [63:0] ALU Resultout,
    Array[1] = 8'b10000000;
                                            output reg [4:0] rdOut,
    Array[2] = 8'b01010000;
                                            output reg MemtoRegOut, RegWriteOut
    Array[3] = 8'b000000000;
                                         );
    Array[4] = 8'b10010011;
                                         always @(posedge clk)
    Array[5] = 8'b00000010;
                                         begin
    Array[6] = 8'b10010000;
                                            if (reset==1'b0)
    Array[7] = 8'b000000000;
                                            begin
                                            ReadDataout=ReadDatain;
```

```
Array[8] = 8'b01100011;
                                           ALU Resultout=ALU Resultin;
    Array[9] = 8'b10000110;
                                           rdOut=rd;
    Array[10] = 8'b01010000;
                                           MemtoRegOut=MemtoReg;
    Array[11] = 8'b000000000;
                                           RegWriteOut=RegWrite;
                                           end
    Array[12] = 8'b00010011;
                                         end
    Array[13] = 8'b10001010;
                                         always@(reset)
    Array[14] = 8'b10100010;
                                         begin
    Array[15] = 8'b00000101;
                                           if (reset==1'b1)
                                           begin
    Array[16] = 8'b01100011;
                                           ReadDataout=64'b0;
    Array[17] = 8'b10010100;
                                           ALU Resultout=64'b0;
    Array[18] = 8'b01010000;
                                           rdOut=5'b0;
    Array[19] = 8'b000000000;
                                           MemtoRegOut=1'b0;
                                           RegWriteOut=1'b0;
    Array[20] = 8'b10110011;
    Array[21] = 8'b10000010;
                                           end
    Array[22] = 8'b00010010;
                                         end
    Array[23] = 8'b01000000;
                                         endmodule
 end
always @ (Inst Adress)
begin
       Instruction={Array[Inst Adress+
3], Array[Inst_Adress+2],
Array[Inst Adress+1], Array[Inst Adress
]};
end
endmodule
module mux 3(
                                         module parser (
  input [1:0] sel,
                                                input [31:0] instruction,
  input [63:0] a,
                                                output reg [6:0] opcode,
 input [63:0] b,
                                                output reg [4:0] rd,
 input [63:0] c,
                                                output reg [2:0] func3,
 output reg [63:0] three_muxout
                                                output reg [4:0] rs1,
                                                output reg [4:0] rs2,
);
always @(*)
                                                output reg [6:0] func7
begin
                                         );
                                         always@(instruction)
  case(sel)
  2'b00: three muxout=a;
                                         begin
                                         assign opcode=instruction[6:0];
  2'b01: three muxout=b;
  2'b10: three muxout=c;
                                         assign rd =instruction[11:7];
  endcase
                                         assign func3=instruction[14:12];
                                         assign rs1 =instruction[19:15];
end
endmodule
                                         assign rs2=instruction[24:20];
                                         assign func7=instruction[31:25];
                                         end
                                         endmodule
```

```
module Program Counter(
                                          module tb();
input clk, reset,
                                          reg clk;
input [63:0] Pc In,
                                          reg reset;
input PCWrite,
                                          Riscv rp(.clk(clk), .reset(reset));
output reg [63:0] PC_Out
                                          initial
);
                                          begin
always @(posedge clk or posedge
                                          clk= 1'b0;
reset)
                                          reset=1'b1;
begin
                                          #7 reset =1'b0;
       if (reset)
                                          end
               PC Out<=0;
                                          always
        else
               if (PCWrite==1'b1)
                                          #5 clk=~clk;
               begin
                                          endmodule
               PC_Out<=Pc_In;
               end
end
endmodule
module registerFile(
                                          module Riscv(
input [4:0] Rs1, [4:0] Rs2, [4:0] Rd,
                                          input clk, reset
input [63:0] WriteData,
                                          );
input RegWrite,
                                          wire [63:0] b;
                                          assign b= 16'h0000000000000004;
input clk, reset,
output reg [63:0] ReadData1, reg [63:0]
ReadData2
                                          wire [63:0] PC_Out;
);
                                          wire [63:0] PC Outidex;
                                          wire [63:0] adder1 out;
reg [63:0] Array[31:0];
                                          wire [63:0] adder2 out;
initial
begin
                                          wire [63:0] adder out ex;
 Array[0] <= 64'd0;
                                          wire [63:0] mux1 out;
 Array[1]<=64'd1;
                                          wire [31:0] Instruction_m;
                                          wire [31:0] Instruction_if;
 Array[2]<=64'd2;
 Array[3]<=64'd3;
                                          wire[6:0] opcode;
 Array[4] <= 64'd4;
                                          wire[4:0] rd parser;
 Array[5]<=64'd5;
                                          wire[4:0] rd_id;
                                          wire[4:0] rd mem;
 Array[6]<=64'd6;
 Array[7]<=64'd7;
                                          wire[4:0] rd ex;
 Array[8]<=64'd8;
                                          wire[2:0] func3;
 Array[9]<=64'd9;
                                          wire[4:0] rs1;
 Array[10]<=64'd10;
                                          wire [4:0] rs2;
 Array[11]<=64'd11;
                                          wire[4:0] rs1_id;
 Array[12]<=64'd12;
                                          wire [4:0] rs2_id;
 Array[13]<=64'd13;
                                          wire[6:0] func7;
 Array[14]<=64'd14;
                                          wire Branch, MemRead, MemtoReg, MemWrite, ALUSrc,
 Array[15]<=64'd15;
                                          RegWrite;
```

```
Array[16]<=64'd16;
                                        wire Branch mux, MemRead mux, MemtoReg mux,
 Array[17]<=64'd17;
                                        MemWrite mux, ALUSrc mux, RegWrite mux;
 Array[18]<=64'd18;
                                        wire [1:0] Aluop mux;
 Array[19]<=64'd19;
                                        wire [1:0] Aluop;
 Array[20]<=64'd20;
                                        wire Branch id, MemRead id, MemtoReg id, MemWrite id,
 Array[21]<=64'd21;
                                        ALUSrc id, RegWrite id;
 Array[22]<=64'd22;
                                        wire Branch_ex, MemRead_ex, MemtoReg_ex,
 Array[23]<=64'd23;
                                        MemWrite ex, RegWrite ex;
                                        wire MemtoReg mem, RegWrite mem;
 Array[24]<=64'd24;
 Array[25]<=64'd25;
                                        wire [1:0] Aluop id;
 Array[26]<=64'd26;
                                        wire [63:0] imm data;
 Array[27]<=64'd27;
                                        wire [63:0] imm dataOut;
 Array[28]<=64'd28;
                                        wire [3:0] Operation;
 Array[29]<=64'd29;
                                        wire [63:0] mux2 out;
                                        wire [63:0] ReadData1;
 Array[30]<=64'd30;
 Array[31]<=64'd31;
                                        wire [63:0] ReadData2;
                                        wire[63:0] ReadData1Out;
end
                                        wire [63:0] ReadData2Out;
always@(negedge clk)
                                        wire [63:0] ReadData2 ex;
begin
                                        wire [63:0] Result;
       if (RegWrite==1)
                                        wire [63:0]ALU Result out;
       begin
                                        wire [63:0] ALU Resultout mem;
               Array[Rd]<=WriteData;
                                        wire Zero;
                                        wire zero ex;
       end
                                        wire[63:0] Read_data;
end
                                        wire [63:0] ReadDataout;
                                        wire [63:0] mux3_out;
                                        wire [63:0] if pc out;
always @(Rs1, Rs2, reset, Array, clk,
                                        wire [3:0] funct;
                                        wire [1:0] Forward A in;
posedge clk)
begin
                                        wire [1:0] Forward B in;
       if (reset)
                                        wire [63:0] forward b muxout;
       begin
                                        wire [63:0] alu 64 a;
               ReadData1<=64'b0;
                                        wire PcWrite ctrl;
               ReadData2<=64'b0;
                                        wire ifid write ctrl;
       end
                                        wire mux_sel_ctrl;
                                        wire PCSrc;
       else
       begin
                                        wire [63:0] control idex out;
       ReadData1<=Array[Rs1];
                                        Program Counter p1(.clk(clk),
                                        .reset(reset),.PCWrite(PcWrite ctrl),
       ReadData2<=Array[Rs2];
                                        .Pc In(mux1 out),.PC Out(PC Out));
                                        adder a1(.a(PC Out),.b(b),.out(adder1 out));
       end
end
                                        assign PCSrc=Branch ex & zero ex;
endmodule
                                        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),
.lfid write(ifid write ctrl),
.mux_sel(mux_sel_ctrl)
);
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
```