FINAL PROJECT REPORT

CE-321L/CS-330L: Computer Architecture

Array Data Sorter

Muhammad Youshay
Rabia Shahab
Sidra Aamir
Iqra Azfar

CE-321L/CS-330L: Computer Architecture

Array data Sorter

26/04/2023

Description:

We made a 5 stage pipelined processor that was used to sort an array of data in which we also introduced hazard detection system in it and resolved it using forwarding, flushing the pipeline and stalling.

Introduction:

As mentioned in description above, our problem statement was to build a 5 stage pipelined processer to sort an array, and develop in it, hazard detection and resolution system. Our objective of this project was to construct this processor, which is able to sort an array, while capable of detecting and resolving hazards, if any occurs.

Methodology:

- Explanation of the project (task wise).
    1. Firstly, for sorting data, we used bubble sort as our sorting algorithm. We converted its pseudo code to assembly language and tested it on https://venus.kvakil.me/ Once our sorting algorithm correctly sorted the sample data, we proceeded towards making processor.
    2. We modified our single cycle processor now in order to run the sorting algorithm on it. Firstly, we converted bubble sort algorithm to machine code instructions, these instructions were put in instruction memory, so instruction memory is modified. Secondly data memory is modified to adjust according to our sorting algorithm. Thirdly, a branch control unit is introduced to deal with branch (blt) type instructions, this unit was not present in the single cycle processor we made in lab 11.
    3. Now, once our single cycle processor worked properly, we had to convert it to a pipelined processor with 5 stages. This processor have 4 new modules for 4 registers, that are; IF/ID, ID/EX, EX/MEM and MEM/WB. There's a forwarding unit module that looks after forwarding technique where ever data hazard is encountered.
    4. Moving forward, we had to introduce hazard detection unit in our pipelined processor. We encountered data hazards only, and techniques are used to resolve them, that are forwarding, stalling and flushing the pipeline.

SNIPPETS OF CODE:

Note: Only some important modules are shown above, others can be seen on this Github link!

1. **SINGLE CYCLE BUBBLE SORT:**

Branch Control Unit Code:

```
module branch_control(
        input [3:0] Funct,
        input less,
        input  CarryOut,
        output reg comp
);

always @ (*)
begin
        if ((Funct[2:0] == 3'b000) && ( CarryOut == 1'b1))
                comp = 1'b1;  //for beq
        else if ((Funct[2:0] == 3'b100) && ( less == 1'b1))
                comp = 1'b1; //for blt
        else
                comp = 1'b0;
end

endmodule
```

Instruction Memory:

```
module Instruction_Memory(
        input [63:0] Inst_Address,
        output reg [31:0] instruction
);

reg [7:0] inst_mem[159:0];
initial
begin
        inst_mem[0] = 8'b10010011;
        inst_mem[1] = 8'b00000101;
        inst_mem[2] = 8'b01000000 ;
        inst_mem[3] = 8'b00000000;
        inst_mem[4] = 8'b00010011;
        inst_mem[5] = 8'b00000011 ;
        inst_mem[6] = 8'b00000000 ;
        inst_mem[7] = 8'b00000000 ;
        inst_mem[8] = 8'b10010011;
        inst_mem[9] = 8'b00000011;
        inst_mem[10] = 8'b00000000 ;
        inst_mem[11] = 8'b00000000 ;
        inst_mem[12] = 8'b10010011;
        inst_mem[13] = 8'b00000010 ;
        inst_mem[14] = 8'b00000000 ;
        inst_mem[15] = 8'b00000000 ;
```

```
        inst_mem[16] = 8'b01100011;
        inst_mem[17] = 8'b00001000 ;
        inst_mem[18] = 8'b10110011 ;
        inst_mem[19] = 8'b00001000 ;
        inst_mem[20] = 8'b10010011;
        inst_mem[21] = 8'b00001010 ;
        inst_mem[22] = 8'b00000000 ;
        inst_mem[23] = 8'b00000000 ;
        inst_mem[24] = 8'b10110011;
        inst_mem[25] = 8'b10001010  ;
        inst_mem[26] = 8'b01101010  ;
        inst_mem[27] = 8'b00000000 ;
        inst_mem[28] = 8'b10110011;
        inst_mem[29] = 8'b10001010;
        inst_mem[30] = 8'b01101010;
        inst_mem[31] = 8'b00000000;
        inst_mem[32] = 8'b10110011;
        inst_mem[33] = 8'b10001010;
        inst_mem[34] = 8'b01101010;
        inst_mem[35] = 8'b00000000;
        inst_mem[36] = 8'b10110011;
        inst_mem[37] = 8'b10001010;
        inst_mem[38] = 8'b01101010;
        inst_mem[39] = 8'b00000000;
        inst_mem[40] = 8'b10110011;
        inst_mem[41] = 8'b10001010;
        inst_mem[42] = 8'b01101010;
        inst_mem[43] = 8'b00000000;
        inst_mem[44] = 8'b10110011;
        inst_mem[45] = 8'b10001010;
        inst_mem[46] = 8'b01101010;
        inst_mem[47] = 8'b00000000;
        inst_mem[48] = 8'b10110011;
        inst_mem[49] = 8'b10001010;
        inst_mem[50] = 8'b01101010;
        inst_mem[51] = 8'b00000000;
        inst_mem[52] = 8'b10110011;
        inst_mem[53] = 8'b10001010;
        inst_mem[54] = 8'b01101010;
        inst_mem[55] = 8'b00000000;
        inst_mem[56] = 8'b00010011;
        inst_mem[57] = 8'b00000011  ;
        inst_mem[58] = 8'b00010011  ;
        inst_mem[59] = 8'b00000000  ;
        inst_mem[60] = 8'b10010011;
        inst_mem[61] = 8'b00000011 ;
        inst_mem[62] = 8'b00000000 ;
        inst_mem[63] = 8'b00000000 ;
        inst_mem[64] = 8'b10010011;
        inst_mem[65] = 8'b00000011;
        inst_mem[66] = 8'b00000011;
        inst_mem[67] = 8'b00000000;
        inst_mem[68] = 8'b11100011;
        inst_mem[69] = 8'b10000110 ;
        inst_mem[70] = 8'b10110011 ;
```

```
inst_mem[71] = 8'b11111100 ;
inst_mem[72] = 8'b00110011;
inst_mem[73] = 8'b00001011 ;
inst_mem[74] = 8'b01111011 ;
inst_mem[75] = 8'b00000000 ;
inst_mem[76] = 8'b00110011;
inst_mem[77] = 8'b00001011;
inst_mem[78] = 8'b01111011;
inst_mem[79] = 8'b00000000;
inst_mem[80] = 8'b00110011;
inst_mem[81] = 8'b00001011;
inst_mem[82] = 8'b01111011;
inst_mem[83] = 8'b00000000;
inst_mem[84] = 8'b00110011;
inst_mem[85] = 8'b00001011;
inst_mem[86] = 8'b01111011;
inst_mem[87] = 8'b00000000;
inst_mem[88] = 8'b00110011;
inst_mem[89] = 8'b00001011;
inst_mem[90] = 8'b01111011;
inst_mem[91] = 8'b00000000;
inst_mem[92] = 8'b00110011;
inst_mem[93] = 8'b00001011;
inst_mem[94] = 8'b01111011;
inst_mem[95] = 8'b00000000;
inst_mem[96] = 8'b00110011;
inst_mem[97] = 8'b00001011;
inst_mem[98] = 8'b01111011;
inst_mem[99] = 8'b00000000;
inst_mem[100] = 8'b00110011;
inst_mem[101] = 8'b00001011;
inst_mem[102] = 8'b01111011;
inst_mem[103] = 8'b00000000;
inst_mem[104] = 8'b10010011;
inst_mem[105] = 8'b10000011 ;
inst_mem[106] = 8'b00010011 ;
inst_mem[108] = 8'b00110011;
inst_mem[109] = 8'b10000111 ;
inst_mem[110] = 8'b01011010 ;
inst_mem[111] = 8'b00000000 ;
inst_mem[112] = 8'b10110011;
inst_mem[113] = 8'b00000111  ;
inst_mem[114] = 8'b01011011  ;
inst_mem[115] = 8'b00000000 ;
inst_mem[116] = 8'b00010011;
inst_mem[117] = 8'b00001011  ;
inst_mem[118] = 8'b00000000  ;
inst_mem[119] = 8'b00000000 ;
inst_mem[120] = 8'b00000011;
inst_mem[121] = 8'b00111000  ;
inst_mem[122] = 8'b00000111  ;
inst_mem[123] = 8'b00000000  ;
inst_mem[124] = 8'b10000011;
inst_mem[125] = 8'b10111000 ;
inst_mem[126] = 8'b00000111 ;
```

```
        inst_mem[127] = 8'b00000000 ;
        inst_mem[128] = 8'b11100011;
        inst_mem[129] = 8'b00000010 ;
        inst_mem[130] = 8'b00011000 ;
        inst_mem[131] = 8'b11111101 ;
        inst_mem[132] = 8'b11100011;
        inst_mem[133] = 8'b01000000 ;
        inst_mem[134] = 8'b00011000 ;
        inst_mem[135] = 8'b11111101 ;
        inst_mem[136] = 8'b00110011;
        inst_mem[137] = 8'b00001001 ;
        inst_mem[138] = 8'b00000000 ;
        inst_mem[139] = 8'b00000001 ;
        inst_mem[140] = 8'b00110011;
        inst_mem[141] = 8'b00001000 ;
        inst_mem[142] = 8'b00010000 ;
        inst_mem[143] = 8'b00000001 ;
        inst_mem[144] = 8'b00100011;
        inst_mem[145] = 8'b00110000 ;
        inst_mem[146] = 8'b00000111 ;
        inst_mem[147] = 8'b00000001 ;
        inst_mem[148] = 8'b10110011;
        inst_mem[149] = 8'b00001000 ;
        inst_mem[150] = 8'b00100000 ;
        inst_mem[151] = 8'b00000001 ;
        inst_mem[152] = 8'b00100011;
        inst_mem[153] = 8'b10110000   ;
        inst_mem[154] = 8'b00010111   ;
        inst_mem[155] = 8'b00000001   ;
        inst_mem[156] = 8'b11100011 ;
        inst_mem[157] = 8'b00000100  ;
        inst_mem[158] = 8'b00000000  ;
        inst_mem[159] = 8'b11111010 ;


   end
        always @(Inst_Address)
        begin

        instruction={inst_mem[Inst_Address+3],inst_mem[Inst_Address+2],inst_mem[Inst_Addre
ss+1],inst_mem[Inst_Address]};
        end
endmodule
```

Data Memory

```
module Data_Memory(
          input [63:0] Mem_Addr, Write_Data,
          input clk, MemWrite, MemRead,
          output reg [63:0] Read_Data,
          output [63:0] element1,
    output [63:0] element2,
    output [63:0] element3,
    output [63:0] element4
);

 reg [7:0] DataMemory [63:0];
  integer i;
  integer x;

   initial
    begin
      for (i = 0; i < 64; i = i + 1)
      begin
        DataMemory[i] = 0;
      end
    end
  assign element1 ={DataMemory[7],DataMemory[6],  DataMemory[5] , DataMemory[4] ,
DataMemory[3] , DataMemory[2] , DataMemory[1] , DataMemory[0]};
  assign element2 = {DataMemory[15], DataMemory[14], DataMemory[13] ,
DataMemory[12] , DataMemory[11], DataMemory[10] , DataMemory[9] ,
DataMemory[8]};
assign element3 =
{DataMemory[23],DataMemory[22],DataMemory[21],DataMemory[20],DataMemory[19]
,DataMemory[18],DataMemory[17],DataMemory[16]} ;
 assign element4= {DataMemory[31] ,DataMemory[30], DataMemory [29],
DataMemory[28], DataMemory[27] , DataMemory[26] ,DataMemory[25]
,DataMemory[24]};

  always @( posedge clk)
    begin
    if (MemWrite == 1'b1)
    begin
      DataMemory[Mem_Addr] = Write_Data[7:0];
      DataMemory[Mem_Addr + 1] = Write_Data[15:8];
      DataMemory[Mem_Addr + 2] = Write_Data[23:16];
      DataMemory[Mem_Addr + 3] = Write_Data[31:24];
      DataMemory[Mem_Addr + 4] = Write_Data[39:32];
      DataMemory[Mem_Addr + 5] = Write_Data[47:40];
      DataMemory[Mem_Addr + 6] = Write_Data[55:48];
      DataMemory[Mem_Addr + 7] = Write_Data[63:56];
    end
    end
  always @(*)
```

```
    begin
      if (MemRead == 1'b1)
        begin
      Read_Data = {DataMemory[Mem_Addr + 7], DataMemory[Mem_Addr +
6],DataMemory[Mem_Addr + 5], DataMemory[Mem_Addr + 4],
DataMemory[Mem_Addr + 3], DataMemory[Mem_Addr + 2],DataMemory[Mem_Addr +
1], DataMemory[Mem_Addr]};
        end

    end
endmodule
```

Top Module of Single Cycle Processor:

```
module Single_Cycle_Processor(
    input clk, reset,
    output wire [63:0] element1,element2,element3,element4
);
    // Inputs
    wire [31:0] instruction;

    // Control signals
    wire [6:0] opcode;
    wire [4:0] rs1;
    wire [4:0] rs2;
    wire [4:0] rd;
    wire [1:0] ALUOp;
//    wire [3:0] Funct;
    wire [2:0] funct3;
    wire [6:0] funct7;
    wire [3:0] Operation;
    wire less, comp, Branch, Zero, MemRead, MemToReg, MemWrite, ALUSrc, RegWrite,
data_memory_select;

    // Intermediate signals
    wire [63:0] imm_data;
    wire [63:0] register_file_mux_out;
    wire [63:0] alu_result;
    wire [63:0] data_memory_read_out;
    wire [63:0] WriteData;
    wire [63:0] readData1;
    wire [63:0] readData2;

    // Outputs
    wire [63:0] pc_default_out;
    wire [63:0] pc_branch_out;
    wire [63:0] pc_mux_out;
    wire [63:0] PC_Out;
Instruction_Memory IM(.Inst_Address(PC_Out),.instruction(instruction));
ImmDataExtractor ID(.instruction(instruction),.imm_data(imm_data));
```

```
InstructionParser
IP(.opcode(opcode),.instruction(instruction),.rs1(rs1),.rs2(rs2),.rd(rd),.funct3(funct3),.funct7(funct7
));
ALU_Control ALU_C(.ALUOp(ALUOp),.Funct({instruction[30],
instruction[14:12]}),.Operation(Operation));
Control_Unit
CU(.opcode(opcode),.Branch(Branch),.ALUOp(ALUOp),.RegWrite(RegWrite),.MemRead(MemR
ead),.MemToReg(MemToReg),.MemWrite(MemWrite),.ALUSrc(ALUSrc));
branch_control BC(.Funct({instruction[30],
instruction[14:12]}),.less(less),.CarryOut(Zero),.comp(comp));
registerFile
rf(.WriteData(WriteData),.rs1(rs1),.rs2(rs2),.rd(rd),.readData1(readData1),.readData2(readData2),.
RegWrite(RegWrite),.clk(clk),.reset(reset));
mux Mux_3_RF(.sel(ALUSrc),.a(imm_data),.b(readData2),.data_out(register_file_mux_out));
mux
Mux_2_DM(.sel(MemToReg),.a(data_memory_read_out),.b(alu_result),.data_out(WriteData));
Data_Memory
DM(.MemWrite(MemWrite),.MemRead(MemRead),.Mem_Addr(alu_result),.Write_Data(readDat
a2),.clk(clk),.Read_Data(data_memory_read_out), .element1(element1), .element2(element2),
.element3(element3), .element4(element4));
alu_64
ALU(.a(readData1),.b(register_file_mux_out),.ALUOp(Operation),.Result(alu_result),.CarryOut(Z
ero),.less(less));
mux PC_Mux(.sel(Branch & comp),
.a(pc_branch_out),.b(pc_default_out),.data_out(pc_mux_out));
Adder Pc_Adder(.a(PC_Out),.b(64'd4),.out(pc_default_out));
Adder Pc_branch_Adder(.a(PC_Out),.b(imm_data << 1), .out(pc_branch_out));
Program_Counter Prog_count(.PC_In(pc_mux_out),.PC_Out(PC_Out),.clk(clk),.reset(reset));
```

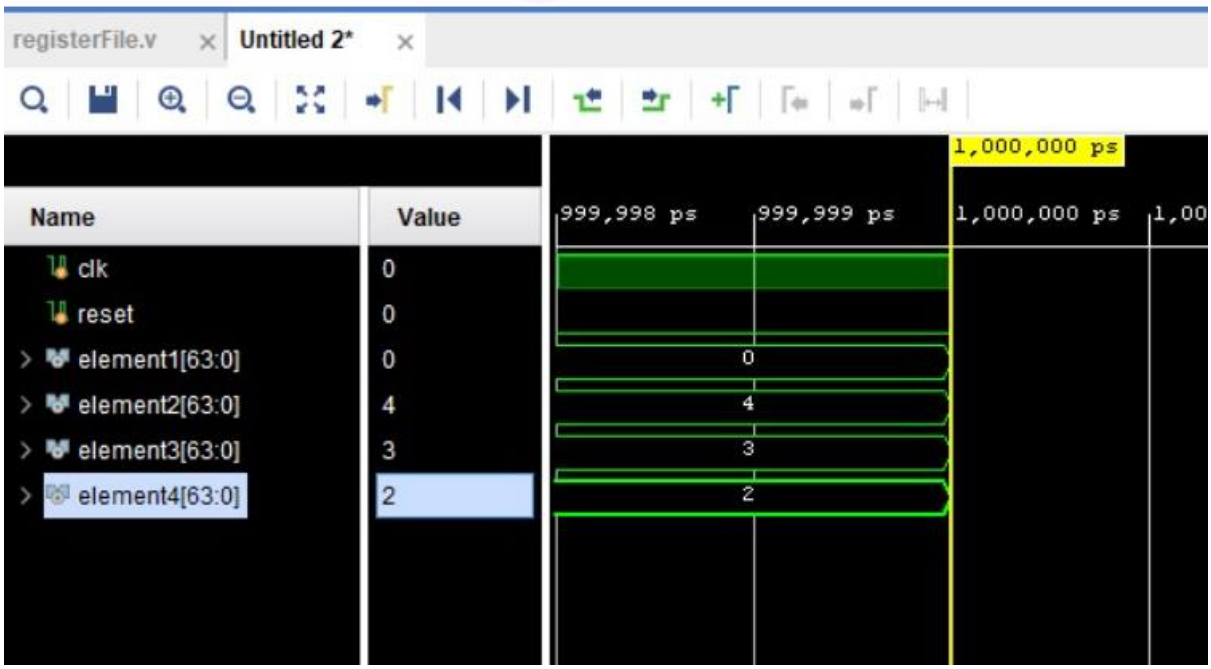
Register File

```
module registerFile(
        input [63:0] WriteData,
        input [4:0] rs1,
        input [4:0] rs2,
        input [4:0] rd,
        output reg [63:0] readData1,
        output reg [63:0] readData2,
        input clk, reset, RegWrite
);

reg [63:0] Registers [31:0];
integer i;
initial begin
for (i=0; i<64; i=i+1)
begin
Registers [i] = 8'h0;
end
Registers [11] = 8'h8;
end
always @(*)
   begin
     if (reset == 1)
       begin
```

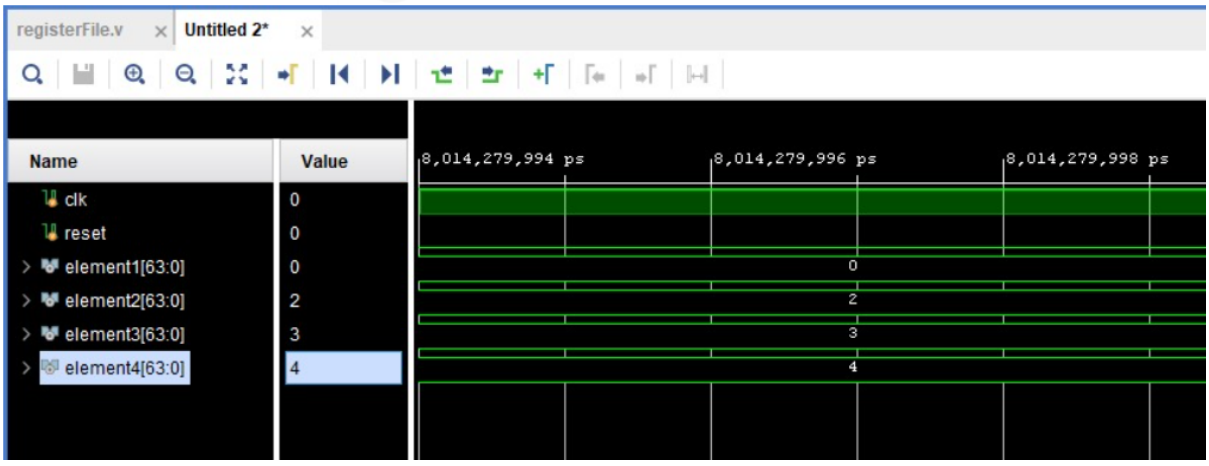
```
        readData1 = 64'b0;
        readData2 = 64'b0;
      end
    else
      begin
        readData1 = Registers[rs1];
        readData2 = Registers[rs2];
      end
  end

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

endmodule
```

SIMULATION results (Single Cycle)

Before (unsorted)



After (sorted)




```
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:21 ; elapsed = 00:00:27 . Memory (MB): peak = 1024.426 ; gain = 0.000
```

2. Pipelined Processor Testing:

Top Module

```
module Pipelined_Processor(input clk, reset);

//IF/ID
wire [31:0] IF_ID_Ins;
wire [63:0] IF_ID_PC;
//ID/EX
wire [3:0] ID_EX_Instruction;
wire [4:0] ID_EX_rs1;
wire [4:0] ID_EX_rs2;
wire [4:0] ID_EX_rd;
wire [63:0] ID_EX_imm_data;
wire [63:0] ID_EX_ReadData1;
wire [63:0] ID_EX_ReadData2;
wire [63:0] ID_EX_PC;
wire [1:0] ID_EX_ALUOp;
wire ID_EX_ALUSrc;
wire ID_EX_Branch;
wire ID_EX_MemRead;
wire ID_EX_MemtoReg;
wire ID_EX_MemWrite;
wire ID_EX_RegWrite;
//EX/MEM
wire [4:0] EX_Mem_rd;
wire [63:0] EX_Mem_Adder, EX_Mem_ALU_Result, EX_Mem_ForwardB_MUX;
wire EX_Mem_Branch, EX_Mem_MemRead, EX_Mem_MemtoReg, EX_Mem_MemWrite,
EX_Mem_RegWrite;
wire[63:0] EX_Mem_Zero;
//MEM/WB
wire [63:0] Mem_WB_Read_Data;
wire [63:0] Mem_WB_ALU_Result;
wire[4:0] Mem_WB_rd;
```

```
wire Mem_WB_MemtoReg;
wire Mem_WB_RegWrite;
//Others
wire [1:0] Fwd_A, Fwd_B;
wire [63:0] MuxA_Out, MuxB_Out;
wire [63:0] mux_out_1;
wire [63:0] mux_out_2;
wire [63:0] mux_out_3;
wire [63:0] PC_out;
wire [63:0] out_1;
wire [63:0] out_2;
wire [31:0] Instruction;
wire [6:0] opcode;
wire [4:0] rd;
wire [2:0] funct3;
wire [4:0] rs1;
wire [4:0] rs2;
wire [6:0] funct7;
wire [63:0] ReadData1;
wire [63:0] ReadData2;
wire [1:0] ALUOp;
wire [63:0] immediate;
wire [3:0] Operation;
wire [63:0] Result;
wire [63:0] Read_Data;
wire branch;
wire MemRead;
wire MemtoReg;
wire MemWrite;
wire ALUSrc;
wire RegWrite;
wire Zero;


Program_Counter pc (.clk(clk), .reset(reset), .PC_In(mux_out_1), .PC_Out(PC_out));

Adder add1(.a(PC_out), .b(64'd4), .out(out_1));

Adder add2(.a(ID_EX_PC), .b(ID_EX_imm_data << 1), .out(out_2));

Mux a(.a(out_1), .b(EX_Mem_Adder), .SEL(EX_Mem_Branch & EX_Mem_Zero),
.data_out(mux_out_1));

Mux b(.a(MuxB_Out), .b(ID_EX_imm_data), .SEL(ID_EX_ALUSrc),
.data_out(mux_out_2));

Mux c(.a(Mem_WB_ALU_Result), .b(Mem_WB_Read_Data),
.SEL(Mem_WB_MemtoReg), .data_out(mux_out_3));

mux3by1
muxA(.data_out(MuxA_Out),.S(Fwd_A),.A(ID_EX_ReadData1),.B(mux_out_3),.C(EX_Me
m_ALU_Result));
```

```
mux3by1
muxB(.data_out(MuxB_Out),.S(Fwd_B),.A(ID_EX_ReadData2),.B(mux_out_3),.C(EX_Me
m_ALU_Result));

Instruction_Memory im(.Inst_Address(PC_out), .Instruction(Instruction));

Instruction_Parser IP(.instruction(IF_ID_Ins), .opcode(opcode), .rd(rd), .funct3(funct3),
.rs1(rs1), .rs2(rs2), .funct7(funct7) );

Control_Unit CU(.Opcode(opcode), .ALUOp(ALUOp), .Branch(branch),
.MemRead(MemRead), .MemtoReg(MemtoReg), .MemWrite(MemWrite),
.ALUSrc(ALUSrc), .RegWrite(RegWrite));

Register_File rf(.WriteData(mux_out_3),.rs1(rs1), .rs2(rs2),
.rd(Mem_WB_rd),.RegWrite(RegWrite),.clk(clk),.reset(reset), .ReadData1(ReadData1),
.ReadData2(ReadData2));

Immediate_Data_Generator ig(.instruction(IF_ID_Ins), .immediate(immediate));

ALU_Control alu_c(.ALUOp(ID_EX_ALUOp), .Funct(ID_EX_Instruction),
.Operation(Operation));

ALU_64_bit alu (.a(MuxA_Out), .b(mux_out_2),.ALUOp(Operation), .Result(Result),
.Zero(Zero));

Data_Memory dm(.clk(clk),.Mem_Addr(EX_Mem_ALU_Result),
.Write_Data(EX_Mem_ForwardB_MUX), .MemWrite(EX_Mem_MemWrite),
.MemRead(EX_Mem_MemRead),.Read_Data(Read_Data));


IF_ID ifid(.clk(clk), .reset(reset), .Instruction(Instruction),.PC(PC_out),
.IF_ID_Ins(IF_ID_Ins), .IF_ID_PC(IF_ID_PC));

ID_EX ID_EX(
   .clk(clk), .reset(reset),
.IF_ID_Instruction({IF_ID_Ins[30],IF_ID_Ins[14:12]}),.rs1(rs1),.rs2(rs2),.rd(rd),.imm_data(i
mmediate),.ReadData1(ReadData1),.ReadData2(ReadData2),.PC(IF_ID_PC),.ALUOp(ALU
Op),.Branch(branch), .MemRead(MemRead), .MemtoReg(MemtoReg),
.MemWrite(MemWrite),
.ALUSrc(ALUSrc),.RegWrite(RegWrite),.ID_EX_Instruction(ID_EX_Instruction),.ID_EX_r
s1(ID_EX_rs1),.ID_EX_rs2(ID_EX_rs2),.ID_EX_rd(ID_EX_rd),.ID_EX_imm_data(ID_EX
_imm_data),.ID_EX_ReadData1(ID_EX_ReadData1),.ID_EX_ReadData2(ID_EX_ReadDat
a2),.ID_EX_PC(ID_EX_PC),.ID_EX_ALUOp(ID_EX_ALUOp),.ID_EX_ALUSrc(ID_EX_
ALUSrc),.ID_EX_Branch(ID_EX_Branch),.ID_EX_MemRead(ID_EX_MemRead),.ID_EX
_MemtoReg(ID_EX_MemtoReg),.ID_EX_MemWrite(ID_EX_MemWrite),.ID_EX_RegWri
te(ID_EX_RegWrite));

Forwarding_Unit
FU(.ID_EX_rs1(ID_EX_rs1),.ID_EX_rs2(ID_EX_rs2),.EX_Mem_rd(EX_Mem_rd),.EX_Me
m_RegWrite(EX_Mem_RegWrite),.Mem_WB_rd(Mem_WB_rd),.Mem_WB_RegWrite(Me
m_WB_RegWrite),.Forward_A(Fwd_A),.Forward_B(Fwd_B)  );

EX_Mem EXMem(.clk(clk), .reset(reset),.ID_EX_Branch(ID_EX_Branch),
.ID_EX_MemRead(ID_EX_MemRead),.ID_EX_MemtoReg(ID_EX_MemtoReg),
.ID_EX_MemWrite(ID_EX_MemWrite), .ID_EX_RegWrite(ID_EX_RegWrite),
```

```
        .Adder(out_2),.Zero(Zero),.ALU_Rslt(Result),.ForwardB_MUX(MuxB_Out),.ID_EX_rd(ID
        _EX_rd),.EX_Mem_Branch(EX_Mem_Branch),.EX_Mem_MemRead(EX_Mem_MemRead
        ),.EX_Mem_MemtoReg(EX_Mem_MemtoReg),
        .EX_Mem_MemWrite(EX_Mem_MemWrite),.EX_Mem_RegWrite(EX_Mem_RegWrite),.E
        X_Mem_Adder(EX_Mem_Adder),.EX_Mem_Zero(EX_Mem_Zero),
        .EX_Mem_ALU_Result(EX_Mem_ALU_Result),.EX_Mem_ForwardB_MUX(EX_Mem_F
        orwardB_MUX),.EX_Mem_rd(EX_Mem_rd));

        Mem_WB MemWB(.clk(clk),
        .reset(reset),.EX_Mem_MemtoReg(EX_Mem_MemtoReg),.EX_Mem_RegWrite(EX_Mem_
        RegWrite),.Read_Data(Read_Data),
        .EX_Mem_ALU_Result(EX_Mem_ALU_Result),.EX_Mem_rd(EX_Mem_RD),.Mem_WB_
        MemtoReg(Mem_WB_MemtoReg),.Mem_WB_RegWrite(Mem_WB_RegWrite),.Mem_WB
        _Read_Data(Mem_WB_Read_Data),.Mem_WB_ALU_Result(Mem_WB_ALU_Result),.Me
        m_WB_rd(Mem_WB_rd));



        always @(posedge clk)
           begin
           $monitor(
              "Instruction =%b",Instruction,
              " RS1 =%d",rs1,
              " RS2 =%d",rs2,
              " RD =%d",rd,
              " Result =%d", Result,
              " ForwardA =%d",Fwd_A,
              " ForwardB =%d",Fwd_B
              );
            end


endmodule
```

Forwarding Unit:

```
        module Forwarding_Unit(
          input [4:0] ID_EX_rs1,
          input [4:0] ID_EX_rs2,
          input [4:0] EX_Mem_rd,//
          input EX_Mem_RegWrite,
          input [4:0] Mem_WB_rd,
          input Mem_WB_RegWrite,
          output reg [1:0] Forward_A,
          output reg [1:0] Forward_B
        );

          always @(*) begin
            // Check for hazard on rs1/Forward_A from EX stage
            Forward_A = (EX_Mem_RegWrite && EX_Mem_rd != 0 &&
        (EX_Mem_rd == ID_EX_rs1)) ? 2'b10 :
                    // Check for hazard on rs1/Forward_A from MEM stage
                    (Mem_WB_RegWrite && Mem_WB_rd != 0 &&
        (Mem_WB_rd == ID_EX_rs1) &&
```

```
                    !(EX_Mem_RegWrite && EX_Mem_rd != 0 && EX_Mem_rd
        == ID_EX_rs1)) ? 2'b01 :
                    // No hazard on rs1/Forward_A
                    2'b00;

            // Check for hazard on rs2/Forward_B from EX stage
            Forward_B = (EX_Mem_RegWrite && EX_Mem_rd != 0 &&
        (EX_Mem_rd == ID_EX_rs2)) ? 2'b10 :
                    // Check for hazard on rs2/Forward_B from MEM stage
                    (Mem_WB_RegWrite && Mem_WB_rd != 0 &&
        (Mem_WB_rd == ID_EX_rs2) &&
                    !(EX_Mem_RegWrite && EX_Mem_rd != 0 && EX_Mem_rd
        == ID_EX_rs2)) ? 2'b01 :
                    // No hazard on rs2/Forward_B
                    2'b00;
          end
endmodule
```

Data Memory

```
module Data_Memory
(
  input clk,
  input [63:0] Mem_Addr,
  input [63:0] Write_Data,
  input MemWrite,
  input MemRead,
  output reg [63:0] Read_Data
);

  reg [7:0] Data_Memory [63:0];
  integer i;
  initial
    begin
      for (i=0; i<64; i=i+1)
        begin
          Data_Memory [i] = 8'd0;
        end
    end
always @(*)
    begin
      if(MemRead == 1)
        begin Read_Data = {
          Data_Memory[Mem_Addr+7], Data_Memory[Mem_Addr+6],
          Data_Memory[Mem_Addr+5], Data_Memory[Mem_Addr+4],
          Data_Memory[Mem_Addr+3], Data_Memory[Mem_Addr+2],
          Data_Memory[Mem_Addr+1], Data_Memory[Mem_Addr+0]
        };
        end
    end
always @(posedge clk)
    begin
      if (MemWrite == 1)
        begin
```

```
        Data_Memory[Mem_Addr+7] = Write_Data[63:56];
        Data_Memory[Mem_Addr+6] = Write_Data[55:48];
        Data_Memory[Mem_Addr+5] = Write_Data[47:40];
        Data_Memory[Mem_Addr+4] = Write_Data[39:32];
        Data_Memory[Mem_Addr+3] = Write_Data[31:24];
        Data_Memory[Mem_Addr+2] = Write_Data[23:16];
        Data_Memory[Mem_Addr+1] = Write_Data[15:8];
        Data_Memory[Mem_Addr+0] = Write_Data[7:0];
      end
   end
endmodule
```

Instruction Memory

```
          module Instruction_Memory(
            input [63:0] Inst_Address,
            output reg [31:0] Instruction

          );

            reg [7:0] Instruction_Memory [7:0];

            initial
              begin
          //    //TESTCASE 1
          //       //Addi x1,x0,8
          //    Instruction_Memory[3] = 8'b00000000;
          //    Instruction_Memory[2] = 8'b10000000;
          //    Instruction_Memory[1] = 8'b00000000;
          //    Instruction_Memory[0] = 8'b10010011;

          //       //Addi x4,x0, x1
          //    Instruction_Memory[7] = 8'b00000000;
          //    Instruction_Memory[6] = 8'b00010000;
          //    Instruction_Memory[5] = 8'b00000010;
          //    Instruction_Memory[4] = 8'b00110011;
          //TESTCASE 2
                //00800093 //Addi x1,x0,8
              Instruction_Memory[3] = 8'b00000000;
              Instruction_Memory[2] = 8'b10000000;
              Instruction_Memory[1] = 8'b00000000;
              Instruction_Memory[0] = 8'b10010011;

              //00008213 //Addi x4,x1,2
              Instruction_Memory[7] = 8'b00000000;
              Instruction_Memory[6] = 8'b00000000;
              Instruction_Memory[5] = 8'b10000010;
              Instruction_Memory[4] = 8'b00010011;



              end

            always @ (Inst_Address)
              begin
```

```
        Instruction = {Instruction_Memory[Inst_Address+3],
      Instruction_Memory[Inst_Address+2],
      Instruction_Memory[Inst_Address+1],
      Instruction_Memory[Inst_Address+0]};
        end

      endmodule
```

Test Cases:

```
//    //TESTCASE 1         //Addi x1,x0,8
    Instruction_Memory[3] = 8'b00000000;
    Instruction_Memory[2] = 8'b10000000;
    Instruction_Memory[1] = 8'b00000000;
    Instruction_Memory[0] = 8'b10010011;

       //Addi x4,x0, x1
    Instruction_Memory[7] = 8'b00000000;
    Instruction_Memory[6] = 8'b00010000;
    Instruction_Memory[5] = 8'b00000010;
    Instruction_Memory[4] = 8'b00110011;
////TESTCASE 2
//     //00800093 //Addi x1,x0,8
//    Instruction_Memory[3] = 8'b00000000;
//    Instruction_Memory[2] = 8'b10000000;
//    Instruction_Memory[1] = 8'b00000000;
//    Instruction_Memory[0] = 8'b10010011;

//    //00008213 //Addi x4,x1,2
//    Instruction_Memory[7] = 8'b00000000;
//    Instruction_Memory[6] = 8'b00000000;
//    Instruction_Memory[5] = 8'b10000010;
//    Instruction_Memory[4] = 8'b00010011;
```

TCL Console output for Test Case 1

```
# run 1000ns
Instruction =00000000100000000000000010010011 RS1 = 0 RS2 = 0 RD = 0 Result =                0 ForwardA =0 ForwardB =0
Instruction =00000000100000000000000010010011 RS1 = 0 RS2 = 8 RD = 1 Result =                0 ForwardA =0 ForwardB =0
Instruction =00000000000010000000001000110011 RS1 = 0 RS2 = 8 RD = 1 Result =                8 ForwardA =0 ForwardB =0
Instruction =xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx RS1 = 0 RS2 = 1 RD = 4 Result =                8 ForwardA =0 ForwardB =0
Instruction =xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx RS1 = x RS2 = x RD = x Result =                x ForwardA =X ForwardB =2
Instruction =xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx RS1 = x RS2 = x RD = x Result =                x ForwardA =x ForwardB =x
Instruction =xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx RS1 = x RS2 = x RD = x Result =                x ForwardA =X ForwardB =X
Instruction =xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx RS1 = x RS2 = x RD = x Result =                x ForwardA =0 ForwardB =0
```

CL Console output for Test Case 2

```
Instruction =00000000100000000000000010010011 RS1 = 0 RS2 = 0 RD = 0 Result =                0 ForwardA =0 ForwardB =0
Instruction =00000000100000000000000010010011 RS1 = 0 RS2 = 8 RD = 1 Result =                0 ForwardA =0 ForwardB =0
Instruction =00000000000000001000001000010011 RS1 = 0 RS2 = 8 RD = 1 Result =                8 ForwardA =0 ForwardB =0
Instruction =xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx RS1 = 1 RS2 = 0 RD = 4 Result =                8 ForwardA =0 ForwardB =0
Instruction =xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx RS1 = x RS2 = x RD = x Result =                8 ForwardA =2 ForwardB =X
```

3. Hazard Detection in Pipelined Processor (Part3):

Top Module:

```
module Pipelined_Processor_Task_3
(clk, reset,element1,element2,element3,element4);
input clk,reset;
output wire [63:0] element1,element2,element3,element4;
wire [63:0] PC_to_IM;
wire [31:0] IM_to_IFID;
wire [6:0] opcode_out;
wire[4:0] rd_out;
wire [2:0] funct3_out;
wire [6:0] funct7_out;
wire [4:0] rs1_out, rs2_out;
wire Branch_out, MemRead_out, MemtoReg_out, MemWrite_out, ALUSrc_out, RegWrite_out;
wire Is_Greater_out;
wire [1:0] ALUOp_out;
wire [63:0] mux_to_reg;
wire [63:0] mux_to_pc_in;
wire [3:0] ALU_C_Operation;
wire [63:0] adder_send_exmem;
wire [63:0] ReadData1_out, ReadData2_out;
wire [63:0] imm_data_out;
wire [63:0] int_4 = 64'd4;
wire [63:0] PC_plus_4_to_mux;
wire [63:0] alu_mux_out;
wire [63:0] alu_result_out;
wire zero_out;
wire [63:0] imm_to_adder;
wire [63:0] imm_adder_to_mux;
wire [63:0] DM_Read_Data_out;
wire [1:0] fwd_A_out, fwd_B_out;
wire [63:0] muxout_a, muxout_b;
wire pc_mux_sel_wire;
wire PCWrite_out;
wire IDEX_Branch_out, IDEX_MemRead_out, IDEX_MemtoReg_out,
IDEX_MemWrite_out, IDEX_ALUSrc_out, IDEX_RegWrite_out;
wire [63:0] IDEX_PC_addr, IDEX_ReadData1_out, IDEX_ReadData2_out,IDEX_imm_data_out;
wire [3:0] IDEX_funct_in;
wire [4:0] IDEX_rd_out, IDEX_rs1_out, IDEX_rs2_out;
wire [1:0] IDEX_ALUOp_out;
wire [63:0] IFID_PC_addr;
wire [31:0] IFID_parse;
wire controlmux_sel_bit;
wire IFID_Write_out;
wire EXMEM_Branch_out, EXMEM_MemRead_out,
EXMEM_MemtoReg_out,EXMEM_MemWrite_out, EXMEM_RegWrite_out;
wire EXMEM_zero_out, EXMEM_Is_Greater_out;
wire [63:0] EXMEM_PC_plus_imm, EXMEM_alu_result_out,EXMEM_ReadData2_out;
wire [3:0] EXMEM_funct_in;
wire [4:0] EXMEM_rd_out;
wire Flush_signal;
wire [1:0] ALUop_IDEXin;
```

```
wire [3:0] funct_in;
wire MEMWB_MemtoReg_out, MEMWB_RegWrite_out;
wire [63:0] MEMWB_DM_Read_Data_out, MEMWB_alu_result_out;
wire [4:0] MEMWB_rd_out;
mux_2by1 mux_from_pc( .a(EXMEM_PC_plus_imm),.b(PC_plus_4_to_mux),
.sel(pc_mux_sel_wire), .data_out(mux_to_pc_in));
Program_Counter prog_count (.clk(clk), .reset(reset), .PCWrite(PCWrite_out),
.PC_In(mux_to_pc_in), .PC_Out(PC_to_IM));
Adder from_pc_adder( .A(PC_to_IM), .B(int_4), .out(PC_plus_4_to_mux));
Instruction_Memory inst_mem(.Inst_Address(PC_to_IM), .Instruction(IM_to_IFID));
IF_ID ifid(.clk(clk), .Flush(Flush_signal), .IFID_Write(IFID_Write_out),  .PC_addr(PC_to_IM),
.Instruc(IM_to_IFID), .PC_store(IFID_PC_addr), .Instr_store(IFID_parse));
Hazard_Detection hazard_det_unit(.IDEX_rd(IDEX_rd_out), .IFID_rs1(rs1_out),
.IFID_rs2(rs2_out), .IDEX_MemRead(IDEX_MemRead_out),
.IDEX_mux_out(controlmux_sel_bit), .IFID_Write(IFID_Write_out), .PCWrite(PCWrite_out));
Instruction_Parser instr_parser(.instruc(IFID_parse), .opcode(opcode_out), .rd(rd_out),
.funct3(funct3_out), .rs1(rs1_out), .rs2(rs2_out), .funct7(funct7_out));
Control_Unit cont_unit(.Opcode(opcode_out),  .Branch(Branch_out),  .MemRead(MemRead_out),
.MemtoReg(MemtoReg_out), .MemWrite(MemWrite_out), .ALUSrc(ALUSrc_out),
.RegWrite(RegWrite_out), .ALUOp(ALUOp_out));
Register_File reg_file(.clk(clk),  .reset(reset), .RegWrite(MEMWB_RegWrite_out),
.WriteData(mux_to_reg), .RS1(rs1_out), .RS2(rs2_out),  .RD(MEMWB_rd_out),
.ReadData1(ReadData1_out), .ReadData2(ReadData2_out));
imm_data_generator imm_data_gen(.instruc(IFID_parse), .imm_data(imm_data_out));
ID_EX idex_reg(.clk(clk), .Flush(Flush_signal), .PC_addr(IFID_PC_addr),
.read_data1(ReadData1_out), .read_data2(ReadData2_out), .imm_val(imm_data_out),
.funct_in(funct_in), .rd_in(rd_out),  .rs1_in(rs1_out), .rs2_in(rs2_out),
.RegWrite(RegWrite_IDEXin), .MemtoReg(MemtoReg_IDEXin), .Branch(Branch_IDEXin),
.MemWrite(MemWrite_IDEXin), .MemRead(MemRead_IDEXin), .ALUSrc(ALUSrc_IDEXin),
.ALU_op(ALUop_IDEXin), .PC_addr_store(IDEX_PC_addr),
.read_data1_store(IDEX_ReadData1_out), .read_data2_store(IDEX_ReadData2_out),
.imm_val_store(IDEX_imm_data_out), .funct_in_store(IDEX_funct_in),
.rd_in_store(IDEX_rd_out), .rs1_in_store(IDEX_rs1_out), .rs2_in_store(IDEX_rs2_out),
.RegWrite_store(IDEX_RegWrite_out), .MemtoReg_store(IDEX_MemtoReg_out),
.Branch_store(IDEX_Branch_out),  .MemWrite_store(IDEX_MemWrite_out),
.MemRead_store(IDEX_MemRead_out), .ALUSrc_store(IDEX_ALUSrc_out),
.ALU_op_store(IDEX_ALUOp_out));
ALU_Control alu_control(.ALUOp(IDEX_ALUOp_out), .Funct(IDEX_funct_in),
.Operation(ALU_C_Operation));
mux_2by1 mux_with_ALU(.a(IDEX_imm_data_out), .b(muxout_b), .sel(IDEX_ALUSrc_out),
.data_out(alu_mux_out));
mux_3by1 fwd_a_mux(.a(IDEX_ReadData1_out), .b(mux_to_reg), .c(EXMEM_alu_result_out),
.sel(fwd_A_out), .data_out(muxout_a));
mux_3by1 fwd_b_mux(.a(IDEX_ReadData2_out), .b(mux_to_reg), .c(EXMEM_alu_result_out),
.sel(fwd_B_out), .data_out(muxout_b));
ALU_64_bit alu_64(.a(muxout_a), .b(alu_mux_out), .ALUOp(ALU_C_Operation),
.Result(alu_result_out), .Zero(zero_out), .Greater_Check(Is_Greater_out));
Forwarding_Unit fwd_unit(.EXMEM_rd(EXMEM_rd_out), .MEMWB_rd(MEMWB_rd_out),
.IDEX_rs1(IDEX_rs1_out), .IDEX_rs2(IDEX_rs2_out),
.EXMEM_RegWrite(EXMEM_RegWrite_out), .EXMEM_MemtoReg(EXMEM_MemtoReg_out),
.MEMWB_RegWrite(MEMWB_RegWrite_out), .fwd_A(fwd_A_out), .fwd_B(fwd_B_out));
Adder imm_to_PC(.A(IDEX_PC_addr), .B(imm_to_adder), .out(adder_send_exmem));
EX_MEM ex_mem_reg(.clk(clk), .Flush(Flush_signal), .RegWrite(IDEX_RegWrite_out),
.MemtoReg(IDEX_MemtoReg_out), .Branch(IDEX_Branch_out), .Zero(zero_out),
.Greater_Check(Is_Greater_out), .MemWrite(IDEX_MemWrite_out),
```

```
.MemRead(IDEX_MemRead_out), .PCplusimm(adder_send_exmem),
.ALU_result(alu_result_out), .WriteData(muxout_b), .funct_in(IDEX_funct_in),
.rd(IDEX_rd_out), .RegWrite_store(EXMEM_RegWrite_out),
.MemtoReg_store(EXMEM_MemtoReg_out), .Branch_store(EXMEM_Branch_out),
.Zero_store(EXMEM_zero_out), .Is_Greater_store(EXMEM_Is_Greater_out),
.MemWrite_store(EXMEM_MemWrite_out), .MemRead_store(EXMEM_MemRead_out),
.PCplusimm_store(EXMEM_PC_plus_imm), .ALU_result_store(EXMEM_alu_result_out),
.WriteData_store(EXMEM_ReadData2_out), .funct_in_store(EXMEM_funct_in),
.rd_store(EXMEM_rd_out));
Branch_Control branch_cont(.Branch(EXMEM_Branch_out), .Flush(Flush_signal),
.Zero(EXMEM_zero_out), .Greater_Check(EXMEM_Is_Greater_out), .funct(EXMEM_funct_in),
.switch_branch(pc_mux_sel_wire));
Data_Memory
dm(EXMEM_alu_result_out,EXMEM_ReadData2_out,clk,EXMEM_MemWrite_out,EXMEM_M
emRead_out,DM_Read_Data_out,element1,element2, element3,element4);
MEM_WB mem_wb_reg(.clk(clk), .RegWrite(EXMEM_RegWrite_out),
.MemtoReg(EXMEM_MemtoReg_out), .ReadData(DM_Read_Data_out),
.ALU_result(EXMEM_alu_result_out), .rd(EXMEM_rd_out),
.RegWrite_store(MEMWB_RegWrite_out), .MemtoReg_store(MEMWB_MemtoReg_out),
.ReadData_store(MEMWB_DM_Read_Data_out), .ALU_result_store(MEMWB_alu_result_out),
.rd_store(MEMWB_rd_out));
mux_2by1 muxx(.a(MEMWB_DM_Read_Data_out),.b(MEMWB_alu_result_out),
.sel(MEMWB_MemtoReg_out), .data_out(mux_to_reg));
assign MemtoReg_IDEXin = controlmux_sel_bit ? MemtoReg_out : 0;
assign RegWrite_IDEXin = controlmux_sel_bit ? RegWrite_out : 0;
assign Branch_IDEXin = controlmux_sel_bit ? Branch_out : 0;
assign MemWrite_IDEXin = controlmux_sel_bit ? MemWrite_out : 0;
assign MemRead_IDEXin = controlmux_sel_bit ? MemRead_out : 0;
assign ALUSrc_IDEXin = controlmux_sel_bit ? ALUSrc_out : 0;
assign ALUop_IDEXin = controlmux_sel_bit ? ALUOp_out : 2'b00;
assign funct_in = {IFID_parse[30],IFID_parse[14:12]};
assign imm_to_adder = IDEX_imm_data_out<< 1;
endmodule
```

Instruction Memory:

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

reg [7:0] inst_mem[159:0];
initial
begin

        inst_mem[0] = 8'b10010011;
        inst_mem[1] = 8'b00000101;
        inst_mem[2] = 8'b01000000 ;
        inst_mem[3] = 8'b00000000;
        inst_mem[4] = 8'b00010011;
        inst_mem[5] = 8'b00000011 ;
        inst_mem[6] = 8'b00000000 ;
        inst_mem[7] = 8'b00000000 ;
        inst_mem[8] = 8'b10010011;
        inst_mem[9] = 8'b00000011;
```

```
inst_mem[10] = 8'b00000000 ;
inst_mem[11] = 8'b00000000 ;
inst_mem[12] = 8'b10010011;
inst_mem[13] = 8'b00000010 ;
inst_mem[14] = 8'b00000000 ;
inst_mem[15] = 8'b00000000 ;
inst_mem[16] = 8'b01100011;
inst_mem[17] = 8'b00001000 ;
inst_mem[18] = 8'b10110011 ;
inst_mem[19] = 8'b00001000 ;
inst_mem[20] = 8'b10010011;
inst_mem[21] = 8'b00001010 ;
inst_mem[22] = 8'b00000000 ;
inst_mem[23] = 8'b00000000 ;
inst_mem[24] = 8'b10110011;
inst_mem[25] = 8'b10001010  ;
inst_mem[26] = 8'b01101010  ;
inst_mem[27] = 8'b00000000 ;
inst_mem[28] = 8'b10110011;
inst_mem[29] = 8'b10001010;
inst_mem[30] = 8'b01101010;
inst_mem[31] = 8'b00000000;
inst_mem[32] = 8'b10110011;
inst_mem[33] = 8'b10001010;
inst_mem[34] = 8'b01101010;
inst_mem[35] = 8'b00000000;
inst_mem[36] = 8'b10110011;
inst_mem[37] = 8'b10001010;
inst_mem[38] = 8'b01101010;
inst_mem[39] = 8'b00000000;
inst_mem[40] = 8'b10110011;
inst_mem[41] = 8'b10001010;
inst_mem[42] = 8'b01101010;
inst_mem[43] = 8'b00000000;
inst_mem[44] = 8'b10110011;
inst_mem[45] = 8'b10001010;
inst_mem[46] = 8'b01101010;
inst_mem[47] = 8'b00000000;
inst_mem[48] = 8'b10110011;
inst_mem[49] = 8'b10001010;
inst_mem[50] = 8'b01101010;
inst_mem[51] = 8'b00000000;
inst_mem[52] = 8'b10110011;
inst_mem[53] = 8'b10001010;
inst_mem[54] = 8'b01101010;
inst_mem[55] = 8'b00000000;
inst_mem[56] = 8'b00010011;
inst_mem[57] = 8'b00000011  ;
inst_mem[58] = 8'b00010011  ;
inst_mem[59] = 8'b00000000  ;
inst_mem[60] = 8'b10010011;
inst_mem[61] = 8'b00000011 ;
inst_mem[62] = 8'b00000000 ;
inst_mem[63] = 8'b00000000 ;
```

```
inst_mem[64] = 8'b10010011;
inst_mem[65] = 8'b00000011;
inst_mem[66] = 8'b00000011;
inst_mem[67] = 8'b00000000;
inst_mem[68] = 8'b11100011;
inst_mem[69] = 8'b10000110 ;
inst_mem[70] = 8'b10110011 ;
inst_mem[71] = 8'b11111100 ;
inst_mem[72] = 8'b00110011;
inst_mem[73] = 8'b00001011 ;
inst_mem[74] = 8'b01111011 ;
inst_mem[75] = 8'b00000000 ;
inst_mem[76] = 8'b00110011;
inst_mem[77] = 8'b00001011;
inst_mem[78] = 8'b01111011;
inst_mem[79] = 8'b00000000;
inst_mem[80] = 8'b00110011;
inst_mem[81] = 8'b00001011;
inst_mem[82] = 8'b01111011;
inst_mem[83] = 8'b00000000;
inst_mem[84] = 8'b00110011;
inst_mem[85] = 8'b00001011;
inst_mem[86] = 8'b01111011;
inst_mem[87] = 8'b00000000;
inst_mem[88] = 8'b00110011;
inst_mem[89] = 8'b00001011;
inst_mem[90] = 8'b01111011;
inst_mem[91] = 8'b00000000;
inst_mem[92] = 8'b00110011;
inst_mem[93] = 8'b00001011;
inst_mem[94] = 8'b01111011;
inst_mem[95] = 8'b00000000;
inst_mem[96] = 8'b00110011;
inst_mem[97] = 8'b00001011;
inst_mem[98] = 8'b01111011;
inst_mem[99] = 8'b00000000;
inst_mem[100] = 8'b00110011;
inst_mem[101] = 8'b00001011;
inst_mem[102] = 8'b01111011;
inst_mem[103] = 8'b00000000;
inst_mem[104] = 8'b10010011;
inst_mem[105] = 8'b10000011 ;
inst_mem[106] = 8'b00010011 ;
inst_mem[107] = 8'b00000000 ;
inst_mem[108] = 8'b00110011;
inst_mem[109] = 8'b10000111 ;
inst_mem[110] = 8'b01011010 ;
inst_mem[111] = 8'b00000000 ;
inst_mem[112] = 8'b10110011;
inst_mem[113] = 8'b00000111  ;
inst_mem[114] = 8'b01011011  ;
inst_mem[115] = 8'b00000000 ;
inst_mem[116] = 8'b00010011;
inst_mem[117] = 8'b00001011  ;
inst_mem[118] = 8'b00000000  ;
```

```
        inst_mem[119] = 8'b00000000 ;
        inst_mem[120] = 8'b00000011;
        inst_mem[121] = 8'b00111000  ;
        inst_mem[122] = 8'b00000111  ;
        inst_mem[123] = 8'b00000000  ;
        inst_mem[124] = 8'b10000011;
        inst_mem[125] = 8'b10111000 ;
        inst_mem[126] = 8'b00000111 ;
        inst_mem[127] = 8'b00000000 ;
        inst_mem[128] = 8'b11100011;
        inst_mem[129] = 8'b00000010 ;
        inst_mem[130] = 8'b00011000 ;
        inst_mem[131] = 8'b11111101 ;
        inst_mem[132] = 8'b11100011;
        inst_mem[133] = 8'b01000000 ;
        inst_mem[134] = 8'b00011000 ;
        inst_mem[135] = 8'b11111101 ;
        inst_mem[136] = 8'b00110011;
        inst_mem[137] = 8'b00001001 ;
        inst_mem[138] = 8'b00000000 ;
        inst_mem[139] = 8'b00000001 ;
        inst_mem[140] = 8'b00110011;
        inst_mem[141] = 8'b00001000 ;
        inst_mem[142] = 8'b00010000 ;
        inst_mem[143] = 8'b00000001 ;
        inst_mem[144] = 8'b00100011;
        inst_mem[145] = 8'b00110000 ;
        inst_mem[146] = 8'b00000111 ;
        inst_mem[147] = 8'b00000001 ;
        inst_mem[148] = 8'b10110011;
        inst_mem[149] = 8'b00001000 ;
        inst_mem[150] = 8'b00100000 ;
        inst_mem[151] = 8'b00000001 ;
        inst_mem[152] = 8'b00100011;
        inst_mem[153] = 8'b10110000   ;
        inst_mem[154] = 8'b00010111   ;
        inst_mem[155] = 8'b00000001   ;
        inst_mem[156] = 8'b11100011 ;
        inst_mem[157] = 8'b00000100  ;
        inst_mem[158] = 8'b00000000  ;
        inst_mem[159] = 8'b11111010 ;
   end
        always @(Inst_Address)
        begin

        Instruction={inst_mem[Inst_Address+3],inst_mem[Inst_Address+2],inst_mem[Inst_Addre
ss+1],inst_mem[Inst_Address]};
        end
endmodule
```

Data Memory:

```
module Data_Memory
(
        input [63:0] Mem_Addr,
        input [63:0] Write_Data,
        input clk, MemWrite, MemRead,
        output reg [63:0] Read_Data
,
        output [63:0] element1,
   output [63:0] element2,
   output [63:0] element3,
   output [63:0] element4
);
  reg [7:0] DataMemory [63:0];
  integer i;
   initial
    begin
      for (i = 0; i < 64; i = i + 1)
      begin
        DataMemory[i] = 0;
      end
      DataMemory[0]  = 8'd6;
      DataMemory[8]  = 8'd8;
      DataMemory[16] = 8'd9;
      DataMemory[24] = 8'd7;



    end
  assign element1 ={DataMemory[7],DataMemory[6],  DataMemory[5] , DataMemory[4] ,
DataMemory[3] , DataMemory[2] , DataMemory[1] , DataMemory[0]};
assign element2 = {DataMemory[15], DataMemory[14], DataMemory[13] , DataMemory[12] ,
DataMemory[11], DataMemory[10] , DataMemory[9] , DataMemory[8]};
assign element3 =
{DataMemory[23],DataMemory[22],DataMemory[21],DataMemory[20],DataMemory[19],DataMe
mory[18],DataMemory[17],DataMemory[16]} ;
 assign element4= {DataMemory[31] ,DataMemory[30], DataMemory [29], DataMemory[28],
DataMemory[27] , DataMemory[26] ,DataMemory[25] ,DataMemory[24]};



  always @( posedge clk)
    begin
    if (MemWrite == 1'b1)
    begin
      DataMemory[Mem_Addr] = Write_Data[7:0];
      DataMemory[Mem_Addr + 1] = Write_Data[15:8];
      DataMemory[Mem_Addr + 2] = Write_Data[23:16];
      DataMemory[Mem_Addr + 3] = Write_Data[31:24];
      DataMemory[Mem_Addr + 4] = Write_Data[39:32];
      DataMemory[Mem_Addr + 5] = Write_Data[47:40];
      DataMemory[Mem_Addr + 6] = Write_Data[55:48];
      DataMemory[Mem_Addr + 7] = Write_Data[63:56];
```

```
    end
    end
  always @(*)
    begin
      if (MemRead == 1'b1)
        begin
      Read_Data = {DataMemory[Mem_Addr + 7], DataMemory[Mem_Addr +
6],DataMemory[Mem_Addr + 5], DataMemory[Mem_Addr + 4], DataMemory[Mem_Addr + 3],
DataMemory[Mem_Addr + 2],DataMemory[Mem_Addr + 1], DataMemory[Mem_Addr]};
        end

    end
endmodule
```

Hazard Detection Unit:

```
module Hazard_Detection
(
    input [4:0] IDEX_rd, IFID_rs1, IFID_rs2,
    input IDEX_MemRead,
    output reg IDEX_mux_out,
    output reg IFID_Write, PCWrite
);

always@(*) begin

    if (IDEX_MemRead && (IDEX_rd == IFID_rs1 || IDEX_rd == IFID_rs2))
    begin
        IDEX_mux_out = 0;
        IFID_Write = 0;
        PCWrite = 0;
    end
    else begin
        IDEX_mux_out = 1;
        IFID_Write = 1;
        PCWrite = 1;
    end

end

endmodule
```

Branch Control:

```
module Branch_Control
(
    input Branch, Zero, Greater_Check,
    input [3:0] funct,
    output reg switch_branch, Flush
);

always @(*) begin
    if (Branch) begin
        case ({funct[2:0]})
            3'b000: switch_branch = Zero ? 1 : 0;
            3'b001: switch_branch = Zero ? 0 : 1;
```

```
            3'b101: switch_branch = Greater_Check ? 1 : 0;
            3'b100: switch_branch = Greater_Check ? 0 : 1;
            default: switch_branch = 0;
        endcase
    end else begin
        switch_branch = 0;
    end
end

always @(switch_branch) begin
    if (switch_branch) begin
        Flush <= 1;
    end else begin
        Flush <= 0;
    end
end

endmodule
```

IF/ID – ID/EX – EX/MEM

```
module IF_ID (
    input clk, IFID_Write, Flush,
    input [63:0] PC_addr,
    input [31:0] Instruc,
    output reg [63:0] PC_store,
    output reg [31:0] Instr_store
);

    always @(posedge clk) begin
        if (Flush) begin
            PC_store <= 0;
            Instr_store <= 0;
        end
        else if (!IFID_Write) begin
            PC_store <= PC_store;
            Instr_store <= Instr_store;
        end
        else begin
            PC_store <= PC_addr;
            Instr_store <= Instruc;
        end
    end
endmodule


module ID_EX(
input clk, Flush,
input [63:0] PC_addr,
input [63:0] read_data1, read_data2,
input [63:0] imm_val,
input [3:0] funct_in,
input [4:0] rd_in, rs1_in, rs2_in,
input MemtoReg, RegWrite,
```

```
input Branch, MemWrite, MemRead,
input ALUSrc,
input [1:0] ALU_op,
output reg [63:0] PC_addr_store,
output reg [63:0] read_data1_store, read_data2_store,
output reg [63:0] imm_val_store,
output reg [3:0] funct_in_store,
output reg [4:0] rd_in_store, rs1_in_store, rs2_in_store,
output reg MemtoReg_store, RegWrite_store,
output reg Branch_store, MemWrite_store, MemRead_store,
output reg ALUSrc_store,
output reg [1:0] ALU_op_store
);

always @(posedge clk) begin

if (Flush) begin
PC_addr_store <= 0;
read_data1_store <= 0;
read_data2_store <= 0;
imm_val_store <= 0;
funct_in_store <= 0;
rd_in_store <= 0;
rs1_in_store <= 0;
rs2_in_store <= 0;
RegWrite_store <= 0;
MemtoReg_store <= 0;
Branch_store <= 0;
MemWrite_store <= 0;
MemRead_store <= 0;
ALUSrc_store <= 0;
ALU_op_store <= 0;
end

else begin

PC_addr_store <= PC_addr;
read_data1_store <= read_data1;
read_data2_store <= read_data2;
imm_val_store <= imm_val;
funct_in_store <= funct_in;
rd_in_store <= rd_in;
rs1_in_store <= rs1_in;
rs2_in_store <= rs2_in;
RegWrite_store <= RegWrite;
MemtoReg_store <= MemtoReg;
Branch_store <= Branch;
MemWrite_store <= MemWrite;
MemRead_store <= MemRead;
ALUSrc_store <= ALUSrc;
ALU_op_store <= ALU_op;
end
end

endmodule
```

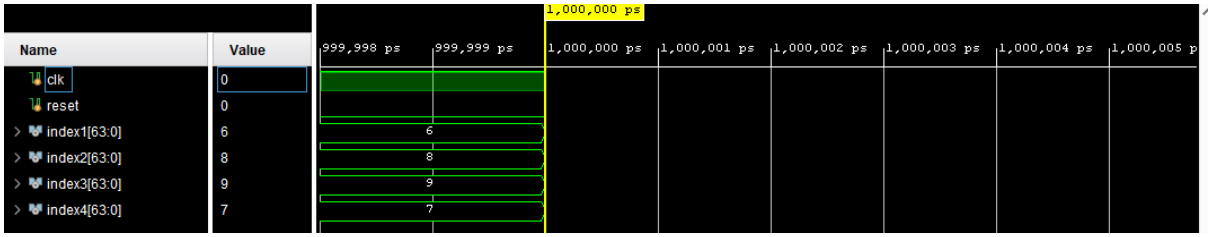
```
module EX_MEM
(
    input clk, Flush,
    input RegWrite, MemtoReg,
    input Branch, Zero, MemWrite, MemRead, Greater_Check,
    input [63:0] PCplusimm, ALU_result, WriteData,
    input [3:0] funct_in,
    input [4:0] rd,

    output reg RegWrite_store, MemtoReg_store,
    output reg Branch_store, Zero_store, MemWrite_store,
            MemRead_store, Is_Greater_store,
    output reg [63:0] PCplusimm_store, ALU_result_store,
          WriteData_store,
    output reg [3:0] funct_in_store,
    output reg [4:0] rd_store
);


always @(posedge clk) begin
    if (Flush) begin
    RegWrite_store = 0;
    MemtoReg_store = 0;
    Branch_store = 0;
    Zero_store = 0;
    Is_Greater_store = 0;
    MemWrite_store = 0;
    MemRead_store = 0;
    PCplusimm_store = 0;
    ALU_result_store = 0;
    WriteData_store = 0;
    funct_in_store = 0;
    rd_store = 0;
    end else begin
    RegWrite_store = RegWrite;
    MemtoReg_store = MemtoReg;
    Branch_store = Branch;
    Zero_store = Zero;
    Is_Greater_store = Greater_Check;
    MemWrite_store = MemWrite;
    MemRead_store = MemRead;
    PCplusimm_store = PCplusimm;
    ALU_result_store = ALU_result;
    WriteData_store = WriteData;
    funct_in_store = funct_in;
    rd_store = rd;
    end
end


endmodule
```

Task Bench:

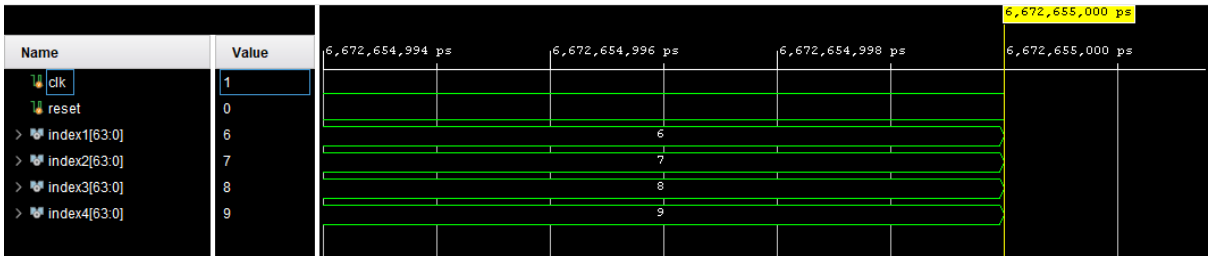
```
module tb_task3
();

reg clk, reset;
wire [63:0] index1,index2,index3,index4;
Pipelined_Processor_Task_3 tsk3
(
    .clk(clk),
    .reset(reset),
    .element1(index1),
    .element2(index2),
    .element3(index3),
    .element4(index4)

);

initial

 begin

  clk = 1'b0;

  reset = 1'b1;

  #10 reset = 1'b0;
 end


always

 #5 clk = ~clk;

endmodule // tb_RISC_V
```

SIMULATION results (PIPELINED PROCESSOR – Task 3)

Before (unsorted)



After (sorted)



Part 4: Compare the performance of running the array sorting program on Single Cycle Processor with RISC-V Processor in terms of execution time.

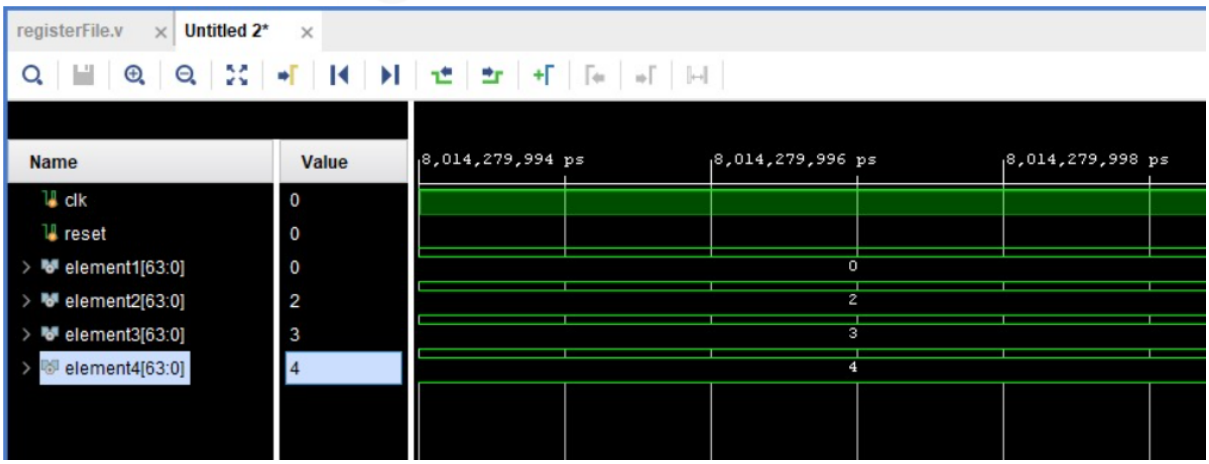
When running the code on single cycle, the time taken from CPU was 00:00:21. When running the code on pipeline without hazard, the time taken from CPU is 00:00:09. As we can see, the time taken by single cycle is more than the time taken with hazard (21>09).

Generally, a RISC-V processor will probably beat a single-cycle processor for executing an array sorting program. This is due to the fact that single-cycle processors can only execute one instruction per clock cycle, whereas RISC-V processors typically have a pipeline architecture that allows them to execute instructions in parallel.

A RISC-V processor's performance advantage may be even greater regarding array sorting algorithms like the bubble sort that we tested. This is due to the efficient execution of these algorithms on a pipelined processor, which involves numerous comparisons and conditional branches.

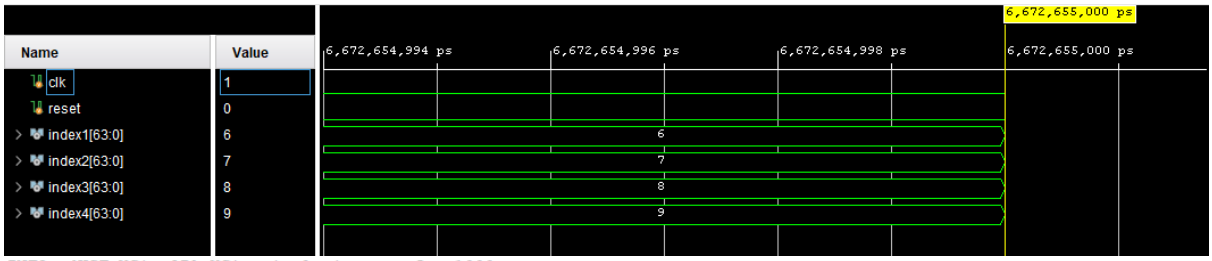
Hence when running array sorting programs, a RISC-V processor is likely to perform better than a single-cycle processor. Our results (shown on the next page) are a proof of this statement.

Results:
Single cycle:




```
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:21 ; elapsed = 00:00:27 . Memory (MB): peak = 1024.426 ; gain = 0.000
```

Pipelined:




```
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:09 ; elapsed = 00:00:23 . Memory (MB): peak = 1521.395 ; gain = 0.000
```

We have shown both result times above.

- When running the code on single cycle, the time taken from CPU was 00:00:21.
- When running the code on pipeline without hazard, the time taken from CPU is 00:00:09.

As we can see, the time taken by single cycle is more than the time taken with hazard (21>09). This is because certain implementations in pipeline such as forwarding units make it easier for the code to be executed without having to wait for all instructions to finish to load the next one. This saves time and hence pipelining had shorter CPU time than single cycle for bubble sort.

Challenges:
Firstly, we faced challenge in adding branching part to the processor, it wasn't done before and hence was a tricky thing to add. Secondly, setting our instruction and data memory according to the bubble sorting algorithm was also a little tough. Moreover, task 3 was also challenging as implementing flushing and branching control for it was very tricky.

Task Division:

Rabia: task 1, 3
Iqra: task 1, 3
Youshay: task 1, 3
Sidra: task 2

Conclusion:
Our project was a success, we were able to able to implement single cycle, pipelined with forwarding and hazard detection. We faced some difficulty in the last task but were able to overcome it with our Computer Architecture theory knowledge and extensive research.

References:

- CA Course TextBook & PPTs provided by our Instructors

Appendices:

- https://github.com/MuhammadYoushay/ComputerArchitectureProject.git