Computer Architecture Assignment 3

Student1 = Yogya Chawla 2020AAPS1776H

STUDENT 2 = GAUTHAM GUTTA 2020AAPS2204H

1) We have used stalling and forwarding unit to overcome the difficulties in pipelining.

2) Truth Table for instruction

Opcode	operation	RegDst	RegWrite	AluSrc	MemWrite	aluop	MemtoReg	MemRead	jump	Branch
000000	r-type	1	1	0	0	10	0	0	0	0
100011	i-type (load)	0	1	1	0	00	1	1	0	0
101011	i-type (store)	0	0	1	1	00	0	0	0	0
000100	i-type (beq)	0	0	0	0	01	0	0	0	1
000101	i-type (bne)	0	0	0	0	11	0	0	0	1
001000	i-type (addi)	0	1	1	0	00	0	0	0	0
001001	i-type (subi)	0	1	1	0	01	0	0	0	0
000010	j-type (jump)	0	0	0	0	00	0	0	1	0

ALU operations:

alucontrol	operation
0000	and
0001	or
0010	add
0110	sub
1000	sll
1001	set arithmetic/logical for sr
1010	decide barrel shifter direction
0100	zero for bne
0101	zero for beq
0111	slt

3) The program that you load in the instruction memory. (4 instructions minimum for each type)

Instruction Formats:

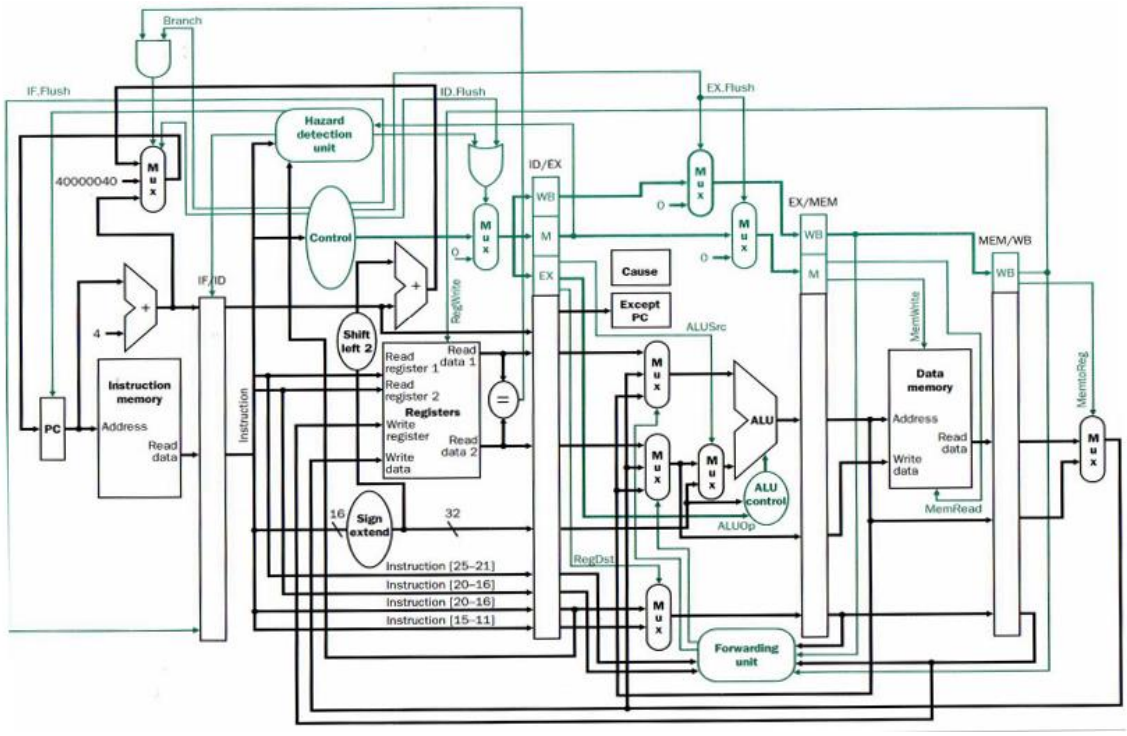
1) R-type instructions:
   Format:

   Rd<= Rs op Rt

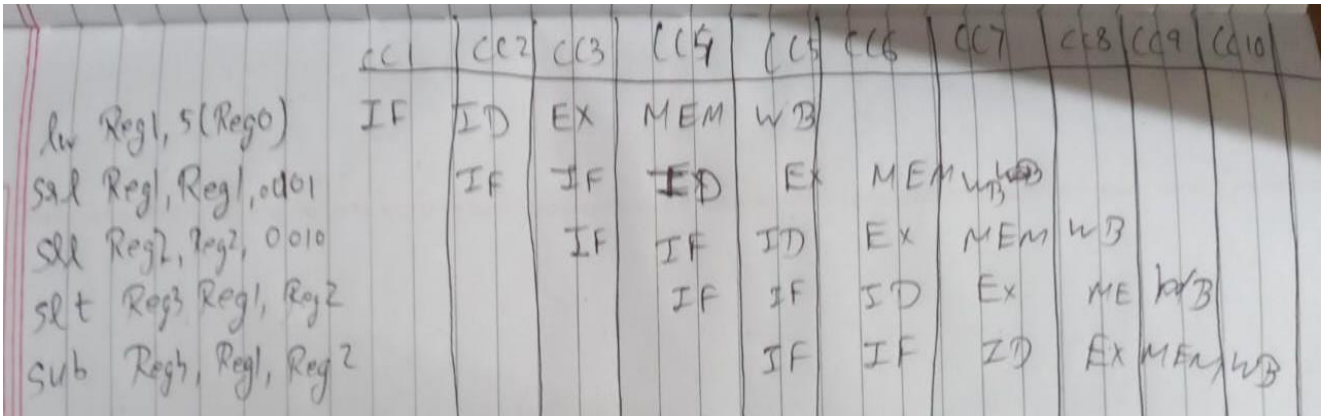
Opcode (6)	Rs (5)	Rt (5)	Rd (5)	Shamt (5)	Funct (6)

Opcode	**Funct**	**Shamt**	**Operation**
000000	100000	XXXXXX	Add
000000	100010	XXXXXX	Sub
000000	100100	XXXXXX	And
000000	100101	XXXXXX	Or
000000	101010	XXXXXX	Set Less Than
000000	010100	shamt	Left Shift
000000	010101	shamt	Right Shift
000000	010111	shamt	Barrel Shift

4) Block Diagram of MIPS Pipeline processor



5) Few instructions being stalled and their pipeline diagram shown below. Here for stalling actions we have repeated the instructions.



6) Output Waveform

7) Instructions loaded

100011 00000 00010 0000000000000001 //lw $Reg2,005(Reg0)

000100 00001 00010 0000000000001110 //- Branch to <end> if $Reg1=$Reg2

000000 00000 00001 00001 00001 010101 //- srl $Reg1, $Reg1, 00001

000000 00001 00010 00011 00000 101010 // slt $Reg3, $Reg1, $Reg2

000000 00010 00001 00101 00000 100010 sub $Reg5, $Reg1, $Reg2

000000 00010 00001 00110 00000 100101 or $Reg6, $Reg1, $Reg2

101011 00000 00101 0000000000000000 sw $Reg5, 000($Reg0)

000010 00000000000000000000001100 jump to end

000000 00010 00001 00101 00000 100000 add $Reg5, $Reg1, $Reg2

000000 00010 00001 00110 00000 100100 - and $Reg6, $Reg1, $Reg2

000010 00000000000000000000000111 jump to end//

001000 00010 01000 0000000000000101 - addi $Reg8, $Reg2, 101

8) Verilog files

Top.v

```
module top(
   input clock,
```

```
    input rst,
    output [31:0] aluout,
    output [31:0] Instruction,
    output [31:0] ALU_A,
    output [31:0] ALU_B,

    output memread,
    output memtoreg,
    output [1:0] aluop,
    output [2:0] alucontrol,
    output memwrite,
    output alusrc,
    output regdst,
    output regwrite,
    output [4:0] writereg,
    output [31:0] Read_reg_data_2,
    output [31:0] Readdata

    );
wire [31:0] instruction;
wire [31:0] PC_plus_4;
wire [31:0] PC;
wire Jump;
wire jr;
wire reg1;
wire jal;
wire Branch;
wire bne;
wire MemRead;
```

```
wire MemtoReg;
wire [1:0] ALUOp;
wire MemWrite;
wire ALSrc;
wire RegDst;
wire [31:0] write_data;
wire RegWrite;
wire [4:0] write_reg;
wire [31:0]read_reg_data_1;
wire [31:0]read_reg_data_2;
wire [31:0] extended;
wire zero;
wire [31:0] ALU_out;
wire [2:0] ALU_control;
wire [31:0] readdata;
wire signed [31:0] shifted;
wire [31:0] adder2_result;
wire [31:0] mux4_out;
wire [27:0] Out;
wire [31:0] jraddress;
wire [4:0] readreg1;
wire [31:0] jaladdress;
wire [4:0] writereg1;
wire [31:0] writedata1;

wire  ex_mem_Branch, ex_mem_MemRead, ex_mem_MemWrite, ex_mem_RegWrite,
ex_mem_MemtoReg, ex_mem_zero;
wire [31:0] ex_mem_read_reg_data_2, ex_mem_ALU_out, ex_mem_adder2_result;
wire  [4:0] ex_mem_write_reg;
```

```
wire [2:0] stall;

instruction_fetch if1(
.clock(clock),
.rst(rst),
.Jump(Jump),
.jr(jr),
.jal(jal),
.Branch(Branch),
.bne(bne),
.zero(zero),
.shifted(shifted),
.Jump_address(Out),
.jraddress(jraddress),
.jaladdress(jaladdress),
.instruction(instruction),
.PC(PC),
.stall(stall)
);

wire [31:0] read_address;

/*instruction_memory im1(
   .clock(clock),
   .read_address(PC),
   .rst(rst),
   .instruction(instruction)
);*/
```

```
wire [31:0] in;

mux m0(
   .A(PC_plus_4),
   .B(ex_mem_adder2_result),
   .sel(ex_mem_Branch & ex_mem_zero),
   .Out(in)
);

PC pc (
   .clock(clock),
   .rst(rst),
   .in(in),
   .PC(PC)
);


wire [31:0] if_id_instruction;
wire [31:0] if_id_PC_plus_4;


IF_ID if_id_uut(
   .clock(clock),
            .PC_plus_4(PC_plus_4),
            .instruction(instruction),
            .if_id_PC_plus_4(if_id_PC_plus_4),
            .if_id_instruction(if_id_instruction),
            .stall(stall[2])
```

```
);

wire  id_ex_Branch, id_ex_MemRead, id_ex_MemWrite, id_ex_RegWrite, id_ex_MemtoReg,
id_ex_RegDst, id_ex_ALUSrc;

wire  [1:0] id_ex_ALUOp;

wire [4:0] id_ex_readreg1;

wire  [31:0] id_ex_PC_plus_4, id_ex_read_reg_data_1, id_ex_read_reg_data_2, id_ex_extended,
id_ex_instruction;


ID_EX id_ex_uut (
   .clock(clock),
   .ALUSrc(ALUSrc),
   .RegDst(RegDst),
   .MemtoReg(MemtoReg),
   .Branch(Branch),
   .MemRead(MemRead),
   .MemWrite(MemWrite),
   .RegWrite(RegWrite),
   .ALUOp(ALUOp),
   .if_id_PC_plus_4(if_id_PC_plus_4),
   .read_reg_data_1(read_reg_data_1),
   .readreg1(readreg1),
   .read_reg_data_2(read_reg_data_2),
   .extended(extended),
   .if_id_instruction(if_id_instruction),
   .id_ex_ALUSrc(id_ex_ALUSrc),
   .id_ex_RegDst(id_ex_RegDst),
   .id_ex_MemtoReg(id_ex_MemtoReg),
   .id_ex_Branch(id_ex_Branch),
```

```
    .id_ex_MemRead(id_ex_MemRead),
    .id_ex_MemWrite(id_ex_MemWrite),
    .id_ex_RegWrite(id_ex_RegWrite),
    .id_ex_ALUOp(id_ex_ALUOp),
    .id_ex_PC_plus_4(id_ex_PC_plus_4),
    .id_ex_read_reg_data_1(id_ex_read_reg_data_1),
    .id_ex_readreg1(id_ex_readreg1),
    .id_ex_read_reg_data_2(id_ex_read_reg_data_2),
    .id_ex_extended(id_ex_extended),
    .id_ex_instruction(id_ex_instruction),
    .stall(stall[2])

);



EX_MEM ex_mem_uut (
    .clock(clock),
    .id_ex_Branch(id_ex_Branch),
    .id_ex_MemRead(id_ex_MemRead),
    .id_ex_MemWrite(id_ex_MemWrite),
    .id_ex_RegWrite(id_ex_RegWrite),
    .id_ex_MemtoReg(id_ex_MemtoReg),
    .zero(zero),
    .ALU_out(ALU_out),
    .id_ex_read_reg_data_2(id_ex_read_reg_data_2),
    .adder2_result(adder2_result),
    .write_reg(write_reg),
    .ex_mem_Branch(ex_mem_Branch),
    .ex_mem_MemRead(ex_mem_MemRead),
```

```
    .ex_mem_MemWrite(ex_mem_MemWrite),
    .ex_mem_RegWrite(ex_mem_RegWrite),
    .ex_mem_MemtoReg(ex_mem_MemtoReg),
    .ex_mem_zero(ex_mem_zero),
    .ex_mem_ALU_out(ex_mem_ALU_out),
    .ex_mem_read_reg_data_2(ex_mem_read_reg_data_2),
    .ex_mem_adder2_result(ex_mem_adder2_result),
    .ex_mem_write_reg(ex_mem_write_reg),
    .stall(stall[2])
);

wire mem_wb_MemtoReg, mem_wb_RegWrite;
wire [4:0] mem_wb_write_reg;
wire [31:0] mem_wb_readdata, mem_wb_ALU_out, mem_wb_write_data;

MEM_WB mem_wb_uut (
    .clock(clock),
    .ex_mem_RegWrite(ex_mem_RegWrite),
    .ex_mem_MemtoReg(ex_mem_MemtoReg),
    .ex_mem_ALU_out(ex_mem_ALU_out),
    .readdata(readdata),
    .ex_mem_write_reg(ex_mem_write_reg),
    .mem_wb_RegWrite(mem_wb_RegWrite),
    .mem_wb_MemtoReg(mem_wb_MemtoReg),
    .mem_wb_write_reg(mem_wb_write_reg),
    .mem_wb_ALU_out(mem_wb_ALU_out),
    .mem_wb_readdata(mem_wb_readdata),
    .stall(stall[2])
);
```

```
control c1(
   .opcode(if_id_instruction[31:26]),
   .RegDst(RegDst),
   .Jump(Jump),
   .Branch(Branch),
   .MemRead(MemRead),
   .MemtoReg(MemtoReg),
   .ALUOp(ALUOp),
   .MemWrite(MemWrite),
   .ALUSrc(ALUSrc),
   .RegWrite(RegWrite),
   .jr(jr),
   .reg1(reg1),
   .jal(jal),
   .bne(bne),
   .stall(stall[0]),
   .clock(clock)
);
//mux m_c(
//    .A(
//    .B(
//);

adder a1(
.A(PC),
.B(4),
.sum(PC_plus_4)
);
```

```
mux m1(
   .A(id_ex_instruction[20:16]),
   .B(id_ex_instruction[15:11]),
   .sel(id_ex_RegDst),
   .Out(write_reg)
);

mux m6(
   .A(if_id_instruction[25:21]),
   .B(if_id_instruction[4:0]),
   .sel(reg1),
   .Out(readreg1)
);

mux m7(
   .A(mem_wb_write_reg),
   .B(5'b11111),
   .sel(jal),
   .Out(writereg1)
);

mux m8(
   .A(mem_wb_write_data),
   .B(jaladdress),
   .sel(jal),
   .Out(writedata1)
);
```

```
mux m3(
   .A(mem_wb_ALU_out),
   .B(mem_wb_readdata),
   .sel(mem_wb_MemtoReg),
   .Out(mem_wb_write_data)
);

data_memory d1(
   .address(ex_mem_ALU_out),
   .write_data(ex_mem_read_reg_data_2),
   .readdata(readdata),
   .mem_read(ex_mem_MemRead),
   .mem_write(ex_mem_MemWrite),
   .clock(clock)
);

register_file r1(
   .readreg1(readreg1),
   .readreg2(if_id_instruction[20:16]),
   .write_reg(writereg1),
   .write_data(writedata1),
   .read_reg_data_1(read_reg_data_1),
   .read_reg_data_2(read_reg_data_2),
   .RegWrite(mem_wb_RegWrite),
   .clock(clock)
);

assign jraddress = read_reg_data_1;
```

```
sign_extender ex1(
   .A(if_id_instruction[15:0]),
   .OUT(extended)
);

mux m2(
   .A(id_ex_read_reg_data_2),
   .B(id_ex_extended),
   .sel(id_ex_ALUSrc),
   .Out(ALU_B)
);

ALU_control ALUc(
   .ALUOp(id_ex_ALUOp),
   .funct(id_ex_instruction[5:0]),
   .ALU_control(ALU_control)
);

wire [1:0] forwardA, forwardB;

forwarding_unit fu (
   .ex_mem_RegWrite(ex_mem_RegWrite),
   .mem_wb_RegWrite(mem_wb_RegWrite),
   .clock(clock),
   .id_ex_readreg1(id_ex_readreg1),
   .id_ex_readreg2(if_id_instruction[20:16]),
   .ex_mem_write_reg(ex_mem_write_reg),
   .mem_wb_write_reg(writereg1),
   .forwardA(forwardA),
```

```
    .forwardB(forwardB)
);
wire [31:0] alu_a, alu_b;

mux2 m01 (
    .A(id_ex_read_reg_data_1),
    .B(writedata1),
    .C(ex_mem_ALU_out),
    .sel(forwardA),
    .Out(alu_a)
);
mux2 m02 (
    .A(ALU_B),
    .B(writedata1),
    .C(ex_mem_ALU_out),
    .sel(forwardB),
    .Out(alu_b)
);


ALU alu1(
    .ALU_src_1(alu_a),
    .ALU_src_2(alu_b),
    .ALU_control(ALU_control),
    .shamt(id_ex_instruction[10:6]),
    .ALU_out(ALU_out),
    .zero(zero),
    .clock(clock)
);
```

```
shifter s1(
   .A(id_ex_extended),
   .Out(shifted)
);

adder a2(
   .A(id_ex_PC_plus_4),
   .B(shifted),
   .sum(adder2_result)
);

/*mux m4(
   .A(PC_plus_4),
   .B(adder2_result),
   .sel(Branch & zero),
   .Out(mux4_out)
); */



shifter2 s2(
   .A(instruction[25:0]),
   .Out(Out)
);

/*mux m5(
   .A(mux4_out),
   .B({PC_plus_4, Out}),
```

```
   .sel(Jump),
   .Out(PC)
);*/

stalling_unit su (
   .clock(clock),
   .id_ex_MemRead(id_ex_MemRead),
   .if_id_Rs(readreg1),
   .if_id_Rt(if_id_instruction[20:16]),
   .id_ex_Rt(id_ex_instruction[20:16]),
   .stall(stall)
);

assign aluout = ALU_out;
assign Instruction = instruction;
assign ALU_A = read_reg_data_1;


assign memread = MemRead;
assign memtoreg = MemtoReg;
assign aluop = ALUOp;
assign memwrite = MemWrite;
assign alusrc =ALUSrc;
assign regdst = RegDst;
assign writedata = write_data;
assign regwrite = RegWrite;
assign writereg = write_reg;
assign Read_reg_data_2 = read_reg_data_2;
assign Readdata = readdata;
```

```
assign alucontrol = ALU_control;

endmodule
```

instruction_fetch.v

```
module instruction_fetch(
   input clock,
   input rst,
   input Jump,
   input jr,
   input Branch,
   input bne,
   input zero,
   input jal,
   input signed [31:0] shifted,
   input [27:0] Jump_address,
   input [31:0] jraddress,
   output [31:0] jaladdress,
   output [31:0] instruction,
   output reg [31:0] PC,
   input stall
   );

instruction_memory I1 (
.read_address(PC),
.rst(rst),
.instruction(instruction)
);
```

```
wire [31:0] PC_plus_four;
assign PC_plus_four = PC + 4;
assign jaladdress = PC + 8;



always@(negedge clock)
begin
if(rst == 0)
        PC <= 0;

else if((Branch && zero) & !stall)
   PC <= PC + 4 + shifted;

else if((bne && !zero) & !stall)
   PC <= PC + 4 + shifted;

else if((Jump) & !stall)
   PC <= {PC_plus_four[31:28],Jump_address};

else if((jr) & !stall)
   PC <= jraddress;

else if (!stall)
        PC <= PC + 4;
end




endmodule
```

instruction_memory.v

```
module instruction_memory(
   input [31:0] read_address,
   input rst,
   output [31:0] instruction
   );

reg [7:0] Mem [115:0];
assign instruction = {Mem[read_address], Mem[read_address+1], Mem[read_address+2],
Mem[read_address+3]};

initial
begin
                                $readmemh("instruction_mem.mem", Mem);
end

endmodule
```

mux.v

```
module mux(
   input [31:0] A,
   input [31:0] B,
   input sel,
   output  [31:0] Out
);

assign Out = sel? B:A;
endmodule
```

PC.v

```
module PC(
   input clock,
   input rst,
   input [31:0] in,
   output [31:0] PC
   );

assign PC = rst ? in: 0;

endmodule
```

IF_ID.v

```
module IF_ID(
   input clock,
            input [31:0] PC_plus_4, instruction,
            output reg [31:0] if_id_PC_plus_4, if_id_instruction,
            input stall
            );



            always@ (negedge clock)
            begin
            if(!stall) begin
          if_id_PC_plus_4 <= PC_plus_4;
          if_id_instruction <= instruction;
```

```
   end
        end
endmodule
```

ID_EX.v

```
module ID_EX(
        input Branch, MemRead, MemWrite, RegWrite, MemtoReg, RegDst, ALUSrc, clock,
        input [1:0] ALUOp,
        input [31:0] if_id_PC_plus_4, read_reg_data_1, read_reg_data_2, extended, if_id_instruction,
        input [4:0] readreg1,

        output reg id_ex_Branch, id_ex_MemRead, id_ex_MemWrite, id_ex_RegWrite,
id_ex_MemtoReg, id_ex_RegDst, id_ex_ALUSrc,
        output reg [1:0] id_ex_ALUOp,
        output reg [4:0] id_ex_readreg1,
        output reg [31:0] id_ex_PC_plus_4, id_ex_read_reg_data_1, id_ex_read_reg_data_2,
id_ex_extended, id_ex_instruction,
        input stall

        );

        always@ (negedge clock) begin
      if(!stall) begin
         id_ex_Branch <= Branch;
         id_ex_MemRead <= MemRead;
         id_ex_MemWrite <= MemWrite;
         id_ex_RegWrite <= RegWrite;
```

```
  id_ex_MemtoReg <= MemtoReg;
  id_ex_RegDst <= RegDst;
  id_ex_ALUSrc <= ALUSrc;
  id_ex_ALUOp <= ALUOp;
  id_ex_PC_plus_4 <= if_id_PC_plus_4;
  id_ex_read_reg_data_1 <= read_reg_data_1;
  id_ex_readreg1 <= readreg1;
  id_ex_read_reg_data_2 <= read_reg_data_2;
  id_ex_extended <= extended;
  id_ex_instruction <= if_id_instruction;
  end
else
  begin
     id_ex_Branch <= 0;
     id_ex_MemRead <= 0;
     id_ex_MemWrite <= 0;
     id_ex_RegWrite <= 0;
     id_ex_MemtoReg <= 0;
     id_ex_RegDst <= 0;
     id_ex_ALUSrc <= 0;
     id_ex_ALUOp <= 0;
     id_ex_PC_plus_4 <= if_id_PC_plus_4;
     id_ex_read_reg_data_1 <= read_reg_data_1;
     id_ex_readreg1 <= readreg1;
     id_ex_read_reg_data_2 <= read_reg_data_2;
     id_ex_extended <= extended;
     id_ex_instruction <= if_id_instruction;
  end
```

```
        end

endmodule
```

EX_MEM.v

```
module EX_MEM(
        input id_ex_Branch, id_ex_MemRead, id_ex_MemWrite, id_ex_RegWrite, id_ex_MemtoReg,
clock, zero,
        input [31:0] id_ex_read_reg_data_2, ALU_out, adder2_result,
        input [4:0] write_reg,
        output reg ex_mem_Branch, ex_mem_MemRead, ex_mem_MemWrite, ex_mem_RegWrite,
ex_mem_MemtoReg, ex_mem_zero,
        output reg [31:0] ex_mem_read_reg_data_2, ex_mem_ALU_out, ex_mem_adder2_result,
        output reg [4:0] ex_mem_write_reg,
        input stall
        );



        always@ (negedge clock) begin

    ex_mem_Branch <= id_ex_Branch;
    ex_mem_MemRead <= id_ex_MemRead;
    ex_mem_MemWrite <= id_ex_MemWrite;
    ex_mem_RegWrite <= id_ex_RegWrite;
    ex_mem_MemtoReg <= id_ex_MemtoReg;
    ex_mem_zero <= zero;
            ex_mem_write_reg <= write_reg;
    ex_mem_read_reg_data_2 <= id_ex_read_reg_data_2;
```

```
        ex_mem_ALU_out <= ALU_out;
        ex_mem_adder2_result <= adder2_result;

          end

endmodule
```

MEM>WB.v

```
module MEM_WB(
          input ex_mem_RegWrite, ex_mem_MemtoReg, clock,
          input [31:0] ex_mem_ALU_out, readdata,
          input [4:0] ex_mem_write_reg,
          output reg mem_wb_MemtoReg, mem_wb_RegWrite,
          output reg [4:0] mem_wb_write_reg,
          output reg [31:0] mem_wb_readdata, mem_wb_ALU_out,
          input stall
          );


          always@ (negedge clock)
           begin

        mem_wb_RegWrite <= ex_mem_RegWrite;
        mem_wb_MemtoReg <= ex_mem_MemtoReg;
        mem_wb_readdata <= readdata;
        mem_wb_ALU_out <= ex_mem_ALU_out;
        mem_wb_write_reg <= ex_mem_write_reg;
           end
```

```
endmodule
```

adder.v

```
module adder(
input [31:0] A,
input [31:0] B,
output [31:0] sum
);

assign sum = A+B;



endmodule
```

control_unit.v

```
module control(
    input [5:0] opcode,
    output reg RegDst,
    output reg Jump,
    output reg Branch,
    output reg MemRead,
    output reg MemtoReg,
    output reg [1:0] ALUOp,
    output reg MemWrite,
    output reg ALUSrc,
    output reg RegWrite,
    output reg jr,
    output reg reg1,
    output reg jal,
```

```
    output reg bne,
    input stall,
    input clock
    );
always@(*)
begin
case(opcode)
        6'b000000 : begin      // R - type
                                        RegDst = 1;
                                        ALUSrc = 0;
                                        MemtoReg = 0;
                                        RegWrite = 1;
                                        MemRead = 0;
                                        MemWrite = 0;
                                        Branch = 0;
                                        ALUOp[1] = 1;
                                        ALUOp[0] = 0;
                                        Jump = 0;
                                        jr = 0;
                                        reg1 = 0;
                                        jal = 0;
                                        bne = 0;
                                        end
        6'b100011 : begin              //LW
                                        RegDst = 0;
                                        ALUSrc = 1;
                                        MemtoReg = 1;
                                        RegWrite = 1;
                                        MemRead = 1;
```

```
                    MemWrite = 0;
                    Branch = 0;
                    ALUOp[1] = 0;
                    ALUOp[0] = 0;
                    Jump = 0;
                    jr = 0;
                    reg1 = 0;
                    jal = 0;
                    bne = 0;
                    end
6'b101011 : begin   //SW
                    RegDst = 0;
                    ALUSrc = 1;
                    MemtoReg = 0;
                    RegWrite = 0;
                    MemRead = 0;
                    MemWrite = 1;
                    Branch = 0;
                    ALUOp[1] = 0;
                    ALUOp[0] = 0;
                    Jump = 0;
                    jr = 0;
                    reg1 = 0;
                    jal = 0;
                    bne = 0;
                    end
6'b000100 : begin   //beq
                    RegDst = 0;
                    ALUSrc = 0;
```

```
                                MemtoReg = 0;
                                RegWrite = 0;
                                MemRead = 0;
                                MemWrite = 0;
                                Branch = 1;
                                ALUOp[1] = 0;
                                ALUOp[0] = 1;
                                Jump = 0;
                                jr = 0;
                                reg1 = 0;
                                jal = 0;
                                bne = 0;
                                end
6'b100000 : begin               //bne
                                RegDst = 0;
                                ALUSrc = 0;
                                MemtoReg = 0;
                                RegWrite = 0;
                                MemRead = 0;
                                MemWrite = 0;
                                Branch = 0;
                                bne = 1;
                                ALUOp[1] = 0;
                                ALUOp[0] = 1;
                                Jump = 0;
                                jr = 0;
                                reg1 = 0;
                                jal = 0;
                                end
```

```
6'b000001 : begin                    //addi
                                     RegDst = 0;
                                     ALUSrc = 1;
                                     MemtoReg = 0;
                                     RegWrite = 1;
                                     MemRead = 0;
                                     MemWrite = 0;
                                     Branch = 0;
                                     ALUOp[1] = 0;
                                     ALUOp[0] = 0;
                                     Jump = 0;
                                     jr = 0;
                                     reg1 = 0;
                                     jal = 0;
                                     bne = 0;
                                     end
6'b000010 : begin                    //j
                                     RegDst = 0;
                                     ALUSrc = 0;
                                     MemtoReg = 0;
                                     RegWrite = 0;
                                     MemRead = 0;
                                     MemWrite = 0;
                                     Branch = 0;
                                     ALUOp[1] = 0;
                                     ALUOp[0] = 0;
                                     Jump = 1;
                                     jr = 0;
                                     reg1 = 0;
```

```
                                jal = 0;
                                bne = 0;
                                end
6'b000011 : begin               //jr
                                RegDst = 0;
                                ALUSrc = 0;
                                MemtoReg = 0;
                                RegWrite = 0;
                                MemRead = 0;
                                MemWrite = 0;
                                Branch = 0;
                                ALUOp[1] = 0;
                                ALUOp[0] = 0;
                                Jump = 0;
                                jr = 1;
                                reg1 = 1;
                                jal = 0;
                                bne = 0;
                                end
6'b000111 : begin               //jal
                                RegDst = 0;
                                ALUSrc = 0;
                                MemtoReg = 0;
                                RegWrite = 1;
                                MemRead = 0;
                                MemWrite = 0;
                                Branch = 0;
                                ALUOp[1] = 0;
                                ALUOp[0] = 0;
```

```
                    Jump = 1;
                    jr = 0;
                    reg1 = 0;
                    jal = 1;
                    bne = 0;
                    end
6'b001111 : begin   //jalr
                    RegDst = 0;
                    ALUSrc = 0;
                    MemtoReg = 0;
                    RegWrite = 1;
                    MemRead = 0;
                    MemWrite = 0;
                    Branch = 0;
                    ALUOp[1] = 0;
                    ALUOp[0] = 0;
                    Jump = 0;
                    jr = 1;
                    reg1 = 1;
                    jal = 1;
                    bne = 0;
                    end
default : begin
                    RegDst = 0;
                    ALUSrc = 0;
                    MemtoReg = 0;
                    RegWrite = 0;
                    MemRead = 0;
                    MemWrite = 0;
```

```
                                                    Branch = 0;
                                                    ALUOp[1] = 0;
                                                    ALUOp[0] = 0;
                                                    Jump = 0;
                                                    jr = 0;
                                                    reg1 = 0;
                                                    jal = 0;
                                                    bne = 0;
                                                    end
endcase

end

/*always@(*)
begin
if(stall) begin
        Branch = 0;
   MemRead = 0;
   MemWrite = 0;
   RegWrite = 0;
   MemtoReg = 0;
   RegDst = 0;
   ALUSrc = 0;
   ALUOp = 0;
end
end*/


endmodule
```

register_file.v

```
module register_file(
   input [4:0] readreg1,
   input [4:0] readreg2,
   input [4:0] write_reg,
   input [31:0] write_data,
   output [31:0] read_reg_data_1,
   output [31:0] read_reg_data_2,
   input RegWrite,
   input clock
   );

reg [31:0] regMem [31:0];
initial begin
        $readmemh("register_mem.mem", regMem);
        end


always@(posedge clock)
begin
        if(RegWrite)
                regMem[write_reg] = write_data;
end

assign read_reg_data_1 = regMem[readreg1];
assign read_reg_data_2 = regMem[readreg2];
```

```
endmodule
```

data_memory.v

```
module data_memory(
   input [31:0] address,
   input [31:0] write_data,
   output reg [31:0] readdata,
   input mem_read,
   input mem_write,
   input clock
   );

reg [7:0] dataMem [127:0];
initial begin
        $readmemh("data_mem.mem", dataMem);
        end

always@(posedge clock)
begin
        if(mem_write)
        begin
                {dataMem[address+3], dataMem[address+2], dataMem[address+1], dataMem[address]}
<= write_data;
        end

        if(mem_read)
        begin
```

```
                readdata <= {dataMem[address+3], dataMem[address+2], dataMem[address+1],
dataMem[address]};
        end

end

endmodule
```

sign_extender.v

```
module sign_extender(
   input [15:0] A,
   output reg [31:0] OUT
   );
always@(*)

begin
OUT[15:0] = A;
OUT[31:16] = {16{A[15]}};
end

endmodule
```

ALUControl.v

```
module ALU_control(
   input [1:0] ALUOp,
   input [5:0] funct,
   output reg [2:0] ALU_control
```

```
   );
always@(*)
begin
if(ALUOp == 2'b00)

        ALU_control = 3'b010;
else if(ALUOp == 2'b01)
        ALU_control = 3'b110;
else if(ALUOp[1]) begin
        case(funct[3:0])
        4'b0000 : ALU_control = 3'b010;  //add
        4'b0010 : ALU_control = 3'b110;  //sub
        4'b0100 : ALU_control = 3'b000;  //and
        4'b0101 : ALU_control = 3'b001;  //or
        4'b1010 : ALU_control = 3'b111;  //sll
        4'b1011 : ALU_control = 3'b011;  //srl

        endcase
end
end
endmodule
```

forwarding_unit.v

```
odule forwarding_unit(

        input ex_mem_RegWrite, mem_wb_RegWrite, clock,
        input [4:0] id_ex_readreg1, id_ex_readreg2, ex_mem_write_reg, mem_wb_write_reg,
        output reg [1:0] forwardA, forwardB
```

```
	);


	always@ (negedge clock) begin
		if(ex_mem_RegWrite & (ex_mem_write_reg != 0) & (ex_mem_write_reg == id_ex_readreg1))
			forwardA <= 2'b10;
		else if(mem_wb_RegWrite & (mem_wb_write_reg != 0) & (mem_wb_write_reg == id_ex_readreg1))
			forwardA <= 2'b01;
		else
			forwardA <= 2'b00;
	end


	always@ (negedge clock) begin
		if(ex_mem_RegWrite & (ex_mem_write_reg != 0) & (ex_mem_write_reg == id_ex_readreg2))
			forwardB <= 2'b10;
		else if(mem_wb_RegWrite & (mem_wb_write_reg != 0) & (mem_wb_write_reg == id_ex_readreg2))
			forwardB <= 2'b01;
		else
			forwardB <= 2'b00;
	end



endmodule
```

mux2.v

```
module mux2(
```

```
   input [31:0] A,
   input [31:0] B,
   input [31:0] C,
   input [1:0] sel,
   output  [31:0] Out
);

assign Out = sel[1]? C:(sel[0]? B:A);
endmodule
```

ALU.v

```
module ALU(
   input [31:0] ALU_src_1,
   input [31:0] ALU_src_2,
   input [2:0] ALU_control,
   input [4:0] shamt,
   output reg [31:0] ALU_out,
   output reg zero,
   input clock
   );

always @(*) begin
            case(ALU_control)
            3'b000 : ALU_out = ALU_src_1 & ALU_src_2;
            3'b001 : ALU_out = ALU_src_1 | ALU_src_2;
            3'b010 : ALU_out = ALU_src_1 + ALU_src_2;
            3'b110 : ALU_out = ALU_src_1 - ALU_src_2;
            3'b111 : ALU_out = ALU_src_1 << shamt;
```

```
        3'b011 : ALU_out = ALU_src_1 >> shamt;
        default: ALU_out = 32'b 0;
        endcase
end

always@(*)
begin
if(ALU_out == 0)
        zero = 1;
else
        zero = 0;
end

endmodule
```

shifter.v

```
module shifter(
   input [31:0] A,
   output reg signed [31:0] Out
   );

always@(*)
begin
Out = A<<2;
end
endmodule
```

shifter2.v

```
module shifter2(
   input [25:0] A,
   output reg [27:0] Out
   );

always@(*)
begin
Out = {2'b00,(A<<2)};
end
endmodule
```

stalling_unit.v

```
module stalling_unit(
        input clock, id_ex_MemRead,
        input [4:0] if_id_Rs, if_id_Rt, id_ex_Rt,
        output reg [2:0] stall
        );

        always@ (posedge clock) begin
                if(id_ex_MemRead & ((id_ex_Rt == if_id_Rs) | (id_ex_Rt == if_id_Rt)))
                        stall <= 1;
                else
                        stall <= 0;
        end

        /*reg check;
```

```
always@(*)
check <= id_ex_MemRead & ((id_ex_Rt == if_id_Rs) | (id_ex_Rt == if_id_Rt));

always@(*) begin
if(check)
                stall <= 1;
        else
                stall <= 0;
end*/

//assign stall = (id_ex_MemRead && ((id_ex_Rt == if_id_Rs) || (id_ex_Rt == if_id_Rt)))? 1:0;

Endmodule
```