**EECE 2323 Digital Logic Design Lab Report**

Lab #4 Adding Register File to ALU

| Student Name: Masha Drapkin | Section #: 1 |
| --- | --- |
| Instructor Name: Emad Aboelela | Date: 7/29/22 |
| Lab TAs: Keshav Bharadwaj Vaidyanathan, Srinidi Somasundaram Subbulakshmi, Yuchao Su | |

**Background & Purpose**

…

**Pre-Lab Response**

*Verilog HDL Code*

module mux\_temp(input[7:0] in1,

input[7:0] in2,

input select\_line,

output reg [7:0] out

);

always@(\*)

begin

if(select\_line)

out <= in2;

else

out <= in1;

end

endmodule

module eightbit\_alu(

input signed [7:0] a,

input signed [7:0] b,

input [2:0] s,

output signed [7:0] f,

output ovf,

output take\_branch

);

assign f = (s == 0)? a + b:

(s == 1)? ~b:

(s == 2)? a & b:

(s == 3)? a | b:

(s == 4)? a >>> 1:

(s == 5)? a << b:

0;

assign ovf = (s == 0)? (a[7] == b[7]) && (a[7] != f[7]):

0;

assign take\_branch = (s == 6)? a == b:

(s == 7)? a != b:

0;

endmodule

module reg\_file(

input rst,

input clk,

input wr\_en,

input [1:0] rd0\_addr,

input [1:0] rd1\_addr,

input [1:0] wr\_addr,

input [8:0] wr\_data,

output [7:0] r0\_data,

output [7:0] r1\_data

);

reg [8:0] RegFile [3:0];

assign r0\_data = RegFile[rd0\_addr];

assign r1\_data = RegFile[rd1\_addr];

integer i = 0; //for the loop

always @ (posedge clk, posedge rst)

if (rst) //clear the contents of all registers

for (i=0; i < 2\*\*2; i=i+1) RegFile[i] <= 0;

else if (wr\_en == 1'b1) RegFile[wr\_addr] <= wr\_data;

endmodule

module alu\_regfile(input ALUSrc1,

input ALUSrc2,

input [7:0] Instr\_i,

input rst,

input clk,

input wr\_en,

input [1:0] rd0\_addr,

input [1:0] rd1\_addr,

input [1:0] wr\_addr,

input [8:0] wr\_data,

input wr\_en,

input [2:0] ALUOp,

output [7:0] result,

output ovf,

output take\_branch,

output [7:0] input1,

output [7:0] input2

);

reg [7:0] r0\_data, r1\_data;

reg [7:0] zero\_reg = 8'b0;

mux\_temp

mux1(.in1(r0\_data[7:0]),.in2(zero\_reg),.select\_line(ALUSrc1),.out(input1));

mux\_temp mux2(.in1(r1\_data[7:0]),.in2(Instr\_i),.select\_line(ALUSrc2),.out(input2));

reg\_file reg1(.rst(rst),.clk(clk),.wr\_en(wr\_en),.rd0\_addr(rd0\_addr),.rd1\_addr(rd1\_addr),.wr\_addr(wr\_addr),.wr\_data(wr\_data),.r0\_data(input1),.r1\_data(input2));

eightbit\_alu alu1(.a(input1),.b(input2),.s(ALUOp),.f(result),.ovf(ovf),.take\_branch(take\_branch));

endmodule

*Test Vectors*

| **rst** | **wr\_en** | **r0\_addr** | **r1\_addr** | **wr\_addr** | **wr\_data** | **Instr\_i** | **ALUSrc1** | **ALUSrc2** | **ALUOp** | ***result*** | ***input1*** | ***input2*** | ***ovf*** | ***take\_branch*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | - | - | 0 | 0 | - | - | - | - | - | - | - | - | - |
| 0 | 1 | - | - | 00 | -8’d86  10101010 | - | - | - | - | - | - | - | - | - |
| 0 | 0 | 00 | - | - | - | - | 0 | - | 001  (inv) | - | -8’d86  10101010 | - | 0 | 0 |
| 0 | 1 | - | - | 01 | 8d’85  01010101 | - | - | - | - | - | - | - | - | - |
| 0 | 0 | 00 | 01 | - | - | - | 0 | 0 | 010  (and) | 8’d0 | -8’d86 | 8d’85 | 0 | 0 |
| 0 | 0 | 00 | 01 | - | - | - | 0 | 0 | 000  (+) | -8’d1 | -8’d86 | 8d’85 | 0 | 0 |
| 0 | 0 | - | - | 01 | -8’d86  10101010 | - | - | - | - | - | - | - | - | - |
| 0 | 0 | 00 | 01 | - | - | - | 0 | 0 | 011  (or) | -8’d1  11111111 | -8’d86 | 8d’85 | 0 | 0 |
| 0 | 1 | - | - | 10 | -8’d86  10101010 | - | - | - | - | - | - | - | - | - |
| 0 | 1 | - | - | 11 | -8d’100  10011100 | - | - | - | - | - | - | - | - | - |
| 0 | 0 | 10 | 11 | - | - | - | 0 | 0 | 000  (+) | 8’d70  01000110 | -8’d86  10101010 | -8d’100  10011100 | 1 | 0 |
| 0 | 0 | 10 | 11 | - | - | -8d’2 11111110 | 0 | 1 | 100  (sra) | -8d’43 11010101 | -8’d86  10101010 | -8d’2 11111110 | 0 | 0 |
| 0 | 0 | 10 | 11 | - | - | - | 1 | 0 | 110  (beq) | 8’d0 | 8’d0 | -8d’100  10011100 | - | 0 |
| 0 | 0 | 10 | 11 | - | - | -8d’2 11111110 | 1 | 1 | 110  (beq) | 8’d0 | -8d’2 11111110 | 0 | 0 | 0 |
| 0 | 0 | 10 | 11 | - | - | -8d’2 11111110 | 1 | 1 | 111  (bne) | 8’d0 | -8d’2 11111110 | 0 | 0 | 1 |

*Testbench Code*

`timescale 1ns / 1ps

module alu\_reg\_file\_tb;

reg ALUSrc1, ALUSrc2, wr\_en, clk, rst = 1;

reg [1:0] rd0\_addr, rd1\_addr, wr\_addr;

reg [2:0] ALUOp;

reg [7:0] Instr\_i;

reg [8:0] wr\_data;

wire [7:0] result, input1, input2;

wire ovf, take\_branch;

alu\_regfile UUT(

.ALUSrc1(ALUSrc1),

.ALUSrc2(ALUSrc2),

.wr\_en(wr\_en),

.clk(clk),

.rst(rst),

.rd0\_addr(rd0\_addr),

.rd1\_addr(rd1\_addr),

.wr\_addr(wr\_addr),

.ALUOp(ALUOp),

.Instr\_i(Instr\_i),

.wr\_data(wr\_data),

.result(result),

.input1(input1),

.input2(input2),

.ovf(ovf),

.take\_branch(take\_branch));

initial

begin

clk = 0;

forever #20 clk = ~clk;

end

initial

begin

// reset

#102 rst = 1;

// turn off reset

#100 rst = 0;

// writing to spot 0 the value -86

#100

wr\_en = 1'b1;

wr\_addr = 2'b00;

wr\_data = 8'b010101010;

// inverting -86 to 85 (input1 = -86)

#100

wr\_en = 1'b0;

rd0\_addr = 2'b00;

ALUSrc1 = 1'b0;

ALUOp = 3'b001;

// writing to spot 1 the value 85

#100

wr\_en = 1'b1;

wr\_addr = 2'b01;

wr\_data = 8'b01010101;

// -86 and 85 = 0 (input1 = -86, input2 = 85)

#100

wr\_en = 1'b0;

rd0\_addr = 2'b00;

rd1\_addr = 2'b01;

ALUSrc2 = 1'b0;

ALUOp = 3'b010;

// -86 + 85 = -1 (input1 = -86, input2 = 85)

#100

ALUOp = 3'b000;

// writing to spot 1 the value 85

#100

wr\_addr = 2'b01;

wr\_data = 8'b010101010;

// writing to spot 1 the value 85 when wr\_en is off and -86 or 85 = -1

#100

ALUOp = 3'b011;

// writing to spot 2 the value -86

#100

wr\_en = 1'b1;

wr\_addr = 2'b10;

wr\_data = 8'b010101010;

// writing to spot 3 the value -100

#100

wr\_addr = 2'b11;

wr\_data = 8'b010011100;

// -86 + -100 = 70 ovf 1 (input1 = -86, input2 = -100)

#100

wr\_en = 1'b0;

rd0\_addr = 2'b10;

rd1\_addr = 2'b11;

ALUOp = 3'b000;

// -86 (Instr\_i in mux) SRA = -43 (input1 = -86, input2 = -2)

#100

Instr\_i = 8'b11111110;

ALUSrc2 = 1'b1;

ALUOp = 3'b100;

// -86 beq 0 = take\_branch 0 (input1 = 0, input2 = -100)

#100

ALUSrc1 = 1'b1;

ALUSrc2 = 1'b0;

ALUOp = 3'b110;

// -2 beq 0 = take\_branch 0 (input1 = 0, input2 = -2)

#100

ALUSrc2 = 1'b1;

// -2 bne 0 = take\_branch 1 (input1 = 0, input2 = -2)

#100

ALUOp = 3'b111;

#100 $finish;

end

initial

begin

end

initial

begin

$monitor("result = ", result);

$monitor("input1 = ", input1);

$monitor("input2 = ", input2);

$monitor("ovf = ", ovf);

$monitor("take\_branch = ", take\_branch);

end

endmodule

*Simulation Waveforms*

Values in decimal

…

**Results & Analysis**

…

**Conclusion & Recommendations**

…

module alu\_regfile\_vio\_top(

input clk, // clock for vio and RegFile

input reset, // BTN0 for for reset

output [7:0] led, // add-on board led[5:0], + LD0, LD1

output wire ovf\_ctrl // LD3

);

wire [7:0] alu\_input1, alu\_input2, alu\_input2\_instr\_src;

wire [7:0] alu\_output;

wire [2:0] ALUOp;

wire alu\_ovf;

wire take\_branch;

wire RegWrite; //Write enable

wire [1:0] regfile\_ReadAddress1; //source register1 address

wire [1:0] regfile\_ReadAddress2; //source register2 address

wire [1:0] regfile\_WriteAddress; //destination register address

wire [8:0] regfile\_WriteData; //result data

wire [8:0] regfile\_ReadData1; //source register1 data

wire [8:0] regfile\_ReadData2; //source register2 data

wire ALUSrc1, ALUSrc2;

reg [7:0] zero\_register = 0;

// Assign LEDs

assign led = alu\_output;

assign ovf\_ctrl = alu\_ovf;

// Instantiate RegFile module here

reg\_file reg1(.rst(reset),.clk(clk),.wr\_en(RegWrite),.rd0\_addr(regfile\_ReadAddress1),.rd1\_addr(regfile\_ReadAddress2),.wr\_addr(regfile\_WriteAddress),.wr\_data(regfile\_WriteData),.r0\_data(regfile\_ReadData1),.r1\_data(regfile\_ReadData2));

// Instantiate Muxes here

mux\_temp mux1(.in1(regfile\_ReadData1),.in2(zero\_register),.select\_line(ALUSrc1),.out(alu\_input1));

mux\_temp mux2(.in1(regfile\_ReadData2),.in2(alu\_input2\_instr\_src),.select\_line(ALUSrc2),.out(alu\_input2));

// Instantiate ALU module here

eightbit\_alu alu1(.a(alu\_input1),.b(alu\_input2),.s(ALUOp),.f(alu\_output),.ovf(alu\_ovf),.take\_branch(take\_branch));

// Instantiate VIO module here

vio\_0 vio (

.clk(clk),

.probe\_in0(alu\_output),

.probe\_in1(alu\_ovf),

.probe\_in2(alu\_take\_branch),

.probe\_in3(regfile\_ReadData1),

.probe\_in4(regfile\_ReadData5),

.probe\_in5(alu\_input1),

.probe\_in6(alu\_input2),

.probe\_out0(alu\_input2\_instr\_src),

.probe\_out1(ALUOp),

.probe\_out2(ALUSrc1) ,

.probe\_out3(ALUSrc2),

.probe\_out4(RegWrite),

.probe\_out5(regfile\_ReadAddress1),

.probe\_out6(regfile\_ReadAddress2),

.probe\_out7(regfile\_WriteAddress),

.probe\_out8(regfile\_WriteData)

);

endmodule