module pdatapath\_top(

input wire clk,

input wire rst\_general,

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 Reg;//Write enable

//wire RegRead;//Read 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;

wire [8:0] alu\_result;

reg [7:0] zero\_register = 0; //ZERO constant

wire MemtoReg;

wire MemWrite;

wire [8:0] Data\_Mem\_Out;

assign alu\_result = {alu\_ovf, alu\_output};

// Assign LEDs

assign led = alu\_output;

assign ovf\_ctrl = alu\_ovf;

/\* Instantiate the reg-file, MUXes, ALU that you have created here\*/

reg\_file reg1(

.rst(rst\_general),

.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));

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));

mux\_temp mux3(

.in1(alu\_result),

.in2(Data\_Mem\_Out),

.select\_line(MemtoReg),

.out(regfile\_WriteData));

eightbit\_alu alu1(

.a(alu\_input1),

.b(alu\_input2),

.s(ALUOp),

.f(alu\_output),

.ovf(alu\_ovf),

.take\_branch(take\_branch));

/\* Instantiate the data memory that you have created here\*/

data\_memory data\_mem (

.a(alu\_output), // input wire [7 : 0] a

.d(regfile\_ReadData2), // input wire [8 : 0] d

.clk(clk), // input wire clk

.we(MemWrite), // input wire we

.spo(Data\_Mem\_Out) // output wire [8 : 0] spo

);

/\* Instantiate the VIO that you have created here,

make sure the number of probes and their width are correctly configured \*/

vio\_0 vio (

.clk(clk), // input wire clk

.probe\_in0(regfile\_WriteData), // input wire [9 : 0] probe\_in0

.probe\_in1(regfile\_ReadData1), // input wire [8 : 0] probe\_in1

.probe\_in2(regfile\_ReadData2), // input wire [8 : 0] probe\_in2

.probe\_in3(alu\_input1), // input wire [7 : 0] probe\_in3

.probe\_in4(alu\_input2), // input wire [7 : 0] probe\_in4

.probe\_in5(take\_branch), // input wire [0 : 0] probe\_in5

.probe\_in6(alu\_ovf), // input wire [0 : 0] probe\_in6

.probe\_in7(alu\_output), // input wire [7 : 0] probe\_in7

.probe\_in8(Data\_Mem\_Out), // input wire [8 : 0] probe\_in8

.probe\_out0(RegWrite), // output wire [0 : 0] probe\_out0

.probe\_out1(alu\_input2\_instr\_src), // output wire [7 : 0] probe\_out1

.probe\_out2(ALUSrc1), // output wire [0 : 0] probe\_out2

.probe\_out3(ALUSrc2), // output wire [0 : 0] probe\_out3

.probe\_out4(ALUOp), // output wire [2 : 0] probe\_out4

.probe\_out5(MemWrite), // output wire [0 : 0] probe\_out5

.probe\_out6(MemtoReg), // output wire [0 : 0] probe\_out6

.probe\_out7(regfile\_ReadAddress1), // output wire [1 : 0] probe\_out7

.probe\_out8(regfile\_ReadAddress2), // output wire [1 : 0] probe\_out8

.probe\_out9(regfile\_WriteAddress) // output wire [1 : 0] probe\_out9

);

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 signed [8:0] wr\_data,

output [7:0] r0\_data,

output [7:0] r1\_data

);

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

integer i = 0; //for the loop

always @ (posedge clk, posedge rst)

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

for (i=0; i < 4; i=i+1) RegFile[i] = 0;

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

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

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

endmodule

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 intruction\_decoder(

input [15:0] instruction,

output RegWrite,

output [7:0] Instr\_i,

output ALUSrc1,

output ALUSrc1,

output [2:0] ALUOp,

output MemWrite,

output [3:0] opcode,

output MemToReg,

output [1:0] rs\_addr,

output [1:0] rt\_addr,

output [1:0] rd\_addr);

assign opcode = instruction[15:12];

always@(\*)

begin

end

endmodule

| **Opcode** | **RegDst** | **RegWrite** | **ALUSrc1** | **ALUSrc2** | **ALUOp[2:0]** | **MemWrite** | **MemToReg** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *0000 (lw)* | 1’b0 | 1’b1 | 1’b0 | 1’b1 | 3’b000 | 1’b0 | 1’b1 |
| *0001 (sw)* | 1’bx | 1’b0 | 1’b0 | 1’b1 | 3’b000 | 1’b1 | 1’bx |
| *0010 (add)* | 1’b1 | 1’b1 | 1’b0 | 1’b0 | 3’b000 | 1’b0 | 1’b0 |
| *0011 (addi)* | 1’b0 | 1’b1 | 1’b0 | 1’b1 | 3’b000 | 1’b0 | 1’b0 |
| *0100 (inv)* | 1’b1 | 1’b1 | 1’b0 | 1’b0 | 3’b001 | 1’b0 | 1’b0 |
| *0101 (and)* | 1’b1 | 1’b1 | 1’b0 | 1’b0 | 3’b010 | 1’b0 | 1’b0 |
| *0110 (andi )* | 1’b0 | 1’b1 | 1’b0 | 1’b1 | 3’b010 | 1’b0 | 1’b0 |
| *0111 (or)* | 1’b1 | 1’b1 | 1’b0 | 1’b0 | 3’b011 | 1’b0 | 1’b0 |
| *1000 (ori)* | 1’b0 | 1’b1 | 1’b0 | 1’b1 | 3’b011 | 1’b0 | 1’b0 |
| *1001 (sra)* | 1’b1 | 1’b1 | 1’b0 | 1’b0 | 3’b100 | 1’b0 | 1’b0 |
| *1010 (sll)* | 1’b0 | 1’b1 | 1’b0 | 1’b0 | 3’b101 | 1’b0 | 1’b0 |
| *1011 (beq)* | 1’bx | 1’b0 | 1’bx | 1’b0 | 3’b110 | 1’b0 | 1’bx |
| *1100 (bne)* | 1’bx | 1’b0 | 1’bx | 1’b0 | 3’b111 | 1’b0 | 1’bx |
| *1101 (clr)* | 1’bx | 1’b1 | 1’b1 | 1’b0 | 3’b010 | 1’b0 | 1’b0 |