Prelab 4

Timothy Van Slyke

February 12, 2018

1 Part 1

```
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 reg [8:0] rd0_data,
  output reg [8:0] rd1_data
);
  reg [3:0] [8:0] storage;

  always @(posedge clk) begin
    if(rst) begin
      storage[0] <= 0;
      storage[1] <= 0;
      storage[2] <= 0;
      storage[3] <= 0;
    end else if(wr_en) begin
      storage[wr_addr] <= wr_data;
    end else begin
      rd0_data <= storage[rd0_addr];
      rd1_data <= storage[rd1_addr];
    end
  end
endmodule
```

2 Test Cases

time (ns)	ALU operand 1	ALU operand 2	ALU output	ALU ovf	ALU take_branch
0	xxxxxxxx	xxxxxxxx	xxxxxxxx	x	x
10	xxxxxxxx	xxxxxxxx	xxxxxxxx	x	0
20	xxxxxxxx	xxxxxxxx	xxxxxxxx	x	0
30	xxxxxxxx	xxxxxxxx	xxxxxxxx	x	0
40	10000000	10000001	00000001	1	0
50	10000000	10000001	00000001	1	0
60	10000000	11100111	01100111	1	0
70	10000000	11100111	01100111	1	0
80	10000000	11100111	01100111	1	0
90	10000000	11100111	01100111	1	0
100	10000000	11100111	01100111	1	0
110	00000000	11100111	11100111	0	0
120	10000000	11111111	01111111	1	0
130	00000000	11111111	11111111	0	0
140	10000000	11100111	01100111	1	0

Figure 1: Outputs for each test vector.

time (ns)	rst	write enable	write data	instr_i	mux 1 select	mux 2 select	ALU op
5	0	0	000000000	00000000	0	0	000
15	0	1	010000000	00000000	0	0	000
25	0	1	010000001	00000000	0	0	000
35	0	0	010000001	00000000	0	0	000
45	0	1	011100111	00000000	0	0	000
55	0	0	011100111	00000000	0	0	000
65	1	0	011100111	00000000	0	0	000
75	0	1	011100111	00000000	0	0	000
85	0	1	010000000	00000000	0	0	000
95	0	0	010000000	11111111	0	0	000
105	0	0	010000000	11111111	1	0	000
115	0	0	010000000	11111111	0	1	000
125	0	0	010000000	11111111	1	1	000
135	0	0	010000000	11111111	0	0	000
145	0	0	010000000	11111111	0	0	000

Figure 2: Inputs for each test vector.

Other Modules Used for Simulation

```
module alu_reg(
  // inputs
  input rst,
  input clk,
  input RegWrite,
  input [1:0] ReadAddr1,
  input [1:0] ReadAddr2,
  input [1:0] WriteAddr,
  input [8:0] WriteData,
  input [7:0] Instr_i,
  input ALUSrc1,
  input ALUSrc2,
  input [2:0] ALUOp,
  // outputs
  output reg [7:0] result,
  output wire [7:0] input1,
  output reg [7:0] input2,
  output reg ovf,
  output reg take_branch
);
  // reg file outpus
  reg [8:0] reg_data0;
  reg [8:0] reg_data1;
  // mux'd inputs to alu

  // register file instantiation
  reg_file RegFile(
    .rst(rst),
    .wr_en(RegWrite),
    .clk(clk),
    .rd0_addr(ReadAddr1),
    .rd1_addr(ReadAddr2),
    .wr_addr(WriteAddr),
    .wr_data(WriteData),
    .rd0_data(reg_data0),
    .rd1_data(reg_data1)
  );

  assign input1 = ALUSrc1 ? 8'b0000_0000 : reg_data0;
  assign input2 = ALUSrc2 ? Instr_i : reg_data1;

  // ALU instantiation
  eightbit_alu EightbitALU(
    .a(input1),
    .b(input2),
    .s(ALUOp),
    .f(result),
```

```
    .ovf(ovf),
    .take_branch(take_branch)
  );

endmodule

module alu_reg_tb();
  reg clk;
  always #5 clk = !clk;
  reg rst;
  reg write_enable;
  reg [1:0] read_address1;
  reg [1:0] read_address2;
  reg [1:0] write_address;
  reg [8:0] write_data;
  reg [7:0] instr_i;
  reg alu_src1;
  reg alu_src2;
  reg [2:0] alu_op;

  wire [7:0] alu_result;
  wire [7:0] alu_input1;
  wire [7:0] alu_input2;
  wire ofv;
  wire take_branch;

  alu_reg AluReg(
    // inputs
    .rst(rst),
    .clk(clk),
    .RegWrite(write_enable),
    .ReadAddr1(read_address1),
    .ReadAddr2(read_address2),
    .WriteAddr(write_address),
    .WriteData(write_data),
    .Instr_i(instr_i),
    .ALUSrc1(alu_src1),
    .ALUSrc2(alu_src2),
    .ALUOp(alu_op),
    // outputs
    .result(alu_result),
    .input1(alu_input1),
    .input2(alu_input2),
    .ovf(ovf),
    .take_branch(take_branch)
  ); // AluReg();

  integer test_duration = 150;

  always @(posedge clk) begin
```

```
  if ($time < test_duration) begin
    $display("Clock␣Tick␣(posedge␣t␣=␣%d):␣", $time);
    $display({"Inputs:\n",
      "rst␣=␣%b,␣",
      "write␣enable␣=␣%b,␣",
      "write␣data␣=␣%b,␣",
      "instr_i␣=␣%b,␣",
      "mux␣1␣select␣=␣%b,␣",
      "mux␣2␣select␣=␣%b,␣",
      "ALU␣op␣=␣%b,␣"
    }, rst, write_enable, write_data, instr_i, alu_src1, alu_src2, alu_op);
  end
end
always @(negedge clk) begin
  if ($time < test_duration) begin
    $display("Clock␣Tick␣(negedge␣t␣=␣%d):␣", $time);
    $display({"Ouputs:\n",
      "ALU␣operand␣1␣=␣%b,␣",
      "ALU␣operand␣2␣=␣%b,␣",
      "ALU␣output␣=␣%b,␣",
      "ALU␣ovf␣=␣%b,␣",
      "ALU␣take_branch␣=␣%b,␣"
    }, alu_input1, alu_input2, alu_result, ovf, take_branch);
  end
end
initial begin
  clk = 0;
  rst = 0;
  read_address1 = 0;
  read_address2 = 1;
  write_enable = 1;
  write_address = 0;
  write_data = 0;
  instr_i = 0;
  alu_src1 = 0;
  alu_src2 = 0;
  alu_op = 0;
  #10  // load value into addr 0
  write_address = 0;
  write_data = 8'b1000_0000;
  #10 // load value into addr 1
  write_address = 1;
  write_data = 8'b1000_0001;
  #10 // read values into respective operands
  write_enable = 0;
  read_address1 = 0;
  read_address2 = 1;
  #10 // load into address 2
  write_enable = 1;
  write_address = 2;
```

```
write_data = 8'b1110_0111;
#10 // read address 2 into operand 2
write_enable = 0;
read_address2 = 2;
#10 // test reset functionality
rst = 1;
#10 //load values back into registers
rst = 0;
write_enable = 1;
write_address = 2;
write_data = 8'b1110_0111;
#10
write_address = 0;
write_data = 8'b1000_0000;
#10 // read values we just loaded into operands
write_enable = 0;
read_address1 = 0;
read_address2 = 2;

// test multiplexers
instr_i = 8'b1111_1111;
#10
alu_src1 = 1;
alu_src2 = 0;
#10
alu_src1 = 0;
alu_src2 = 1;
#10
alu_src1 = 1;
alu_src2 = 1;
#10
alu_src1 = 0;
alu_src2 = 0;
#10
alu_op = 0; // no-op
// test ALU opcodes
// #10
// alu_op = 1;
// #10
// alu_op = 1;
// #10
// alu_op = 2;
// #10
// alu_op = 3;
// #10
// alu_op = 4;
// #10
// alu_op = 5;
// #10
// alu_op = 6;
```

```
    // #10
    // alu_op = 7;
    // #10
    // alu_op = 7; // noop
  end

endmodule
```