Lab 8

Jiasen Zhou, Jon Johnston

March 18, 2019

1 Executive Summary

The Executive Summary section should be a single concise paragraph that describes the following items. Please note that this should be a paragraph, not an itemized list like I have below. The list is just to tell you what I'm looking for...it is not there for you to fill it in on your lab report. See the Lab 1 Example Report.pdf on Canvas to see a good example of what I'm looking for.

1. The goal of the lab
2. What modules you created and what they do. This should describe how the modules function and how they fit into the overall processor that we are building. For instance, for Lab 1, you will want to describe the operation of the register and the fact that it is used to store the program counter, which is used to keep track of which instruction to execute next.
3. Whether your lab was successful. If not successful, please state what is not currently working.

2 Test Report

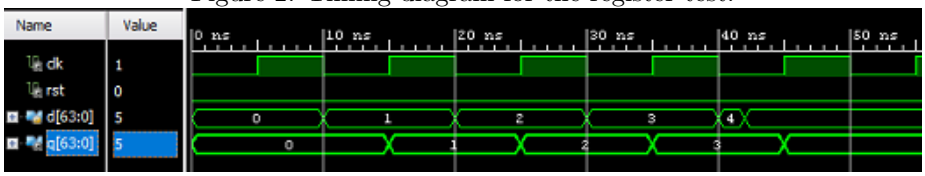
To verify operation of this/these module(s), this lab requires N test bench(es). (You can just copy this statement and fill it in with the correct number of test benches)

1. iExecute Test Bench

Figure 1: Expected Results of the register test.

Time(ns)	0-10	10-15	15-20	20-25	25-30	30-35	35-40	40-42	42-45	45+
rst	0	0	0	0	0	0	0	0	0	0
d	0	1	1	2	2	3	3	4	5	5
q	0	0	1	1	2	2	3	3	3	5

Figure 2: Timing diagram for the register test.



3 Code Appendix

Listing 1: Verilog code for testing a register.

```
`include "definitions.vh"


module register_test;

wire clk;
reg rst;
reg [`WORD - 1:0] d;
wire [`WORD - 1:0] q;

oscillator clk_gen(clk);

register UUT(
    .clk(clk),
    .reset(rst),
    .D(d),
    .Q(q)
    );

initial
begin
    rst = 0;
    d<='WORD'd0; #`CYCLE;
    d<='WORD'd1; #`CYCLE;
    d<='WORD'd2; #`CYCLE;
```

```
    rst = 1;
    d<='WORD'd3; #`CYCLE;
    d<='WORD'd4; #(`CYCLE/5);
    rst = 0;
    d<='WORD'd5; #(`CYCLE*4/5);
end

endmodule
```

Listing 2: Verilog code for iExecute module

```
`include "definitions.vh"
module iExecute(
    input [`WORD-1:0] pc_in,
    input [`WORD-1:0] read_data1,
    input [`WORD-1:0] read_data2,
    input [`WORD-1:0] sign_extend,
    input [10:0] opcode,
    input [1:0] alu_op,
    input alu_src,
    output [`WORD-1:0] alu_result,
    output zero,
    output [`WORD-1:0] branch_target
    );

    wire [`WORD-1:0] mux_out;
    wire alu_ctrl_out;
    wire [`WORD-1:0] shift_result;

    mux MUX (
        .a_in(read_data2),
        .b_in(sign_extend),
        .control(alu_src),
        .mux_out(mux_out));

    ALU alu(
        .a_in(read_data1),
        .b_in(mux_out),
        .alu_control(alu_ctrl_out),
        .alu_result(alu_result),
        .zero_flag(zero)
        );

    alu_control alu_ctrl(
        .ALUOp(alu_op),
        .opcode(opcode),
```

```
        .ALU_control(alu_ctrl_out));

        //shift sign extend left by 2 = *4
    assign shift_result = sign_extend*4;

    adder pc_adder(
        .a_in(pc_in),
        .b_in(shift_result),
        .add_out(branch_target)
        );

endmodule
```