Lab 2 - Program Counter

Steve Potter

January 30, 2019

1 Executive Summary

The goal of this lab is to create an adder module and a mux module. These modules will initially be used in the Fetch stage of our 64-bit ARMv8 processor. The adder will be used to increment the Program Counter (PC). The incremented PC will be used for sequential program execution. The mux will be used to set the PC to either the incremented PC or to a branch address. This selection will be based on the mux control line, which specifies whether the program should branch or continue running sequentially. The module works correctly as shown by the test report below

2 Test Report

To verify operation of these two modules, this lab requires two separate test benches.

1. Adder Test Bench
2. Mux Test Bench

Figure 1: Expected Results of the adder test.

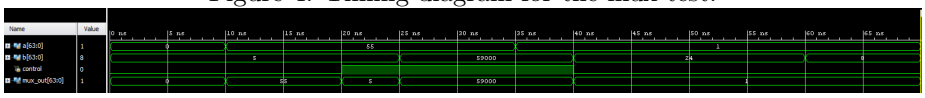
	0-10	10-20	20-30	30-35	35-55	55-65
a	0	55	55	1	1	1
b	5	5	59000	59000	24	8
add_out	5	60	59055	59001	25	9

Figure 2: Timing diagram for the adder test.

Figure 3: Expected Results of the mux test.

	0-10	10-20	20-25	25-35	35-40	55-75	75-85
a	0	55	55	55	1	1	1
b	5	5	5	59000	59000	24	8
control	0	0	1	1	1	0	0
add_out	0	55	5	59000	59000	1	1

Figure 4: Timing diagram for the mux test.



3 Code Appendix

Listing 1: Verilog code for testing the adder.

```
`include "definitions.vh"

module adder_test;

reg [`WORD - 1:0] a;
reg [`WORD - 1:0] b;
wire [`WORD - 1:0] add_out;

adder UUT(
    .a_in(a),
    .b_in(b),
    .add_out(add_out)
    );

initial
begin
    a<='WORD'd0;
    b<='WORD'd5; #`CYCLE;
    a<='WORD'd55; #`CYCLE;
    b<='WORD'd59000; #`CYCLE;
    a<='WORD'd1; #(`CYCLE/2);
    b<='WORD'd24; #(2*`CYCLE);
    b<='WORD'd8; #`CYCLE
    $finish;
end
```

```
endmodule
```

Listing 2: Verilog code for testing the mux.

```
`include "definitions.vh"

module mux_test;

reg [`WORD - 1:0] a;
reg [`WORD - 1:0] b;
reg control;
wire [`WORD - 1:0] mux_out;

mux#(64) UUT(
    .a_in(a),
    .b_in(b),
    .control(control),
    .mux_out(mux_out)
    );

initial
begin
    control<=1'b0;
    a<='WORD'd0;
    b<='WORD'd5; #`CYCLE;
    a<='WORD'd55; #`CYCLE;
    control<=1'b1; #(`CYCLE/2);
    b<='WORD'd59000; #`CYCLE;
    a<='WORD'd1; #(`CYCLE/2);
    control<=1'b0;
    b<='WORD'd24; #(2*`CYCLE);
    b<='WORD'd8; #`CYCLE
    $finish;
end
endmodule
```

Listing 3: Verilog code for the adder.

```
`include "definitions.vh"

module adder(
    input [`WORD-1:0] a_in,
    input [`WORD-1:0] b_in,
    output [`WORD-1:0] add_out
    );
    assign add_out = a_in+b_in;
```

```
endmodule
```

Listing 4: Verilog code for the mux.

```
`include "definitions.vh"

module mux#(
    parameter SIZE=8)(
    input [SIZE-1:0] a_in,
    input [SIZE-1:0] b_in,
    input control,
    output [SIZE-1:0] mux_out
    );
    assign mux_out = control?b_in:a_in;
endmodule
```