Lab 3 Fetch Stage

Justin Roessler and Jon Johnston

February 5, 2019

1 Executive Summary

The purpose of this lab is to finish the fetch stage of our arm processor. The fetch stage updates the PC with each positive clock edge(add 4), reads the value of the program counter, takes the information at that memory location, and sends it out to the next stage to be executed. The fetch stage can also jump to a branch address if the pc_src is set to 1.

A mux was used to switch between the program counter's address and the branch address depending on the pc_src. Also, an add module was used to add 4 to the PC with each positive clock edge. Lastly, an instruction memory module was created to take the address location from the mux and output the contents of that address to the decode stage of the processor. After implementing some delays and these modules, the lab was a success and the fetch stage is operating as expected.

2 Test Report

To verify operation of this/these module(s), this lab requires 2 test benches.

1. Instruction Memory Test Bench
2. Fetch Test Bench

Figure 1: Expected Results of the Instruction Memory test.

Time(ns)	0-5	5-10	10-15	15-20	20-25	25-30	30-35
clk	0	1	0	1	0	1	0
instruction	X	2332623529	2332623529	2332623530	2332623530	2332623531	2332623531
address	0	0	4	4	8	8	32

Time(ns)	35-40	40-45	45-50	50-55	55-60	60-65
clk	1	0	1	0	1	0
instruction	2332623537	2332623537	2332623533	2332623533	2332623539	2332623539
address	32	16	16	40	40	40

Figure 2: Timing Diagram for the Instruction Memory test.

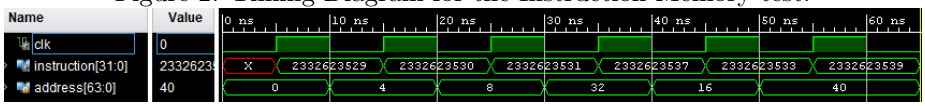
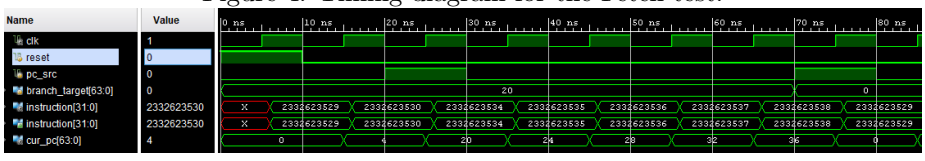


Figure 3: Expected Results of the Fetch test.

Time(ns)	0-5	5-10	10-15	15-20	20-25	25-30	30-35	35-40
clk	0	1	0	1	0	1	0	1
reset	1	1	0	0	0	0	0	0
pc_src	0	0	0	0	1	1	0	0
branch_target	20	20	20	20	20	20	20	20
instruction	X	2332623529	2332623529	2332623530	2332623530	2332623534	2332623534	2332623535
cur_pc	0	0	0	4	4	20	20	24

Time(ns)	40-45	45-50	50-55	55-60	60-65	65-70	70-75	75-80
clk	0	1	0	1	0	1	0	1
reset	0	0	0	0	0	0	0	0
pc_src	0	0	0	0	0	0	1	1
branch_target	20	20	20	20	20	20	0	0
instruction	2332623535	2332623536	2332623536	2332623537	2332623537	2332623538	2332623538	2332623529
cur_pc	24	28	28	32	32	36	36	0

Figure 4: Timing diagram for the Fetch test.



3 Code Appendix

Listing 1: Verilog code for testing a Instruction Memory.

```
`include "definitions.vh"

module instr_mem_test;

    wire clk;
    wire [`INSTR_LEN - 1:0] instruction;
    reg [`WORD - 1:0] address;

    oscillator clock(
        .clk(clk)
        );

    instr_mem memory(
        .clk(clk),
        .address(address),
        .instruction(instruction)
        );

    initial
    begin
        address = 0; #10;
        address = 4; #10;
        address = 8; #10;
        address = 32; #10;
        address = 16; #10;
        address = 40; #15;
        $finish;
    end

endmodule
```

Listing 2: Verilog code for implementing a Instruction Memory.

```
`include "definitions.vh"

module instr_mem#(
    parameter SIZE=1024)(
     input  [`WORD - 1:0] address,
     input clk,
     output reg [`INSTR_LEN - 1:0] instruction
    );
```

```
    // imem is the instruction memory itself
    // imem is initially populated by reading
    // a file one time in the initial section
    reg [`INSTR_LEN - 1:0] imem [SIZE-1:0];

    // Add code here to output the correct instruction
    always @(posedge(clk))
    begin
        instruction <= imem[address/4];
    end
    // initialize instruction memory from a file
    initial
        $readmemb(`IMEMFILE, imem);

endmodule
```

Listing 3: Verilog code for testing the Fetch Stage.

```
`include "definitions.vh"

module fetch_test;

    wire clk;
    reg reset, pc_src;
    reg [`WORD-1:0] branch_target;
    wire [`INSTR_LEN-1:0] instruction;

    oscillator clock(.clk(clk));

    fetch iFetch(.clk(clk), .reset(reset), .branch_target(branch_target),
                 .pc_src(pc_src), .instruction(instruction));

    initial
    begin
        reset = 1;
        pc_src= 0;
        branch_target= 20;#10;
        reset = 0;#10;
        pc_src= 1;#10;
        pc_src = 0; #40;
        branch_target= 0;
        pc_src= 1;#10;
        pc_src = 0; #100;
        $finish;
    end
```

```
endmodule
```

Listing 4: Verilog code for implementing the Fetch Stage.

```
`include "definitions.vh"

module fetch #(parameter SIZE = 64)
    (input [`WORD-1:0]  branch_target,
    input pc_src, clk, reset,
    output [`INSTR_LEN-1:0] instruction);

    wire [`WORD-1:0] cur_pc,new_pc, incremented_pc,clkplus1;

    mux MUX (.a_in(incremented_pc), .b_in(branch_target), .control(pc_src), .mux
    register PC(.clk(clk), .reset(reset), .D(new_pc), .Q(cur_pc));
    adder ADD(.a_in(cur_pc), .b_in(4), .add_out(incremented_pc));

    delay DELAY(.a(clk), .a_delayed(clkplus1));
    instr_mem iMEM(.address(cur_pc), .clk(clkplus1), .instruction(instruction));

endmodule
```