Lab 4

Sam Sahli and Shaun Gordon

February 19, 2017

1 Introduction

After the instruction memory had been constructed and its output had been tested, it was evident the output of the instruction memory lagged by one cycle from the program counter. To fix this problem, a buffer module was constructed and introduced to the instruction memory in this lab. If the new buffer module was implemented correctly, the new output of the Fetch test would show no lag from the program counter.

2 Interface

In Lab 3, the outputs from the Fetch stage were the instruction, from the instruction memory, and the next value of the program counter, from the adder. These outputs will remain the same for this Lab, except now the outputs from the memory and adder will go to the buffer module first. The buffer module will hold these values while a delay is run, and then output the unaltered next program counter value and instruction from the Fetch stage. This delay, or buffer, allows the Fetch group to calculate new values while the next stage is allowed to operate on fixed values.

3 Design

The buffer module is designed with four inputs and two outputs. There are single bit clock and relay inputs. The clock controls the module's operation, while the reset, when raised, will set the outputs to zero. The other two inputs are the next program counter value and the instruction from memory. These inputs each come in on a 32-bit wire. They are then stored on the two corresponding 32-bit output registers. No changes are made to these values, as the only needed function of the buffer is a time delay and the storage of previously determined values. It is useful to think of the buffer as a register that hold these values for a specific time interval.

4 Implementation

This lab required the complete construction of one Verilog file and the modification of another. Of course the buffer module was the file that needed to be fully constructed and implemented. The code for this module is displayed in Listing 1. It can be observed that the module is created with a parameter titled DELAY. This is the value that determines the size of the time delay between the input and output. After the creation of the variables, an always statement is triggered at the negative edge of the clock and the positive edge of the reset. This means that as the clock runs, all the operations of the buffer will be executed as the clock goes from high to low. This allows the rest of the Fetch stage to continue operating during the high phase of a clock cycle, and the outputs are given time to be carried through all modules successfully. Next a delay is run to allow the instruction to be fetched and not lag the value of the program counter. Then, if the reset is raised, both outputs are set to zero, and if not, both outputs are set to both inputs after the delay.

The modified Verilog file is the Fetch module constructed in lab 3. The modified code for this stage is displayed in Listing 2. The only difference between this code and the code from Lab 3 is the inclusion of the buffer module and the creation of a 32-bit internal wire called instruction. The instruction wire is used to carry the instruction from the memory to the buffer. The buffer's DELAY parameter is set to the value DELAY found in the Verilog header file. Then all the inputs and outputs of the buffer module are matched with the already existing internal wires and outputs in the Fetch stage. This links the buffer and Fetch module.

Listing 1: Verilog code for implementing a buffer.

```
`include "definitions.vh"

module buffer_ifid#(parameter DELAY=0)(
    input clk,
    input reset,
    input [`WORD-1:0] nPC_if,
    input [`WORD-1:0] IR_if,
    output reg[`WORD-1:0] nPC_id=`ZERO,
    output reg[`WORD-1:0] IR_id=`ZERO
    );

    always @(negedge(clk), posedge(reset))
    begin
      #DELAY;
      if (reset)
      begin
        nPC_id<=`ZERO;
        IR_id<=`ZERO;
      end
```

```
        else
        begin
          nPC_id<=nPC_if;
          IR_id<=IR_if;
        end
    end
endmodule
```

Listing 2: Verilog code for modifying the iFetch stage.

```
`include "definitions.vh"


module iFetch#(parameter STEP=32'd1, SIZE=1024)(
    input clk,
    input reset,
    input PCSrc,
    input [`WORD-1:0] BrDest,
    output [`WORD-1:0] nPC,
    output [`WORD-1:0] IR
    );
    wire [`WORD-1:0] PC;
    wire [`WORD-1:0] new_PC;
    wire [`WORD-1:0] nextPC;
    wire [`WORD-1:0] instruction;


    mux#(`WORD) PCsel(
    .Ain(nextPC),
    .Bin(BrDest),
    .control(PCSrc),
    .mux_out(new_PC)
    );

    register myPC(
    .clk(clk),
    .reset(reset),
    .D(new_PC),
    .Q(PC)
    );

    adder incrementer(
    .Ain(PC),
    .Bin(STEP),
    .add_out(nextPC)
    );
```

```
    instr_mem#(SIZE) iMemory(
    .clk(clk),
    .pc(PC),
    .instruction(instruction)
    );

    buffer_ifid#(`DELAY) buffer(
    .clk(clk),
    .reset(reset),
    .nPC_if(nextPC),
    .IR_if(instruction),
    .nPC_id(nPC),
    .IR_id(IR)
    );
endmodule
```

5 Test Bench Design

The only test bench that needs to be created in this lab is the buffer test bench. The Fetch test bench required no modification but is still included in this report for reference in Listing 4. The buffer test bench can be seen in Listing 3. The purpose of this test bench is to observe that the buffer delays the output of both inputs by the value that the DELAY parameter is set to. This is done by assigning input values and observing outputs with the same value after the time delay. A low, middle, and high 32-bit value should be tested to guarantee the buffer works for all possible input values. The reset also needs to be raised to test whether the output is set to zero.

Listing 3: Verilog code for testing a buffer.

```
`include "definitions.vh"


module buffer_ifid_test;


    wire clk;
    reg reset=0;
    reg [`WORD-1:0] nPC_if;
    reg [`WORD-1:0] IR_if;
    wire [`WORD-1:0] nPC_id;
    wire [`WORD-1:0] IR_id;

    oscillator clk_gen(clk);
```

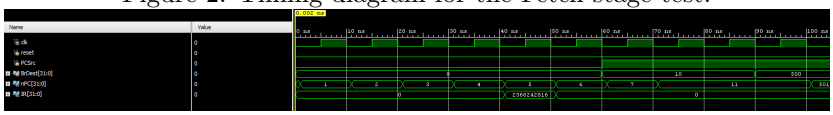
```
    buffer_ifid#(`DELAY) UUT (
        .clk(clk),
        .reset(reset),
        .nPC_if(nPC_if),
        .IR_if(IR_if),
        .nPC_id(nPC_id),
        .IR_id(IR_id)
    );

        initial begin
            nPC_if = 0;
            IR_if = 0;
            #`CYCLE;

        nPC_if = 1;
        IR_if = 1;
        #`CYCLE;

        nPC_if = 2;
        IR_if = 2;
        #`CYCLE;

        nPC_if = 3;
        IR_if = 3;
        #`CYCLE;

        nPC_if = 1023;
        IR_if = 1023;
        #`CYCLE;

        nPC_if = 4294967295;
        IR_if = 4294967295;
        #`CYCLE;

        reset = 1;
        #`CYCLE;

        reset = 0;
        #`CYCLE;

    end

endmodule
```

Listing 4: Verilog code for testing a Fetch stage.

```
`include "definitions.vh"


module iFetch_test;

    wire clk;
    reg reset=0;
    reg PCSrc=0;
    reg [`WORD-1:0] BrDest;
    wire [`WORD-1:0] nPC;
    wire [`WORD-1:0] IR;

    oscillator clk_gen(clk);

        iFetch UUT (
                .clk(clk),
        .reset(reset),
        .PCSrc(PCSrc),
        .BrDest(BrDest),
        .nPC(nPC),
        .IR(IR)
        );

        initial begin
            BrDest = 0;
            #`CYCLE;
        #`CYCLE;
        #`CYCLE;
        #`CYCLE;
        #`CYCLE;
        #`CYCLE;

        BrDest = 10;
        PCSrc = 1;
        #`CYCLE;
        #`CYCLE;
        #`CYCLE;

        BrDest = 500;
        #`CYCLE;
        #`CYCLE;

        BrDest = 20;
        #(`CYCLE/5);
```

Figure 1: Timing diagram for the buffer test.

Figure 2: Timing diagram for the Fetch stage test.



```
        BrDest = 1023;
        #(`CYCLE*4/5);

        PCSrc = 0;
        #`CYCLE;

        reset = 1;
        #`CYCLE;

        reset = 0;
        #`CYCLE;

    end


endmodule
```

6 Simulation

The timing diagram for the buffer test is shown in Figure 1. This timing diagram shows that if the inputs are set to a 32-bit value the output will be delayed by the determined DELAY parameter. The diagram also shows that when reset is raised, both outputs are set to zero. This confirms that the buffer is performing the desired operation.

The timing diagram for the Fetch stage is shown in Figure 2. The important thing to observe in this diagram is the delay of the output and the outputs being trigger on the negative edge of the clock. Both of these traits are observed in the diagram, so this simulation confirms the buffer is successfully integrated into the Fetch stage.

7 Conclusions

The purpose of this lab was to build and test a buffer module and integrate it into the Fetch stage. A buffer was successfully built and testing was utilized to confirm that it was operating correctly. The purpose of the buffer was to eliminate the lag between the instruction output and the value of the program counter. This was done by using a time delay and triggering the output on the negative edge of the clock. This allows the fetch module to continue calculating the next value during the positive cycle of the clock. Now that the buffer's operation has been confirmed and implemented, the construction of the MIPS computer Fetch stage is complete.