Lab title

your names

November 20, 2017

1 Introduction

Introduction with problem overview, your design procedure, and rationale. Be as brief as possible to let me know the big picture of the lab.

2 Interface

Since we have already shown our inputs and outputs for the non-pipelined datapath, there is no need to detail every input and output again. Therefore, you can leave this section blank.

3 Design

This section should show your pipeline analysis spreadsheet where we showed the contents of the pipeline buffers. If you can display it in a readable way in this document, then please do so. Otherwise, include the document or an image of the document in your Git repository and point to that document (it does not have to be a link, just a description of where it is in your repository).

4 Implementation

Show your Verilog code that is pertinent to pipelining. This should include (but is not limited to) datapath.v, iFetch.v, iDecode.v, iExecute.v, iMemory.v, and iWriteBack.v. Code should be included in the report like Listing 1 on page 1.

Listing 1: Verilog code for implementing a register.

```
`include "definitions.vh"

module register(
    input clk,
    input reset,
    input  [`WORD-1:0] D,
    output reg [`WORD-1:0] Q=`WORD'b0
    );

    always @(posedge(clk),posedge(reset)) begin
        if (reset==1'b1)
            Q<='WORD'b0;
        else
            Q <= D;
    end

endmodule
```

5 Test

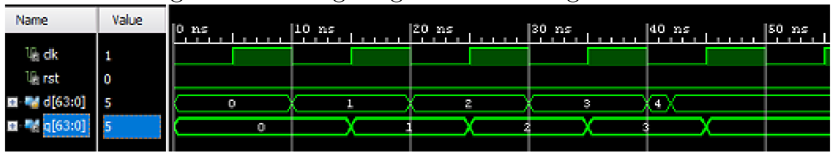
This is where you should show the code you used to test your pipeline and the results. Please include:

1. List of assembly commands that you used to test your pipeline
2. Binary for the commands you have been using to test your datapath since the iDecode stage
3. Simulation Results from these commands. Please make sure that I can see all values in the table. This might be difficult, as there are a lot of signals. You can use multiple diagrams if necessary. A sample simulation diagram is in Figure 1 on page 3.

Listing 2: Verilog code for testing a register.

```
`include "definitions.vh"


module test_regs;

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

oscillator clk_gen(clk);

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

Figure 1: Timing diagram for the register test.



```
initial
begin
        d<=`WORD' d0;  #`CYCLE;
        d<=`WORD' d1;  #`CYCLE;
        d<=`WORD' d2;  #`CYCLE;
        d<=`WORD' d3;  #`CYCLE;
        d<=`WORD' d4;  #(`CYCLE/5);
        d<=`WORD' d5;  #(`CYCLE*4/5);
end

endmodule
```

6 Conclusions

Describe the behavior you achieved in your pipeline. Did it work correctly? How long did it take to execute? How long would it take on a non-pipelined datapath? Can you make it run faster by reducing the cycle time? What can be done in future labs to make it run faster?