Lab 10 Memory

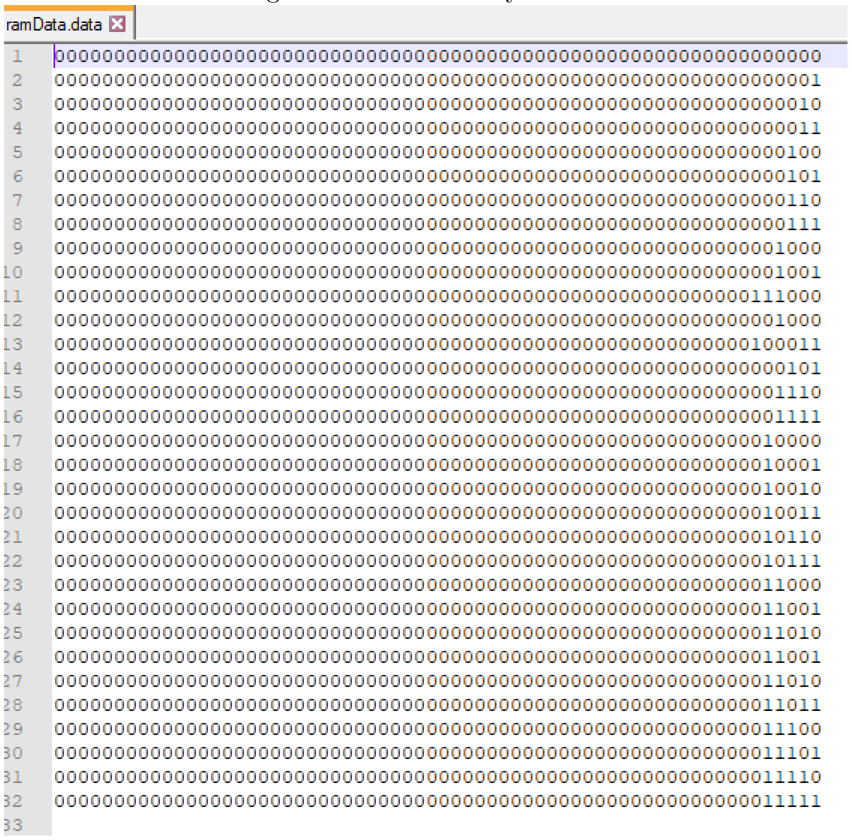
Matthew Carrano and Breana Leal

March 28, 2018

1 Results

The data memory and iMemory modules work correctly. Both modules were verified through the iMemory testbench which consisted of six tests, each a cycle long. Inside the test bench, the input variables are mem_read, mem_write, zero, branch, uncond_branch, alu_result, and read_data2. All variable names correspond to the name in the datapath picture. The output variables are or_result (the branch decision) and read_data. read_data corresponds to the picture, and or_result is simply the result of the OR gate in the iMemory module. The goal of the first test was to verify the memory read capability as well as the or_result when all branch related inputs are 0. It is seen from the simulation in Figure 2 on page 5 that on the first positive edge of the clock, or_result is 0 and read_data is 1. This is expected because zero, branch, and uncond_branch are all set to 0 which means the inputs of the OR gate are 0. Additionally, mem_read is set to 1 and alu_result to 2. This means the module should read the data from address 1 of data memory, and output that data through read_data. Data memory was initialized with Figure 1 on page 2. Therefore, address 1 of data memory contains the value 1. This is why read_data equals 1. On the two next cycles, the memory write functionality was tested. This was done by turning mem_read off and mem_read on, and then vice versa in the following cycle. alu_result was also changed to 2 during the write test. In this way, the value of read_data2, arbitrarily set to 1234 in the first cycle, is written to address 2 of data memory and in the next cycle (mem_read = 1, mem_write = 0) the value of address 2 is read from data memory. Therefore, alu_data is 1234 at the start of the third cycle, as seen in the simulation. Finally, the branch decision was tested further. First, branch was set to 1, while the other branch-related inputs remained at zero. With the zero flag set to 0, the AND gate will output 0, and therefore both inputs of the OR gate are zero. Therefore, the or_result is 0. Next, zero was set to 1. This means the AND gate will output 1, and therefore the or_result is 1. Lastly, branch and zero are set to 0, but uncond_branch to 1. Thus, one of the OR gate inputs is 1 and so its output is also 1.

Figure 1: Data Memory initial values.



```
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000001
0000000000000000000000000000000000000000000000000000000000000010
0000000000000000000000000000000000000000000000000000000000000011
0000000000000000000000000000000000000000000000000000000000000100
0000000000000000000000000000000000000000000000000000000000000101
0000000000000000000000000000000000000000000000000000000000000110
0000000000000000000000000000000000000000000000000000000000000111
0000000000000000000000000000000000000000000000000000000000001000
0000000000000000000000000000000000000000000000000000000000001001
0000000000000000000000000000000000000000000000000000000000111000
0000000000000000000000000000000000000000000000000000000000001000
0000000000000000000000000000000000000000000000000000000000100011
0000000000000000000000000000000000000000000000000000000000000101
0000000000000000000000000000000000000000000000000000000000001110
0000000000000000000000000000000000000000000000000000000000001111
0000000000000000000000000000000000000000000000000000000000010000
0000000000000000000000000000000000000000000000000000000000010001
0000000000000000000000000000000000000000000000000000000000010010
0000000000000000000000000000000000000000000000000000000000010011
0000000000000000000000000000000000000000000000000000000000010110
0000000000000000000000000000000000000000000000000000000000010111
0000000000000000000000000000000000000000000000000000000000011000
0000000000000000000000000000000000000000000000000000000000011001
0000000000000000000000000000000000000000000000000000000000011010
0000000000000000000000000000000000000000000000000000000000011001
0000000000000000000000000000000000000000000000000000000000011010
0000000000000000000000000000000000000000000000000000000000011011
0000000000000000000000000000000000000000000000000000000000011100
0000000000000000000000000000000000000000000000000000000000011101
0000000000000000000000000000000000000000000000000000000000011110
0000000000000000000000000000000000000000000000000000000000011111
```

2 Code

Listing 1: Verilog code for implementing the data_memory module.

```
`include "definitions.vh"

module data_memory#(
parameter SIZE=1024)(
    input clk, mem_write, mem_read,
    input [`WORD - 1:0] address, write_data,
    output reg [`WORD - 1:0] read_data
    );

    reg[`WORD - 1:0] dmem [SIZE-1:0];

        //handle output
         always @(posedge(clk))
            if(mem_read==1) begin
                read_data <= dmem[address];
            end

        always @(posedge(clk))
            if(mem_write==1) begin
                dmem[address] <= write_data;
            end

        //initialize memory from file
        initial
            $readmemb(`DMEMFILE, dmem);

endmodule
```

Listing 2: Verilog code for implementing the iMemory module.

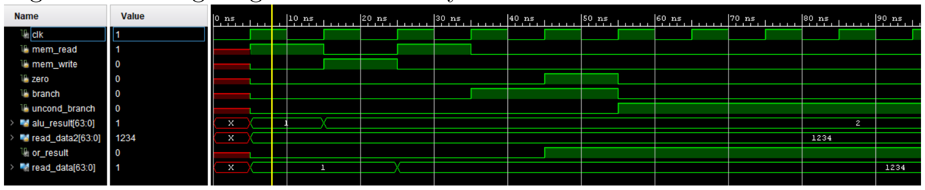
```
`include "definitions.vh"

module iMemory(
    input mem_read, mem_write, zero, branch, uncond_branch, clk,
    input [`WORD-1:0] alu_result, read_data2,
    output or_result,
    output [`WORD-1:0] read_data
    );

    wire and_result;

    assign and_result = branch & zero;
    assign or_result = uncond_branch | and_result;

    data_memory dataMemory(
        .clk(clk),
        .mem_write(mem_write),
        .mem_read(mem_read),
        .address(alu_result),
        .write_data(read_data2),
        .read_data(read_data)
        );

endmodule
```

Listing 3: Verilog code for implementing the iMemory Testbench.

```
`include "definitions.vh"

module iMemory_test;

    reg mem_read, mem_write, zero, branch, uncond_branch;
    reg [`WORD-1:0] alu_result, read_data2;
    wire or_result, clk;
    wire [`WORD-1:0] read_data;

    oscillator clk_gen(clk);

    iMemory UUT(
        .mem_read(mem_read),
        .mem_write(mem_write),
        .zero(zero),
        .branch(branch),
        .uncond_branch(uncond_branch),
        .clk(clk),
        .alu_result(alu_result),
        .read_data2(read_data2),
        .or_result(or_result),
        .read_data(read_data)
        );

    initial begin
        #(`CYCLE/2)
        alu_result <= 1;
        read_data2 <= 1234;
        mem_read <= 1;
        mem_write <= 0;
        zero <= 0;
        branch <= 0;
        uncond_branch <= 0;
        #(`CYCLE)
        alu_result <= 2;
        mem_read <= 0;
        mem_write <= 1;
        #(`CYCLE)
        mem_read <= 1;
        mem_write <= 0;
        #(`CYCLE)
        mem_read <= 0;
        branch <= 1;
        #(`CYCLE)
        zero <= 1;
        #(`CYCLE)
        branch <= 0;
        zero <= 0;
        uncond_branch <= 1;

    end


endmodule
```

3 Simulation

Figure 2: Timing diagram for iMemory module test. All values are in decimal.



Name	Value
clk	1
mem_read	1
mem_write	0
zero	0
branch	0
uncond_branch	0
alu_result[63:0]	1
read_data2[63:0]	1234
or_result	0
read_data[63:0]	1