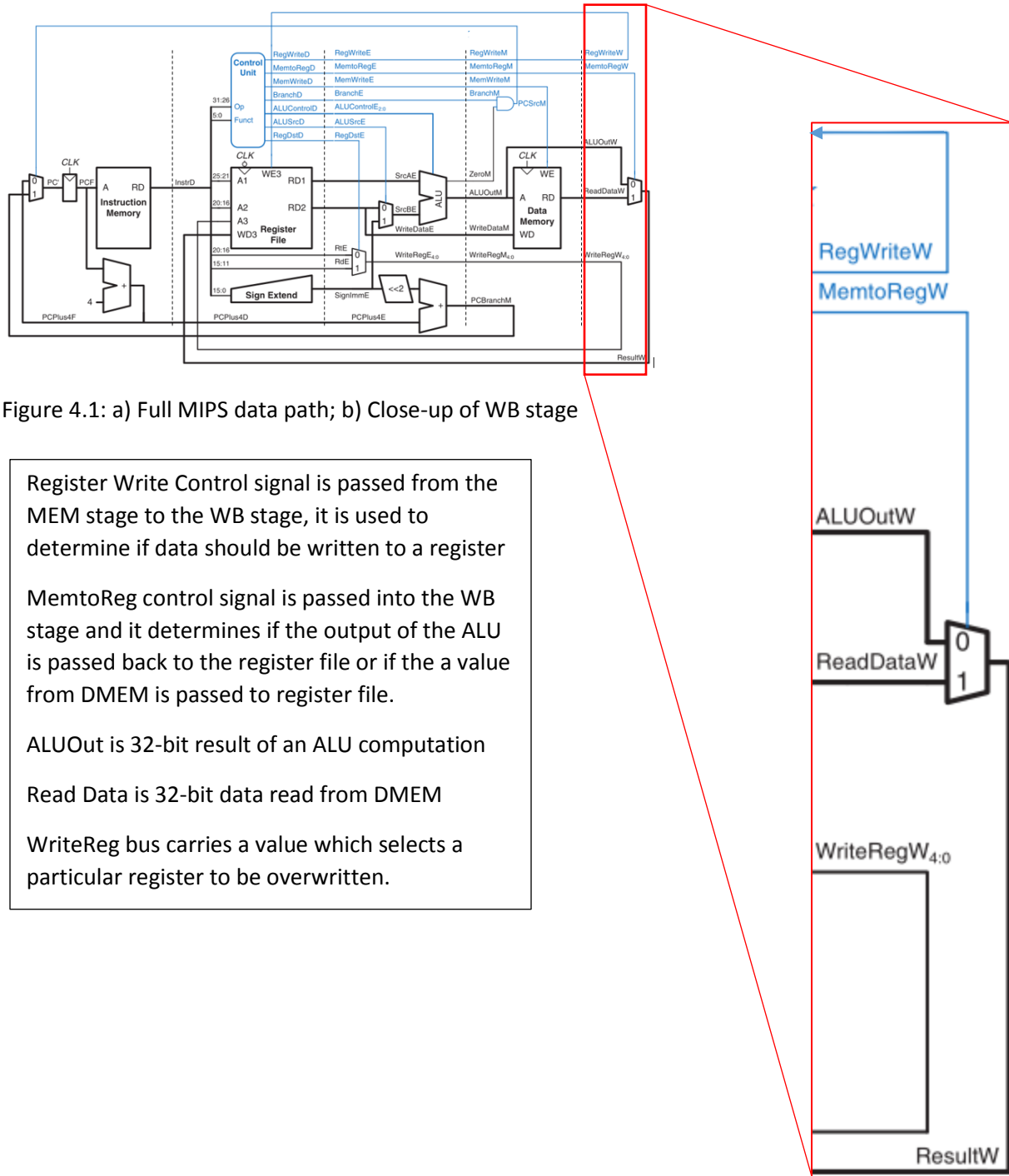


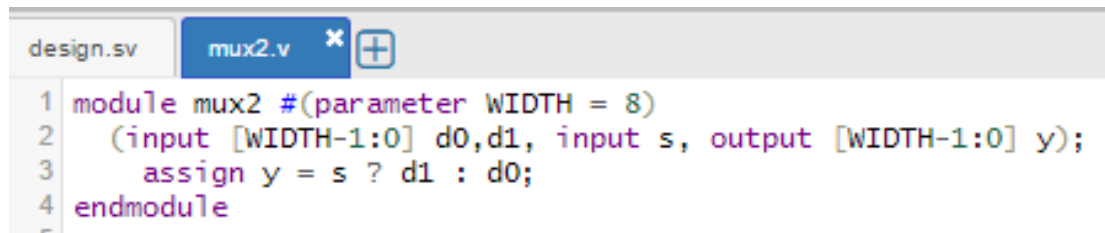
LAB 4

MIPS Write Back

In the single cycle MIPS processor, there are 5 stages of execution. The simplest stage is the Write Back stage (WB) which involves a single mux and a bunch of buses:



A parameterized mux is provided for use in this and other muxes, as shown in the Figure 4.2.



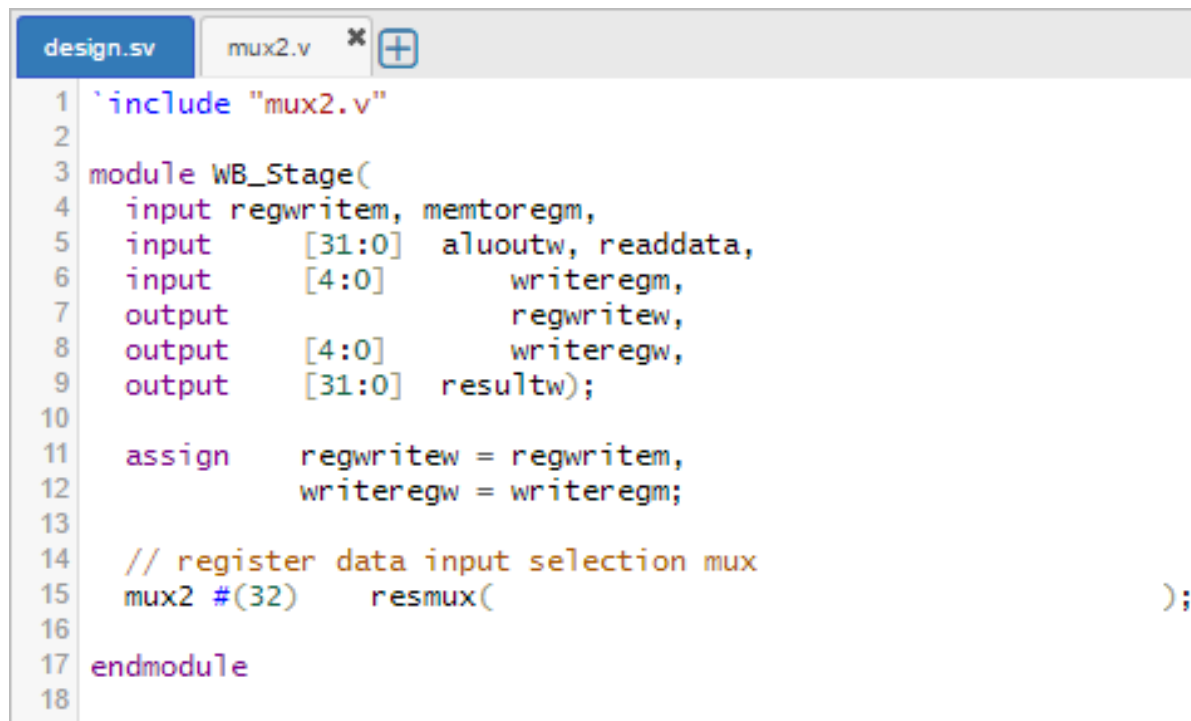
```

1 module mux2 #(parameter WIDTH = 8)
2   (input [WIDTH-1:0] d0,d1, input s, output [WIDTH-1:0] y);
3   assign y = s ? d1 : d0;
4 endmodule

```

Figure 4.2: Code for 2-to-1 Mux

Complete the module skeleton in Figure 4.3 to create the Write Back stage.



```

1 `include "mux2.v"
2
3 module WB_Stage(
4   input regwritem, memtoregm,
5   input [31:0] aluoutw, readdata,
6   input [4:0] writeregm,
7   output regwritew,
8   output [4:0] writeregw,
9   output [31:0] resultw);
10
11   assign regwritew = regwritem,
12          writeregw = writeregm;
13
14   // register data input selection mux
15   mux2 #(32) resmux(
16
17 endmodule
18

```

Figure 4.3:Code for Write Back stage

Finally test your design to ensure it works correctly using the Verilog test fixture code in Figure 4.4.

Correct test results are shown in the Figure 4.5.

WHAT TO SUBMIT

Once you have verified proper functionality of your project, copy the contents of **WB_Stage module (design.sv)** to a text file named **WB_Stage.txt** and upload the BeachBoard Dropbox for Lab 4. Additionally, upload your Lab report (see the LabReportTemplate in the Documents folder on BeachBoard).

NOTE: keep these lab files as they will be needed for future labs!

Created by Josh Hayter, updated by Jelena Trajkovic

```

1 module t_WB_Stage();
2   reg regwritem, memtoregm;
3   reg [31:0] aluoutw, readdata; reg [4:0] writeregm;
4   wire regwritew;      wire [4:0] writeregw;
5   wire [31:0] resultw; integer i;
6
7   WB_Stage dut( regwritem, memtoregm, aluoutw, readdata,
8                 writeregm, regwritew, writeregw, resultw);
9   initial begin
10     for(i = 0; i < 5; i = i + 1) begin
11       aluoutw = $random; readdata = $random;
12       {writeregm,regwritem} = $random; memtoregm = i;
13       #1 testcase();
14     end
15     #1 $finish;
16   end
17   task testcase(); begin
18     $display("Test Case %d",i);
19     $display("aluoutw = %h\treaddata = %h",aluoutw,readdata);
20     $display("memtoregm = %b\tresultw = %h",memtoregm,resultw);
21     if((!memtoregm && resultw != aluoutw) ||
22        (memtoregm && resultw != readdata)) begin
23       $display("TEST FAILED: resmux has malfunctioned"); $finish; end
24     $display("regwritem = %b\tregwritew = %b",regwritem,regwritew);
25     $display("writeregm = %h\twriteregw = %h",writeregm,writeregw);
26     if((regwritem!=regwritew)|| (writeregm!=writeregw)) begin
27       $display("TEST FAILED: signal passing malfunctioned"); $finish; end
28     $display("");
29   end endtask
30 endmodule
31

```

Figure 4.4: Testbench for WB stage.

```
Test Case          0
aluoutw = 12153524      readdata = c0895e81
mentoregm = 0    resultw = 12153524
regwritem = 1    regwritew = 1
writeregm = 04   writeregw = 04

Test Case          1
aluoutw = b1f05663      readdata = 06b97b0d
mentoregm = 1    resultw = 06b97b0d
regwritem = 1    regwritew = 1
writeregm = 06   writeregw = 06

Test Case          2
aluoutw = b2c28465      readdata = 89375212
mentoregm = 0    resultw = b2c28465
regwritem = 1    regwritew = 1
writeregm = 00   writeregw = 00

Test Case          3
aluoutw = 06d7cd0d      readdata = 3b23f176
mentoregm = 1    resultw = 3b23f176
regwritem = 1    regwritew = 1
writeregm = 1e   writeregw = 1e

Test Case          4
aluoutw = 76d457ed      readdata = 462df78c
mentoregm = 0    resultw = 76d457ed
regwritem = 1    regwritew = 1
writeregm = 1c   writeregw = 1c
```

Figure .4.5: Expected output