LAB 5
MIPS Memory Access Stage

In the single cycle MIPS processor, there are 5 stages of execution. The Memory access stage is where the data memory can be written to for a store word instruction or read from for a load word instruction. Also there is logic to see if a branch will be taken:

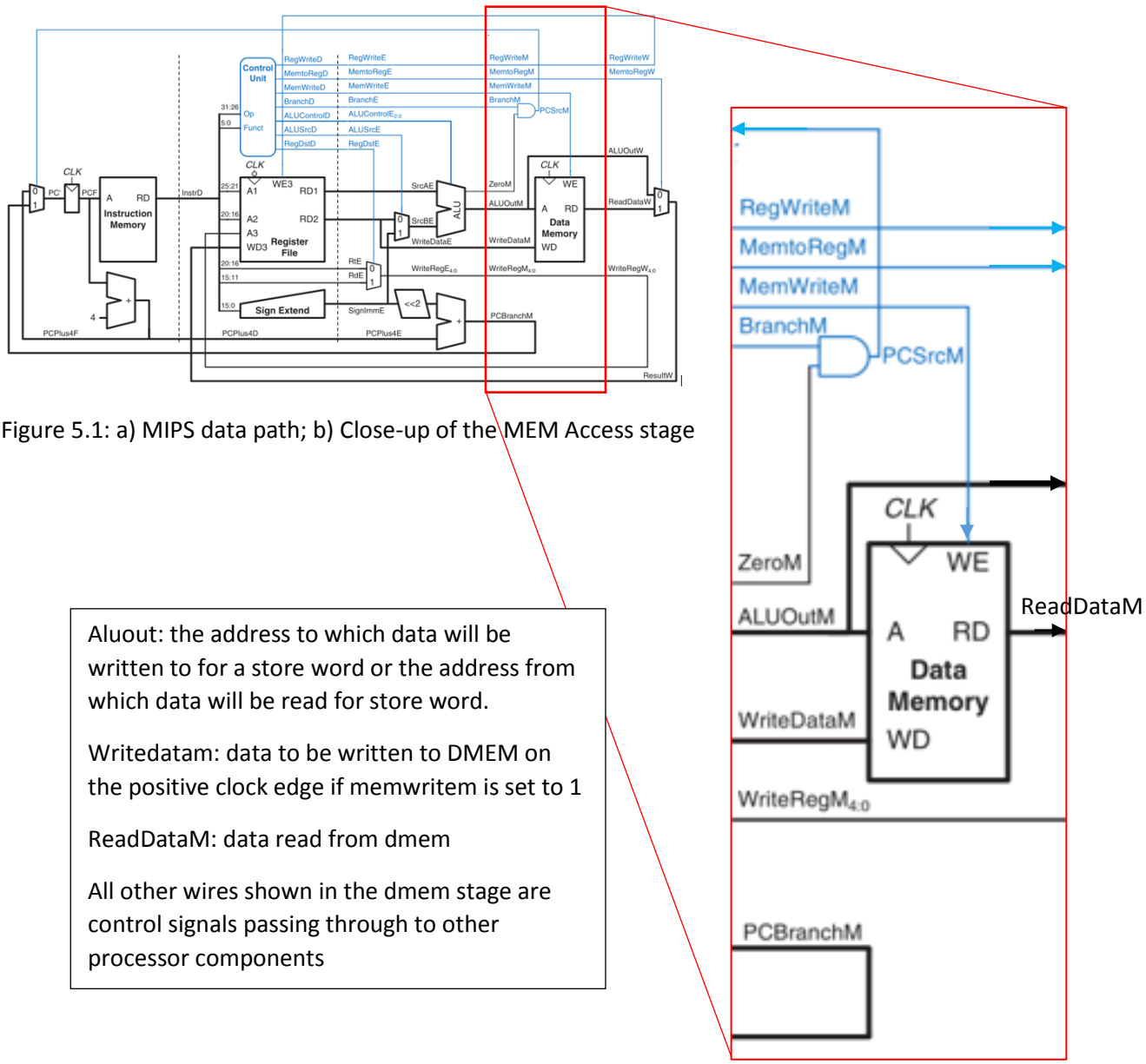

Figure 5.1: a) MIPS data path; b) Close-up of the MEM Access stage

Aluout: the address to which data will be written to for a store word or the address from which data will be read for store word.

Writedatam: data to be written to DMEM on the positive clock edge if memwritem is set to 1

ReadDataM: data read from dmem

All other wires shown in the dmem stage are control signals passing through to other processor components

Create Data Memory in a file named **dmem.v** and name the module **dmem**, as per Figure 5.2.

design.sv dmem.v

```
module dmem(clk, we,a,wd,rd);
  input clk, we;
  input [31:0] a, wd;
  output [31:0] rd;
  reg [31:0] RAM[63:0];
  assign rd=RAM[a[31:2]];
  always @(posedge clk)
    if (we) RAM[a[31:2]] <= wd;
endmodule
```

Figure 5.2: Code for dmem module

Use the block diagram on page 1 and complete the module skeleton given in Figure 5.3 to create the mem stage:

design.sv dmem.v

```
`include "dmem.v"
module MEM_Stage(
                        clk,
                        regwritee,
                        memtorege,
                        memwritem,
                        branchm,
                        zerom,
                        aluoute,
                        writedatam,
                        writerege,
                        pcsrcm,
                        pcbranche,
                        regwritem,
                        memtoregm,
                        aluoutm,
                        readdatam,
                        writeregm,
                        pcbranchm
                        );
  input clk, regwritee, memtorege, memwritem, branchm, zerom;
  input [31:0] aluoute, pcbranche, writedatam;
  input [4:0]    writerege;
  output pcsrcm, regwritem, memtoregm;
  output [31:0] aluoutm, readdatam, pcbranchm;
  output [4:0] writeregm;

  assign    regwritem = regwritee, memtoregm = memtorege,
            pcsrcm =    branchm & zerom, aluoutm = aluoute,
            writeregm = writerege, pcbranchm = pcbranche;

  dmem dmem(                                                  );
endmodule
```

Figure 5.3: Skeleton of the code for MEM stage module

Test your design to ensure it works correctly using the Verilog test fixture code in Figure 5.4. Correct test results are shown on the Figure 5.5.

testbench.sv

```
module t_MEM_Stage();
  reg   clk, regwritee, memtorege, memwritem, branchm, zerom;
  reg   [31:0] aluoute, pcbranche, writedatam;
  reg   [4:0] writerege;
  wire  pcsrcm, regwritem, memtoregm;
  wire  [31:0] aluoutm, readdatam, pcbranchm;
  wire  [4:0] writeregm;    integer i;

  MEM_Stage dut(clk, regwritee, memtorege, memwritem, branchm, zerom,
                aluoute, writedatam, writerege, pcsrcm, pcbranche,
                regwritem, memtoregm, aluoutm, readdatam, writeregm,
                pcbranchm);
  always #5 clk = ~clk; //clock pulse generation
  initial begin clk = 1;// $dumpfile("dump.vcd"); $dumpvars(0,dut);
    @(negedge clk)  rndctrlsig(); showmem(); checksignals();
    memwritem = 1; aluoute = 32'h0; writedatam = 32'h01010101;
    @(negedge clk)  rndctrlsig(); showmem(); checksignals();
    memwritem = 1; aluoute = 32'h4; writedatam = 32'h12121212;
    @(negedge clk)  rndctrlsig(); showmem(); checksignals();
    memwritem = 1; aluoute = 32'h8; writedatam = 32'h23232323;
    #10 showmem(); $finish;
  end
  task checksignals; begin  @(posedge clk) #1
    if(regwritem != regwritee || memtoregm != memtorege ||
      pcsrcm != (branchm & zerom) || aluoutm != aluoute ||
      writeregm != writerege || pcbranchm != pcbranche)
      begin $display("Test Failed: Control Signals faulty"); $finish; end
    else begin $display("Control signals OK"); end
  end endtask
  task rndctrlsig;
    {regwritee, memtorege, branchm, zerom, writerege, pcbranche} = $random;
  endtask
  task showmem; begin @(posedge clk) #1
    for(i = 0; i < 4; i = i + 1)
      $display("dmem[%h] = %h",i,dut.dmem.RAM[i]);
  end endtask
endmodule
```

Figure 5.4:Testbench for the MEM_Stage module

WHAT TO SUBMIT

Once you have verified proper functionality of your project, copy the contents of **MEB_Stage module (design.sv)** to a text file named **MEM_Stage.txt** and upload the BeachBoard Dropbox for Lab 5. Additionally, upload your Lab report (see the LabReportTemplete 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

Log Share

```
[2017-03-20 10:41:41 EDT] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
dmem[00000000] = xxxxxxxx
dmem[00000001] = xxxxxxxx
dmem[00000002] = xxxxxxxx
dmem[00000003] = xxxxxxxx
Control signals OK
dmem[00000000] = 01010101
dmem[00000001] = xxxxxxxx
dmem[00000002] = xxxxxxxx
dmem[00000003] = xxxxxxxx
Control signals OK
dmem[00000000] = 01010101
dmem[00000001] = 12121212
dmem[00000002] = xxxxxxxx
dmem[00000003] = xxxxxxxx
Control signals OK
dmem[00000000] = 01010101
dmem[00000001] = 12121212
dmem[00000002] = 23232323
dmem[00000003] = xxxxxxxx
Done
```