CECS 341 – LAB 5

MIPS Memory Access Stage

06 April 2020

Rifa Safeer Shah

017138353

I certify that this submission is my original work

Lab Report: Lab Assignment 5 – MIPS Memory Access Stage

1. Goal: The goal of the MIPS Memory Access Stage Lab Assignment is to use Structural Verilog to complete the memory access stage where the data memory can be written to for a store word instruction or read from for a load word instruction in the MIPS processor.

2. Steps:
    a. Go to edaplayground.com and log in to your account.
    b. Create dmem.v file with the data memory.
    c. Rewrite the outline provided with this lab.
    d. Identify what is to be done to produce specified output.
    e. Check for errors
    f. Run the code for test cases.

3. Results: In this lab assignment we create the memory stage in the MIPS processor. The program was correctly executed, and the desired results were achieved.

```
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
```

4. Conclusion: In the conclusion of this lab, I learned about the different stages of execution in the MIPS processor. We focused on the MIPS Memory Access Stage. I learned the process that MIPS processor goes through in this stage. The challenging part was to figure out how dmem.v file works and understanding the figure 5.2. This stage is a connection between load word instruction and store word instruction.

5. Notes: ALUOut is the address to which data will be written to for a store word or the address from which data will read for store word. Writedatam is the data to be written to DMEM on the positive clock edge if memwritem is set to 1. ReadDataM is data read from dmem. All other wires shown in the dmem stage are control signals passing through to other processor components.

6. Lab Screenshots:

testbench.sv

```
// Code your testbench here
// or browse Examples
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;
  initial begin clk = 1;
    @(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
```

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(clk,memwritem,aluoutm,writedatam,readdatam);
endmodule
```

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
```