**Lab 3: MEMORY - Instruction Memory, Register File, and Data Memory**COEN 122L - Computer Architecture  
Contact: Gregory Bokides Fay (gfay@scu.edu)   
TA Office Hours: By appointment   
Note: Email any question prior to class for

**Information**

The final pipeline will require three main memory components. The instruction memory will hold the binary instructions that the pipeline will execute. The program counter (PC) is used to index the Instruction Memory that should be executed. We are not creating the PC right now, therefore it is up to your discretion to prove that Instruction Memory will work (hardcode values into the IM and PC). I recommend using the testbench to hardcode the PC aspect into it. When the project comes around, the instruction set will be hardcoded into it. The register file holds the register memory for the pipeline. It contains 64 available register entries each with a size of 32 bits. Memory is the final memory component, used to load/store data into deeper memory. It has 32 bit size entries, and the number of entries is up to your discretion. [65536:0] is plenty for our design, no idea where Nate got that number. Seems chill ¯\\_(ツ)\_/¯

**Assignment**

Part 1) Create modules for the: 1) Instruction Memory, 2) Register, and 3) Data Memory. These will be used in your project. Part 2) test them and present your results. The modules should be tested concurrently in one testbench with one clock. Use the ‘posedge’ of the clock cycle to initiate changes within each module. Test each component individually but concurrently. For example, you should change the inputs for all three components, pause for 1 clock cycle, change all inputs again, pause for 1 clock cycle, etc, until you’ve tested each case. Warning, it’s easy to over think this lab. There are a lot of components that come in to play that are out of our scope. For us, try to focus on the inputs, black box, and outputs of each component individually. In the project we will focus on tying them all together.

**Assignment Overview**  
There are 2 parts to this assignment.   
1) Module creation (create three components)

* Instruction Memory
* Register File
* Data Memory

2) Testing (create 1 testbench module that tests all three on 1 waveform)

* Prove your work is correct.
* Practice your presentation skills using the GUI
  + Change Radix (left click on each input)
  + Change color by left clicking also. What is a good color scheme?
  + Other GUI interfaces - up to your discretion

-----------------------------------------------------------------------------------------------------------------------------------

**Part 1: Module Creation**

-----------------------------------------------------------------------------------------------------------------------------------

**1) Instruction Memory** (possible in < 15 lines)

Inputs

* PC\_address – 32 bits (this is used to index which instruction should be outputted)
* Clock – 1 bit
* \_\_\_\_\_\_ optional input. Since this isn’t a complete project, you must decide some way to hardcode in the PC and loaded Instruction Memory set. Don’t worry about this aspect too much; it will make sense once we create the PC and load in our Instruction Set into the Instruction Memory.

Black box (research / reference class notes)

Questions the designer (you!) get to answer: What is the length of each entry? (32 bits) How many entries are there in total? (your discretion. 255 is good). We have not created the PC yet, therefore assume the PC is inputted correctly and index your local register appropriately. You will hardcode your local register / testbench to put in the right values. I recommend researching Instruction memory online, and examining a complete Data path image. If you understand how a full datapath works, you will do well in this class.

Outputs

* Instruction – 32 bits (will go to control, register, and sign extend in our project)

-----------------------------------------------------------------------------------------------------------------------------------

**2) Register** (possible in 16 lines)

Inputs

* RegWrite – 1 bit
* 3 Data Addresses – 6 bits each (rs, rt, rd)
* Clock – 1 bit
* Data in – 32 bits (comes from data memory, used when write flag = 1)

Black box

Use “*reg [31:0] block [63:0];*” to create memory like a classic register.

\*hint use: always *@(posedge clk)* begin…

Outputs

* rs – reg 32 bits (make a register testing purposes)
* rt - reg 32 bits (make a register testing purposes)

-----------------------------------------------------------------------------------------------------------------------------------

**3) Memory** (possible in 13 lines)

Inputs

* Clock – 1 bit
* MemWrite – 1 bit
* Mem\_read – 1 bit
* Data address – 32 bits
* Data input – 32 bits

Black box

Use: *reg [31:0] block [XXX:0];* to create memory with XXX entries of 32 bit size.

\*hint: The write flag has big consequences for what this component should do.

Outputs

* Data read – 32 bits (a.k.a. the data our inputs requested, goes to register)

-----------------------------------------------------------------------------------------------------------------------------------

**Part 2: Testing**

-----------------------------------------------------------------------------------------------------------------------------------

Overview

We will test all components in one testbench module. The ‘posedge’ of the clock will be used to trigger each reaction. The waveform output should prove the correctness of each three modules in a **clear** presentable manner.

**Hints**:

* Initialize all 3 at the start as we did with the ALU
* Initialize your clock. (initial begin clk = 0; forever #5 clk = ~clk; end)
* Initial begin your main testing
* Change the inputs for IM, then Register, then Memory
* Pause for 1 clock cycle then continue changing inputs as necessary

**Questions**:

* Did you account for appropriate edge cases?
* Was each flag triggered?
* Does the code do what it’s supposed to?
* How is the presentation of the waveform?
* Would changing the display of numbers to unsigned be better? (left click 🡪 Radix)
* Would color coding make it better? (left click 🡪 color)
* Is your work clear? Would a brief readme.txt be helpful?

**Grading:**

80% - Correctness 20% - Presentation

To turn in:

* Verilog Code (80%)
  + Code containing modules for: instruction memory, register, memory, and a testbench
* Proof of Work (20%)
  + Waveform screenshot (with name in top left corner)
  + Readme.txt - optional