University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Fall 1997 D.A. Patterson
Handout #5

Assignment 4
Lab e-mail due to TA Tuesday 9/30/97.
Exercises due Friday 10/3/97 at 4PM in the boxes in 283 Soda.
Design review in discussion section Thursday 10/2/97.
Lab due Thursday 10/9/97 in discussion section.
CS152 Computer Architecture and Engineering

PLEASE put the DISCUSSION SECTION NUMBER (or time) that you attend as well as your name, as homeworks will be handed out in discussion. No number, no homework.

Homework Policy: Assignments are due at 4PM on Fridays. No late homeworks will be accepted, as we want to be able to discuss the assignments in section. Study groups are encouraged, but what you turn in must be your own work. As we agreed on the first day of class, the penalty for cheating is no credit for the full assignment. (If things are tough, it's better to skip pieces of one assignment rather than take a chance on getting no credit.)

Exercises: Every student should turn his/her own solutions to the following problems by Friday 10/3 from P&H: **5.1, 5.2, 5.12, and 5.13**.

Lab: In this assignment you will build a single-cycle datapath like the one discussed in class and in chapter 5, and verify that it executes a subset of the MIPS instruction set. The components that you need for the datapath, such as MUXes, registers, ALUs, etc. are provided. Look under directory `~cs152/fa97/hw4` for the components. You will have to add this directory to your search path by editing your `viewdraw.ini` file. (Add the line `DIR [r] /home/mm/cs152/fa97/hw4` to the end of your `viewdraw.ini` file.) You will also learn to write a VHDL description of the control logic required to run your datapath.

This lab assignment is to be completed *with* your project partners. To help you build a successful machine, we will give you intermediate milestones for this assignment. By Tuesday 9/30 you must e-mail your TA the names of your team members, your spokesperson, the assignments for each team member, and copies of your on-line log thus far. During discussion section on Thursday 10/2, you will have a design review sitting in front of a machine with your TA(s). Depending on the size of the discussion section, these will last approximately 15 minutes. Time slots will be assigned in discussion section the week before. Be sure to email your TAs a coherent version of the on-line log the day before your meeting so that they can review your progress off-line.

Problem 0: VHDL

Work through the VHDL tutorial (available on the web page).

Problem 1: Datapath Design

The following components are provided for this assignment and future ones.

1. alu: arithmetic logical unit
2. ram2048x32: memory of capacity 2048 words

You will most likely design the following components while working on this assignment. We suggest breaking them into these sub-components.

1. `bts4x32`: 32-bit tri-state buffer
2. `extend`: sign/zero extender
3. `m16x21h`, `m32x21h`, `m32x31h`, `m32x51h`, `m5x21h`: MUXes with various widths and number of inputs.
4. `reg32`: 32-bit register
5. `regfile`: register file
6. `shifter`: arithmetic logical multibit shifting unit

All the components should be implemented in VHDL. You can treat the components as symbols similar to the gates we used in the previous assignment when building a schematic. We suggest that you adhere to standards when building your modules — it will make your life easier.

For instance, you could do the following:

1. In all the components the label *DOUT* refers to the data output, and if there is only one data input (e.g., `reg32`), it is called *DIN*.
2. If there is more than one data input, they are labeled *A*, *B*, *C*, ...
3. MUX output selectors are labeled *SEL0*, *SEL1*, ..., with *SEL0* representing the least significant bit of the selector. *A* is routed to the output when the select bits represent *0*, *B* when they represent *1*, and so on.
4. Control signals such as the write enable on `reg32` have a suffix *_H* or *_L* to indicate whether they are active high or active low.
5. All clock inputs to the modules are called *CLK*.

All modules should be falling-edge triggered. Please look through the comments (lines preceded by `--`) at the beginning of the corresponding VHDL description files in directory `~cs152/fa97/hw4/<comp>.vhd` for more information regarding the signals for `<comp>`.

Using the components listed above, and the components in the standard cell library, build a single-cycle datapath that implement the following subset of MIPS instructions: `add`, `addi`, `sub`, `ori`, `lw`, `lui`, `sll`, `srl`, `slt`, `sw`, `jr`, `beq`. Make sure that your design is clean — you will be breaking it apart to build a pipelined processor in future assignments. In this assignment you don't have to put in the PC and instruction memory. Just set the values of the 32-bit instruction and the control signals in `viewsim`, and test your datapath.

To test your datapath, follow these steps:

1. Write a simple diagnostic program to test each of the instructions. Put the source in a `.s` file (e.g. `foo.s`).
2. Run `mipsasm` to create a `.dat` file. (The executable `mipsasm` should be under `~cs152/fa97/hw4/mipsasm`). For example, `mipsasm < foo.s > foo.dat`.
3. Set the clock (stepsize).
4. Run `viewsim` with your datapath.
5. For each of the encoded instructions you have in `foo.dat`, manually set the value of the 32-bit instruction bus. Also set the control signals. Doing this is easier if you create a `viewsim` script file (file name `<something>.cmd`) and invoke the script file under `viewsim` (by typing `<something>`) rather than typing each command individually. **Turn in a copy of your diagnostic program and a log of your simulation showing that the instructions worked.** The objective of this question is to make sure that you have all the parts of the datapath in place to execute all the instructions and you have a clear idea of what the control signals have to be for each instruction. The testing need not be exhaustive, but should be complete.
6. To look at the state of the memory at any time, use the `dumpm` command. For example `dumpm imem\ram(0:255)` will display the contents of memory words 0 through 255. `Viewsim`'s `mem` module holds a maximum of 2048 x 32 bits. It takes a 11-bit word address as input.

Problem 2: VHDL Controller for the Datapath

Now we are ready to build the controller for the datapath you built in problem 1. The first step is to create truth tables or boolean equations for the datapath you have just created. You will use these to create a VHDL control module. Make this into a module with pins labeled appropriately and connect it to the datapath you built in problem 1. This controller should take the bits from the 32-bit instruction and implement a combinational circuit that generates the control signals. You should also add in the instruction memory, *PC* and *nPC*, and the next instruction logic here.

To load the instruction memory in `viewsim`, type `loadm imem\ram < foo.dat`. This will load your compiled program into `viewsim`'s memory assuming your instruction memory unit is labeled as imem.

You should write a diagnostic program to test this mini-MIPS processor. Your program should be written such that the result of the test should be stored in memory which you can peek at using the `dumpm` command under `viewsim`. In general, you will not have access to the contents of the registers like you did in `spim`. It is important to write good diagnostic programs to test your hardware because it will simplify your life later on when you have to debug a more complicated processor.

Another good debugging tool to have is a VHDL decoder module that reads the contents of the instruction register and prints out the name of the instruction and its operands in a readable format.

Turn in a copy of your diagnostic program(s), a simulation log that shows the correct execution of the program, and your on-line logs. How much time did your team spend on this assignment?