The UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Comp 541 Digital Logic and Computer Design
Prof. Montek Singh
Spring 2017

Lab #9: A Full Single-Cycle MIPS Processor

Issued Wed 3/29/17; Due Wed 4/5/17 (11:59pm)

You will learn the following in this lab:

- Integrating ALU, registers, etc., to form a full datapath
- Integrating the control unit with the datapath
- Integrating memory units with a processor
- Encoding instructions
- More practice with test fixtures for testing a processor

Part 0: Carefully review the single-cycle MIPS processor design of Lecture 12.

Study the lecture slides carefully, and review Chapter 7.1-7.3 of textbook. Also review the Patterson Hennessy front inside green flap for summary of the MIPS instruction set; a PDF version of this ("green card") is posted on the class website. Wherever there are differences between the Patterson Hennessy textbook and the Harris and Harris textbook, we will adhere to the former. We will do so because the MARS simulator, which you will use to assemble your MIPS assembly code follows the former.

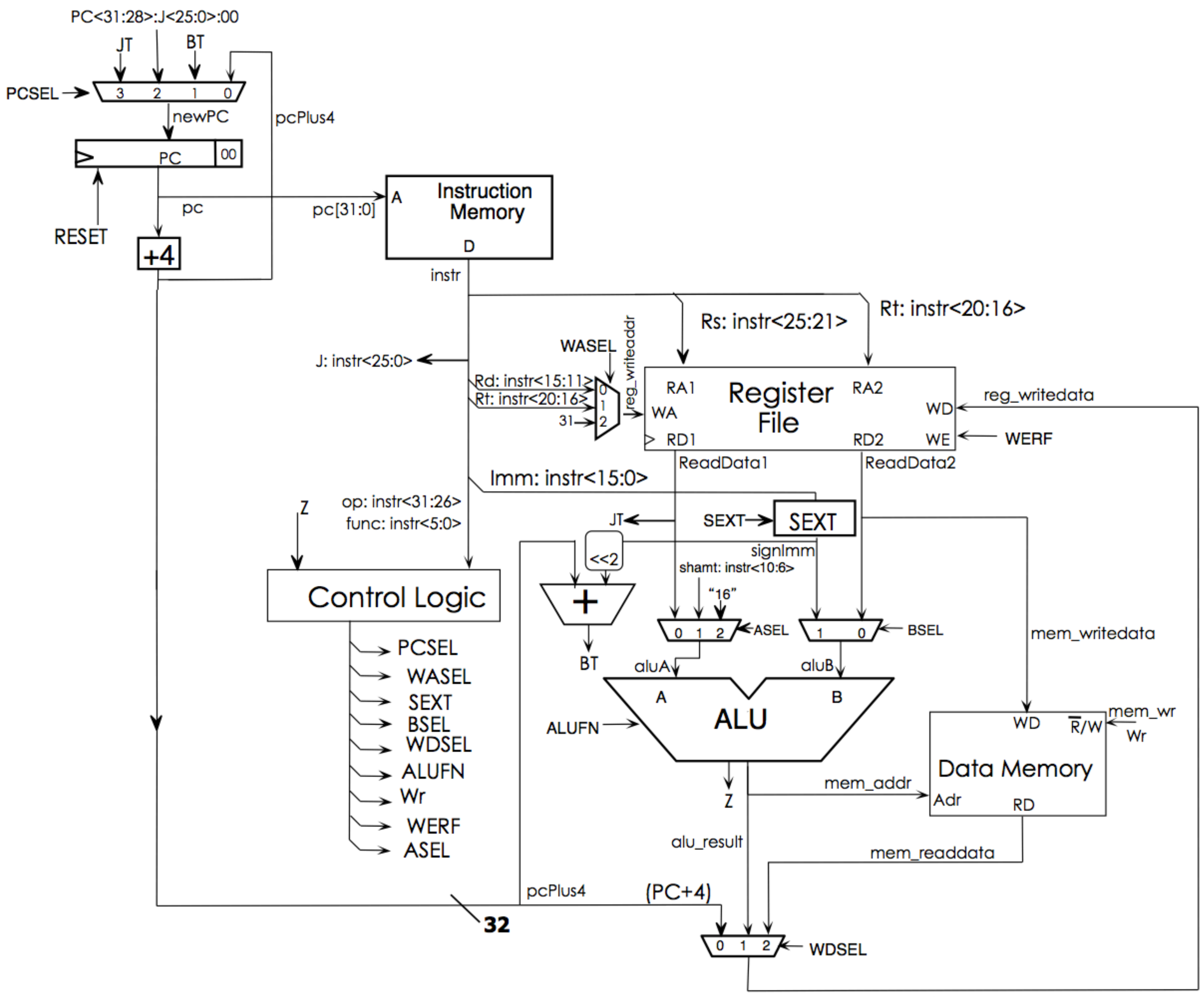
Part 1: Design a full single-cycle MIPS.

Put all the pieces together to create a full single-cycle MIPS CPU as discussed in class. Verilog templates for some of the higher-level modules are on the course website. In particular, do the following:

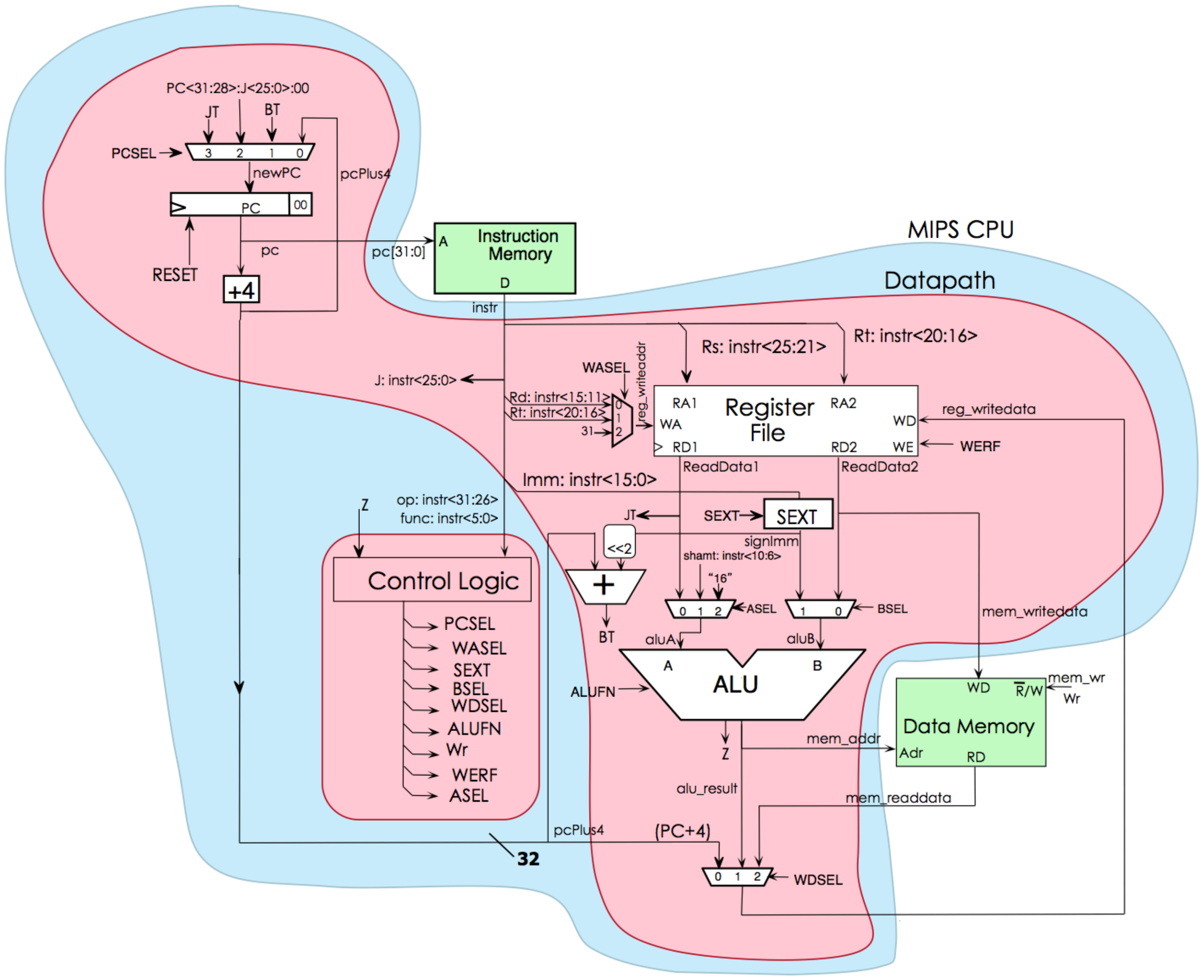
- **Design the top-level module, with instruction and data memories and the MIPS CPU,** as shown on Slide #4 of Lecture 12. Name this file `top.sv`. You may look at Slide #5 for guidance. For now, choose a reasonably small size for each memory, e.g. 64 memory locations. *Note:* The addresses generated by the CPU to access these memories will still be 32 bits long, even though fewer address bits will actually be used inside these memories. Use the template `ram.sv` from the previous lab, and make the following modifications:
  - o Send full 32-bit PC to instruction memory; strip the last two bits inside the instruction memory to derive a word address from the byte address.
  - o Similarly, send full 32 bits of address to data memory; strip the last two bits inside the data memory to derive a word address.
  - o Both of these memories should return a full 32-bit word (i.e., `Nloc` = 64, `Dbits` = 32, and `initfile` is the file name for initialization values). Your instruction memory module should be in its own file called `imem.sv`, and the data memory module in its own file called `dmem.sv`. Both should be directly based on the template `ram.sv` from Lab 8.
- **Initialize the instruction and data memories,** using the method explained in Lab 8. The file that contains the initial values for the instruction memory will contain a 32-bit assembly coded instruction

per line (either in hex or in binary). The file that contains the initial values for the data memory will contain some 32-bit data values, again one per line. Note: You can create these files either within Vivado, or outside of Vivado and then add these to the project. Make sure to set their type to "Memory File."

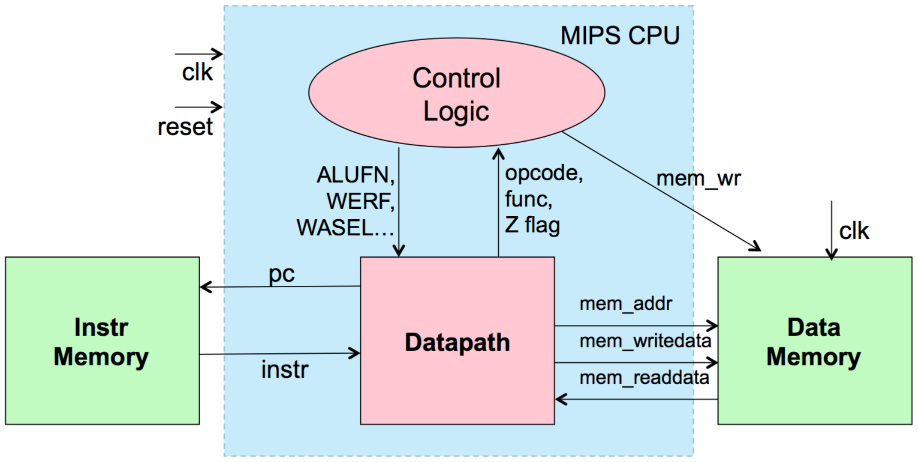
- **Design the top-level processor, containing controller and datapath modules,** using the figure below for guidance. *Note:* There are differences between the MIPS design in the Harris and Harris textbook and what we are designing in the lab. Our lab version has a much more sophisticated ALU. Therefore, do not blindly follow the information in the book; instead, follow the lecture slides and the lab writeups. The datapath should be 32-bit wide (i.e., registers, ALU, data memory and instruction memory, all use 32-bit words).
- **Complete all the little pieces,** so the design looks like that in Lecture 12, also reproduced below.



The following figure shows the top-level hierarchical decomposition. Your design is required to follow this hierarchy.



If you hide the implementation details, the simplified picture looks like this:



Small amounts of logic can be "inlined" (instead of written as a separate module), e.g., muxes, sign extension, adders, shift-by-2, etc.

- **Remember to use the directive `` `default_nettype none ``** to make it easier to catch missing declarations or name mismatches due to typos, etc.
- **Implement ALL of the instructions from Part 3 of Lab 8.**
- **Use test fixtures to test your CPU via simulation.** Two self-checking testers are provided on the class website. These were initially written in MIPS assembly, then compiled using MARS and converted to hex machine code, which should be used to initialize your instruction memory. Store the machine code into the file that is used to initialize the instruction memory. Be sure to initialize the program counter (PC) inside your MIPS design to zero, so that it starts executing from the beginning of the instruction memory. Similarly, to initialize your data memory, put the initial values in the corresponding file.
  - **sqr** (`tester_sqr.sv`)**:** For this tester, the program has 21 instructions, so the instruction memory should be at least that big (32 or 64 locations would be fine). Check the `Nloc` parameter for instruction memory in `top.sv`. To run this test, check that, in the `top` module, the names of the files that have initialization values for the instruction and data memories are "`sqr_imem.mem`" and "`sqr_dmem.mem`", respectively.
  - **full** (`tester_full.sv`)**:** For this tester, the program has 49 instructions, so let us make the instruction memory be 64 locations; set the `Nloc` parameter for instruction memory accordingly. Also, in the `top` module, modify the names of the files that have initialization values to "`full_imem.mem`" and "`full_dmem.mem`", respectively.
- **For now, you will not be implementing your design on the board.** You will do so next week.
- **Start thinking about what you would like to build for your final project!** Every project must use a VGA display as output. Nexys 4 boards have audio output built-in. Keyboard/mouse inputs will be available to every project. The Nexys 4 boards have accelerometers built-in; a limited number of other input devices (joysticks, keypads, etc.) may be available to use instead. Start thinking!

Okay, good luck!

What to submit:

- **ALL of the Verilog files for Part 1, but skip the ALU and its submodules.**
- **A couple of sentences within the email body stating whether your design worked, any errors/bugs you couldn't resolve, any special observations, etc.**
- **Two screenshots of the simulation waveforms for Part 1, one for each of the self-checking testers provided on the website.**

How to submit: **Please submit your work by email by 11:59pm, Apr 5 (Wed), as follows:**

- **Send email to: `comp541-submit-s17@cs.unc.edu`**
- **Use subject line: Lab 9**
- **Include all of the attachments, and your statement/observation, as specified above**
