Lab 9: Multicycle Processor (Part 2)

Multicycle Processor Completion

This lab needs a lot of effort, but when you are done, you will be glad you did it! In this lab, you will complete your own multicycle MIPS processor. Please refer to Part 1 in Lab 8 for an overview of the processor. By this point, you should be a competent digital designer, so few directions will be provided.

You can start from the design project in Part 1, so you don't have to copy the cunit. In the same project, create the iunit, munit and eunit, taking the inputs and outputs specified in Part 1. You can reuse hardware from earlier labs as well (such as the ALU, sgnext and register file) wherever possible.

Refer to Figure 1 in Part 1 of the lab or Figure 5.28 in the text to set up each of the units. For example in the iunit, you need to generate the register for the PC counter. You will need a 3-to-1 multiplexer to select one of these cases for the PC input: from the ALURESULT, from the ALUOUT or from part of the INSTRUCTION in the case of Jump command. In addition, the PC is enabled by a PCENABLE signal generated in the cunit.

In the munit, you need to have a RAM to store instructions and memory data. Both read and write are possible. As you can see, the address for the RAM can come either from the PC (if you are accessing an instruction) or from ALUOUT (if you are accessing data) and it is determined by the IorD (Instruction or Data) signal from the cunit. Once the content is fetched from the RAM, it is sent to one of two different registers for instruction and for memory data, respectively. Both RAM and registers can be set up properly using the CORE generator.

In the eunit, there are more functions to implement. The first item to set up is the sign extension module, which takes 16 bits and sign extends to 32 bits. The other key element is the register file that allows reading from two different addresses and, at the same time, writing data to another address. You can connect two sets of Dual-Port Block Memory to set up the register file. Assuming you set up one of the dual-port block memories as "doublereg," you can connect two of them for the eunit. Set up additional multiplexers to choose the addresses for read and write according to the diagram.

There are a few *very important* issues related to the dual-port memory block:

1. One trick to allow both writing and reading properly is to let the reading operation be done at the falling edge of the clock instead of the rising edge. This way, the reading address comes from the correct instruction, which is only valid until slightly after the clock rising edge. You can choose the active edge when you set up the register in CORE generator.

2. Secondly, using CORE generator, the names of the ports for the dual port memory block are lowercase. If your schematic names the relating input ports using uppercase, you need to manually change the names generated by CORE generator to match the connection. To do so, find the file *doublereg.v*, for example, which should be in the folder of your project, open it using either Notepad or ModelSim and change any port names from lowercase to uppercase.

3. The problem in (2) might also be present when you name everything in lowercase in the schematic, because sometimes CORE generator does name ports with uppercase, depending on which module it is working with. Since we haven't tested that aspect, keep the issue mind when you work with lowercase in your schematic.

You can copy additional function units, such as the ALU, from previous labs. Finally, when you have finished all four units, connect them together in a top-level schematic. The design should only take CLK and RESET as inputs. All of your flip-flops should take a RESET input into the Async Control terminal to reset the initial value to a known state. The Instruction Register and PC also require CLK_EN enable inputs. You should label the internal signals and busses so you can view them during simulation. Pay careful attention to bus connections; they are an easy place to make mistakes. For example, remember that the IorD multiplexer needs bits (5:2) of the PC or bits (5:2) of ALUOUT as the address, so don't try to connect the 32-bit bus directly to the 4-bit multiplexer inputs. Be sure to enter your test program (list of instructions, each 32 bits) from Part 1 in Lab 8 into the content of the munit RAM. Check the schematic before simulation.

Simulate your processor for 370 ns. Use a 10 ns CLOCK and a RESET signal that is high for the first 10 ns. Display, at a minimum, the PC, INSTRUCTION, FSM state (from within your cunit), SRCA and SRCB (from within

your eunit), ALURESULT and ZERO. You will likely want to add other signals to help debug. Check that your results match the expectations from Part 1. In particular, does the value of MemoryData at time 310 ns match what the program should produce? If there are any mismatches, debug your design.

Debugging

Hopefully, your lab will have at least one error so you will get to hone your debugging skills! Here are some hints:

- Be sure you thoroughly understand how the MIPS processor is supposed to work. This system is too complex to debug by trial and error. You should be able to predict what value every signal takes on at every point in time while debugging.
- In general, trace problems by finding the first point in a simulation where a signal has an incorrect value. Don't worry about later problems, because they could have been caused by the first error. Identify which circuit element is producing the bad output and add all its inputs to the simulation. Repeat until you have isolated the problem.
- If you use a symbol made from a schematic or CORE generator, make sure all the lower-level components are also included. Be sure all the components target the same device.
- Remember that input A of a multiplexer is chosen when the select signal is low and input B is chosen when the select signal is high. If you flip the multiplexer upside down for convenience, be sure to connect the data inputs to the proper ports.
- Many people have made errors with bus taps, such as not labeling a bus tap, connecting the wrong number of bits or mislabeling bits. Some of these errors can be checked using a schematic check, but some of them, if the bus widths are the same, will only affect your simulation results. So double check the bus connections to avoid headaches in debug.