

CSE 141 Lab #2: 8-bit CPU Internals

due Friday, Feb. 7

In this lab assignment, you will design the ALU (arithmetic logic unit) and the internal storage of your CPU, connect them together, and demonstrate that they work together through schematic and timing printouts. The internal storage means the register file, stack, or accumulator (depending on your instruction set architecture). If your ISA has none of these, or if your machine operates completely out of memory, you will have to rig up something to supply operands to the ALU to demonstrate it. An accumulator machine will also need to supply a second operand from somewhere.

The internal storage (from this point I'll just call it a register file, but substitute your own type) should be connected to the ALU in a manner like that shown in the accompanying sample schematic. In this picture, the ALU has two 8-bit inputs, an 8-bit output, and three control bits. This register file has two 8-bit outputs, one 8-bit input, and various control signals. The SRCA (Read Reg Address A) signals are the address of the first source register, SRCB the address of the second source register, and DEST (Reg Write Address) the address of the destination register. The multiplexor in this picture allows me to initialize the registers to non-zero values to demonstrate the operation of the circuit, since I'm not yet ready to do this by executing code. This description is only to help you understand the example, obviously your design will be different. In particular, your ALU operations are likely to be quite different.

You will demonstrate your design in two ways. First, with schematics such as the one shown. Obviously, you must also show all relevant internal circuits. Second, you must demonstrate correct operation of all ALU operations with a timing diagram. An example of a (partial) timing diagram is also included; yours will be longer. The timing diagram should demonstrate all ALU operations (this includes math to support load address computation or branch target computation, but not PC increment), each with a couple of interesting inputs. Make sure corner/unusual cases are demonstrated, if relevant. If you support instructions that do multiple computations at the same time, you may as well demonstrate them as a unit, to minimize changes for the next lab. Instructions that do no computation (e.g., branch to address in register) need not be demonstrated. The timing diagram should also demonstrate everything your internal storage is supposed to do, e.g., register writes as well as register read operation. In particular, you should demonstrate that you can use the same register as a source and destination operand of the same instruction (if appropriate). The schematics and timing diagrams will be difficult for us to understand without a great deal of annotation. Good organization and use of hierarchical design also help.

The lab report will contain the following:

- I. Introduction and general comments. Overview of report.
- II. Summarize your ISA from Lab 1 (operations supported, with full detailed descriptions).
- III. A listing of ALU operations you will be demonstrating, including the instructions they are relevant to.
- IV. Full schematics, hierarchically organized (starting with the highest-level circuit).
- V. Timing diagrams. It should be clear everything works. If your presentation leaves doubt, we'll assume it doesn't

VI. Answer the following questions:

1. How many (and what) ALU operations do you support?
2. Will your ALU be used for non-arithmetic instructions (e.g., address calculation, branches, ...)? If so, how does that complicate your design?
3. Are there instructions that do more than “one thing” in the ALU, or need multiple ALUs? If so, how does that complicate your design?
4. Are your ALU control signals the same as your opcode bits? If not, how much translation is needed? Could you have designed your ISA to eliminate (or reduce) that translation?
5. Now that your ALU is designed, are there any instructions that would be particularly straightforward to add given the hardware that is already there?
6. Is there anything you could have done in your ISA to make your ALU design job easier?
7. Is there anything you could have done in your ISA to make your register file (or ??) design job easier?
8. Give a qualitative description of your expected cycle time, as dictated by the ALU and register file design – relatively fast, average, slowwww... Explain. What could you do to improve cycle time, if it were a factor?

Hints for increasing the readability of your reports and making the design easier.

1. Assume the reader is somewhat ignorant (because, in the context of your project, they are).
2. Use subcircuits to create hierarchical designs. Be aware of the printable (single-page) area of each circuit when designing it.
3. Annotate heavily, as already mentioned.
4. Combine signals for numerical (eg, hexadecimal) values whenever possible in the timing window.
5. Don't be afraid to design your own circuits, even for low-level functionality. Often, using a library part that is *almost* right causes more headaches than designing your own from primitives.
6. Never place logic between the clock and the clock input of a device. This causes devices to be out of sync, and can also create phantom clock edges.
7. Remember that you are free to set the clock cycle to anything you want to increase readability of the timing diagrams, or just to get it to work.
8. Account for problems with Xilinx tools. Things will break or disappear when least convenient. Leave time for things to go wrong. Back up files frequently. Keep multiple backups.

Other information:

1. You are to use the visual logic entry interface for Xilinx. No VHDL, etc.
2. You can use any circuit you find in the XC4000E library. You are not allowed to use logiblox circuits except for creating memory devices in lab 3.
3. You are in general not allowed to change your ISA unless there is a correctness issue.
4. You are not allowed to change/merge/divide groups.