

CSE 141 Lab 2

8-bit CPU Internals

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. External storage (ie data or instruction memories) will not be handled in this stage of the lab. 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 main goal of this lab is to show most of your instruction types “works” independently.** Exceptions are that those instructions that access memory or deal with modifying the program counter (PC). You WILL want to test these to the best of your ability as they rely on ALU operations – but we won’t be testing use of external memory or the PC just yet. You don’t have to be able to execute a sequence of instructions.

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. (NOTE: In the final design, you will not be able to initialize your registers to non-zero values before the beginning of the program – you will be able to initialize them to zero).

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. It is highly recommended that you print schematics and then label them names and additional detail and explanation of what each part is and what it does. Obviously, you must also show all relevant internal circuits. You will likely need to print and annotate more than one schematic in order to clearly describe your implementation. Your design should be organized and hierarchical.

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 may be MUCH 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.

All designs will use schematic entry with the Xilinx tools (no vhdl/verilog entry except for multipliers). You may use any of the devices/symbols available for the Spartan 2 device xc2s30-5tq144.

The lab report will contain the following:

1. Introduction and general comments. Overview of report.
2. Summarize your ISA from Lab 1 (operations supported, with full detailed descriptions – at least as detailed as Lab1).
3. A listing of ALU operations you will be demonstrating, including the instructions they are relevant to or used with.
4. Full schematics, hierarchically organized. Each schematic needs a name, a general description, and annotations that explain what each part does and how it is used.
5. Timing diagrams. It should be clear everything works. If your presentation leaves doubt, we'll assume it doesn't. These need to be annotated **COPIOUSLY** as well. Please help us save our eyesight by writing in all values and providing explanations of what each part of the diagram is testing, as demonstrated in the example timing diagram. **Again, each supported instruction must be tested at least once (usually more with some “interesting” inputs.** Only those instructions which do NO computation or only increment the PC need not be tested. The more you test now, the happier you will be later.

Answer the following questions (you can actually number these in your report in a section called “Design Questions”):

6. How many (and what) ALU operations do you support?
7. Will your ALU be used for non-arithmetic instructions (e.g., address calculation, branches)? If so, how does that complicate your design?
8. Do any instructions need to use the ALU twice (or need multiple ALUs)? If so, how does that complicate your design and how did you choose between using multiple ALUs and having more than one ALU?
9. What is your most complex instruction, from the standpoint of the ALU?

10. Are your ALU control signals easily derived from your opcode bits? How much translation is needed? In hindsight, how could you have designed your ISA to eliminate (or at least reduce) that translation?
11. Now that your ALU is designed, are there any instructions that would be particularly straightforward to add given the hardware that is already there (assuming you had opcode instruction bits available)?
12. Is there anything you could have done in your ISA to make your ALU design job easier?
13. Do you have a single internal storage type or more than one “class” of internal storage? Is there anything you could have done in your ISA to make your register file (or whatever internal storage you use) design job easier?
14. Give a qualitative (that is not with numbers, but with reasoned English words) 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 design goal?
15. At this point, if you could start over and change one thing, what would it be?