Lab 4: Building an ALU

Worth 50 points (45 lab + 5 report)

Lab Objective:

You now know enough about combination and sequential logic to build the LC-3 ALU, or at least something like it. In this lab you will build the logic necessary to perform any of the LC-3 **operation** instructions (ADD, AND, and NOT), but only for 4-bit inputs and a 4-bit output. The purpose of this lab is to gain a better understanding of how a processor works. You are expected to practice your schematic skills by making your design clean and easy to understand by using multiple pages and nice routing of wires. This lab will build on lab #3 and you should start from that design and enhance it as described below.

Lab Description:

Start by making a "top" level page that has the interface to your ALU. This page should contain the input switches for setting the opcodes, LEDs to show which operation is being performed, a keypad as in lab #3 for writing data into your register file, clock push button, reset push button, two 7-segment displays for Latch A and Latch B, and control switches (see below). You should use the "Signal Sender" and "Signal Receiver" objects from the Palette to pass data across pages. This also helps make your schematic much neater. There is a sample ALU design (examples/devices/ALU.lgi) that you can play with, you will NOT be using the ALU tool for this lab; you will be making your own ALU!
To get an overview picture of what this lab does take a look at the Block Diagram.pdf file to see how things are connected and how data will "flow" in your circuit.

Control Signals:

Here is a description of the control signals for your design; these should all be on the top sheet. These are the signals that will control how data flows through your design:

- Clock – this button causes things to "move" from one storage element to another, only when this button is pressed should any memory get updated (that would be register file locations as well as Latch A and B). You need to set up all your control signals before pressing the clock.
- Reset – this button causes all the storage elements to output 0, this would be the 4x4 register file, Latch A, and latch B. You only need to do this once at the start.
- Address – Since the memory is 4 deep, we need a 2-bit address, use two switches for this.
- RW – This will control whether you are writing data into the register file at the location addressed (RW = 1) or if you are reading data from the register file into one of the 2 latches, which are the inputs to the ALU.
- In_sel – this switch will control the mux which selects what the write data will be for the register file, either from the keypad or from the ALU.
- LA_we – this switch enables writing to Latch A of the memory location being addressed

- LB_we – this switch enables writing to Latch B of the memory location being addressed
- opcode – this 2-bit value set from switches controls what function the ALU performs.
    - 00 – NOT: Take the value from Latch A and invert all the bits
    - 01 – AND: Perform the bit-wise AND of Latch A and Latch B
    - 10 – ADD: Do two's complement addition of Latch A and Latch B
    - 11 – PASS: Take the value from Latch A and pass it through, used to move values from one register location to another

The ALU:

Make one page for each of the functions "insides" of the ALU of the LC-3. Remember, you are only supporting 4-bit inputs and results. You need to build the logic necessary to perform each of the following operations:

- AND – Perform a bit-wise AND on the top inputs. If the inputs are A and B, then you would produce:
- Result[3] = A[3] AND B[3]
- Result[2] = A[2] AND B[2]
- And so forth
- ADD – have a carry-in input and a carry-out output also. You can connect the carry-in to GND and the carry-out will need to go to some overflow detect logic. Build a full-adder cell and "chain" together 4 of them to form the adder.
- NOT – simply invert the A input, you will ignore the B input.
- PASS – simply pass the value of A through

You should have another page that "decodes" the opcode and selects the correct result to be the final output of the ALU, this would basically be a 4-1 mux.
You should also have an overflow detection circuit. Put an LED on the top sheet that will light up if there is an overflow from the ALU. When does the overflow occur? For extra credit you will need to refer to your lecture notes to see how two's compliment addition works and when an arithmetic overflowing can happen.

Data Flow:

Test your circuit to make sure it works. If you want to come up with some inputs and outputs you can use the calculator on the Windows machines and put it in scientific mode so you can choose binary mode. You can now come up with inputs for the AND, ADD, and NOT and verify the outputs. ☺
Here are the steps on how you would store the value 0xF into the register file at location 1:

1. Reset all the memories (do this just once)
2. Press F on the keypad (you should connect up a 7-segment display to show what you pressed)
3. Set the Address switches to be 2'b01
4. Set the RW switch to be a 1 for write
5. Set In_sel switch to be a 1 to select the keypad as the write data
6. Press the clock button
7. Now the value 0xF is stored into the register file location [1]

To add memory[0] to memory[1] and store the result in memory[2] you would need to read the value out of memory[0] into a Latch (A or B) and then read memory[1] into the other Latch, this would take 2 clock presses. You would then set the opcode to be 2'b10 (ADD), the Address to be 2'b10, and In_sel to be 0 to select the ALU output as the write data. You would then press the clock button to store the result to the register file at address 2.

LC-3:

The setting of all the control lines in a real processor would be done by a finite state machine during the execution of an instruction. We will go into that in more detail when we look at the LC-3 architecture but doing this lab and testing your circuit should give you an idea of how the execution is done in steps, where each step would be a state in our state machine.
To make this be the "real" LC-3 ALU you would need to modify the inputs and output to support 16-bits. In the real LC-3 there would also be decodes for the operand bits of the instruction to specify the register data to use as the inputs. Also you would have to expand the possible 5-bit immediate as both the ADD and AND support immediate operands. You are not asked to do this now but you should understand how you would really build the data path of the LC-3 and what control lines it would require.

Collaboration: ***You are expected to work on this lab on your own though you may discuss at a high level how you plan on doing the lab to others in your section.***

Files to Submit:

- **Lab4.lgi**
- **Lab4_report.txt**

Please ask the tutor or see the class newsgroup for information on how to submit these files.

Report/Code comments Requirement:

- Please put minimal block comments on the top page of you schematic including your name, UCSC email address, section, date, and lab number. You should also have comments on all the other pages that includes a title and short description of the contents of the page.
- Please follow the regular write up rules.

Check-off: In general you are required to always demonstrate your lab when it is finished.

Grading template (total points possible: 50):

Sign off is worth at most 45 points:
A: 40-45 points. Have a completely functional circuit that is labeled well and laid out neatly using multiple sheets
B: 36-40 points. Have a completely function circuit that is messy, has minimal comments, and may not use enough sheets
C: 31-35 points. Circuit that mostly works with some functional issues or minor glitches, looks like everything is present
D: 27-30 points. Circuit looks to have had some effort put into it but is not complete or functional.

Lab write up is worth up to 5 points.

Extra credit

You have a simple overflow detection as part of the lab, a simple LED connected to the Carry Out of the MSB full adder. Since all numbers in the LC-3 are two's complement numbers add logic to detect when an arithmetic overflow occurs when adding two's complement numbers.

Happy Designing!!