CMPE 12L Lab 2 - Winter 2015
Arithmetic Logic Units

Prof. HP Dommel
Due: Monday February 2, 2015 at 5:00pm
50 Points (20 Report, 30 Check-off)

Administrative

For this assignment, you must work with a partner in your lab section. If you are having trouble finding a partner, ask on Piazza or your lab TA.

Objective

ALU stands for Arithmetic Logic Unit. It's the major part of hardware responsible for most of the computations (arithmetic and logic) a computer performs. In this lab, you will design and build a multi-page schematic for the operation of an 4-bit ALU capable of performing addition (ADD), negation (NOT), and bitwise AND. You do not need to know the details yet, but the LC-3 ALU operates on 16-bit numbers. The ALU you will design in this lab is only a 4-bit ALU, because a 16-bit ALU would be really messy to implement in MML.

Prerequisites

- Read through this **entire** lab assignment.
- Read the textbook chapter 2.
- Review the lecture notes on arithmetic and logical operations.

Tutor/TA Review

Your lab tutor/TA will cover the following items in the first portion of the first lab:

- Using the connector to make clean wire routing.
- Using the signal sender and signal receiver tools to do multiple pages.
- Use input/outputs in MML
- What's required

Requirements

You must submit a MML design called lab2_[username].lgi, as well as your partial work ALU_[username].lgi and MEMORY_[username].lgi and the associated lab write-up. Your design should be synchronous and well-organized. You must use pages.

Requirements for the ALU

The ALU you design must meet at least the following criteria. It must ...

- Have two 4-bit inputs using two keypads
- Display both keypad inputs using a 7-segment display for each input
- Capture the input in D flip-flops (labeled "SR1" and "SR2" for source register)
- Have a clock button (labeled "CLK")
- Have a global reset to reset all registers and displays (labeled "RST")
- Allow the user to input LC-3 opcodes using switches, and store the opcode in an instruction register (labeled "IR")
- Display the operation in progress with LEDs or other output device
- Be capable of performing a bitwise AND of the two operands
- Be capable of performing a bitwise NOT of the first operand
- Be capable of performing an ADD of the two operands using a ripple-carry full adder
- Capture the result of the operation with D flip-flops (labeled "DR")
- Display the result of the operation on a 7-segment display
- Display with an LED if the result of an ADD had a carry-out
- Display with an LED if the output of the ALU is zero (labeled "Z")
- Display with an LED if the output of the ALU is positive (labeled "P")
- Display with an LED if the output of the ALU is negative (labeled "N")
- Be able to recognize a bad opcode and light an LED representing bad opcode (labeled "BC"). **The LEDs should be turned off if the opcode is invalid and the output should display 0xBC.**

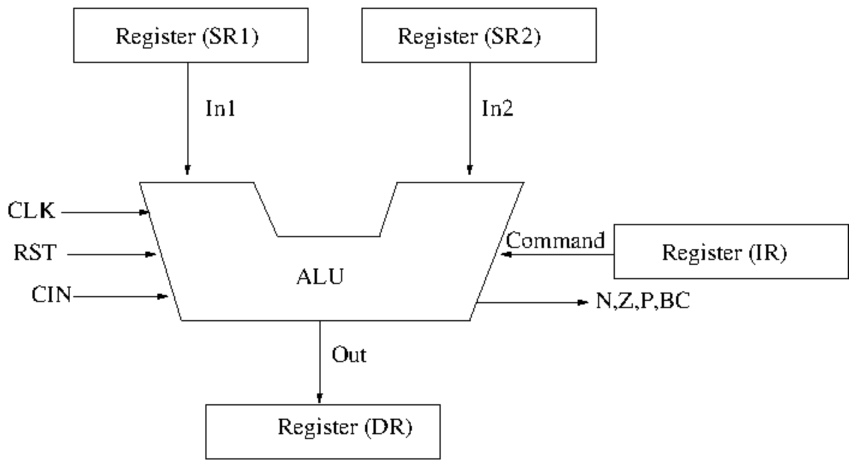

Figure 1: The ALU has two operands (In1 and In2 stored in SR1 and SR2 registers), a command input (stored in IR), and an output (out stored in DR). The ALU will need an additional clock signal CLK (like a "go" button), a global reset RST, and a carry-in CIN for the adder.

Organizing the design

You should organize your design into multiple pages. The following organization is a suggestion; yours may be different.

- Inputs and outputs
- Source registers implementation
- Opcode resolution – decode based on instruction opcode, and "bad code" logic
- Logic for the two bitwise functions AND and NOT
- Logic for the ADD instruction (could be spaced on two or more pages)
- Destination register multiplexor (a selector switch)
- Destination register implementation

Example: Designing the I/O Layer

Below is an example design of the I/O layer. Yours may be different.

I start at the top layer, the page (or sheet) of my design that will contain all the user-interactive pieces. The ALU will take two inputs and a set of commands, and will produce an output. As an example, my design will be similar to Figure 1. The design also needs a clock signal and a global reset signal. Also, for the adder, I will need a carry-in signal (which we will cover later).

The command in the first diagram is also called an opcode, or operation code. This is a set of bits (a number) that will tell the ALU which action to perform. I can get the LC-3 opcodes for ADD and NOT and ADD from the book, or they are listed later in this assignment.

Next, I need to figure out what kind of inputs and outputs I will need. For example, the inputs require a keypad and a 7-segment display (for verification of input); the output will need a display as well; the control signals will need some switches and lights. You get the idea.

The condition codes are also outputs from the ALU. These include:

- "Z" should be the "is zero?" LED. It will turn on if the result from the ALU is zero (that is, each bit is equal to 0), and will be off otherwise.
- "N" should be the "is negative?" LED. It will turn on if the result from the ALU is negative (that is, the leading bit is a 1), and will be off otherwise.
- "P" should be the "is positive?" LED. It will turn on if the result from the ALU is positive (how do I know?), and will be off otherwise.
- "BC" should be the "is a bad opcode?" LED. It will turn on if the command is anything but an AND, NOT or ADD (how do I know?), and will be off otherwise.

Materials

Now I make a list of all the materials accumulated so far. This list is just an example; yours may be different.

- Two 4-bit inputs
- One 4-bit output
- Two keypads for 4-bit input
- Three 7-segment displays (2 for input, 1 for output)
- A bunch of switches for the opcode
- A bunch of LEDs
- One button for clock
- One button for reset
- One switch for carry-in

Most of these elements are already provided in the ALU.lgi file Next, we will work out the logic for page 2.

Control logic

Control logic is the part of the logic that decodes the opcode, checks for the reset button, checks the clock, and so on. This is a major part of the project, and should be addressed. In the attached pdf file, the inputs and outputs are as follows:

- SR1[3..0] is the 4-bit first operand

- SR2[3..0] is the 4-bit second operand
- DR[3..0] is the 4-bit result; DR stands for destination register
- Cin is the carry-in to the adder
- Cout is the carry-out from the adder
- IR[3..0] is a 4-bit opcode from the instruction register
- Clk is the clock
- Reset is the asynchronous global reset

Table 1: The control logic for the ALU operations. Four bits are needed because we are only implementing a few of the many operations. How many opcodes are in LC3?

Operation	Opcode
ADD	0001
NOT	1001
AND	0101
BC	Everything Else

Ripple carry full adder

There are lots of ways to make an adder. The one we will implement here is called a ripple-carry full adder. The ripple-carry part means that when we need to carry a 1, it will "ripple" down the schematic; full adder means we have a (mandatory) carry-out bit.

The adder in this ALU will add the two inputs and the carry-in, and produce a sum and a carry-out. Table 2 shows the truth table for adding one-bit values A and B, and a carry-in. This is the same adder we saw in class. Figure 2 shows the truth table for this adder.

You can probably infer the logic equation for this device, but your tutor may show you in lab how to make a ripple-carry full adder.

Note that the table in Figure 2 shows you one bits' worth of truth values. To make a multiple-bit ripple-carry adder, you connect several of these pieces together. The carry-out of one adder will be the carry-in of its neighbor. Figure 2 shows a diagram of a 4-bit adder.

Suggestions

- Verify that your logic works for each step of the design. Bugs are easiest to fix when they are caught early! Use the probe tool to see the values at important nodes. Attach LEDs to everything you care about, if it helps you to visualize your design.
- Check your work with actual numbers. Work out the solutions on paper.

Table 2: The truth table for a one-bit full adder

A	B	c_{in}	c_{out}	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

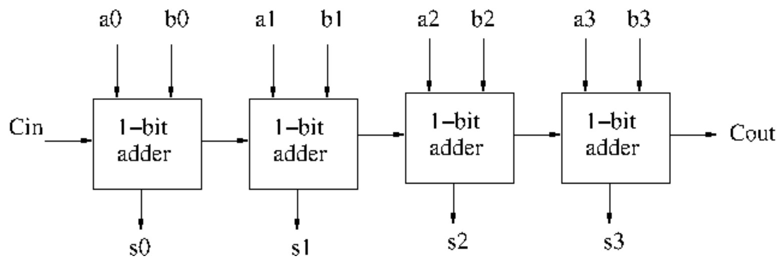

Figure 2: A 4-bit ripple carry adder

- Feel free to use other materials than those listed here. It may make your design more understandable or fun!
- **Get your memory checked by a tutor/TA before continuing onto the ALU. This will reduce confusion.**

Extra credit

In order to receive extra credit points, the entire basic design must be fully functional, and the additional functionality must be added. For 3 extra credit points, add logic to perform a SUB instruction with an opcode of 1111. That is, subtract the second operand from the first (out = in1 - in2). All three values – both inputs and the output – must be two's complement numbers (negative numbers must be represented). Your design must work in one clock cycle.

Lab Submission

Your lab will be submitted via your eCommons account. Please log in to eCommons using your UCSC account and attach the following files to your "Lab2" assignment submission:

- lab2_[username].lgi
- ALU_[username].lgi

- MEMORY_[username].lgi
- lab2_report_[username].pdf

Note that the final report must be submitted in PDF format. Make sure to confirm that your assignment is SAVED and SUBMITTED before the deadline. You may resubmit your assignment an unlimited number of times up until the due date.

Check-off

For this lab, as with most labs, you will need to demonstrate your lab when it is finished to a TA/tutor and get it signed off.

Grading template

This is a suggested grading rubric. It is also a good general guideline before submitting your lab to check off these points.

ALU requirements

☐ (10 pts) Inputs and outputs

- ☐ (1 pts) Have two flip-flopped ("SR1" and "SR2") 4-bit inputs, and display both inputs on 7-segment displays
- ☐ (1 pts) Have one flip-flopped ("DR") 4-bit output of the ALU, displayed on a 7-segment display
- ☐ (1 pts) Have flip-flopped ("IR") LC-3 opcode bits (see textbook if you want details, but enough is provided later in this lab)
- ☐ (1 pts) Have a global asynchronous reset button (labeled "Reset") to reset all registers and displays
- ☐ (1 pts) Have a clock button (labeled "CLK") which acts as a flip-flopped signal for all sequential logic
- ☐ (1 pts) Display the operation in progress with LEDs or other output device
- ☐ (1 pts) Light an LED if the result of an ADD had a carry-out
- ☐ (1 pts) Light an LED if the output of the ALU is zero (labeled "Z"), positive (labeled "P"), or negative ("N")
- ☐ (1 pts) Light an LED whether the opcode was a bad opcode (labeled "BC", other opcode LEDs should be unlit)
- ☐ (1 pts) Display the value "0xBC" on the ALU output when the opcode is a bad opcode

☐ (20 pts) Control logic and ALU operation

- ☐ (5 pts) Be capable of performing a bitwise AND of the two operands

  - ☐ (5 pts) Be capable of performing a bitwise NOT of the first operand
  - ☐ (5 pts) Be capable of performing an ADD of the two operands using a ripple-carry full adder
  - ☐ (5 pts) Be able to recognize a bad opcode

The following is OPTIONAL extra credit:

☐ (3 pts) Implement subtraction instruction in one clock cycle

Lab write-up requirements

In the lab write-up, we will be looking for the following things. The lab report is worth 20 points. We do not break down the point values; instead, we will assess the lab report as a whole while looking for the following content in the report.

- How is sequential logic useful; i.e., why are the inputs and outputs latched?
- What did materials did you need to implement the ALU?
- How many opcodes (types of instructions) does LC3 have?
- Why isn't a SUB instruction (subtract) included in the LC-3 instruction set?
- Why isn't an NAND instruction included in the LC-3 instruction set?
- How would you modify your particular design to implement a NAND instruction also?
- What is the purpose of the N, Z, P, and BC LEDs?