CMPE 12L Lab 2 – Spring 2014
Arithmetic Logic Unit and Memory

100 Points (40 report, 60 work)

Due Date: April 27, 2014

Objective

This lab is broken down into 3 parts. First, you will design and build a multi-page schematic for memory with four registers capable of storing four bits per register. You will then build a schematic to implement the operation of a 4-bit arithmetic logic unit (ALU) capable of performing addition (ADD), negation (NOT), and bitwise AND. Finally, you will combine the memory and ALU.

You will have four lab sessions to complete the work. That means this lab is more challenging than the average lab and may require time outside your lab section. Please plan accordingly. **A partner is mandatory in this lab.**

Part 1: The Memory

From class you learned that combinational logic is just a function of current inputs and that sequential logic is a function not only of current input, but some past sequence of inputs. To store that past sequence information we need storage devices.

The file name MEMORY.lgi contains the memory part of this lab. However, it requires additional logic to make it work. You will add functionality to both read a four-bit number from a register, and write a 4-bit number into a particular register.

Part 2: The ALU

ALU stands for Arithmetic Logic Unit. It's the major part of hardware – in microprocessors in general and the LC-3 in particular – responsible for most of the computations (arithmetic and logic) you perform. This ALU will perform operations on 4-bit words.

Part 3: The ALU and memory combination

With the memory as a storage unit and the ALU responsible for computation, the combination of the two along with their control units, the inputs and outputs form a very simple model of a computer. An actual computer will be more complex than what you will make in this lab but this lab will give you an idea of what is happening inside. You will have control of the clock, the reset, address selection, etc, which in an actual computer will be controlled by other hardware.

What is required?

You will submit two files to the class locker: *ALU_mem.lgi* that contain your schematic and a lab report that includes discussion of all parts of the lab assignment.

Your design should be synchronous and well-organized. You must use multiple pages and proper logic elements (a D flip-flop versus a bunch of NAND gates, etc)

In the finished schematic, your memory should interact with the ALU. That is, the user should be able to store the ALU result into a specific register, and use register contents for the inputs into the ALU.

Requirement for the memory/registers

- You can read from each register individually by selecting the register using the *address select switches*. The result is displayed on the 7-segment display.
- You can write to each register individually be selecting the register using the *address select switches*, entering the number on the keypad, and pressing the clock button.
- You can clear all the latches asynchronously (at the same time, and regardless of the clock) using the clear button.
- The state of the registers change only when the clock or reset buttons are pressed.

Requirement for the ALU

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

- Have two 4-bit inputs using two keypads.
- Have one 4-bit output using 7-segment displays
- Be capable of displaying both inputs using a 7-segment display for each
- Allow the user to input opcodes using switches.
- Display the operation in progress with LEDs
- 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 ripple-carry full adder
- Display the result of the operation on a 7-segment display
- Display with an LED if the result of an ADD had a 2's complement overflow
- Display with an LED if the output of the ALU is zero, positive, or negative.

Requirement for the ALU-memory system

- The first ALU operand is selected by the SR2 selector switches
- The second ALU operand is always from register 3 (R3)
- The result of the ALU is displayed.
- To store the ALU result into a register in memory, KP/ALU signal must be set to ALU, the corresponding DR selector switches must be set, and the clock button must be pushed.
- To store a keyboard value into a register in memory, KP/ALU signal must be set to ALU, the corresponding DR selector switches must be set, and the clock button must be pushed.

Organizing the Design

You will create each part of the lab separately, and test each part thoroughly before combining them. You should organize your design into multiple pages. The following organization is a suggestion; yours may be different

- Memory page 1: User interface (inputs and outputs)
- Memory page 2: Registers
- Memory page 3: Address Decoder and write logic
- Memory page 4: Output Multiplexion
- ALU: Opcode resolution – decode based on instruction opcode
- ALU: Logic for the two bitwise functions AND and NOT
- ALU: logic for the add instructions
- ALU: Output Multiplexion

Procedure for Part 1: The Memory

You will build a register file. You will have four registers in the memory, and each register will store four bits. You will only access one of the four registers at a time.

A block diagram of the memory is shown in the attached file named *ALU_memory_blockdiagram.pdf.* Most of the components for your memory were already implemented in *MEMORY.lgi*. Your job is to implement the missing circuits to make your memory work. The missing circuits are the following:

- Address decoder logic
- Output multiplexion

In the *.lgi* file, you can find descriptions and components needed to build the missing circuits.

Procedure for Part 2: The ALU

In Part 2 you will build an ALU with three possible operations: AND, NOT and ADD. Consider what your design will look like. Organize it onto multiple pages and draw it out in paper. Refer to the attached block diagram in *ALU_memory_blockdiagram.pdf* to give you an idea. **Your lab tutor will not help you if you have not brainstormed in paper first.**

Input/Output

The first page contains all of your input/output (I/O). This first page is available on the class website under *Attachements* in the file called *ALU.lgi*. Inputs are:

- Two keypads that will be the inputs to the ALU
- Carry-in bit for the full adder
- Two opcode switches to determine which operation will be displayed

The outputs are:

- Result from the selected ALU operation
- System status
- Current operation in progress

Opcode Switchs and Operation in Progress

The opcode command is a set of bits (a number) that will tell the ALU what operation to perform. The opcodes for ADD, AND, and NOT are:

Operation	Opcode
ADD	00
AND	01
NOT	10

Current operation LEDs indicate the current ALU operation. The attached block diagram may help you design your control logic encode the opcode. ’The ‘Bad Code’ LED will turn on if the command is anything but an AND, NOT, or ADD, and will be off otherwise.

Condition codes

The system status informs the user what the output of the ALU is. These are called condition codes. These includes:

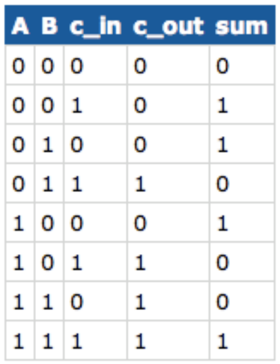
- ‘Z’ should be the ‘is zero?’ LED. It will turn on if the result from the ALU is zero (that is, each bit is equal to zero), and will be off otherwise.
- ‘N’ should be the ‘is negative?’ LED. It will turn if the result from the ALU is negative (that is, the leading bit is a 1), and will be off otherwise.
- ‘P’ should be the ‘is positive?’ LED. It will turn on if the result from the ALU is positive, and will be off otherwise.
- ‘Overflow’ indicates if the ADD operation resulted in an 2’s complement overflow

AND and NOT

The AND operation does a bitwise AND on the two ALU inputs. The NOT operation does a bitwise NOT on the first ALU input.

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.



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 1: Truth table for a 1 bit full adder

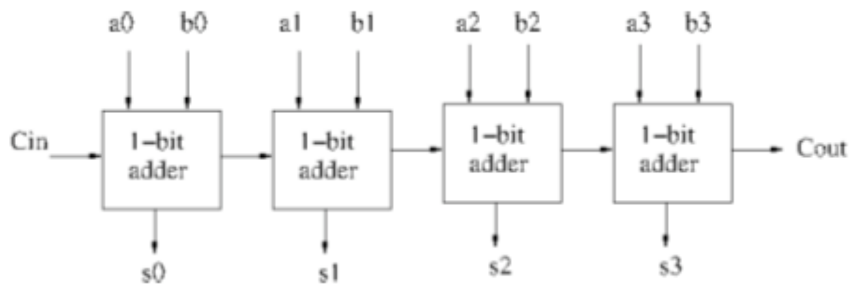


Figure 2: A 4-bit ripple carry adder

The adder in this ALU will add the two inputs and the carry-in, and produce a sum and a carry-out. Table 4 shows the truth table for adding one-bit values A and B, and a carry-in.

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 4 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 5 shows a diagram of a 4-bit adder.

Procedure for Part 3: ALU and memory combination

To combine the ALU and memory, connect one ALU input to register 3, and the second ALU input will come from a set of multiplexers that will select one of the registers. The output of the ALU can be written to a register, or a key pad value can be written to a register. See *ALU_memory_blockdiagram.pdf* for more details.

Extra Credit

- Get the memory part of the lab checked-off by the 2nd lab session.
- Add logic to perform a SUB instruction with an opcode of 11. That is, subtract the second operand from the first (out = in1 – in2). All three values – both inputs and output – must be two's complement numbers (negative numbers must be represented). Your design must work in one clock cycle.

Grading Template

This is a suggested grading rubric. Your tutor may or may not use this rubric to grade by, but is a good general guideline before submitting your lab to check off these points.

Memory requirements: 15(+5) point Total

- (5 pts) Address decoding network is correct
- (5 pts) Ability to display each register bank on Output Display
- (5 pts) Ability to write to each register bank using clock

Extra Credit

- (5 pts) Get the memory part checked-off by the 2nd session.

ALU requirements: 30 (+5) points total

Inputs and outputs

- (5 pts) Have opcode bits that select an ADD, AND, or NOT operation.
- (4 pts) Display the operation in progress with LEDs or other output device.
- (2 pts) Display with an LED if the result of an ADD had a 2's complement overflow
- (3 pts) Display with three LED if the output of the ALU is zero, positive, or negative.

Control Logic and ALU operation

- (4 pts) Be capable of performing a bitwise AND of 2 operands
- (4 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. (<u>Adder implementation should be minimized</u>)
- (3 pts) Be able to recognize a bad code

Extra Credit

- (5 pts) Implement subtraction instruction in 1 clock cycle

Combination requirements: 15 points

- (1 pts) Reset signal works as intended
- (1 pts) Design is organized on multiple pages
- (6 pts) Ability to store ALU result into memory
- (7 pts) Ability to use memory as inputs to ALU

60 (+ 10) Total points

Lab write-up requirements

The lab report is worth 40 points. We do not break down the point values; instead, we will assess the lab report as a whole. Along with the usual items in the report you should answer the following questions:

- How is sequential logic useful; i.e. why are the inputs and outputs latched?
- What is the clock signal used for?
- What number representation did you use? And how do you check for a negative number?
- What is the maximum and minimum number that your ALU can accept?
- Your ALU only accepts 4-bits. What changes do you need so that your ALU can accept 8-bits?