Hui Shi Li

Section 6

2/2/15

Partner: Dan Savulescu

TA: Daphne Gorman

Lab 2

Introduction: The objective of this lab is to understand how the memory, ALU(arithmetic logic unit) works individually and combined. For memory, you write an input to an address, which is decoded using an address decoder to choose which register to read and write from. It also consists of four multiplexers and each multiplexer which selects one bit of the register selected to output to the destination register.

The ALU takes two inputs and does a computation, which can be add, and, invert(not), subtract. It then outputs the result to the destination register. There is also system status LEDs that shows whether the result is has overflowed or underflowed, or is negative, zero or positive. The current operation LEDs light up depending on what opcode you select, which represents the operation the ALU is currently performing.

The combination of memory and ALU allows the user to store two inputs via a single keypad, and choose what operation to perform. The addition of the KB allows the user to store the result of an operation as an input, more specifically into any address the user selects.

Procedure:

- Memory: First, my partner and I implemented the memory part to store the inputs in the address selected. Implementing the address decoder helped us figure out which bit goes in what register. From there, we used the multiplexers to select which bit to output depending on the address selected.
- ALU: To begin the ALU, we first implemented the not operation and the and operation, which were the easiest of the four. The not operation basically flips/invert each bit of the input which results in the output. The and operation does the “and” of each bit of the two inputs. For example, the leftmost bit of input 1 is “anded” with the leftmost bit of input 2 and so on.
- Moving on to the add operation, we first think about the truth table for an add operation, which includes the two inputs, carry in, carryout and sum.

A	B	Cin	Cout	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

We use the sum of products to obtain the equations:

Sum = A’B’Cin + A’BCin’ + AB’Cin’ = $A \oplus B \oplus C$

Cout = A’BCin + AB’Cin + ABCin’ + ABCin = AB + ($A \oplus B$)Cin

From these two equations, we can combine them to create the adder for one bit and connect four of them for four bits.

- As for subtraction, we build off the adder. To subtract is simply equivalent to adding the inverse of the 2’s complement plus 1. For example, 3 – 2, would be 0011 added to 1110 which equals 0001. So, we add an inverter for the second input and have a carry in of 1 to add 1.
- Combining the memory and ALU was simply modifying the names of the inputs and making sure everything was linked up correctly. Since this part requires only one keypad, we take note that the second input will always be stored in the address 11, the register R3. For the KB, storing the result as an input requires us to use a 2 to 1 mux, which selects between the keyboard input and the result to store in memory.

Results:

The most confusing part of the lab was memory, but once I understood how memory works. The rest of the lab was very straightforward with the exception of using one keypad to store two inputs. Originally, I thought that we would need two keypads to store the two input values. But looking over the diagram helped with figuring out that one is sufficient, since the second input would always be stored in the 11 address.

Discussion:

Sequential logic is useful because it not only depends on the present inputs but also the previous inputs or inputs from the past. This allows us to store information and do our computations.

The materials needed to implement the ALU are 2 keypads, 3 of the 7 segment display(two for inputs, one for output), a carry-in switch, two switches for opcode select, 5 LEDs for system status(overflow, negative, zero, positive, underflow), and 4 LEDs to show the current operation.

The LC-3 has 15 opcodes, 2^4 -1.

SUB and NAND are not included in the LC-3 instruction set because they can be constructed from using AND, NOT and ADD. To keep the LC-3 opcode set small and to not be redundant, we exclude SUB and NAND from the LC-3 instruction set.

To create NAND, I would modify the AND design to flip each bit of the result using an inverter.

The purpose of the N, Z, P and BC LEDs is to check if the operation is performed correctly. For example, if you add 2 and 3, you should expect the result to be positive. It also shows if the result is negative, which you don’t necessarily know from the seven segment display for the output.

Conclusion:

From this lab, I learned how memory works, and what happens for an input to be stored in an address and register. I also learned to use gates and the notion of the 2’s complement to implement four basic operation. Lastly, my takeaway from the lab is that the memory and ALU functions differently from each other, but when combined together, it builds on top of one another to work.