Hui Shi Li

Section 6

2/9/2015

TA: Daphne Gorman

# Lab 3

**Introduction**: The objective in this lab is to understand how to write code in assembly and use a LC3 simulator to run the code. In this lab, we create a program in LC3 assembly language that will take in two single digit inputs. Then it will perform subtraction, multiplication and division on these inputs and store them in memory locations, x3100 to x3103.

**Procedure**: Given the echo.asm file as a starting point, I used added to this by taking the second input and storing in another label called userinput2. Then I found that the inputs were stored as ASCII values. So, I had to subtract 48 from the value stored in the labels, because 0 starts at 48, 1 is 49 and so on. To output it, I had to convert back to the ASCII value, so adding 48 back would allow me output to the screen.

Next I decided to implement the three functions, subtract, multiply and divide. To do so, I created subroutines and JSR'd to them. For the multiplication and division subroutines, I had to use BRp, BRn, or BRzp to loop a specific part of the subroutine, which will run that specific part depending on the branch condition. This is key for these two operations, because multiplying is simply adding your first input, x amount of times (where x is your second input). Division is simply subtracting (or adding the two's complement plus 1) divisor from the dividend as many times as possible until you have a negative dividend. Storing the results of these operations require me to use STI to store at a specific memory location.

Results: This lab took a lot of time debugging, especially when I implemented the subroutines, specifically the division subroutine. Stepping through the entire program helped me find me errors and correct them. For instance, stepping into the division subroutine made me realize that I needed to implement the case where it breaks out of the division loop but the remainder was still negative and the quotient counter was 1 more than expected.

**Discussion**:

- Algorithms:

```
Subtract(int A, int B) {
    int result = 0;
    B = Invert(B) + 1;
    Result = A + B;
    Store(result, x3100);
}
```

Example output: A= 6, B= 5, result = x0001, which is 1 in hex

```
**Multiply**(int A, int B) {
    int result = 0;
    while(B>0) {
        result += A;
        B--;
    }
    Store(result, x3101);
}
```

Example output: A=6, B= 5, result = x001E, which is 30 in hex

```
Division(int A, int B) {
    int counter = 0;
    int remainder = A;
    int divisor = Invert(B) + 1;
    while(remainder>=0) {
        counter++;
        remainder= remainder – divisor;
    }
    while(remainder<0) {
        counter= counter - 1;
        remainder = remainder + B;
        Store(counter, x3102);
        Store(remainder, x3103);
    }
    Store(counter, x3102);
```

```
                    Store(remainder, x3103);
            }
```

Example output: A= 6, B = 5, counter= x0001, remainder= x0001

A branch instruction basically loops while the condition code is true. For example, BRz loops as long as the last result is zero and BRzp would loop as long as the last result is zero or positive.

I used STI to store the results in the correct memory location, because it allows us to store the contents of a register to an address. What is stored in the memory of that address is the address of the location to which the data is stored. To better understand this, STI R3 product means store the contents of R3 to product, which is an address pointer.

An addressing mode is what allows us to specify where an operand is located, it can be in memory, in a register or part of an instruction. The five addressing modes are immediate(literal), register, PC-relative, indirect and Base+offset.

Examples:

Immediate- LEA

Register- ADD, NOT AND

PC-relative- LD, ST

Indirect-LDI, STI

Base+offset-  LDR, STR

The last register, R7 is used to put return values for TRAP instructions.

PUTS outputs/displays the string to the screen. It is executed by storing one character in R0 and the other registers hold the rest of the characters. Then it is converted into ASCII before outputting to the screen.

**Conclusion**: This lab taught me the importance of debugging, especially when you have subroutines or larger amounts of code. Stepping through each line of my code helped me understand what is being executed. Also commenting my code helped me keep track of what I was doing and helped me understand where I left off every time I revisited my code.