Jason Ou (jaou 1385128)
Partner: Calvin Pham
CMPE 12L
23 FEB 2015
TA: Daphne Gorman
Section: 6

Lab Write-Up #4

OVERVIEW/INTRODUCTION

The purpose of this lab was to create a LC-3 program that first convert two two-digit base 10 numbers into half-precision floating-point (16-bits) numbers. Second, we had to multiply these floating point numbers. The two operands and the product must be stored in memory. The LC-3 program must be able to handle negative numbers and a well as zero. The starting location of our code will be at 0x3000, and the product of the half-precision floating-point must be stored at address 0x3200.

METHODOLOGY/PROCEDURES

PART A: STORE TWO, TWO DIGIT USER INPUT AS DECIMAL

First the LC-3 program takes in two two-digit user inputs that can also check for a negative case as well. These user inputs are then stored as integers and then converted into decimal form, or integers by using an ASCII offset of -48 to convert the ASCII user inputs into decimal.

PART B: CONVERT RESULTING INTEGERS TO HALFPRECISION FLOATINGPOINT NUMBERS

The two two-digit integers that the users inputted are then converted to half-precision floating-point numbers. To do this, we first convert the numbers the users inputted into binary, then shift the binary number to 2^10 and then remove the leading 1. This is the mantissa for the integer. Afterwards, depending on how many times the user shifted their number to the left, we add this to the number 127 to get the exponent of the floating-point number which we store in 2^14 to 2^10. Depending on whether or not the number is positive (0) or negative (1), we store that into the sign bit at 2^15.

PART C: MULTIPLY THE TWO HALFPRECISION FLOATINGPOINT NUMBERS & STORE

First we add the sign bits of the two half-precision floating-point numbers. We then add the exponents, then we take the first and second fractions, add a leading 1 to them, shift the mantissas to the right 4 times, and multiply the mantissas. We then AND the result with X2000 If the result is 0, then we shift the multiplication result to the correct location, else we add 1 to the exponent before shifting it the correct location. We then remove the leading 1 and store the product into x3200.

RESULTS

PART A: STORE TWO, TWO DIGIT USER INPUT AS DECIMAL

The storing of the two two-digits went well. It was simple to implement and store since we just had to convert the values into decimal before storing them.

PART B: CONVERT RESULTING INTEGERS TO HALFPRECISION FLOATINGPOINT NUMBERS

Converting the resulting integers to half-precision floating-point numbers was a bit tougher, but after some work, we were able to get it working by. The hardest part was shifting the numbers to the right place, but after using masks correctly, we were able to get the program working.

PART C: MULTIPLY THE TWO HALFPRECISION FLOATINGPOINT NUMBERS & STORE

Multiplying the two half-precision floating-point numbers and storing it was not as difficult thanks to the lab 4 flow chart. The Hardest part would have been to shifting the binary number many times to get the numbers to multiply correctly.

Jason Ou (jaou 1385128)
Partner: Calvin Pham
CMPE 12L
23 FEB 2015
TA: Daphne Gorman
Section: 6

DISCUSSION

1. **What's the largest number that can be representing in half-precision floating-point format?**

    - The largest number that can be represented in half-precision floating-point format is 65,504.

2. **What's the smallest positive number that can be represented in half-precision floating-point format?**

    - The smallest number that can be represented in half-precision floating-point format is 2^-14 in binary or 6.10x10^-5 in decimal.

3. **How do JSR and RET work in LC3?**

    - JSR is an instruction that LC-3 supports that allows the user to invoke program fragments multiple times within the same program without having to specfy its details all over again in the source program each time it is needed. JSR allows the user to implement code in a subroutine and at the end of the subroutine, the instruction RET is used to return to an instruction after the JSR instruction that called the subroutine that instructed the RET.

CONCLUSION

This lab helped me understand how to program extensively in LC-3's assembly language. It has helped me become more familiar with the assembly language and become more comfortable and confident when using LC-3 to write a program in assembly. I learned the importance of writing more comments as the program becomes more difficult as each line of code becomes more and more complicated. I also learned the importance of clearing certain registers and as well as using assembly for loading, storing, and calling subroutines. Lab 4 was much more complicated compared to lab 3, but by the end of this programming assignment, I was able to learn how to program a limited calculator for half-precision floating-point numbers.