##############################
Name: Priscilla Usmani- pusmani
Lab section: 3 MW TA: Jay Roldan
Due: 5/19/13
Lab Partner: Stephen Arredondo
##############################

Title:

Lab 4: Floating Point Multiplication

Purpose:

The purpose of lab 4 is to convert two-digit base 10 numbers into half-precision floating-points in 16 bit numbers and multiply the two floating point numbers. The two operands and the products should then display as a sequence of zeros and ones to the terminal.

Procedure:

Before we started the lab, we prepared a flowchart that takes a two-digit sequence of characters from the digits 0-9 and converts them into an integer. Next, we convert the resulting integers from 0 to 99 to a half precision floating point number and outputting a floating point value in R0 as a 16 bit binary sequence with the 3 fields separated by a space.

After the pre-lab was checked by the TA, we started coding.

First step was to make the program ask to input a two digit number. The first number of the two digit number entered it placed in the two digit place number and converted into ASCII. We then use a loop to multiply the first digit by ten. Next, the second digit is inputted and also converted, after being converted the two digits are added and stored together into a variable. The process is repeated for the other two digit number, and is then stored into another variable.

The second step is to convert the integers into a half precision floating point. In order to convert them, we need three subroutines (mantissa, exponent, and sign). For the mantissa, we load a mask and a count. The mask is used to move the 1s and 0s to the spot needed in the 10 bit fraction part. Next is the exponent, the floating point number is in bias 15 so we added 15, and an additional 10 to count for the placing in the exponent. We then negate the mask to put the 0 and 1s in their spots, and shift the exponents. The last part is the sign, we store 0 in a variable and use that for the floating point number. Now that all three parts of the floating point number are set, we add the parts together and store each two digit user input number into a corresponding variable.

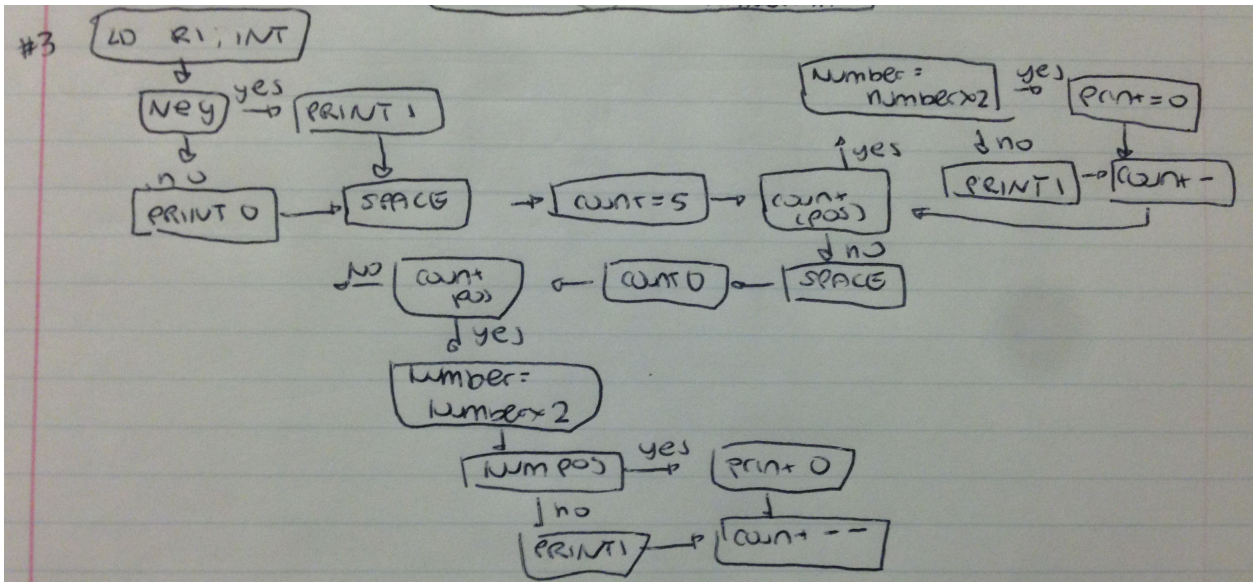
Lastly, we multiply the two numbers together. The first part of the subroutine is where the sign is found, (which is 0 because they are positive numbers), and store it in R0. Due to the bias, we negate the offset by adding 1 to it. Then we added the new offset to our added exponent, which gives us our exponent that is stored. For the multiplication of the mantissa, we loaded the mantissas from the two inputted numbers and the masks. We added the masks to each of the mantissas, and store the first mask in R1 and the second mask to R4. In order to multiply it, we created a loop. We put the contents of R1 or R2 and subtract 1 from the second mantissa. It continues looping until the second mantissa is no longer positive, the R2 then holds the correct product of the mantissa. We have to load another mask and AND it with the result of the mantissa. If the result is zero, we have to normalize the mantissa and go to

another loop. We add 3 into an empty register and enter another loop, we then add the mantissa to itself three times in order to normalize it. Finally we combine the sign exponent and mantissa to print it out with spaces inbetween the sign, exponent, and mantissa for each inputter number and product.

Algorithms and other data:

Convert half precision floating point number
Integer -> Binary -> Result=AND(mash, integer) -> result=0
result=0 (if yes)-> double integer
result=0 (if no)-> exponent-25-count-- -> number=AND(not mash, integer)-> sign =0

Multiply:



What Went Wrong or What Were The Challenges:

The main challenge in this program was getting the write output for the multiplications. The product of the mantissas was difficult because we had to understand how to mask. After getting the concept, it was more doable. The next challenge was trying to get it to do negative, but finding out that it was only extra credit, we didn't try.

Other Information:

What is the largest number that you can represent in half precision floating-point format?
65504
What is the smallest positive number that can be represented in half-precision floating-point format?
2^-24
How does a JSR and RET instruction work in LC3?
JSR: Juumps to a location (like an unconditional branch) and saves current PC (address of next instruction) in R7
RET: gets us back to the calling routine

Enumerate the subroutines you implemented. For each subroutine, include its function, the register(s) that hold the argument values, and the register(s) that holds the return values.

CLRREGISTER= clears the registers to 0

MUL 10: loop that multiplies the first digit by 10 to mark the tens place, uses R2 as a count, R0 is then the number being multiplied

GETINTEGER: gets an integer

toDecimal: substracts 16, converts integer to half precision float

Conclusion:

In this lab we learned to convert a 2 digit sequence of characters to an integer, conver an integer to a half precision floating point number, and recognize the correct sign. I heave learned to modify my input and multiply routines to accept and multiply positive numbers.