TCSS371 – Machine Organization
Homework 6 – Routines

20 Points

This homework will test your understanding of Routines that we covered in lecture from Chapter 8, 9 & 10 of your textbook. Submit your homework on Canvas before the due date and time. Answers to the problems must be submitted as a single zip file with the name, **uwnetidhw6.zip or uwnetid1uwnetid2hw6.zip** containing Crypt.asm and Mult.asm (if attempting extra credit, don't include other file extensions).

No late work or email submissions will be accepted.

Instructions

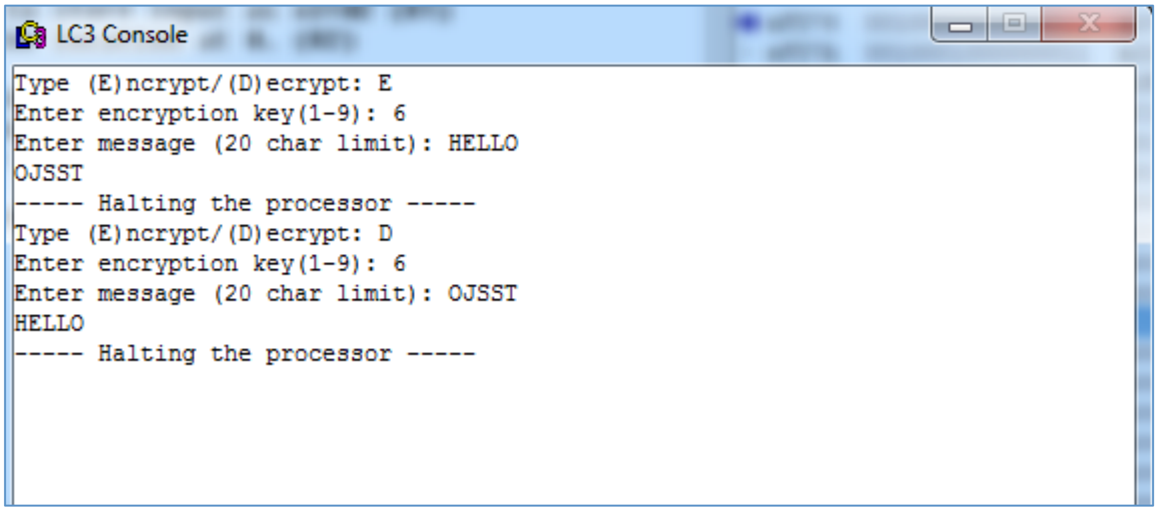
- Write down the steps and draw a flowchart (if it helps) before you start with the Simulator and editor. You don't need to submit these but will help greatly in completing the work.
- Start your program at x3000 so that it will be easier for me to grade it.
- Each program must include a header comment that has your name and a brief description of the program.
- Make sure to comment logic of your assembly program. You will be graded on comments.

Background: Cryptography, from the Greek word kryptos, meaning "hidden", deals with the science of hiding a message from eyes you do not want to understand the contents of the message. The desire to do this has existed ever since humankind was first able to write. There are many ways to keep something secret. One way is to physically hide the document. Another way is to encrypt the text you wish to remain secret. Some people use this to keep others from understanding the contents of the files in their computer. Encrypting a file requires an input message, called the "plain text," and an encryption algorithm. An encryption algorithm transforms "plain text" into "cipher text." Just like a door needs a key to lock and unlock it, an encryption algorithm often requires a key to encrypt and decrypt a message. Just like a homeowner can selectively give the key to the front door to only those he wants to allow unaccompanied access, the author of a message can selectively give the encryption key to only those he wants to be able to read the message. In order for someone to read the encrypted message, he has to decrypt the cipher text, which usually requires the key.

For example, suppose the plain text message is HELLO WORLD. An encryption algorithm consisting of nothing more than replacing each letter with the next letter in the alphabet would produce the cipher text IFMMP XPSME. If someone saw IFMMP XPSME, he would have no idea what it meant (unless, of course, he could figure it out, or had the key to decrypt it.) The key to encrypt in this case is "pick the next letter in the alphabet," and the decryption key is "pick the previous letter in the alphabet." The decryption algorithm simply replaces each letter with the one before it, and presto: the plain text HELLO WORLD is produced.

Implement, in LC-3 assembly language, an encryption/decryption program that meets the following requirements:

Sample Output:



```
Type (E)ncrypt/(D)ecrypt: E
Enter encryption key(1-9): 6
Enter message (20 char limit): HELLO
OJSST
----- Halting the processor -----
Type (E)ncrypt/(D)ecrypt: D
Enter encryption key(1-9): 6
Enter message (20 char limit): OJSST
HELLO
----- Halting the processor -----
```

Input: Your program should prompt the user for three separate inputs from the keyboard, as follows:

1. The prompt: Type (E)ncrypt/(D)ecrypt :
    a. The user will type E or D. We will assume in this assignment that the user can type an E or D correctly. **No validation is necessary**.
    b. Your program will accept that character, must store it in x3100, and use it, as we shall see momentarily.

2. The prompt: Enter encryption key (1-9):

   a. The user will type a single digit, from 1 to 9. We will assume in this assignment that the user can hit digit keys on the keyboard correctly. **No validation is necessary**.
   b. Your program will accept this digit, must store it in x3101, and use it to encrypt or decrypt the message.

3. The prompt: Enter message (20 char limit):
   a. The user will input a character string from the keyboard, terminating the message with the <ENTER> key. We will assume in this assignment that the user can hit digit keys on the keyboard correctly and enter the correct number of characters. **No validation is necessary**.
   b. Your program will store the message, starting in location x3102. You must reserve locations from x3102 to store the message.

Algorithm

The encryption algorithm is as follows.

1. The low order bit of the code will be toggled. That is, if it is a 1, it will be replaced by a 0; if it is a 0, it will be replaced by a 1.
2. The key will be added to the result of step 1 above.

For example, if the input (plain text) is A and the encryption key is 6, the program should take the ASCII value of A, 01000001, toggle bit [0:0], producing 01000000 and then add the encryption key, 6. The final output character would be 01000110, which is the ASCII value F.

The decryption algorithm is the reverse.

1. Subtract the encryption key from the ASCII code.
2. Toggle the low order bit of the result of step 1.

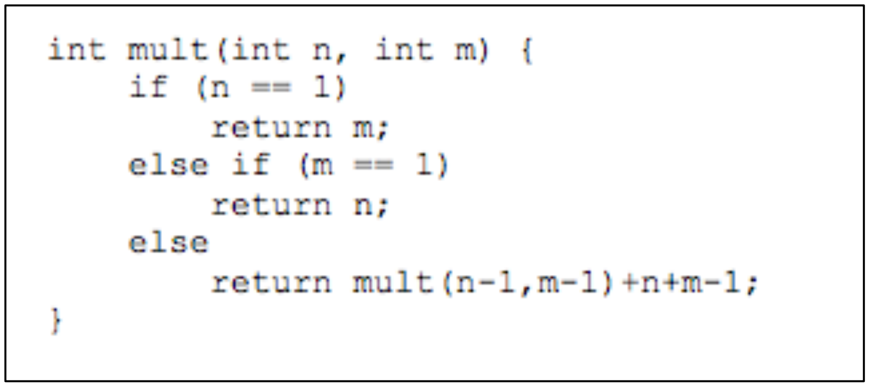
For example, if the input (cipher text) is F, and the encryption key is 6, we first subtract the encryption key (i.e., we add -6) from the ASCII value of F, 01000110 + 11111010, yielding 01000000. We then toggle bit [0:0], producing 01000001, which is the ASCII value for A.

The result of the encryption/decryption algorithm should be stored in locations starting from x3117.

Name your program **Crypt.asm.** ***You must use routines and they must be meaningful in this program to get full credit.***

Extra Credit (10%): Write an LC3 program that includes- a main function (code located at the top of your program)- a function MULT that calculates the product of two non-negative numbers
The main function should load two values from variables named FACT1 and FACT2, initialized to 3 and 8. Call MULT using those values, store the result in a variable named PRODUCT.

The function MULT must be a close translation of the code shown below. It **must** be recursive and make use of the **stack** for local variables (when saving register values, for example). Use registers to send parameters and return values. The callee must save and restore registers as necessary. There is a simpler recursive version of multiply, but you must implement this one.



```
int mult(int n, int m) {
    if (n == 1)
        return m;
    else if (m == 1)
        return n;
    else
        return mult(n-1,m-1)+n+m-1;
}
```

Name your program **Mult.asm**