General notes

** Both CollectKey and CollectText use lastAddress in order to keep track of the location of the most recent added char in either asciiBuff (where key is stored) or MESSAGE (where 10-char-string is stored). Since decrypt only asks for the key to be input, we must temporarily store the lastAddress after CollectText called during Encryption so that the new call to acquire the key does not overwrite the lastAddress.*

**MESSAGE may be overwritten during decrypt due to a wrong key. To avoid this, temporarily store MESSAGE in TempMessage and then restore it after the decrypting.*

**There are two lastAddress labels that contain the same address (lastAddress and lastAddress2). There are also two MESSAGE labels that contain the same address (MESSAGE and MESSAGE2). This is due to the inconvenience that LC3 will not recognize labels too far away from where they are being used. However, do the code being very long and the many places that require the MESSAGE pointer, there simply isn't a place MESSAGE can be placed to serve every place that its being used.*

;==========

Main subroutine:

Display greeting message
Initialize lastAddress pointer to beginning of MESSAGE

while:

prompts command instructions
Get inputted character
if (input ascii equals E)
Branch to Encrypt
if (input ascii equals e)
Branch to Encrypt
if (input ascii equals D)
Branch to Decrypt
if (input ascii equals d)
Branch to Dncrypt
if (input ascii equals X)
Branch to DONE
if (input ascii equals x)
Branch to DONE
Branch back to while ;If PC reaches here, an incorrect input had been detected

Encrypt

Call CollectKey
Call CollectText
Call Caesar
Call Vigenere
Call Shift
Branch back to while ;Await further instructions

```
        Decrypt
                Store lastAddress in tempLastAddress
                Call CollectKey
                Store tempLastAddress in lastAddress ; Restore lastAddress
                Store MESSAGE in TempMessage ;
                Call sDecrypt ;shift decrypt
                Call Vigenere ;Vigenere encrypt and decrypt are the same
                Call cDecrypt ; Caesar decrypt
                Store TempMessage in MESSAGE ; Restore MESSAGE
                Branch back to while ;Await further instructions
        DONE
                Erase encrypted data ;Set value contained at each memory address to 0
                HALT
;=====================================================================
;Caesar subroutine
; This subroutine implements a Caesar encryption on the chars stored starting at MESSAGE. It takes each char, adds the yValue key onto it, and then does modulo 128 in order to get the encrypted ascii value. Stores it back in its memory position in MESSAGE
;=====================================================================
Caesar:
        Save return address
        R6 = lastAddress - MESSAGE ; Acquire the number of characters inputted
        R4 = MESSAGE               ; Acquire MESSAGE pointer
        R5 = yValue                ; Acquire yValue we've calculated during encrypt
        R1 = N                     ; Acquire N value

        caesarLoop
                Load char into R0       ; Load char into R0
                R0 = R0 + R5            ; Add the yValue to char ascii value
                Call Modulo             ; Call mod function for mod 128
                Store R2 in R4          ; Store encrypted char in position in R4
                Increment R4            ; Increment MESSAGE pointer
                Decrement R6            ; Decrement char counter
                Branch to caesarLoop if R6 still positive

        Restore return address
        Return
;=====================================================================
;Vigenere subroutine
; This subroutine implements a Vigenere subroutine on the chars stored starting at MESSAGE. It takes each char and then does XOR K(x1) to get the encrypted ascii value. Stores it back in
```

its memory position in MESSAGE. This subroutine is also used for decryption purposes. Decryption is simply the XOR between encrypted ascii value and K(x1)

```
;=====================================================================
Vigenere:
        Save return address
        R6 = lastAddress - MESSAGE ; Acquire the number of characters inputted
        R4 = MESSAGE               ; Acquire MESSAGE pointer
        R1 = ASCIIBUFF+1           ; Acquire K (x1) from ASCIIBUFF

        vigLoop
                Load char into R0      ; Load char into R0
                Call XOR               ; Call XOR function (R0 XOR R1)
                Store R2 in R4         ; Store encrypted char in position in R4
                Increment R4           ; Increment MESSAGE pointer
                Decrement R6           ; Decrement char counter
                Branch to vigLoop if R6 still positive

        Restore return address
        Return
;=====================================================================
;Shift subroutine
; This subroutine implements a Shift subroutine on the chars stored starting at MESSAGE. It takes each char and then shifts it left by K (z1) amount to get the encrypted ascii value. Stores it back in its memory position in MESSAGE. Shifting a value left is simply multiplying it by 2. We are performing K(z1) doubling operations on the original ascii value in order to get the encrypted ascii value.
;=====================================================================
Shift:
        Save return address
        R6 = lastAddress - MESSAGE ; Acquire the number of characters inputted
        R4 = MESSAGE               ; Acquire MESSAGE pointer
        R1 = ASCIIBUFF+NegASCIIOffset ; Acquire K (z1) then subtract ascii offset

        shiftLoop
                Load char into R0      ; Load char into R0
                Call Shift_Func        ; Call Shift_func (R0 Shift left by R1)
                Store R2 in R4         ; Store encrypted char in position in R4
                Increment R4           ; Increment MESSAGE pointer
                Decrement R6           ; Decrement char counter
                Branch to shiftLoop if R6 still positive

        Restore return address
        Return
```

```
;=====================================================================
;Caesar decrypt subroutine
;This subroutine decrypts a the Caesar encryption. It takes the encrypted ascii value stored at
MESSAGE, adds 128 to it, then subtracts the K(yValue) in order to get the decrypted ascii
value. However, if the original character input ascii + yValue was actually less than 128, we
need a way to account for that. To do this, the Caesar decrypt checks to see if the encrypted
ascii value + 128 - yValue is greater than 127. If this is true, the decrypted value is simply the
encrypted value minus K (yValue). Otherwise, the decrypted value is encrypted ascii value +
128 - yValue.
;=====================================================================
cDecrypt:
        Save return address
        R6 = lastAddress - MESSAGE ; Acquire the number of characters inputted
        R4 = MESSAGE                ; Acquire MESSAGE pointer
        R5 = -yValue                ; We need it to be -yValue for subtraction
        R1 = N                      ; Postive N
        R3 = -N                     ; Negative N for comparison purposes
        cdLoop
                Load char into R0   ; Load char into R0
                R0 = R0 + R1        ; ADD N to R0 ascii value
                R2 = R0 + R5        ; Subtract yValue and load into R2
                R0 = R2 + R3        ; R2 - N
                Branch to greater if R0 is zero or positive ; If (char + key) > 127
                ;Otherwise continue to store result
        storeResult
                Store R2 in R4      ; Store decrypted char into R4
                Increment R4        ; Increment MESSAGE pointer
                Decrement R6        ; Decrement char counter
                Branch to cdLoop if R6 still positive
        Restore return address
        Return

        greater
                Reload char into R0
                R2 = R0 + R5        ; Subtract key in order to get result
                Branch to storeResult

;=====================================================================
;Shift decrypt subroutine
;The shift decryption is simply shifting right by K (z1). Since shifting left once is multiplying by 2,
shifting right once is dividing by 2. One case we need to be careful with is if the ascii value is 0.
0 shifting should still be 0.
;=====================================================================
```

```
sDecrypt:
        Save return address
        R6 = lastAddress - MESSAGE ; Acquire the number of characters inputted
        R4 = MESSAGE               ; Acquire MESSAGE pointer

        sdLoop
                R1 = ASCIIBUFF - NegASCIIOffset ; Acquire K (z1) and subtract offset
                Load char into R0       ; Load char into R0
                R0 = -R0                ; Acquire -R0 for comparison purposes later
                Call sd_Func
                Store R2 in R4          ; Store decrypted char into R4
                Increment R4            ; Increment MESSAGE pointer
                Decrement R6            ; Decrement char counter
                Branch to sdLoop if R6 still positive

        Restore return address
        Return

sd_Func Subroutine:
        R2 = 0
        R5 = 0
        If R0 == 0, branch return
        divide
                R2=R2+1
                R5=R5+2
                R3 = R5 + R0            ; Comparison: If R5 == R0
                Branch divide if NOT 0 ; If R5 != R0, branch to divide
        R0 = R2
        R0 = -R0                ; Set R0 = -R2 and continue cycle
        Decrement R1            ; Decrement key counter (how many times we must divide by 2)
        Branch sd_Func if positive

        Return          RET     ; Return with decrypted value in R2

;=====================================================================
;CollectText subroutine
*This subroutine simply clears the MESSAGE buffer first, then collects up to 10 user inputted
characters and stores it. If more than 10 characters are entered, it will wait until the user presses
enter, then prompts the message that too many characters had been entered
;=====================================================================

CollectText:
        Save return address
```

```
PushChar
        Display instruction to enter text
        Load MESSAGE pointer into R1
        Clears MESSAGE
        Load MaxLength into R2
CharLoop
        Acquire character input
        Test for carriage return
        If character detected, branch GoodInput

        Decrement R2
        If R2 reaches negative, input is too large
        Branch TooLargeInput

        Store last character read
        Increment R1 ; Increment Message pointer

        Store R1 pointer in lastAddress
        Branch to CharLoop

GoodInput
        Restore return address
        Return
TooLargeInput
        Spin until carriage return is detected
        Display too many chars message
        Branch to PushChar
```

```
;=====================================================================
;CollectKey subroutine
```
**This subroutine collects the user inputted key and stores in ASCIIBUFF. It has error checking to make sure the key is entered in the correct format. The key must begin with a digit between 1 and 7 inclusive. The next character must be a non-digit character. The last three characters must be digits that have a combined decimal value based on their position that is stored at yValue. Three digits must be entered (enter 004 for 4, 023 for 23, 127 for 127). This combined value (yValue) must be between 0 and 127.*

```
;=====================================================================
CollectKey:
        Save return address
        Clear current ASCIIBUFF
```

```
PushValue
        Prompt for key input
        R1 points to string being generated
        R2 = MaxChar ;Makes sure that key inputted is 5 characters long
ValueLoop
        Acquire user input
        Test for carriage return
        If carriage return detected, branch to Continue

        Test if input length is not 5
        If input length not 5, branch to InvalidNoLoop
        Otherwise, branch to CheckInput

Continue
        Decrement R2
        If R2 is negative, branch to InvalidInput

        Store last character read
        Increment R1 ;Incrementer pointer to the key being generated

        Store lastAddress
        Branch back to ValueLoop

CheckInput
        Test if first character entered is greater than 0
        Test if first character entered is less than 8
        If both cases pass continue
        If either case fails, branch to InvalidNoLoop

        Tests if second character NOT a digit
        If second character is a digit, branch to InvalidNoLoop
        Otherwise, continue

        First, calls CheckDigit with each of the last three chars
        If any are not digits, branch InvalidNoLoop
        Otherwise, continue
        Call ASCIItoBinary to acquire value of three digits in order
        Check to see if value collected is within range (0-127)
        Store value at yValue

Restore return address
Return
```

InvalidInput
    Spin until character return is detected
InvalidNoLoop
    Prompt invalid input
    Clear current buffer
    Branch to PushValue

CheckDigit
    Checks that the character is a digit
    return
;===
;Helper methods below
;===
XOR subroutine
    Takes R0 and R1 and loads the XOR of the two values into R2.
    ; Same code as submitted in homework

MODULO subroutine
    Takes R0 and R1 and loads the MODULO of the two values into R2
    ; Same code as submitted in lab

Shift_Func
    Shifts left by a certain amount. Every shift left is essentially multiplying the current value by 2. As a result, the decryption (shifting right) is simply dividing by 2.

ASCIItoBinary
    Taken from the calculator files. It converts ASCII values into decimal value used for yValue conversion from user input into decimal.

General Flowchart