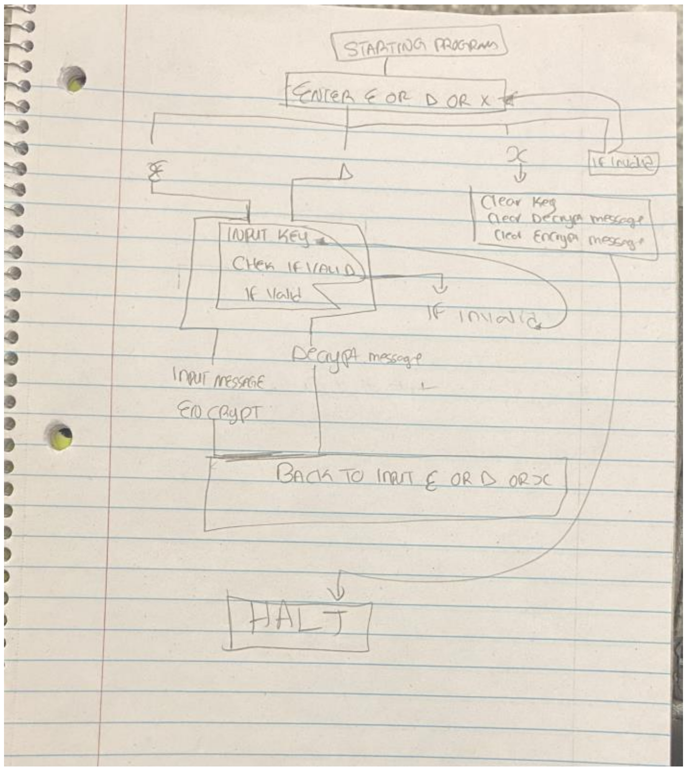
OLAMIDE TREASURE OLUWALADE
PROJECT 4 REPORT

FLOWCHART



DESCRIPTION OF SUBROUTINES

In my code implementation, there are several subroutines.

- **KEYADDR** – In this subroutine, the user inputs their 5-digit key. After inputting the key, I branched to another subroutine called CHECK.
  Note; I saved R7 before branching to the second subroutine and then loaded it back when I returned from the check subroutine.
- **CHECK** – This subroutine checks if the user input is valid or not. If it is valid, it returns and if it is Invalid, it branches back to KEY asking the user to input their key again.
- **goVigenere** – In this subroutine, the encryption code is implemented. Starting with the XOR and then to the modulo. After the modulo, it returns to where it was called and then branches back to the start of the program asking the user for the next input.
- **GoDEC** – In this subroutine, the decryption code is implemented. Starting with the modulo, and then the XOR. After the XOR decryption, it returns to where it was called and branches back to the start of the program. Asking the user for their next input.

ALGORITHM DESCRIBING IMPLEMENTATION

My implementation for the code is as follows

- Print starting program
- Request user for input
- If E, branch the label ENCRYPT

  In the encrypt

  - Go to the subroutine KEYADDR
  - After returning, Ask user for their 10-character message
  - Go to the subroutine goEVigenere
  - After returning, go back to the start of the program and ask for users input again

- If D, branch to the label DECRYPT

  In decrypt,

  - Go to the subroutine KEYADDR
  - After returning, go to the subroutine goDEC
  - After returning, go back to the start and ask the user for input

- IF X, branch to XS

DESCRIPTION OF ENCRYPT AND DECRYPT

- **ENCRYPT:** In the encryption part of this program, using the second digit of the key the 10-digit cypher were encrypted. The second digit key was XORED with each of the 10 digits and then stored in x4000 – x4009.

  After the XOR part, then the encrypted message is encrypted again using MODK. The last three digits of the key are added together and then we add it with the cipher message (this is done index by index.) and then modulo it with the value 128 which becomes (three digits added together + cipher message) Modulo 128. The final encrypted message is stored in x4000 – x4009.

- **DECRYPT:** In the decryption part of this program, I do the opposite of what we did in the encryption part. I started with the MODK part but instead of addition, I did subtraction. Making it (three digits added together - cipher message) Modulo 128. After the MODK part we then XOR using the result from the decrypted MODK part. The final decrypted message is stored in x5000 – x5009

- Note, my key is stored in x3500 - x3504

ANSWERS TO UNDERLINED PART

- **Shifting left in LC3** – To shift left in LC3, we multiply the number we are shifting and 2 to the power of the number of times we are shifting. For example, 5 << 2 = 5 x 2^2 = 20

- **Shifting right in LC3** – To shift right in LC3, we divide the number we are shifting and 2 to the power of the number of times we are shifting. For example, 20/2^2 = 20/2 = 10, 10/2 = 5.