Tharun Saravanan

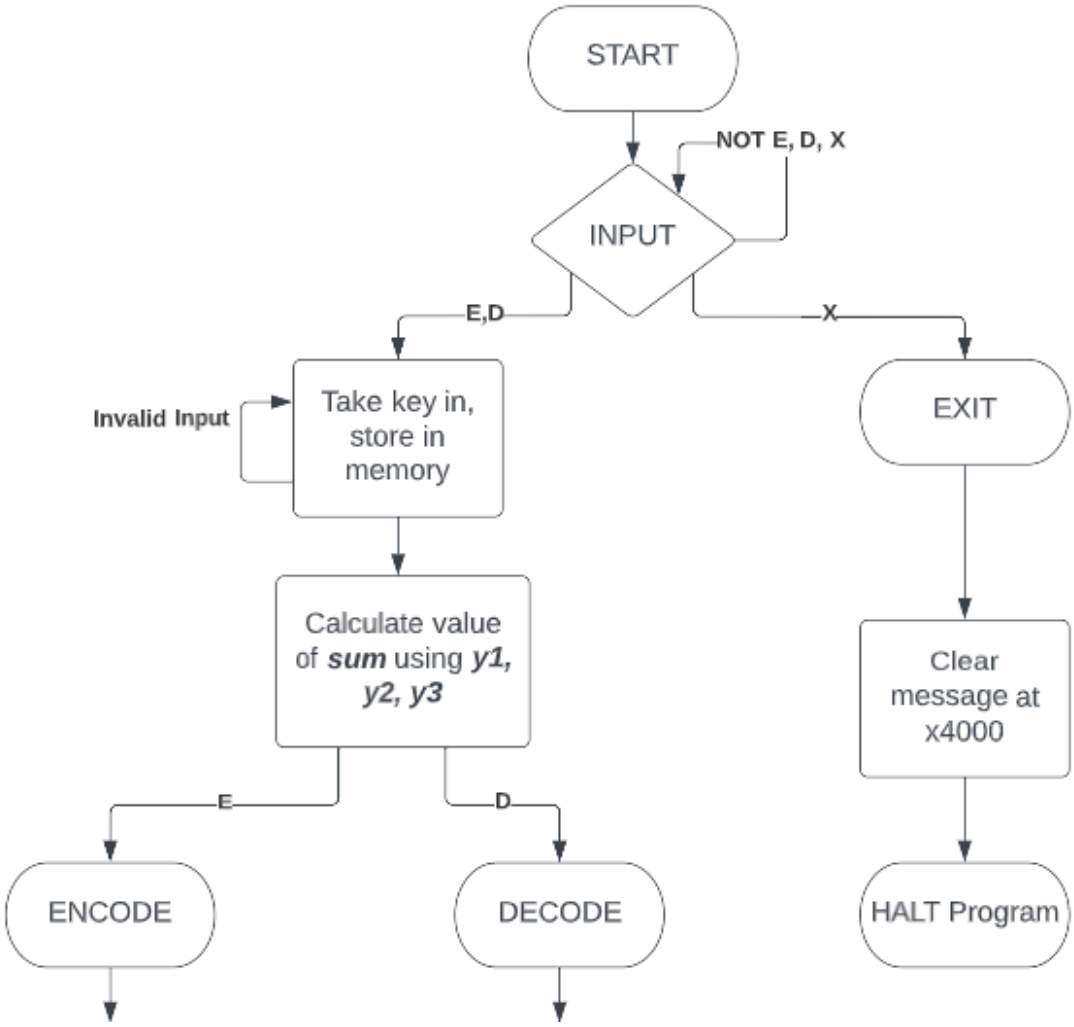
Professor Narahari

Computer Architecture

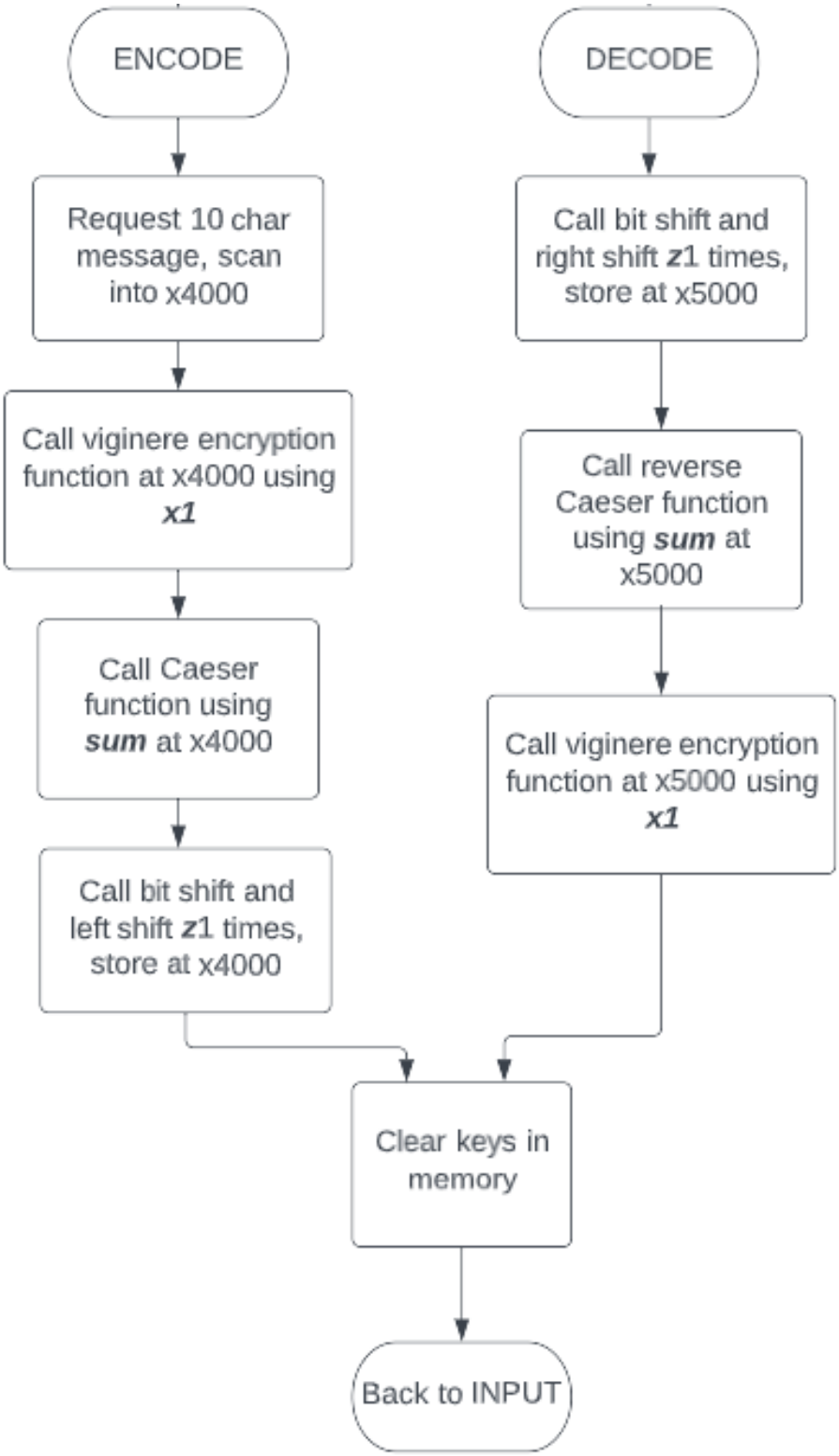
19 November 2022

Project 4 Report

Flowchart of overall sequence of operations:



Encode/Decode flowchart of operations:



a)

Overall description of algorithm:

This program begins by prompting the user to input a character E, D, or X. An E will result in the Encode operations being called, a D will result in the Decode operations being called, and an X will result in the exit operations being called. Any other input at this initial step will produce an invalid input output followed by the prompt being asked again. If Encode or Decode is called, the program will then ask for a key and call the KEYIN subroutine, which will take the next 5 characters input from the user and check if they are valid as a key. If they pass an initial check of being within an acceptable range of values, then y1 through y3 is computed as a 3 digit number and stored into a sum variable. If the 3 digit number is greater than the maximum of 127 or does not pass the initial check, then the user is prompted for the key once again. This is repeated until a valid key is found as part of KEYIN. The program at this point changes behavior depending on whether encode or decode has been chosen.

In the case that encode has been chosen, the program will call the goIN subroutine, which will scan 10 characters from the user and store them in the address of MESSAGE at x4000. After doing this, subroutine goEncryptVI is first called to encrypt using the Vigenere cipher, subroutine goENCRYPTCA is called second to encrypt using the Caesar cipher, and goENCRYPTBIT is called third to encrypt using left bit shifts. The result of this is processed and stored in x4000 as the subroutines are called. Subroutine clearKEY is called finally to erase the

key values from memory. The program from here then branches back to prompt the user for input E, D, or X once more.

In the case that decode has been chosen, the program will call the decrypt methods in the complementary order to the encrypt methods. Subroutine goDECRYPTBIT is called first to decrypt using right shifts, then goDECRYPTCA is called second to decrypt using the Caesar cipher, and goENCRYPTVI is called third to decrypt using Vigenere cipher. The result of this is processed and stored in x5000 as the subroutines are called. Subroutine clearKEY is similarly called finally to erase the key values from memory.

Exit being chosen erases all locations where the encrypted message was stored and where the decrypted message was stored (Starting at x4000 and x5000 respectively and erasing the 10 following locations in memory).

Subroutines:

KEYIN

Pseudocode

```
KEYIN()
    for(int i = 1; i <= 5; i++)
        key[i] = IN
    if(key[1] < 48 or key[1] > 56) //If outside of range
        Branch to KEYIN
    if(48 < key[2] < 58) //If numeric
        Branch to KEYIN
    if(key[3] < 48 or key[3] > 57) //If outside of range
        Branch to KEYIN //Not a number
    if(key[4] < 48 or key[4] > 57) //If outside of range
        Branch to KEYIN //Not a number
    if(key[5] < 48 or key[5] > 57) //If outside of range
        Branch to KEYIN //Not a number
    sum += 100 * key[3]
    sum += 10 * key[4]
    sum += key[5]
    if(sum > 127) //If sum exceeds range
        Branch to KEYIN
    return
```

This subroutine checks each input value's (input from the user) ascii value to see if it's in the allowable ascii value range. If it is not, then it will branch to the start and retry. Further as a check of the last 3 digits, the number is computed into a sum by multiplication with its corresponding place, and if the sum is outside of the acceptable range, then it will also branch to the start and retry

goIN

Pseudocode:

```
goIN()
    print("ENTER MESSAGE") //Print asking for input
    for(int j = 0; j < 10; i++) //Take 10 characters
        int k = IN //Take input
        Store(k, x4000 + j) //Store at respective address
    return
```

This subroutine simply requires taking in an input from the user 10 times and then storing this input at its respective location in memory starting from x4000 (Address of MESSAGE) + (n - 1) (the n-1th character being input)

encryptVI

Pseudocode:

```
encryptVI(int x1, address) //Address will be x4000 or x5000
                           //Depending on whether encrypting or decrypting
    int k
    for(int j = 0; j < 10; i++) //Take 10 characters
        Load(k, address + j) //Store at respective address
        k = k XOR x1
        Store(k, address + j) //Store at respective address
    return
```

This subroutine needs to parse each of the values at either x4000 or x5000 depending on whether the program is encrypting or decrypting. Then it calls the XOR between the value stored in memory and key value x1. This works because the opposite operation of an XOR is just another XOR with the same key, hence why we can use the same routine to encrypt and decrypt the stored message.

encryptCA

```
encryptCA(int sum)
    int k
    for(int j = 0; j < 10; i++) //Take 10 characters
        Load(k, x4000 + j) //Load from respective address
        k = k + sum //Add sum
        k = mod(k, 128) //Perform mod to get encrypted value
        Store(k, x4000 + j) //Store at respective address
```

This function simply requires adding sum to each of the values retrieved from memory at x4000, and performing modulus between this value and 128 to encrypt using the Caesar Cipher.

decryptCA

```
decryptCA(int sum)
    int k
    for(int j = 0; j < 10; i++) //Take 10 characters
        Load(k, x5000 + j) //Load from respective address
        k = k - sum //subtract the sum from the value
        k = mod(k, 128) //Perform mod to get original value
        Store(k, x5000 + j) //Store at respective address
```

This function gets the reverse Caesar Cipher value by subtracting the sum from the value stored, and then performing mod 128 on it, as this is derived by solving for the output in the original encrypt function ci = (pi + K) modulo N.

encryptBIT

```
encryptBIT(int z1)
    int k
    for(int j = 0; j < 10; i++) //Take 10 characters
        Load(k, x4000 + j) //Load from respective address
        k = k * 2^z1 //Multiply by 2 z1 times to left shift z1 bits
        Store(k, x4000 + j) //Store at respective address
```

We can perform a left bit shift of n bits by taking each value in memory and multiplying the value by 2 a total of z1 times (2 to the power of z1). This works because performing a left bit shift which shifts all of the bits left by 1 doubles the value.

decryptBIT

```
decryptBIT(int z1)
    int k
    for(int j = 0; j < 10; i++) //Take 10 characters
        Load(k, x4000 + j) //Load from 4000 (first decrypt operation)
        for(int i = 0; i < z1; i++) //Divide by 2 a total of z1 times
            k = k / 2
        Store(k, x5000 + j) //Store at 5000 for next decryption algorithms
```

Similar to the left bit shift, we can perform a right bit shift of n bits by taking each value in memory and dividing the value by 2 a total of z1 times. This works because performing a right bit shift which shifts all the bits right by 1 halves the value.

clearKEY

```
CLEARKEY()
    sum = 0 //Clear sum (Holds y1, y2, y3)
    for(int i = 1; i <= 5; i++)
        key[i] = 0 //Simply parse through all key values and set = 0
```

Clearing the key simply requires setting the 5 key values and the computed sum of y1, y2, y3 equal to 0. Called at the end of encode and decode to erase the key.

b)

How do you decrypt a message that has been encrypted using this scheme?

- The opposite of an XOR operation is simply another XOR operation. Thus if we call A XOR B to get C, we can call C XOR B to get A once again.

How do you shift left in LC3? How do you right shift in LC3?

- Performing a left bit shift which shifts all of the bits left by 1 doubles the value. Thus we can simply multiply the ascii value by 2 for a total of z1 times to achieve a left shift of z1 bits. Similarly, performing a right bit shift which shifts all of the bits right by 1 halves the value. Thus we can simply divide the ascii value by 2 for a total of z1 times to achieve a right shift of z1 bits.