OVERVIEW:

Goal:

Create a procedure that is used by the Top Secret Agency to encypt data sent between their agents.

Definitions:

- key_map: The string of keys that determine the rule by which we map a message.
- message_ar: Ths string containing the encrypted/decrypted message.
- mode_number: An integer from -3 to 0 that specify the mode of operation.
- valid input: Integer from -3 to 0. (Any other input will select decoy mode)

Modes of Operation (Strategy for modularizing code):

* Decoy:
* Encryption:
* Decryption:
* Key Generation:

COMPUTE Procedure:

```
Goal:
    The procedure provides functionality needed for the Top Secret Agency
    as part of their data encryption program.

Description:
    This procedure contains 4 operation modes. Each operation mode
    is selected based on the 32-bit signed integer passed on the stack
    one line before calling the COMPUTE procedure. This integer ranges from
    [0,-3]. It will perform the task as specified below by each operation mode.

    * Note: All general purpose registers are stored and restored
        by the Compute procedure. Number of general-purpose
        registers were adequate during procedure design.

    * Potential Upgrades:
        1. Incorporate data validation inside the Compute procedure.
        2. Implement the key generation algortihm to use
           a binary-search algorithm instead of a linear search.
        3. Implement a method to dynamically calculate the length
           of the new key_map to be generated.

Requirements:
        - No global variables.
```

```
        - All parameters must be passed on the stack.
        - Only compute procedure must be submitted.
        - No null-terminating string as key_map.

Resources Used:
    Assembly Language for x86 Processors (7th edition) textbook
    by Kip Irvine provided sufficient information to create and
    design the Compute procedure.
```

Basic Outline:

```
Input:
    Parameter1: May vary,
    Parameter2: May vary.
    Parameter3: Must be a 32-bit integer in range [-3, 0]

Enter procedure.

    Validate input. (Reprompt user until valid input is entered.)
    After valid input was obtained:

    IF input == -3,
        If true, proceed to Key Generation mode.
        This will substitute the current key with a new
        randomly generated key.

    IF input == -2,
        If true, proceed to Descryption mode.

    IF input == -1,
        If true, proceed to Encryption mode.

    Else,
        Proceed to Decoy mode.

Output:
    IF input ==  3,
        A new key_map is generated that replaces the old key_map.
        (Do not create a new key_map if you still have some messages
         that are yet to be decrypted. Old key_map will be lost.)
    IF input ==  -2,
        The encrypted string passed by address is gets decrypted.
        (That is, the message in normal English.)
    IF input ==  -1,
        The string passed by address on the stack will be encrypted based on
        the rules specified by the key_map.
```

```
    Else,
        The address passed to the procedure will contain the sum
        of the other two integers.
```

1. Decoy (0):

```
Goal: Take two 16-bit words, add them and store the result in  at a given
    memory location.
```

2. Encryption (-1):

```
Goal: Take a string, key_map, and a 32-bit signed integer and encrypt
    the string based on the rules specified by the key_map.
```

3. Decryption(-2):

```
Goal: Take an encrypted string, key_map, and a 32-bit signed integer,
    and decrypt the string using the rules specified in the key_map.
```

4. Key Generation(-3):

```
Goal: Create a new random key_map and replace it with the existing one.

Descryption:
    The main loop will iterate 26 times. That is after each iteration of the
    main loop, one random charater will be stored in the array given.

Psuedo Code:
    cx = 26
    For cx > 0:
        *push   cx
        *push   current location in string.
        *Inside Loop:

            -Create a random number between [97,122], inclusively.
            -If number already in the string, create new random
             number and repeat.
             Continue this search untill a number found that is not in
             the string.
            -Break loop.

        *pop    current lcoation in string.
        *Insert the above integer found in the string. (the integer represents
         the ascii value of the charater)
        *Walk one byte along in the string.
        *pop    ecx
```

```
        *dec    ecx

    Return to start of loop.
```

Testing:

```
Example of testcases used:

Case 1: For Encryption and Decryption Operation Modes.
------------------------------------------------------
    Input:
        key_map: "qwertyuiopasdfghjklzxcvbnm"
        message1: "rugby ball"
        message2: "This is an extremely long message thet will
                need to be decoded"
    Output:
        key_map: "qwertyuiopasdfghjklzxcvbnm"
        message1: "kxuwn wqss"
        message2: "Tiol ol qf tbzktdtsn sgfu dtllqut ziqz voss
                fttr zg wt rtegtr"

    *Note: message1 and message2 are successfully decrypted.

Case 2: For Encryption and Decryption Operation Modes.
------------------------------------------------------
    Input:
        key_map: "mnbvcxzasdfghjklpoiuytrewq"
        message1: "'History does not repeat itself, but it often rhymes'
                - Mark Twain"
        message2: "Go Beavers!!!"
    Output:
        key_map: "mnbvcxzasdfghjklpoiuytrewq"
        message1: "'Hsiukow vkci jku oclcmu suicgx, nyu su kxucj oawchi'
                - Mmof Trmsj."
        message2: "Gk Bcmtcoi!!!"

    *Note: message1 and message2 are successfully decrypted.

* Remarks: The capital letters will not get encrypted because the condition
       for encryption only encrypts lowercase letters.
```

Extra Credit:

1. In decoy mode the Compute procedure will correclty calculate the sum of any two 16-bit integers and store the result in the 32-bit memory location provided.
2. Passing -3 as the parameter on the stack will generate a new key_map and replace it with the previous key_map.

Rough Work: