Lab 5: Subroutines

Due 15 March 2019 11:59 PM

Minimum Submission Requirements

- Your Lab5 folder must contain the following files (note the capitalization convention):
  - Lab5.asm
  - Diagram.pdf
  - README.txt
- Commit and Push your repository

Lab Objective

In this lab, you will learn how to implement subroutines and manage data on the stack.

Lab Preparation

Read chapters 5, 6, 8.1, and 8.2 from Introduction To MIPS Assembly Language Programming.

Description

This program will allow the user to encrypt or decrypt strings using a Caesar Cipher. In order to generate the Caesar Cipher shift value, the user will also enter a key. The program will calculate the checksum of the key, and use that checksum to shift each letter in the string that should encrypted/decrypted. Then, the encrypted and decrypted strings are displayed to the user.

Caesar Cipher

The Caesar Cipher is a simple encryption method in which every letter in the string to be encrypted is shifted some amount to the right or left in the alphabet. For example, if you are using a right shift of 2, and the string you want to encrypt is "ABC XYZ", then the encrypted version of that string would be "CDE ZAB". Each letter in the original string was shifted two letters to the right, and letters at the end of the alphabet were looped back around to the beginning of the alphabet. For more information about Caesar ciphers, check out the Wikipedia article: https://en.wikipedia.org/wiki/Caesar_cipher

Specification

User Input

The strings that will be given to the user as prompts are contained in the .data segment defined in the Lab5Main.asm file. The addresses of these strings will be part of the arguments to your subroutines. Your code will be responsible for saving the user input in allocated memory.

Allocating Memory (Arrays)

You will need to allocate three chunks of memory (arrays) in the .data section for the user input string to be encrypted or decrypted and resulting string as well as the user input key. These should be able to store a max of 100 characters. You will also need to reserve a byte of memory to keep track of the users choice of E, D, or X.

Checksums

You will be calculating a checksum from the key string given by the user. The checksum is a single value which will be used as the shift value for the Caesar Cipher. In order to calculate the checksum, you will take the user's key string, and xor each byte together. For example, if the user inputs "test" as their key string, you will calculate the checksum as follows:

checksum = ('t' ⊕ 'e' ⊕ 's' ⊕ 't') mod 26

= (0x74 ⊕ 0x65 ⊕ 0x73 ⊕ 0x74) % 26

The above example results in a checksum of 22. Therefore, if the user inputs 'test' as the key, you should use a Caesar Cipher right shift of 22 to encrypt the string. Read more about checksums here:
https://en.wikipedia.org/wiki/Checksum#Parity_byte_or_parity_word

The Stack

In this lab, you will use the program stack to handle the preservation of certain register values at the start of a subroutine so that they can be restored at the end of a subroutine. The most notable use of this will be preserving the jump and link return address, $ra, so you can navigate out of nested subroutines. In addition, the values in registers $s0 - $s7 must be preserved across subroutine calls.

Subroutines

Lab5.asm will contain the subroutines to display and encrypt or decrypt the string. You must implement all of the subroutines listed, and you may create more of your own subroutines. If you plan on using any of the saved registers, $s0 - $s7, you must save these registers appropriately on the stack (push at the beginning of your subroutine, then pop at the end of the subroutine).

Nested Subroutines

Three of the following subroutines will be called from within another subroutine. Specifically, *compute_checksum*, *encrypt*, and *decrypt* will be called from inside of *cipher*, and *check_ascii* will be called from inside *encrypt* and *decrypt*. In order to properly execute these nested subroutines, you must use the stack to handle the return address register, $ra.

Subroutines

```
#-----------------------------------------------------------------
# give_prompt
#
# This function should print the string in $a0 to the user, store the user's input in
# an array, and return the address of that array in $v0. Use the prompt number in $a1
# to determine which array to store the user's input in. Include error checking for
# the first prompt to see if user input E, D, or X if not print error message and ask
# again.
#
# arguments:  $a0 - address of string prompt to be printed to user
#             $a1 - prompt number (0, 1, or 2)
#
# note:             prompt 0: Do you want to (E)ncrypt, (D)ecrypt, or e(X)it?
#                   prompt 1: What is the key?
#                   prompt 2: What is the string?
#
# return:     $v0 - address of the corresponding user input data
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# cipher
#
# Calls compute_checksum and encrypt or decrypt depending on if the user input E or
# D. The numerical key from compute_checksum is passed into either encrypt or decrypt
#
# note: this should call compute_checksum and then either encrypt or decrypt
#
# arguments:  $a0 - address of E or D character
#             $a1 - address of key string
#             $a2 - address of user input string
#
# return:     $v0 - address of resulting encrypted/decrypted string
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# compute_checksum
#
# Computes the checksum by xor'ing each character in the key together. Then,
# use mod 26 in order to return a value between 0 and 25.
#
# arguments:  $a0 - address of key string
#
# return:     $v0 - numerical checksum result (value should be between 0 - 25)
#-----------------------------------------------------------------
```

```
#-----------------------------------------------------------------
# encrypt
#
# Uses a Caesar cipher to encrypt a character using the key returned from
# compute_checksum. This function should call check_ascii.
#
# arguments:  $a0 - character to encrypt
#             $a1 - checksum result
#
# return:     $v0 - encrypted character
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# decrypt
#
# Uses a Caesar cipher to decrypt a character using the key returned from
# compute_checksum. This function should call check_ascii.
#
# arguments:  $a0 - character to decrypt
#             $a1 - checksum result
#
# return:     $v0 - decrypted character
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# check_ascii
#
# This checks if a character is an uppercase letter, lowercase letter, or
# not a letter at all. Returns 0, 1, or -1 for each case, respectively.
#
# arguments:  $a0 - character to check
#
# return:     $v0 - 0 if uppercase, 1 if lowercase, -1 if not letter
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# print_strings
#
# Determines if user input is the encrypted or decrypted string in order
# to print accordingly. Prints encrypted string and decrypted string. See
# example output for more detail.
#
# arguments:  $a0 - address of user input string to be printed
#             $a1 - address of resulting encrypted/decrypted string to be printed
#             $a2 - address of E or D character
#
# return:     prints to console
#-----------------------------------------------------------------
```

Extra Credit

As extra credit, add a subroutine to print the decrypted string using a rail fence cipher with 3 rails, and the corresponding ciphertext. Call your new rail_fence subroutine from print_strings. The rail fence cipher should ignore punctuation and spacing. See this Wikipedia page for an example
https://en.wikipedia.org/wiki/Rail_fence_cipher#Method

Testing

Your program must work properly with the main program provided (on Canvas) Lab5Main.asm.

Output

An example of the expected output is shown below. In this example, the user first encrypts the string “Hello World!” using the key “cmpe12”. Then, the user decrypts the string “Fcjjm Umpjb!” using the same key “cmpe12” in order to get back the original string, “Hello World!”. As shown below, capital letters should remain capitalized when encrypted or decrypted, and non-letter characters should not be encrypted/decrypted.

Sample Output

```
Welcome to the Caesar Cipher program!

Do you want to (E)ncrypt, (D)ecrypt, or e(X)it? E
What is the key? cmpe12
What is the string? Hello World!

Here is the encrypted and decrypted string
<Encrypted> Fcjjm Umpjb!
<Decrypted> Hello World!

Do you want to (E)ncrypt, (D)ecrypt, or e(X)it? D
What is the key? cmpe12
What is the string? Fcjjm Umpjb!

Here is the encrypted and decrypted string
<Encrypted> Fcjjm Umpjb!
<Decrypted> Hello World!


Do you want to (E)ncrypt, (D)ecrypt, or e(X)it? g
Invalid input: Please input E, D, or X.

Do you want to (E)ncrypt, (D)ecrypt, or e(X)it? X

Goodbye!
-- program is finished running --
```

Sample Output With Extra Credit

```
Welcome to the Caesar Cipher program!

Do you want to (E)ncrypt, (D)ecrypt, or e(X)it? E
What is the key? cmpe12
What is the string? Hello World!

Here is the encrypted and decrypted string
<Encrypted> Fcjjm Umpjb!
<Decrypted> Hello World!

Extra Credit
<Cipher>
H . . . O . . . L .
. E . L . W . R . D
. . L . . . O . . .
<Cipher Text> HOLELWRDLO

Do you want to (E)ncrypt, (D)ecrypt, or e(X)it? D
What is the key? cmpe12
What is the string? Fcjjm Umpjb!

Here is the encrypted and decrypted string
<Encrypted> Fcjjm Umpjb!
<Decrypted> Hello World!

Extra Credit
<Cipher>
H . . . O . . . L .
. E . L . W . R . D
. . L . . . O . . .
<Cipher Text> HOLELWRDLO


Do you want to (E)ncrypt, (D)ecrypt, or e(X)it? g
Invalid input: Please input E, D, or X.

Do you want to (E)ncrypt, (D)ecrypt, or e(X)it?
X

Goodbye!
-- program is finished running --
```

Files

Diagram.pdf

This file will contain a block diagram or flowchart of how the different components of your code work together.

Lab5.asm

This file contains your implementation of all subroutines. It must assemble on its own. See the *Code Documentation* section from previous labs for instructions on proper code formatting. Pseudocode is not required for this lab, though you may include it

after the header comment. You must include a block comment on register usage as indicated in previous labs.

README.txt

This file must be a plain text (.txt) file. It should contain your first and last name (as it appears on Canvas) and your CruzID. Your CruzID is your email address before @ucsc.edu. Your answers to the questions should total at least 8 sentences with complete thoughts.

Your README should adhere to the following template:

```
------------------------
Lab 5: Subroutines
CMPE 012 Winter 2019

Last Name, First Name
CruzID
-------------------------

What was your design approach?
Write the answer here.

What did you learn in this lab?
Write the answer here.

Did you encounter any issues? Were there parts of this lab you found
enjoyable?
Write the answer here.

How would you redesign this lab to make it better?
Write the answer here.

Did you collaborate with anyone on this lab? Please list who you collaborated
with and the nature of your collaboration.
Write the answer here.
```

Grading Rubric

Overall (5 pt)

5 pt all subroutines work correctly with main code

Subroutines (27 pt)

Note: credit will only be given for this section if Lab5.main assembles on its own

3 pt give_prompt
- 2 pt handles error checking with prompt and proper formatting
- 1 pt correctly returns memory address of user input

```
4 pt   cipher
       1 pt  calls compute_checksum with proper input arguments
       1 pt  calls encrypt/decrypt with proper input arguments using jal
       1 pt  manages stack to save $ra register
       1 pt  correctly returns memory address of resulting string
1 pt   compute_checksum
          -  correctly computes the checksum from the user input key and returns
             numerical value between 0-25 from checksum result
1 pt   check_ascii
          -  correctly checks character range and returns value indicating range
4 pt   encrypt
       1 pt correctly encrypts and return lower case character
       1 pt correctly encrypts and return upper case character
       1 pt correctly handles non-letter character
       1 pt calls check_ascii using jal
4 pt   decrypt
       1 pt correctly decrypts and return lower case character
       1 pt correctly decrypts and return upper case character
       1 pt correctly handles non-letter character
       1 pt calls check_ascii using jal
3 pt print_strings
       2 pt  prints correct string with formatting
       1 pt  prints encrypted and decrypted strings in the correct order
7 pt   stack usage
       1 pt  per subroutine: manages stack to save $s register values if using them
```

Documentation / Style (8 pt)

```
8 pt   style and documentation
       2 pt diagram effectively describes flow of program (-1 if hand drawn)
       1 pt comment on register usage
       1 pt useful and sufficient comments
       1 pt labels, instructions, operands, comments lined up in columns
       2 pt README contains at least 6 sentences total with complete thoughts
       1 pt complete headers for code and README
       Note: program header must include name, CruzID, date, lab name, course name,
       quarter, school, program description; README must include name, CruzID, lab
       name, course name
```

Extra Credit (8 pt)

```
8 pt   print the decrypted string using a rail fence cipher with 3 rails, and the
corresponding ciphertext
```