COS284 Practical Assignment 1: The Spy's Dilemma

University of Pretoria
Department of Computer Science

Online Publication: 09 August 2024

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Plagiarism Policy

- All work submitted must be your own. You may not copy from classmates, textbooks, or other resources unless explicitly allowed.
- You may discuss the problem with classmates, but you may not write or debug code with other classmates, other than those in your group.
- If you use any advanced material, you must cite these sources.

The Spy's Dilemma

In the heart of a bustling metropolis, a top-secret government agency known as the CipherGuard operates silently behind the scenes. The agency is tasked with protecting the nation's most sensitive communications from prying eyes. As promising new recruits, you and your fellow classmates have been selected to undergo a special training program to hone your skills in cryptography and secure communication. Your mission, should you choose to accept it, involves five critical tasks that will determine your future as elite CipherGuard operatives.

Mission Objectives

1. Understand basic input/output operations in assembly language.
2. Implement conditional logic to control program flow based on user input.
3. Develop a basic encryption algorithm to transform plaintext into ciphertext.
4. Create a decryption algorithm to revert ciphertext to plaintext.

5. Integrate the above functionalities into a cohesive program that processes user input and performs encryption or decryption.

Mission Details

At a high level, the experience should simulate a real-world cryptographic operation:

```
Welcome agent. What do you want to do, Encrypt [1] or Decrypt [2]?
Choice: 1
Enter plaintext to encrypt: MOVE
The cipher text is: 1930507175 1930507143 1930506775 1930507047
```

Although this looks like a simple task, there is quite a lot going on here. The following sections will break it down and provide hints where needed.

Analysis

The program receives input from the user and makes decisions based on that input. It must also perform encryption and decryption operations. Each task will build on the last, culminating in a final program that integrates all components.

Task 1: Greeting

The first requirement is to output a welcome message to the user, prompting them to choose whether to encrypt or decrypt a message. The welcome message should look as follows:

```
Welcome agent. What do you want to do, Encrypt [1] or Decrypt [2]?
```

You will write all the necessary code for this greeting inside the provided file called **greeting.asm**. A partial skeleton has been provided. You will need to fill in the blanks.

Task 2: User Choice

Your next task is to prompt the user for input to either encrypt or decrypt a message. Implement logic to read an integer input (1 for encryption, 2 for decryption) from the user.

```
Choice: 1
```

You will write all the necessary code for this inside a file called **get_user_choice.asm**. A partial skeleton has been provided. You will need to fill in the blanks.

Task 3: Encrypt Functionality

The third step requires you to implement a simple encryption algorithm, using a basic Caesar cipher. First, the bits of each character have to be rotated left by 4 bits, and secondly, an exclusive or operation will have to be applied to all of the character's bits. The exclusive or operation that should be applied to each character should use this hexadecimal constant:

```
0x73113777
```

You will write all the necessary code for this inside a file called **encrypt_and_print.asm**. A partial skeleton has been provided. You will need to fill in the blanks.

After successfully implementing this, the output should look as follows:

```
Enter plaintext to encrypt: MOVE
The cipher text is: 1930507175 1930507143 1930506775 1930507047
```

Task 4: Decrypt Functionality

The fourth step is to implement the decryption algorithm that reverses the encryption process. You will write all the necessary code for this inside the file called **decrypt_and_print.asm**. A partial skeleton has been provided. You will need to fill in the blanks.

After successfully implementing this, the output should look as follows:

```
Enter cipher text to decrypt: 1930507175 1930507143 1930506775 1930507047
The plaintext is: MOVE
```

Task 5: Integrate Functions

Now, combine all the functionalities to create a cohesive program. This final task will use the greeting, get the user's input, and perform the appropriate encryption or decryption based on the input.

You won't need to write any more code for this task. Thus far, we marked the files you implemented for each task and overrode the rest to allow for partial marks; this time, all your implementations will be used as is and called together.

Assumptions

You may assume the following:

- The user will provide valid input for choice, encryption, and decryption that is fitting for them.
- The user will always provide exactly 4 characters to encrypt.

Submission

You will submit your assignment via **FitchFork** at ff.cs.up.ac.za. You will submit all of the files as an archive (.zip), and they will be graded individually. You will be graded on the following files:

```
greeting.asm
get_user_choice.asm
encrypt_and_print.asm
decrypt_and_print.asm
```

You need to submit all of these to be graded on all of the tasks. Partial marks are possible if some tasks are incomplete.

Marking

The total is 10 marks, with a breakdown as follows:

- Task 1: **1 mark**
- Task 2: **1 mark**
- Task 3: **2 marks**
- Task 4: **2 marks**
- Task 5: **4 marks**

Function Prototypes

This is a high-level understanding of the functions you will be writing.

```
void greeting();
int get_user_choice();
void encrypt_and_print(const char* input);
void decrypt_and_print(int num1, int num2, int num3, int
    ↪ num4);
```

Debugging

Please learn how to use GDB to debug this assignment. This is trivial compared to the rest of the semester. An example to get you started:

```
section .data
message db "Hello, Agent"
secret db "This is a secret message"

section .text
mov rax, message
mov rdi, [message]
mov rsi, 34
    ; break here
```

Now, assuming we have a breakpoint, let's see what different expressions evaluate to:

```
$rax: 4210836
$rsi: 34
(char)message: 'H'
(char)$rdi: 'H'
*(char*)$rax: 'H'
*(char*)&message@12: 'H', 'e', 'l', 'l', 'o', ',', ' ', 'A', 'g', 'e', 'n', 't'
*(char*)$rax@12: 'H', 'e', 'l', 'l', 'o', ',', ' ', 'A', 'g', 'e', 'n', 't'
```

Keep this in mind when working with pointers. Hopefully, you understand why the following happens:

```
*(char*)$rax@30: 'H', 'e', 'l', 'l', 'o', ',', ' ', 'A', 'g', 'e', 'n', 't',
'T', 'h', 'i', 's', ' ', 'i', 's', ' ', 'a', ' ', 's', 'e', 'c', 'r', 'e', 't',
' ', 'm', 'e', 's', 's', 'a', 'g', 'e'
```

If you do not understand why that happens, review the data storage section in the textbook before attempting this assignment.

Advice

You will be splitting your logic into separate functions. Do not assume that a function will respect the values in your registers when calling them. There are only 16 general-purpose registers. Although there is a calling convention to be respected, you do not know about those specifics yet. For this assignment, always assume any value in a register will be overwritten by a function call.