COS284 Assignment 1

University of Pretoria
Department of Computer Science

Online Publication: 20 July 2023

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 together.
- If you use any advanced material, you must cite these sources.

Mission Brief: Secure the Bank

Greetings agents,
We have a mission of utmost importance for you. A new banking system needs to be built, and we trust only you with this responsibility. The architecture? YASM x86 64-bit assembler. You'll need to rely on your skills in user input and output, mathematical operations, and bitwise operations. The specifics are detailed below.

Mission Objectives

1. Prompt the user for information
2. Take input from the user
3. Convert between data formats
4. Perform calculations to determine account number and balance
5. Develop an obfuscation protocol

Mission Details

On a high level the experience should look as follows:

```
Welcome to the Bank of <<Redacted>>
Enter 4-digit PIN: 1234
Your account number is:
11234
Your balance is:
50800
Your obscured PIN is:
;<=>
```

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 only receives one form of user input - the pin on line 2. Everything else is generated by the program through hard-coded messages or calculations. The program should be able to perform the following actions:

Task 1: Greeting

The first requirement is to output a welcome message to the user. The welcome message should look as follows:

```
Welcome to the Bank of <<Redacted>>
```

When outputting to the console, make sure you are careful with exactly what you output. The welcome message should be exactly as above, with no extra spaces or characters. Your first attempt might look something like this:

```
Welcome to the Bank of <<Redacted>>[?]
```

```
Welcome to the Bank of <<Redacted>>[%]
```

or if you really messed up, you might see something like

```
[?][?][?][?][?]
```

All of these mean the pointer you have passed in to the system call to write was either populated incorrectly or the size passed in was incorrect. If you are unable to do this rudimentary task, the rest of the mission will be impossible. You should then rendezvous with the bureau's intelligence team (tutors) for assistance. You will write all the necessary code for this greeting inside a file called **greeting.asm**. A partial skeleton has been provided for you. You will need to fill in the blanks.

Task 2: Get PIN

Your next task is to prompt the user for a pin.

```
Enter 4-digit PIN:

Enter 4-digit PIN: 1234
```

This will incur additional challenges as the input will be provided as an ASCII string. You will need to convert this string to a 32-bit integer. You will write all the necessary code for this inside a file called **get_pin.asm**. A partial skeleton has been provided for you. You will need to fill in the blanks.

- **NB**: The pin needs to be stored in the register **rax** at the end of the function. This will be how the value is returned to the caller.

Task 3: Calculate Account Number

The third step requires you to calculate the user's account number. The account number is calculated as follows:

```
uint32_t calculate_account(uint32_t pin) {
  return pin + 10000;
}
```

This means you will need to use the integer value of the pin and add 10000. You will write all the necessary code for this inside a file called **calculate_account.asm**. A partial skeleton has been provided for you. You will need to fill in the blanks.

There is an output portion for the account number, however that requires converting the number to ASCII so we will first discuss the rest of the "calculations" before that. The function itself does not handle the conversion or output.

- **NB**: The account number needs to be stored in the register **rax** at the end of the function. This will be how the value is returned to the caller.

Task 4: Calculate Balance

The fourth step requires you to calculate the user's balance: You must use the following mathematical formula to calculate the balance:

Let A be the account number
and P be the PIN number. Then, the balance (B) can be calculated as follows:

$$B' = ((A + P) \times P) \wedge (P \oplus A)$$
$$B = (B' \mod 50000) + 50000$$

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

- **NB**: The balance needs to be stored in the register **rax** at the end of the function. This will be how the value is returned to the caller.

Task 5: Obfuscate PIN

We cannot store the PIN in plain text. We need to obfuscate it. The following is the obfuscation protocol:
Let's assume the original 4-digit pin as $d_1 d_2 d_3 d_4$ where each d_i represents a digit of the pin.

The `obscure_pin` function can then be defined as:

$$\texttt{obscure\_pin}(d_1 d_2 d_3 d_4) = d'_4 d'_3 d'_2 d'_1$$

where,

$$d'_i = ((d_i - 48) \oplus 0xF) + 48$$

And d'_i represents the obscured version of digit d_i. Here, $\oplus$ represents the bitwise XOR operation, 48 is the ASCII value of the digit 0, and $0xF$ is the hexadecimal representation of the decimal number 15. The operation $d_i - 48$ is used to convert the ASCII representation of d_i to its numerical value, and the operation $d'_i + 48$ is used to convert the numerical value of d'_i back to its ASCII representation.
With a function prototype as follows:

```
void obscure_pin(char* pin)
```

- **NB**: This function mutates the existing ASCII string pointer. It does not return a new string.

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

Task 6: Create Account

Up until now, you have only been building the cogs of the machine. By this point you should have a function to do each of the following:

- Welcome the user
- Get the PIN
- Calculate the account number
- Calculate the balance
- Obscure the PIN

Now you will actually make use of those values, convert them to ASCII where necessary and output them as previously shown. You will write all the necessary code for this inside a file called **create_account.asm**. A partial skeleton has been provided for you. You will need to fill in the blanks.

Assumptions

Our sources have informed us that you may make the following assumptions about this mission:

- **The user of your program is not an idiot**

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), however they will be graded individually. Only the final task (Task 6) relies on all the other tasks. Thus you will be graded on the following files:

```
greeting.asm
get_pin.asm
calculate_account.asm
calculate_balance.asm
obscure_pin.asm
create_account.asm
```

You need to submit all of these, to be graded on all of the tasks. For example, if you only submit:

```
greeting.asm
calculate_account.asm
```

You will be graded on Task 1 and Task 3. You will receive a 0 for the other tasks. Thus it is possible to receive partial marks if you struggle with a specific task.

Marking

The total is 20 marks with a break down as follows:

- Task 1: **1 mark**
- Task 2: **4 marks**
- Task 3: **1 mark**
- Task 4: **4 marks**
- Task 5: **5 marks**
- Task 6: **5 marks**

Additional information

This practical will provide backbone and structure where necessary. You have not officially been taught how conditional statements, loop structures, and functions work. Thus for this first practical that will all be provided. You will not be so fortunate in the future.

Function Prototypes

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

```
void greeting();
uint32_t get_pin();
uint32_t calculate_account(uint32_t pin);
uint32_t calculate_balance(uint32_t acc, uint32_t pin);
void obscure_pin(char* pin);
void create_account(char *acc, char *pin, char *bal);
```

Debugging

Please learn how to use GDB to debug this assignment. This is trivial in comparison to the rest of the semester. If you think tracing an integer through the program is difficult, wait until you need to work with data structures from COS212. A simple example to get you started:

```
section .data
    message db "Hello, world"
    useless db "This is a useless string"

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

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

```
$rax: 4210836
$rsi: 34
(char)message: 'H'
(char)$rdi: 'H'
*(char*)$rax: 'H'
*(char*)&message@12: 'H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd'
*(char*)$rax@12: 'H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd'
```

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

```
*(char*)$rax@30: 'H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd',
'T', 'h', 'i', 's', ' ', 'i', 's', ' ', 'a', ' ', 'u', 's', 'e', 'l', 'e', 's',
 's', ' ',
```

If you do not understand why that happens, I recommend you go through 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.