Exercise 01

x86 Assembly

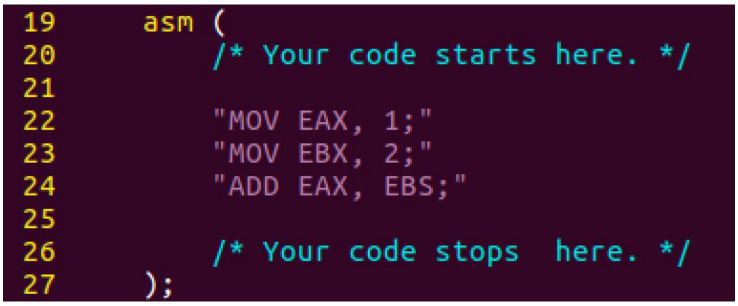
Log into your VM (`user` / `1234`), open a terminal and type in `infosec pull 1`.

- When prompted, enter your username and password
- Once the command completes, your exercise should be ready at `/home/user/1/`

Question 1 (30 pt)

In the **`q1.c`** file, write an x86 Assembly program that receives an integer in `EBX`, computes its **greatest prime factor**, and stores the result in `EAX`; if the integer is less than or equal to `1`, the result should be `0`.

Add your assembly instructions as strings to q1.c between our comments, like so:



```
    asm (
        /* Your code starts here. */

        "MOV EAX, 1;"
        "MOV EBX, 2;"
        "ADD EAX, EBS;"

        /* Your code stops  here. */
    );
```

- To compile your program, run[1] `gcc q1.c -masm=intel -o q1`.
- If compilation succeeds without errors, it will create a program named `q1` within the same directory
- Test your code as follows:
  - To run it, just run `./q1 <number>`
  - For example, `./q1 10` should print `5`, and `./q1 -5` should print `0`

Question 2

Part A (30 pt)

In the **`q2a.c`** file, write an x86 Assembly program that receives an integer in `EBX`, **computes its Fibonacci number <u>using recursion</u>**, and stores it in `EAX`; if the integer is less than 0, the result should be 0.

[1] `gcc` = the compiler, `q1.c` is our input file, `-masm=intel` means we use the intel x86 syntax, `-o q1` means to write the result as `q1`.

Fibonacci numbers are the numbers of the sequence 0, 1, 1, 2, 3, 5, 8… defined as:

$$a_0 = 0\,, \qquad a_1 = 1\,, \qquad a_n = a_{n-1} + a_{n-2}$$

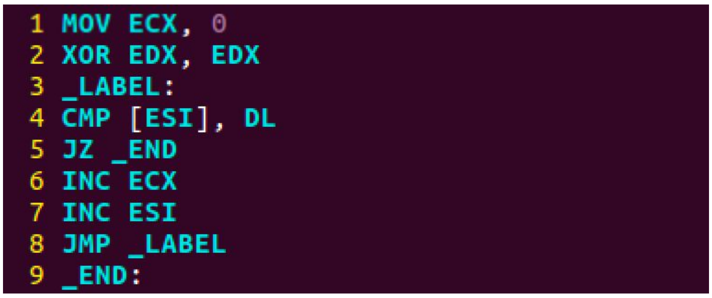
Add your assembly instructions as strings as in question 1, and compile and test in a similar way.

Part B (20 pt)

In the `q2b.c` file, as before write an x86 Assembly program to compute a Fibonacci number, **this time <u>without recursion</u>**.

Question 3 (20 pt)

Read the following x86 Assembly program, and describe what it does in `q3.txt`.



```
MOV ECX, 0
XOR EDX, EDX
_LABEL:
CMP [ESI], DL
JZ _END
INC ECX
INC ESI
JMP _LABEL
_END:
```

Note: Telling us what every line does, is NOT a valid answer. We want to **the key idea of what this code does**, not a translation from Assembly to English.

Final notes:

1. Consider edge cases (i.e. negative numbers, etc.)
2. **Document your code**
    a. You can use C comments to add documentation between the strings of the Assembly
3. Don't use any additional third party libraries that aren't already installed on your machine (i.e. don't install anything)
4. If your answer takes an entire page, you probably misunderstood the question.