Exercise 01

x86 Assembly / `ex-x86`

Welcome to the first exercise! To be able to run this exercise, and all following exercises, you will need to follow both these documents:

- Setup instructions - Installing the VM and registering on the course website. You will need this to run all exercises.
- Exercise Submission Guidelines - General instructions and course policies that apply to all exercises in the course. You **must** read this document before submitting any exercise.
- In case you need help debugging, copying files to/from the VM, etc., see our official How do I do <whatever>? Document
    - You don't need to read this now, but if you are stuck on anything try checking this out

Ready? Awesome, let's actually begin :)

- Start your VM
    - This might take 3-4 minutes where the screen might seem frozen
    - This is OK, it should start in a few minutes
- Log into your VM if prompted (`user` / `1234`)
- Open a terminal and type in `infosec pull ex-x86`.
    - When prompted, enter your **course website credentials**
    - I.e. the username and password you used to register to the course website (these are unrelated to your university user)
- Once the command completes, your exercise should be ready at `/home/user/ex-x86`

When you finish solving the assignment, submit your exercise with `infosec push ex-x86`.

- This will run some sanity tests to make sure your submissions seems to be OK (it is not a full test of the homework)
- It will submit the homework even if the tests fail
- The last submission is what that matters
    - You can see your submitted files on the course website

Question 1 (30 pt)

In the `q1.s` file, write an x86 Assembly function that receives an integer, computes its exact square root and returns it; if the integer is less than `1`, or if there is no exact (integer) root, the result should be `0`.

Details

- Write your code where it says `<<<< PUT YOUR CODE HERE >>>>`.
- For your convenience, we already read the user input into `EBX`.
- To return the result from a function, store it in `EAX` right before the code returns.

Running your code

- To compile your program, from within the exercise directory, run `make q1`.
  - This will link your assembly function into a C program that calls it (the C program is defined in `skeleton.c`).
- If compilation succeeds without errors, it will create a program named `q1` within the same directory
- To test your code, just run it - `./q1 <number>`
  - For example, `./q1 16` should print `4`, and `./q1 6` should print `0`

Question 2

Part A (30 pt)

In the `q2a.s` file, write an x86 Assembly function that receives an integer, computes its **Squarebonacci number** **using recursion**, and returns it; if the integer is less than `0`, the result should be `0`.

In a similar fashion to Fibonacci numbers, **Square**bonacci numbers are the numbers of the sequence 0, 1, 1, 2, 5, 29, … defined as:

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

Details

Add your assembly code as in question 1, and compile and test in a similar way (`make q2a` and `./q2a <number>`).

Important Note: Due to automatic testing constraints, please **avoid** using tail-recursion, and make sure each parent call invokes at least 2 recursive calls. You can read more about tail recursion in this Wikipedia explanation.

Part B (20 pt)

In the `q2b.s` file, as before, write an x86 Assembly program to compute a Squarebonacci number, **this time without recursion**.

Question 3 (20 pt)

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

```
1        XOR    EDX, EDX
2   _LABEL1:
3        CMP    [EDI], DL
4        JZ     _LABEL2
5        INC    EDI
6        JMP    _LABEL1
7
8   _LABEL2:
9        MOV    AL, [ESI]
10       MOV    [EDI], AL
11       INC    ESI
12       INC    EDI
13       CMP    AL, DL
14       JNZ    _LABEL2
15
16  _END:
```

Please note that the program receives input via 2 registers: `EDI` and `ESI`.

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

Final notes

- Consider edge cases (i.e. negative numbers, etc.)
- **Document your code**
    - You can prefix lines in assembly files with `#` to mark the rest of the line as a comment
- If your answer takes an entire page, you probably misunderstood the question