Assignment 2: A Calculator in Assembly

Due: Friday, January 24, 10pm

Language: x86_64 Assembly, AT&T syntax

Starter code: See Assignment 2 on Canvas for the Github Classroom link.

Submission: Submit the contents of your repository via Gradescope. See Deliverables below for what to submit.

This is an *individual* assignment.

This assignment asks you to implement a simple terminal calculator in assembly.

You may work on Github Codespaces, or use our Docker container. You can also work on Khoury Login. We will assume that the submitted code was tested on one of these platforms.

Task

Your task is to complete the `main` function (under the label `main`) in `calculator.s` so that it implements the following functionality:

1. The calculator can be invoked on the command line as follows:

   ```
   $ ./calculator
   ```

2. After launching, the calculator will accept a line of input in the following format:

   ```
   $ <integer1> <operation> <integer2>
   ```

   The four valid operations are:

   - `+`: add
   - `-`: subtract
   - `*`: multiply
   - `/`: divide

3. You may assume that the line will always contain a valid integer, a single character, followed by a valid integer.

4. When the input consists of 2 valid *signed long* integers with a valid operation in between, the result of the corresponding operation should be printed to the terminal. An error message should be printed if the operation cannot be performed, that is, when it would result in an operating system exception. *There should be only one line of output and that line should only contain a single long integer or an error message.*

   A sample interaction should be as follows:

```
$ ./calculator
50 - 51
-1
```

5. If the operation is invalid (i.e., not one of `+`, `-`, `*`, or `/`), the error message should be `Unknown operation`. Other error messages are up to you. You do not have to handle overflow, but you should handle division by zero.
6. The main function shall return `0` on success and `1` if there was an error.

Here are a few more examples of running your program:

```
$ ./calculator
12 + 43
55
$ ./calculator
33*78
2574
$ ./calculator
3 / 11
0
$ ./calculator
57 ^ 3
Unknown operation
```

Using `scanf`

The most convenient way to read input for our simple example, is to use the `scanf` function from the C library. Follow the class example posted on Piazza. The starter code already contains labelled "slots" for the two integers and a character. You just need to pass them to `scanf` and use their contents after it returns. Make sure you understand when to put `$` in front of a label.

We recommend experimenting with `scanf` by writing a very simple C program.

Do not be afraid to ask questions.

Code Layout

When writing assembly, follow these simple rules:

1. Labels, `.global`, `.text`, and `.data` are aligned on the left edge.
2. Instructions and data definition directives are indented by 2, 4, or 8 spaces (pick one and be consistent)
3. Think of groups of instructions as logically corresponding to an individual higher level operations, e.g., calling a function entails loading argument registers and the `call` instruction itself. Separate logical groups by empty lines. Add comments to indicate your intent (e.g., "Add left-hand side and right-hand site, storing the result in rax").

Using the Makefile

We have provided a very basic Makefile for you. You can compile your program simply by running `make` on the command line. Running `make clean` should help you prepare your working directory for committing to git and for submission. Read the comments in the makefile and look at this tutorial if you have questions: https://makefiletutorial.com/.

Deliverables

Submitting machine generated code for this assignment automatically results in 0 points and may be considered cheating. This includes, but is not limited to, using a C compiler to translate C into assembly.

- Modify the file `calculator.s` and commit it to your repository.
- Do not include any executables, object files, or any other binary, intermediate or hidden files. That means, e.g., no `.o` files, no files or directories starting with `.` (like `.git`), etc.
- Finally, download a ZIP archive of your repository (available on Github) and submit it to our Gradescope.

Hints

- Make sure you understand the provided starter code, particularly where to store the elements of the input line and how to use them.
- When using `scanf`, each part of the input line will be stored to a memory location, according to the format passed as the first argument. We have prepared labelled locations in the starter code for this purpose.
- It might be useful to know how specify character literals in assembly: https://sourceware.org/binutils/docs/as/Constants.html
- Pay close attention to the calling conventions and the use of registers when calling C functions.
- The instruction for division is used differently from the other arithmetic instructions. You will need to do some research and reading. The materials linked from our website are a good start.
- Use examples from the lectures and the lab to help you get unstuck and ask questions.