Assignment 2: A Calculator in Assembly

Due: Friday, January 27, 10pm

Starter code: See Assignment 2 on Canvas for the Github 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 command line calculator in assembly. The provided starter code retrieves the command line arguments, you have to fill in the rest of the code.

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 <operation> <number1> <number2>
   ```

2. The four valid operations are:

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

3. You may assume that the first argument is always a single character (which might or might not be a valid operation) and that the 2nd and 3rd argument are valid long integers.

4. When the program is invoked with a valid operation and 2 valid *signed long* integers, 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 of the output from your program should look something like the following.

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

   To make it a little clearer:

```
$ ./calculator - 50 51
-1
 ^      ^      ^ ^  ^--- long2
 |      |      | +------ long1
 |      |      +-------- subtract '-'
 |      +--------------- name of executable
 +---------------------- long1 - long2
```

5. If the operation is not valid (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.

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
```

Note that you need to quote the asterisk `"*"` on the command line to prevent the shell from interpreting it as a special character (https://tldp.org/LDP/GNU-Linux-Tools-Summary/html/x11655.htm).

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. If you want to know more about Makefiles (and you should!), try this tutorial: https://makefiletutorial.com/.

Deliverables

> Submitting machine generated code for this assignment automatically results in 0 points and may be considered cheating. This includes, for example, 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, grab a ZIP archive of your repository (which can be downloaded from Github) and submit it to our Gradescope.

Hints

- Make sure you understand the provided starter code, in particular where we store the arguments for you.
- Each argument is a *string*, that is the value is actually a memory address pointing to the first character of the string in memory.
- You will need to use the `atol` C function to convert the two strings into the corresponding long integers. You may need to call other C functions as well. Use `man 3 atol`.
- 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.
- Use examples from the lectures and the lab to help you get unstuck and ask questions.

Changes

01/23/2023

- Added a warning about machine generated code.