Assignment 4 — Subroutines

Due: Monday, February 7th

While monolithic, straight-line code can theoretically be used to express any computation, code that incorporates some form of procedural organization is far easier to read, debug, and maintain. Subroutines allow common code to be grouped into callable units, so that it can easily be reused later.

Deliverables:

GitHub Classroom:

Required Files: `stack.asm`, `utils.asm`, `calculator.asm`

Optional Files: *none*

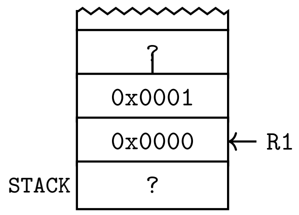
Part 1: A Stack Implementation

Recall that stacks can be implemented in assembly by allocating a contiguous block of memory and maintaining the *stack pointer*, the address of the top-of-stack, in a register.

Implement an assembly stack, growing from high addresses to low addresses, by completing the subroutines in `stack.asm`. The following code:

```
LEA R1, STACK
AND R2, R2, #0
JSR PUSH
ADD R2, R2, #1
JSR PUSH
JSR POP
```

...should place `0x0001` into `R2` while producing the following in memory:



Your subroutines must follow the LC-3 assembly conventions presented in lecture for saving and restoring any registers that they modify.

Part 2: Integer Input and Output

The input and output macros provided in LC-3 assembly are only capable of reading and writing textual data. Complete the subroutines in `utils.asm` to support reading and writing integers:

- Your subroutines must support all integers in the range $\{-9, \ldots, 9\}$. Thus, you may assume that only single digits will be required, but you must differentiate between positive and negative integers.
- You may make use of the existing `GETC` and `OUT` macros to read and write characters. You do *not* need to implement input and output from scratch.

Furthermore, your subroutines must follow the LC-3 assembly conventions presented in lecture for saving and restoring any registers that they modify. Note that these subroutines, while not necessarily long, will likely be considerably longer than those in `stack.asm`.

Part 3: Stack-Based Arithmetic

Consider the assembly program in `calculator.asm`: this file contains the skeleton of a program to perform stack-based calculations[1], supporting the following user commands:

- `#`*`N`* — Push *N* onto the stack, where *N* is an integer in the range $\{-9, \ldots, 9\}$.
- `a` — Pop two integers off of the stack, **a**dd them together, and push their sum.
- `s` — Pop two integers[2] off of the stack, **s**ubtract one from the other, and push their difference.
- `n` — Pop one integer off of the stack, **n**egate it, and push its negation.
- `q` — Print the integer on top of the stack, then **q**uit.

All of these commands will be followed by the user's pressing "Enter". The majority of the code for parsing user commands has been given. Additionally, code has been included so that the subroutines in `stack.asm` and `utils.asm` can be called as though they were written in `calculator.asm`[3].

Complete the five indicated cases in `calculator.asm`, making use of your previously implemented subroutines:

- You may *not* change any of the given code, however, you may add additional subroutines to the end of `calculator.asm`, if desired.
- All of your code must follow the LC-3 conventions presented in lecture for saving and restoring any registers modified by subroutines.
- It must be possible to rerun your program by manually resetting the PC to `0x3000`. It should not require that the LC-3 be reinitialized or that any files be reloaded.

You may assume that all user input will be valid, structured such that an integer outside the range $\{-9, \ldots, 9\}$ will never need to be printed, and that the size of the stack will never exceed 16 elements.

For example:

```
Enter a command (#N/a/s/n/q): #1
Enter a command (#N/a/s/n/q): #-2
Enter a command (#N/a/s/n/q): s
Enter a command (#N/a/s/n/q): q
Value on top of the stack: 3
```

Your program will be tested using `diff`, so its printed output must match *exactly.*

Part 4: Submission

The following files are required and must be pushed to your GitHub Classroom repository by the deadline:

- `stack.asm` — Working implementations of assembly subroutines for stacks, as specified.
- `utils.asm` — Working implementations of assembly subroutines for integer I/O, as specified.
- `calculator.asm` — A working assembly program for stack-based arithmetic, as specified.

The following files are optional:

- *none*

Any files other than these will be ignored.

[1] Expressions written in this form are said to be in *postfix notation.*
[2] The first integer popped is the subtrahend; the second, the minuend.
[3] Note that the LC-3 simulator allows multiple assembly files to be loaded, but each file sets the PC accordingly on load. Thus, in order to run this program, `calculator.asm` should be the final file assembled and loaded into the simulator.