CS4430 HOMEWORK 2 (20 POINTS TOTAL)

Issued: Monday, February 12th, 2018.
Due: Monday, February 19th, 2018 by 11:59pm.

DIRECTIONS

To turn in your solution, please email me the code (harrisonwl@missouri.edu) with the subject "CS4430 HW2". It is **important** that you get this small detail right because otherwise I may miss your submitted solution.

PROBLEM DESCRIPTION

In class, we've been discussing "return-oriented programming" or ROP. As a simple example, consider the following ThreeAddr code:

```
0:    mov R0 #0;
      mov Rx R0;
      mov R1 #9;
      write Rx;
1:    mov Rx R1;
      mov R2 #8;
      mov Rx R2;
      write Rx;
2:    exit;
```

This code, in ROP-style, is:

```
0:    pushl 14;  // lines 0-3 are the "preamble"   //
1:    pushl 9;   // only use pushl in the preamble //
2:    pushl 4;
3:    ret;
4:    mov R0 #0;
5:    mov Rx R0;
6:    mov R1 #9;
7:    write Rx;
8:    ret;
9:    mov Rx R1;
10:   mov R2 #8;
11:   mov Rx R2;
12:   write Rx;
13:   ret;
14:   exit;
```

PROBLEM ASSIGNMENT

Your job is to translate by-hand the example `foobar.tac` into ROP style. Put your translated version in a file called `rop-foobar.tac`. Note that the code in `foobar.tac` includes a number of write instructions that print out a numbers throughout the program's execution. Use the function `interp` defined in `ThreeAddrInterp.hs` to make sure that `foobar.tac` and `rop-foobar.tac` produce the same output.

Specific directions:

1. Download and unpack "ROP.zip" from the course website. Within the ROP directory is `ThreeAddressInterp.hs` and a subdirectory called `tac`. Within tac are a number of examples that can help you, including:

    - `branchzero1.tac`, `branchzero2.tac`, `jump.tac`, `loop.tac`, and their ROP translations (prefixed with "`rop-`"). These examples illustrate how to translate jump and branch instructions into operations on the `SP` register.

    - The file to be translated, `foobar.tac`, is also in the `tac` subdirectory. Rewrite this program into one that (1) produces the same output and (2) does not include a single jump or branch instruction.

2. For the purposes of this exercise, a gadget is a sequence of non-control flow instructions ending in either a `ret` or an `exit` instruction. The definition of "control-flow instruction" is subtly different from the one we used in defining basic blocks in ThreeAddr. Specifically, labels are not necessarily control-flow instructions. Your answer must be written entirely in terms of gadgets! It is **not** sufficient that the program you turn in merely has the same output and no jumps or branches. Once you think you have a solution, you should come back to this paragraph and make sure that you have properly gadget-ified your program.