Some Assembly Required

CMS 230, Fall 2017

Due Thursday, November 30, 11:59:59 PM

Description

Translate the following C programs into ARM assembly language. This project will give you practice writing assembly language programs with function calls.

I'll grade your programs by downloading them and running them on the Raspberry Pi. Each program has one obvious correct output that it should print by calling `printf`. There are 4 problems worth 25 points each.

Name your problems `problem1.c`, `problem2.c`, and so forth. Include a `Makefile` to build your programs from source. A failure to successfully build with `make` is a 10-point deduction. We don't have precise style guidelines for ARM programs like we do for C and Java, but your program should be well-formatted in a way that shows its logical structure. I reserve the right to deduct points from programs that have incoherent or impossible-to-understand formatting.

Use a recursive function if the problem asks for one. You will be ***HEAVILY PENALIZED*** if you use a loop for a recursive problem.

Make sure to include descriptive comments in your programs and make notes to your self as you're developing—assembly language programming is really hard if you don't keep track of what's going on at each point in the program.

Upload your work to GitHub by the due date. Note that this requires either having your Raspberry Pi connected to a network, or if that isn't possible, copying the text of your files to your main computer or Cloud 9 and uploading it from there.

Tips

Remember the calling convention for ARM programs! In particular, remember that the called function (the *callee*) is allowed to modify `r0` to `r3` freely. This is particularly important when calling `printf`: you can never assume the values in `r0-r3` after a call to `printf` are the same as they were before the call! The

best thing to do is save the registers on the stack before the call and restore them afterwards:

```
push {r0-r3}  // save registers on stack

// Load arguments and call printf

pop {r0-r3}  // restore registers saved by push
```

The Problems

FizzBuzz II: Electric FizzBuzzoogaloo

Solve the FizzBuzz problem in ARM:

- Loop through the numbers from 1 to 100
- If a number is divisible by 3, print `Fizz`; if it's divisible by 5, print `Buzz`; if it's divisible by 3 and 5, print `FizzBuzz`; if it's not divisible by 3 or 5, print the number.

You'll probably want to write a function that implements the mod operation using repeated subtraction in a loop, as in the following C code:

```
int mod(int a, int b) {
    int rem = a;
    while ( rem > b) {
        rem -= b;
    }
    return rem;
}
```

We Are All Made of Stars

Write an ARM program that prints a slope of stars, like the following. Your program must use a variable and loops to control the height of the slope. You don't have to write any functions for this problem, but using one might help you organize your code.

```
*
**
***
****
*****
```

Hobbies, Again

I enjoy building stone ziggurats in my backyard. To build an N-level ziggurat, I first build an $N \times N$ square of stones on the ground. Then I build an $N-1 \times N-1$ square of stones for the second level, then an $N-2 \times N-2$ square of stones for the third level, and so forth, until I finally place a single stone on the top level.

Write a ***recursive*** ARM assembly program that calculates the number of stones in a ten-level ziggurat.

Hint: The number of stones in a ten-level ziggurat is the number in a nine-level ziggurat plus 10^2. In general,

$$stones(N) = stones(N-1) + N^2$$

Oh yeah, how are you going to calculate N^2?