Some Assembly Required

CMS 230, Fall 2017

Due Thursday, November 16, 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 local and global variables, conditional statements, and loops.

I'll grade your programs by downloading them and running them on the Raspberry Pi. Each program has one obvious correct output that should print when you use `echo $?` after running the program. There are five problems and each one counts for 18 points.

Name your problems `problem1.c`, `problem2.c`, and so forth. Include a `Makefile` to build your programs from source. A working build using `make` counts for 10 points. 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.

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.

The Problems

Basic

```
// Add up three global variables

// Global variable
int a = 10;
```

```
int b = 20;
int c = 30;

void main() {
    // Use registers for local variables
    int sum = a + b + c;
    return sum;
}
```

Triple Max

```
int main() {
    // Use registers for local variables
    int x = 10;
    int y = 5;
    int z = 20;

    int  max = x;

    if (y > max) {
        max = y;
    }

    if (z > max) {
        max = z;
    }

    return max;
}
```

Mod

```
// Return remainder after integer division

// Global variables
int a = 4;
int b = 30;

int main() {
    // Use registers for local variables
    int remainder;
    int divisor;
```

```
    if (a > b) {
        remainder = a;
        divisor = b;
    } else {
        remainder = b;
        divisor = a;
    }

    while (remainder >= divisor) {
        remainder -= divisor;
    }

    return remainder;
}
```

Fibonacci in a Loop

```
// Calculate the 10th Fibonacci number in a loop

// Global variables
int fib = 1;
int prev = 0;

int main() {
    // Use registers for local variables
    int n = 1;
    int sum = 0;

    while (n <= 10) {
        sum = fib + prev;
        prev = fib;
        fib = sum;
        n++;
    }

    return fib;
}
```

The Euclidean Algorithm

The greatest common divisor of two numbers a and b, written $gcd(a, b)$, is the largest integer that evenly divides both.

The Euclidean algorithm, named after the ancient Greek mathematician, finds the gcd by exploiting the fact that

$$gcd(a, b) = gcd(a - b, b)$$

where a is the larger of the two values. This is not obvious, but arises from the fact that any number that divides a and b must also divide $a - b$. Let d be a common divisor of both a and b; it must be possible to write

$$a = dm$$
$$b = dn$$

for some numbers m and n. Therefore,

$$a - b = d(m - n)$$

and d is also a divisor of $a - b$.

Therefore, you can find the gcd of two integers by repeatedly subtracting the smaller from the larger until the two values are equal, as in the code below. Try some example values and verify that it works. Note that if $gcd(a, b) = 1$ the numbers are coprime: they have no common divisors.

```
// Return gcd of a and b

int a = 60;
int b = 25;

int main() {
    while (a != b) {
        if (a > b) {
            a = a - b;
        } else {
            b = b - a;
        }
    }
    return a;
}
```