Some Assembly Required

CMS 230, Fall 2019

Due Thursday, November 14, 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. However, you should test other values for the variables and make sure your program is correct and works for more than just the given values. I will be testing your programs with other variable values. There are five problems and each one counts for 16 points.

Name your problems `problem1.s`, `problem2.s`, and so forth. Modify the `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. Include indentation for statements which are not labels. 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 yourself 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.

Remember to complete the `Readme.md` file with your submission.

Setup

Begin by accepting the assignment via the link on Canvas. This will setup a repo with the starter code for you on GitHub.

Copy the link for your newly created repo (big green button on GitHub).

Boot your Raspberry Pi, login, and check that it is connected to the network.

At the command line prompt on your Pi, issue the command:

```
git clone paste-link-to-your-repo-here
```

Work on the following problems in the directory created by the `git` command. Commit your code and push it to Github.com with the following sequence of commands:

```
git add .
git commit -m "commit message"
git push origin master
```

Note the message in the 2nd command should be descriptive rather than the generic message "commit message". Something like "finished problem 4" or "fixed bugs in problem 1" would be more appropriate. Note that you are one dropped/lost Raspberry Pi away from disaster as there are no cloud backups for your Pi. Thus, I recommend committing/pushing frequently as this transfers your work to the cloud and reduces the probability of disaster.

Push your work to GitHub by the due date. Note that this requires having your Raspberry Pi connected to a network so take that into account as you plan your time. I grade the last thing you submit.

The Problems

Problem 1: Triple Max

```
int main() {
    // Use registers for local variables -- we'll
    // learn how to use the stack for local vars later
    int x = 10;
    int y = 5;
    int z = 20;

    int  max = y;

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

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

    return max;
}
```

Problem 2: 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;
}
```

Problem 3: Pythagorean Theorem

A mathematical oldie but a goodie is always the Pythagorean Theorem: $a^2 + b^2 = c^2$. Return the value of c^2. *Hint: Look at the "multiplication by repeated addition" algorithm and code from class.*

Problem 4: 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;
}
```

Problem 5: 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. For example, $gcd(15, 24)$ is 3 and $gcd(12, 18)$ is 6.

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 $a - b$.

Practically, 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 = 40;
int b = 15;

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

Tips and Hints

1. Remember that you can easily copy files to create a new file. If you have finished `problem3.s` and want to use that file as a basis for problem 4, you can issue the command `cp problem3.s problem4.s`. This will create a copy of `problem3.s` named `problem4.s`. You can then edit `problem4.s` using `nano` as usual without effecting `problem3.s`.

2. You may find it useful to open 2 windows (via either PuTTy or Terminal) to your RaspberryPi. This allows you to open `nano` in one and leave it open. In the other, you can issue commands (like `gcc`, `make`, run your programs). as needed. This process was demonstrated in class as a way to minimize the need to open/close `nano` frequently.

3. Pay attention to your program flow. The most frequent mistake on this assignment is forgetting an unconditional branch in a loop or an if-statement.

4. `nano` does not display line numbers by default. However, if you receive an error from `gcc`, it will report the line number, and line numbers in general are useful in debugging. You can jump directly to a line of code in `nano` by typing "`^_`". This means to hold the Control key while pressing the underscore key. You can then type in a specific line number, press Enter and `nano` will jump to that line. You can also start `nano` with the command `nano -c problemX.s`. The `-c` flag will cause nano to display a status bar across the bottom which shows what line number the cursor is currently on (as well as a lot of other information).

5. Remember that you can always write, compile, and run these programs in a high-level language to check your answers.