CS 352 Homework
Armoir

1 Overview

Your responsibility in this homework is to learn about assembly programming using the ARM architecture, low-level memory manipulation, bit twiddling, and branching. You will do this in the context of writing several utilities functions (an *armoir* of them) and one executable program.

Compiling for ARM requires either that you compile on an ARM machine or use a *cross-compiler* when on a different kind of machine. Likewise, running an ARM executable requires an ARM processor, either physical or virtual. Cross-compilers and emulators are freely available on Ubuntu, but reliable and free tools are harder to find for other operating systems. Therefore, the grading script for this homework assumes you are running on Ubuntu. Install the cross-compiler and the Qemu emulator at the command-line:

```
sudo apt-get install gcc-arm-linux-gnueabihf qemu
```

Compile an assembly file with this command:

```
arm-linux-gnueabi-gcc -static myassembly.s
```

Execute an application compiled for ARM with this command:

```
qemu-arm myexecutable
```

2 Requirements

Implement the following functions in the specified files using ARM assembly. Each should compile should with `gcc`.

- Write function `inbox` in file `inbox.s`. It accepts six `int` parameters, in the following order:
  - a rectangle's left x-coordinate
  - a rectangle's bottom y-coordinate
  - a rectangle's right x-coordinate
  - a rectangle's top y-coordinate
  - a point's x-coordinate
  - a point's y-coordinate

  If the point is inclusively within the rectangle's bounds, the function returns 1. Otherwise it returns 0.

- Write function `pack` in file `color.s`. It accepts three `int` parameters, in the following order:
  - a red intensity in [0, 255]

  - a green intensity in [0, 255]
  - a blue intensity in [0, 255]

  It returns a single `int` with this byte layout:

byte 3	byte 2	byte 1	byte 0
0	red	green	blue

- Write function `red` in file `color.s`. It accepts one `int` parameter in the form returned by `pack`. It returns the red intensity in [0, 255] as an `int`.
- Write function `green` in file `color.s`. It accepts one `int` parameter in the form returned by `pack`. It returns the green intensity in [0, 255] as an `int`.
- Write function `blue` in file `color.s`. It accepts one `int` parameter in the form returned by `pack`. It returns the blue intensity in [0, 255] as an `int`.
- Write function `opposed` in file `opposed.s`. It accepts two `int`s. It returns an `int`, which is 1 if one number is negative and the other is not. Otherwise it returns 0. Write this without branching; use predicated instructions if necessary.
- Write function `max` in file `max.s`. It accepts two parameters in this order:
  - a `const` pointer to an `int` array (assembly has no notion of `const`, which is something only a compiler can enforce; just don't modify the array)
  - the number of elements in the array, of type `int`

  It returns the maximum element in the array.
- Write function `vecsum` in file `vecsum.s`, which adds two n-dimensional vectors together. It accepts three parameters in this order:
  - a `const` pointer to a first `int` array
  - a `const` pointer to a second `int` array
  - the number of elements in either array, of type `int`

  It dynamically allocates a new array using `malloc` and assigns each element as the sum of the corresponding elements in the parameter arrays.
- Write function `main` in file `echo.s`. Unlike the utility functions above, this is an executable program that mimics the builtin `echo` utility. It prints to standard output the command-line parameters, separated from one another by a space. As in a C program, the number of command-line parameters (`argc`) is given as the first parameter, and the parameter array (`argv`) is given as the second. Do not print out the executable name, which is the first element of `argv`.

3 Later-week Submission

As this assignment is due before the first day of final exams, this assignment is not eligible for later-week submission.

4 Submission

To submit your work for grading:

1. Run the grading script from your homework directory using `../specs/grade`.
2. Commit and push your work to your repository.
3. Verify that your solution is on Bitbucket by viewing your repository in a web browser.

A passing grading script does not guarantee you credit. Your grade is conditioned on a few things:

- You must meet the requirements described above. The grading script checks some of them, but not all.
- You must successfully submit your code to your repository. Expect to have issues with Git.
- You must not plagiarize. Write your own code. Talk about code with your classmates. Ask questions of your instructor or TA. Do not look at others' code. Do not ask questions specific to your homework anywhere online but Piazza. Your instructor employs a vast repertoire of tools to sniff out academic dishonesty, including: drones, moles, and a piece of software called MOSS that rigorously compares your code to every other submission. You don't want to live in a world serviced by those who squeaked by through questionable means. For your future self, career, and family, do your own work.

The grading script allows you to signal your instructor when requirements are met. You only need to send an email if you qualified for later week submission and are resubmitting after the original deadline.