EECS 470 Lab 4 Assignment

Note:

- The lab should be completed individually.
- The lab must be submitted by the end of Friday the following week and checked off by a GSI.

1 Introduction

You have just learned about the BASH shell and some of the useful utilities provided by the standard GNU/Linux environment. Today you will be using this knowledge to automate the testing of Project 3. The general form of this script will closely resemble the actual Project 3 Autograder, though there are a number of additional testcases which are not released to you.

2 Assignment

You are required to automatically check that a modified version of the Verisimple pipeline produces correct output. The following parts lay out how we recommend you do this, but you are free to do this however you would like.

2.1 Ground Truth

First, we need a *ground truth*, a set of known-correct outputs. To build this, you should write a simple script to loop through files in a directory (the assembly testcases) and call a command or commands on each of them. This should look something like:

```
for file in test_progs/*.s; do
        file=$(echo $file | cut -d'.' -f1)
        echo "Assembling $file"
        # How do you assemble a testcase?
        echo "Running $file"
        # How do you run a testcase?
        echo "Saving $file output"
        # How do you want to save the output?
        # What files do you want to save?
done
```

2.2 Testing

Now that you have a script that generates some ground truth data, it is time to modify it to compare that data with the output of some unknown version of the Verisimple pipeline. This will be done in several steps. First, you will need to run the script you wrote in section 2.1 to generate your ground truth. Then, you will need to make sure the executable you are running is up-to-date. After that, for each of your testcases you will need to

- Simulate the testcase. Do you need to rebuild the executable for this?
- Check that the `writeback.out` files match exactly with the ground truth version. What utility does this?
- Check that the lines beginning with `@@@` in the `program.out` files match exactly. What utility lets us find only these lines, before using the one from above to compare?
- Print whether the testcase passed or failed.

3 Check Off

In order to get your work checked off, you will need to show your **well-commented** scripts to a GSI. Once you think you have your scripts working, please add yourself to the help queue and mark that you wish to be checked off. While you wait, you might consider implementing one of the optional features listed below or working on the project itself.
In addition, you also need to show your git repository of project 3 and grant Admin access to the account `eecs470staff@umich.edu`.

A Optional Features

This was a very basic version of this script, and we can improve it significantly from here forward. Here are a few things you could do to make this script better/prettier/more useful for the final project...

- Use BASH functions to modularize this script. I would recommend adding this to your `.bash_profile` so that you can call (e.g. `source .test.bash`).
- Colorize your output. I find it more satisfying to read something like `PASSED` than to read `PASSED` without the color. The Linux Documentation Project has a good description of BASH colors.
- You may want to move some significant portion of the handling of the testcases/assembly files into the Makefile/test bench. For instance, the `$readmemh` command in the test bench can be pointed at a file provided by a `` `define ``, which in turn can be provided by an environment variable/Makefile macro. The dependency resolution capabilities of make is also very useful for making sure that you are running the right test case.
- Detecting and killing infinite loops or hung simulations is extremely useful, but also very hard.