EECS 470 Lab 4 Assignment

Note:

- The lab should be completed individually
- The lab check off is due by **Friday, September 26th**

1 Introduction

In this lab you will write BASH scripts to automate the testing of project 3. You will develop basic scripts using the utilities mentioned in the lab slides to check that your modifications to the VeriSimpleV pipeline still produce the correct register writeback and memory outputs.

2 Generate Ground Truth

The first thing we'll need is a *ground truth*, a set of known-correct outputs to compare against. Luckily we already have a correct processor! The initially provided VeriSimpleV processor produces correct memory and register writeback output, so we can use it to make our ground truth data.

Start by downloading your project 3 repository and making a copy in a new directory to house the original project files.

Then, write a short BASH script which will use the Makefile to compile the original processor and run it on each of the simulation files generating output in the new directory.

We will walk through the script below together during the lab presentation. You can copy the script as is to generate the ground truth files. When you copy the script, be sure to avoid copying invisible formatting characters inserted by the PDF.

Once you generate the ground truth files, copy the `.out` and `.wb` files from the `output` directory of the original repository into the `correct_out` directory of your project 3 repository.

```
echo "Generating ground truth outputs from original processor"

cd ~/470/your_p3_original

# This iterates through *.s and *.c files.
for source_file in programs/*.s programs/*.c; do
    if [ "$source_file" = "programs/crt.s" ]
    then
        continue
    fi
    program=$(echo "$source_file" | cut -d '.' -f1 | cut -d '/' -f 2)

    echo "Running $program"

    make $program.out

done
```

Figure 1: Example BASH script for generating ground truth data

Note: do not use the `make simulate_all` command yet. You should copy this BASH for-loop instead. This will make it easier to do per-output operations later.

3 Comparing to the New Processor

Now that you have a script which generates ground truth outputs, you will write one more script to compare those truthy files to the output of your project 3 implementation.

Change back to your actual project 3 directory and write one more BASH script which does the following for each test program:

1. Run the program on your processor and generate fresh output.

   Which Make target does this?

2. Check the register writeback output: The `output/*.wb` files should match exactly with the ground truth version.

   Which shell utility can you use to find the difference between files?

3. Check the memory output: The `output/*.out` files should have all lines which start with "`@@@`" match the ground truth `@@@` lines. Note that the `output/*.cpi` files will not be accurate for the final pipelined processor.

   What utility lets us *G*lobally search for *R*egular *E*xpressions and *P*rint the matching lines? How can we save this output to a file?

4. Print whether the test `Passed` or `Failed`.

   How can you choose one or the other? Can you store some of the values of the earlier checks to a variable?

```
echo "Comparing ground truth outputs to new processor"

cd ~/470/your_p3_implementation

# This only runs *.s files. How could you add *.c files?
for source_file in programs/*.s; do
    if [ "$source_file" = "programs/crt.s" ]
    then
        continue
    fi
    program=$(echo "$source_file" | cut -d '.' -f1 | cut -d '/' -f 2)
    echo "Running $program"

    echo "Comparing writeback output for $program"

    echo "Comparing memory output for $program"

    echo "Printing Passed or Failed"

done
```

Figure 2: Example BASH script for comparing against ground truth data

3.1 Test It

Change some non-trivial part of the processor and try running your scripts to ensure you can print incorrect outputs properly.

Many modifications, even very small ones, will make your processor run infinitely on some test programs. One modification that won't cause your processor to run infinitely is to change `$fdisplay` on line 295 in

`test/cpu_test.sv` to use `memory.unified_memory[0]` instead of `memory.unified_memory[k]`.

3.2 Some Notes

Project 3 comes with a "correct_out" folder that contains the correct output from 3 programs. However, you will need more exhaustive test cases to make sure everything is working properly. The goal in this lab is to supplement those instructor-given files with correct functional output from all test programs (and let you easily check outputs for other programs you may write).

Also, each program generates 4 outputs: *.out, *.wb, *.cpi, and *.ppln files. You should only use *.out and *.wb files for checking functional correctness. The cpi of your lab 3 processor will change once you pipeline your processor, so the *.cpi files you generate here will not be valid for the final processor. The only *.cpi files you should use to check your final processor are the 3 instructor-given ones. The *.ppln file should be used for debugging.

4 Submission

For lab 4, be ready to show your BASH script comparing the outputs of your project 3 to the correct ground_truth files. Be sure your compare script prints "passed" if the outputs of project 3 are correct, and "failed" if not. The instructor will ask you to introduce a bug into your project 3 to test the failure case.

Place yourself on the help queue during lab or office hours once you're confident you've completed the lab satisfactorily. Turn in your check-off to Gradescope by the end of the day of next week's lab.

5 Optional Features

This was a very basic checking script, and we can improve it significantly with a few modifications. 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`).
- Detecting and killing infinite loops or hung simulations is very useful, but also very hard.
- 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. Or you could steal the BASH function below:

```bash
# this function takes a color by number, then prints the rest of the arguments
# call it like: echo_color 2 this is a message
echo_color() {
    # check if in a terminal and in a compliant shell
    # use tput setaf to set the ANSI Foreground color based on the number 0-7:
    #   0:black, 1:red, 2:green, 3:yellow, 4:blue, 5:magenta, 6:cyan, 7:white
    #   other numbers are valid, but not specified in the man page
    if [ -t 0 ]; then tput setaf $1; fi;
    # echo the message in this color
    echo "${@:2:$#}"
    # reset the terminal color
    if [ -t 0 ]; then tput sgr0; fi
}

for i in 1 2 3 4 5 6 7; do
    echo_color $i Hi from the lab assignment!
done
```

Figure 3: Example BASH script for comparing against ground truth data