Introduction to Simulation

CS 370 Project 1 – Spring 2019
Due: February 13, 2019 at 11:00pm

1 Introduction

Please read this entire document. It is lengthy and contains many details – but I want to make sure you get off on the right foot. Save yourself some headache and read this carefully.

This is the first project for CS 370. The purpose of the project is to give you an opportunity to get tools set up and running, acclimate you to running command line tools, give you an opportunity to refresh yourself on assembly and C, and introduce you to processor modeling and simulation.

There are two main tools that we will use for the projects this semester. The first tool is an assembler, called *lasm* (which stands for the La Crosse Assembler). Recall that assemblers are used to convert programs written in assembly source code to executable files.

We have no way of running the executables emitted by *lasm* because your laptops, desktops and our departmental servers all have Intel processors, rather than MIPS processors. It is common in computer architecture research to **simulate** the execution of programs, rather than execute them directly. So, the second tool we will use for our projects will be *lsim* (the La Crosse Simulator). The simulator opens the executable file, and executes the program in software rather than hardware. You may be familiar with emulators for retro game systems – in fact *lsim* is both a simulator and an emulator.

Both *lasm* and *lsim* are command line tools, and this project is meant to simply get these tools up and running as a precursor to future projects. You will run both *lasm* and *lsim* to produce the output that you should submit (in addition to the miniscule amount of code you need to modify) to D2L for grading.

This project must be done individually. The tools have been written such that they produce unique output when run in your environment, even with the same input as other students. Any evidence of copied code or outputs will be reported to the Student Life Office as an academic integrity violation.

You must run your assignments on the departmental Linux server. *lasm* has been fully pre-compiled and it is not guaranteed to run anywhere other than the departmental server. Similarly, *lsim* is partially pre-compiled (you will do the final compilation), so also needs to be done on the departmental server.

The due date for this project is **Wednesday, February 13th, 2019 at 11:00pm**. Project code and output files must be submitted to D2L. Please submit individual files, do `not` zip files together for submission. Late projects cannot be accepted. It is in your best interest to start this project as early as possible in case you have issues running the tools. There are several supplementary and reference documents on D2L to help you get started.

2 Project Specification

There are several steps in this project, first you must download *lasm* and the partially compiled *lsim*, copy them to your working directory on the departmental server, then modify a given assembly program, and assemble that program with *lasm*. You must then lightly modify some C code to *lsim*, finish the compilation

of *lsim*, and then run *lsim* with the assembly program from the previous steps. We will take each step one-at-a-time in the remainder of this document.

2.1 Logging in to the Server

First, you need to log in to one of the CS department servers. Using `ssh`, log in to `compute.cs.uwlax.edu`. If you have never used `ssh` (or forgot), it is a tool for remotely connecting to the command-line interface of another machine. See the document on D2L named "computing.environment.pdf" (in the Reference section of D2L) for details on setting up `ssh`.

`compute.cs` is a 40-core, 128GB memory, a real bad-mamba-jamba – perfect for our high-performance simulation needs. The simulator and assembler that you will be using/modifying for projects for the rest of the semester have been tested on `compute.cs`, so it is in your best interest to compile and run *lasm* and *lsim* there. If you try these tools elsewhere, you will be given no mercy if something goes wrong when I grade your assignments.

As a sanity check, once you're logged in, you should see a command prompt (usually ends with a dollar sign). If you issue the Linux command `ls` (just type `ls`, then hit enter) – that is used to list the files in your current directory (folder), right after you log in, you will always be in your home directory. You will likely see directories named `Desktop`, `Documents`, etc. and any files that you've previously saved to your campus drive. I presume that the Linux command-line is at least vaguely familiar to you. D2L has a document (again, in the Reference section) named "linux.howto.pdf" that gives you some common commands, and places to find more info. Linux is the bees knees for developers, if you're not familiar with it... it's time to embrace it.

You will next need to get the project files (from D2L) onto your network storage (your home directory). You can either `scp` the files, and then work on them from your `ssh` session, or you can mount your network directory to your laptop/desktop so that it appears like any other directory/folder on your machine. If you have never used `scp` or mounted a network drive, see the computing environment document on D2L.

Once the zip file is in your home directory, you can go back to your `ssh` session, and unzip with:

```
unzip project1.zip
```

You can then delete the zip file if you want:

```
rm project1.zip
```

Be careful when deleting files in Linux, there is no way to retrieve files once they've been deleted.

2.2 Assembling a Program

If you go into the `project1` directory (hint, the `cd` command) and list the files, you will see an `assembler` directory, a `simulator` directory, and a file named `scramble.asm`. The `assembler` directory only has the executable for *lasm*. The `simulator` directory has several subdirectories and files. This subdirectory structure is common for a C program that is in development. The `bin` subdirectory is empty right now, but will eventually hold the simulator executable. The `src` subdirectory has `.h` and `.c` source code files. The `obj` subdirectory holds object files (files that have been compiled, but not yet linked). You will see right now it has several files that I have compiled for you already. Finally, a `makefile` has been provided, so that you can quickly build the simulator.

For this step, you should first try running *lasm*. `cd` back to the `project1` directory. From here, you should be able to run:

```
./assembler/lasm
```

And when it runs without any command-line flags, it simply prints a "usage" message and ends. You can see there are a handful of command-line flags that you can use when running *lasm*, but at the very least, you need to provide the filename of the assembly source file that you want to assemble – in this case, `scramble.asm`. However, before assembling the scramble program, I want you to modify one of the values at the top of the program.

Open `scramble.asm` with a plain text editor of your choice (some options are discussed in the computing environment document on D2L), and at the top you will see a data value with the name `seed`. Currently that memory location is initialized with the value `0xdeadbeef`. You should change that value to something else, any value of your choosing. Pick something with a mix of 1 and 0 in several bit positions, 0-9a-f in hex (i.e. don't do anything boring like `0x00000000`). Keep in mind that this value must be 32-bits. Save this file with your modified value (same filename is fine).

Now assemble your program. You could simply run:

```
./assembler/lasm scramble.asm
```

... which will produce a MIPS executable with the default name of `a.out`. Alternatively, you could give a name to the executable with something like:

```
./assembler/lasm -out fred scramble.asm
```

... which would produce an executable with the name `fred.out`. You can name your executable anything you wish.

For more (a lot more) discussion on *lasm*, see the `lasm.pdf` document in the Reference section of D2L.

2.3 Compiling *lsim*

In the previous step, you assembled the scramble program into a MIPS executable. To run the scramble program, you will need to finish the compilation of *lsim*. To do so, you can `cd` into the `simulator` directory, and simply issue the `make` command. This will use the `makefile`, which contains the rules for building *lsim*. However, you should make another small modification to *lsim* before building.

Open `src/main.c` with your favorite plain text editor. This is the `main()` function of *lsim*. You will see the code that I use to parse the command-line options to *lsim*, some code that initializes the emulator and simulator, and most importantly, the main simulation loop (the `do...while loop`). This loop is where most of the magic occurs. The calls to functions named `fetch()`, `decode()`, etc. are all tasks that a processor will perform when executing instructions. So, those functions model those processor tasks.

lsim also has a built-in emulator. The difference between a simulator and an emulator is that emulation only mimics the functionality of instructions, whereas a simulator mimics the functionality of instructions while also maintaining timing information. *lsim* runs the simulator and emulator redundantly – so your MIPS programs are actually executed twice by *lsim*. You might ask why? In future projects, you will be writing your own versions of some of the simulator functions (your own implementations of `fetch()`, `decode()`, etc.). To help you debug, the emulator has a `check()` function, that I will always provide with future projects. When you call that `check()` function, as can be seen at the end of the `do...while` loop, then the simulator state is compared to the emulator state. If the simulator differs from the emulator, then the simulator is immediately ended, and debug info printed. So the emulator is there to help you debug your future projects.

After the simulation loop, you will see a call to a function named `probe_system()`. This function is querying some of the properties of the machine that you are using to run *lsim*, and it prints this info, as well as a

checksum of that info. This is how I will determine on which system you are running your simulations to ensure that you are doing this assignment yourself.

Before building *lsim*, you should modify the value that is being passed to `probe_system()`. Similar to the scramble value, you can pick any value that you wish to pass to this probe function, but pick something interesting with lots of 1s and 0s throughout (0-9a-f in hex). Again, this should be a 32-bit value.

Save your modified `main.c` file, quit, and then issue the `make` command to build *lsim* – its executable should appear in the aforementioned `bin` directory.

More info on *lsim* can be found in the `lsim.pdf` in the Resource section of D2L. That document will be modified throughout the semester, as we make changes for future assignments.

2.4 Running a Simulation

Now that you have a MIPS program (your scramble executable), and the simulator (the `bin/lsim`), you are ready to run a simulation. You can `cd` back to your `project1` directory, and simply:

```
./simulator/bin/lsim a.out
```

This assumes that you used the default name when assembling scramble. If you had named your scramble program `scramble.out`, then you would issue:

```
./simulator/bin/lsim scramble.out
```

The scramble program expects user input, the initials of your name. It will first print a prompt, then you should type your initials – three letters without any spaces between. So if your name is Joe T. Schmoe, then you would enter `jts` or `JTS`. Be sure to use **your** initials, and always three letters. If you don't have a middle name, then use the letter "x" for your middle initial. I will be using your initials when I replicate your results when I grade.

After entering your initials, and pressing enter, the rest of the scramble program will run, and print an integer value before it ends. After printing the value, *lsim* takes over and prints several statistics that it gathered about the program it ran.

lsim doesn't print any information as it simulates the MIPS program by default. If you want to see the results of instructions as they are executed, then you can re-run the scramble program with the `-trace` option:

```
./simulator/bin/lsim -trace a.out
```

Don't forget that scramble is still expecting you to enter your initials, so a bit of information about a handful of instructions will be printed, and then the simulation will appear to hang... but of course it is just waiting for you to type your initials. Once you do, you will see that *lsim* spits out a ton of instructions, one per row. Each instruction has helpful debugging information, like values that each register had, results, addresses, and so on. `lsim.pdf` on D2L outlines the meaning of all of the trace information.

Re-run *lsim* with the scramble program one more time without trace, but this time, redirect the output to a text file. For any program that prints output to the console (stdout), Linux allows you to save that output to a file instead of printing to the console. Run the simulation with:

```
./simulator/bin/lsim a.out > output.txt
```

This time, it will appear that nothing is happening, but in fact, the simulator is running, and it did print the prompt for your initials, but that prompt is now in the `output.txt` file, so you won't see it. Just enter

your initials again, hit enter, and you should then be returned to the prompt. If you open `output.txt` in your favorite plain text editor, you will see your output has safely been stashed away. This is one of the powerful aspects of Linux. If you want to run your simulation again, you will first need to delete the existing file (with `rm output.txt`), since this type of file redirection requires that the file doesn't already exist.

You can also redirect an existing text file to the **input** of a running program. If you wanted, you could create a new text file (again in your favorite editor) and save your initials to that text file – let's say you called it `input.txt`. You could then run:

```
./simulator/bin/lsim a.out > output.txt < input.txt
```

...which will fully automate your simulation and save the output. Pretty neat, eh?

3 Submission and Grading

When you are done, you should submit your project to the D2L dropbox named "Project 1". Please submit only the files you modified, and the output.txt file with the example run of scramble on *lsim* with your initials. Please **do not** zip your project files, I only want:

```
scramble.asm
main.c
output.txt
```

The project deadline is February 13th, at 11:00pm. Late projects can only be accepted with a university-approved excused absence.

The grading for this assignment is straight-forward. I'm going to compile *lsim* with your modified `main.c`, then re-run the tools with your modified `scramble.asm`. I will then double-check the results of the integer value printed by scramble to make sure it matches the value in your `output.txt`. I will also check the information printed by the `probe_system()` function to make sure it matches your environment, and that the checksum matches the expected value. If these items are all in order, then you can expect full credit. If something doesn't match, I will likely email you about it, in case I'm not using the same seed or probe value.