Lab #0: MIPS & Broken SPIM

Lab 0 is due Friday, March 1st. Your lab report and source code must be submitted by **10:10 AM** before class. The late policy applies to this lab project.

This lab is to be done individually. Get started early! Also, remember to **put your name** on the lab report! The required format for lab reports can be found in the resource web page.

Problem: Writing diagnostic programs

You are given a buggy version of SPIM, called `spim_broken` (setup instructions below). 3 of the 12 instructions you are asked to verify are buggy in some way or under some conditions. 9 of the instructions always execute perfectly. Your task is to write a set of diagnostic programs to identify the 3 buggy instructions, and to describe each bug. (You will verify lots of other things work. Include programs that confirm correct instructions too.)

Here is a list of things you should check for:

1. Register zero always has the value zero. Please test the case that the register zero is used as both the target register and one of the source registers. (Note: Register zero may appear to behave abnormally for all instructions in the *interactive mode*, that is when you print the value in the register zero, the register may contain non-zero values if the instruction just executed writes to the register. The error appears in both the correct version and the buggy version of SPIM. It is related to how the register zero is implemented in SPIM. To bypass this error, please use only batch mode to test the value of register zero).
2. All the branch and jump instructions should jump to the right address under the right condition.
3. SLT(I) and SLTI(I) must work properly with bit patterns representing negative two's complement integers or large natural numbers.
4. All the arithmetic and logical instructions with immediate field should extend it properly.
5. Shifting instructions work correctly and extend the MSB appropriately.

You should make sure that the fundamental things work before you test some other things that depend on them. Once you have verified that simple compares and branches work, you can build longer programs with a sequence of tests. After each test, you can use a compare followed by a branch to either go to the next test, or to a "failure label" which will put in an identifier for the failed test in a register. As long as there are no failures, it will sequence through all the tests. You can easily look at the "failure register" at the end of execution to determine which (if any) of the tests failed. Your test code should look something like the following:

```
test case a
beq caseA_result, expected_Result, go_to_caseB
jump to failure A

caseB:
test case b
```

```
...
...

failure A:
addiu $t0, $0, 1
j endOfTest

failure B:
addiu $t0, $0, 2
j endOfTest

...
...
endOfTest:
```

A sample diagnostic program is provided. Writing diagnostic programs can deepen your understanding of the instruction sets. The main purpose of this assignment, however, is not only to refresh your knowledge of MIPS and understand the diagnostic process. More importantly, this lab is supposed to teach you how to create programs that **can be used to validate your design later in the semester**, since the final project only requires you to implement your own ISA. Therefore, diagnostic programs will only be useful if they are written using the limited set of instructions that are available on a CPU. Hence, you must write the diagnostics using only the MIPS instructions below:

arithmetic: addu, addiu
logical: and, or, andi
compare: slt, slti
control: beq, bne, j

Helpful Hints

The 3 bugs were introduced by modifying a simulator, not by modifying hardware. Thus, do not think in terms of "I think one shift opcode has a bug, and therefore the other shift opcodes must have a bug too". Instead, you should think in terms of hunting for 3 independent bugs.

Just as a reminder, some of you will find 6 or 7 opcodes with bugs, and you will feel certain that all of these bugs exist. In actuality, what is happening is that one of the instructions you think is correct is in fact buggy, and that bug is deceiving you into thinking that correct instructions are buggy.

What to Turn In

For this lab project, turn in all of your diagnostic programs. Describe the testing methodology that motivates the design of your program suite. Tell us how your programs systematically test the processor, and explain why these tests were able to find the bugs you found.

Also, turn in a description of the specific errors that you found, and note which test program excited each error.

Point out the machine language instruction sequences in your diagnostic suite that tests each of the "things you should check for" listed earlier in this section. Points will be taken off if you did not test for all of these items, even if you found all of the bugs! We want to see that you can design systematic tests for an ISA.

In particular, don't ignore the list of "things you should check for", and just write tests for whatever ideas came to mind.

How to Submit

Copy your lab report, which is a .pdf, a .doc, or a .html file, and all your source code into an empty directory. Assuming the directory is "submission", make a tar ball of the directory using the following command:

```
tar czvf [your_first_name]_[your_last_name]_lab1.tar.gz submission.
```

Replace [your_first_name] and [your_last_name] with your first name and your last name.

Use the submission function of Canvas to submit the tar ball file.

Setup Instructions:

All files are located on mlb.acad.ece.udel.edu. You need an EECIS "ACAD" account to access the machine. If you don't already have one, you can apply for an account at www.eecis.udel.edu.

After log on, first you need to copy exception.s to your work directory. When you run spim or spim_broken, exceptions.s must be in the current directory, that is, the directory "./". All the spim programs and exception.s can be found at /usa/xli/spim/.

For mlb.acad.ece.udel.edu and other departmental linux machines:

spim_broken: Buggy version of spim.
spim: Correct version of spim.