CS243 Computer Architecture

Extra Credit for the Midterm Exam: Optimize Wordcount

Your job in this lab is to rewrite some compiled C++ code in MIPS so it runs faster than what the compiler generated.

This assignment is also a contest. See how fast you can make your improved code run. I'll announce the five fastest submissions in class.

All work must be your own. No use of AI or starting from other people's work. You may work in a pair, though. If you work in a pair, please submit only one source file for both of you and include a comment at the start of the file telling both your names.

Wordcount

The program you'll be optimizing is a simplified version of the standard Unix utility *wc*, which counts the number of characters, words, and lines in a text file. Once you extract the *tar* archive for this homework assignment, you should have a directory containing these files:

`hwwc.pdf`	This PDF file.
`wc.cpp`	A C++ implementation of *wc*.
`wc-compiled.asm`	The MIPS assembly language from the compiler.
`wc.asm`	A test file in MIPS assembly language for you to start from.
`kant`	A text file to use for testing.

If you compile `wc.cpp` and run it on `kant`, you should see this:

```
$ ./wc kant
lines: 166
words: 1284
chars: 7806
```

A line is a sequence of characters terminated by a newline (`\n`). A word is defined as a sequence of one or more non-whitespace characters. The whitespace characters are:

name	*C literal*	*ASCII code*
tab	`'\t'`	9
newline	`'\n'`	10
carriage return	`'\r'`	11
vertical tab	`'\v'`	12
form feed	`'\f'`	13
space	`' '`	32

What to Do

1. Read `wc.cpp` and understand how the `wc_impl` function works. It's simple but it might use a programming technique that you haven't seen before. (It counts words using a very small "finite state machine".) Notice also the use of pointers to walk through the string and to indicate where in memory to store the results (in the `WcResult` structure).

2. Read the `wc-compiled.asm` file to see the code that *gcc* generated for `wc_impl` and `is_space`. This file includes a `main` function and it includeds the `kant` text file in the data segment so it's easy to run and test. If you run it in MARS, the numbers you see should match those output by the C++ program.

   I have added comments to the code to make it easier to understand. As you read through the code, you may find parts that seem strange or pointless. Compilers often write code like that. The more you notice that seems unnecessary, the better, because those will give you ideas for simplifications.

3. Now open `wc.asm` and write your own version of `wc_impl` (and `is_space` if you call it). Your code should generate correct results for any text file passed to it, and it should run at least 10% faster than the compiler-generated code. Correctness includes restoring all callee-preserved registers. (Hint: if you don't trash any callee-preserved registers, you won't need to restore any.)

   We will count execution time simply as the number of instructions executed. (We will see a little later in the course that real execution time is more complex than this.) All that counts is the number of instructions executed while in `wc_impl` (and any functions that it calls), not the code in `main`. To measure execution time, use the `Instruction Counter` tool from the `Tools` menu in MARS. To verify that you are measuring the number of instructions correctly, measure the execution time in `wc-compiled.asm`; it should be 156,371 instructions. So, a 10% speed-up would be completing `kant` in 140,733 instructions.

   The use of 0xCAFED00D to initialize the members of the `WcResult` struct is a debugging technique. If those numbers occur in the output (they will be huge negative numbers), then most likely your program never updated the `WcResult` in memory.

4. Once you've gotten a correct version working, play around with it and see how much faster you can make it. Consider small tweaks as well as wildly different ideas. You don't need to mimic the C++ code at all. You can implement it using any MIPS assembly code you can think of, as long as it's correct. It's possible to shave off quite a lot more than 10%. Consider other programming techniques that have been shown in the course so far, like a look-up table.