Inf2C Computer Systems

Coursework 1

Deadline: Thu 27 Oct 2011, 4pm

Paul Jackson

1 Description

The aim of this assignment is to introduce you to writing MIPS assembly-code programs. The assignment asks you to write two MIPS programs and test them using one or more of the SPIM simulators. For details on SPIM, see the first lab script, available at:

`http://www.inf.ed.ac.uk/teaching/courses/inf2c-cs/labs/lab1.html`

and the Patterson and Hennessy text book.

This is the first of two assignments for the Inf2C Computer Systems course. It is worth 50% of the coursework mark for Inf2C-CS and 12.5% of the overall course mark.

Please bear in mind the guidelines on plagiarism which are linked to from the Informatics 2 Course Guide.

1.1 Task A: Counting characters

This first task is a warm-up exercise for the second task. It helps you get familiar with the basic structure of MIPS programs and with using one of the SPIM simulators.

Write a MIPS program that counts the number of characters in a sequence of characters entered from the terminal. The program should expect the sequence to end with a `$` character and should ignore whitespace characters. Assume the sequence otherwise does not contain any `$` characters.

Whitespace characters you should handle are

- *space* (' ', 0x20)
- *tab* ('`\t`', 0x09)
- *newline* (a.k.a *line-feed*) ('`\n`', 0x0a)
- *carriage-return* ('`\r`', 0x0d)

Most of the C-syntax names here for the characters (e.g. '\t') do not work in SPIM, so you will need to use the hex codes.

Name your program `count.s`.

A sample interaction with your program should look like:

```
Enter text, followed by $:
School of
Informatics$

Count: 19
```

where `%` is the command line prompt.

You will find that the exact nature of the interaction will depend on the tool you use. For example, with `spim`, the command-line shell only sends input to your program when *Enter* is keyed, whereas with `xspim`, your program will see the `$` as soon as you type it.

Please use the same wording as above in the text printed by your program. This will simplify testing and marking your program.

You might well want to make use of the `read_char` syscall to read in typed characters. If you do so, bear in mind that the documentation for it in the SPIM Appendix linked to from the Lab web page has a typo. The table on page A-44 claims that the result character is returned in register `$a0`. This should be `$v0`. This typo is corrected in recent editions (e.g. 4th) of the Patterson and Hennessy book.

Be careful about when you choose to use `$t*` registers and when `$s*` registers. Assume that values of `$t*` registers are not guaranteed to be preserved across `syscall` invocations, whereas values of `$s*` registers will be preserved. However, do not just use `$s*` registers in your code: also make use of `$t*` registers when appropriate.

1.2 Task B: Counting letter frequencies

Task B builds on Task A, so you will probably want to start with a copy of the code you wrote for Task A. Task B involves more complicated branching structure and data manipulations, and brings in working with an array.

Write a MIPS program that reads in a sequence of characters from the terminal, again terminated with a `$` character, and this time that counts the number of occurrences of each letter of the alphabet A-Z. The program should not distinguish between upper and lower case versions of each letter. Counts of character occurrences should be held in an array of 26 32-bit integers.

Name your program `freq.s`.

A sample interaction with your program should look like:

```
Enter text, followed by $:
The quick brown fox jumps
over the lazy dog
$
A: 1
B: 1
C: 1
```

```
D: 1
E: 3
F: 1
G: 1
H: 2
I: 1
J: 1
K: 1
L: 1
M: 1
N: 1
O: 4
P: 1
Q: 1
R: 2
S: 1
T: 2
U: 2
V: 1
W: 1
X: 1
Y: 1
Z: 1
```

You should have at least one instance of a load or store to the array of counts that

(a) just uses primitive instructions to compute the address and do the load or store, and

(b) makes use of pseudo-instructions and the richer addressing modes, as shown on page A-45 of the online Appendix on SPIM.

You might start with only instances of kind (b) and then deduce what to write for (a) by looking at the primitive instructions as reported by one of the SPIM tools.

To see information on the standard ASCII encoding of characters, type `man ascii` at a DICE command line prompt.

2 Program Development and Testing

You are free to develop your programs using different versions of the SPIM simulator, e.g. the linux command line version `spim`, the Linux X-Windows version `xspim` or the Windows PC version `pcspim`. Further information on these versions and the new `qtspim` is available from `http://spimsimulator.sourceforge.net/`. The new `qtspim` version is not currently installed on DICE, as we have had problems building it.

Ultimately, you must ensure that your programs work using the command line `spim` tool, as currently installed on all DICE machines. This is the tool that will be used to check your programs. To run a MIPS program on DICE using `spim`, type

`spim -f` *filename*`.s`

at a command-line prompt.

3 Submission

Submit your work using the command

```
submit inf2c-cs cw1 count.s freq.s
```

at a command-line prompt on a DICE machine. Unless there are special circumstances, late submissions are not allowed. Please consult the online Informatics 2 Course Guide for further information on this.

4 Assessment

Your programs will be primarily judged according to correctness, completeness, code size, and the correct use of registers.

In assembly programming, commenting a program and keeping it tidy is very important. Make sure that you comment the code throughout and format it neatly. A proportion of the marks will be allocated to these.

One approach to commenting assembly code is to include C-like pseudo-code in the comments, in addition to natural-language phrases. This pseudo-code can introduce more meaningful names for variables and constants, show the expressions being computed and make the control structure clearer. Such pseudo-code has to be crafted carefully so it is easy to see and check its relationship to the assembly code.

5 Questions

If you have questions about this assignment, ask for help from the lab demonstrators, your Inf2C-CS tutor or the course organiser.

13th October 2011