Informatics 2C – Computer Systems

Coursework 1

Deadline: Friday, 29 October 2010, 12:00 noon

David Braude & Paul Jackson

1 Description

The aim of this practical is to increase your familiarity with programming a MIPS processor. The practical asks you to write and submit two assembly programs for the MIPS processor, as implemented by the SPIM simulator, and a short report. For details on SPIM, see the first lab script available at:

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

This is the first of two practicals 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 Reversing a string

The first task is to write a program to read in an input string from the terminal, reverse the order of the characters, and print the reversed string to the terminal. The following are key requirements:

- The input string will be terminated with a newline.
- The input string will be no longer than 40 characters (not including the newline).
- The reversal should be accomplished by pushing the input characters onto a stack and then popping them off this stack.
- The output string should be terminated with a newline.

Your program need not handle the error condition of there being more than 40 characters in the input string. For implementing a stack, do not use the stack pointer register $sp. Instead choose one of the other general-purpose registers for your stack pointer.

Of course, the use of a stack is not strictly necessary for reversing a string. We ask for it here, because you will be programming a stack again in the 2nd part of this practical.

Please do not include in your program output any extra text, like a prompt for the input string. We will be running some automatic tests on your program, and the test harness will not expect this extra output. If we see extra text, marks might be deducted. An example test program will be made available before the practical hand-in date.

Save your program in a file named **`reverse.s`**.

1.2 Bracket parsing

The second task is to write a program to check that an input string is properly bracketed, that is, for every open bracket there is a corresponding close bracket, there are not too many close brackets, and brackets are properly nested. There are two types of brackets you must consider: round

brackets '`( )`' and square brackets '`[ ]`'. Your program should work through the input string, pushing open brackets onto a stack and popping the stack when a close bracket is encountered. Error conditions your program should report are

- *Mismatching close bracket*. For example, at the last character of "`((abc)]`".
- *Unexpected close bracket*. For example, at the 6th character of "`(def)]g`".
- *One or more open brackets are unclosed*. For example, after the last character of "`[[ghi`".

As before, you can assume the input string will contain no more than 40 characters.

The output messages should have form

```
At <c>: mismatching close bracket
At <c>: unexpected close bracket
At <c>: unclosed open bracket(s)
```

Messages should be terminated with a newline character immediately after the last printing character. The **<c>** should be replaced with the position of the character at which the error condition is detected. For the 3rd error condition, **<c>** should be the position of the newline terminating the input string. Position numbers should start at 1 for the 1st character of the string.

Your program should make a best effort to continue after detecting an error. For example, if a mismatching close bracket is found, assume the bracket is a mis-typed correct closing bracket. If a close bracket is unexpected, just treat it as you treat non-bracket characters. Because of this best effort behaviour, your program might report multiple error conditions for a single input string. For example, on the input string "`ab]cde(fgh])(ij`", the output

```
At 3: unexpected close bracket
At 11: mismatching close bracket
At 12: unexpected close bracket
At 16: unclosed open bracket(s)
```

would be expected.

If no errors are detected in the input string, print the message

```
At <c>: all brackets matched
```

where **<c>** is the position one past the input string end.

The printing of messages should be accomplished by a simple function call where a pointer to the message string without the position information is passed as an argument. Function calls should use a jump-and-link (jal) instruction and returns the jump register (jr) instruction. As there is no recursion, there is no need to save the return address on the program stack pointed to by $sp, and, in general, there should be no need to use the program stack for any other purposes. The body of the function should take care of adding the character position information to the message.

As with the first task, an example test program will be made available before the hand-in date.

Save your program in a file **`brackets.s`**.

1.3 Report

In addition to the programs you must submit a short report. It should be approximately a page in length. In this report you give an explanation of the two algorithms you used. You should include psuedo-code or flowcharts in your discussion to help with your explanation. You should also explain your choice of registers. The report should be converted to PDF format and saved in a file

named **`report.pdf`**.

2 Submission

Submit your work using the command

```
submit inf2 inf2c-cs cw1 reverse.s  brackets.s  report.pdf
```

at a command-line prompt on a DICE machine.

Unless there are special circumstances, late submissions are not allowed. Please consult the Informatics 2 course guide for information for details.

3 Assessment

We will use SPIM to run your programs, so ensure that they load and run on the simulator. Your programs will be primarily judged according to correctness, completeness, performance, 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.

The report will be marked not only for completeness but also your ability to communicate. It is suggested that you get a first language English speaker to check it, if you are not confident with your work. Cite any work that is appropriate.

4 Questions

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