Inf2C Computer Systems

Coursework 1

MIPS Assembly Language Programming

Deadline: Wed, 26 Oct (Week 6), 16:00

Boris Grot, Priyank Faldu

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 the MARS IDE. For details on MARS, see the first lab script, available at:

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

The assignment also gets you to write C code and use C code versions of the programs as models for the MIPS versions.

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 20% of the overall course mark.

Please bear in mind the guidelines on plagiarism which are linked to from the online Undergraduate Year 2 Handbook.

1.1 Task A: Word Finder

This first task is a warm-up exercise. It helps you get familiar with the basic structure of C and MIPS programs, and with using the MARS IDE.

Write a MIPS program `find_word.s` that given an input sentence, outputs the words in it. For the purpose of this exercise, a word is defined as a sequence of characters and/or numbers without any white spaces or punctuation marks in it. The only punctuation marks that can appear in the input sentence are: , . ! ? _ - ( ) (comma, period, exclamation, question mark, underscore, hyphen, parentheses, and white-space). You do not need to check for invalid characters in the input. Finally, a word may or may not have a meaning e.g. vk8s92p is a valid word.

A good way to go about writing a MIPS program is to first write an equivalent C program. It is much easier and quicker to get all the control flow and data manipulations correct in C than in MIPS. Once the C code for a program is correct, one can translate it to an equivalent MIPS program statement-by-statement. To this end, a C version of the desired MIPS program is provided in the file `find_word.c`. You can compile this program at the command prompt on the DICE machines with the command:

```
gcc -o find_word find_word.c
```

This creates an executable `find_word` which you can run by entering:

```
./find_word
```

Sample interaction with your program should look like:

```
./find_word
input: The first INF2C-CS (!!!!!!) coursework is due on 26-Oct (Wednesday).
output:
The
first
INF2C
CS
coursework
is
due
on
26
Oct
Wednesday
```

For convenience, the C program includes definitions of functions such as `read_string` and `print_string` which mimic the behaviour of the SPIM system calls with the same names. Derive your MIPS program from this `find_word.c` C program.

Don't try optimising your MIPS code, just aim to keep the correspondence with the C code as clear as possible. Use your C code as comments in the MIPS code to explain the structure. Then put additional comments to document the details of MIPS implementation.

As a model for creating and commenting your MIPS code, have a look at the supplied file `hex.s` and the corresponding C program `hex.c`. These are versions of the `hexOut.s` program from the MIPS lab which converts an entered decimal number into hexadecimal.

You will need to choose what kind of storage to use for each of the variables in the C code. The programs for Task A and Task B (explained next) are small enough that byte- or word-sized variables should likely all fit in registers. However any arrays will need to go into the data segment. Use the `.space` directive to reserve space in the data segment for arrays, preceeding it with an appropriate `.align` directive if the start of the space needs to be aligned to a word or other boundary.

Be careful about when you choose to use `$t*` registers and when `$s*` registers. Remember that values of `$t*` registers are not guaranteed to be preserved across `syscall` invocations,

whereas values of `$s*` registers will be preserved. However, do not use only `$s*` registers in your code, make use of `$t*` registers when appropriate. Note that you are expected to abide by the MIPS register convention across function calls even for the functions you write.

1.2 Task B: Find Most-Frequently Occurring Word

In this task, you will write a program in both C and MIPS that finds and prints the most-frequently occurring word in the input sentence. The definition of "word" stays the same as in Task A. Words are case sensitive, so word "case" and "Case" are considered different words. If multiple unique words have the same (highest) frequency of occurence, you should print any one of them.

A good way to approach this problem is to generate a histogram of word frequencies. For that, you will need to maintain a list of unique words and the count of occurrences each word.

When storing and printing a string, a special terminating character is appended at the end of a string to mark the boundry of a string. In C, this special null character is represented by '\0'; in MIPS, it is 0x0. So, the word "home" should be stored as five consecutive characters 'h', 'o', 'm', 'e' and '\0'.

Some rules to follow:

- The valid characters are as follows: 'a-z', 'A-Z', '0-9', and , . ! ? _ - ( ) (comma, period, exclamation, question mark, underscore, hyphen, parentheses and white-space). Other characters will not be used in test inputs, so it doesn't matter whether your code handles them or not.
- The length of the input sentence is limited to 1000 characters.
- The length of a word in the input sentence is limited to 50 characters.
- Words are case sensitive. ("case" and "Case" are considered different words)
- If multiple words have the same highest frequency, any one of these words can be output
- Only the following C library functions are allowed: `getchar, fgets, putchar and printf`.

Your program should output the following (words in italics themselves should not be printed):
line 1: *num_unique_words* : number of unique words in a sentence
line 2: *max_frequency* : frequency of most occuring word(s)
line 3: *num_words_with_max_frequency* : number of words with the highest frequency
line 4: *word* : any word with the highest frequency of occurrence

For your convenience, the output function to print the above is provided in the most_freq_word.c file. You need to populate all the parameters of the output function appropriatly. The parameters supplied to the output functions are as follows:

- num_unique_words
- max_frequency
- num_words_with_max_frequency
- word

The first three parameters are integers, while ‘‘word” is a character array.

Sample execution of the C program:

```
> ./most_freq_word

input: How many unique words are present in this unique sentence?
output:
9
2
1
unique
```

Here ’>’ is the Unix command-line prompt, ‘`input: `’ is a prompt printed by the program, and ‘`How many unique words are present in this unique sentence?`’ is text input. After this text is keyed in and the return key is pressed, the program prints the output. This process continues until the user exits the program (e.g. CTRL+C).

Few more examples:

```
> ./most_freq_word

input: The first INF2C-CS (!!!!) coursework is due on 26-Oct (Wednesday) and
second INF2C-cs coursework is due on 23-Nov (Wednesday).
output:
16
2
6
INF2C
```

Any of these are valid answers : INF2C, coursework, is, due, on, Wednesday

```
> ./most_freq_word

input: ().
output:
0
-1
0
```

Approach this task by first writing an equivalent C program `most_freq_word.c` and then translating this C program into a MIPS program `most_freq_word.s`. Before translating, you should compile and test your C program. Ensure it is working correctly before starting on

the MIPS code. To help you get started with the C program, a skeleton `most_freq_word.c` is supplied. Furthermore, you should reuse the Word Finder of Task A to get different words in the sentence and then generate the list of word-frequency pairs for all uniuqe words.

As in Task A, use the C code to comment your MIPS code, and put additional comments for MIPS specific details. In your C program, your comments can focus on explaining the higher-level features of your program, so your comments in the MIPS code can mainly just concern themselves with the MIPS implementation details.

Finally, the output of the MIPS program must have the same exact format as specified above and found in the C code provided to you. Failure to follow the formatting convention in the output will likely fail the automated testing part of marking, resulting in a points deduction.

2 Program Development and Testing

Ultimately, you must ensure that your MIPS programs assemble and run without errors when using MARS. When testing your programs, we will run MARS from the command line. Please check that your MIPS programs run properly in this way before submitting them. For example, if the MARS JAR file is saved as `mars.jar` in the same directory as a MIPS program `prog.s`, the command

```
java -jar mars.jar sm prog.s
```

at a command-line will assemble and run prog.s. The `sm` option tells MARS to start running at the `main` label rather than with the first instruction in the code in `prog.s`. When running MARS with its IDE, checking the setting *Initialize Program Counter to global 'main' if defined* on the *Settings* menu achieves the same effect.

MARS supports a variety of pseudo-instructions, more than are described in the MIPS appendix of the Hennessy and Patterson book. In the past we have often found errors and misunderstandings in student code relating to the inadvertent use of pseudo-instructions that are not documented in this appendix. For this reason, only make use of pseudo-instructions explicitly mentioned in the appendix.

When writing MIPS program, use `read_string` instead of `read_char` syscall as `read_char` has been found to be unreliable.

Finally, ensure that your C program for Task B compiles without warnings or errors and runs properly, as it will be assessed as well.

3 Submission

Submit your work using the following command from a prompt of a DICE machine:

```
submit inf2c-cs 1 find_word.s most_freq_word.c most_freq_word.s
```

Note that you are allowed to submit multiple times (i.e., re-submit) as long as it is before the deadline. New submissions overwrite old ones, and lateness will be judged based on the time stamp of the files present in the directory. We have no way to verify or retrieve files you may have submitted before the deadline, so make sure your final submission is on time.

Unless there are special circumstances, late submissions are not allowed. Please consult the online Undergraduate Year 2 Handbook for further information on this. If you feel you have special circumstances that warrant an extension, contact the ITO using their support form. Do *not* contact the lecturer, TA, etc. – they cannot grant you an extension.

4 Assessment

Your programs will be primarily judged according to correctness, completeness, and code quality. In terms of code quality, we are looking for well-structured and easy-to-read code. This means using functions, comments, and neat formating (such as indentation) in both C and MIPS programs.

When editing your code, please make sure you do not use tab characters for indentation. Different editors and printing routines treat tab characters differently, and, if you use tabs, it is likely that your code will not look pretty when we come to mark it. If you use `emacs`, the command `(m-x)untabify` will remove all tab characters from the file in a buffer.

5 Similarity Checking and Plagiarism

You must submit your own work. Any code that is not written by you must be clearly identified and explained through comments at the top of your files. Failure to do so is plagiarism. Note that you are required to take reasonable measures to protect your assessed work from unauthorised access. Detailed guidelines on what constitutes plagiarism and other issues of academic misconduct can be found at:

`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

All submitted code is checked for similarity with other submissions using software tools such as MOSS. These tools have been very effective in the past at finding similarities and are not fooled by name changes and reordering of code blocks.

6 Questions

If you have questions about this assignment, ask for help from the lab demonstrators, your Inf2C-CS tutor or the course organiser. Alternatively, ask your questions on the course Discussion Forum (linked to from the course home page).

October 11, 2016