# Inf2C Computer Systems

# Coursework 1

# MIPS Assembly-Language Programming

**Deadline: Tue 27 Oct 2015, 16:00**

Boris Grot, Rakesh Kumar

## 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 `word_finder.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). Furthermore, a word may or may not have a meaning e.g. vk8s92p is a valid word.

Sample interaction with your program should look like:

```
input: The first INF2C-CS coursework is due on 27-Oct-2015 (Tuesday).
output:
The
first
INF2C
CS
coursework
is
due
on
27
Oct
2015
Tuesday
```

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 `word_finder.c`. You can compile this program at the command prompt on the DICE machines with the command:

```
gcc -o word_finder word_finder.c
```

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

```
./word_finder
```

For convenience, this C program includes definitions of functions such as `read_string` and `print_char` which mimic the behaviour of the SPIM system calls with the same names. Derive your MIPS program from this `word_finder.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 all single byte or single word variables can be held just in registers rather than the data segment. 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.

## 1.2 Task B: Palindrome Finder

In this task, you have to write a palindrome finder in both C and MIPS that finds and prints all the palindromes in the input sentence. A palindrome is a word, phrase, number or other sequence of characters which reads the same backward or forward. However, for the purpose of this task, a valid palindrome is restricted to a single word. (The definition of "word" stays same as in Task A)

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.
- A valid palindrome can only be a single word, number or alphanumeric. Examples:

  Valid palindromes: "level", "8448", "a9c9a". Invalid palindrome: "My gym"
- Shortest palindrome is at least 2 characters long.
- Palindromes are case insensitive. ("KayAk" and "A9c9a" are a valid palindromes)
- Palindromes should be printed in the order they are encountered.
- If there are no matches, the program should print the message "No palindrome found".
- The only C library functions that can be used are `getchar, fgets, putchar and printf`.

Sample execution of the C program:

```
> ./palin_finder

input: not-a-nun
output:
nun

input: I am a man.
output:
No palindrome found

input: I got my Honda Civic in 2002.
output:
Civic
2002
```

Here '>' is the Unix command-line prompt, '`input: `' is a prompt printed by the program, and '`not-a-nun`' is text input. After this text is keyed in and the return key

is pressed, the program prints the palindromes found. The process continues until the user exits the program (e.g. CTRL+C). Note that only one palindrome should be printed per line. Palindromes can be printed either in the same case as they appear in the input sentence or completely upper/lower case.

Approach this task by first writing an equivalent C program `palin_finder.c` and then translating this C program into a MIPS program `palin_finder.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, an outline `palin_finder.c` is supplied. Furthermore, you can reuse the Word Finder of Task A to get different words in the sentence and then check whether the individual word is a palindrome or not. We recommend structuring the code into functions for ease of development and readability.

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.

# 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.

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

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.

# 3 Submission

Submit your work using the command

```
submit inf2c-cs 1 word_finder.s palin_finder.c palin_finder.s
```

at a command-line prompt on a DICE machine. Unless there are special circumstances, late submissions are not allowed. Please consult the online Undergraduate Year 2 Handbook 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 both C and MIPS 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 aspects of your code.

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. Detailed guidelines on what constitutes plagiarism can be found at:

`http://web.inf.ed.ac.uk/infweb/admin/policies/guidelines-plagiarism`
.

All submitted code is checked for similarity with other submissions using the MOSS system[1]. MOSS has been effective in the past at finding similarities. It is 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).

12th October 2015

[1] `http://theory.stanford.edu/~aiken/moss/`