Inf2C Computer Systems

Coursework 1

MIPS Assembly Language Programming

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

Instructor: Boris Grot

TA: Priyank Faldu

The aim of this assignment is to introduce you to writing MIPS assembly code and C 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 asks 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 academic misconduct from Undergraduate year 2 student handbook available at http://web.inf.ed.ac.uk/infweb/student-services/ito/students/year2.

1 Preliminaries

For the purpose of this exercise, input sentence consists of one or more characters from alphabets (`a-z` and `A-Z`), punctuation marks `,.!?-` (i.e., comma, period, exclamation, question mark, hyphen) or white-spaces in any order and combination. A word is defined as a sequence of one or more characters from alphabets (`a-z` and `A-Z`) or a hyphen (`-`). A word may not contain punctuation marks `,.!?` (i.e., comma, period, exclamation, question mark). As explained next, hyphen is a special character which can be part of a valid word or can be considered a punctuation mark based on the surrounding

characters. Note that a word or a sentence might not have a meaning in English or any other language.

Hyphenated words: If hyphen is immediately surrounded by valid word characters (i.e., `a-z A-Z`) on both sides with no spaces between the hyphen and immediately preceding and following characters, the word is considered a *hyphenated word* and should be treated as a normal single word. In all other cases, a hyphen is considered a punctuation character that is not part of the word. For instance, `energy-efficiency` is considered a single word, while `energy- efficiency` is considered two words: `energy` and `efficiency` with hyphen (along with following white space) as a punctuation mark separating the words.

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.

Task consists of writing a MIPS program `find_word.s` that outputs the list of words from an input sentence. It outputs all the valid words in a given input sentence - one each per line.

Sample interaction with your program should look like:

```
Enter input: The first INF-CS   -   course-work,  is due on last-wednesday of Oct.
output:
The
first
INF-CS
course-work
is
due
on
last-wednesday
of
Oct
```

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`.

For convenience, the 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 `find_word.c` C program.

Do not try optimizing your MIPS code. 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.

1.2 Task B: PigLatin Converter

In this task, you will write a program in both C and MIPS that given an input sentence, converts all of its individual words into PigLatin[1] using the set of rules defined below while preserving the relative position of punctuation marks and white-spaces.

For a word that contains at least one vowel (i.e., `a`, `e`, `i`, `o`, `u`, `A`, `E`, `I`, `O`, `U`):

- Remove all the characters before the first vowel (if any), append the removed character sequence at the end of the word followed by `ay`. (e.g., `thanks` becomes `anksthay`)
- If the word starts with a capital letter (i.e., `A-Z`), transformed word should also start with a capital letter. (e.g., `Say` becomes `Aysay`)
- If the word contains only capital letters, transformed word should also contain only capital letters including the suffix 'AY'. Note that hyphenated capitalized words are allowed, and the hyphen should be preserved. (e.g., `SIMP-LE` becomes `IMP-LESAY`).

For a word that does not contain any vowels: `ay` should be appended at the end of the word with appropriate capitalization rules explained above and no other changes to the word (e.g., `ty` becomes `tyay`).

Sample interaction with your program should look like:

```
input: Aye-aye captain, Say    SIMPLE  thanks! ty.
output: Aye-ayeay aptaincay, Aysay    IMPLESAY  anksthay! tyay.
```

Approach this task by first writing an equivalent C program `convert_pig_latin.c` and then translating this C program into a MIPS program `convert_pig_latin.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 `convert_pig_latin.c` is supplied. Furthermore, you can reuse the Word Finder of Task A to get different words in the sentence and then convert each of them individually. 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.

[1] https://en.wikipedia.org/wiki/Pig_Latin. Note that the rules used in this assignment slightly differ from those on the Wikipedia page. You must follow the rules provided here.

1.3 Assumptions and Restrictions on Program Inputs

Your programs in both tasks will be tested against input sentences that adhere to the following rules:

- An input sentence will be a maximum of 1000 characters including white-spaces.
- Multiple consecutive white spaces or punctuation marks are allowed and should be preserved as-is.
- A single word will contain a maximum of 50 characters.
- Each word must satisfy one and only one of the following capitalization constraints
  - Exactly one capital letter at the beginning of the word (e.g., `Fir-st`), or
  - All capital letters which may also contain hyphen (e.g., `ALL-CAPITAL`), or
  - No capital letters anywhere in the word (e.g., `none`).
- A word may contain at most one hyphen character. Hyphens, being a non-letter character, are not subject to capitalization constraints.

1.4 Restrictions on the use of C Library Functions

The only C library functions that can be used are `fgets and printf` which are called within the functions `print_char, print_int, print_string and read_string` provided with the C files.

1.5 Output

For your convenience, the `output` function is provided in both C files. In convert_pig_latin.c file, you are required to populate `output_sentence`, a character array parameter of the `output`, with your converted sequence that terminates by a null character. Make sure your program doesn't print anything other than what is printed by the `output` function as it may interfere in automated marking. In MIPS programs, you should output exactly the same way as done in C programs.

2 Program Development and Testing

2.1 C

You can compile C program written in file, say prog.c at the command prompt on the DICE machines with the following command:

```
gcc -o prog prog.c
```

If compilation succeeded without any errors, it creates an executable `prog` which can then run by entering:

```
./prog
```

2.2 MIPS

You will need to choose what kind of storage to use in MIPS for all variables from the C code. The programs for Task A and Task B are small enough that all single byte or single word variables can be held just in registers rather than the data segment. However, if you use any arrays, they will need to go into the data segment. Use the `.space` directive to reserve space in the data segment for arrays, preceding 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.

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`, running following command at the command-line, assembles and then runs prog.s.

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

Caveats:

1) 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, marking the check-box for *Initialize Program Counter to global 'main' if defined* on the *Settings* menu achieves the same effect.

2) MARS supports a variety of pseudo-instructions, more than the ones that 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, make sure you only use pseudo-instructions that are explicitly mentioned in the appendix.

3 Submission

Submit your work using the command

```
submit inf2c-cs cw1 find_word.s convert_pig_latin.c convert_pig_latin.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 student 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 Academic Misconduct

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 [2]. 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 any questions about this assignment, you may talk to Inf2C-CS lab demonstrators, tutor or the course instructor. Alternatively, you may also post questions on the online course discussion forum at https://piazza.com/ed.ac.uk/fall2017/inf2ccs/home.

October 10, 2017

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