CMSC 216 Exercise 4 Fall 2021

MIPS Assembly Basics Exercise **See Piazza for due date**

1 Overview

The objective is to start to prepare you to write functions in assembly. Assembly has a different vocabulary from C and Java (eg, instructions on registers instead of expressions on variables), and a different structure (eg, branches and labels instead of conditionals and blocks).

You can discuss this exercise with your classmates, but you may not exchange any code nor write code together.

2 Context: Use of Assembly in Practice

Assembly language is used in several key situations. First, when a machine first starts up, the code that reads in the operating system is written in assembly. You might call this the "boot loader". Second, when software needs to interact with hardware (eg, IO adaptors), it uses special instructions (eg, in, out) or special registers that aren't exposed to C code. Third, "hand tuned" assembly is used when a routine needs to be optimized for size, speed, or to ensure specific timing. Only extremely important or frequently used code gets this treatment, but on small microcontrollers without much memory for instructions, optimization for size is common.

In practice, assembly routines are called from C, but we won't do that here (because our testing platforms currently don't have a C compiler that generates MIPS assembly).

3 Grading Criteria

Your assignment grade will be determined with the following weights:

Public tests 100%

IMPORTANT: We will look at your code. You will lose most of the points if you use a C-compiler to generate MIPS code, or if you do not follow conventions on stack accessing and function calling (see section 7).

4 Academic integrity statement

Please **carefully read** the academic honesty section of the course syllabus. We take academic integrity matters seriously. Please do not post assignment solutions online (e.g., Chegg, github) where others can see your work. Posting code online can lead to an academic case where you will be reported to the Office of Student Conduct.

5 Starter files

This description is in the 216public exercise_descriptions directory. Copy the directory `assembly_basic` from the 216public exercises directory to your 216 directory. The .submit file for this exercise is in that folder.

6 Specifications

You will implement three functions in MIPS assembly. Their prototypes, in C syntax, are given below.

- `int max(int x, int y)`: Parameters `x` and `y` are passed in registers. Returns the max of `x` and `y` in a register. Eg, `max(8, 9)` returns `9`.
- `int mash(int x, int y)`: Parameters `x` and `y` are passed in the stack. Returns the value of `10*x + y` in a register. Eg, `mash(8, 9)` returns `89`. Assume inputs do not cause overflow.
- `int strlen(char* str)`: Parameter `str` is passed in a register. Assume that `str` points to a nul-terminated string. Returns the length of the string in a register.

We supply template files `max.s`, `mash.s` and `strlen.s`, one for each function. We also supply drivers for these functions, in files `max_driver.s`, `mash_driver.s` and `strlen_driver.s`. Consult the drivers to determine the precise locations of arguments and return values, ie, which registers and/or where in stack.

The key reference documentation for the SPIM instruction set is HP_AppA.pdf, pages 48–72 (also available in the class page resources).

7 Important

1. **To run spim and qtspim in a shell in Grace, you must first run the command "`module load spim`" in that shell**. To avoid typing this for each new shell, add a line with this command to your `~/.cshrc.mine` file (and then run "`source ~/.cshrc.mine`" or logout and login).

2. Your code must be written in the template files `max.s`, `mash.s` and `strlen.s`.

3. Labels representing the functions are already included in the template files. You may need additional labels for branch targets. Do not have any label starting with "`main`". (Each driver has a `main` function.)

4. Do not remove the existing comments in the template files. Add comments to your code as you see fit.

5. Do not modify the driver files.

6. To test your function, concatenate the driver file and your file and execute that resulting file. For example, do

   ```
   cat max_driver.s max.s > max_prog.s
   ```

   and execute file `max_prog.s` in qtspim or in command-line spim. The latter is illustrated below (see `max_test1`); the first three lines are typed in, the last line is output.

   ```
   % spim -file max_prog.s
   8
   9
   9
   %
   ```

7. Your grade will be based on the results obtained from the submit server. It is your responsibility to verify that your program generates the expected results on the submit server.

8. **Obey conventions for accessing quantities on the stack, otherwise you will lose points**. In particular, access arguments and local variables on stack via offsets to the frame pointer.

9. **Obey function calling conventions, otherwise you will lose points**. If your function uses a callee-saved register, it should be saved before using it and restored before returning, **regardless of whether the supplied driver depends upon that register**. **For testing, we may use other drivers with the same passing conventions for arguments and return value**.

Although this assignment is not graded for style, use good style. Labels are flush-left, instructions are indented, and comments at the end of code lines are further indented. Instructions should line up, and comments after instructions should line up. The assembler, unlike `make`, does not *need* a literal tab character for indenting; spaces are fine. Emacs "assembler-mode" will try to help with this formatting; other editors may do the same.

8 Submitting your assignment

In the assignment directory (assembly_basic) execute the command **submit**. For a web submission, submit a zip of `max.s`, `mash.s` and `strlen.s`.