Tags: #cec32x, #lab, cec32x_lab_44–Lab9_sorting_in_asm

Lab 9. Sorting in Assembly and Unit Test

Introduction

We have practiced unit test and TDD before. In this lab, we use TDD to develop assembly functions for array sorting.

The main work of this lab is to program assembly functions to sort **int16_t** arrays in ascending order or descending order. You **are suggested to use** the code on page 16 of the class notes for Module 10; this way, you have something with correct algorithm to stat with.

In the provided code, there are two C functions, named `rank_ascending_c` and `rank_decending_c`, which *are programmed* to sort the **int16_t** arrays. The assembly functions to be programmed will do the same thing as these two C functions.

Lab requirements

There are a couple of requirements:

- When writing your assembly code, use the same naming style as the C function; replace the trailing `c` with `s` to signify the functions are the assembly version.
- You need to use `PUSH` and `POP` instructions to save/recover the environment for the caller if you use registers beyond and include r4 in your assembly function.

Lab demo

Using TDD, the lab demo is extremely simple—we just need to run the test target; as long as we can pass all the tests, we are pretty sure the functions you write should be fine if the tests are well defined.

Tasks for the workshop

Download the zipped file for this lab and unzip it.

You will see that there are two targets for this project. (See the text window on the right of the **Load** button.) You should be able to build the **Application** target. However, you **cannot build** the **Unit test** target; you will see errors and warnings when you build it. You need to work out the following tasks to finish the building and pass the unit tests of this test target. For all tasks below, we assume we use the the **Unit test** target.

Task 1. (10 points) Removing the errors and the warnings for mismatch of the argument types

To remove the building errors for the **Unit test** target, you need to comment out statements on lines 63 and 64 in `testFunctions_all.c` as you do not have these functions defined (programmed) yet.

To fix the warnings for the mismatch of the argument types, you need to fix the mismatch of the data types in the functions under test, `swap_c_`, `rank_ascinding_c`, and `rank_decending_c`, and the provided test data. These files are defined in the `functions.c` file. Note that we need **int16_t** arrays, but the provided swapping/sorting functions are designed for **unint32_t** arrays. Fix these problems by changing the data type of the arguments of the C functions under test. Change the internals of these functions as well when needed so that you can pass the tests in lines 59 to 61.

Note that the implementation of the `rank_ascinding_c` function is different than the implementation provided on page 16 in class notes of Module 10. The key difference is in the manor of loop control, which can be done in different ways.

Task 2. (10 points) Adding new tests for the assembly functions

You need to duplicate the tests in `test_rank_ascending_c` and `test_rank_descending_c` and change the duplicated version to these for testing the assembly functions to be programmed in this lab. Note that you need to change the trailing `c` to `s` to signify these new functions are for the assembly functions. Note that you have to make sure you use the `s` version of functions in these new test functions. You also need to have a skeleton code for each of the assembly functions to start running the test cases.

Task 3. (45 points). Programming the assembly function for ranking in ascending order.

Now, you need to program the assembly function for ranking in ascending order step by step. Again, you are suggested to get started from the provided assembly code on page 16 of the class notes for Module 10. (Starting from the C code of `test_rank_ascending_c` is more difficult and error prone, but this is absolutely doable.)

Task 4. (25 points). Programming the assembly function for ranking in descending order.

Now, you need to copy the code in Task 3 and modify it so that it will perform ranking in descending order. You are suggested to change as less code as possible when writing this function from the previous one.

Submission of the work

(10 points for cleanliness).

Again, there is no demo for this lab. Just submit the pdf file containing the following parts.

- A proof for the completion of Task 1; a screenshot for the build result showing 0 warning will do.

- A proof for the completion of Task 2; include the code snippet in the report.
- A proof for the completion of Tasks 3 and 4; the screenshot of the printf results of the final running of the **Unit test** target will do.