EE 180

Winter 2022

EE 180 Lab 1

Merge sort in MIPS Assembly

Due: Tuesday 18th, 2022 (11:59pm)

1. Goals

This assignment is intended to familiarize you with the MIPS instruction set and to test your understanding of pointer implementations and procedure call conventions. In this lab you will use SPIM, a 32-bit MIPS simulator, to execute MIPS instructions.

This assignment must be completed in groups of two students. You may discuss strategies with other groups, but the code each group submits should be the work of its members only. Use gradescope code KYWVRP. **Please do not submit two copies of the same subsmission under two different names.**

2. Your Task

Implement merge sort using the MIPS instruction set outlined in the SPIM documentation and in lectures.

Look at `mergesort.c` for a C version of the merge sort routine. Your job is to add the required assembly code to the skeleton file, `mergesort.s`, such that the assembly code has the same functionality as the C code.

Download the starter code from Canvas. To access rice and copy the starter code, use the following commands. This will create a `lab1` directory in the current directory with the necessary files for completing, running, and testing the lab.

Note: In order to access the Rice servers outside of the Stanford campus, students must setup their VPN to access the Stanford systems remotely: https://uit.stanford.edu/service/vpn

```
scp lab1_starter.tar.gz yoursunetid@rice.stanford.edu:~
```

EE 180

Winter 2022

```
ssh yoursunetid@rice.stanford.edu

cd ~

mkdir your-path-here

mv lab1_starter.tar.gz your-path-here

cd your-path-here

tar -xf lab1_starter.tar.gz
```

Comments within `mergesort.s` clearly specify the functionality that needs to be added.

2.1. Running the Task Locally (Optional)

Students running a Linux distribution on their own machine or a virtual machine may choose to run the assignment by installing SPIM locally. Those running a Debian distribution such as Ubuntu may install SPIM 8.0 as follows:

```
sudo apt-get update && sudo apt-get upgrade && sudo apt-get
install build-essential spim
```

Students running other flavors of Unix may manually build SPIM from the source code we provide on Canvas. We recommend using either the Rice servers or Ubuntu. SPIM is an old software, as such, depending on your local system, you may need to individually resolve library issues to build SPIM from scratch.

`Spim.tar.gz`

2.2. Running the Task on the provided VM

A VM is provided through virtual box, which students may choose to use for this assignment. The provided VM should having spim already pre-installed.

Please refer to EE180 ISE VM document for installation instructions.

3. Running the Code

You will run your merge sort implementation with:

```
spim -file mergesort.s < inputfile
```

The input file should be a text file containing integers separated by newlines. The first number in the file specifies the number of integers in the array to be sorted. The subsequent integers are the actual array elements.

Included with the starter code is a script, `test.sh`, which will diff the output of your `mergesort.s` with that of mergesort.c after compiling it with GCC. To use this test script, run the script using:

```
./test.sh inputfile
```

One test case input file `sampleinput.txt` is provided for your convenience, but you should construct several of your own test cases to test your code.

4. Guidelines

- Be sure to follow the procedure call convention detailed in Appendix A.6. You may also find Appendix A.9 and A.10 very helpful.
- Make sure that all function arguments are passed according to the established convention!
- Follow the register and procedure call conventions (i.e., which registers are intended for what use and the sequence of steps in a procedure call). Appendix A of the textbook is a good reference in this respect. Compilers have to follow the conventions, so you should too.

5. Hints

- Be sure you have a good understanding of how a procedure call works before you attempt to code.
- The assignment is reasonably straightforward and should take on the order of 100-200 lines of assembly.

6. Requirements

- Write a README file containing the name and Stanford email address of you and your partner.
- By 11:59 PM of the due date, submit your `mergesort.s` and `README` files on gradescope under the Lab 1 assignment. If you're not already added to the course on gradescope, you can find the code for the class on gradescope on the course's canvas website. Be sure to submit as a group (you can add two people to one submission on gradescope).
- The TAs will be testing your code using the SPIM installation that runs on the rice machines. SPIM is available from apt-get and from MacPorts, so feel free to download it and implement your solution on your own computer, but remember that it is **your** responsibility to make sure that your code runs correctly on the rice machines.
- Add code to `mergesort.s`, but do not change any of the original skeleton code, which controls the input prompts and printing the output. If you think there's a bug with this section of the code, please email the TAs via the staff e-mail list. Modifying the I/O instructions may cause your assignment to be graded incorrectly.

7. Notes on Grading

- We will be using an automatic script to grade your submission, so the format of your output must exactly match that of the C program. (Verify that the output of `test.sh` is empty for all valid input files.)
- Coding efficiency will not be graded (i.e., total number of instructions executed, instructions per loop, efficient memory usage, etc.). All we are looking for is functionality. However, you should implement your code the same way it is implemented in the `mergesort.c` file; when there is a recursive call, you should make a recursive call, when there is dynamically allocated memory, you should dynamically allocate memory, etc. Also, use reasonable formatting and white space in your code.
- We will use several inputs in addition to the provided sample to grade your submission, but they will all have the same format as the sample. We will not try to break your code with pathological inputs, such as numbers which exceed the range of 32-bit integers or huge lists of numbers that could cause your stack and heap to collide. However, you should test on large numbers and large array lengths to ensure your code is robust.