CS 350c, Spring 2016, Laboratory 2
Sorting and Performance Estimation

Assigned: Thursday, March 10, 2016
Due: Tuesday, April 5, 2016, by 10 am

1   Introduction

In this lab, you will learn about and use the specification of our simple microprocessor (SM) by implementing the quicksort and mergesort algorithms, and then using them in a way that minimizes execution time. This will involve coding these two algorithms and then running them on several test data files.

The SM specification gives a time in SM specification-level cycles; this is a crude approximation of the time that would be required of a simply pipe-lined implementation of SM. In fact, in Laboratory 3 (to be assigned in early April), you will be asked to provide some implementation details for a more accurate pipe-lined SM model.

This laboratory is designed to help you more thoroughly understand the effect caches may have on your programs. And, given an understanding of how caches actually operate, you will compare, contrast, and customize your use of the above two algorithms to achieve the best performing sort for several different sized datasets.

2   Logistics

You are expected to work on this lab alone. However, you may communicate with others concerning your understanding of C code and the various tools, e.g., the compiler, assembler, linker, loader, and other systems issues. And, you may discuss the specification of the class microprocessor and any tools that you use – including anything provided to you for our class. The results that you submit in response to laboratory must be created and provided by you alone.

Any clarifications and revisions to the laboratory will be posted on the top-level course web page.

https://en.wikipedia.org/wiki/Quicksort and **https://en.wikipedia.org/wiki/Merge_sort** provide information about these sorting algorithms, as do many other references. As the SM memory holds 16-bit integers, each of your implementations (of quicksort and mergesort) should sort an array of 16-bit integers.

3 Handout Instructions

You will find the file `sm-programming.tar` referenced on the homework page of the class website. You will need to download this file so you can use its contents. There may be changes or updates, so please be sure to download the latest version – check the top-level class web page for any updates or corrections.

This "tar" file will contain a copy of our C-language-based SM microprocessor specification. You will note that this specification loads an initial memory image from a `<filename>` specified on the command line when running the SM emulator.

The format of the SM emulator input file is designed to be simple. Each line of an input file must contain two numbers: a natural-number memory address (0..65535) and an integer value (-32768..65535) – this *expanded* integer range for the input allows one to read either natural numbers or integers; allows for easier input for instructions. The version of the SM microprocessor specification included with this laboratory is designed to accept three arguments: `<n>` `<size>` `<filename>`. `<n>` is the number of instructions you want the SM emulator to execute, `<size>` is the size of the array to sort, and `<filename>` contains lines with address-value pairs.

We will supply test datasets. These datasets will be in the same format as the SM emulator input file. In fact, all you need to do is to concatenate your "object" file with the dataset file(s).

4 Evaluation

You are expected to write a report that explains timing results of running your mergesort and quicksort code on several datasets. Below is a very simple dataset. We will provide more datasets a week before the due date. All of our datasets will start at memory location 4096.

```
4096 3
4097 8
4099 9
4098 -8
4100 -3
4101 2
4104 22
```

Note, these addresses are not contigious, but the region we sort will be contiguous. If we forget a value, it will be 0 (zero) because each memory location is initialized to 0 (except the top memory location that contains the size). There is not particular order required, as shown in the example above.

Likely, you will need to "tune" the usage of your sort routines. What do we mean? Well, the overall cost, as reported by the SM emulator, may be lower by using one kind of sort or the other or both, but on various parts of the sorting problem. For instance, it may be "cheaper" to break the sorting problem into smaller pieces, sort the pieces with one algorithm, and then use the other algorithm to finish the sorting process.

The maximum score for this laboratory is 100 points. The value of the individual components is as follows.

- An explanation of the code that you wrote that documents your sorting algorithms and your rationale for the correctness of your SM code. And, you must submit your source code (40 points).

- Timing results (as provided by the SM emulator) from running your sorting algorithm on different dataset sizes (20 points).

- An explanation of what sort algorithms you choose for various datasets (to be provided) of different sizes. (20 points).

- A presentation of the timing results of each run (10 points). The various runs are described below.

- Answers to the challenge questions that are listed below (10 points).

5 Running the SM Emulator

To use the SM emulator, you will need to compile "simple-micro-0.6.c" that is provided with the laboratory; this code has been compiled successfully on the departmental Linux machines as well as an Apple Macintosh running OSX 10.11.3. Below is a command sufficient to compile this program so you may run it; you only need to type:

```
gcc -Wall -O2 -o sm simple-micro-0.6.c
```

The resulting compiled program can be run by just typing:

```
./sm <num_of_instructions> <num_of_elements> <filename_with_memory_image>
```

The emulator will print the final value of register 0.

You will spend some of your time for this laboratory studying the cache model of the SM microprocessor. It's important that you understand and can answer the questions posed below, as the answers to these questions will no doubt help your selection of your two sorting algorithms for the various datasets.

After reading the input file, the SM emulator will begin execution at memory address 0; that is, the PC will be set to 0. So, your program needs to work starting at address 0. The first thing your program should do is to set the stack pointer to 0xFFFF; that is, put the stack pointer at the top of the memory.

We will provide datasets that are also in the `<address>` `<value>` format, where each line of the file will contain exactly one such pair of numbers. The `<address>` must be a natural number in the range 0..65535, and the `<value>` must be an integer in the range -32768..32767. Since the program and data input file formats are identical, they may simply be concatenated – be careful.

Below are some questions about this laboratory. It is important that you give thoughtful answers to these questions (jointly worth 10%). Your answers need not be more than a few paragraphs, but your answers need to reveal clearly that you understand how the SM memory hierarchy operates.

- How large is the SM cache? How is it organized (e.g., set sizes, number of sets, and so on.)? What is the cache replacement policy? How did you decide which sorting algorithm or algorithms to run?

Hints

First, get the sorting algorithms to work correctly. You can use your own assembler or you can adapt the original SM assembler to your cause.

Hand In Instructions

Please follow the instructions below for turning in your work.

- Make sure you have included your identifying information in your submission. In this laboratory, you will provide several items: the code you wrote to assemble and produce memory images, the memory image that contains both algorithms compiled into SM machine code – and any startup code that you require (e.g., like initializing the stack pointer).

- To submit your lab, you should provide a tar file that contains all the tools you write and your compiled SM binary. This file should be named `<YourUTID>_Code.tar`.

- To handin your sorting laboratory report, please submit it as a PDF file. Name this file with your report as `<YourUTID>.pdf`. The Canvas system identifies you in this manner.

  Submit your laboratory report by using CANVAS.