CS 350c, Spring 2017, Laboratory 2
A y86 Simulator, and y86 Programming

Assigned: Tuesday, March 7, 2017
Due: Tuesday, April 4, 2017, by 10 am

1 Introduction

In this lab, you will learn about and use a specification of the y86 microprocessor by first creating a y86 specification and then implementing the quicksort and fibnocci algorithms. This will involve writing the y86 simulator, coding these two algorithms in y86 assembler, and then running them on several test data files.

Your y86 specification function should exactly specify what the y86 instruction-set architecture means (by way of the C-language program you will write), and as it executes, it will emulate each y86 instruction.

This laboratory is designed to help you more thoroughly understand what it means to specify an instruction set. And, the two algorithms you are asked to implement will help assure that your y86 simulator is functioning correctly.

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/Fibonacci_number** provide information about the quicksort and Fibonacci algorithms – as do many other references. Your algorithms should both be recursive; that is, instead of a iterative solution, you are expected to write recursive y86 programs. A formal specification of the y86 instruction-set architecture will be provided on the class Homework and Laboratory webpage; the careful reader can use this specification to answer specific questions. Those finding and first reporting errors in the class specification will be given extra credit.

3 Handout Instructions

You will find the file `y86-programming.tar` referenced on the homework page of the class website. You may 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 update or correction announcements.

The format of the y86 emulator input file is designed to be simple. Each line of an input file must contain two numbers: a natural-number memory address (0..16777216) and a natural number byte value (0..255). The format for each byte is:

```
(
 ( <address_0> . <byte_0> )
 ( <address_1> . <byte_1> )
...
 ( <address_N> . <byte_N> )
)
```

Before loading and attempting to simulate a y86 program (with the format just above), your y86 microprocessor specification should initialize all registers, the flags, and the RIP to 0. Thus, all of your programs will begin execution at memory address zero.

To keep the memory requirements of your y86 simulator modest, you only need to implement a 16 megabyte (2^{24}) byte memory. All memory addresses should be truncated to the lower 24-bits before any memory read/write access is performed.

We will supply test a test dataset. This dataset will be in the same format as the y86 emulator input file.

4 Evaluation

You are expected to write a report that explains the results of running your quicksort and fibonacci code on any examples we provide Below is a very simple dataset. We will provide a dataset for your quicksort a week before the due date. Any dataset that we provide will start at memory location 4096.

```
(
 ( 4096 . 3 )  ; 8-byte length of array to sort
 ( 4097 . 8 )
 ( 4099 . 0 )
 ( 4098 . 0 )
...
 ( 4104 . 53 ) ; Beginning of array to be sorted
 ( 4105 . 222 )
 ( 4106 . 22)
 ...
)
```

Addresses need not be contigious, but the region we sort will be contiguous, and this region starts at address 4014. Remember, the array is a quad-word array; each entry is 8 bytes in size. Also, remember that array entries may be negative – even though the way we specify memory contents is with numbers in the range 0..255.

If we forget a value, it will be 0 (zero) because each memory location should be initialized to 0 before your simulator starts loading its simulated memory.. There is no particular order required, even though the example above has addresses in ascending order.

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

- Your simulator source code (50 points). This should be documented so we are convinced that you understand what your simulator is doing. We will run your simulator on our own examples; these examples are designed to stress your simulator.
- Your y86 assembler-based quicksort program (15 points).
- Your y86 assembler-based Fibonacci program (15 points). Includes a table indicating how many operations your y86 simulator needs to step to calculate fib(n), for n from 1 to 20.
- A report that describes your simulator and the results of executing the two (quicksort and Fibonacci) assigned programs (10 points).
- Answers to the challenge questions that are listed below (10 points).

5 Running your y86 Emulator

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. This format has been discussed above. Since the program and data input file formats are identical, they may simply be concatenated – so 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 what you have done.

- How many instructions/second can your simulator emulate? How did you organize the memory for your simulator? When programming, did you follow the register assignment discipline? Did you use all of the available y86 instructions? If not, why not? Could your recursive Fibonacci program have executed faster by ignoring the stack discipline? What was the most useful tool you used in developing your simulator? Why was it useful? Other than your assembler and simulator, what tools, if any, did you develop?

Hints

First, get a very simple program (e.g., add two numbers) to work correctly. You should use your own assembler to aid your cause. Once a very simple program is working, you can incrementally increase the

complexity of the program(s) you create. Remember, you have to debug what you create. And, this will, no doubt, cause you to create functions to explore the state of your simulator (the registers, flags, and memory locations) after some number of simulator steps.

Talk with your colleagues, your instructor, and your lab assistant for ideas to implement useful debugging aids or other tools. Remember, you are creating a new tool chain – it is often worth some effort to make your tools easy and productive to use.

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 implement your y86 assembler (already submitted for HW 6, but we want a complete system) and your y86 simulator. Provide your quicksort and Fibonacci code as two y86 assembler-level programs; we should be able to assemble your code using either your assembler or our assembler.

- To submit your lab, you should provide a tar file that contains all the tools you write and instructions to compile y86 tools (assembler and simulator). This file should be named `<YourUTID>_Code.tar` – and this code better run on the departmental Linux computers, because we will use these computer to test and evaluate your submission.

- To handin your 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 everything else for your C-language, y86 simulator, your y86 assembler (already submitted as HW 6), and your two, y86 assembly language programs as a tar file. Additional submission instruction will appear on the Homework and Laboratory webpage close to the time for submission.