Lab 4: Life of Pi

Peter Johnson[1]

E155 Fall 2018 Section 2, October 4, 2018

***Abstract*— This paper describes the set up, programming and testing of a Raspberry Pi such that it can run an assembly language program to sort an array of 12 signed bytes on the Pi.**

I. Introduction

In this lab, the MuddPi Mark IV utility board was not utilized at all. Instead, the lab was centered on setting up and gaining familiarity with the Raspberry Pi. Once the Pi was running I wrote an assembly language program to sort 12 signed bytes on the installed nano editor, compiled it using GCC and then tested the output in the kdbg debugger.

II. Design and Testing Methodology

I approached the design of the assembly-language sorting algorithm by considering available algorithms online. I selected based on effectiveness and ease of implementation. I ended up choosing to implement Bubble Sort. Bubble Sort is probably the most intuitive algorithm to understand. In the algorithm, a value is compared to the value in the next index. If it is greater, they are swapped. This continues until the element has made it to the end or there is a larger element in the next index. The entire list is then looped through. This simplicity means it is easier to implement in assembly than something like merge sort which would require calling sub-functions. In exchange for this ease of implementation, Bubble Sort's run time is $O(n^2)$ where n is the length of the array to be sorted. However this is not really a problem for this application. So this is an acceptable tradeoff.

Once I decided on Bubble Sort, I wrote a version of the algorithm in C instead of going straight into assembly. Once I had the C code written I then translated it into assembly like we did in class. Since we we are using bytes I used LDRB instead of LDR. This worked when they were all positive, but we want them to be dealt with as signed 2's complement numbers. So I switched to LDRSB and this dealt with the signed bits correctly. During this process, I utilized the kdbg debugger to check what my outputs were.

My testing methodology was simple and just consisted of several test cases I wrote. To make it more convenient to see the sorted values I loaded the sorted values into individual registers since it was a little tedious to parse through memory since the list is not displayed linearly. I started with a basic list of just random values both positive and negative. In a more comprehensive test case I used the most positive value, the most negative value, and repeated values. I also wrote a case where every value was the same, a case where the list was already sorted, and a case where the list was the reverse of the sorted version.

III. Code and Schematics

Refer to attached the code. There are no schematics for this lab.

IV. Results and Discussions

All of the prescribed tasks were accomplished: the Raspberry Pi was setup and I wrote an assembly language program that successfully sorts an array of 12 signed bytes on the Pi. The algorithm itself as well as register use are not optimized. The version of the algorithm that I implemented will always run $O(n^2)$ even if the list comes sorted. To optimize it, I would stop the algorithm if the inner loop did not cause a swap. If I was to optimize the algorithm, its worst case would still be $O(n^2)$ but its best case would be $O(n)$ and intermediate cases could be better than $O(n^2)$. After last class, I learned that instead of creating a temporary variable (and another register) when swapping register values, I could accomplish the swap using just XOR. This could be implemented if register usage was an issue. I did all of my access of the Pi directly through a mouse and a keyboard. In the future I may decide to use wireless access over putty, but I did not feel the need to this time. One thing that was odd, was I was able to SSH in without using xlaunch, just Putty. I'm not sure if this matters, as all I did was ls and open my sort.s file in nano.

V. Conclusions

The lab was successfully completed. The Raspberry Pi runs Raspbian, emails its IP to me and can run my assembly-language program to sort 12 signed bytes. I gained experience with setting up Pis, linux environments and writing assembly-language programs. In total I spent 7 hours.

[1]Peter Johnson is with Department of Engineering Harvey Mudd College, Claremont, CA 91711 pjohnson@hmc.edu