Assignment 4: Eight Bits of Madness!

Due by: March 16, 2018 before 10:00 pm
Collaboration: None
Grading: Packaging 10%, Style 10% (where applicable), Design 10% (where applicable), Correctness/Functionality 70% (where applicable)

Overview

The fourth assignment is **all** about hacking 6502 assembly code using the Easy6502 simulator. For most problems we give you a bunch of starter code, so all you have to do is fill in one or two subroutines in the way described (and in the code itself, be sure to read it for all the gory details). Still, working in 6502 assembly is bound to bend your mind, so enjoy! **And don't wait too long to start, you'll have to master a new language and that'll take time!**

Problem 1: Long Arithmetic (20%)

The 6502 is an 8-bit processor (albeit with a 16-bit address space) so it can only do arithmetic on 8-bit quantities. For this problem you need to write two subroutines that can perform **32-bit** addition and subtraction.

Once you understand how the `ADC` and `SBC` instructions work that's actually not too complicated, essentially you just need to do four "small" 8-bit operations to get the desired 32-bit operation. Of course the proverbial devil is in the details, specifically the carry (or borrow) flag. Make sure you take the time to understand how it influences the execution of the basic `ADC` and `SBC` instructions!

Your job? Add the necessary subroutines to the file `thirtytwo.s` of course!

Problem 2: Plotting Assemblies (30%)

The Easy6502 simulator comes with limited graphics capabilities. However, you cannot simply set a pixel at a certain (x, y) coordinate to a certain color, instead you have to figure out where **in memory** that pixel actually lives. As you saw in lecture, **that's tedious!**

So for this problem you'll write a subroutine that performs the tedious calculations automatically. The details are in the starter code, but essentially you'll have to figure out how you can translate a certain (x, y) coordinate into the correct memory address to get the nice animation we showed you in lecture. And just to be clear: (0, 0) is the pixel in the top-left corner whereas (31, 31) is the pixel in the bottom-right corner.

Your job? Add the necessary subroutine to the file `plot.s` of course!

BTW, Peter's best solution takes about 40 cycles to plot a point; how many cycles do you need?

Problem 3: Out of Sorts (45%)

Sorting is always a popular programming problem, and now it's time to hack a sorting algorithm in 6502 assembly. But don't worry, this is not going to be boring at all. The starter code for this problem has everything you need to **animate** the sorting process just like we showed you in lecture.

All you have to do is decide on a simple sorting algorithm based on swapping array elements and then implement it for the 6502. We recommend **selection sort.**

Note that you may get a little tired of how slow the animation is while you debug your code. There's a comment in the starter code that gives a hint as to why it's so slow. If you're excited about 6502 assembly, why not go ahead and fix that inefficiency? (Warning: That's a good bit harder than just writing the sorting algorithm.)

Your job? Add the necessary subroutine to the file `sorting.s` of course! If you choose to improve the animation code as well, please clearly note that in your `README` file and describe how you did it.

Problem 4: A Life of Gambling? (5%)

This is the **challenge problem** for the assignment. Note that it's worth **very little**, but please trust us that it's **a lot** of work. (Doing the problem is extremely rewarding, but since it's **way** out there in terms of complexity we didn't want to force everybody to do it.)

Your job? Implement a graphical (that is animated!) version of Conway's Game of Life! It's up to you whether you start with a random configuration or a specific "famous" pattern, but the entire 32 x 32 pixel screen should be used.

Please note that we are **well** aware of this version and several other published versions, so if you're thinking of copying it, think again. Those 5% are certainly not worth getting caught cheating over, are they?

Deliverables

Please turn in your assignment on Gradescope.

Include a `README` file that briefly explains what your programs do and contains any other notes you want us to check out before grading.

Grading

For reference, here is a short explanation of the grading criteria; some of the criteria don't apply to all problems, and not all of the criteria are used on all assignments.

Packaging refers to the proper organization of the stuff you hand in, following the guidelines for Deliverables above.

Style refers to either to programming style if we are talking about a programming problem (things like consistent indentation, appropriate identifiers, useful comments, generally simple, clean, readable code), or to the clarity and readability of your solution for a written problem.

Design refers to proper modularization and the proper choice of algorithms and data structures for a programming problem, and the proper choice of abstractions for all problems.

Functionality refers to your programs being able to do what they should according to the specification given above; if the specification is ambiguous and you had to make a certain assumption, defend that assumption in your `README` file.

If your programs cannot be built you will get no points whatsoever for programming problems. If your programs fail miserably even once, i.e. terminate with an exception of any kind or dump core, we will take off 10%; do proper error checking and handling! Finally, make sure to include your name and email address in *every* file you turn in (well, if there *is* a way to include that information, otherwise you can leave it out)!