This class was surprisingly interesting. Let’s be honest, nobody gets excited about programming in assembly, so this came as a bit of a shock to me. This class seems to serve as a critical bridge from everything that we’ve done previously (programming 1 and 2, and DSA) to the operating systems class. But I think most importantly, it is also the first class that bridges the gap between CS in theory and CS in practice. I believe that this is the primary motivation behind having the course in the curriculum. One of the sections from the book that really stood out to me was one where they performed an analysis of the runtime of two different sorting algorithms. Prior to this class, to figure out which one was faster we’d use what we learned in DSA and say, “algorithm A has a lower big O complexity than algorithm B, therefore, it is faster. The End.” But in this analysis, when they ran it, they found the opposite. And it was due to differences in how effectively the algorithms utilized the cache, with algorithm A making more calls to main memory and algorithm B being more successful at utilizing the cache. This was a great example of one of my absolute favorite quotes in actions, “in theory there is no difference between theory and practice, in practice there is.” This was my main takeaway from the course, remembering that all the theory in the world still needs to touch real-world hardware to run.

This course was also really important in one other aspect. I typically breakdown the knowledge that a programmer requires into a few major areas. Programming language skills which include things like basic syntax and learning how to accomplish things in a specific language. CS theory which covers DSA knowledge and learning how to structure and organize programs. And there are several other broad areas, but the one this class really helped with was thinking computationally, basically, thinking like a computer in order to solve various problems with a computer. Assembly programming, while admittedly being incredibly frustrating at times (Ok, most of the time), is the purest form of computational thinking that I’ve ever encountered. Programs are executed step by step, with nothing abstracted away from the programmer. I think that this is something that the course could really lean into in the future in order to develop more well-rounded graduates (AKA, more programming assignments please!).

PS. I remember from one of the earlier lectures where you mentioned that past feedback had complained about the level of sassiness in the class. I would like to respectfully disagree with the assessment of that being a negative. Too many classroom lectures today have absolutely no life to them, and are just so, so, boring. Thank you for making this class fun.