### Project Purpose

### The purpose of this project was to emulate the interactions between a computer processor and memory, using processes. By implementing these processes, our goal was to be able to understand the intricacies of inter-process communication while also solidifying our understanding of lower-level operating systems concepts such as main memory, input/output handling, instruction execution cycles, register usage, stack behavior, interrupt handling and memory protections. This project was aimed to build off the fundamentals of computer architecture and solidify the need for processes in operating systems.

### Project Implementation

I implemented this project in C programming language and started by analyzing all the project requirements. From here, I recognized that I would need to use the fork() functions to create two parallel processes (one for processor and the other for memory). At the basic level, the processor would fetch, decode, and execute the instructions from memory. This guided me towards first ensuring the memory process was working. I allocated space for 2000 integer values in memory to store instructions and data. From here, the program would parse the file argument provided at command line execution to load each instruction or data in sequential memory indexes. Special cases for interrupt and offset handling were added to ensure that the sample files provided were loaded into memory.

From here, I turned my attention towards understanding how the processor would handle the instruction execution cycle (Fetch, Decode, Execute) and recognized that the cycle would run until the CPU process read the value 50. This helped establish a while loop I could implement to keep fetching instructions into the corresponding registers, to then decode and execute. For the latter two steps, the project document provided the instruction set which I could then model using a lengthy set of switch cases, based on the value fetched into the instruction register. The hardest part in my project implementation was understanding how the user stack and system stack worked. To overcome this challenge, I drew out a basic diagram of memory, to see how the user stack and system stack grew. From this, I was able to complete my instruction cycle. I further optimized my code by using functional and modular programming principles, implementing functions for common instructions such as reading and writing from memory, initializing memory from a sample file, or even checking if indexed addresses were out of bounds seeing the common case present in several switch cases.

### Project Takeaways

My biggest takeaway from the project was understanding the nuances of inter process communication. Since the processes were running concurrently, I had issues with synchronization of the processes, especially in sample programs which would loop. Using forks and pipes was tricky at first, especially since I didn’t have much exposure to it in prior classes. Understanding how data is communicated between two processes, and the role of pipe descriptors was part of the learning curve with this project. However, through this experience I feel like I’m a lot more competent and comfortable with inter process communication.

The conceptual part aside, I recognized the importance of time management, especially in debugging situations. I’m glad that I started on the project early on, and worked incrementally, as coding the instruction set wasn’t too difficult. However, the challenges arose in trying to structure the logic such that the desired outputs would not only be attained for one sample file, but all of them. Early on, the issue I’d run into in designing my switch cases was when I made a change for instance in how the stack was programmed (write then decrement approach) to satisfy the desired outputs in one sample file would conversely create issues when running another sample file. Debugging is often the most time consuming and laborious part of programming, so being able to have the time to design and test every aspect of the project was critical.