The purpose of this project is to simulate a simple computer system that consists of a CPU and Memory. The CPU and Memory are different processes in the program with the CPU doing all the logic of the program and the Memory storing and fetching instructions given by the terminal/text file. The CPU and Memory cannot communicate with each other through simple means but instead required a pipe for communication. In addition to the computer system, this program is to complete our understanding of forked processes and the communication between them.

I implemented this project through a C program which had fork the main process to create a child process and pipes for communicating between the parent and child processes. The child process in my program is the Memory which loads in the instructions which are given in the samples (text files and assigned at the terminal) and is only able to read and write to memory with an address (writing also includes the data needing to be stored). For the communication, I used two pipes, one for the parent to write and the other for the child to write. The child process will continue in a while loop until the CPU tells the child to quit the program (shutting down the main memory). On the other hand, the CPU of the system which is in the parent process runs inside of a do while loop until CPU calls for instruction 50 which will end the parent and child processes. The CPU will first check for a time interrupt and then if there isn’t a time interrupt called then will load the instruction to be executed from memory. The instructions are executed through if statements and if the instructions call upon another address outside of the user/kernel mode then the CPU will call the termination of the program and close the Memory process. The CPU also stores and has the time interrupt which is executed every x instructions and then reset for the next time instruction. Like stated in the instructions for the project, the user and the kernel are divided into the program and the stack where the stack stores data while the instructions execute, and fetches addresses for other instructions are at pointed at the different ends with different registers. The CPU has the given 6 registers plus any variables required for executing the time interrupt or holding the instruction to be executed.

In my experience, I find that having multiple samples that test different parts of my code helpful for debugging especially when I need to switch between addresses and system calls. For the longest of time, I had bugs for pushing and popping the stack in which was resolved by tweaking the instructions around since I had the address in the wrong order of the read and write to memory instructions. This is where sample2 helped me understand where and which instruction had the problem. I also had a weird bug in which characters were printed twice and later found out that I was not executing instruction 9 correcting. For a while I had problems with sample3 since when I ran the code with a small time interrupt (usually an integer smaller then 20), my code would never complete executing and continue to call for the time interrupt. However, during class it came to my knowledge that this was normal for the code to have this complication. Overall, I found this project helpful in my understanding of the CPU and Memory interactions and how processes work.