**Project Summary**

In this project, we simulated a simple processor interacting with memory. We also demonstrated how processes exchange information with one another when implementing this project. The two processes that were executed were CPU and memory. The memory would contain all the instructions that the CPU needed to fetch and execute. We also gained more understanding of basic operating system like processor interaction with memory, instruction and register behaviors, instruction execution and interrupt handling. Overall the project very well demonstrated the concepts that we have learned in the first unit of this course, and it was good to physically put them into practice.   
 My project was done in C++, so in order for the two separate processes to execute and communicate with each other I had to use the fork and pipe calls. When the fork is called, it creates two processes, a parent process, and a child process that run at the same time. The pipe call is used to send information back and forth between the parent and the child process. The child process was the memory, and once it was called, the program file was read into a memory array of 2000, which represented my simple memory that my CPU would fetch instructions from. After memory was instantiated, the only thing that memory did was read instructions to CPU, or write to the memory array based on the information given by the CPU process. The CPU process was a little more complicated because it consisted of keeping track of all the registers needed in the system, and incrementing the program counter, so we could fetch and execute the instructions in the proper order. So with a program count, and access to reading/writing to memory the CPU was able to read instructions in order to execute them accordingly, and write data to memory as was necessary. The CPU also successfully handled interrupts and system calls. The CPU would process interrupts by entering kernel mode, and transitioning to the system stack, rather than the user stack in user mode. The CPU needed to save all the old registers to the stack before entering the interrupt, so that once the interrupt was processed, the CPU would go back to execution normally.

I do feel that I have learned a lot from this project. Before this I have not really used fork and pipes to this extent, however at the end of this I feel much more comfortable running processes and communicating between them using pipes. I also have a better understanding of the basics of how a processor and memory interact with each other. I did learn concepts like this in computer architecture but they did not stick with me, but after completing this assignment, since I had to manually code the sequence of how CPU interacts with memory myself the concept is much more ingrained in my knowledge. I have to admit I did not have a concrete understanding of jump instructions and interrupts before this assignment, but now these concepts seem almost easy to me!