CS 4348 Project 1 Summary

**Project Purpose**

The project purpose is to simulate a very simple computer system containing a CPU and Memory by using separate processes. With these separate processes, the project facilitates the usage of, in my case, pipes to communicate in between the processes. Additionally, the project features interrupts within the CPU. By completing the project, the goal is to gain understanding of these multiple processes, how the CPU interacts with memory, CPU instruction behavior, how registers are commonly used, how user and system stacks work, interrupt handling, and memory protection. The full list of topics covered by the project is well-discussed within the instructions given.

**Implementation**

I have chosen to create my project in C++ within Ubuntu 20.04.2. The program is broken into five files with two being header files. Note: All function descriptions for the CPU are within the CPU.h file.

The first file, main.cpp is the main driver code. Inside this file, I first read the command line arguments and instantiate the CPU and Memory objects. With the command line argument of text file, we read each instruction and write them to memory. Only then is the main task forked into two processes and the pipes to and from the two tasks are instantiated. We then have the main task, CPU, and the subtask Memory. Within the main CPU task, there is a timer that constantly ticks upon every instruction and is used for timer interrupts. When the timer is the same as the interrupt timer, the timer resets back to zero. Then, in the memory task, the task handles all the messages sent from and to the CPU task. These messages are handled in the task as the Memory itself has no processing aside from reading and writing from the memory.

The second file, CPU.cpp, is the CPU file. Once inside this file, it is essentially single threaded. For every loop, it checks for interrupts, reads the instruction, performs the instruction, and then loops again. For instructions involving memory and stack, the task sends a message through a pipe to the memory task asking it to perform the task. The CPU task then waits for a response, if needed.

The final file, Memory.cpp, is the memory file. This class is very straightforward with an integer array of size 2000 containing the memory followed by two functions for reading and writing to an address.

**Note**:

* In-depth explanations of all files are within the header files.
* When an interrupt happens, unless the instruction set has been written specifically to resume where it left off, there is a high chance the output will not be as expected due to not saving the AC, X, and Y values not being saved.

**Personal Experience**

The overall experience was pleasant. Previously, I had a significant amount of C++ experience, but my last job focused more on Python. As such I was a bit rusty. This program gave me a good chance to brush up on C++. A more project specific experience was very pleasant and straightforward once I overcame the hurdle of understanding the instructions. Some of the instructions were unclear to me and I had to ask other class members what they meant. After fully understanding the instructions, the process was straightforward. Although, there were a few ways to implement certain things, such as the stack pointer, that I was unsure of the ‘correct’ way to implement to adhere to the output tests. Overall a fun project.