Tim Sorsby

CS4348.005

Summary

The purpose of this project was to explore the interactions between the processor and memory and how the two coexist in the fetch-execute cycle as well as understanding how processes could be utilized in order to emulate that environment.

My implementation used Java and I split my code primarily into two classes, one for the processor called CPU.java and the other for memory called Memory.java. Initially, the CPU class created a process targeting the Memory file, which proceeded to read in an input file containing instruction codes and data into an array. It was from this array, the subsequent requests from the CPU would’ve been processed from.

The CPU was partially responsible for sending requests over to the process via piping to its input and output streams. This was done through read and write requests. A read request sent a specific address over to memory, from that address the process executing the memory file would look up the memory address of the desired instruction from the array, and then it will return that instruction back to the processor. A write request sent a specific address along with some data over to memory, from there the process would look up the address location in the array and overwrite the data stored with the new data.

Another responsibility of the CPU was, of course, to process and/or execute the instructions once they were fetched out of memory. Each instruction code was equipped with a description, based on the specifications in the project instructions, that would go on to manipulate a variety of registers (pc, sp, ir, ac, x, and y). I put each instruction definition, the logic for the instruction itself, inside of its own function labeled generally as “op[instruction code]”. So, for example, the instruction code 1 was a function called “op1” in my code. Once all of the instruction definitions were created, I made a processing loop to continually pipe read commands to the process and fetch the *next* instruction from memory. With each new instruction read from memory, a simple if-else-if branching technique was used to determine how that instruction should be processed.

In addition to the responsibilities, the CPU class was also keeping track of the current mode in use as well as an interrupt timer. Initially, user mode was being used. Per the project instructions, user mode resides in those addresses in the range 0-999. While in user mode, all read and write requests to memory from the processor stayed within that range; if in the case an address was trying to be accessed outside of that range, an error was thrown to the console alerting of a memory violation. There were three mechanisms in place that could switch the mode, two for switching *into* kernel mode (first being an instruction code 29; next being the interrupt timer expiring), and the other for switching *out of* kernel mode and back into user mode, restoring its pc and sp.

My experience implementing all of this was interesting. The biggest obstacles that I faced while doing this project was in the beginning as well as the end. From the start, I found it difficult to visualize what *should* happen. I had tried several architectures but decided upon using two classes in the end, instead of the three or four I had thought of previously. I also had a hard time wrapping my head around the piping aspect at first. I didn’t quite understand how two files could communicate. I understood a process could execute another program while another was being run, but I didn’t know how to send commands from one to the next. What I discovered was through manipulating the input and output streams of the process from the source program that created it, I could configure a protocol to pipe commands to that other program. Once I made that connection, the project came together with an immense amount of research and looking back at the notes for the class. Although, towards the end of the project, I faced another roadblock. In my testing of the project, I had tried to parse through the lines of the sample files in an attempt to trace a bug via the Windows command prompt. It ended up being a lot of me looking up what proper commands were and when I’d try something new, I’d get an error, go look back at documentation, get another error, then back to documentation, and so on. I realized I was spending more time trying to get Windows to understand than I was actually tracing the bug. So, I switched over to MobaXterm and accessed the school’s cs1 server to test my code instead. Had I realized this earlier, I think I could’ve saved a lot of time. When trying to use MobaXterm, there were a few complications on trying to connect to the cs1 server due to me being outside of the school network. I had to get the UTDallas VPN in order to resolve this issue.

Overall, this project was a very educational experience. I got more perspective on how the CPU and Memory communicate with one another and how the fetch-execute cycle works. The parts that really stuck out to me during this assignment were the working with process streams via the piping mechanism, the roles of the registers and what they should be used for, and the logic behind instruction codes and how they commingled with the entire environment. This was very much a project where knowledge was key. After knowing the concepts of what is supposed to happen, implementing the concepts, the programming itself was a lot easier. So overall I thought that this project gave me a deeper understanding on everything that was going on between the CPU and memory and how the data and registers are being handled.