Rahat Ahmed

rxa129130

CS 4348 Project 1 Summary

The purpose of this project was to simulate a simple computer system consisting of a CPU and memory. Both components are to be implemented as two separate processes that communicate. The memory component supports simple reading and writing of a fixed size array of integers. The CPU component has a few registers and supports 31 instructions.

I decided to implement the project in Java using the Runtime.exec method to create a new process. Java automatically pipes the input and output of child processes, so I just needed to use getInputStream and getOutputStream to get the streams to communicate between the processes. The project didn’t specify any protocol for cross-process communication, so I created my own. In the protocol, each command from the CPU to the memory process is separated by new lines and is interpreted as ASCII text. The command is given by a single character followed by comma-separated arguments if necessary. The read command is in the form “r[address]”. The write command is in the form “w[address],[value]”. The end command is simply “e”, and it ends the memory process. The memory only responds to read commands, outputting the requested value and a new line to separate responses. The Memory class implements this protocol and trivially simulates memory with an integer array.

The CPU class simulates registers with simple integer class members. It contains convenience methods to send commands to the memory as described above. It has a run method that executes the fetch-execute loop until the execute method decides the program has stopped. The fetch method simply reads the next instruction to IR and increments PC. The execute method uses a switch statement with cases for each instruction to handle the current instruction in IR. Most instructions were implemented by manipulating the registers and using the convenience methods to read and write memory, as well as for pushing and popping from the stack. In cases where instructions had an argument, the execute method simply fetched the next line of the program into IR. I wrote a method for entering kernel mode that handled the stack switching and register saving, which is called by the timeout interrupt or manual interrupt.

Apart from creating the inter-process communication protocol, most of the project was fairly straight forward. Implementing the CPU instructions was very easy once I had written convenience methods to handle reading and writing to memory. The biggest problem was debugging the program. Since the program was split between two processes, the error stack trace gave no useful information as to what the problem was. Even using a thread to watch the error stream of the memory process didn’t help. I ended up having to add log messages to most instructions and tracing the input program and output myself to see what went wrong. Overall, it was an easy project as long as you didn’t mess up, because once you do, it is very difficult to fix it.