Michael Winkler

Senior Research

Professor Joel Hollingsworth

Change Log

4/8/17:

* Implemented the quantum counter (referred to from here on as the QC). For now, the quantum value is hardcoded into the chip itself, but I’m not committed to this approach; in the future, the QC could read the value from somewhere else.

4/9/17-4/10/17:

* Began integrating the QC into my CPU. More specifically:
  + Added the QC chip to the CPU
  + Added two or gates that or the output of the QC with the signals that determine with the D register should load a value and whether the program counter should load a value. I’ve included these because when the quantum is reached, I want both the D register and the PC to load a new value.
  + Added a not gate that takes the QC’s output as input and has as output ~qcOut. This is fed into the increment input of the program counter, which determines whether the PC increments during a given clock cycle, because I want it to pause for one clock cycle when the quantum is reached.
  + Added a register that holds
  + Added a 16-bit multiplexor that chooses between the output of the A register and the address at which the code that jumps to the control function begins. That multiplexor feeds into the program counter’s input and is controlled by the output of the quantum counter. The result of this setup is that when the quantum is reached, the program counter loads as its next value the address of the beginning of the code that jumps to the control function.
  + Added another 16-bit multiplexor that chooses between the ALU output and the address output by the program counter. Its output is fed into the D register and it’s controlled by the quantum counter’s output. Before I made any modifications, the D register always took the output of the ALU into its input; now, when the quantum is reached, it stores the current PC value. This is important because the code that jumps to the control function expects the previous PC value to be stored in the D register so that it can be pushed onto the stack as a return address.

4/11/17:

4/13/17:

* Realized that I can use the existing function call method in my compiler to call the program control method, rather than writing the code to call it from scratch. However, the method did require some modifications; to that end, I added a function called writeControllCall() into my codeWriter class. I also modified the writeInit() function to call the new function.

4/15/17:

* Focused on testing today. I’ve been testing each component as I go along, but today I made sure everything works together. Specifically, I ensured that the following steps function correctly:
  1. The compiler completes the jack -> virtual machine language step correctly, including the Sys file which contains my control method.
  2. The compiler also completes the virtual machine language -> assembly language step correctly, including the additions I made that allow the jump to the control function to occur.
  3. When I step through a program running on my modified CPU, an interrupt occurs after the number of steps specified by the quantum and a jump to the control function is affected.
  4. After the control function executes, control is correctly returned to the function that was executing before the interrupt occurred.

4/16/17:

* Implemented the “disk” chip, which is not really a separate disk but will be treated as such for the purposes of this project

4/20/17:

* Began integrating my new “disk” chip into the computer. It turns out that this is a much taller task than I’d anticipated because the computer, as it exists, is 16-bit, which means that its registers aren’t big enough to store all the addresses that exist within the disk. I’ll also need to modify the instruction set to contain a ‘write to disk’ command in addition to the existing ‘write to memory (RAM)’ instruction. I’m going to set this aside until I make some more progress on modifying the compiler to compile multiple programs to be run simultaneously.
* Also, I set up the file/folder structure that I’ll use to test the next set of modifications I’m going to make to the compiler, which will allow it to compile multiple programs to be run in parallel. Due to the way the computer is structured, all the programs to be run simultaneously will have to be compiled together. It’s not ideal, and obviously a real computer wouldn’t do it this way, because users need to be able to start and stop programs at their discretion. But adding that level of functionality would require changing the computer to a degree that it would be beyond the scope of this project.