Lab 10: The fetch-execute cycle

Lab note: *This is the final laboratory assignment for the semester. It will extend over two cycles and is worth 40 points.*

Objectives

- to reason about a complex sequential circuit.
- to understand the architecture of a simple computer, particularly how the fetch-execute cycle works in practice.

Chapter 13 of the textbook describes the underlying circuitry of most of the HYMN implementation pictured on page 166. In this lab, we will complete HYMN by designing its control unit. By working on the control unit, we will see how the pieces of the computer fit together and gain a good understanding of how HYMN works.

CPU cycles

In HYMN, the execution of each instruction takes two clock cycles.

Fetch cycle: The current instruction goes into the IR (IR $\leftarrow$ **M**[PC]).

Execute cycle: The computer executes the instruction in the IR. If the PC is not otherwise changed by the instruction, it should increase by 1 (PC $\leftarrow$ PC + 1).

Each cycle consists of two phases — first, the clock is 0, then it becomes 1. During the 0 phase, the data should propagate through the circuit, reaching the registers and memory, but these should not change yet. During the 1 phase, the control unit will send 1 to the registers and memory that should change, so that they will load the data waiting on their input. Data computed during the 0 phase should remain on the wires throughout the 1 phase.

For example, in the fetch cycle, while the clock is 0 the control unit would tell the memory to load the data at the address in the PC. While the clock is 0, the data will propagate through the circuit, reaching the IR, but the IR should not change as long as the clock is at 0. When the clock reaches 1, the control unit sends a 1 to the IR, so that its value changes to the value propagated during the clock's 0 phase. The control unit should make sure the same data from memory continues to be sent into the IR throughout both phases; otherwise, the new information going into the IR could outrun the signal to the IR to change, and the IR would change to the wrong value.

Within your control unit, you will want a D flip-flop to track whether the CPU is in the fetch cycle or the execute cycle. Note that, because a cycle begins with a clock's 0 phase, this D flip-flop should change when the clock changes to 0. Thus, the clock value sent into the flip-flop should be the NOT of the overall circuit's clock.

Note: When the computer reaches a HALT instruction, the computer should execute no further instructions. One good way to do this is to "freeze" the computer by refusing to change the PC during the execute cycle of a HALT instruction. If you do this, then the computer will fetch the same HALT instruction in the next fetch cycle, and thus it will be in a tight loop.

The control unit's interface

The control unit takes six bits as input.

Clock gives the clock's current phase.

Flags give the zero and positive flags computed based on the AC value.

Instruction op code includes the top three bits of the instruction contained in the IR. Note that $instr_7$ represents the topmost bit in the instruction (and the op code), while $instr_5$ would be the lowest bit in the op code. Thus, for an LOAD instruction (op code 100), $instr_7$ would be 1, and $instr_6$ and $instr_5$ would be 0.

The circuit should produce six bits as output.

PC select selects whether the data sent into the PC should be the lower five bits of the instruction in IR (if it is 0) or the value PC + 1 (if it is 1).

PC write tells the PC register to change its value.

Memory address selector selects whether the address sent to memory should the lower five bits of the instruction in IR (if it is 0) or the PC's value (if it is 1).

Memory write tells memory to change the value at the currently selected address.

IR write tells the IR register to change its value.

AC write tells the AC register to change its value.

Creating the circuit

Before continuing with this assignment, you should design the circuit on paper. Each output for the control unit will be some Boolean function of seven variables, representing the six inputs and the D flip-flop's value.

When you are ready to try your circuit, you can create it within Logisim.

1. The "`getcs 210 10`" command will fetch the Logisim data file `cpu.cir`. This file contains all the pieces of the CPU, except that the control unit is empty.
2. After starting Logisim and opening the circuit file, select Advanced Tools from the Options menu. Also, select Small Gates.
3. Open the control unit subcircuit by going to the Project menu and selecting "control unit." Enter your complete circuit for the control unit in the space given.

   Do not move the switches and output LEDs! Logisim uses their relative ordering to determine the ordering of the pins in the main circuit. Thus, if you were to switch the order in which the switches and LEDs appear, you would also need to rewire the main circuit. This would cause a lot of unnecessary confusion.

Testing the circuit

1. Return to the main circuit by going to the Project menu and selecting "main."
2. Enter a program into memory by selecting the poke tool (represented by the down and up arrows in the toolbar), clicking on the memory address you want to change in memory, and typing the hexadecimal number on the keyboard. The displayed memory contents should change as you type.
3. To start the clock toggling, right-click on the poke tool in the toolbar and select Resume Clocks. This will begin the clock automatically toggling once each second. When you want the clock to stop, select Pause Clocks from the poke tool's menu.

   If something goes wrong (and it probably will), you may find it convenient to replace the clock with an input switch, repeatedly clicking the switch manually to toggle the value, so that you can verify the control unit's actions with each clock phase change, at your own pace of clicking.

What to hand in

Run "`handincs 210 10`" to submit your completed `cpu.cir` file for testing during the grading process.

Your lab report should incorporate an introduction, a conclusion, and a printout of your control unit circuit. If there were any bugs you were unable to fix, describe these in your report too, as thoroughly as you can, so that I can award partial credit for your understanding of the bugs.