SDC Simulator, Part 2

CS 350: Computer Organization & Assembler Language Programming
Lab 6 due Tue Oct 25 [2 week lab]

Note:

- Lab 6 extends Lab 5 and the Final Project is similar to Lab 6, so complete Lab 5 before working on this lab, and get this lab to work even if you can't hand it in on time.

A. Why?

- Implementing the von Neumann architecture helps you understand how it works.

B. Outcomes

After this lab, you should be able to

- Run a simulator for a simple von Neumann computer.

C. Programming Problems

- This lab builds upon the previous one to produce a simple line-oriented simulator in C for the Simple Decimal Computer (SDC). In Lab 5, you built the initial part, in which you initialize memory by reading its values from a text file.
- For this lab, you are to add the simulator commands, which let the user execute SDC instructions and inspect the registers and memory.
- There's a sample executable solution on fourier as `~sasaki/sdc`.. For input, the sample *`.sdc` files from Lab 5 can be used, but you should also create and use your own input files for testing.

D. Lab 6 Programming Assignment [50 points]

For this lab, you have to add a command-processing loop: You read a line containing a simulator command and execute it, read another line and execute it, and so on until you are given the quit command. There are six commands (`q` for quit, `d` for dump control unit and memory, `h` and `?` for help, an integer (to execute that many instruction cycles), or an empty line (to execute one instruction cycle).

1. To start the command loop, prompt for and read a command line. (You'll want to use the `fgets` / `sscanf` technique from the previous lab.) See if the line is empty or has a command. If you hit end-of-file on standard input, exit the program.
    2. For command `q`, note that you've seen a quit command and exit the program. (Don't dump things back out, just quit.)
    3. For command `d`, dump out the control unit and the memory values.
    4. For `h` or `?`, print out a help message.
    5. For an integer (let's call it *N*), execute the instruction cycles that many times but make sure *N* is reasonable first.
        5a. If $N < 1$, complain to the user and go on to the next command.
        5b. If *N* is unreasonably large, warn the user and change *N* to a sane limit.
        5c. Now run the instruction cycle *N* times. Check after each cycle to make sure the running flag is still true; if it becomes false, skip the rest of *N*.
    6. If the command was empty (a newline), run the instruction cycle once (if the running flag is true; if it's false, tell the user that the CPU has halted).
7. Continue the loop (go to step 1). Note you do this if even if CPU execution has halted; that way the user has the option of entering a `d` command before quitting.

E. Programming Notes

- `Lab6_skel.c` includes the framework for this part of the simulator. For brevity, I've omitted the CPU and memory initialization code — you'll need to copy in your Lab 5 solution code for those parts. Don't work on this lab until you finish Lab 5.

F. Grading Guide [50 points total]

- **Setup [2 points total]**
    - [2 pts] Include your name and section in the program and in your output.
    - Use your Lab 5 code to read an SDC input file into memory.

- **The Command Loop [12 points total]**
  - [2 points] Read the command line from standard input
  - [1 point] Command loop repeats until user enters `q` or end-of-file on input
  - [2 points] Handle the `d`, `h`, and `?` commands.
  - If the command is a integer *N*,
    - [3 points] Handle $N \leq 0$ or N insanely large
    - [2 points] Do *N* instruction cycles (stopping early on `HALT`)
  - [2 points] If the command line is empty, do one instruction cycle.
- **Execute an Instruction Cycle [5 points total]**
  - [2 points] Verify that the CPU is running. Also, exit the simulation if the PC is illegal.
  - [3 points] Fetch / decode instruction, call function to handle SDC instruction.
- **Execute an SDC Instruction [25 points total]**
  - [6 = 3 * 2 points] `LD`, `ST`, `ADD` work
  - [4 points] Load and Add immediate work (note positive / negative cases)
  - [2 points] `BR` works
  - [4 points] Branch conditional works (note positive / negative cases)
  - [6 = 3 * 2 points] `GETC`, `OUT`, `PUTS` work
  - [3 = 3 * 1 point] `DMP`, `MEM` work; unused I/O commands skipped
- **Commenting and Style [6 points total]**
  - [4 points] Functions and variables are well-named and commented, code is well-formatted and concise.
  - [2 points] Line or section comments are included when doing something tricky.

Note: If compiling your program on `fourier` (with our usual `gcc`) doesn't produce an executable, your program gets zero points.

