SDC Simulator, Part 2

CS 350: Computer Organization & Assembler Language Programming
Lab 7 due Wed Mar 9

Note: The Final Project will extend your solution to this lab,
so complete it even if you can't hand it in on time

Links: `Lab7_skel.c`

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 6, 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 `alpha` as `~sasaki/Lab7_soln`.. For input, the sample sdc file from Lab 6 can be used, but you should also create and use your own input files for testing.

D. Lab 7 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 CPU and memory, `h` and `?` for help, an integer (to execute that many instruction cycles), or the 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 Lab 6.) 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 CPU (program counter, instruction register, and data registers) 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, 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

- `Lab7_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 6 solution code for those parts.
- Remember, `sscanf` returns the number of items it was able to read. E.g., `x = sscanf(s, "%d", &y);` tries to read an integer from string `s` into variable `y`. If it succeeds, `x` is set to `1`; if not, `x` is set to `0`.

F. Grading Guide [50 points total]

- **Setup:**
    - [2 pts] Include your name and section in the program and in your output.
    - Use your Lab 6 code to read an SDC input file into memory.
- **The Command Loop**:
    - [5 pts] (Steps 1 – 4 above) Read the command line from standard input and handle the `q`, `d`, `h`, and `?` commands. (Exit the simulator on quit or end-of-file.)
    - (Step 5 above) If the command is a number *N*.
        - [3 pts] Handle $N \leq 0$ or N insanely large
        - [2 pts] Do *N* instruction cycles (stopping early on HALT)
    - [2 pts] (Step 6 above) If the command line is empty, do one instruction cycle (see step 6 above).
    - Repeat command loop.
- **Executing an instruction cycle**:
    - [2 pts] Check that the CPU is running, exit the simulation if the PC is illegal.
    - [3 pts] Fetch and decode instruction, call function to handle SDC instruction
- **Executing an SDC instruction**:
    - [4 pts] `LD`, `ST`, `ADD` work;
    - [5 pts] `LDM`, `ADDM` work
    - [2 pts] `BR` works; [4 pts] `BRC` works
    - [2 pts] `GETC` works; [2 pts] `OUT` works; [3 pts] `PUTS` works
    - [2 pts] `DMP`, `MEM` (dump CPU, memory) work
    - [1 pts] Skip unused I/O command
- **Commenting and Style**:
    - [4 pts] Functions and variables are well-named and commented, code is well-formatted and concise.
    - [2 pts] Line or section comments are included when doing something tricky.