SDC Simulator

CS 350: Computer Organization & Assembler Language Programming
Lab 9, due Fri Nov 1

A. Why?

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

B. Outcomes

After this lab, you should

- Have a simulator for a simple von Neumann computer, in C.

C. Programming Problem 1: The SDC Simulator [75 points total]

- Lab 9 adds instruction execution to the the Lab 8 SDC framework, resulting in a full simulator. If you didn't complete Lab 8, then complete it as part of this lab.

Changes to the Lab 8 SDC Framework

- Add `pc` and `ir` declarations to the CPU declarations; initialize them in `initCPU`.
- Fill in the help message function.
- (The main change:) Make the `instruction_cycle` function actually decode and execute the current instruction
  - Set `ir` to `mem[pc]`
  - Increment `pc`
  - Isolate the opcode, register and memory address fields of the `ir`
  - Execute the instruction. (You can drop the `call_nbr` and `min` definitions above `instruction_cycle`.) You'll need 10 cases (one for each opcode); plus, opcode 9 (I/O) has subcases you need to look at.
- Instead of setting `running` to false the tenth time `instruction_cycle` is called, set it to false when you encounter a `HALT` instruction.

Sample Solution; Sample Program and Output

- You an run a sample solution by executing `~sasaki/Lab09_soln` on `alpha`. In `Lab09_sample_pgm.txt`. and `Lab09_sample_out.txt`, you'll find a large sample program and the output it produces.

Program Output

- For each instruction, print output that fully describes the actions and results of that instruction.
- You don't have to have exactly the same format of output as the sample solution, but you should print out all the information it prints out:
  - The operation, the register and memory location, and the old and values of the register or memory location.
    - E.g., for `ADDM` (Add Immediate), say which register you modified, give its old value, the increment, and the new value of the register.
  - You can omit items that aren't relevant for a given instruction. E.g., `HALT` ignores the `R` and `MM` fields of the instruction; `LDM` uses `MM` as an immediate value, so the value at location `MM` isn't relevant.

Point Breakdown [75 points]

- [5 pts] General program structure — code well-organized [avoid too many levels of nested statements, functions that are too long
- [60 = 12 * 5 points each] Handle the `LD`, `LDM`, `ST`, `ADD`, `ADDM`, `NEG`, `BR`, `BRP`, `IO 1`, `IO 2`, `IO 3`, and `IO 4` instructions. (Of the 11 points, 6 are for correctness of the operation and 5 are for the output.)
- [10 pts] Formatting and comments: Indent your code correctly; include your name and section number; include enough comments so that the TA can understand what you're doing but not so many that the TA loses track of what you're doing.

D. Programming Problem 2: An SDC Program [25 points]

- Write a program in SDC machine language that divides the number `X` in location `20` by `10`, leaving the quotient in `R1` and the remainder in `R0`. (Assume `X` ≥ `0`.)

- Remember, an SDC program is a sequence of 4-digit numbers. To test your SDC program, run your SDC simulator and enter your SDC program as the input to the simulator
- Here's some pseudocode for the program. The key property is the relationship between `X`, `R0`, and `R1`: Every time we hit the while loop test, `X = 10*R1 + R0` with `R1 ≥ 0` and `R0 ≥ 0` (this is known as a loop invariant). we stop when `R0` < `10`:

```
R0 = X
R1 = 0
while R0 >= 10
   R0 = R0 - 10
   R1 = R1 + 1
HALT
```

- Since we can only test a register for > 0, we need `while R0 - 9 ≥ 1`. Rather than constantly subtracting and adding `9`, let's just subtract `9` once before the loop begins and add it back after the loop ends:

```
R0 = X
R0 = R0 - 9
R1 = 0
while R0 >= 1
   R0 = R0 - 10
   R1 = R1 + 1

       R0 = R0 + 9
       HALT
```

- Remember to write your code so that `X` is stored at location `20`; the TAs will rely on this for grading purposes.
- Feel free to store `-9` and `-10` somewhere so that you can use them in your program.
- ***Point Breakdown [25 points]***
    - Roughly 2 points for correctness of each instruction and constant.

