COM S 227
Spring 2019
Miniassignment 2 A simple computer
40 points

Due Date: Friday, March 15, 11:59 pm (midnight)
5% bonus for submitting 1 day early (by 11:59 pm March 14)
10% penalty for submitting 1 day late (by 11:59 pm March 16)
No submissions accepted after March 16, 11:59 pm

General Information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus, http://www.cs.iastate.edu/ cs227/syllabus.html#ad , for details.
You will not be able to submit your work unless you have completed the *Academic Dishonesty policy acknowledgement* on the Homework page on Canvas. Please do this right away.
If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Note: This is a miniassignment and the grading is automated. If you do not submit it correctly, you will receive at most half credit.

Overview

In this assignment you will implement a simple instruction set for a toy accumulator architecture. An *accumulator architecture* is a class of computer architecture in which most or all instructions work on an accumulator register. This is in contrast with a *register architecture*, which has many registers that may be selected and used for computation and control.

Accumulator architectures were very common in the early computers from the 1940s through the 1960s, but as the science of computing matured, we came to understand that register architectures were superior in many ways, so accumulator architectures disappeared from the computing landscape.

But that accumulator architectures no longer exist in practice does not imply that they are not interesting machines! There is much to be learned by both neophyte and experienced computer scientists about the practice of computing by looking at the design of an accumulator architecture, and by programming one.

Our architecture, called CS227COMP, is digital (base 10, as opposed to binary, base 2) and has only 13 instructions, and yet that is sufficient to solve non-trivial problems; in fact, it's theoretically as "powerful" as any modern computer!

Implementing this processor architecture will give you practice working with arrays and loops. The javadoc provides details and examples of the implementation.

Advice

Before you write any code for a method, **work through the problem with a pencil and paper on a few concrete examples**. Make yourself write everything down; in particular, write down things that you need to remember from one step to the next (such as indices, or values from a previous step). Try to explain what you are doing in words. Write your algorithm in pseudocode.

Another key problem-solving strategy is to try solving part of the problem, or solving a related, simpler problem. Often it's useful to break a problem into one or more sub-problems that can be solved in methods

of their own. "Helper" methods, like these, are for internal (inside a class) use only and should have *private* access; you, the author of the class call them to help you implement the functionality of the class, but allowing the user of the class to call them could potentially corrupt class state. Our CS227COMP implementation has 17 private methods to handle things like executing each of the instructions, updating the instruction counter, and dealing with integer overflow. This is not to say that you should have helper methods! Only to get you to think about how a good design may look.

Here are some ideas for getting started:

- Among other initialization, your constructors should allocate your memory array.
- Remember that accessors ("getters") should get and mutators ("setters") should set. Don't *combine* the functionality!
- `runProgram()` simply runs the next instruction in a loop!
- To get started on `loadMemoryImage()`, can you
  1. determine the size of the smaller of two arrays?
  2. iterate over an array, checking values and copying them?
- To get started on loadProgramFromConsole(), can you
  1. prompt for input from a user?
  2. read an unknown number of integers from a scanner until you see a sentinel?
  3. check that the integers are in the representable range?
- To get started on loadProgramFromFile(), see `loadProgramFromConsole()`. This is essentially a simpler version without prompts.
- `nextInstruction()` is the meat of this program. It may seem intimidating when you consider that you have to correctly execute any of the 13 instructions, but, done well, this method should be simpler (longer, true, but simpler) than either of the two load methods. For instance, as mentioned above, the instructions themselves are most cleanly implemented in private helper functions. Given that, this method is a matter of figuring out which instruction is being executed and calling the appropriate helper.

  Ideally, instruction counter update would occur at exactly one place directly within `nextInstruction()`; however, the jump instructions throw a wrench in the works. It's not impossible to special-case the jumps and do your instruction counter update here, but its probably cleaner to do it in each of your instruction helpers instead.

  Helper methods to handle instruction overflow and arithmetic overflow can be called here or in your instruction helpers.

  Crashes can occur here *and* in your load methods. It might be useful to have a helper that handles crashes for you that you can simply call when you need it.

My code's not working!!

Developing loops can be hard. If you are getting errors, a good idea is to take a simple concrete example, and trace execution of your code by hand (as illustrated in section 6.2 of the text) to see if the code is doing what you want it to do. You can also trace what's happening in the code by temporarily inserting `println` statements to check whether variables are getting updated in the way you expect. (Remember to remove the extra `println`'s when you're done!)

Overall, the best way to trace through code with the debugger, as we are practicing in Lab 6. Learn to use the debugger effectively, and it will be a lifelong friend.

Always remember: one of the wonderful things about programming is that within the world of your own code, you have absolute, godlike power. If the code isn't doing what you want it to do, you can decide what you really want, and make it so. **You are in complete control!**

(If you are not sure what you want the code to do, well, that's a different problem. Go back to the "Advice" section.)

Testing and the SpecChecker

A SpecChecker will posted shortly that will perform an assortment of functional tests. As long as you submit the assignment correctly, your score will be exactly the score reported by the specchecker.

However, when you are debugging, it is much more helpful if you have a simpler test case of your own that reproduces the error you are seeing. For example:

```
import mini2.CS227Comp;

public class SimpleTest
{
 public static void main(String[] args)
 {
   String msg = "(LOAD␣4)␣For␣machine␣with␣IC=0,␣"          +
                "AC=0,␣memory=[3004,␣0,␣0,␣0,␣42,␣5],␣"     +
                "after␣nextInstruction(),␣AC␣should␣be␣42.";
   int[] arr = {3004, 0, 0, 0, 42};
   comp = new CS227Comp(0, 0, arr);
   comp.nextInstruction();
   System.out.println(msg + (42 == comp.getAC()));
 }
}
```

Since no test code is being turned in, you are welcome to post your tests on Piazza for others to use and comment on.

Documentation and style

Since this is a miniassignment, the grading is automated and in most cases we will not be reading your code. Therefore, there are no specific documentation and style requirements. However, writing a brief descriptive comment for each method will help you clarify what it is you are trying to do.

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder `miniassignment2`. If you dont find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag `miniassignment2`. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post source code for general Java examples that are not being turned in. (In the Piazza editor, use the button labeled pre to have Java code formatted the way you typed it.)

If you have a question that absolutely cannot be asked without showing part of your source code, make the post "private" so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form read all my code and tell me whats wrong with it will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any posts from the instructors on Piazza that are labeled Official Clarification are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q&A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Homework page on Blackboard, before the submission link will be visible to you.

Please submit, on Canvas, the zip file that is created by the SpecChecker. The file will be named `SUBMIT_THIS_mini2.zip`. and it will be located in the directory you selected when you ran the SpecChecker. It should contain one directory, `mini2`, which in turn contains one file, `CS227Comp.java`.

Always LOOK in the zip file the file to check what you have submitted!

Submit the zip file to Canvas using the Miniassignment2 submission link and verify that your submission was successful. If you are not sure how to do this, see the document "Assignment Submission HOWTO" which can be found in the Piazza pinned messages under Syllabus, office hours, useful links.

We strongly recommend that you just submit the zip file created by the specchecker. If you mess something up and we have to run your code manually, you will receive at most half the points.

> We strongly recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory `mini2`, which in turn should contain the file `CS227Comp.java`. You can accomplish this by zipping up the `src` directory of your project. The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and not a third-party installation of WinRAR, 7-zip, or Winzip