Where do I start and what all do I turn to for making progress

Suggest Resource list is as follows:

Conceptual understanding of writing assembly programs in HACK (lecture format)

- **Intro Video: https://www.youtube.com/watch?v=KvVc5RdPPh0 (minute marker 20:00 and onwards)**
- **Concept Refresh (quick bites):**
  - **HACK Assembly Code easier than machine code: https://youtu.be/4M2UZNhoc08**
  - **HACK Assembly Indirect Addressing: https://www.youtube.com/watch?v=bgBHaa2aT64**
  - **HACK Assembly Symbols: https://www.youtube.com/watch?v=dt98g2piUBI**
- **HACK Programming Reference Sheet: https://canvas.tamu.edu/courses/207384/modules/items/7069159**
- **HACK Programming Cheat Sheet: https://canvas.tamu.edu/files/50209381/**

Basic skill development in writing assembly programs in HACK (lab format)

- **Introductory Play Exercise (Lab 11) https://youtu.be/jjzsayw-UWs**
- **Medium Level Exercise (Lab 12) https://youtu.be/AGsCudPN5x0**

Advanced skill development in writing assembly programs in HACK (lab format)

- **Advanced Program Control Flow for Loops and Branches (Lab 13) https://youtu.be/GGI-aHjPbSg**

CPU Emulator

- **Short Video summarizing key features and productivity tips https://www.youtube.com/watch?v=XOcsLxRUglo**
- **Presentation containing details of CPU Emulator Features https://docs.wixstatic.com/ugd/44046b_24b3a15aa628404fbf6dacd86d7da3af.pdf**

Project Assistance

- **Project 4 Guidance: https://canvas.tamu.edu/files/38209886/download?download_frd=1**
- **Extended QnA in Global Community (if you have an esoteric question that's not handled in the FAQ)**
  - **http://nand2tetris-questions-and-answers-forum.52.s1.nabble.com/Project-4-f32605.html**
  - **http://nand2tetris-questions-and-answers-forum.52.s1.nabble.com/Chapter-4-f32603.html**

My program works when I run it manually but it fails in the test file -- what should I do?

Very likely that this is an optimization problem. Our CPU emulator test scripts work by running the emulator for an arbitrary but constant number of cycles and then checking the output. These aren't intended to be very tight bounds but it is possible you'll run into them anyway. You can change the number of loops in the test cases if you need to. You can do this by finding the failing test case and increasing the loop length. For example, an repeat construct of length 50 would look like this:

repeat 50 {

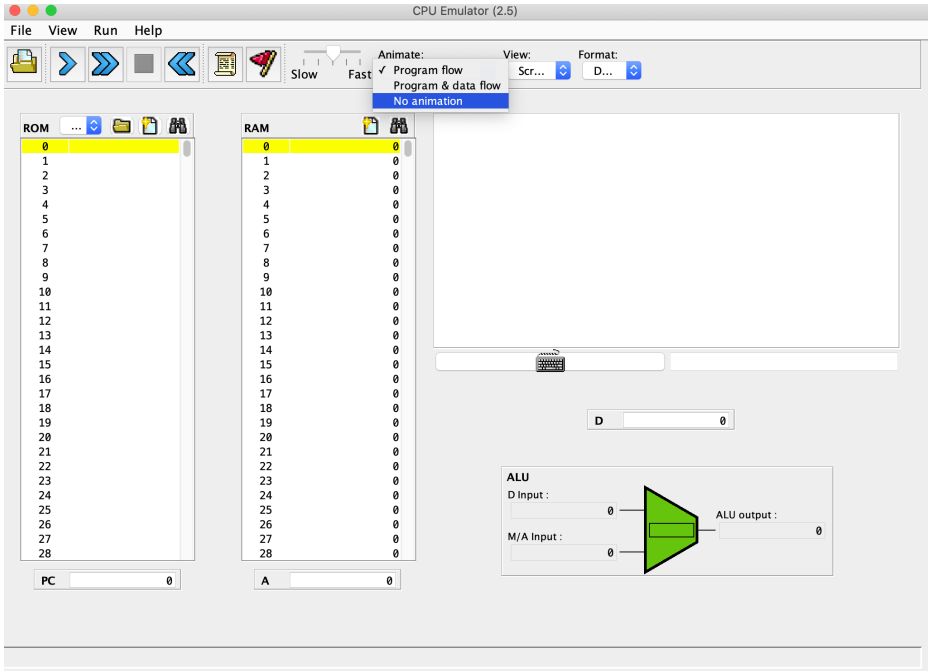
}

How can I make a test case which ignores some outputs when comparing?

Sometimes you will want to ignore the output of some field in a test case -- for example if you are testing the behavior of a script which stores the output in different locations based on the input. You can do this by using the asterisk as a wildcard in your cmp file. It will match one arbitrary character when comparing so you will still need to match the length. For example, look at some entries for RAM[4] in the calc.cmp file provided as part of the Project4 test collateral.

My CPU Emulator is really slow, how can I speed it up?

Turn off animation. It can be nice to look at but it slows the execution down significantly.



Is it possible to specify a constant using binary?

Unfortunately not, the A instruction only supports (non-negative) decimal numbers.

I get an error “At line 32767: Can't continue past last line”

The PC is bounded by the maximum signed 16-bit integer (32767). The CPU emulator will error if it overflows instead of looping around. What this error means is that you have attempted to execute the 32768th line which does not exist because the PC cannot support it. Although this might just be a program length bound, the most likely scenario is that you forgot to add a terminating infinite loop. An empty instruction is considered to be a no-op instruction and so the emulator will just do nothing and move on to the next line until it runs to the end of the ROM. You can fix this by adding an infinite loop to the end of your code or just ignore it, it isn’t problematic if your code works.

The Hack language is case-sensitive

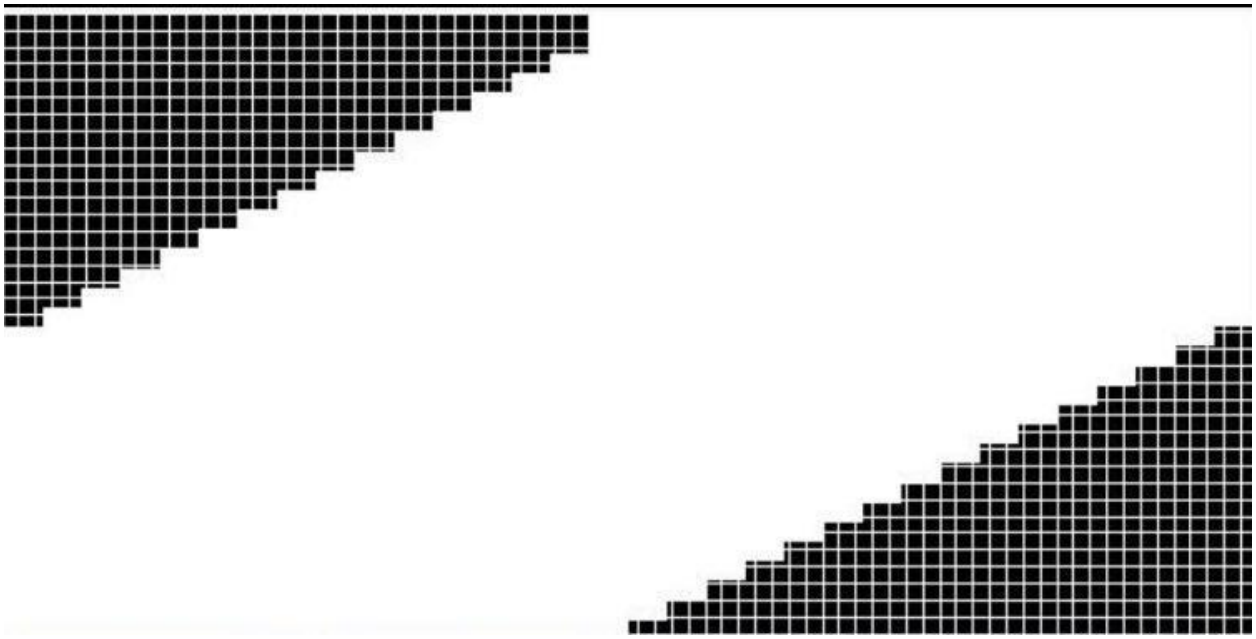
A common error occurs when one writes, say, "@foo" and "@Foo" in different parts of one's program, thinking that both labels are treated as the same symbol. In fact, the assembler treats them as two different symbols. This bug is difficult to detect, so you should be aware of it.

I get an error message when loading my program, what should I do?

The line to look at is generally the line after the one which is mentioned in the error. A couple thing to note, although this isn’t meant to be comprehensive:

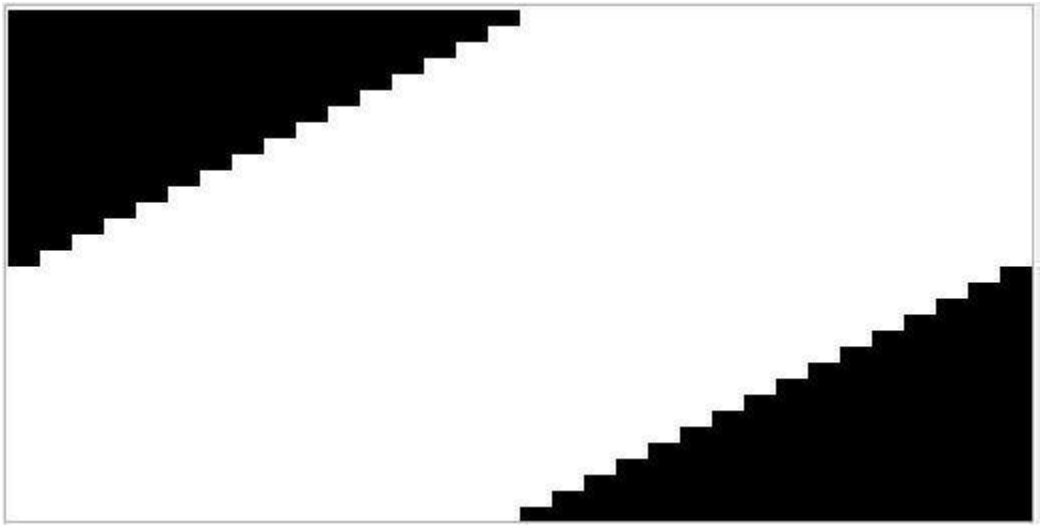
- Registers are case sensitive -- “a” is not a valid register, you should use “A” instead
- You can not load constants into memory by any way other than using an A instruction. “D=100” is not a valid instruction, for example.

My aggie.asm has a bunch of white lines/visual artifacts/etc, what should I do?



This is a visual bug that the CPU emulator has -- it won't affect your grade.

Is it okay if my aggie.asm has a staircase shape?



The lines should be smooth -- we'll be applying a 5 point penalty if they are blocky like the above image.

My algorithm is really inefficient, will that affect grading?

Efficiency is not part of the grade and our test cases will use purposefully high clock cycles to avoid it, but it is not impossible there will be false negatives because of how the test scripts work. If this happens to you, reach out to your TA and have them adjust and rerun the scripts.

Should I reset RAM[4], RAM[1024], any values which might not be updated in the general program flow?

You should ensure that they accurately represent the value and you aren't giving values from previous computations. Resetting them is probably the simplest way to manage that.

Why do we do an infinite loop at the end of the program?

It's essentially a way to freeze program execution when you are done with computation. If you do not do that, your program will continue executing blank lines after your program (which are no-operation instructions) and eventually cause an error from attempting to access a line that does not exist in the ROM address space.