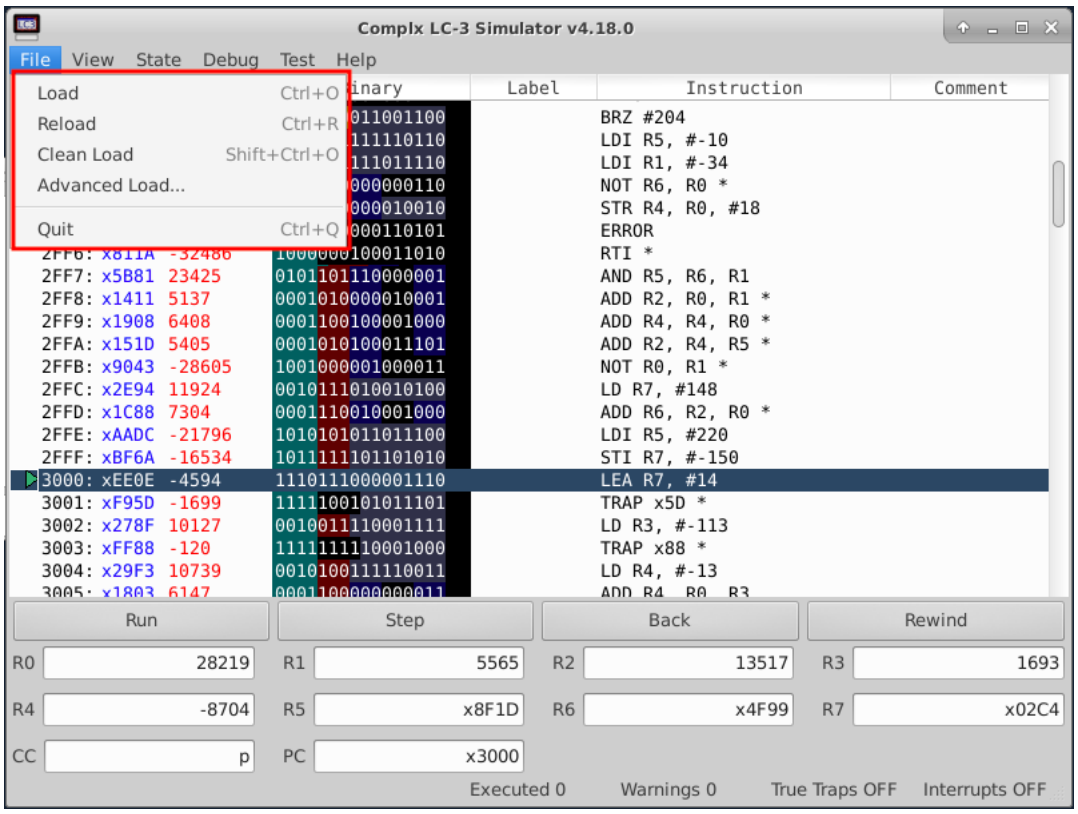
Introduction to Complx

Your friendly neighborhood 2110 TAs

Have Fun Debugging!

1 Components

1.1 Loading Files in Complx



Complx comes with 4 different ways to load in your assembly program:

1. Load
   This loads a selected assembly program into memory, with the memory and registers assigned random values. It is important that your code works with this, since you can never assume any value in memory or a register.

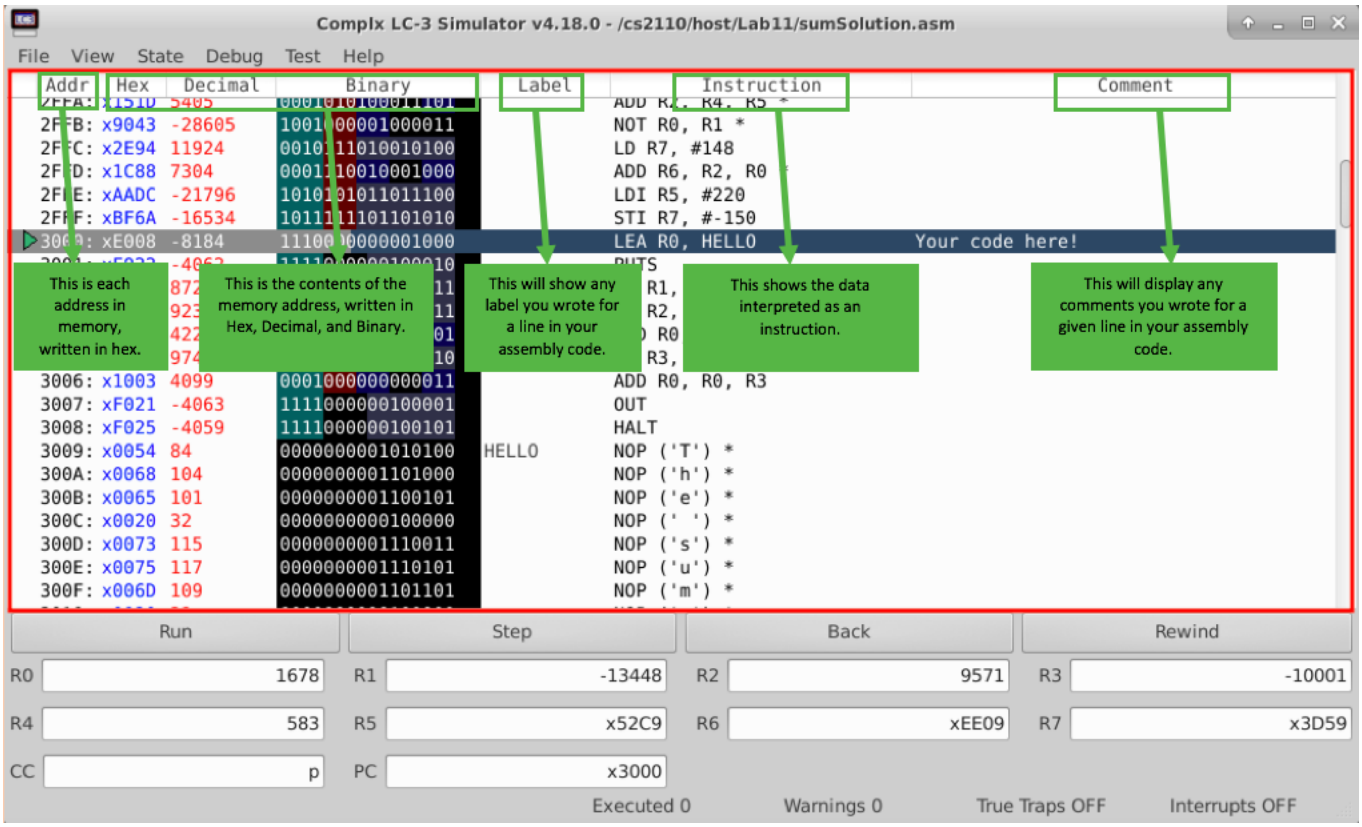
2. Reload
   This reloads the current assembly file loaded into Complx, again with random memory and registers.

3. Clean Load
   This is load but memory and the registers are initialized to 0. This is helpful when debugging to make what your code does more clear.

4. Advanced Load
   This allows you to specify starting values for registers and memory as random, 0, or some specific value, specify initial console input if needed, or change the starting value of the PC.

1.2 Different Components of Complx

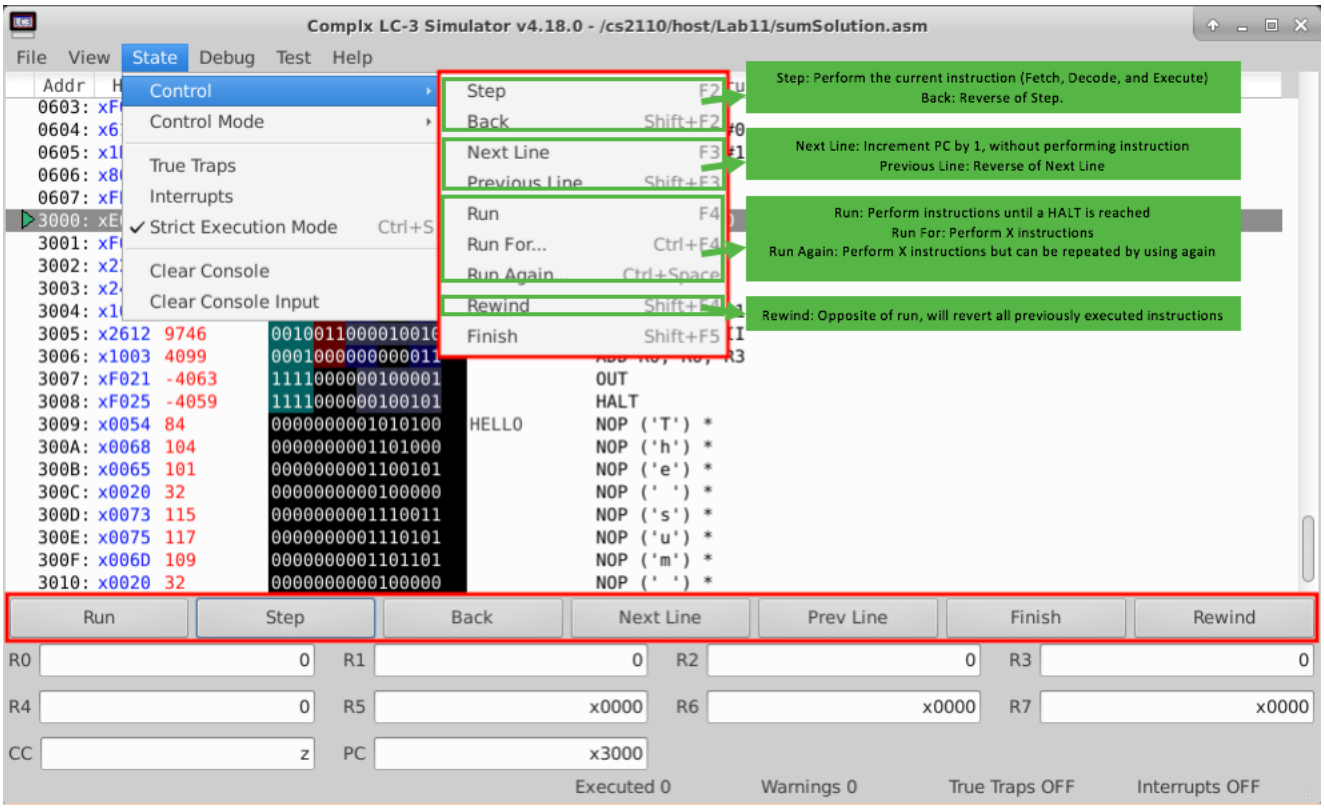
1.2.1 Memory View



Useful tips!

1. You can go to any memory address by pressing ctrl+g and typing the memory address (for example x3000).
2. If you want to view two different areas of memory simultaneously, you can open another memory view window using view → New View, or ctrl+v.
3. You can use view → Hide Addresses to configure which addresses are displayed in the memory view.
4. You can use view → disassemble to change how the instructions are displayed.
   (a) Basic: Labels are replaced with calculated offsets and TRAPS are replaced with TRAP and the corresponding trap vector.
   (b) Normal: Default display of instructions.
   (c) High Level: Instructions are displayed in C-like syntax.

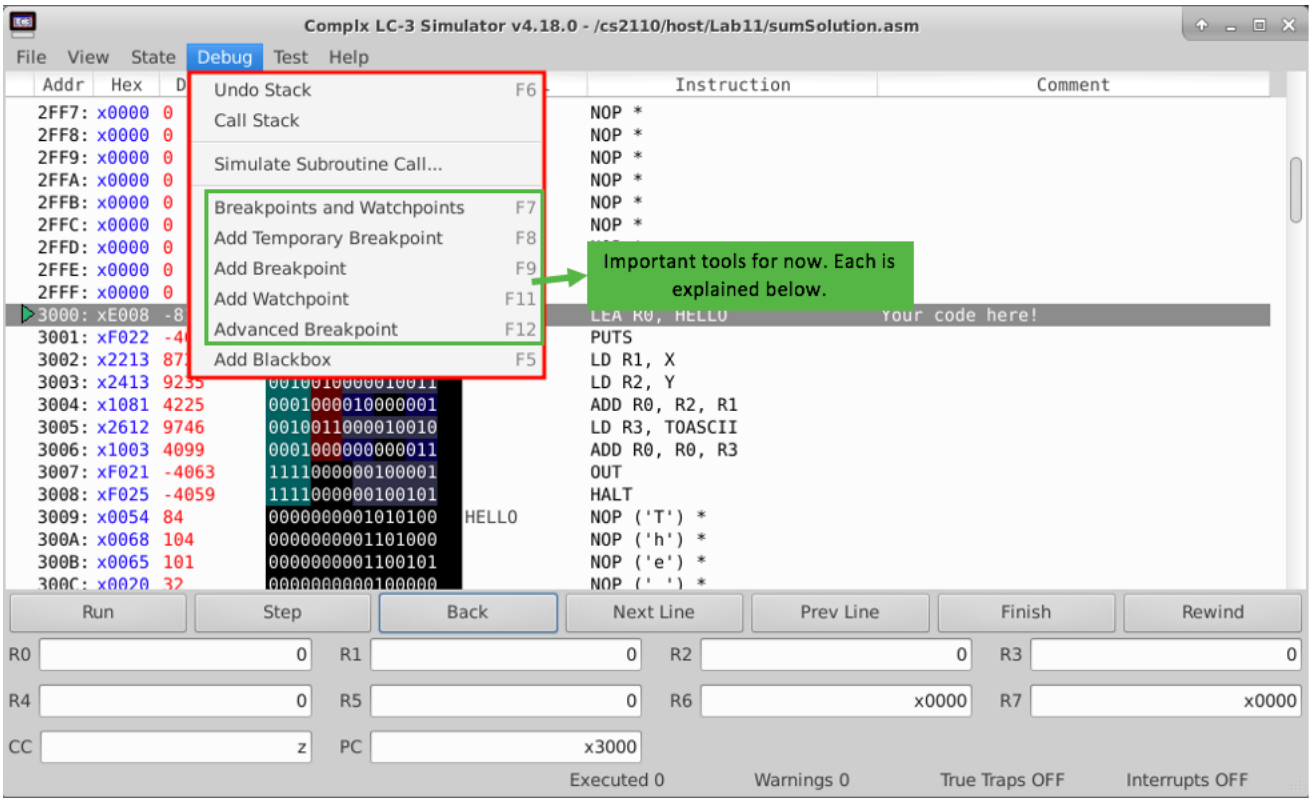
1.2.2 Controls



1.2.3 Registers

The bottom of Complx displays the values in all 8 registers, as well as the PC and CC. You can double-click on a register to cycle through displaying it as decimal, binary, and hexadecimal!

1.3 Breakpoints and Watchpoints



1.3.1 Breakpoints

Breakpoints are lines in our assembly code that we specify Complx to stop at **before** executing it. So if we were to use the Run command, it would stop at a breakpoint instead of a HALT instruction.
This is really useful if we want to figure out the status of our program sometime during its execution! Each breakpoint has the following properties:

1. Name: A name for the breakpoint, which can be anything.
2. Address: The address where the breakpoint is being added. This is the instruction we are breaking at.
3. Condition: A condition for when the breakpoint actually stops execution upon being hit. This can be 1 (true) if you always want it to stop execution, or checking something like: R1==0, for example, or mem[x3000]==5.
4. Times: How many times it should be hit before stopping. Use -1 for an indefinite amount.
5. Hits: Not modifiable, but the number of times the breakpoint has been reached.
6. Enabled: Whether or not the breakpoint will stop execution.

To create a new breakpoint you can select an instruction in Memory View and use any of the following (Note: the selected line is highlighted, the green arrow refers to where the PC is pointing):

1. Add Temporary Breakpoint: This creates a single breakpoint that will be destroyed after being hit once.

2. Add Breakpoint: This creates a single breakpoint with Times = -1.

3. Advanced Breakpoint: This creates a single breakpoint and lets you speciy the different properties.

You can also create a breakpoint directly in your assembly code! You would add one as a comment to a line with the following syntax:

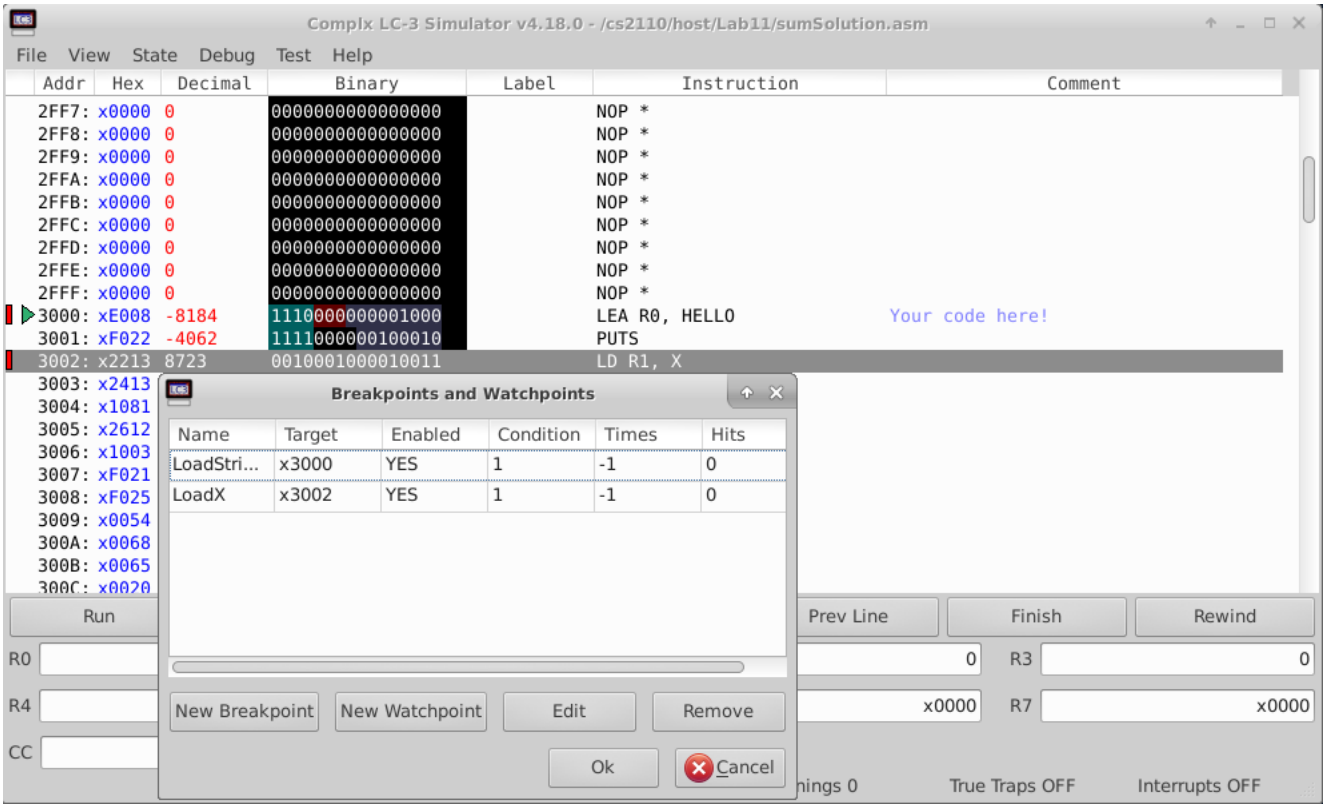
;@break name= condition=

For example:

LEA R0, HELLO ;@break name=LoadStringAddr, condition=1

would create a new breakpoint for the instruction LEA R0, HELLO.

Breakpoints show up in Complx denoted by a red rectangle on the line of the breakpoint, but you can also see them listed in Debug→Breakpoints and Watchpoints.



1.3.2 Watchpoints

Watchpoints are very similar to breakpoints except you specify a memory address or register, and they trigger when the value at the memory address or register changes. Just like breakpoints they also have properties for: name, condition, times, hits, and enabled. You can create these with Debug → New Watchpoint or from within the Breakpoints and Watchpoints table.

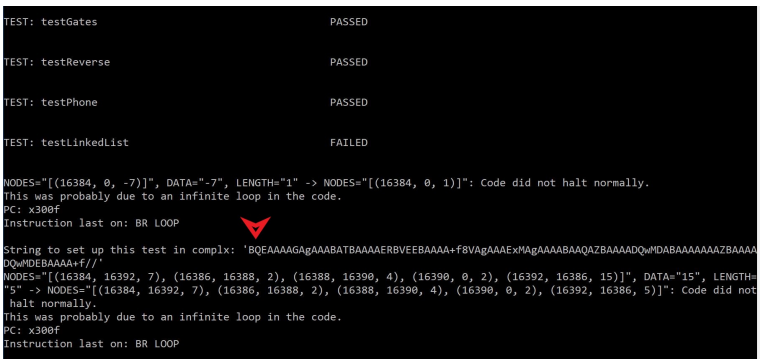
2 Debugging in Complx/Docker

When you turn in your files on gradescope for the first time, you might not receive a perfect score. Does this mean you change one line and spam gradescope until you get a 100? No! You can use a handy tool known as tester strings.

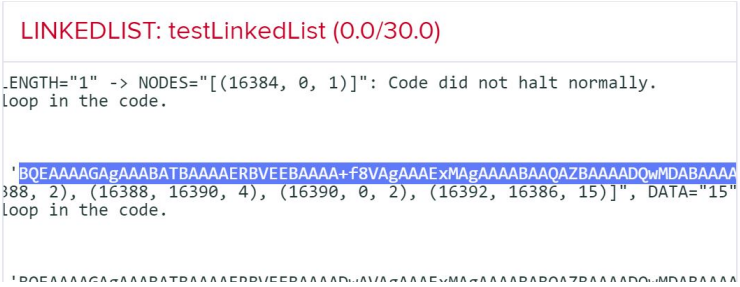
1. First off, we can get these tester strings in two places: the local grader or off of gradescope. To run the local grader:

   - Mac/Linux Users:
     (a) Navigate to the directory your homework is in. **On your computer, not the docker image**
     (b) Run the command `sudo chmod +x grade.sh`
     (c) Now run `./grade.sh`
   - Windows Users:
     (a) On **docker quickstart**, navigate to the directory your homework is in
     (b) Run `./grade.sh`

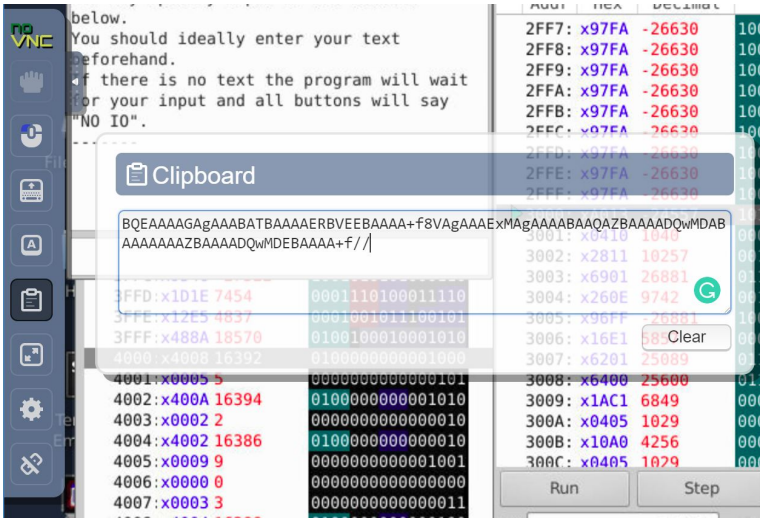
   When you run the script, you should see an output like this:



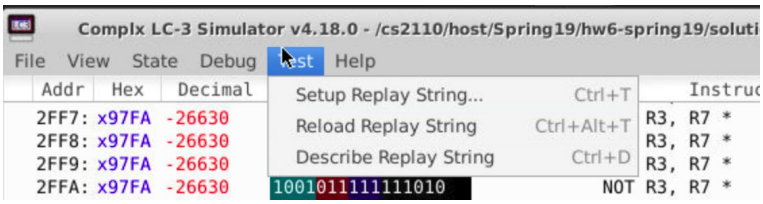
   Copy the string, starting with the leading 'B' and ending with the final backslace. Do not include the quotations.
   Side Note: If you do not have docker installed, you can still use the tester strings to debug your assembly code. In your gradescope error output, you will see a tester string. When copying, make sure you copy from the first letter to the final backslace and again, don't copy the quotations.



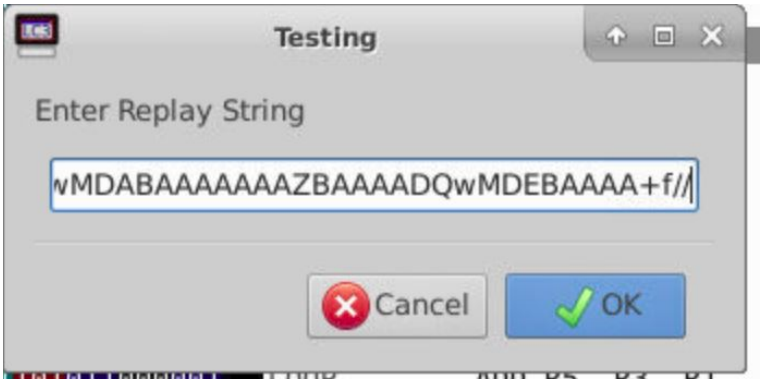
2. Secondly, navigate to the clipboard in your docker image and paste in the string.



3. Next, go to the Test Tab and click Setup Replay String

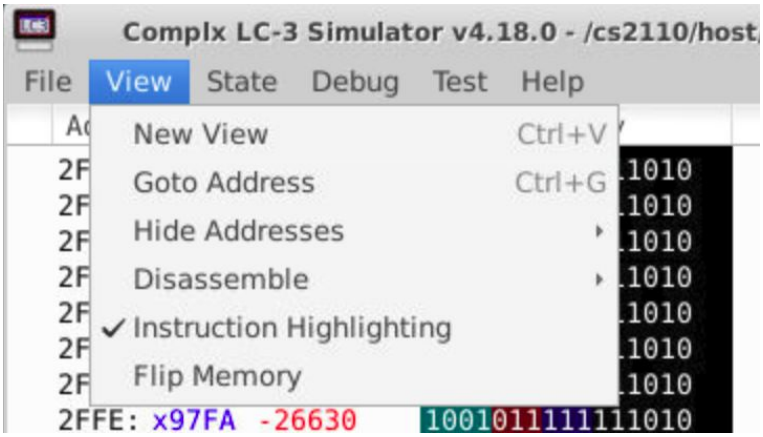


4. Now, paste your tester string in the box!

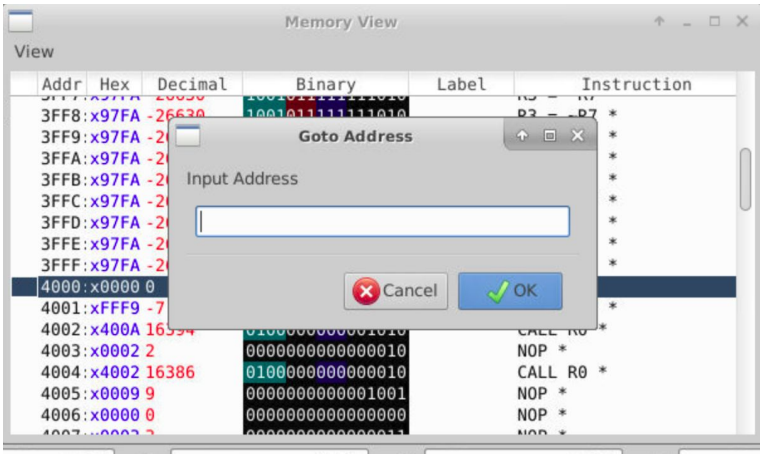


5. Now, complx is set up with the test that you failed! The nicest part of complx is the ability to step through each instruction and see how they change register values. To do so, click the step button. To change the number representation of the registers, double click inside the register box.

6. If you are interested in looking how your code changes different portions of memory, click the view tab and indicate 'New View'



7. Now in your new view, go to the area of memory where your data is stored by CTRL+G and insert the address



Ta-Da! You now have a handy-dandy way to debug your assembly code.
Please Note: If you ask your TAs to debug your assembly code, we will ask to see if you have already stepped through your code using complx. If you have not, we will show you how to debug and let you debug your code yourself.