

CS 2110 Lab 12

Debugging in Complx

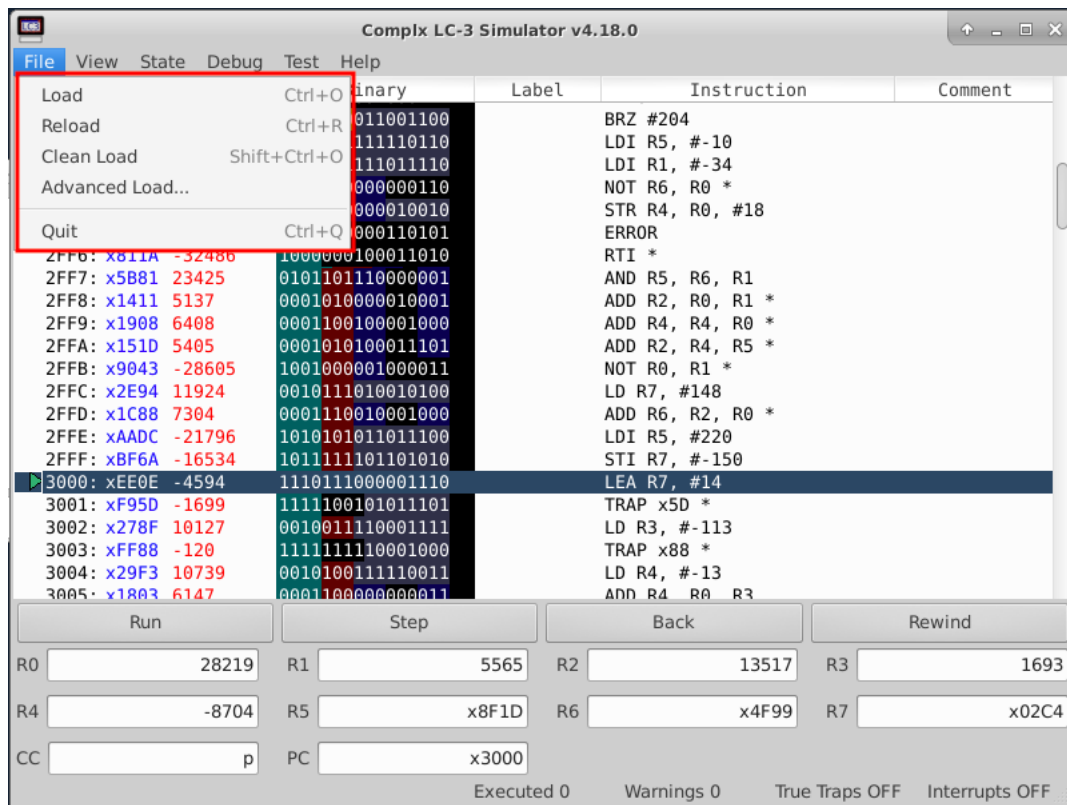
The TAs :) (and the amazing creator of Complx, Brandon)

10/03/2018

1 Recitation Outline

1. Loading Files in Complx
2. Different Components of Complx
3. Breakpoints and Watchpoints
4. Replay Strings

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

Complx LC-3 Simulator v4.18.0 - /cs2110/host/Lab11/sumSolution.asm

Addr	Hex	Decimal	Binary	Label	Instruction	Comment
2FFA	x1510	-5495	000101010001101		ADD R2, R4, R5 *	
2FFB	x9043	-28605	100100001000011		NOT R0, R1 *	
2FFC	x2E94	11924	001011010010100		LD R7, #148	
2FFD	x1C88	7304	0001110010001000		ADD R6, R2, R0	
2FFE	xAADC	-21796	101010101101100		LDI R5, #220	
2FFF	xBF6A	-16534	101111101101010		STI R7, #-150	
3006	x0008	-8184	1110010000001000		LEA R0, HELLO	Your code here!
3007	xF021	-4063	1111000000100010		OUT	
3008	xF025	-4059	1111000000100101		HALT	
3009	x0054	84	0000000001010100	HELLO	NOP ('T') *	
300A	x0068	104	0000000001101000		NOP ('h') *	
300B	x0065	101	0000000001100101		NOP ('e') *	
300C	x0020	32	0000000001000000		NOP (' ') *	
300D	x0073	115	0000000001110011		NOP ('s') *	
300E	x0075	117	0000000001110101		NOP ('u') *	
300F	x006D	109	0000000001101101		NOP ('m') *	

Run Step Back Rewind

R0 1678 R1 -13448 R2 9571 R3 -10001

R4 583 R5 x52C9 R6 xEE09 R7 x3D59

CC p PC x3000

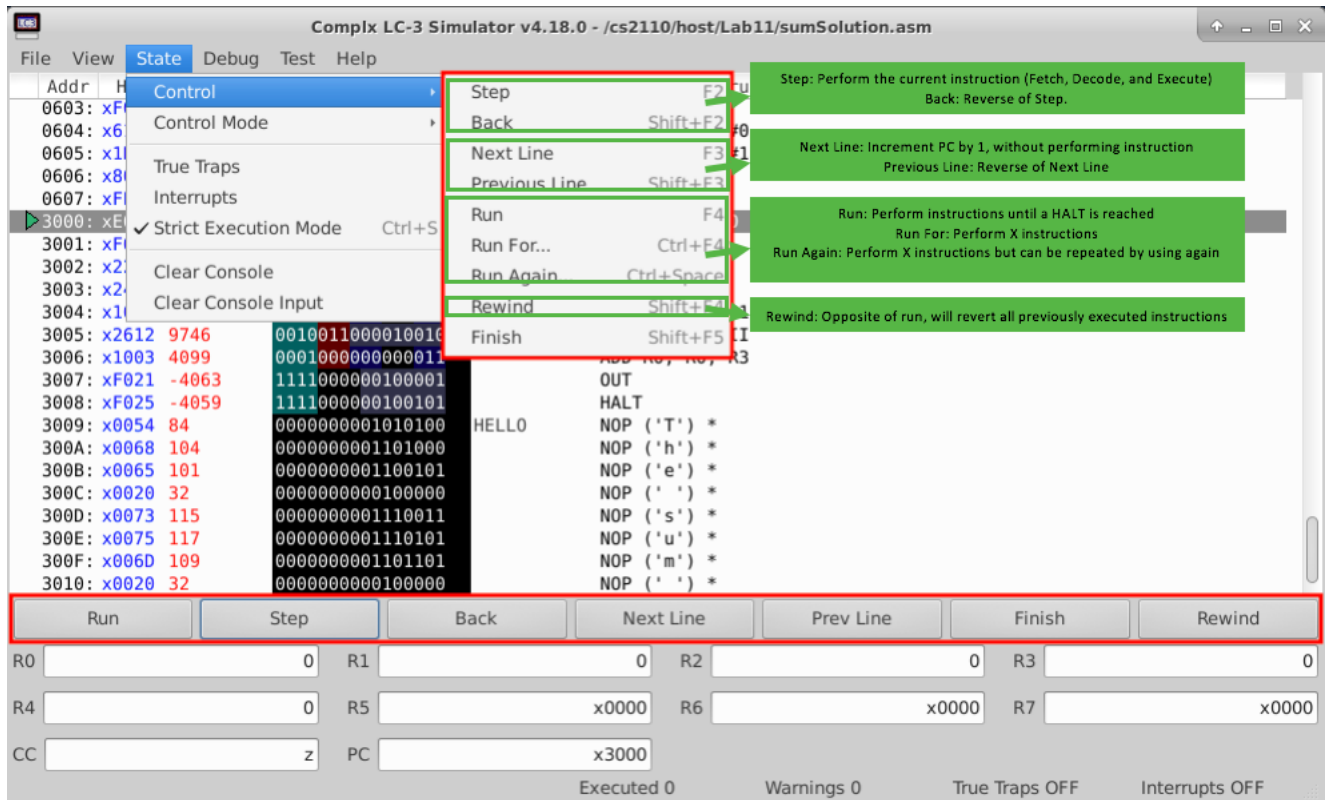
Executed 0 Warnings 0 True Traps OFF Interrupts OFF

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.

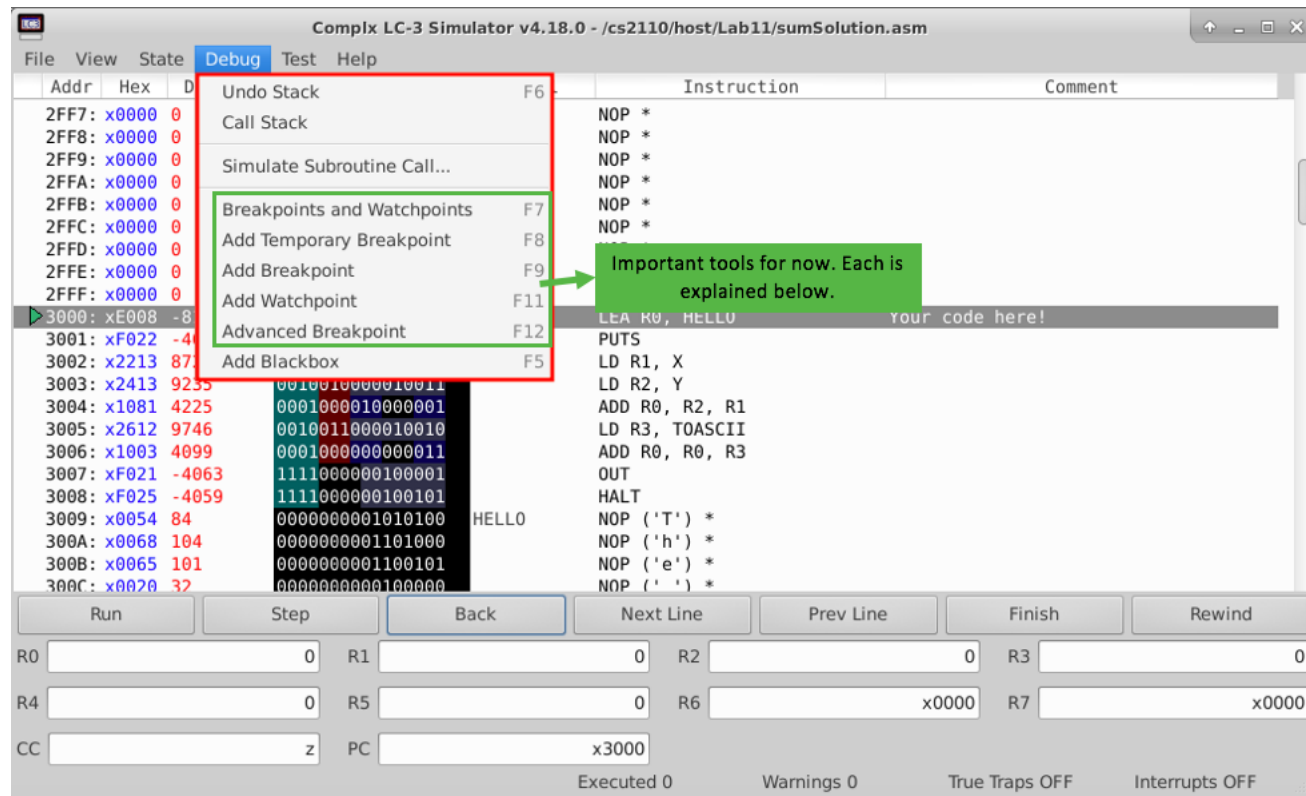
1.2.2 Controls



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 specify 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:

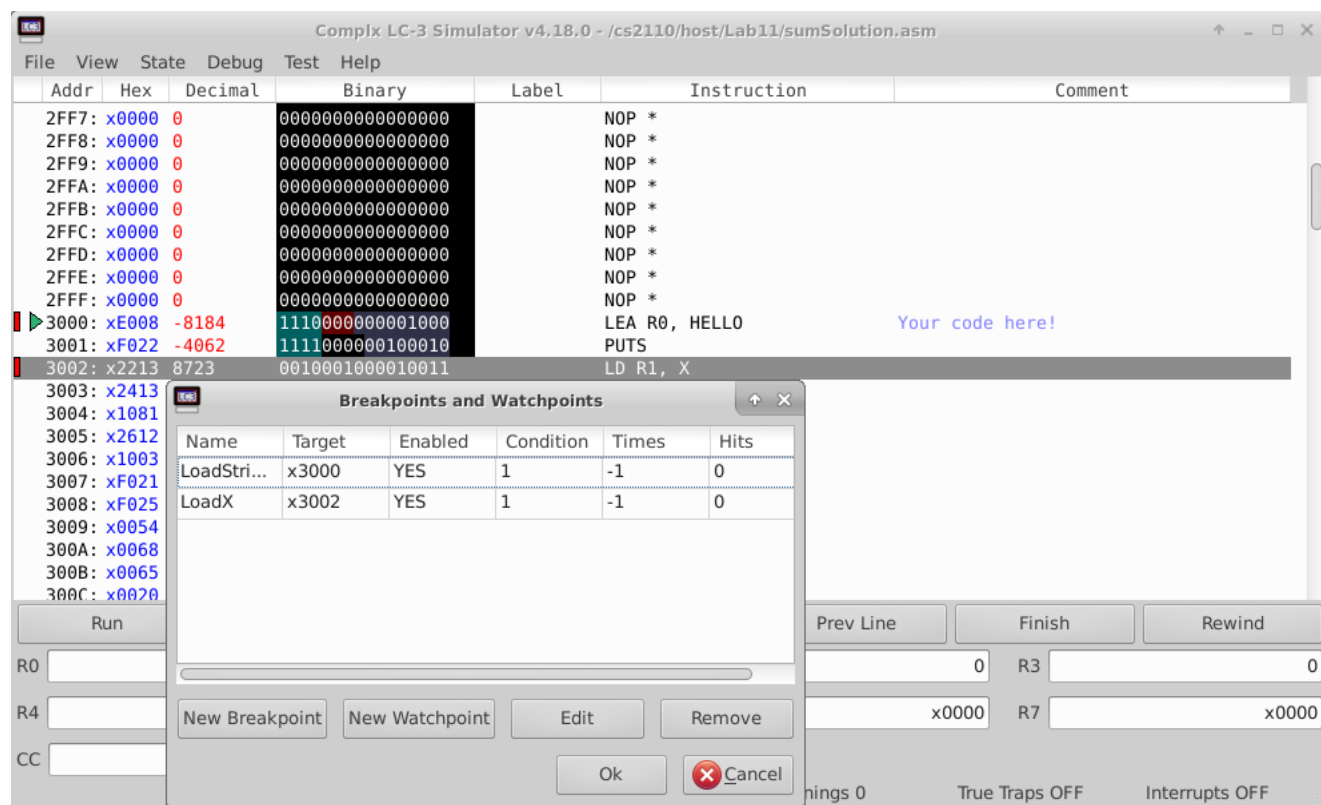
```
;@break name= condition=
```

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.

1.4 Replay Strings

Say you have written all of your code and then upload it to gradescope. Once uploaded, you see that you failed all of the test cases. Oh no! Luckily for you, there are replay strings that can reset up the testing environment locally in complx! Just copy the replay string found at the end of your gradescope message. (Shown below)

Autograder Results

Results

Code

TestAddSubtract: testAddSubtract (0.0/100.0)

MEM[RESULT] was expected to be (-2 xfff9) but code produced (-7 xfff9)

'BQAAAAAGAAAAABAVBQAAAEFSUKFZBAAAAP//AAAAAAEAEwYAAABMRU5HVEgBAAAABAD/'
-> 5: MEM[RESULT] was expected to be (5 x0005) but code produced (17763 x0005)

'BQAAAAAGAAAAABAVBQAAAEFSUKFZBwAAAAMABQABAAQABgAIAAwAEwYAAABMRU5HVEgBAAAABAD'

Inside complx, go to Test -> Setup Replay String. Paste your replay string inside the box, hit ok, and you are good to start locally testing!

