
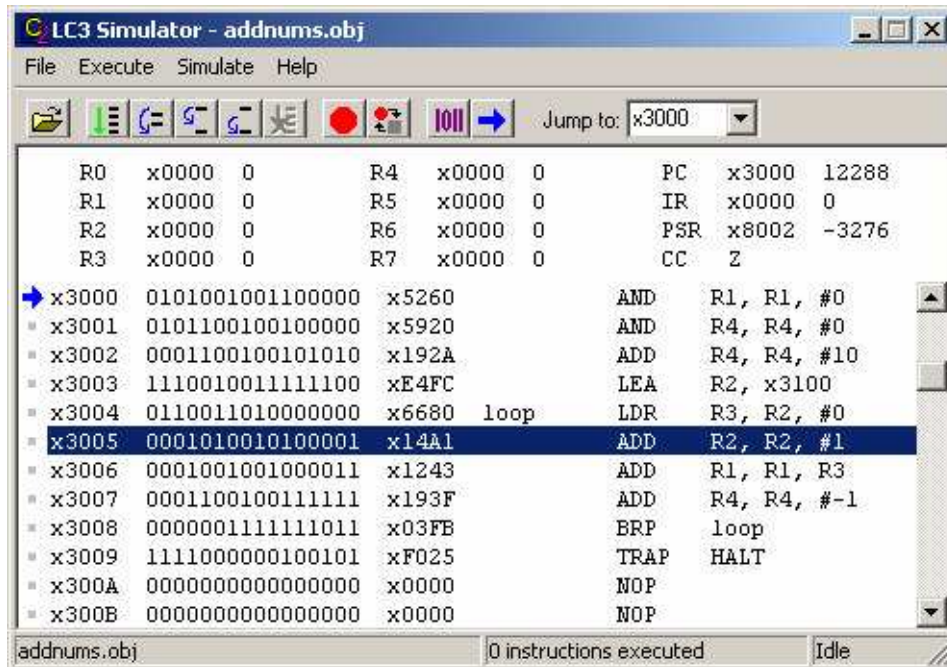


Chapter 3

Running a program in the simulator

Now you're ready to run your program in the simulator. Open the simulator, and then click the Load Program button . Browse and choose *addnums.obj*. Notice that you only get the option to open one type of file in the simulator: the .obj file. This is what you'll see when your program is loaded:



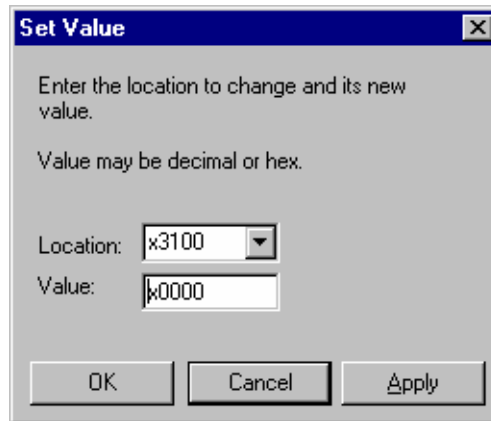
Notice that the first line of your program, no matter what format you originally used, is gone. That line specified where the program should be loaded in memory: x3000. As you can see if you scroll up a line or so, the locations before x3000 are still all 0s. Since nothing has happened yet (you haven't started running or stepping through your program), the temporary registers (R0 through R7) still contain all 0s, the PC is pointing to the first line of your program (as is the blue arrow), and the IR is empty.

Loading the data (ten numbers) into memory

There are several ways to get the ten numbers that you're planning to add into the memory of the LC-3 simulator. You want them to begin at location x3100.

First way: click in the "Jump to" box to the right of the buttons on the toolbar, and type the hex number x3100. When you press return, you'll jump x100 locations ahead in the memory display, so that location x3100 is the first one shown.

Now double-click anywhere on line x3100. You'll get this popup window:



In the “Value” box, type the hex number x3107, and choose OK. Now your first data location will look like this:

```

      x3100 001100001000000111  x3107          ST  R0, x3107

```

Notice that you get to see the binary representation, the hex representation, and some silly, useless assembly language representation (ST R0, x3107). Of course, this is data, not an instruction, but the LC-3 simulator doesn’t know that. In fact, the contents of all memory locations are equal in the eyes of the LC-3, until they get run as instructions or loaded as data. Since you have a halt instruction far before the place where you’re putting this data, it will never be treated as an instruction. So ignore that assembly language interpretation.

You can double-click on each line in turn and enter the data. If you only want to open the popup window once, keep changing the value of the Location field, enter the next number in the Value field, and click the Apply button. When you’re done, click OK.


Second way: go back to the LC3Edit program, and enter this code in hex.


```

3100          ;data starts at memory location x3100
3107          ;the ten numbers we want to add begin here
2819
0110
0310
0110
1110
11B1
0019
0007
0004

```

Save this code as *data.hex* by clicking on .

As usual, the first line is the address where we want the data to begin. The other lines are the actual data we want to load into memory. Click on , since you typed your program in hex. Now, a file called *data.obj* will exist wherever you saved .hex file.

Now go back to the simulator, choose Load Program  once again, and select *data.obj*. Note that you can load multiple .obj files so they exist concurrently in the LC-3 simulator's memory. The memory locations starting with x3100 will look like this:

```

▪ x3100 0011000100000111 x3107 ST R0, x3008
▪ x3101 0010100000011001 x2819 LD R4, x311B
▪ x3102 0000000100010000 x0110 NOP
▪ x3103 00000001100010000 x0310 BRP x3014
▪ x3104 0000000100010000 x0110 NOP
▪ x3105 0001000100010000 x1110 ADD R0, R4, R0
▪ x3106 0001000110110001 x11B1 ADD R0, R6, #-15
▪ x3107 00000000000011001 x0019 NOP
▪ x3108 0000000000000111 x0007 NOP
▪ x3109 0000000000000100 x0004 NOP
▪ x310A 0000000000000000 x0000 NOP

```

Now your data is in place, and you're ready to run your program.

Running your program

Click on the "Jump to" field, and choose x3000 as the location where you want to go.

This next step is VERY important: double-click on the little grey square in front of the line at address x3009.

```

▪ x3009 1111000000100101 xF025 TRAP HALT
↑
double-click
here!

```

That sets a breakpoint on that line. If you don't follow this suggestion, you'll never see your result in R1, because we'll do the trap routine for HALT, which changes R1 before it halts the simulator. (I'll explain breakpoints in more detail in the next chapter.) After you double-click, the line will look like this:


```

● x3009 1111000000100101 xF025 TRAP HALT

```

That red blob is a stop sign. So we'll stop when we get to line x3009, before we run the instruction there.


Now you're ready to run your program. Make sure the PC has the value x3000, because that's where the first instruction is. If it doesn't, double-click on the PC value near the top of the interface, and change it to x3000.


Now for the big moment: click on , the Run Program button!

If you've already added up the ten numbers you put into the data section of your program, you know that x8135 is the answer to expect. That's what you should see in R1 when the program stops at the breakpoint. (In decimal, the result is -32,459. It's negative because the first bit in x8135 is a 1, which is a negative 2's complement number.) You'll get a popup window telling you that you've reached a breakpoint, which in this case is the event when the PC gets the value x3009.

Stepping through your program


So now that you've seen your program run, you know it works. But that doesn't give you a good sense for what's actually going on in the LC-3 during the execution of each instruction. It's much more interesting to step through the program line by line, and see what happens. You'll need to do this quite a bit to debug less perfect code, so let's try it.

First, you need to reset the very important program counter to the first location of your program. So set the PC back to x3000. You can either double-click on it, and enter a new value, or use this quicker method: click on line x3000, and then click on , which sets the PC to that location. Now you're ready to step through your program.


Click , Step Over, one time. A few interesting things just happened:

- R1 got cleared. (If you "cleaned up" by clearing R1 before you started, this won't be an exciting event.)
- The blue arrow, and the PC, both point to location x3001 now, which is the next instruction to run.
- The IR has the value x5260. Look at the hex value of location x3000. That is also x5260. The IR holds the value of the "current" instruction. Since we finished the first instruction, and have not yet run the second, the first instruction is still the current one.

Click , Step Over, for a second time. Again, notice the new values for the PC and IR. The second instruction clears R4.

Click , Step Over, a third time. The PC and IR update once again, and now R4 holds the value x0A, which is decimal 10, the number of times we need to repeat our loop to add ten numbers. This is because the instruction which just executed added x000A to x0000, and put the result in R4.

Continue to step through your program, watching the results of each instruction, and making sure they are what you expect them to be.

At any point, if you "get the idea" and want your program to finish executing in a hurry, click on the Run Program button, , and that will cause your program to execute until it reaches the breakpoint you set on the Halt line.

So now you know how it feels to write a program perfectly the very first time, and see it run successfully. Savor this moment, because usually it's not so easy to attain. But maybe programming wouldn't be as fun if you always got it right immediately. So let's pretend we didn't. The next chapter will walk you through debugging some programs in the simulator.