In this sandbox (ungraded) lab, there are two parts:

1. An assembly language program sandbox, and
2. A Jade lab that, if you opt to do so, allows us to test your circuit by adding names to signals and modules. We will also put the LC3 into a test module, which will then let us drop it into future labs and hookup to new memories easily. In the next labs, you will test your code using our LC-3 Lite solution for a grade, and can also optionally test (ungraded) on your own LC-3 Lite design.

Both of these parts are not graded, and part 2 is optional and only needed if you want to run the assembly program labs on your LC-3 Lite. Feel free to move onto the first assembly language lab.

Assembly Language Program Sandbox

Included below is a sandbox where you can become familiar with the assembly language conversion tool. Remember to use only the instructions supported by the LC-3 Lite: ADD, AND, NOT, LDR, STR, BR, and HALT. We have already entered HALT into each instruction window as the last instruction in your program.

Enter your code in the left hand side window, which will already contain a section of the code for each problem. (**Don't remove this code!**) The .ORIG 0x3000 directive causes the first instruction to be entered into memory at location 0x3000, and subsequent instructions into successive memory locations. The HALT instruction should be left in place as the last instruction in your program. The .DATA 0x4000 directive places the data starting at address 0x4000.

Your code should be entered in the space between .ORIG and HALT. As you do so, the line numbers and binary values for the instructions appear in the right hand side window. The assembler automatically generates the code as you type (it refreshes every 3/4s of a second). The Check button will not respond. It should give errors (red 'X' next to the line number) when it runs into a problem. If new machine code on the right side does not appear, there's usually a problem with your input on the left side.

Credit: This javascript LC-3 assembler was written by Tom Alexander and acquired via github: https://github.com /Tom-Alexander/lc3-assembler. It is used with modifications to support memory address insertion and the .DATA directive.

LC-3 ASSEMBLER SANDBOX (UNGRADED)

Help

LC-3 Assembler

```
.ORIG 0x3000
HALT
.DATA 0x4000
.END
```

```
@0b11000000000000
0b1111000000100011
@0b100000000000000
```

Check button now submits assembly to edX!

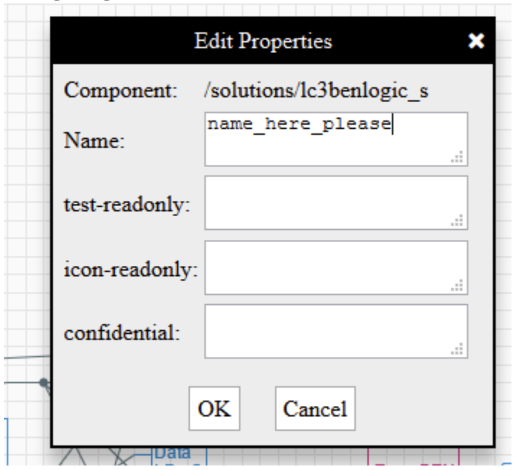
Check

Show Discussion

New Post

Testing submodules in Jade (Optional/Ungraded)

To use your own LC-3 Lite design, which is **completely optional and ungraded**, we need to reach and label signals in lower levels of the hierarchy. To do so, we need to give every module a *name*. You can do this easily by double clicking the module and giving it a name, like so:


Edit Properties

Component: /solutions/lc3benlogic_s

Name: name_here_please

test-readonly:

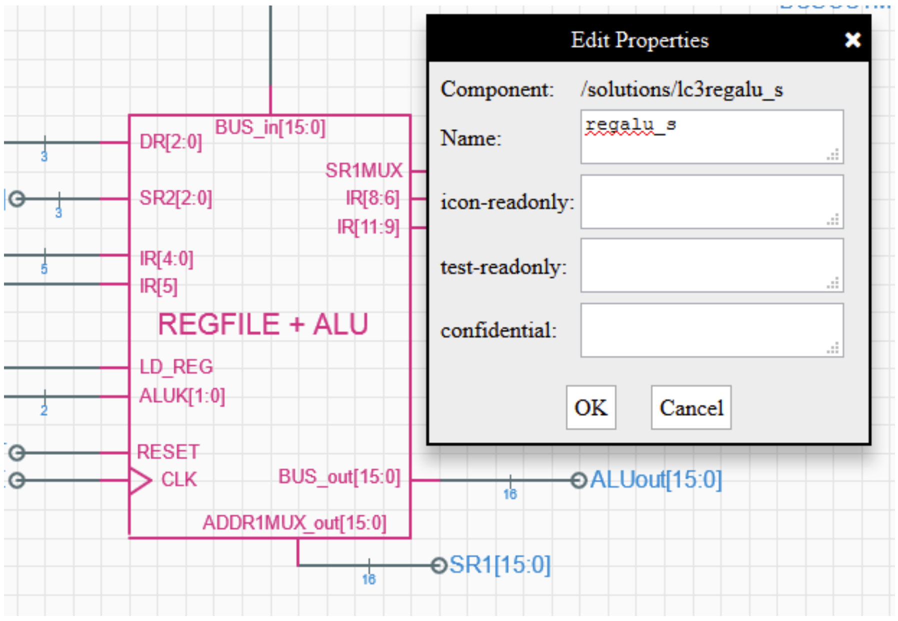
icon-readonly:

confidential:

OK Cancel

We would like to access the registers. They are buried a few levels down, within the '**lc3regfile**' module, and the '**lc3regalu**' module on top of that. In your previous labs, load them up, double click on the modules, name them "regfile_s" and "regalu_s", respectively, and save them. These are shown in the pictures below:

In **lc3top:**



Edit Properties

Component: /solutions/lc3regalu_s

Name: regalu_s

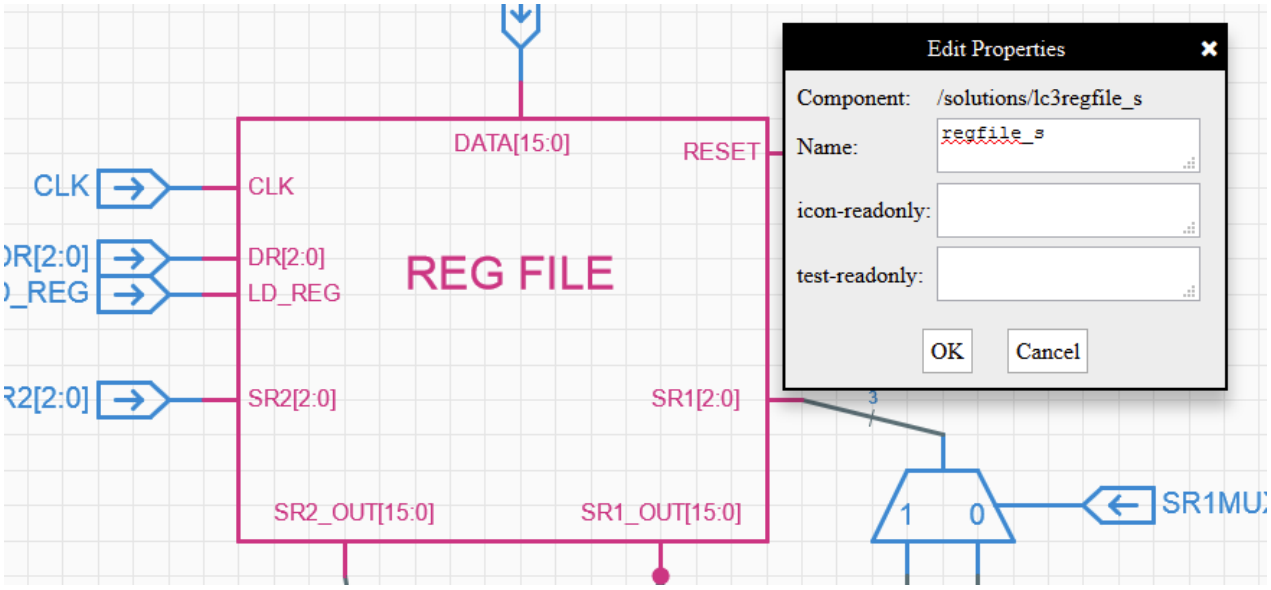
icon-readonly:

test-readonly:

confidential:

OK Cancel

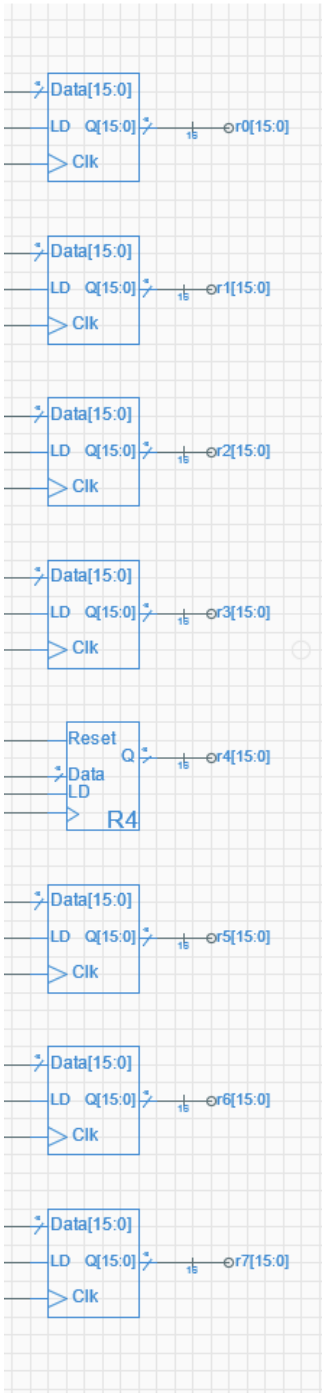
In **lc3regalu:**



Component: /solutions/lc3regfile_s
Name: regfile_s
icon-readonly:
test-readonly:

Finally, in the **lc3regfile** lab, we want to name the registers. Name them "r0[15:0]" , "r1[15:0]" .. and so on. Here is a picture of how this will look:

Help

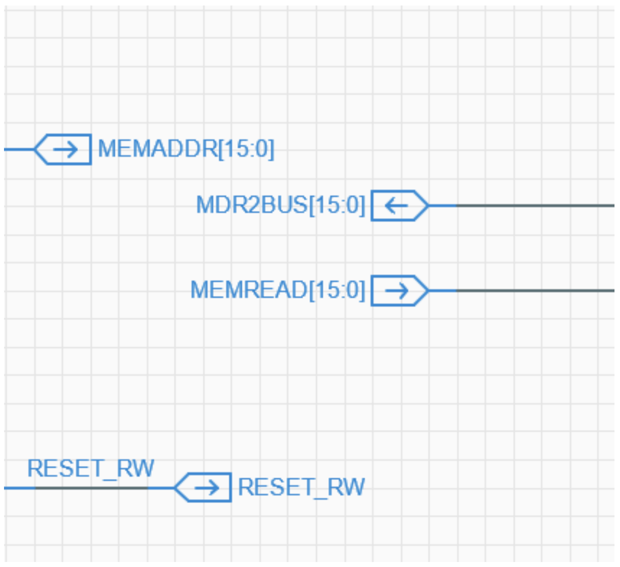


Now that we have things properly named, we can call them from higher levels of hierarchy. In the test tab, use '.' to call names from submodules. For example, a wire or signal "A" in a submodule "outsidemux" would be accessed as "outsidemux.A". In our LC-3 Lite, r5 would be accessed from lc3top as "regalu_s.regfile_s.r5[15:0]". If we go up another level of hierarchy and use the lc3top in a new module, we could name the lc3top module itself "lc3", which would then let us access r5 as "lc3.regalu_s.regfile_s.r5[15:0]".

After you have labeled your LC-3 Lite submodules, now we want to build a version of the LC-3 Lite where we can plugin memories to it.

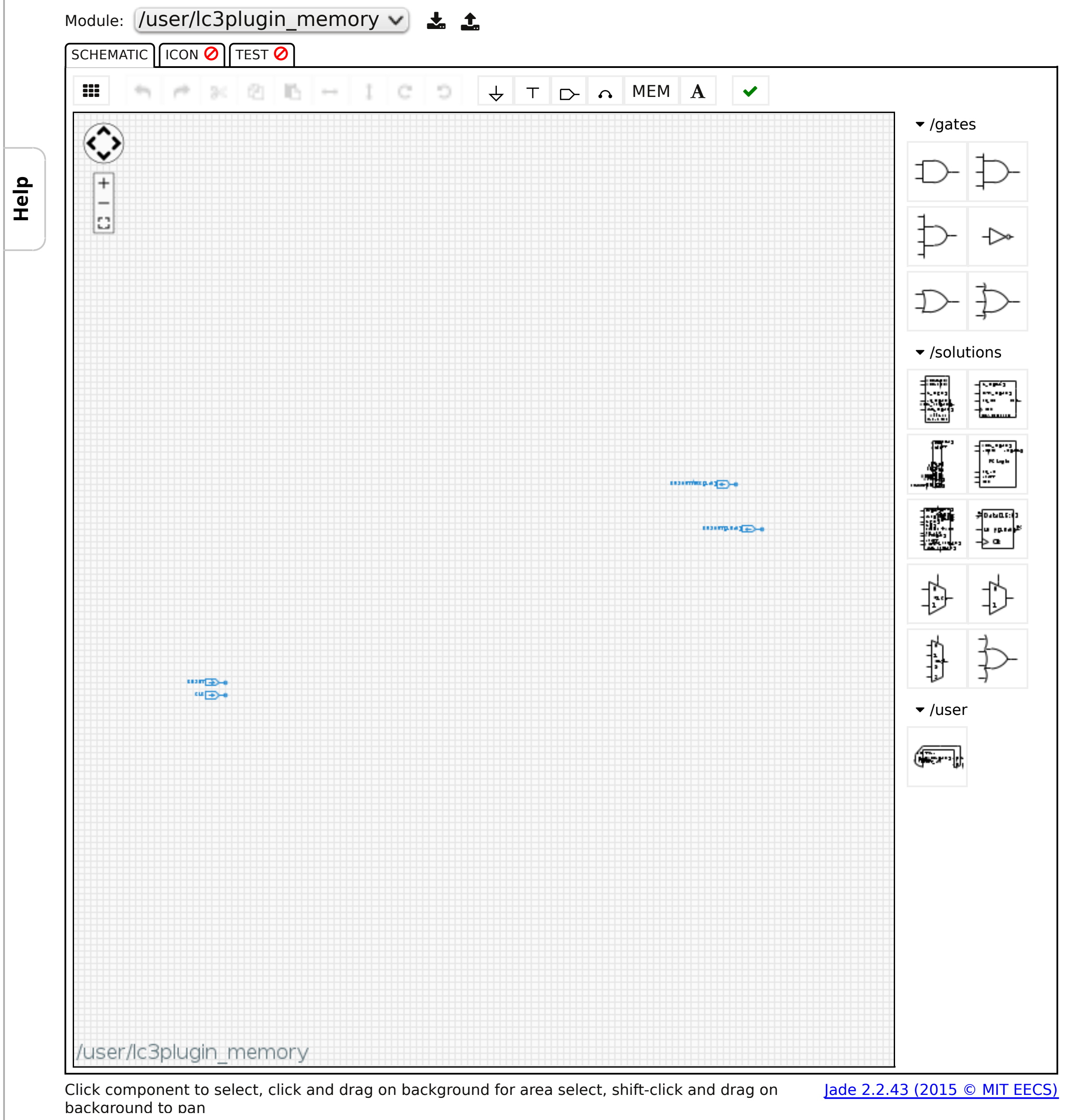
Below is a Jade instance where you can do so. Load your **/user/lc3top** module, and then copy its entire contents over to the **/user/lc3plugin_memory** module.

Then, delete the memory and instead make these 4 signals outputs:



Save the **/user/lc3plugin_memory** circuit. To check it's functionality, we will use it in another Jade window.

MAKING YOUR LC-3 LITE WORK WITH PLUGIN MEMORIES (UNGRADED)



Click component to select, click and drag on background for area select, shift-click and drag on background to pan

Jade 2.2.43 (2015 © MIT EECS)

Check

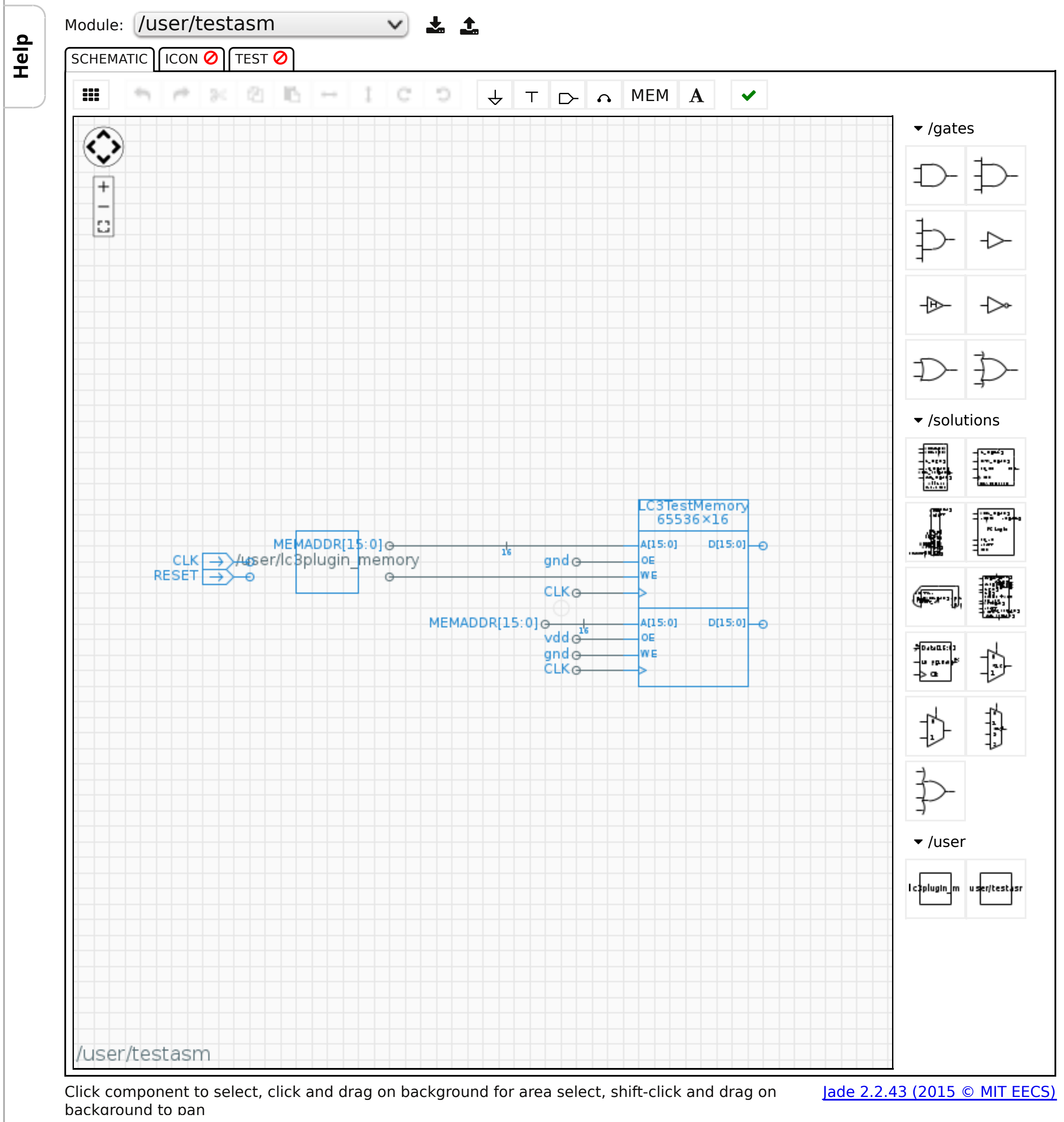
Load your **/user/lc3plugin_memory** module and it should just show up in the schematic below, with all the connections made.

You may need to name the lc3plugin_memory module "lc3".

The memory in the schematic is loaded with the same program as your original LC-3 Lite lab, so if the connections are right, it should pass.

Check to make sure it passes with this test before you use this in the following assembly program labs!

TESTING YOUR LC-3 LITE WITH PLUGIN MEMORY (UNGRADED)



Click component to select, click and drag on background for area select, shift-click and drag on background to pan

Jade 2.2.43 (2015 © MIT EECS)

Check

Show Discussion

New Post

Help

About edX

About

News

Contact

FAQ

edX Blog

Donate to edX

Jobs at edX

Follow Us

Facebook

Twitter

LinkedIn

Google+

Tumblr

Meetup

Reddit

Youtube