← Lab 9: Making the Return Stack component

Finish by midnight on Sunday, 12/2

To help you with the project, you'll be making the Return Stack component.

Components (subcircuits)

A **component** or subcircuit is kind of like a **function:** you have inputs ("parameters") and outputs ("return values"). You **must** use components in your project, and they will be very helpful in organizing the many parts of your CPU.

Components in Logisim have some neat features:

- You can reuse a component as many times as you want (put as many copies as you want)
- You can look inside components while your circuit is running, for debugging
- You can make them look any way you want too!
    - You can be a *component artist* lol

Specifications

Read the specifications **on the project details page here.**

Get your inputs and outputs and tunnels all set up and ready to go. Here's how.

Starting off

Make a new subcircuit (Project > Add Circuit...) and name it something like "Return Stack".

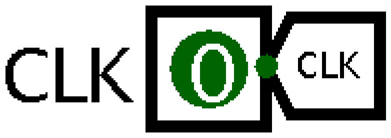
Your return stack will need a number of inputs and outputs. What I like to do is place down **all my inputs and outputs first,** so I don't forget any. It's like the first line of a function.

I also recommend using a tunnel for each input/output, so you can move the inputs and outputs around without disturbing your circuitry.

For the clock input, *use a regular input*. Do not use a clock component. There will be 1 clock component in your CPU, on the main circuit, which you wire into this one.

Saving yourself some tedious work

Here's a tip: let's say you have an input + tunnel like this:

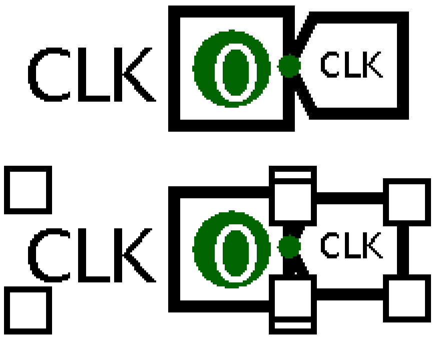


and you want to make **a new input** with a different name and number of bits.

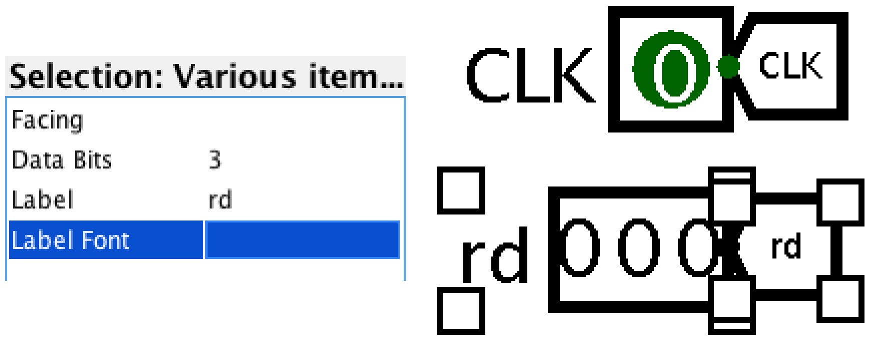
So what you can do is:

1. Draw a box around the input and tunnel to select them both
2. Copy and paste it
3. Draw a box around **both the newly-copied input and tunnel**
4. Change the "Label" and "Data Bits" on the left, and it will change *both things!*

For example, I made a copy and selected it:



and then I changed the data bits and label:


Selection: Various item...

Facing	
Data Bits	3
Label	rd
Label Font	

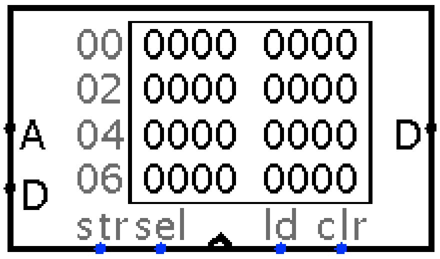
Making the `sp` register and RAM

There is 1 register in this component, `sp`, and it's 8 bits. Make it.

Then make a RAM with the following settings:

- Address bit width: 8
- Data bit width: 16
- Data interface: **separate load and store ports**

It should look like this:



The **str** input is the **write enable**. You can ignore the **sel**, **ld**, and **clr** inputs.

Make sure you hook the clock input up to the register and RAM. It's an easy thing to forget.

Copy and paste tunnels when you can! It will avoid spelling mistakes and saves a lot of time. Grayed-out tunnels are not connected to anything else.

Making the `sp` register count up or down (or neither)

The `is call` and `is ret` inputs will control the behavior of the `sp` register.

If *either one* is 1, the register should change (be enabled). If they're both 0, it should not.

> Do not use comparators for this. Think about the *logic* of it. You can do this with a single gate.

When `is call` is 1, the data going into `sp` should be `sp + 1`.

When `is call` is 0, the data going into `sp` should be `sp - 1`.

Testing it

Set both `is call` and `is ret` to 0. Tick the clock input (just toggle it between 0 and 1). **Nothing should happen.**

Set `is call` to 1. Tick the clock input. **The `sp` value should increase by 1 each tick.**

Set `is call` to 0 and set `is ret` to 1. Tick the clock input. **The `sp` value should decrease by 1 each tick.** It's ok if it wraps around to 0xFF.

Making the RAM work

The only time we write to the RAM is when `is call` is 1. So hook `is call` to its `str` input.

The A (address) input can be one of two things:

- for return instructions, it should be `sp - 1`.
  - *you already have a wire or tunnel which represents this value. use it!*
- otherwise, it should be the current value of `sp`.

The D (data) input should be `PC + 1`.

The D **output** (on the right side) will be your `return address` output.

At this point… you're basically done!

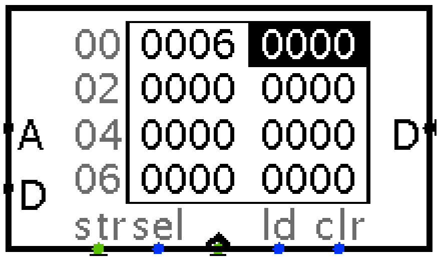
Testing it

Set `is call` and `is ret` to 0. Set the `PC` input to some non-zero value. Tick the clock. **Nothing should change.**

Now change `is call` to 1. The address input to the RAM should be 0 (the current value of `sp`). The data input to the RAM should be `PC + 1`. Tick the clock. Two things should happen:

- `sp` should increase by 1
- The first entry in RAM should become whatever `PC + 1` was

I set `PC` to 5 (binary 101), and ticked the clock, and now my RAM looks like this:

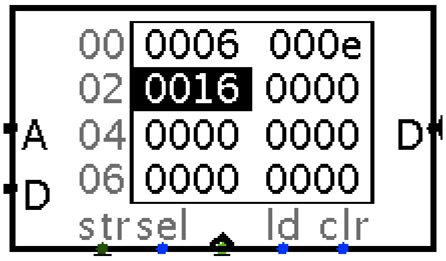


Try pushing some more values by changing `PC` and ticking the clock.

Now let's see if returning (popping) works.

Change `is call` to 0 and `is ret` to 1. The RAM **should not be enabled** (`str` should be 0). The address input to the RAM should be `pc - 1`. The output from the RAM should be the most-recently-pushed value.

I pushed 3 things, so my `sp` = 3, and **before** I tick the clock, things look like this (the highlighted RAM value is the one that is being read):



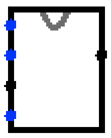
Now, when you tick the clock, **the `sp` should decrease by 1.**

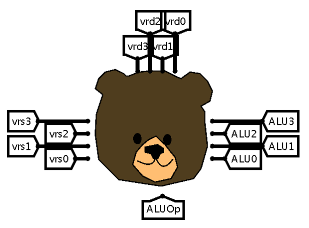
If everything works, congrats! You're done! Now you did a little bit of your project :)

(for fun) Making it look nicer

You don't have to do this, but you can if you want.

Go to the main circuit (double click on the left). Then place your return stack. It looks kinda dumpy by default.





my ALU is a bear, your argument is invalid

If you go into your return stack circuit again, go to *Project > Edit Circuit Appearance*. Here you can move the pins around, draw things, put labels on things, whatever you want. Even something like… a bear.

Submitting

Once you're sure your circuit works, you can submit.

Name your circuit file `username_lab9.circ`, like `jfb42_lab9.circ`.

Submit here.

Drag your asm file into your browser to upload. **If you can see your file, you uploaded it correctly!**

You can also re-upload if you made a mistake and need to fix it.