09-bus-notes

Microcode; On the bus!

[e.g. (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/09-bus/furthur.jpg)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/09-bus/furthur.jpg)

Agenda

- 0. Re-Orienting
- 1. The Microcode Approach
- 2. The Program
- 3. Make ROM for the Logic!
- 4. Abstracting RAM
- 5. Making Buses Work
- 6. Adding a register file
- 7. Doing some work

---

0. Re-Orienting

Electricity to gates

---

gates to higher beasts, such as...

- decoders
- flip-flops
- RAM
- counters

---

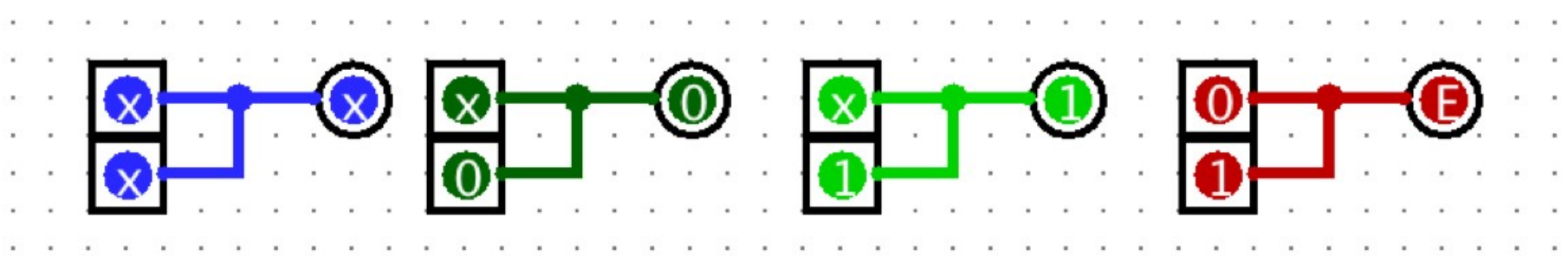
combinatorial logic acted on inputs

A Review of Tri-State

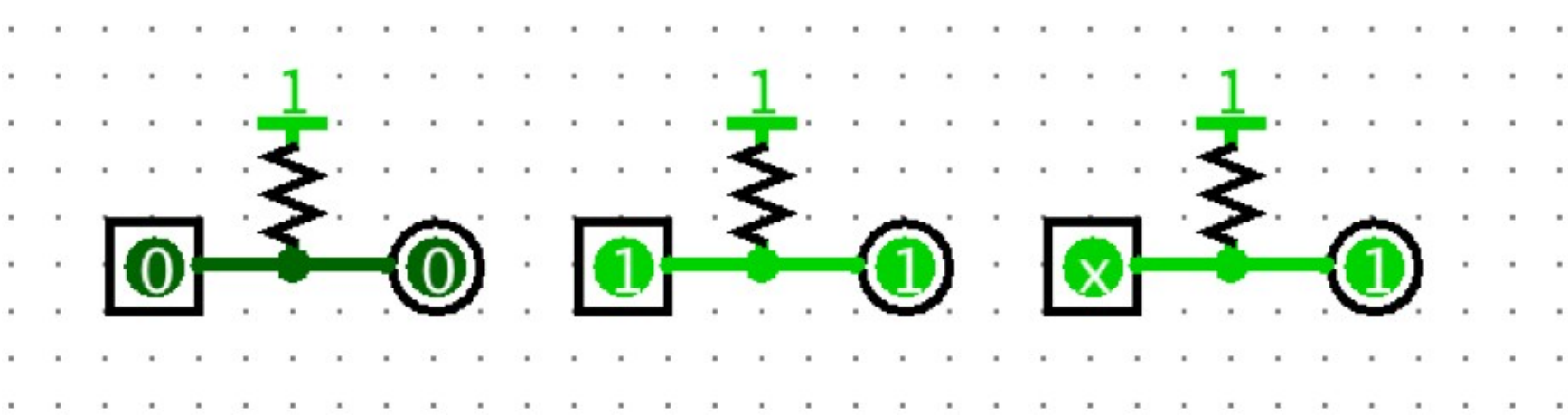
Tri-state controlled buffers can be very useful in your circuits (e.g., they provide a clean way to connect one-of-N byte-wide lines to an output byte).

To review quickly: in addition to 1 and 0, a line can also be "open" or "floating"---not connected to anything.

If you connect two floating lines, they float; if you connect a floating line to a line with a 0 or 1 value, it takes on that value. (But of course, if you connect a 0 to a 1, you get an error)



You can use a **pull-up resistor** to turn a floating line to a 1, but leave a line with another value unchanged.



(LogiSim permits configuring a pull-up resistor to pull to 0, 1, or "error".... but in the real world, you usually see only pull-ups to 1. So stick with that.)

---

sequential logic also acted on stored values... and changed those stored values

---

counters

Expressing a "process" as a finite state machine.... and then building a sequential logic circuit to carry it out

- "state number" is stored in edge-triggered register
- combo logic reads the state registers (and maybe other input), and spits out output, as well as the next state (to be clocked into the register)

Example: the FSM traffic light from Friday.

---

1. The Microcode Approach

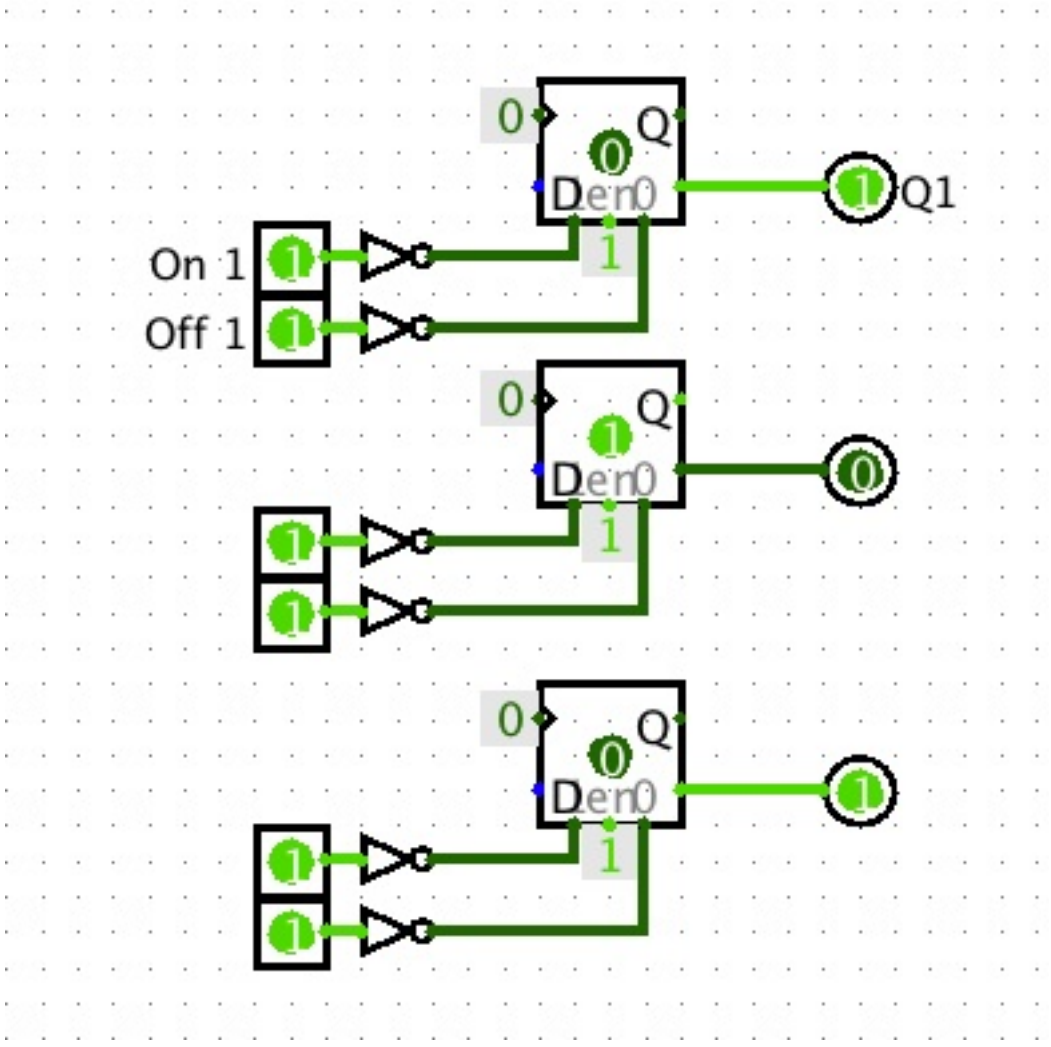
As computer scientists, we like to write programs.

How would you write a PROGRAM that drove the traffic light?

First, we need to think about the "instructions."

Since it's easier to think about "turning a light on" than "keeping it on," we'll add a latch

- 3 bits, one for each light
- (and arranged so the lights will look nice on the right)
- outputs are active-low, so the LED lights up
- inputs are also active-low, because that will let us keep using diodes to connect things up



the "3latch-alow" circuit in [(https://ssl.cs.dartmouth.edu/~sws/cs51-s15/09-bus/demo/traffic-lights.zip>traffic-lights.circ %0D%0A <img%20src=)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/09-bus/demo/traffic-lights.zip>traffic-lights.circ %0D%0A <img%20src=)

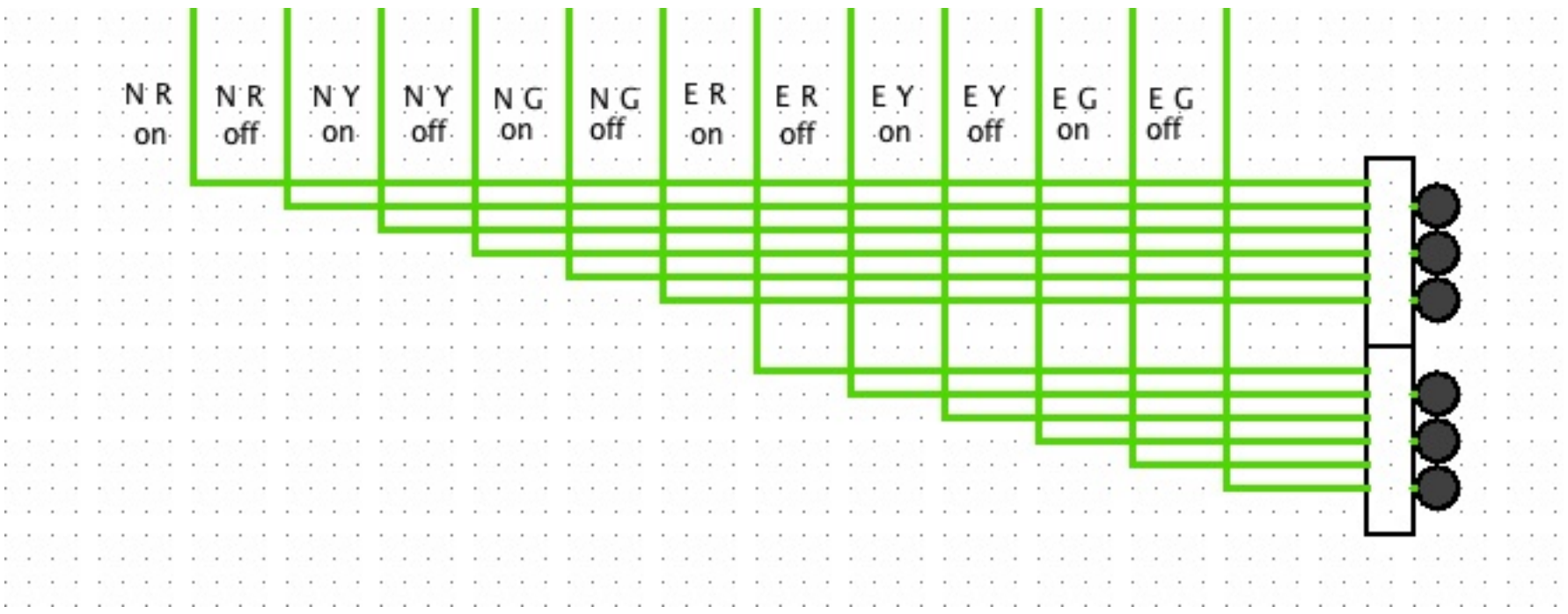
the "latch-demo" circuit in [traffic-lights.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/09-bus/demo/traffic-lights.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/09-bus/demo/traffic-lights.zip)

Maybe we can add dual instructions. "Do this AND do that."

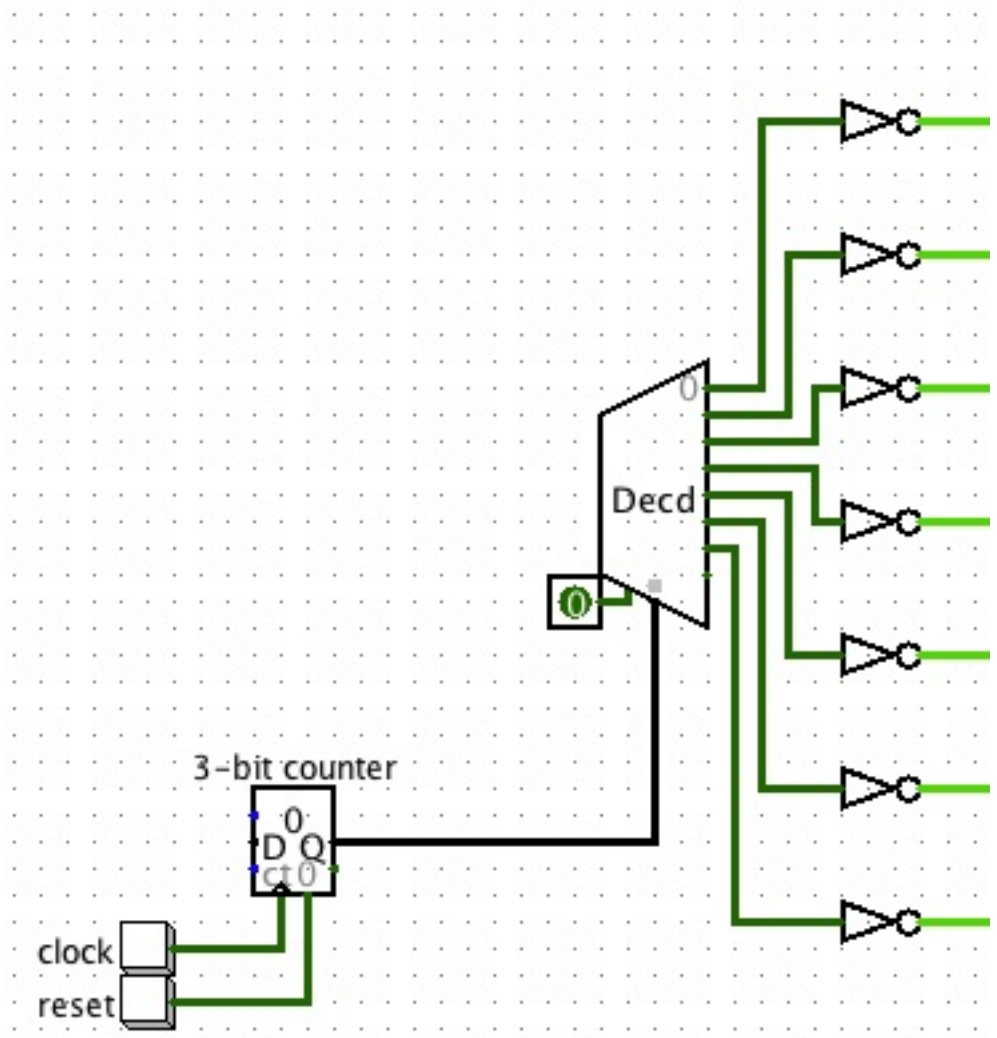
Work through a program for the traffic light

OK, how to build as a circuit?

We need the part that executes the "instructions"

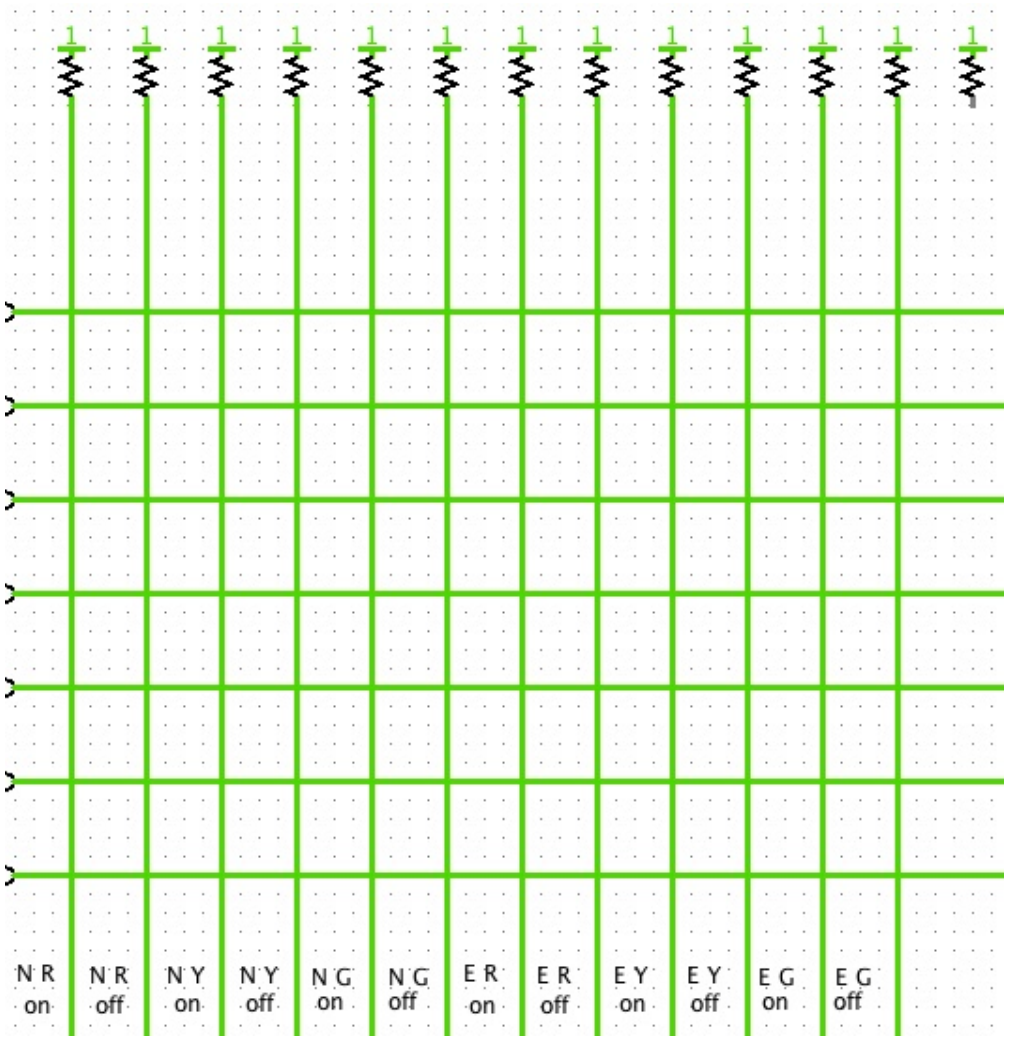


We need to strobe a set of lines, one for each in the program. How can we do that?



Then, we need to wire in the program

the "microcode-skel-0" circuit in [traffic-lights.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/09-bus/demo/traffic-lights.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/09-bus/demo/traffic-lights.zip)



Oops, how are we going to "go to line 0"?

(then.... work through what lines you have to assert low to execute an instruction, and drop diodes there)

2. The Program

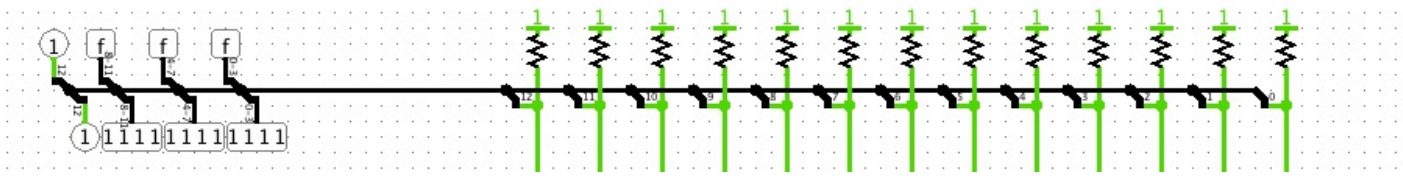
What is the program?

We could think of the instruction code as the bit-sequence across the the control lines.

How could we write our program out?

The hard way: figure out by hand what each instruction word is.

The easy way: take the circuit we built and READ the words off of it!

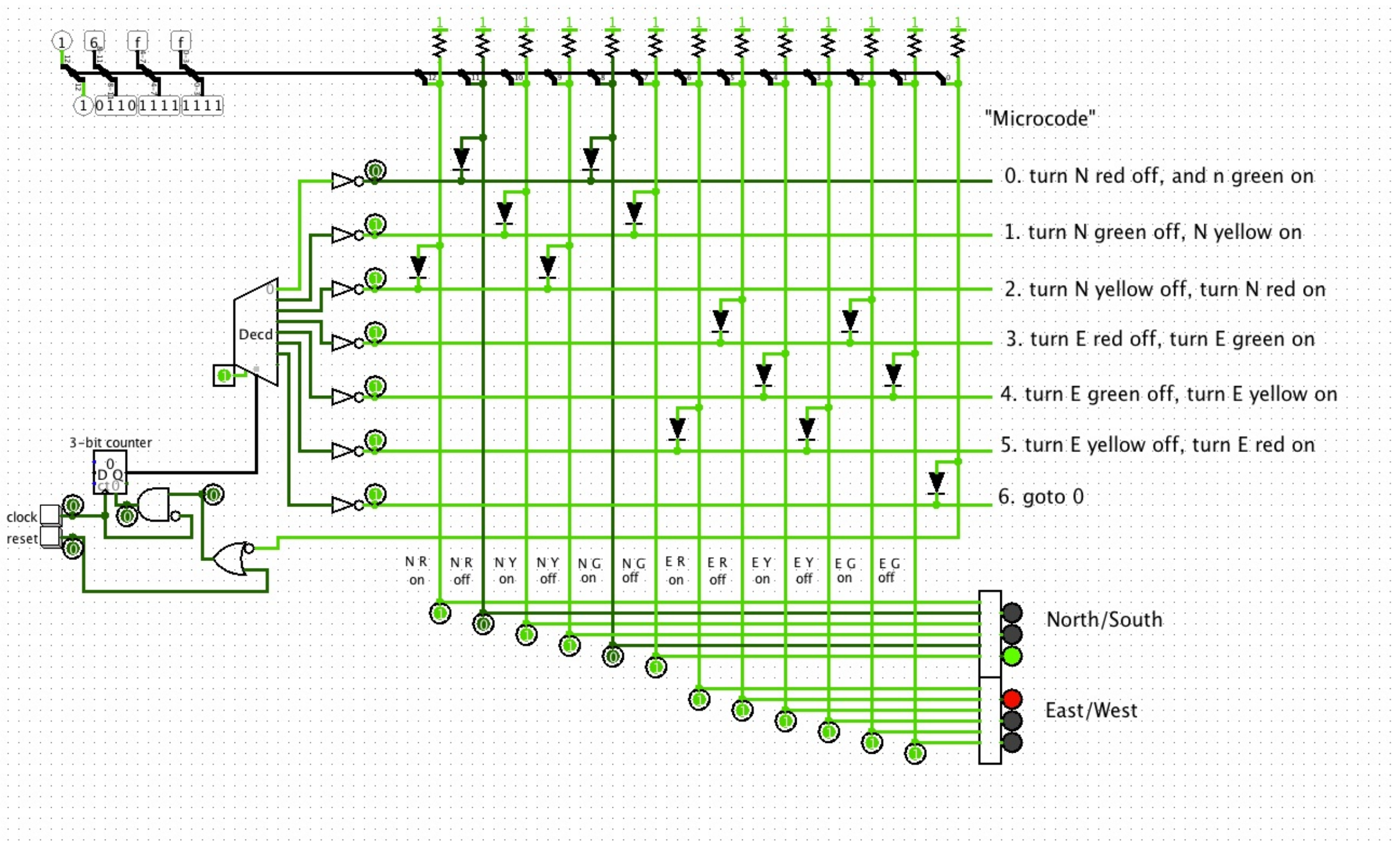


(note use of splitters to pull out each digit's worth of bits from the 13-bit-wide line)

---

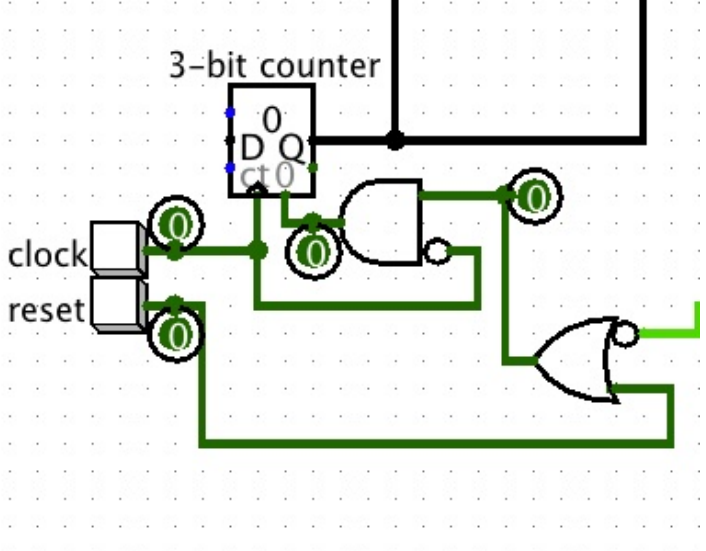
3. Make ROM for the Logic

We end up with "microcode-full" (now annotated with the "program," and cleaned up a bit)



(make you understand how we got here, why the different pieces are in there, and what they do!)

I also cooked up the "goto 0" control so that it doesn't take place until the clock goes down.



At each step in the program, we wiggle the 13 control wires in some specific way.

So we could think of...

- the system governed by a set of 13 control lines
- a specific configuration of the 13 lines constitutes an "instruction"
- and the program constitutes some sequence of instructions

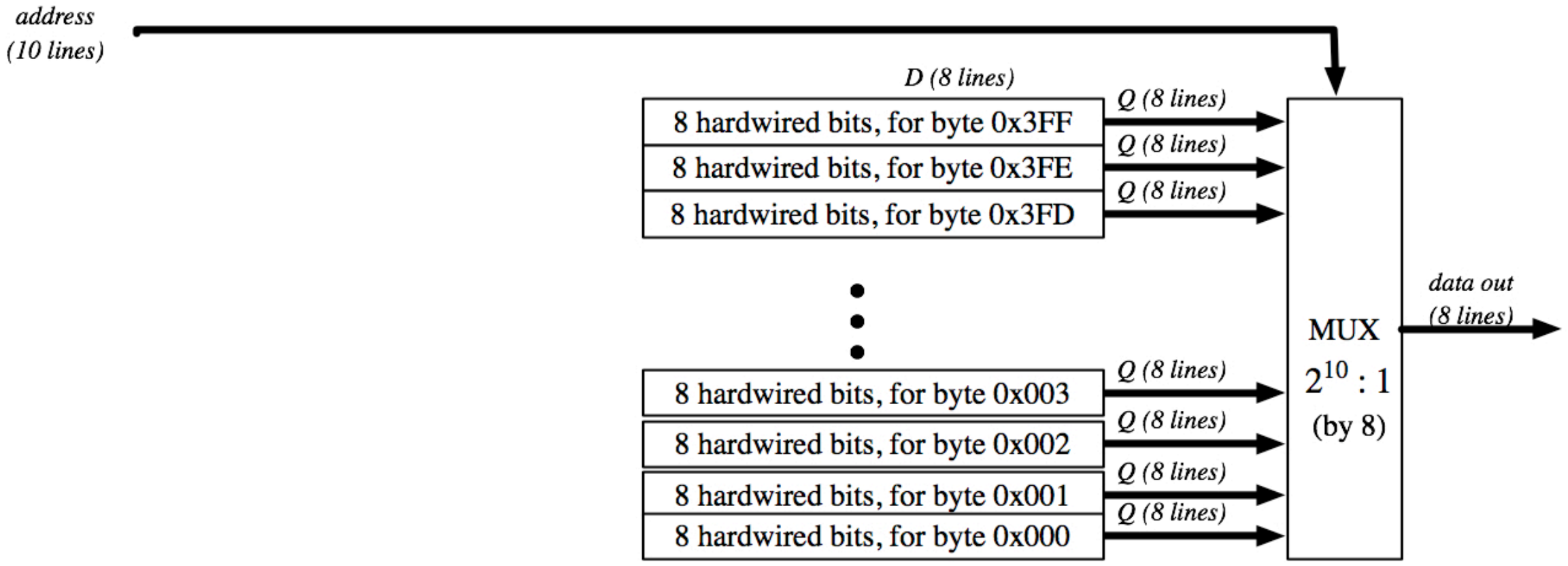
We "play the program" by:

- feeding the address into an active-low decoder
- for each line coming out of the decoder, we wired in the instruction word....
- ...using diodes, to make it easier to elegantly wire it up while also to keep from creating short circuits between the various lines!

---

but we could look at this again, as a box that takes an address in on one side and spits out a control word on the other

Suppose we had a magic box to DO this sort of thing....



1 kilobyte of ROM

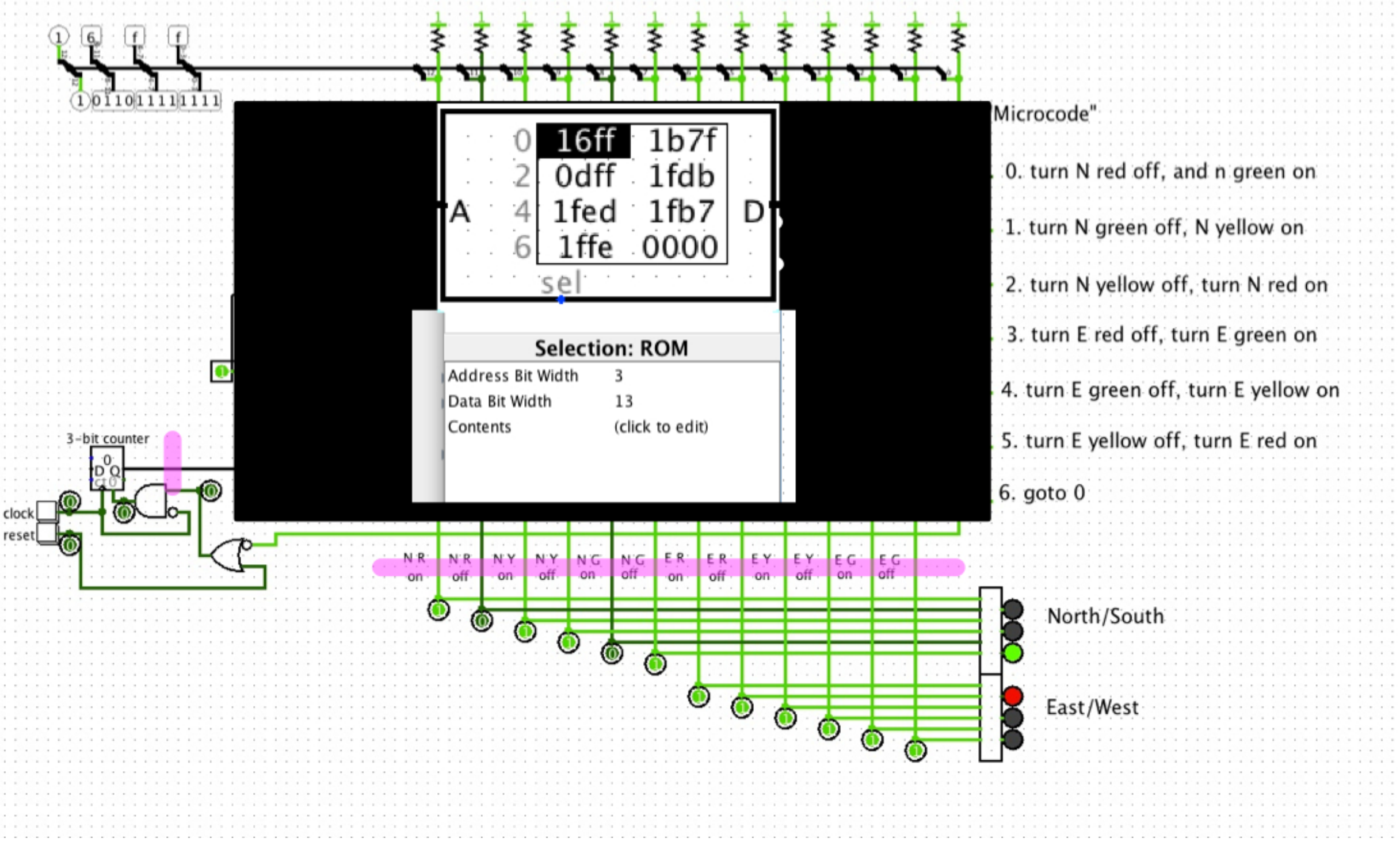
Actually, we do. "ROM"---read-only memory. There are a variety of technologies out there now, some with various ways of programming and then perhaps reprogramming it, thus leading to more complicated acronyms. ("PROM." "EPROM". Etc.)

Contrast with RAM (which is probably better termed "Read-Write Memory", since ROM is random access too, but we're stuck with it now.)

- RAM is rewritable...
- ...and volatile.

...and LogiSim has a "ROM" component...

So we could just code this sequence of instruction up as a ROM... and replace the decoder/diodes/gates with it!



It's starting to look like computation, eh?

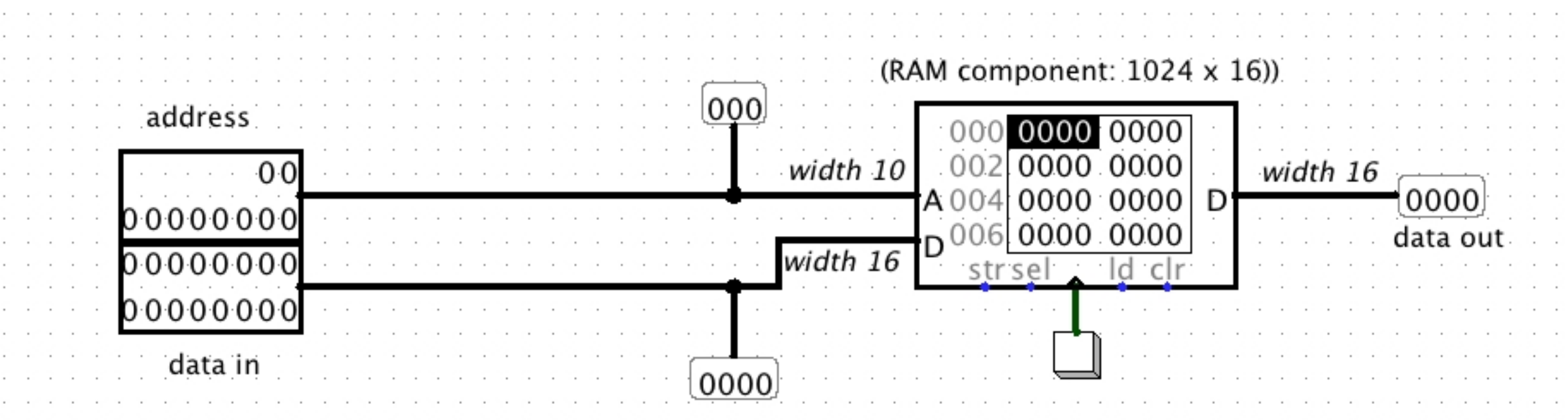
4. Abstracting RAM

We built "big" RAM modules out of flip-flops (and noted how they can also be built from capacitors)

As with ROM, LogiSim also has large RAM modules as built-in components.

E.g. "ramchip-10bit addr" in

bus.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/09-bus/demo/bus.zip)



(Work through the idea of writing to it and reading from it.)

But suppose we want the abstraction of an "address space" even bigger than a single RAM chip.

And we also want to do this in a nice, scalable, pluggable way: just insert more RAM modules into some common lines

---

These common lines are called "buses." In a "textbook" level, we call them things like

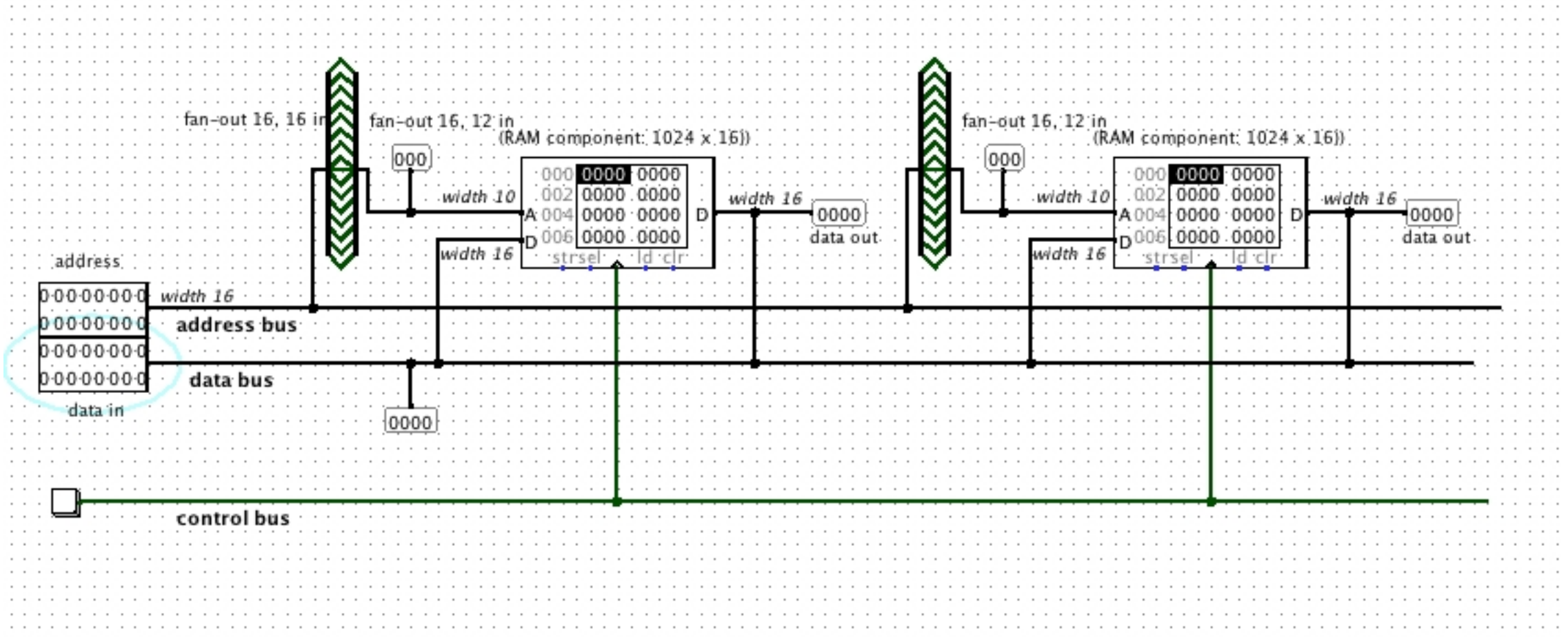
- address bus (send out the address)
- data bus (receive the data, or send it out, if it's a write)
- control bus (e.g., are we reading or writing now?)

But as we'll see later in a "reality check," there are many specific formats of buses (you've probably heard many of the acronyms) and various implementation tricks

---

Let's try extending to a 16-bit address bus. and try making some buses and throwing in some RAM chips. Will this work?

"2 ram chips-v0" in bus.circ



5. Making Buses Work

Depending on how you count, there are at two to four significant problems in the above naive approach.

So, let's try to fix it....(but, for now, WITHOUT using fancier options on the RAM component---the point is to discover what's inside)

"2 ram chips-v1" in bus.circ

Work through the idea of writing to it and reading from it.

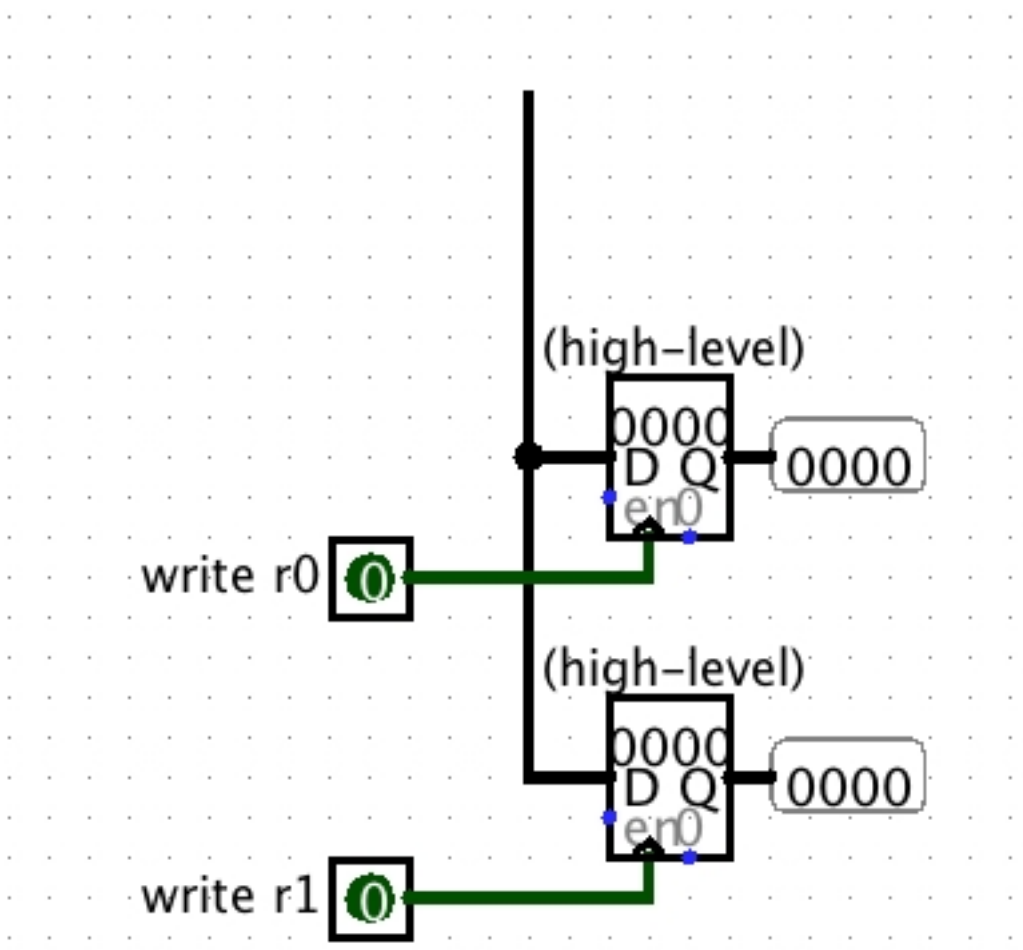
---

6. Adding a register file

Suppose the data source and sink on the left was not really user I/O, but some complicated chip.

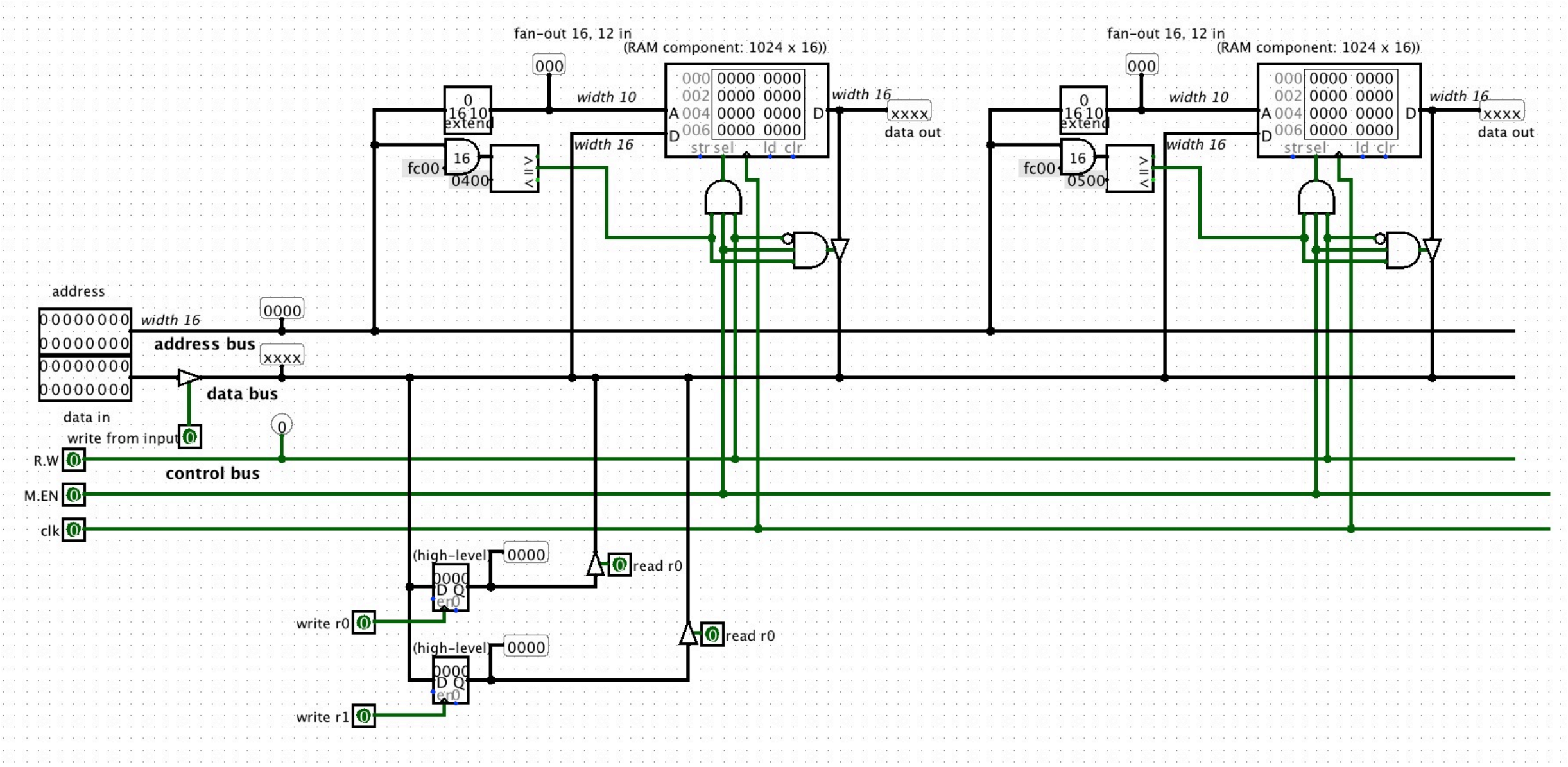
For a number of reasons, we might like to have REGISTERS inside the chip. (Places to store a few words.) (Think: why might we want these?)

"regfile" in bus.circ



7. Doing some work

"full" in bus.circ is the full package...



(and "full-smaller" has some memory thrown out, to make it easier to fit on the screen)

Think about how we might do things like:

- "Move a word from address X to register Y"

How would we wiggle the control lines to do it?

How could we make that automatic?

What about things like...

- "Move a word from register Y to address X"
- "Move a word from register Y to register Z"
- "Add register Y and register Z, and store them in register A"
- "Add the contents of register Y to this address, and move the word at that address to register Z"

---

http://www.youtube.com/watch?v=RXoH3_yL1ng (http://www.youtube.com/watch?v=RXoH3_yL1ng)

(http://www.youtube.com/watch?v=RXoH3_yL1ng)