Sequential Programming, Latches, Flip-Flops

(Not the beach footwear)

Akilesh Praveen — CMSC398E

UMD

April 3, 2020

Agenda

1. Announcements
   - Online Classes
   - Online Resources
   - Project Schedule and Guidelines
2. What is Sequential Logic?
   - Sequential Logic
3. The Idea of 'Memory'
4. Latches
5. Flip Flops
   - Flip Flops vs. Latches
6. Clocks

Announcements

Online Classes!

- Class will be online for the **rest of spring 2020**.
- What does this mean for us?
- Lecture will now be held at **2PM EST** on **Discord**. If you cannot make it, lecture will be recorded and uploaded to an accessible location (TBA on Piazza).
- The CS Submit server is open for business!

Online Resources

- Here's a quick recap of all the resources that we've put together to help you learn and succeed in this class.
  - **Piazza** is the place to ask questions and get answers. Ashwath will be posting frequently here about project and solution releases, so make sure to check in when you can.
  - **ELMS** is where you can take a look at your current grade in the class.
  - The **CS Submit Server** is where all your projects will be turned into. Ashwath's web app is how you'll accomplish this.
  - **The Course Website** is where you can find a cumulative guide for what we've covered in class, separated out by weeks. It's a much more organized way to see our weekly material than Piazza.
  - **Email** is the best and quickest way to reach your instructors.

Discord

- The Discord is up, and assuming that you are watching lecture right now, you've had no issue joining it. For those of you who are looking at this slideset and wish to join the discord, go ahead and check out the relevant ELMS announcement, where Ashwath has uploaded the invite link.
- If you have any issues getting Discord set up, feel free to contact Ashwath or me for help.

Project Schedule

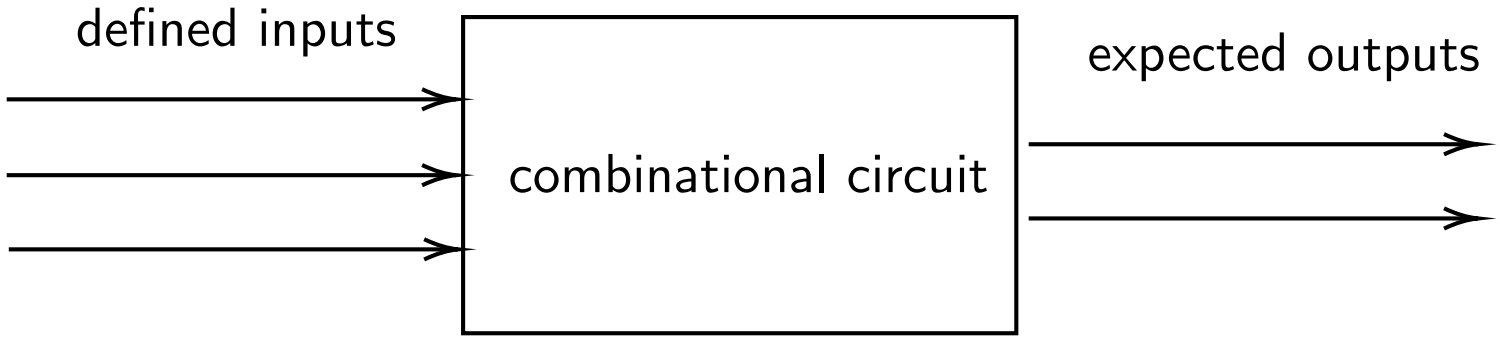
- Circumstances have led us to develop a new project schedule.

New Submission Guidelines

- Now, you can submit your worldfiles to a web app, and testing should take place automatically on the CS submit server.
- More information is available on Piazza, ELMS and Discord.

What is Sequential Logic?

- To understand sequential logic, first we have to make a note of what we know about **combinational logic**.
- Combinational logic basically only relies on the initial inputs to make a decision.

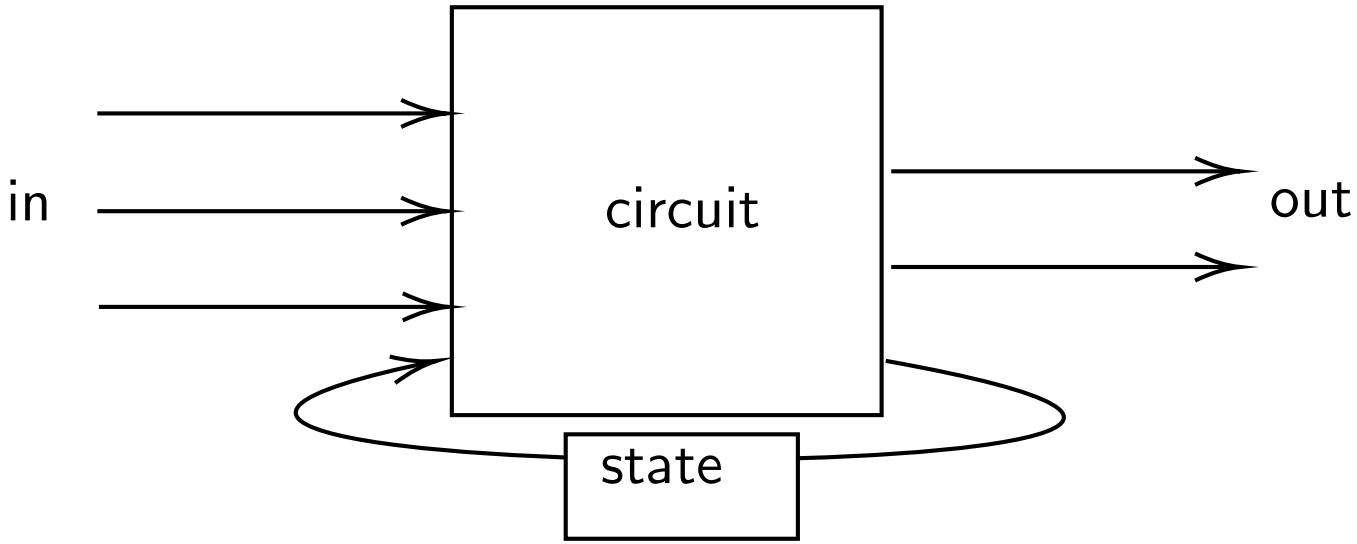


- In other words, we've had a **set truth table** for our combinational circuits; knowing the inputs means you know the outputs.
- *"Your next line will be... "*

Sequential Logic

What is Sequential Logic?

- When it comes to sequential logic, the whole point is **not knowing in one cycle what your output is**.
- In other words, these circuits don't just use the initial inputs, they use the outputs of previous iterations of the circuit to decide the output.



How do we achieve this?

- Actually, it turns out it isn't that bad.
- We're just doing what the diagram says- feeding the outputs of our circuit back into our circuit for further processing!
- If you want a programming analogy, think of a loop statement with a loop variable.
- ```
  def my_loop:
      my_input_variable = 0

      # a perfect example of iterative code
      for i in range(0, 10):
          my_input_variable += 1
  ```
  ```

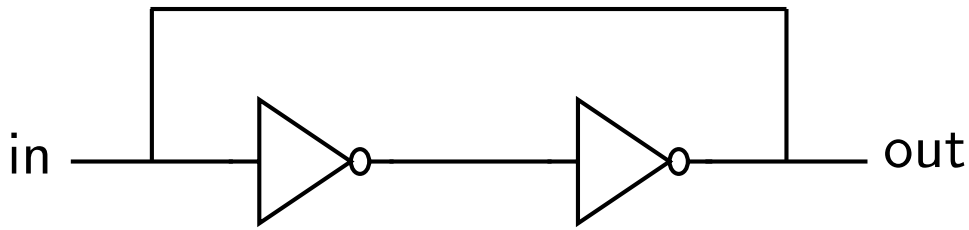
Keeping Track of Values

- Well, that Python segment may have made sense in our heads, but we're missing a key element to help us translate this into digital logic
- We have no idea how to recurrently store the value of a variable every time our circuit executes
- In other words, we're *looping* back to the idea of memory!

Memory: Latches

The First Ever 'Memory'

- The first type of memory ever used was **chained 'not' gates**.
- Go ahead and try to imagine what would happen to the output if the input was 1 or 0. Keep in mind that it's important to consider the output's state before this.



- You'll find that the output stays at 0 until the input becomes a 1. Then, no matter what the input changes back to, the output will still stay at one.
- This keeps track of *something*, but since we can really only set it once, this probably isn't going to help us much in the long run.

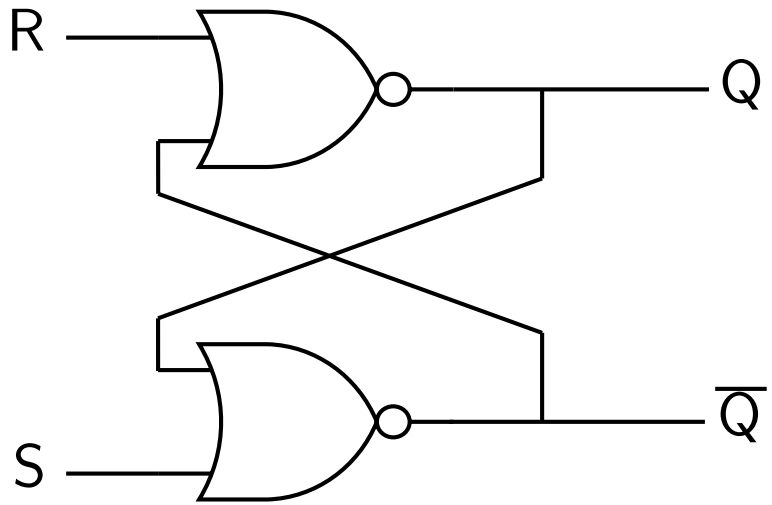
Latches

- We want to do more than set the value of memory just once.
- You could also say that we want to be able to **reset** it.
- I invite you to think for a moment- how can we augment the previous example to allow us to somehow 'reset' the state of our memory?
- Unfortunately, it's not a one-off simple solution, so let's explore it together

Latches

- You can essentially perform this with a commonly-used circuit known as a latch
- There are a few ways to build a latch, but we're going to look at the most basic latch today. (an **SR Latch**)
  - Specifically, a NOR SR Latch (made with NOR gates)
- There are a few other latch designs that I'll be posting about under course resources. You can look at them on your own time, if you'd like.

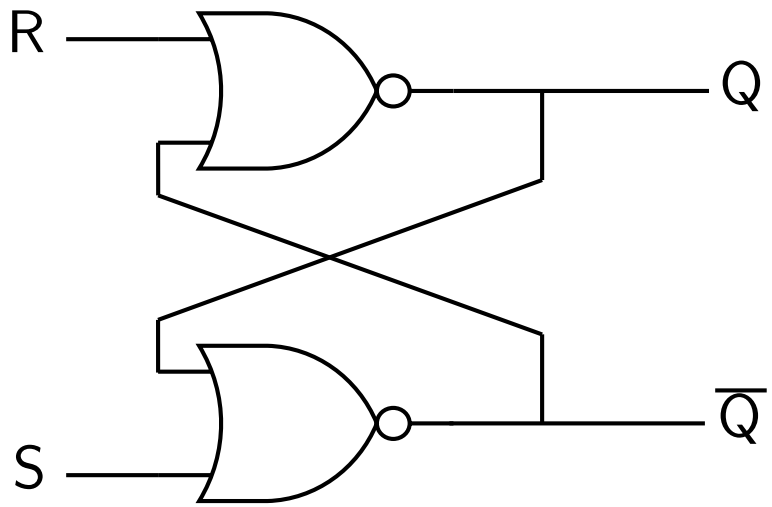
NOR SR Latch



S	R	Q	$\overline{Q}$	
1	0	1	0	'set' state
0	0	1	0	
0	1	0	1	'reset' state
0	0	0	1	

- This is basically a circuit that switches between two ouput states, depending on which of its two inputs has a signal coming through it
- Here's the beauty of it- if there's no input coming in, its state remains unchanged

NOR SR Latch

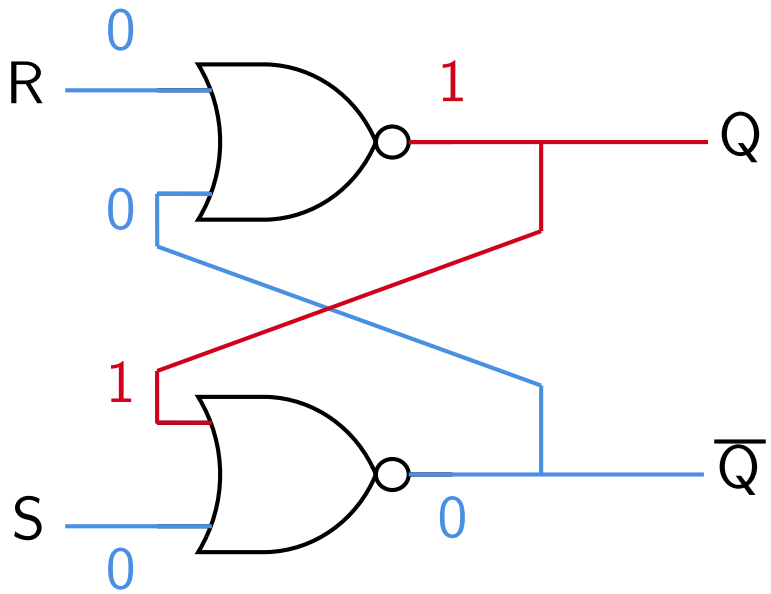


S	R	Q	$\overline{Q}$	
1	0	1	0	'set' state
0	0	1	0	
0	1	0	1	'reset' state
0	0	0	1	

- Note that an input of 1 1 will result in undefined behavior (try it!)
- *Thanks to Ben Eater for the following example. YouTube link can be found under 'Week 8' on course website.*

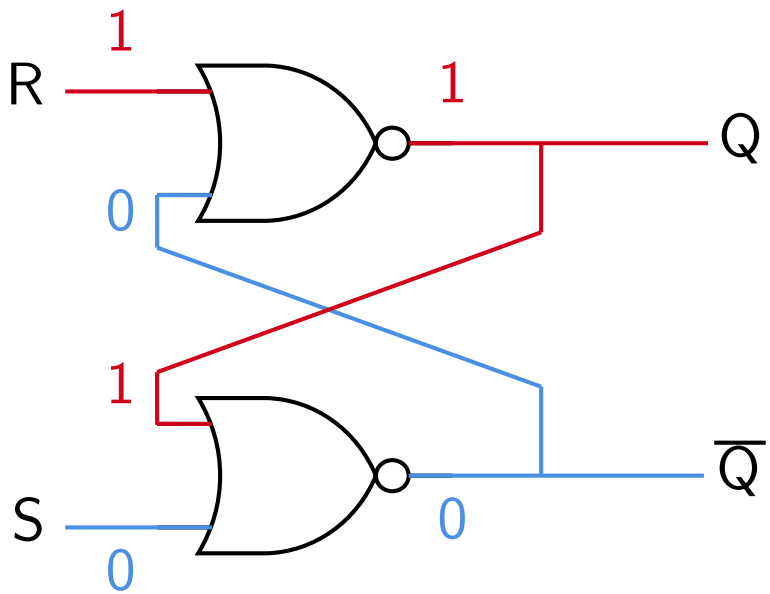
NOR SR Latch

- Now this circuit is a little weird looking, so let's analyze how the signals actually travel.
- You'll notice that we have the potential for infinite looping here, as we talked about a slide or two before.
- Below, you'll see the circuit as it stabilizes with an input of the default `R=0` and `S=0`.



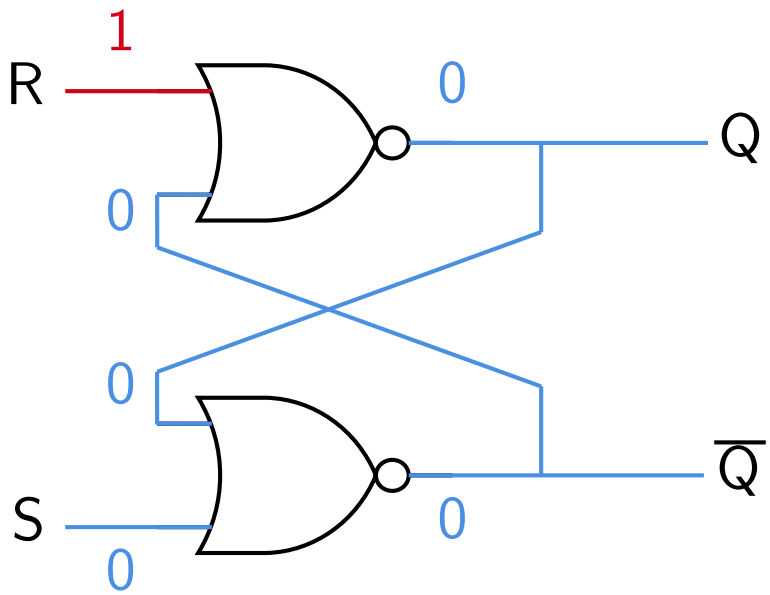
NOR SR Latch

- Now, let's see what happens when we feed the latch a positive signal. Specifically, let's give it a value of '1' from the 'R' input.
- We'll see how the resulting value goes from only Q being on to only $\bar{Q}$ being on
- Here's the **first step**, where we provide R as 1 instead of zero



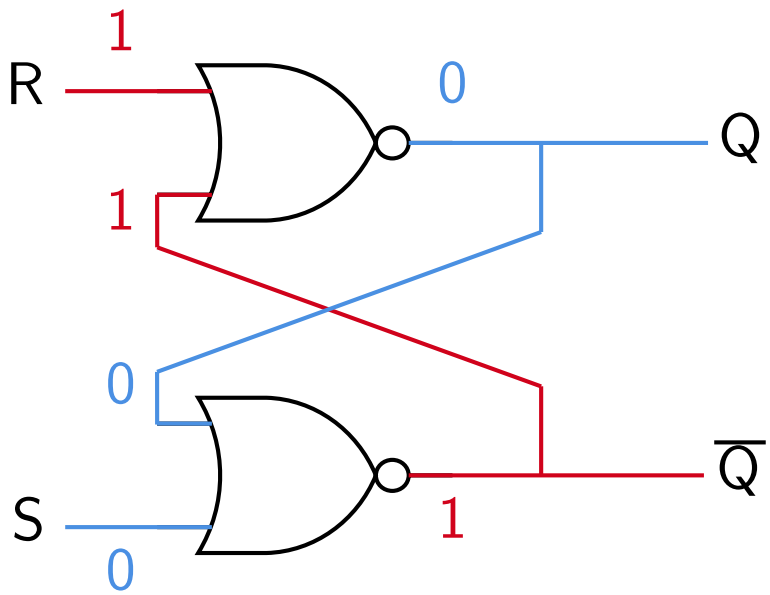
NOR SR Latch

- Here's the **second step**, where the result of the `NOR` gate up top propagates down to the other `NOR` gate.



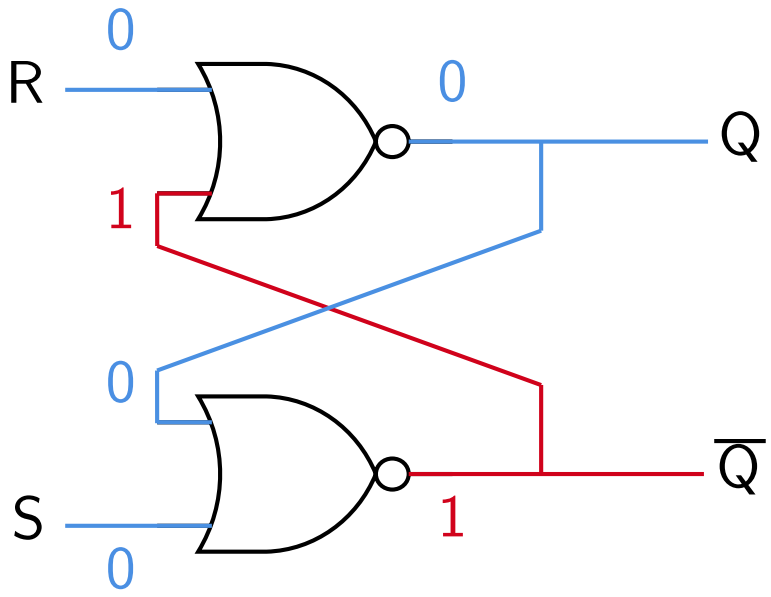
NOR SR Latch

- Here's the third and **final step**. Finally, the result coming out of the bottom `NOR` gate alters the output coming and pointing to $\bar{Q}$



NOR SR Latch

- Note that if we remove the 1 coming in from R at this point, the output states will stay the exact same.



- This is the beauty of latches- they can keep a state for us without us having to keep a signal coming in consistently.
- Circuits like this are the basis for sequential logic.

Other Types of Latches

- First of all, you can make an equivalent latch to the NOR SR Latch by using `NAND` gates. It's basically functionally the same as the NOR SR Latch.
- Additionally, there are other types of Latches. Again, since this is a 1-credit class, this is the one type of latch that we'll be going over in-depth.
- If you want to learn more about latches, we'll be posting some stuff on the course website for you to check out.

Flip Flops and Clocks

What's a Flip Flop?

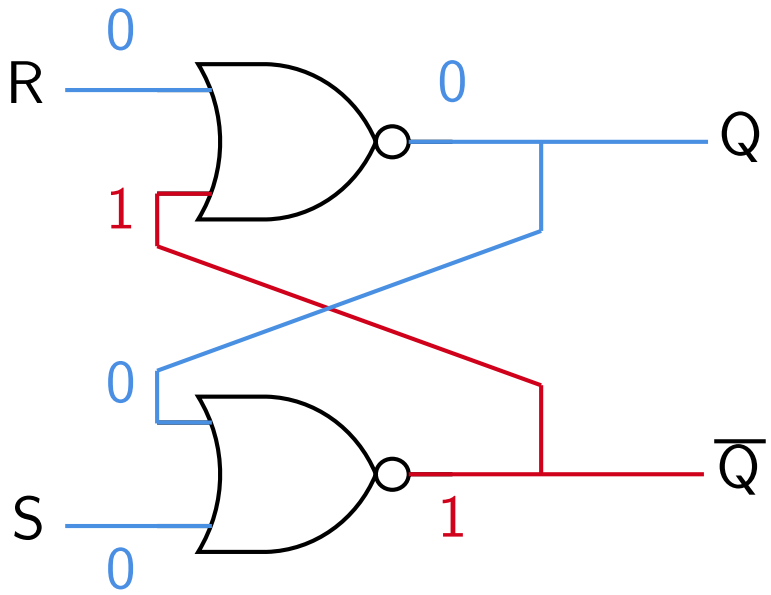
- Certainly not a type of shoe you'd wear to the beach.
- In a sentence, Flip Flops are basically just latches which are only usable when you give a 'control' input is on.
- That may sound silly, but it helps when we want to update memory at certain times

What's a Flip Flop?

- Let's say you wanted your latch(es) to update, but only when you were certain that the inputs for your latch(es) were for sure decided?
- Using a flip flop instead of your latches will only allow the latches to change when that 'control' input switches to on, letting it know that it's okay to update
- If you're having trouble imagining this, think of why we set our test delays to 2+ seconds instead of having them run with no delay

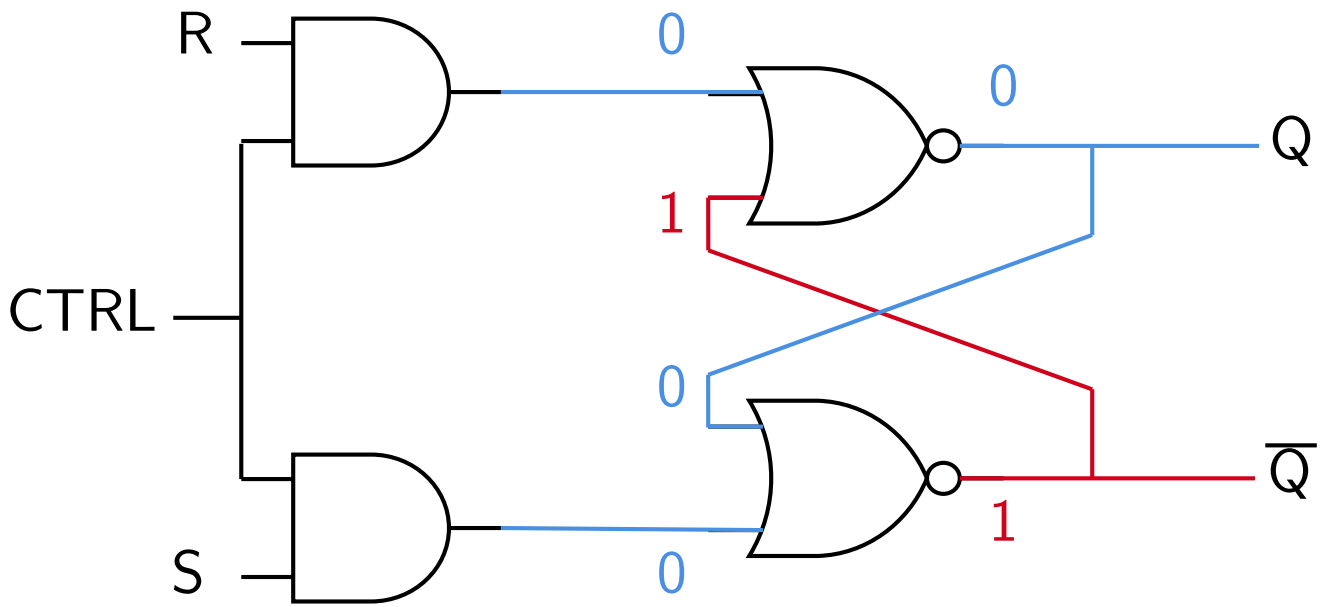
How does that end up looking?

- Let's look at a Flip Flop and a Latch, side by side.
- First, here's a plain old latch.



How does that end up looking?

- And here's the flip flop version.



- Note that without a positive input coming in from the CTRL wire, you wouldn't be able to make any changes with R and S.
- The actual input from the CTRL wire is usually handled by a **clock**, which we'll talk about in a bit.

Flip Flop in Minecraft

- There are plenty of ways to make flip-flops in Minecraft.
- Keep in mind that there are a few variants of flip-flops that you can use, but I'd honestly pick the one here that's easiest to build.
- Minecraft has a feature known as 'locking' redstone repeaters, which we can leverage to make some pretty tiny flip flops.
- Here's a link: https://youtu.be/MPFySvA_Ugs?t=272
- We'll have some videos up on the course website soon, but the above link is a great way to get started.

Clocks

- So what exactly will we be feeding into that control wire?
- Well, we need a way to maintain time- that is, to provide a steady **cycle** for our circuit to periodically update memory.
- Simply speaking, clocks are the way to do that.
- Their only job is to be a circuit that produces a 1, then a 0, then a 1, then a 0, usually on a set timing scheme.

Clocks in Minecraft

- In real life, that's a little tough.
- Long ago, they used to pulse current into Quartz crystal, then use the resulting vibrations to create a reliable clock.
- Luckily, we're just kids playing Minecraft, so Mojang has made Clocks much more simple for us.
- The design below is a little outdated, now we can take advantage of repeaters to 'slow down' a signal for us. Just a few repeaters in a circle should do the trick!
-