Sequential Logic

Introduction to Computer Systems, Fall 2024

Instructors: Joel Ramirez Travis McGaha

Head TAs: Adam Gorka Daniel Gearhardt
Ash Fujiyama Emily Shen

TAs:

Ahmed Abdellah	Ethan Weisberg	Maya Huizar
Angie Cao	Garrett O'Malley Kirsch	Meghana Vasireddy
August Fu	Hassan Rizwan	Perrie Quek
Caroline Begg	Iain Li	Sidharth Roy
Cathy Cao	Jerry Wang	Sydnie-Shea Cohen
Claire Lu	Juan Lopez	Vivi Li
Eric Sungwon Lee	Keith Mathe	Yousef AlRabiah

Poll Everywhere

pollev.com/cis2400

- How are you? Any Questions?

Logistics

- There was ***no check-in*** due today.

- HW05 due @ 11:59 pm on Fri, Oct 11
  - Reminder; only 72 Hour extensions possible

Lecture Outline

- **Sequential Setup**
- R-S Latch
- D Latch & Clock
- D Flip Flops

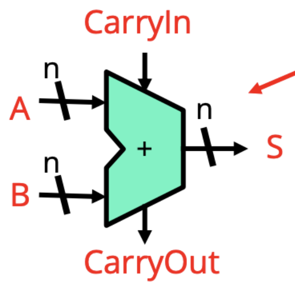
So Far: Combinational Logic

- Always gives the same output for a given set of inputs
  - State-less (i.e., no state or memory)
  - What if I wanted to create something that depended on previous inputs/outputs/other state?

So Far: Combinational Logic

- Always gives the same output for a given set of inputs
  - State-less (i.e., no state or memory)
  - What if I wanted to create something that depended on previous inputs/outputs/other state?

Gate Level Abstraction



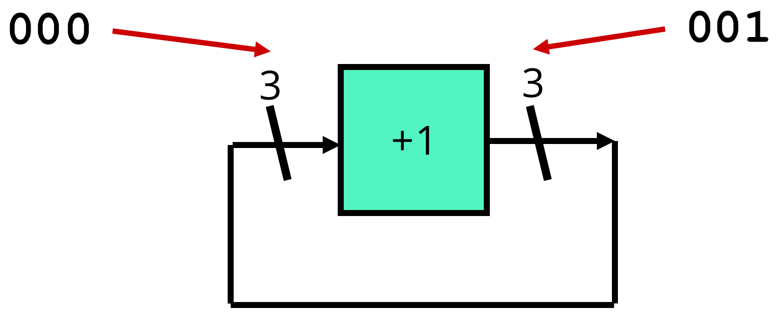
What if I wanted to increment the value S *later*?

Using current methods; we have no way of doing so.

We need to do something different!

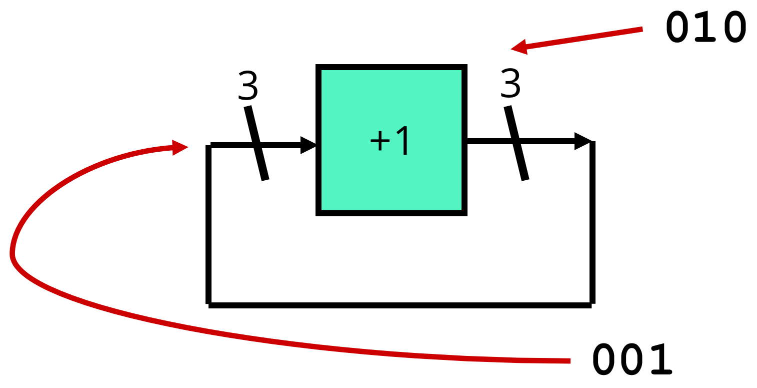
Combinational Counter

- What if we wanted to make a circuit that just continuously incremented an unsigned 3-bit integer *immediately*?
  - We can make this with our usual incrementor
  - What's the Expected Behavior?



Combinational Counter

- What if we wanted to make a circuit that just continuously incremented an unsigned 3-bit integer *immediately*?
  - We can make this with our usual incrementor
  - What's the Expected Behavior?



Poll Everywhere

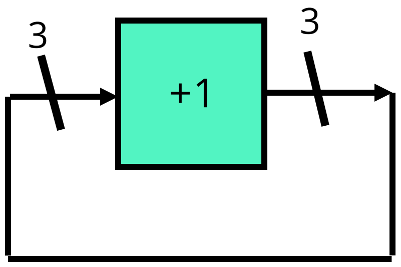
pollev.com/cis2400

- Does the following counter circuit "work” like expected? Will it continuously count up till it overflows and starts at 0 again?

  A. Yes

  B. No

  C. I’m not sure



Poll Everywhere

pollev.com/cis2400

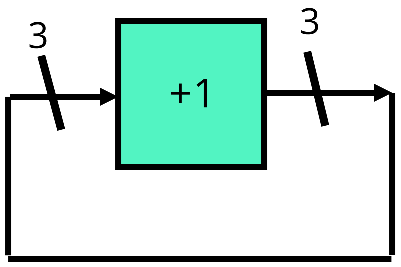
- Does the following counter circuit "work" like expected?

Answer: Depends on how you look at it

A. Yes

B. No

C. I'm not sure



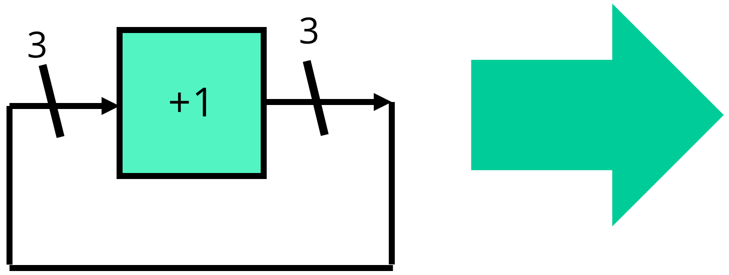
If we interpret the circuit "purely logical", it does count up, but each value is instantly replaced by the next. We'll assume the bits are immediately replaced by the output.

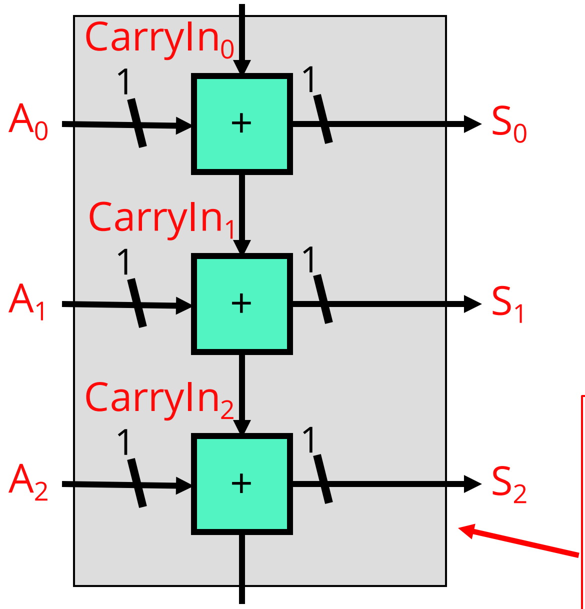
If we consider it as a physical circuit with gate delays...

Combinational Counter

- If we interpret the counter circuit as physical hardware with gate delay: Each bit output depends on the carry in from the previous bits

Circuit is "concurrent". All wires have a voltage, and all transistors are acting simultaneously.





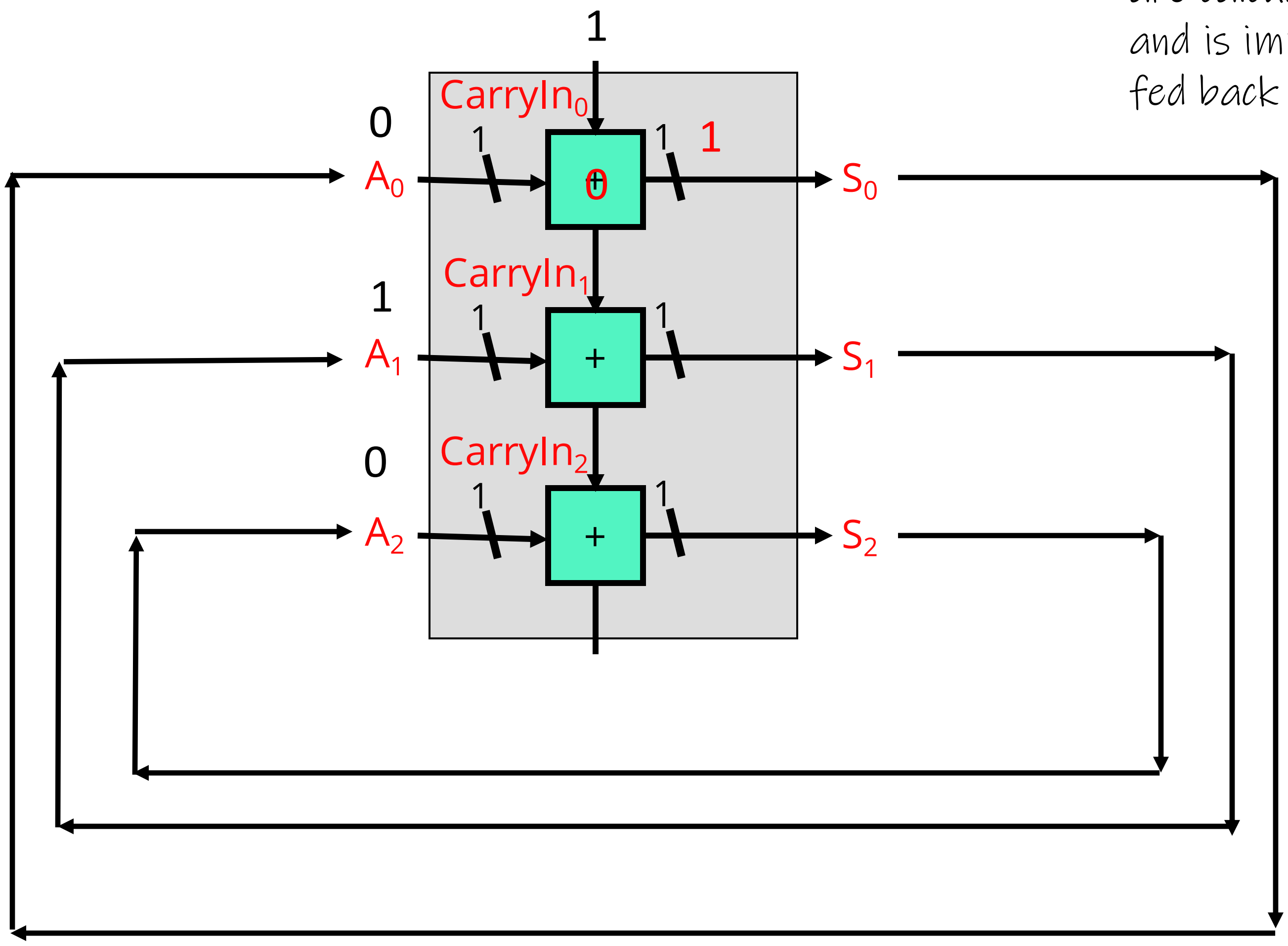
While S_2 is waiting for CarryIn$_2$, S_0 has already been fed back into A_0 and the next increment has already begun

Due to gate delay, the correct output is not calculated before the next increment operation

Combinational Counter

- If we interpret the counter circuit as physical hardware with gate delay:

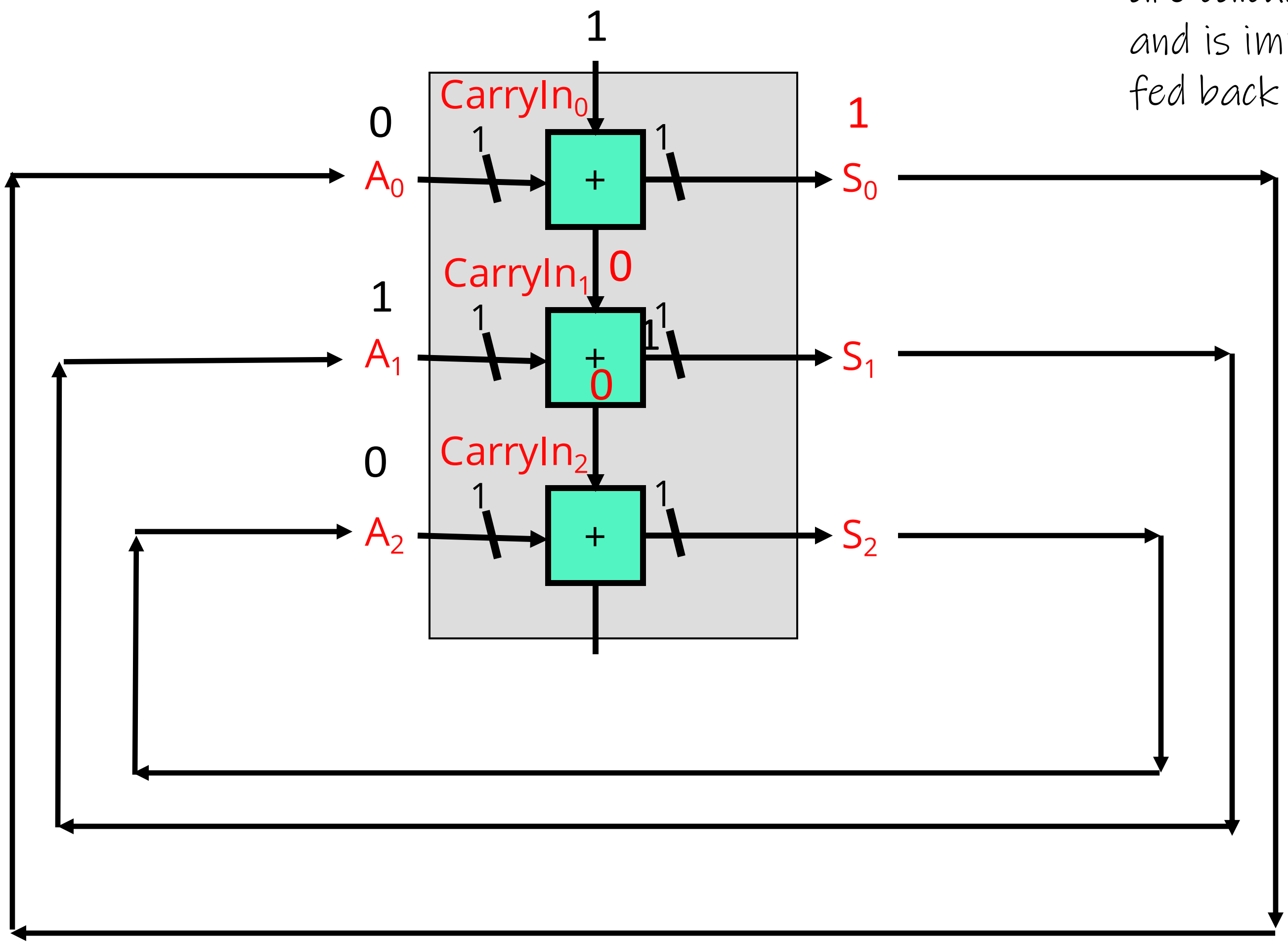
S_0 and $CarryIn_1$ are calculated first and is immediately fed back in



Combinational Counter

- If we interpret the counter circuit as physical hardware with gate delay:

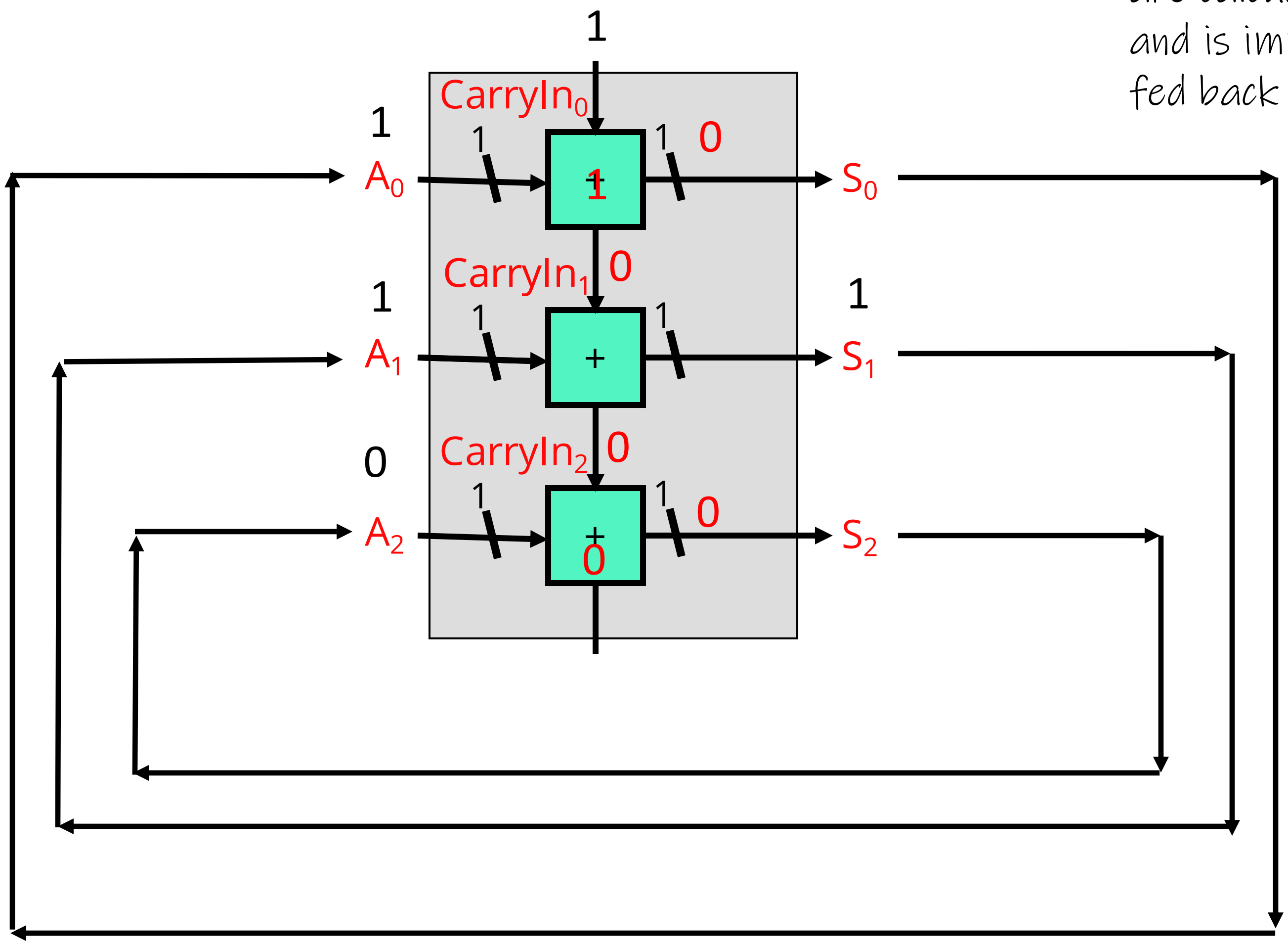
S_0 and CarryIn_1 are calculated first and is immediately fed back in



Combinational Counter

- If we interpret the counter circuit as physical hardware with gate delay:

S_0 and $CarryIn_1$ are calculated first and is immediately fed back in



Combinational Counter Review

- This example was just here to highlight why we need sequential circuits:
  - to be able to store state
  - to synchronize our circuits

- This lecture will be about setting up how we use gates to store state (data) and synchronize (time) signals.

Lecture Outline

- Sequential Setup
- **R-S Latch**
- D Latch & Clock
- D Flip Flops

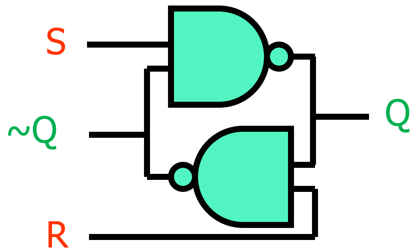
S-R Latch

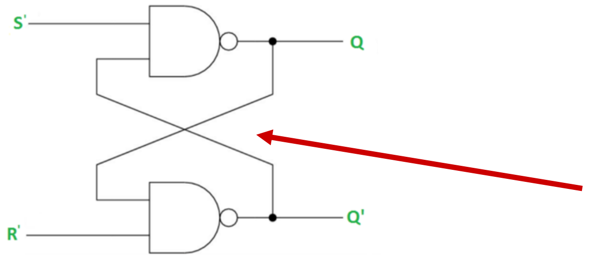
- Can store a bit value depending on its inputs
  - *called a "Latch" because it can "Latch" onto data coming in*
- Is a Bi-stable circuit: Can exist happily in two stable states

- You can push the latch from one state to another by **s**etting or **r**esetting it with S-R signals

S-R Latch Implementation

- Can be implemented by cross coupled NAND gates
  - Two inputs: S (*SET*) & R (*RESET*)
  - Two outputs: Q and NOT(Q)
    - Notation: NOT(Q)= ~Q = Q’ = $\overline{Q}$



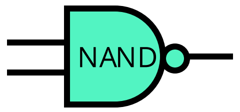


Another common way of drawing the same circuit

Outputs are fed back in!

Analyzing the Operation of a Latch

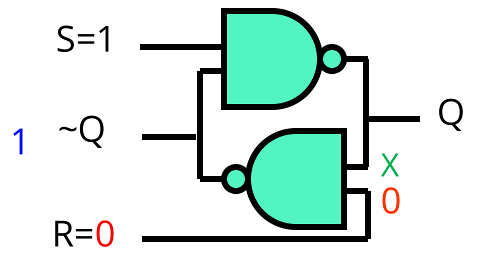
- First, recall truth table for a NAND gate:



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Zero IFF both are 1

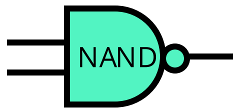
- R-S Latch Operation:
  - Best place to start is S=1, R=0



1st look at lower NAND gate
→ Its inputs are: 0 and X (unknown)
Because Q is unknown at 1st
→ Produces a 1 at its output
(0) NAND (ANYTHING) = 1

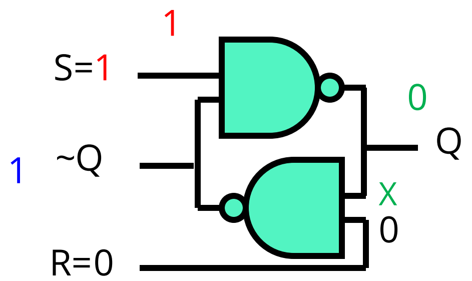
Analyzing the Operation of a Latch

- First, recall truth table for a NAND gate:



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

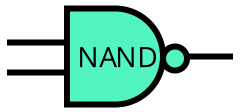
- R-S Latch Operation:
  - Best place to start is S=1, R=0



Next, look at top NAND gate
→ Its inputs are: 1 and 1
Blue 1, comes from lower NAND
→ Produces a 0 at its output
Therefore, when S=1, R=0
The output of latch is: Q=0, ~Q=1
ALWAYS!

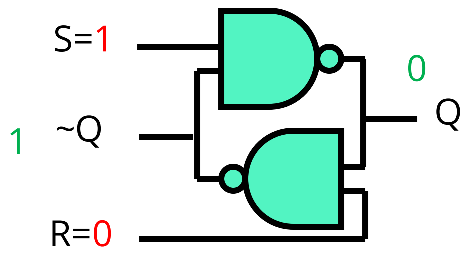
Analyzing the Operation of a Latch

- First, recall truth table for a NAND gate:



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

- R-S Latch Operation:



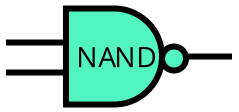
Truth Table for R-S Latch:

ACTION	S	R	Q	~Q
	0	0		
	0	1		
RESET	1	0	0	1
	1	1		

Called the "**RESET**" **action**, Q is "reset" to 0
Also, notice: Q and ~Q opposite

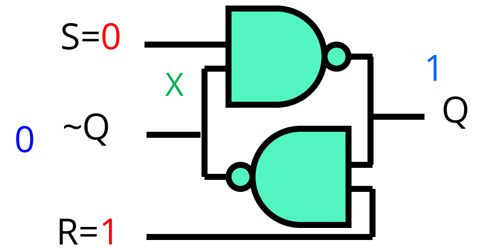
Analyzing the Operation of a Latch

- First, recall truth table for a NAND gate:



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

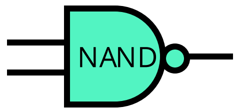
- R-S Latch Operation:
  - Next input case is called the "SET", when inputs are: S=0, R=1



1st look at upper NAND gate
→Its inputs are: 0 and X (anything)
→Produces a 1 at its output
Lower NAND gate
→Inputs are: 1 and 1
→Produces a 0 at its output

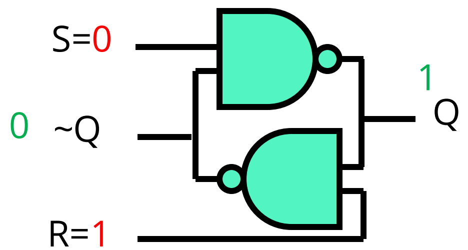
Analyzing the Operation of a Latch

- First, recall truth table for a NAND gate:



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

- R-S Latch Operation:



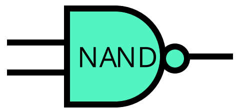
Truth Table for R-S Latch:

ACTION	S	R	Q	~Q
	0	0		
SET	0	1	1	0
RESET	1	0	0	1
	1	1		

***SET*s** LATCH to have a “1” at the output

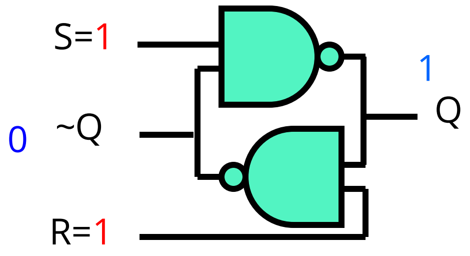
Analyzing the Operation of a Latch

- First, recall truth table for a NAND gate:



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

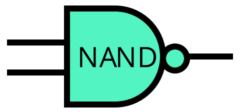
- R-S Latch Operation:
  - Last valid input case is the "HOLD" S=1, R=1
  - If we have just "SET" Latch, we will have Q=1, ~Q=0 already



Upper NAND gate
→ Has S=1 & former value of ~Q=0
→ Produces a 1 at its output
(same Q as when it started)
Lower NAND gate
→ Inputs are: 1 and 1
→ Produces a 0 at its output *(same ~Q)*

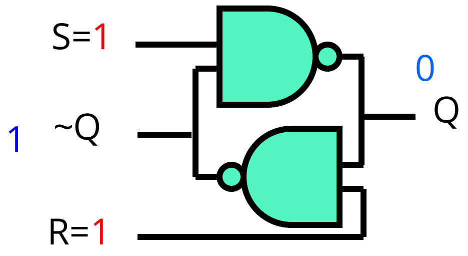
Analyzing the Operation of a Latch

- First, recall truth table for a NAND gate:



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

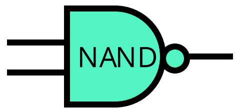
- R-S Latch Operation:
  - Last valid input case is the "HOLD" S=1, R=1
  - If we have just "RESET" Latch, we will have Q=0, ~Q=1 already



Upper NAND gate
→ Has S=1 & value of ~Q=1
→ Produces a 0 at its output
(same Q as when it started)
Lower NAND gate
→ Inputs are: 0 and 1
→ Produces a 1 at its output *(same ~Q)*

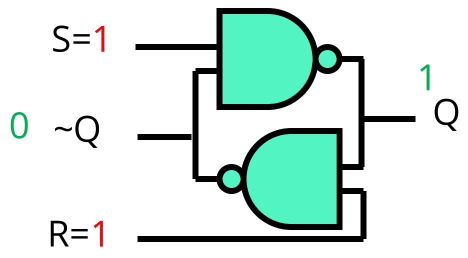
Analyzing the Operation of a Latch

- First, recall truth table for a NAND gate:



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

- R-S Latch Operation:



Truth Table for R-S Latch:

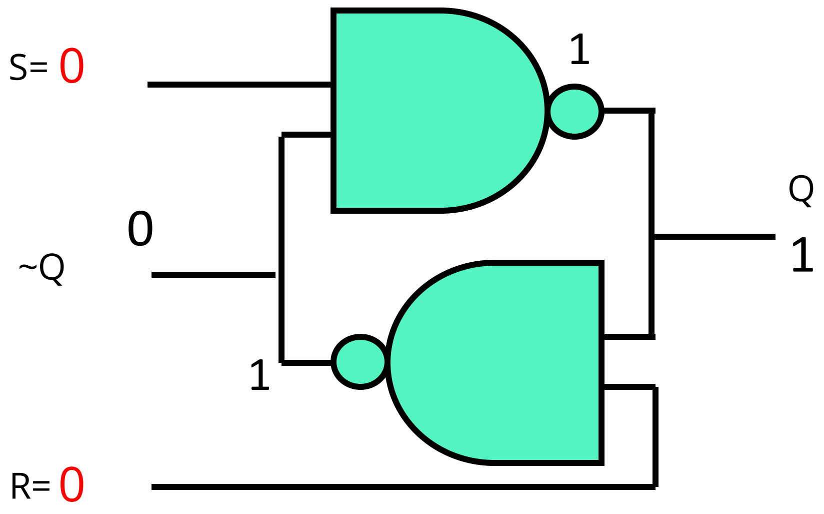
ACTION	S	R	Q	~Q
	0	0		
SET	0	1	1	0
RESET	1	0	0	1
HOLD	1	1	1	0
HOLD	1	1	0	1

THIS HOLD's the last value on its outputs!

Analyzing the Operation of a Latch

- What happens with S=0 and R=0?
  - Short answer: confusion

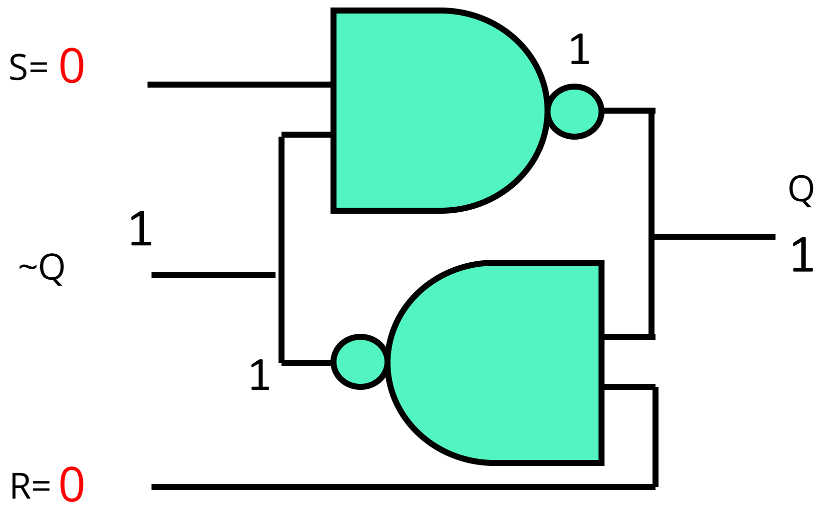
Q = 1 and ~Q = 0 are set before this



Analyzing the Operation of a Latch

- What happens with S=0 and R=0?
  - Short answer: confusion

Q = 1 and ~Q = 0 are set before this

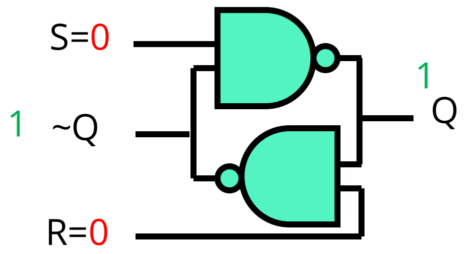


Analyzing the Operation of a Latch

- What happens with S=0 and R=0?
  - Short answer: confusion
  - Real circuits depend on both Q and ~Q
  - Strange things may happen if both are 1

Truth Table for R-S Latch:

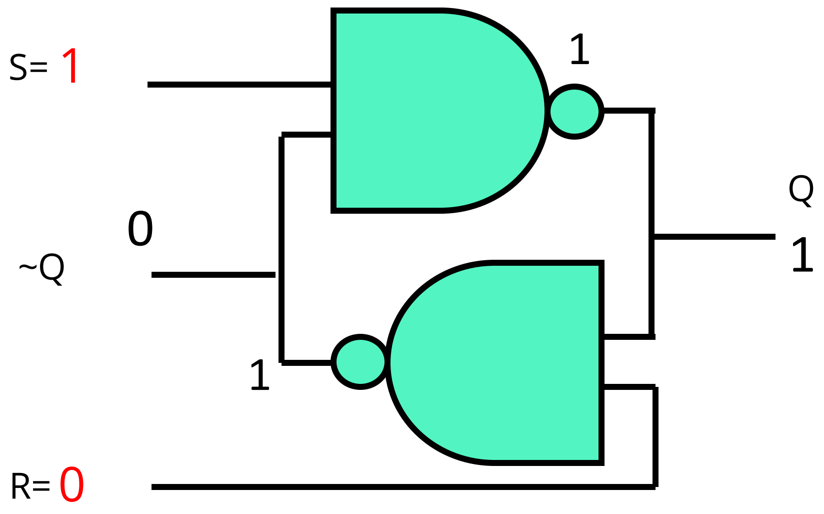
ACTION	S	R	Q	~Q
ILLEGAL	0	0	1	1
SET	0	1	1	0
RESET	1	0	0	1
HOLD	1	1	1	0
HOLD	1	1	0	1



Analyzing the Operation of a Latch

- What "really" happens with S=1 and R=0?
  - Let's see.

Q = 1 and ~Q = 0 are set before this



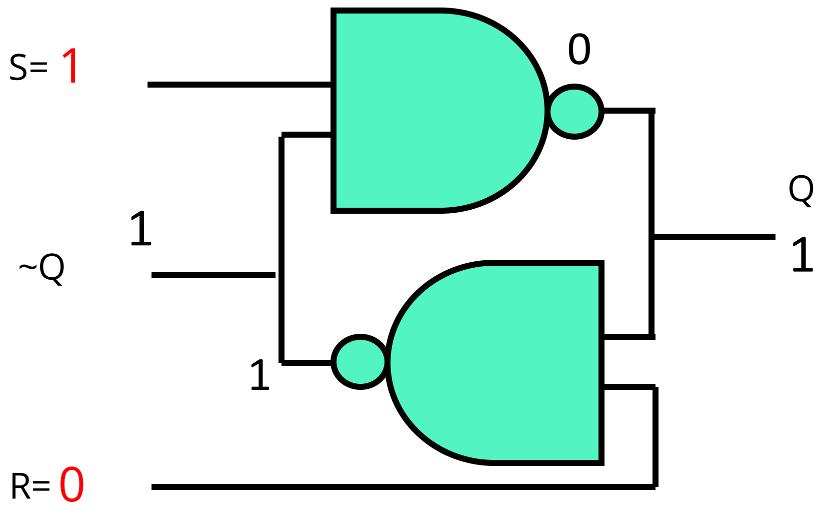
Wait? There both 1?
I thought that only happened in
the illegal state?

Well, let's continue to hold
the signal for another round.

Analyzing the Operation of a Latch

- What happens with S=1 and R=0?
  - Let's see.

Q = 1 and ~Q = 0 are set before this



Ahh, so it 'stabilizes' into the Reset State!

Yup, that's right. 💯

For now, we'll ignore this when we Set or Reset…

Just want to show you how finicky these really are.

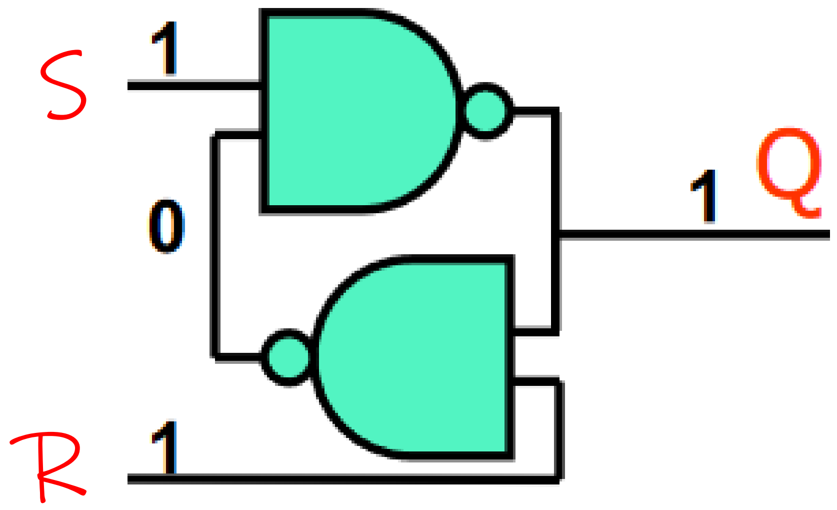
Lecture Outline

- Sequential Setup
- R-S Latch
- **D Latch & Clock**
- D Flip Flops

D Latch

- Goal: Make the RS latch more understandable

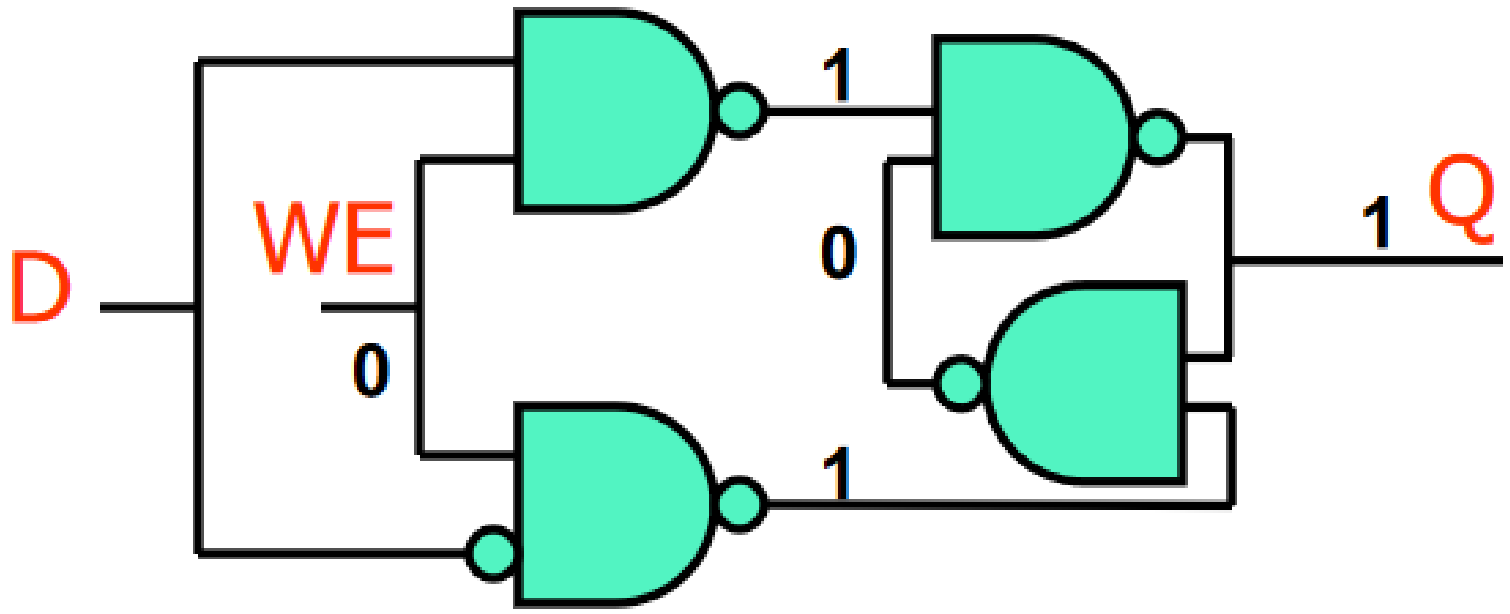
RS LATCH



D Latch

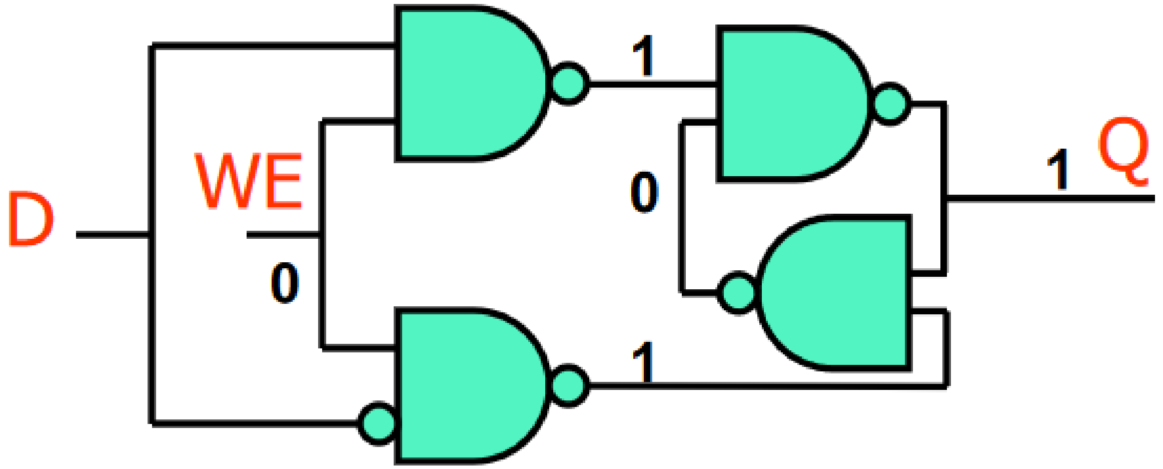
- Goal: Make the RS latch more understandable
  - To make it more understandable, we'll make it more complicated!

D LATCH



D Latch

- Goal: Make the RS latch more understandable
  - Replace R and S with D, add WE for utility
- D the data we want to store (either 1 or 0)
- WE, whether writing (storing that bit) is enabled.
  - If not enabled, Q (the stored data) maintains current value
  - If enabled, then Q is set to be D
  - Impossible for S=0 and R = 0 case to occur



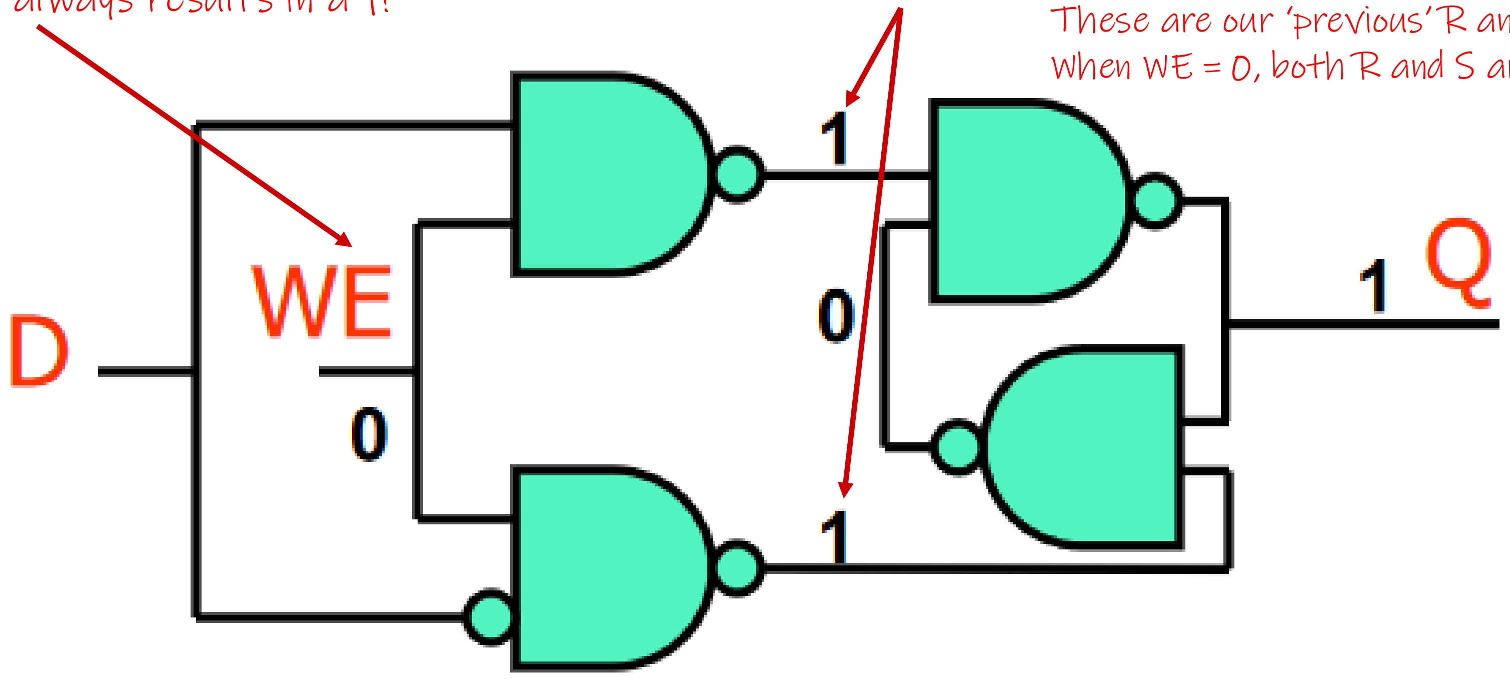
D Latch

- WE set to 0, D is unknown

0 input to a NAND gate always results in a 1!

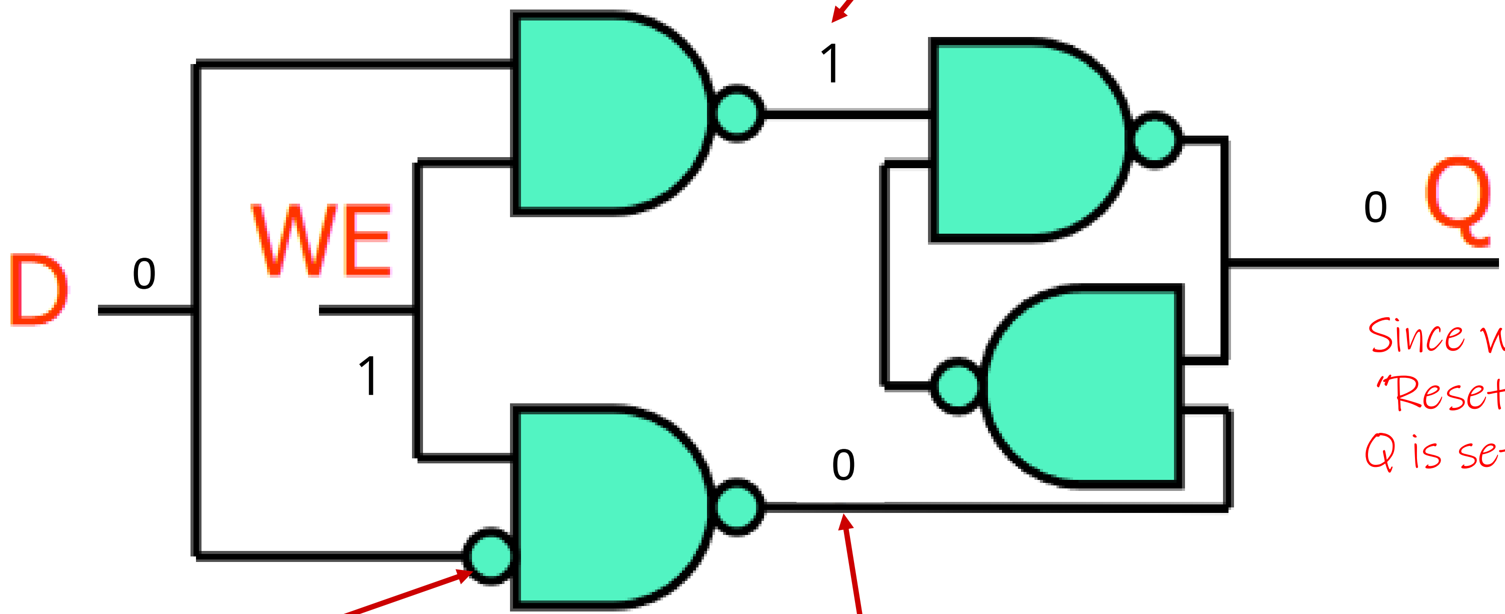
Outputs here are 1!

These are our 'previous' R and S inputs; when WE = 0, both R and S are 1, so we HOLD



D Latch

- WE set to 1, D is 0



This is our previous "Set"

Output here is 1

Since we are "Resetting" Q is set to 0

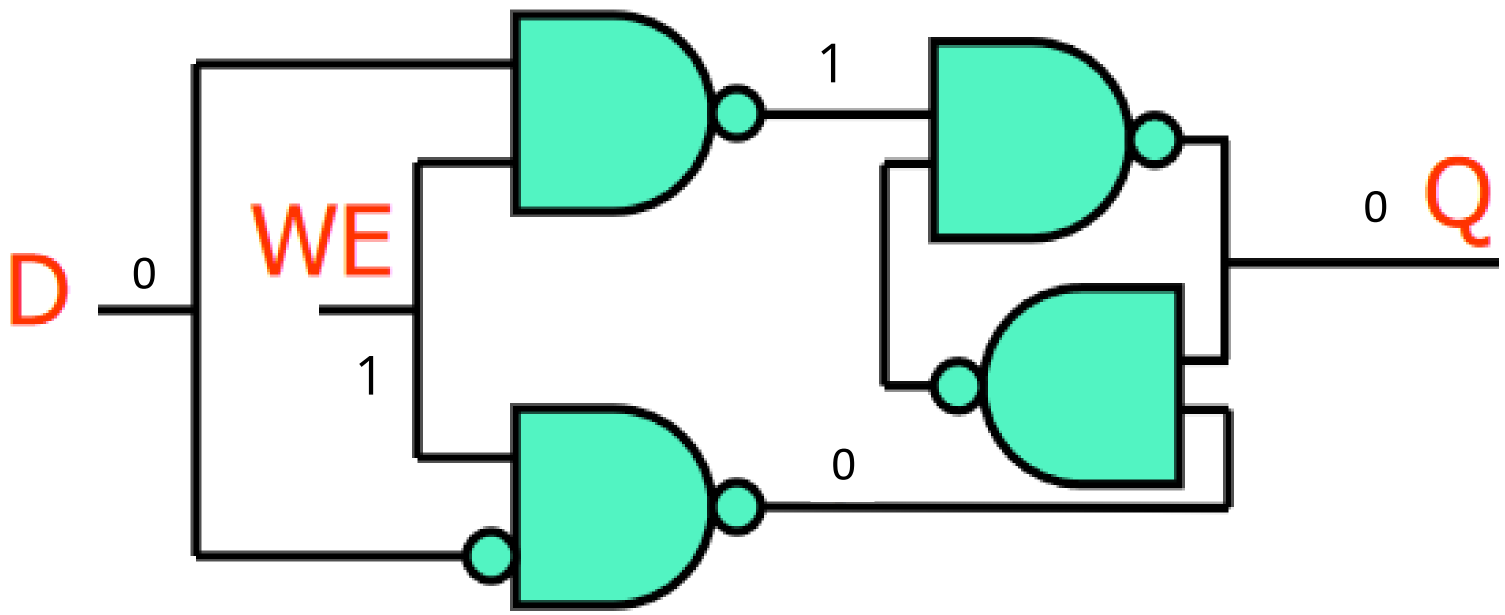
This negates the input!
This gate has input **WE , ~D**

Output here is 0

This is our previous "Reset"

D Latch

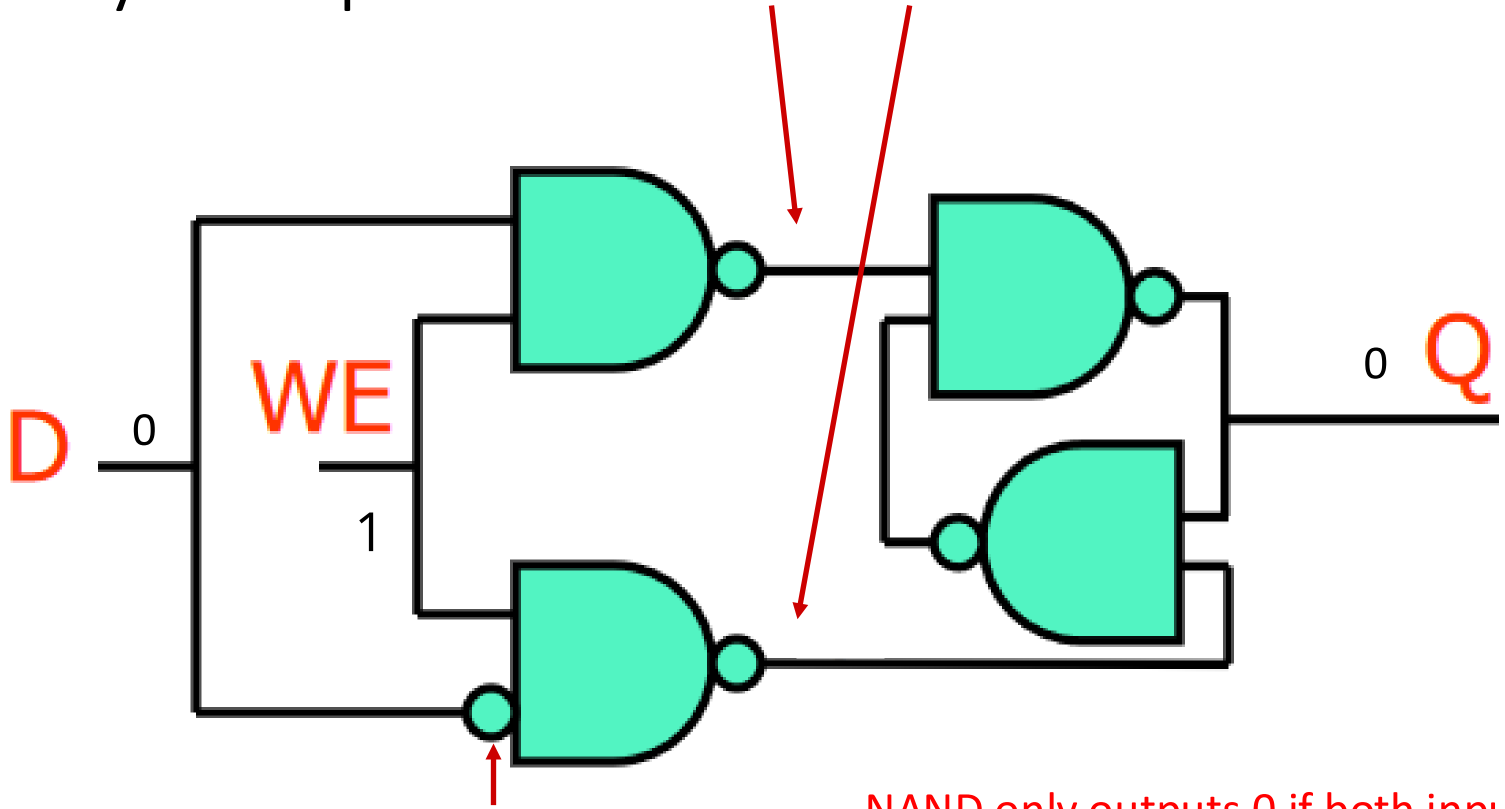
- WE set to 1, D is 0



In D Latch terms, ***Q is equal to the value of D when WE is 1***

D Latch

- Why is it impossible for S = 0 & R = 0?

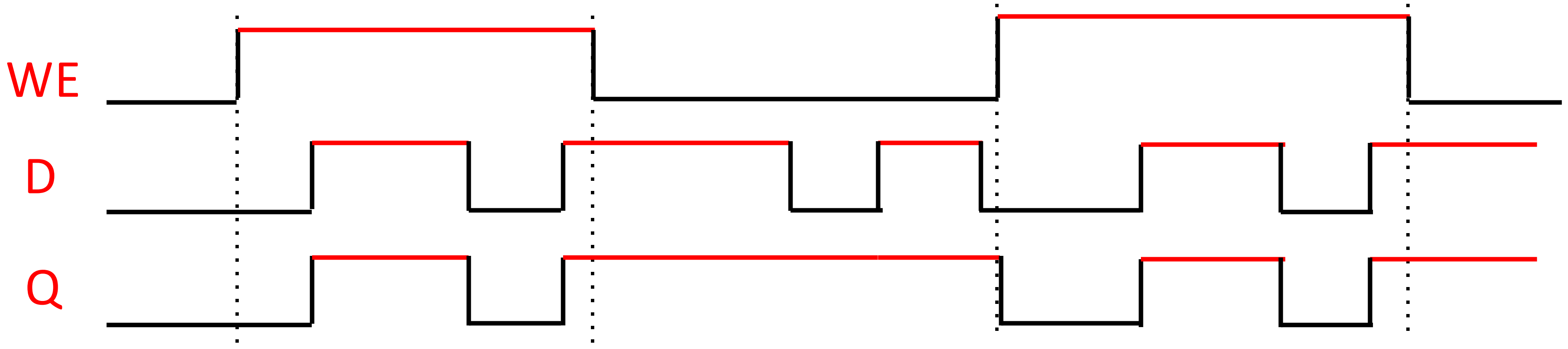


This negation guarantees
S = 0 and R = 0 is impossible

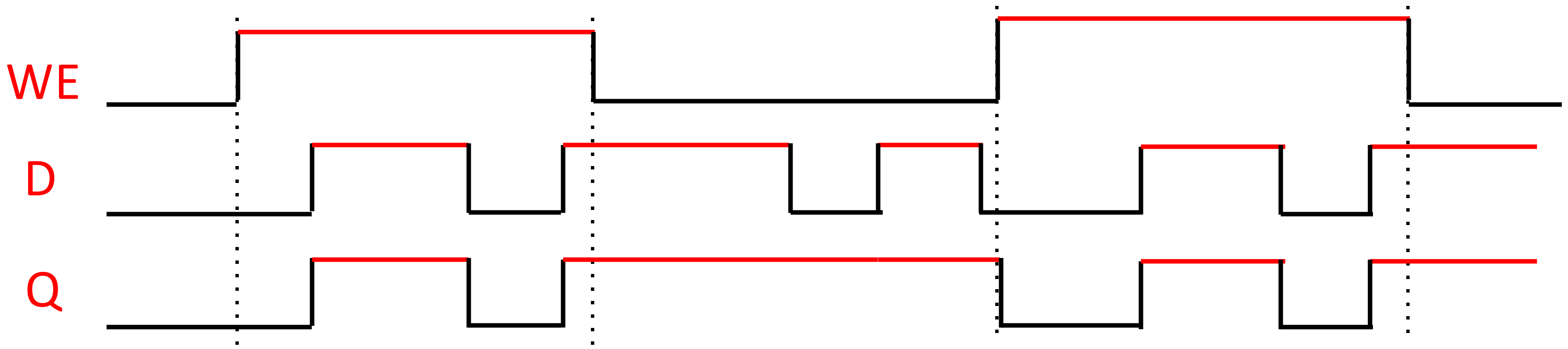
NAND only outputs 0 if both inputs are 1;
the negation makes this impossible.
One will receive D and the other ~D.

Timing Diagrams

- Diagram to represent how signals change over time
- WE: Write Enable Signal
  - WE is **high**: the latch is **open** and Q is the same as D.
  - WE is **low**: the latch is **closed** and Q stays the same. “Holds”
    - The input signal should be stable a certain amount of time before the WE signal is lowered for proper operation. This is referred to as the **setup time**.



Timing Diagrams



Poll Everywhere

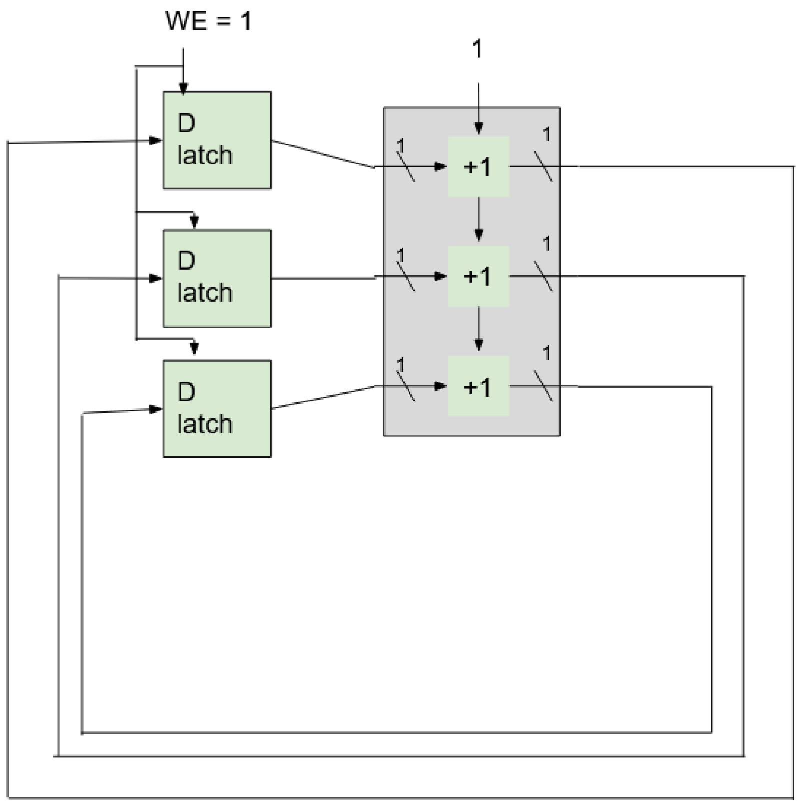
pollev.com/cis2400

- Does the following counter circuit "work" better than the previous counter circuit? Assume WE is hard-coded as 1.

A. Yes

B. No

C. I'm not sure



Poll Everywhere

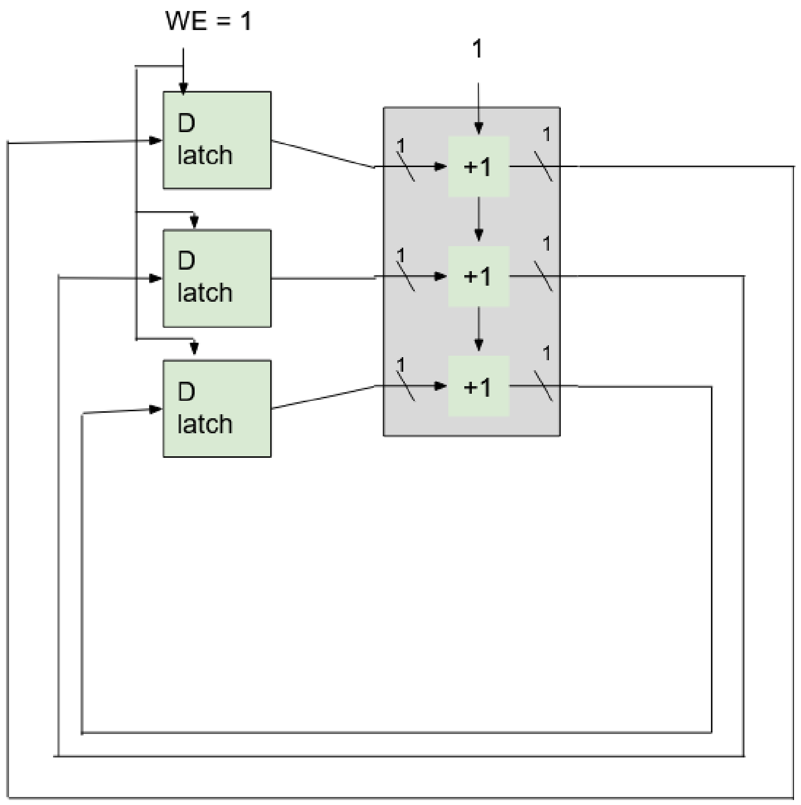
pollev.com/cis2400

- Does the following counter circuit "work" better than the previous counter circuit? Assume WE is hard-coded as 1.

A. Yes

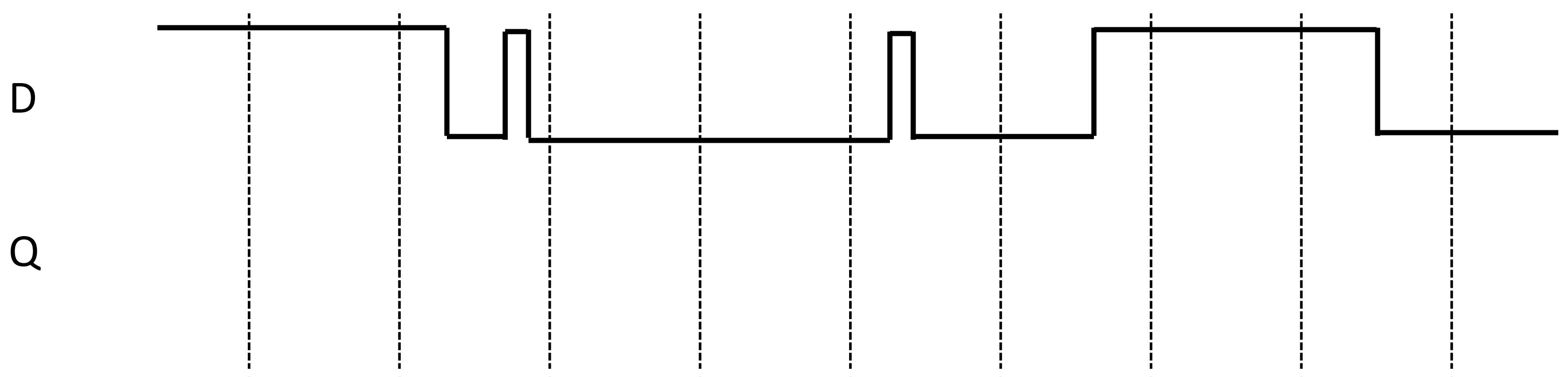
[✓ B. No]

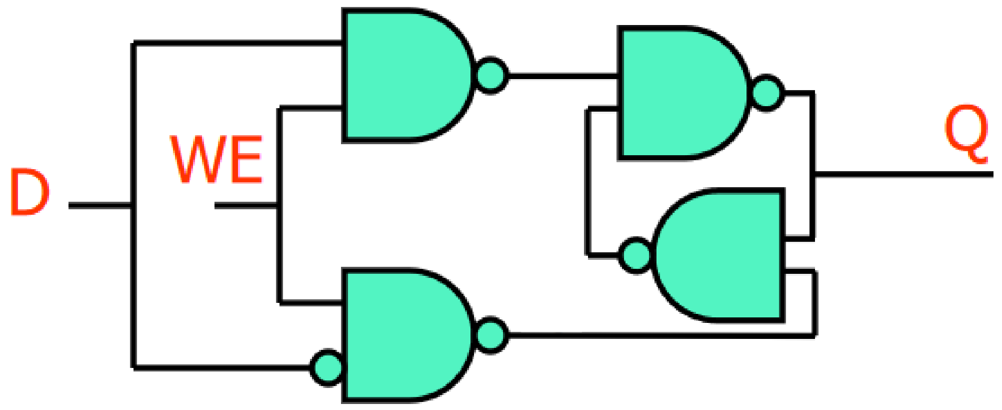
C. I'm not sure



Transparency

- Consider the following signal. What is the signal output (Q) of our D Latch?
  - Assume that WE is 1

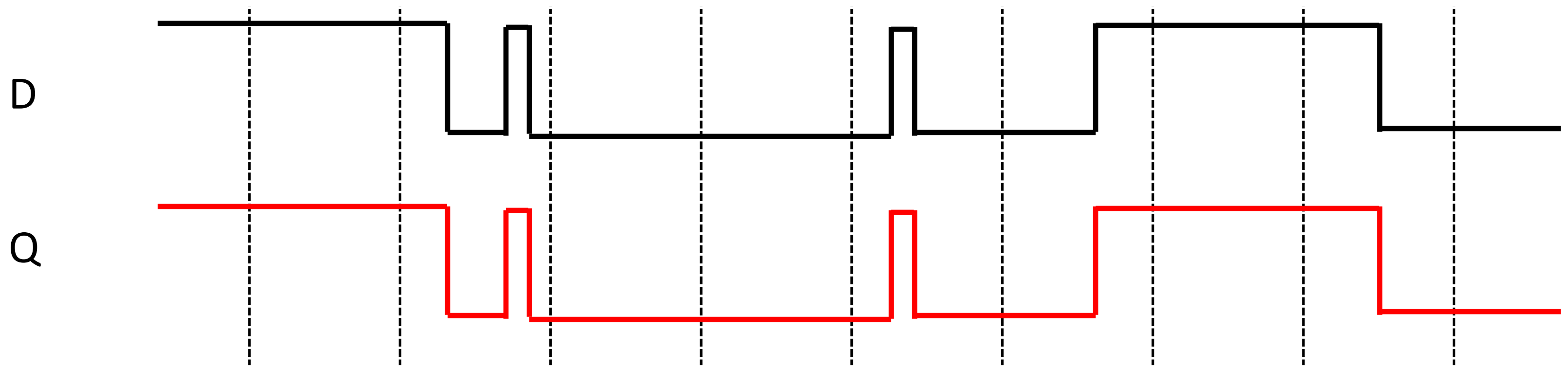


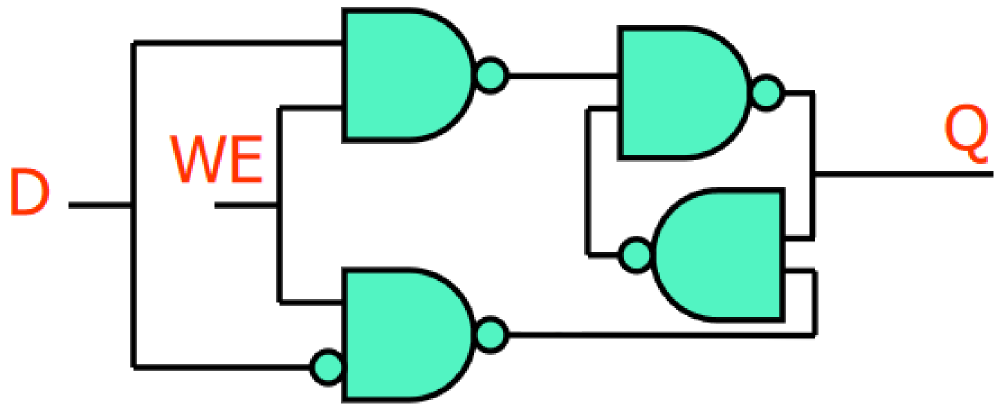


Transparency

- Consider the following signal. What is the signal output (Q) of our D Latch?
  - Assume that WE is 1

We get all the signal all the time
i.e. Q = D, no "Synchronization"



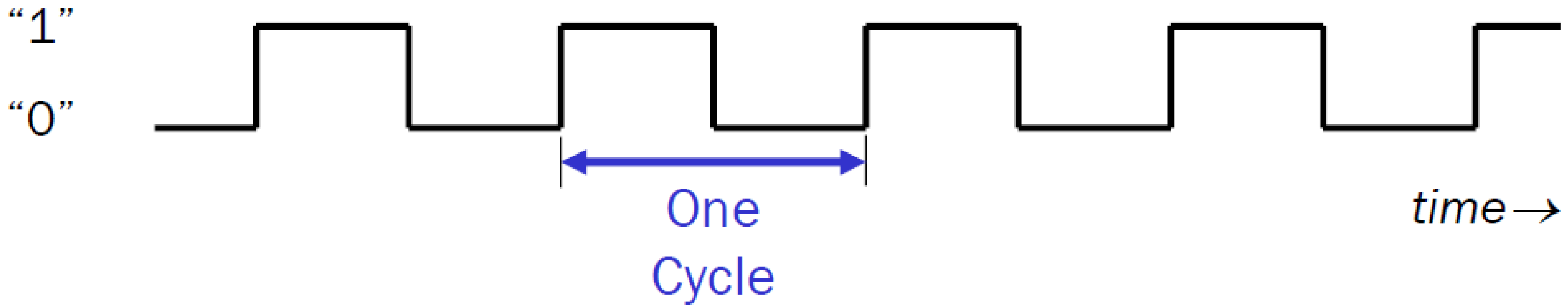


We can call this "Transparent"

The Clock

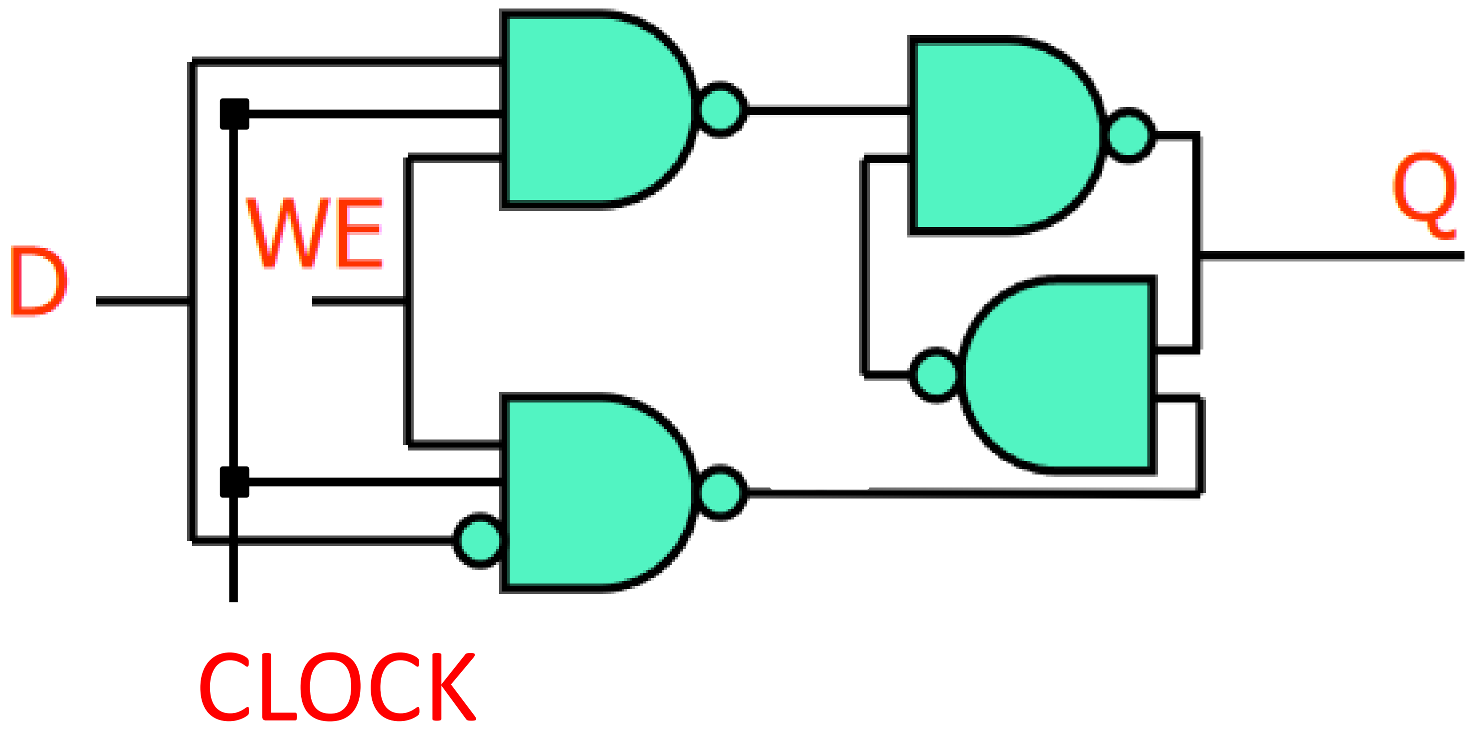
- A regular up & down signal which can be used for timing & synchronization.
  - Is the "heartbeat" of our system.
  - Sort of like a metronome

- Clock Period = Duration of one clock cycle
- Clock Frequency = 1/Period
  - Typical frequency: 2.5GHz = 2.5e9 Hz
  - Typical period: 0.4 nanoseconds



D Latch with Clock

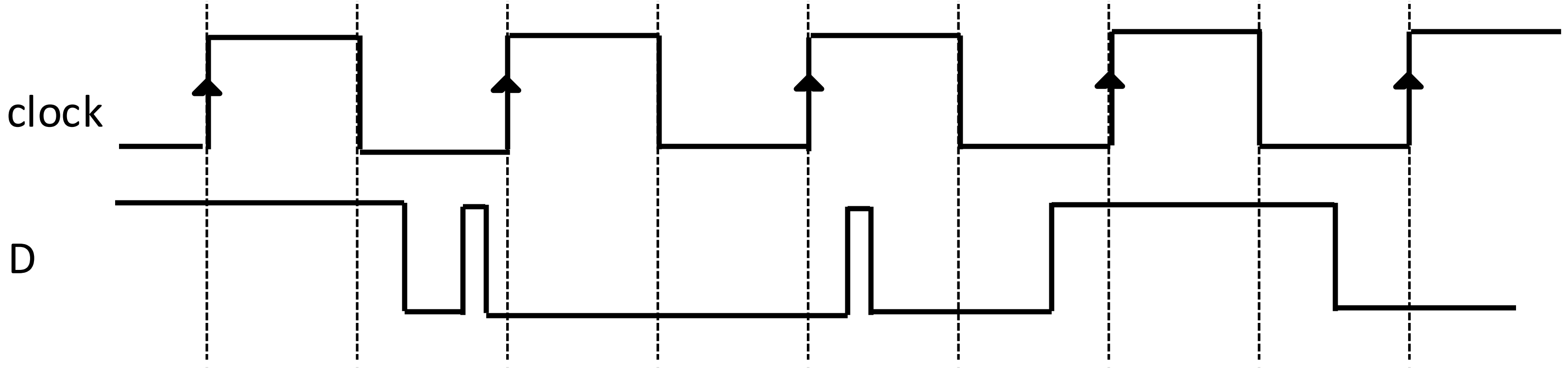
- Clock could be included into our D-Latch
  - This affects out transparency (see next slide)

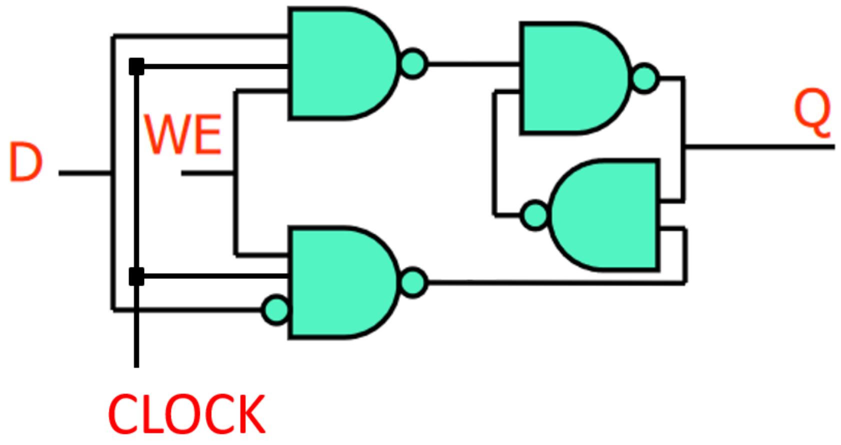


Transparency?

- Consider the following signal. What is the signal output (Q) of our D Latch?
  - Assume that WE is 1

Q only changes when clock is high

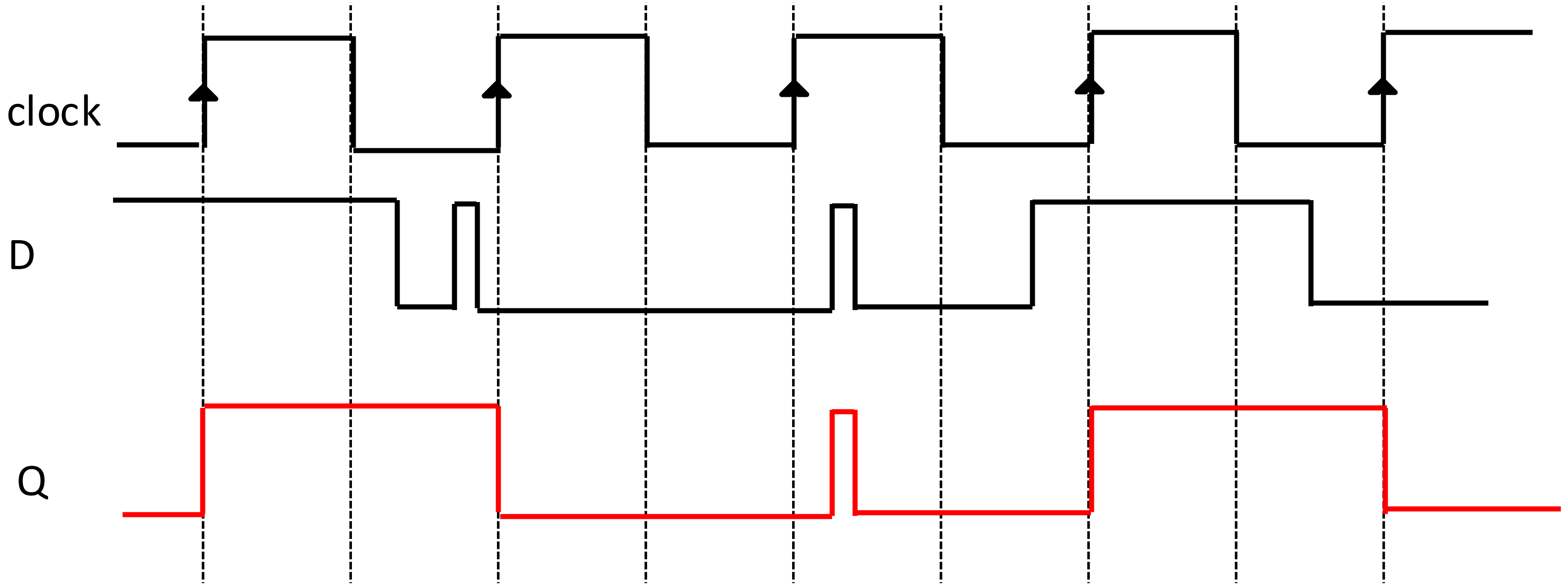




Semi-Transparency

- Consider the following signal. What is the signal output (Q) of our D Latch?
  - Assume that WE is 1

Q only changes when clock is high

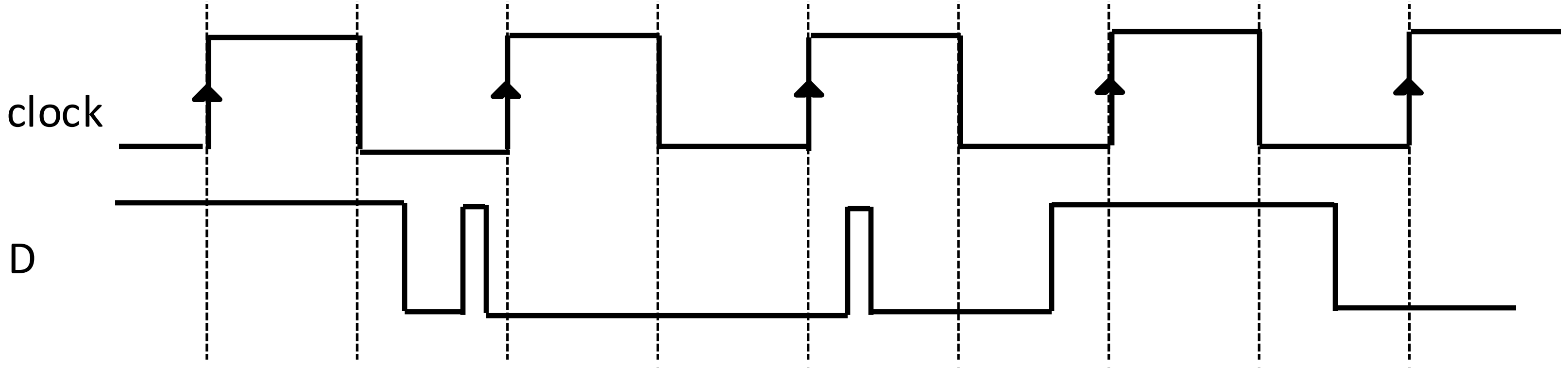


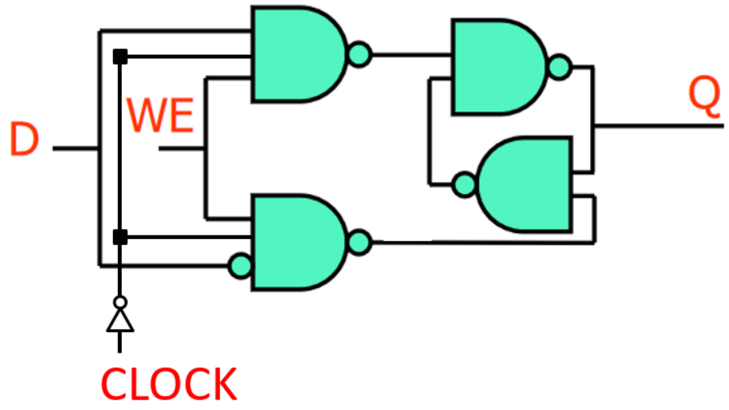
This is called
"Transparent-High"

Semi-Transparency?

- Consider the following signal. What is the signal output (Q) of our D Latch? (Note how the clock is inverted)
  - Assume that WE is 1

Q only changes when clock is low

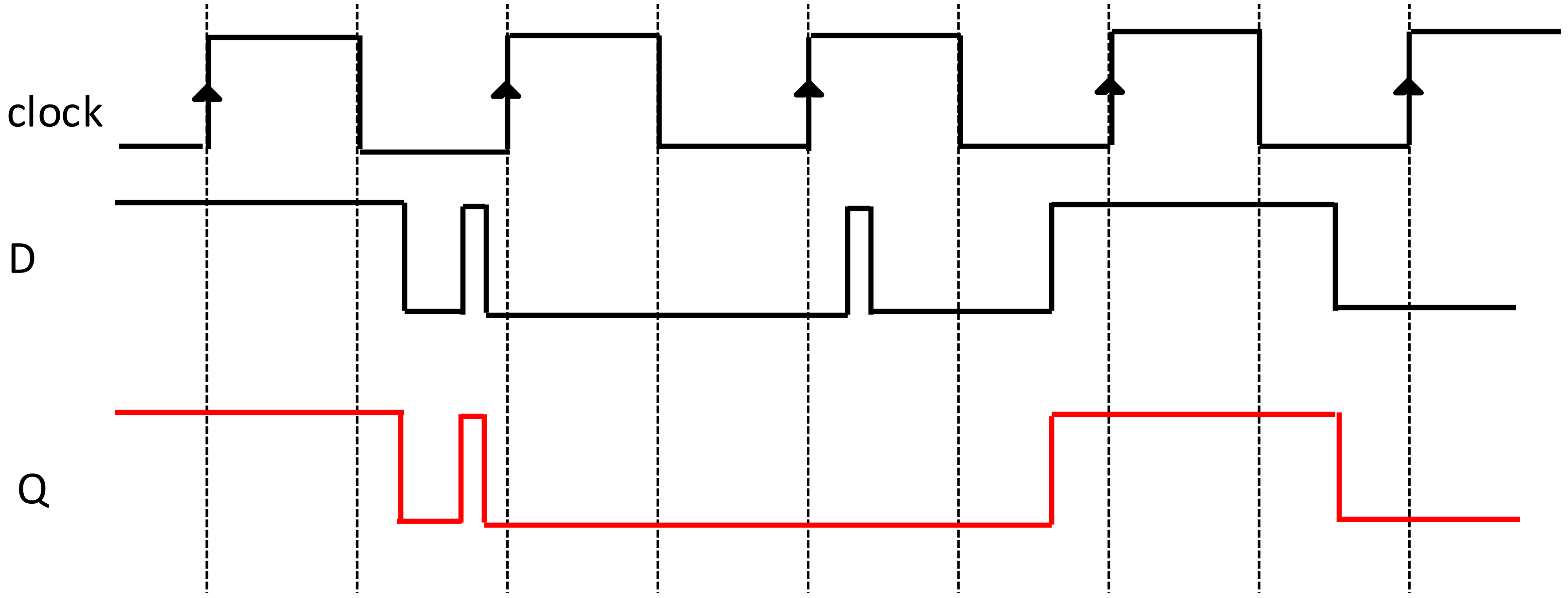




Semi-Transparency?

- Consider the following signal. What is the signal output (Q) of our D Latch?
  - Assume that WE is 1

Q only changes when clock is low



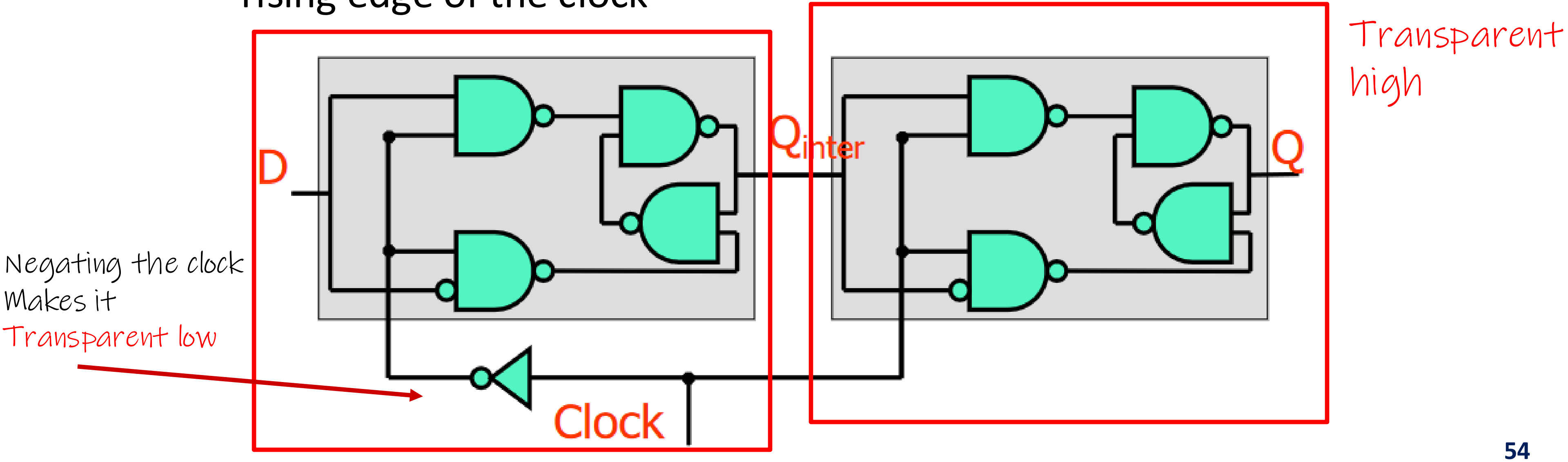
This is called "Transparent-low"

Lecture Outline

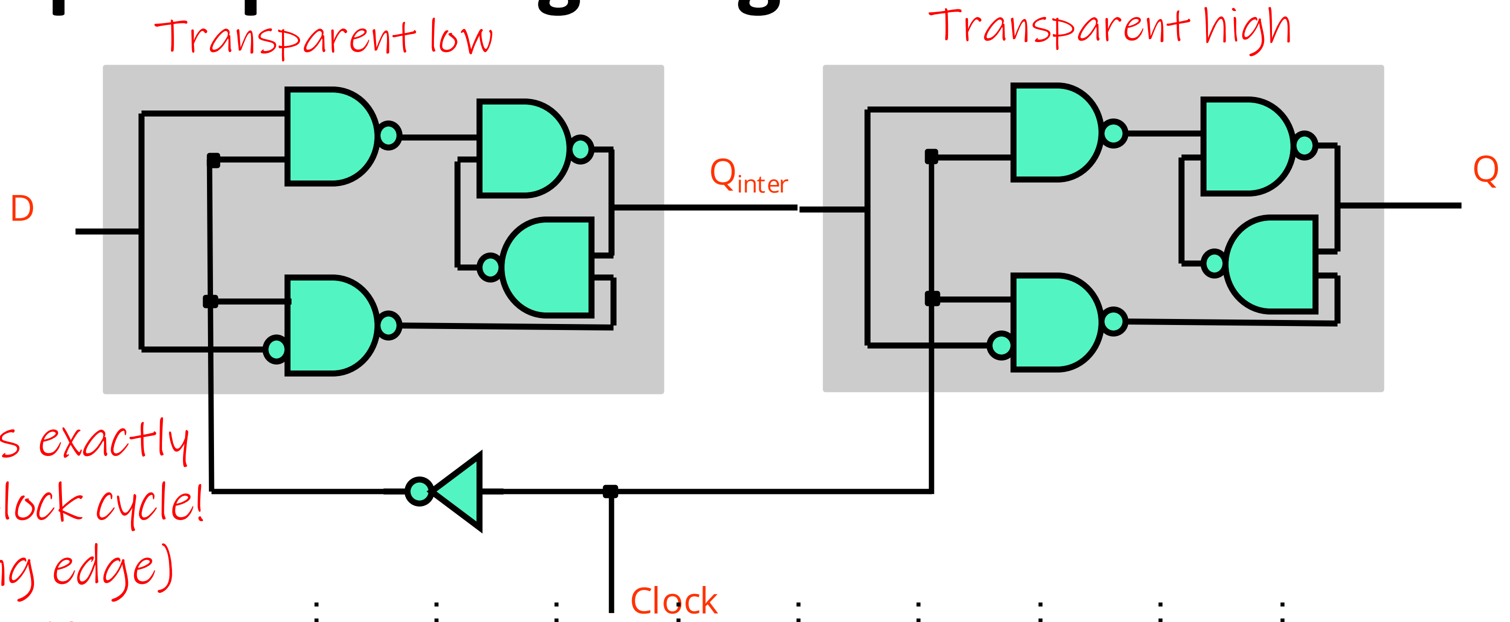
- Sequential Setup
- R-S Latch
- D Latch & Clock
- **D Flip Flops**

D Flip Flop

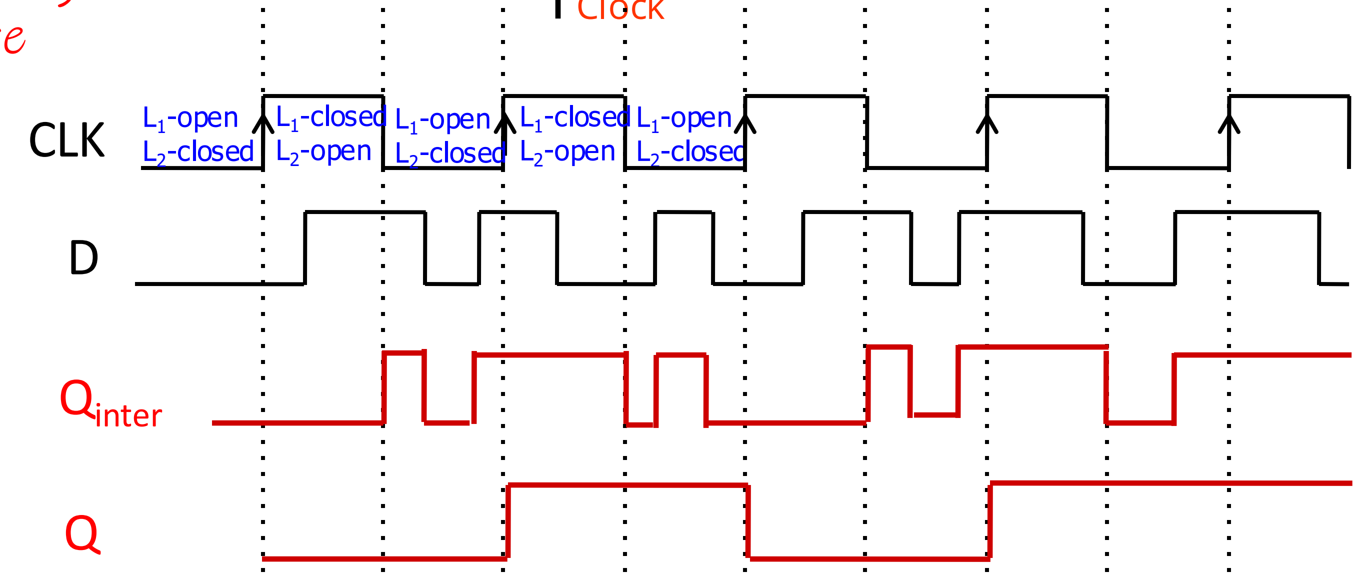
- Made by:
  - Appending a transparent-high onto a transparent-low latch
- Rules:
  - Q_{inter} is the result of passing D through a transparent-low latch
  - Q is the result of passing Q_{inter} through a transparent-high latch
    - Effectively, this makes Q only "sensitive" to the signal value at the rising edge of the clock



D Flip Flop Timing Diagram

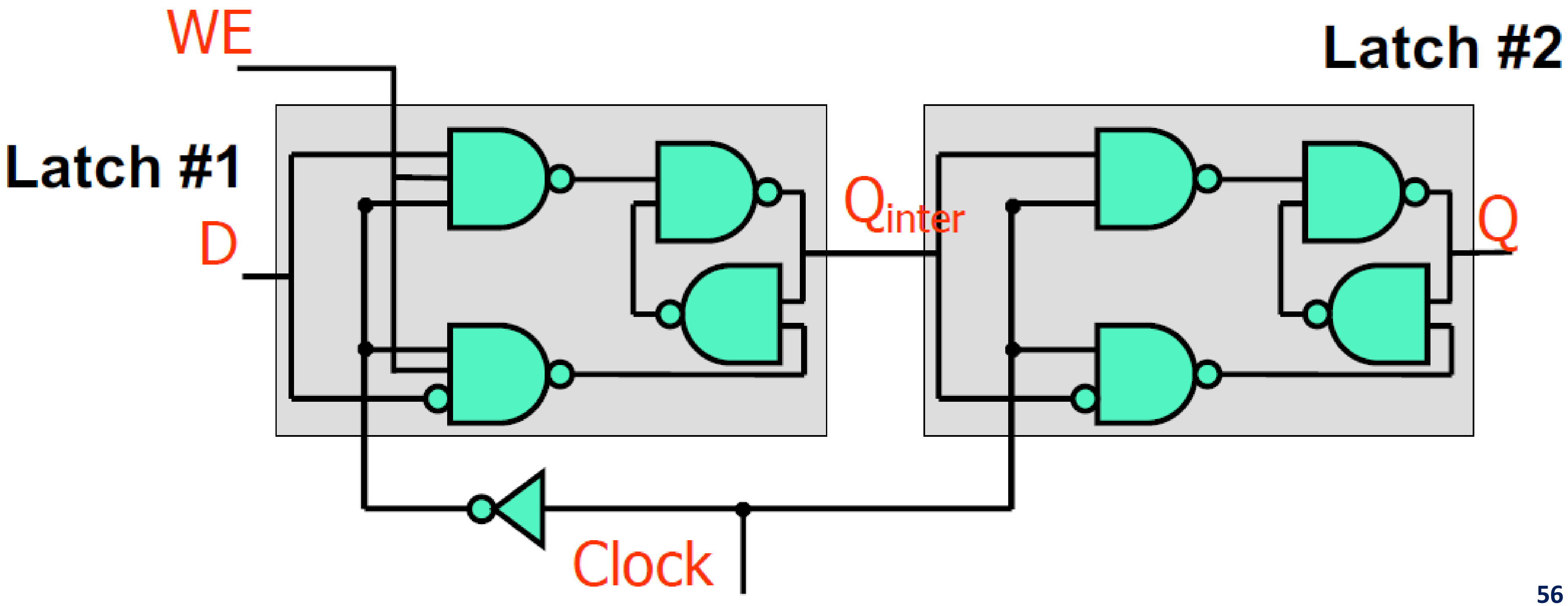


Q updates exactly once per clock cycle! (on a rising edge)
Value is more "stable"



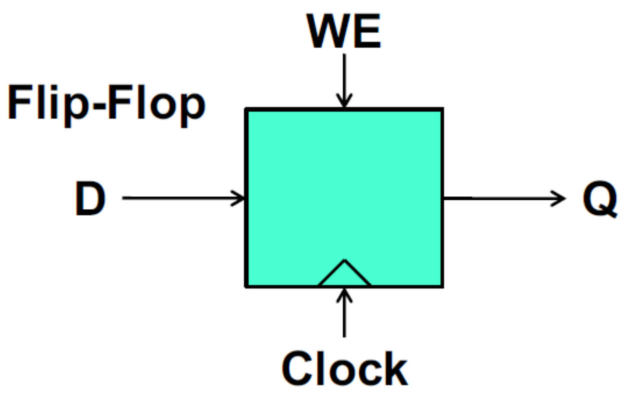
Flip Flop with WE

- Can attach WE to the first latch to enable WE for the flip-flop.
- When WE is low, latch #1 is closed and Q_{inter} cannot change. Q will become Q_{inter} if not already.



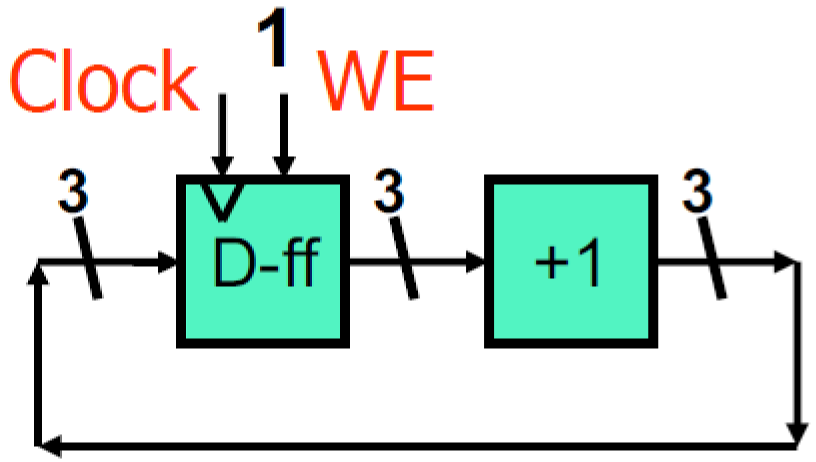
Flip Flops Summary

- We can abstract away the details of a Flip Flop as a 1-bit storage container.
  - Takes in an input D
  - Has an output or stored value "Q“
  - Takes a clock input (often represented with a triangle)
  - Usually has a WE to control if it will update on the next rising edge
  - A set of D flip flops can be grouped together to form a register (storage for a multiple-bit value, more on this next lecture)



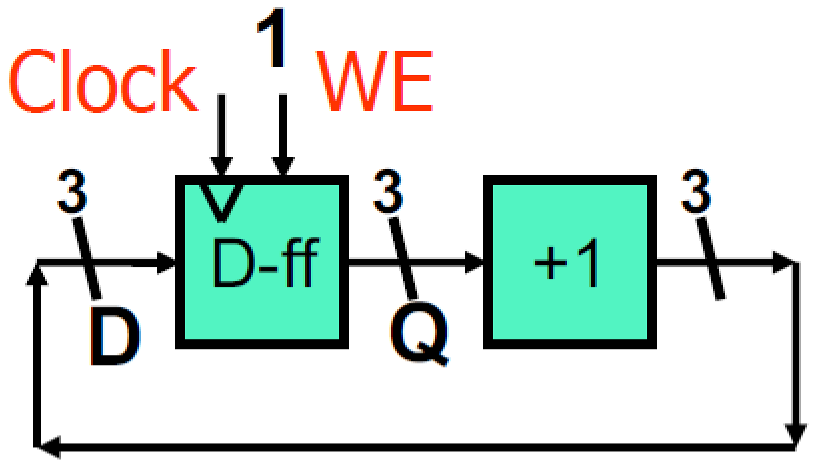
Working Counter

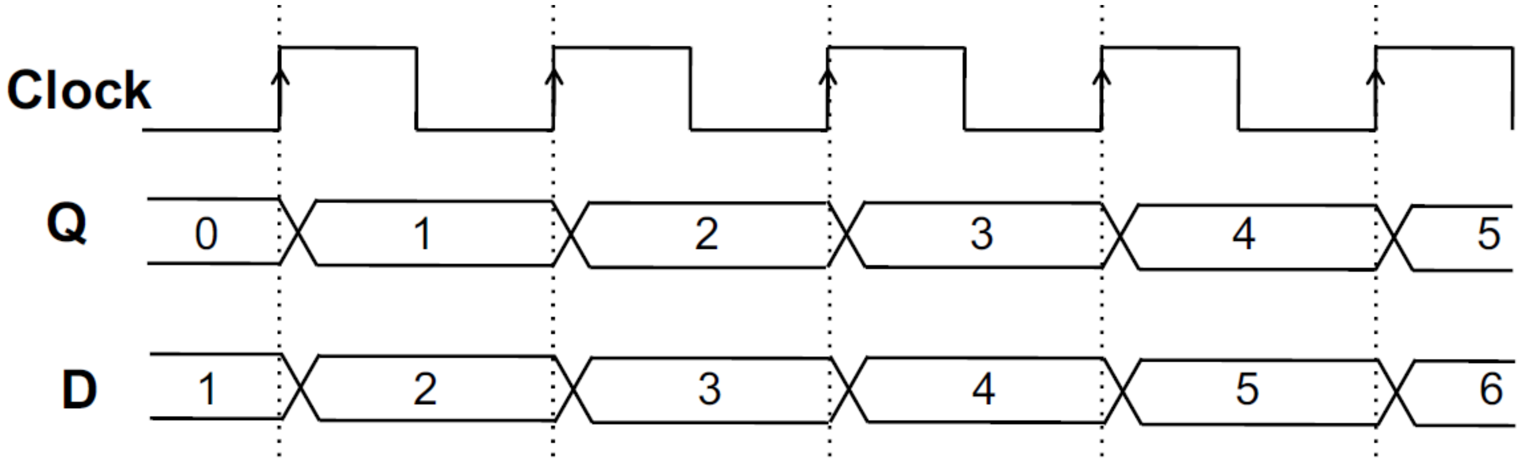
- Use a clocked 3-bit register (storage) made of D flip-flops



Counter Timing Diagram

- Incrementor computes input +1, the next value of the register





Next Lecture

- Memory
  - What really is it and how is it accessed?
- Registers
- More Clock!