Signed subtraction

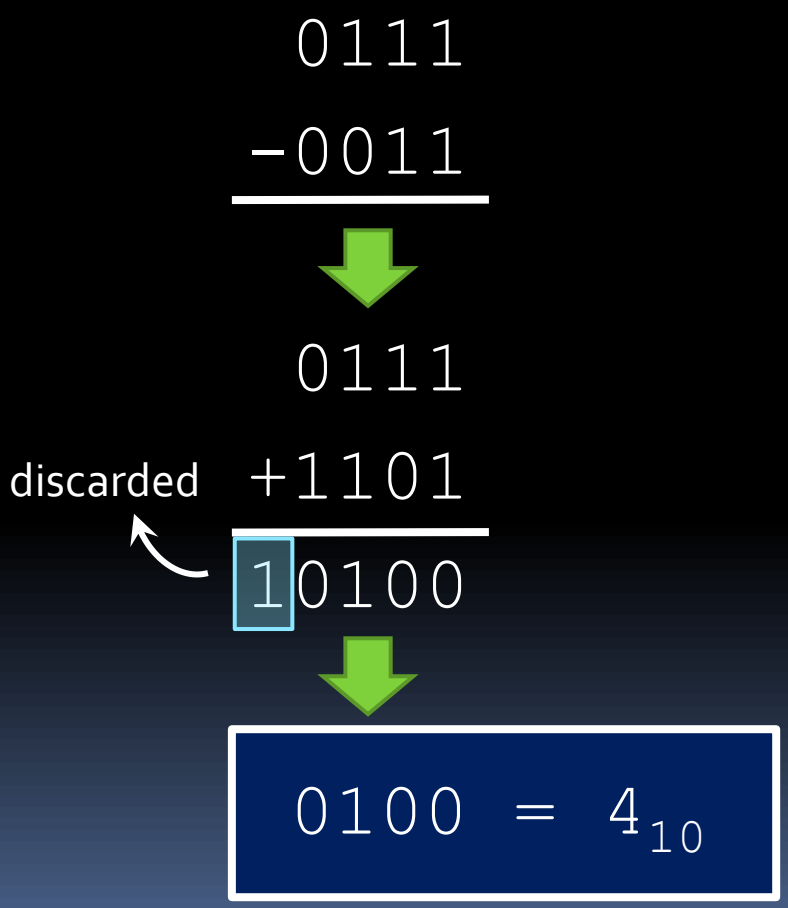
- Negative numbers are generally stored in 2's complement notation.
  - Reminder: 1's complement → bits are the bitwise NOT of the equivalent positive value.
  - 2's complement → 1's complement value plus one; results in zero when added to equivalent positive value.
- Subtraction can then be performed by using the binary adder circuit with negative numbers.

At the core of subtraction

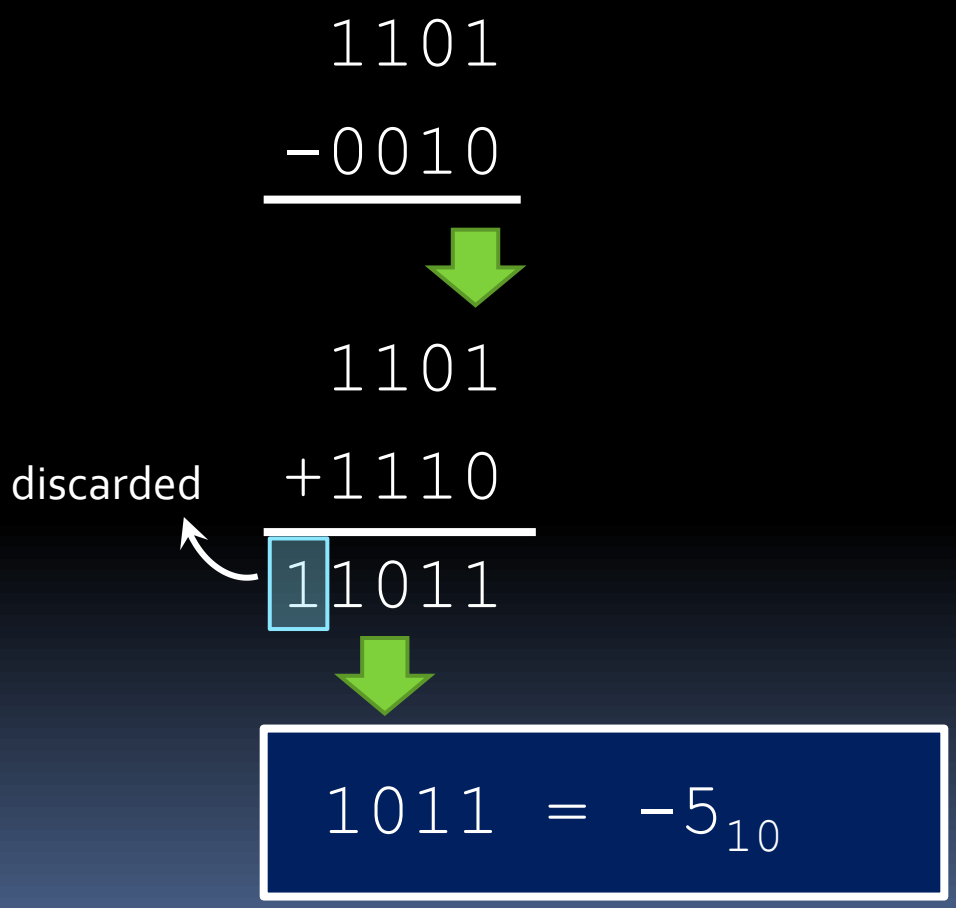
- Subtraction of a number is simply the addition of its negative value.
- This the negative value is found using the 2's complement process.
  - $7 - 3 = 7 + (-3)$
  - $-3 - 2 = -3 + (-2)$

Signed Subtraction example

- 7 − 3



- -3 − 2



What about bigger numbers

- $53 - 27$

```
  00110101
 -00011011
```

```
  00110101
 +11100101
```

discarded

```
 100011010
```

$00011010 = 26_{10}$

- $27 - 53$

```
  00011011
 -00110101
```

```
  00011011
 +11001011
```

discarded

```
 011100110
```

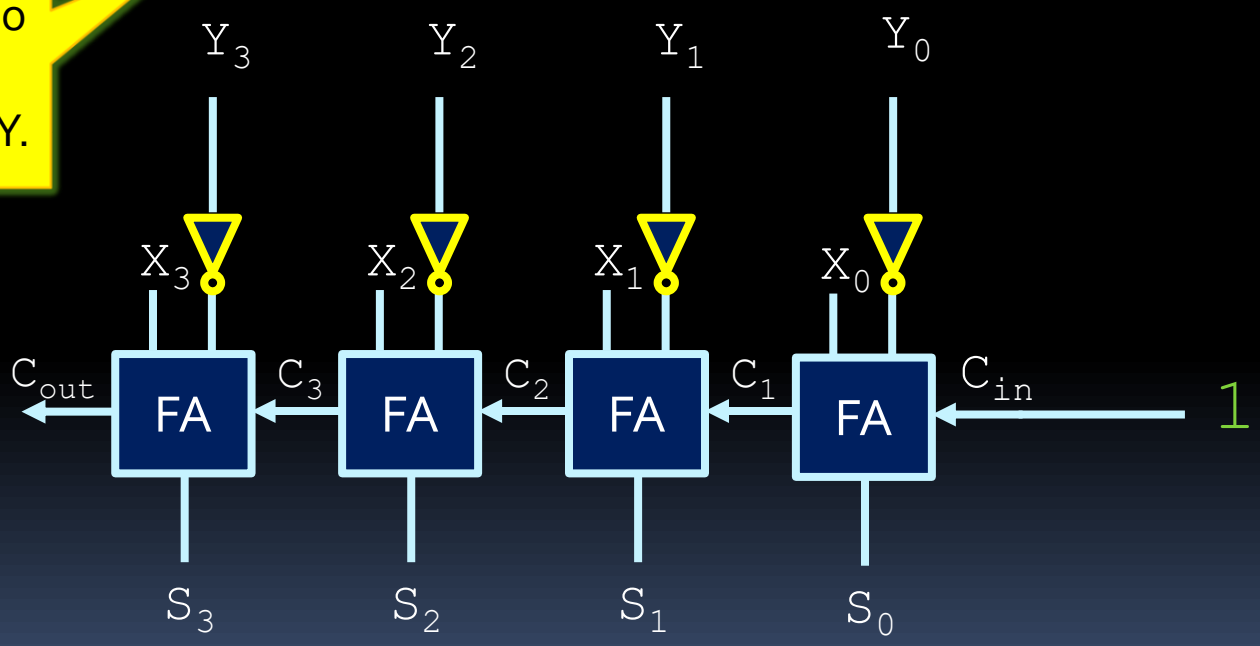
$11100110 = -26_{10}$

Subtraction circuit

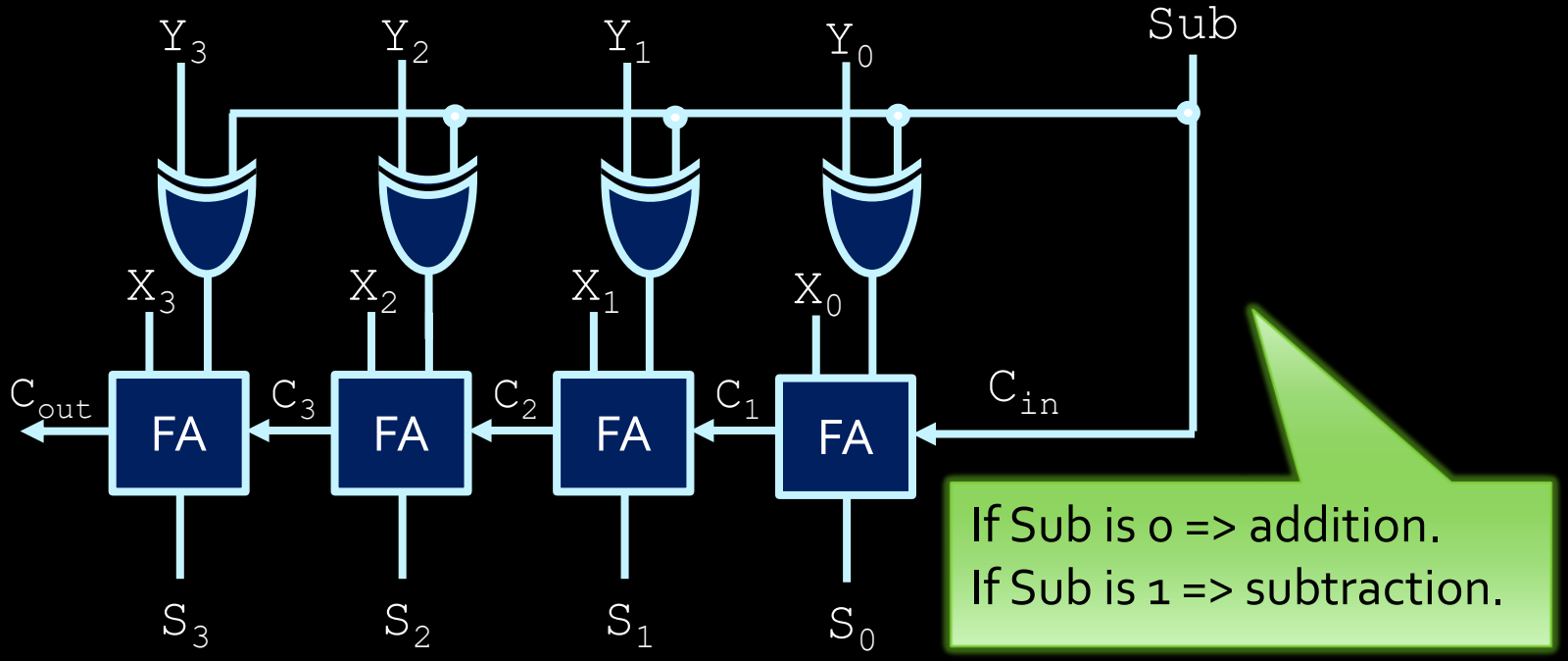
- 4-bit subtractor: X – Y
  - ➔ X plus 2's complement of Y
  - ➔ X plus 1's complement of Y plus 1

Feed 1 as Carry-In in the least significant FA.

Use NOT gates to get the 1's complement of Y.



Addition/Subtraction circuit



- The full adder circuit can be expanded to incorporate the subtraction operation
  - Remember: 2's complement = 1's complement + 1
  - We connect Sub to Cin

Food for Thought

- What happens if we add these two positive signed binary numbers `0110` + `0011` (i.e., `6` + `3`)?
  - The result is `1001`.
  - But that is a negative number (`-7`)! ☹
- What happens if we add the two negative numbers `1000` + `1111` (i.e., `-8` + (`-1`))?
  - The result is `0111` with a carry-out. ☹
- We need to know when the result might be wrong.
  - This is usually indicated in hardware by the Overflow flag!
  - More about this when we'll talk about processors.

Subtracting unsigned numbers

- General algorithm for X - Y:
  (for sign-and-magnitude representation)
  1. Get the 2's complement of the subtrahend Y (the term being subtracted).
  2. Add that value to the minuend X (the term being subtracted from).
  3. If there is an end carry (C_{out} is high), the final result is positive and does not change
     (set sign bit of output to 0).
  4. If there is no end carry (C_{out} is low), get the 2's complement of the result and set the sign bit of output to 1.

Unsigned subtraction example

- 53 – 27

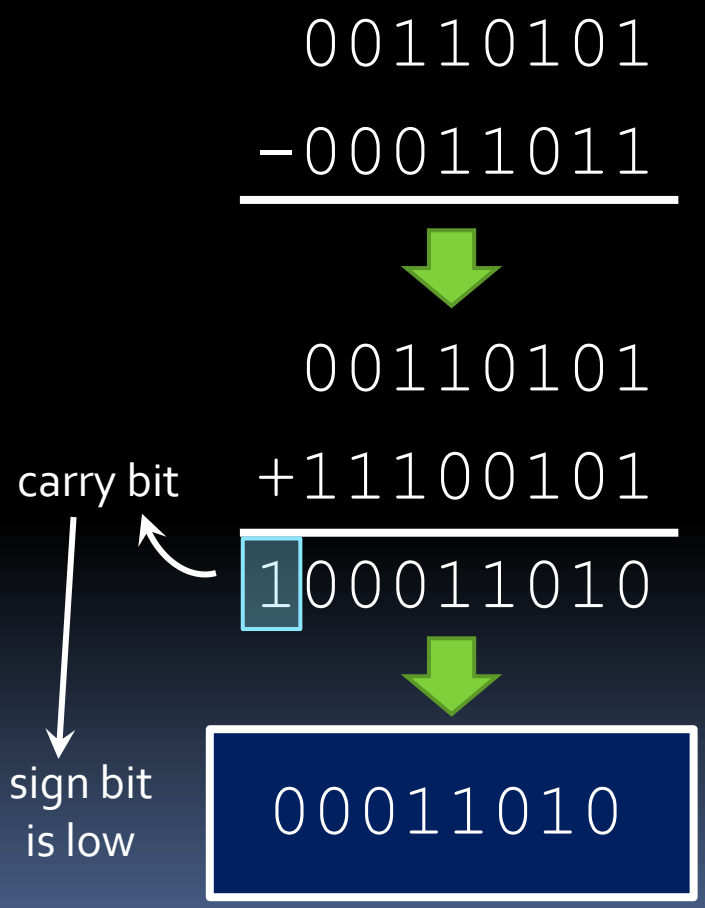
```
  00110101
 -00011011
 ---------
```

- 27 – 53

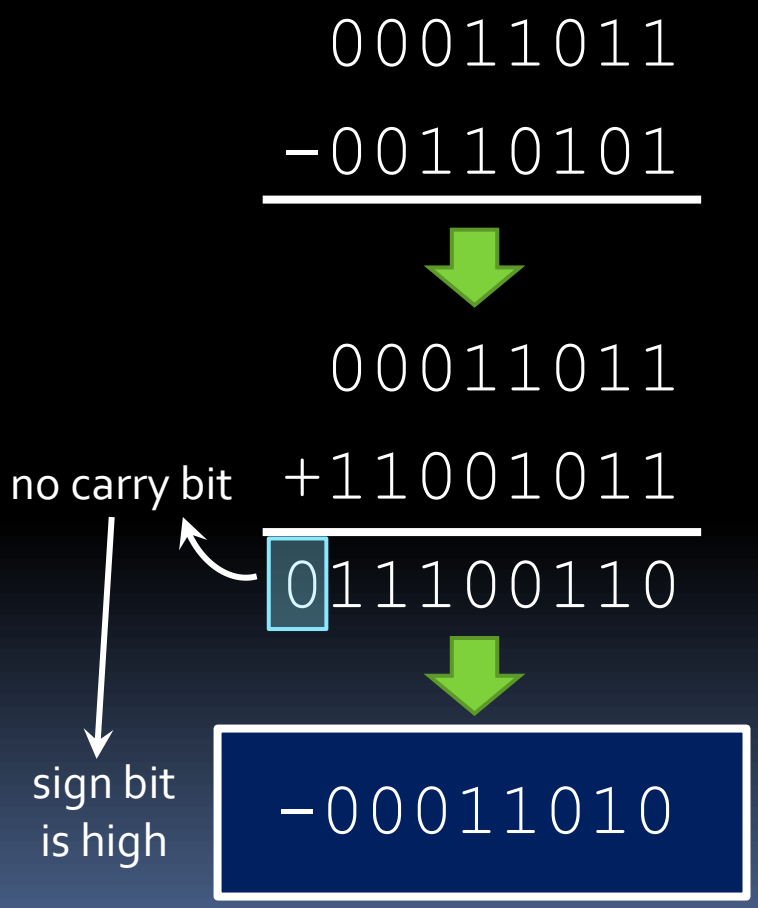
```
  00011011
 -00110101
 ---------
```

Unsigned subtraction example

- 53 – 27

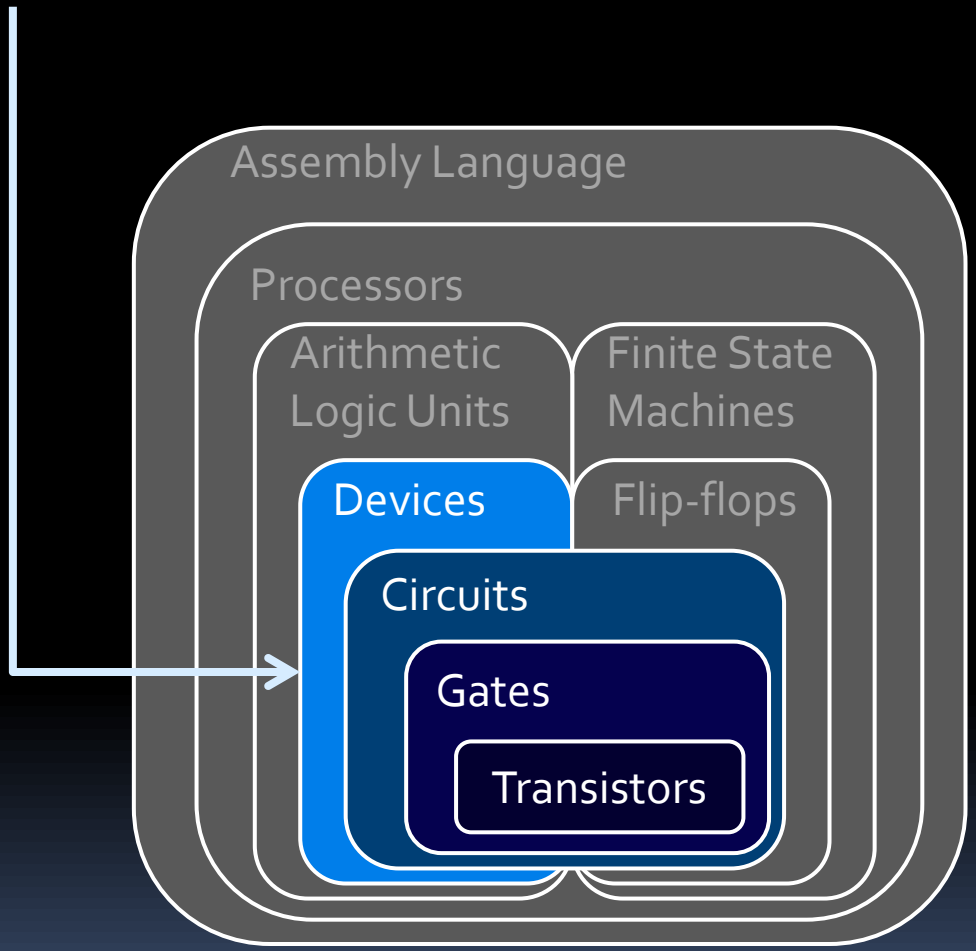


- 27 – 53



Week 4: Sequential Circuits

Last Week


Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Gates
Transistors

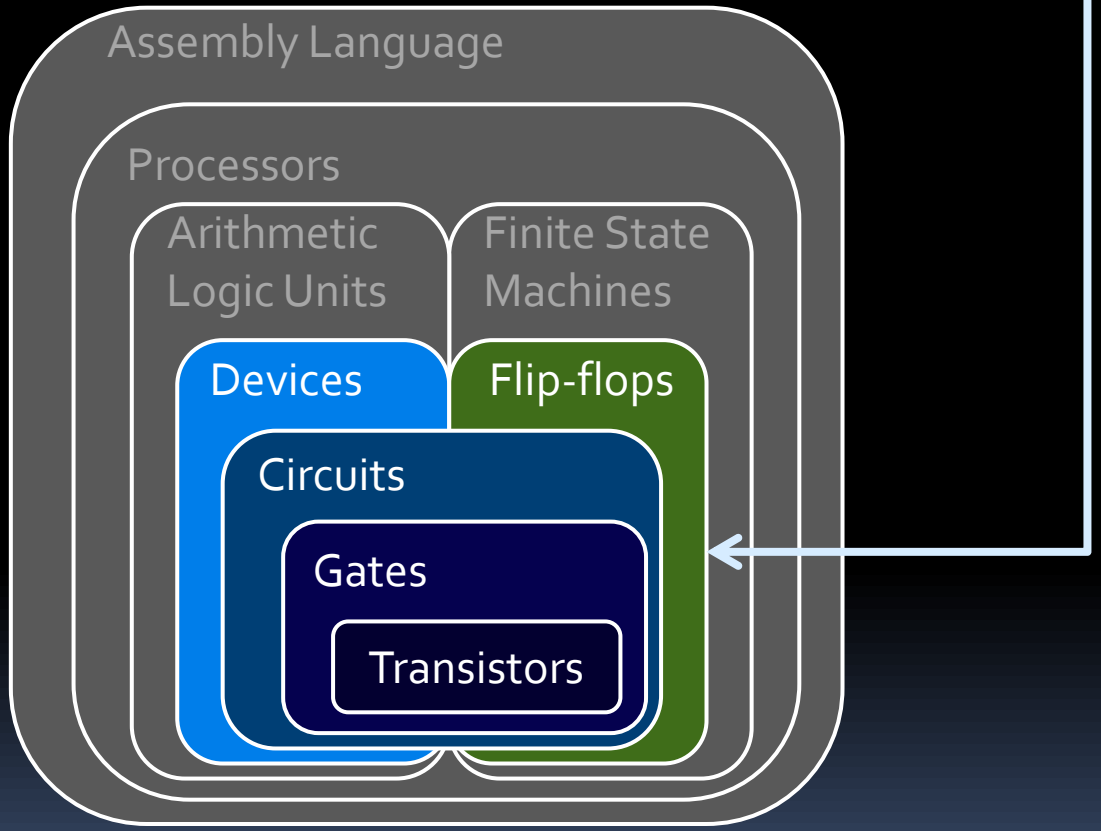
Build A Counter?

- Can you build a device that counts number of key presses?
  - After key is pressed once, hex display should show 1
  - After key is pressed again, show 2
  - Third time: show 3
  - Then reset to 0

Something else to consider…

- Computer specs use terms like "16 GB of RAM" and "5GHz processors".
  - What do these terms mean?
    - RAM = Random Access Memory
    - 16GB = 16 billion `chars` (bytes)
    - 5 GHz = 5 billion clock pulses per second.
  - But what does this mean in circuitry?
    - How do you use circuits to store values?
    - What is the purpose of a clock signal?

This Week

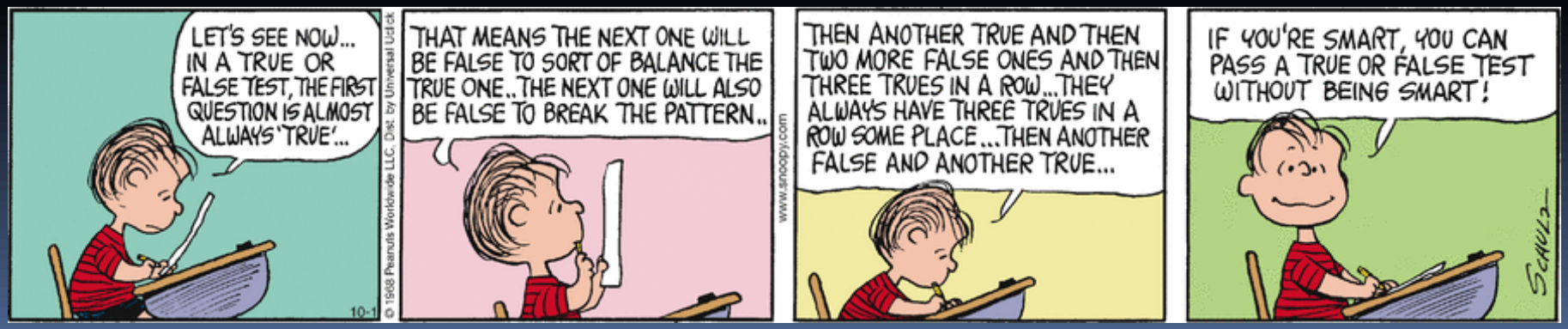

Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Gates
Transistors

Two kinds of circuits

- So far, we've dealt with combinational circuits:
  - Circuits where the output values are entirely dependent and predictable from the input values.
- Another class of circuits: sequential circuits
  - Circuits that also depend on both the inputs and the previous state of the circuit.

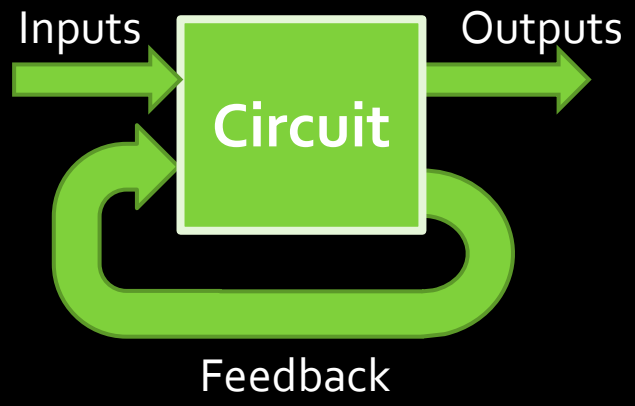
Sequential circuits

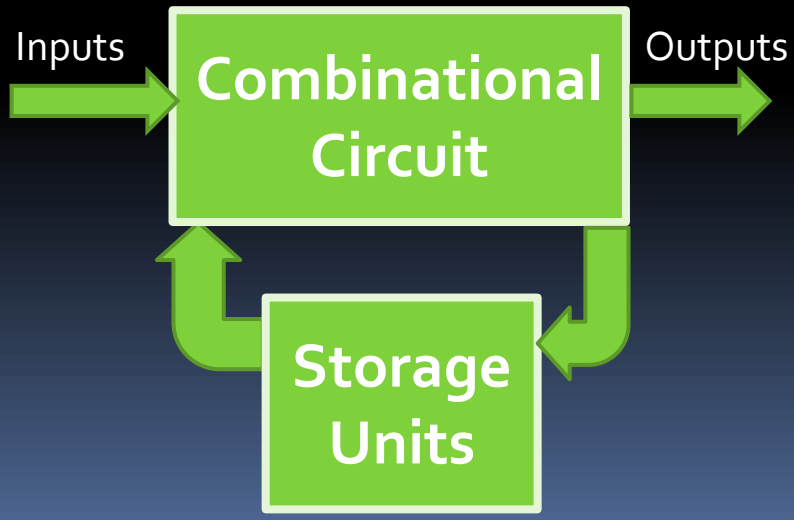
- This creates circuits whose internal state can change over time, where the same input values can result in different outputs.
- Why would we need circuits like this?
  - Memory values
  - Reacting to changing inputs



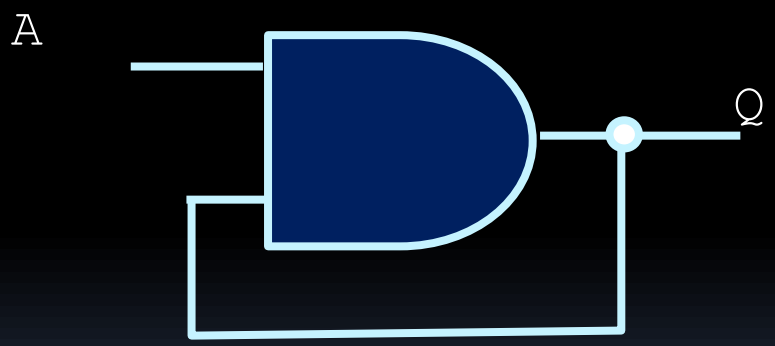
Creating sequential circuits

- Essentially, sequential circuits have feedback in the circuit.
  - How is this accomplished?
  - What is the result of having the output of a component or circuit be connected to its input?



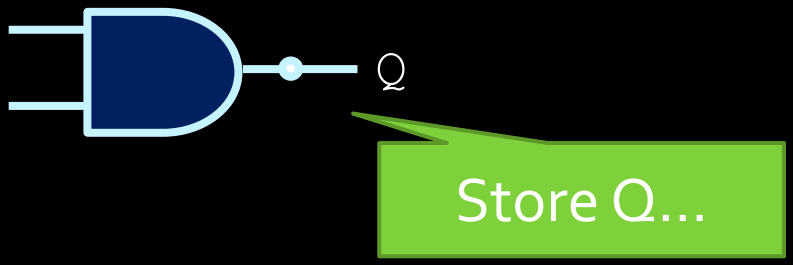


Feedback



Storing and Reusing Values

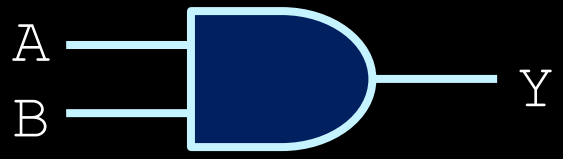
- I want to store the output of an AND gate and reuse it as input.

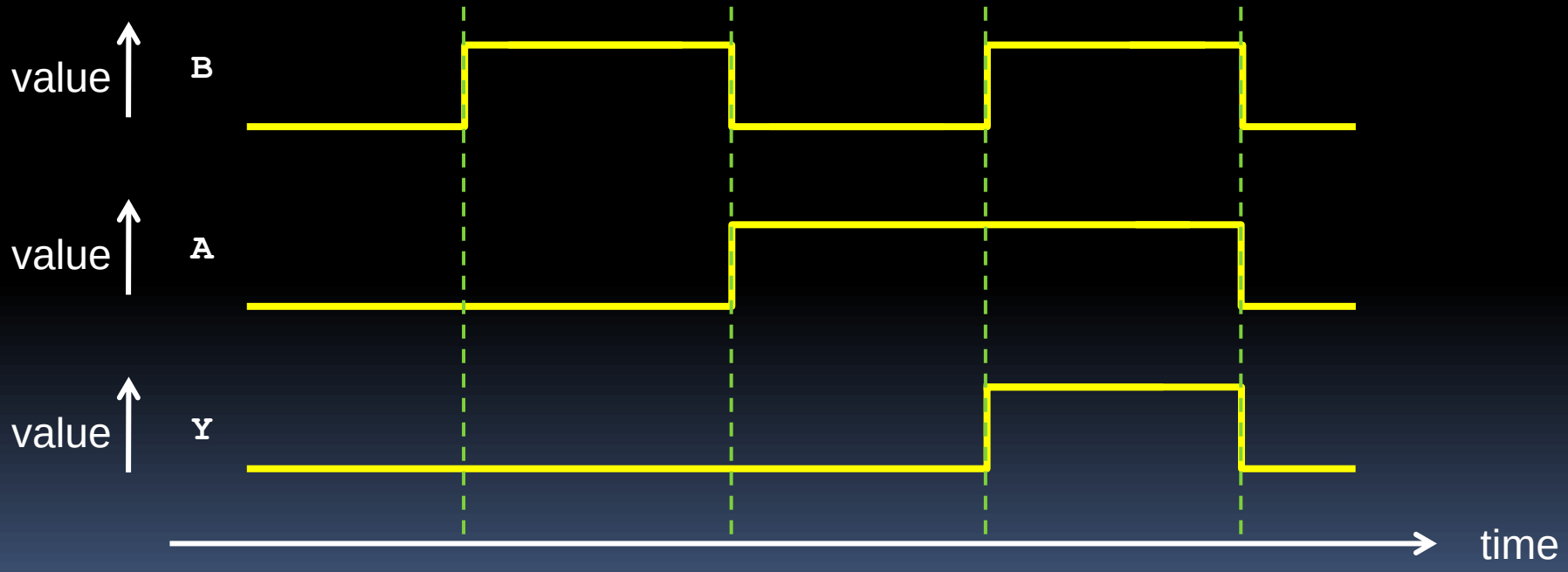


- Does the following work?

- How do we reason about this circuit?

Waveform Diagrams



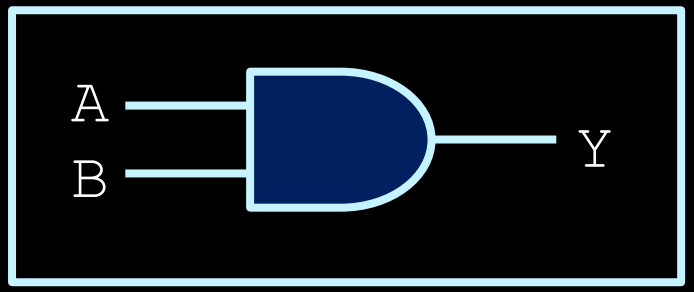


Gate Delay

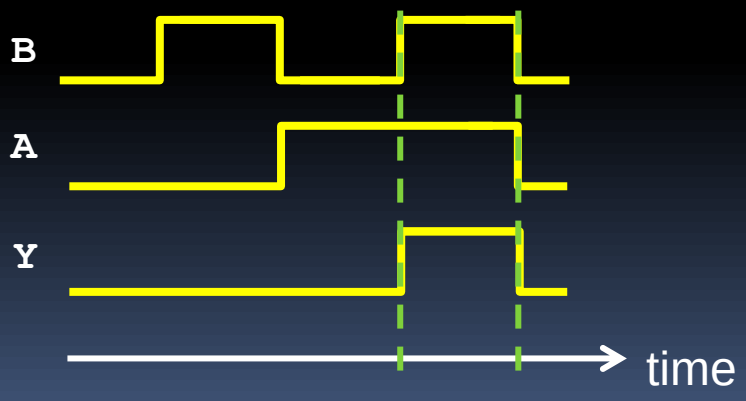
- Outputs don't change instantaneously.
  - Electrons have to move, transistors open/close…
  - Even in combinatorial circuits.

- Gate Delay or Propagation Delay:
  - "The length of time it takes for an input change to result in the corresponding output change."

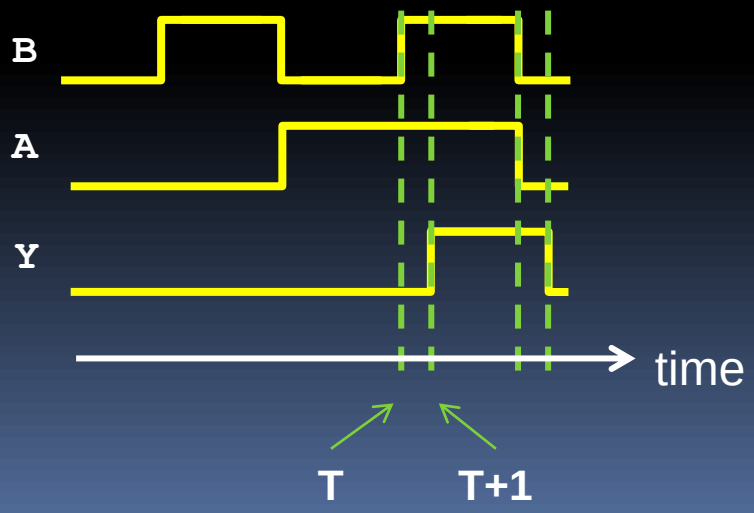
Gate Delays



Ideal

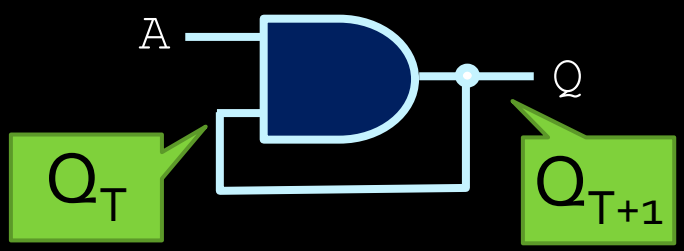


Considering delays



Feedback Circuit Example (AND)

- Some gates don't have useful results when outputs are fed back on inputs.



Q_T and Q_{T+1} represent the values of Q at a time T, and a point in time immediately after (T+1)

A	Q_T	Q_{T+1}
0	0	0
0	1	0
1	0	0
1	1	1

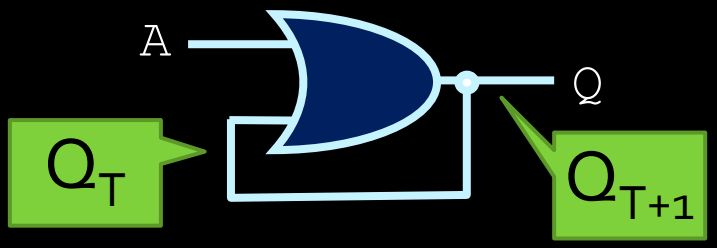
If A=0, Q_{T+1} becomes 0 no matter what Q_T was.

What happens next for later values of A?

Q_{T+1} gets stuck at 0 and cannot change ☹

Feedback Circuit Example (OR)

- Some gates don't have useful results when outputs are fed back on inputs.



In this truth table, Q_T and Q_{T+1} represent the values of Q at a time T, and a point in time immediately after (T+1)

A	Q_T	Q_{T+1}
0	0	0
0	1	1
1	0	1
1	1	1

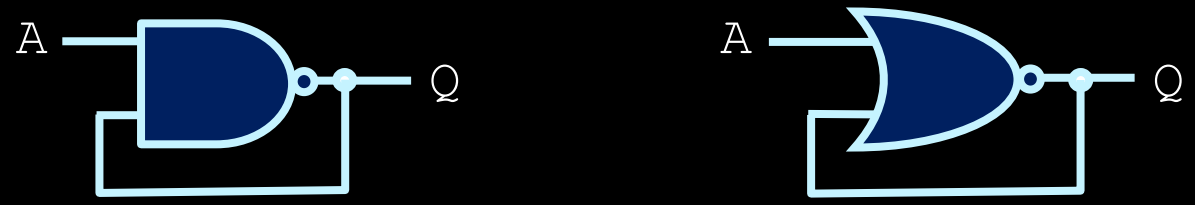
If A=1, Q_{T+1} becomes 1 no matter what Q_T was.

What happens next for later values of A?

Q_{T+1} gets stuck at 1. Not very useful ☹

Feedback Examples (NAND, NOR)

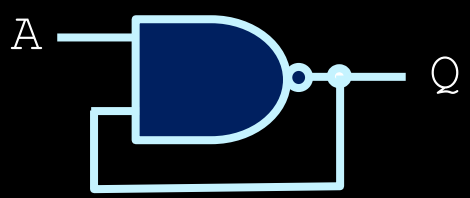
- NAND, NOR gates w/ feedback have more interesting characteristics, which lend themselves to storage devices.



- What makes NAND and NOR feedback circuits different?
  - Unlike the AND and OR gate circuits (which get stuck), the output Q_{T+1} can be changed, based on A.

Feedback Example (NAND)

- Let's assume we set A=0
  - Then, output Q will go to 1.
  - If we leave A unchanged we can store 1 indefinitely!
- If we set A=1, Q's value can change, but there's a catch!

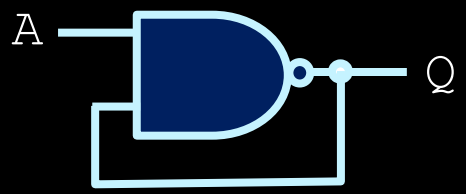


A	Q_T	Q_{T+1}
0	0	1
0	1	1
1	0	1
1	1	0

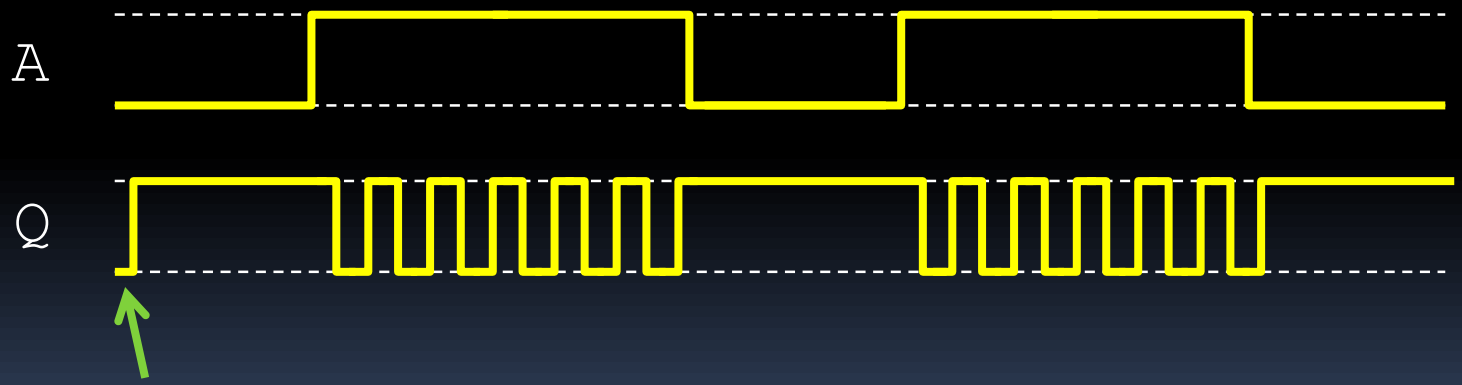
What happens in these last two scenarios?

Unsteady state! Can't store 0 long!

NAND waveform behaviour



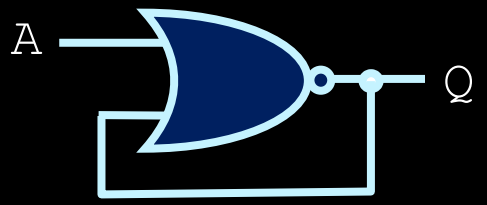
A	Q_T	Q_{T+1}
0	0	1
0	1	1
1	0	1
1	1	0



Gate delay. Output does not change instantaneously

Feedback Example (NOR)

- Let's assume we set A=1
- Then, output Q will go to 0.
- If we leave A unchanged we can store 0 indefinitely!
- If we flip A, we can change Q, but there's a catch here too!



A	Q_T	Q_{T+1}
0	0	1
0	1	0
1	0	0
1	1	0

Feedback behaviour

- NAND behaviour

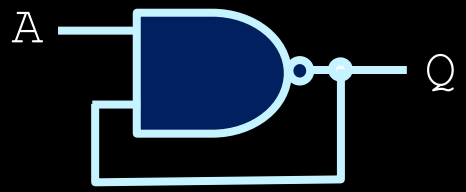
A	Q_T	Q_{T+1}
0	0	1
0	1	1
1	0	1
1	1	0

- NOR behaviour

A	Q_T	Q_{T+1}
0	0	1
0	1	0
1	0	0
1	1	0

- Output Q_{T+1} can be changed, based on A.
- However, gates like these that feed back on themselves enter an unstable state.
  - The output is not stable.

NAND waveform behaviour

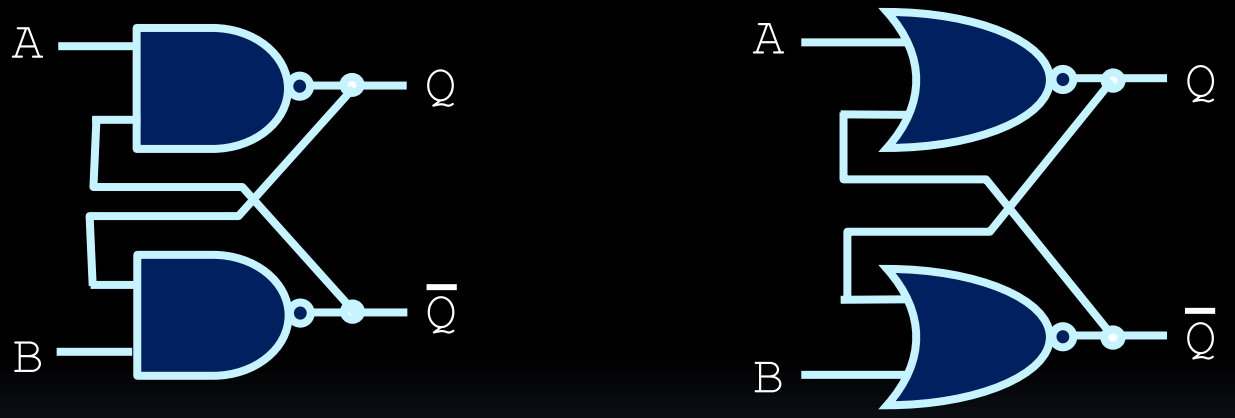


A	Q_T	Q_{T+1}
0	0	1
0	1	1
1	0	1
1	1	0

We want to avoid this. We should be able to store high and low values for as long as we want, and change those values as needed.

Latches

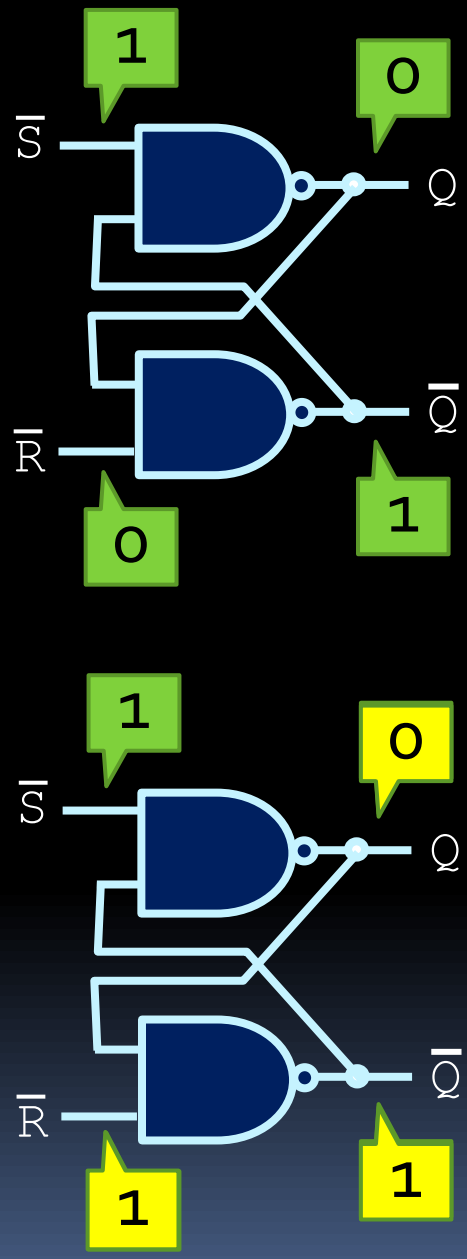
- If multiple gates of these types are combined, you can get stable behaviour.



- These circuits are called latches.

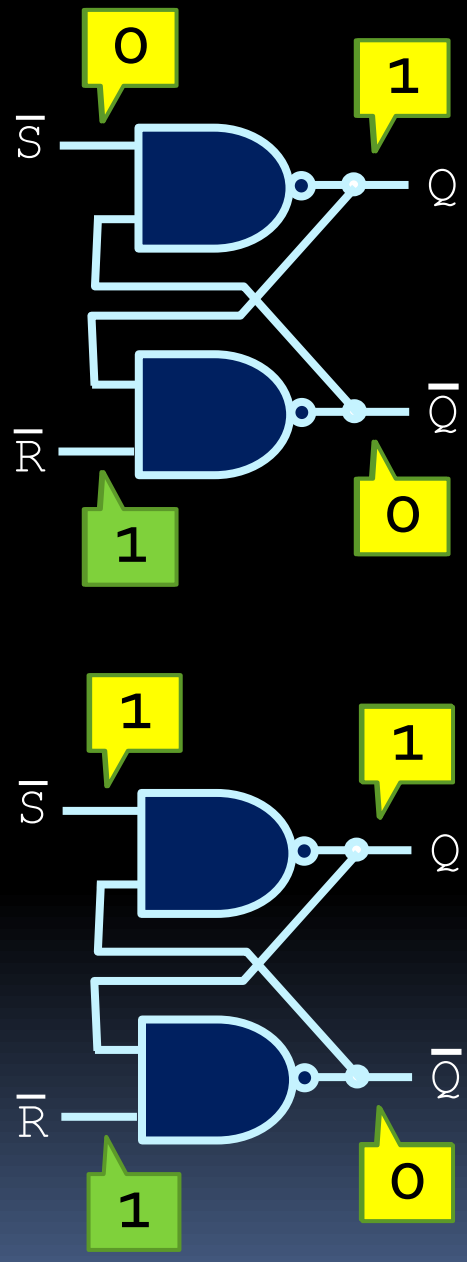
$\overline{S}\overline{R}$ latch

- Let's see what happens when the input values are changed…
  - Assume that $\overline{S}$ and $\overline{R}$ are set to 1 and 0 to start.
  - The $\overline{R}$ input sets the output $\overline{Q}$ to 1, which sets the output Q to 0.
  - Setting $\overline{R}$ to 1 keeps the output value $\overline{Q}$ at 1, which maintains both output values.

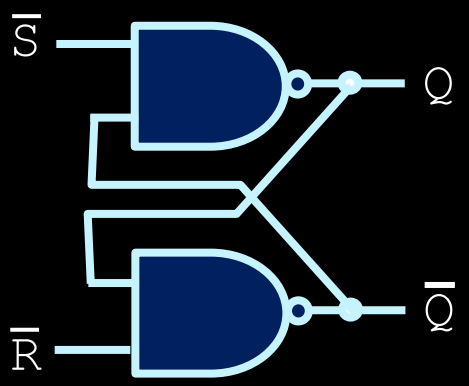


$\overline{S}\overline{R}$ latch

- (continuing from previous)
  - $\overline{S}$ and $\overline{R}$ start with values of 1, when $\overline{S}$ is set to 0.
  - This sets output Q to 1, which sets the output $\overline{Q}$ to 0.
  - Setting $\overline{S}$ back to 1 keeps the output value $\overline{Q}$ at 0, which maintains both output values.
- <u>Conclusion:</u> the input 11 maintains the previous output state!



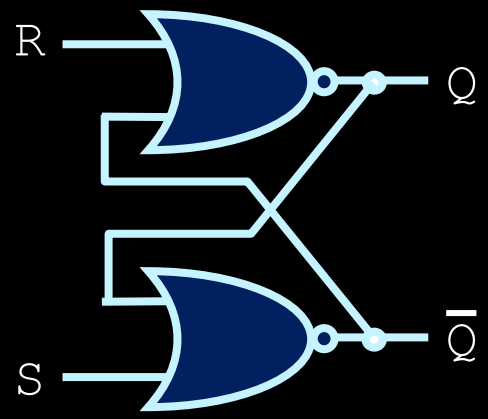
$\overline{S}\overline{R}$ latch



$\overline{S}$	$\overline{R}$	Q_T	$\overline{Q}_T$	Q_{T+1}	$\overline{Q}_{T+1}$
0	0	X	X	1	1
0	1	X	X	1	0
1	0	X	X	0	1
1	1	0	1	0	1
1	1	1	0	1	0

- $\overline{S}$ and $\overline{R}$ are called "set" and "reset" respectively.
- Note how the circuit "remembers" its signal when going from 10 or 01 to 11.
- Going from 00 to 11 produces unstable behaviour!
  - In the sense that we don't know what the output will be. It can be 0 or 1, depending on which input changes first.

SR latch

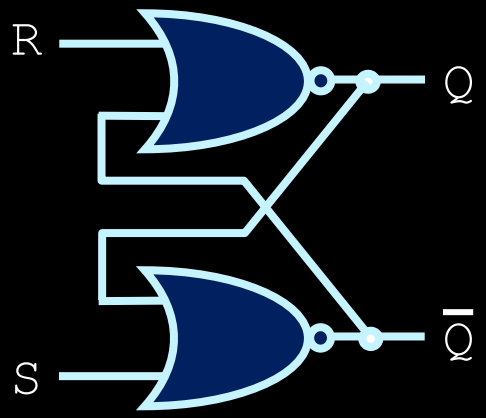


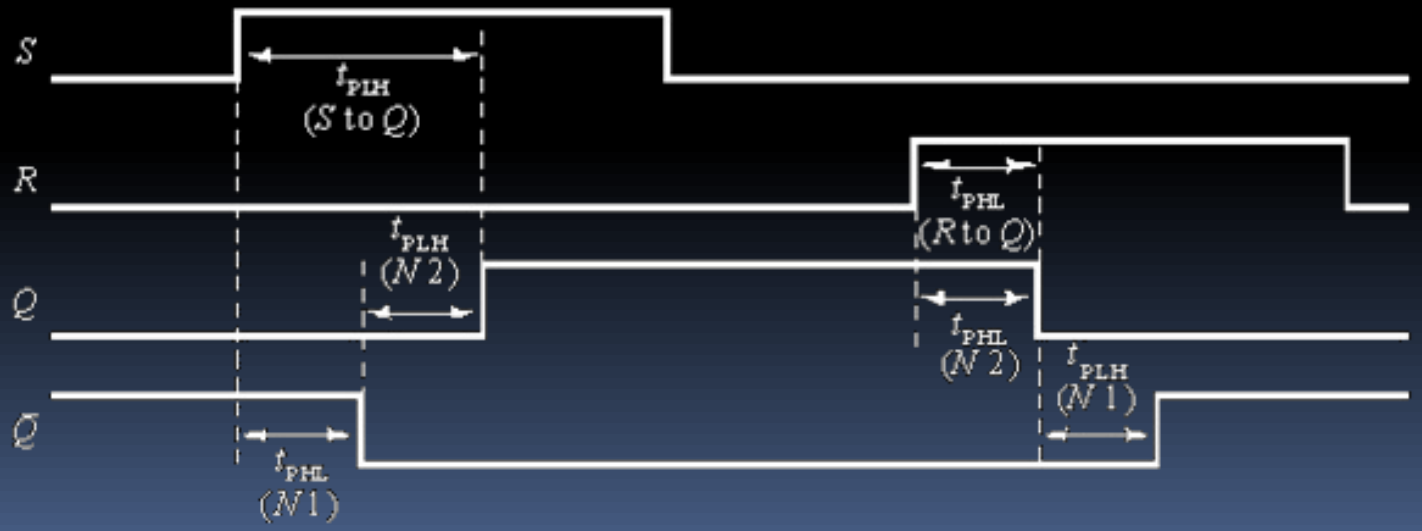
S	R	Q_T	$\overline{Q}_T$	Q_{T+1}	$\overline{Q}_{T+1}$
0	0	0	1	0	1
0	0	1	0	1	0
0	1	X	X	0	1
1	0	X	X	1	0
1	1	X	X	0	0

- In this case, S and R are "set" and "reset".
- In this case, the circuit "remembers" previous output when going from 10 or 01 to 00.
- As with $\overline{S}\overline{R}$ latch, unstable behaviour is possible, but this time when inputs go from 11 to 00.

SR latch timing diagram

- Important to note that the output signals don't change instantaneously.





More on instability

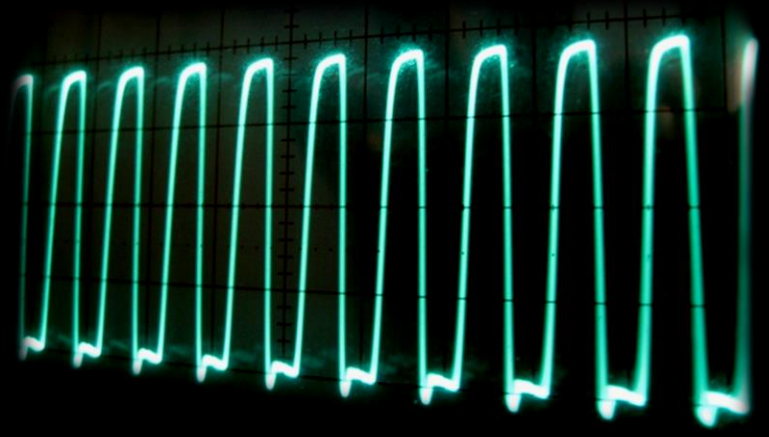
- Unstable behaviour occurs when a $\overline{S}\overline{R}$ latch goes from `00` to `11`, or a SR latch goes from `11` to `00`.
  - The signals don't change simultaneously, so the outcome depends on which signal changes first.
- Because of the unstable behaviour, `00` is considered a forbidden state in NAND-based $\overline{S}\overline{R}$ latches, and `11` is considered a forbidden state in NOR-based SR latches.

Reading from latches

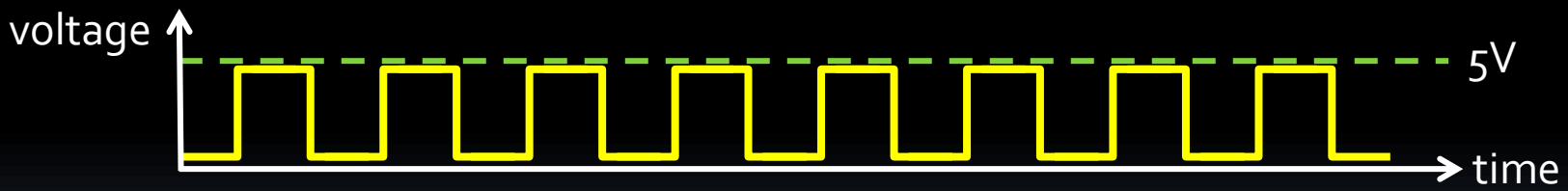
- Now we have circuit units that can store high or low values. When can we read from them?
  - For instance, when do we know when the output is ready to be sampled?
  - If the output is high, how can we tell the difference between a single high value and two high values in a row?
- Need some sort of timing signal, to let the circuit know when the output may be sampled.

  → clock signals.

Clock signals



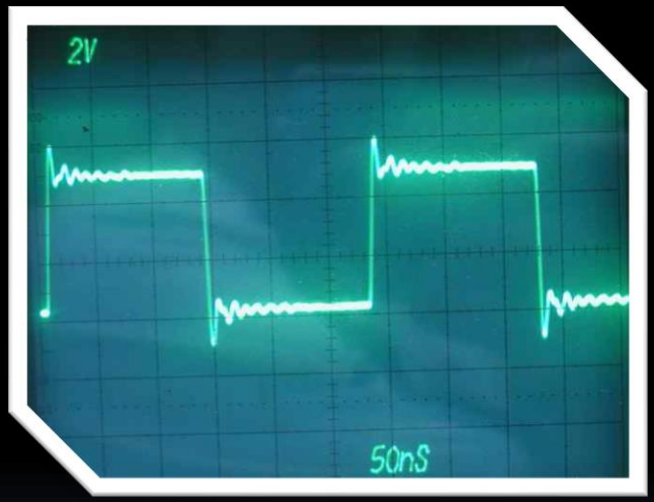
- "Clocks" are a regular pulse signal, where the high value indicates that the output of the latch may be sampled.
- Usually drawn as:



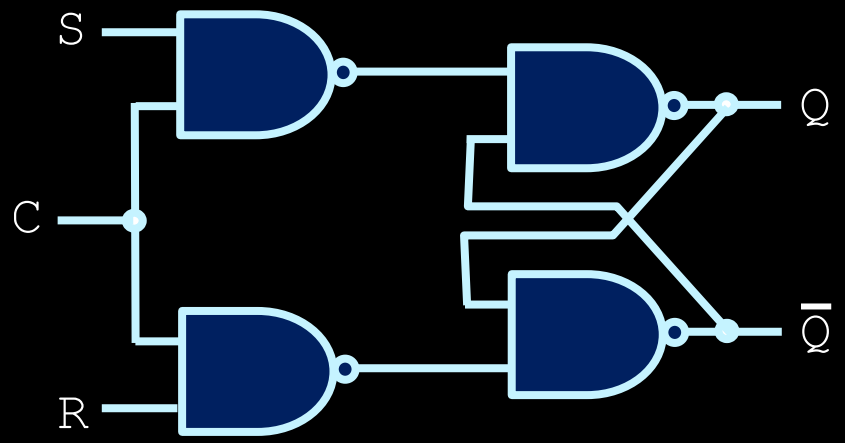
- But looks more like:

Signal restrictions

- What's the limit to how fast the latch circuit can be sampled?
- Determined by:
  - latency time of transistors
    - Setup and hold time
  - setup time for clock signal
    - Jitter
    - Gibbs phenomenon
- Frequency = how many pulses occur per second, measured in Hertz (or Hz).



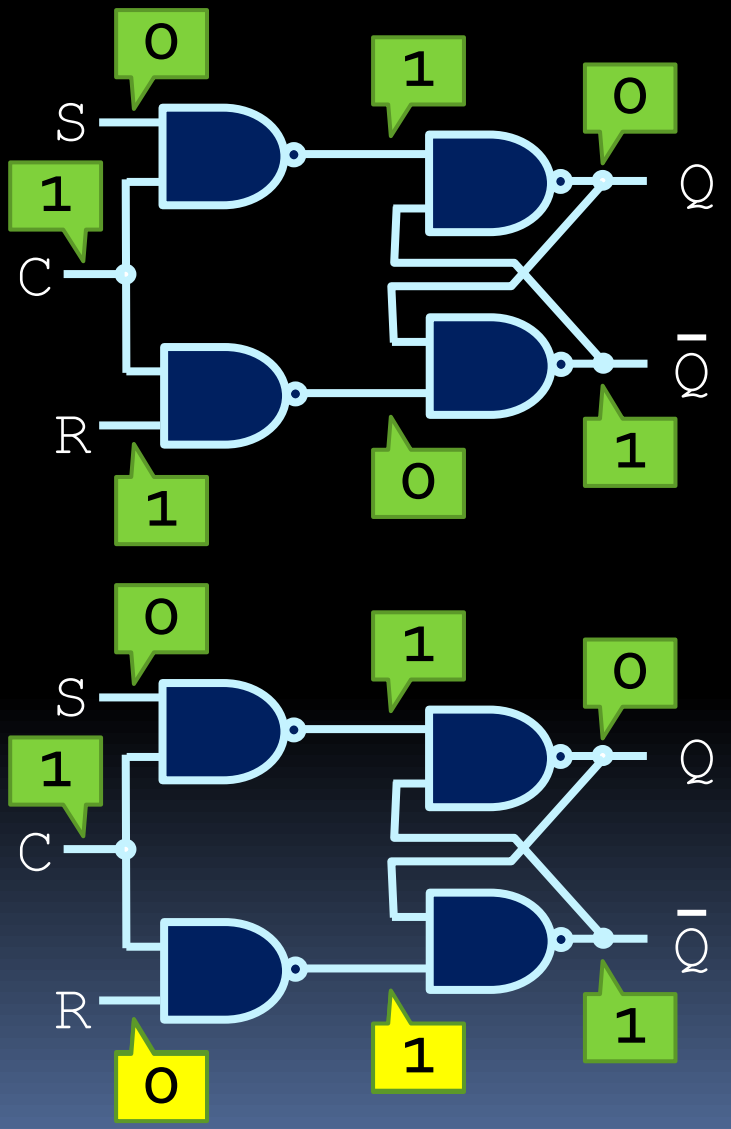
Clocked SR latch



- Adding another layer of NAND gates to the $\overline{S}\overline{R}$ latch gives us a clocked SR latch (or gated SR latch)
  - Basically, a latch with a control input signal `C`.
- The input `C` is often connected to a pulse signal that alternates regularly between `0` and `1` (`clock`)

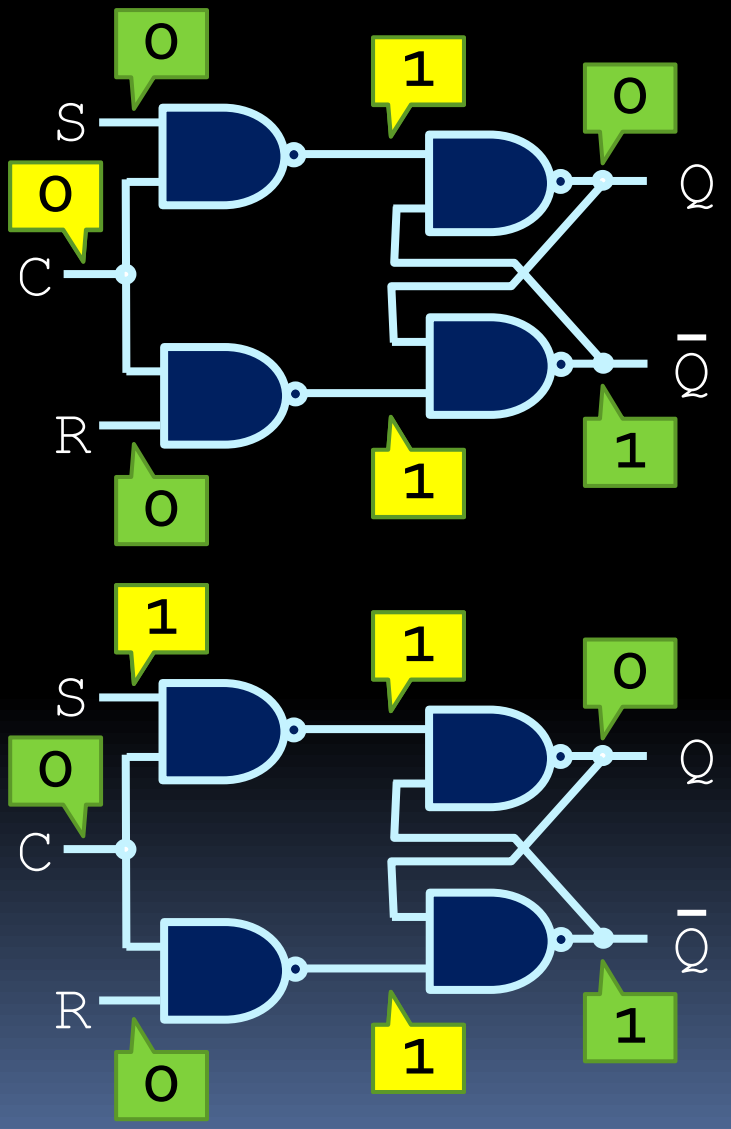
Clocked SR latch behaviour

- Same behaviour as SR latch, but with timing:
  - Start off with `S=0` and `R=1`, like earlier example.
  - If clock is high, the first NAND gates invert those values, which get inverted again in the output.
  - Setting both inputs to `0` maintains the output values.

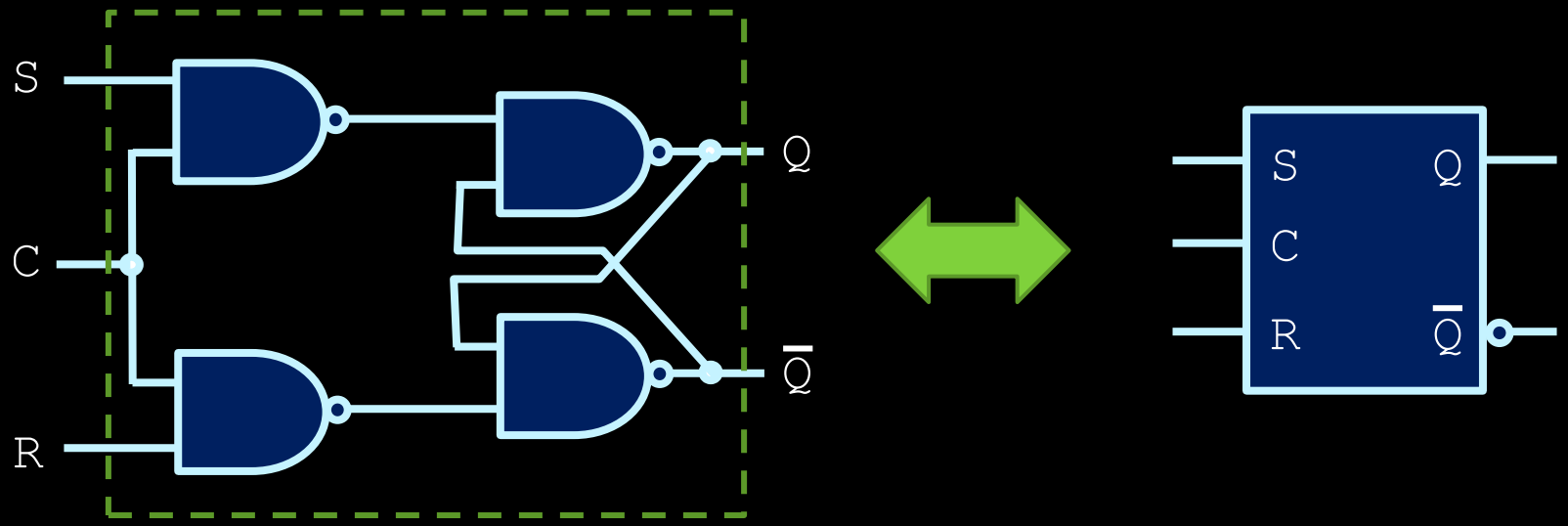


Clocked SR latch behaviour

- Continued from previous:
  - Now set the clock low.
  - Even if the inputs change, the low clock input prevents the change from reaching the second stage of NAND gates.
  - Result: the clock needs to be high in order for the inputs to have any effect.

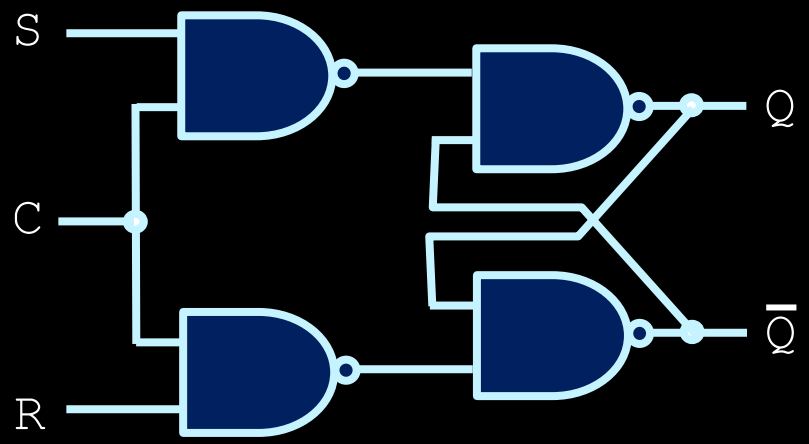


Clocked SR latch



- This is the typical symbol for a clocked SR latch.
- This only allows the S and R signals to affect the circuit when the clock input (C) is high.
- Note: the small NOT circle after the $\overline{Q}$ output is simply the notation to use to denote the inverted output value. It's not an extra NOT gate.

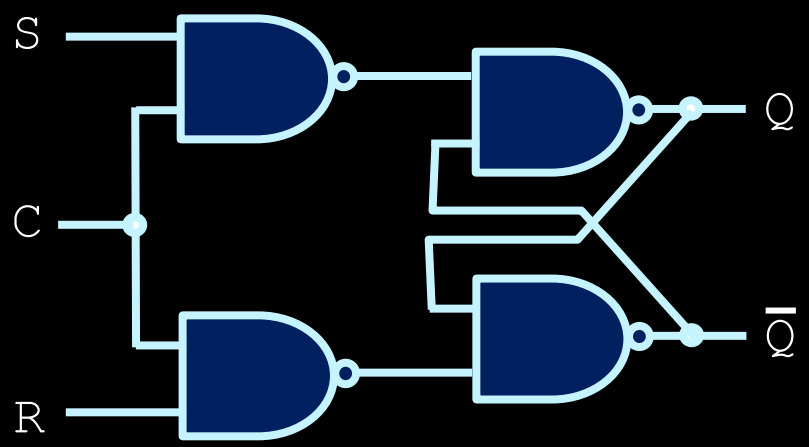
Clocked SR latch behaviour



Q_T	S	R	Q_{T+1}	Result
0	0	0	0	no change
0	0	1	0	reset
0	1	0	1	set
0	1	1	?	???
1	0	0	1	no change
1	0	1	0	reset
1	1	0	1	set
1	1	1	?	???

- Wait!
- Where's the clock?
- There's a better way to look at this….

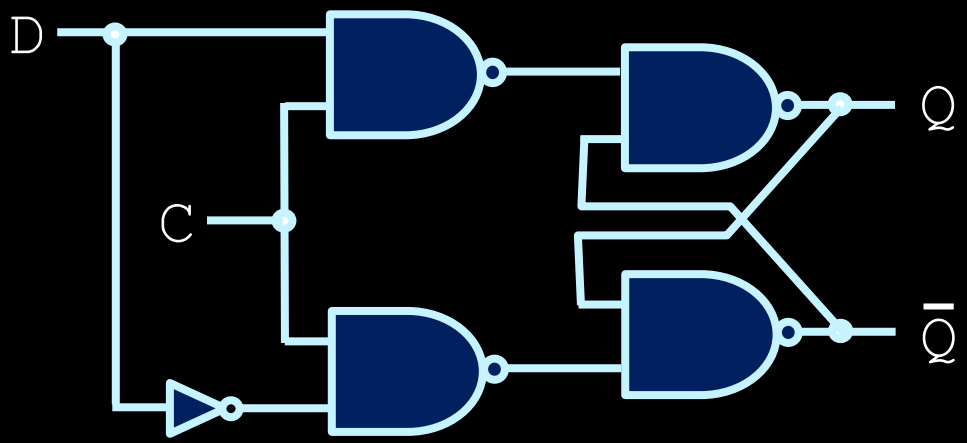
Clocked SR latch behaviour



C	S	R	Q_{T+1}	Result
0	X	X	Q_T	no change
1	0	0	Q_T	no change
1	0	1	0	reset
1	1	0	1	set
~~1~~	~~1~~	~~1~~	~~?~~	~~Undefined~~

- Assuming the clock is 1, we still have a problem when S and R are both 1, since the state of Q is indeterminate.
  - Better design: prevent S and R from both going high.

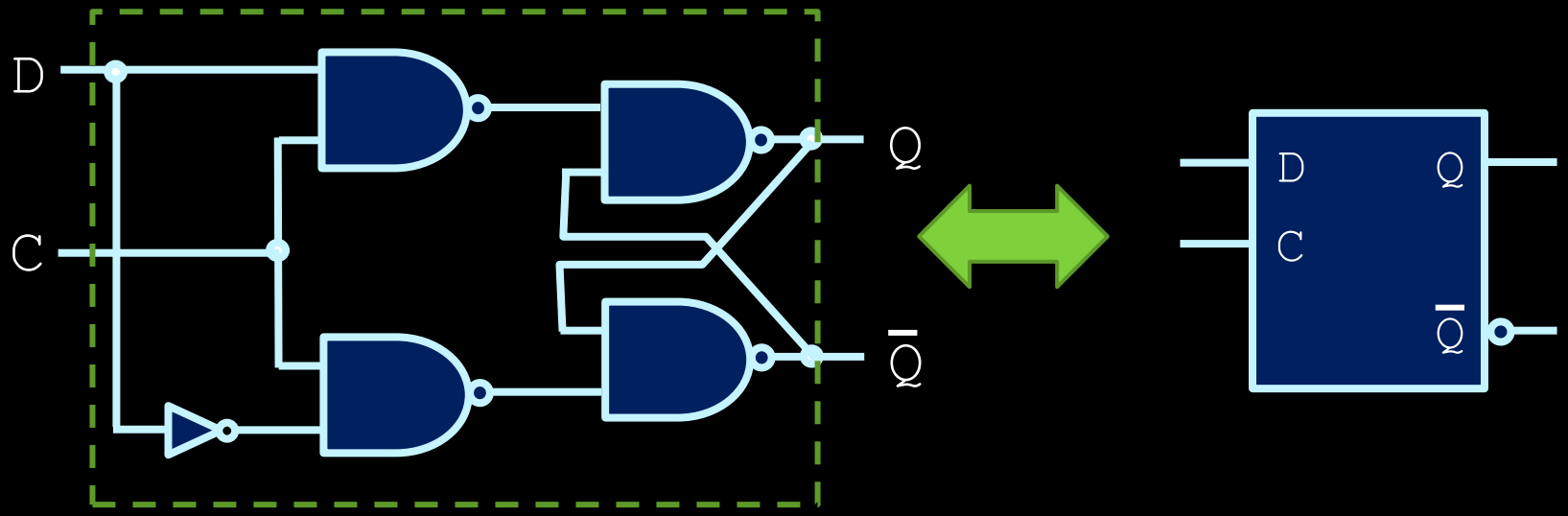
D latch



Q_T	D	Q_{T+1}
0	0	0
0	1	1
1	0	0
1	1	1

- By making the inputs to R and S dependent on a single signal D, you avoid the indeterminate state problem.
- The value of D now sets output Q low or high whenever C is high.

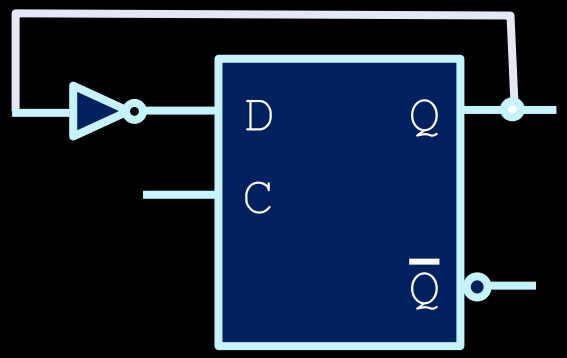
D latch

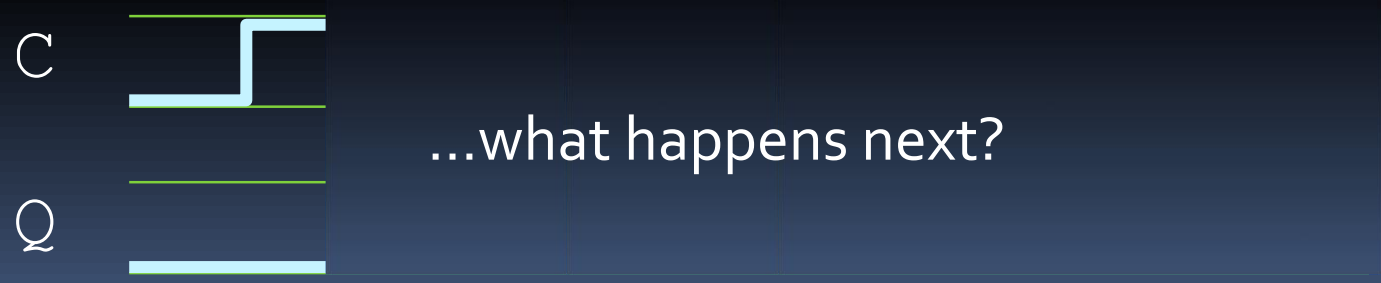


- This design is good, but still has problems.
  - i.e. timing issues.
  - How can we maintain state?

Latch timing issues

- Consider the circuit on the right:
- When the clock signal is high, the output looks like the waveform below:
  - Output keeps toggling back and forth.

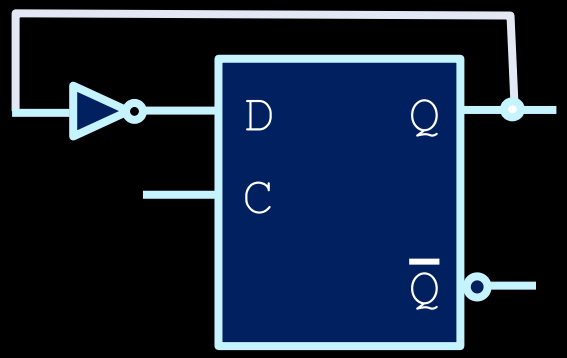




…what happens next?

Latch timing issues

- Consider the circuit on the right:
- When the clock signal is high, the output looks like the waveform below:
  - Output keeps toggling back and forth.



D-Latch is transparent!

- Transparent means that
  - Any changes to its inputs are visible to the output when control signal (Clock) is 1.

- Key Take-away: The "output of a latch **should not** be applied directly or through combinational logic to the input of the same or another latch when they all have the same control (clock) signal."