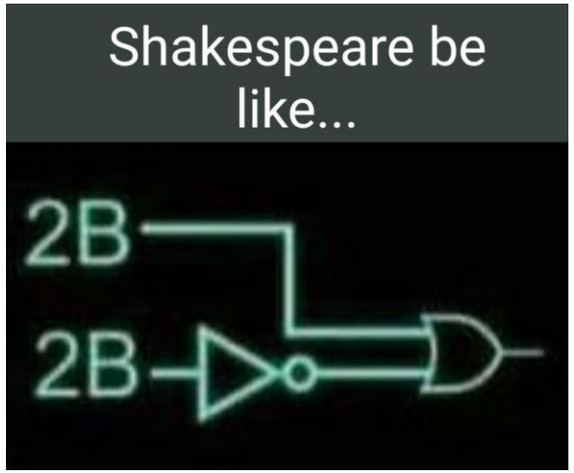


EECS 370 - Lecture 9

Sequential Logic

Poll: What is NOR(0,X)?

Announcements

- Project 2 posted
  - First part due next Thursday
- HW 1 due yesterday
  - Can still submit by tonight for reduced credit
  - HW 2 posted later today

42 Days until US Election Day

- November 5th
- Do you know if you are / can be registered to vote?
  - vote.gov
- Lectures that day will be optional bonus material
  - Not covered in labs / hw / exams

Next few lectures: Digital Logic

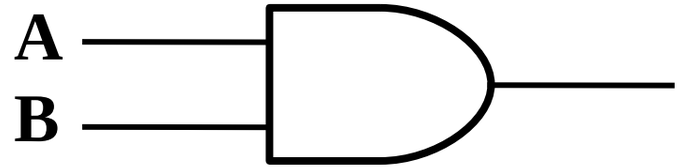
- Lectures 1-7:
  - LC2K and ARMv8/LEGv8 ISAs
  - Converting C to Assembly
  - Function Calls
  - Linking

- Lecture 8:
  - Finish up linking
  - Combinational Logic

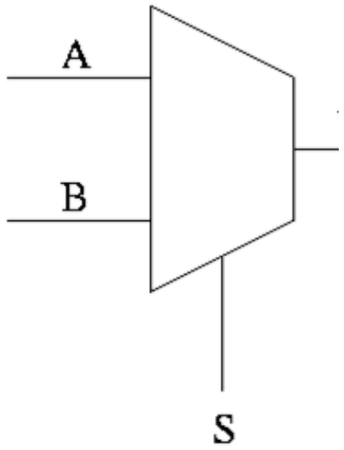
- **Today:**
  - **Continue combinational logic**
  - **Sequential Logic**

Review

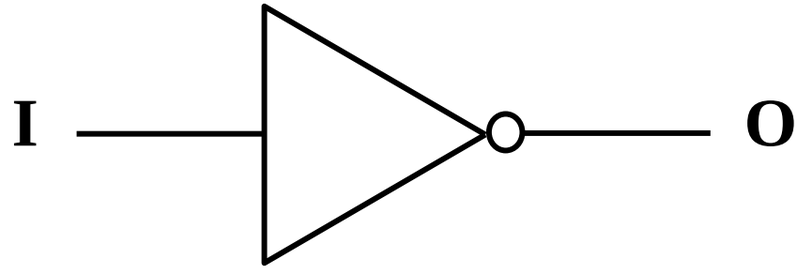
AND



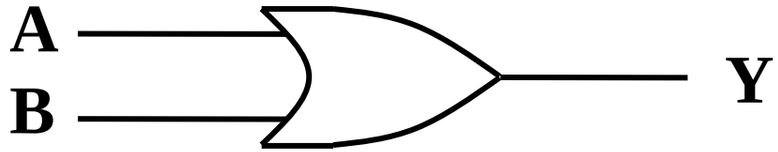
MUX



NOT



OR

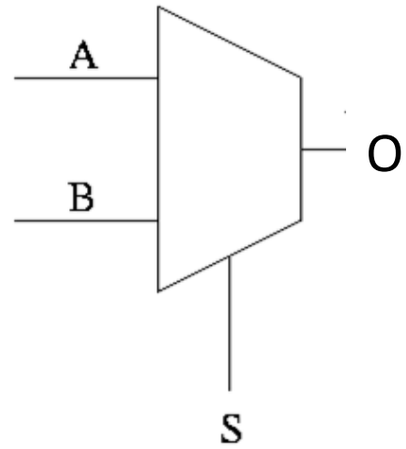


Building Complexity: Selecting

- We want to design a circuit that can select between two inputs (multiplexer or **mux**)
- Let's do a one-bit version
    1. Draw a truth table

A	B	S	O
0	**0**	**0**	
0	**0**	**1**	
0	**1**	**0**	
0	**1**	**1**	
1	**0**	**0**	
1	**0**	**1**	
1	**1**	**0**	
1	**1**	**1**	

Symbol

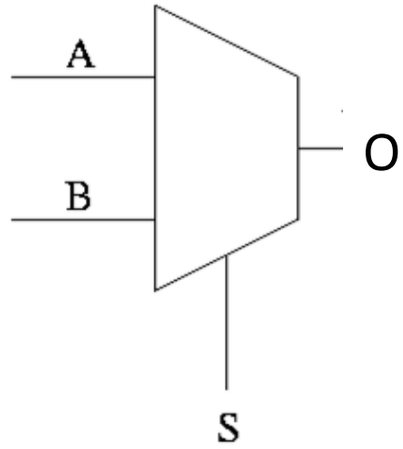


O = S ? B : A

Building Complexity: Selecting

- We want to design a circuit that can select between two inputs (multiplexor or **mux**)
- Let's do a one-bit version
    1. Draw a truth table

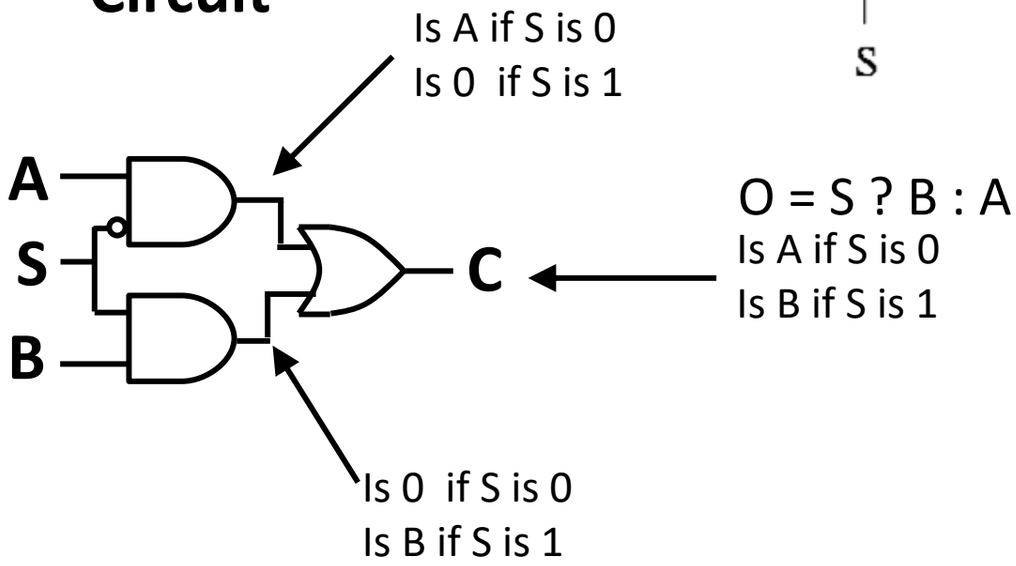
Symbol



A	B	S	O
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Muxes are universal! A 2^N entry truth table can be implemented by passing each ouput value into an input of a 2^N–to-1 mux

Circuit



Building Complexity: Addition

- We want to design a circuit that performs binary addition
- Let's start by adding two bits
  - Design a circuit that takes two bits (A and B) as input
    - Generates a sum and carry bit (S and C)
  1. Make a truth table
  2. Design a circuit

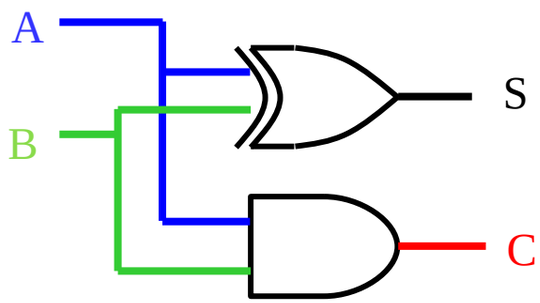
```
  1 0 0 1 1
+ 0 0 1 1 0
-----------
```

A	B	C	S

Building Complexity: Addition

- We want to design a circuit that performs binary addition
- Let's start by adding two bits
  - Design a circuit that takes two bits (A and B) as input
    - Generates a sum and carry bit (S and C)
  1. Make a truth table
  2. Design a circuit

```
  0 1 1 0
  1 0 0 1 1
+ 0 0 1 1 0
-----------
  1 1 0 0 1
```

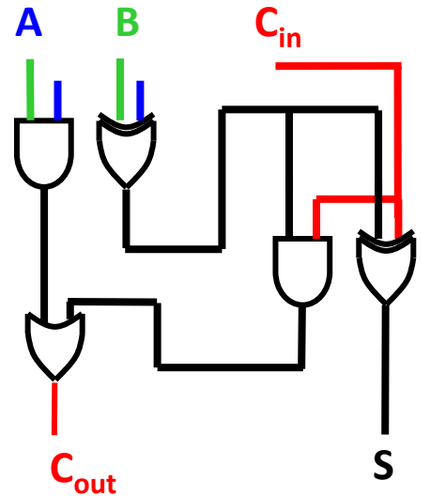


A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Building Complexity: Addition

- Now we can add two bits, but how do we deal with carry bits?
- This is a **full adder**
  - We have to design a circuit that can add three bits
    - Inputs: A, B, Cin
    - Outputs: S, Cout
  1. Design a truth table
  2. Circuit
- This is a **full adder**

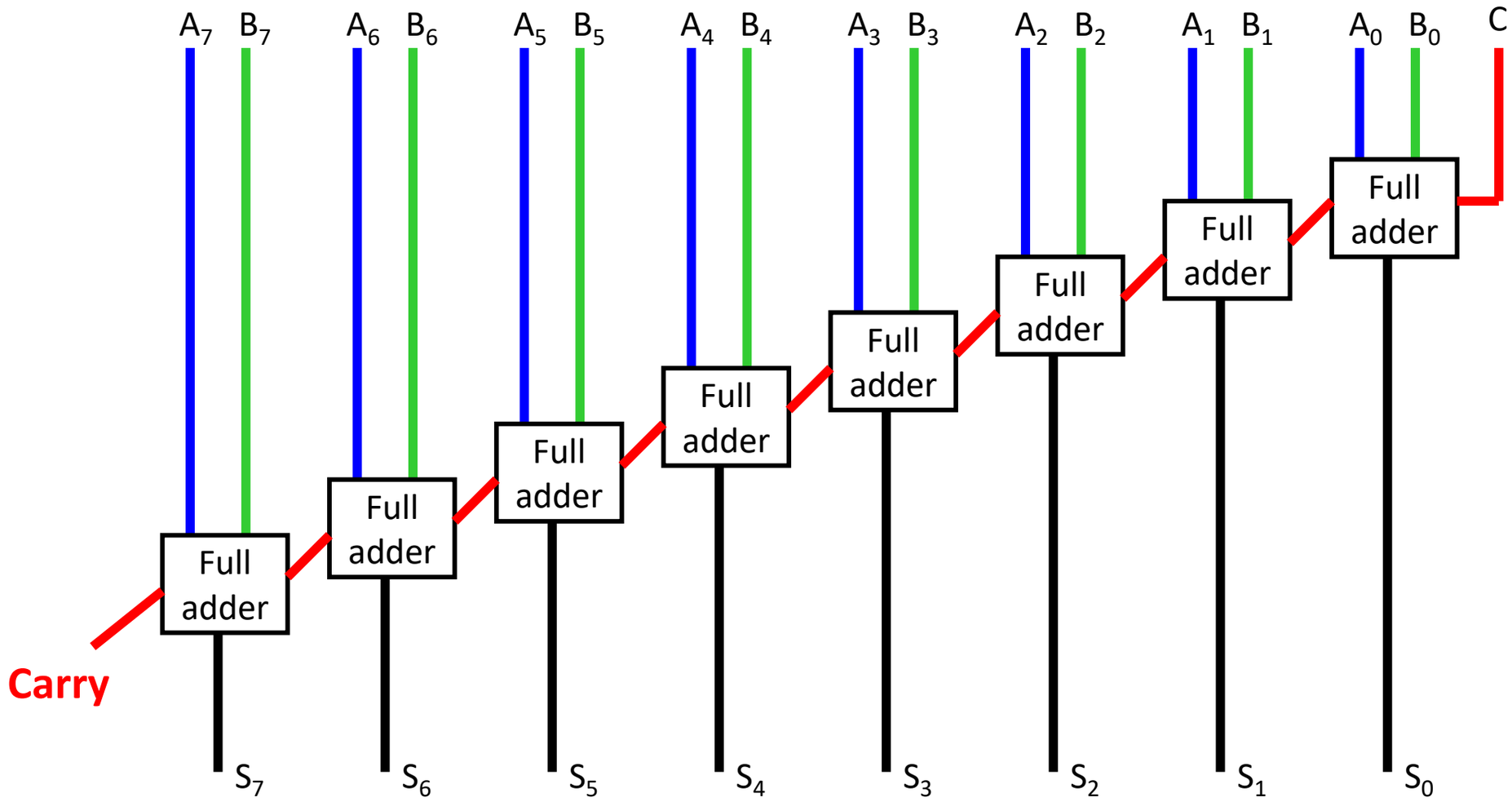
```
  0 1 1 0
  1 0 0 1 1
+ 0 0 1 1 0
-----------
  1 1 0 0 1
```



Cin	A	B	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

8-bit Ripple Carry Adder

If we invert B's bits and set C to 1, we also have a subtractor! Why?

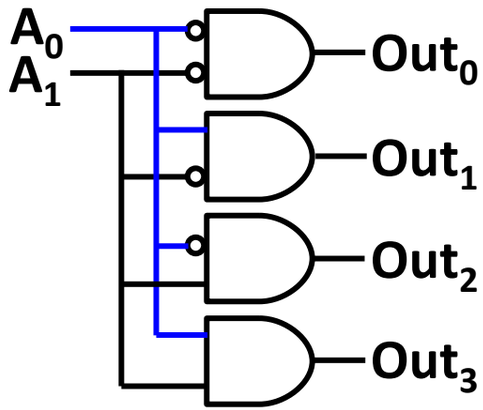


This will be very slow for 32 or 64 bit adds, but is sufficient for our needs

Building Complexity: Decoding

- Another common device is a decoder
  - Input: N-bit binary number
  - Output: 2^N bits, exactly one of which will be high
  - Allows us to **index** into things (like a register file)

Decoder

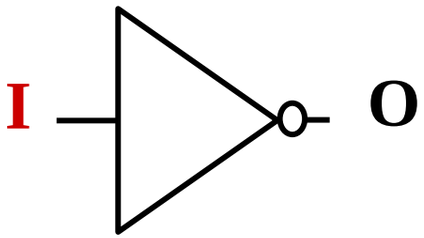


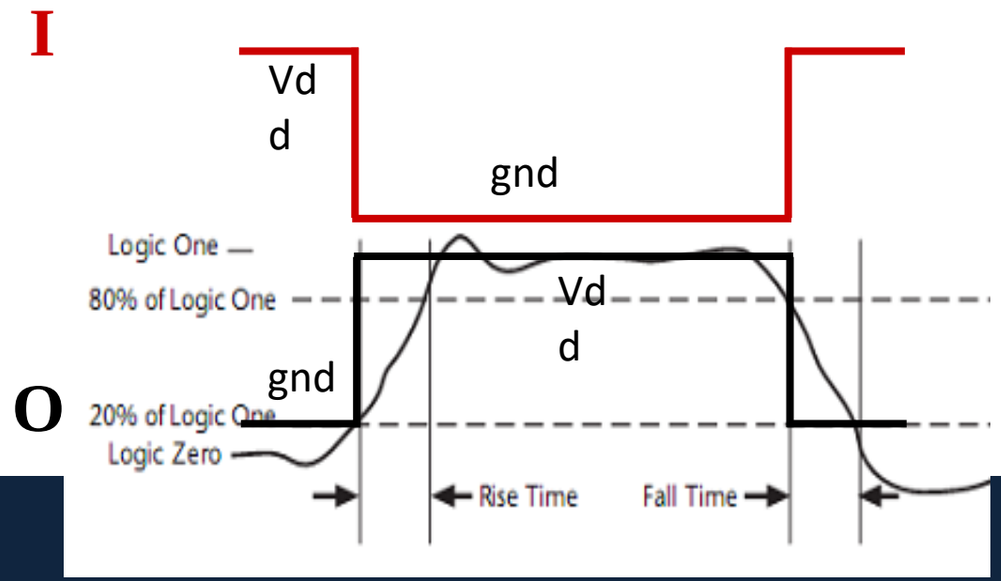
Poll: What will be the output for 101?

Propagation delay in combinational gates

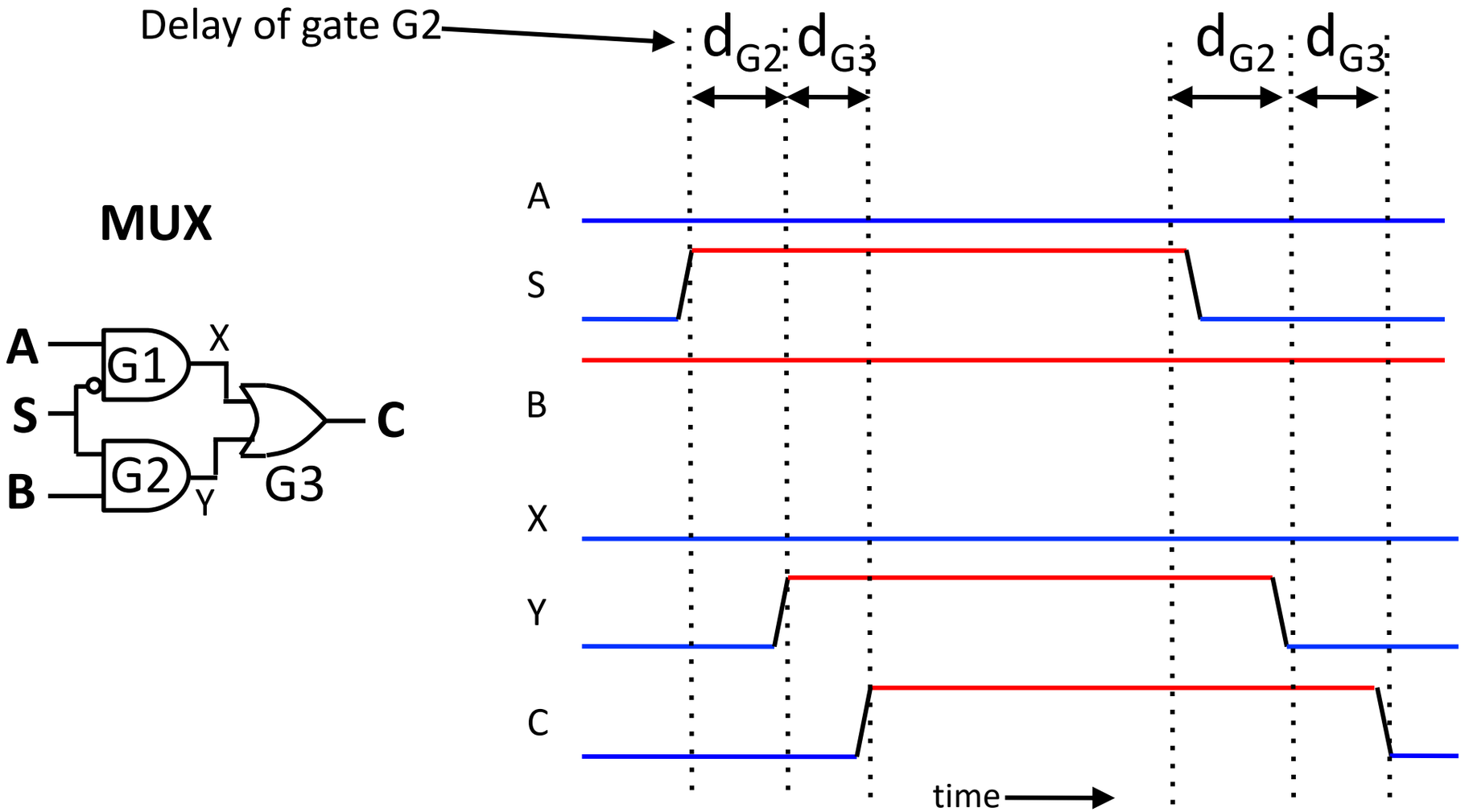
- Gate outputs do not change exactly when inputs do.
  - Transmission time over wires (~speed of light)
  - Saturation time to make transistor gate switch

  ⇒ Every combinatorial circuit has a propagation delay
  (time between input and output stabilization)





Timing in Combinational Circuits

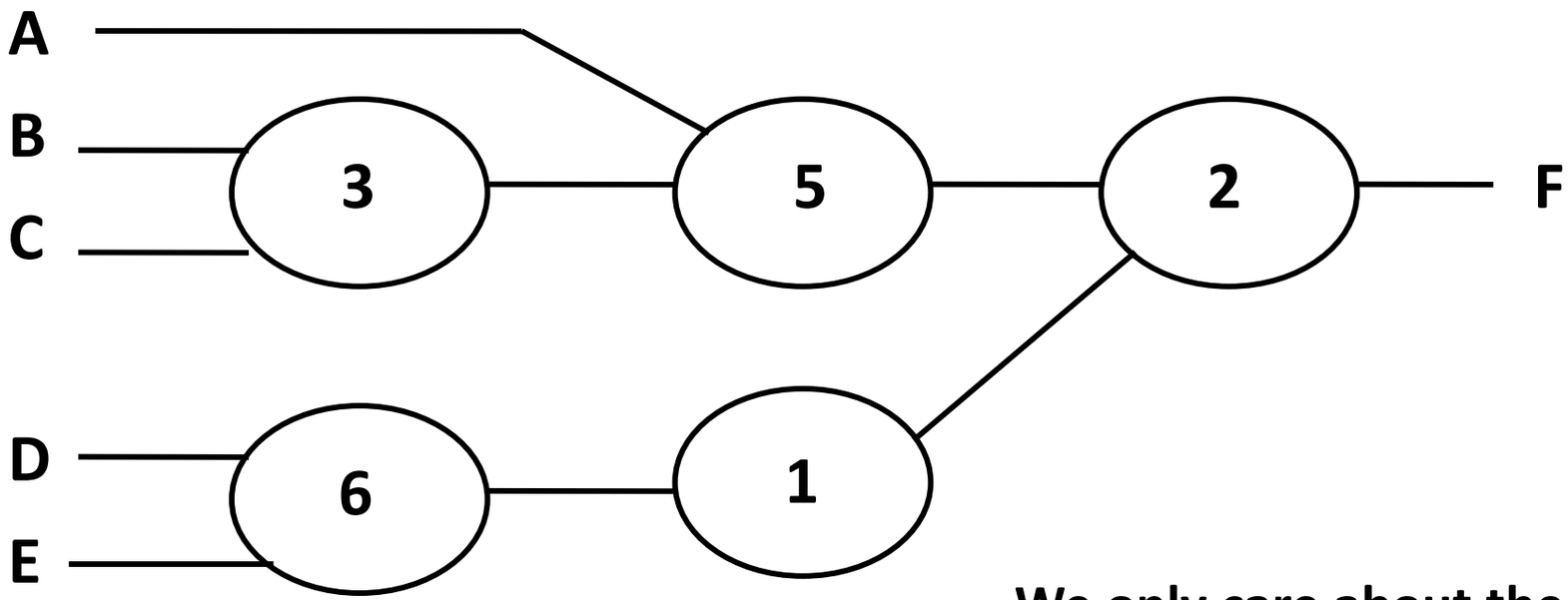


What is the input/output delay (or simply, delay) of the MUX?

What is the delay of this Circuit?

Each oval represents one gate, the type does not matter

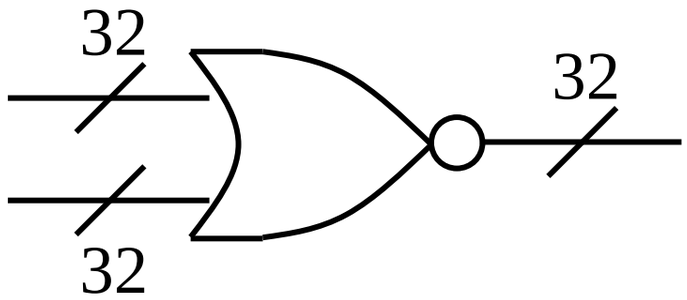
Poll : What is the delay?

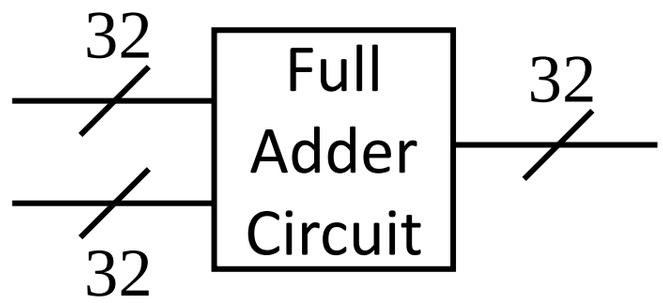


We only care about the longest path, or critical path

Exercise

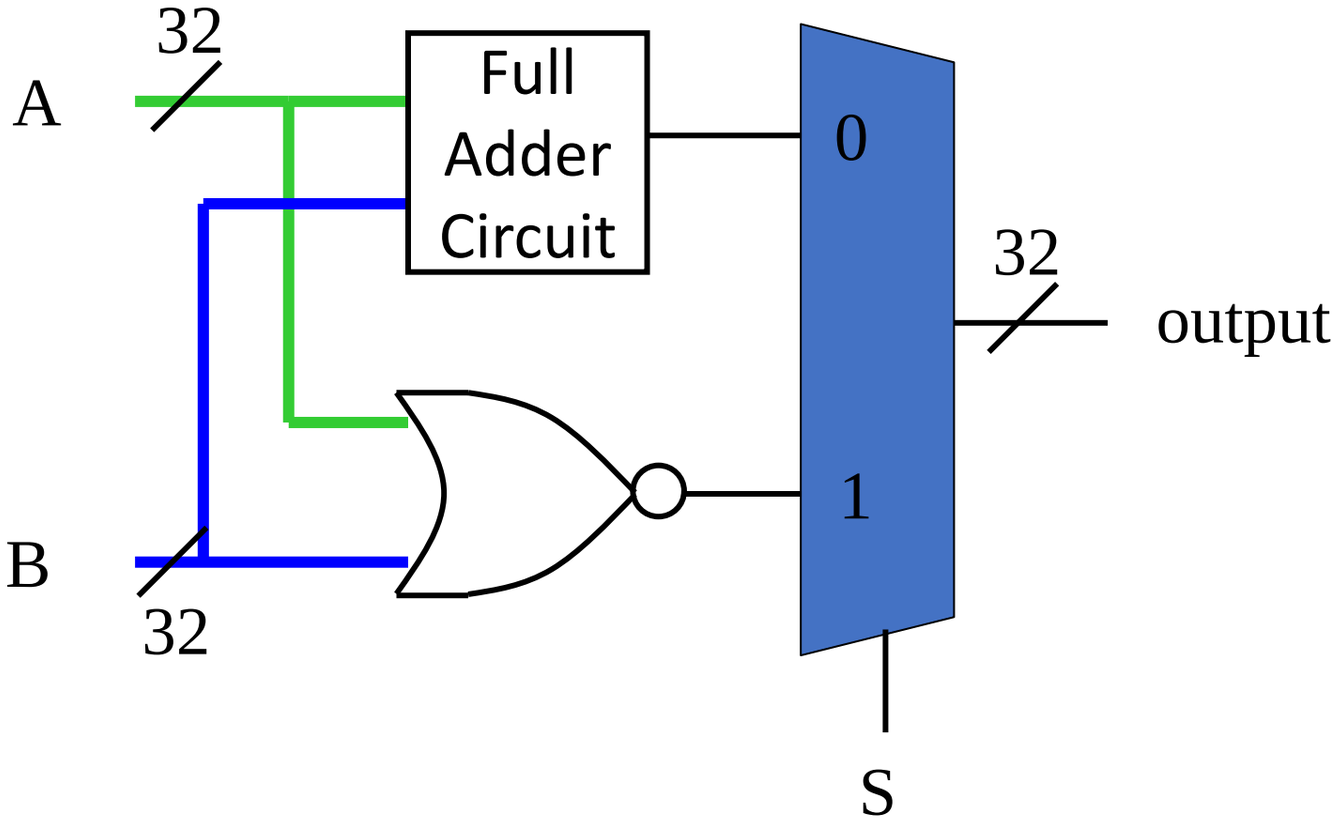
- Use the blocks we have learned about so far (full adder, NOR, mux) to build this circuit
  - Input A, 32 bits
  - Input B, 32 bits
  - Input S, 1 bit
  - Output, 32 bits
  - When S is low, the output is A+B, when S is high, the output is NOR(a,b)
- Hint: you can express multi-bit gates like this:





Exercise

- This is a basic ALU (Arithmetic Logic Unit)
- It is the heart of a computer processor!



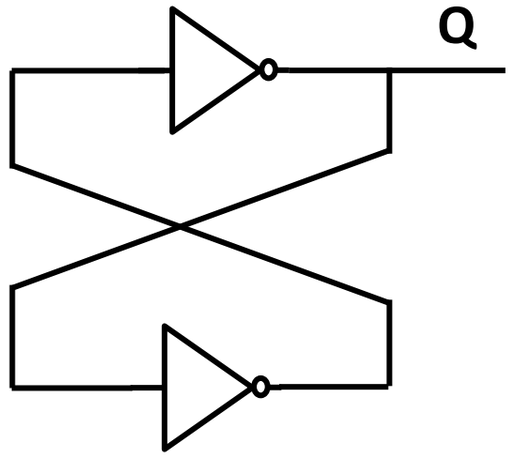
Sequential Logic

- Can we build a processor out of these combinational elements?
- How to build something like a program counter (PC)?
  - Increment it for every instruction… fine, use an adder
  - But only increment it once ready to move on to next instruction
  - That takes a finite amount of time… until then we need to "remember" the current value
- Combinational logic's output is determined from current input
  - But computers have "state" – they remember previous inputs and behave differently based on its history

Sequential Logic

- Examples of state
  - Registers
  - Memory
  - PC
- Sequential logic's output depends not only on current input, but also the current state
- This lecture will show you how to build sequential logic from gates
  - Key is feedback

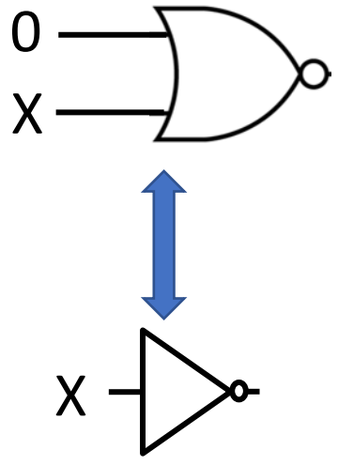
Using feedback to "remember"

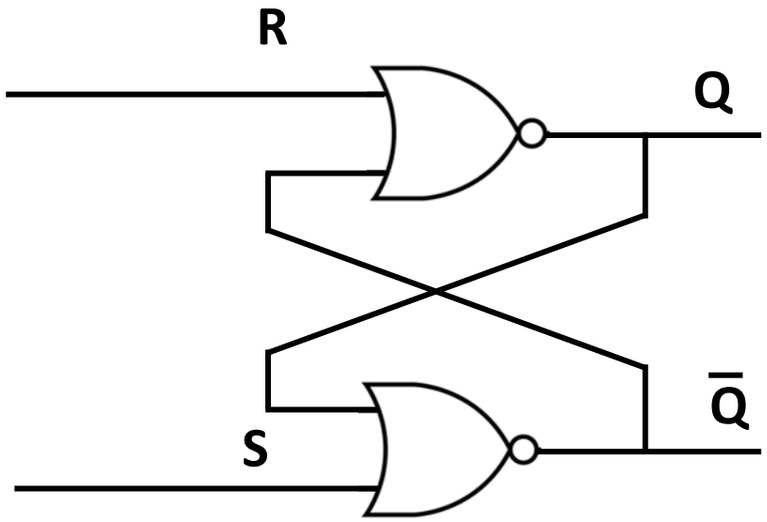


- This remembers its initial value!
- Very basic memory
- What's wrong with this, though?

Let's look at the following circuit

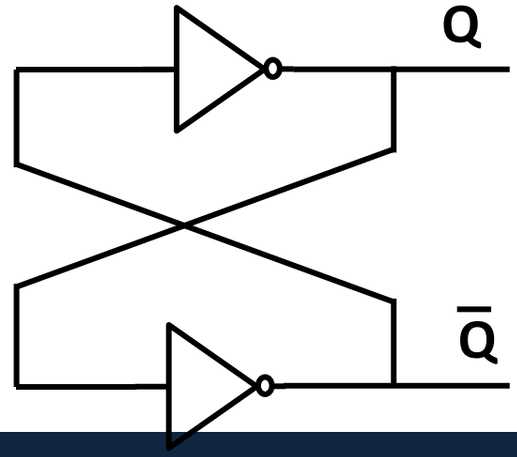
Useful identify:
NOR(0,X)=
NOT(X)



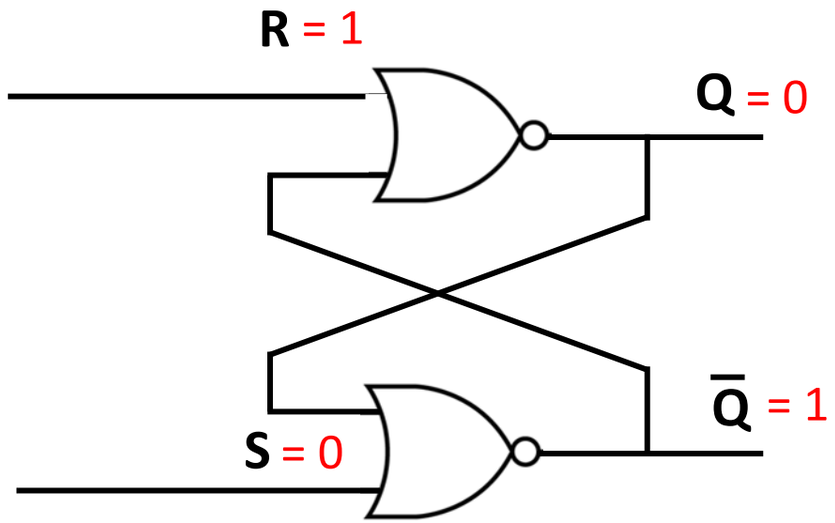


S	R	Q	Q̄
0	0	Q	Q̄

When R=S=0, turns into our memory element!



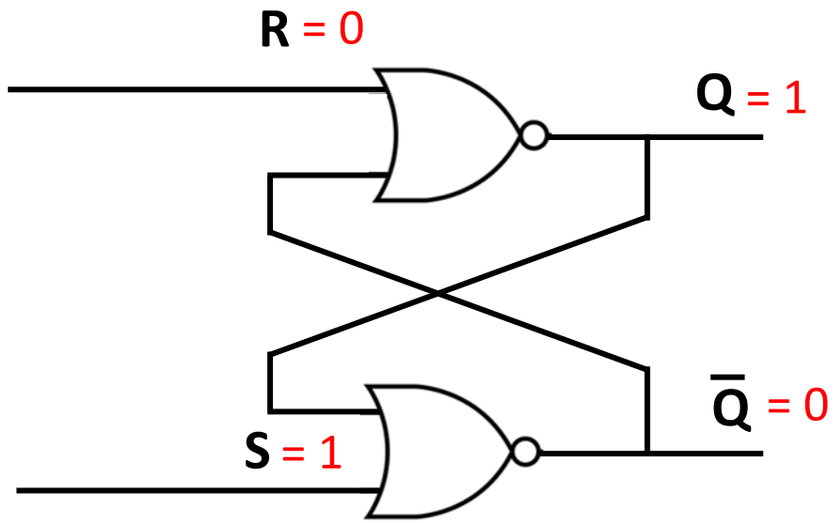
Let's look at the following circuit



S	R	Q	$\bar{Q}$
0	0	Q	Q
0	1	0	1

What is the value of Q if R is 1 and S is 0?

Let's look at the following circuit

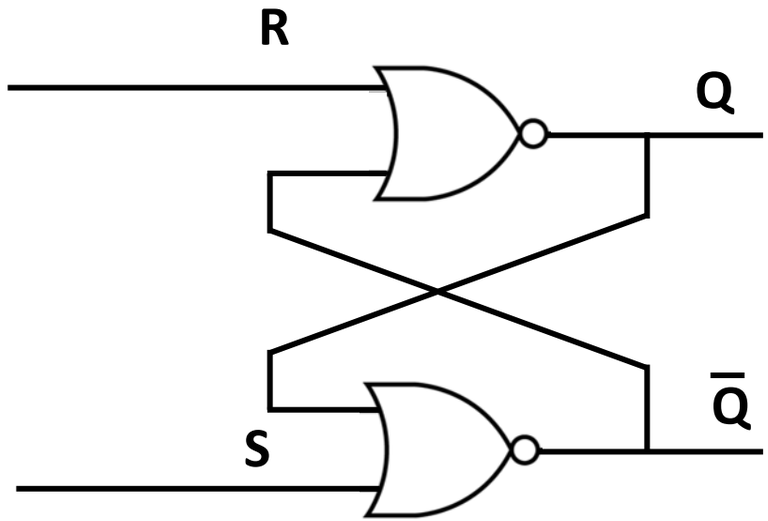


S	R	Q	$\bar{Q}$
0	0	Q	Q
0	1	0	1
1	0	1	0

What is the value of Q if R is 0 and S is 1?

SR Latch

- So this circuit (an SR latch):
  - "Sets" Q to 1 when S=1 R=0
  - "Resets" Q to 0 when S=0 R=1
  - "Latches" Q when S=0 R=0
  - What about when S=1 R=1 ?

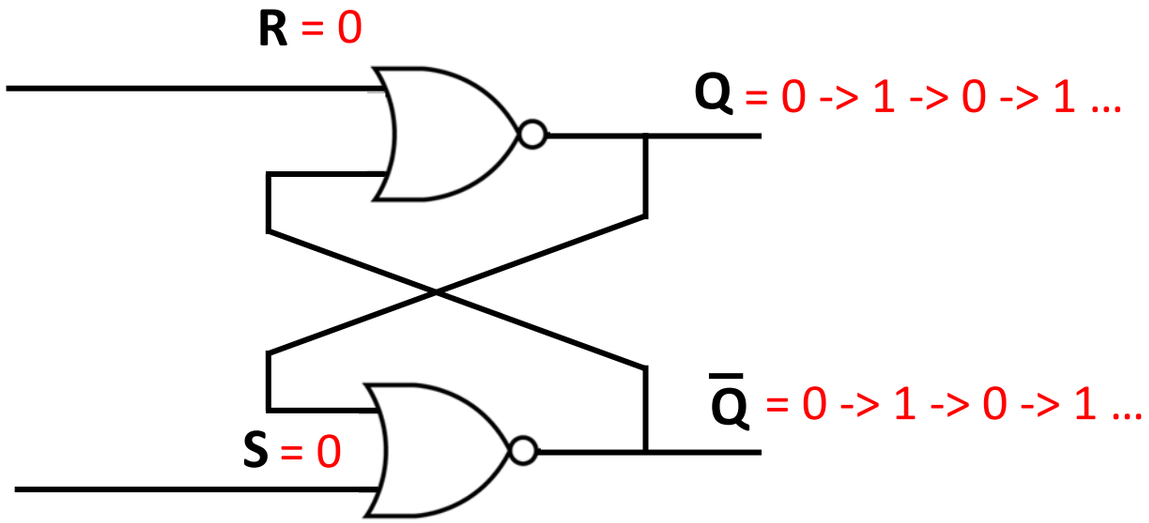


S	R	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	1	0
1	1	0	0

BAD! Why?

SR Latch – Undefined behavior

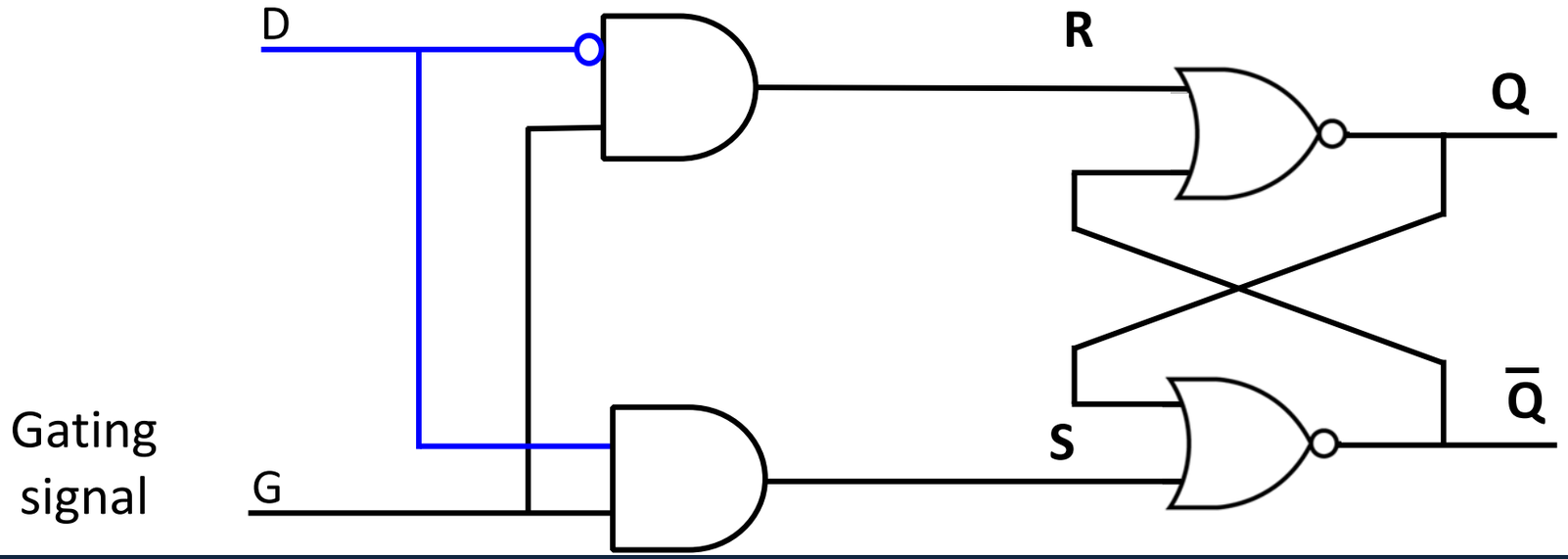
- If S=1, R=1, then Q and it's inverse are both 0
- If inputs then change to S=0, R=0, we get this circuit



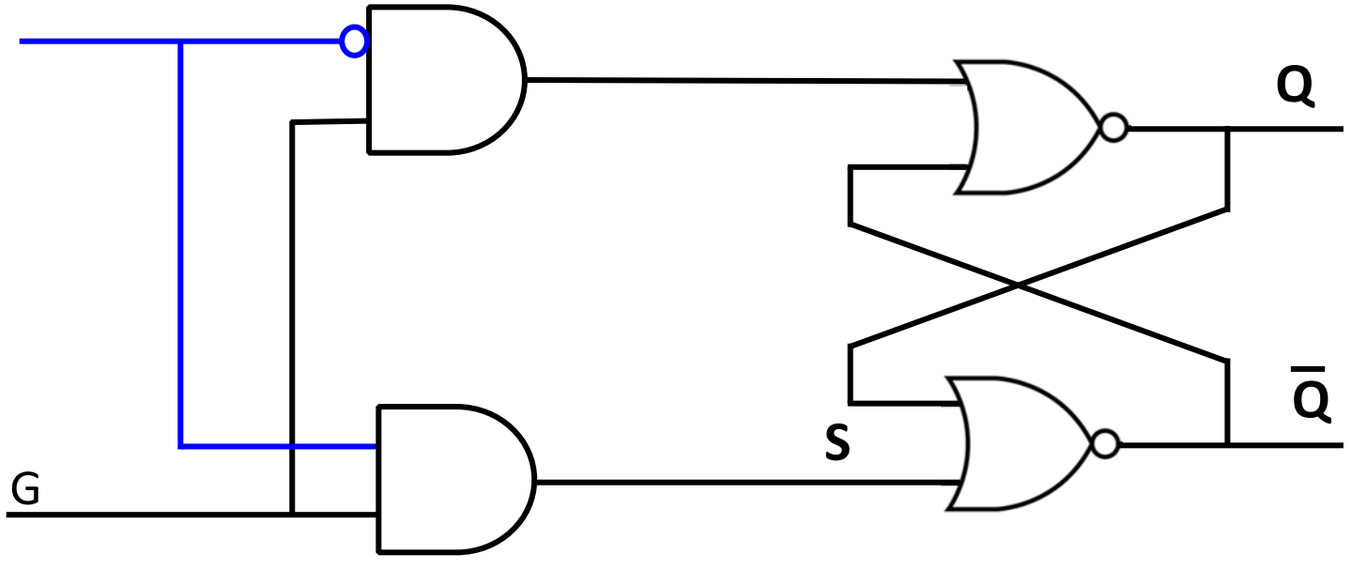
- This is unstable! Output rapidly oscillates between 0 and 1

Improving SR Latch

- SR Latch works great at saving a bit of data…
  - Unless S=R=1, even for a fraction of a second
- Idea: let's prevent that from happening by adding AND gates in front of each input
  - Impossible for R and S to both be 1



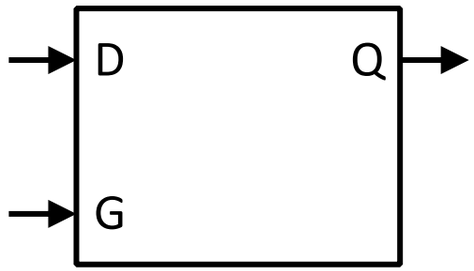
Transparent D Latch



D	G	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	Q	$\bar{Q}$
1	1	1	0

Next state is set

Set state is retained when gate is low



No invalid states ☺

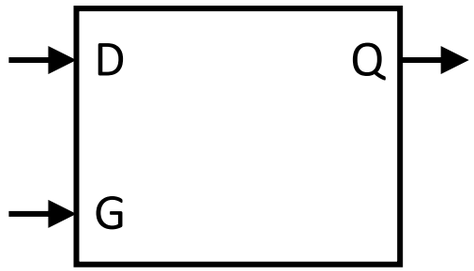
Transparent D Latch

- When G ("gate") is high, Q=D (the latch is "transparent")
- When G is low, Q "latches" to the value of D at that instant and remembers, even if D changes later

Next state is set

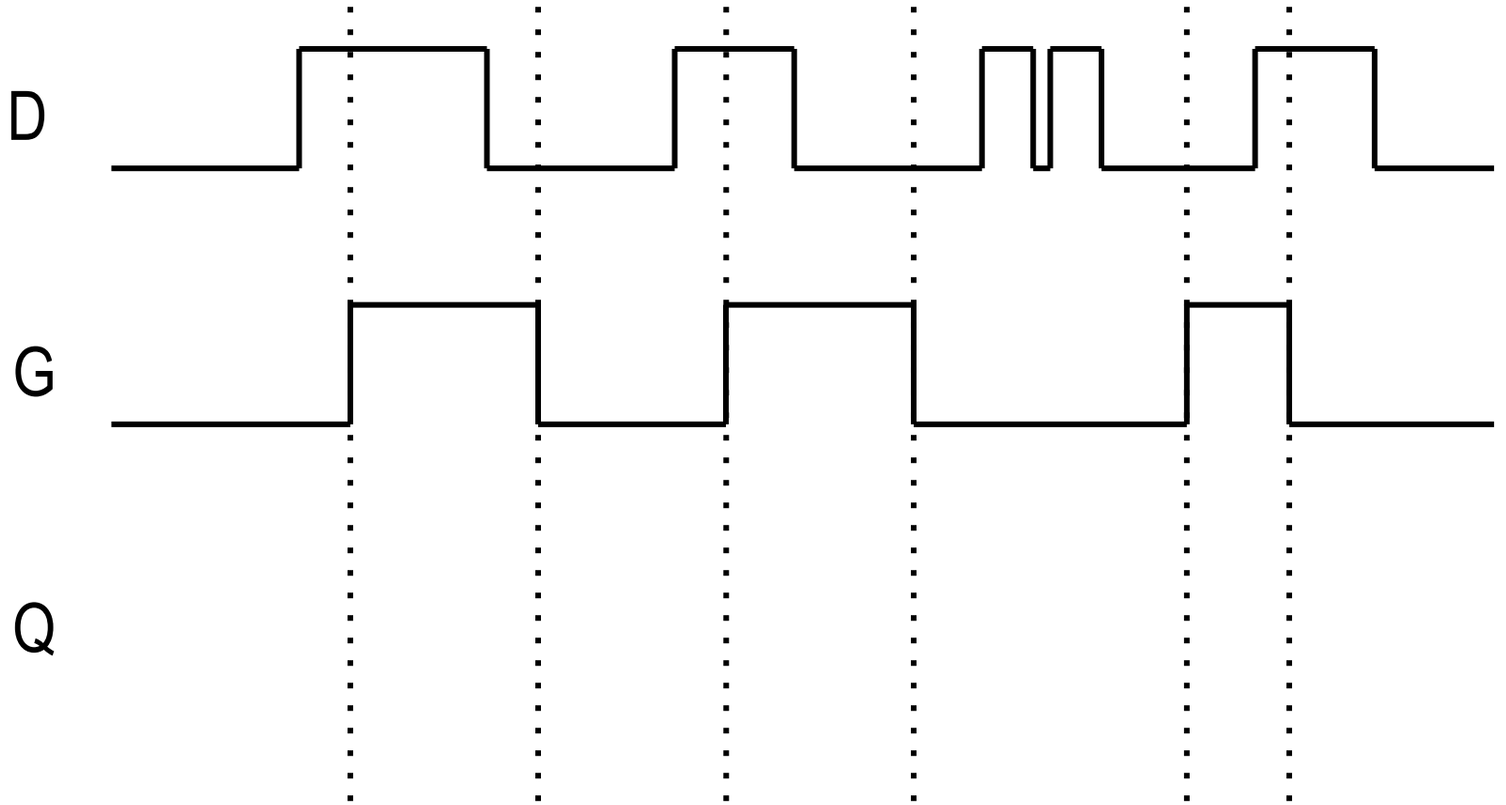
D	G	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	Q	$\bar{Q}$
1	1	1	0

Set state is retained when gate is low

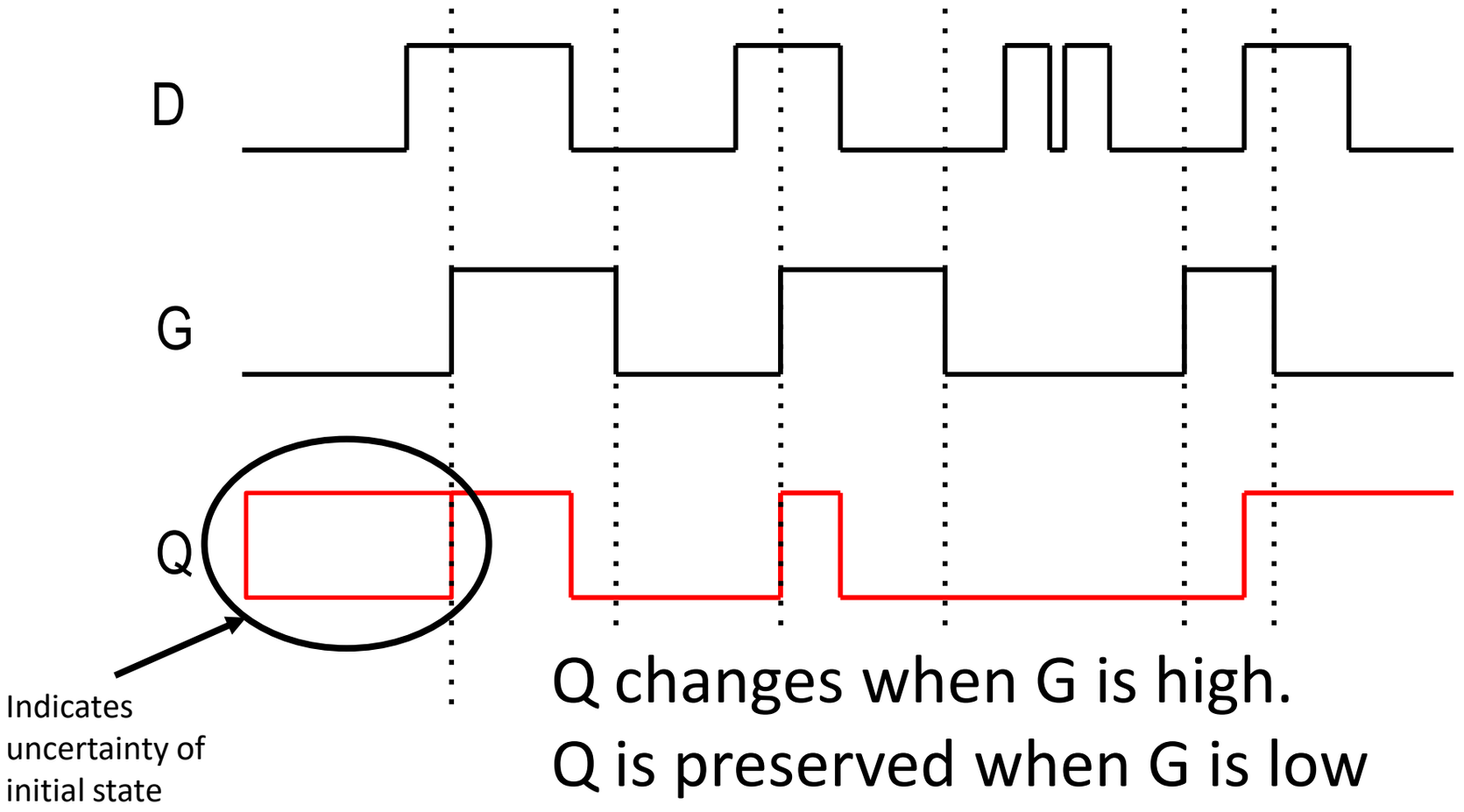


No invalid states ☺

D-Latch Timing Diagram

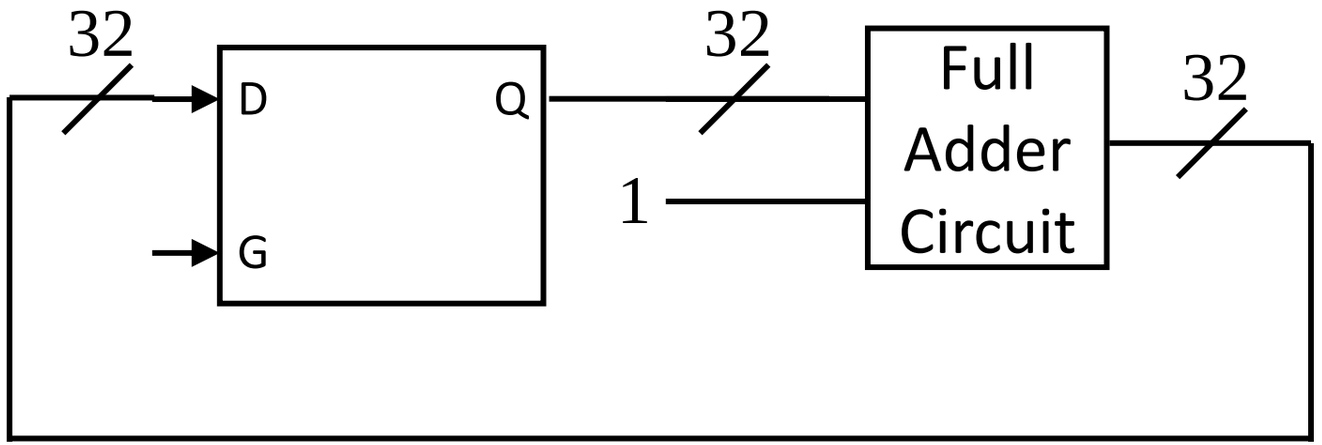


D-Latch Timing Diagram



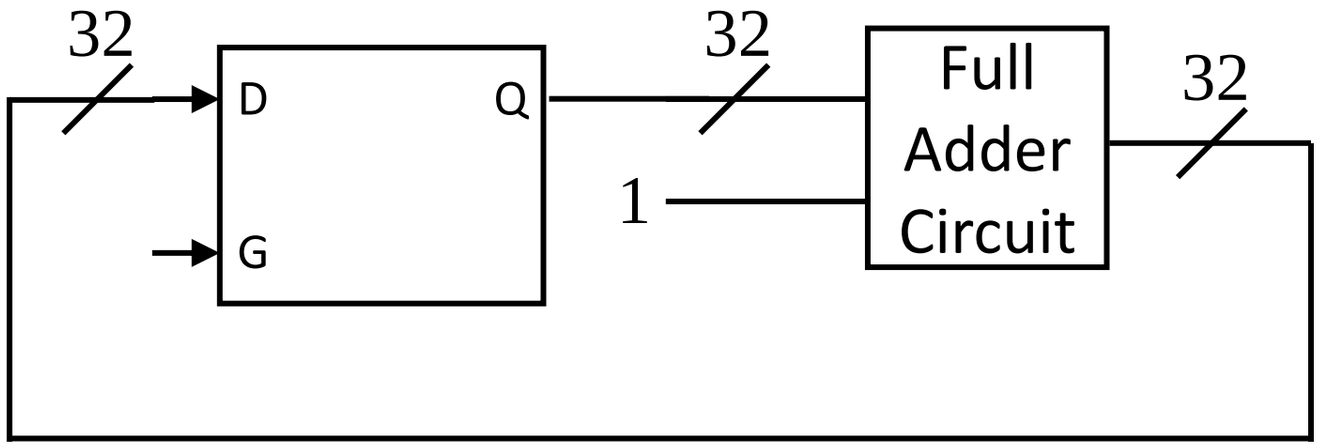
Is D-Latch Sufficient?

- Can we use D-latches to build our PC logic?
- Idea:
  - Use 32 latches to hold current PC, send output Q to memory
  - Also pass output Q into 32-bit adder to increment by 1 (for word-addressable system)
  - Wrap sum around back into D as "next PC"
  - Once ready to execute next instruction, set G high to update



Shortcoming of D-Latch

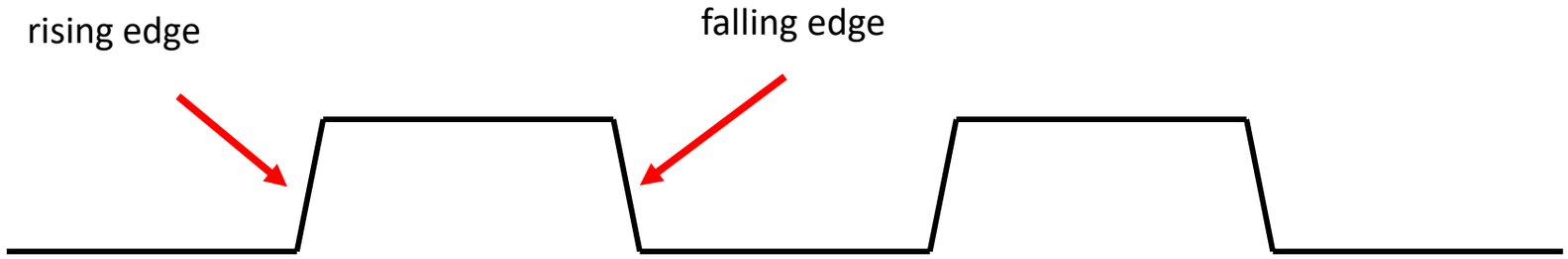
- Problem: G must be set very precisely
  - Set high for too short: latch doesn't have enough time for feedback to stabilize
  - Set high for too long: Signal may propagate round twice and increment PC by 2 (or more)
- Challenging to design circuits with exactly the right durations



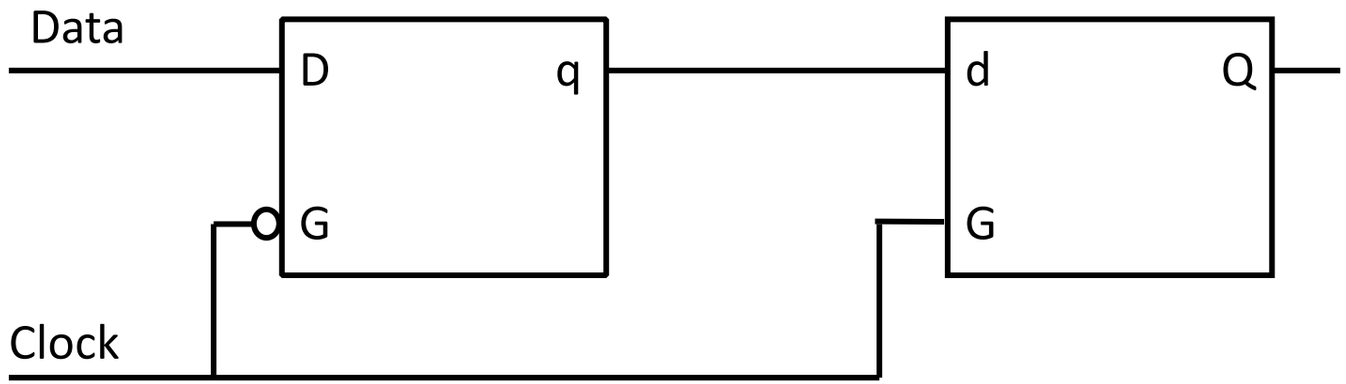
Not just a problem for PC, much of our processor will involve logic like this

Adding a Clock to the Mix

- We can solve this if we introduce a **clock**
  - Alternating signal that switches between 0 and 1 states at a fixed frequency (e.g., 1 GHz)
  - Only store the value the **instant** the clock changes (i.e. the **edge**)



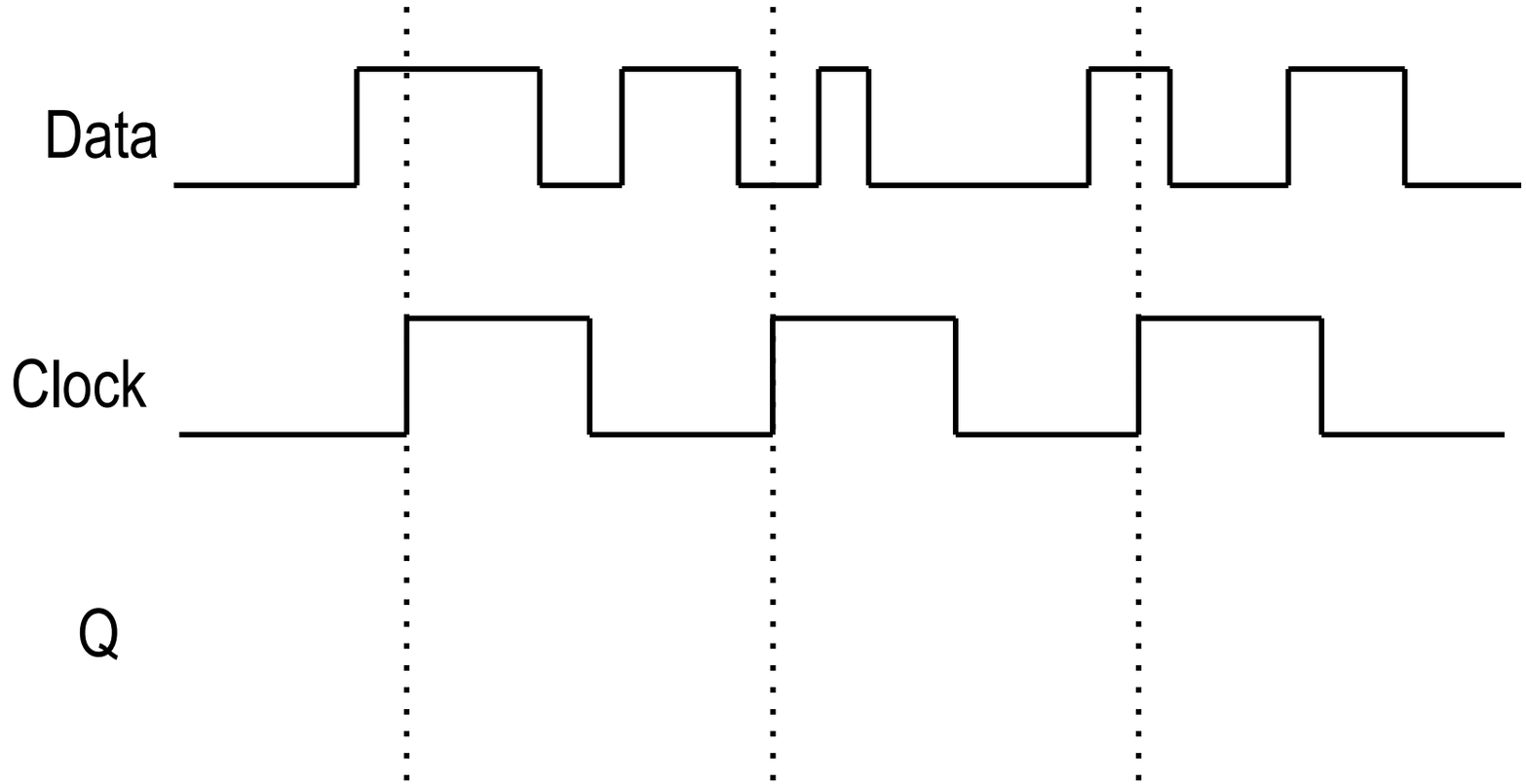
Adding a Clock to the Mix



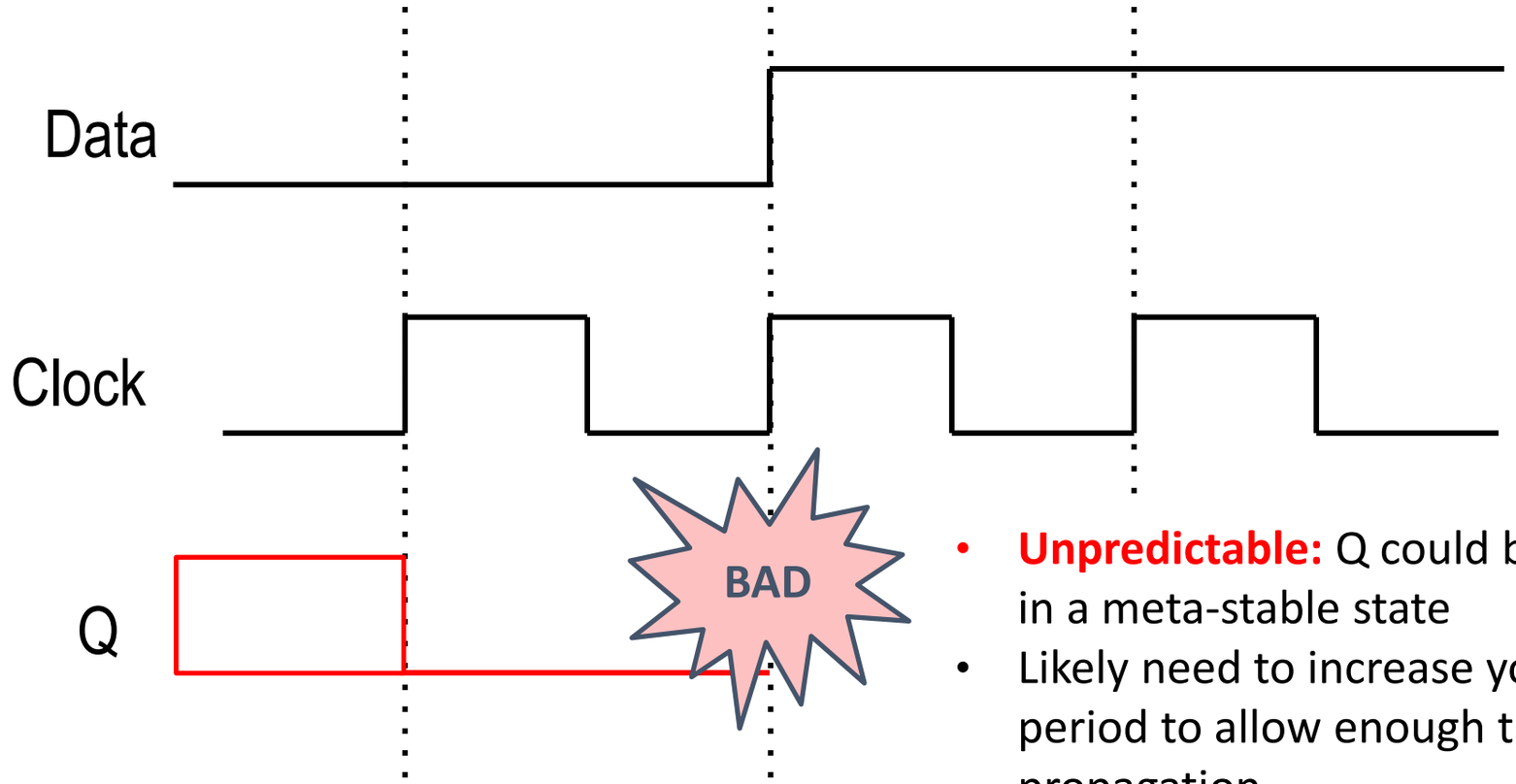
We won't discuss it further here, but this circuit sets Q=D **ONLY** when clock transitions from 0 -> 1

Intuitively, the design works by inverting the Gate signals, so only one passes at a time (like a double set of sliding doors)

D Flip-Flop Timing Diagram



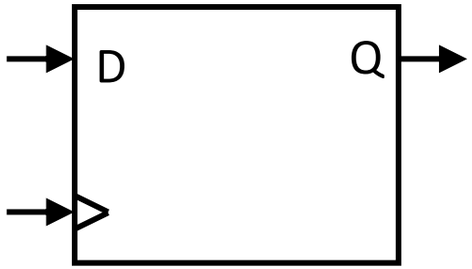
What happens if Data changes on clock edge?



- **Unpredictable:** Q could be high, low or in a meta-stable state
- Likely need to increase your clock period to allow enough time for signal propagation

Latches vs Flip-flops

D Latch



D Flip-flop

Enabled D Flip-flop (only updates on clock edge if 'en' is high)

Next Time

- Finite State Machines
- Introduce first processor implementation