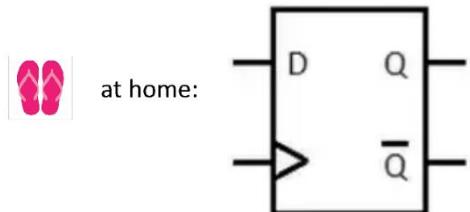


EECS 370 - Lecture 10

Finite State Machine

Me: Mom, can I have  ?

Mom: No we have  at home



Live Poll + Q&A: [slido.com #eeecs370](https://slido.com/#eeecs370)

Poll and Q&A Link

Announcements

- Project 2 posted
 - First part due next Thursday
- First exam 2 weeks from today
 - Covers up through next week's lectures ("Multi-cycle" is last topic)
 - Recommend reviewing earlier material now
 - Rework through previous lab problems
 - Sample exams available on website
 - AI tools have decent hit rate at generating simple questions



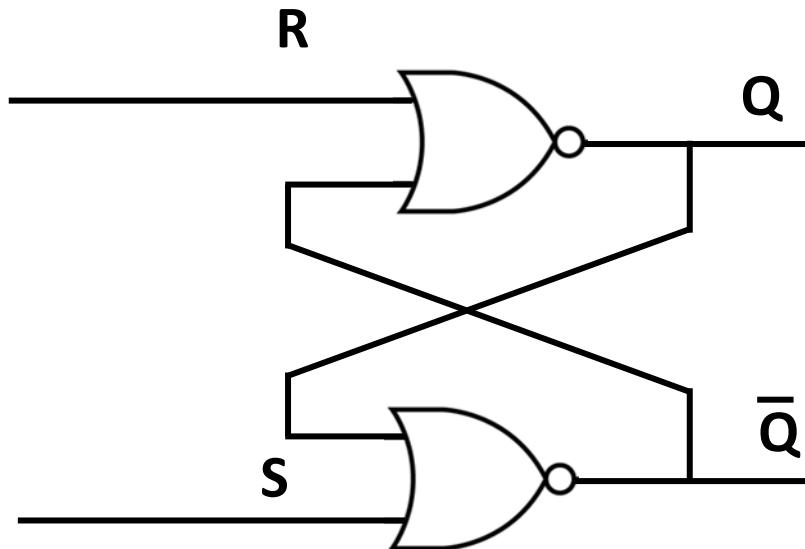
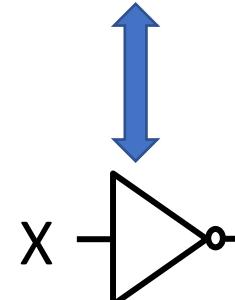
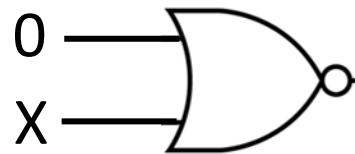
My OH

- Reminder: I have a new office
 - My OH will be in 4812 BBB going forward
- Today's OH only:
 - Pushed from 1:30-2:30 to 3-4
 - As always, check out the Gcal on website for up-to-date info



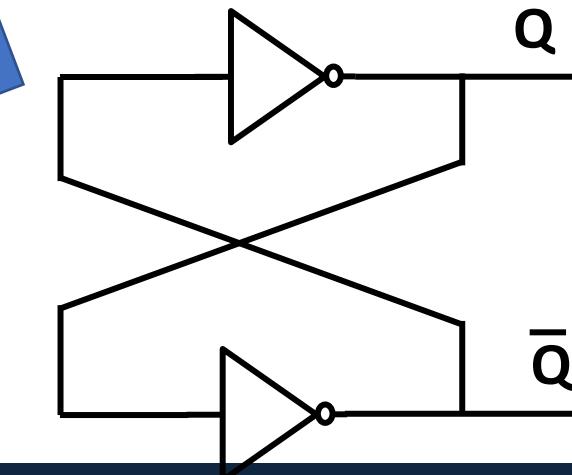
Reminder: “Latch” as a memory element

Useful identity:
 $\text{NOR}(0, X) = \text{NOT}(X)$

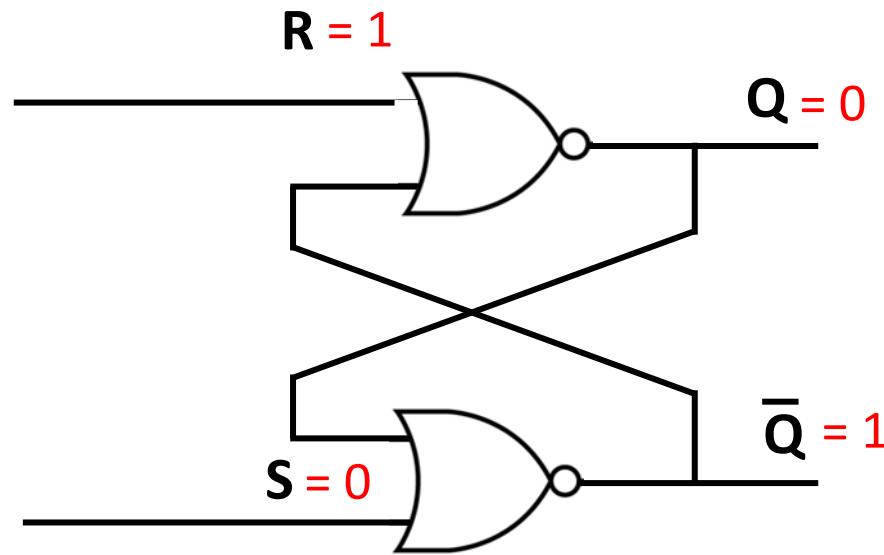


When $R=S=0$, turns
into our memory
element!

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



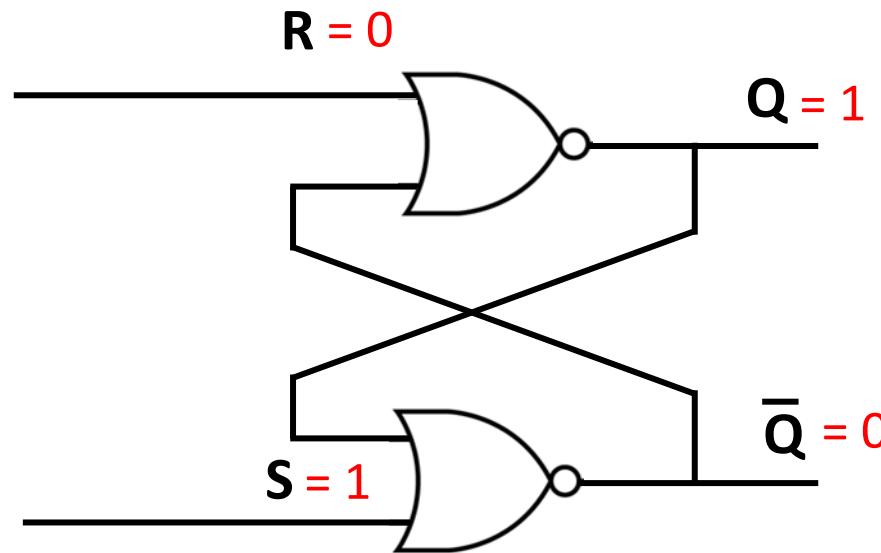
Let's look at the following circuit



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

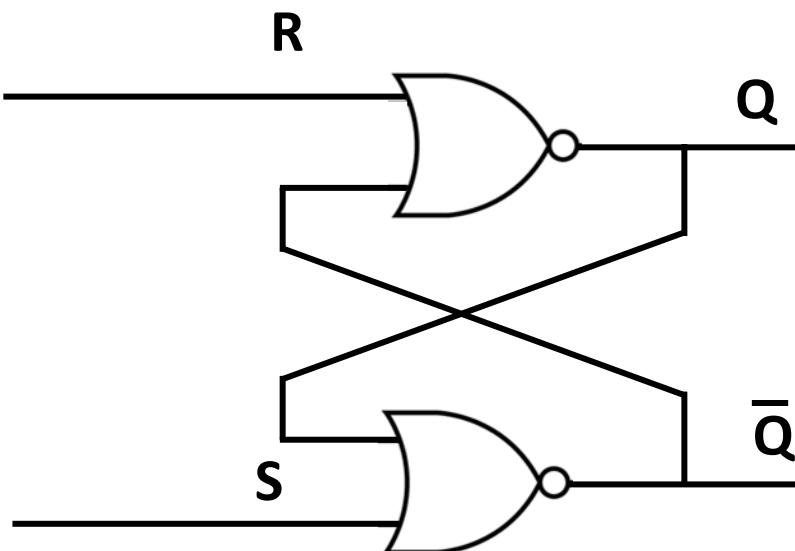


S	R	Q	\bar{Q}
0	0	Q	Q
0	1	0	1
1	0	1	0

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 $S=1 R=1$?



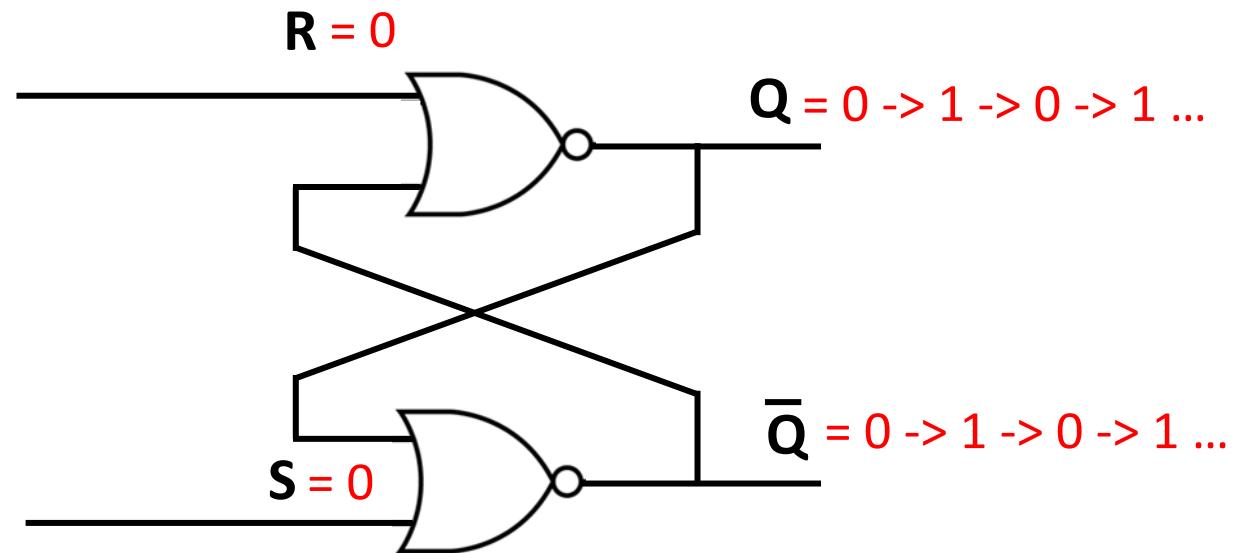
S	R	Q	\bar{Q}
0	0	Q	\bar{Q}
0	1	0	1
1	0	1	0
1	1	0	0

BAD! Why?



SR Latch – Undefined behavior

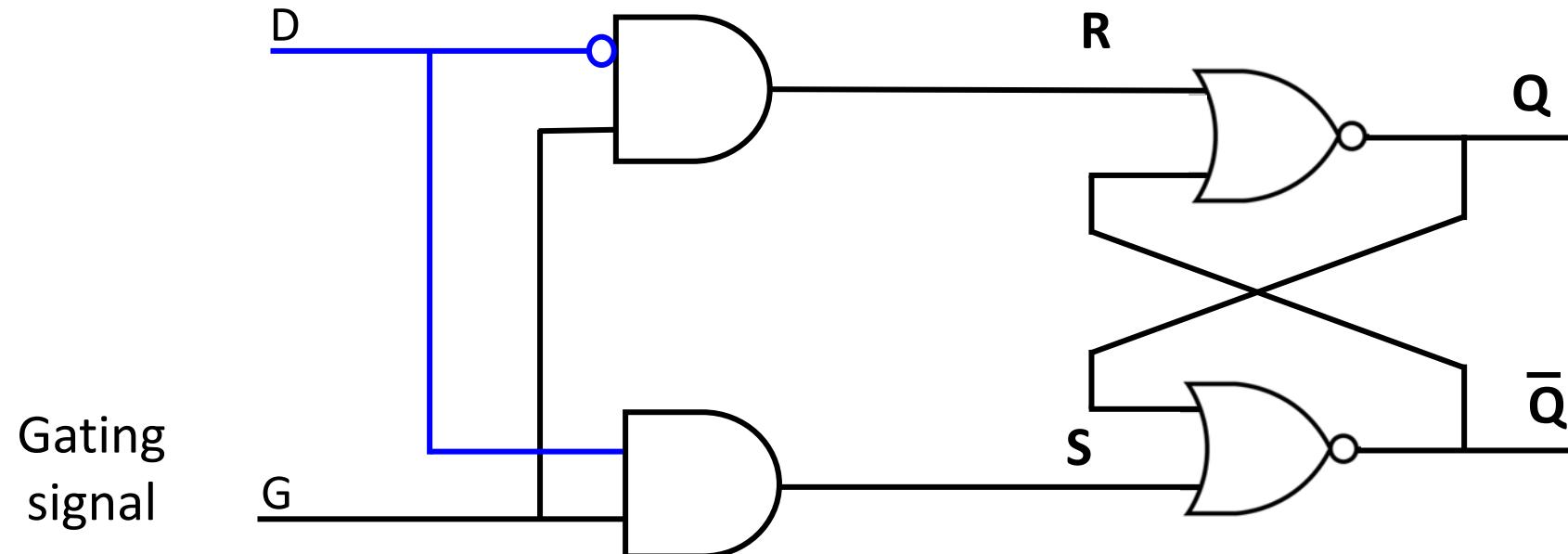
- If $S=1$, $R=1$, then Q and it's inverse are both 0
- If inputs then change to $S=0$, $R=0$, we get this circuit



- 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

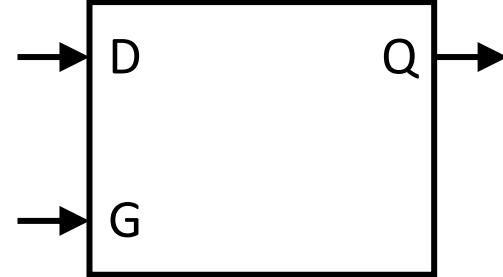
- When G ("gate") is high, Q=D (the latch is "transparent")
- When G is low, Q "latches" to the value of D at that instant and remembers, even if D changes later

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

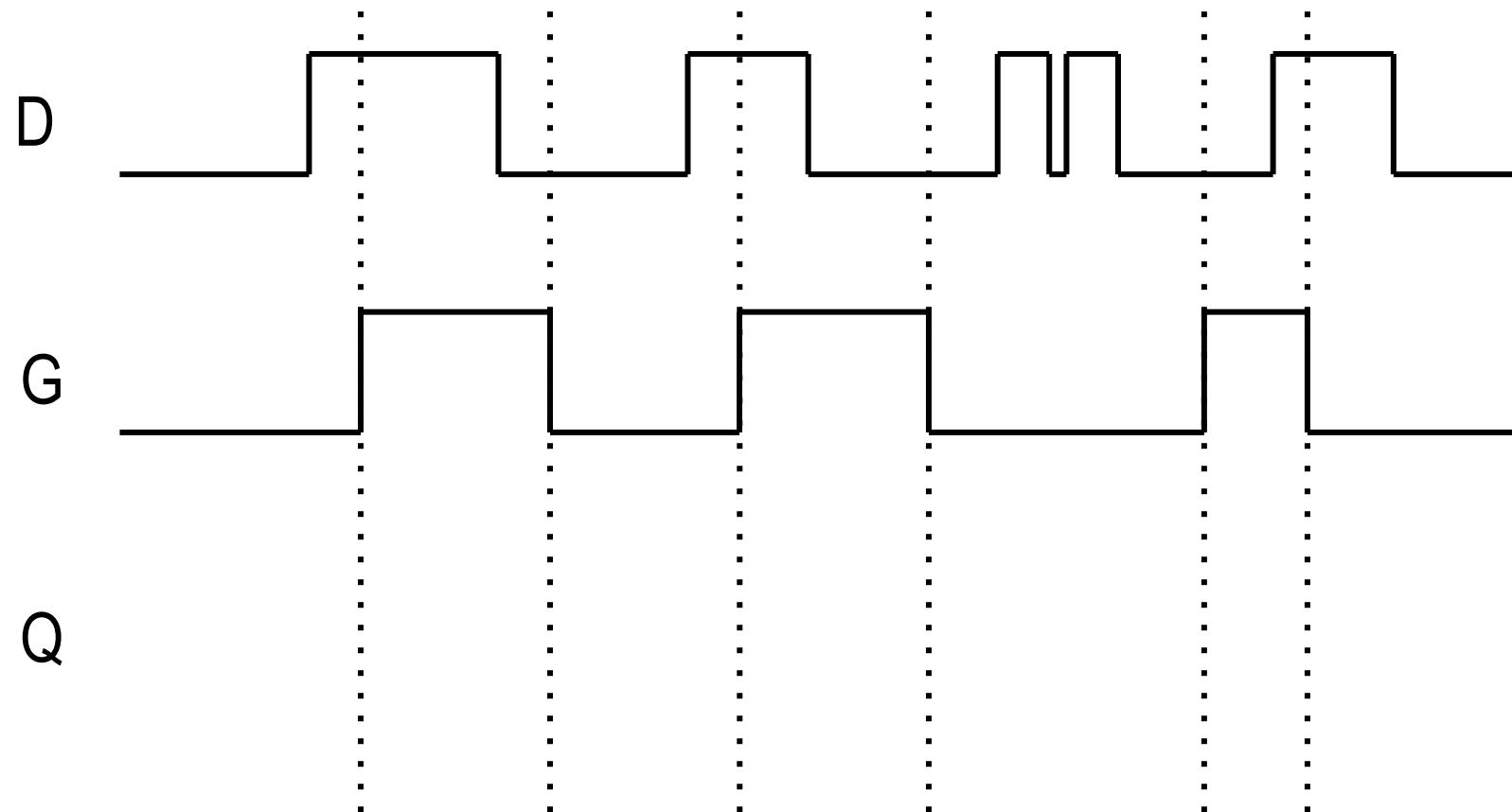
Set state is retained when gate is low

No invalid states 😊

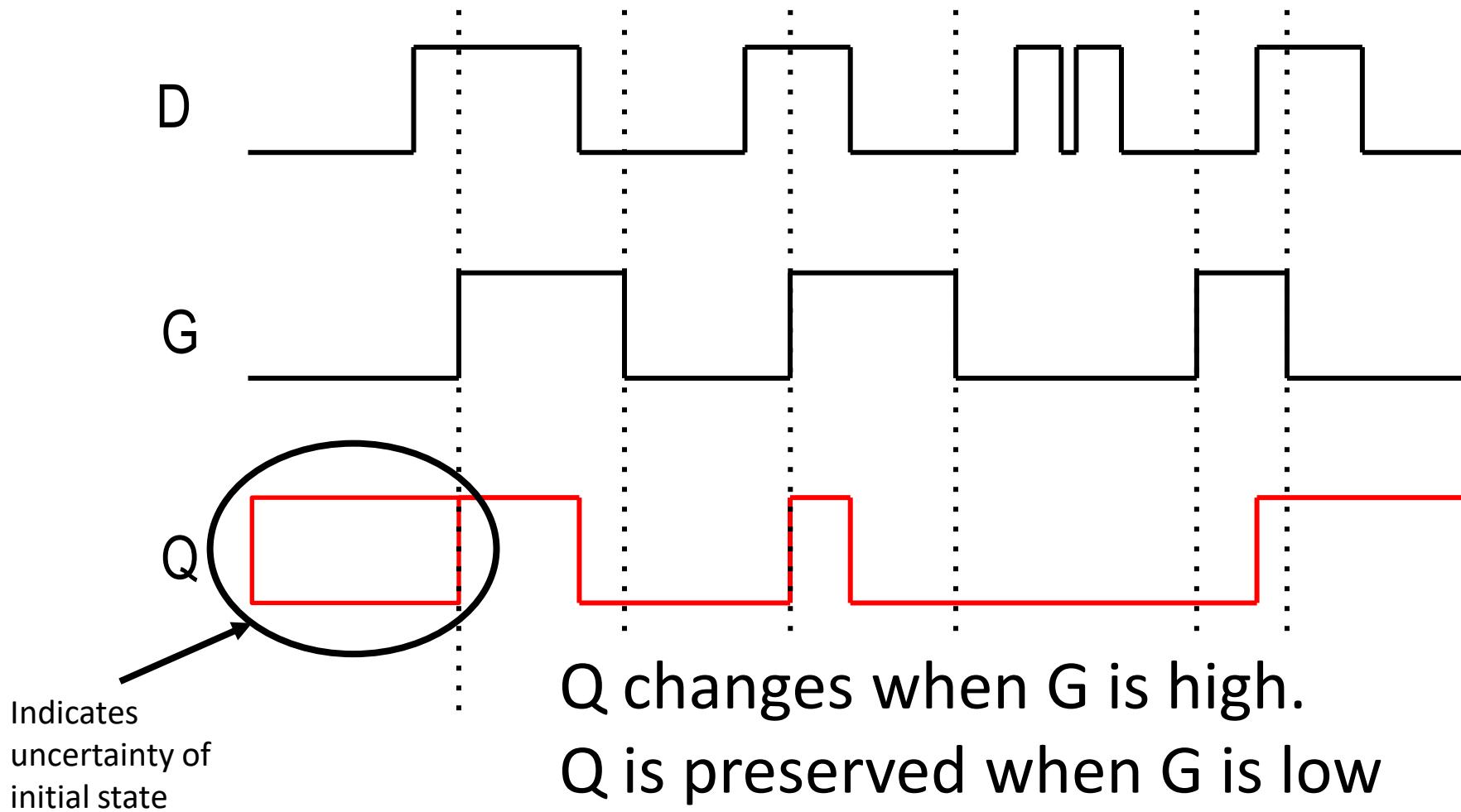


```
graph LR; D((D)) --> Box[ ]; G((G)) --> Box[ ]; Box[ ] --> Q((Q))
```

D-Latch Timing Diagram

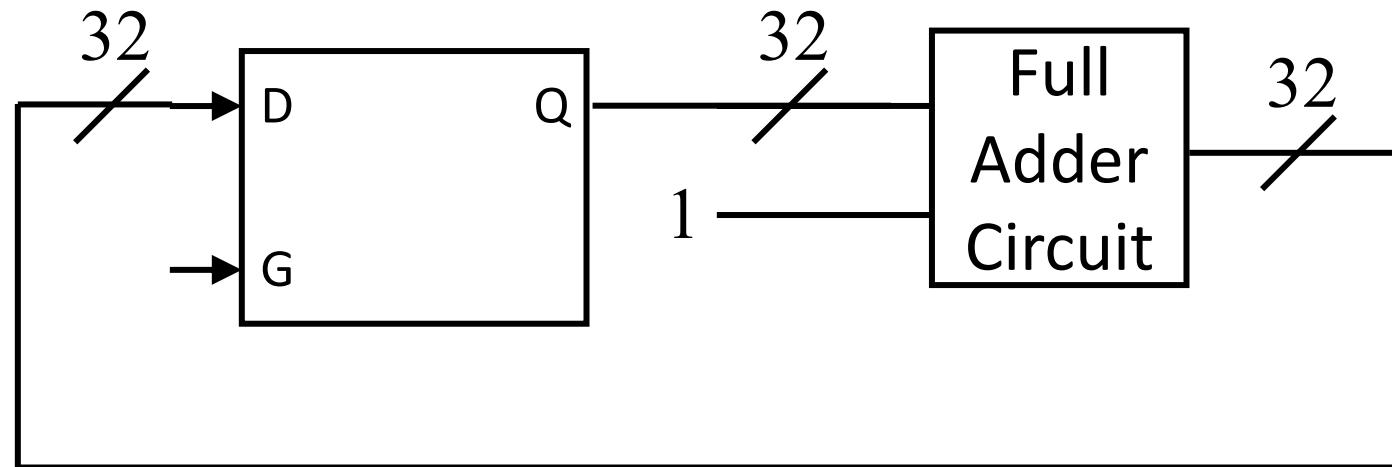


D-Latch Timing Diagram



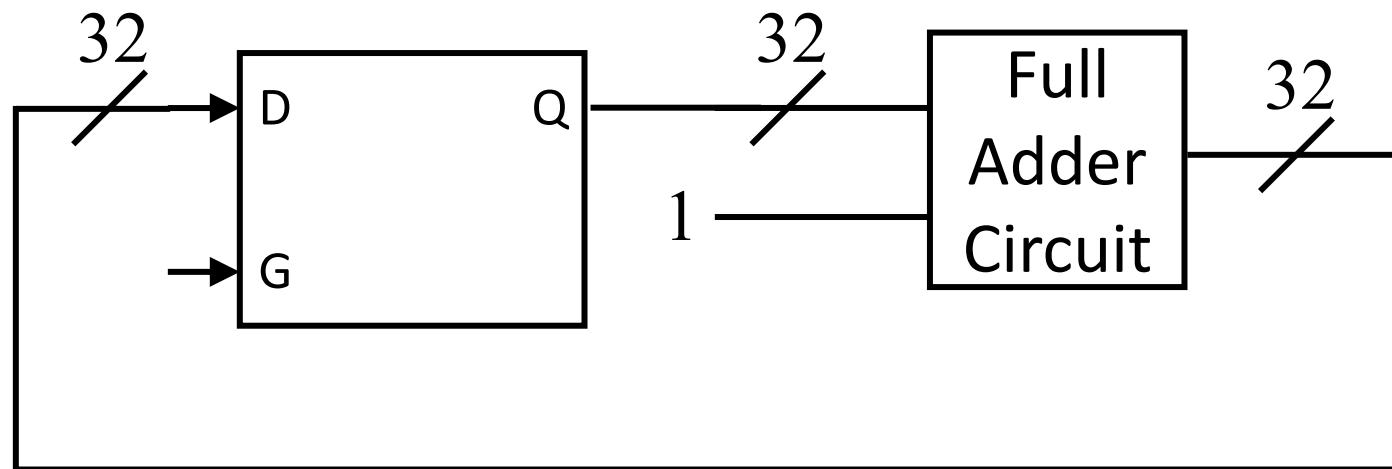
Is D-Latch Sufficient?

- 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

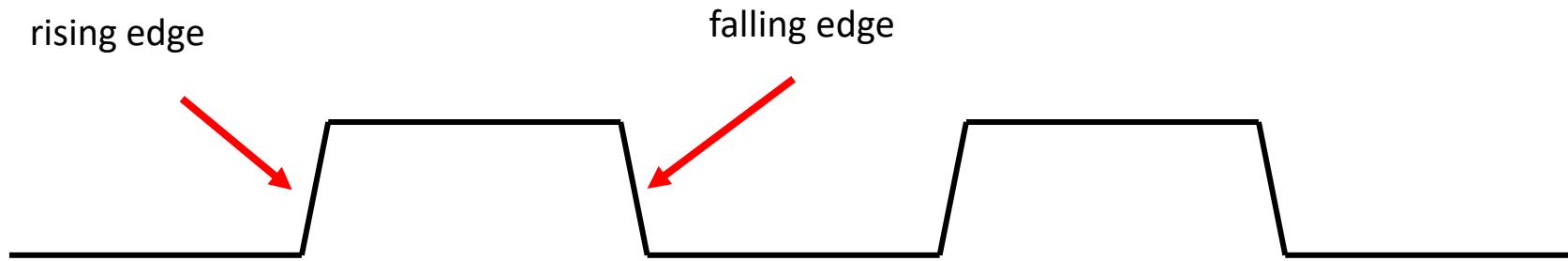
- Problem: G must be set very precisely
 - Set high for too short: latch doesn't have enough time for feedback to stabilize
 - Set high for too long: Signal may propagate round twice and increment PC by 2 (or more)
- Challenging to design circuits with exactly the right durations



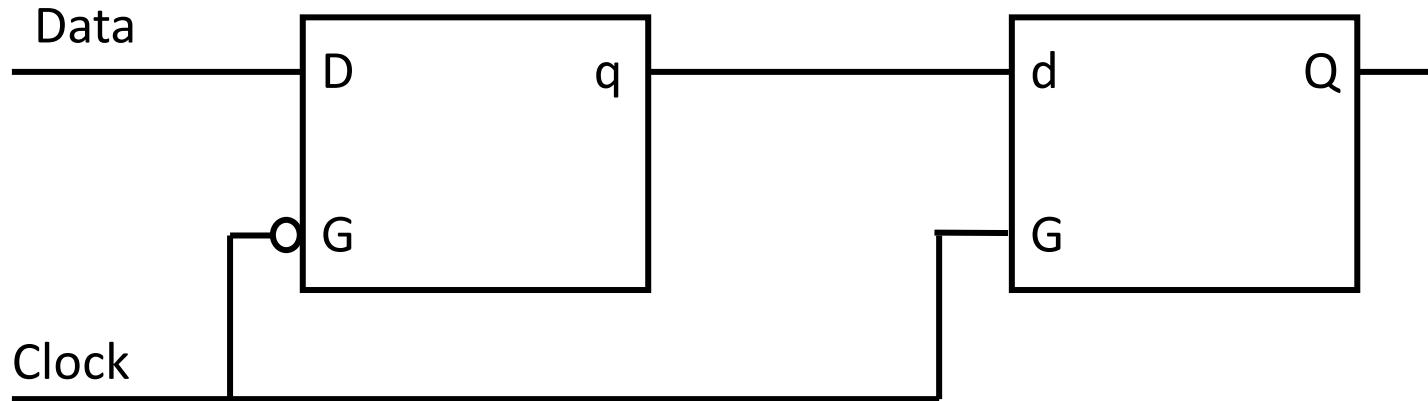
Not just a problem
for PC, much of our
processor will
involve logic like
this

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

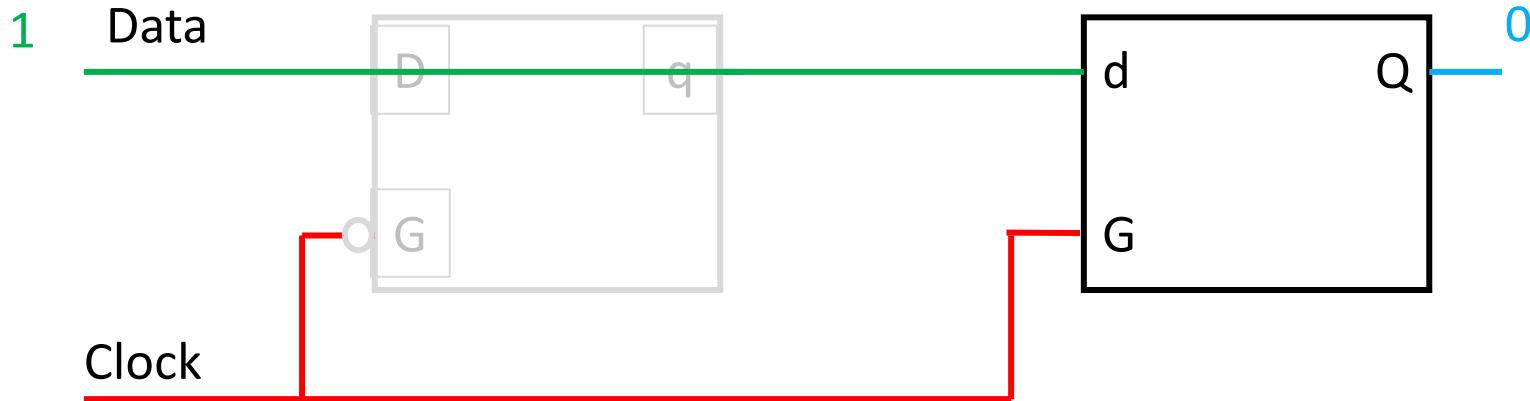


We won't discuss it further here, but this circuit sets $Q=D$ **ONLY** when clock transitions from $0 \rightarrow 1$

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

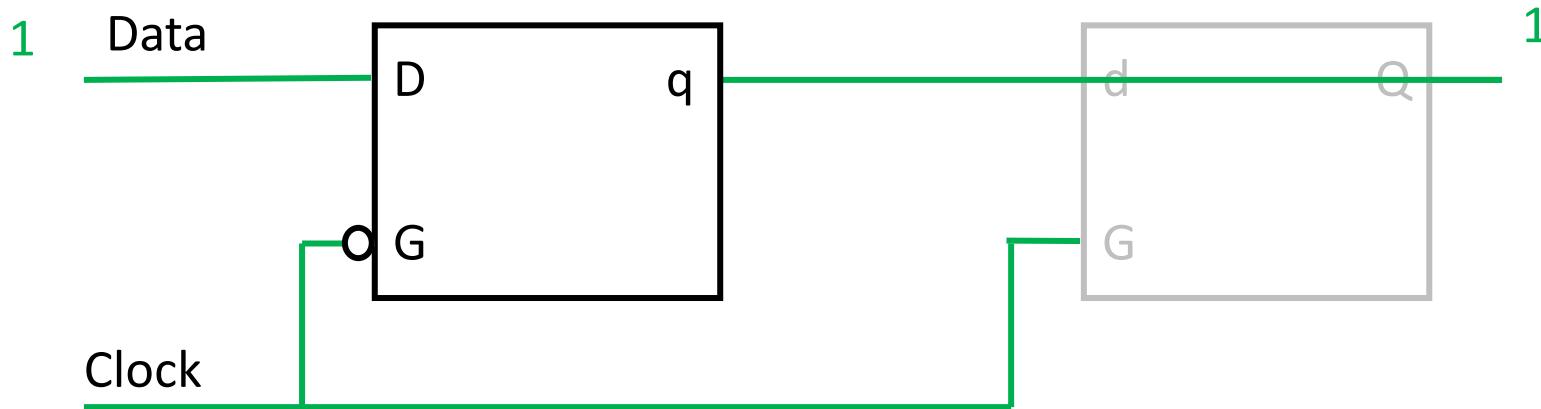


Extra Slides



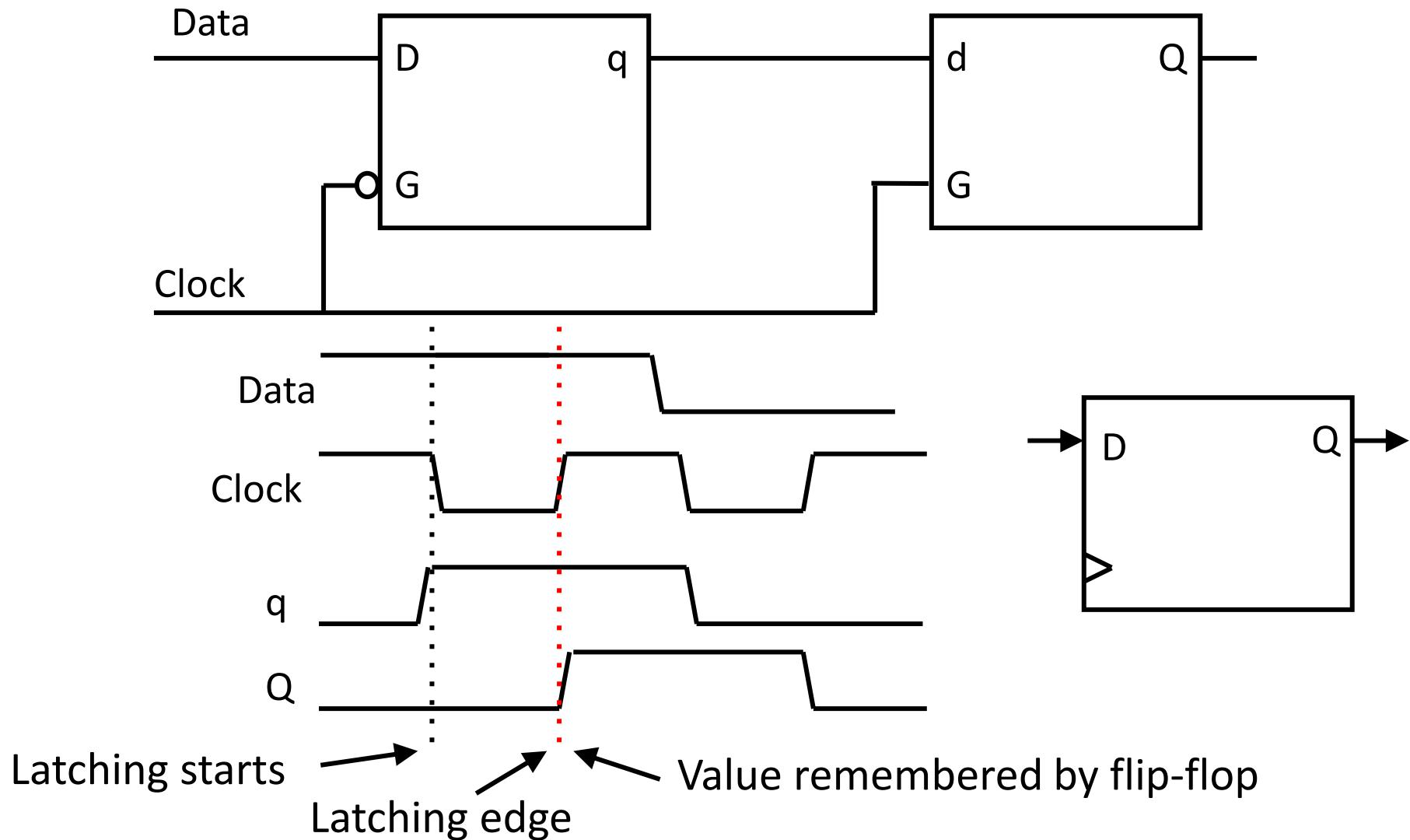
- When "Clock" is low, first latch is transparent
- But Q is latched to previous value of q (in this example, "0")

Extra Slides

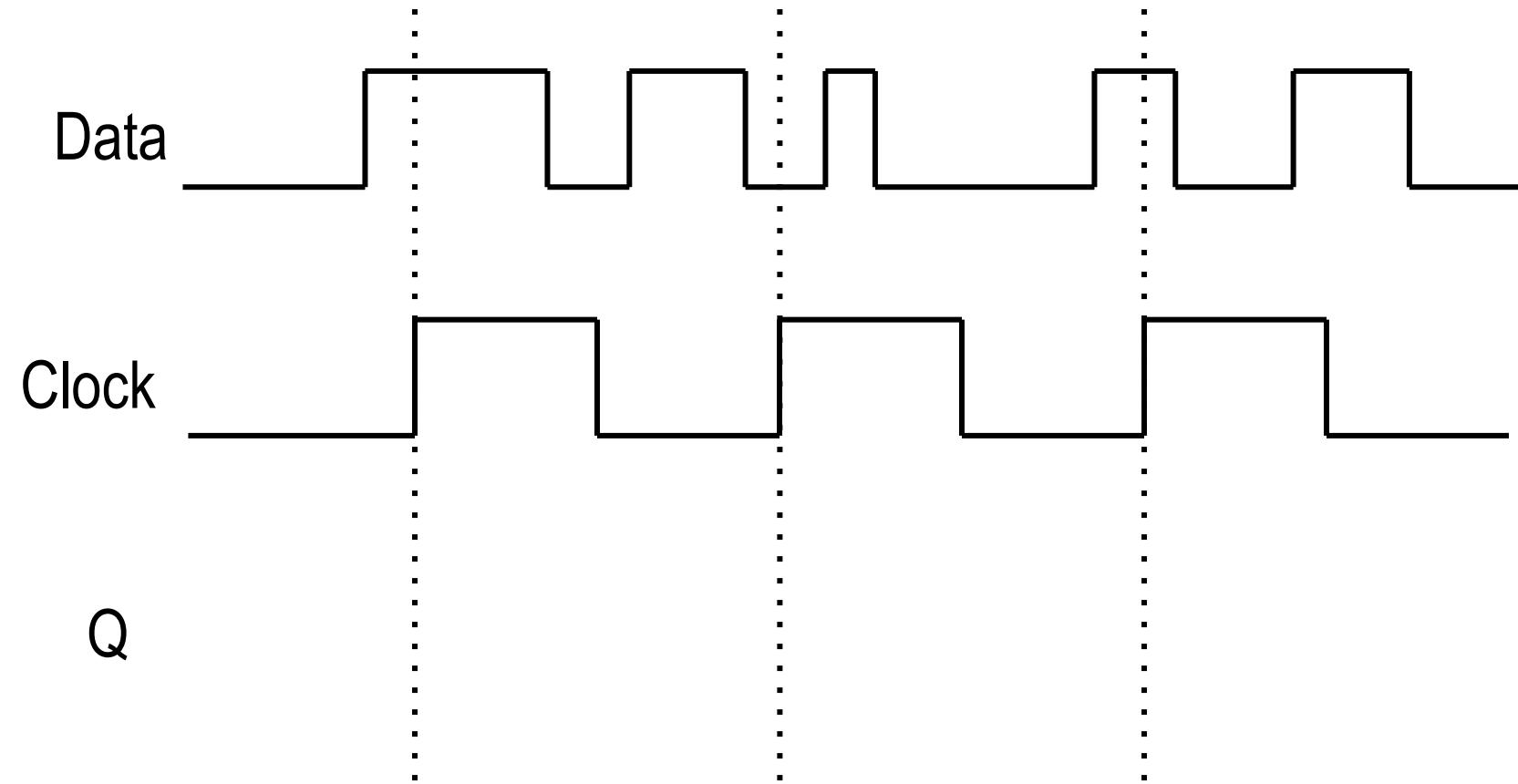


- When "Clock" is high, q latches to D's value
- Second latch is transparent
- Q now holds value of D from the instant Clock went high
- Won't change again until the next instant clock goes from low to high

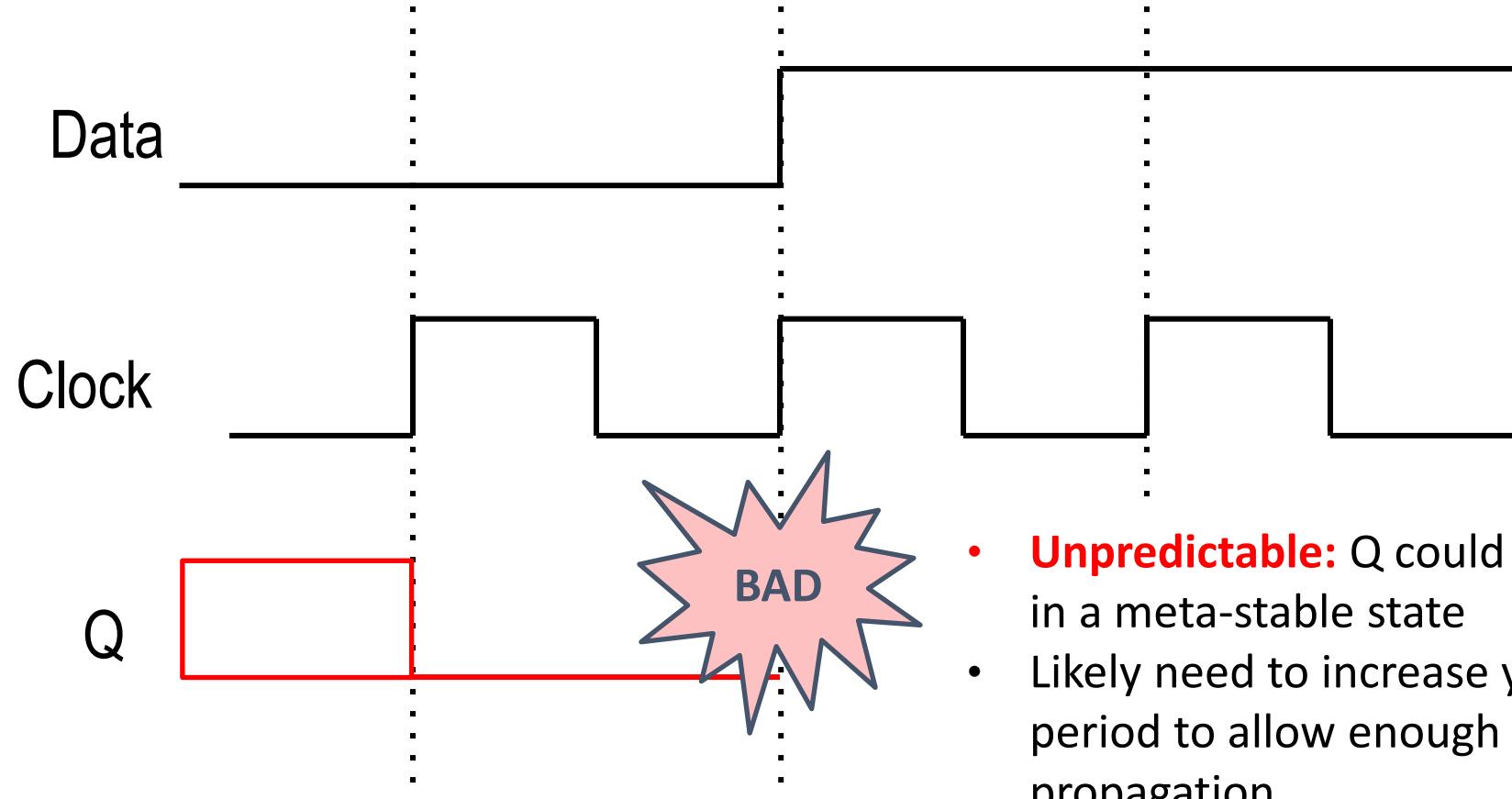
Extra Slides



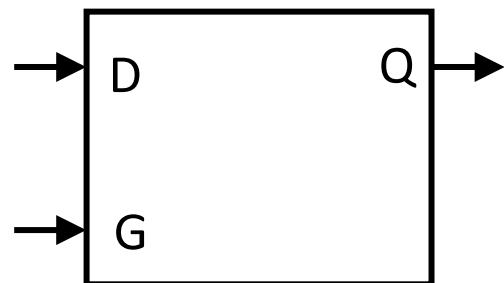
D Flip-Flop Timing Diagram



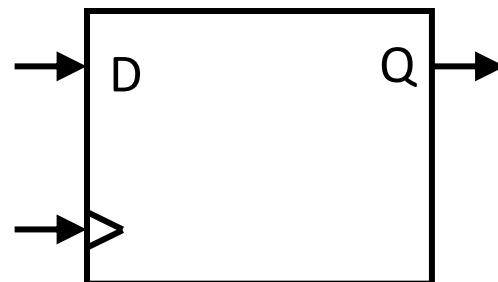
What happens if Data changes on clock edge?



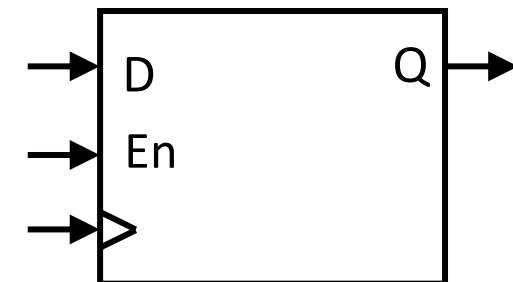
Latches vs Flip-flops



D Latch



D Flip-flop



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

Agenda

- **FSM Implementation**
- ROMs
- Making our FSM more efficient
- Single Cycle Processor Design Overview
- Supporting each instruction
 - ADD / NOR
 - LW / SW
 - BEQ
 - JALR



Finite State Machines

Note: This is very similar to Finite State Automata (FSA) from 376, but with a few differences (the input never ends, and the FSM always outputs something)

- So far we can do two things with gates:
 1. Combinational Logic: implement Boolean expressions
 - Adder, MUX, Decoder, logical operations etc
 2. Sequential Logic: store state
 - Latch, Flip-Flops
- How do we combine them to do something interesting?
 - Let's take a look at implementing the logic needed for a vending machine
 - Discrete states needed: remember how much money was input
 - Store sequentially
 - Transitions between states: money inserted, drink selected, etc
 - Calculate combinationally or with a control ROM (*more on this later*)

Input and Output

- Inputs:
 - Coin trigger
 - Refund button
 - 10 drink selectors
 - 10 pressure sensors
 - Detect if there are still drinks left
- Outputs:
 - 10 drink release latches
 - Coin refund latch



Operation of Machine

- Accepts quarters only
 - All drinks are \$0.75
 - Once we get the money, a drink can be selected
 - If they want a refund, release any coins inserted
-
- **No free drinks!**
 - **No stealing money.**



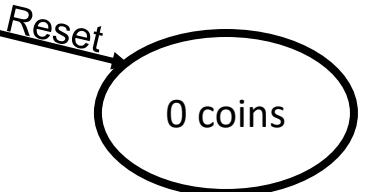
Building the controller

- Finite State Machine
 - An abstract model describing how the machine should be have under a fixed set of circumstances (i.e. finite states)
 - Remember how many coins have been put in the machine and what inputs are acceptable
- Read-Only Memory (ROM)
 - A cheaper way of implementing combinational logic
 - Define the outputs and state transitions
- Custom combinational circuits
 - Reduce the size (and therefore cost) of the controller

Finite State Machines

- A Finite State Machine (FSM) consists of:
 - K states: $S = \{s_1, s_2, \dots, s_k\}$, s_1 is initial state
 - N inputs: $I = \{i_1, i_2, \dots, i_n\}$
 - M outputs: $O = \{o_1, o_2, \dots, o_m\}$
 - Transition function $T(S, I)$ mapping each current state and input to next state
 - Output Function $P(S)$ or $P(S, I)$ specifies output
 - $P(S)$ is a Moore Machine
 - $P(S, I)$ is a Mealy Machine

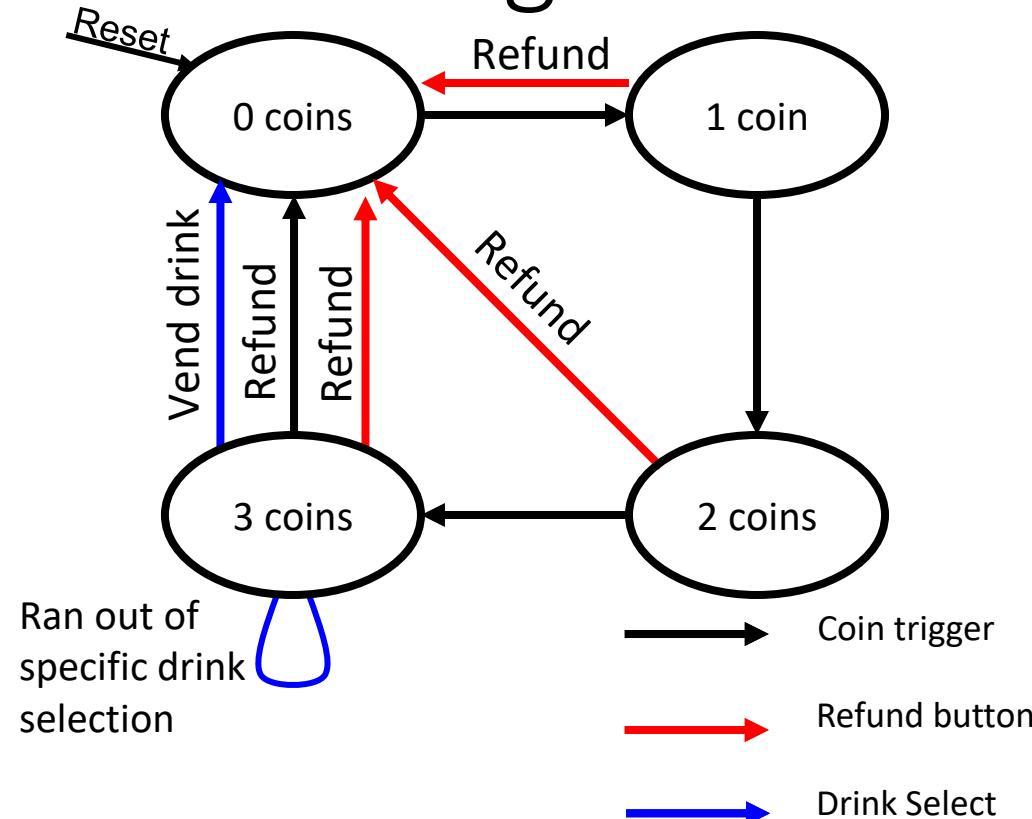
FSM for Vending Machine



- Coin trigger
- Refund button
- Drink Select



FSM for Vending Machine



Is this a Mealy or Moore Machine?

This is Mealy: Mealy output is based on current state *AND* input

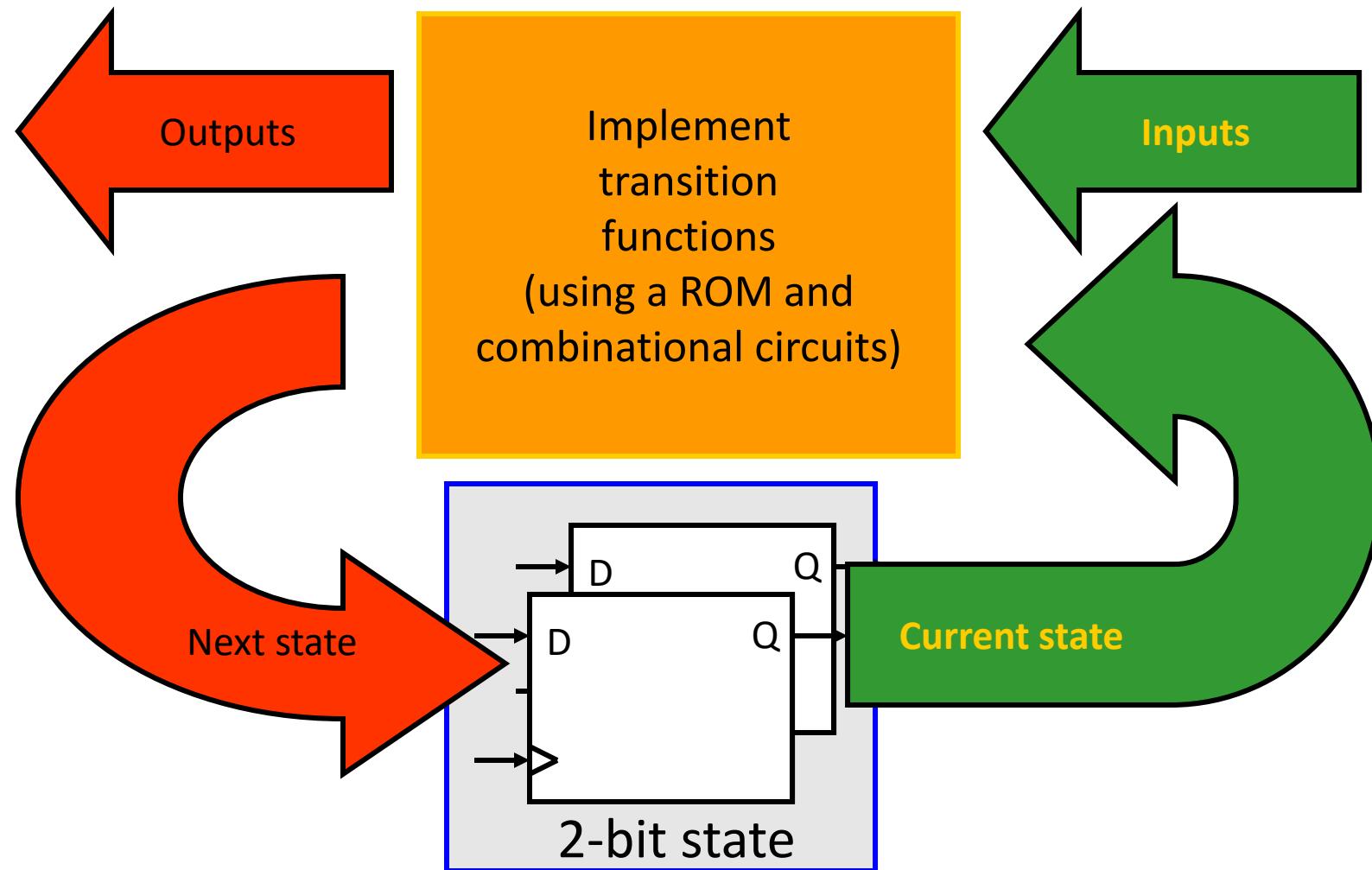
Poll: Mealy or Moore?

Poll: How many flip-flops would we need to remember which state we're in?



Poll: How cheaply do you think we can build one of these controllers?

Implementing an FSM



Implementing an FSM

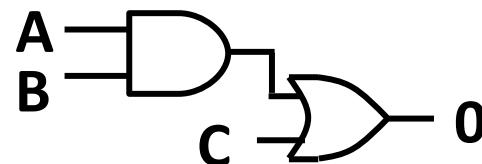
- Let's see how cheap we can build this vending machine controller!
- Jameco.com sells electronic chips we can use
 - D-Flip-flops: \$3, includes several in one package
- For custom combinational circuits, would need to design and send to a fabrication facility
 - Thousands or millions of dollars!!
 - Alternative?

The screenshot shows the Jameco Electronics website. At the top, there is a navigation bar with links for 'Log In/Register', 'Track Orders', 'Cart 0', 'Serving our customers since 1974', 'Customer Care 1-800-831-4242', and 'WORKSHOP'. Below the navigation is a search bar with the placeholder 'Enter Product, Web Code, etc.' and a 'SEARCH' button. The main content area shows a product page for the 'IC 7474 DUAL D TYPE FLIP-FLOP'. The product image is a black integrated circuit package with many pins. To the right of the image, product details are listed: Jameco Part no.: 50551, Manufacturer: Major Brands, Manufacturer p/n: 7474, HTS code: 8542310000, Fairchild Semiconductors [59 KB], Data Sheet (current) [52 KB], and a note that Representative Datasheet, MFG may vary. Below the product details, there is a price of '\$2.95 ea', a quantity selector set to '1', and buttons for 'Add to cart' and '+ Add to my favorites'. A 'View larger image' link is also present. Further down, there is a section titled 'You may also like:' featuring four similar integrated circuit packages: 74HCT74, 74HC74, 74LS74, and 74189, each with 'Major Brands' listed below it. On the left side of the main content area, there is a sidebar with a 'Category' section containing a list of links such as Batteries, Computer Products, Electronic Kits & Projects, Electromechanical, Electronic Design, Fans & Cooling, ICs & Semiconductors, Interconnects, Optoelectronics & LED/Lighting, Passive Components, Power Supplies & Wall Adapters, Test, Tools & Supplies, and Wire & Cable. There are also 'NEW PRODUCTS' and 'CLEARANCE - Additional Savings' sections.

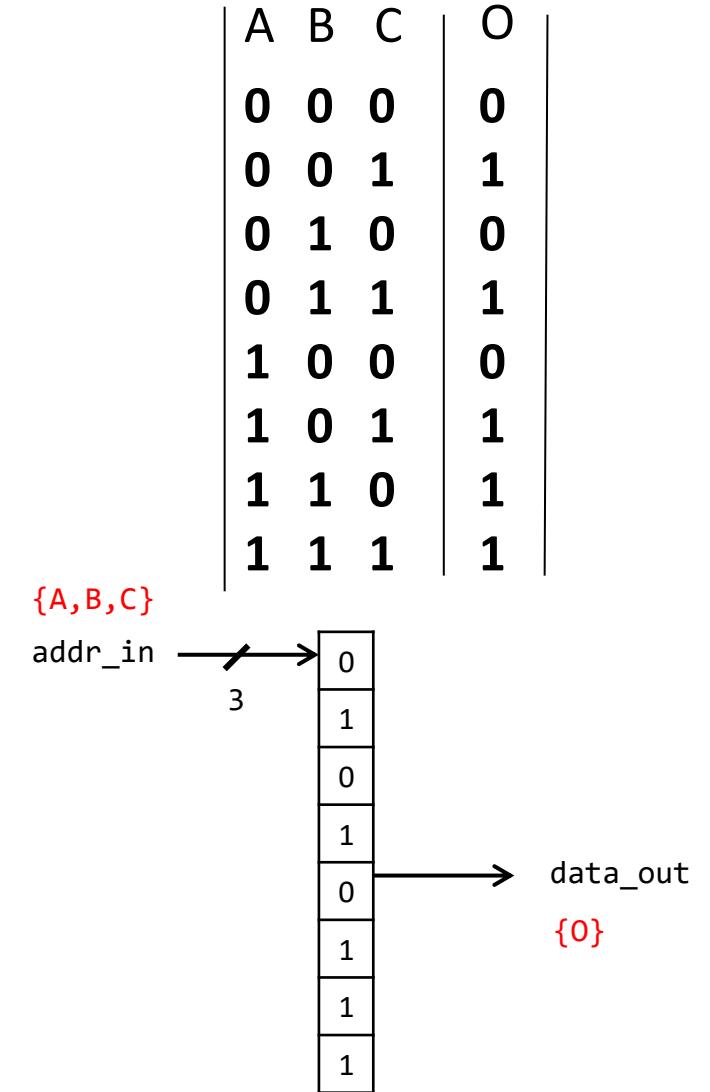
Implementing Combinational Logic

If I have a truth table:

- I can either implement this using combinational logic:



- ...or I could literally just store the entire truth table in a memory and just "index" it by treating the input as a number!
 - Can be implemented cheaply using "Read Only Memories", or "ROMS"



Agenda

- FSM Implementation
- ROMs
- Making our FSM more efficient
- Single Cycle Processor Design Overview
- Supporting each instruction
 - ADD / NOR
 - LW / SW
 - BEQ
 - JALR



ROMs and PROMs

IC 28C256-15 EEPROM 256K-Bit CMOS Parallel



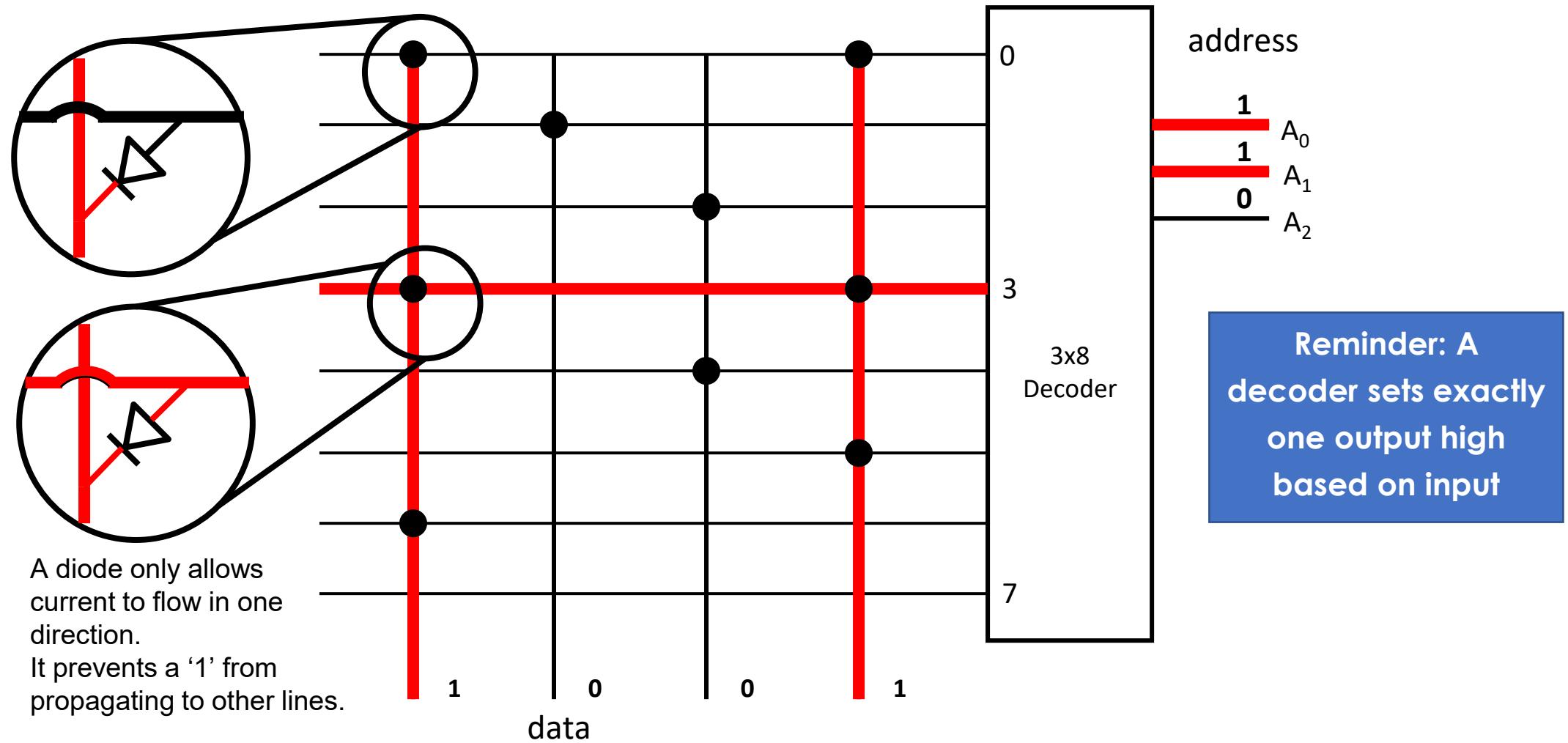
[View larger image](#)

Jameco Part no.: 74843
Manufacturer: Major Brands
Manufacturer p/n: 28C256-15
HTS code: 8542320050
[Data Sheet \(current\)](#) [116 KB]
[Data Sheet \(current\)](#) [499 KB]
ST MICRO [62 KB]
Atmel [371 KB]
Atmel [67 KB]
Representative Datasheet. MFG may vary

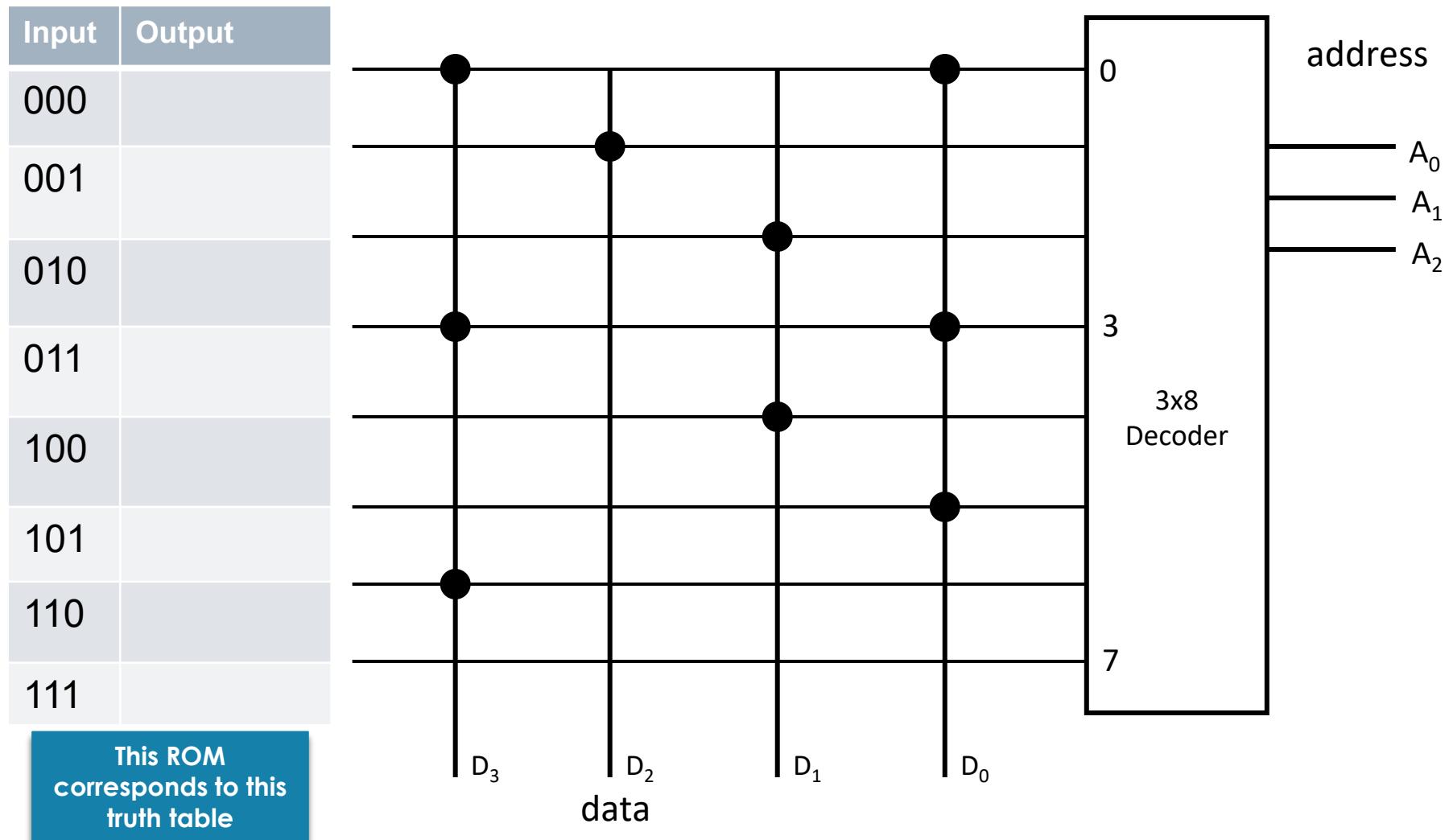
\$12.25 ea
36 In Stock
More Available - 7 weeks
Qty
[Add to cart](#)
[+ Add to my favorites](#)

- Read Only Memory (ROM)
 - Array of memory values that are constant
 - Non-volatile (doesn't need constant power to save values)
- Programmable Read Only Memory
 - Array of memory values that can be written exactly once
- Electronically Erasable PROM (EEPROM)
 - Can write to memory, deploy in field
 - Use special hardware to reset bits if need to update
- 256 KBs of EEPROM costs ~\$10 on Jameco
 - Much better than spending thousands on design costs unless we're gonna make **tons** of these

8-entry 4-bit ROM



8-entry 4-bit ROM



Poll: What's the formula for size of ROM needed?

8-entry 4-bit ROM

