9. Foundations of processor design: Memory Elements and Sequential Logic

EECS 370 – Introduction to Computer Organization – Winter 2023

EECS Department
University of Michigan in Ann
Arbor, USA

Upcoming stuff

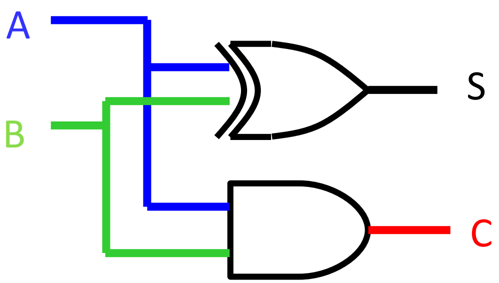
- ❑ Project 1s and 1m
  - Due today!
- ❑ HW2
  - Due Monday 2/6
  - Group part is non-trivial
- ❑ Project 2 is posted
  - 2a due Thursday 2/16
  - 2l due Thursday 2/23
  - 2c due Tuesday 3/10 (after break)
- ❑ HW3 out early next week, due Monday 2/20
- ❑ <u>Midterm: Thursday 3/9 7-9pm</u>

Recap: Combinational Circuits – implement Boolean expressions

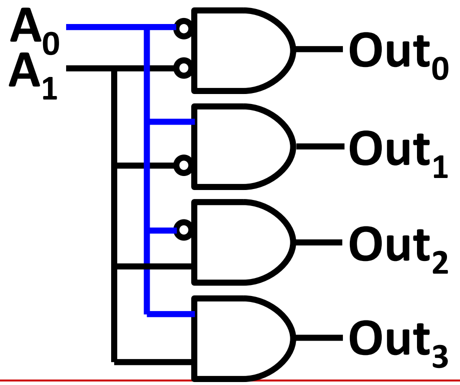
- **No memory**: Output a function only of input
- Undefined input implies undefined output

  - Adder is the basic gate of the ALU
  - Decoder is the basic gate of indexing
  - MUX is the basic gate controlling data movement

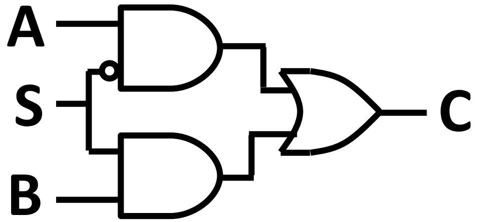
Half Adder



Decoder

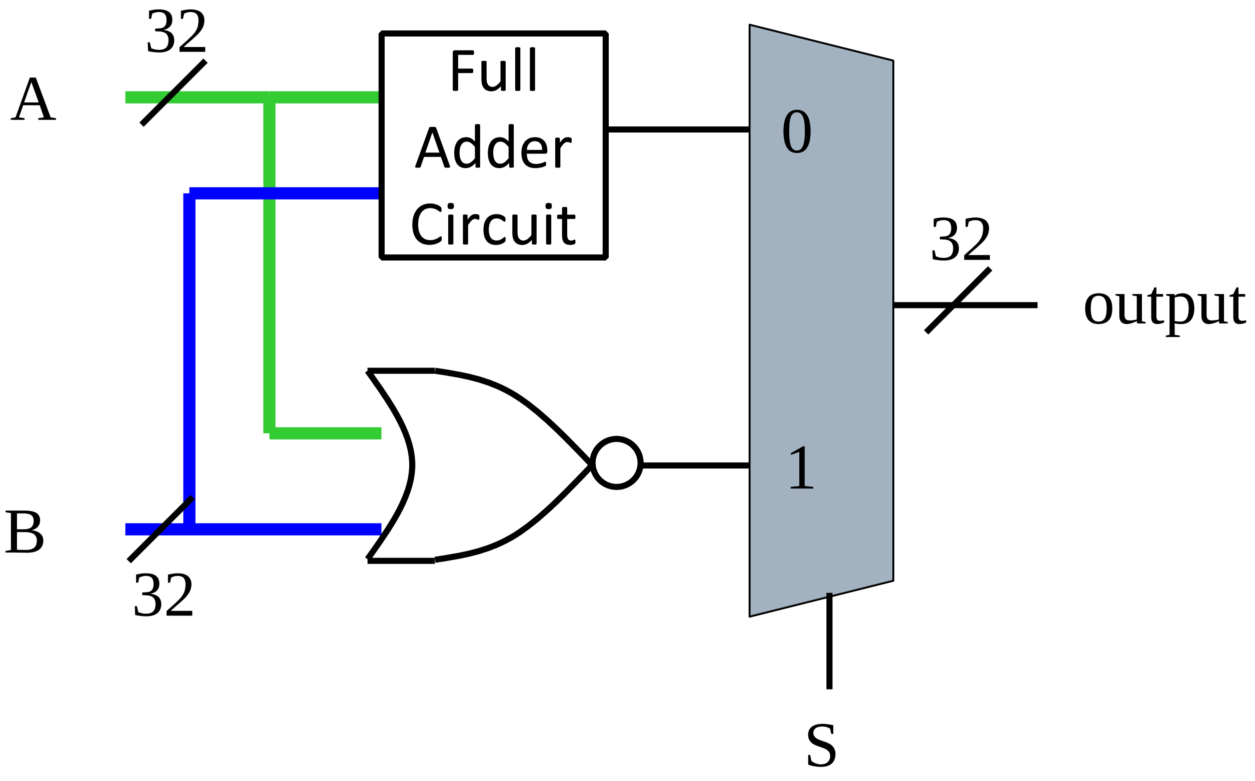


MUX

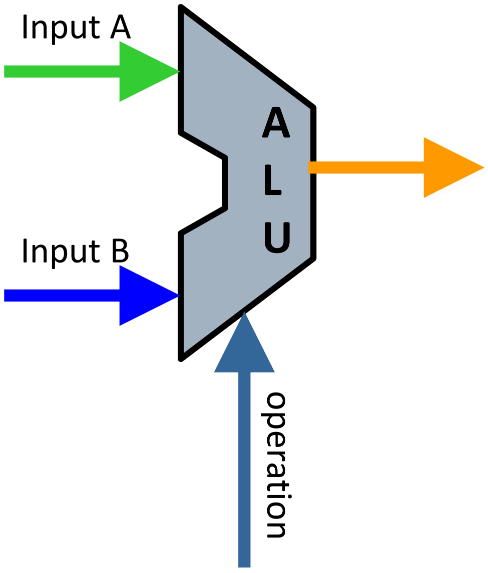


LC2K ALU – From the Previous Lecture...

Circuit



Symbol



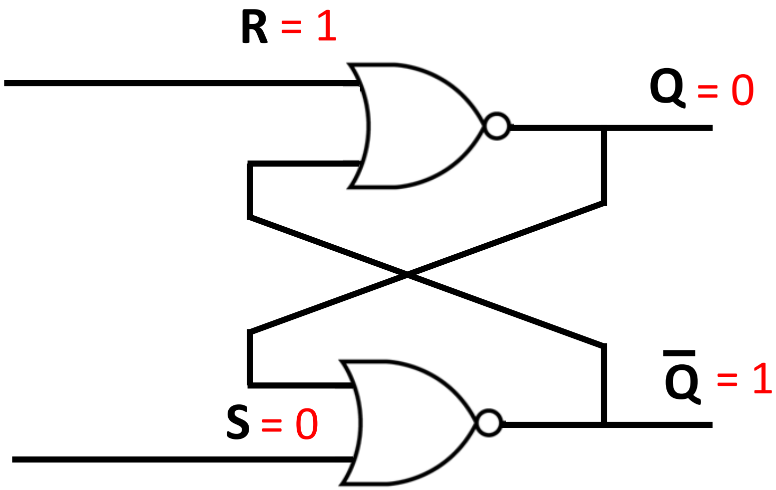
Next topic:

Sequential logic:
giving memory to circuits

What is sequential logic?

- So far, we've covered combinational
  - Output is determined from input
  - But computers don't work that way -- they have state
- Examples of state
  - Registers
  - Memory
- Sequential logic's output depends not only on the current input, but also on its current state
- This lecture will show you how to build sequential logic from gates
  - The key is feedback

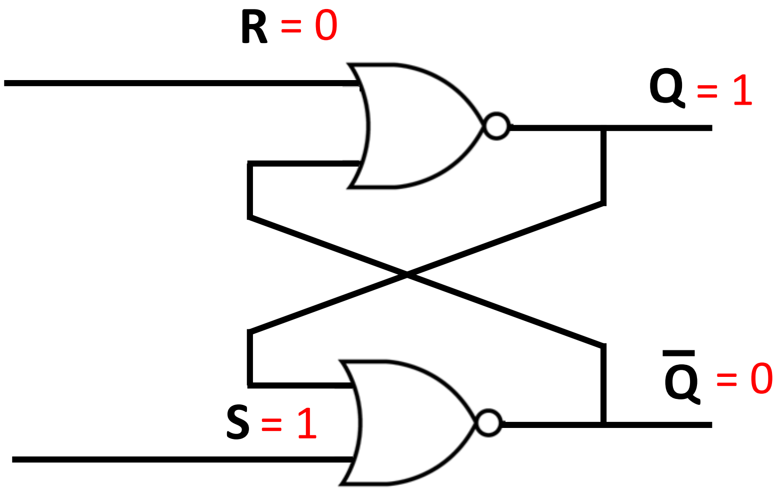
Let’s look at the following circuit



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

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

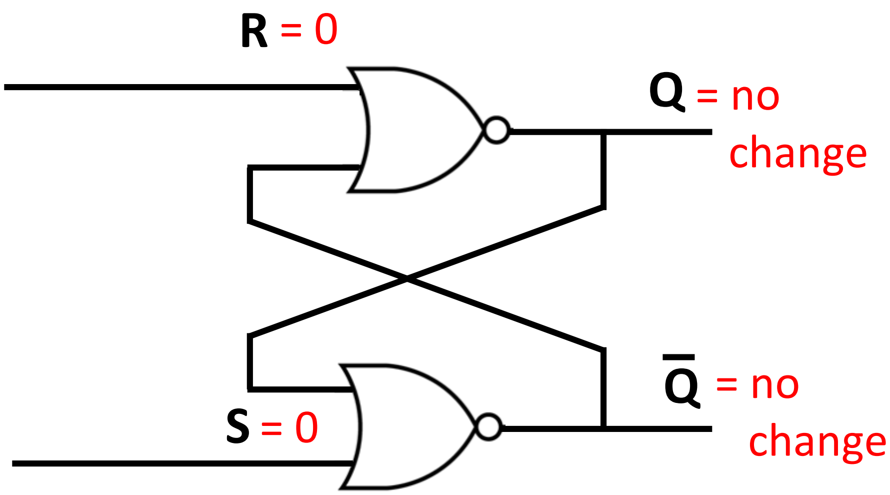
Building the Truth Table



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

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

For a Basic Memory Cell

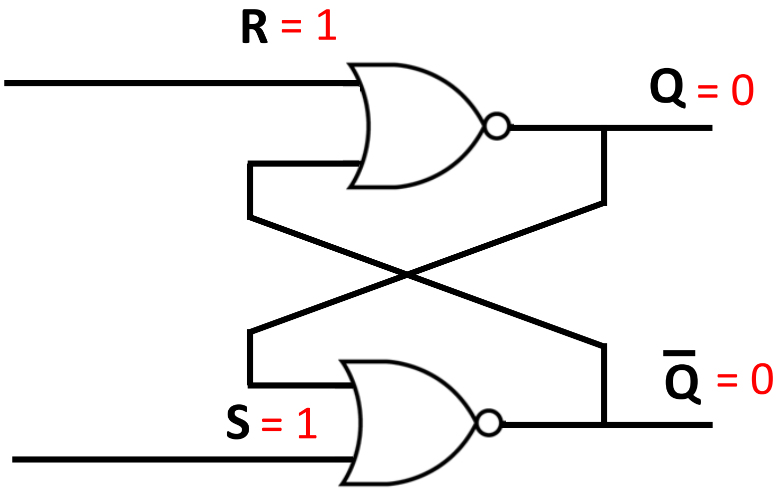


S	R	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	1	0

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

As long as R and S remain 0 then the value Q (and $\bar{Q}$) will remain unchanged. This value is stored in this circuit. **This is a basic memory cell.**

With “invalid” inputs 1,1



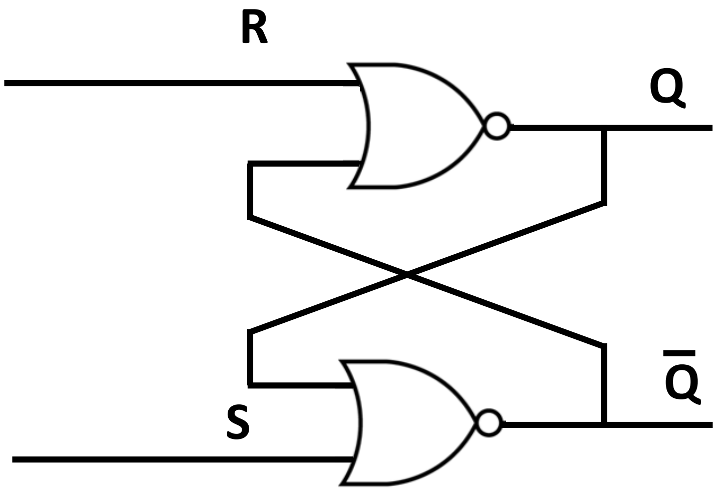
S	R	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	0	1
1	0	1	0
1	1	0	0

Invalid ~ AVOID!

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

Q and $\overline{Q}$ are supposed to be opposite of each other, so **this is a state we avoid.** This state can also lead to unstable future states. Try setting S = 0 and R = 0 now!

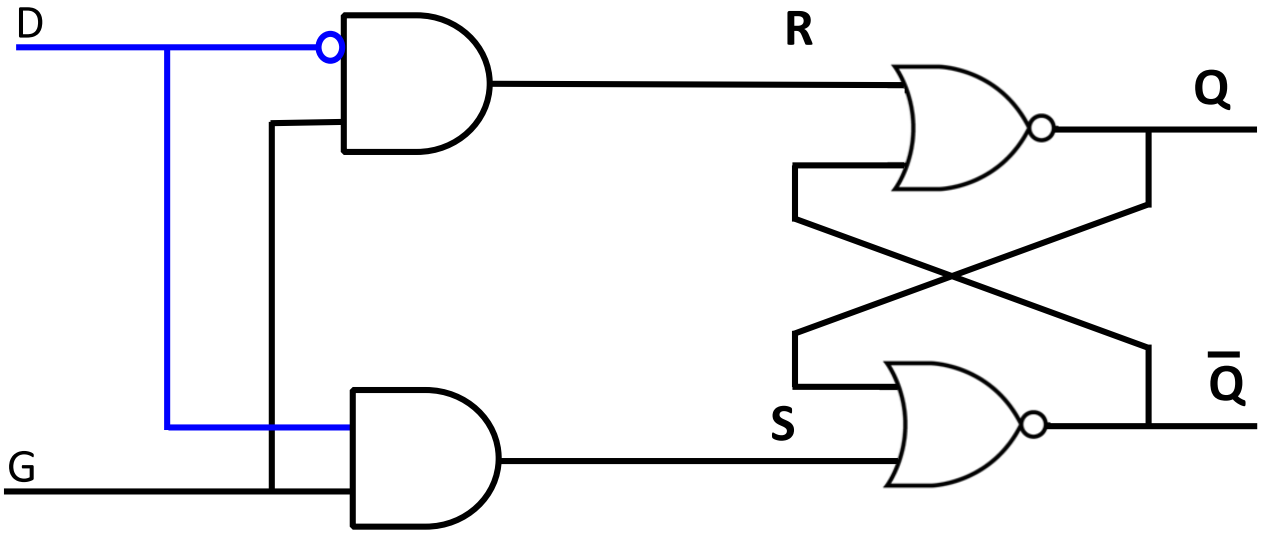
SR Latch



S	R	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	0	1
1	0	1	0
1	1	BAD	

- What do S and R stand for?
- Set and Reset

Transparent D Latch

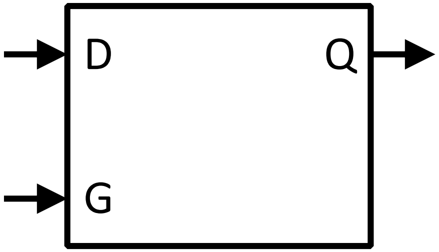


Gating signal

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

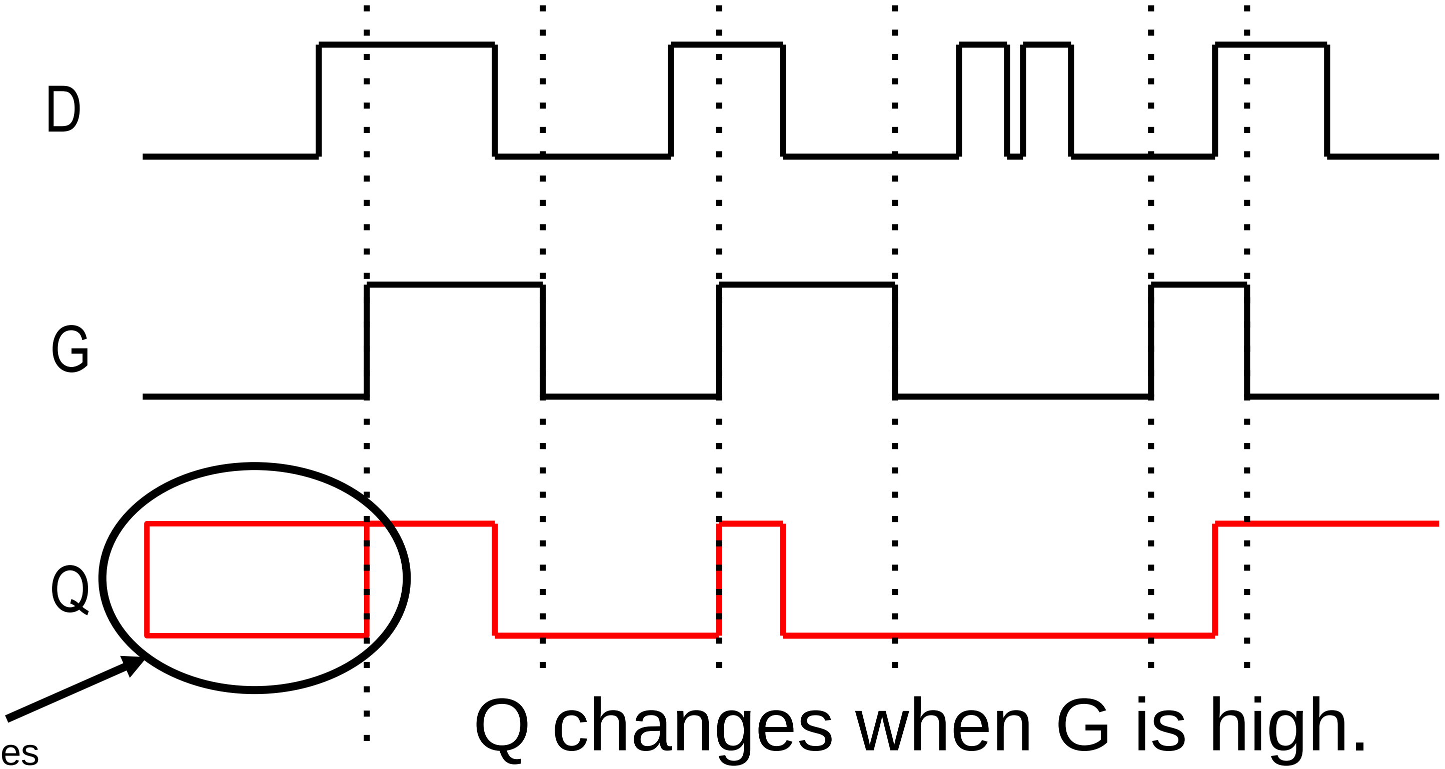
Next state is set

Set state is retained when gate is low



No invalid states ☺

D latch timing

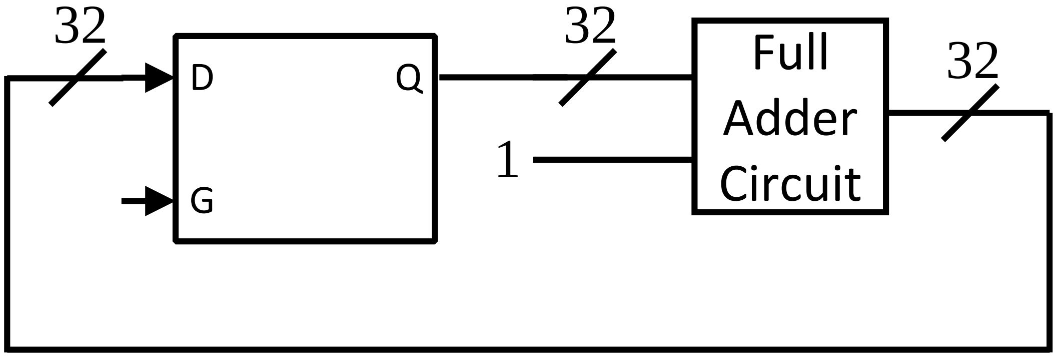


Indicates uncertainty of initial state

Q changes when G is high.
Q is preserved when G is low

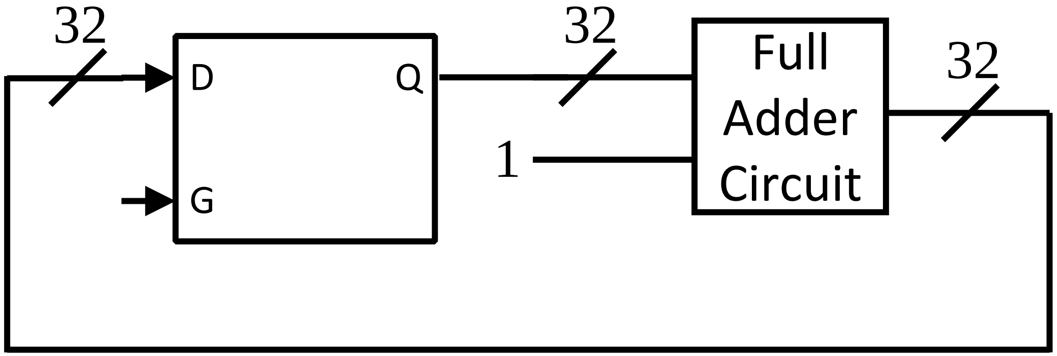
Is D-Latch Sufficient?

- Can we use D-latches to build our Program Counter 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

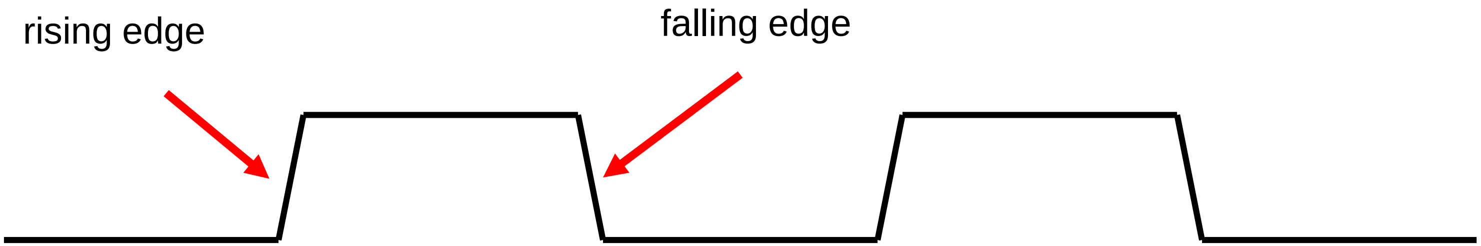
Adding a clock to the mix

- We can design more interesting circuits with a clock
  - Enables a sequential circuit to **predictably** change state (and store information)
  - A clock signal alternates between 0 and 1 states at a fixed frequency (e.g., 100MHz)

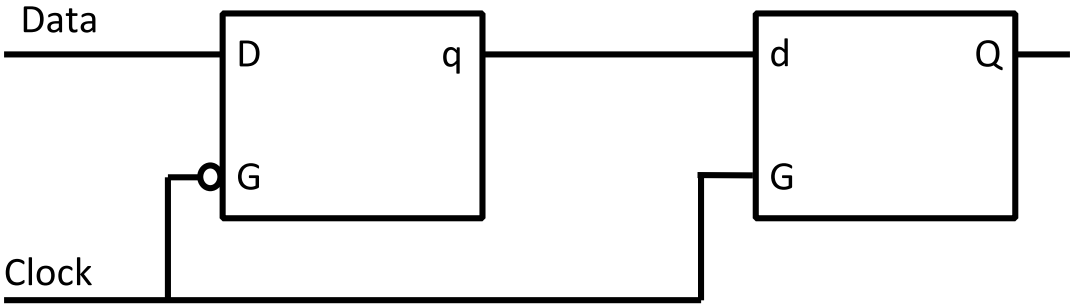
- What should the clock frequency be?
  - It depends on the longest propagation delay between state and next state combination logic
  - And a few other things beyond this class *(shout out 270*)

Clocks

- Clock signal
  - Periodic pulse
  - Generated using oscillating crystal or ring oscillator
  - Distributed throughout chip using clock distribution net



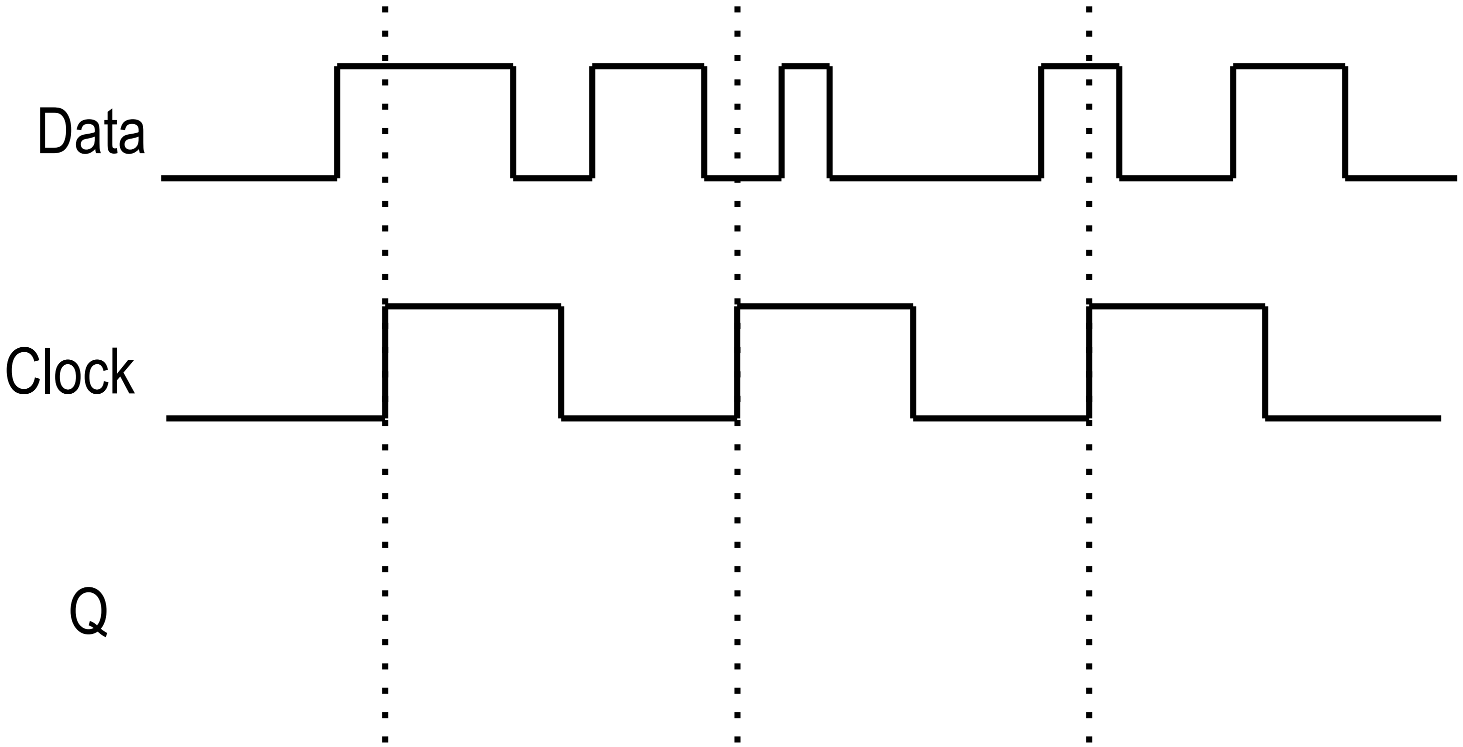
Adding a Clock to the Mix



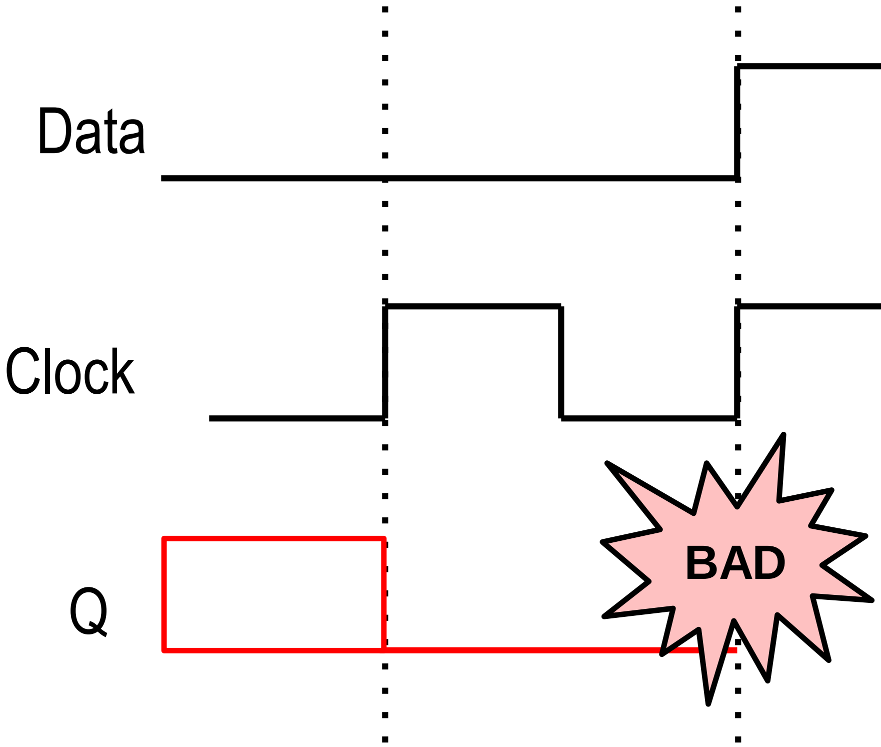
We won't discuss it further here, but this circuit sets Q=D **ONLY** when clock transitions from 0 -> 1

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

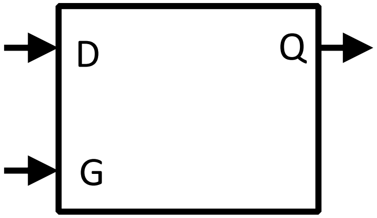


What happens if Data changes on a clock edge?

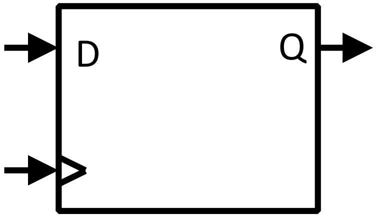


- **Unpredictable:** Q could be high, low or in a meta-stable state
- Likely need to increase your clock period to allow enough time for signal propagation

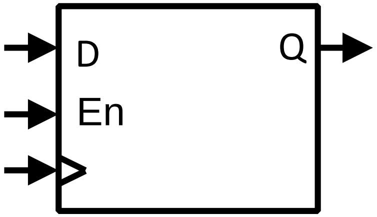
Latches vs Flip-flops



D Latch



D Flip-flop



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

Finite State Machines

- ❑ 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, Memory
- ❑ 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)*

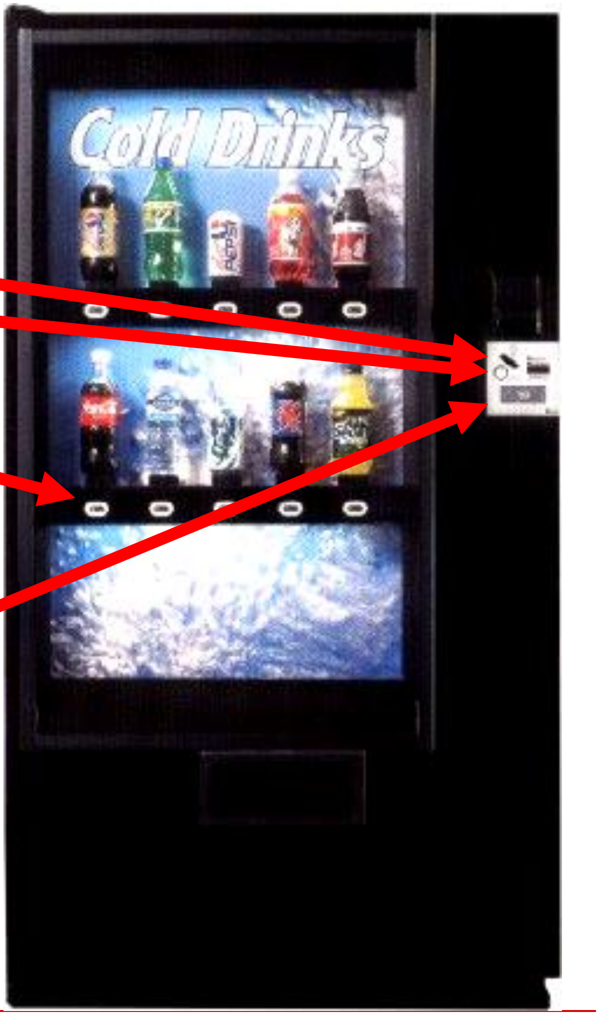
Vending Machine

- ❑ We could use a general purpose processor
- ❑ However, a custom controller will be:
  - Faster
  - Lower power
  - Cheaper to produce in high volume
- ❑ On the other hand, a custom controller:
  - Will be slower to design
  - More expensive in low volume

- ❑ Goals:
  - Take money, vend drinks.

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
  - Remember how many coins have been put in the machine and what inputs are acceptable
- Read-Only Memory (ROM)
  - 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 = {s1, s2, … ,sk}, s1 is initial state
  - N inputs: I = {i1, i2, … ,in}
  - M outputs: O = {o1, o2, … ,om}
  - 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

Two Common State Machines

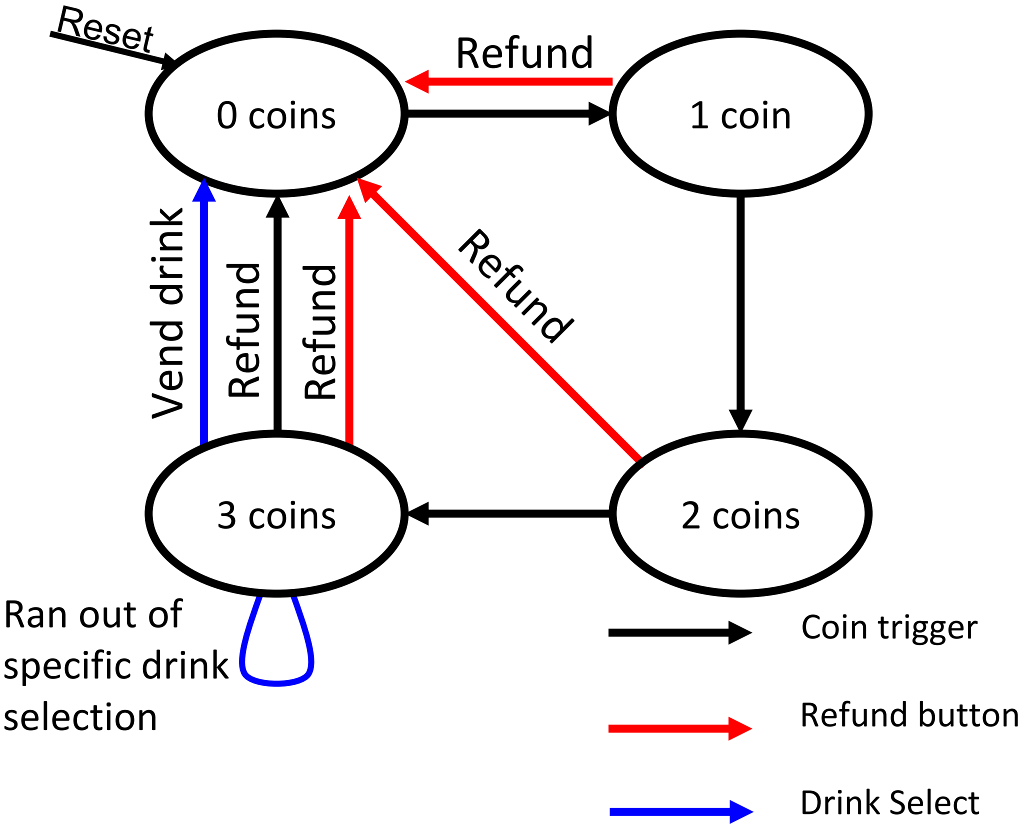
- Moore machine

  output function based on **current state** P(S) only

- Mealy machine

  output function based on **current state** and **current input** P(S,I)

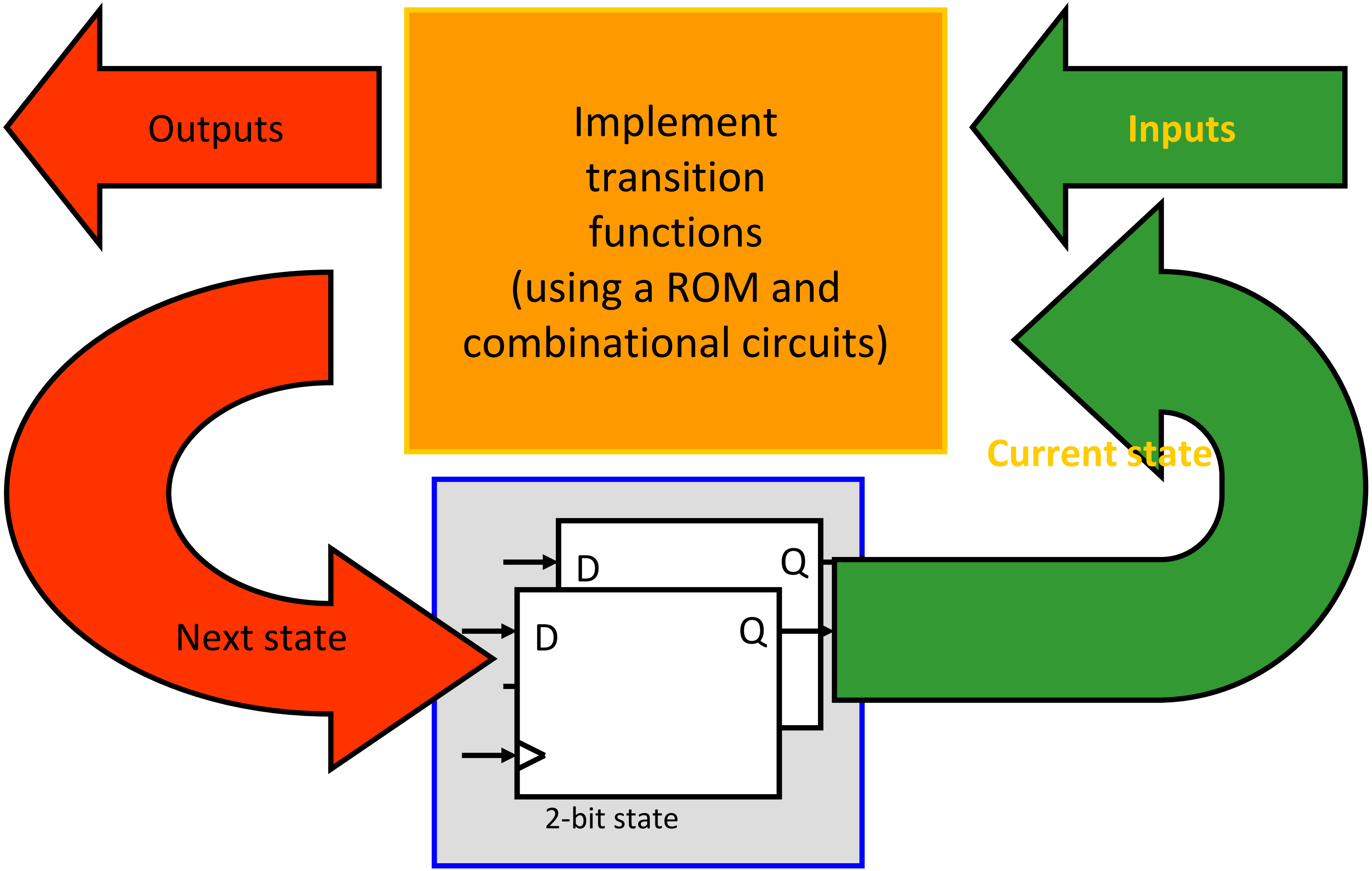
FSM for Vending Machine



Is this a Mealy or Moore Machine?

Mealy ~ output is based on current state *AND* input

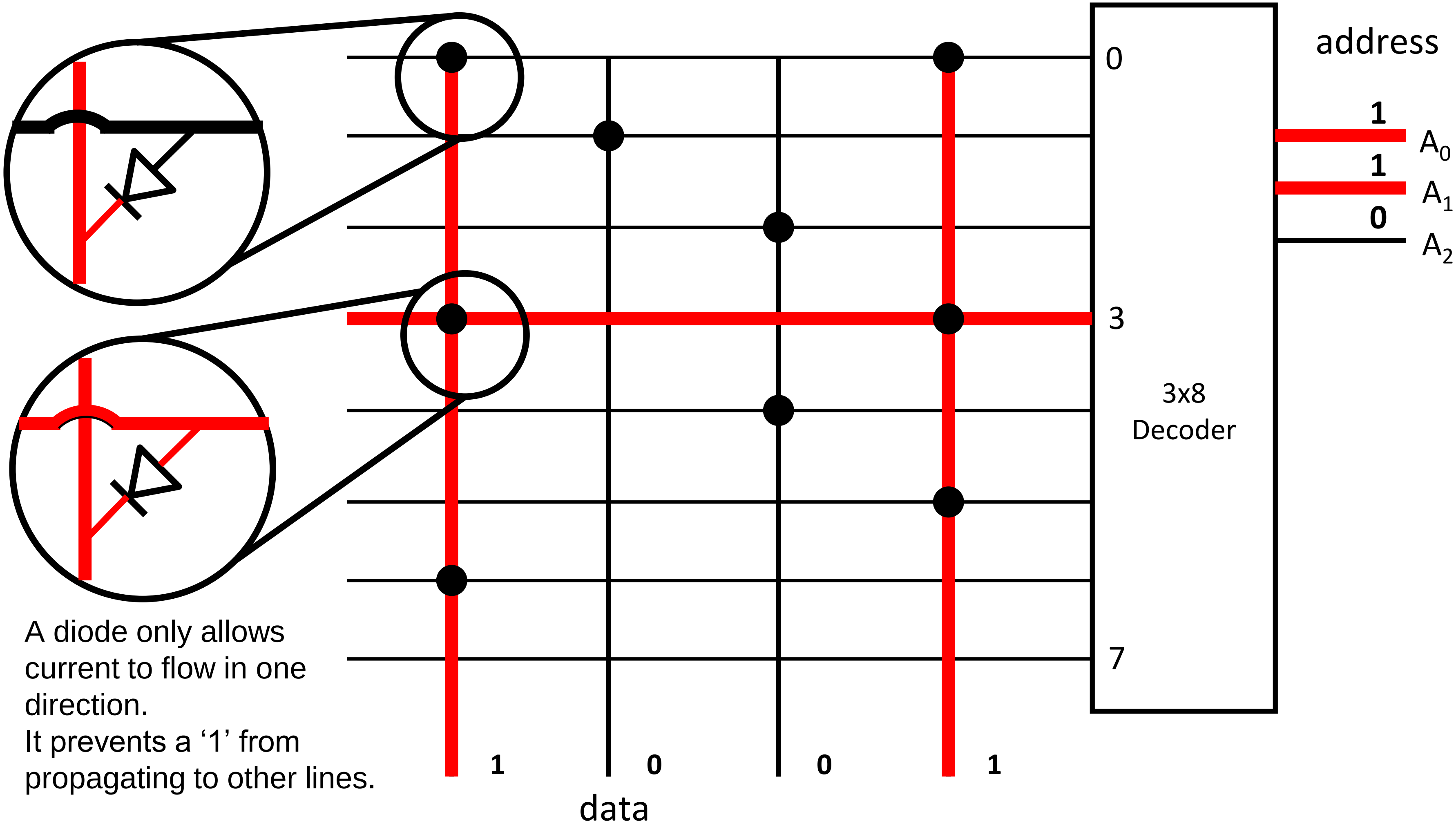
Implementing a FSM


Outputs
Implement transition functions (using a ROM and combinational circuits)
Inputs
Current state
Next state
D
Q
D
Q
2-bit state

ROMs and PROMs

- Read Only Memory
  - Array of memory values that are constant
  - Non-volatile
- Programmable Read Only Memory
  - Array of memory values that can be written exactly once (destructive writes)
- You can use ROMs to implement FSM transition functions
  - ROM inputs (i.e., ROM address): current state, primary inputs
  - ROM outputs (i.e., ROM data): next state, primary outputs

8-entry 4-bit ROM

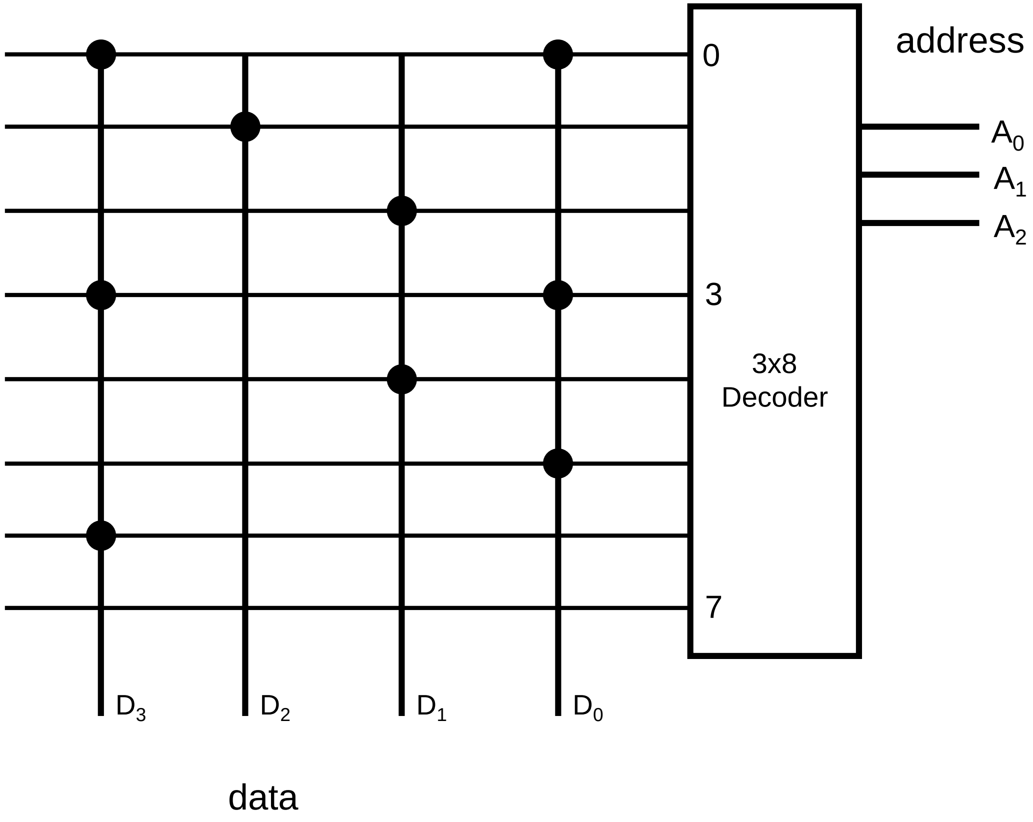


- A diode only allows current to flow in one direction.
- It prevents a ‘1’ from propagating to other lines.

8-entry 4-bit ROM

Input	Output
000	
001	
010	
011	
100	
101	
110	
111	

This ROM corresponds to this truth table

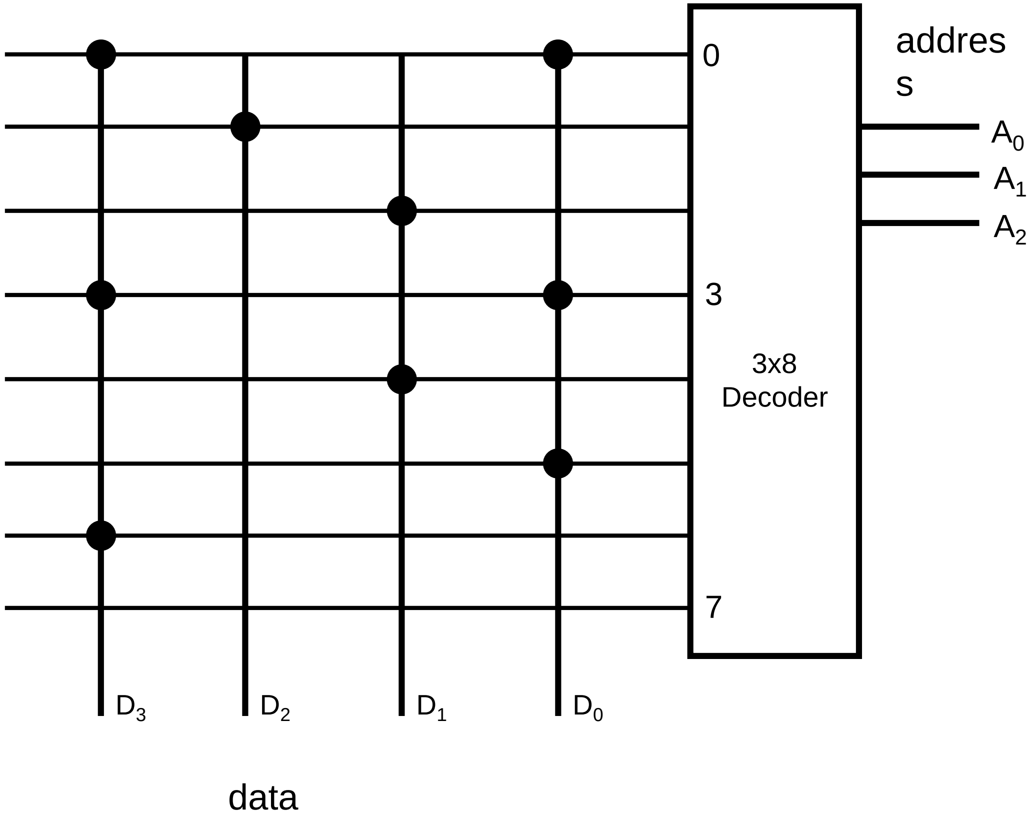


8-entry 4-bit ROM

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

Input	Output
000	1001
001	0100
010	0010
011	1001
100	0010
101	0001
110	1000
111	0000

This ROM corresponds to this truth table



Aside: Other Memories

- Static RAM (random access memory)
  - Built from sequential circuits
    - Takes 4-6 transistors to store 1 bit
    - Fast access (< 1 ns access possible)

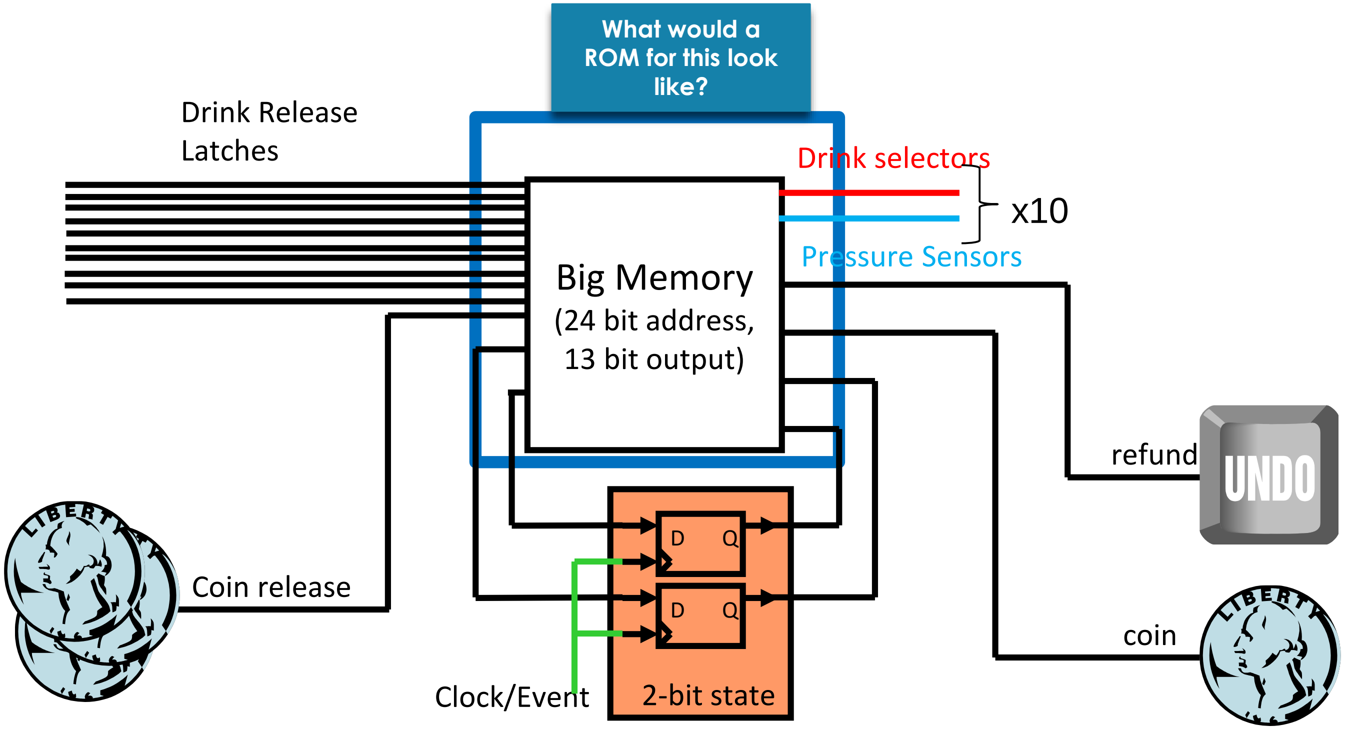
- Dynamic RAM
  - Built using a single transistor and a capacitor
    - 1's must be refreshed often to retain value
    - Slower access than static RAM
    - Much more dense layout than static RAM

- Both require constant power, or they will lose their data (i.e. are **volatile**)
- These will be used to build computer memory hierarchies (later in class)

Implementing Combinational Logic

- Custom logic
  - Pros:
    - Can optimize the number of gates used
  - Cons:
    - Can be expensive / time consuming to make custom logic circuits
- Lookup table:
  - Pros:
    - Programmable ROMs (Read-Only Memories) are very cheap and can be programmed very quickly
  - Cons:
    - Size requirement grows exponentially with number of inputs (adding one just more bit **doubles** the storage requirements!)

Controller Design So far



ROM for Vending Machine Controller

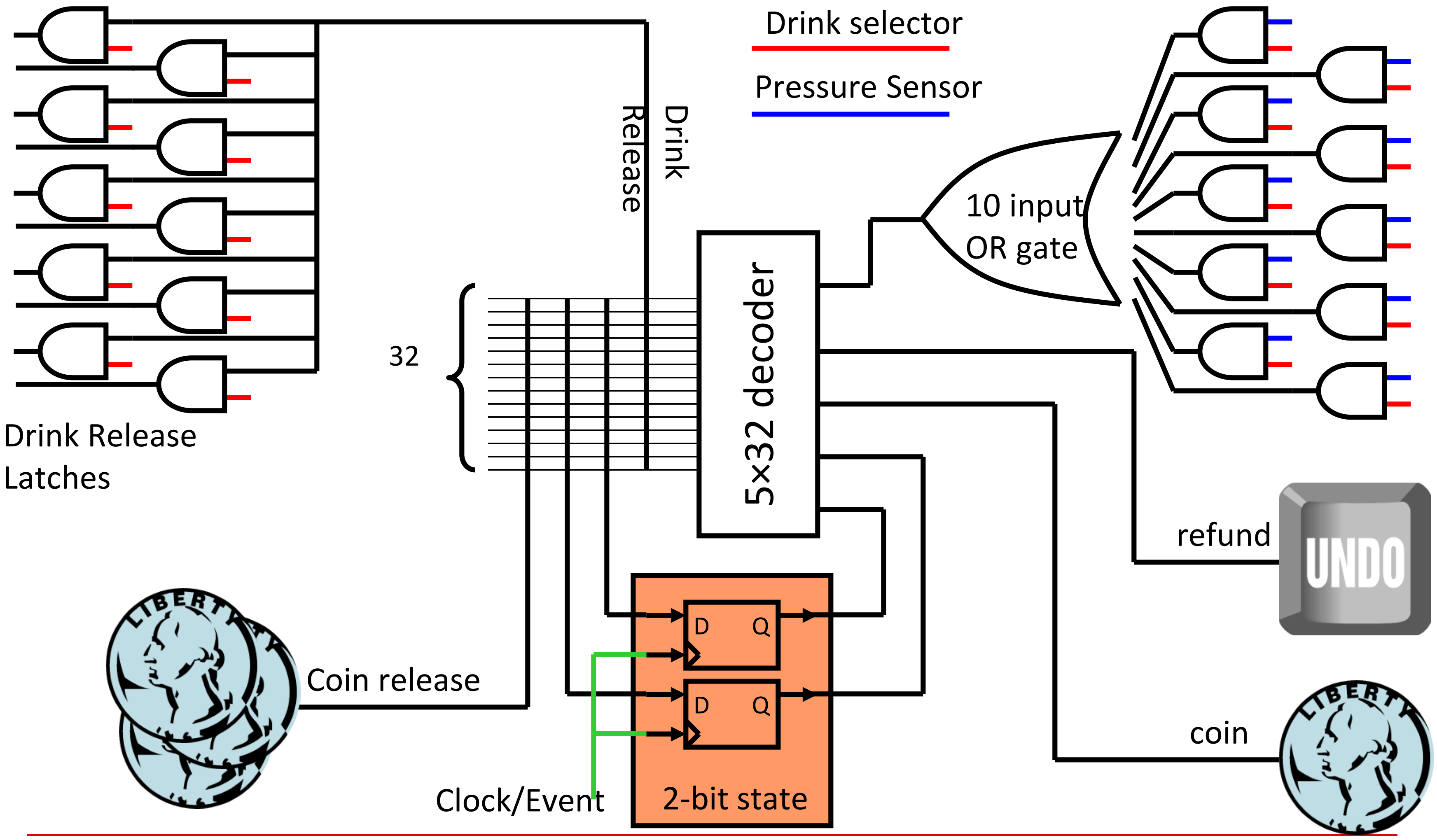
- Use current state and inputs as address
  - 2 state bits + 22 inputs = 24 bits (address)
  - Coin, refund, 10 drink selectors, 10 sensors
- Read next state and outputs from ROM
  - 2 state bits + 11 outputs = 13 bit (memory)
  - Refund release, 10 drink latches
- We need 2^{24} entry, 13 bit ROM memories
  - 218,103,808 bits of ROM seems excessive for our cheap controller

Reducing the ROM needed

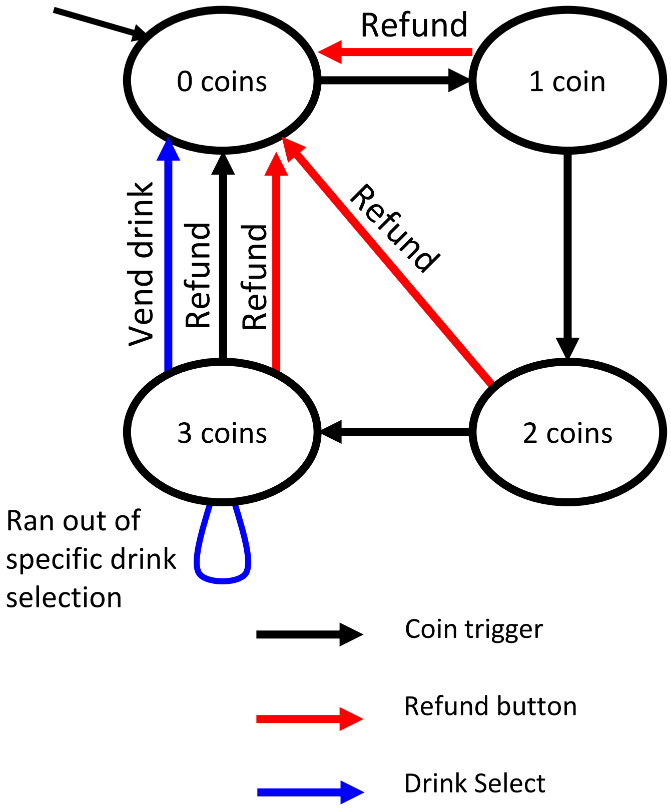
- Replace 10 selector inputs and 10 pressure inputs with a single bit input (drink selected)
  - Use drink selection input to specify which drink release latch to activate
  - Only allow trigger if pressure sensor indicates that there is a bottle in that selection. (10 2-bit ANDs)

- Now:
  - 2 current state bits + 3 input bits (5 bit ROM address)
  - 2 next state bits + 2 control trigger bits (4 bit memory)
  - $2^5 \times 4 = 128$ bit ROM (good!)

Putting it all together



Some of the ROM contents



Current state	Coin trigger	Drink select	Refund button
0 0	0	0	0
0 0	0	0	1
0 0	0	1	0
0 0	1	0	0
0 1	1	0	0
1 0	1	0	0
1 1	0	1	0
1 1	1	0	0
… 24 more entries			

ROM address (current state, inputs)

Next state	Coin release	Drink release
0 0	0	0
0 0	0	0
0 0	0	0
0 1	0	0
1 0	0	0
1 1	0	0
0 0	0	1
0 0	1	0
… 24 more entries		

ROM contents (next state, outputs)

Limitations of the controller

- What happens if we make the price $1.00?, or what if we want to accept nickels, dimes and quarters?
  - Must redesign the controller (more state, different transitions)
  - A programmable processor only needs a software upgrade.
    - If you had written really good software anticipating a variable price, perhaps no change is even needed

- **Next Topic - Our first processor!**

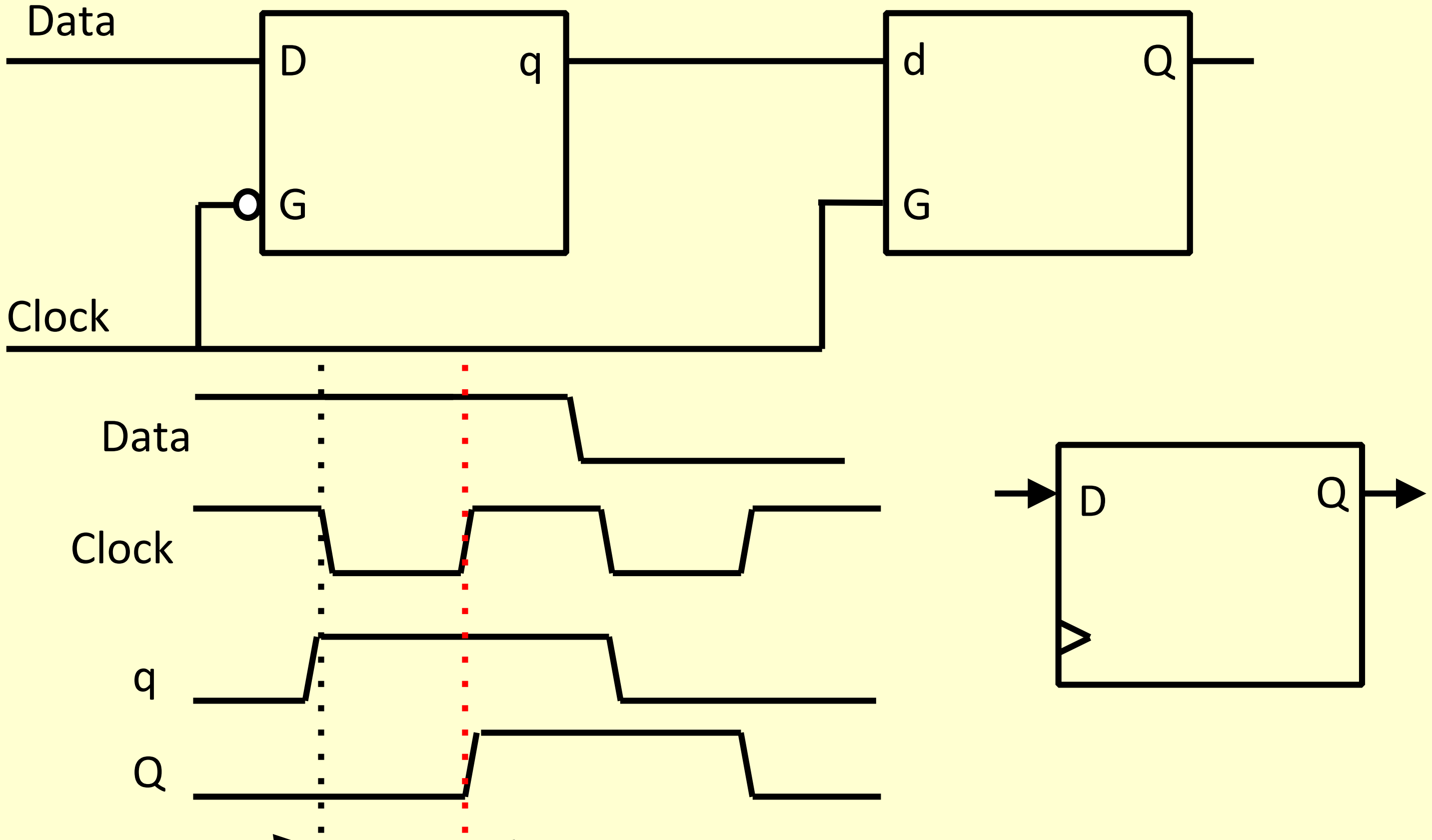
Single-Cycle Processor Design—LC2K

- General-Purpose Processor Design
  - Fetch Instructions
  - Decode Instructions
    - Instructions are input to control ROM
  - ROM data controls movement of data
    - Incrementing PC, reading registers, ALU control
  - Clock drives it all
  - Single-cycle datapath: Each instruction completes in one clock cycle

Extra slides

- The following slides go into more detail about how an edge-triggered flip-flop works
  - And how it is built out of two D-latches and an inverter.
- This is more “for your information” though you may find it helpful in understanding some of this material.

Rising-Edge Triggered D Flip-flop (Extra slide)



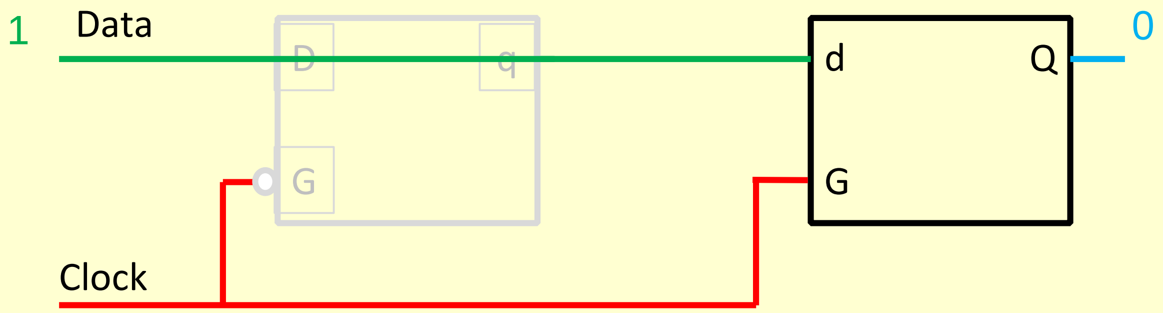
Latching starts

Latching edge

Value remembered by flip-flop

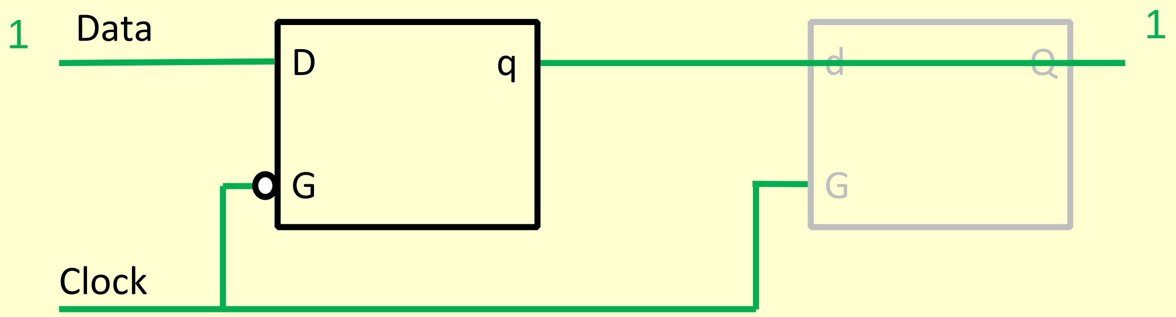
Value should remain valid and stable during latching!

Extra Slides



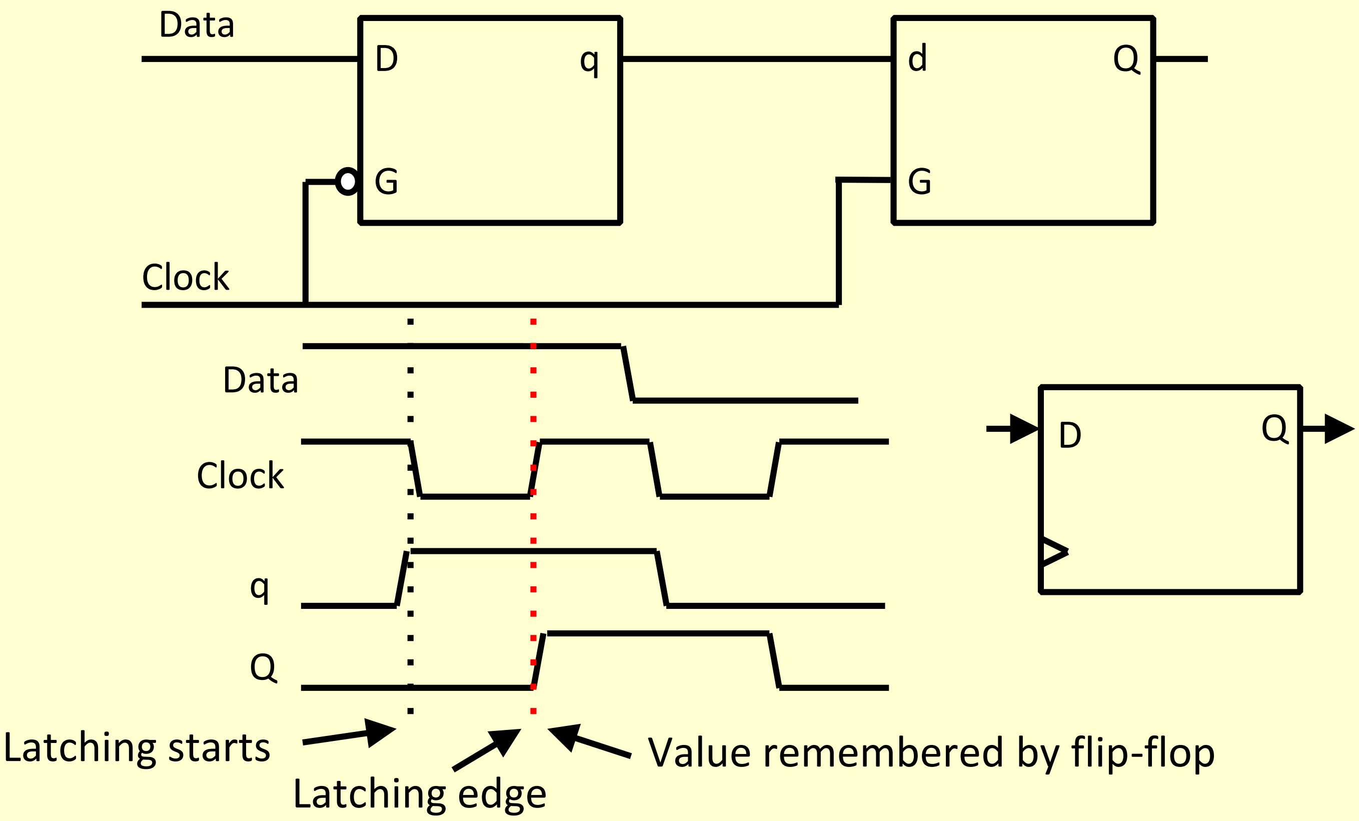
- When "Clock" is low, first latch is transparent
- But Q is latched to previous value of q

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


Data
D
q
d
Q
G
G
Clock
Data
Clock
q
Q
D
Q
Latching starts
Latching edge
Value remembered by flip-flop