9. Sequential logic and FSMs

EECS 370 – Introduction to Computer Organization - Winter 2016

Profs. Valeria Bertacco & Reetu Das

EECS Department
University of Michigan in Ann Arbor, USA

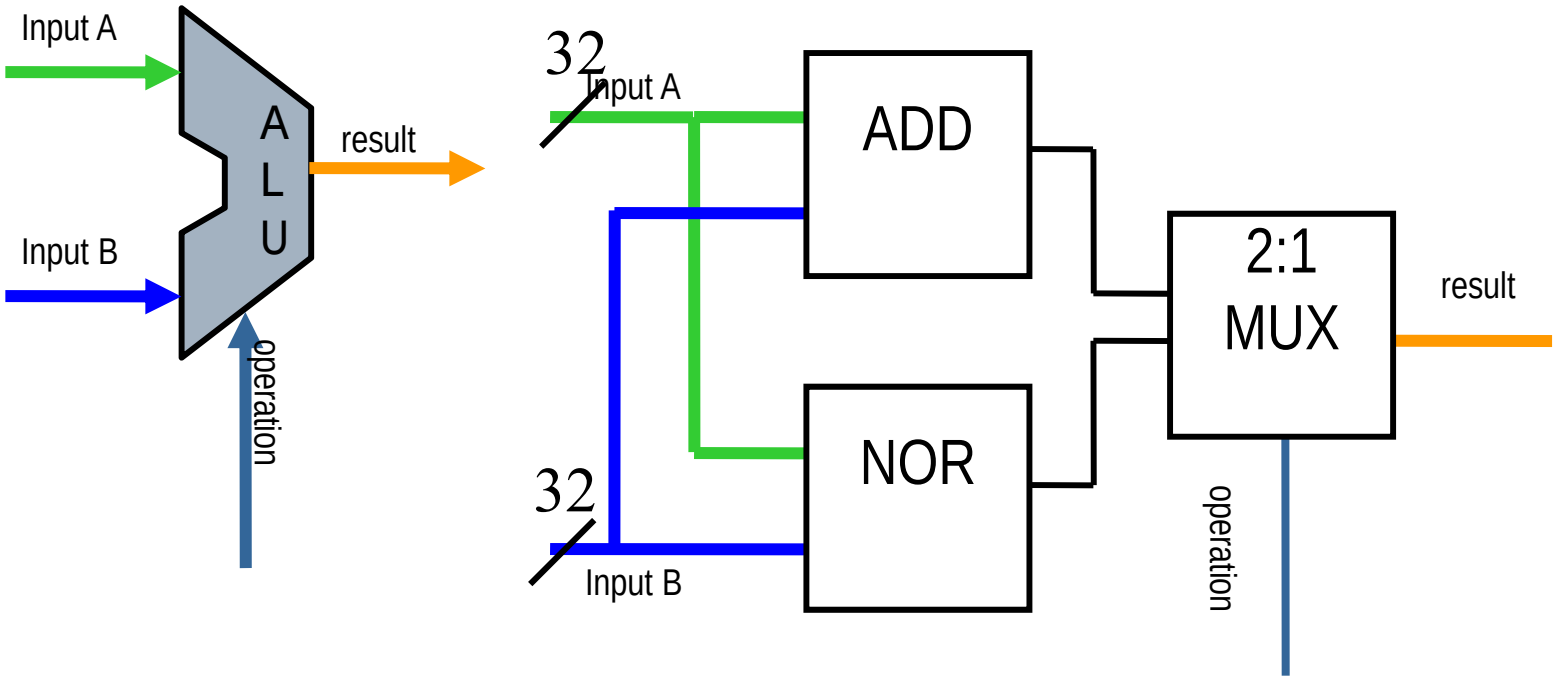
Announcements

- Project 1 due today
- Midterm 1 – February 11, 7pm
  - Students with conflicts or special accommodations will be contacted by Monday by Prof. Bertacco
  - Location will be announced on Tuesday
  - Closed book, notes, calculator, smartphone and anything electronic
  - You are allowed 1 letter-size cheat-sheet, one side only
- Next Tuesday: in-class review (and coke machine)
- Next Wednesday: GSI/IA review – 6pm BBB1670

Recap

- Combinational logic
  - Silicon, CMOS
  - Basic logic gates: AND, OR, NOT, NOR, XOR
  - Mux, Decoder, Half-adder, Full-Adder
  - Ripple-Carry and Carry Look-Ahead Adders

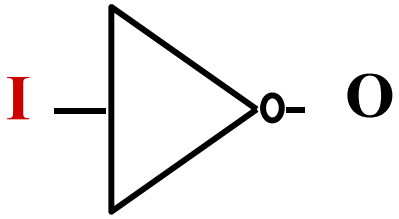
Building combinational circuits: LC-2K ALU

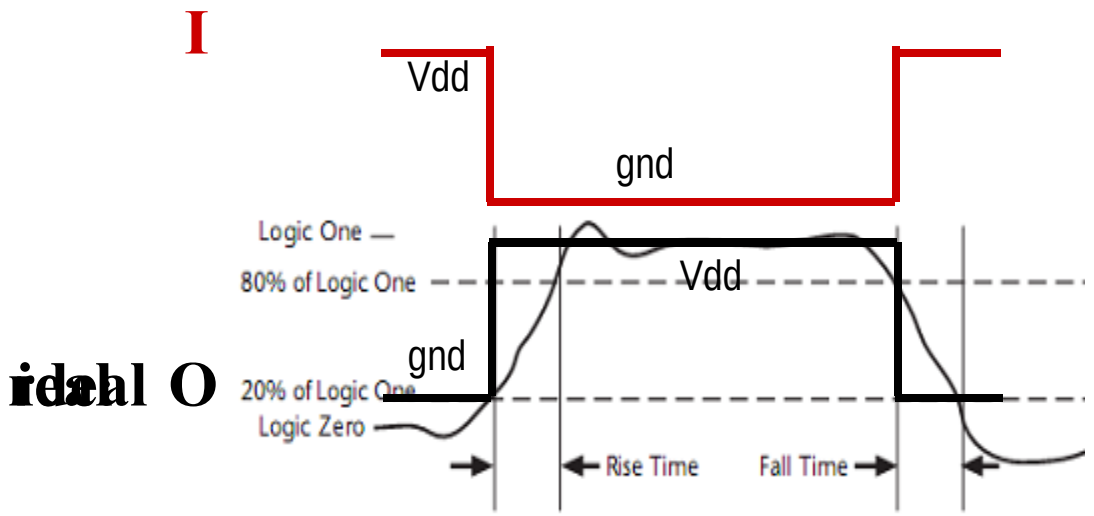


One more problem: propagation delay in combinational gates

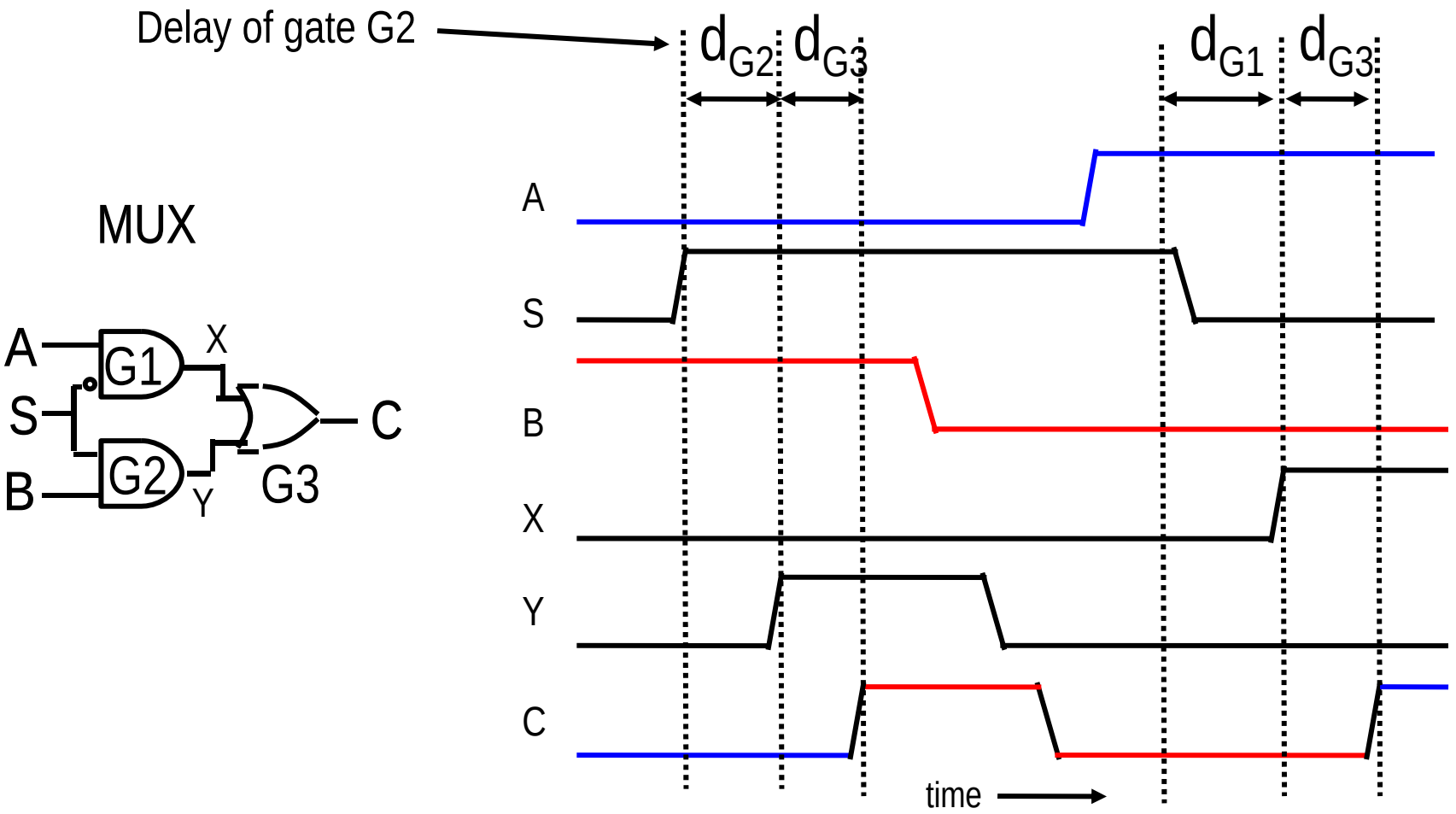
- Gate outputs do not change exactly when inputs do.
  - Transmission time over wires (~speed of light)
  - Saturation time to make transistor gate switch

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



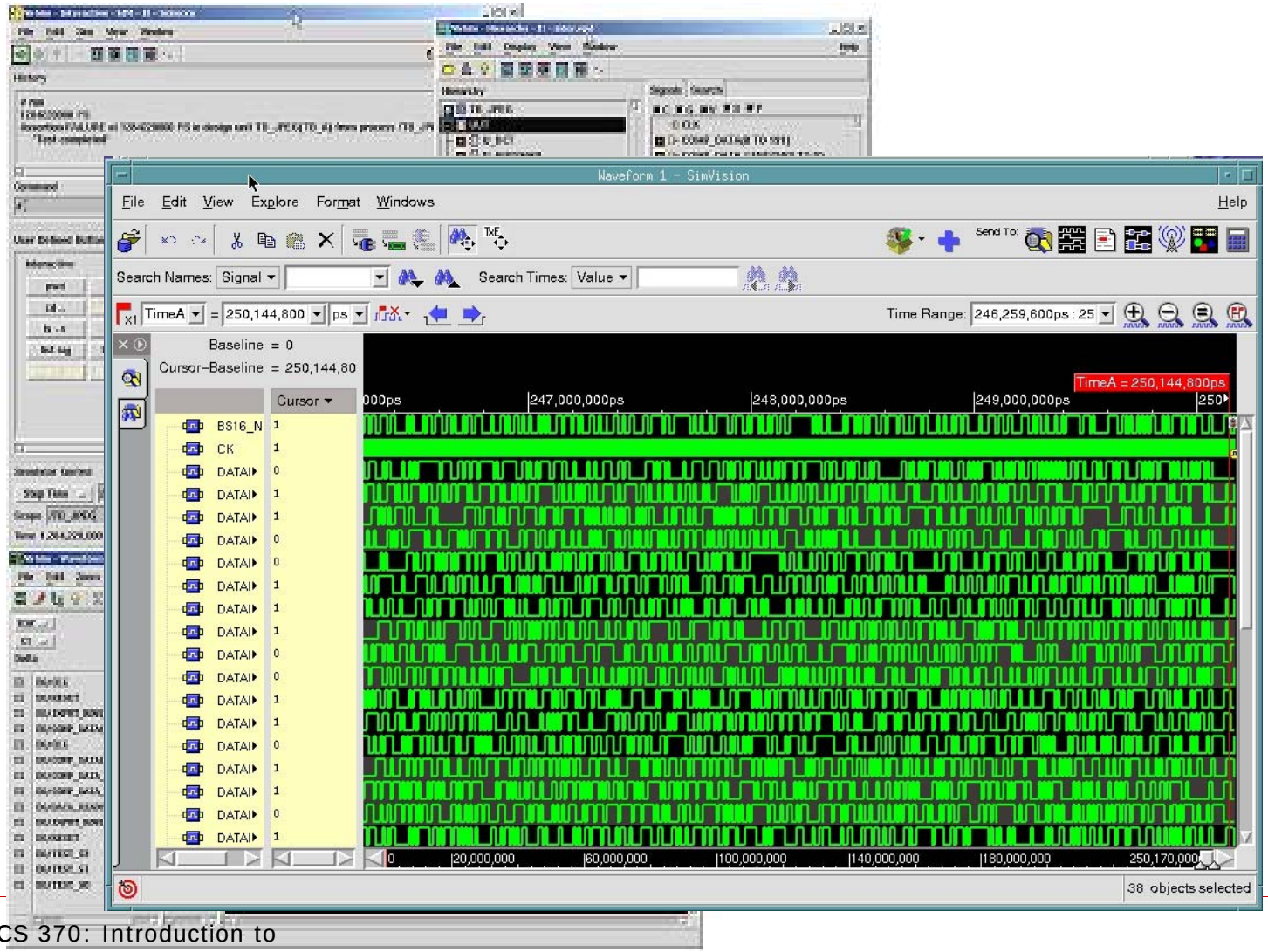


Timing in combinational circuits



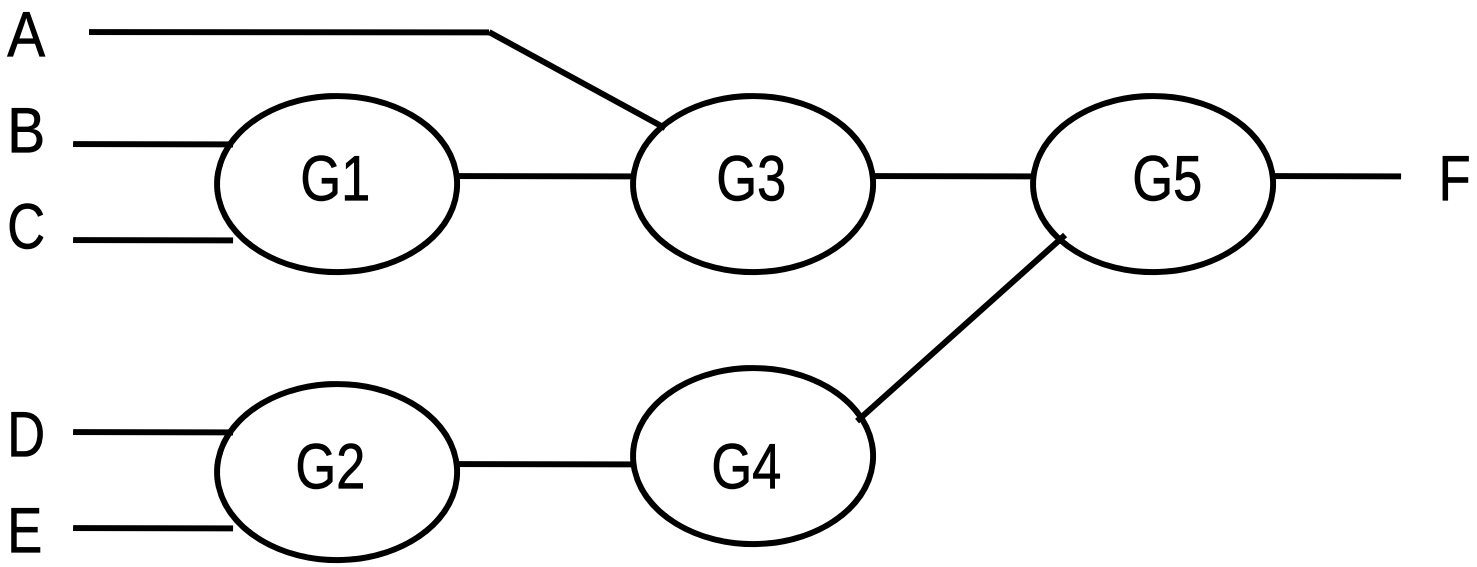
What is the input/output delay (or simply, delay) of the MUX?

Waveform viewers are part of a designer's daily life



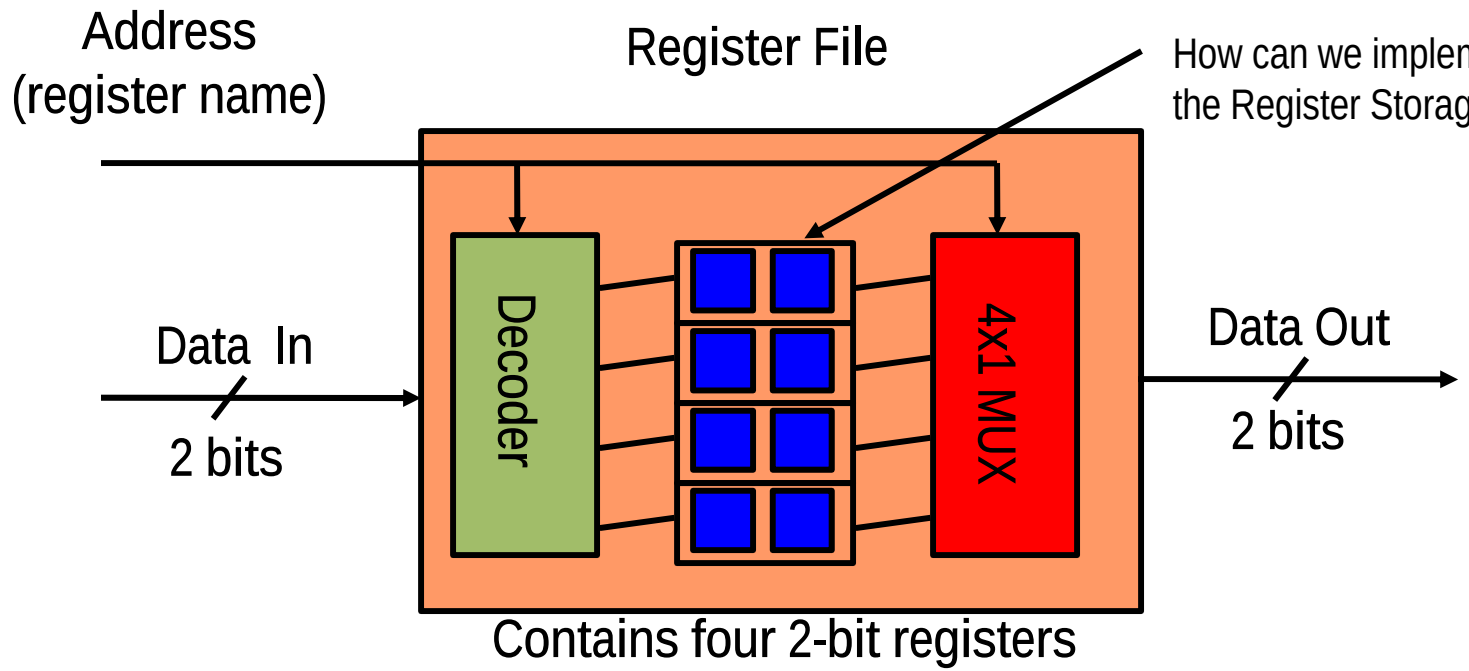
How about the delay of this circuit?

Each oval represents one gate, the type does not matter, every gate has the same propagation delay

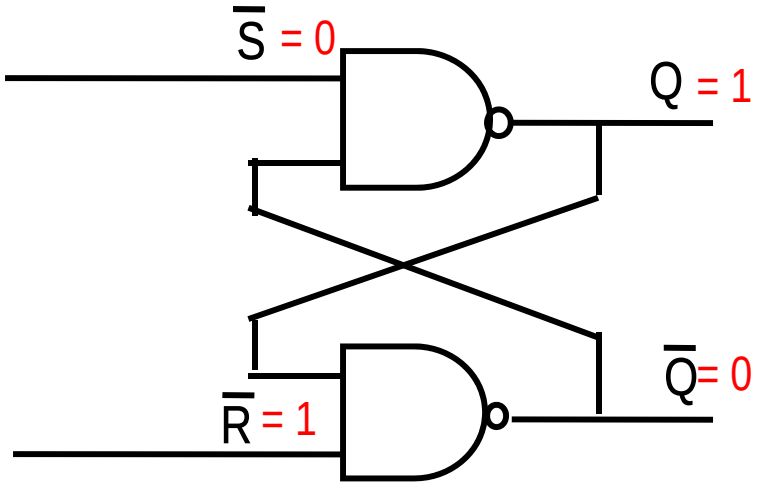


Sequential elements

Why do we need memory – example: REGISTER FILE



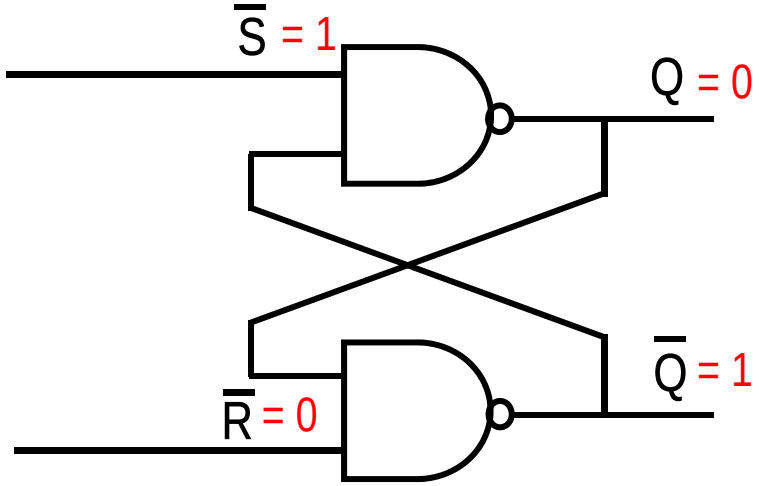
Let's look at the following circuit



S	R	Q	Q
0	1	1	0

What is the value of Q if $\overline{R}$ is 1 and $\overline{S}$ is 0?

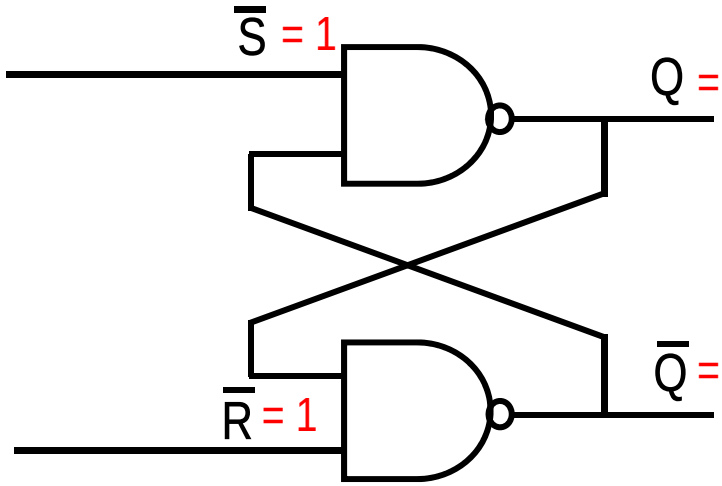
Building the Truth Table



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

What is the value of Q if R is $\overline{0}$ and S is $\overline{1}$?

A Basic Memory Cell

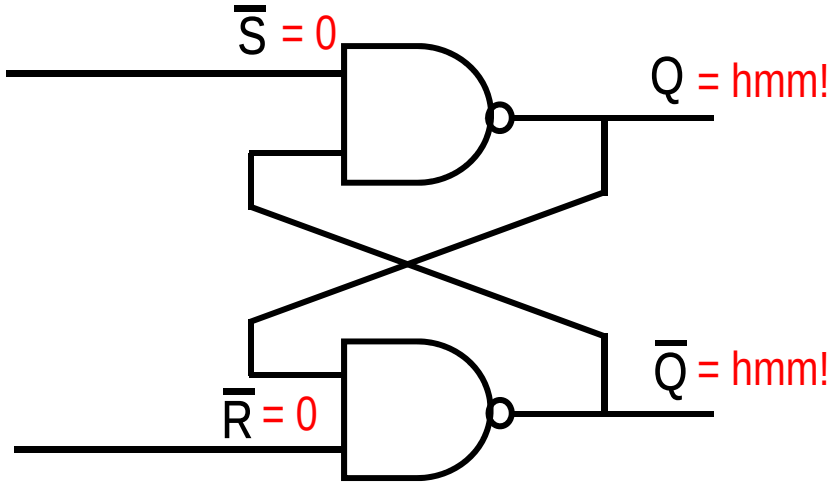


$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	1	1	0
1	0	0	1
1	1	Q	$\overline{Q}$

What is the value of Q if R is $\overline{1}$ and S is $\overline{1}$?

As long as R and S remain 1 then the value Q (and Q) will remain unchanged. This value is stored in this circuit. This is a basic memory cell.

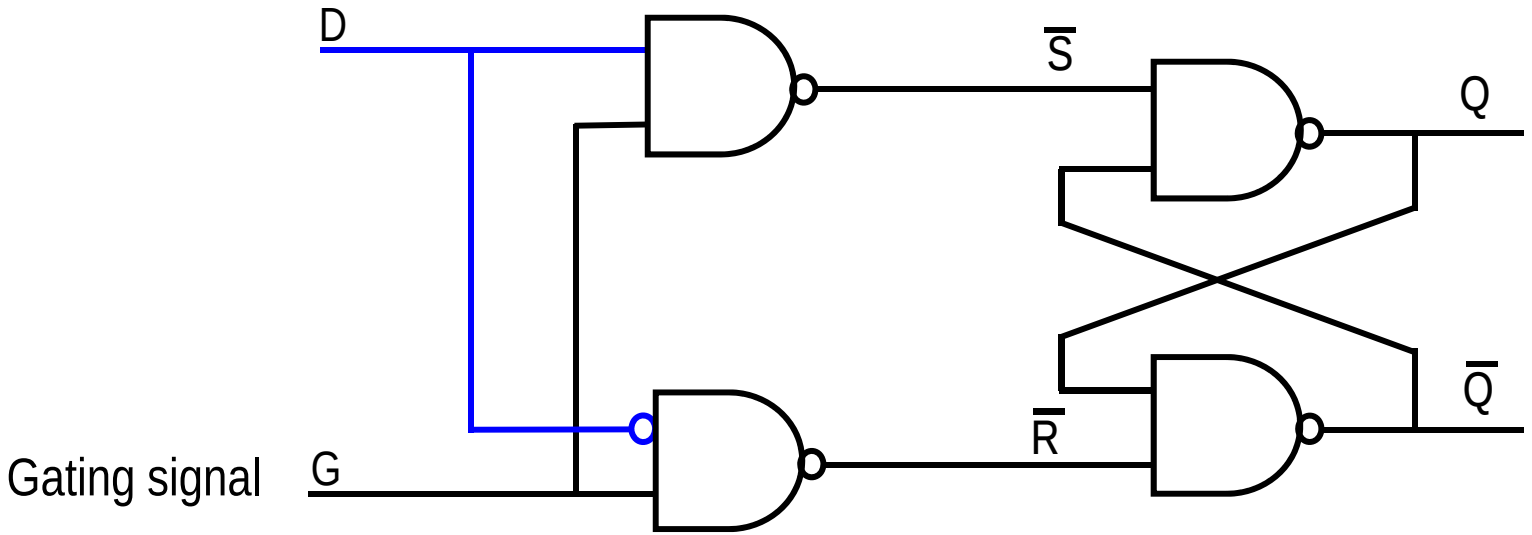
With unstable inputs 0,0



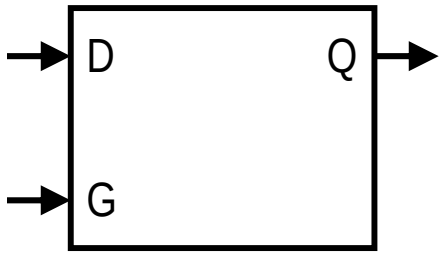
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	0	undef	
0	1	1	0
1	0	0	1
1	1	Q	$\overline{Q}$

What is the value of Q if R is $\overline{0}$ and S is $\overline{0}$?

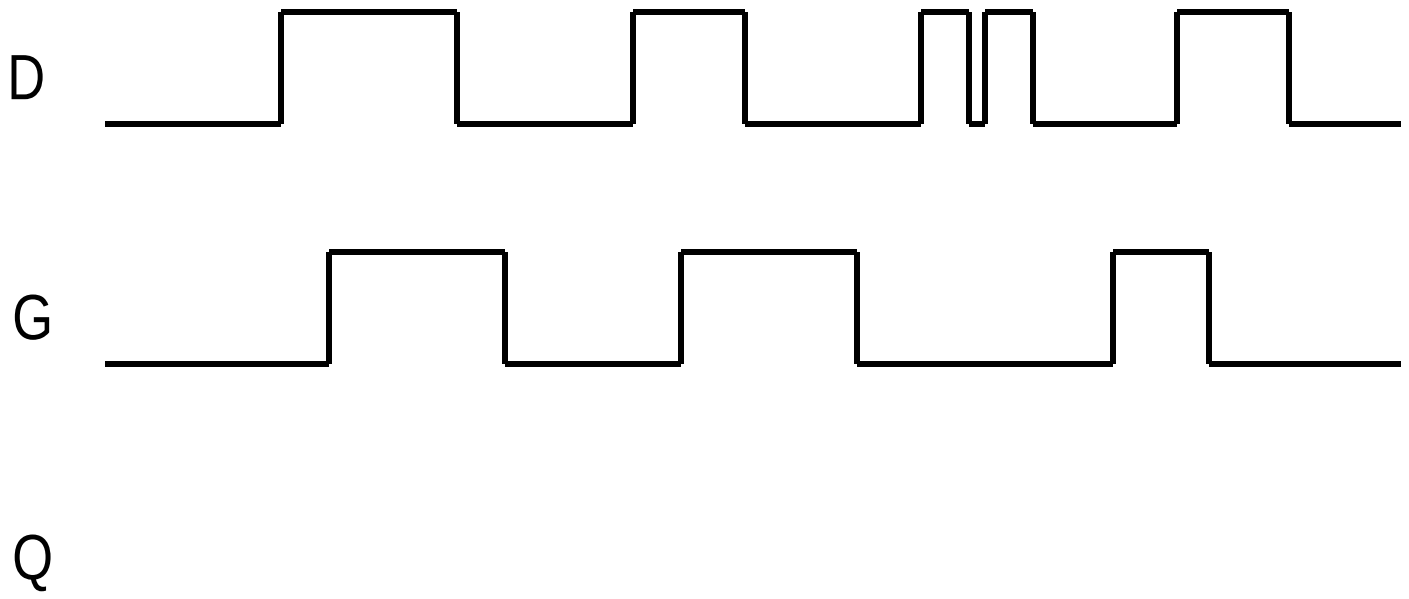
Transparent D Latch



D	G	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	Set state is retained
0	1	0	1	Next state is set
1	0	Q	$\overline{Q}$	
1	1	1	0	



Q?



Adding a clock to the mix

- We can design more interesting circuits if we have a clock signal
- The use of a clock enables a sequential circuit to predictably change state (and store information).

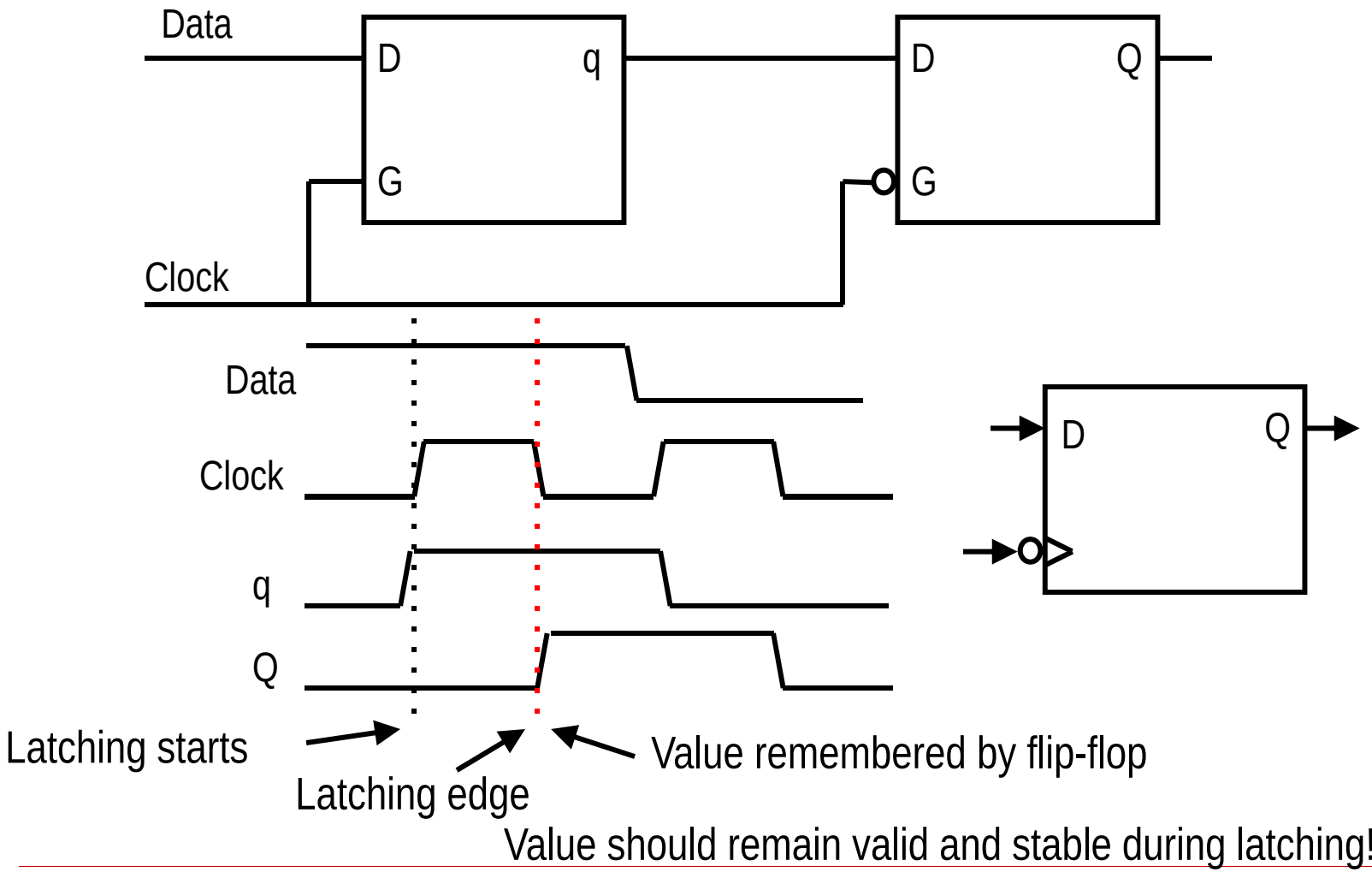
- A clock signal alternates between 0 and 1 states at a fixed frequency (e.g., 100MHz)
- What should the clock frequency be?

Clocks

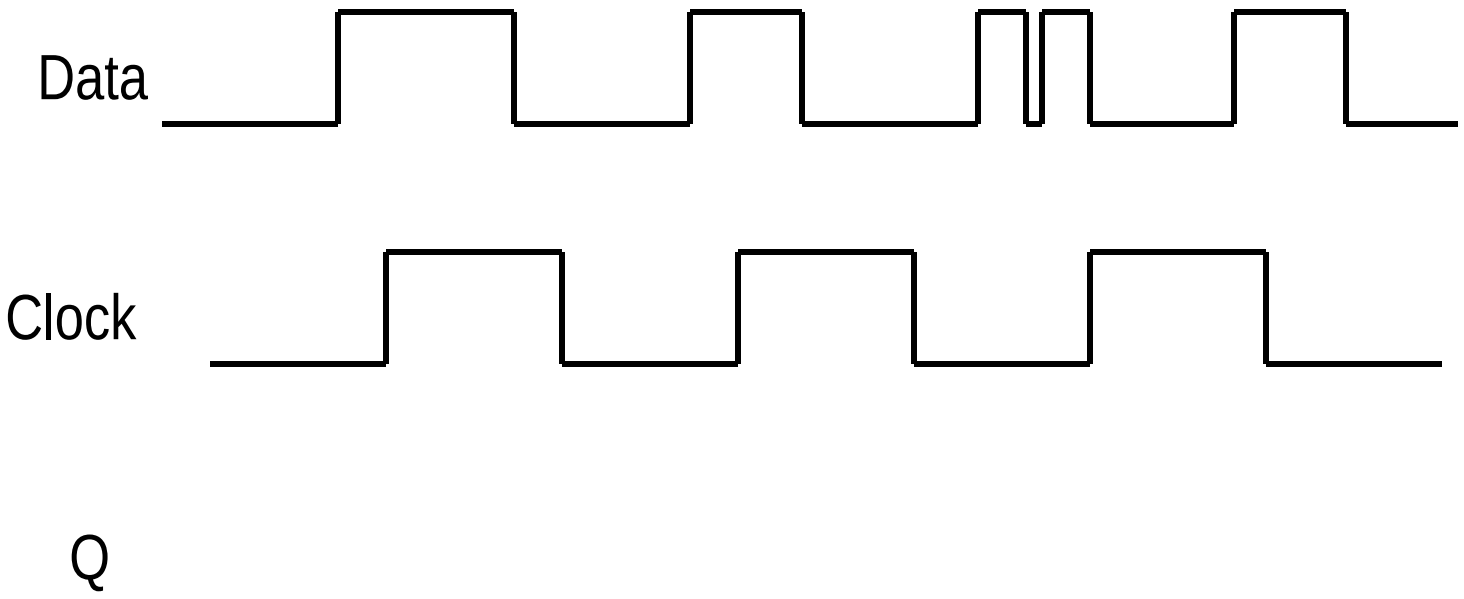
- Clock signal
  - Periodic pulse
  - Generated using oscillating crystal or ring oscillator
  - Distributed throughout chip using clock distribution net

- With clock signals we can create a new class of circuits called **sequential**
  - Output determined by inputs & previous state

Edge-Triggered D Flip-flop

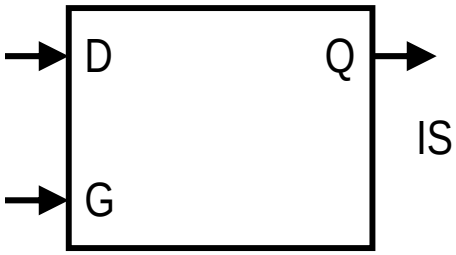


Q?

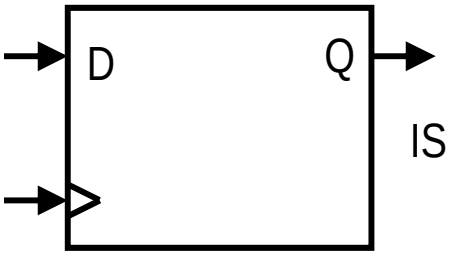


Why edge-triggered flip-flops?

- In edge-triggered flip-flops, the latching edge provides convenient abstraction of “instantaneous” change of state



IS LIKE A TOLL-GATE

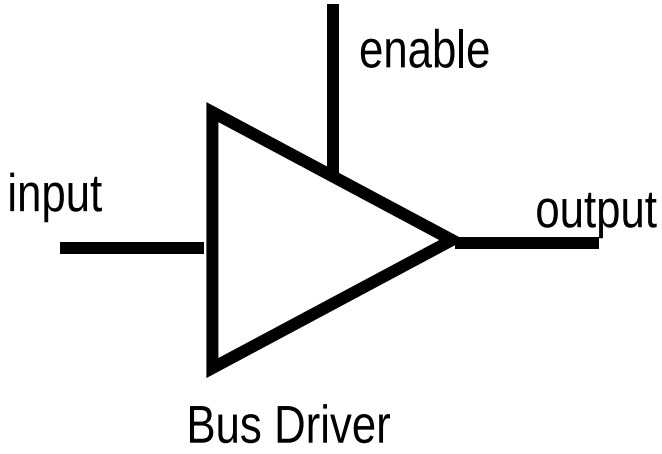


IS LIKE AN AIR-LOCK

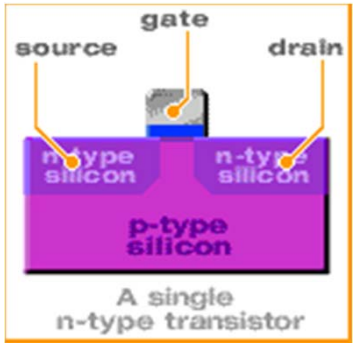
- Edge-triggered FF are useful to design the CONTROL of our processor
- For register file and memory, we need D-FF and one more piece…

Tri-state Logic

- The output of a gate can be any of three different states: one, zero or not connected
  - Need to disconnect the circuit. How?

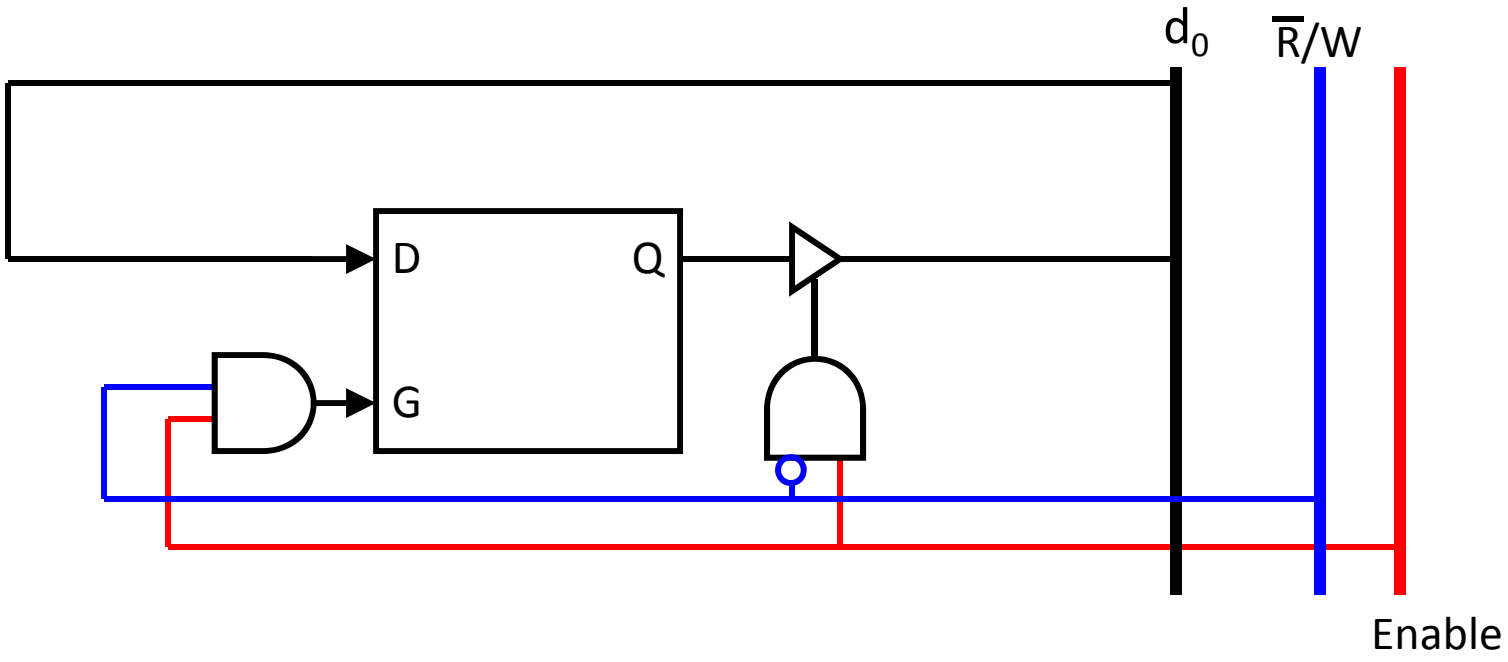


Bus Driver

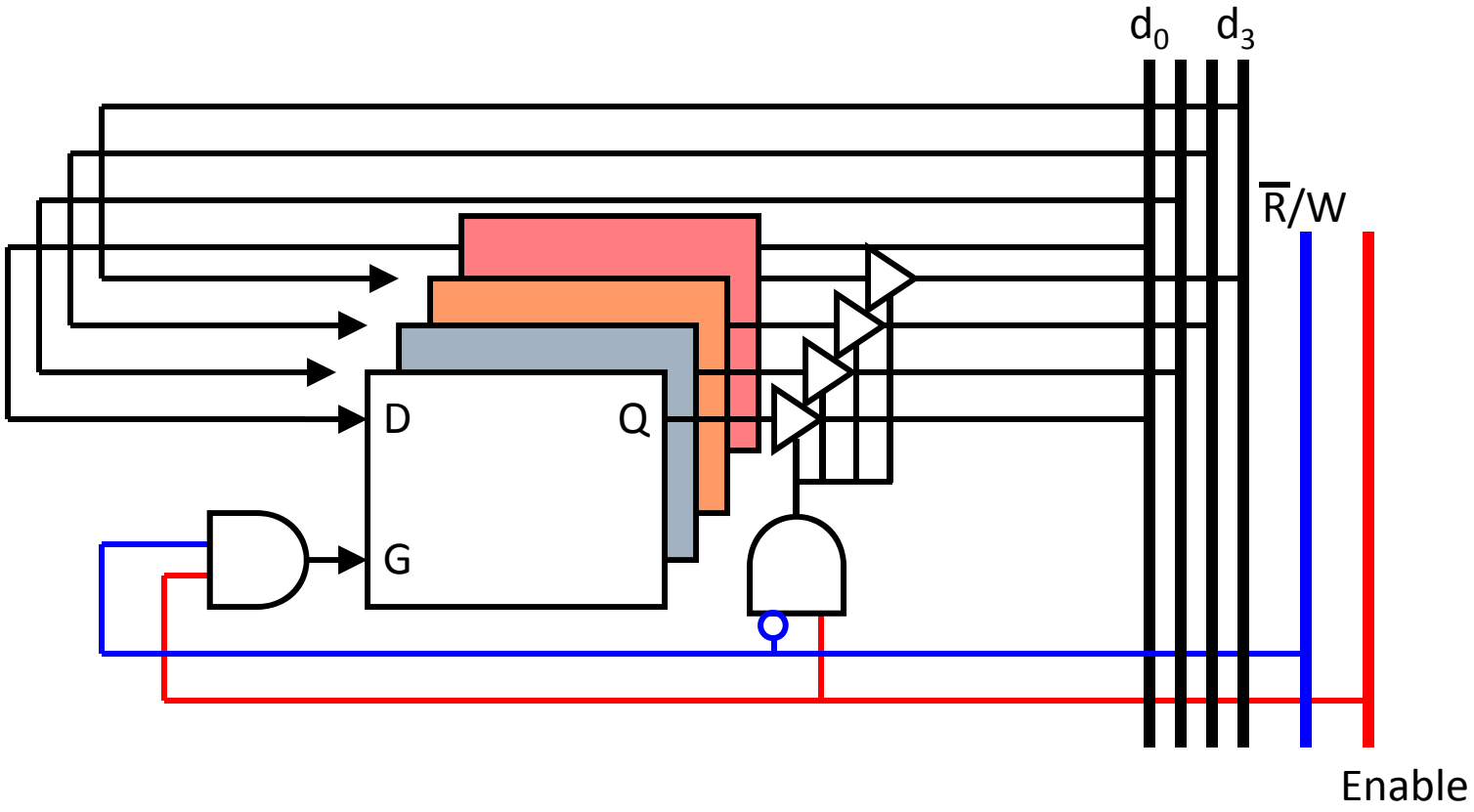


Implemented as a single transistor

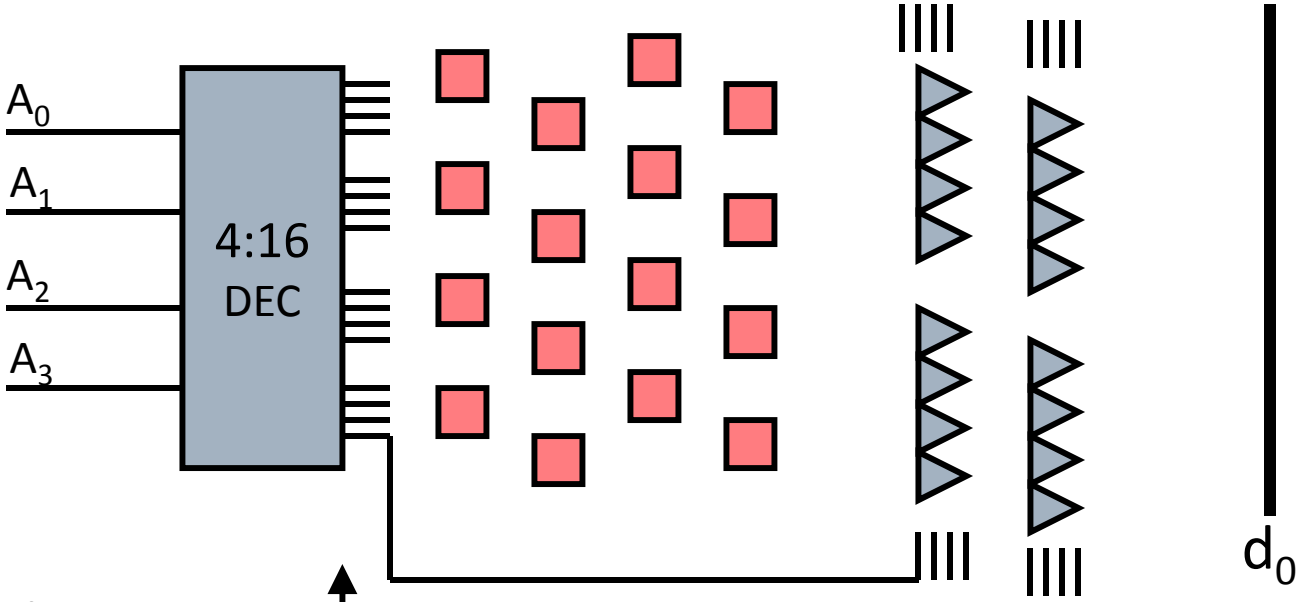
A Static Memory Cell



A 4-Bit Register



Addressing Memory Arrays

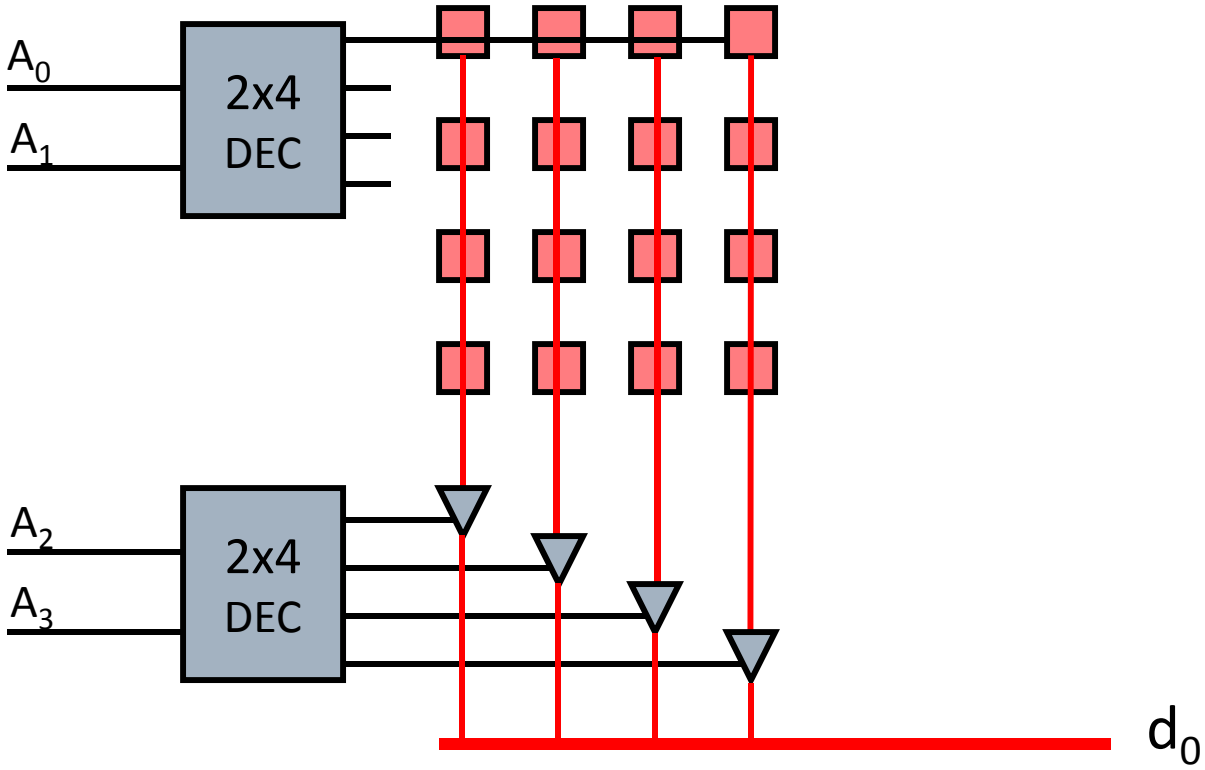


One of the 2^4 enable lines is turned ON

Implements 16:1 MUX

16x1 BIT MEMORY

A Scheme with Fewer Components

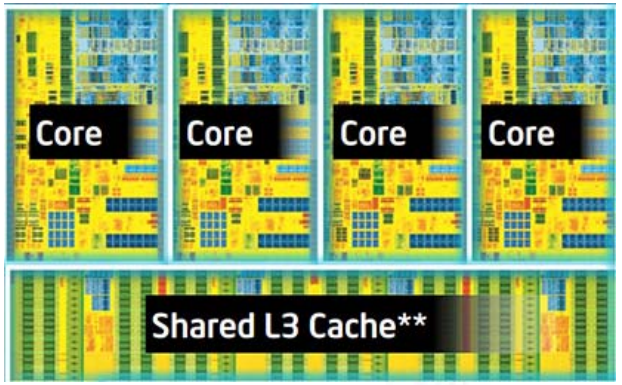


16 X 1 BIT MEMORY

Other Memories

- Static RAM (1-10's MB)
  - Built from sequential circuits
    - Takes 4-6 transistors to store 1 bit
    - Fast access (<< 1 ns access possible)

Intel Haswell



- Dynamic RAM (1-10's GB)
  - Built using a single transistor and a (leaky) capacitor
    - 1's must be refreshed often to retain value
    - Slower access than static RAM (10's ns to access)
    - Much more dense layout than static RAM

- ROM, PROM, Flash memory, PCM, memrister, etc. → Later
  - Even more dense, and more slow

Building Cool Stuff with Memory and Logic

- Build a custom controller for a vending machine.
- Controller, a.k.a. FSM

We could use a general purpose processor, but we might save money with a custom controller.

Take coins, give drinks.

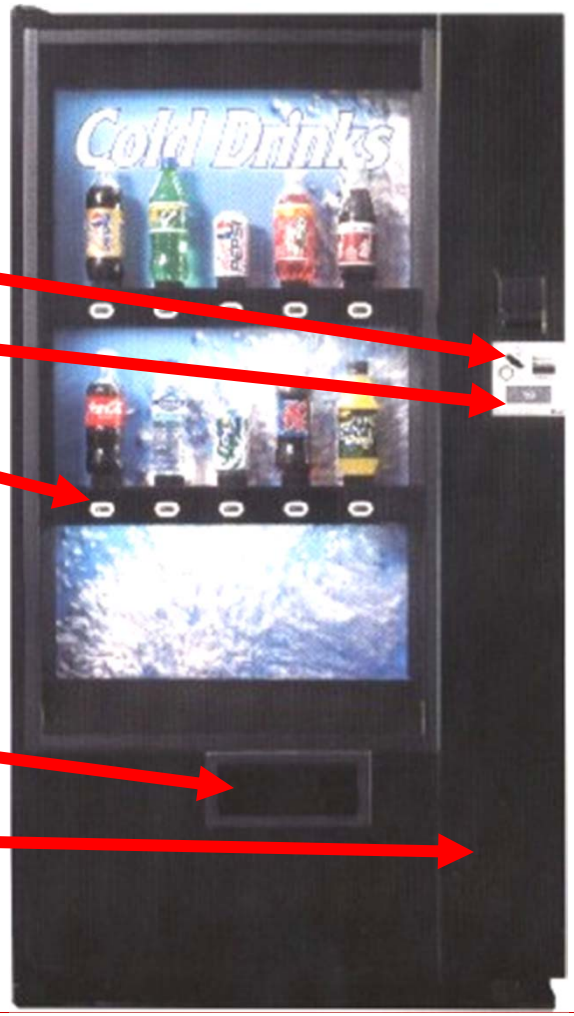
Input and Output

Inputs:

- coin trigger
- refund button
- 10 drink selectors
- 10 pressure sensors (to feel if drink available)

Outputs:

- 10 drink release latches
- Coin refund latch



Operation of the Machine

- All drinks are $0.75
- Accepts quarters only
- Once enough money received, a drink can be selected
- When refund is requested, release all coins received

No free drinks!

No stealing money!

Building the controller

- Finite State
  - Remember how many coins have been inserted and what inputs are acceptable
- Read-Only Memory (ROM)
  - Define the outputs and state transitions

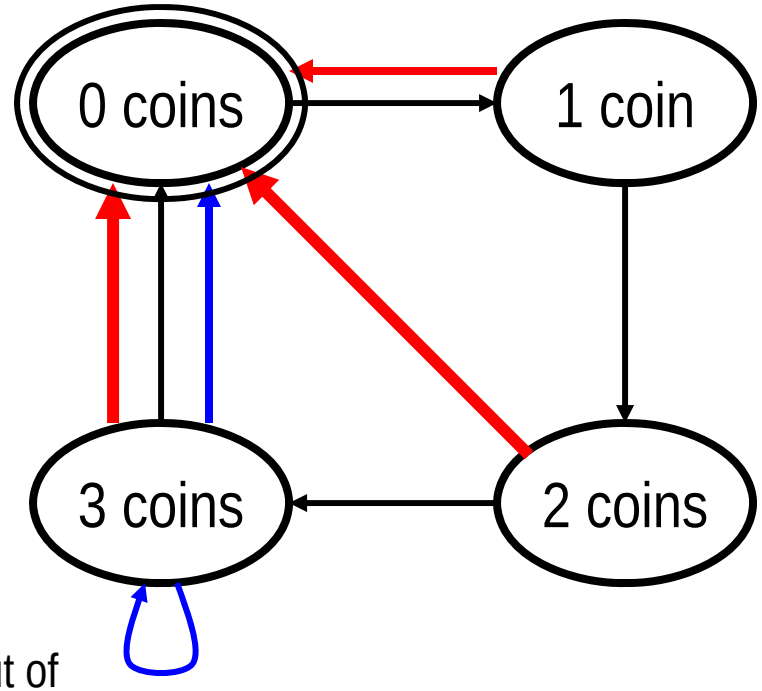
And/or

- Custom combinational circuits
  - Reduce the size (and possibly the 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
- Two flavors:
  - 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



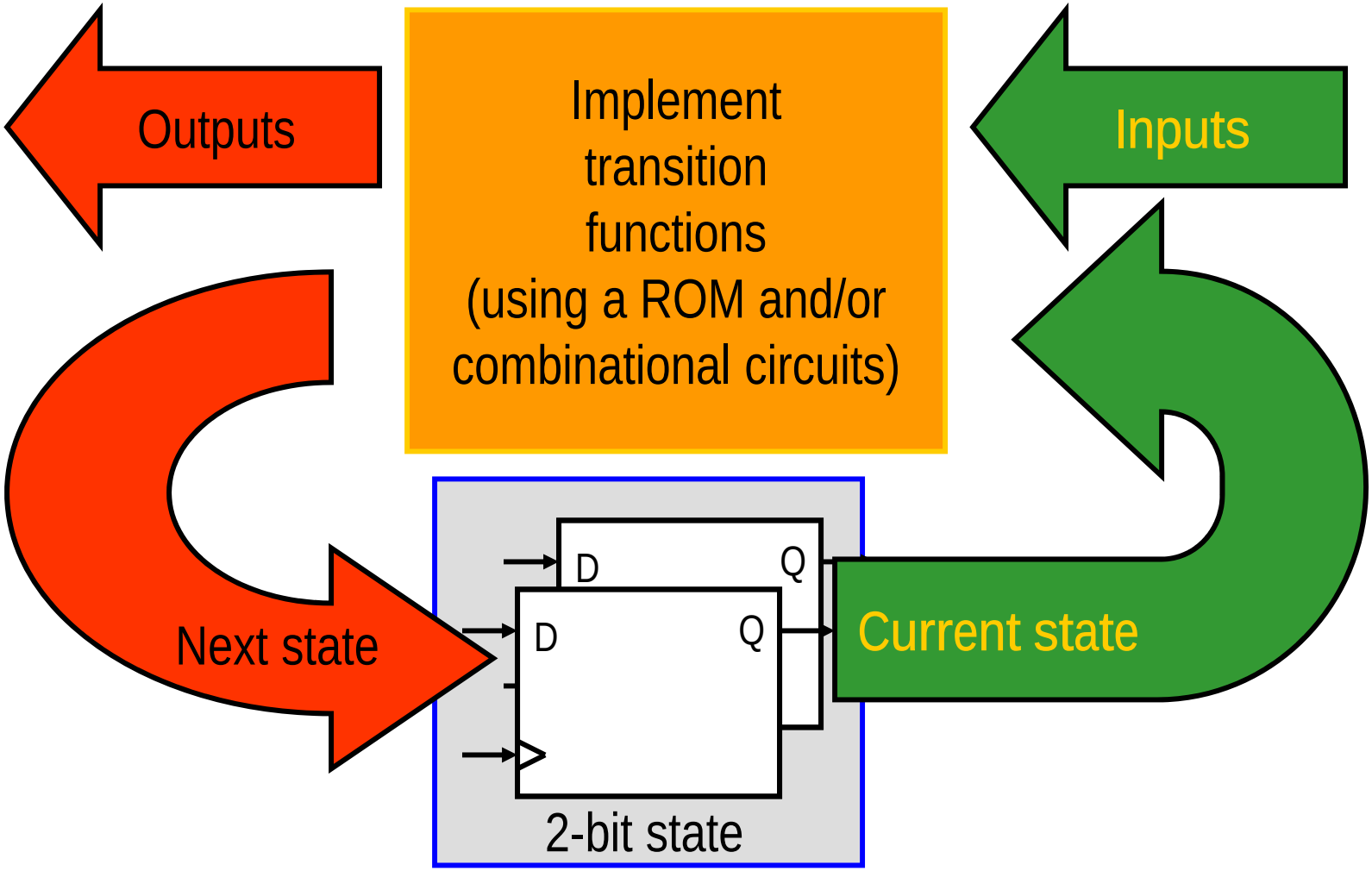
Ran out of specific drink selection

Coin received

Refund button

Drink Select

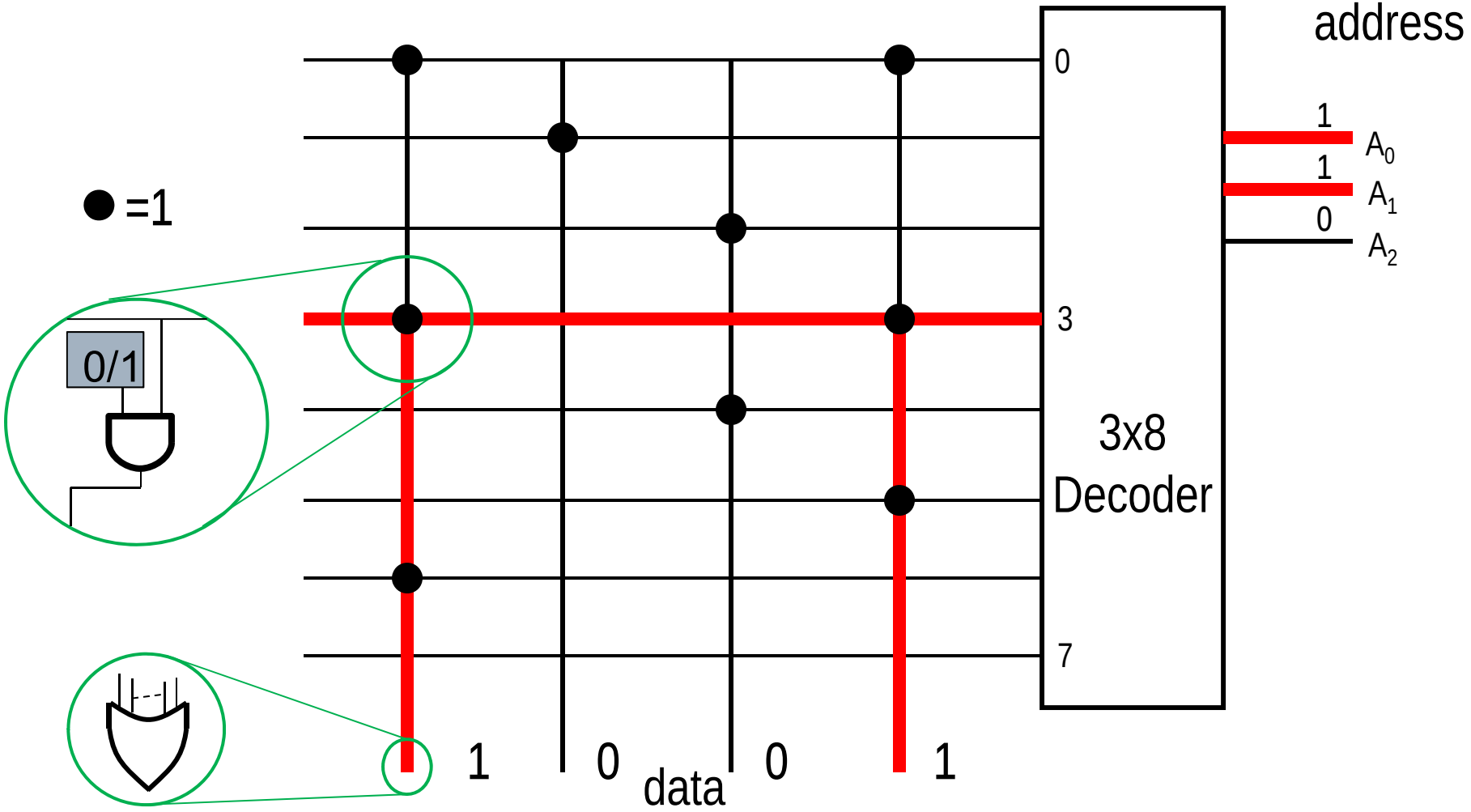
Implementing a FSM



ROMs and PROMs

- ROM = 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



ROM for Vending Machine Controller

- Use current state and inputs as address
  - 2 state bits + 22 inputs = 24 bits (address)
- Read next state and outputs from ROM
  - 2 state bits + 11 outputs = 13 bit (memory)
- 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!)

Some of the ROM contents

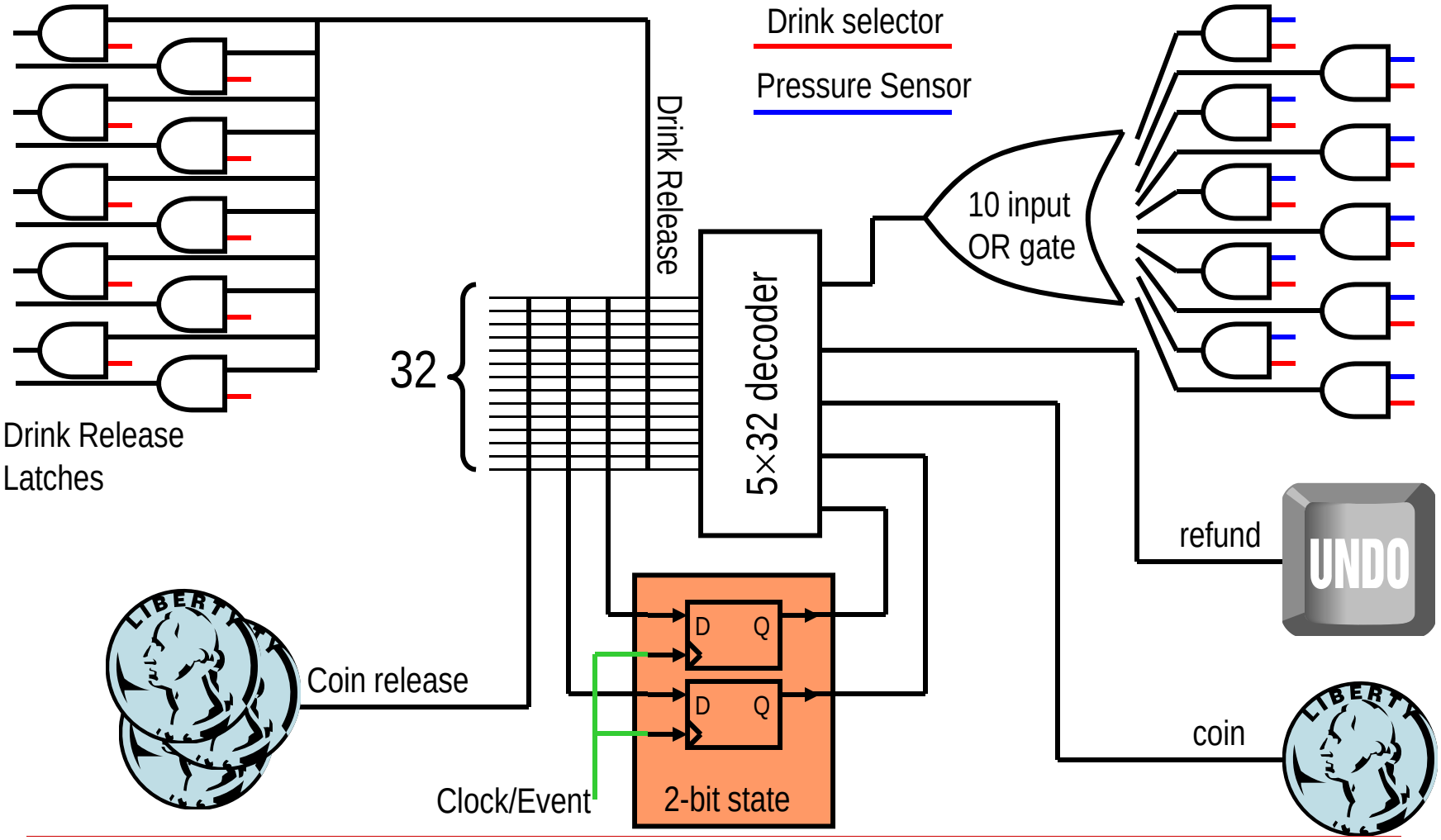
Current state	Coin trigger	Drink select	Refund button
0 0	0	0	0
0 0	0	0	1
0 0	0	1	0
0 0	1	0	0
0 1	1	0	0
1 0	1	0	0
1 1	0	1	0
1 1	1	0	0
… 24 more entries			

ROM address (current state, inputs)

Next state	Coin release	Drink release
… 24 more entries		

ROM contents (next state, outputs)

Putting it all together



Class problem

- What changes would you need to apply to the controller if
  1. the price of a soda is 1.00$
  2. you want to be able to accept dime and nickels, too
  3. and you want to be able to give change back

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

- Now, let's go build our first processor!