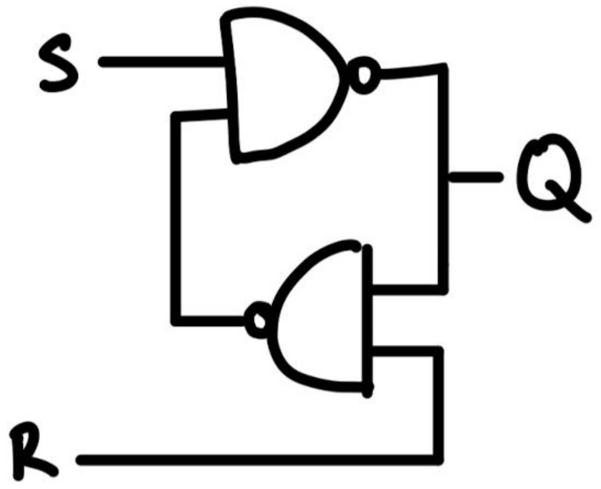
Recitation 10/23

Welcome back :)

R-S latch

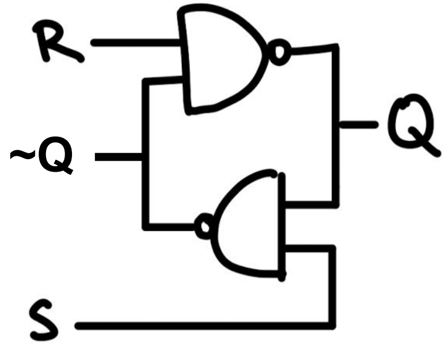
- Uses only 2 gates to hold a state in memory



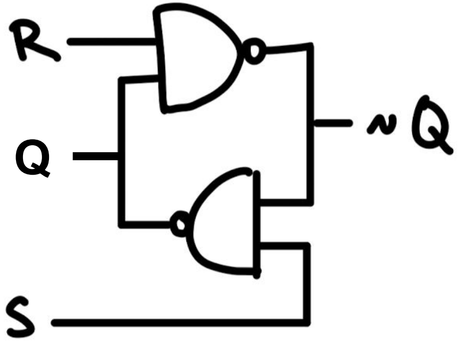
R	S	Q	Action
1	0	1	SET
0	1	0	RESET
1	1	1	HOLD from SET
1	1	0	HOLD from RESET

Question: Which are equivalent to the R-S latch?

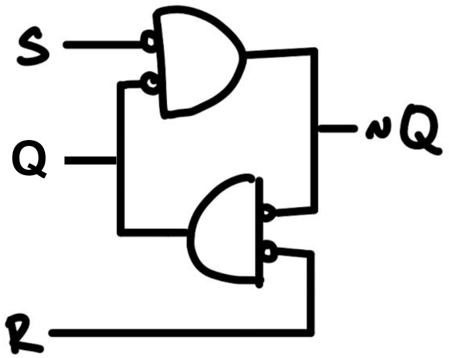
A)



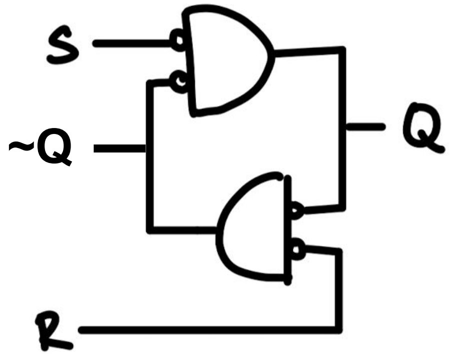
B)



C)

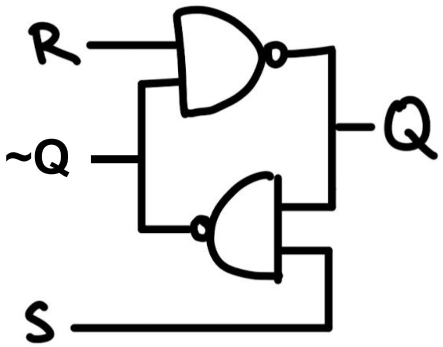


D)

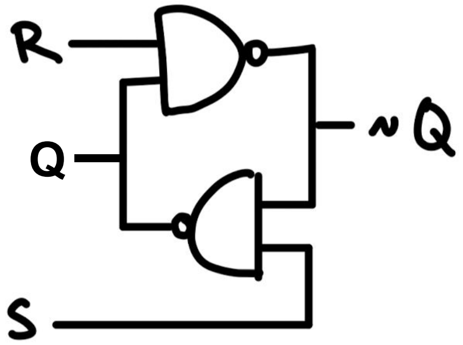


Question: Which are equivalent to the R-S latch?

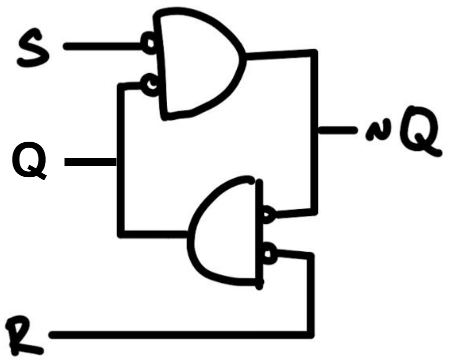
A)



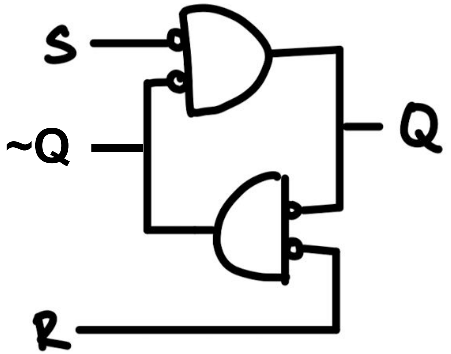
[✓ B)]



C)



D)



RS latch: why R and why S? (not going to be tested)

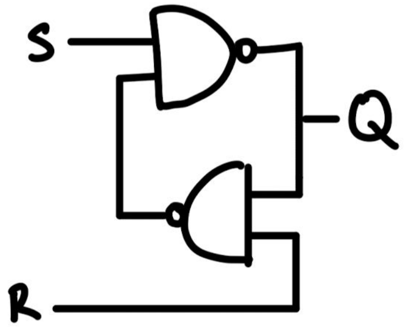
- S = set
- R = reset

But if that’s the case, Q is reset when S = 1 and R = 0. Wouldn’t it make more sense if Q is reset when S = 0 and R = 1?

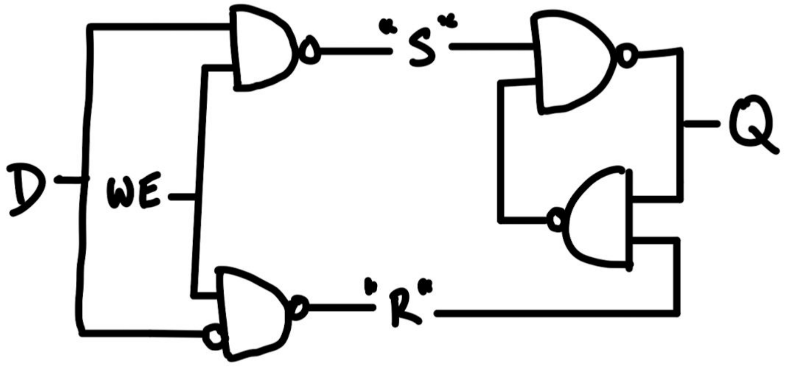
R-S Latch implemented with NAND gates create an “**active low” R-S latch** because **Q is active (1) when S is low (0)**.

R-S Latch implemented with NOR gates create an **“active high” R-S latch** for the opposite reason: **Q is active (1) when S is high (1)**

R-S latch and D-latch comparison



R-S Latch



D-Latch: notice that D and WE are programmed to “make” S and R such that data D is only being held in Q if WE is 1. (what does the truth table for D, WE, R, and S look like?)

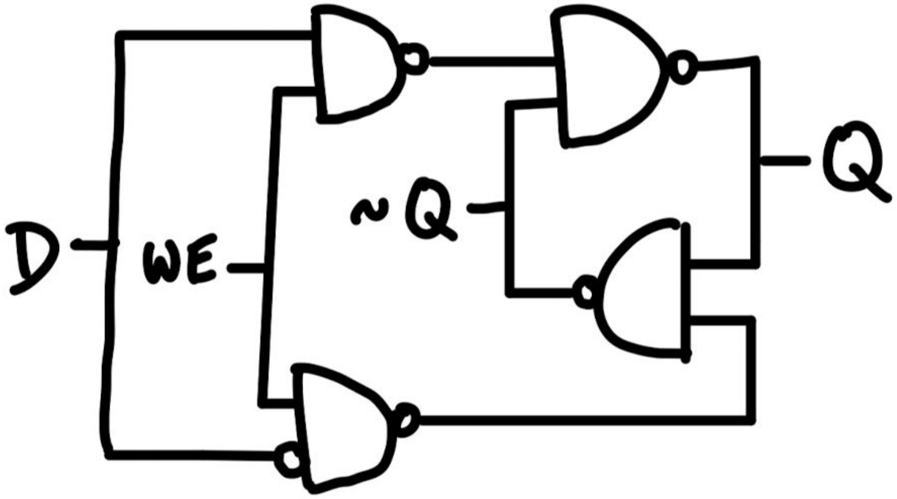
Question: D-Latch + Gate Delay

If an AND gate has a 3ns gate delay and NOT gate has a 1ns delay, how long does the D-latch take to stabilize (i.e. all values stop changing) if we transition from state 1 to state 2?

State 1: D = 0 WE = 0, Q = 1

State 2: D = 0 WE = 1, Q = ?

What is Q in State 2?

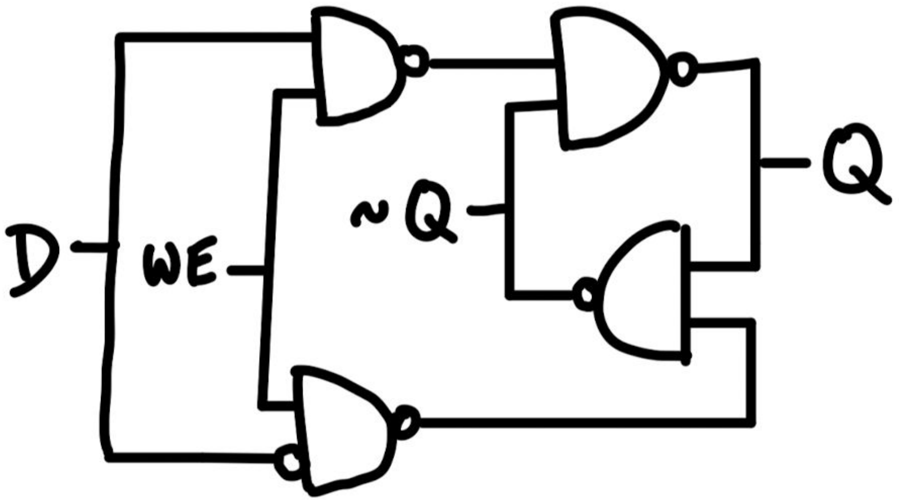


Question: D-Latch + Gate Delay - Solutions

If an AND gate has a 3ns gate delay and NOT gate has a 1ns delay, how long does the D-latch take to stabilize (i.e. all values stop changing) if we transition from state 1 to state 2?

State 1: D = 0 WE = 0, Q = 1

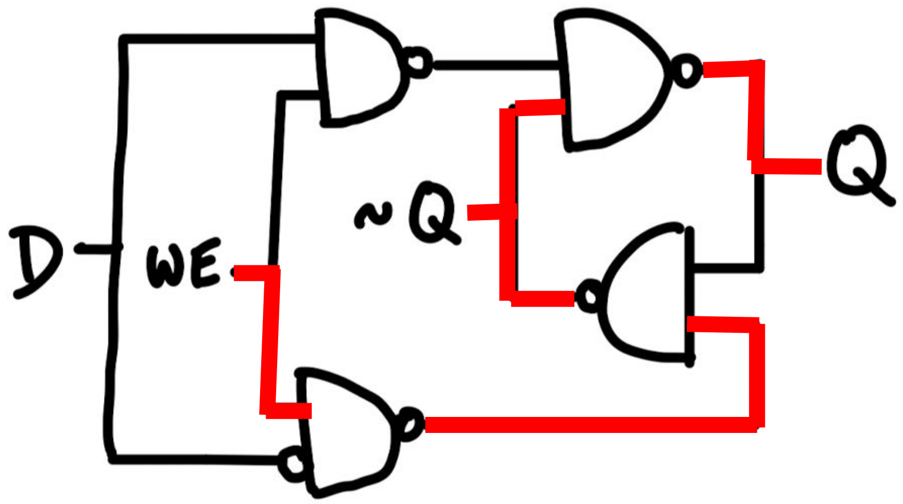
State 2: D = 0 WE = 1, **Q = 0**



Question: D-Latch + Gate Delay - Solutions

Since D does not change, the longest path does not include the NOT gate that D first encounters in the lower branch before hitting the NAND gate.

The longest path for changing values is WE -> Q through the lower branch passing through ~Q, with the total cost being 3(1ns) + 3(3ns) = 12ns

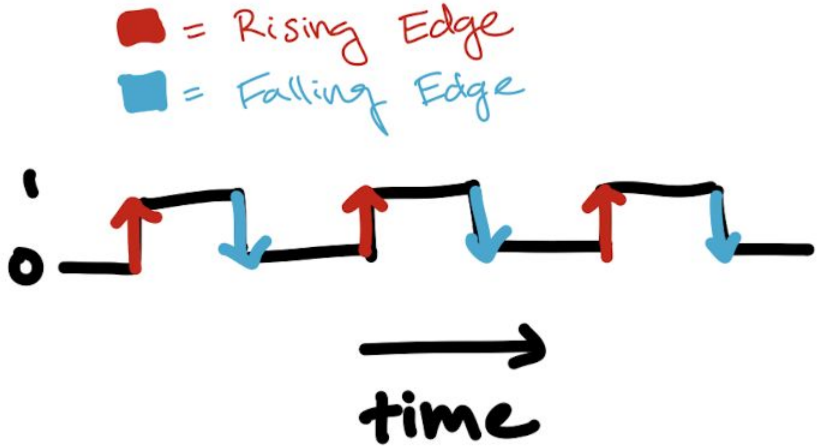


The clock:

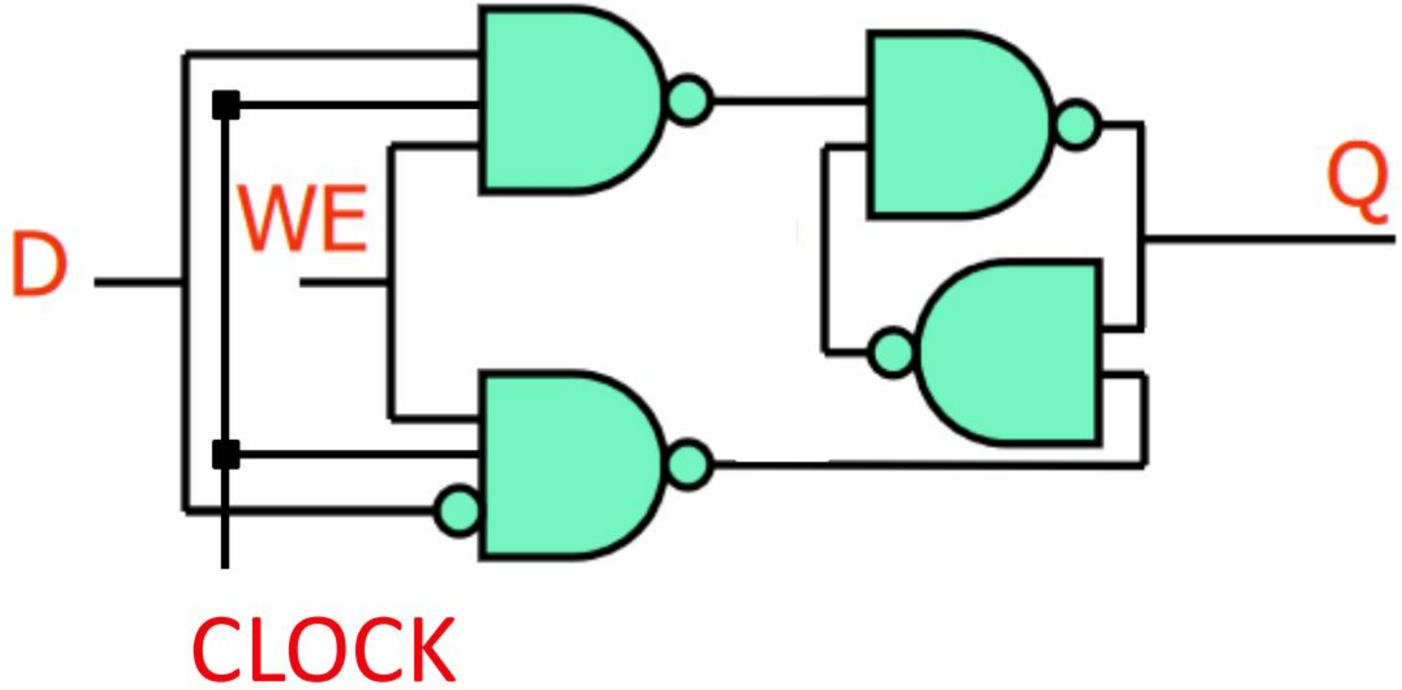
- switches between 0 and 1 at a fixed rate
- Determines (sometimes along with WE) when values are written to memory

Tip: be sure to know the difference between frequency and period!

Period != how long a clock stays high before dropping to low



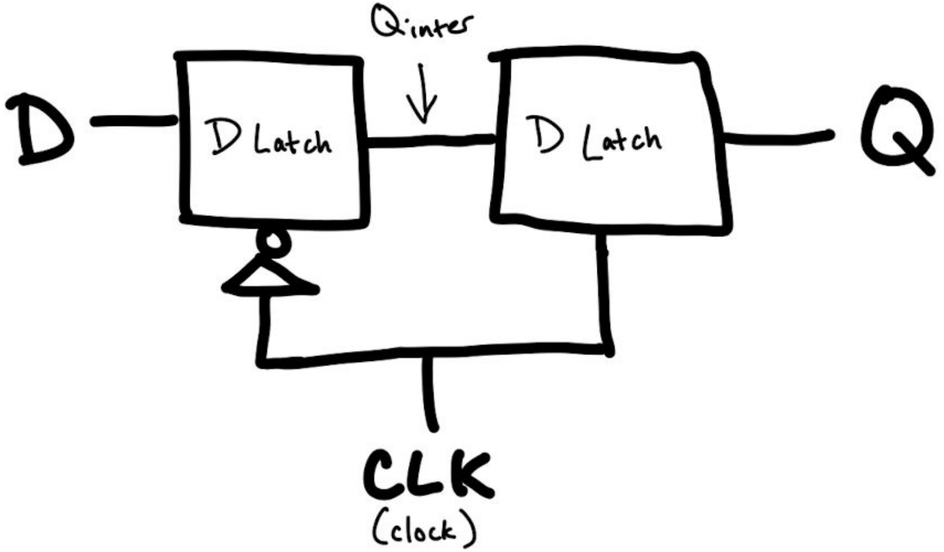
D Latch with Clock



Making the D Flip Flop

Why is the D-Latch on the left called “transparent low,” and the right D-Latch is the “transparent high?”

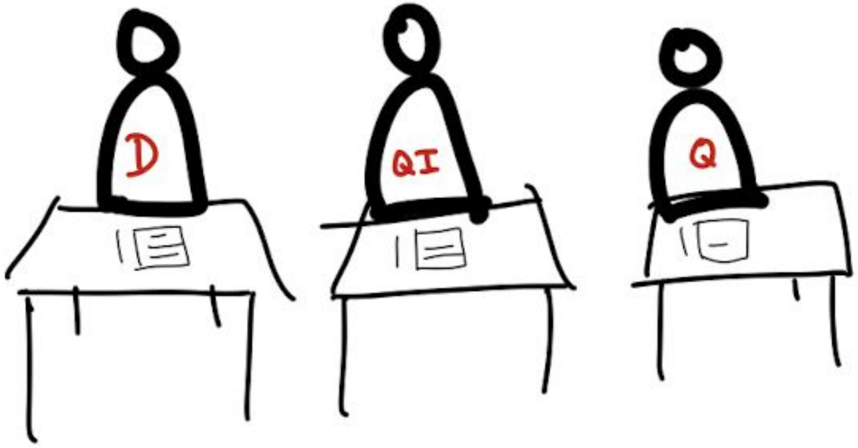
(note that this diagram does not contain WE, so assume CLK = WE)



Story time

D Flip Flop: relation between D, Q_inter, Q, and CLK

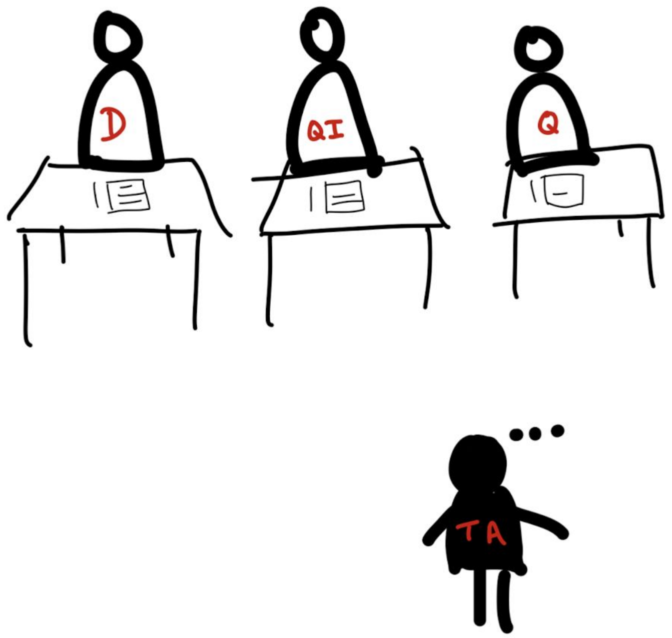
Imagine these four variables as a recitation of 3 students and one TA (CLK). The students sit next to each other (bc they're friends) on the same row, with D on the left, Q on the right, and Q_inter (which I'll now call QI for intermediate) in the middle.



D Flip Flop: relation between D, Q_inter, Q, and CLK

D understands the material being covered by the TA,, while Q and QI do not.

D wants to help her friends out, but QI and Q are easily embarrassed and do not want the TA to know that they are confused.



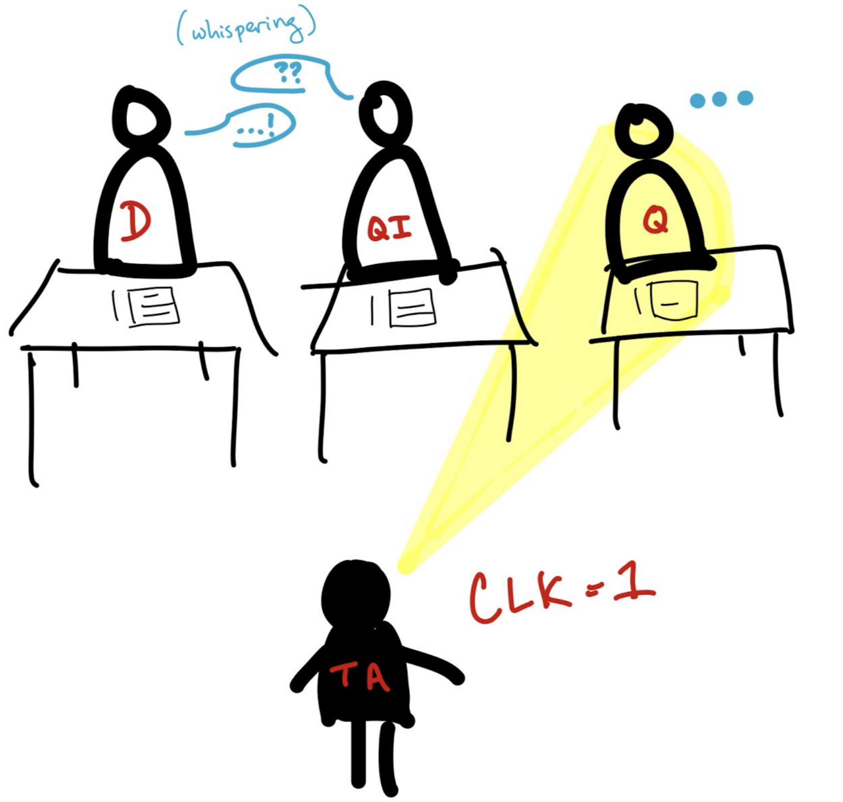
D Flip Flop: relation between D, Q_inter, Q, and CLK

Let’s say when CLK is 1, the TA is looking away from D, but is looking at Q. This allows QI ask D questions and get some answers (which D allows since the attention is also not on her)

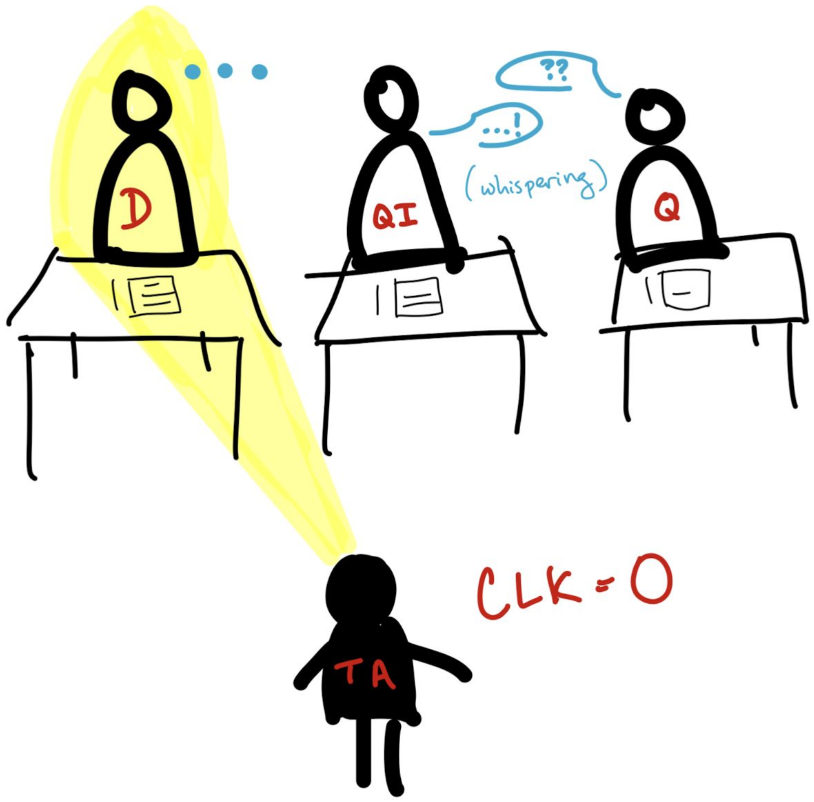
And when CLK = 0, the TA is paying attention to D, so QI can no longer ask D more questions. But! QI can now talk to Q and relay the information that they just received from D, since the TA is not looking at them.

And this system only works if CLK’s switches are slow enough to allow time for the asker to get all the answers to their questions.

QI copies D



Q copies QI



D Flip Flop: relation between D, Q_inter, Q, and CLK

To recap: how do QI and Q get D's answers?

When CLK = 1, QI can copy D, but Q cannot copy QI.

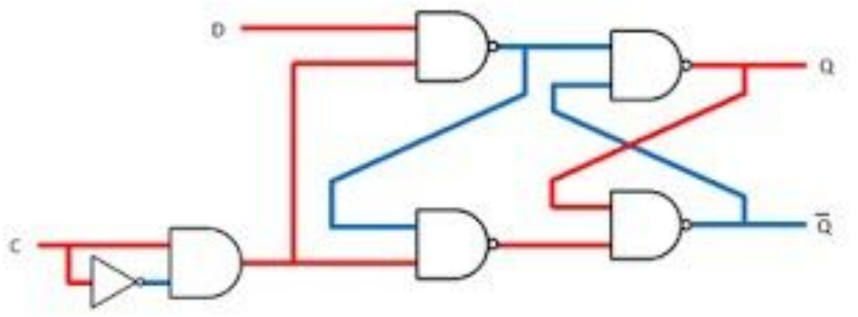
When CLK = 0, QI can no longer copy D. But QI can share what they've recently learned to Q, so Q will copy QI.

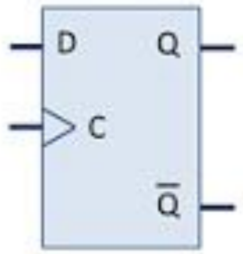
Note: as TA's, this story is not meant to discourage students asking questions in lecture or recitation!!

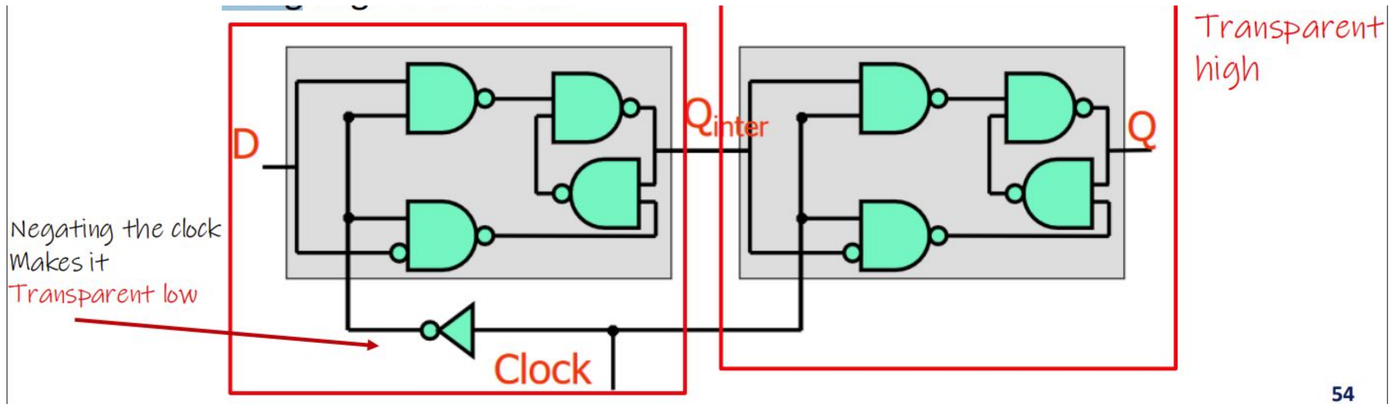
This is just a slightly silly story that helps me remember when values are passed from D to Q_inter to Q in a D Flip Flop.

End Storytime

Timing Diagram with Flip flop



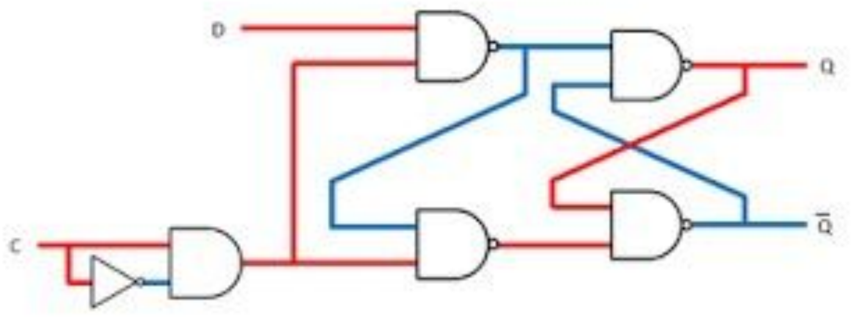


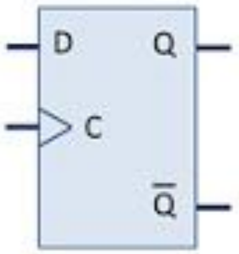


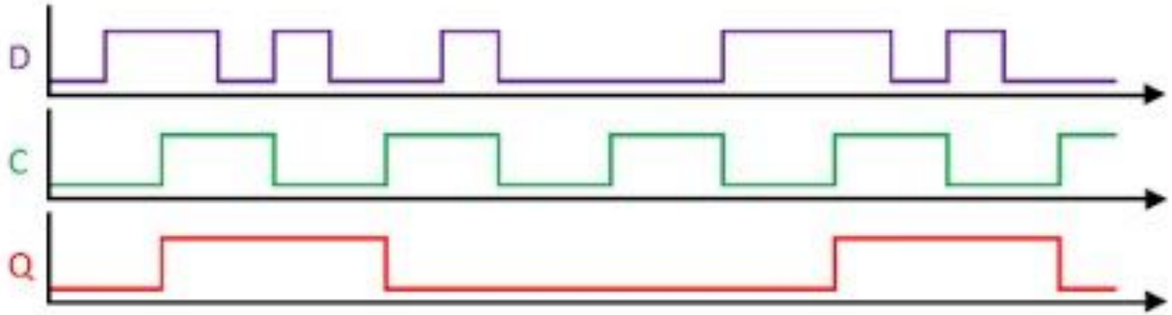
Transparent high

Negating the clock Makes it Transparent low

Timing diagram with flip flop: solution







What clock frequency is required to write 32KB of data to a D flip-flop in 15,000 ns if the data is written sequentially, one bit per clock cycle? (No need to read for this)

Hint: (32KB = 32,768 bits)

What clock frequency is required to write 32KB of data to a D flip-flop in 15,000 ns if the data is written sequentially, one bit per clock cycle?

Number of Clock Cycles Needed: Since the data is written sequentially, and each clock cycle can write 1 bit, the number of clock cycles required to write 32,768 bits is 32,768.

Total Time: The total time to write the data is 15,000 ns.

Clock Period (T): The clock period, which is the duration of one clock cycle, is given by: 15,000 ns / 32768ns = 0.457ns

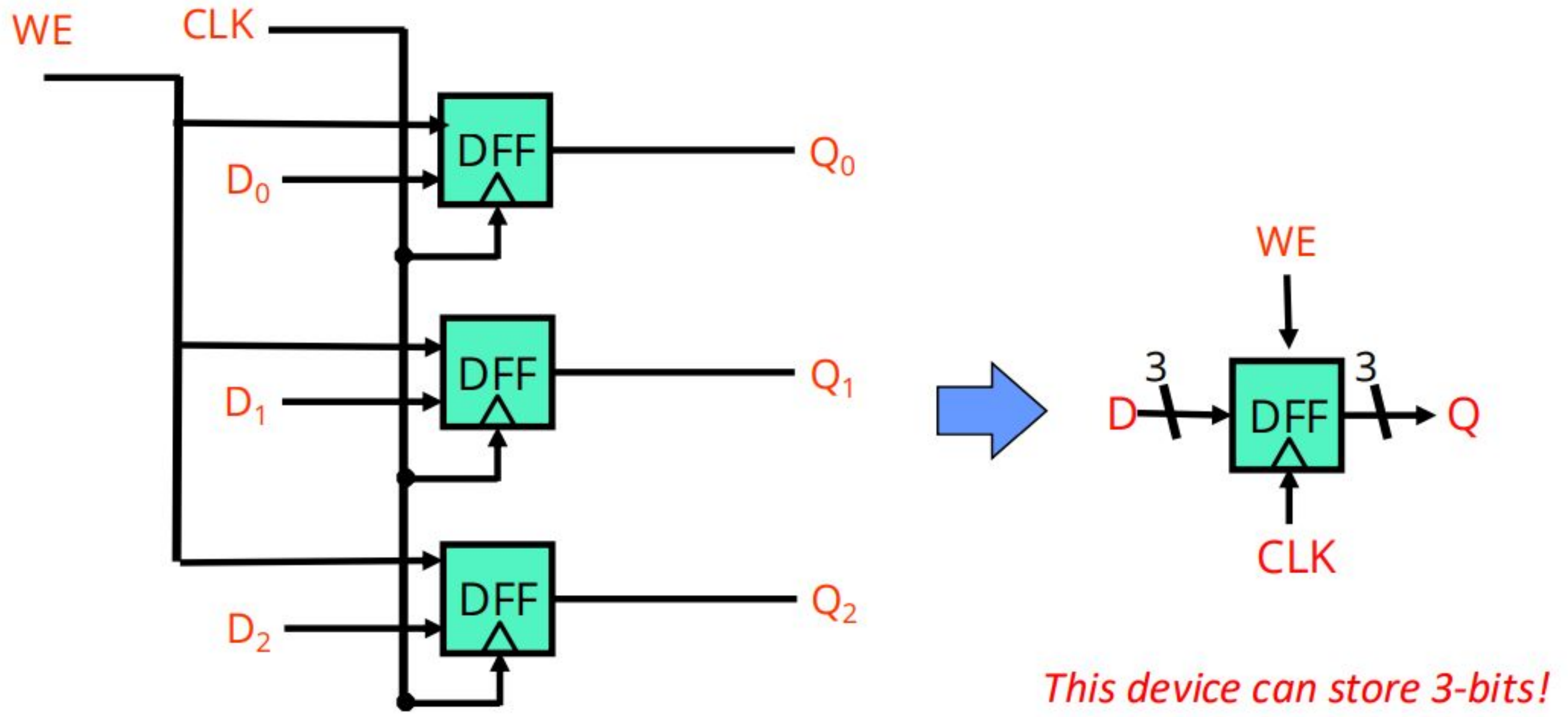
Clock Frequency (f): The clock frequency is the reciprocal of the clock period: 1/0.457ns = 2.19 GHz

Registers

Why is a DFF useful?

IT STORES STUFF!

From lecture:

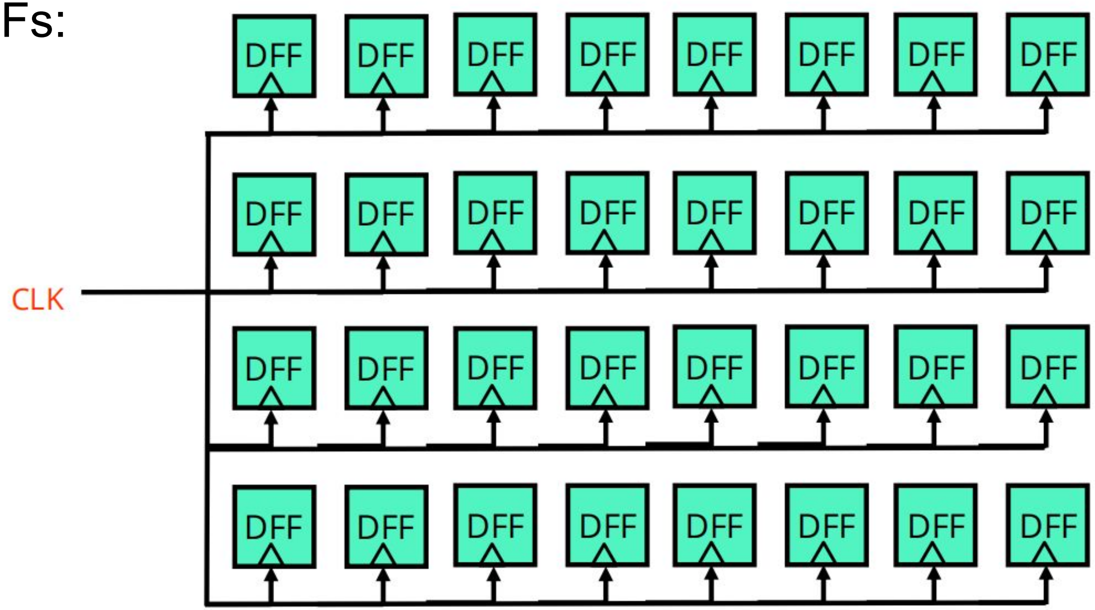


This device can store 3-bits!

Let’s say I want to store a 32-bit number…

Let’s say I want to store a 32-bit number…

- I could use a DFF to represent each bit…
- I would use 32 DFFs:



This looks scary but it’s actually not

We can abstract these 32 DFFS into a “Register”, which holds a 32-bit value:

Reg = 0010 0100 0111 1000 0101 1010 1111 0000

Memories…

In this class, memory is just a really long array of stuff

0	x00
1	xFA
2	xCC
3	xDA
4	xDA
5	xBA
6	xBE
7	x01

Let’s look at some basic memory architectures from lecture

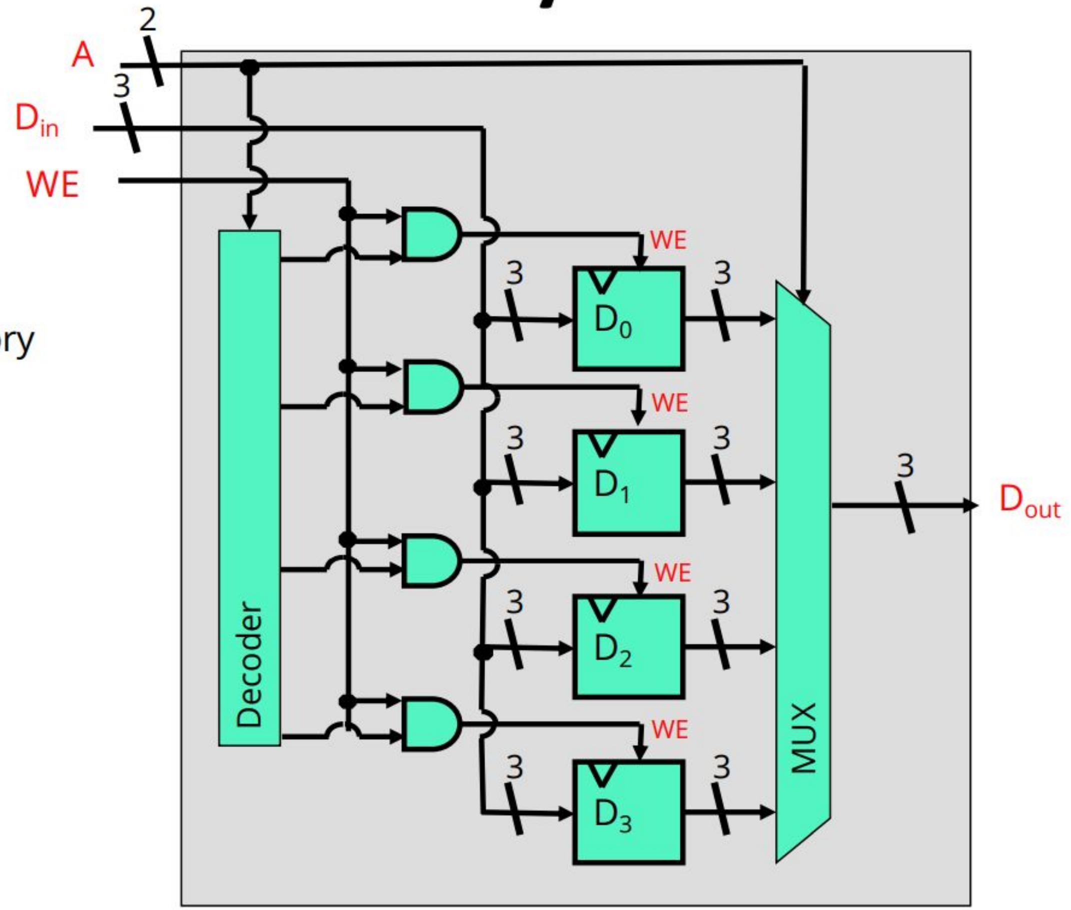
2^2 by 3-bit writeable memory

Write operation

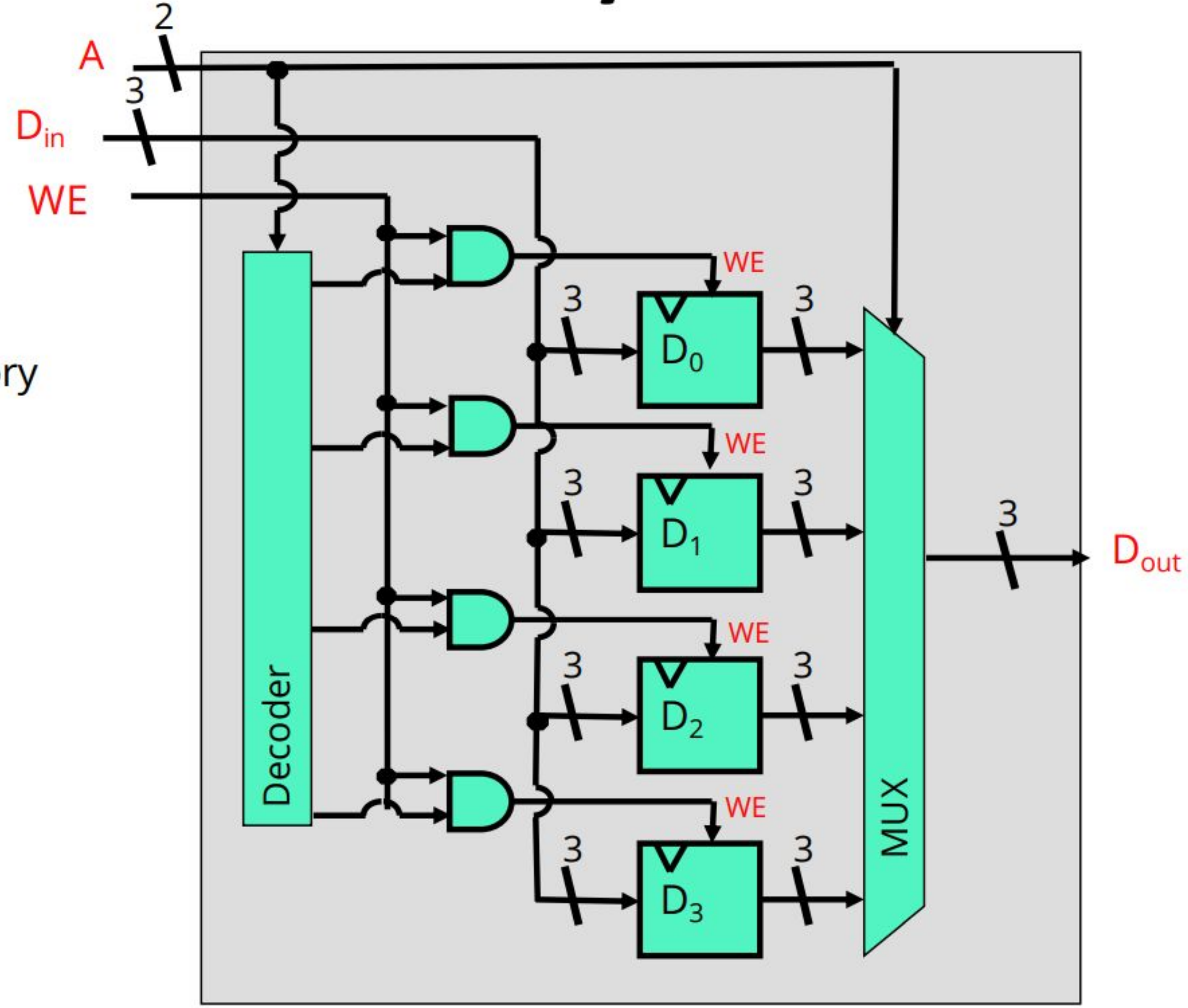
Limitation:
You can only read or write at any given time

This is called "single port" memory

From Lecture

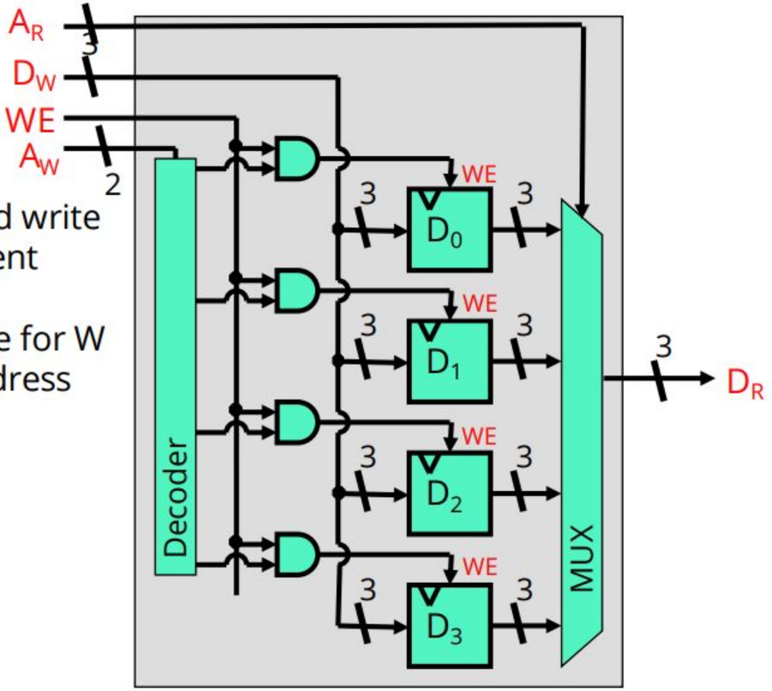


ry



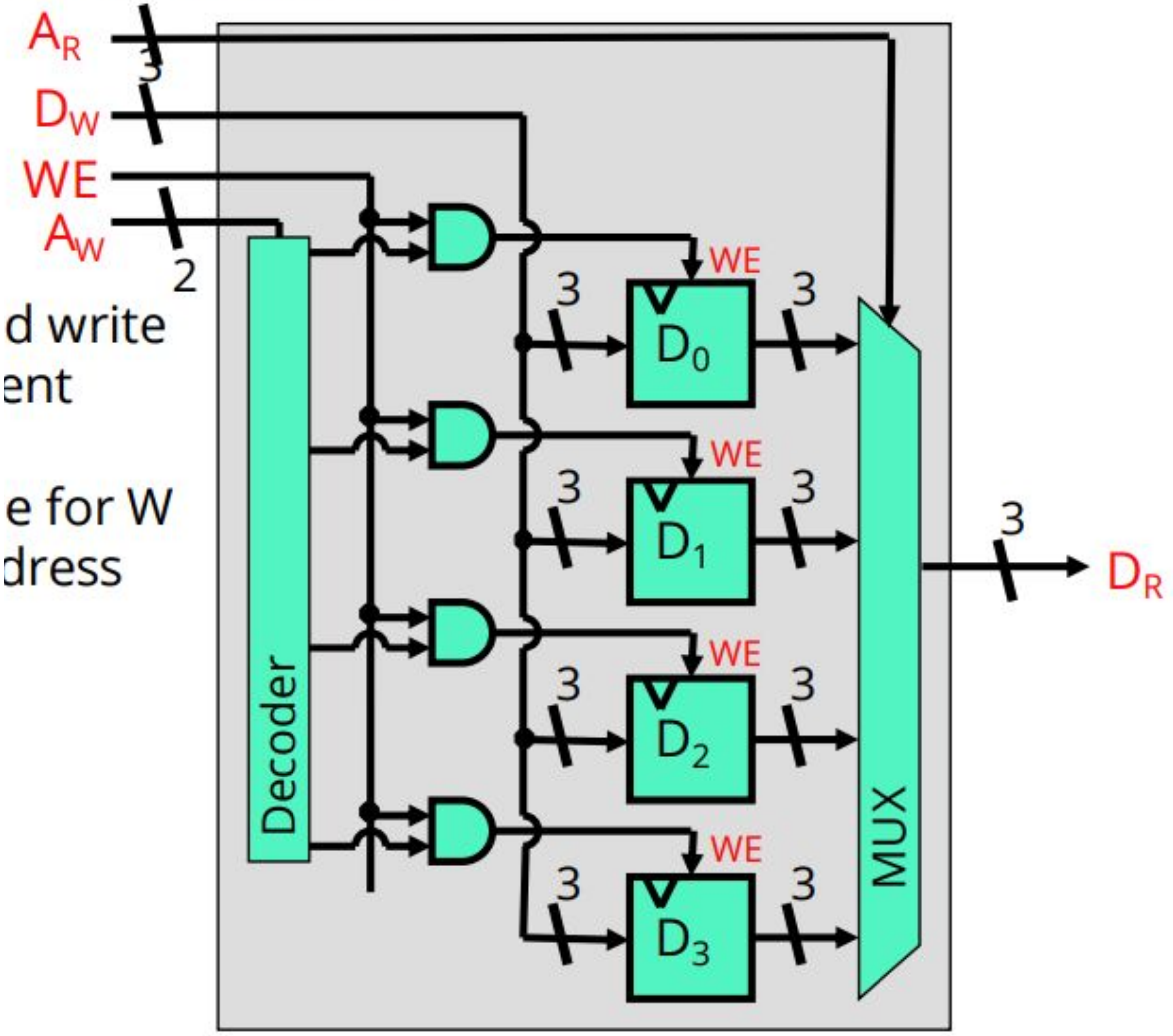
2^2 by 3-bit memory – independent R/W

- Can have independent read/write operations if we take in separate addresses for each.



You can read from one address and write to another with this arrangement

1 address line for R & 1 address line for W
Previously we used the same address

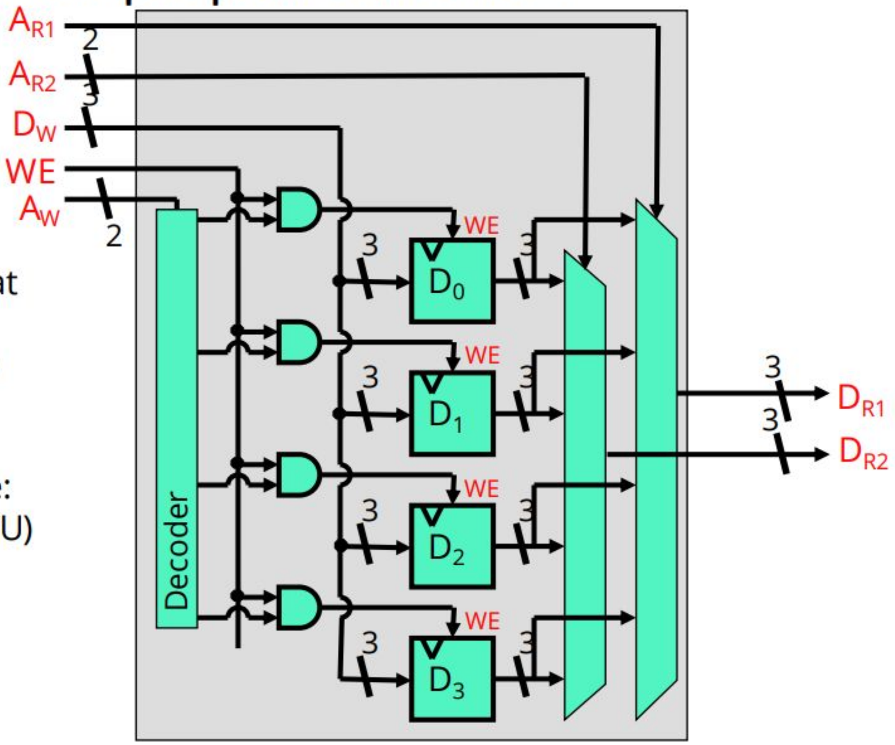


d write
ent

e for W
dress

2^2 by 3-bit memory – Multiple Reads

- Can read from multiple places at once!



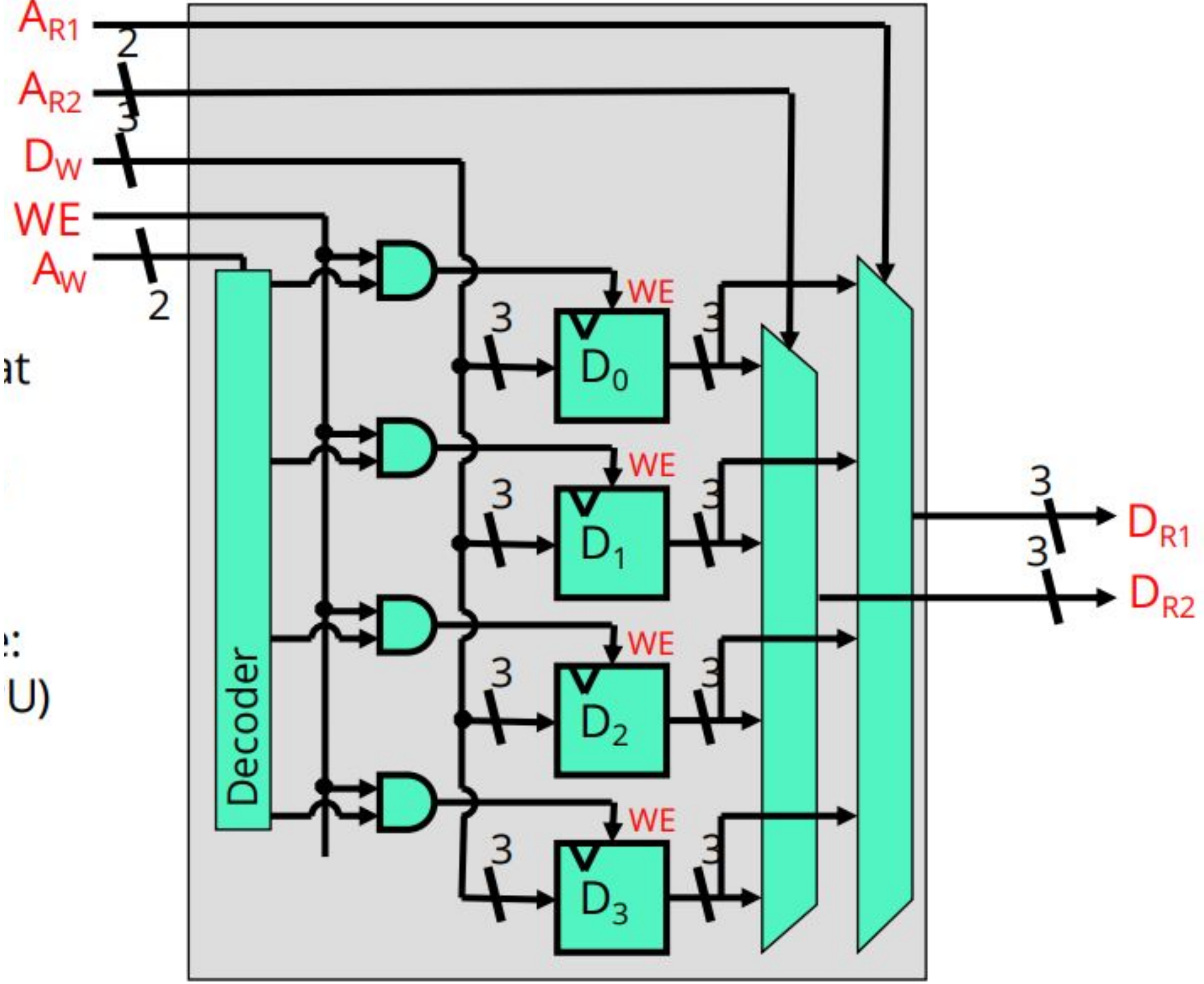
Read from 2 locations at once, write to a third!
(notice 3 address lines)

(We will use this later
In something called the:
“register file” for the CPU)

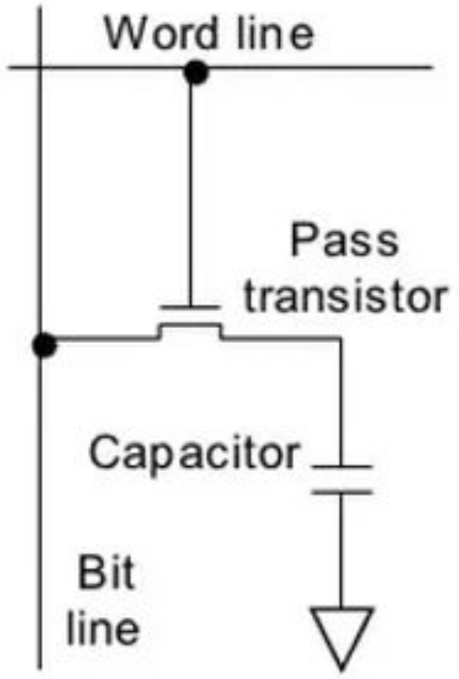
at

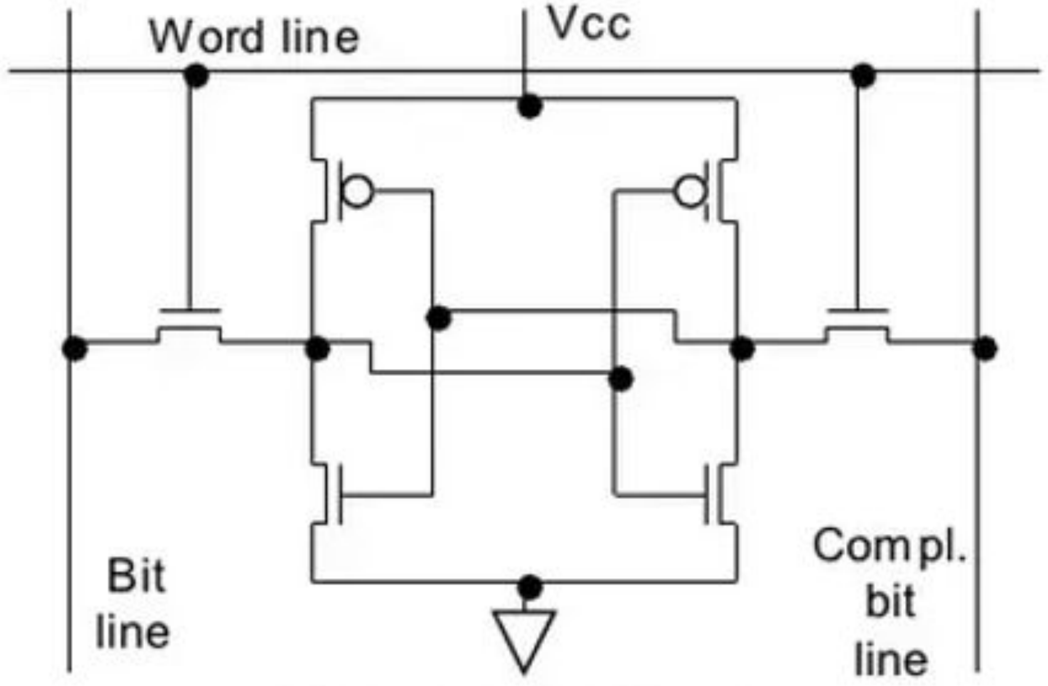
:

U)



SRAM vs DRAM at transistor level (will not be tested)





SRAM

- Fast
- 6 transistors per bit

DRAM

- Slow
- 1 transistors per bit

BID IGEA: Not all memory is same memory