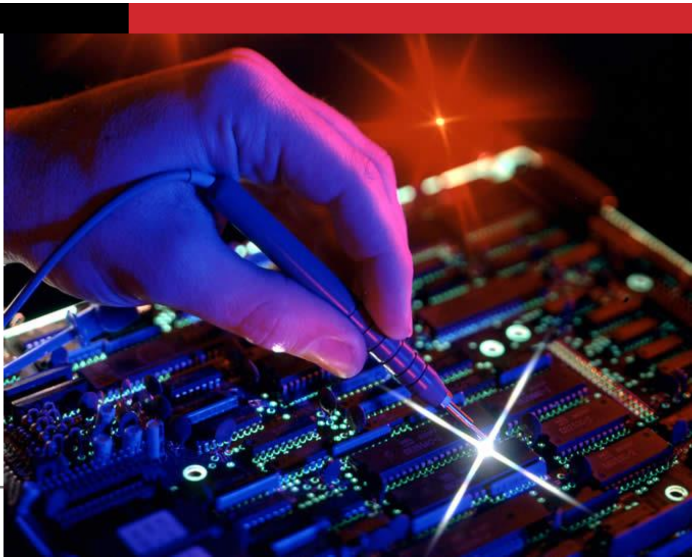
Computer Architecture & Microprocessor System

MEMORY ELEMENTS

Dennis A. N. Gookyi

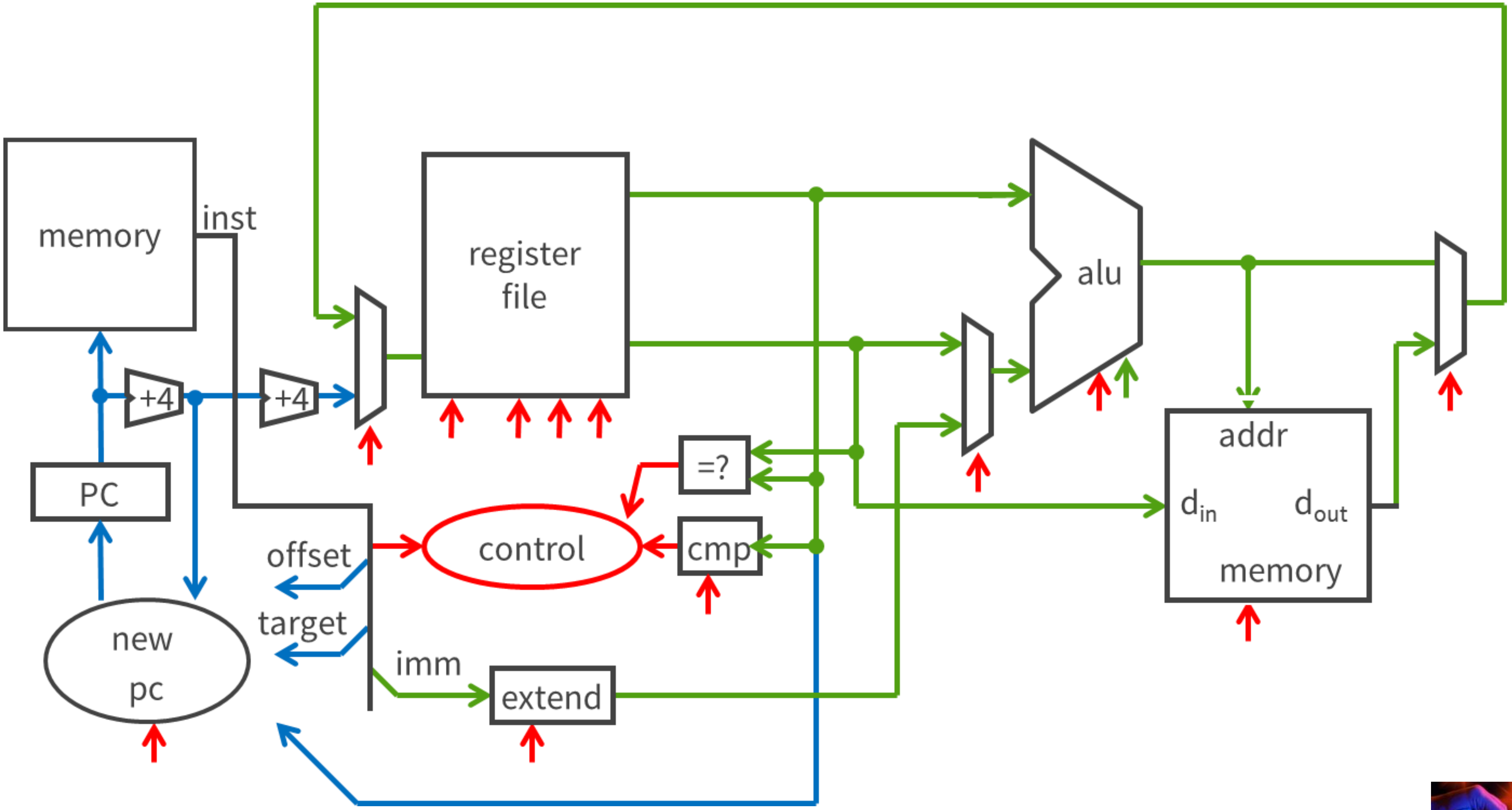


CONTENTS

- Memory Elements

BIG PICTURE: BUILDING A PROCESSOR

- Single cycle processor

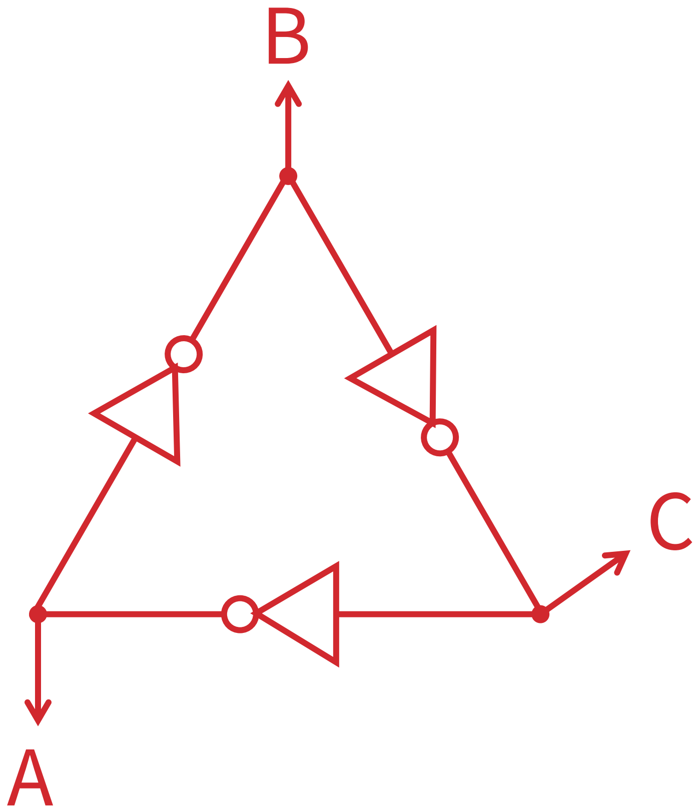


STORING 1-BIT

- How do we store one bit?

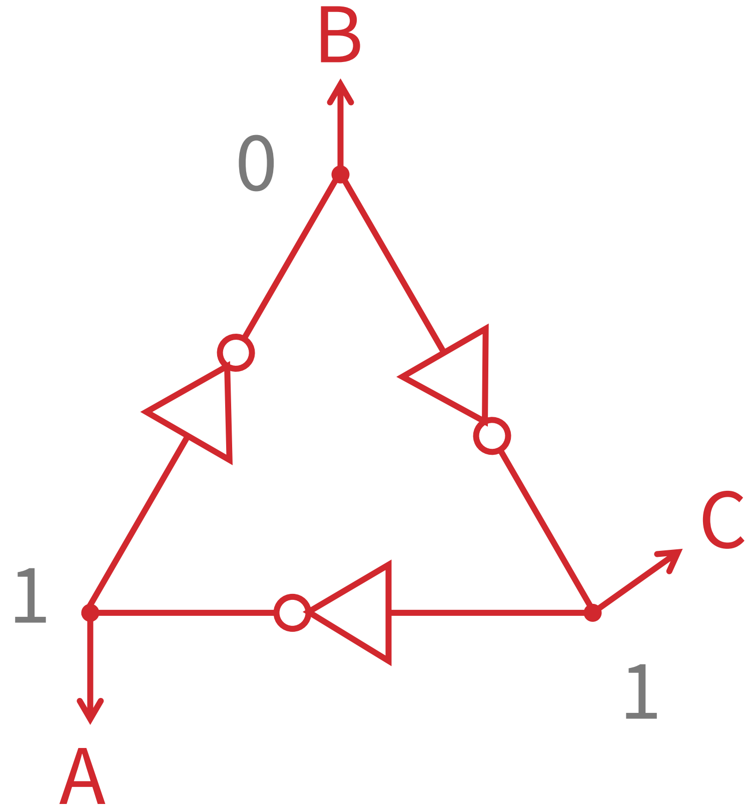
STORING 1-BIT

- First Attempt: Unstable Devices



STORING 1-BIT

- First Attempt: Unstable Devices

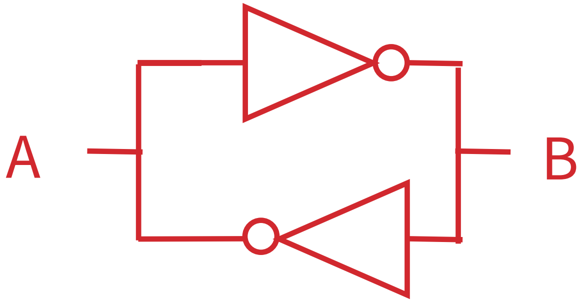


Does not work!

- □ Unstable
- □ Oscillates wildly!

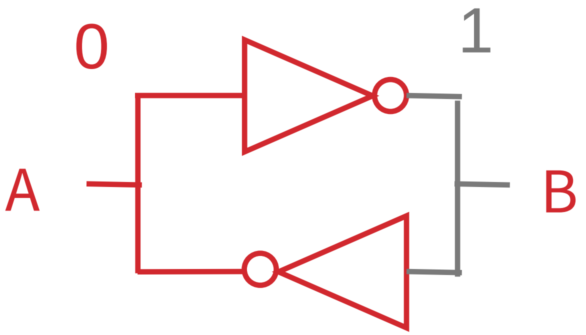
STORING 1-BIT

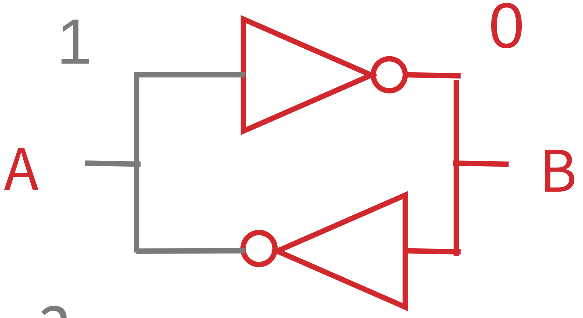
- Second Attempt: Bistable Devices
  - Stable and unstable equilibria



A Simple Device

In stable state, $\overline{A} = B$

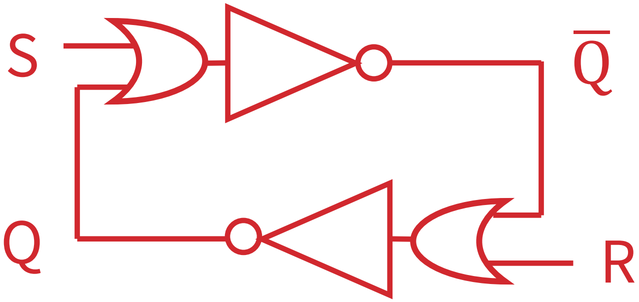




How do we change the state?

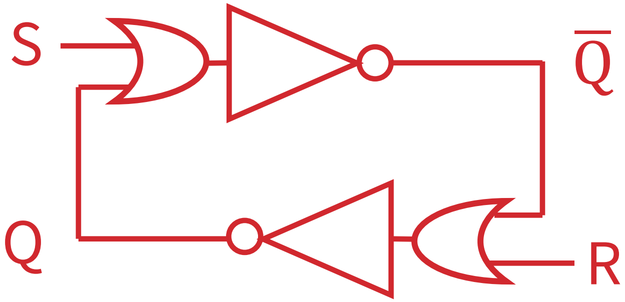
STORING 1-BIT

- Third Attempt: Set-Reset Latch



STORING 1-BIT

- Third Attempt: Set-Reset Latch



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

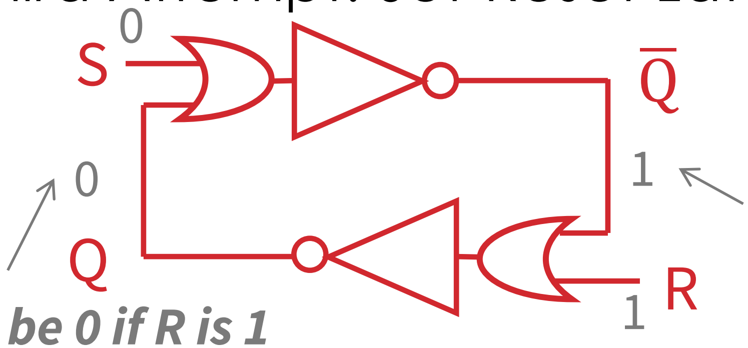
A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Set-Reset (S-R) Latch

Stores a value Q and its complement

STORING 1-BIT

- Third Attempt: Set-Reset Latch



$\overline{Q}$ will be 1

Q will be 0 if R is 1

A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

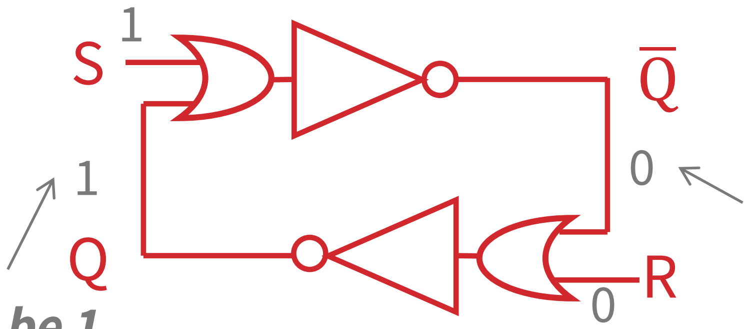
S	R	Q	$\overline{Q}$
0	0		
0	1	0	1
1	0		
1	1		

Set-Reset (S-R) Latch

Stores a value Q and its complement

STORING 1-BIT

- Third Attempt: Set-Reset Latch



Q will be 1

$\overline{Q}$ will be 0 if S is 1

A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

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

Set-Reset (S-R) Latch

Stores a value Q and its complement

What are the values for Q and $\overline{Q}$?

a) 0 and 0

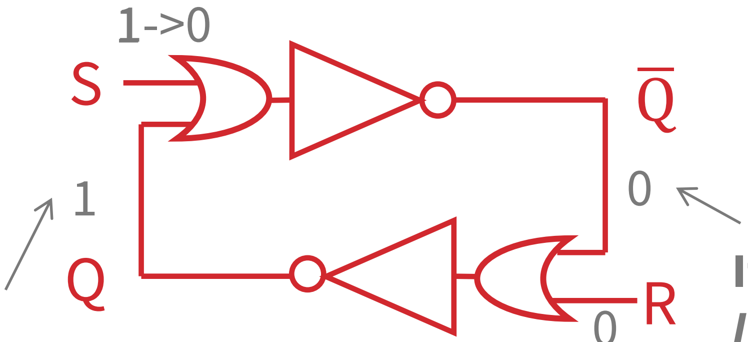
b) 0 and 1

c) 1 and 0

d) 1 and 1

STORING 1-BIT

- Third Attempt: Set-Reset Latch



If $\overline{Q}$ is 0 will stay 0
If $\overline{Q}$ is 1 will stay 1

If Q is 1, will stay 1
if Q is 0, will stay 0

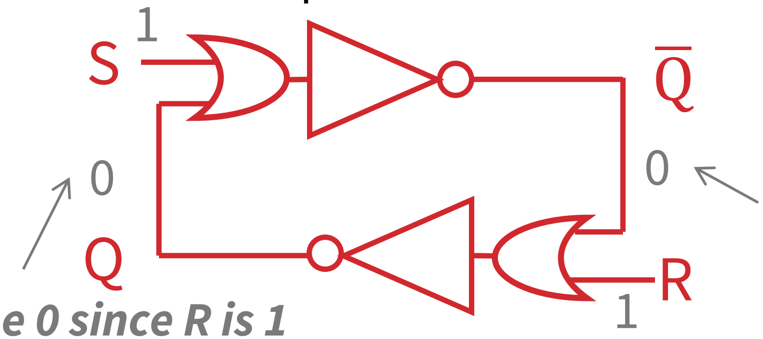
S	R	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	0	1
1	0	1	0
1	1		

A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Set-Reset (S-R) Latch
Stores a value Q and its complement

STORING 1-BIT

- Third Attempt: Set-Reset Latch



$\overline{Q}$ *will be 0 since S is 1*

Q will be 0 since R is 1

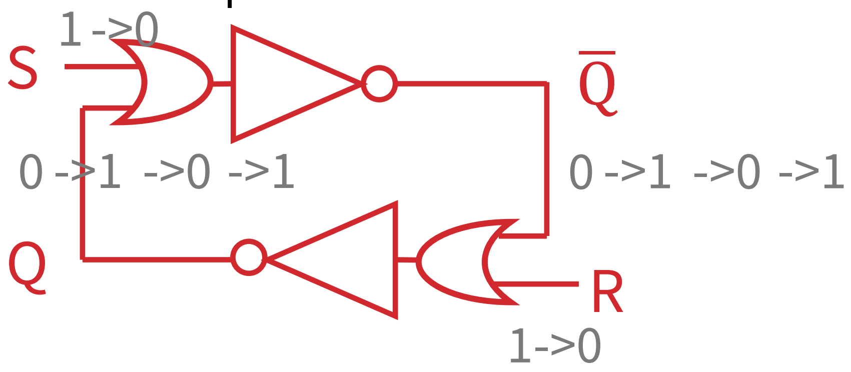
A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

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

What happens when S,R changes from 1,1 to 0,0?

STORING 1-BIT

- Third Attempt: Set-Reset Latch



A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

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

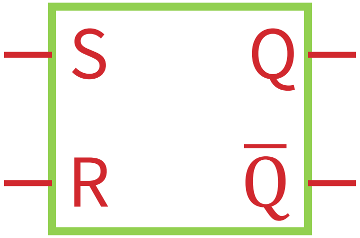
Set-Reset (S-R) Latch
Stores a value Q and its complement

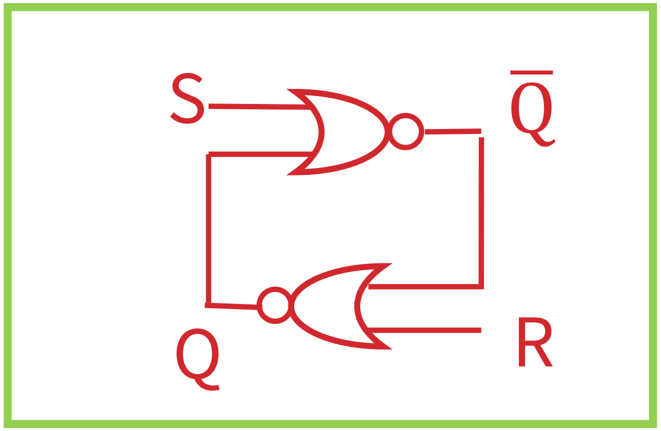
What happens when S,R changes from 1,1 to 0,0?

Q and $\overline{Q}$ become unstable and will oscillate wildly between values 0,0 to 1,1 to 0,0 to 1,1 …

STORING 1-BIT

- Third Attempt: Set-Reset Latch





S	R	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	hold
0	1	0	1	reset
1	0	1	0	set
1	1	forbidden		

Set-Reset (S-R) Latch

Stores a value Q and its complement

STORING 1-BIT

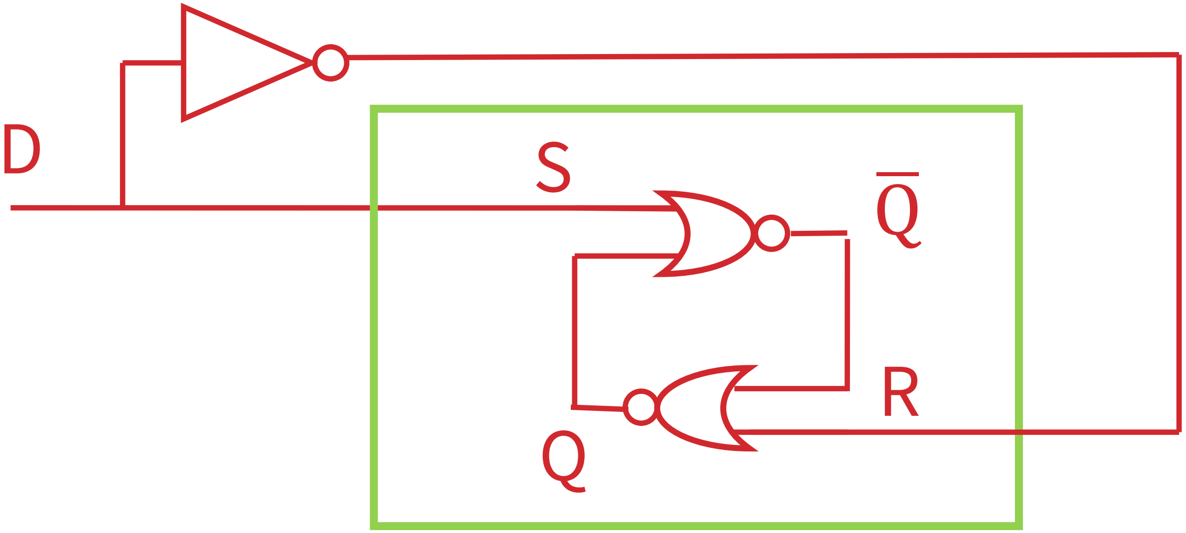
- Third Attempt: Set-Reset Latch

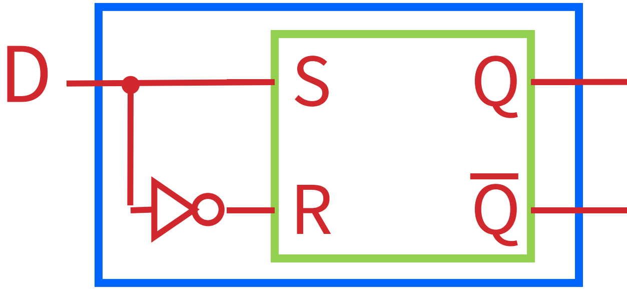
- Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit

  - Issue: SR Latch has a forbidden state

STORING 1-BIT

- Fourth Attempt: (Unclocked) D Latch





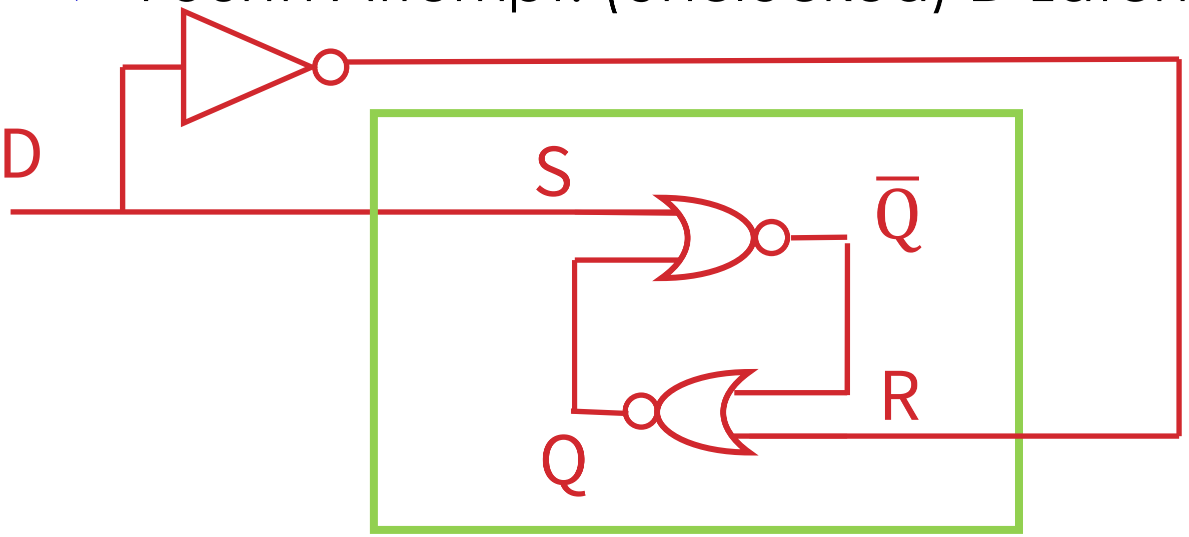
Fill in the truth table?

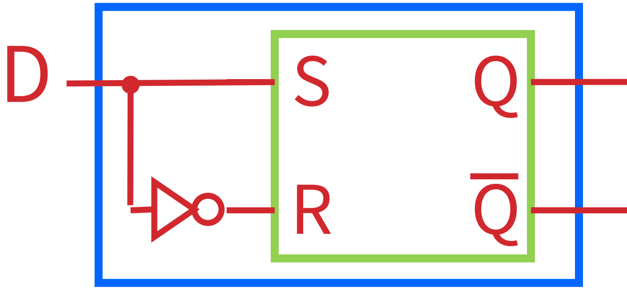
D	Q	$\overline{Q}$
0		
1		

A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

STORING 1-BIT

- Fourth Attempt: (Unclocked) D Latch





D	Q	$\overline{Q}$
0	0	1
1	1	0

Fill in the truth table?

Data (D) Latch

- Easier to use than an SR latch
- No possibility of entering an undefined state

When D changes, Q changes

- … immediately (…after a delay of 2 Ors and 2 NOTs)

Need to control when the output changes

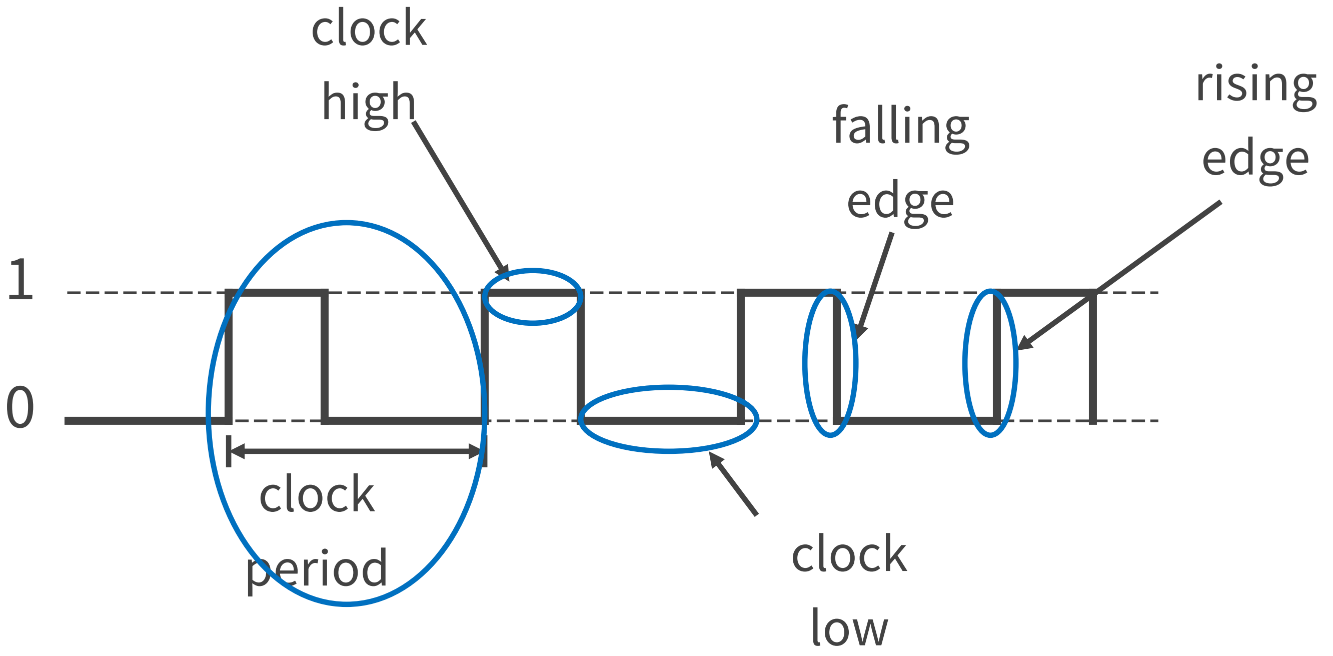
A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

STORING 1-BIT

- Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit
  - □ Issue: SR Latch has a forbidden state
  - □ (Unclocked) D Latch can store and change a bit like an SR Latch while avoiding the forbidden state
    - How do we coordinate state changes to a D Latch?

CLOCKS

- Clock helps coordinate state changes
  - □ Usually generated by an oscillating crystal
  - □ Fixed period
  - □ Frequency = 1/period

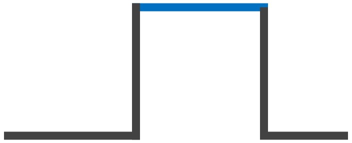


CLOCKS

❖ Clock disciplines

Level sensitive

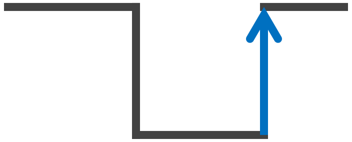
- State changes when clock is high (or low)



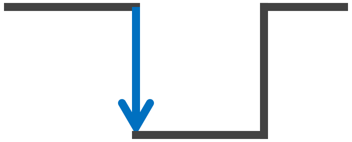
Edge triggered

- State changes at clock edge

positive edge-triggered

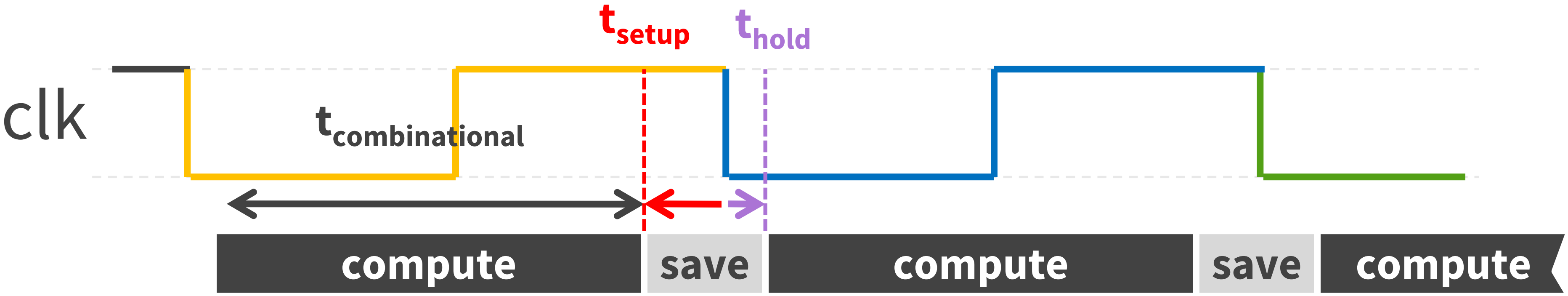


negative edge-triggered



CLOCKS

- Clock Methodology
  - □ Negative edge, synchronous



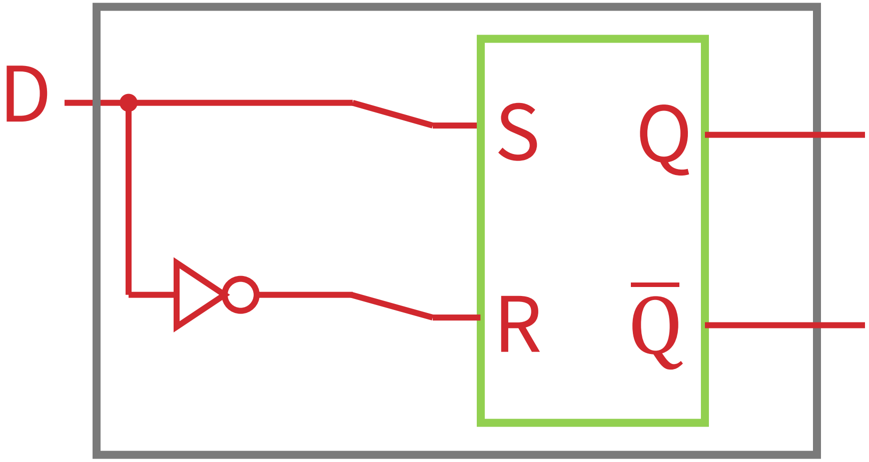
Edge-Triggered → signals must be stable near falling edge

“near” = before and after

t_{setup} t_{hold}

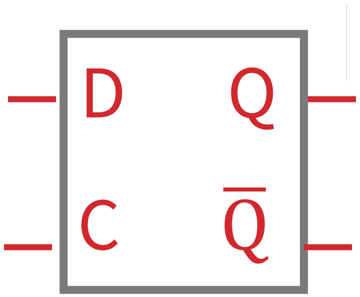
D LATCH

- D Latch



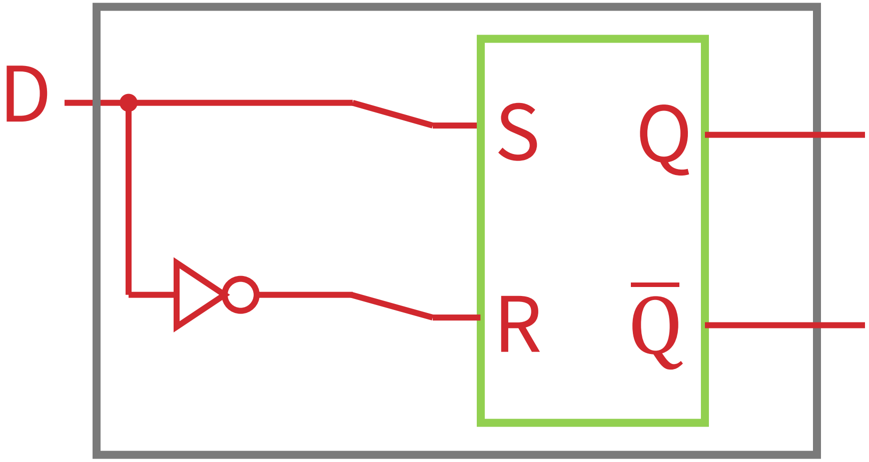
- Inverter prevents SR Latch from entering 1,1 state

	D	Q	$\overline{Q}$	
	0			*Reset*
	1			*Set*

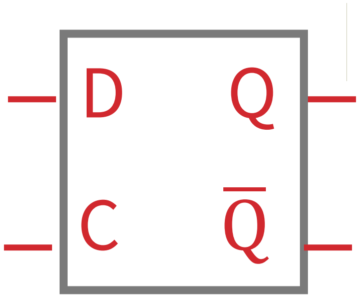


D LATCH

- D Latch

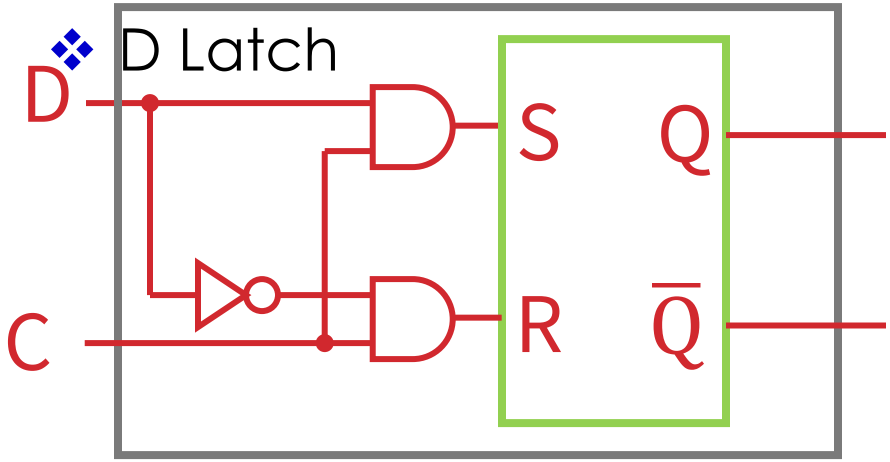


- Inverter prevents SR Latch from entering 1,1 state



	D	Q	$\overline{Q}$	
	0	0	1	*Reset*
	1	1	0	*Set*

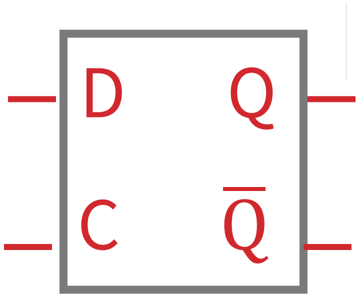
D LATCH



- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state
- C enables changes

C = 1, D Latch *transparent*: set/reset (according to D)

C = 0, D Latch *opaque*: keep state (ignore D)



C	D	Q	$\overline{Q}$
0	0		
0	1		
1	0		
1	1		

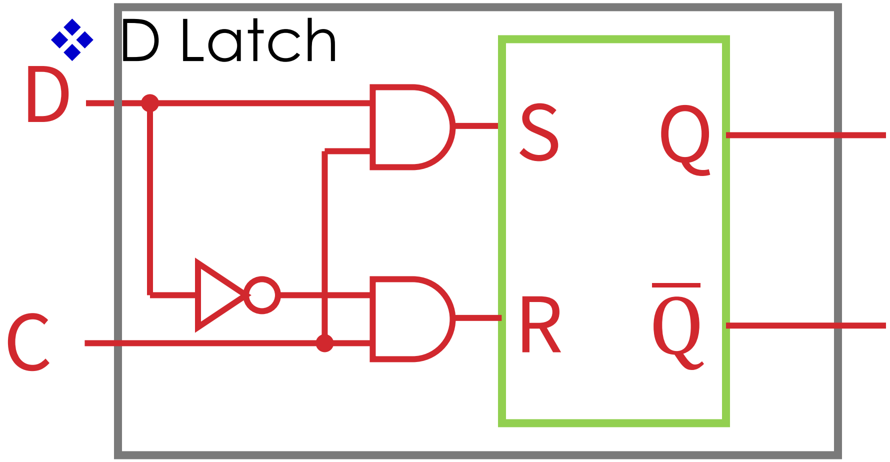
No Change

Reset

Set

D LATCH

- ❖ D Latch

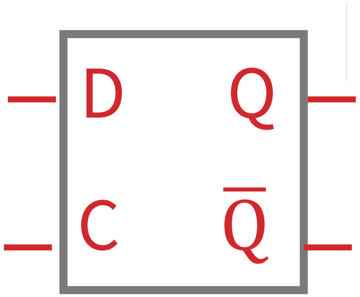


- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state
- C enables changes

C = 1, D Latch *transparent*:
set/reset (according to D)

C = 0, D Latch *opaque*:
keep state (ignore D)

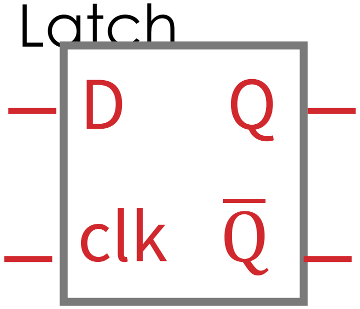
S	R	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	hold
0	1	0	1	reset
1	0	1	0	set
1	1	forbidden		



C	D	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	
0	1	Q	$\overline{Q}$	
1	0	0	1	*Reset*
1	1	1	0	*Set*

D LATCH

- D Latch



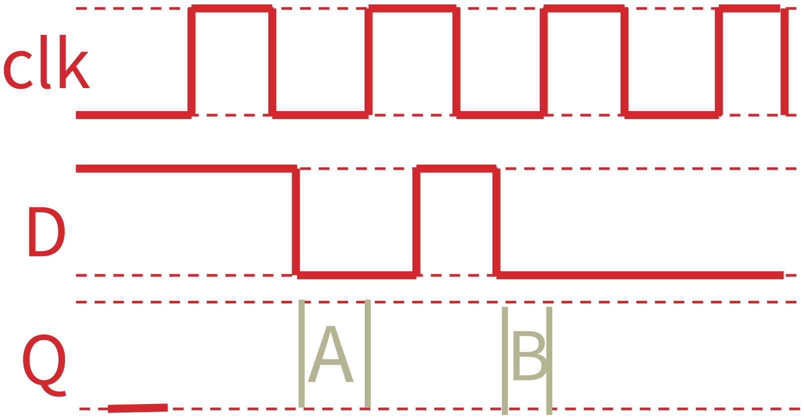
What is the value of Q at A & B?

a) A = 0, B = 0

b) A = 0, B = 1

c) A = 1, B = 0

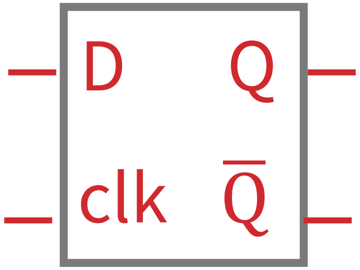
d) A = 1, B = 1



clk	D	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

D LATCH

- D Latch



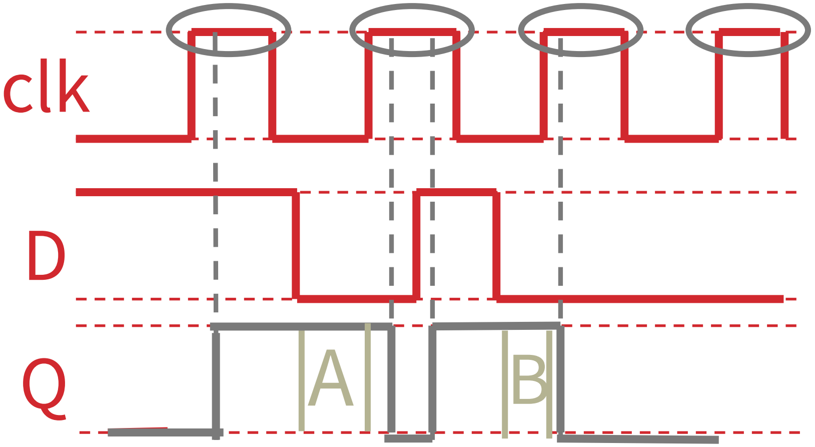
What is the value of Q at A & B?

a) A = 0, B = 0

b) A = 0, B = 1

c) A = 1, B = 0

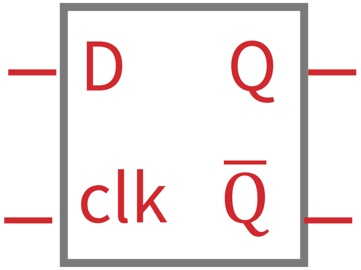
[✓ d) A = 1, B = 1]



clk	D	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

D LATCH

- D Latch



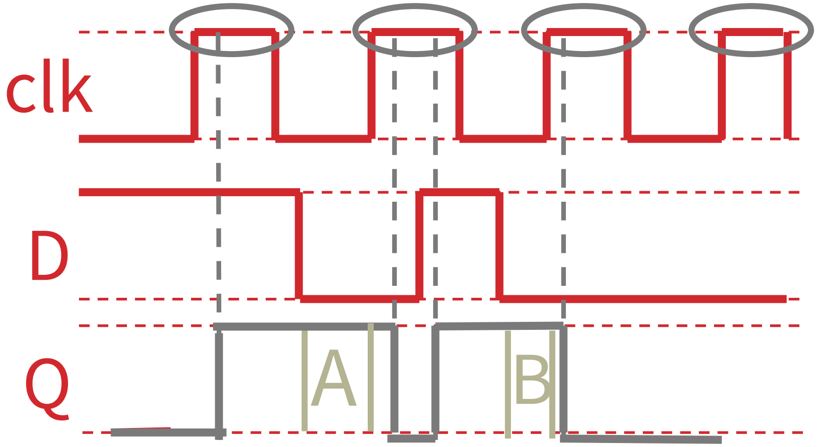
Level Sensitive D Latch

Clock high:

set/reset (according to D)

Clock low:

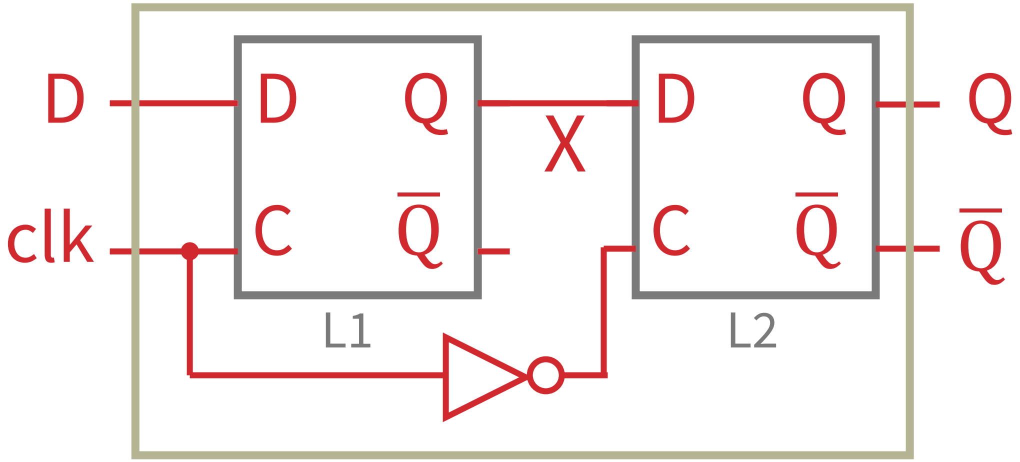
keep state (ignore D)



clk	D	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

D FLIP-FLOP

- D flip-flop



- **Edge-Triggered**
- **Data captured when the clock is high**
- **Output changes only on falling edges**

- Master-Slave Flip Flop
  - □ Outputs change only on falling edges
  - □ Data is captured on rising edges
- 1 cycle delay but works out perfectly – data for the next stage is ready 1 cycle ahead of time

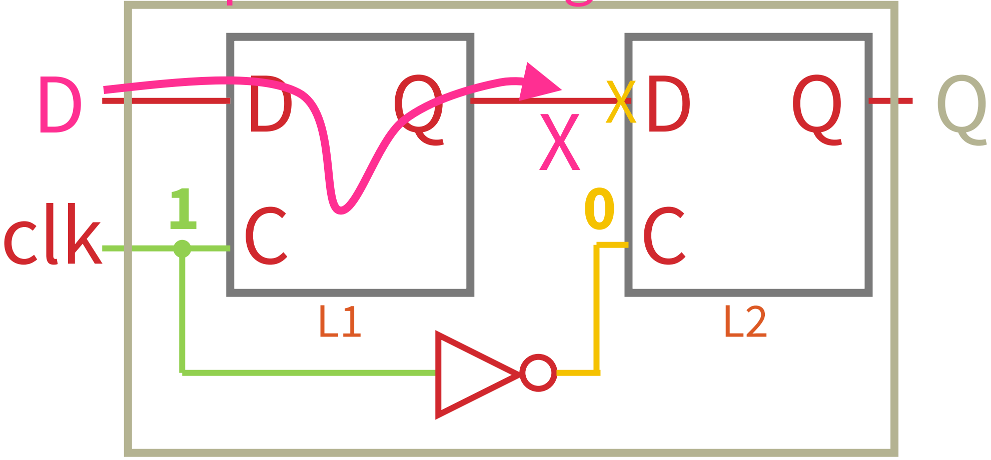
D FLIP-FLOP

- D flip-flop

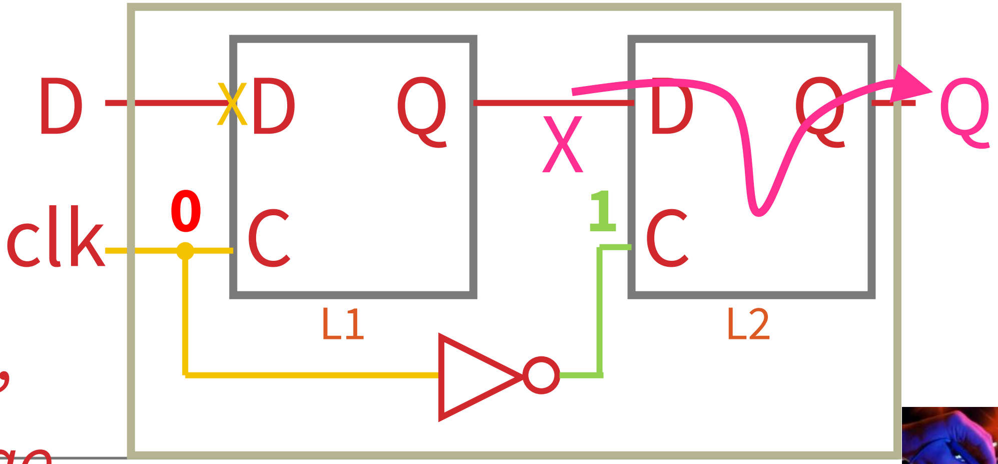
Clock = 1: L1 *transparent*
L2 *opaque*

When CLK rises (0→1),
now X can change,
Q does not change

D passes through L1 to X



X passes through L2 to Q

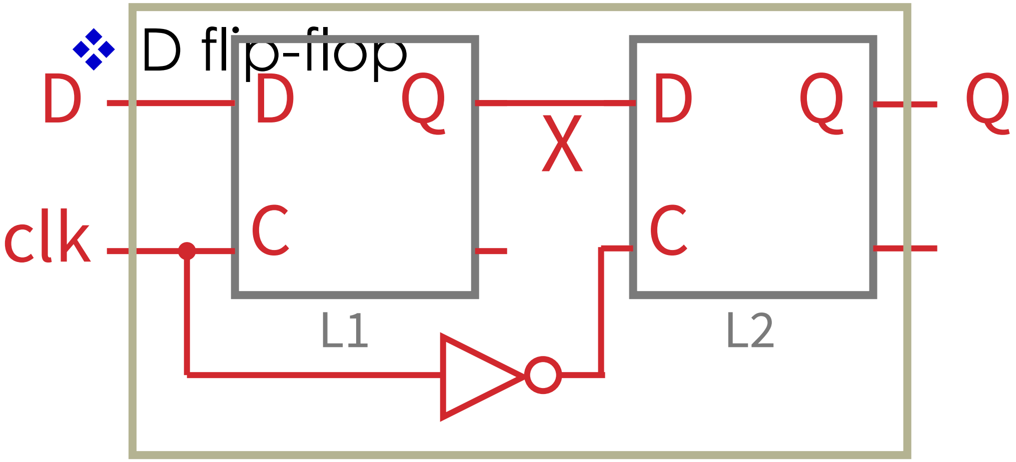


Clock = 0: L1 *opaque*
L2 *transparent*

When ***CLK* falls** (1→0),
Q gets X, X cannot change

D FLIP-FLOP

- D flip-flop



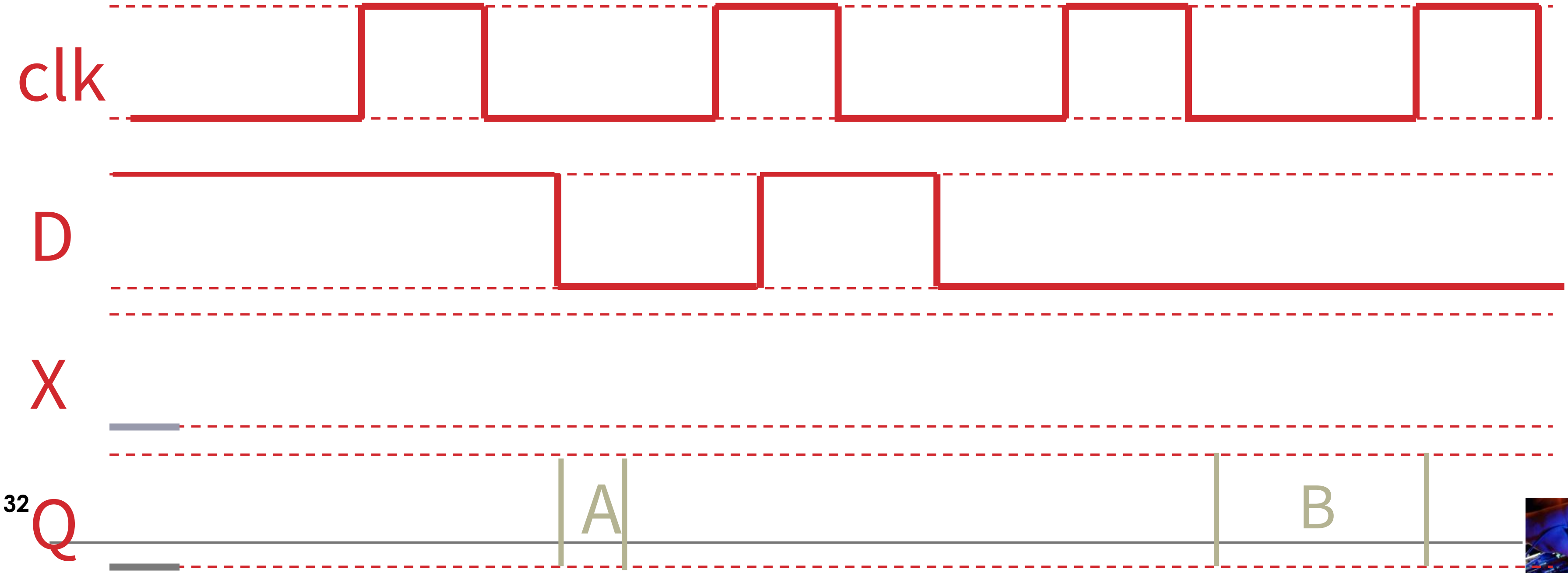
What is the value of Q at A & B?

a) A = 0, B = 0

b) A = 0, B = 1

c) A = 1, B = 0

d) A = 1, B = 1



D FLIP-FLOP

What is the value of Q at A & B?

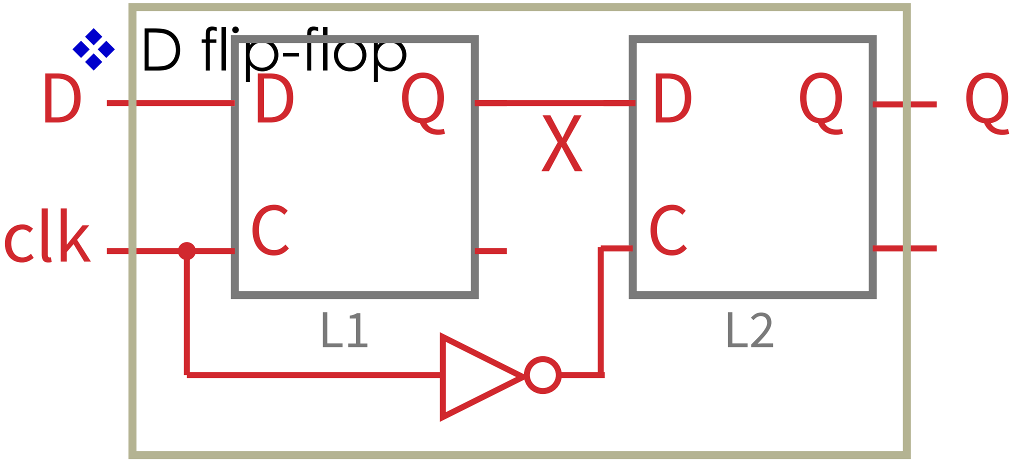
a) A = 0, B = 0

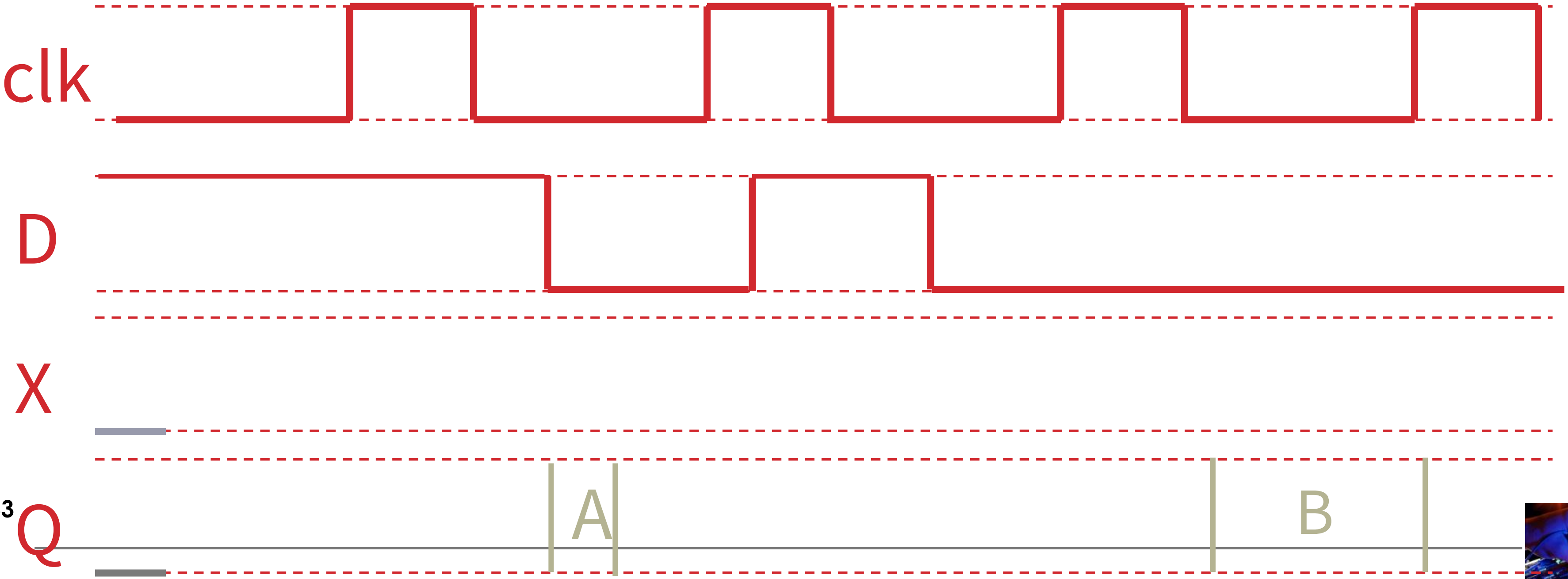
b) A = 0, B = 1

[✓ c) A = 1, B = 0]

d) A = 1, B = 1

- D flip-flop





D FLIP-FLOP

- Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit
  - □ SR Latch has a forbidden state

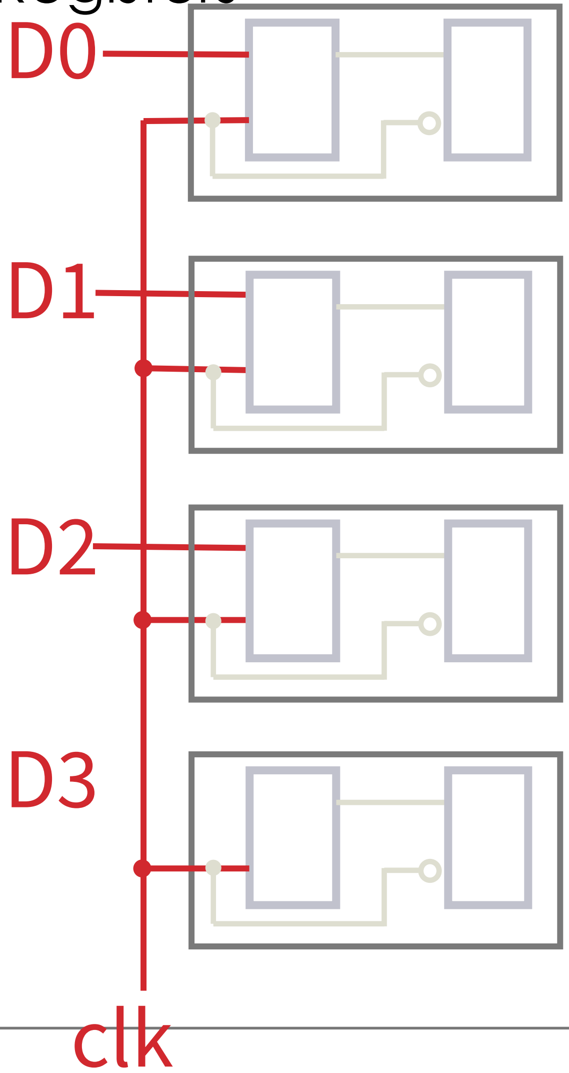
- (Unclocked) D Latch can store and change a bit like an SR Latch while avoiding a forbidden state

- An Edge-Triggered D Flip-Flip (aka Master-Slave D Flip-Flip) stores one bit
  - □ The bit can be changed in a synchronized fashion on the edge of a clock signal

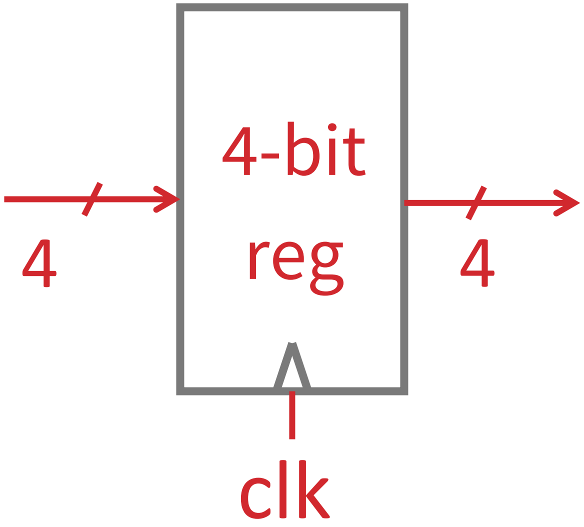
- How do we store more than one bit, N bits?

STORING MORE BITS

- Registers

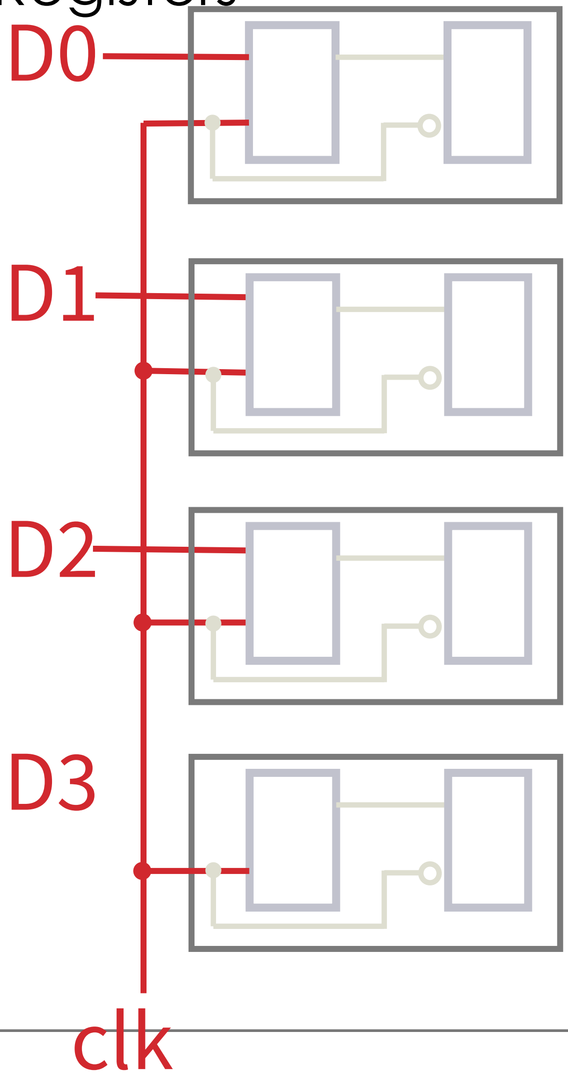


- D flip-flops in parallel
- shared clock
- extra clocked inputs: write_enable, reset, …

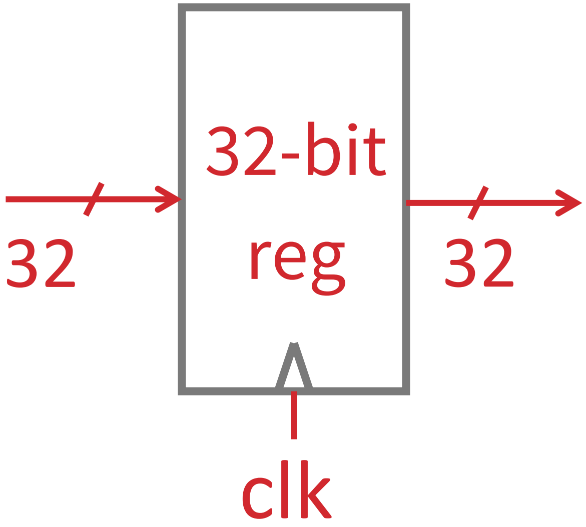


STORING MORE BITS

- Registers

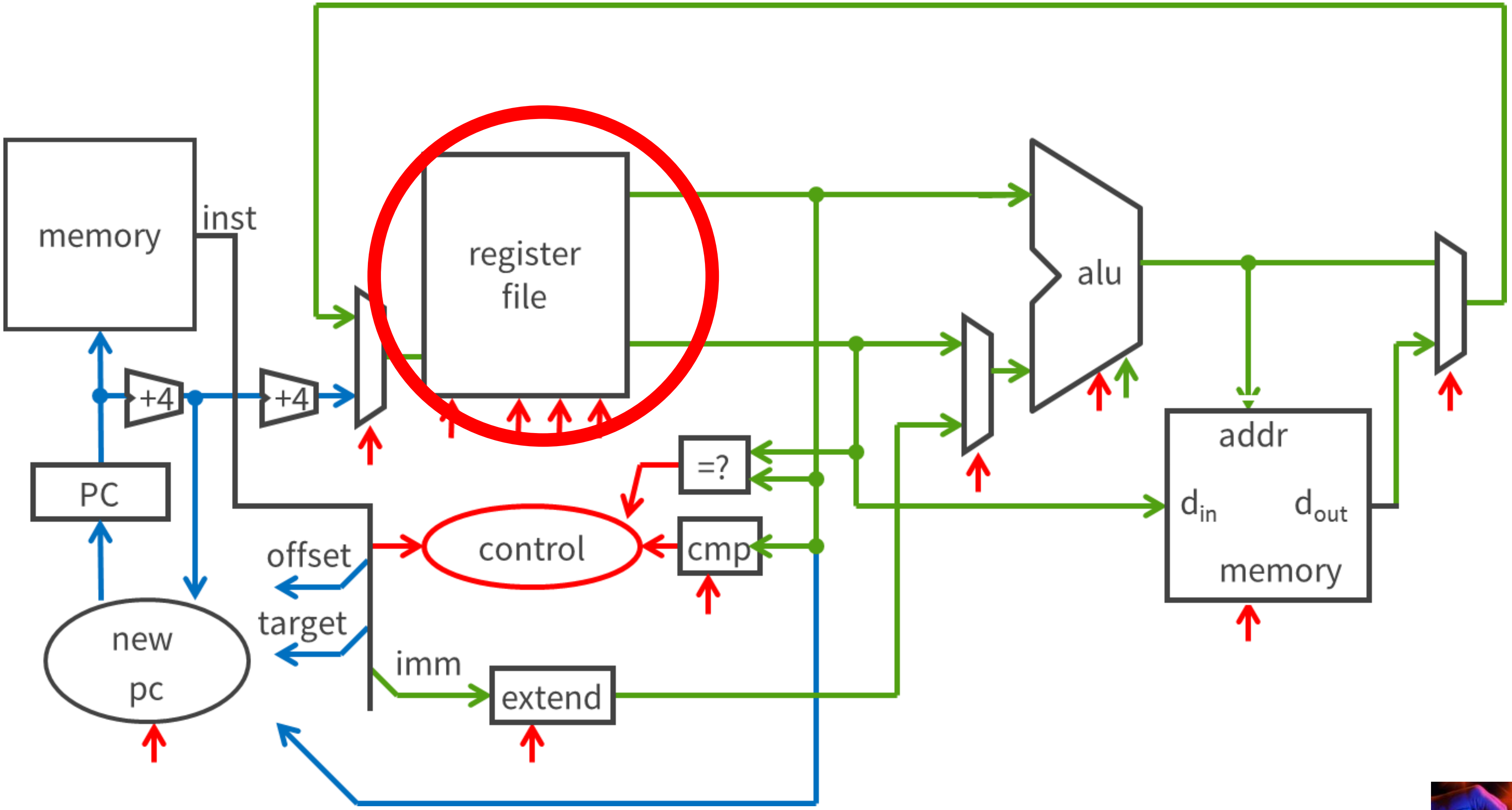


- D flip-flops in parallel
- shared clock
- extra clocked inputs: write_enable, reset, …



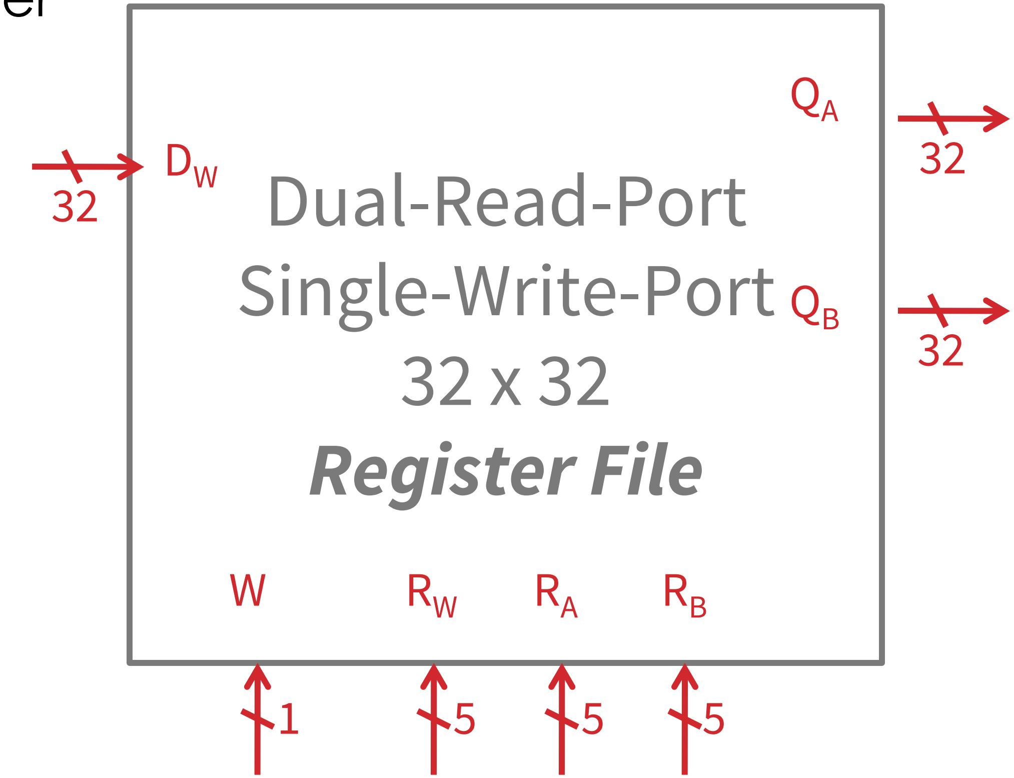
BIG PICTURE: BUILDING A PROCESSOR

- Single cycle processor



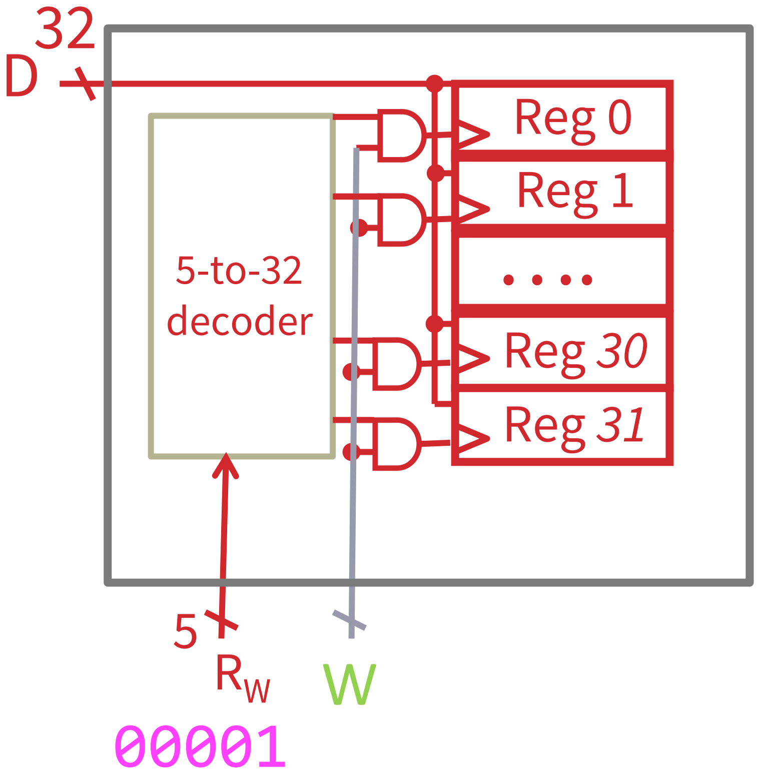
REGISTER FILE

- Register file
  - □ N read/write registers
  - □ Indexed by register number



REGISTER FILE

- Register file
  - □ N read/write registers
  - □ Indexed by register number



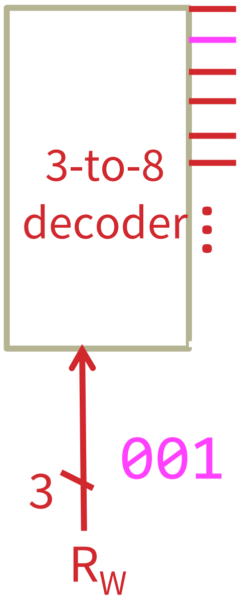
```
addix1, x0, 10
```

- How to write one register in the register file
  - □ Need a decoder

REGISTER FILE

- 3-to-8 decoder truth table and circuit

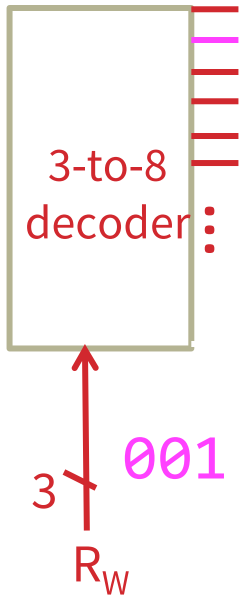
i2	i1	i0	o0	o1	o2	o3	o4	o5	o6	o7
0	0	0								
0	0	1								
0	1	0								
0	1	1								
1	0	0								
1	0	1								
1	1	0								
1	1	1								

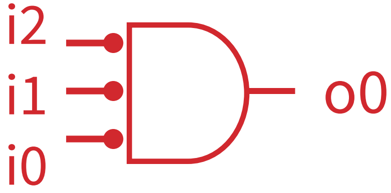


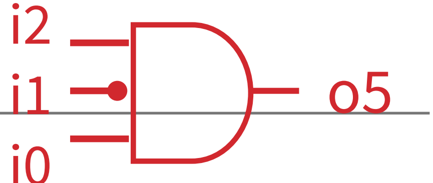
REGISTER FILE

- 3-to-8 decoder truth table and circuit

i2	i1	i0	o0	o1	o2	o3	o4	o5	o6	o7
0	0	0	1							
0	0	1		1						
0	1	0			1					
0	1	1				1				
1	0	0					1			
1	0	1						1		
1	1	0							1	
1	1	1								1







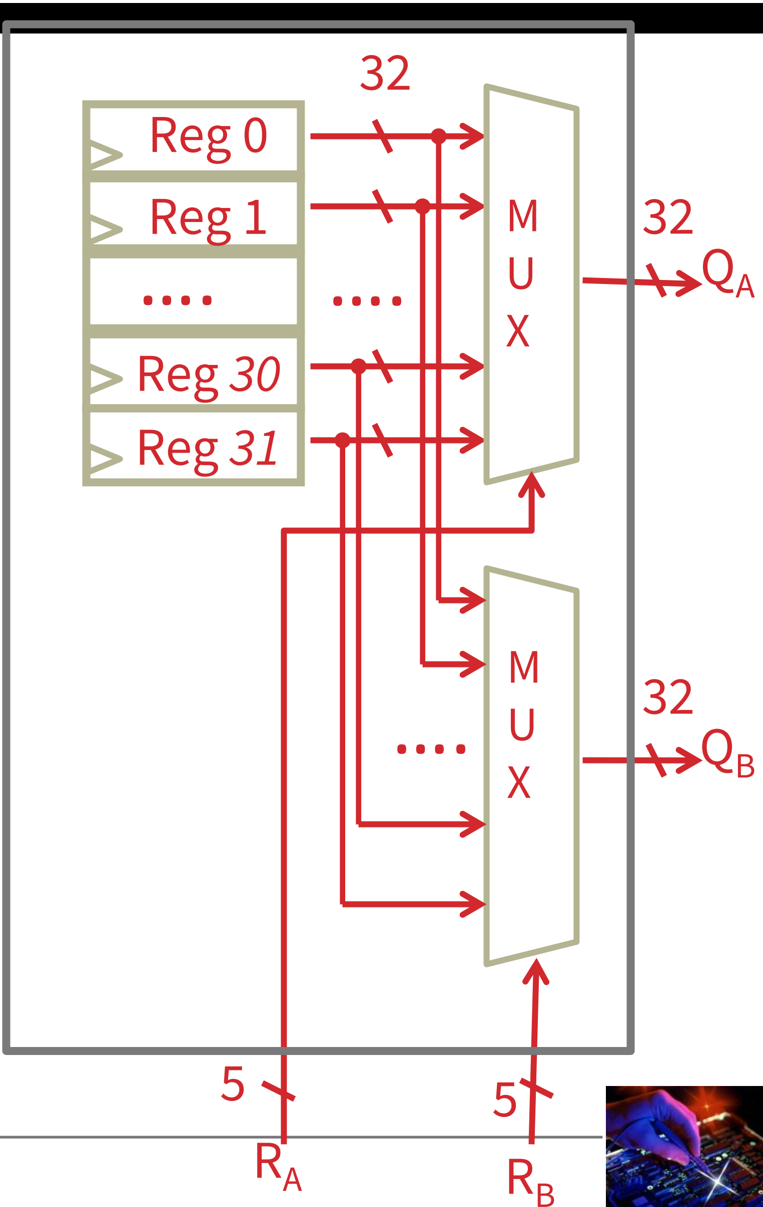
REGISTER FILE

- Register file
  - □ N read/write registers
  - □ Indexed by register number

```
add x1, x0, x5
```

How to read from two registers?

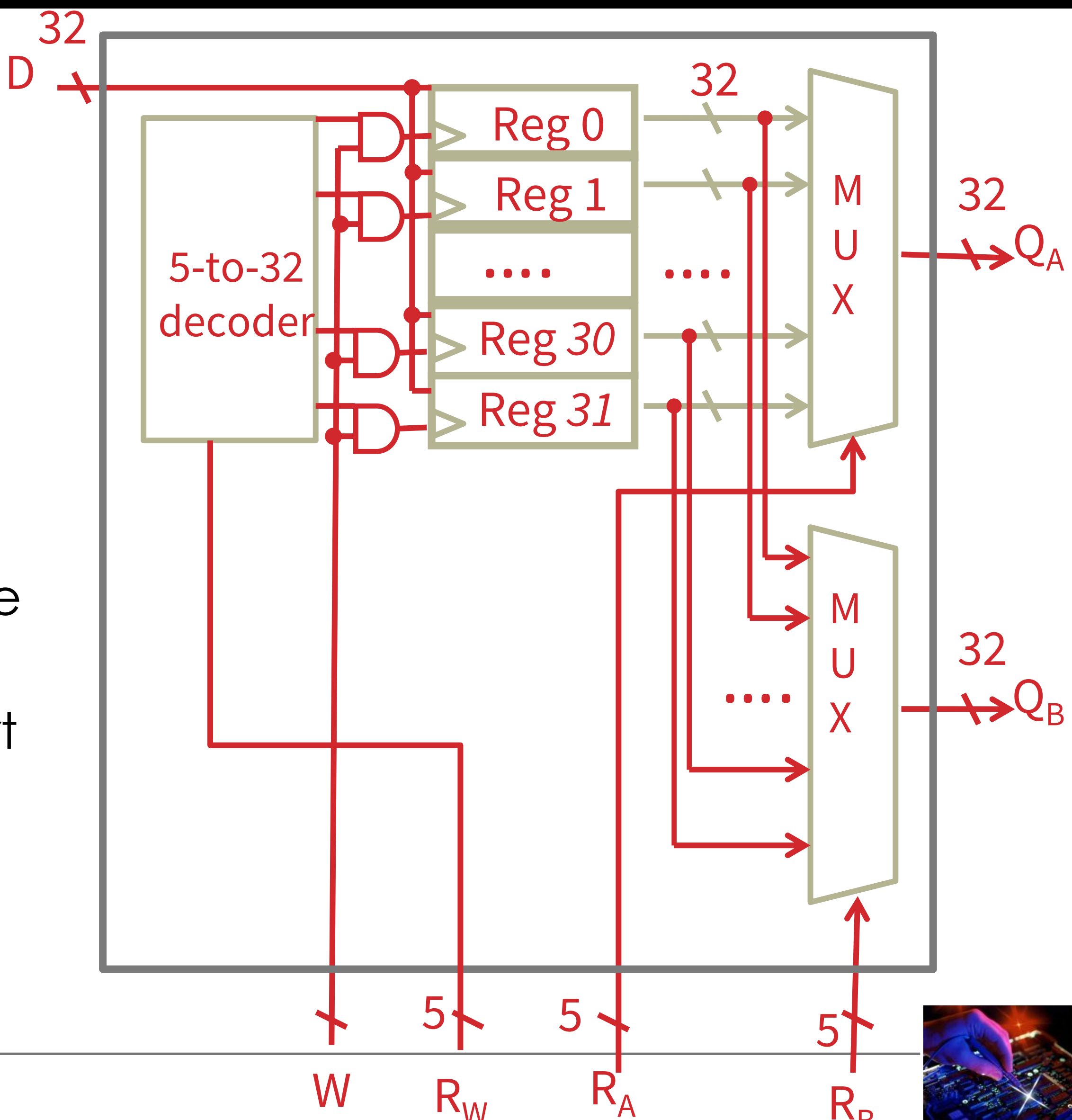
- □ Need a multiplexor



REGISTER FILE

- Register file
  - N read/write registers
  - Indexed by register number

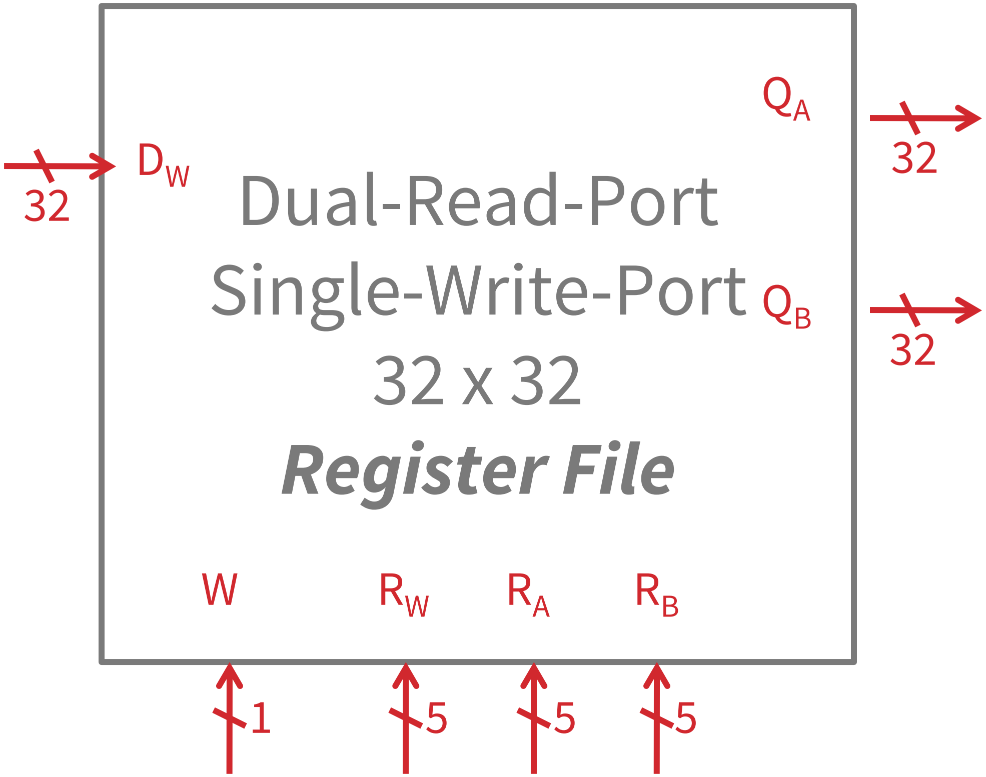
- Implementation:
  - D flip flops to store bits
  - Decoder for each write port
  - Mux for each read port



REGISTER FILE

- Register file
  - □ N read/write registers
  - □ Indexed by register number

- Implementation:
  - □ D flip flops to store bits
  - □ Decoder for each write port
  - □ Mux for each read port



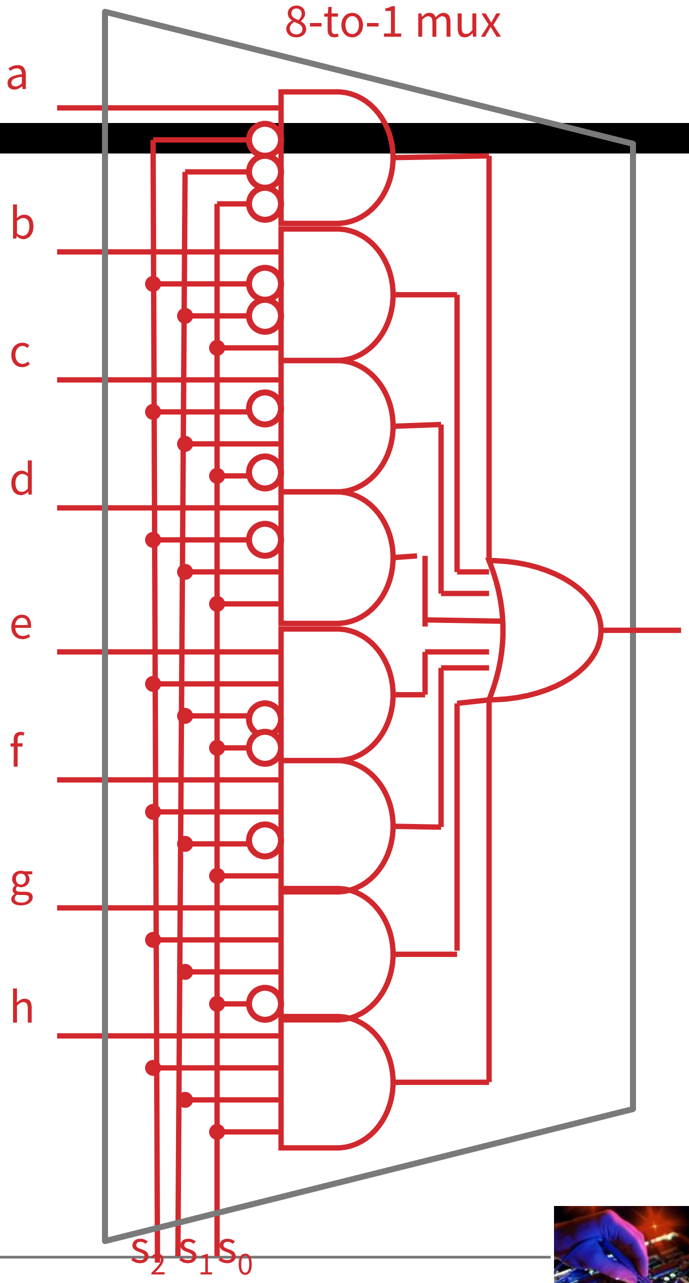
REGISTER FILE

- Register file
  - + Very fast (a few gate delays for both read and write)
  - + Adding extra ports is straightforward
  - – Doesn't scale

    e.g. 32Mb register file with 32 bit registers

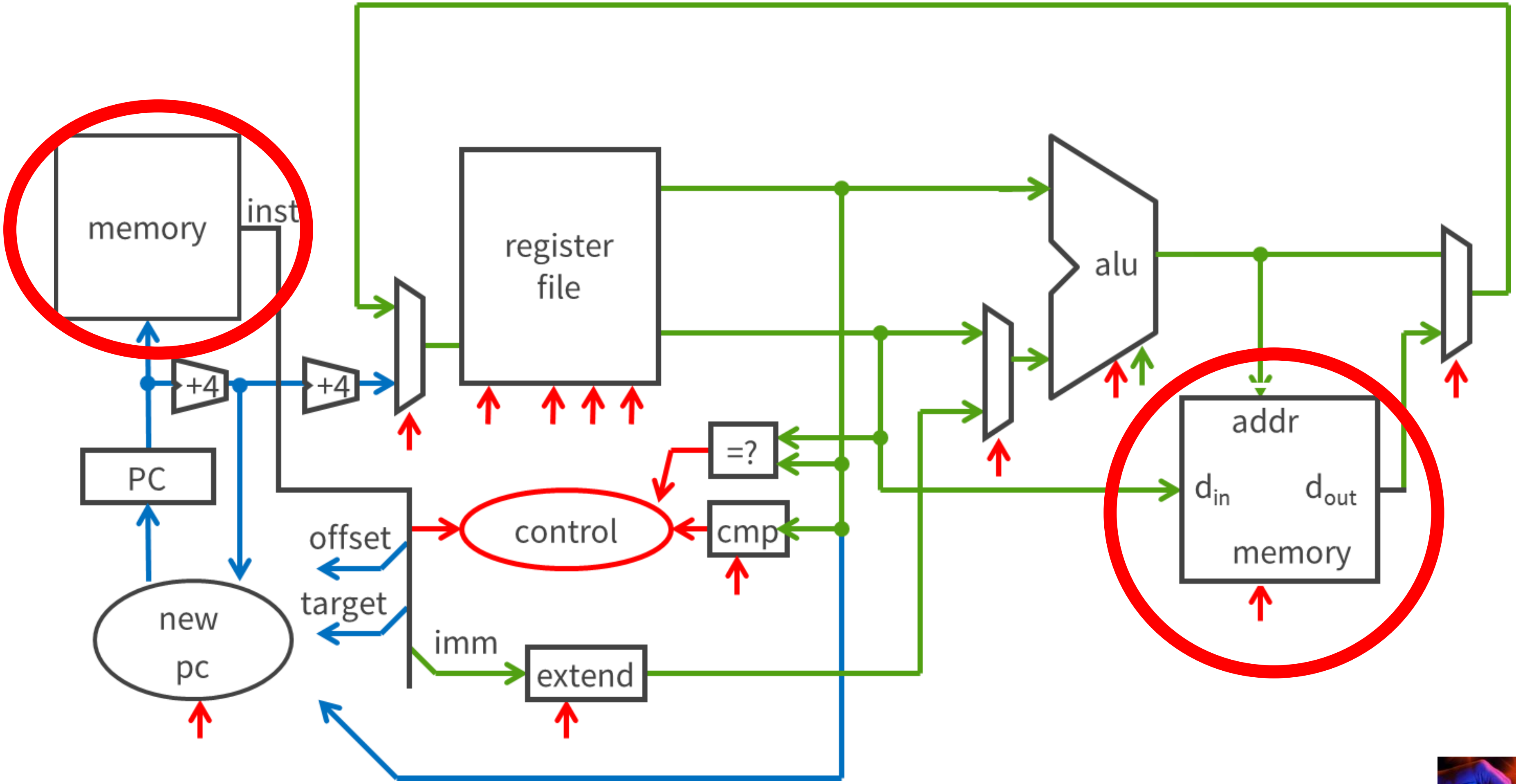
    Need 32x 1M-to-1 multiplexor and 32x 20-to-1M decoder

    How many logic gates/transistors?



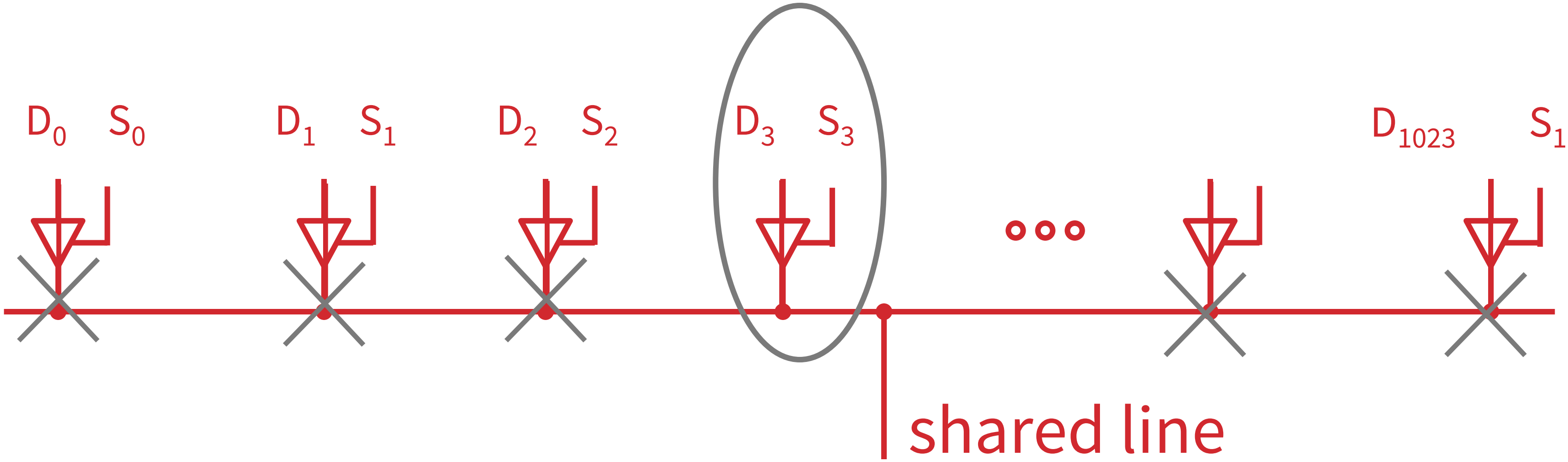
BIG PICTURE: BUILDING A PROCESSOR

- Single cycle processor



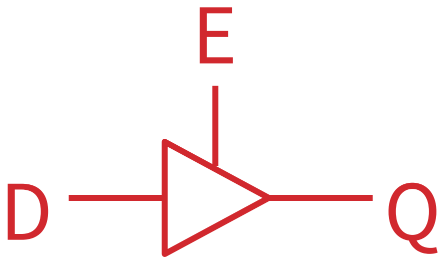
BUILDING LARGE MEMORIES

- Need a shared bus (or shared bit line)
  - □ Many FlipFlops/outputs/etc. connected to single wire
  - □ Only one output drives the bus at a time



BUILDING LARGE MEMORIES

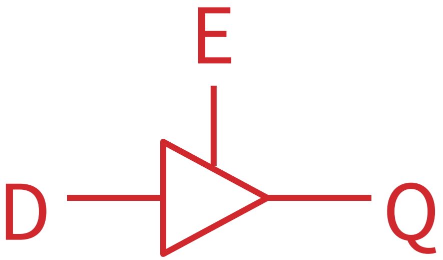
- Tri-State Buffers
  - □ If enabled (E=1), then Q = D
  - □ Otherwise, Q is not connected (z = high impedance)



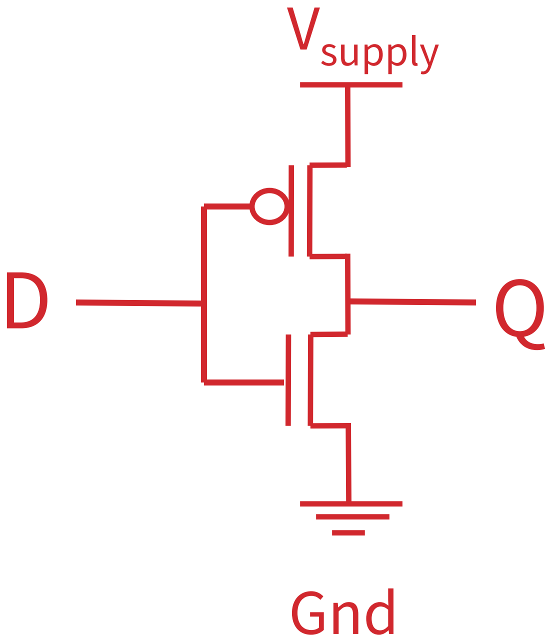
E	D	Q
0	0	z
0	1	z
1	0	0
1	1	1

BUILDING LARGE MEMORIES

- Tri-State Buffers
  - □ If enabled (E=1), then Q = D
  - □ Otherwise, Q is not connected (z = high impedance)

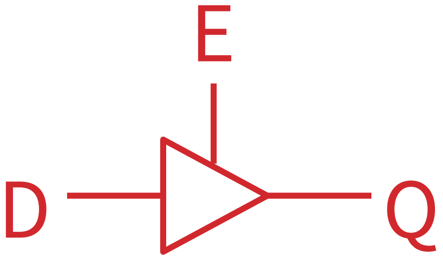


E	D	Q
0	0	z
0	1	z
1	0	0
1	1	1

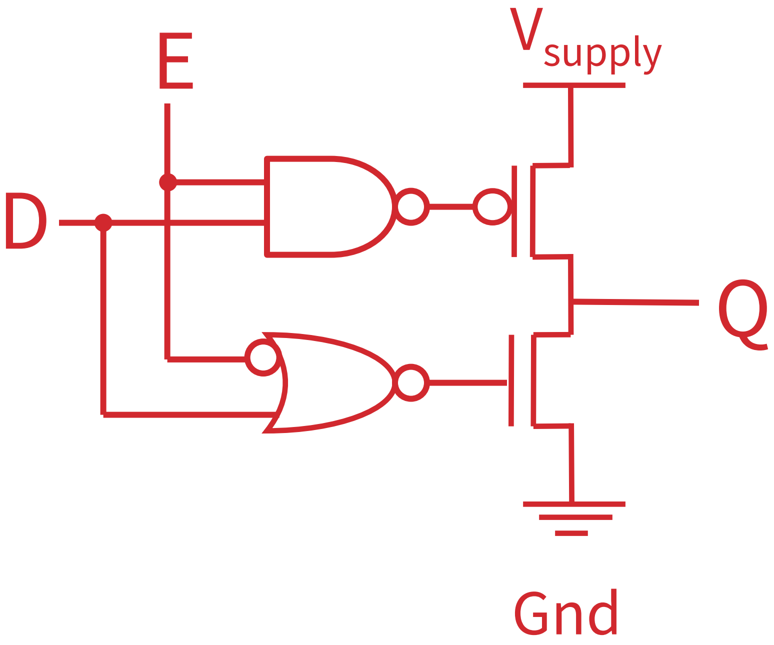


BUILDING LARGE MEMORIES

- Tri-State Buffers
  - □ If enabled (E=1), then Q = D
  - □ Otherwise, Q is not connected (z = high impedance)

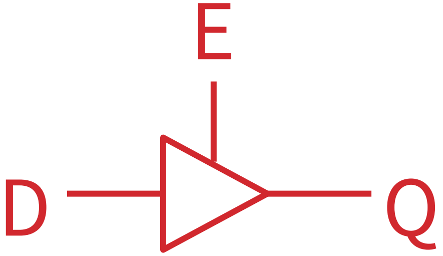


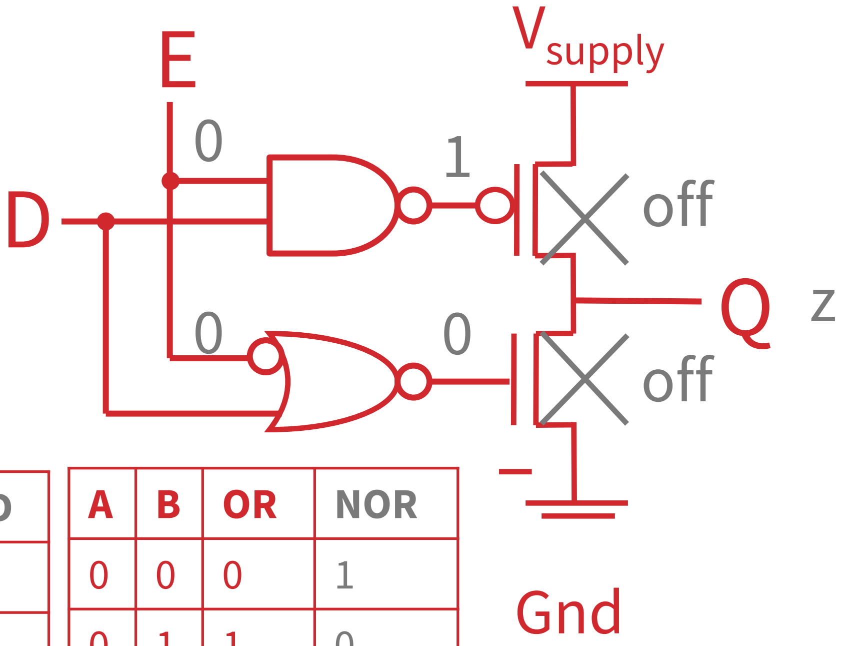
E	D	Q
0	0	z
0	1	z
1	0	0
1	1	1



BUILDING LARGE MEMORIES

- Tri-State Buffers
  - □ If enabled (E=1), then Q = D
  - □ Otherwise, Q is not connected (z = high impedance)





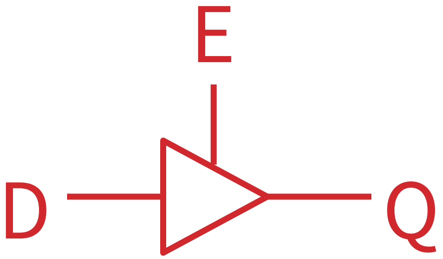
E	D	Q
0	0	z
0	1	z
1	0	0
1	1	1

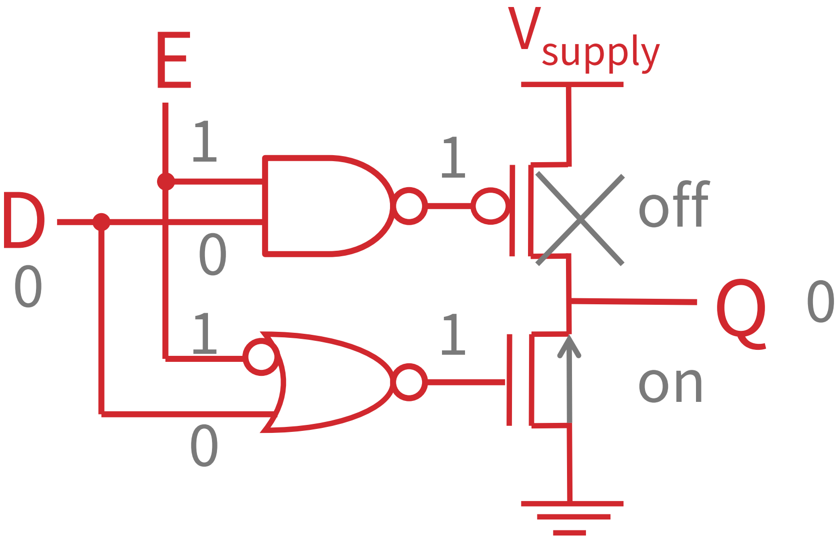
A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

BUILDING LARGE MEMORIES

- Tri-State Buffers
  - □ If enabled (E=1), then Q = D
  - □ Otherwise, Q is not connected (z = high impedance)





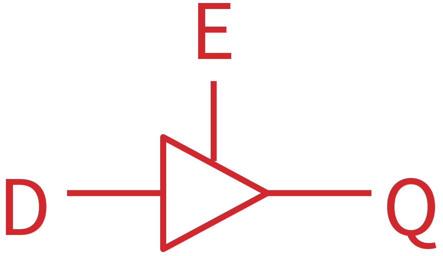
E	D	Q
0	0	z
0	1	z
1	0	0
1	1	1

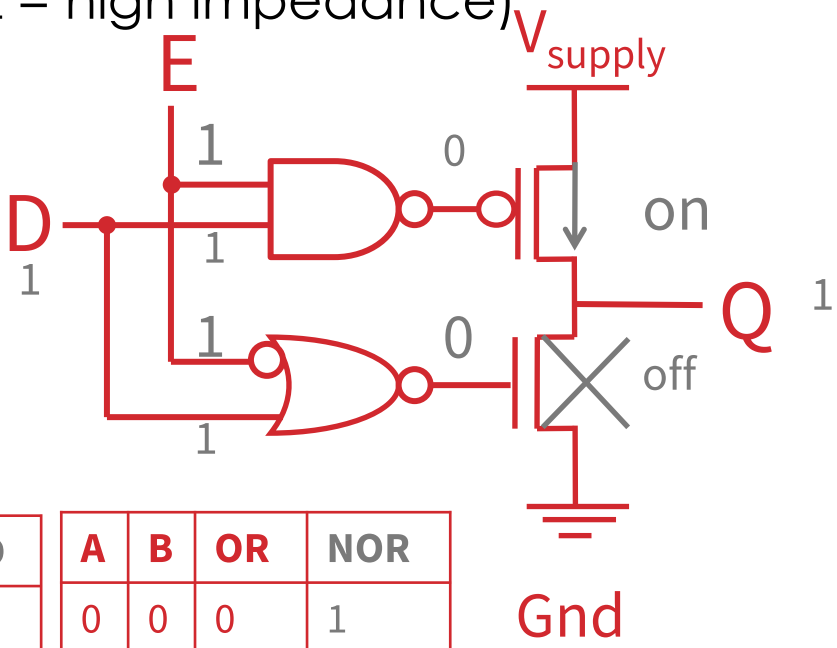
A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

BUILDING LARGE MEMORIES

- Tri-State Buffers
  - If enabled (E=1), then Q = D
  - Otherwise, Q is not connected (z = high impedance)





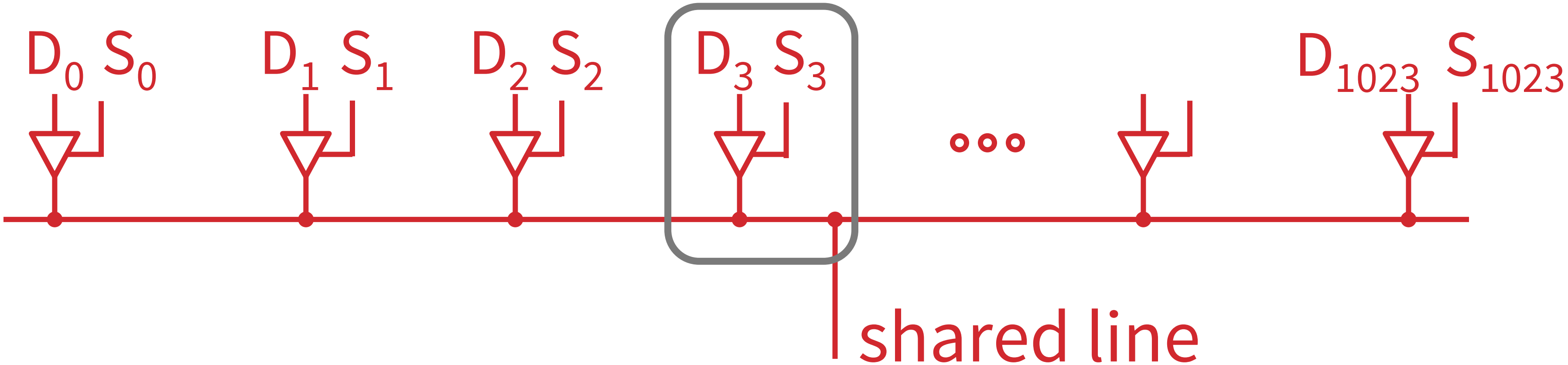
E	D	Q
0	0	z
0	1	z
1	0	0
1	1	1

A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

BUILDING LARGE MEMORIES

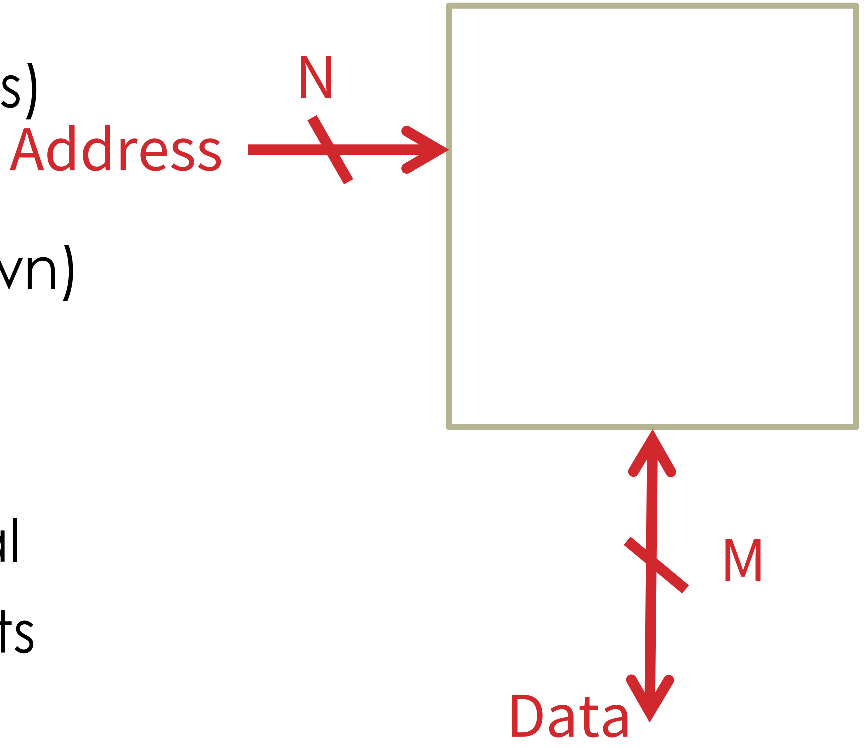
- Shared bus



- Tri-state Buffers allow scaling since multiple registers can be connected to a single output, while only one register actually drives the output

MEMORY

- Storage Cells + bus
- Inputs: Address, Data (for writes)
- Outputs: Data (for reads)
- Also need R/W signal (not shown)

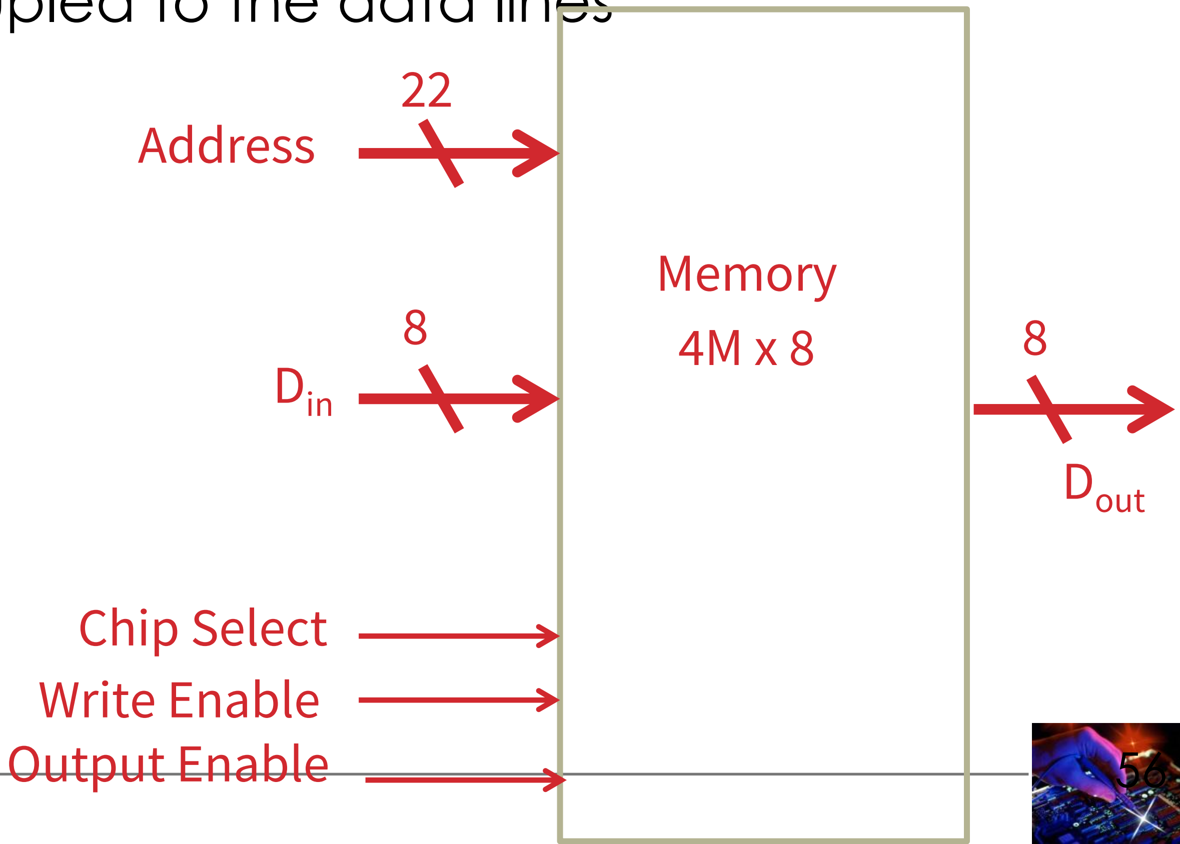


- N address bits -> 2^N words total
- M data bits -> each word M bits

- How many address bits are necessary for a 4M x 8 SRAM module?
  - □ 4M word lines that are each 8 bits wide

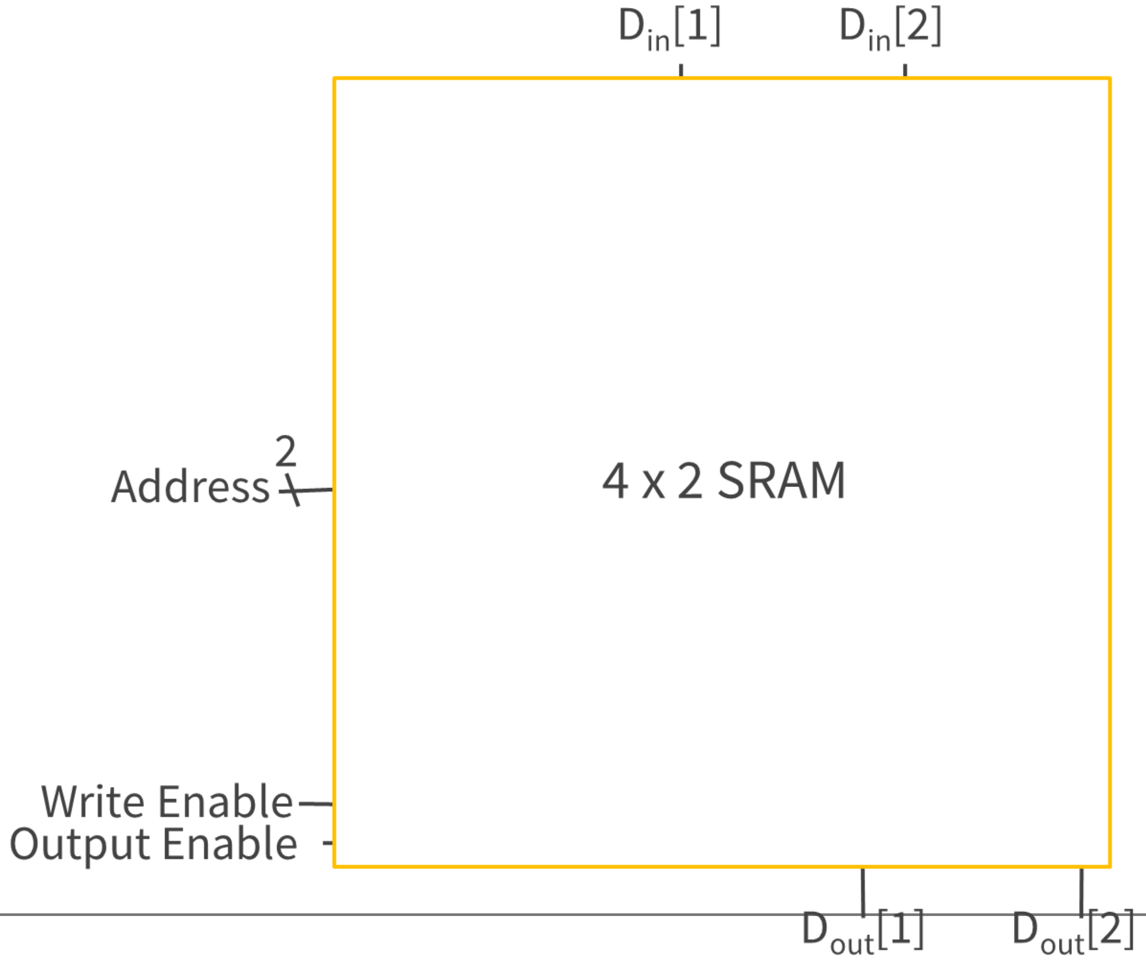
MEMORY

- Storage Cells + bus
- Decoder selects a word line
- R/W selector determines access type
- Word line is then coupled to the data lines



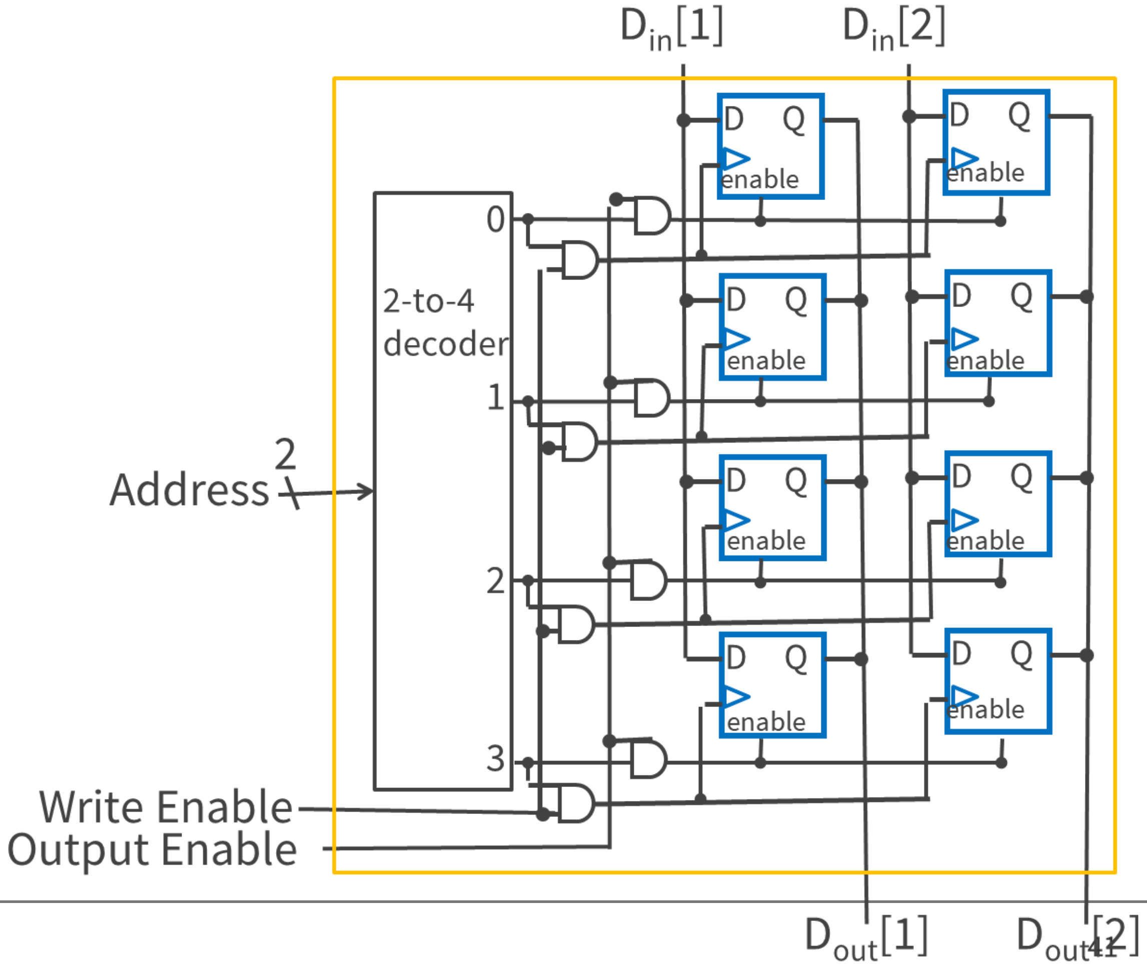
MEMORY

- How do we design a 4 x 2 Memory Module?
  - 4 word lines that are each 2 bits wide



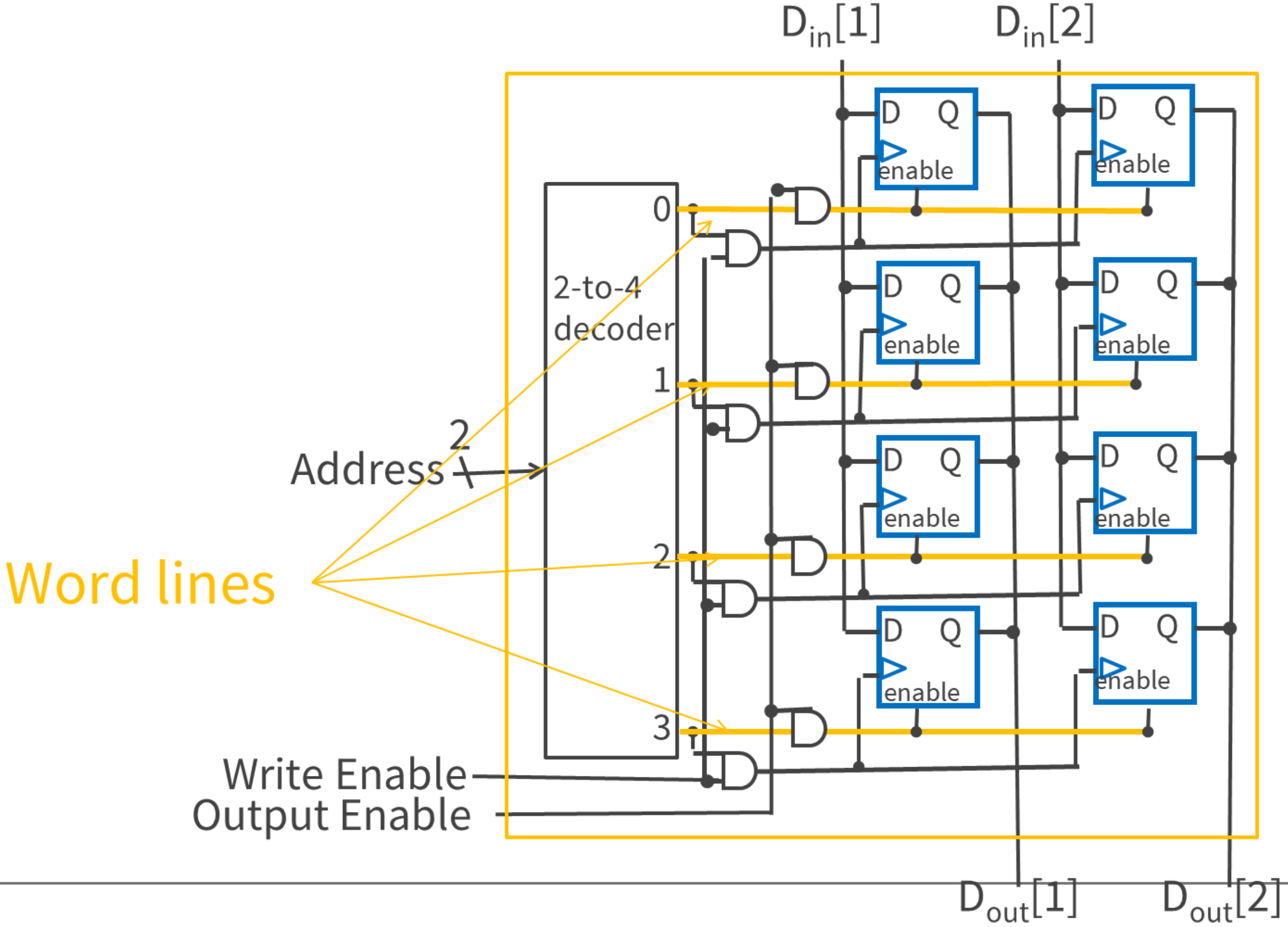
MEMORY

- How do we design a 4 x 2 Memory Module?
  - □ 4 word lines that are each 2 bits wide



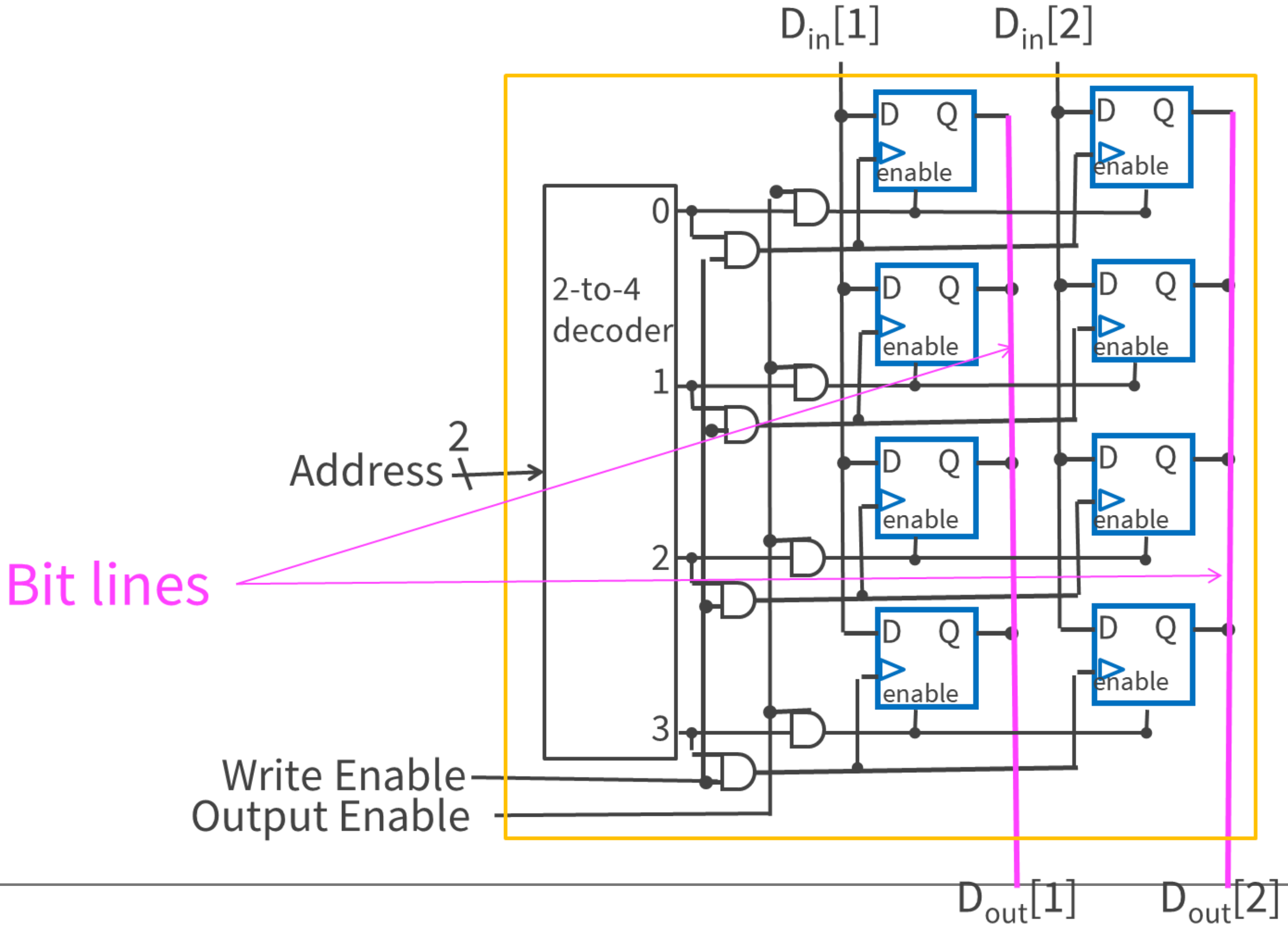
MEMORY

- How do we design a 4 x 2 Memory Module?
  - 4 word lines that are each 2 bits wide



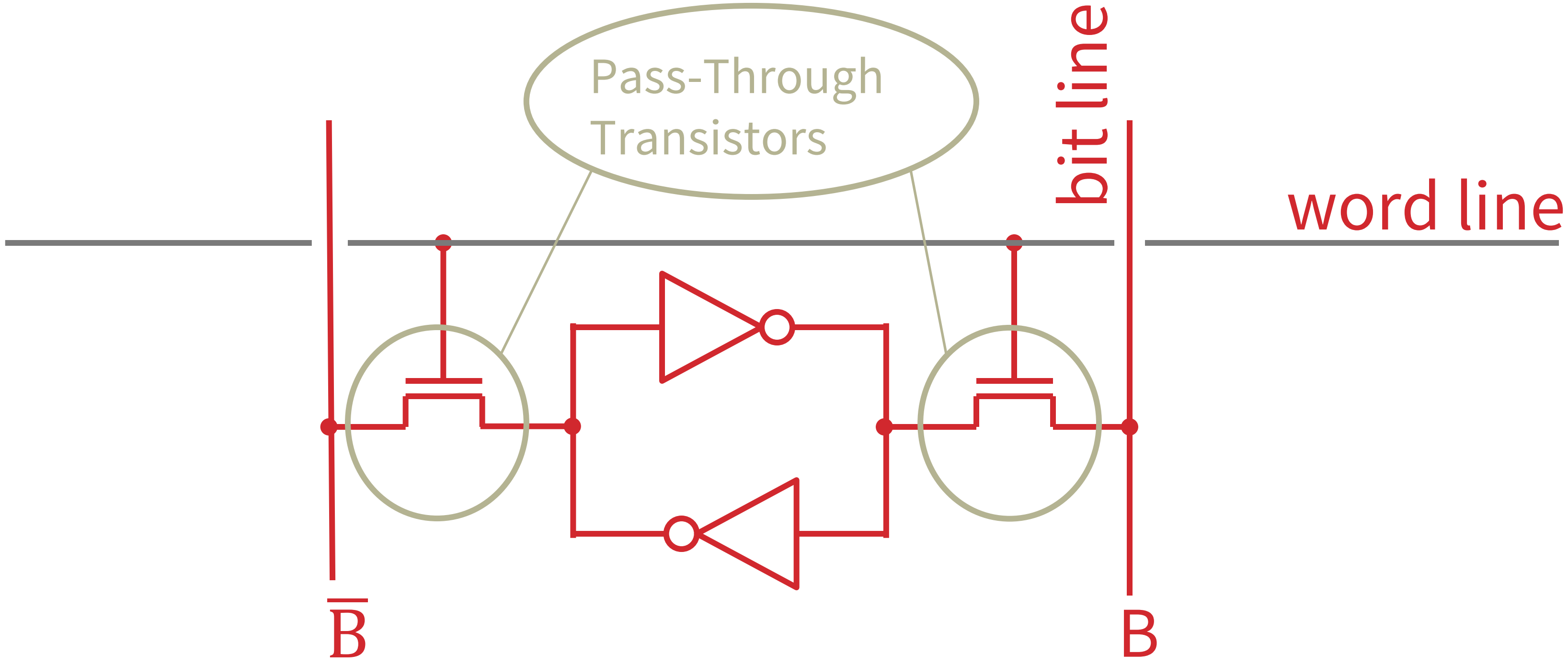
MEMORY

- How do we design a 4 x 2 Memory Module?
  - □ 4 word lines that are each 2 bits wide



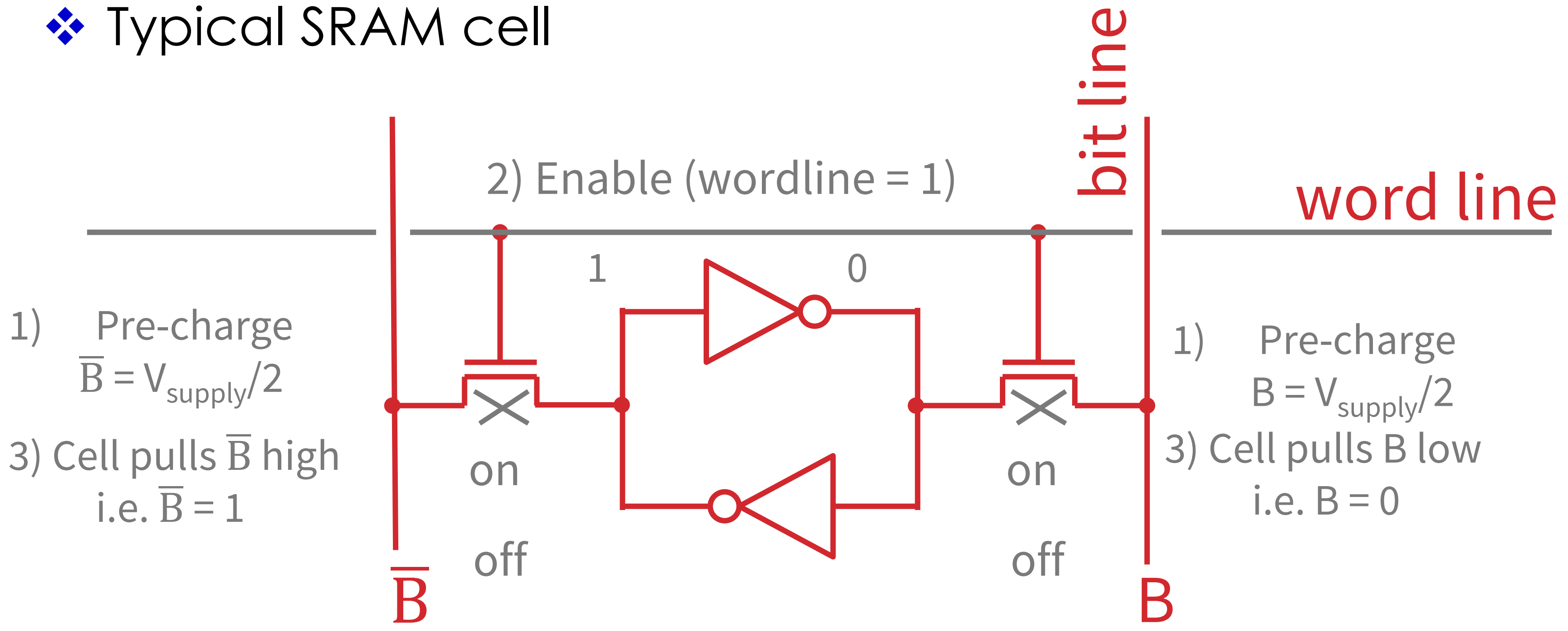
SRAM CELL

- Typical SRAM cell
  - Each cell stores one bit, and requires 4 – 8 transistors (6 is typical)



SRAM CELL

- Typical SRAM cell



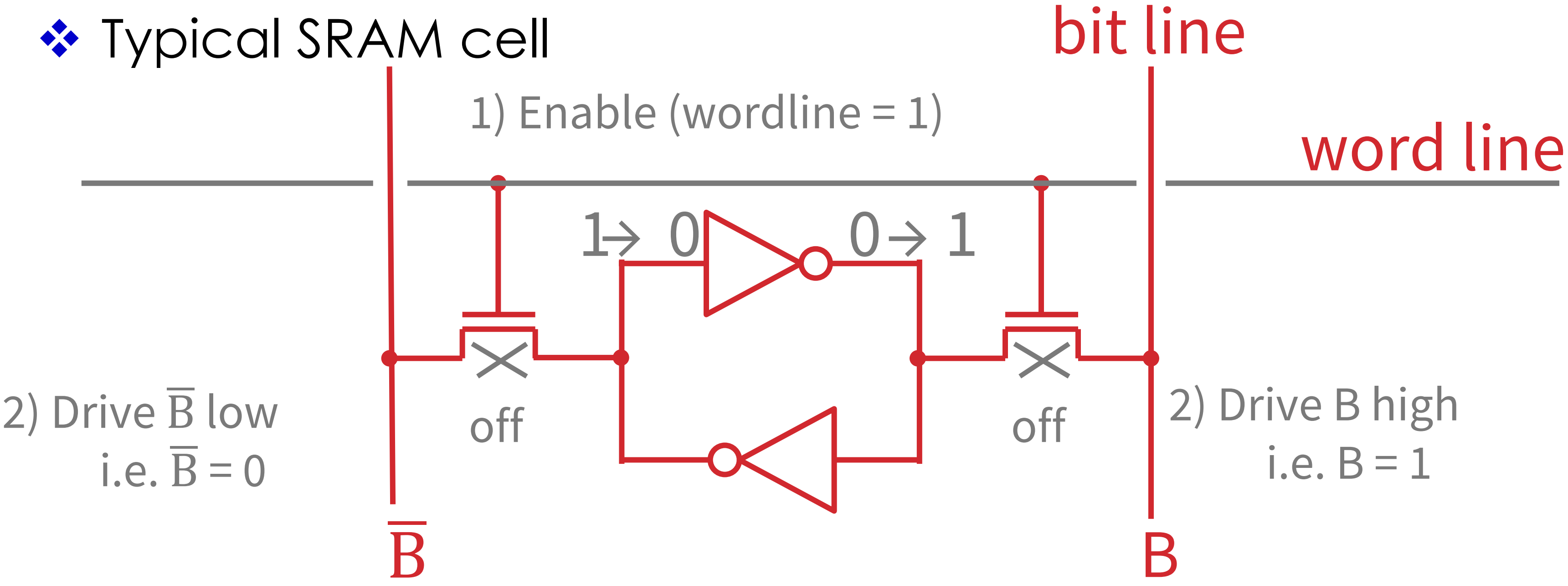
Each cell stores one bit, and requires 4 – 8 transistors (6 is typical)

Read:

- pre-charge B and $\overline{B}$ to $V_{supply}/2$
- pull word line high
- cell pulls B or $\overline{B}$ low, sense amp detects voltage difference

SRAM CELL

- Typical SRAM cell



1) Enable (wordline = 1)

2) Drive $\overline{B}$ low
i.e. $\overline{B}$ = 0

2) Drive B high
i.e. B = 1

Each cell stores one bit, and requires 4 – 8 transistors (6 is typical)

Read:

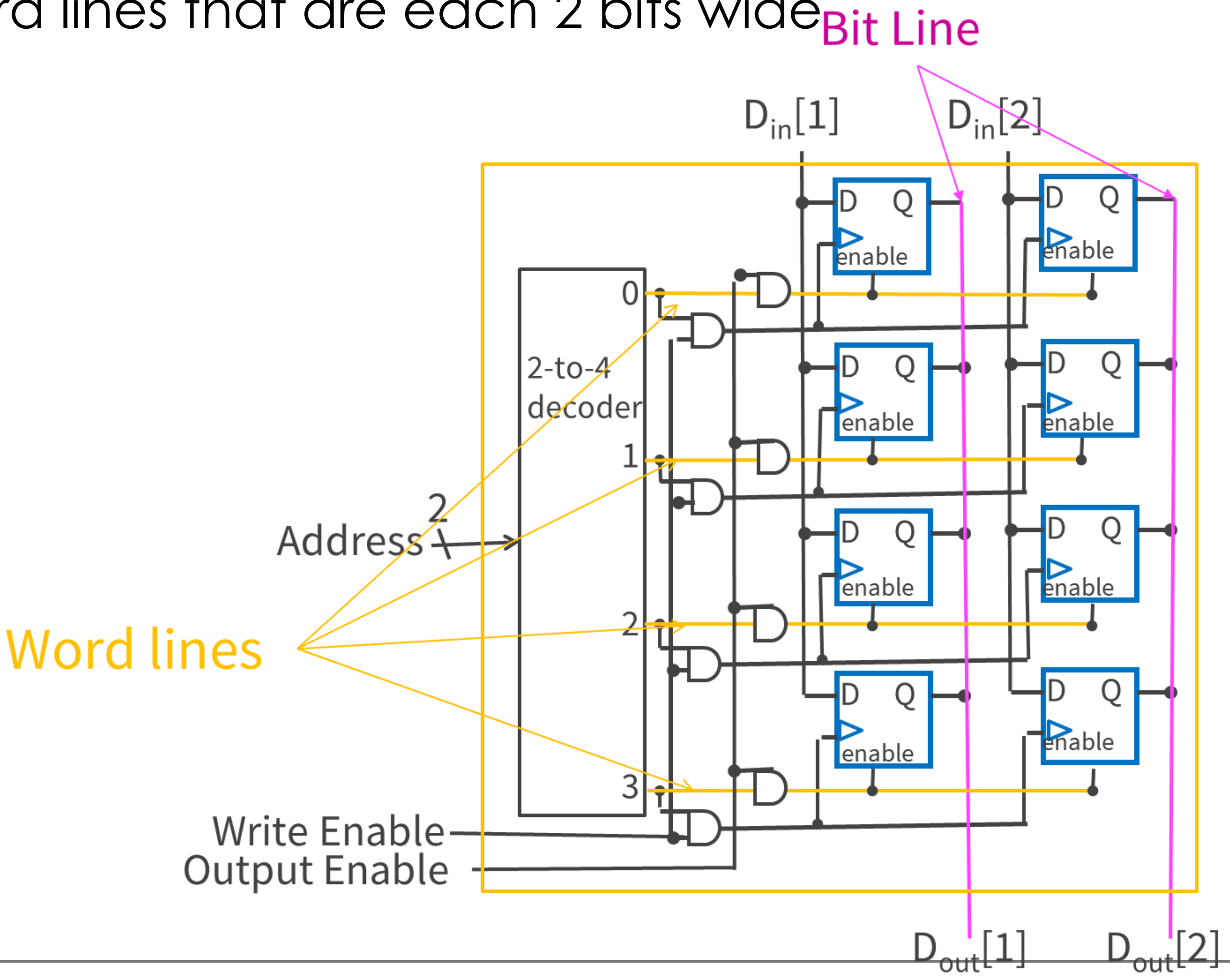
- pre-charge B and $\overline{B}$ to $V_{supply}/2$
- pull word line high
- cell pulls B or $\overline{B}$ low, sense amp detects voltage difference

Write:

- pull word line high
- drive B and $\overline{B}$ to flip cell

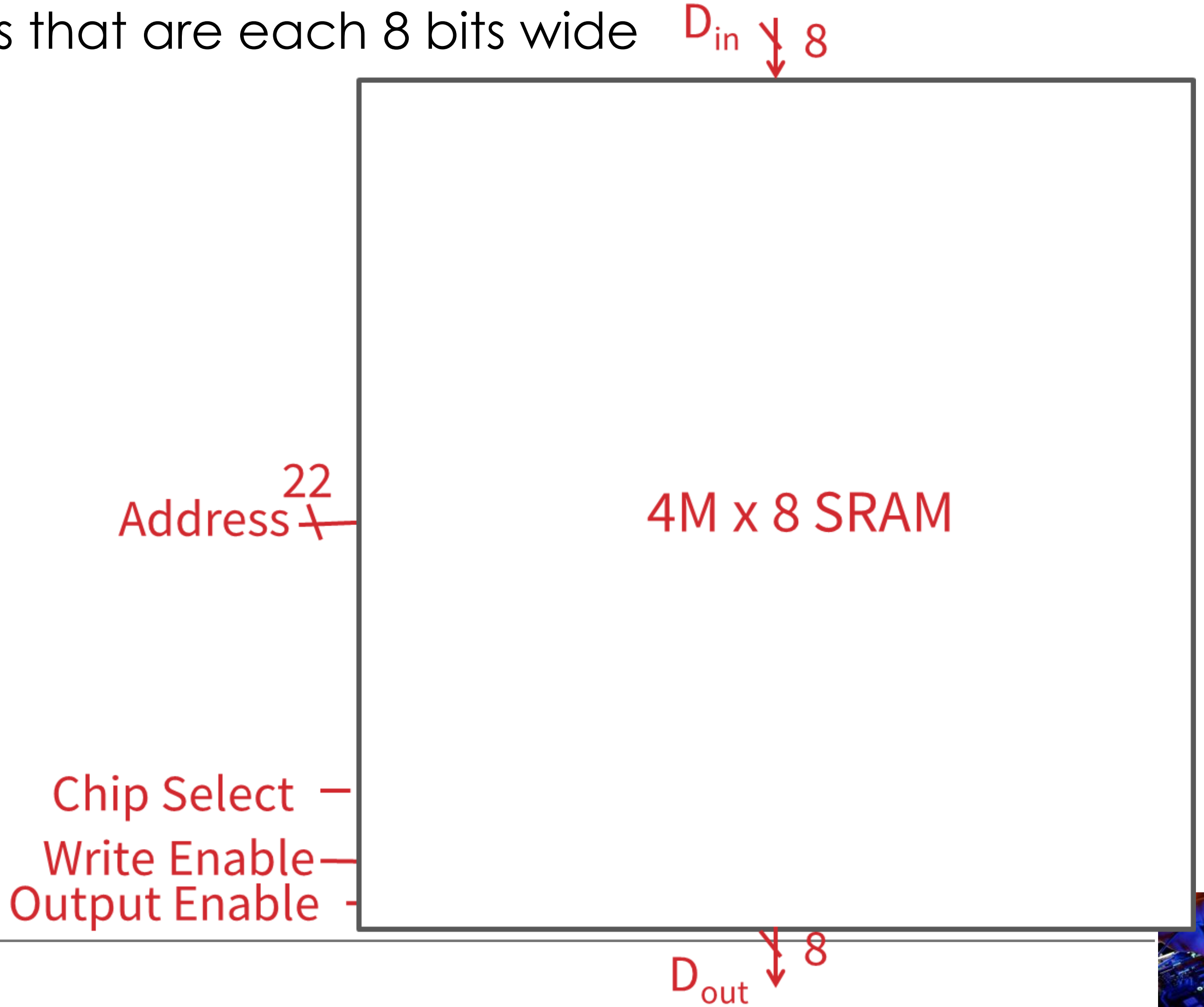
MEMORY

- How do we design a 4 x 2 Memory Module?
  - 4 word lines that are each 2 bits wide



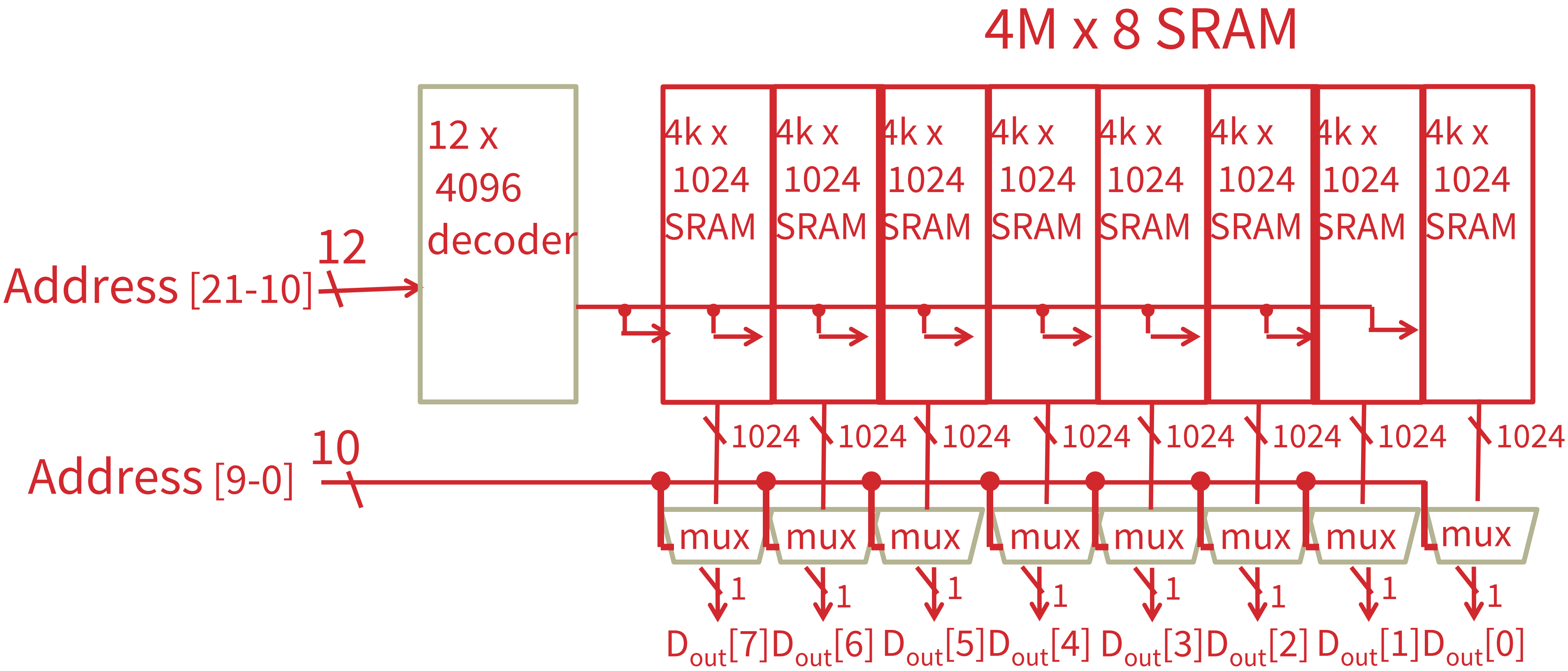
MEMORY

- How do we design a 4M x 8 Memory Module?
  - □ 4M word lines that are each 8 bits wide



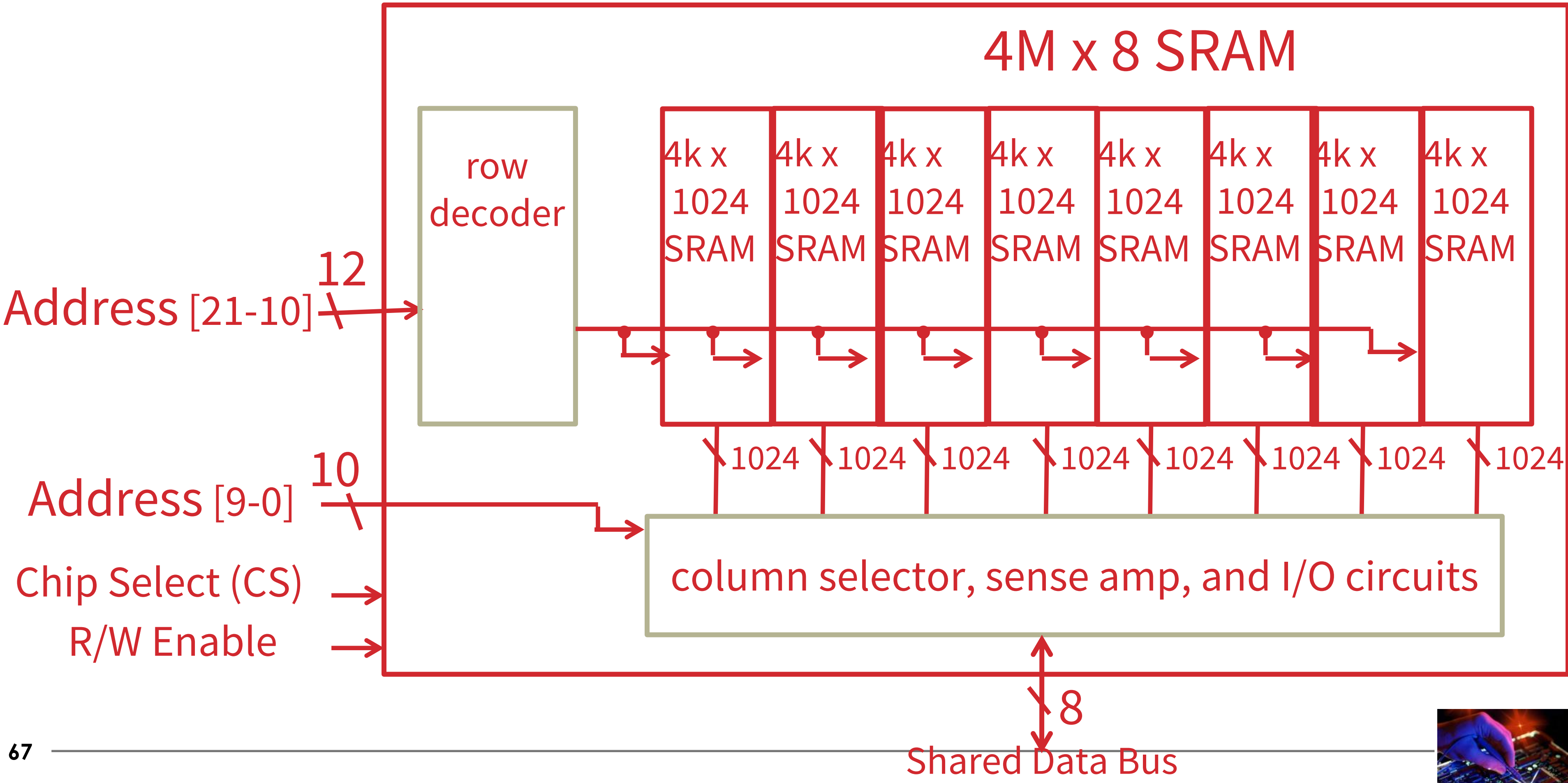
MEMORY

- How do we design a 4M x 8 Memory Module?
  - 4M word lines that are each 8 bits wide



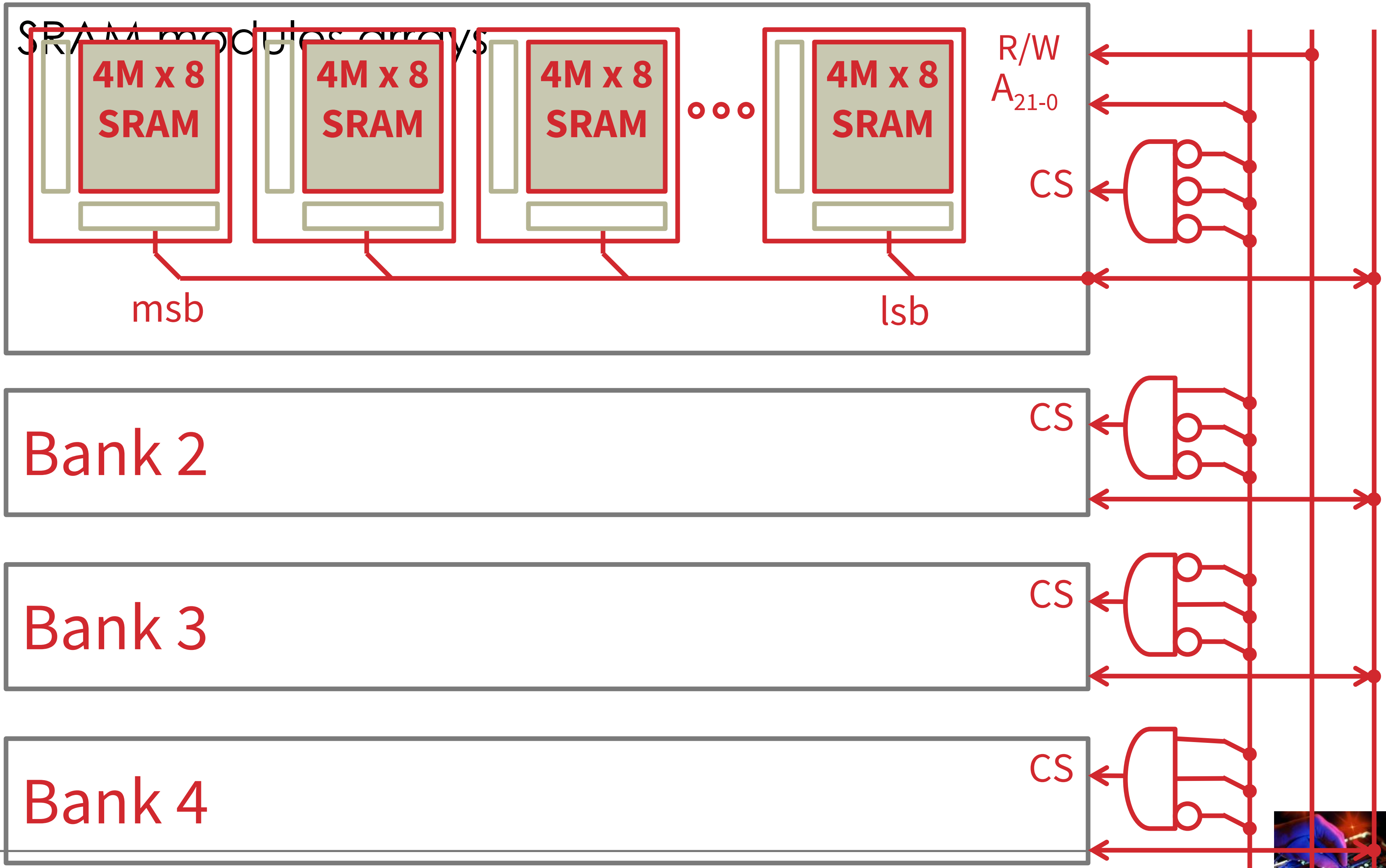
MEMORY

- How do we design a 4M x 8 Memory Module?
  - □ 4M word lines that are each 8 bits wide



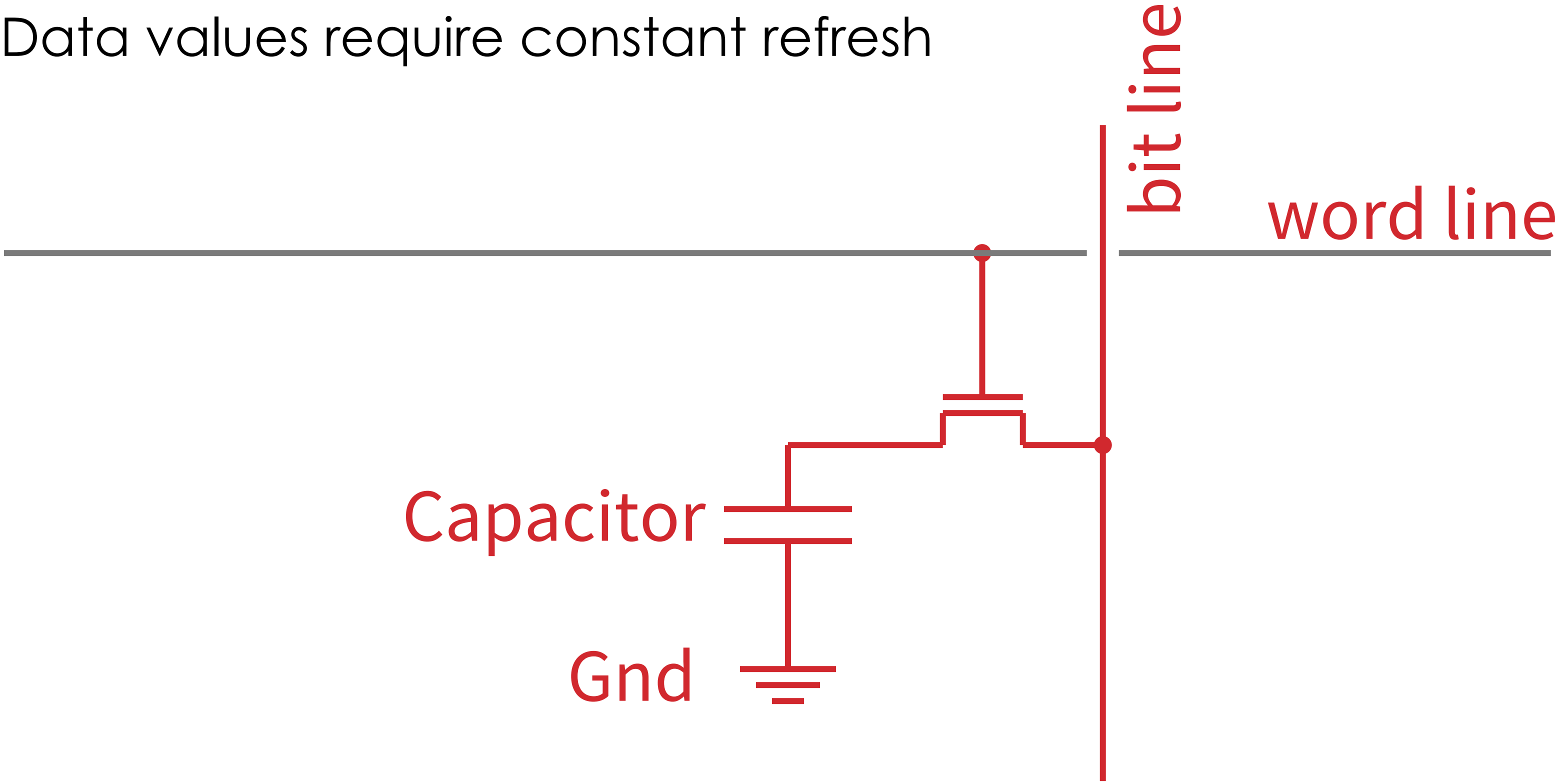
MEMORY

- SRAM modules arrays



MEMORY

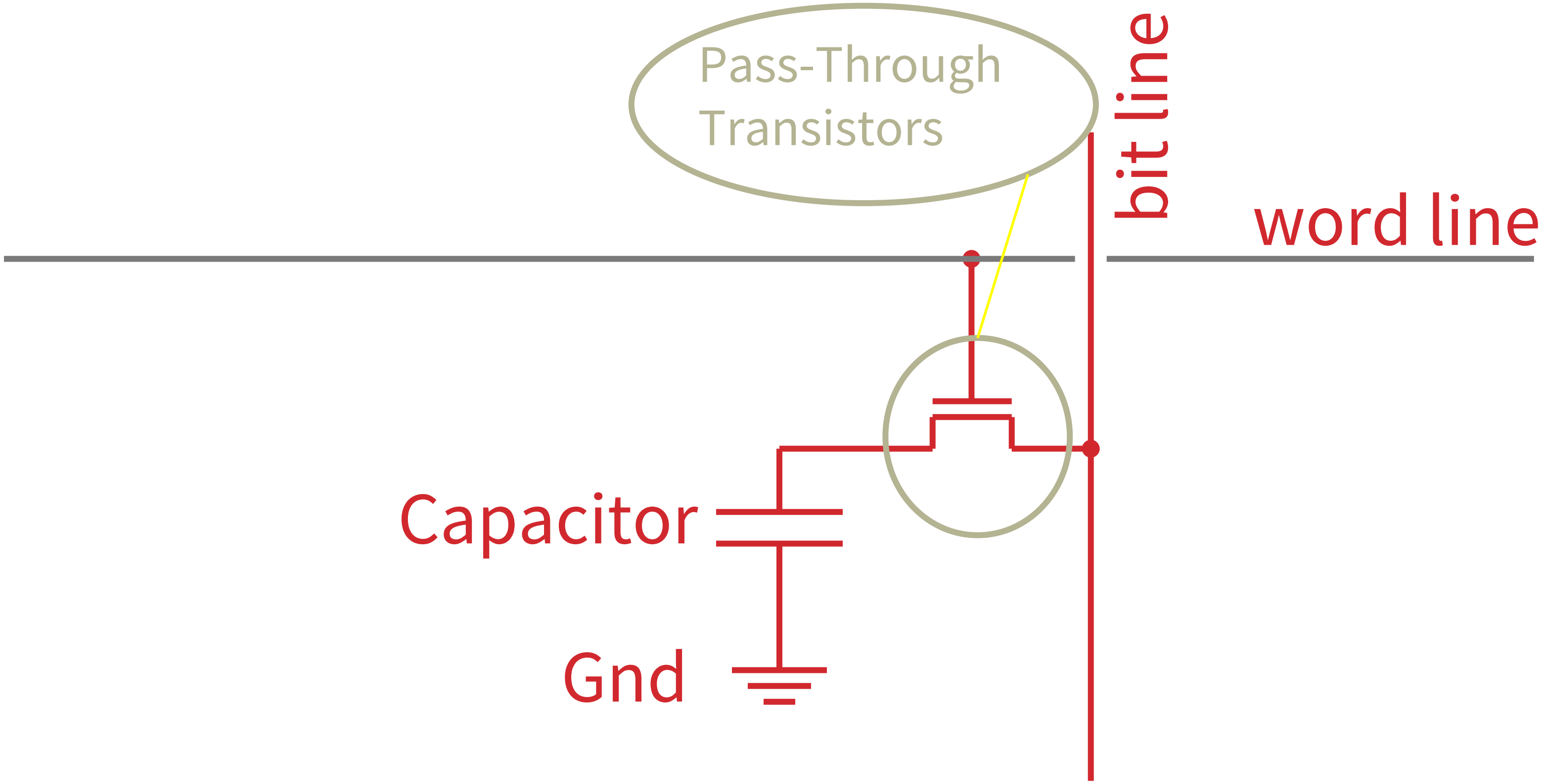
- Dynamic-RAM (DRAM)
  - □ Data values require constant refresh



Each cell stores one bit, and requires 1 transistors

MEMORY

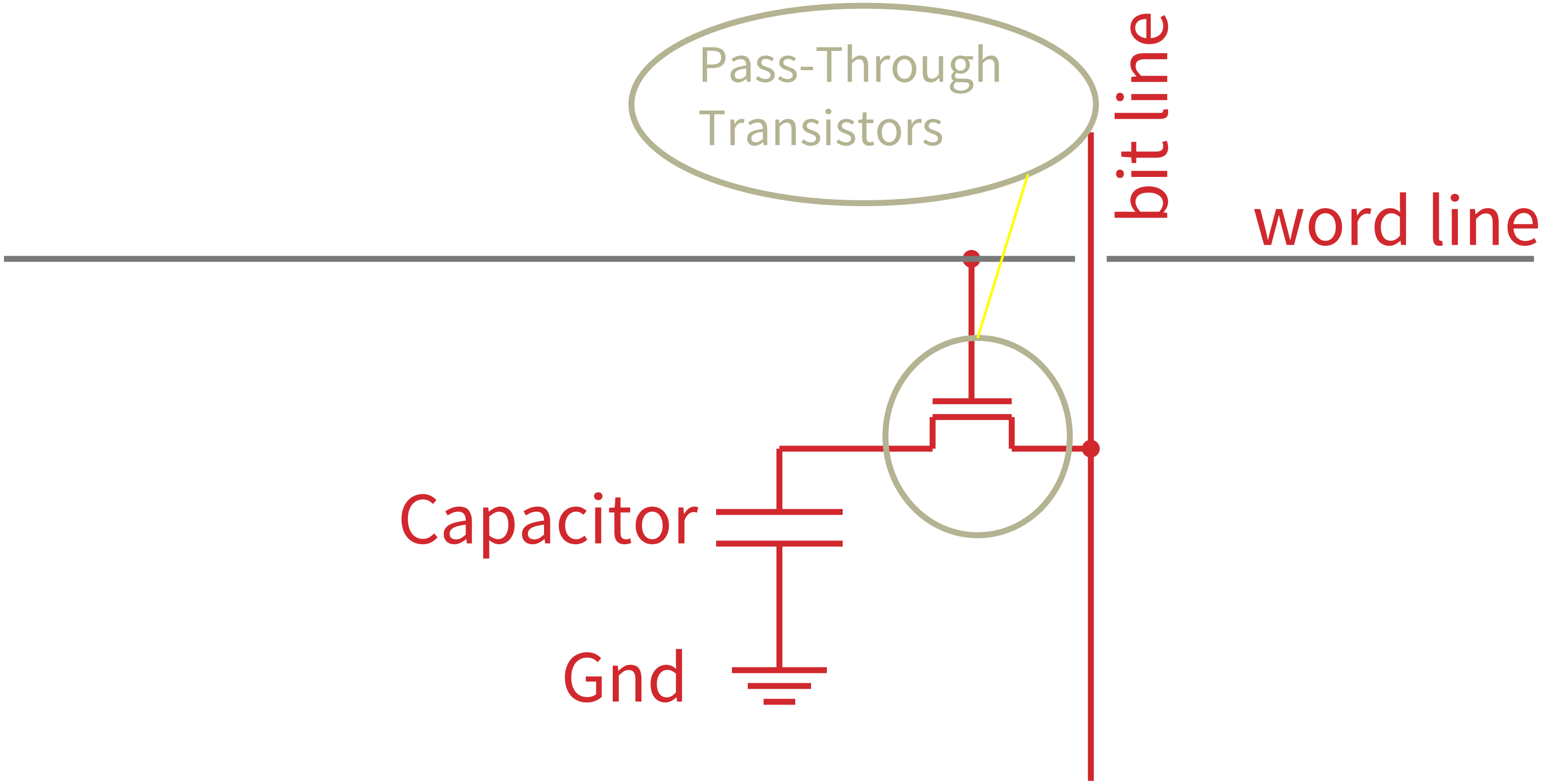
- Dynamic-RAM (DRAM)
  - □ Data values require constant refresh



Each cell stores one bit, and requires 1 transistors

MEMORY

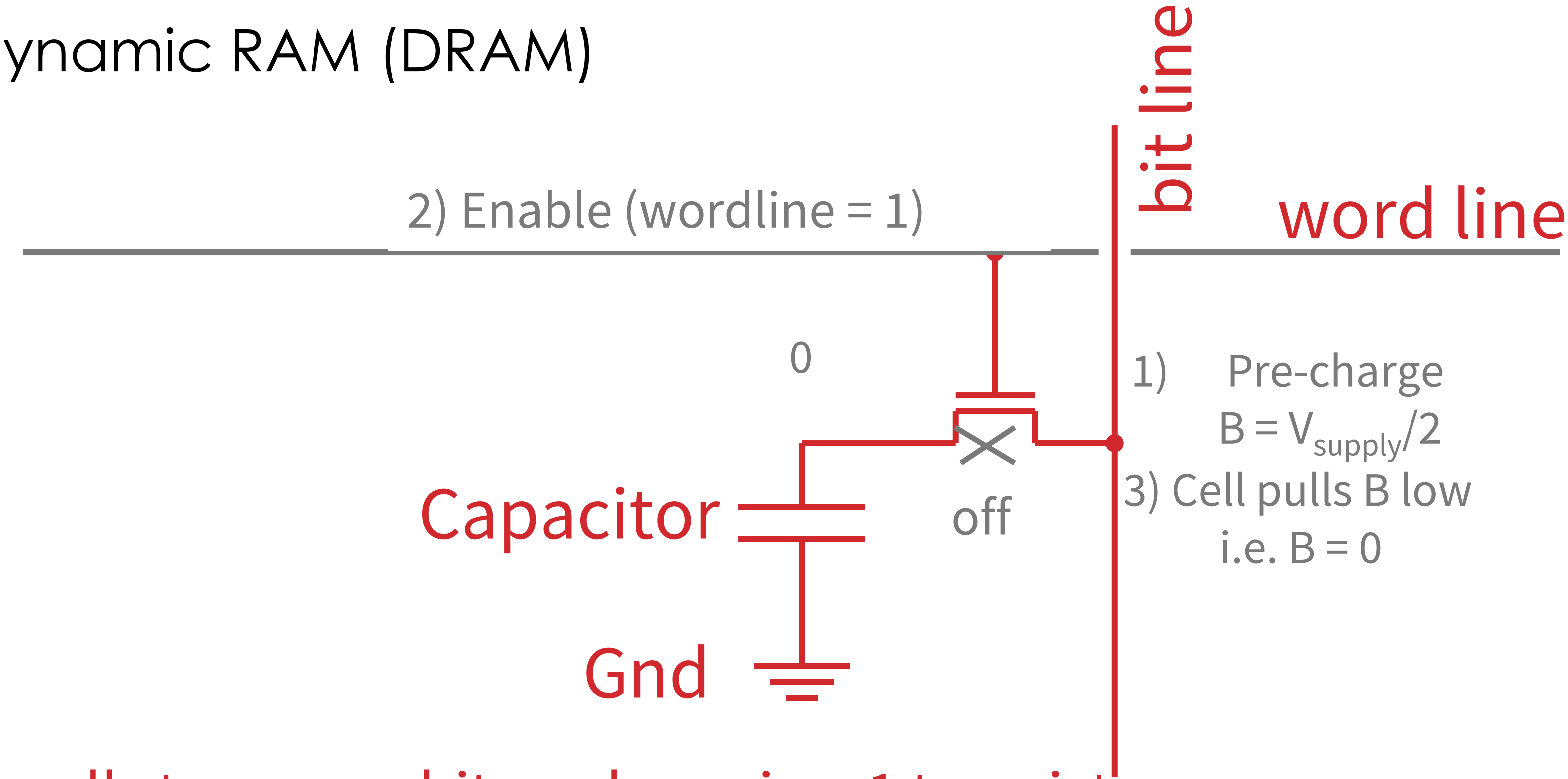
- Dynamic-RAM (DRAM)
  - □ Data values require constant refresh



Each cell stores one bit, and requires 1 transistors

MEMORY

- Dynamic RAM (DRAM)


bit line
2) Enable (wordline = 1)
word line
0
1) Pre-charge B = $V_{supply}/2$
Capacitor
off
3) Cell pulls B low i.e. B = 0
Gnd

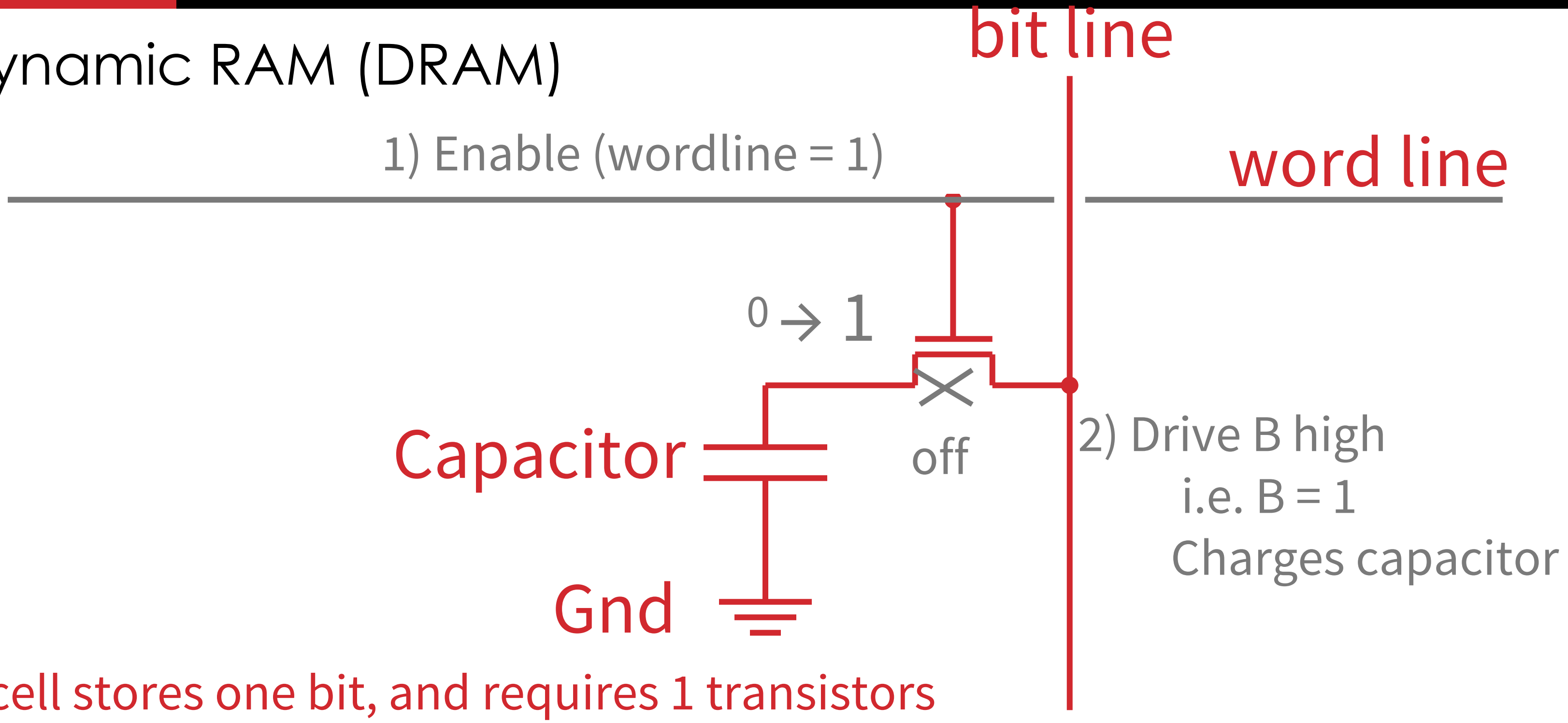
Each cell stores one bit, and requires 1 transistors

Read:

- pre-charge B and $\overline{B}$ to $V_{supply}/2$
- pull word line high
- cell pulls B low, sense amp detects voltage difference

MEMORY

- Dynamic RAM (DRAM)



bit line

word line

1) Enable (wordline = 1)

0 → 1

off

Capacitor

Gnd

2) Drive B high
i.e. B = 1
Charges capacitor

Each cell stores one bit, and requires 1 transistors

Read:

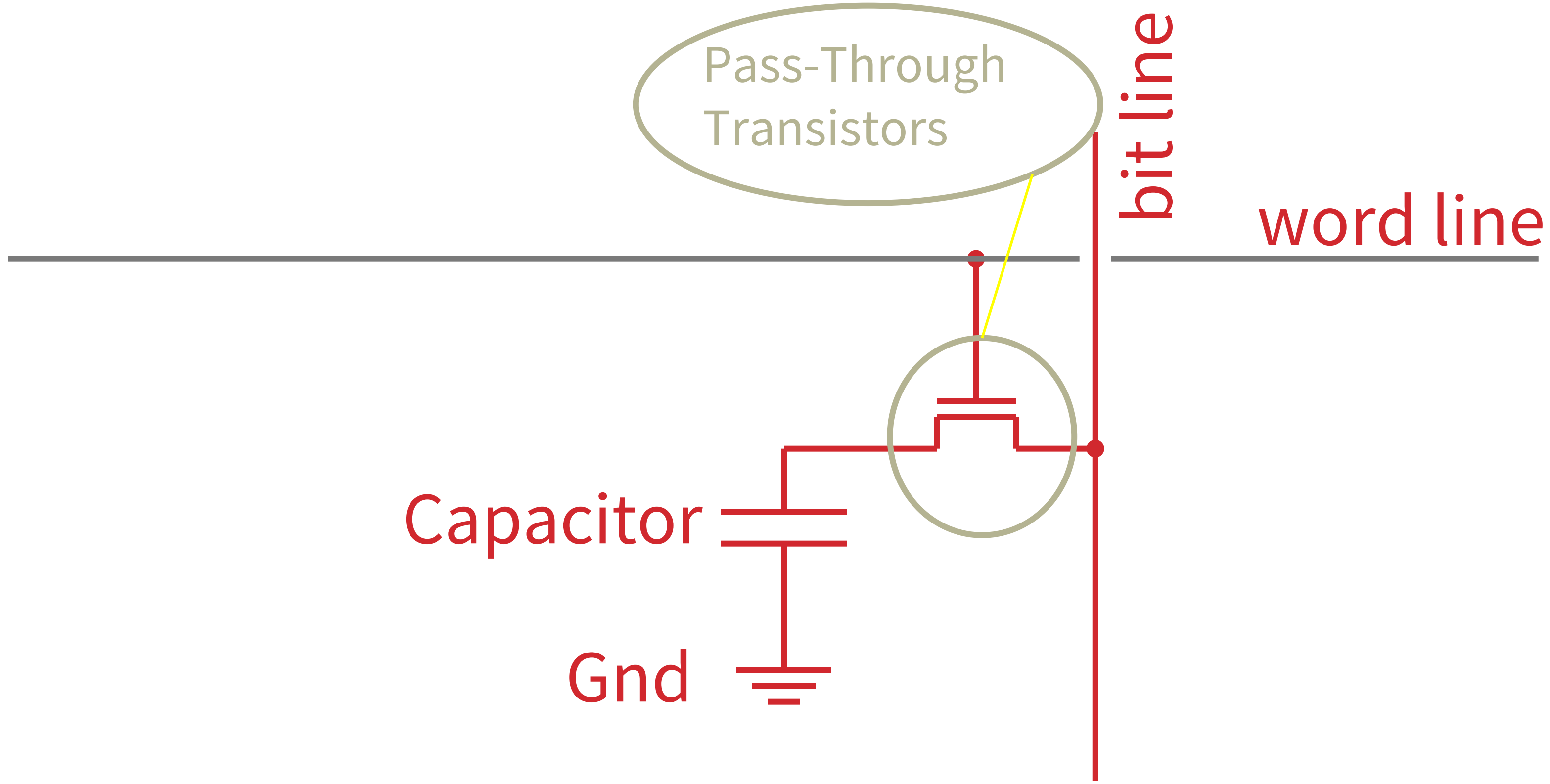
- pre-charge B and $\overline{\mathrm{B}}$ to $V_{supply}/2$
- pull word line high
- cell pulls B low, sense amp detects voltage difference

Write:

- pull word line high
- drive B charges capacitor

MEMORY

- Dynamic-RAM (DRAM)
  - □ Data values require constant refresh



Each cell stores one bit, and requires 1 transistors

MEMORY

- Register File tradeoffs
  - + Very fast (a few gate delays for both read and write)
  - + Adding extra ports is straightforward
  - – Expensive, doesn’t scale
  - – Volatile

- Volatile Memory alternatives: SRAM, DRAM, …
  - – Slower
  - + Cheaper, and scales well
  - – Volatile

- Non-Volatile Memory (NV-RAM): Flash, EEPROM, …
  - + Scales well
  - – Limited lifetime; degrades after 100000 to 1M writes