# Memory

**Prof. Hakim Weatherspoon**
**CS 3410**
Computer Science
Cornell University

[Weatherspoon, Bala, Bracy, and Sirer]

# Announcements

Make sure you are

- Registered for class, can access CMS
- Have a Section you can go to.
- *Lab Sections are required.*

- Project partners are required for projects starting w/ project 2
  - Project partners will be assigned (from the same lab section, if possible)

# Announcements

- Make sure to go to ***<u>your</u>*** Lab Section this week
- Completed **Proj1** due Friday, Feb 15th
- Note, a Design Document is due when you submit Proj1 final circuit
- Work **alone**

**BUT** use your resources

- Lab Section, Piazza.com, Office Hours
- Class notes, book, Sections, CSUGLab

# Announcements

Check online syllabus/schedule

- http://www.cs.cornell.edu/Courses/CS3410/2019sp/schedule
- Slides and Reading for lectures
- Office Hours
- ***Pictures of all TAs***
- Project and Reading Assignments
- **Dates to keep in Mind**
    - Prelims: Tue Mar 5th and Thur May 2nd
    - ***Proj 1: Due next Friday, Feb 15th***
    - Proj3: Due before Spring break
    - Final Project: May 16th

Schedule is subject to change

# Announcements

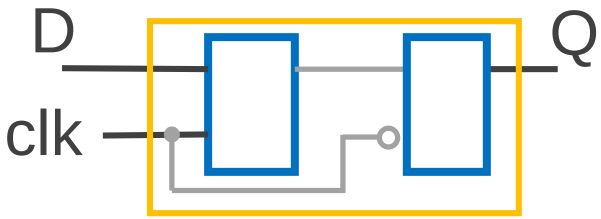
- Level Up (optional enrichment)
  - Teaches CS students tools and skills needed in their coursework as well as their career, such as Git, Bash Programming, study strategies, ethics in CS, and even applying to graduate school.
  - Thursdays at 7-8pm in 310 Gates Hall, starting this week
  - http://www.cs.cornell.edu/courses/cs3110/2019sp/levelup/

# Goals for today

Memory

- CPU: Register Files (i.e. Memory w/in the CPU)
- Scaling Memory: Tri-state devices
- Cache: SRAM (Static RAM—random access memory
- Memory: DRAM (Dynamic RAM)

# Last time: How do we store one bit

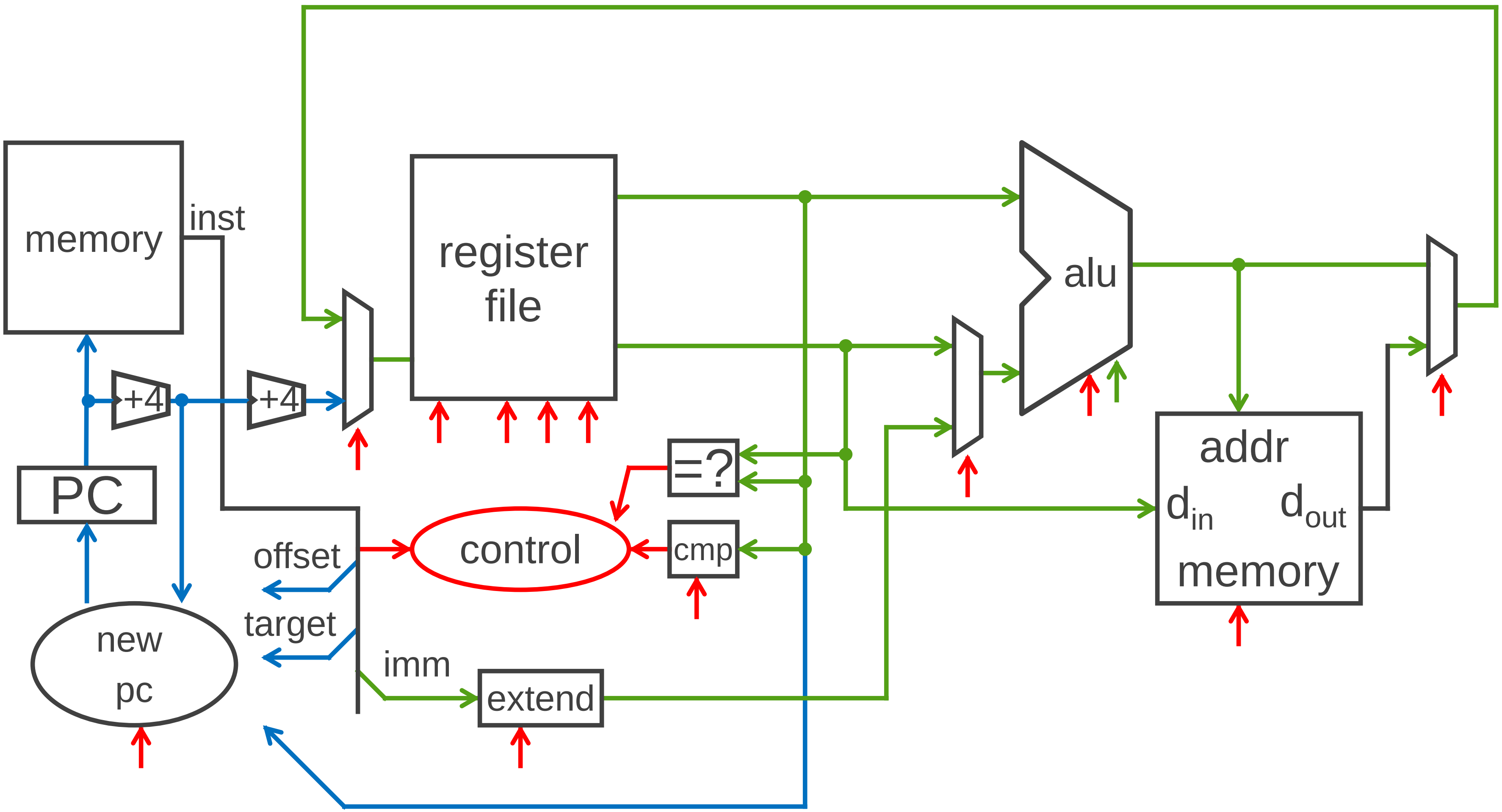



D Flip Flop stores 1 bit

# Goal for today

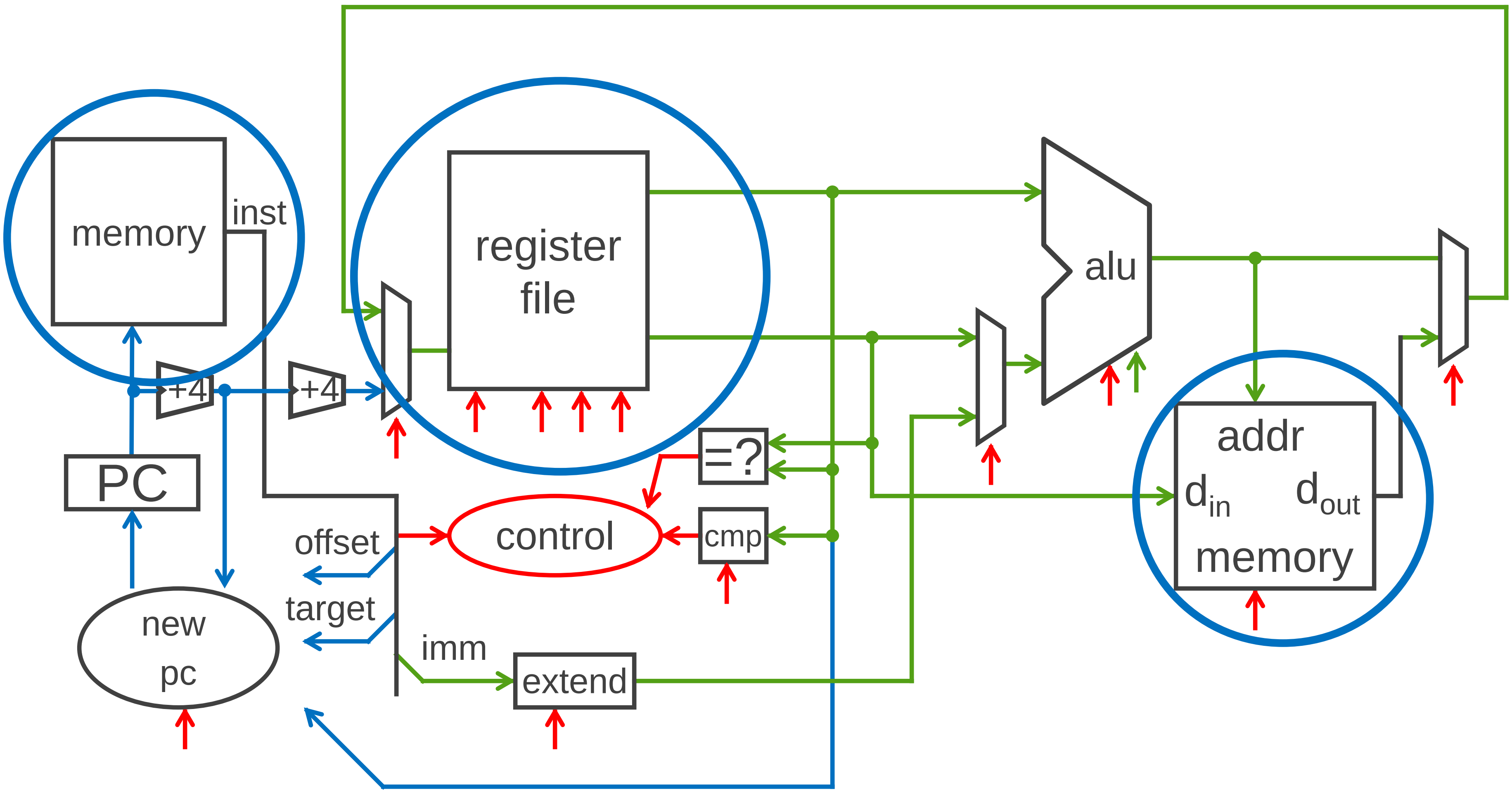
How do we store results from ALU computations?

# Big Picture: Building a Processor




A Single cycle processor

# Big Picture: Building a Processor




A Single cycle processor

# Goal for today

How do we store results from ALU computations?

How do we use stored results in subsequent operations?

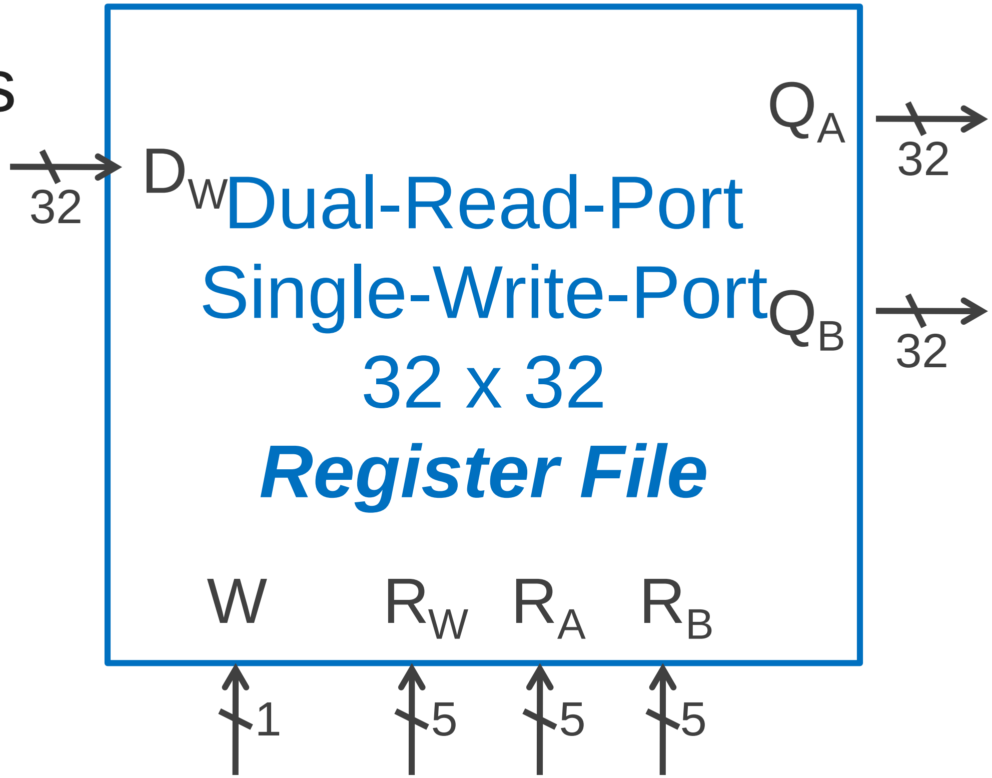
Register File

How does a Register File work? How do we design it?

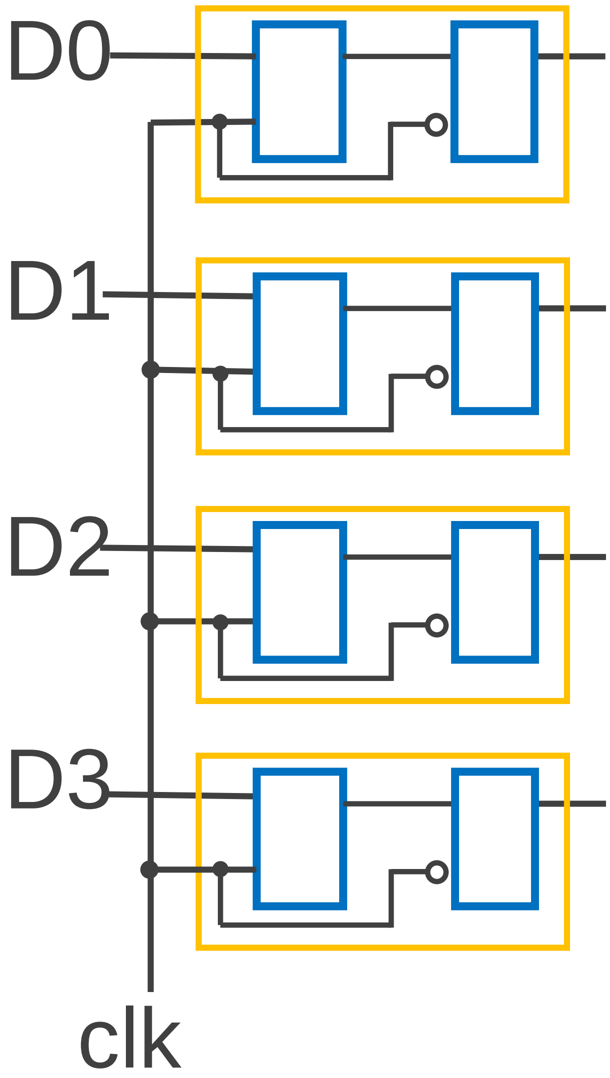
# Register File

## Register File

- N read/write registers
- Indexed by register number

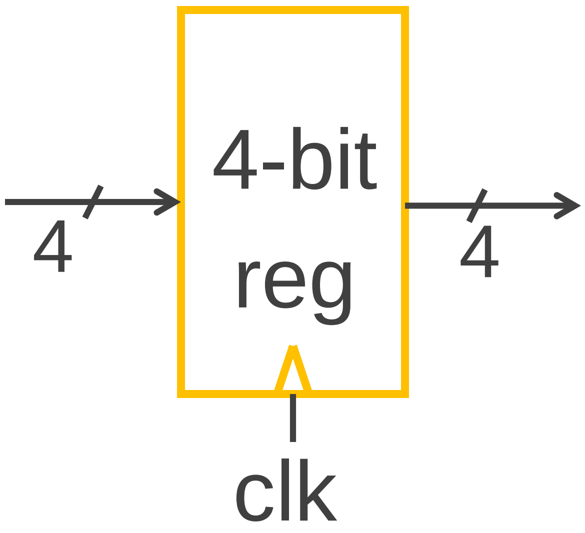


# Register File

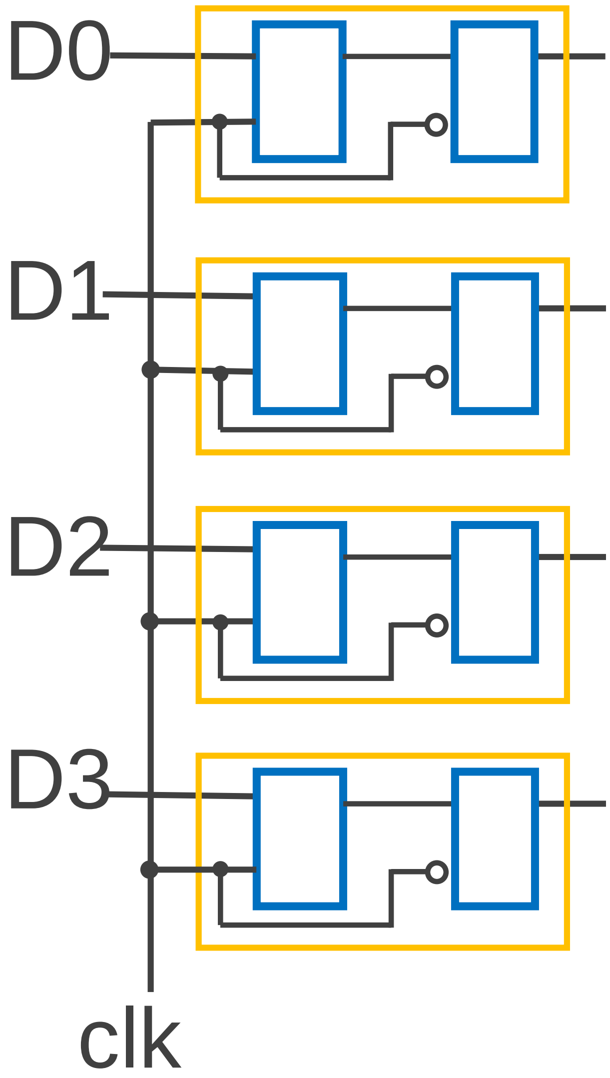



## Recall: Register

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

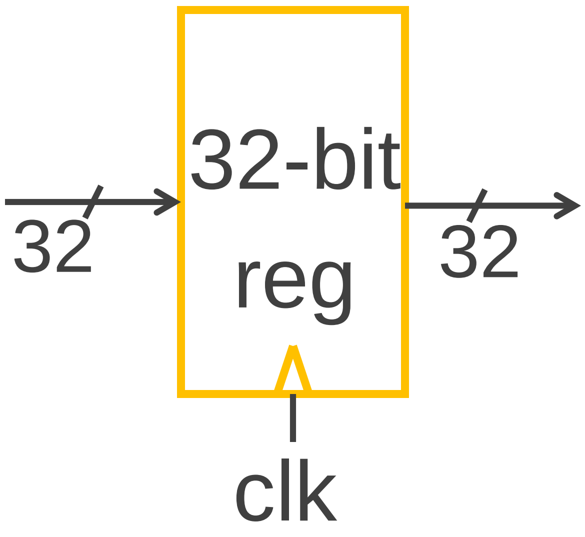


# Register File




## Recall: Register

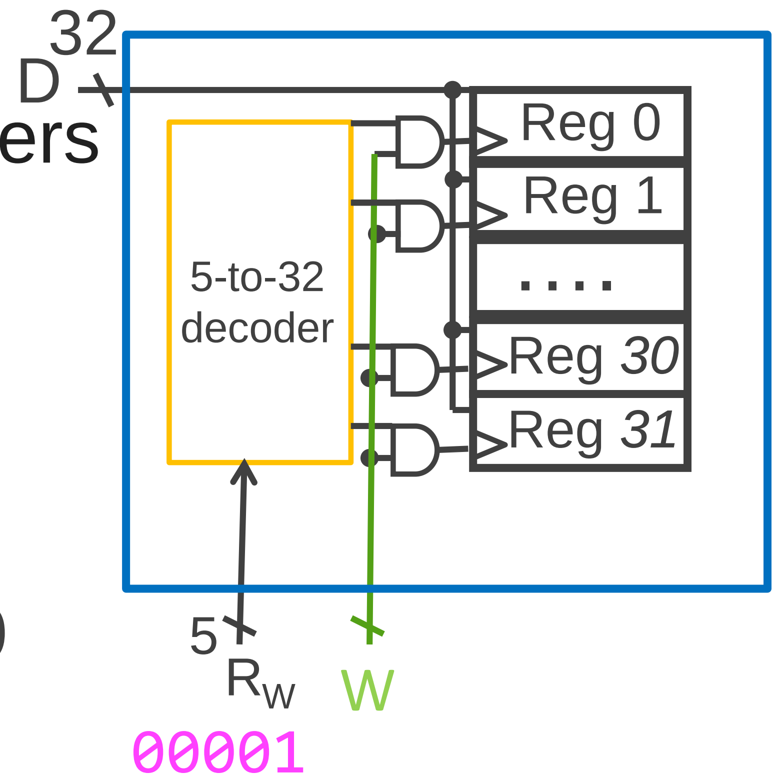
- D flip-flops in parallel
- shared clock
- extra clocked inputs: write_enable, reset, …



# Register File

## Register File

- N read/write registers
- Indexed by register number



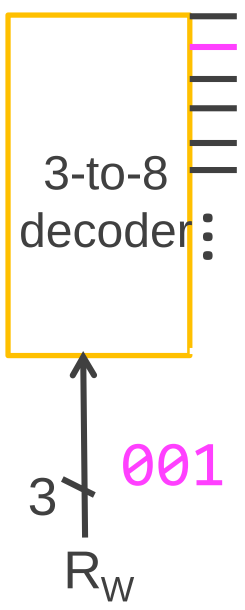

```
addi x1, x0, 10
```

## How to write to ***one*** register in the register file?

- Need a decoder

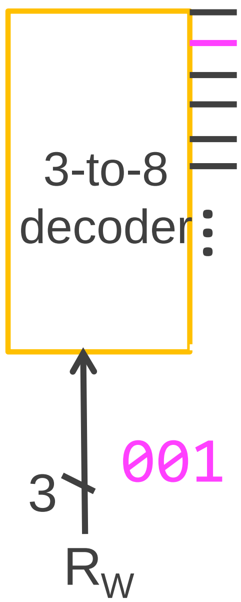
# Aside: 3-to-8 decoder truth table & circuit

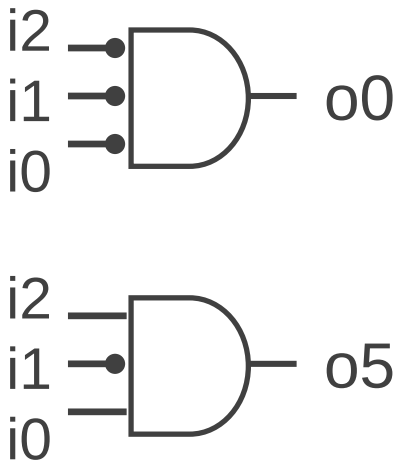
| i2 | i1 | i0 | o0 | o1 | o2 | o3 | o4 | o5 | o6 | o7 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | | | | | | | | |
| 0 | 0 | 1 | | | | | | | | |
| 0 | 1 | 0 | | | | | | | | |
| 0 | 1 | 1 | | | | | | | | |
| 1 | 0 | 0 | | | | | | | | |
| 1 | 0 | 1 | | | | | | | | |
| 1 | 1 | 0 | | |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



# Aside: 3-to-8 decoder truth table & 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 |

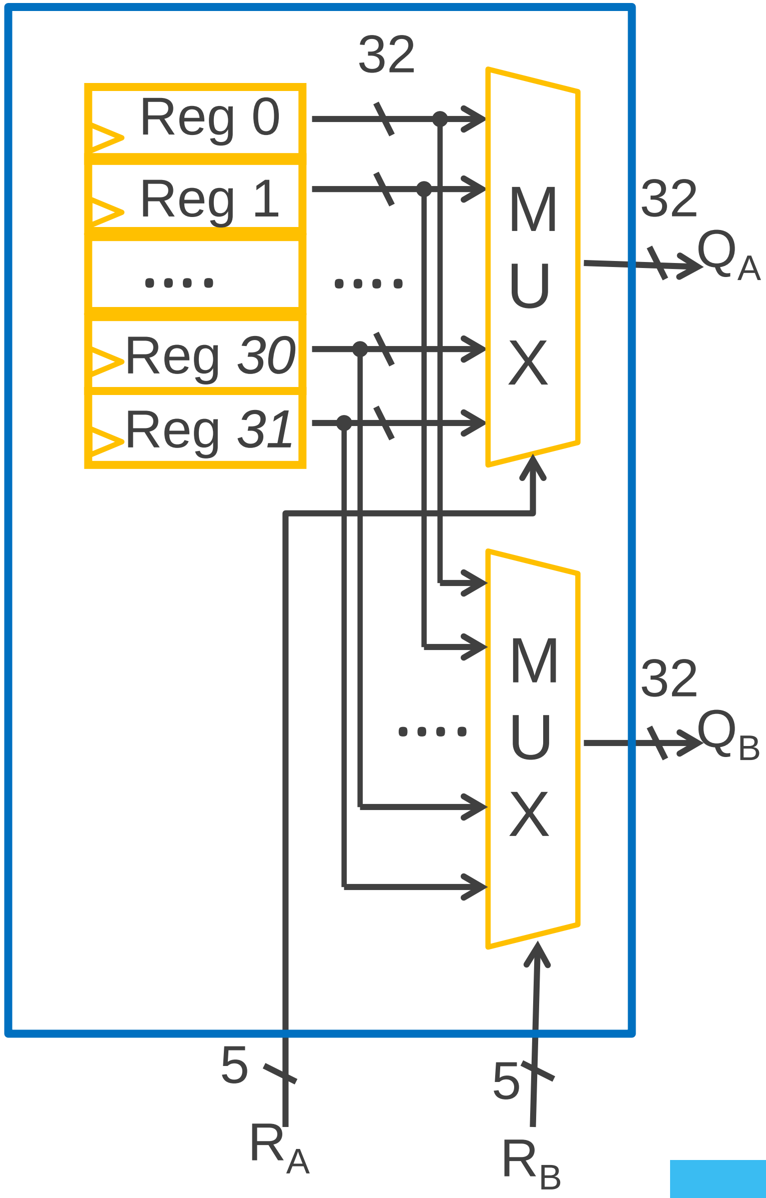


# Register File

## Register File

- N read/write registers
- Indexed by register number




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

## How to read from two registers?

- Need a multiplexor

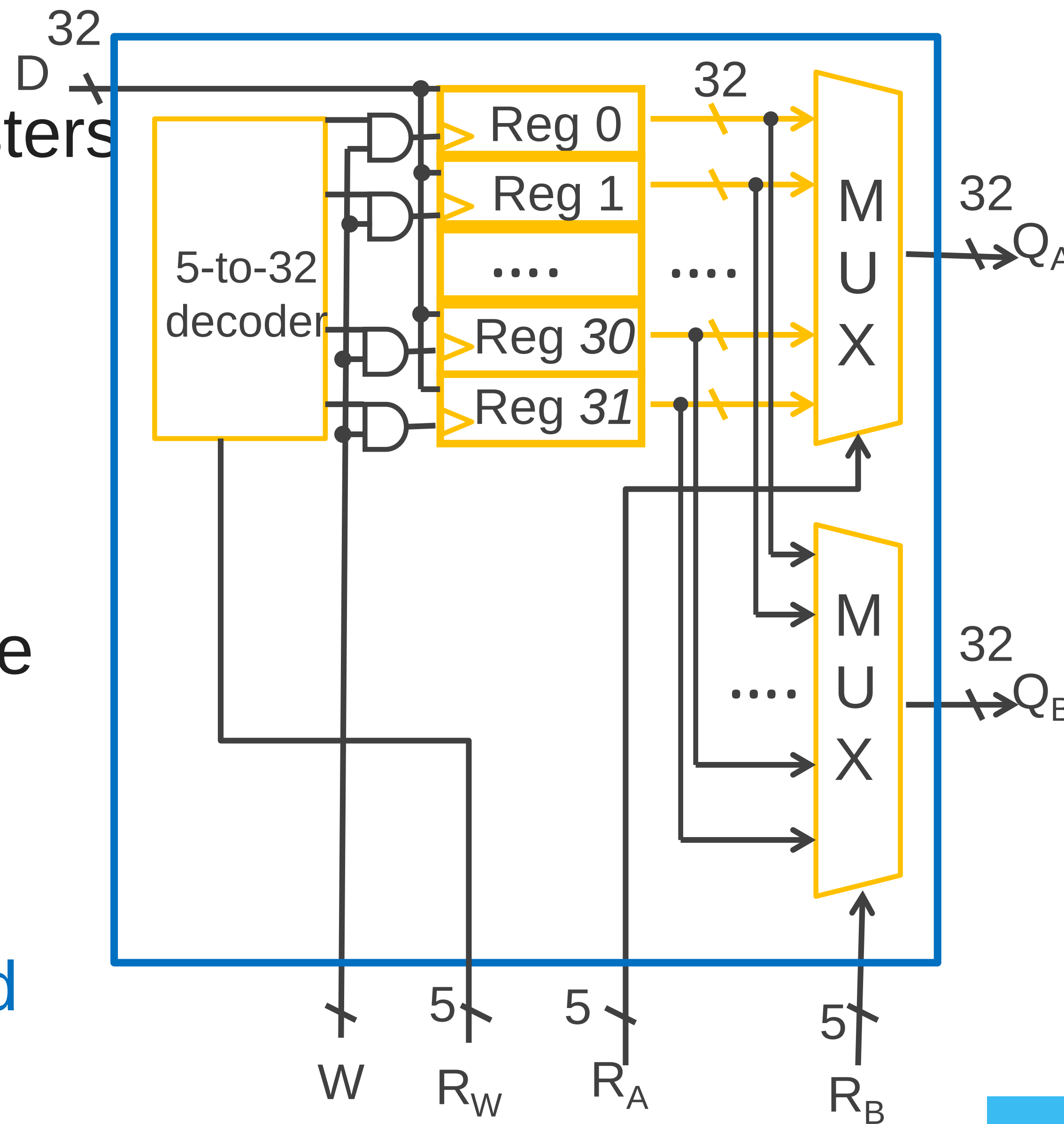
# Register File

## Register File

- N read/write registers
- Indexed by register number

# Implementation:

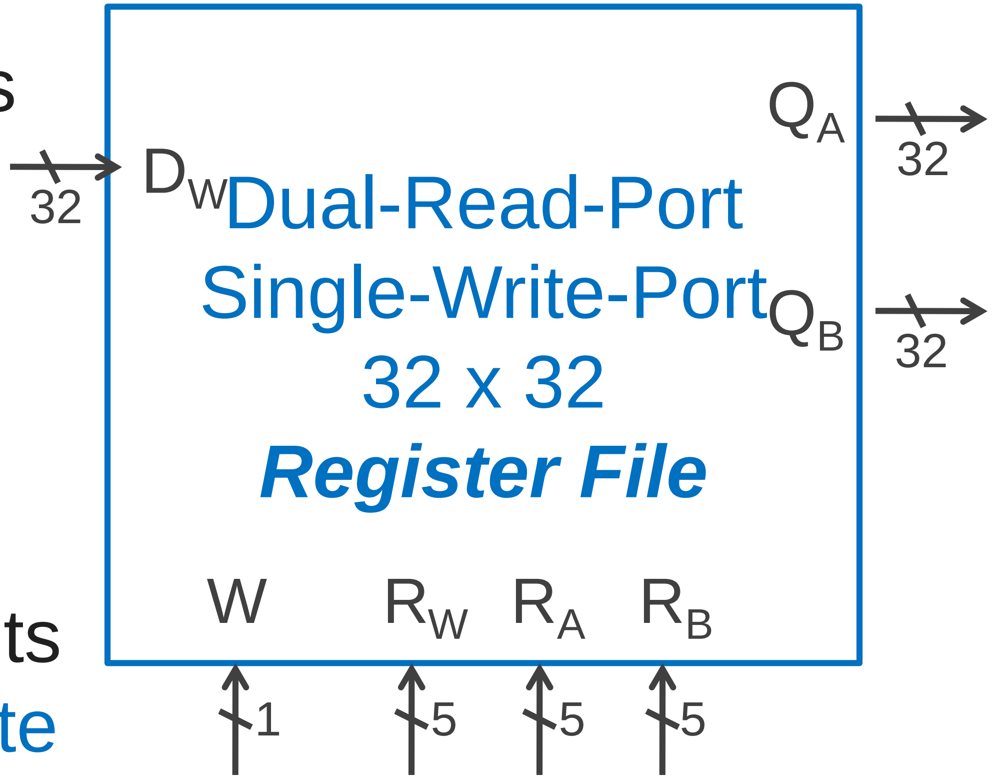
- D flip flops to store bits
- Decoder for each write port
- Mux for each read port



# 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

- N read/write registers
- Indexed by register number

What happens if same register read and written during same clock cycle?

Implementation:

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

# Tradeoffs

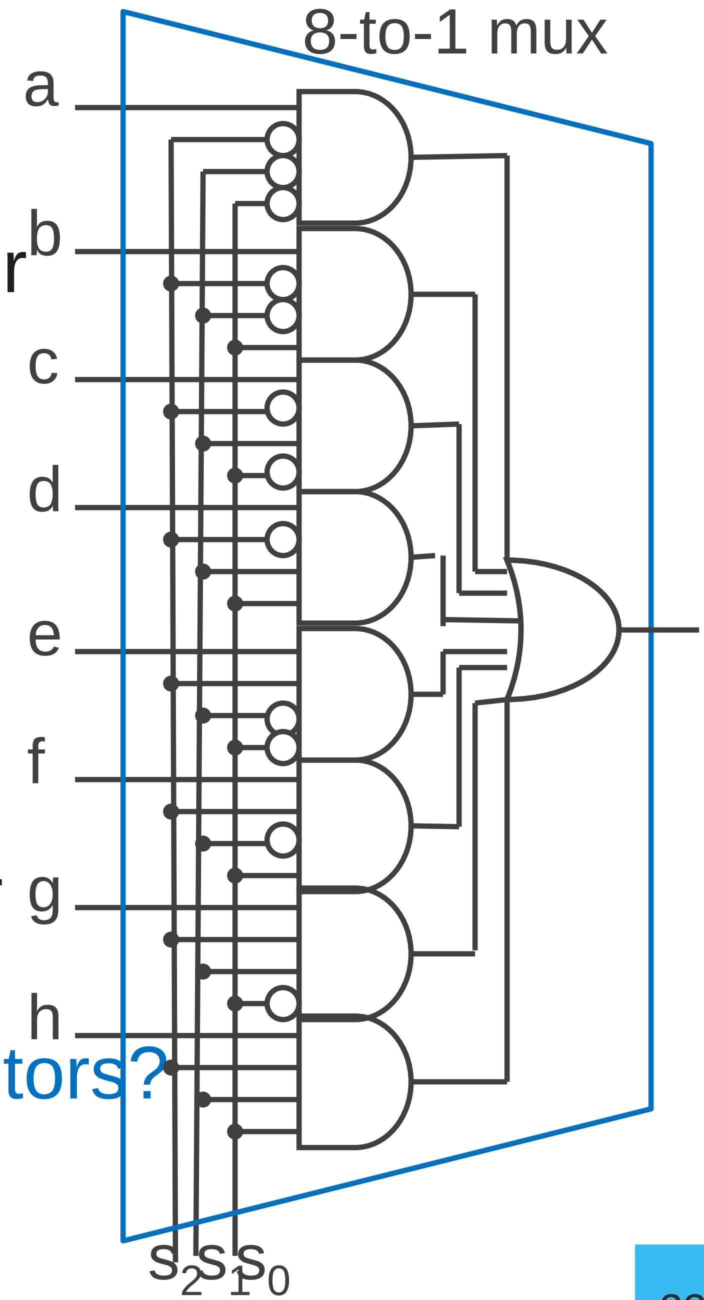
## Register File tradeoffs

- \+ 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?



# Takeway

Register files are very fast storage (only a few gate delays), but does not scale to large memory sizes.

# Goals for today

## Memory

- CPU: Register Files (i.e. Memory w/in the CPU)
- Scaling Memory: Tri-state devices
- Cache: SRAM (Static RAM—random access memory)
- Memory: DRAM (Dynamic RAM)

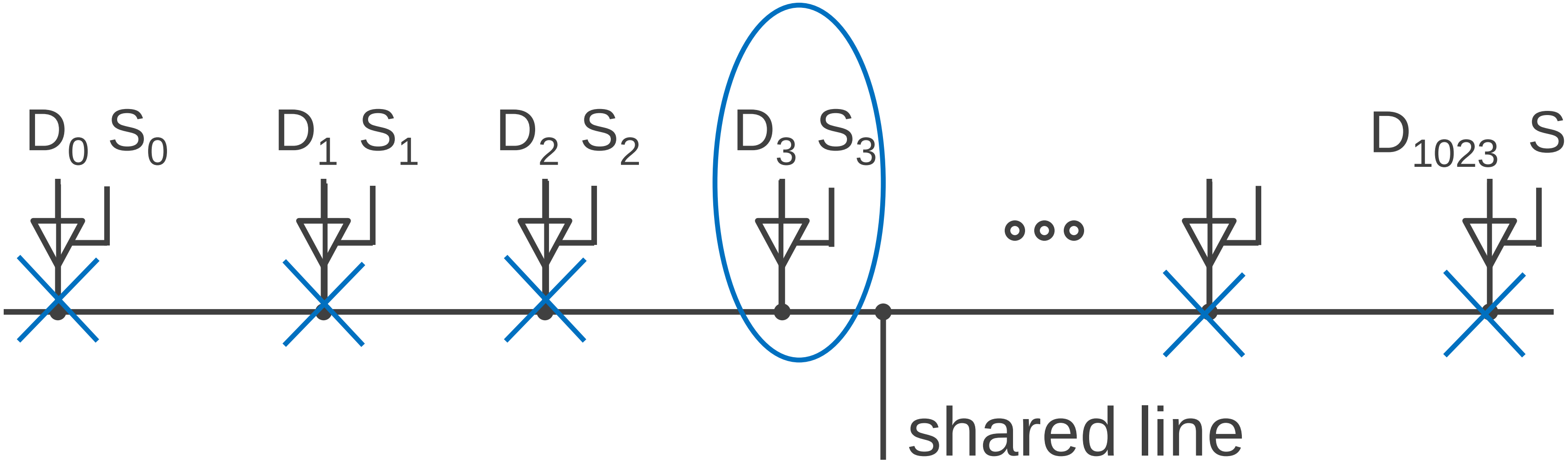
# Next Goal

How do we scale/build larger memories?

# 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

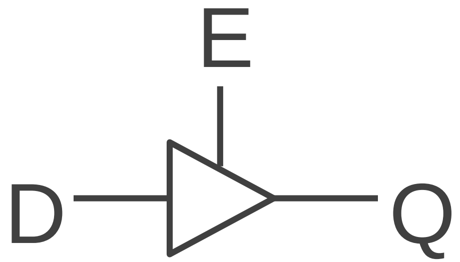



- How do we build such a device?

# Tri-State Devices

## Tri-State Buffers

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

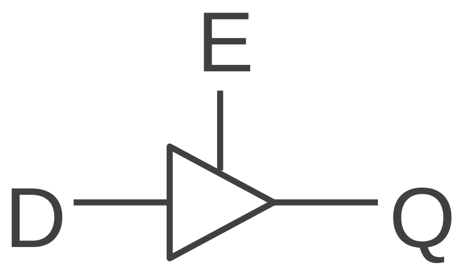



| E | D | Q |
|---|---|---|
| 0 | 0 | z |
| 0 | 1 | z |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

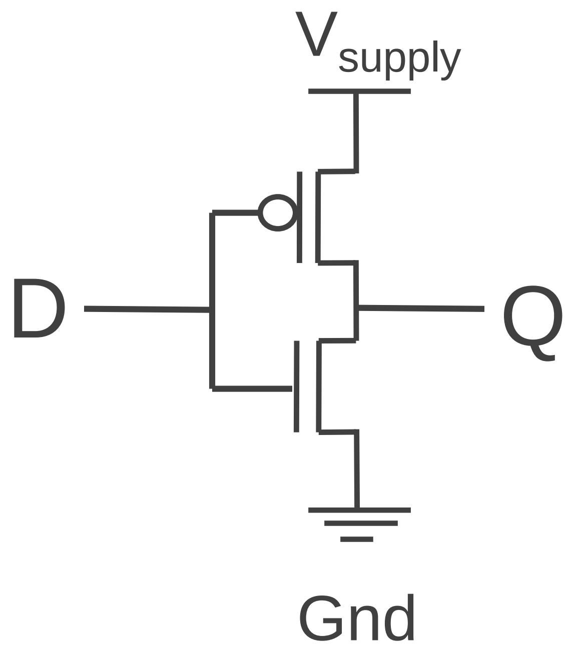
# Tri-State Devices

## Tri-State Buffers

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



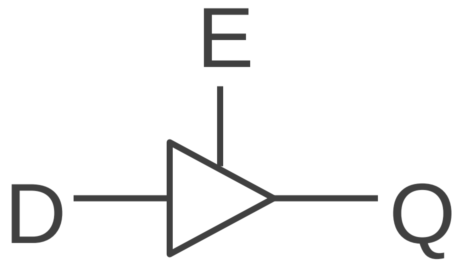

| E | D | Q |
|---|---|---|
| 0 | 0 | z |
| 0 | 1 | z |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



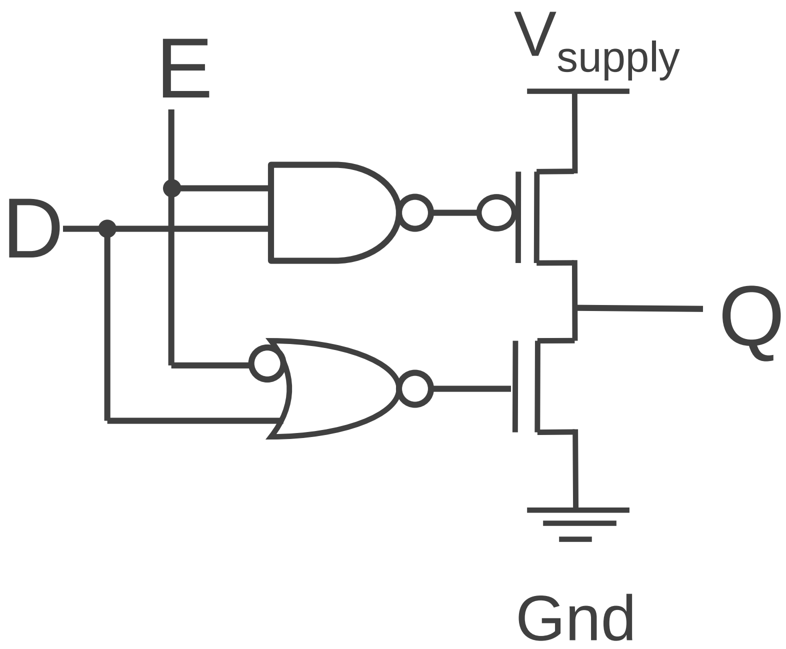
# Tri-State Devices

## Tri-State Buffers

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



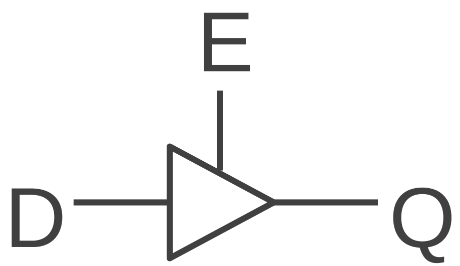

| E | D | Q |
|---|---|---|
| 0 | 0 | z |
| 0 | 1 | z |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



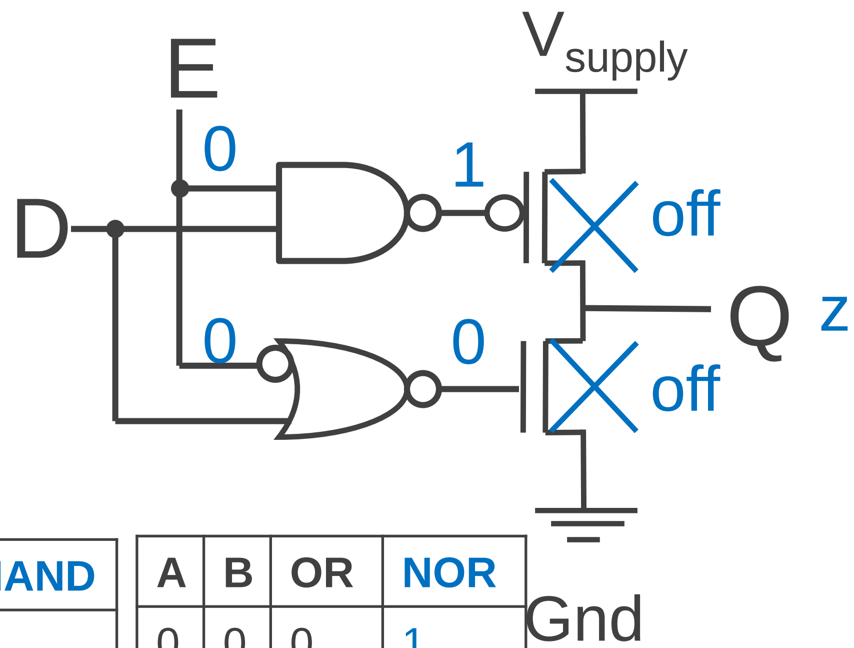
# Tri-State Devices

## 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 |



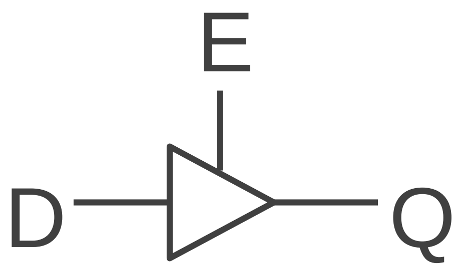

| A | B | AND | NAND |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

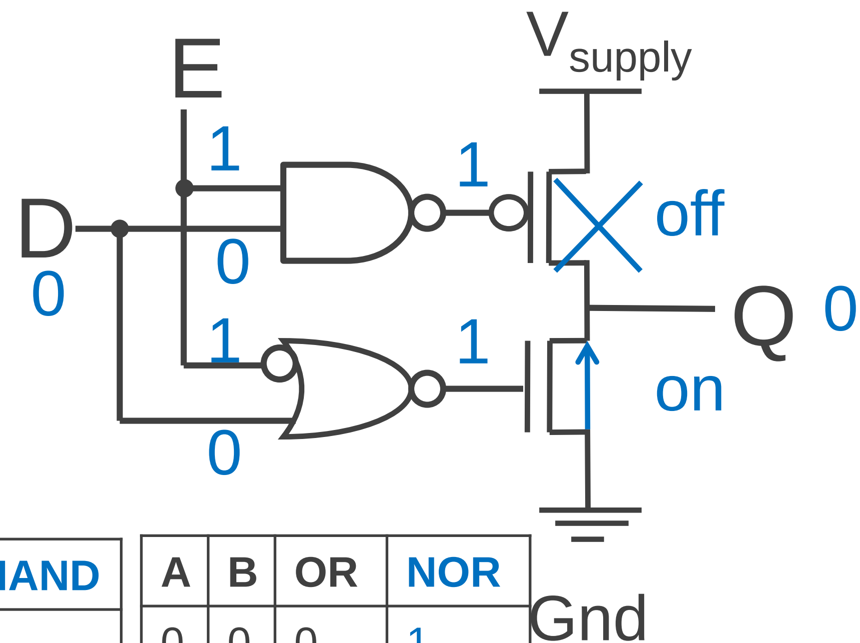
| A | B | OR | NOR |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |

# Tri-State Devices

## 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 |

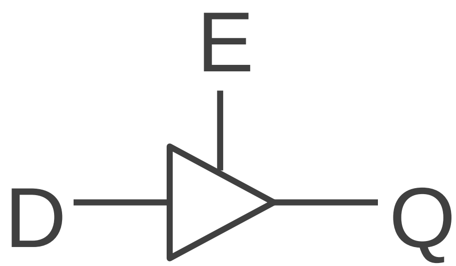
| A | B | AND | NAND |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

| A | B | OR | NOR |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |

# Tri-State Devices

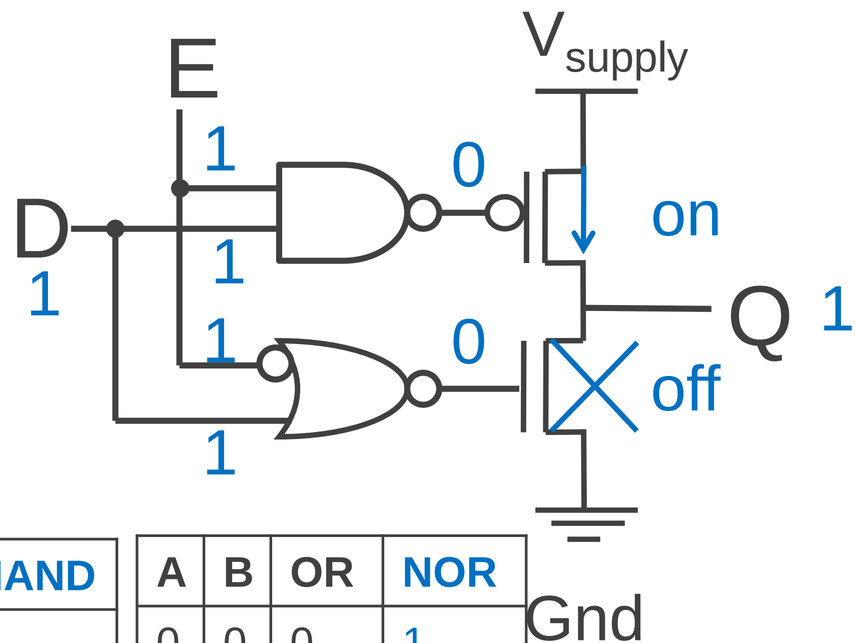
## Tri-State Buffers

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



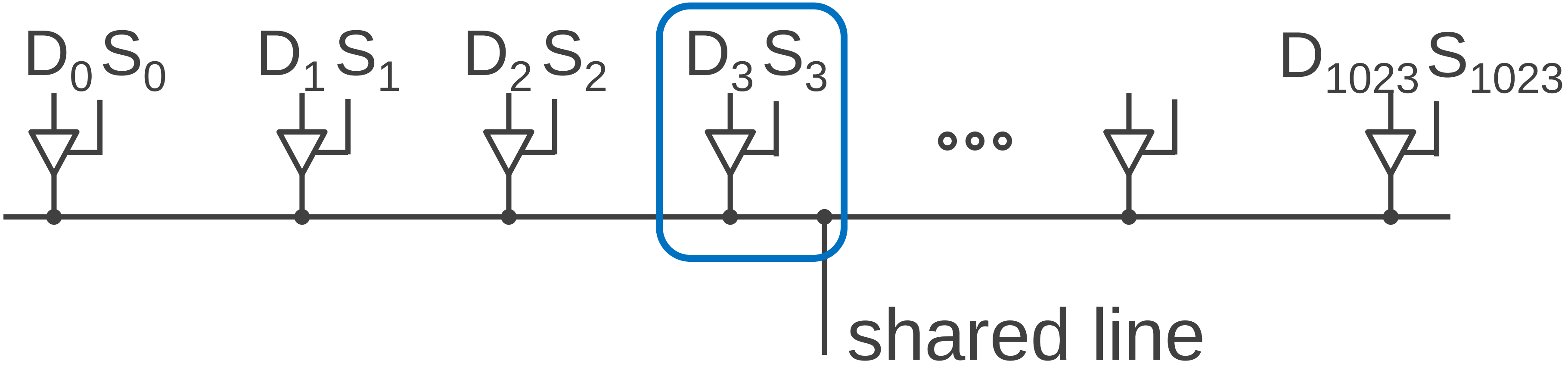

| E | D | Q |
|---|---|---|
| 0 | 0 | z |
| 0 | 1 | z |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| 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 |

# Shared Bus



# Takeway

Register files are very fast storage (only a few gate delays), but does not scale to large memory sizes.

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

# Goals for today

## Memory

- CPU: Register Files (i.e. Memory w/in the CPU)
- Scaling Memory: Tri-state devices
- Cache: SRAM (Static RAM—random access memory)
- Memory: DRAM (Dynamic RAM)

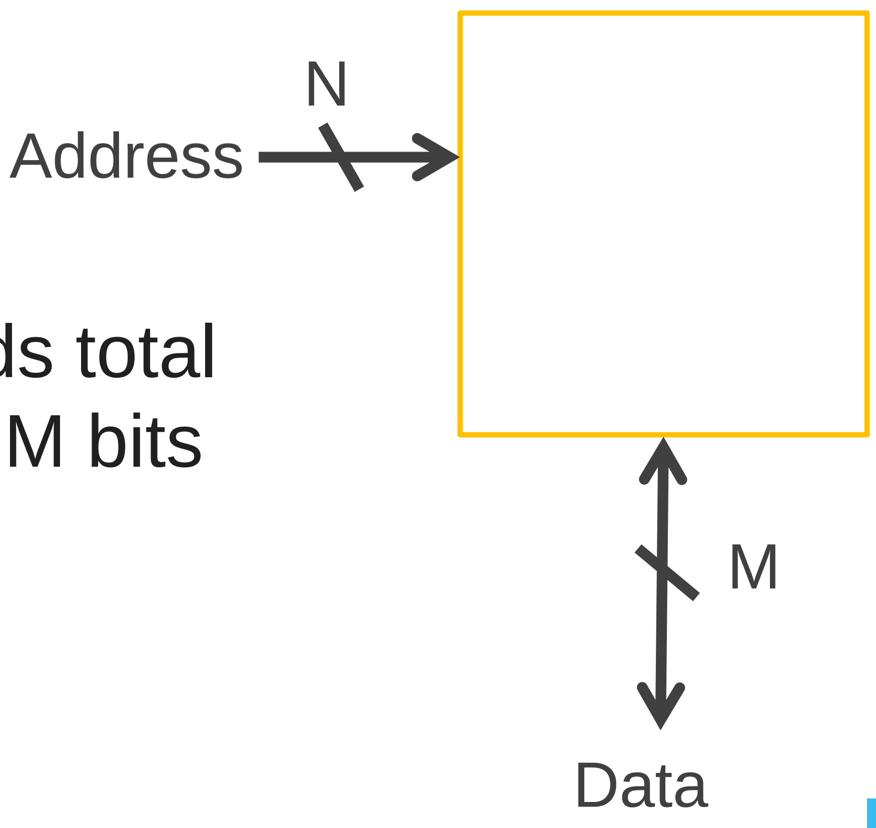
# Next Goal

How do we build large memories?

Use similar designs as Tri-state Buffers to connect multiple registers to output line. Only one register will drive output line.

# Memory

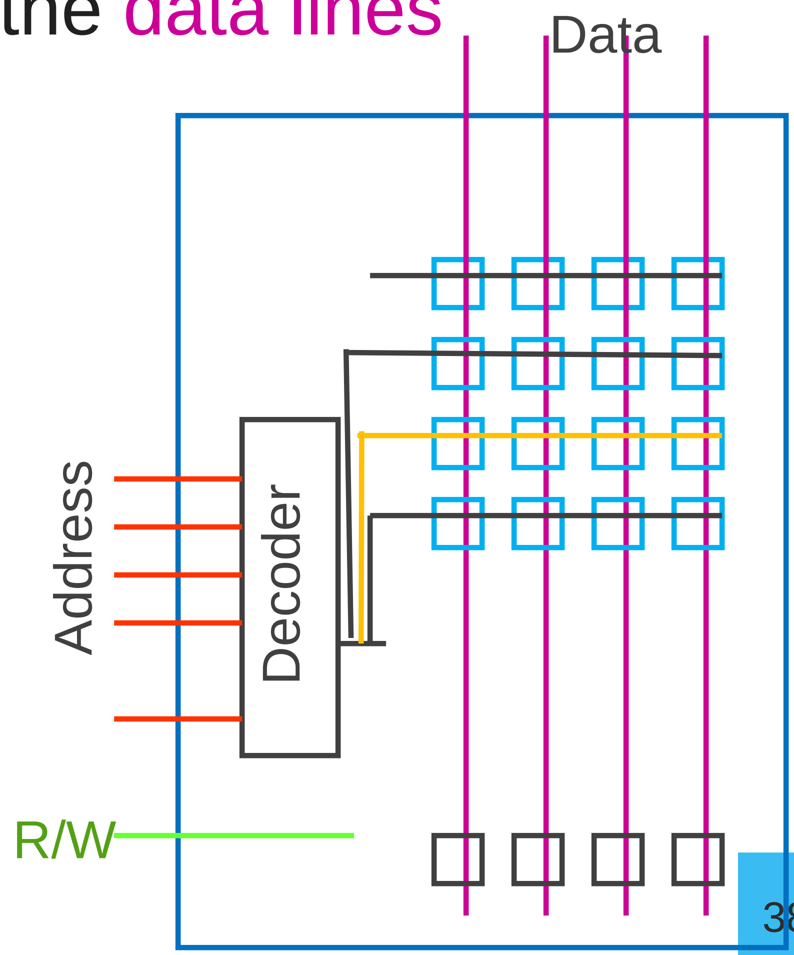
- Storage Cells + bus
- Inputs: Address, Data (for writes)
- Outputs: Data (for reads)
- Also need R/W signal (not shown)




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

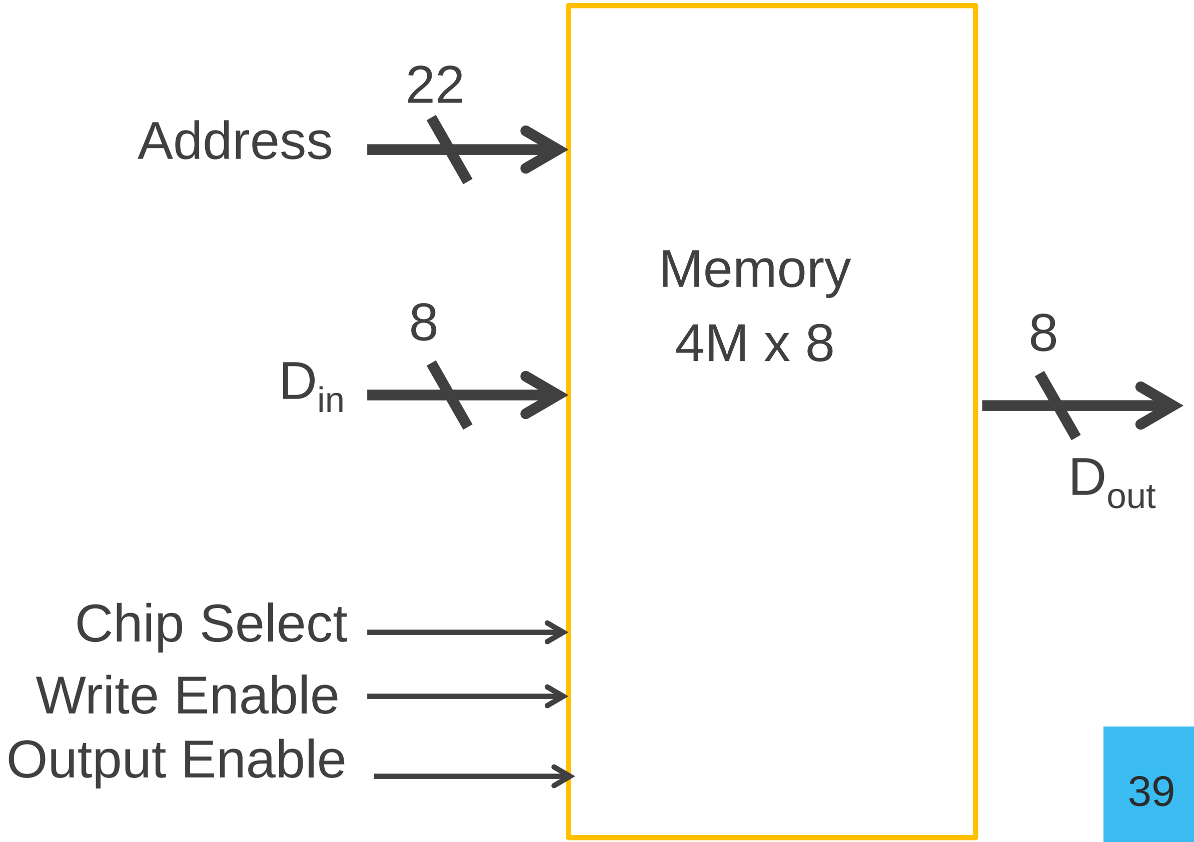
# Memory

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



# Memory

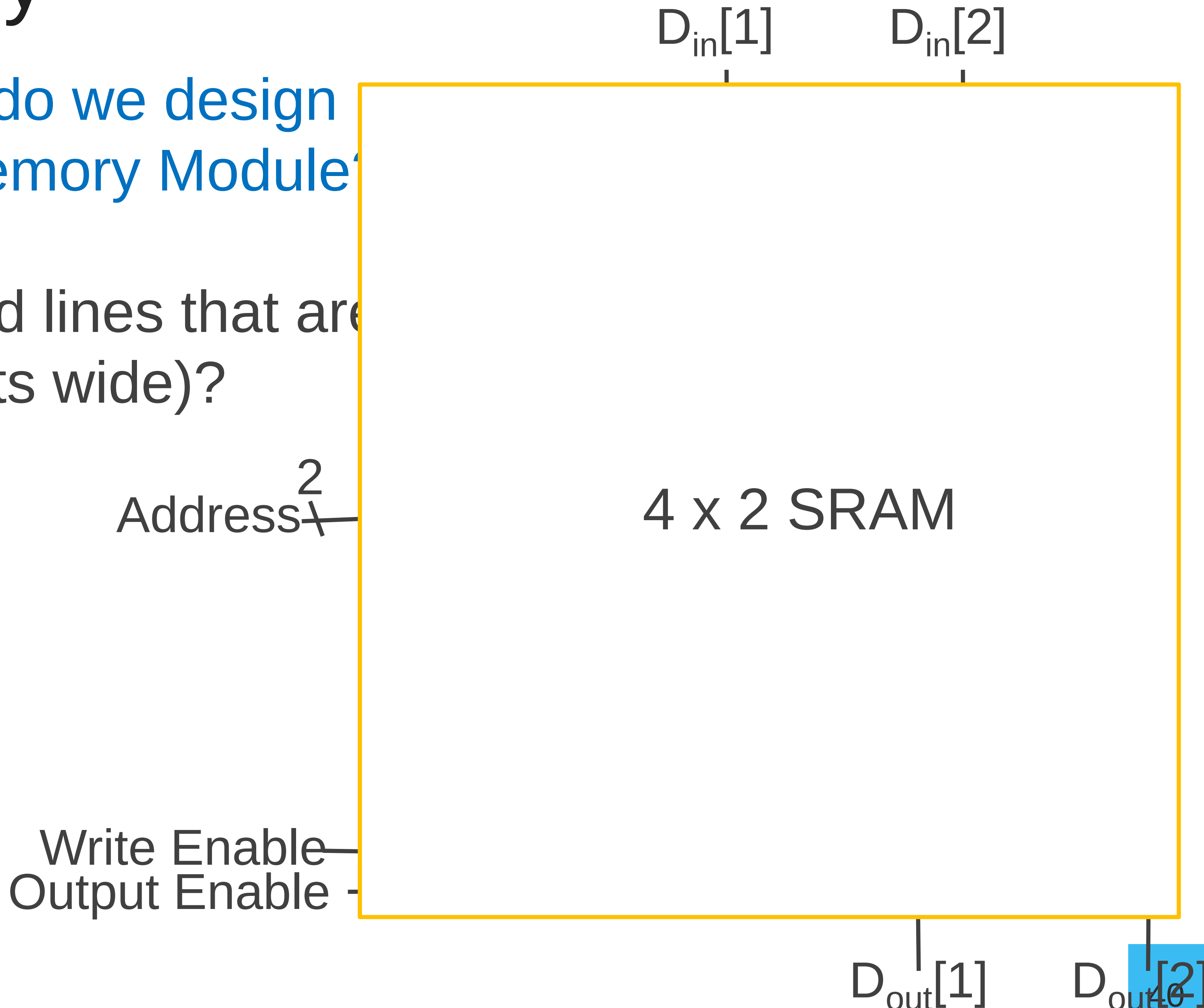
- Storage Cells + bus
- Decoder selects word line
- R/W selector determines access type
- Word line is then coupled to the data lines



# Memory

E.g. How do we design
a 4 x 2 Memory Module?

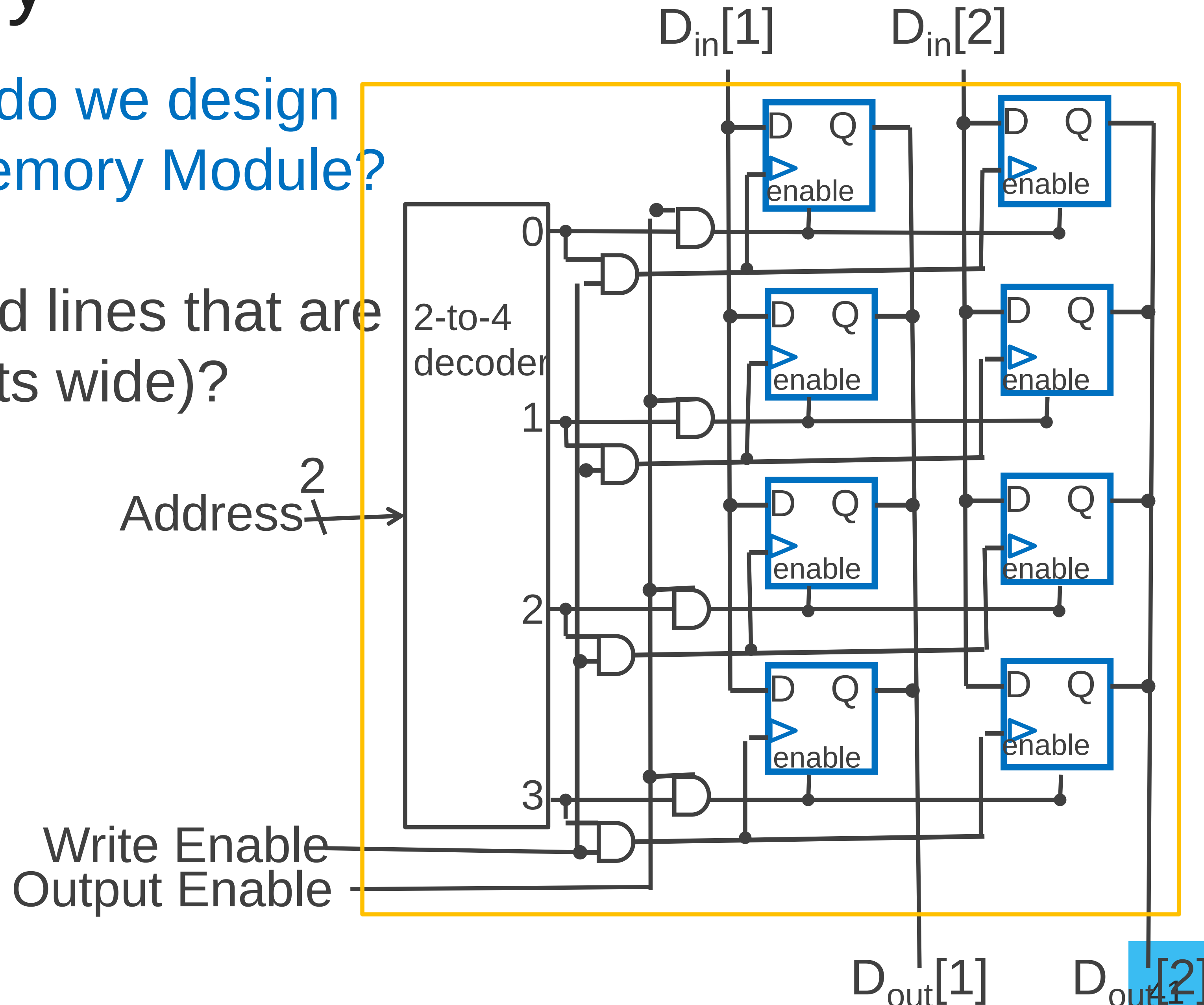
(i.e. 4 word lines that are
each 2 bits wide)?



# Memory

E.g. How do we design a 4 x 2 Memory Module?

(i.e. 4 word lines that are each 2 bits wide)?

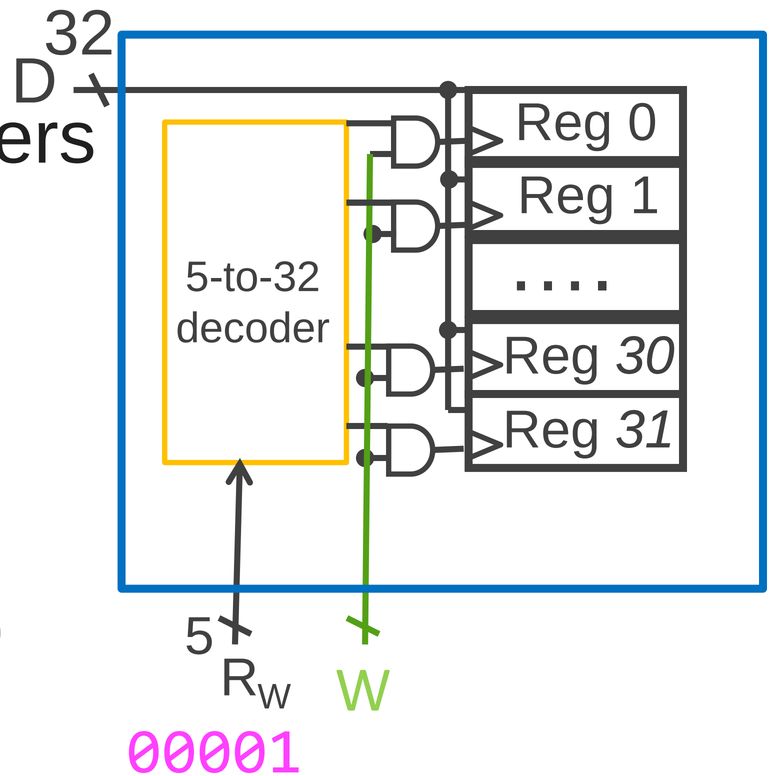


# Register File

## Register File

- N read/write registers
- Indexed by register number

```
addi x1, x0, 10
```




## How to write to ***one*** register in the register file?

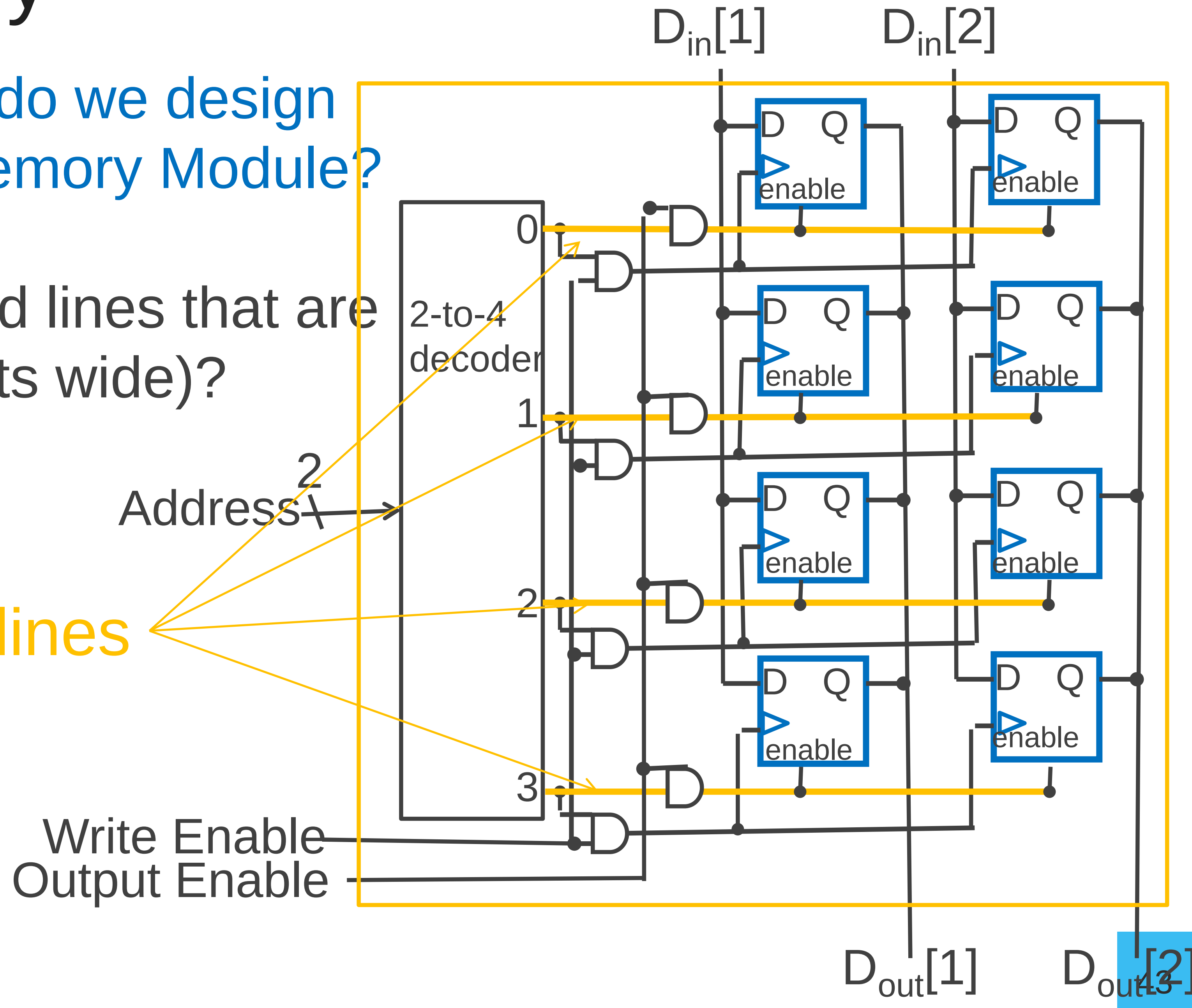
- Need a decoder

# Memory

E.g. How do we design a 4 x 2 Memory Module?

(i.e. 4 word lines that are each 2 bits wide)?

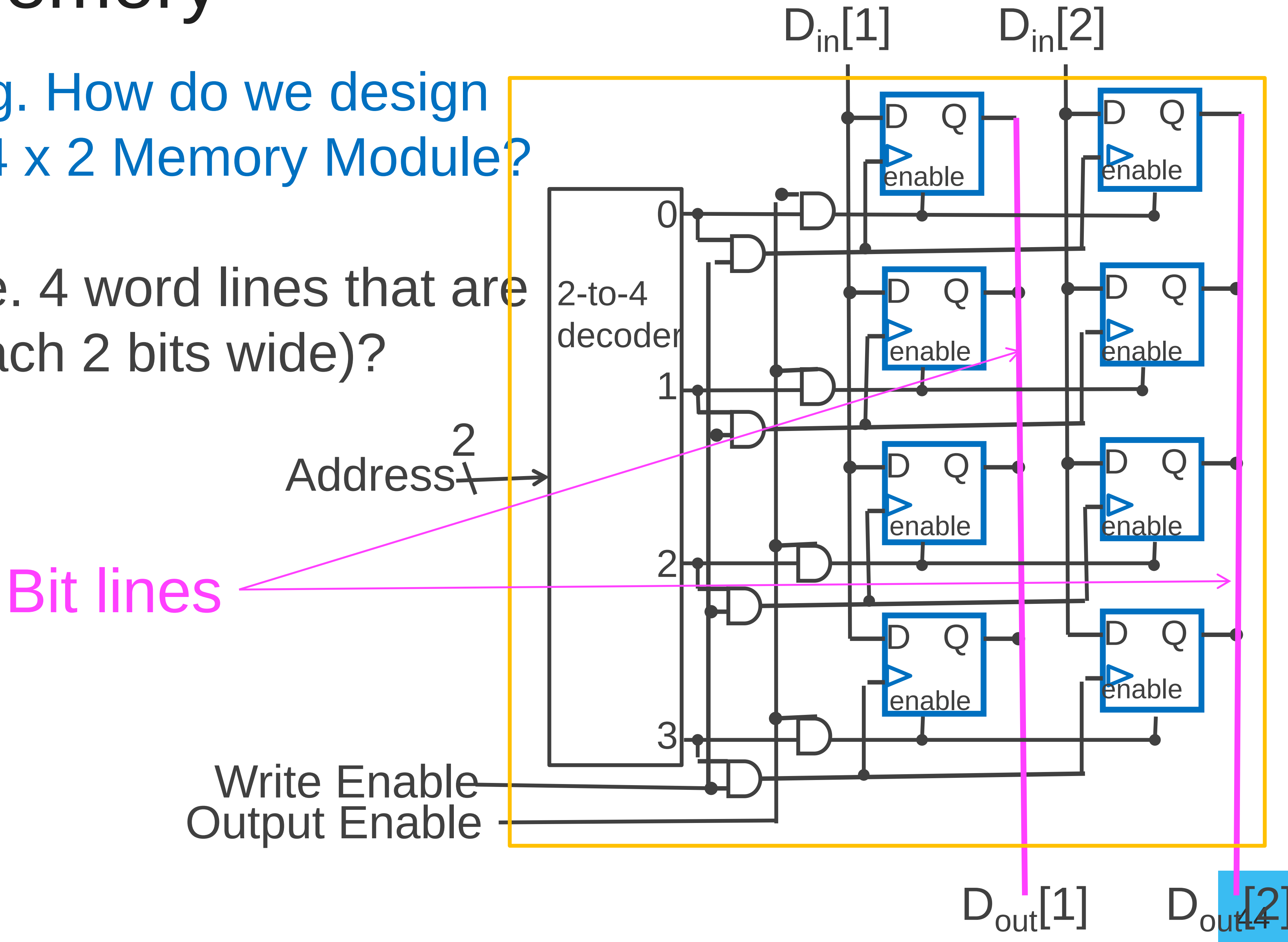
Word lines



# Memory

E.g. How do we design
a 4 x 2 Memory Module?

(i.e. 4 word lines that are
each 2 bits wide)?



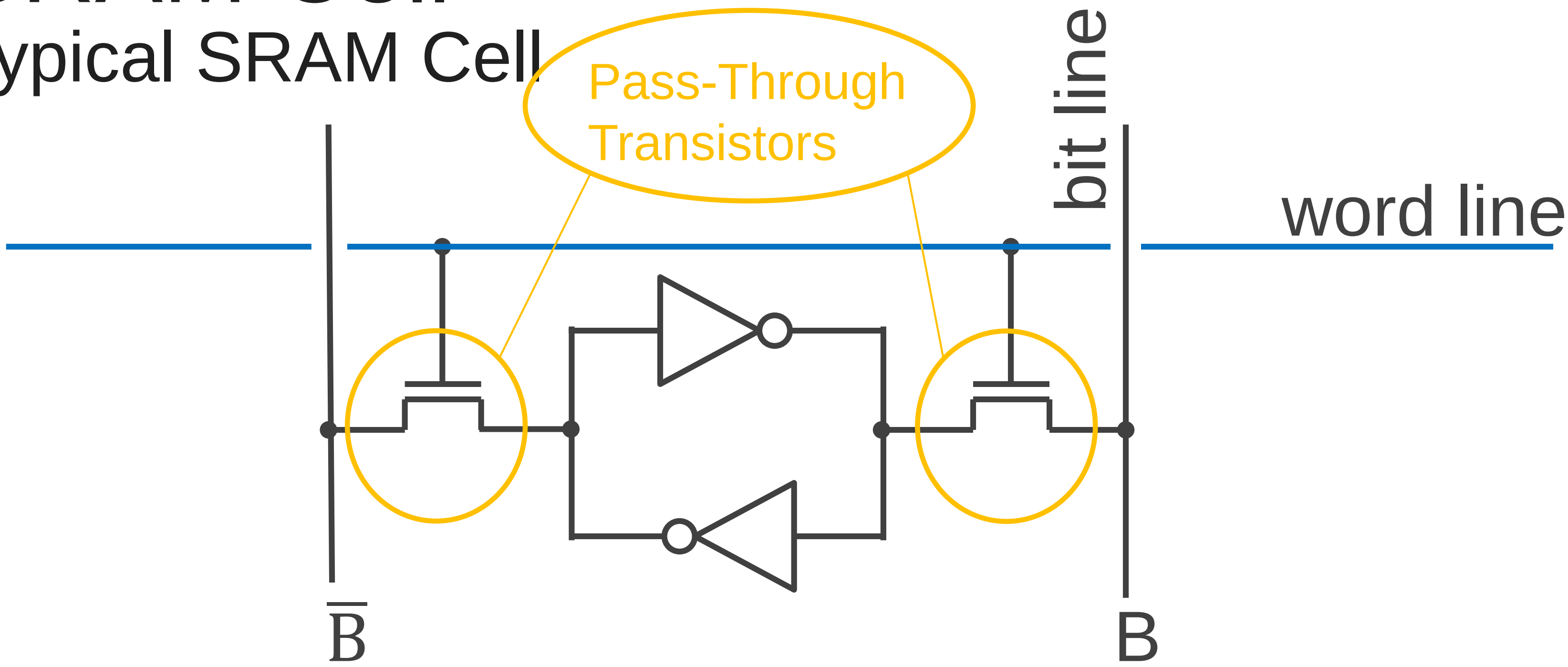
# iClicker Question

What's your familiarity with memory (SRAM, DRAM)?

A. I've never heard of any of this.
B. I've heard the words SRAM and DRAM, but I have no idea what they are.
C. I know that DRAM means main memory.
D. I know the difference between SRAM and DRAM and where they are used in a computer system.

# SRAM Cell

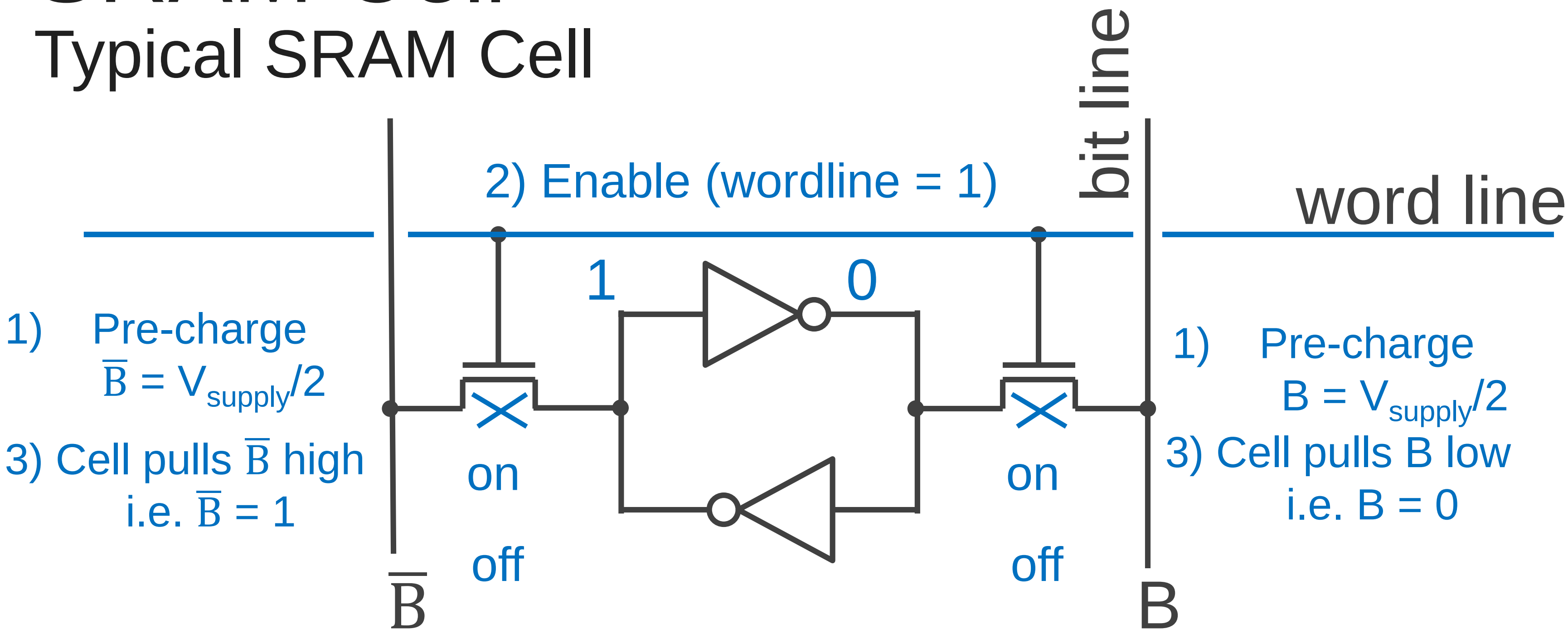
## Typical SRAM Cell




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

# SRAM Cell

## Typical SRAM Cell



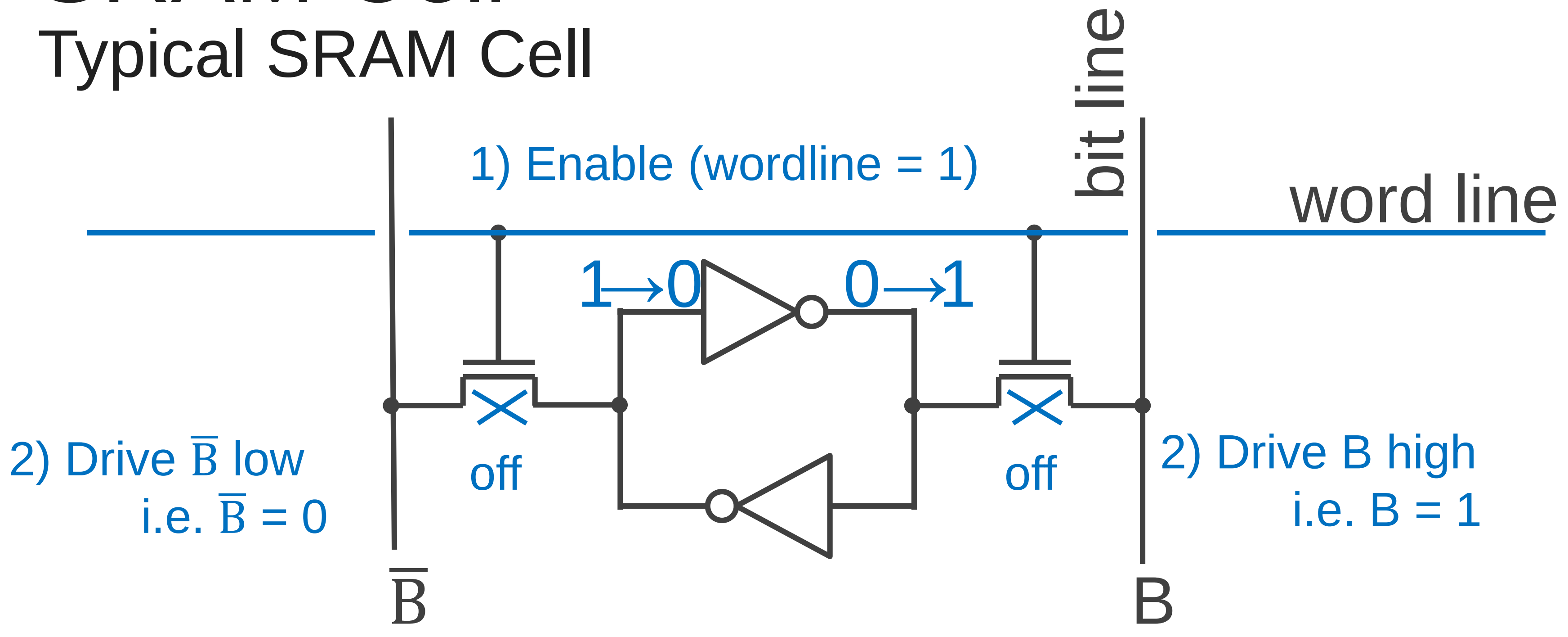

Each cell stores one bit, and requires 4 – 8 transistors (6 is typical)

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




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

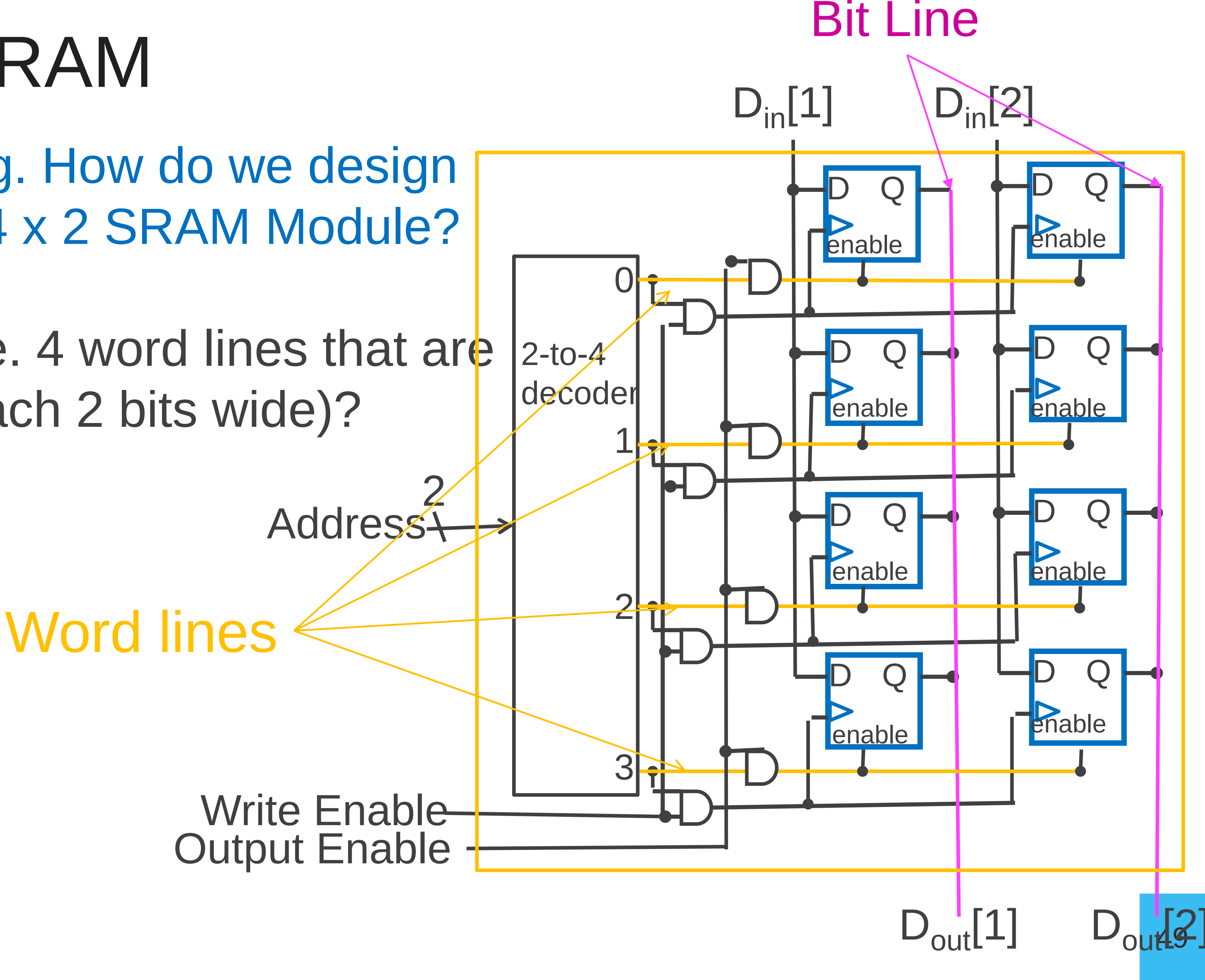
Write:

- pull word line high
- drive

# SRAM

E.g. How do we design a 4 x 2 SRAM Module?

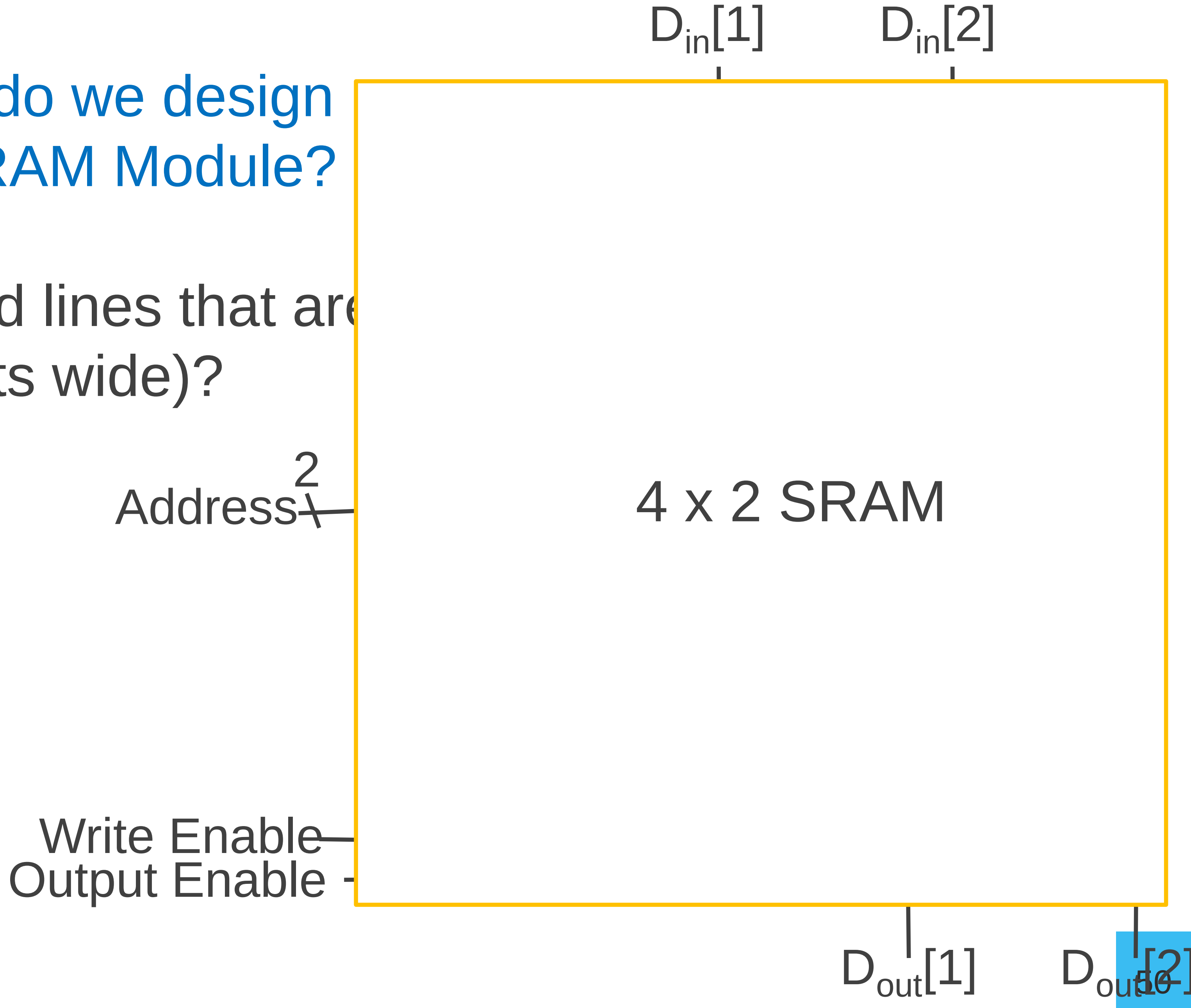
(i.e. 4 word lines that are each 2 bits wide)?



# SRAM

E.g. How do we design a 4 x 2 SRAM Module?

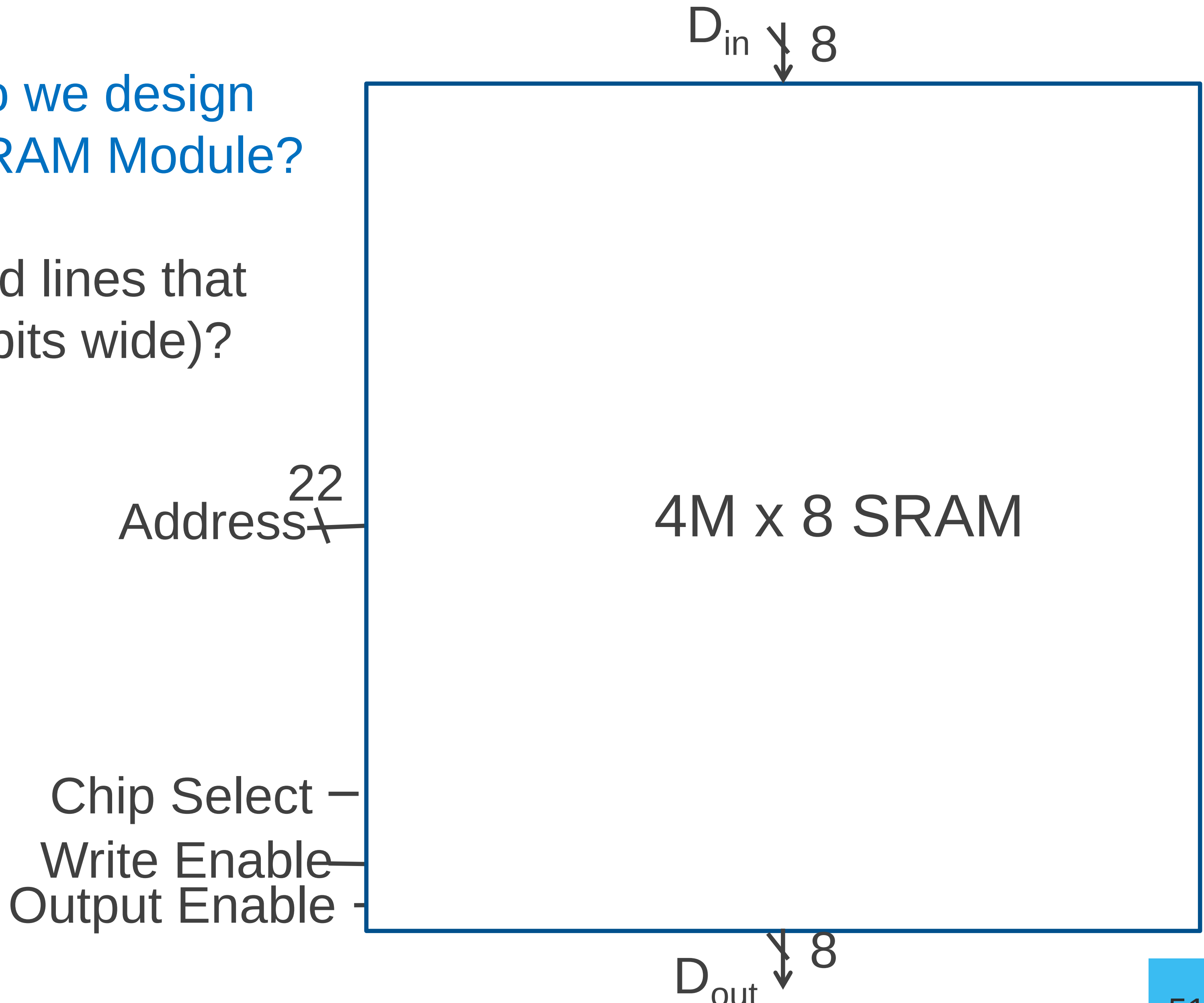
(i.e. 4 word lines that are each 2 bits wide)?



# SRAM

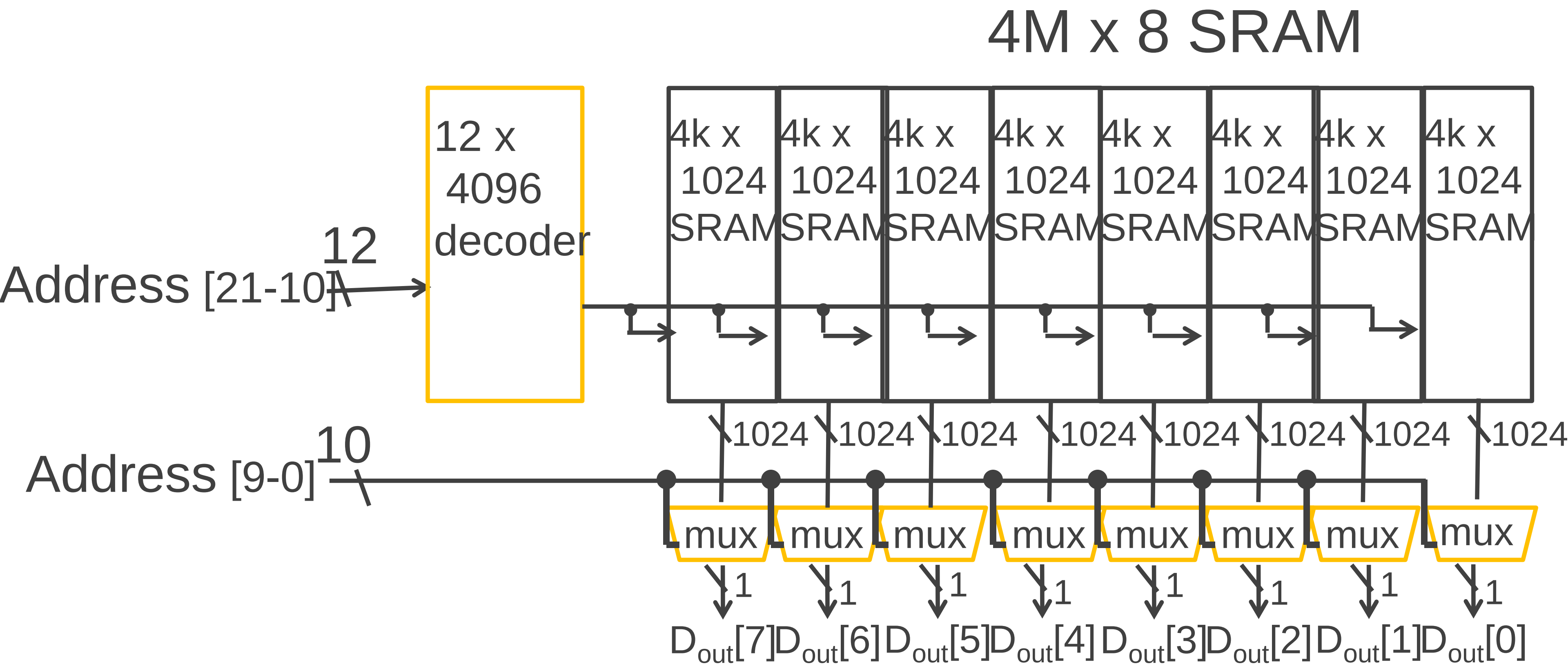
E.g. How do we design a ***4M x 8*** SRAM Module?

(i.e. 4M word lines that are each 8 bits wide)?



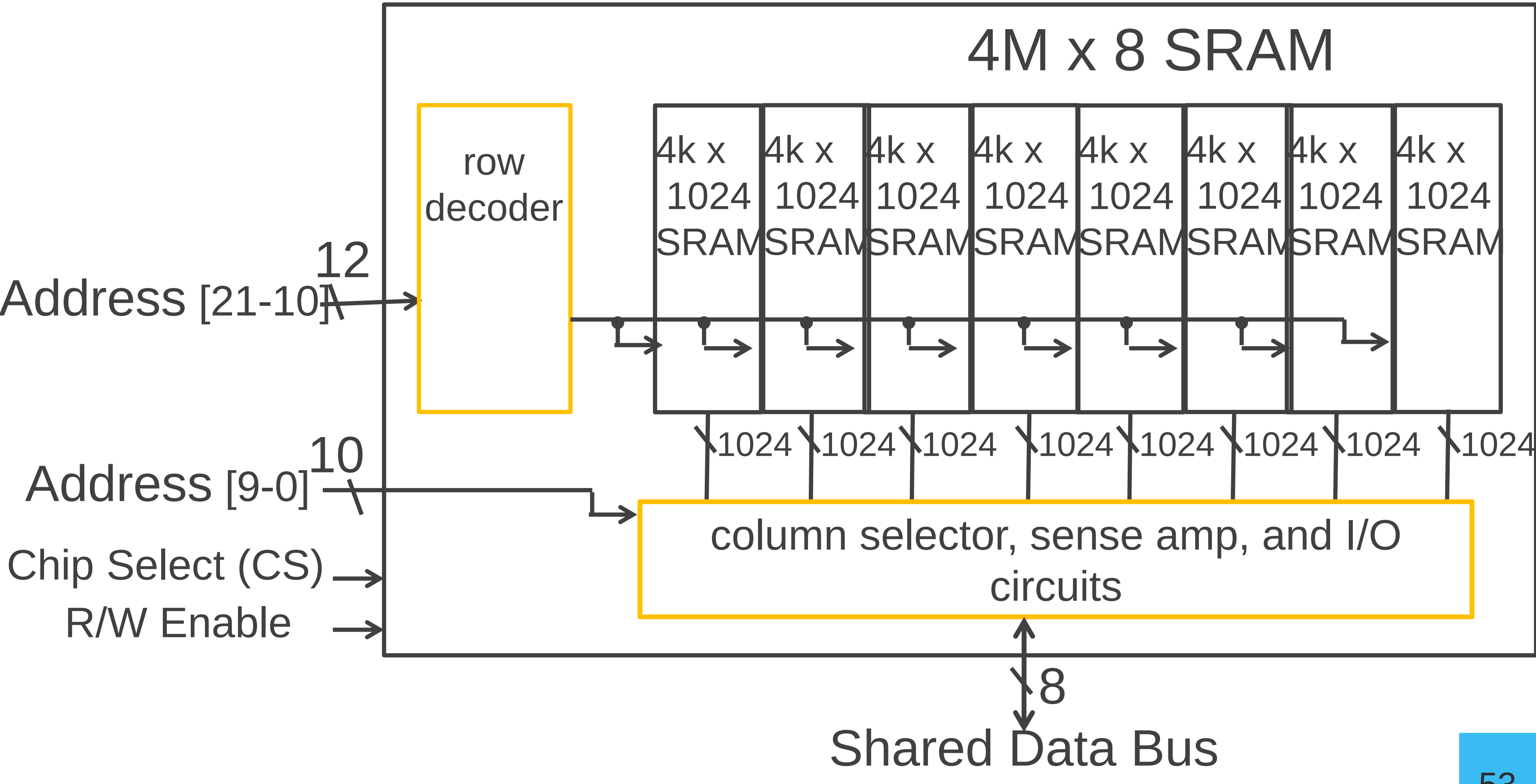
# SRAM

E.g. How do we design
a ***4M x 8*** SRAM Module?

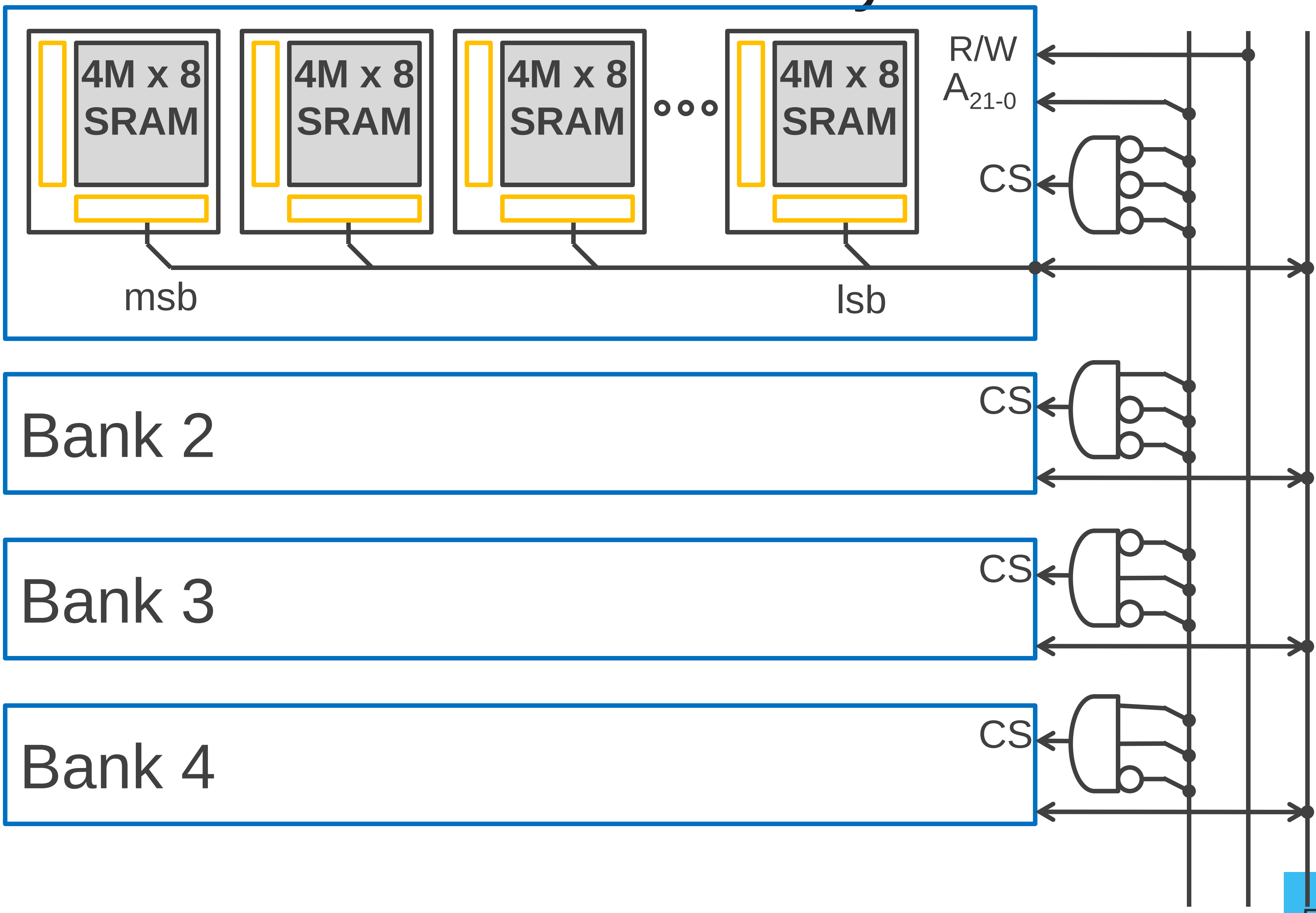


# SRAM

E.g. How do we design
a ***4M x 8*** SRAM Module?



# SRAM Modules and Arrays



# SRAM Summary

## SRAM

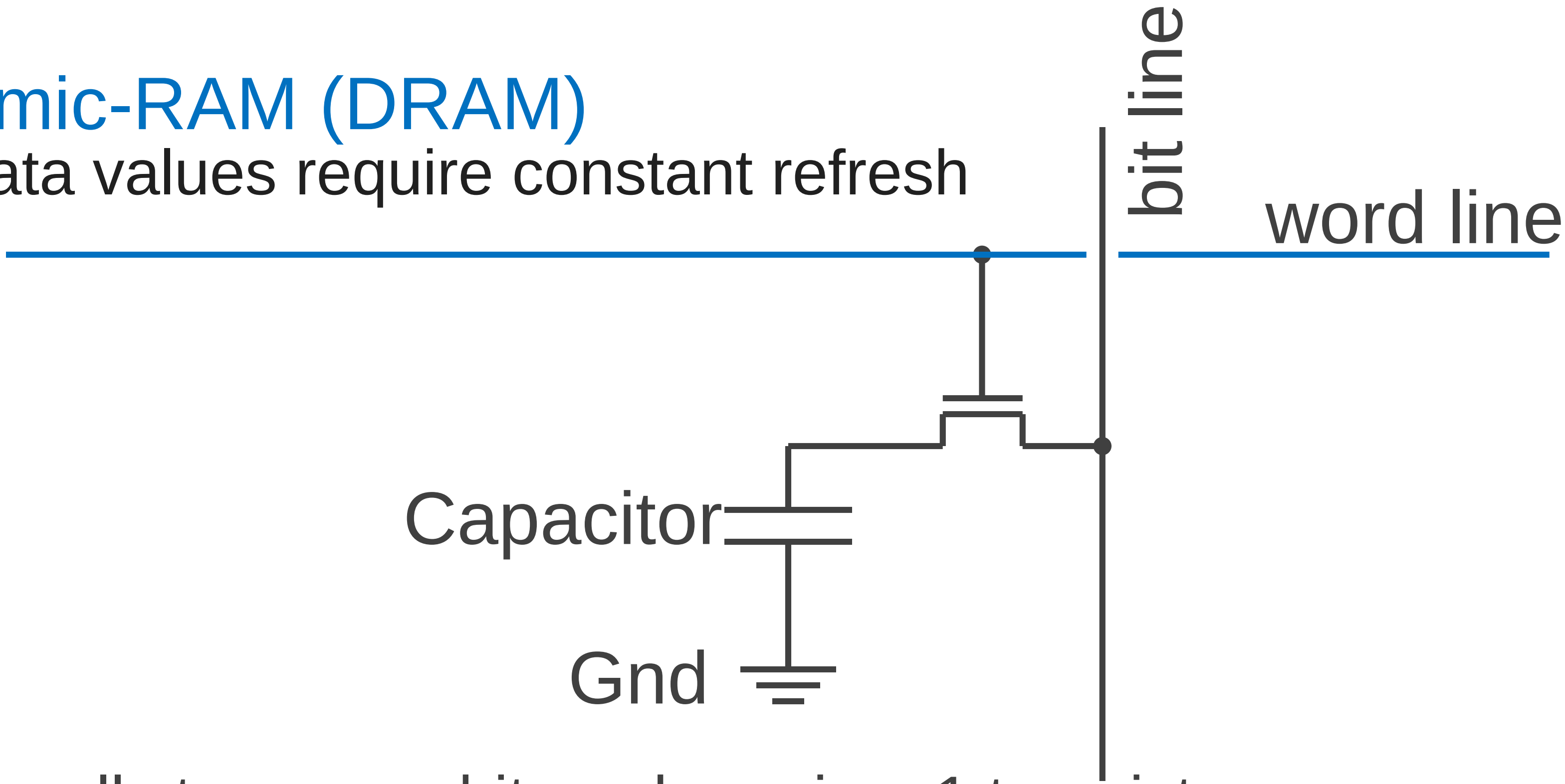
- A few transistors (~6) per cell
- Used for working memory (caches)

- But for even higher density…

# Dynamic RAM: DRAM

## Dynamic-RAM (DRAM)

- Data values require constant refresh

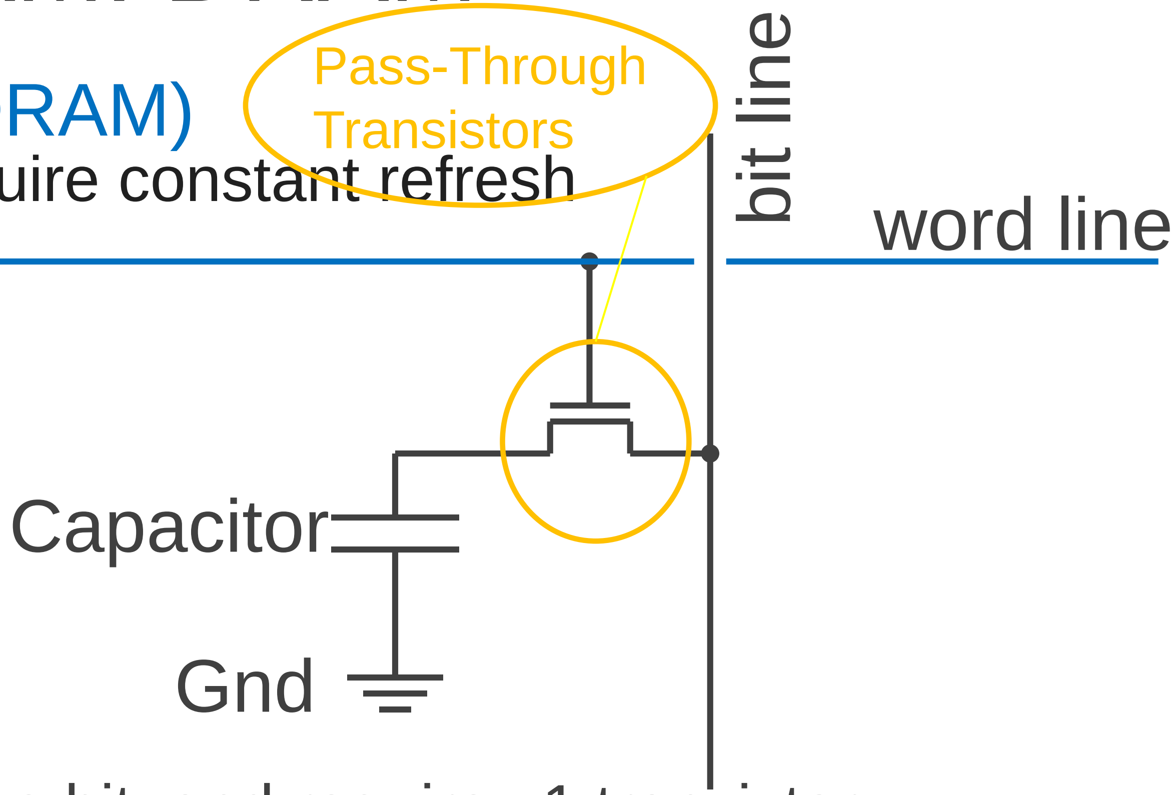



Each cell stores one bit, and requires 1 transistors

# Dynamic RAM: DRAM

## Dynamic-RAM (DRAM)

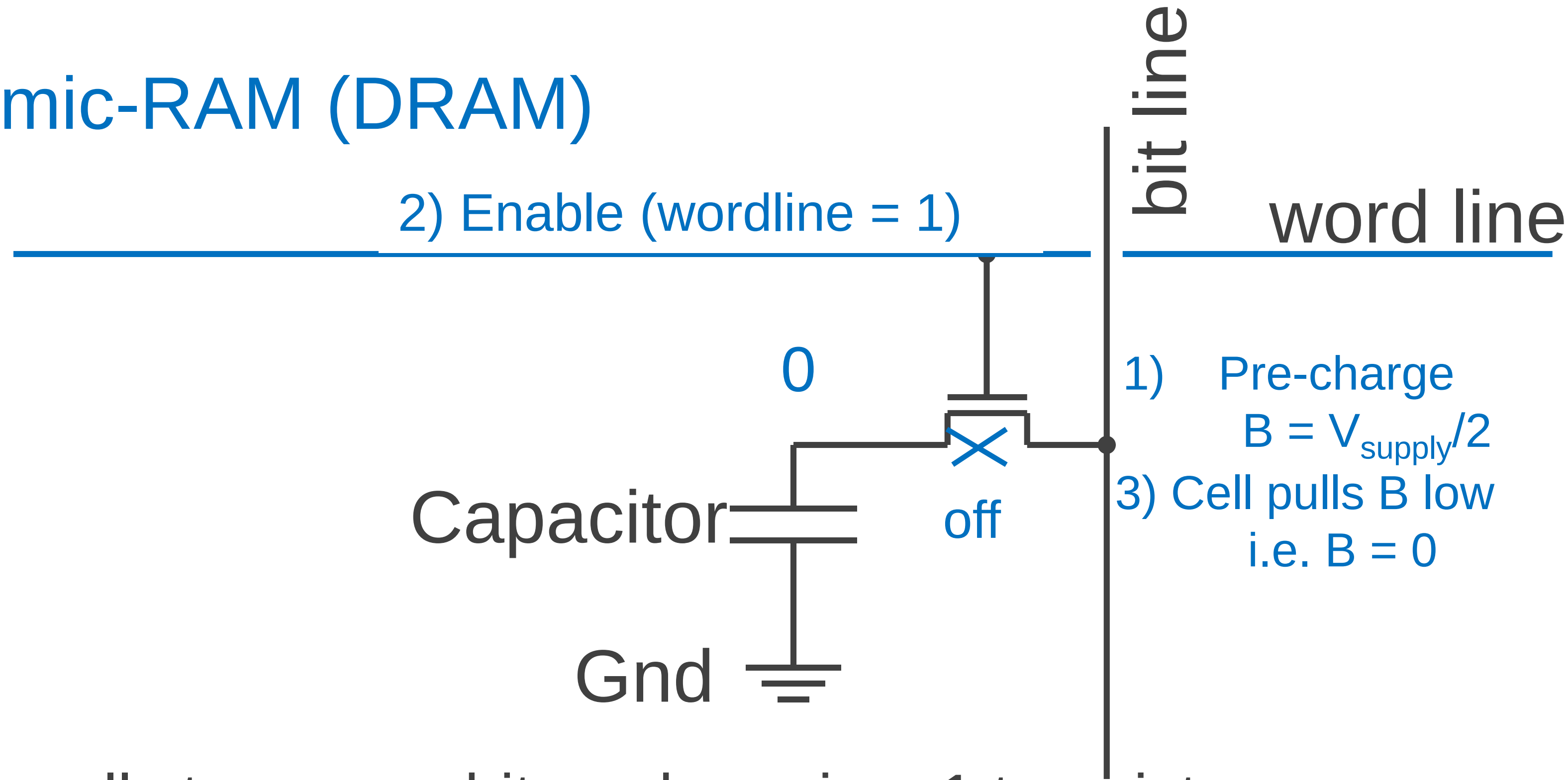
- Data values require constant refresh




Each cell stores one bit, and requires 1 transistors

# Dynamic RAM: DRAM

## Dynamic-RAM (DRAM)



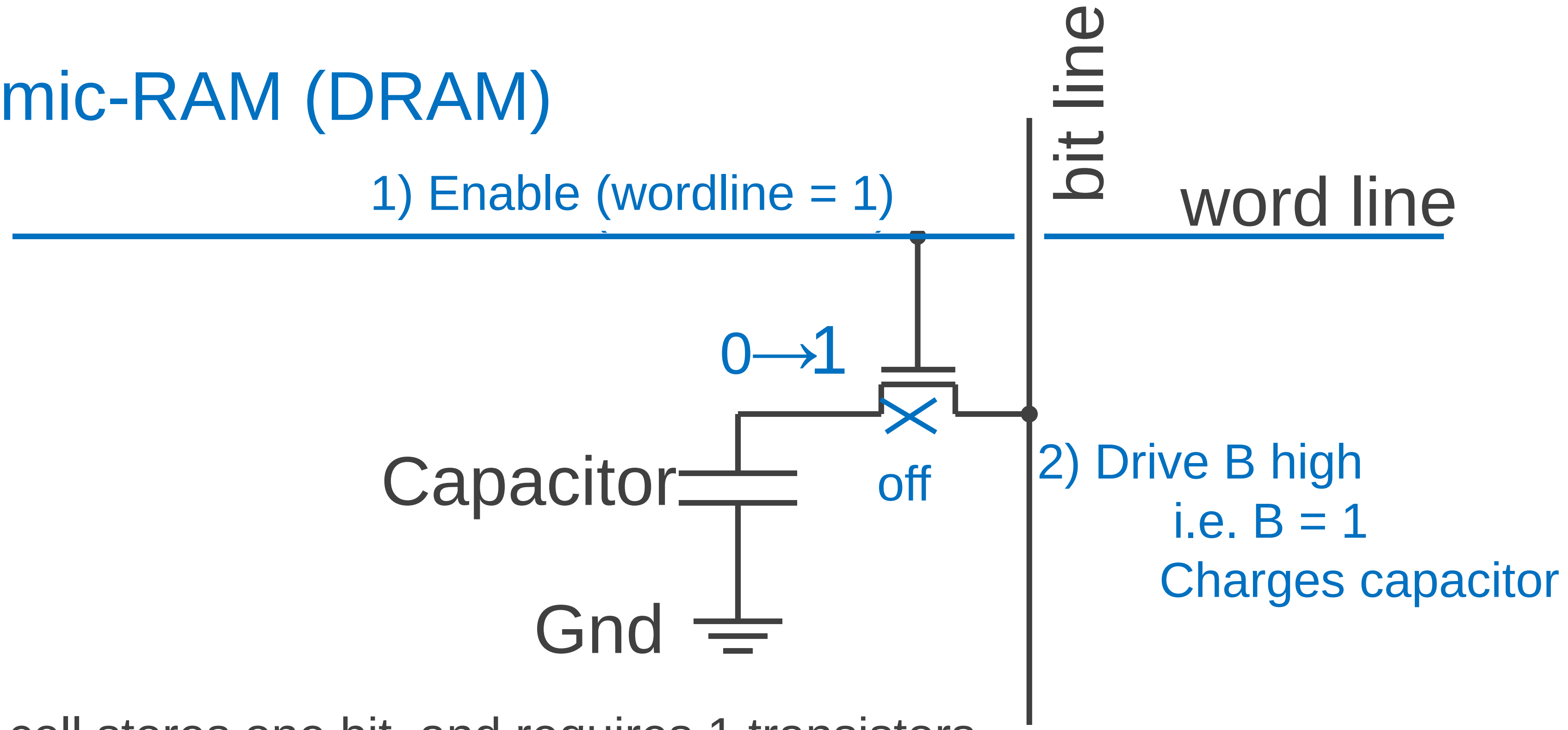

Each cell stores one bit, and requires 1 transistors

Read:

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

# Dynamic RAM: DRAM

## Dynamic-RAM (DRAM)




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

Write:

- pull word line high
- drive B charges capacitor

# DRAM vs. SRAM

Single transistor vs. many gates

- Denser, cheaper ($30/1GB vs. $30/2MB)
- But more complicated, and has analog sensing

Also needs refresh

- Read and write back…
- …every few milliseconds
- Organized in 2D grid, so can do rows at a time
- Chip can do refresh internally

Hence… slower and energy inefficient

# 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

# Summary

We now have enough building blocks to build machines that can perform non-trivial computational tasks

Register File: Tens of words of working memory
SRAM: Millions of words of working memory
DRAM: Billions of words of working memory
NVRAM: long term storage
(usb fob, solid state disks, BIOS, …)

Next time we will build a simple processor!