Memory

Prof. Kavita Bala and Prof. Hakim Weatherspoon

CS 3410, Spring 2014

Computer Science

Cornell University

See P&H Appendix B.8 (register files) and B.9

Administrivia

Make sure to go to ***your*** Lab Section this week

Completed Lab1 due ***before*** winter break, Friday, Feb 14th

Note, a **Design Document** is due when you submit Lab1 final circuit

Work **alone**

Save your work!

- ***Save often***. Verify file is non-zero. Periodically save to Dropbox, email.
- Beware of MacOSX 10.5 (leopard) and 10.6 (snow-leopard)

Homework1 is out

Due a week before prelim1, Monday, February 24th

Work on problems incrementally, as we cover them in lecture (i.e. part 1

Office Hours for help

Work **alone**

Work alone, **BUT** use your resources

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

Administrivia

Check online syllabus/schedule

- http://www.cs.cornell.edu/Courses/CS3410/2014sp/schedule.html

Slides and Reading for lectures

Office Hours

Homework and Programming Assignments

Prelims (in evenings):

  - Tuesday, March 4th
  - Thursday, May 1th

Schedule is subject to change

Collaboration, Late, Re-grading Policies

“Black Board” Collaboration Policy

- Can discuss approach together on a “black board”
- Leave and write up solution independently
- Do not copy solutions

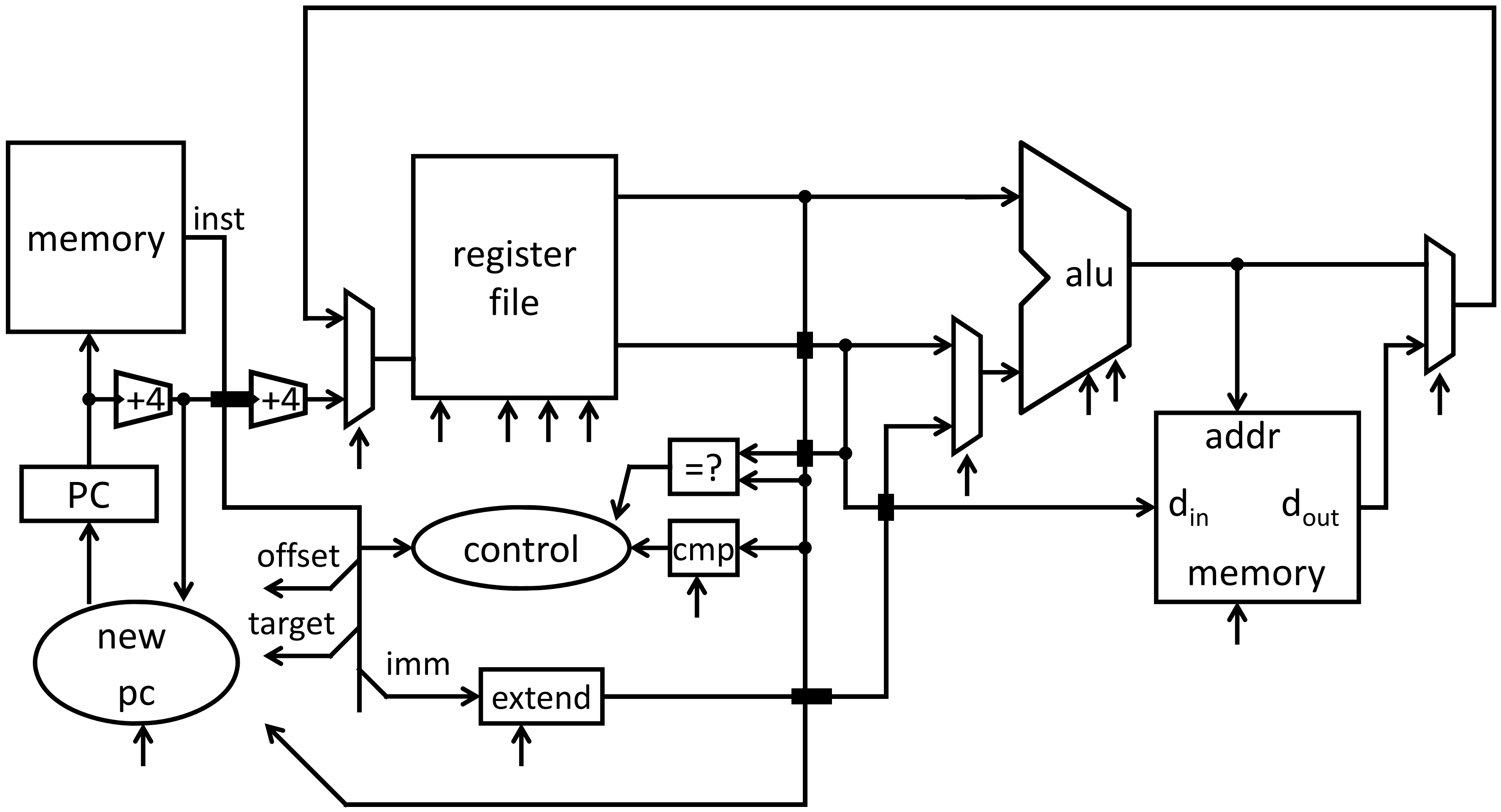
Late Policy

- Each person has a total of ***four*** “slip days”
- Max of ***two*** slip days for any individual assignment
- Slip days deducted first for *any* late assignment,
  cannot selectively apply slip days
- For projects, slip days are deducted from all partners
- ***25%*** deducted per day late after slip days are exhausted

Regrade policy

- Submit written request to lead TA,
  and lead TA will pick a different grader
- Submit another written request,
  lead TA will regrade directly
- Submit yet another written request for professor to regrade.

Big Picture: Building a Processor



A Single cycle processor

Goals for today

Review

- Finite State Machines

Memory

- Register Files
- Tri-state devices
- SRAM (Static RAM—random access memory)
- DRAM (Dynamic RAM)

Which statement(s) is true

(A) In a Moore Machine output depends on both current state and input

(B) In a Mealy Machine output depends on current state and input

(C) In a Mealy Machine output depends on next state and input

(D) All the above are true

(E) None are true

Which statement(s) is true

(A) In a Moore Machine output depends on both current state and input

[✓ (B) In a Mealy Machine output depends on current state and input]

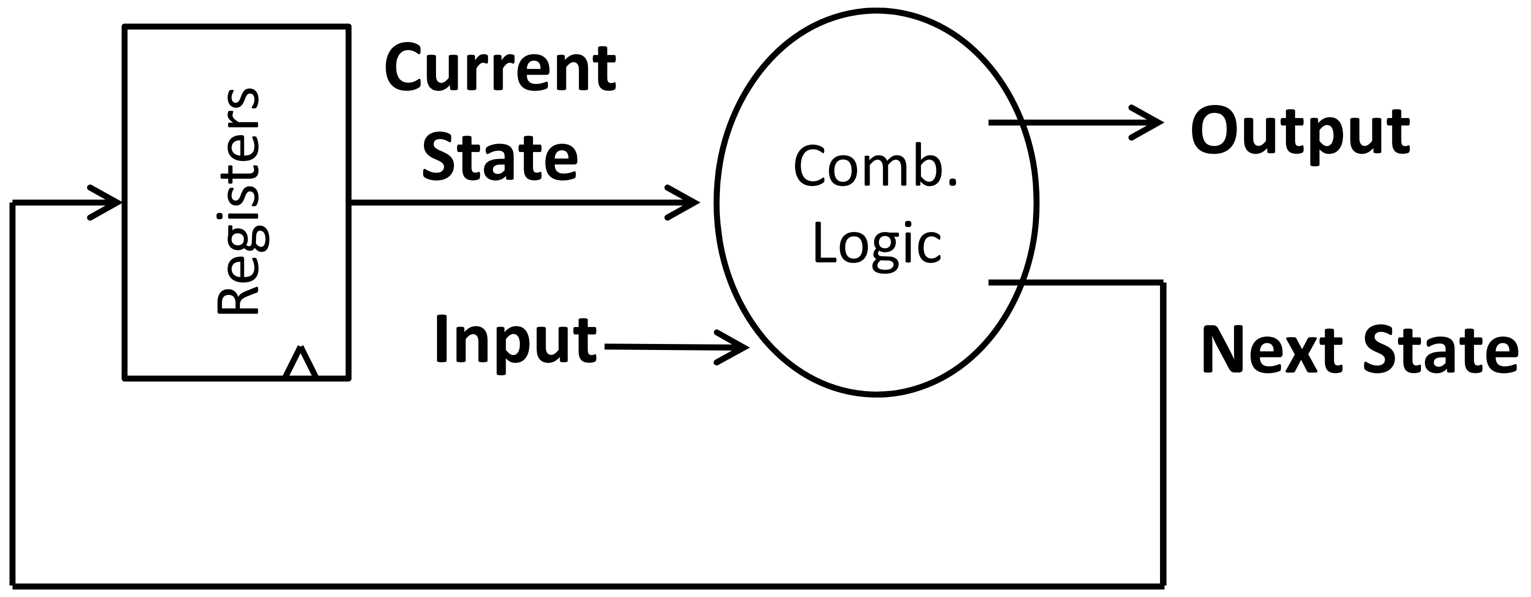
(C) In a Mealy Machine output depends on next state and input

(D) All the above are true

(E) None are true

Mealy Machine

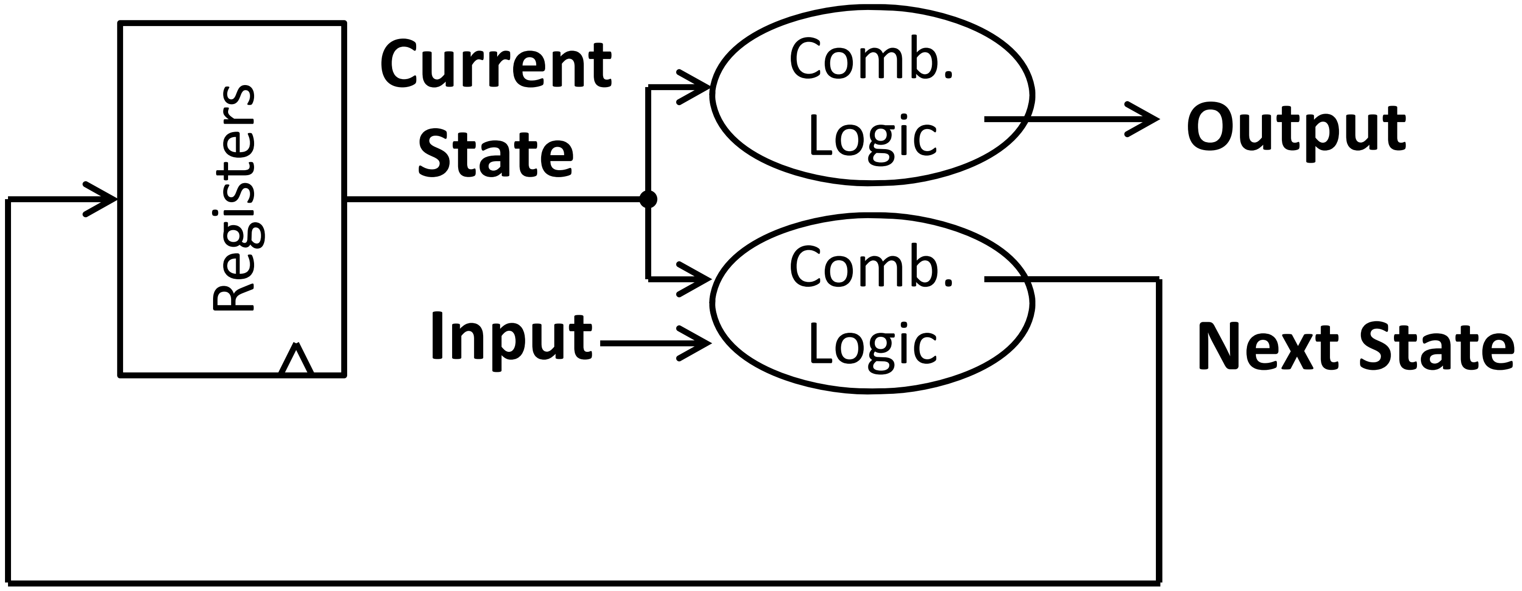
General Case: Mealy Machine



Outputs and next state depend on both current state and input

Moore Machine

Special Case: Moore Machine



Outputs depend only on current state

Example: Digital Door Lock

Digital Door Lock

Inputs:

- keycodes from keypad
- clock

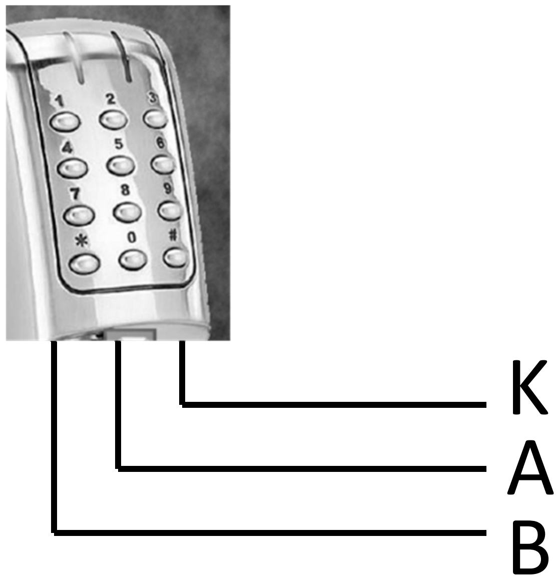
Outputs:

- “unlock” signal
- display how many keys pressed so far

Door Lock: Inputs

Assumptions:

- signals are synchronized to clock
- Password is B-A-B

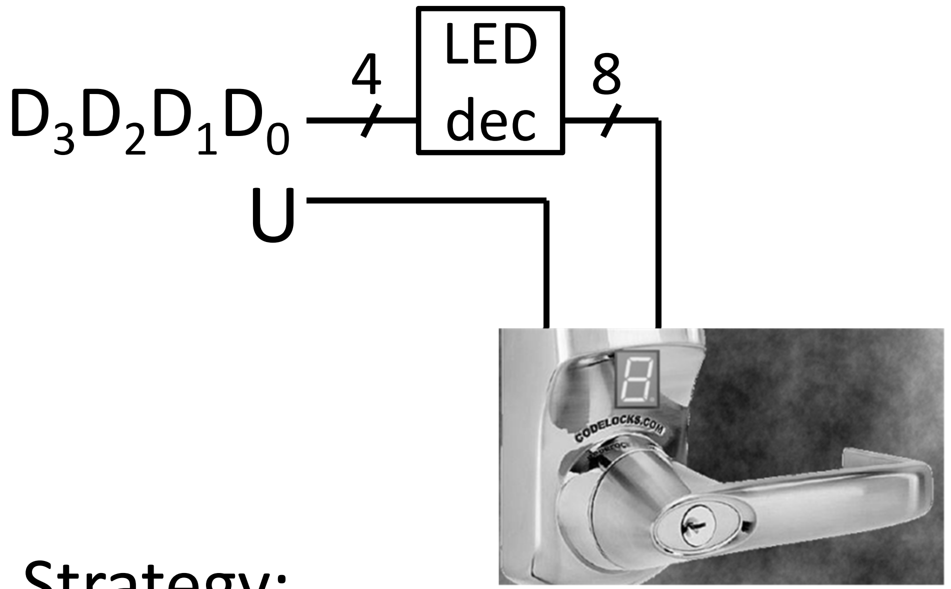


K	A	B	Meaning
0	0	0	Ø (no key)
1	1	0	‘A’ pressed
1	0	1	‘B’ pressed

Door Lock: Outputs

Assumptions:

- High pulse on U unlocks door



Strategy:

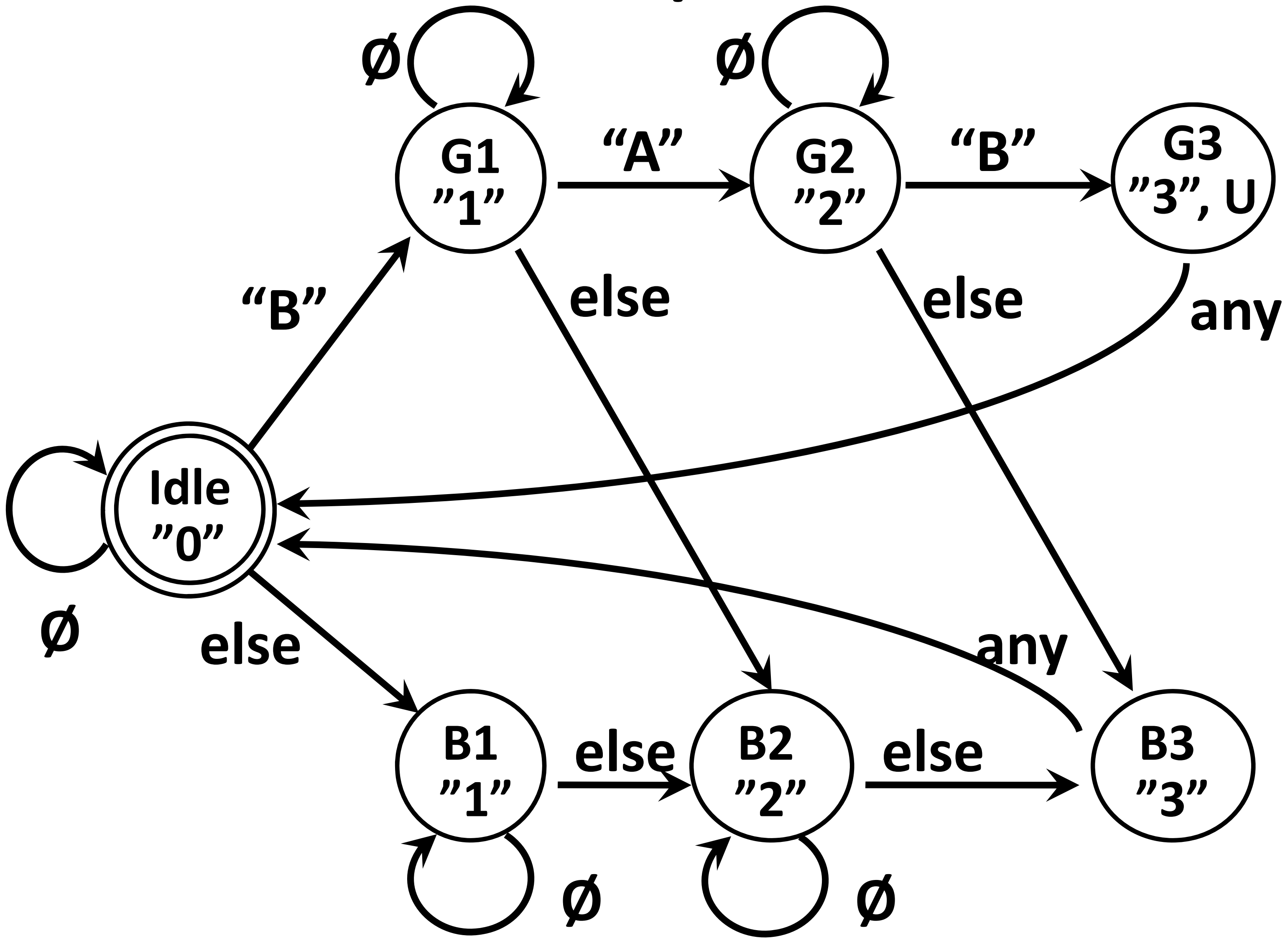
(1) Draw a state diagram (e.g. Moore Machine)

(2) Write output and next-state tables

(3) Encode states, inputs, and outputs as bits

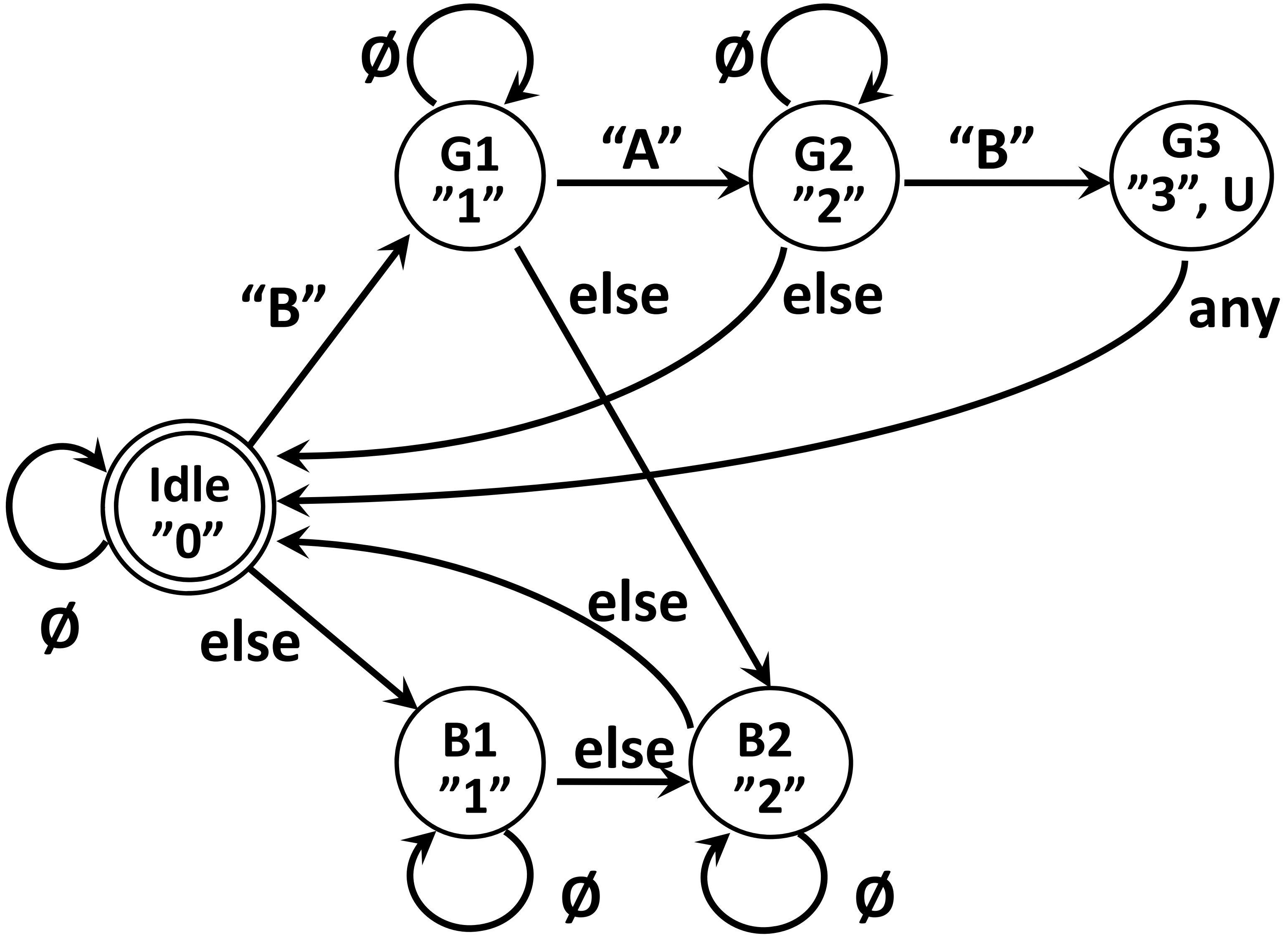
(4) Determine logic equations for next state and outputs

Door Lock: Simplified State Diagram



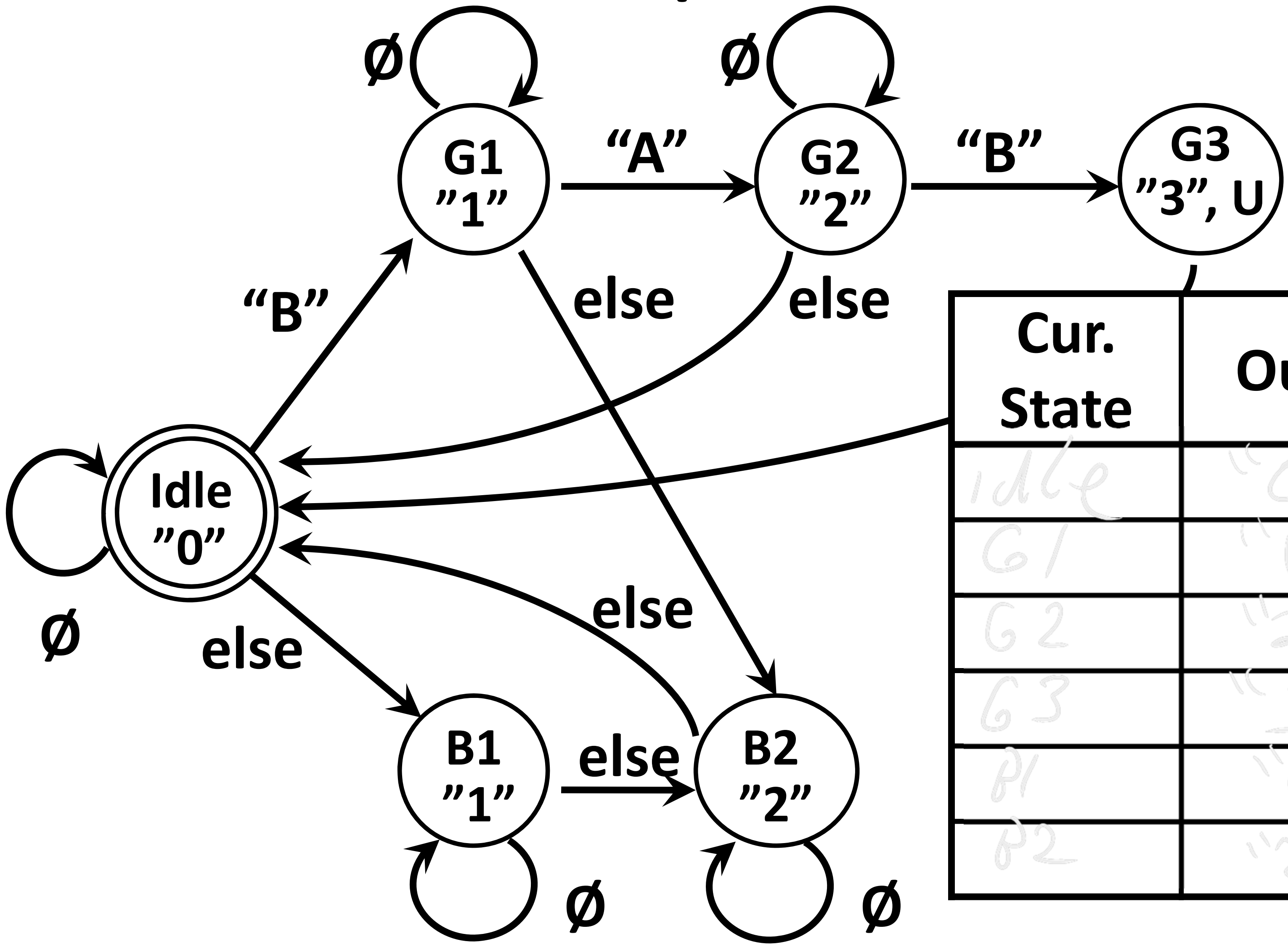
(1) Draw a state diagram (e.g. Moore Machine)

Door Lock: Simplified State Diagram



(1) Draw a state diagram (e.g. Moore Machine)

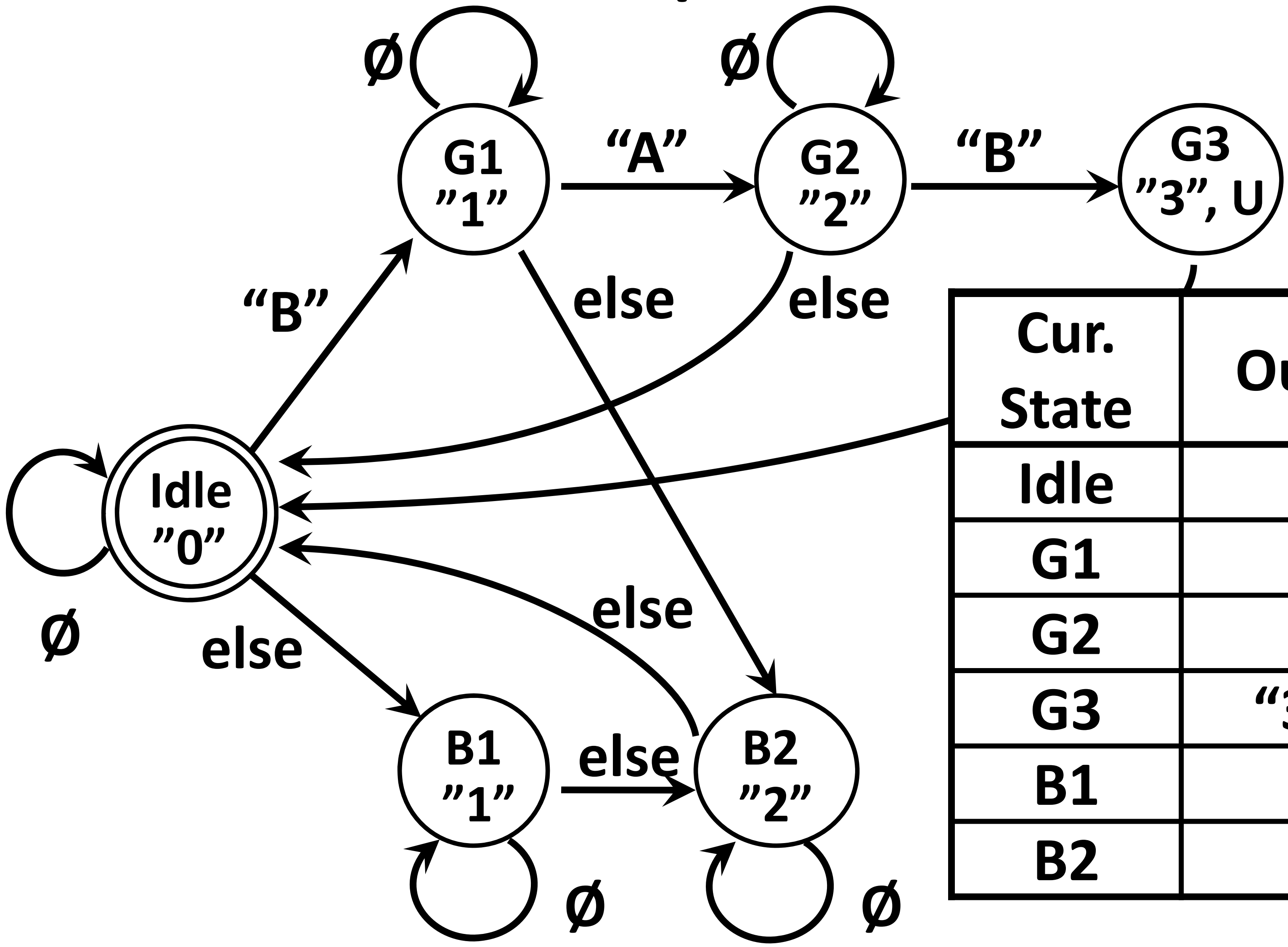
Door Lock: Simplified State Diagram



Cur. State	Output
Idle	"0"
G1	"1"
G2	"2"
G3	"3" U
B1	"1"
B2	"2"

(2) Write output and next-state tables

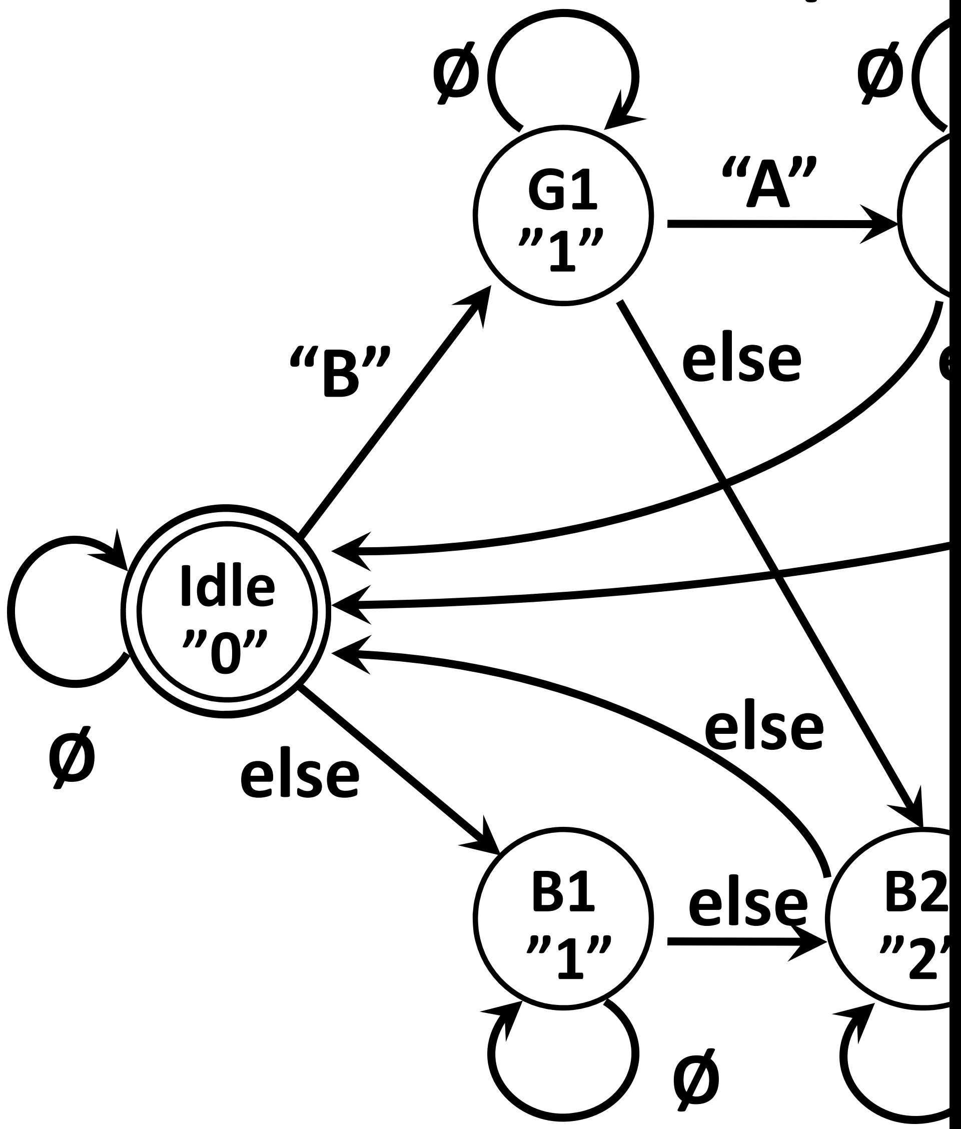
Door Lock: Simplified State Diagram



Cur. State	Output
Idle	“0”
G1	“1”
G2	“2”
G3	“3”, U
B1	“1”
B2	“2”

(2) Write output and next-state tables

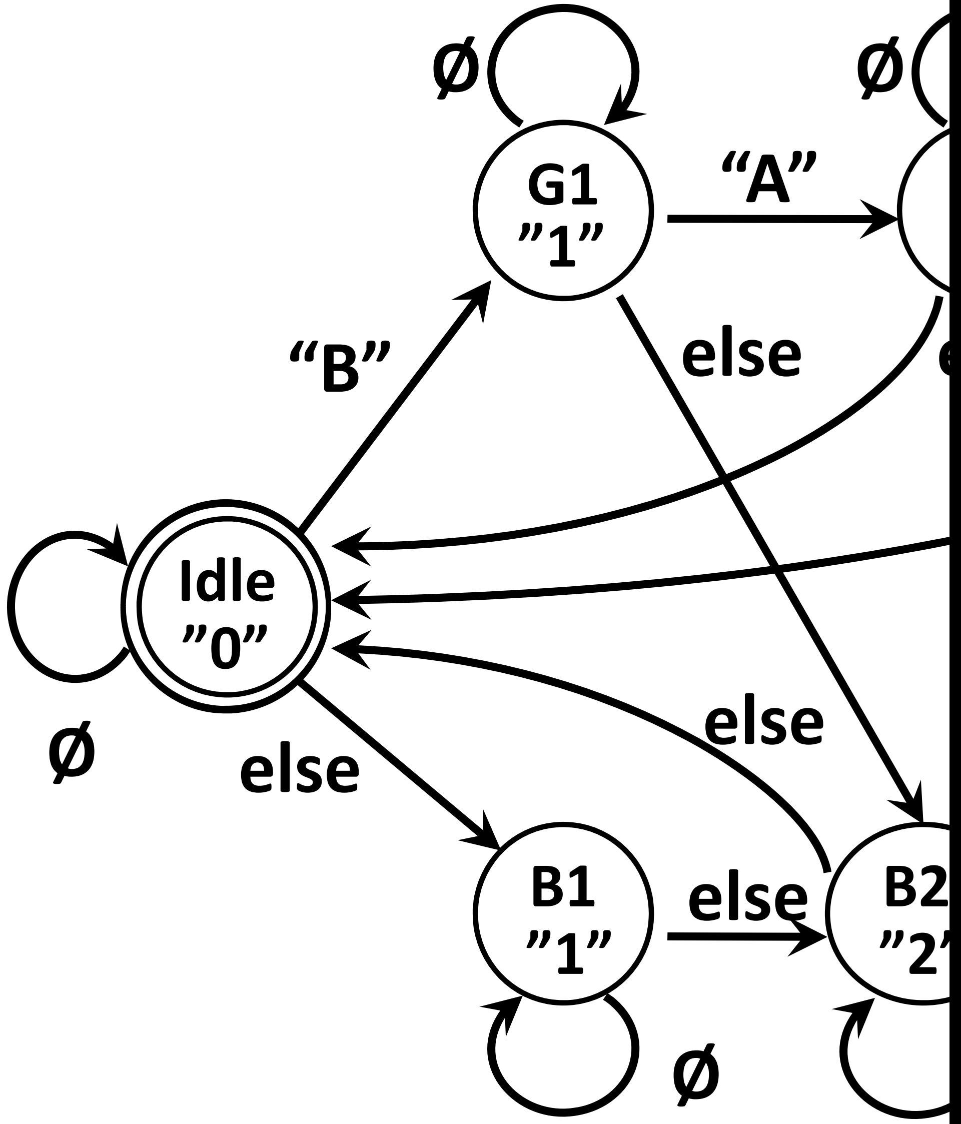
Door Lock: Simplified State Diagram



Cur. State	Input	Next State
Idle	0	Idle
Idle	"B"	G1
Idle	"A"	B1
G1	0	G1
G1	A	G2
G1	B	B2
G2		
G2		
G2		
G3		
B1		
[illegible]		
B2		
B2		

(2) Write output and next-state tables

Door Lock: Simplified State Diagram



Cur. State	Input	Next State
Idle	Ø	Idle
Idle	"B"	G1
Idle	"A"	B1
G1	Ø	G1
G1	"A"	G2
G1	"B"	B2
G2	Ø	B2
G2	"B"	G3
G2	"A"	Idle
G3	any	Idle
B1	Ø	B1
B1	K	B2
B2	Ø	B2
B2	K	Idle

(2) Write output and next-state tables

State Table Encoding

S_2	S_1	S_0	D_3	D_2	D_1	D_0	U
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	0	0	0	1	0
1	0	1	0	0	1	0	0

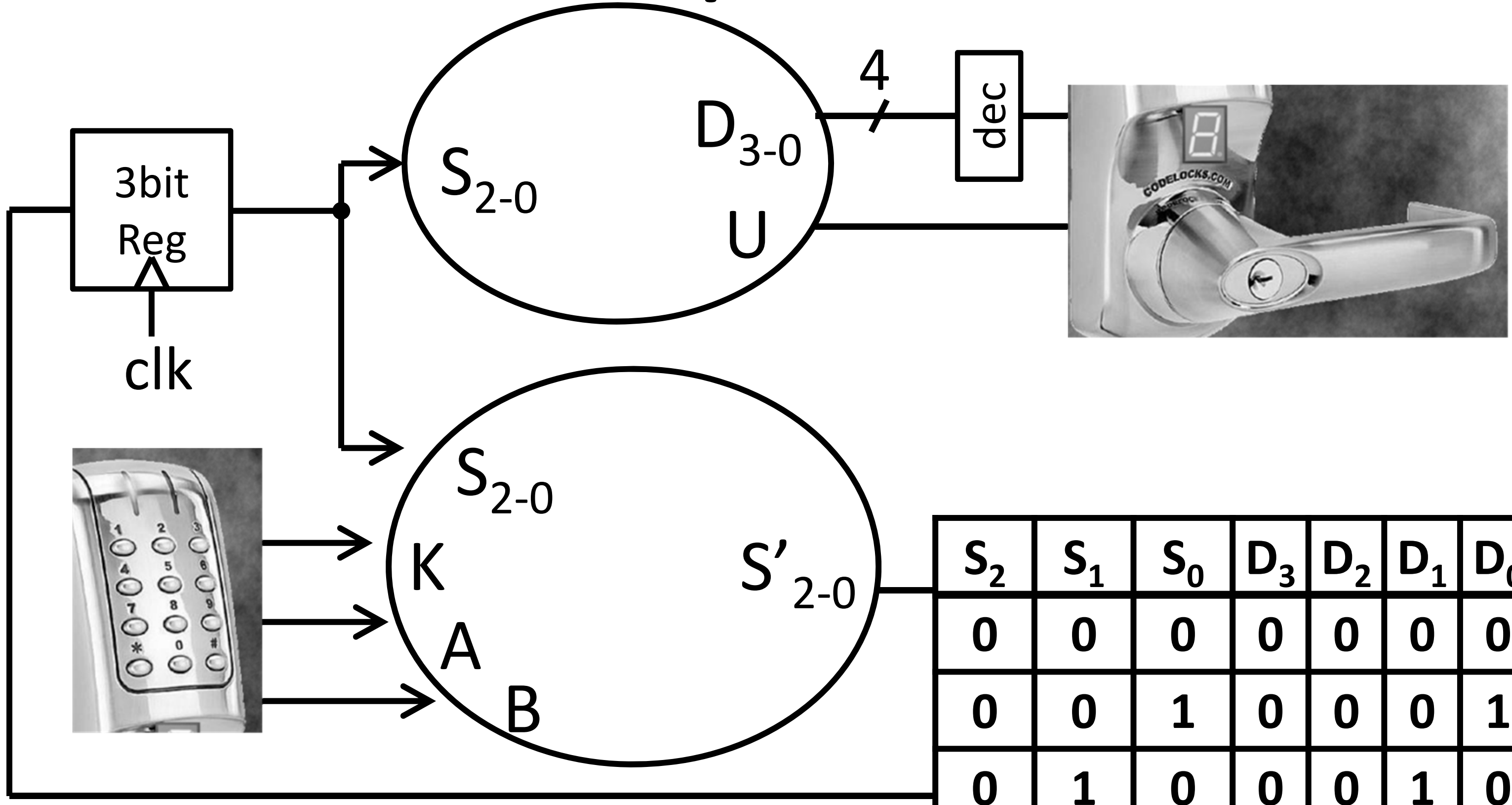
S_2	S_1	S_0	K	A	B	S'_2	S'_1	S'_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	1
0	0	0	1	1	0	1	0	0
0	0	1	0	0	0	0	0	1
0	0	1	1	1	0	0	1	0
0	0	1	1	0	1	1	0	1
0	1	0	0	0	0	0	1	0
0	1	0	1	0	1	0	1	1
0	1	0	1	1	0	0	0	0
0	1	1	x	x	x	0	0	0
1	0	0	0	0	0	1	0	0
1	0	0	1	x	x	1	0	1
1	0	1	0	0	0	1	0	1
1	0	1	1	x	x	0	0	0

State	S_2	S_1	S_0
Idle	0	0	0
G1	0	0	1
G2	0	1	0
G3	0	1	1
B1	1	0	0
B2	1	0	1

D_3D

(ts, and outputs as bits

Door Lock: Implementation

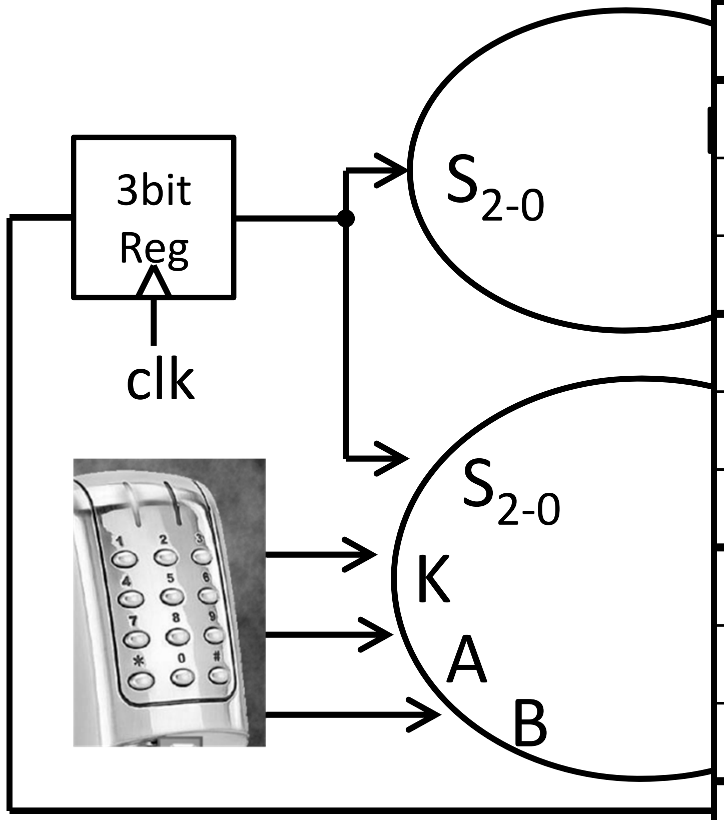


S_2	S_1	S_0	D_3	D_2	D_1	D_0	U
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	0	0	0	1	0
1	0	1	0	0	1	0	0

$$U = \overline{S}_2 S_1 S_0$$
$$D_0 = \overline{S_2 S_1} S_0 + \overline{S}_2 S_1 S_0 + S_2\, \overline{S_1 S_0}$$
$$D_1 = \overline{S}_2 S_1 S_0 + \overline{S}_2 S_1 S_0 + \overline{S}_2 S_1 S_0$$

(4) Determine logic equations for next state and outputs

Door Lock: Implementation



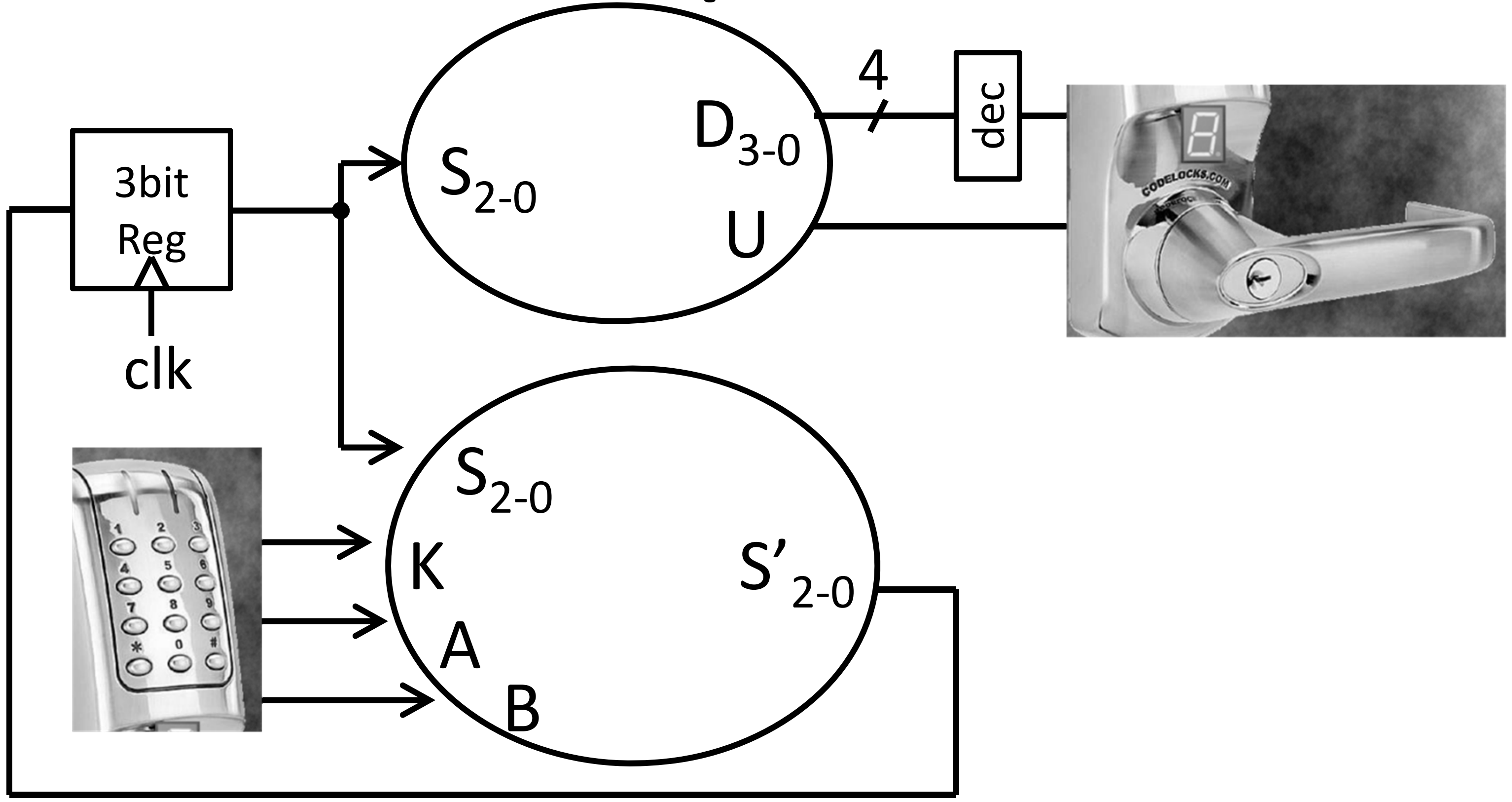
S_2	S_1	S_0	K	A	B	S'_2	S'_1	S'_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	1
0	0	0	1	1	0	1	0	0
0	0	1	0	0	0	0	0	1
0	0	1	1	1	0	0	1	0
0	0	1	1	0	1	1	0	1
0	1	0	0	0	0	0	1	0
0	1	0	1	0	1	0	1	1
0	1	0	1	1	0	0	0	0
0	1	1	x	x	x	0	0	0
1	0	0	0	0	0	1	0	0
1	0	0	1	x	x	1	0	1
1	0	1	0	0	0	1	0	1
1	0	1	1	x	x	0	0	0

$S_0' = ?$

$S_1' = ?$

$S_2' = \overline{S_2S_1S_0}KA\overline{B} + \overline{S_2S_1}S_0K\overline{A}B + S_2\overline{S_1S_2KAB} + \overline{S}_2S_1S_0K + S_2\,\overline{S}_1S_0\,\overline{KAB}$

Door Lock: Implementation



Strategy:

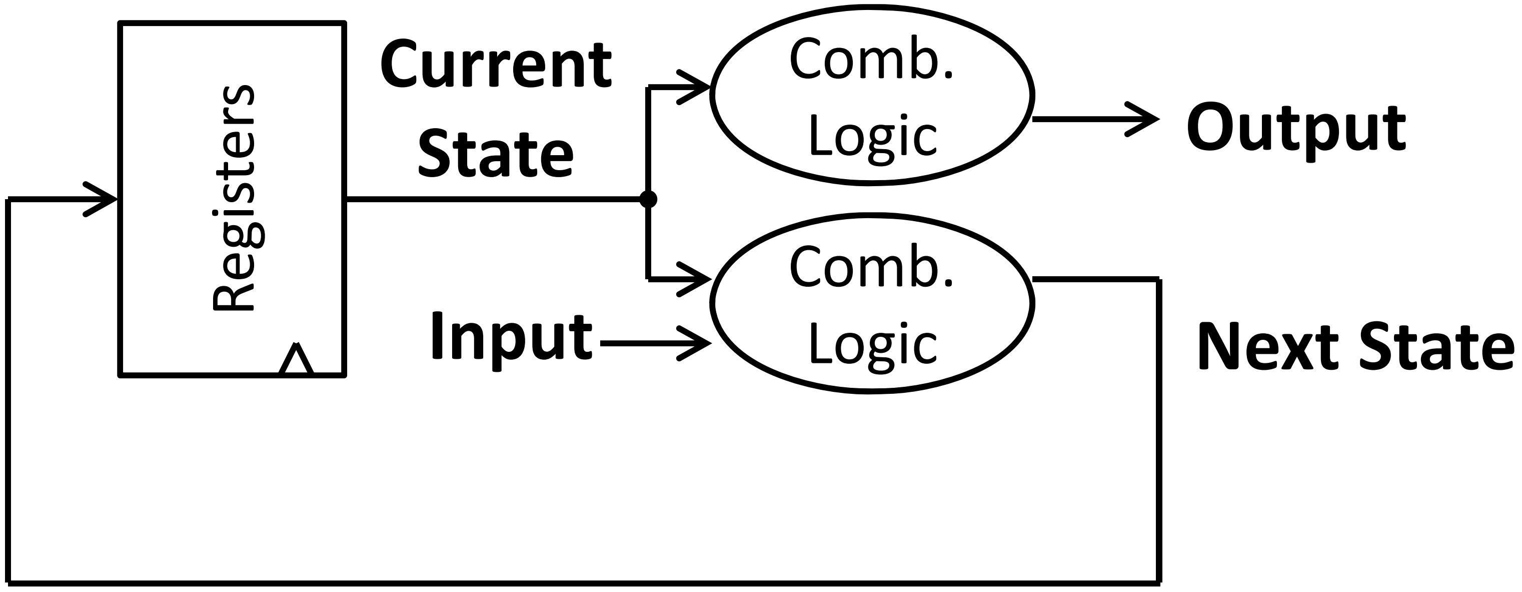
(1) Draw a state diagram (e.g. Moore Machine)

(2) Write output and next-state tables

(3) Encode states, inputs, and outputs as bits

(4) Determine logic equations for next state and outputs

Door Lock: Implementation



Moore Machine

Strategy:

(1) Draw a state diagram (e.g. Moore Machine)

(2) Write output and next-state tables

(3) Encode states, inputs, and outputs as bits

(4) Determine logic equations for next state and outputs

Goals for today

Review

- Finite State Machines

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)

Goal:

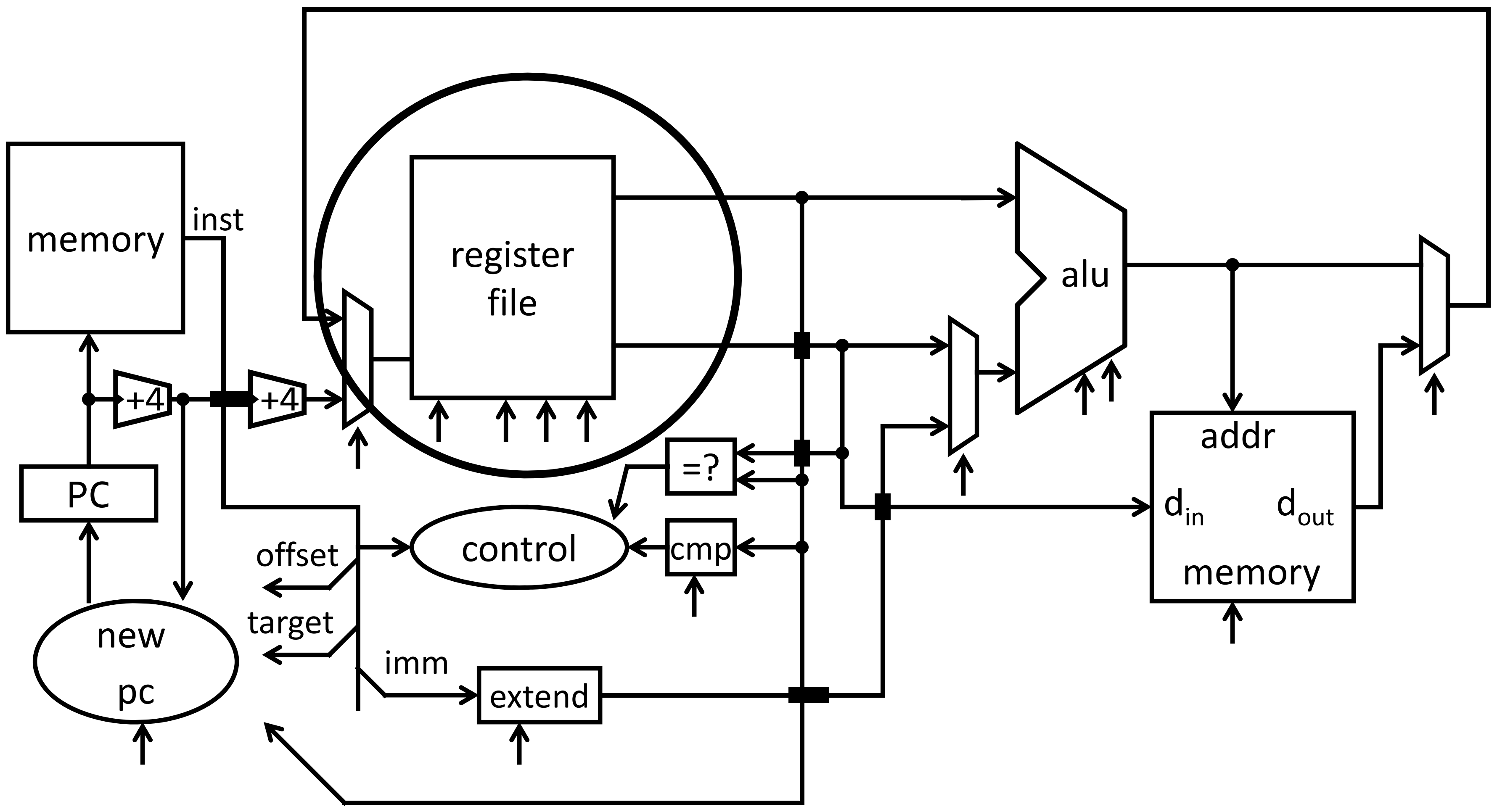
How do we store results from ALU computations?

How do we use stored results in subsequent operations?

Register File

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

Big Picture: Building a Processor

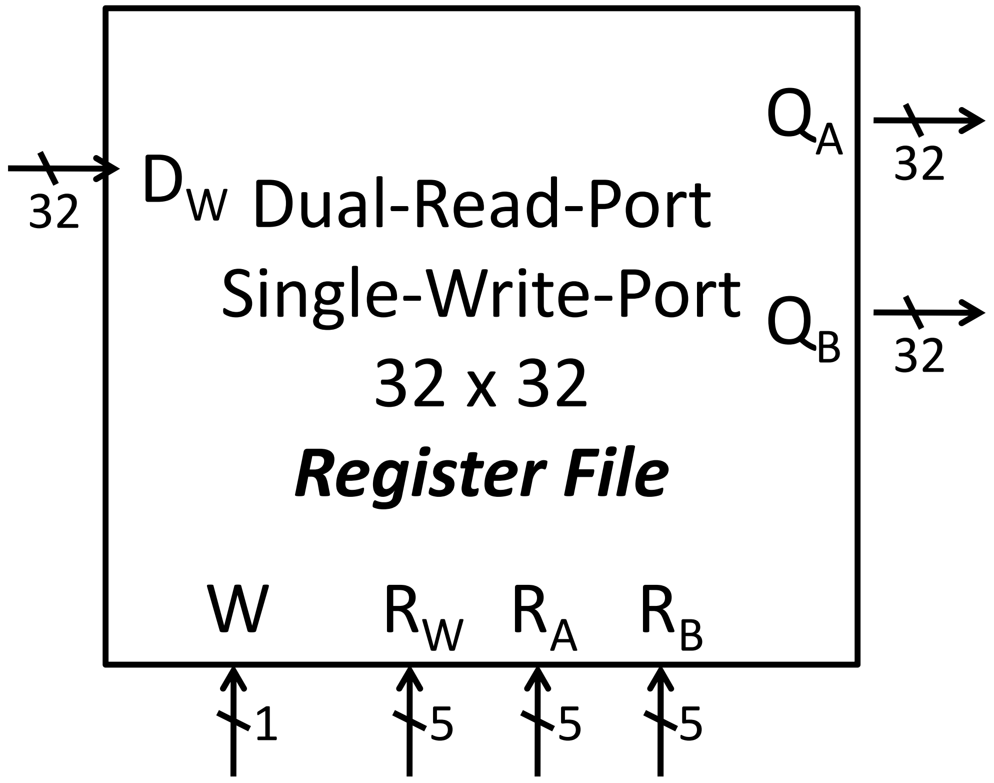


A Single cycle processor

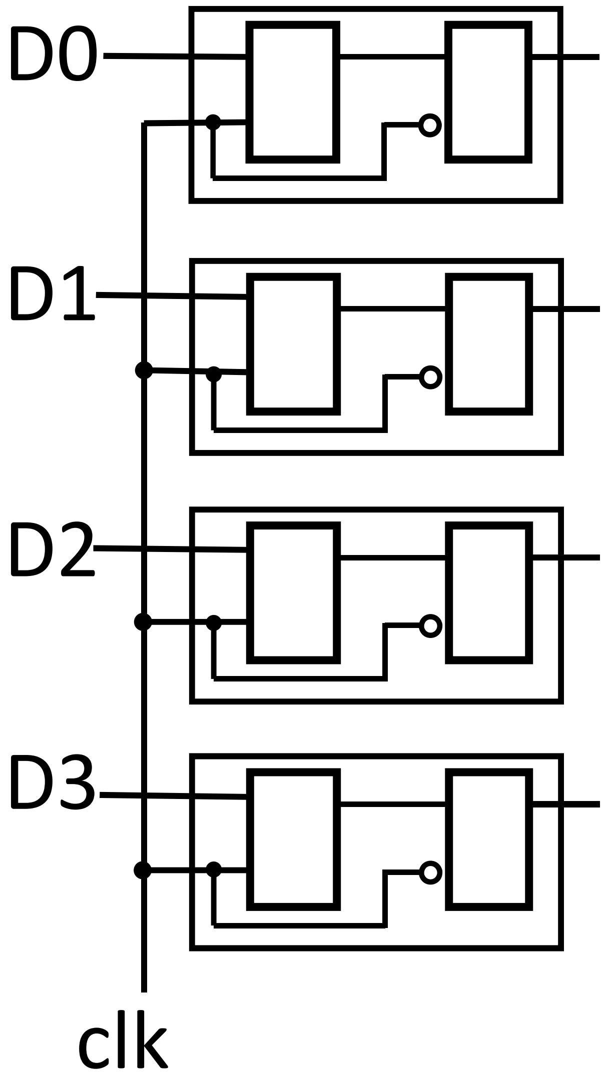
Register File

Register File

- N read/write registers
- Indexed by register number

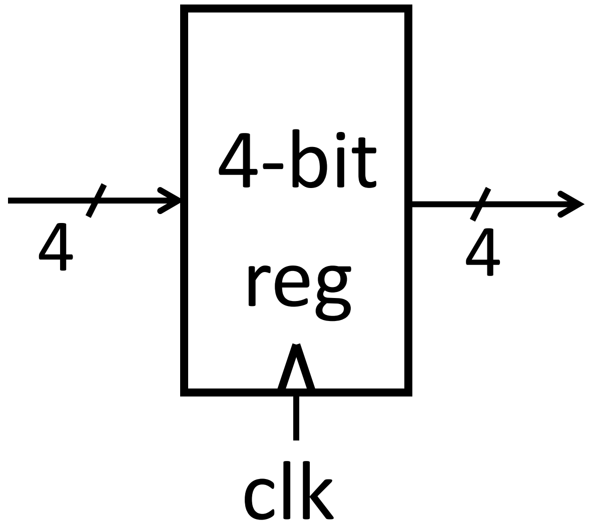


Register File



Recall: Register

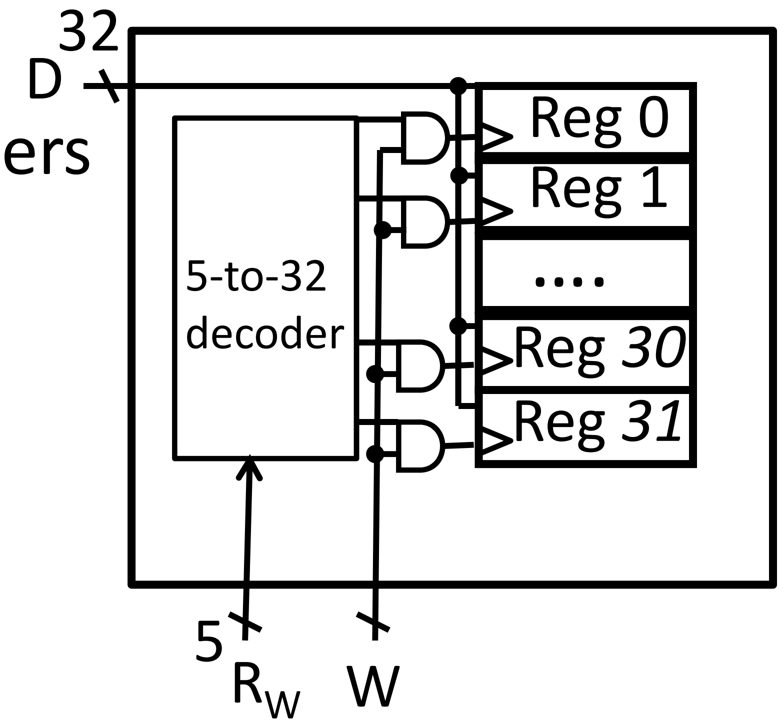
- D flip-flops in parallel
- shared clock
- extra clocked inputs: write_enable, reset, …



Register File

Register File

- N read/write registers
- Indexed by register number



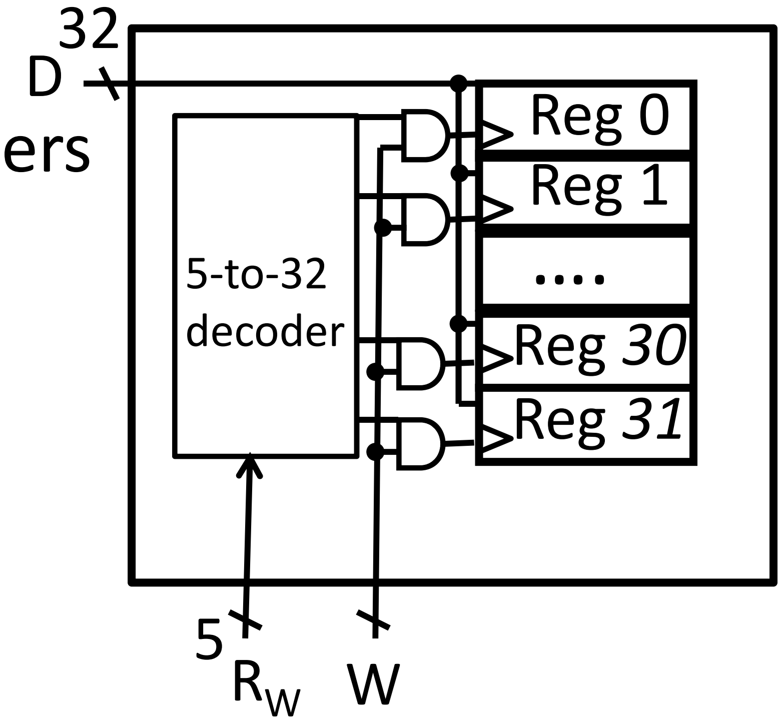
How to write to ***one*** register in the register file?

- Need a decoder

Activity# write truth table for 3-to-8 decoder

Register File

- N read/write registers
- Indexed by register number



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

- Need a decoder

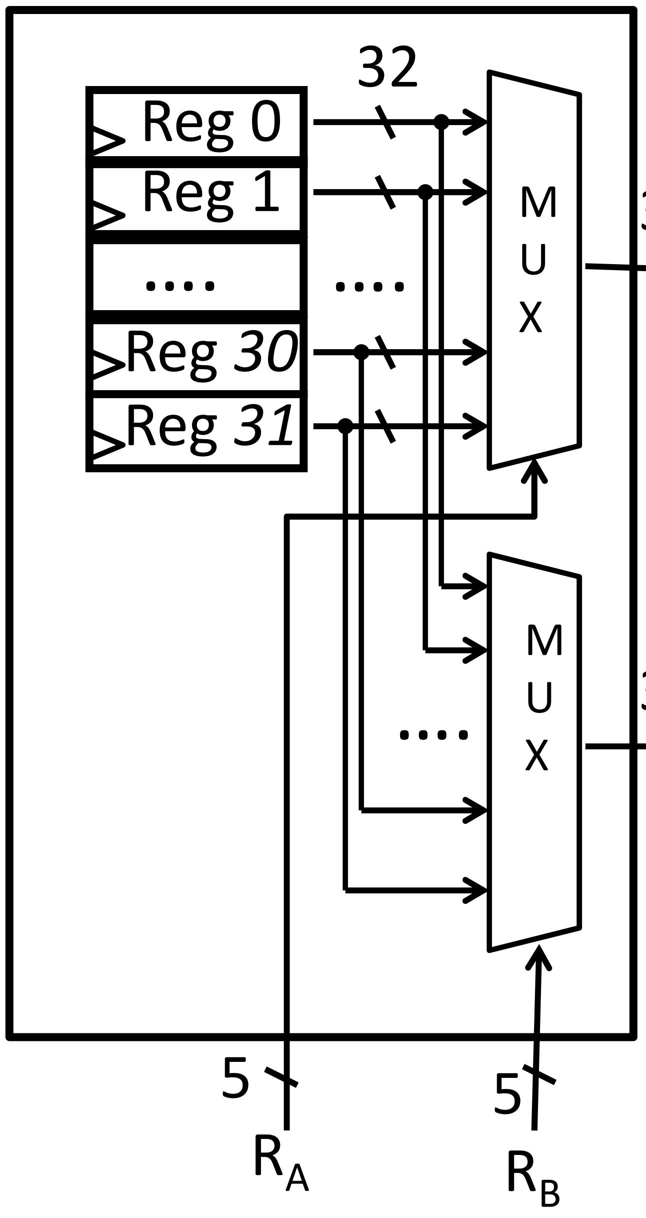
Register File

Register File

- N read/write registers
- Indexed by register number

How to read from two registers?

- Need a multiplexor



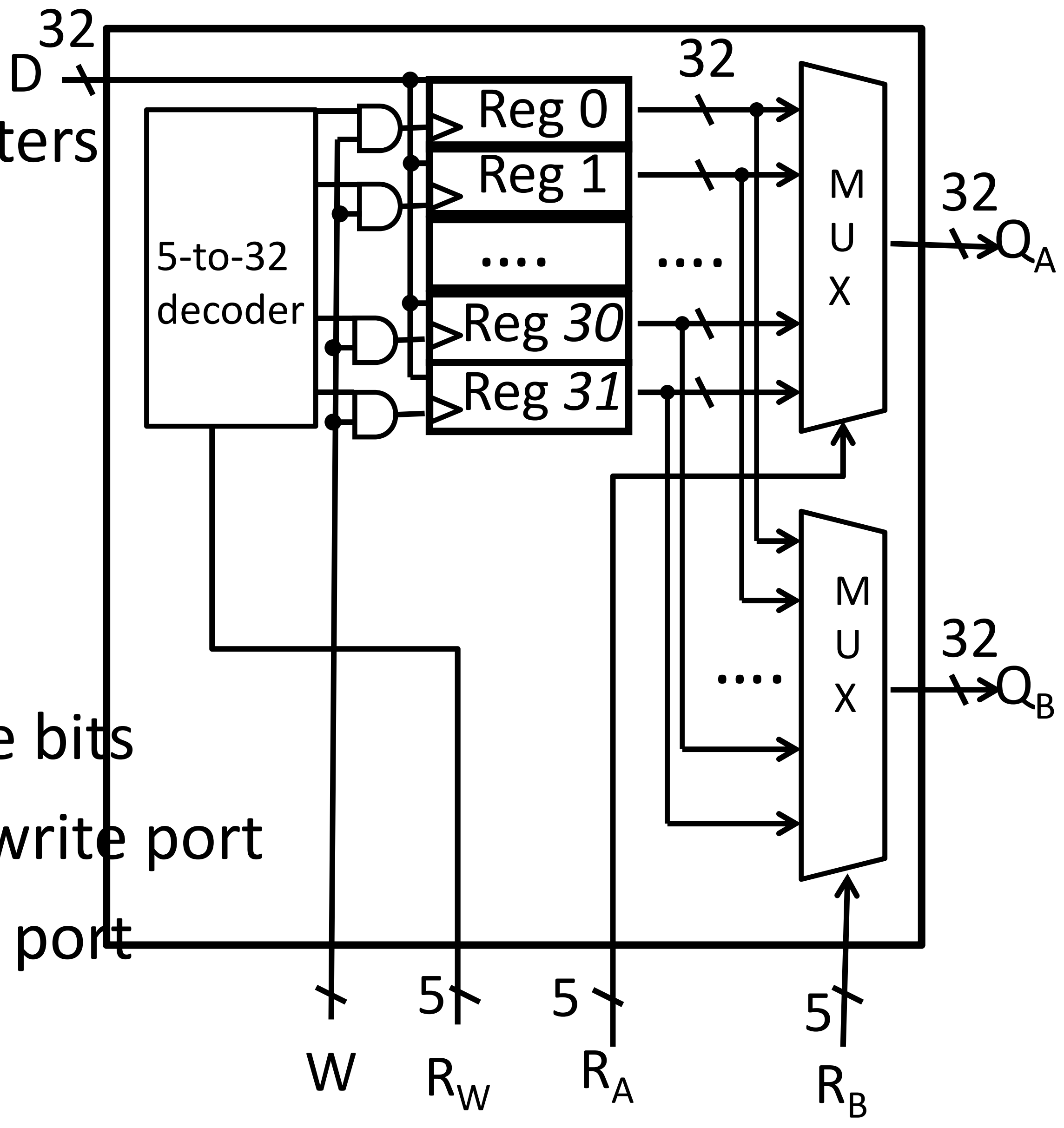
Register File

Register File

- N read/write registers
- Indexed by register number

Implementation:

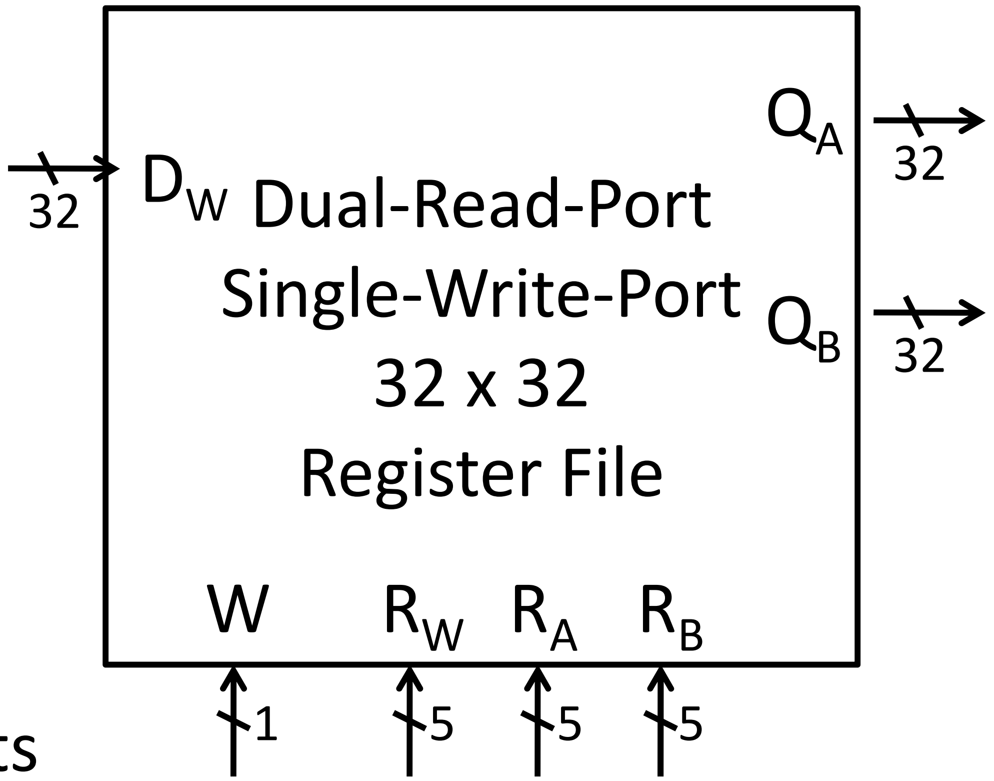
- D flip flops to store bits
- Decoder for each write port
- Mux for each read port



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

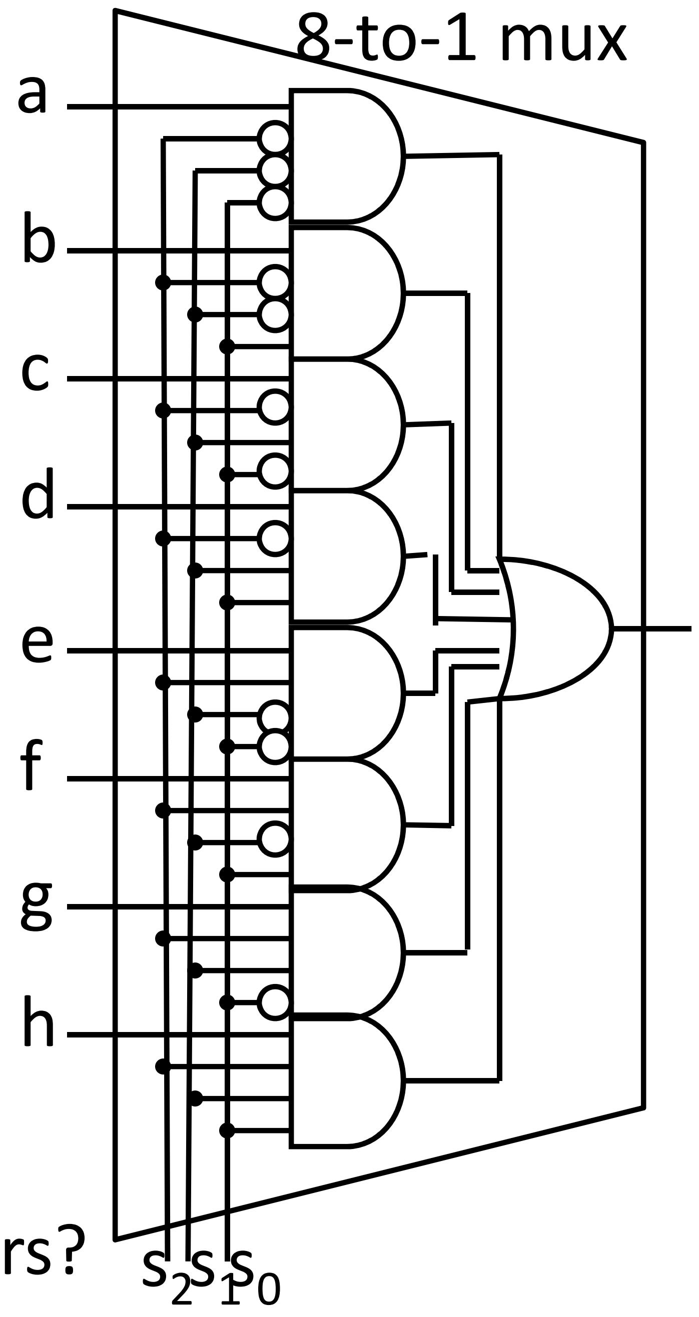
- + Very fast (a few gate delays for both read and write)
- + Adding extra ports is straightforward
- – Doesn't scale

  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

Review

- Finite State Machines

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)

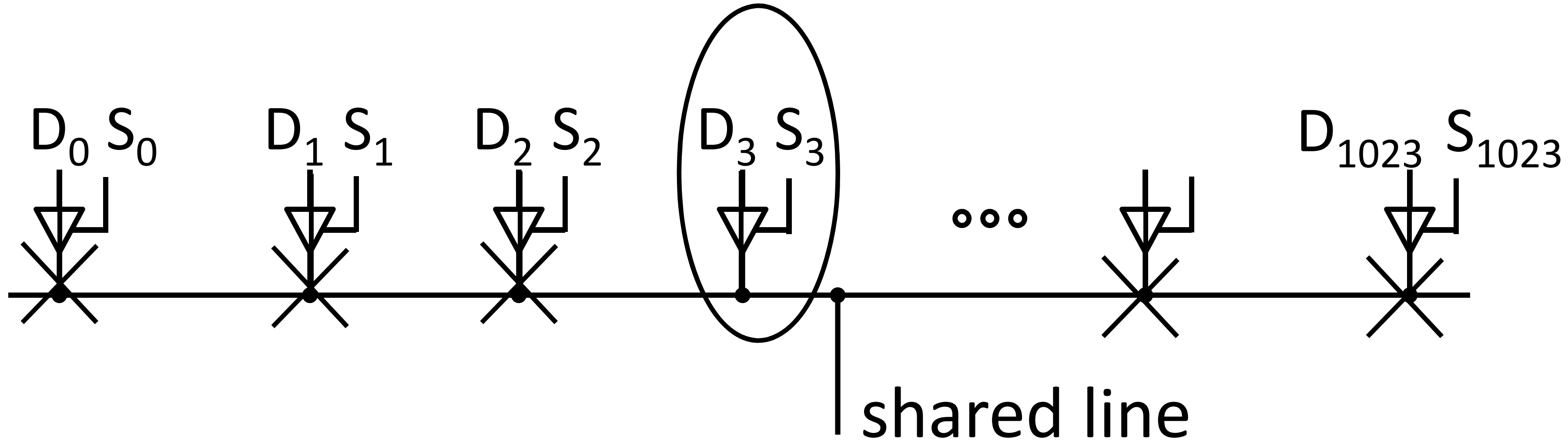
Next Goal

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

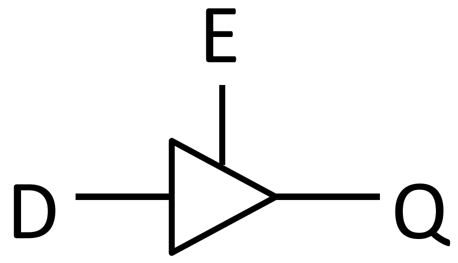


- How do we build such a device?

Tri-State Devices

Tri-State Buffers

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

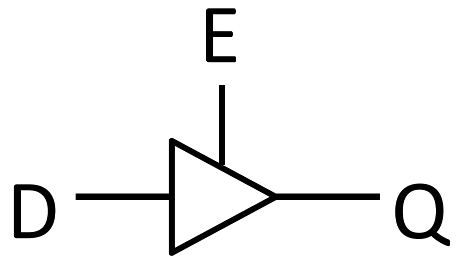


E	D	Q
0	0	z
0	1	z
1	0	0
1	1	1

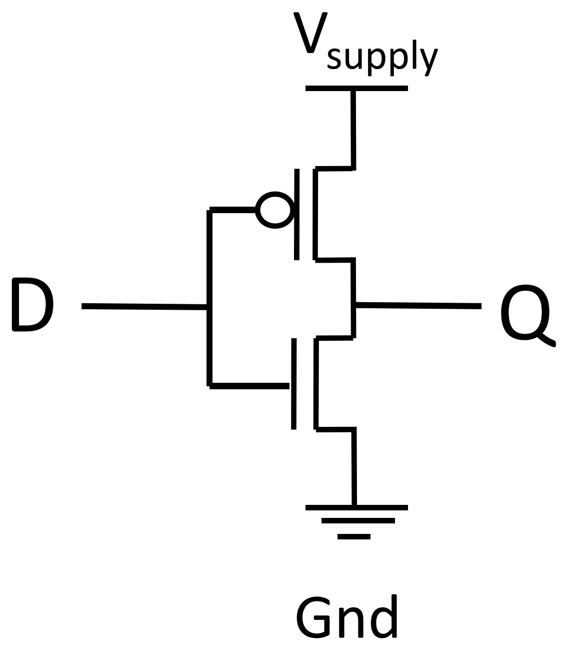
Tri-State Devices

Tri-State Buffers

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



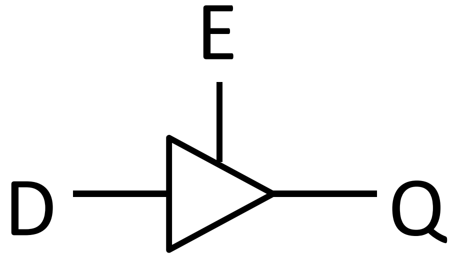
E	D	Q
0	0	z
0	1	z
1	0	0
1	1	1



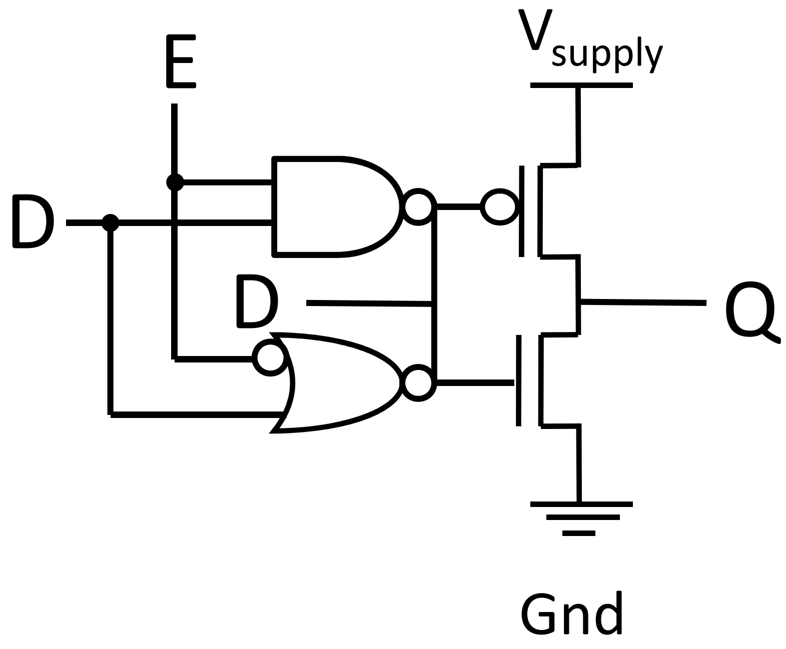
Tri-State Devices

Tri-State Buffers

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



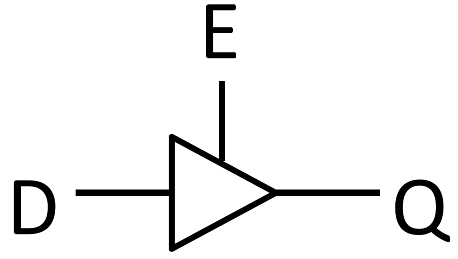
E	D	Q
0	0	z
0	1	z
1	0	0
1	1	1



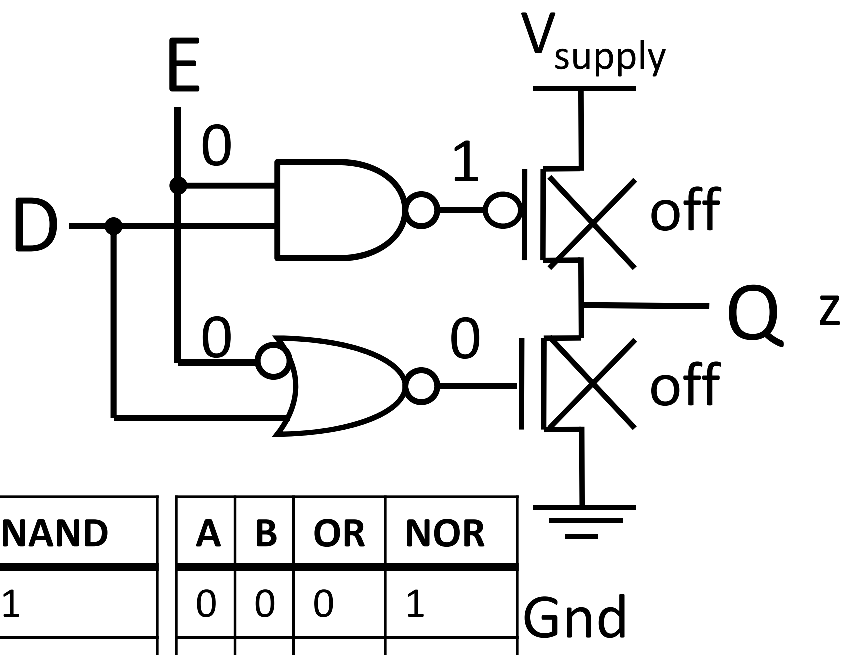
Tri-State Devices

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



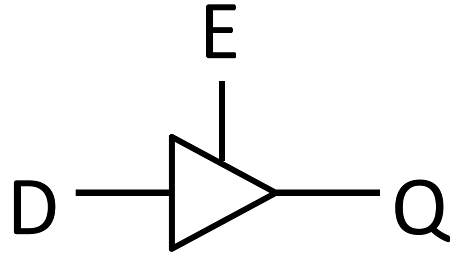
A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

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

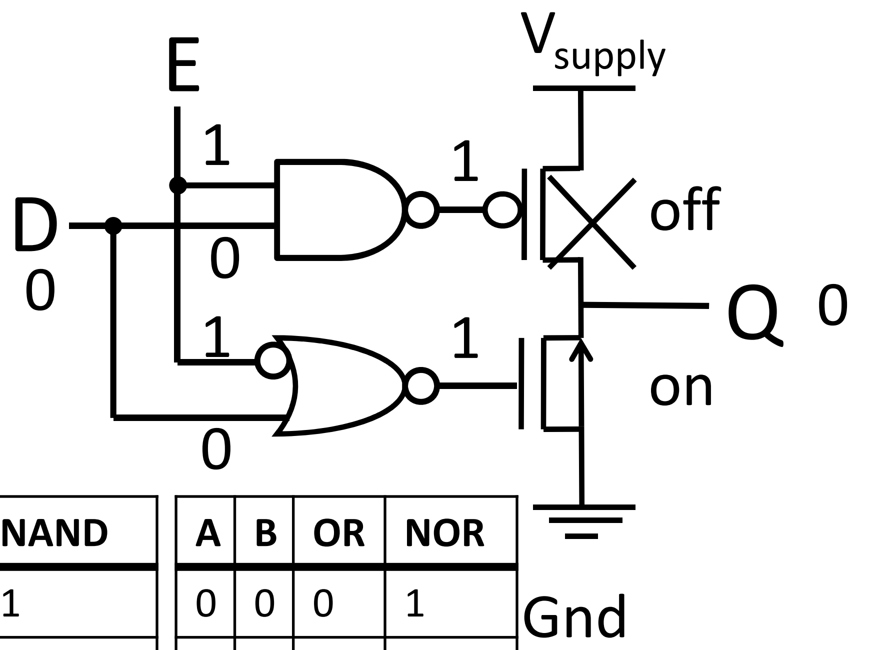
Tri-State Devices

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



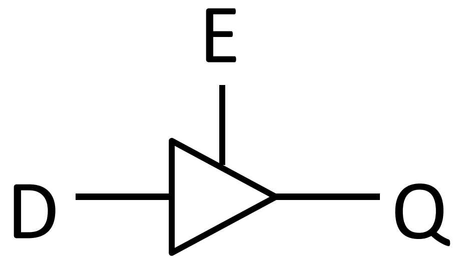
A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

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

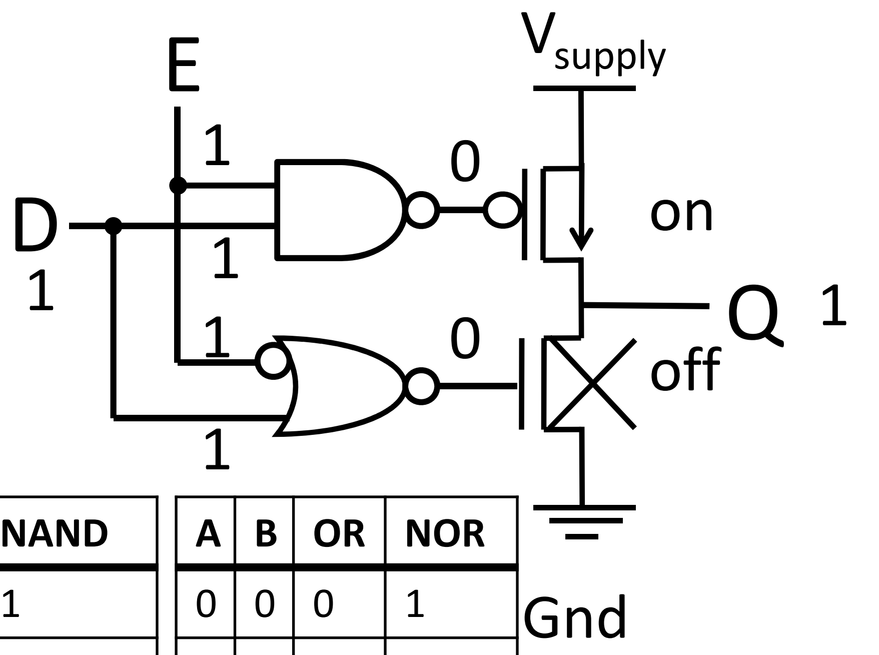
Tri-State Devices

Tri-State Buffers

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



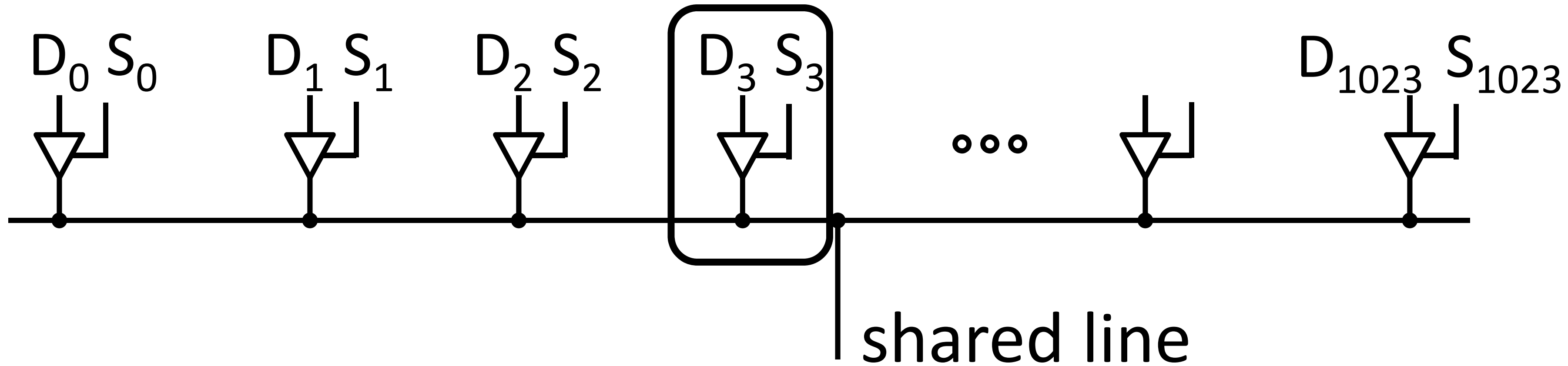
E	D	Q
0	0	z
0	1	z
1	0	0
1	1	1



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

Review

- Finite State Machines

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

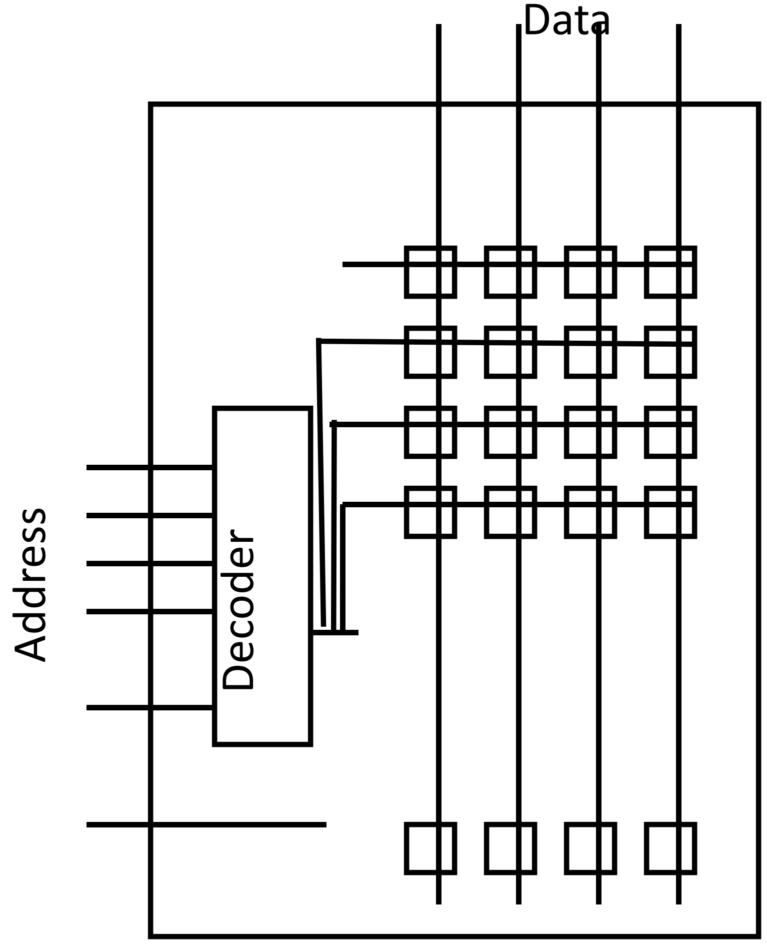
How do we build large memories?

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

SRAM

Static RAM (SRAM)—Static Random Access Memory

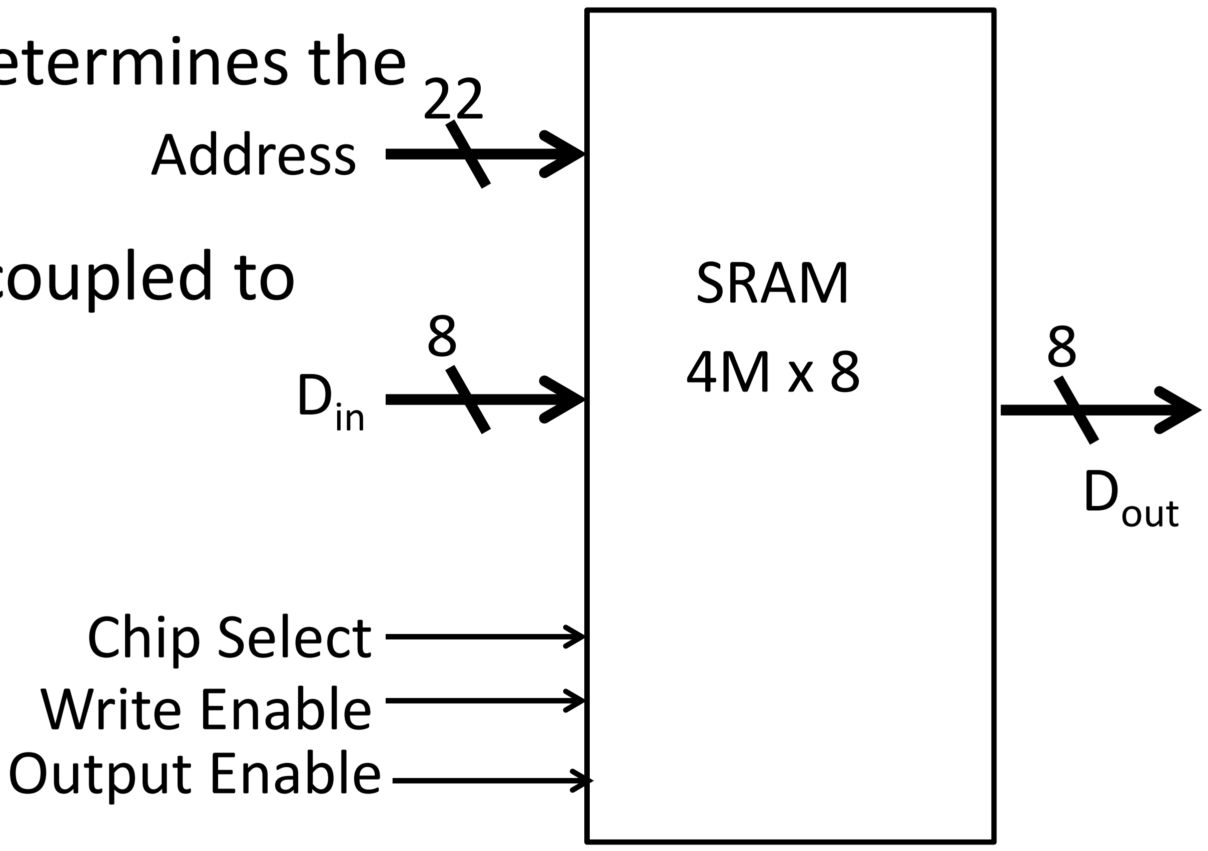
- Essentially just D-Latches plus Tri-State Buffers
- A decoder selects which line of memory to access (i.e. word line)
- A R/W selector determines the type of access
- That line is then coupled to the data lines



SRAM

Static RAM (SRAM)—Static Random Access Memory

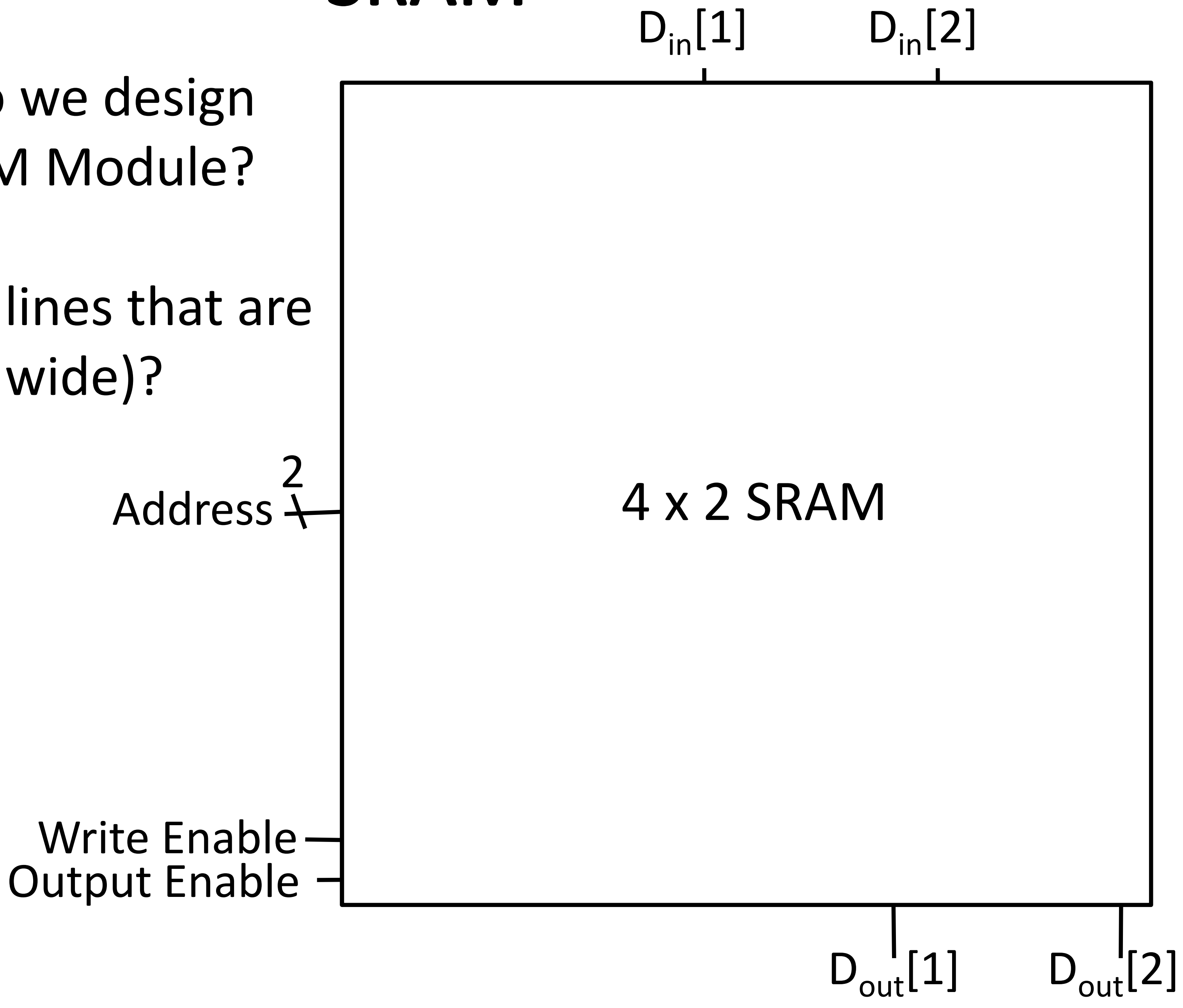
- Essentially just D-Latches plus Tri-State Buffers
- A decoder selects which line of memory to access (i.e. word line)
- A R/W selector determines the type of access
- That line is then coupled to the data lines



SRAM

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

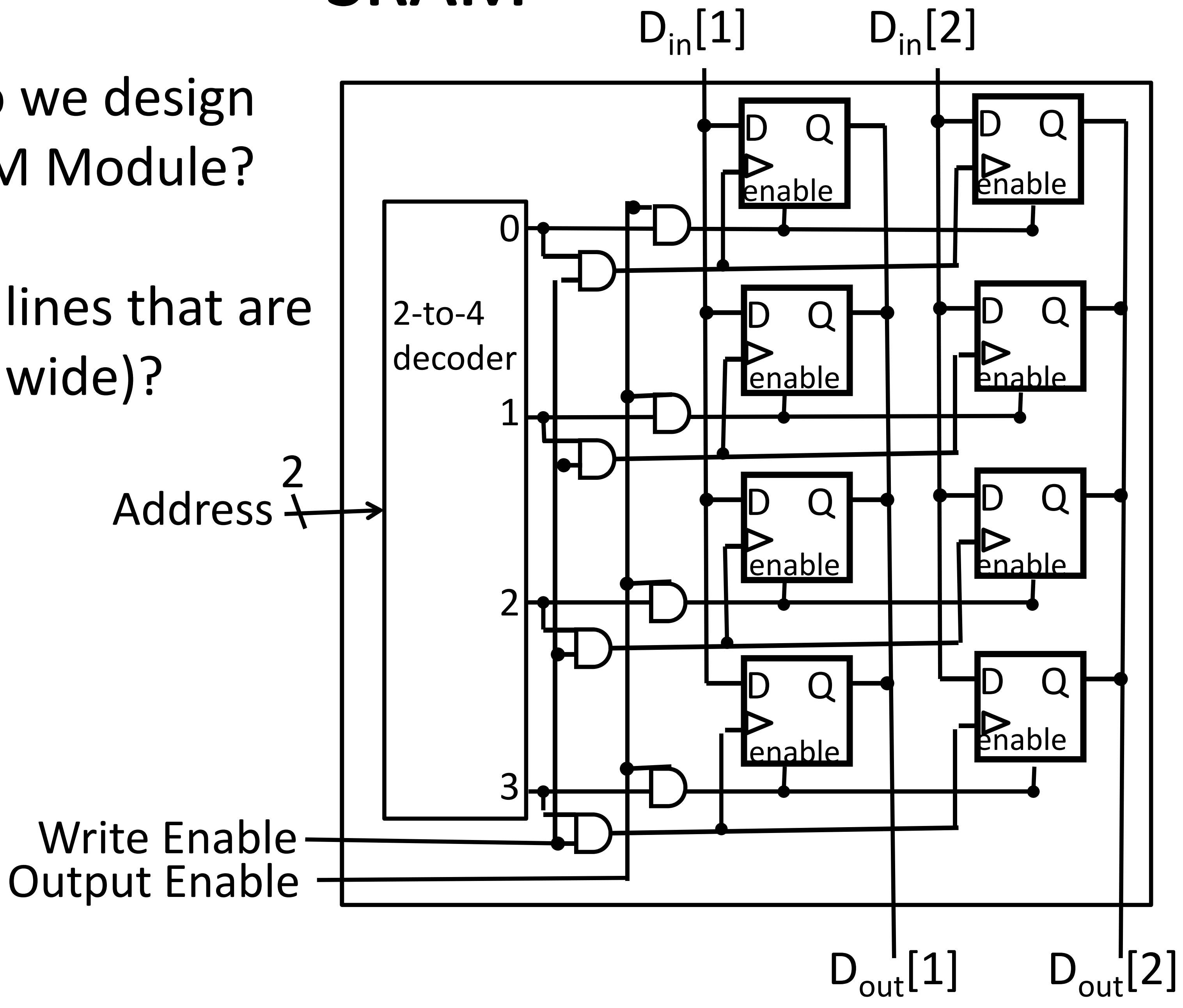
(i.e. 4 word lines that are each 2 bits wide)?



SRAM

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

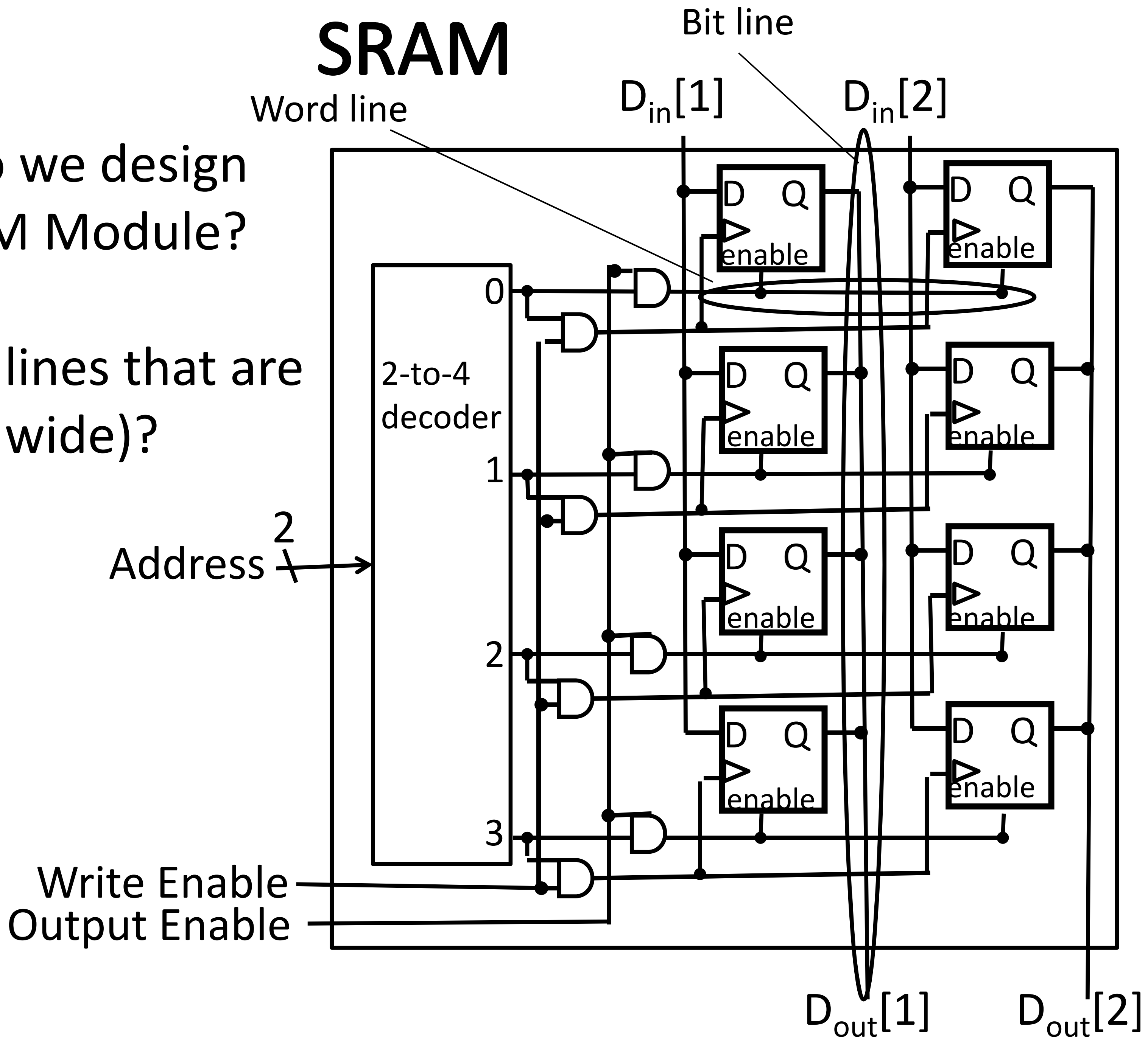
(i.e. 4 word lines that are each 2 bits wide)?



SRAM

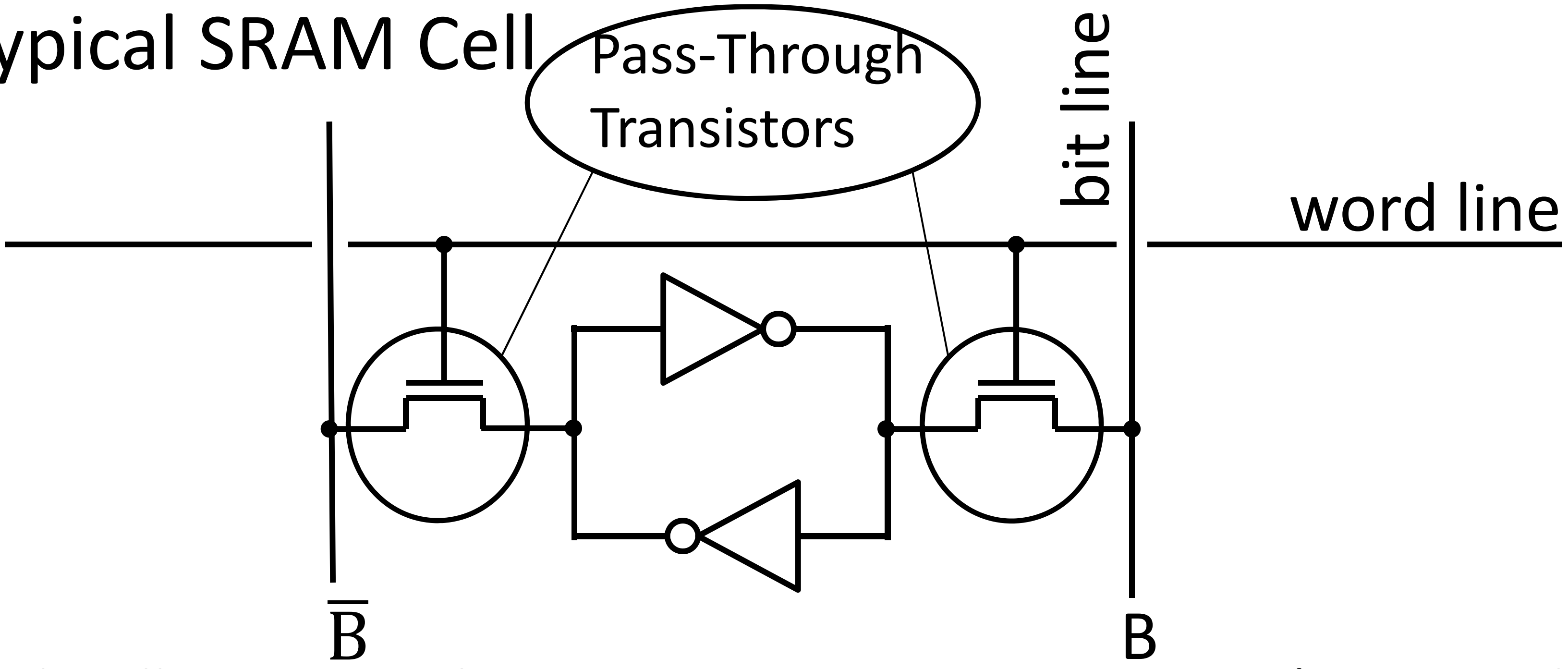
E.g. How do we design a 4 x 2 SRAM Module?

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



SRAM Cell

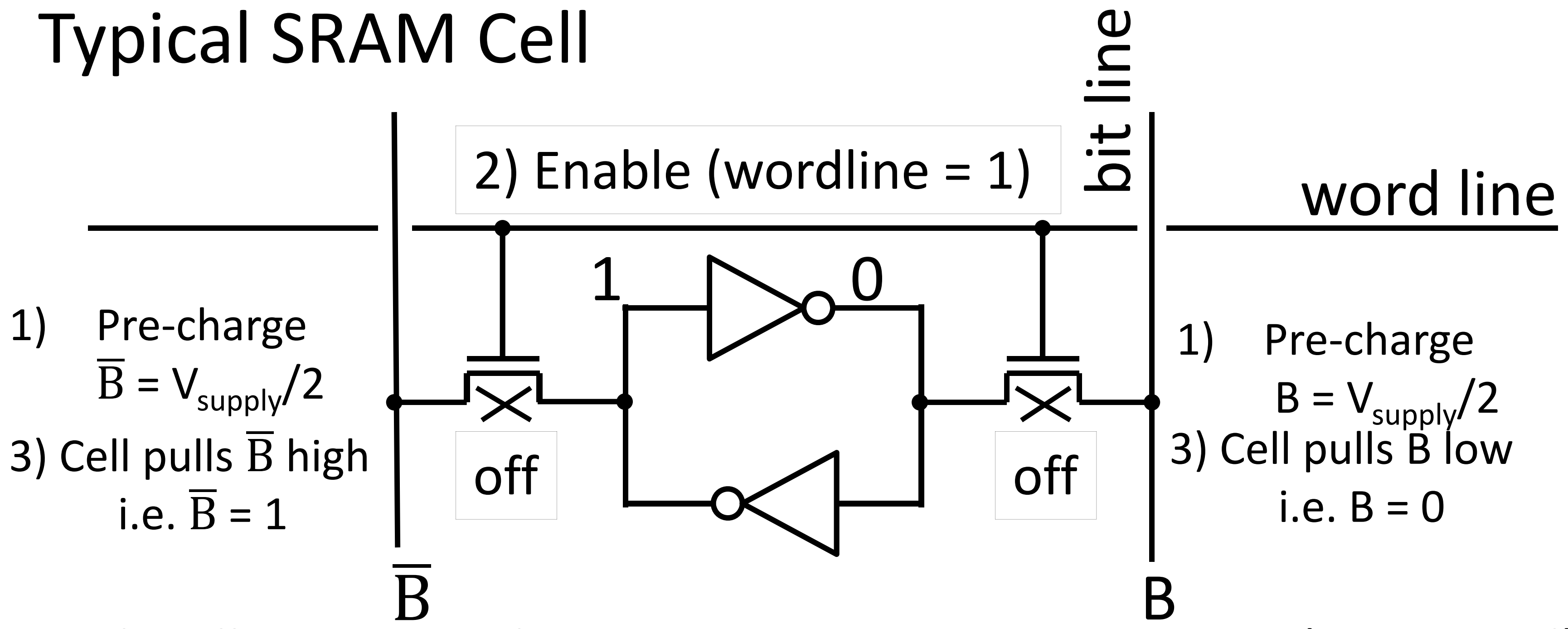
Typical SRAM Cell



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

SRAM Cell

Typical SRAM Cell



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

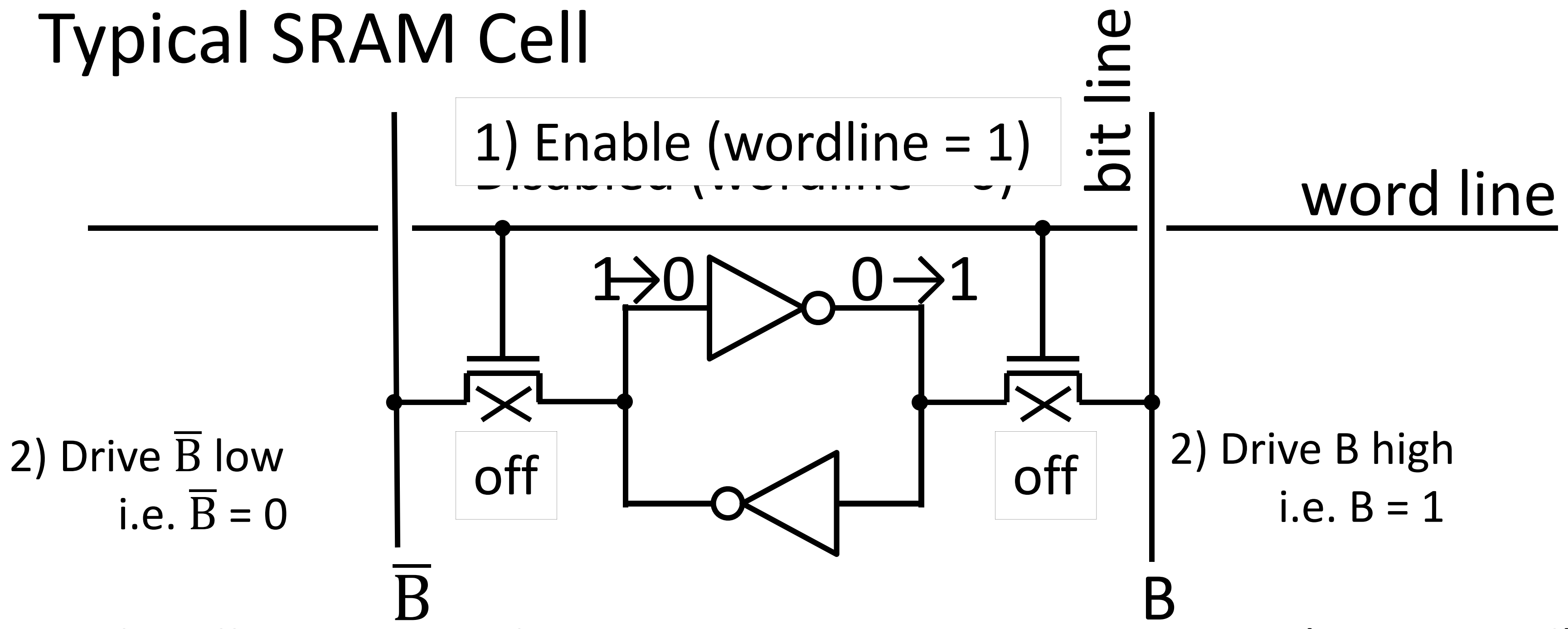
Each cell stores one bit, and requires 4 – 8 transistors (6 is typical)

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

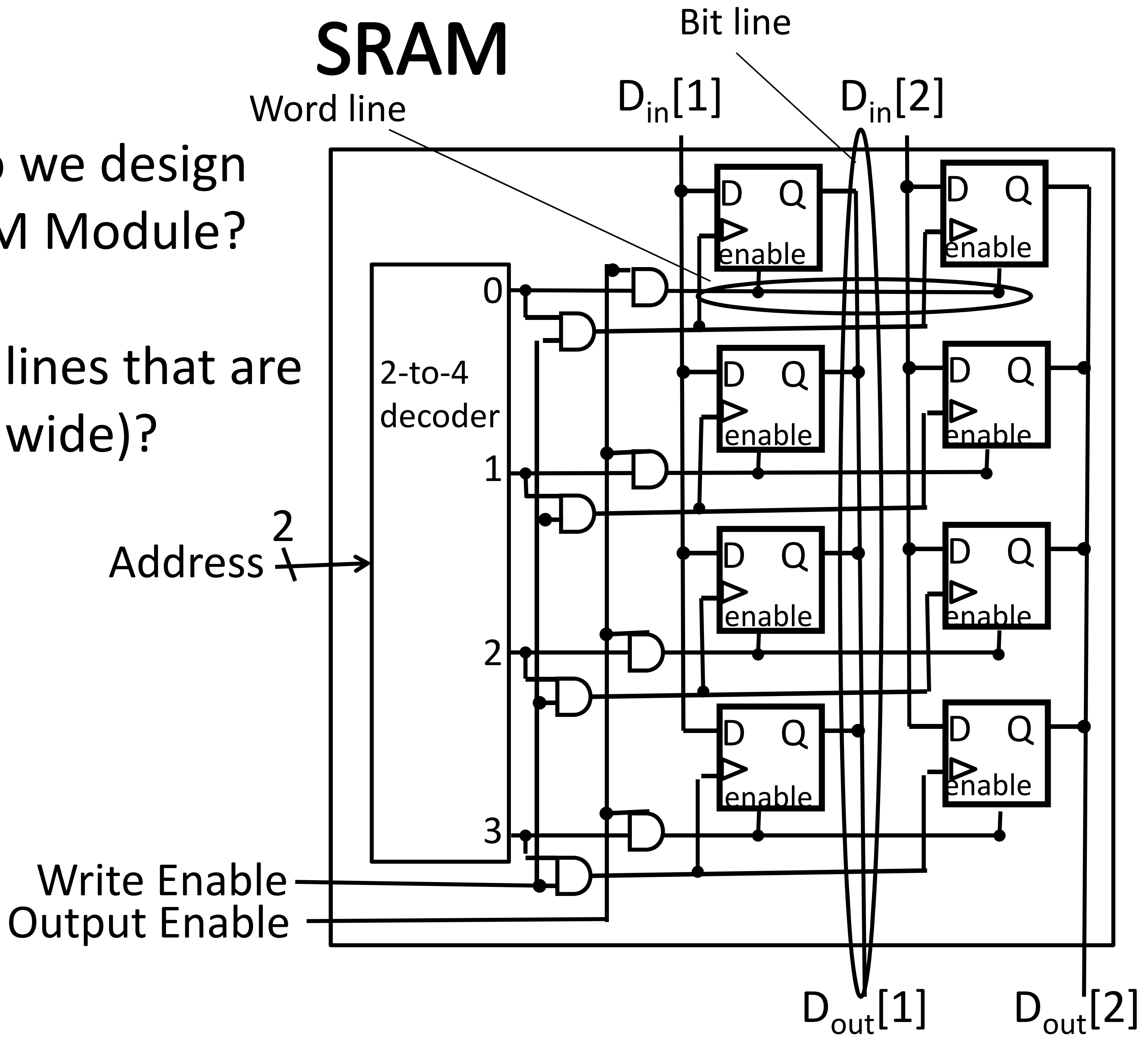
Write:

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

SRAM

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

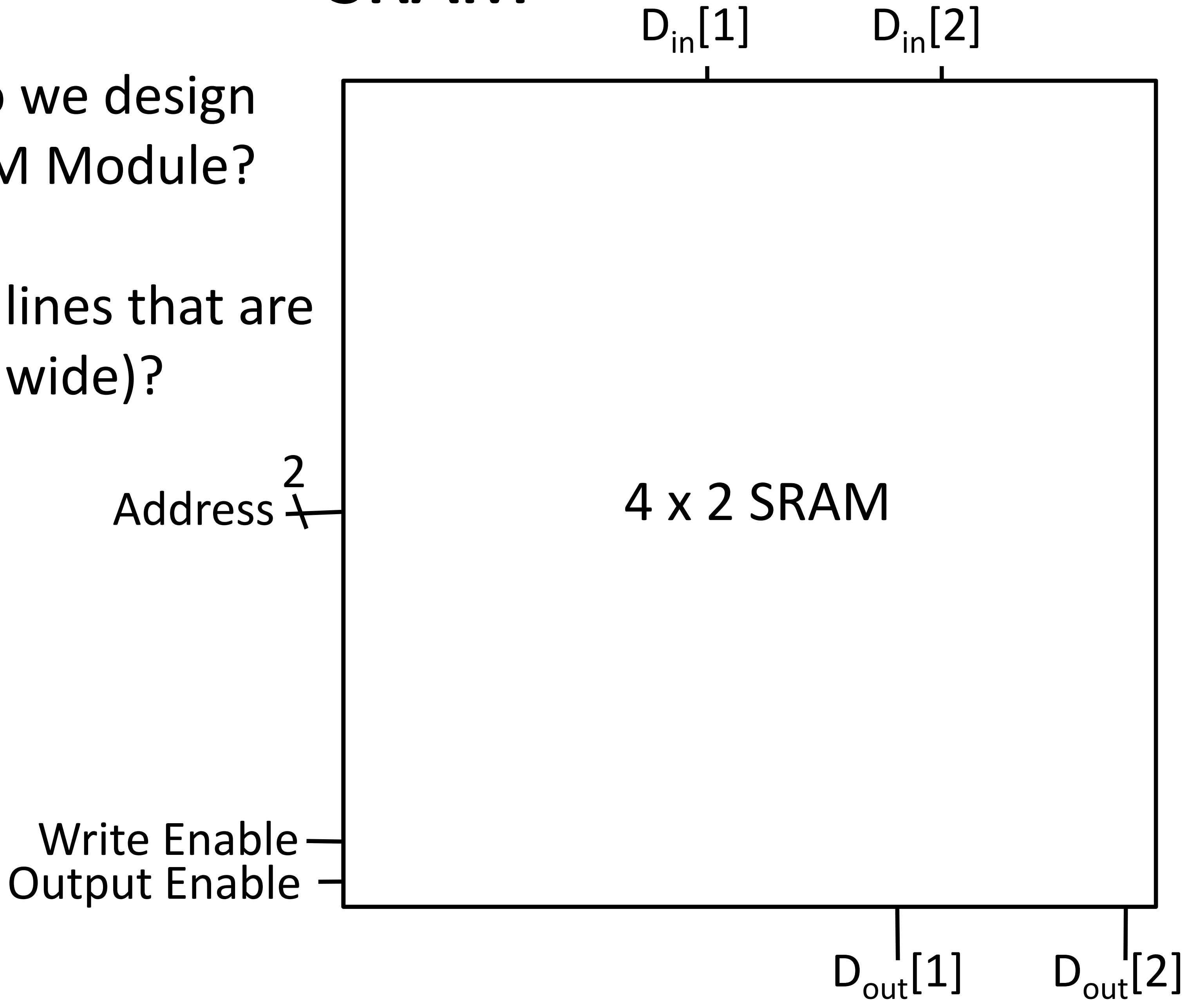
(i.e. 4 word lines that are each 2 bits wide)?



SRAM

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

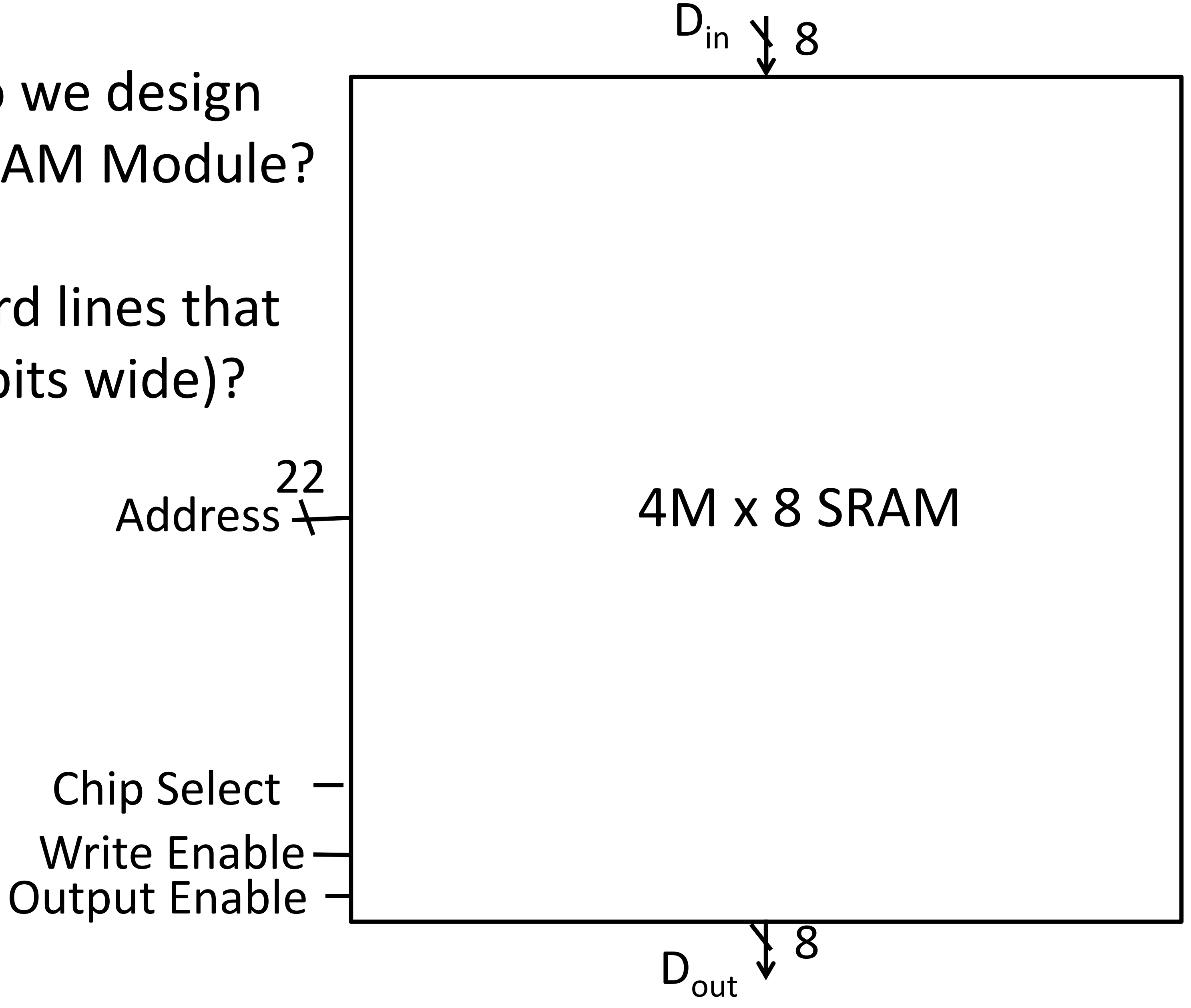
(i.e. 4 word lines that are each 2 bits wide)?



SRAM

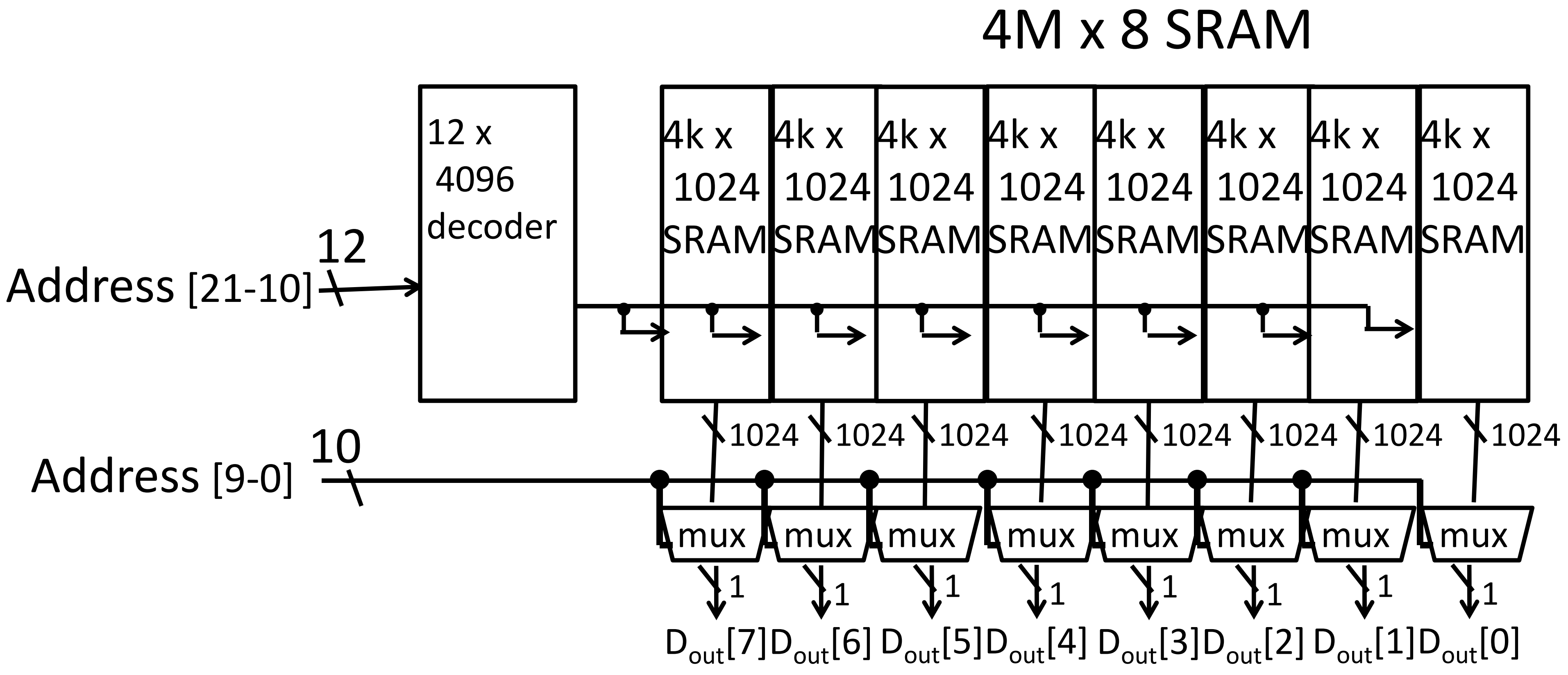
E.g. How do we design a ***4M x 8*** SRAM Module?

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



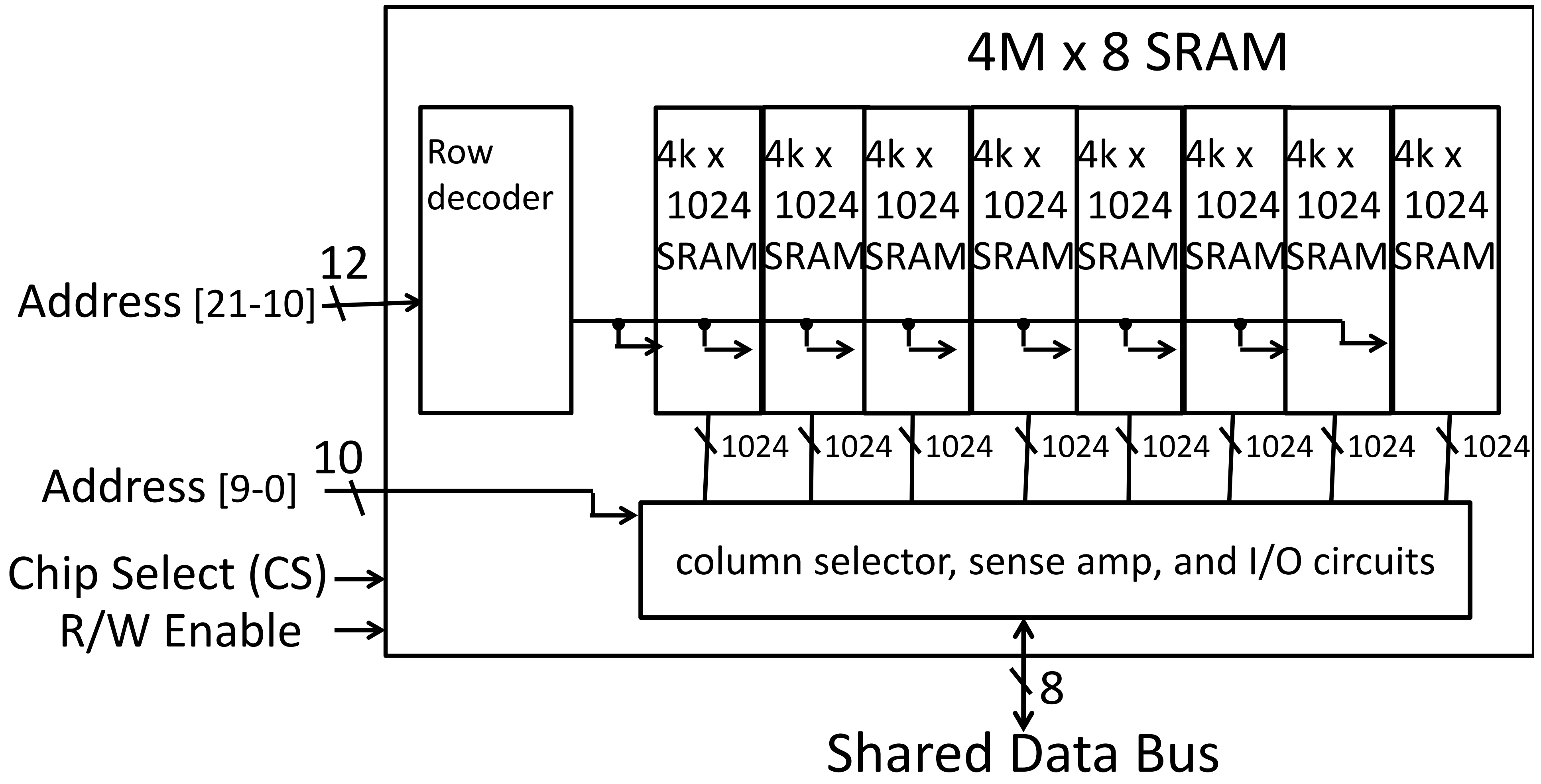
SRAM

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

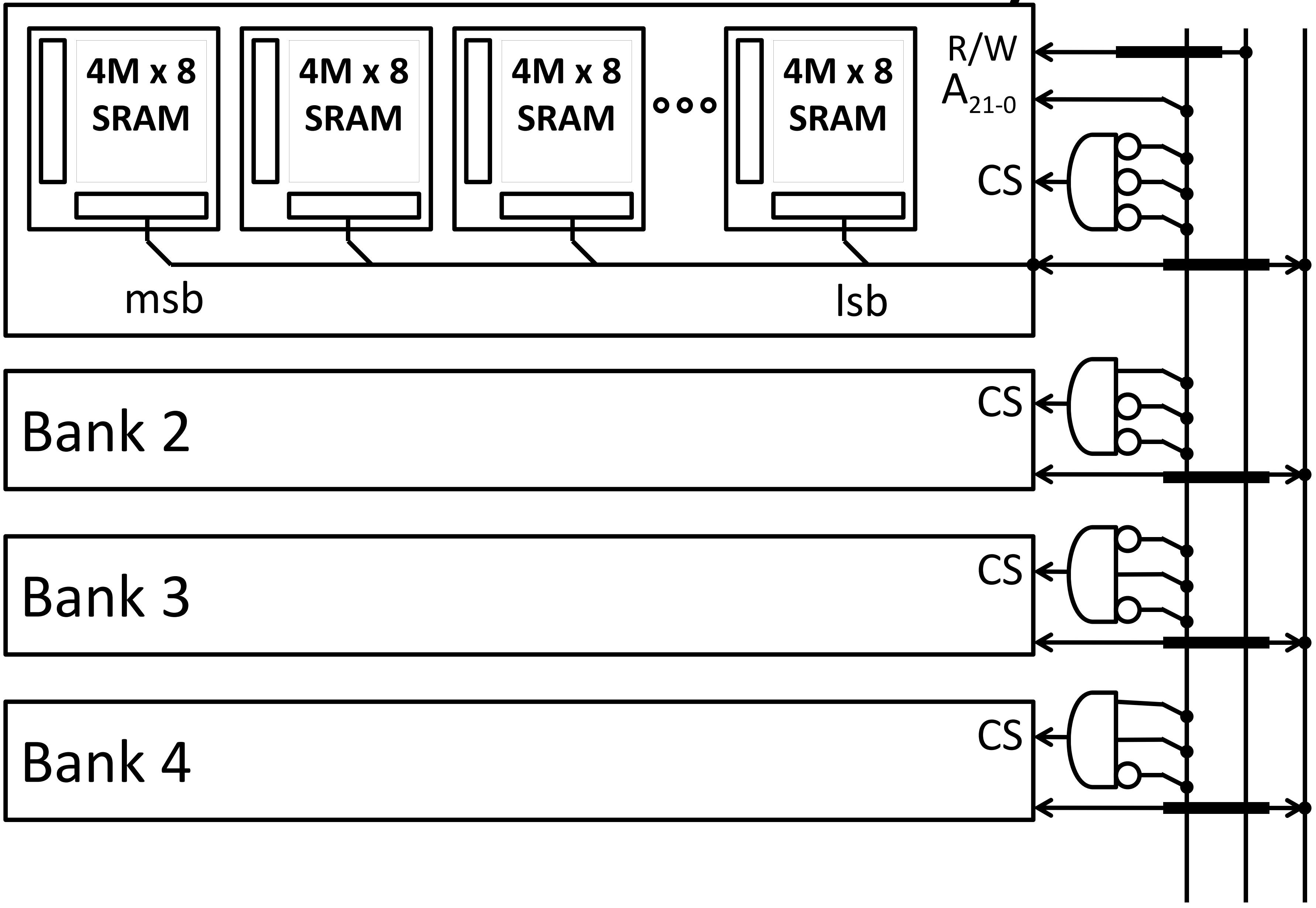


SRAM

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



SRAM Modules and Arrays



SRAM Summary

SRAM

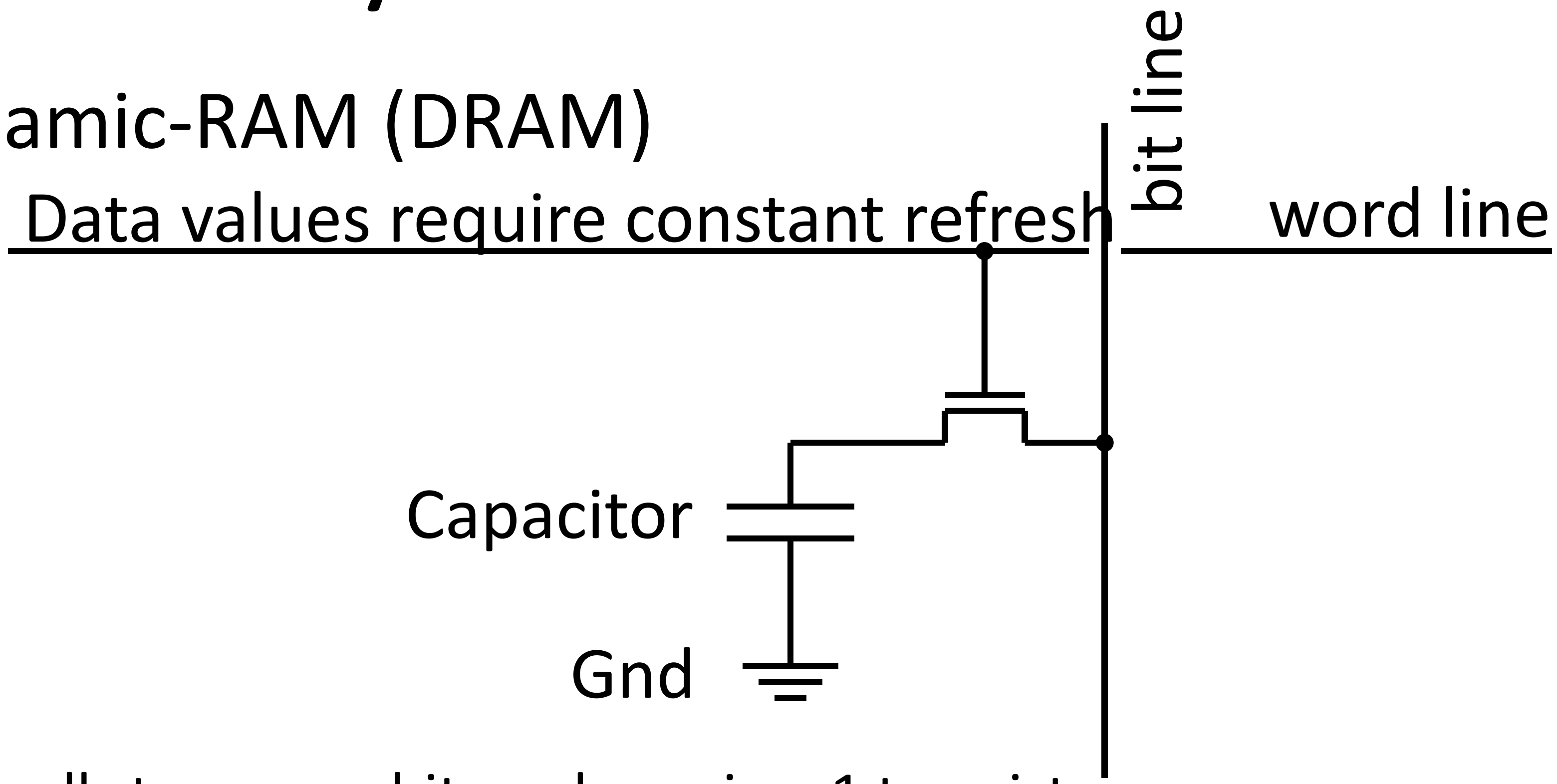
- A few transistors (~6) per cell
- Used for working memory (caches)

- But for even higher density…

Dynamic RAM: DRAM

Dynamic-RAM (DRAM)

- Data values require constant refresh

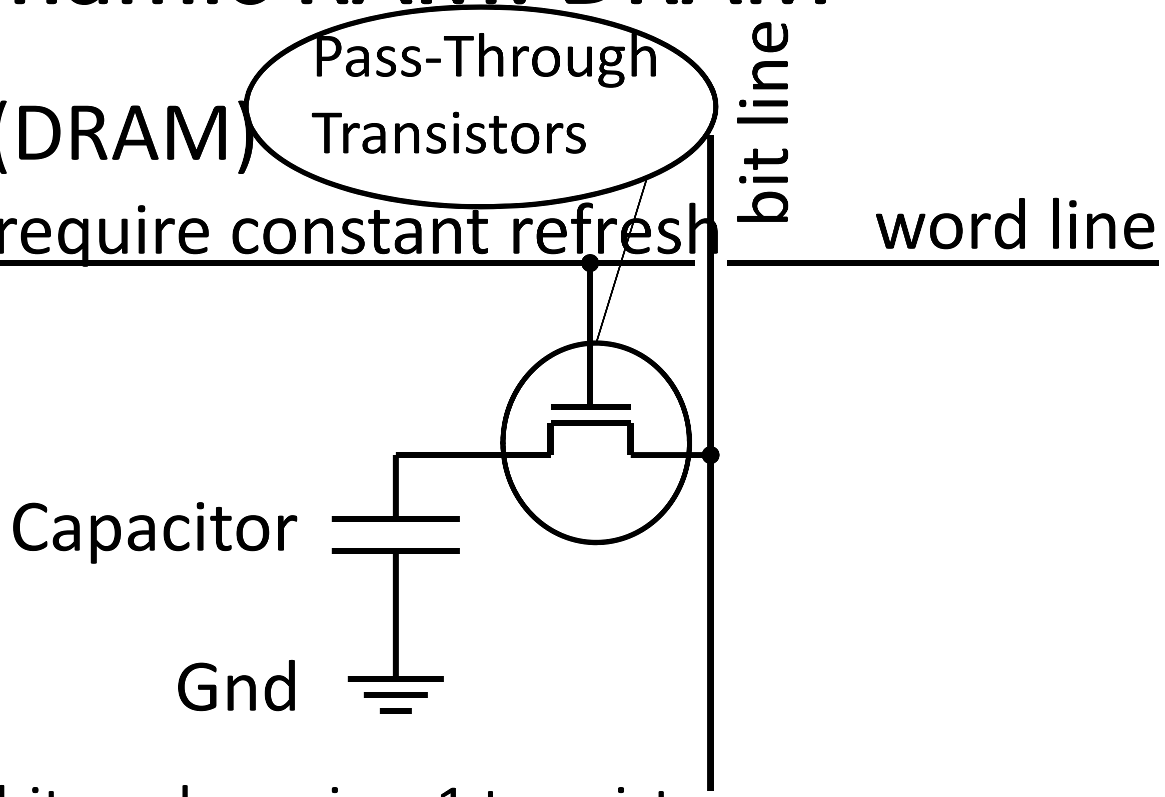


Each cell stores one bit, and requires 1 transistors

Dynamic RAM: DRAM

Dynamic-RAM (DRAM)

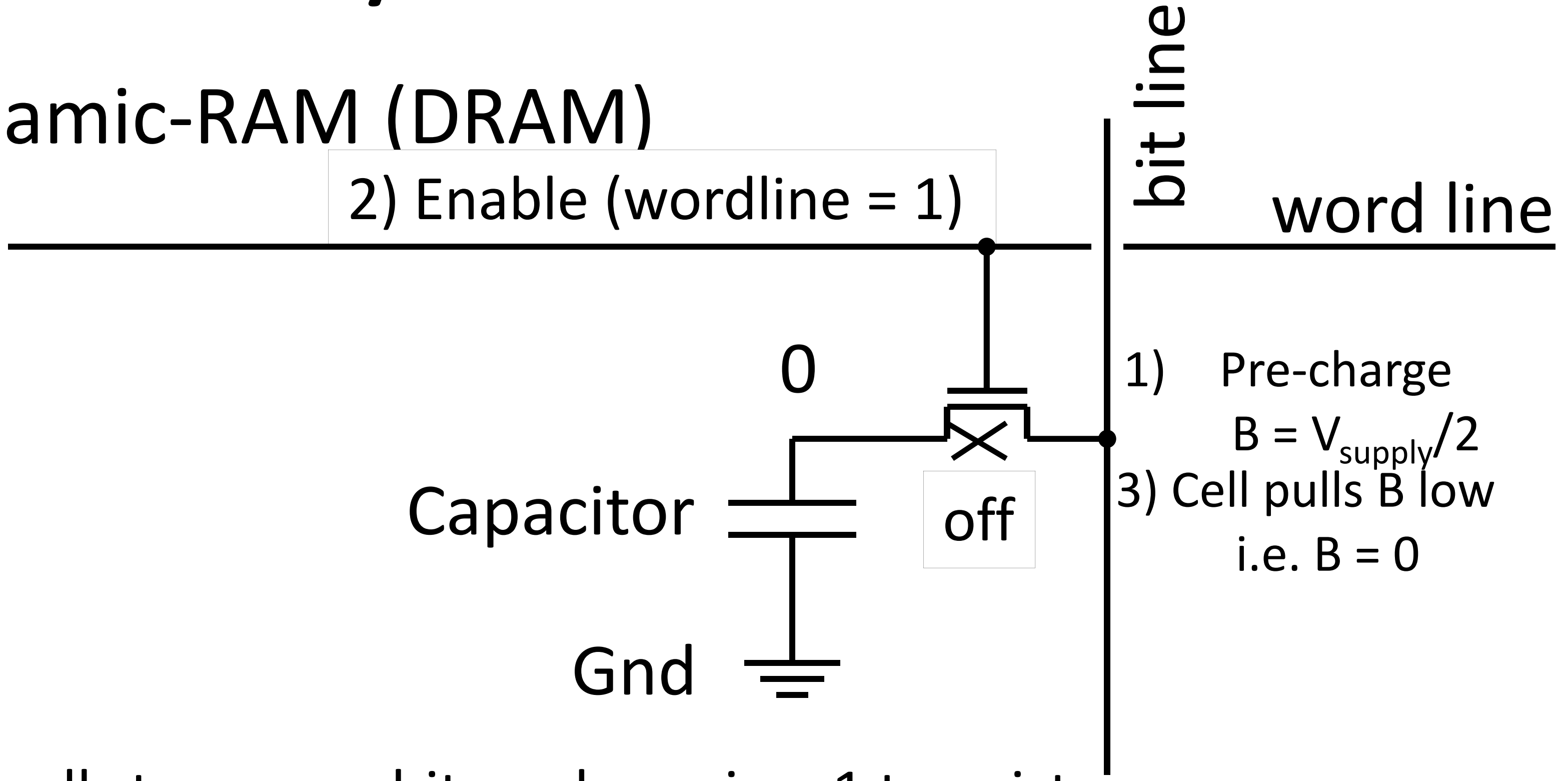
- Data values require constant refresh



Each cell stores one bit, and requires 1 transistors

Dynamic RAM: DRAM

Dynamic-RAM (DRAM)



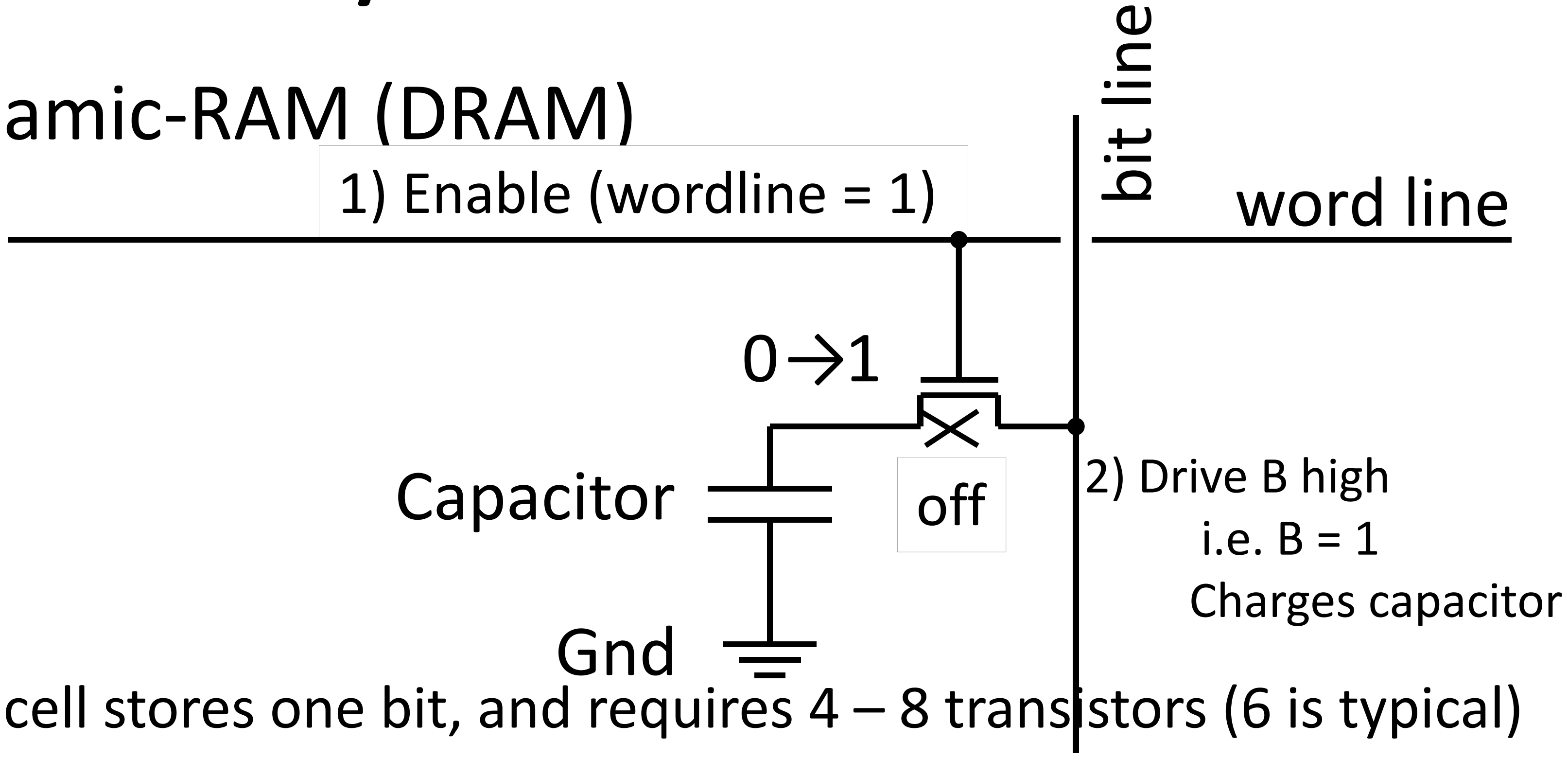
Each cell stores one bit, and requires 1 transistors

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 4 – 8 transistors (6 is typical)

Read:

- pre-charge B and $\overline{\mathrm{B}}$ to $V_{supply}/2$
- pull word line high
- cell pulls B or $\overline{\mathrm{B}}$ 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!