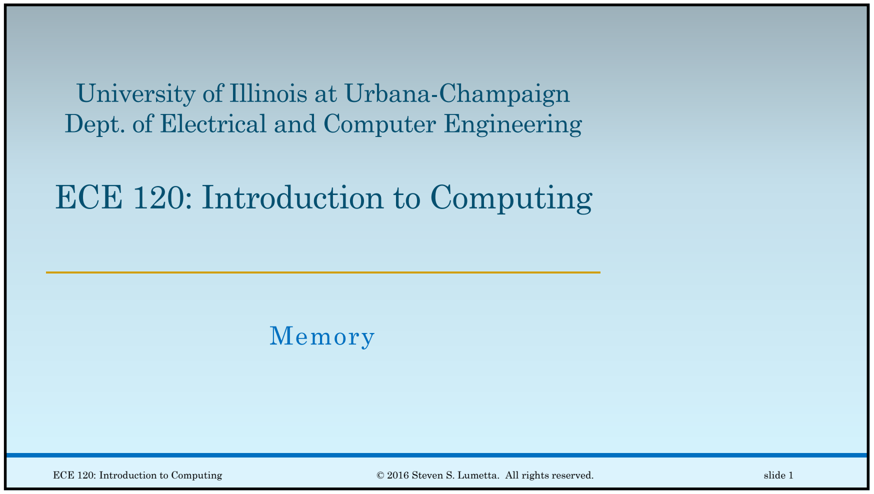
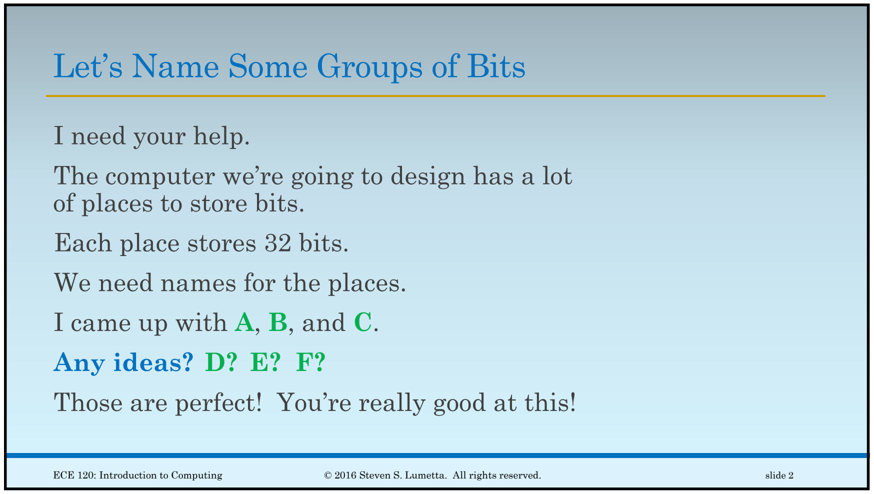


University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Memory



Let's Name Some Groups of Bits

I need your help.

The computer we're going to design has a lot of places to store bits.

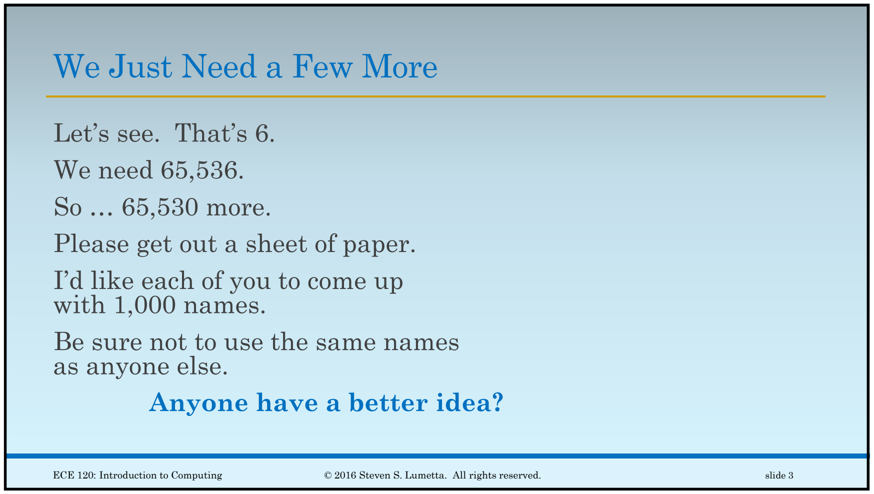
Each place stores 32 bits.

We need names for the places.

I came up with **A**, **B**, and **C**.

Any ideas? D? E? F?

Those are perfect! You're really good at this!



We Just Need a Few More

Let's see. That's 6.

We need 65,536.

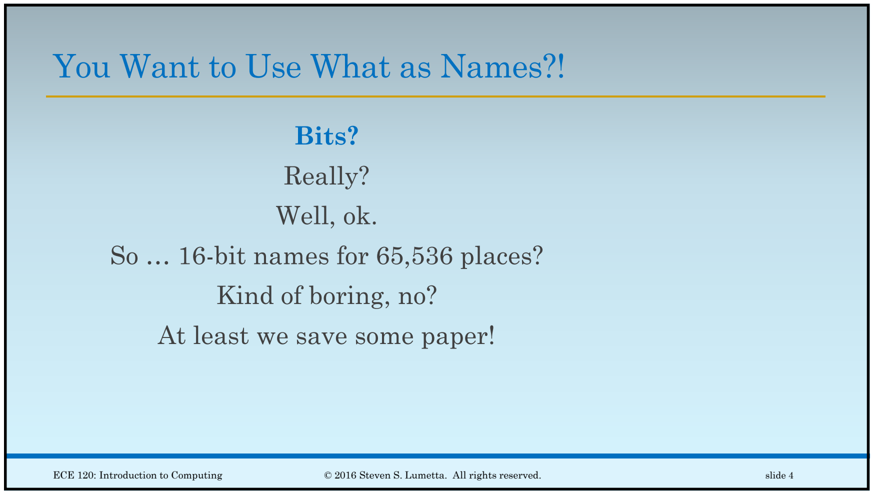
So … 65,530 more.

Please get out a sheet of paper.

I'd like each of you to come up with 1,000 names.

Be sure not to use the same names as anyone else.

Anyone have a better idea?



You Want to Use What as Names?!

Bits?

Really?

Well, ok.

So … 16-bit names for 65,536 places?

Kind of boring, no?

At least we save some paper!

Let's Build a Circuit to Manage Our Bits

If we use bits for names,

- we can probably **build a circuit**
- that lets us **read and write the bits** stored in each place.

Let's call one of our "names" an **address**.

So we have 65,536 = 2^{16} **addresses**.

At each address, we have 32 bits, which we call the **addressability**.

Protocol for Reading and Writing Bits

When we want to **read** the bits at an address:

- Tell the circuit the address we want, **ADDR**
- Then wait for bits to come out. **DATA-OUT**

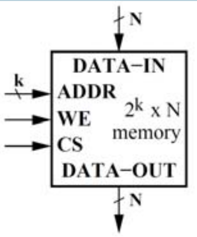
When we want to **write** bits to an address:

- Tell the circuit the address we want, **ADDR**
- And give the circuit the new bits. **DATA-IN**

And we need to tell the circuit whether we want to read or write (write enable). **WE**

A Symbol for Our Memory Circuit

Here's how we might draw the interface to our circuit.



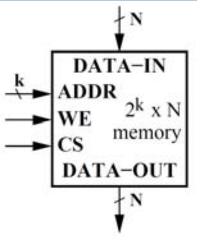
Let's call it a **memory**.

The memory shown has

- **2^k addresses**,
- a **k-bit ADDR** input to specify an address,
- **N-bit addressability**,
- **N-bit** inputs and outputs for data, and
- a **WE** (write enable) input.

Memory Chips Often Have a Chip Select Signal

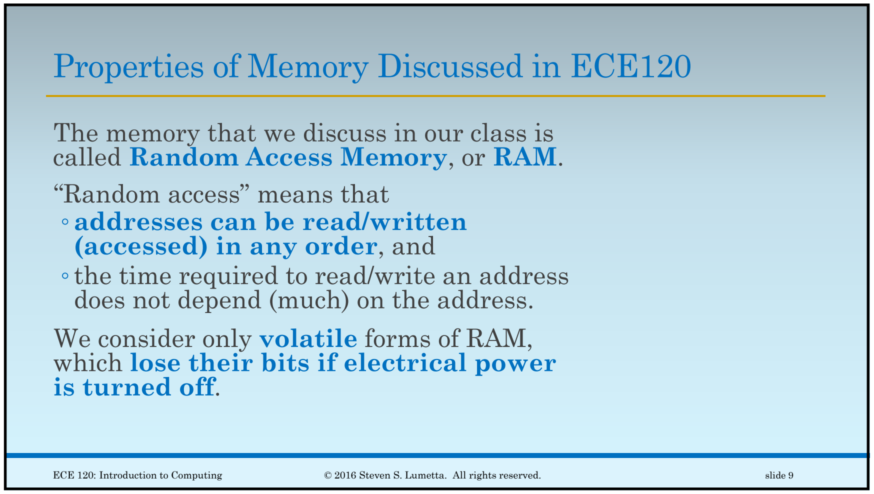
CS means "chip select."



If **CS = 1**, the memory reads or write (as specified by **WE**: **WE = 1** for a write, and **WE = 0** for a read).

If **CS = 0**, the memory does nothing.

CS is used to choose amongst multiple chips.



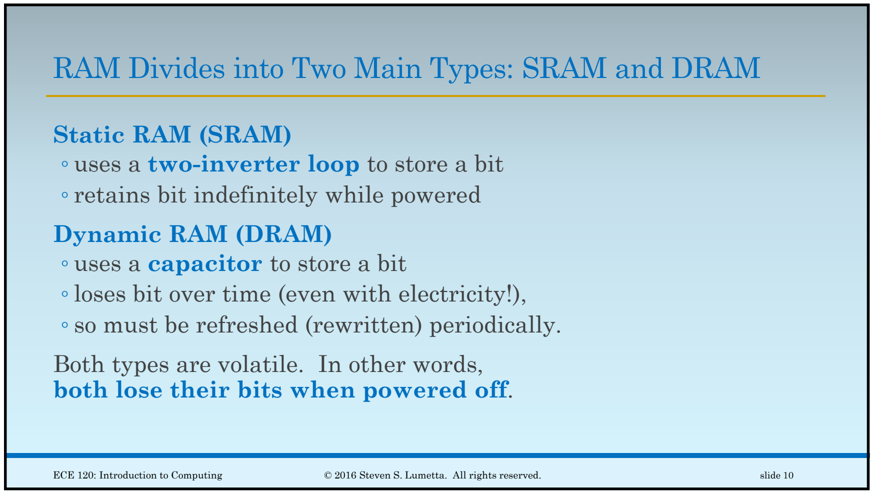
Properties of Memory Discussed in ECE120

The memory that we discuss in our class is called **Random Access Memory**, or **RAM**.

“Random access” means that

- **addresses can be read/written (accessed) in any order**, and
- the time required to read/write an address does not depend (much) on the address.

We consider only **volatile** forms of RAM, which **lose their bits if electrical power is turned off**.



RAM Divides into Two Main Types: SRAM and DRAM

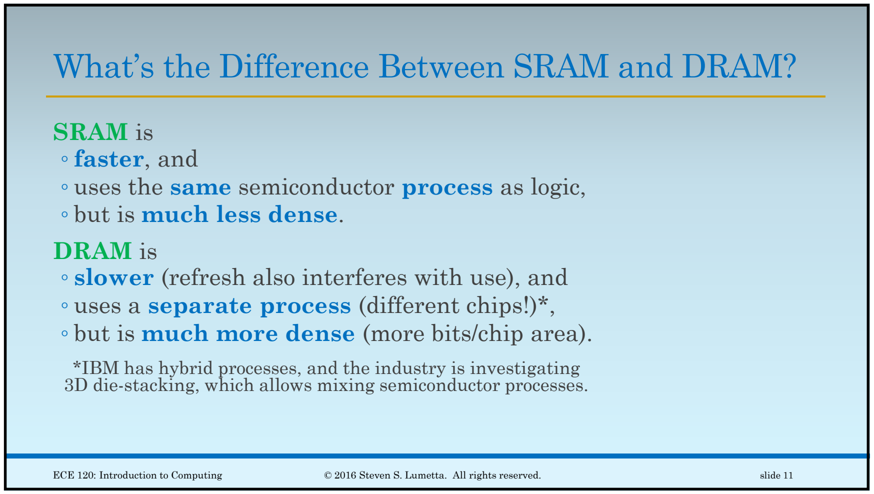
Static RAM (SRAM)

- uses a **two-inverter loop** to store a bit
- retains bit indefinitely while powered

Dynamic RAM (DRAM)

- uses a **capacitor** to store a bit
- loses bit over time (even with electricity!),
- so must be refreshed (rewritten) periodically.

Both types are volatile. In other words, **both lose their bits when powered off**.



What’s the Difference Between SRAM and DRAM?

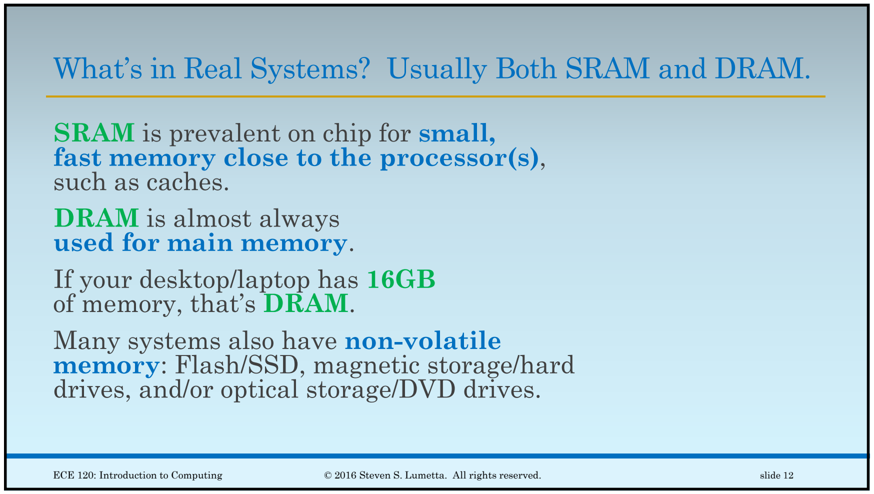
SRAM is

- **faster**, and
- uses the **same** semiconductor **process** as logic,
- but is **much less dense**.

DRAM is

- **slower** (refresh also interferes with use), and
- uses a **separate process** (different chips!)*,
- but is **much more dense** (more bits/chip area).

*IBM has hybrid processes, and the industry is investigating 3D die-stacking, which allows mixing semiconductor processes.



What’s in Real Systems? Usually Both SRAM and DRAM.

SRAM is prevalent on chip for **small, fast memory close to the processor(s)**, such as caches.

DRAM is almost always **used for main memory**.

If your desktop/laptop has **16GB** of memory, that’s **DRAM**.

Many systems also have **non-volatile memory**: Flash/SSD, magnetic storage/hard drives, and/or optical storage/DVD drives.

What Do You Need to Know?

How to use memory: the interface that we developed a few slides back.

The various **terms** that we just introduced.

A little bit about how memories are built (**SRAM and DRAM cells**, **use of decoders to select cells**, **coincident selection**).

How to build bigger memories out of smaller ones (bigger can mean more addresses or wider addressability, or both).

What Don’t You Need to Know?

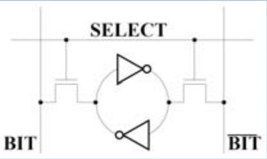
The real circuits that perform reads and writes through the bit lines. These analog circuits are beyond ECE120.*

Details of DRAM operation (see Section 3.6.4* for a brief introduction).

*In next week’s discussion, you’ll see some digital approximations to these circuits with similar logical behavior.

The SRAM Cell Stores One Bit Using a Dual-Inverter Loop

Here’s an **SRAM** cell.



You probably recognize the part in the middle, which **stores the bit**.

Two n-type MOSFETs connect the two inverters to the bit lines (**BIT** and **BIT’**).

When **SELECT = 1**, the bit is connected to the bit lines.

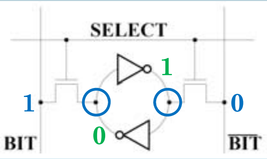
When **SELECT = 0**, this cell is disconnected.

* *

Bit Lines are Held at Fixed Voltages to Write a Bit

To write the bit,

- bit lines are held at opposite values,
- forcing the inverters to store the bit.



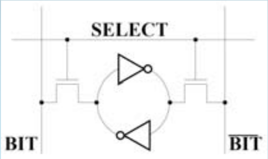
This design does something we told you never to do... **wire together outputs!**

Changing a bit means short circuits, so these analog systems must be designed carefully!

* *

Inverters Drive the Bit onto the Bit Lines for a Read

To read the bit,
- bit lines are left floating, and
- inverters drive the stored bit onto the bit lines.

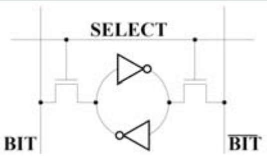


To speed up reads,
- bit lines are pre-charged to **$V_{dd}/2$**, and
- sense amplifiers (analog devices) amplify any changes in voltage between bit lines to 0/1.

* *

This SRAM Cell Circuit is Commonly Used

How many transistors does this cell contain?
- **Two shown**, and
- **two per inverter**.

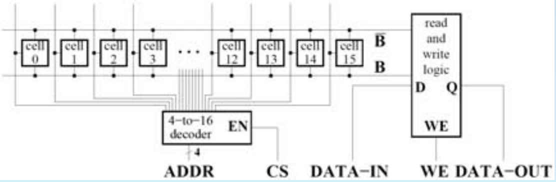


This design is called a **6T cell**.

It balances speed and good reliability with small size, and is one of the most common.

A Memory Requires Many Memory Cells

The circuit below is a **16×1 memory** (16 addresses, 1-bit addressability) marked with the names from our symbolic version of memory.

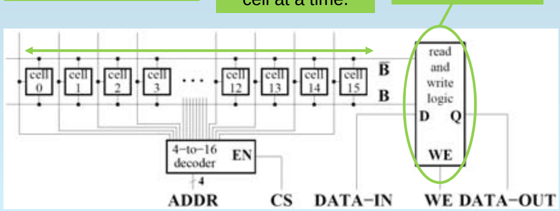


Memory Cells Share Bit Lines

Notice that the cells are rotated: SELECT lines are vertical, and bit lines are horizontal.

All of the cells share bit lines; we access only one cell at a time.

Sense amps and other analog circuits are here.

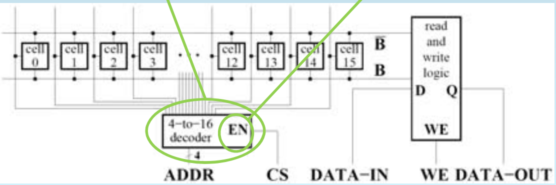


A Decoder is Used to Control Each Cell's SELECT Line

A 4-to-16 decoder drives all 16 cells' SELECT lines.

At most one cell's SELECT line is high at any time.

The decoder's EN (enable) input is based on CS (chip select).

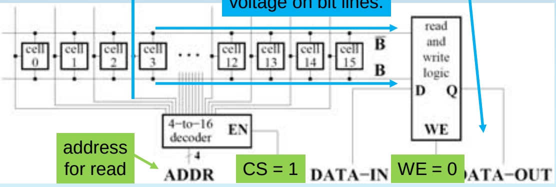


Performing a Read: CS = 1, WE = 0, set ADDR …

"ADDR" cell's SELECT is activated (set to 1) by decoder.

Stored bit changes voltage on bit lines.

Bit becomes available on DATA-OUT



address for read

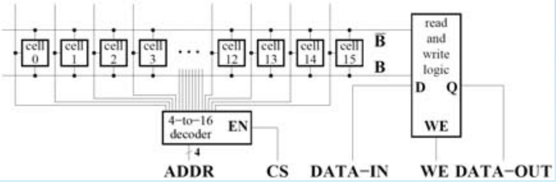
CS = 1

WE = 0

Memory is not Clocked

Did you notice that there's no clock?

Memory operates asynchronously with respect to other logic.



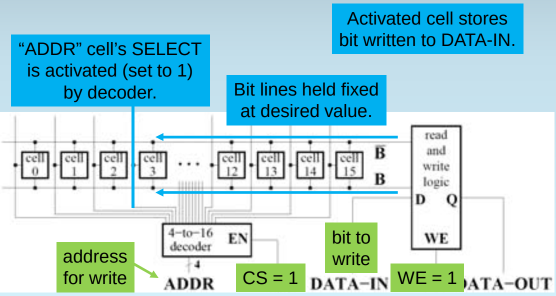
How Do We Know When a Read has Finished?

Two approaches:

- the memory designer specifies a minimum wait time (in the datasheet) for a **read** to complete, or
- the memory raises an output (called **R** in Patt and Patel) to indicate that it is **Ready** for another operation.

You may also hear about Synchronous DRAM (SDRAM), which clocks the interface between memory and the processor to speed up moving bits around. The cells are still unclocked.

Performing a Write: CS = 1, WE = 1, set ADDR …



"ADDR" cell's SELECT is activated (set to 1) by decoder.

Activated cell stores bit written to DATA-IN.

Bit lines held fixed at desired value.

address for write

CS = 1

bit to write

WE = 1

How Do We Know When a Write has Finished?

The same two approaches:

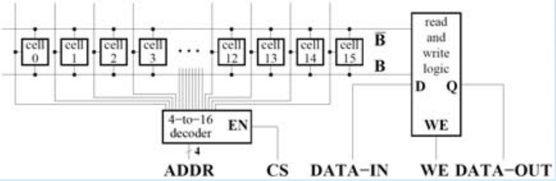
- the memory designer specifies a minimum wait time (in the datasheet) for a **write** to complete, or
- the memory raises an output (called **R** in Patt and Patel) to indicate that it is **Ready** for another operation.

Bit Slices Balance Speed Against Size

This 16×1 memory is a **bit slice**.

The number of bits in a real bit slice is larger.

They balance speed (few bits) against size (few copies of the read/write logic).

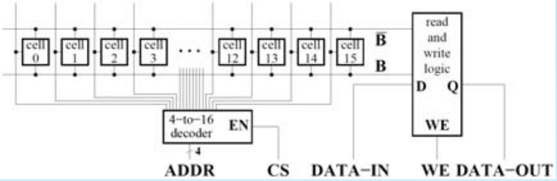


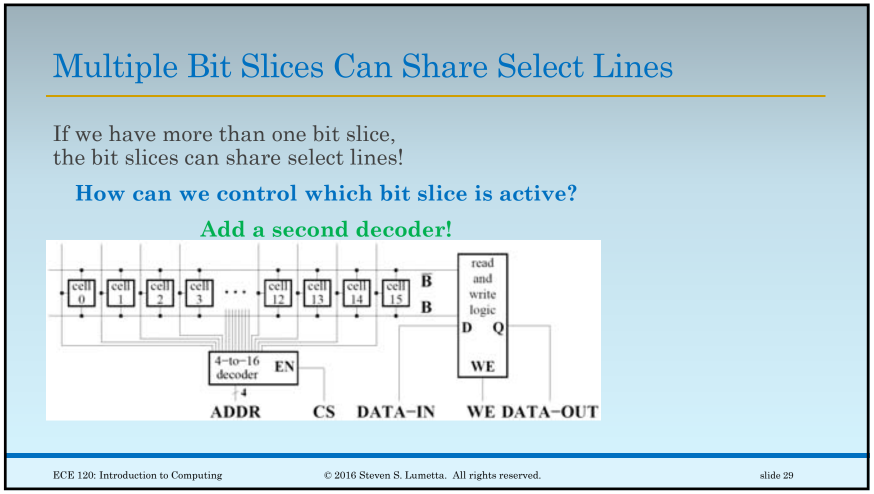
Decoder Also Requires Many Gates

Another major cost for long bit slices: the decoder.

How many gates in an N-to-2^N decoder?

Around 2^N (one AND gate per output).



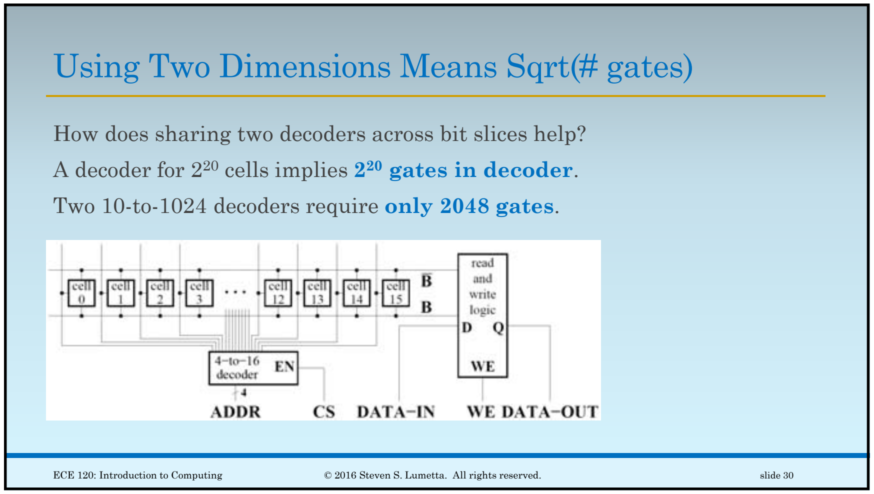


Multiple Bit Slices Can Share Select Lines

If we have more than one bit slice,
the bit slices can share select lines!

How can we control which bit slice is active?

Add a second decoder!

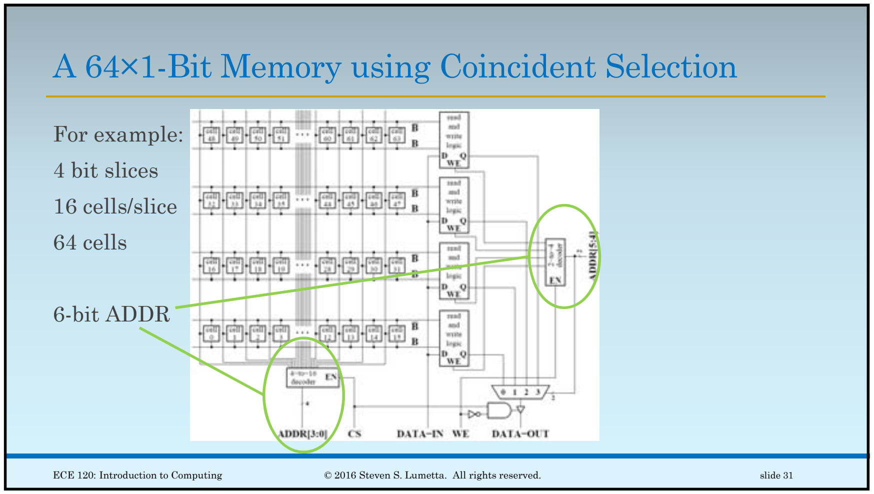


Using Two Dimensions Means Sqrt(# gates)

How does sharing two decoders across bit slices help?

A decoder for 2^{20} cells implies **2^{20} gates in decoder**.

Two 10-to-1024 decoders require **only 2048 gates**.



A 64×1-Bit Memory using Coincident Selection

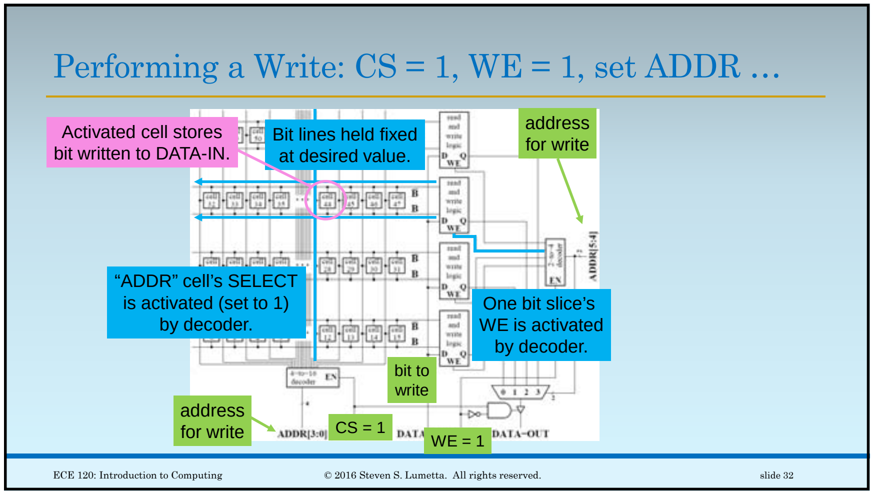
For example:

4 bit slices

16 cells/slice

64 cells

6-bit ADDR



Performing a Write: CS = 1, WE = 1, set ADDR …

Activated cell stores bit written to DATA-IN.

Bit lines held fixed at desired value.

address for write

"ADDR" cell's SELECT is activated (set to 1) by decoder.

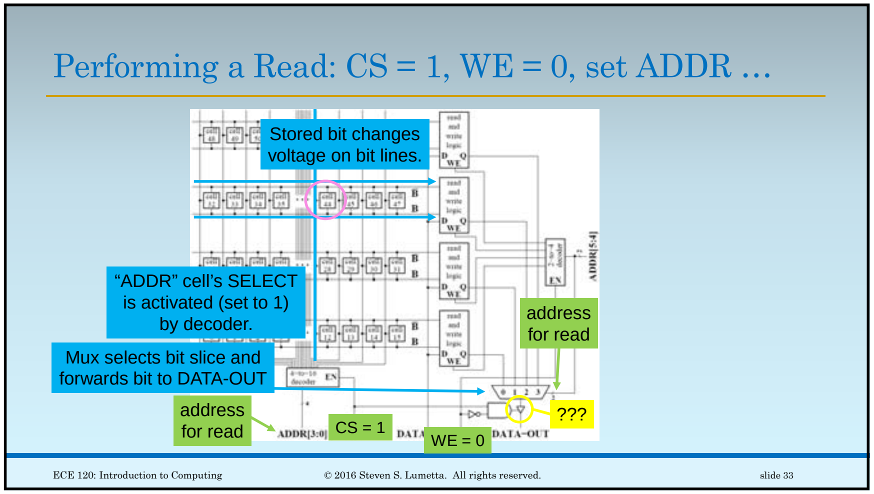
One bit slice's WE is activated by decoder.

bit to write

address for write

CS = 1

WE = 1



Performing a Read: CS = 1, WE = 0, set ADDR …

Stored bit changes voltage on bit lines.

“ADDR” cell’s SELECT is activated (set to 1) by decoder.

Mux selects bit slice and forwards bit to DATA-OUT

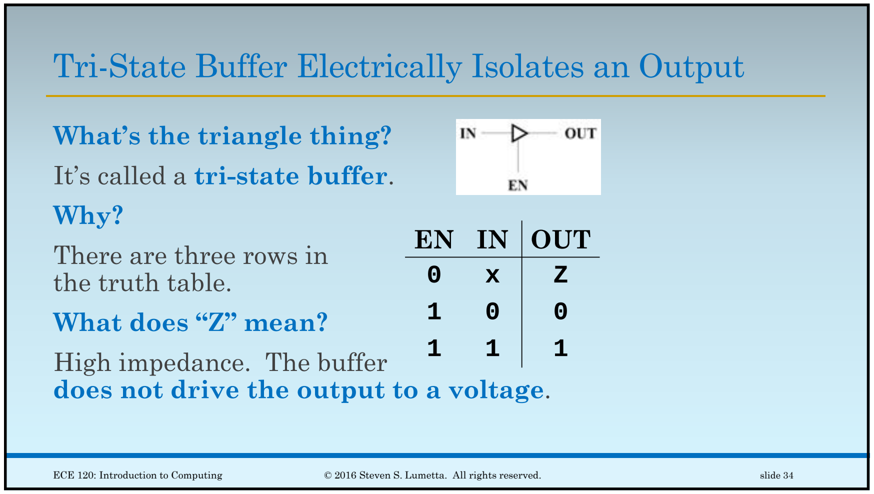
address for read

address for read

CS = 1

WE = 0

???



Tri-State Buffer Electrically Isolates an Output

What’s the triangle thing?

It’s called a **tri-state buffer**.

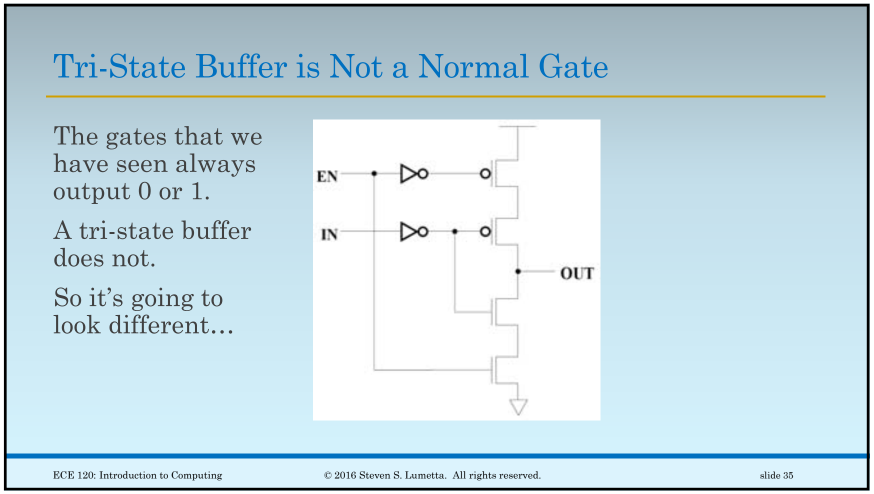
Why?

There are three rows in the truth table.

EN	IN	OUT
0	x	Z
1	0	0
1	1	1

What does “Z” mean?

High impedance. The buffer **does not drive the output to a voltage**.

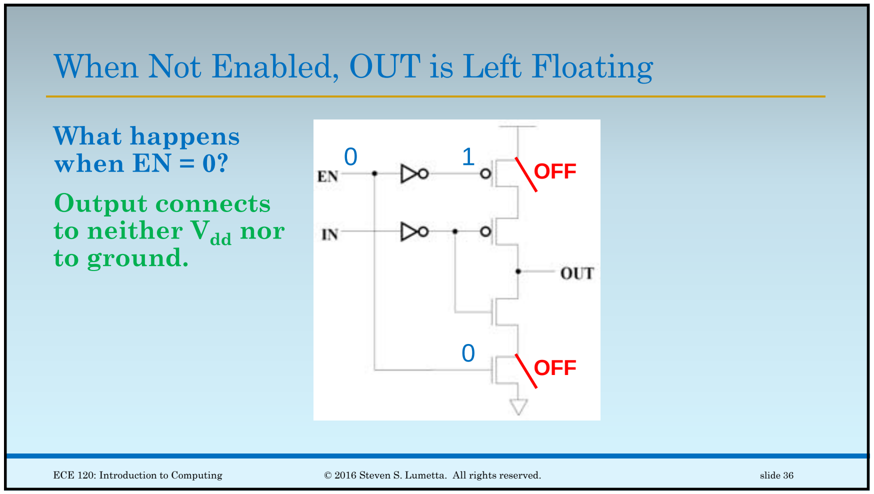


Tri-State Buffer is Not a Normal Gate

The gates that we have seen always output 0 or 1.

A tri-state buffer does not.

So it’s going to look different…



When Not Enabled, OUT is Left Floating

What happens when EN = 0?

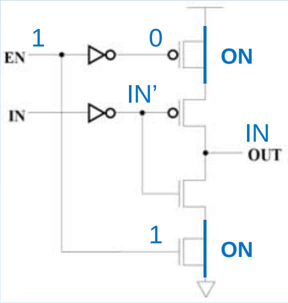
Output connects to neither V_{dd} nor to ground.

When Not Enabled, IN is Copied to OUT

What happens when EN = 1?

Reduces to two inverters!

IN is copied to OUT.



Tri-State Buffers Allow Us to Wire Outputs Together

What good are tri-state buffers?*

One example: distributed mux.

Say that we want to choose amongst four groups of **N** signals.

We can of course use a mux:

- bring **4N** wires together,
- select one group of **N**, and
- distribute the choice with **N** more wires.

*They're becoming uncommon in modern designs, as they can make verification harder and limit performance.

Tri-State Buffers Can Implement a Distributed Mux

Using tri-state buffers, we can instead

- gate each output with tri-state buffers (**4N** buffers, **4** wires to carry **EN** signals),
- connect all outputs with **N** wires.

We call these N wires a bus.

The **4 EN** wires ensure that only one of the four groups of outputs is written to the **N** wires.

In other words, they act as a **distributed mux**.

The LC-3 computer datapath in Patt & Patel uses a bus to move data from component to component.

Tri-State Buffers also Allow Us to Reuse Wires

For our memory design,

- **DATA-OUT** is gated with tri-state buffers,
- so these lines float whenever **CS = 0** or **WE = 1**.

In real memory chips, the same pins (wires) can be used for **DATA-IN** and **DATA-OUT**.

For **writes**, the pins accept bits to store.

For **reads**, the tri-state buffers write the bits from the memory cells onto the pins.

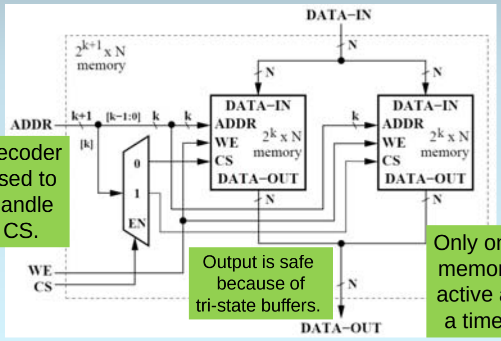
Building a Memory with More Addresses

Given two **$2^k \times N$-bit memories**, how can we construct a **$2^{k+1} \times N$-bit memory**?

That is, **twice as many addresses**?

Notice that each **$2^k \times N$-bit memory** contains **$2^k \times N$** memory cells, so two such memories contain **$2^{k+1}N$** cells.

Building a Memory with More Addresses



Decoder used to handle CS.

Output is safe because of tri-state buffers.

Only one memory active at a time.

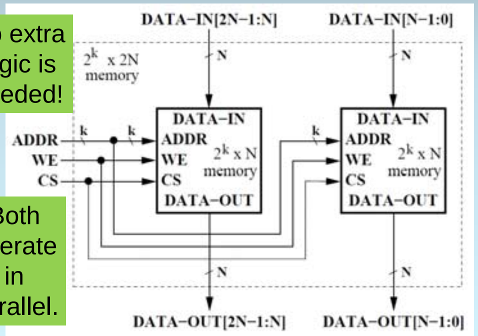
Building a Memory with Wider Addressability

Given two **$2^k \times N$-bit memories**, how can we construct a **$2^k \times (2N)$-bit memory**?

That is, **twice as many bits at each address**?

Notice again that each **$2^k \times N$-bit memory** contains **$2^k \times N$** memory cells, so two such memories contain **$2^k(2N)$** cells.

Building a Memory with Wider Addressability



No extra logic is needed!

Both operate in parallel.