

ECE120: Introduction to Computer Engineering

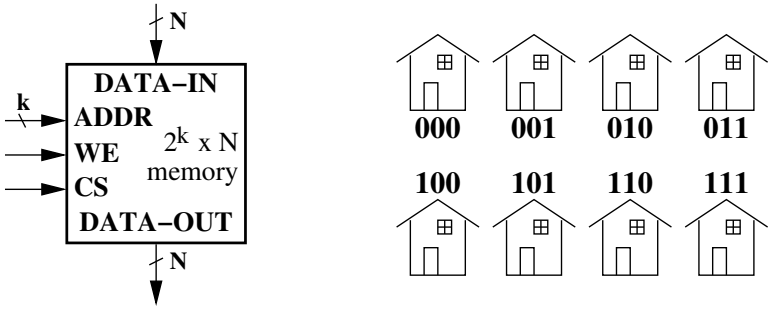
Notes Set 3.6 Random Access Memories

This set of notes describes random access memories (RAMs), providing slightly more detail than is available in the textbook. We begin with a discussion of the memory abstraction and the types of memory most commonly used in digital systems, then examine how one can build memories (static RAMs) using logic. We next introduce tri-state buffers as a way of simplifying ouput connections, and illustrate how memory chips can be combined to provide larger and wider memories. A more detailed description of dynamic RAMs finishes this set. *Sections marked with an asterisk are provided solely for your interest, but you probably need to learn this material in later classes.*

3.6.1 Memory

A computer **memory** is a group of storage elements and the logic necessary to move data in and out of the elements. The size of the elements in a memory—called the **addressability** of the memory—varies from a single binary digit, or **bit**, to a **byte** (8 bits) or more. Typically, we refer to data elements larger than a byte as **words**, but the size of a word depends on context.

Each element in a memory is assigned a unique name, called an **address**, that allows an external circuit to identify the particular element of interest. These addresses are not unlike the street addresses that you use when you send a letter. Unlike street addresses, however, memory addresses usually have little or no redundancy; each possible combination of bits in an address identifies a distinct set of bits in the memory. The figure on the right below illustrates the concept. Each house represents a storage element and is associated with a unique address.



The memories that we consider in this class have several properties in common. These memories support two operations: **write** places a word of data into an element, and **read** retrieves a copy of a word of data from an element. The memories are also **volatile**, which means that the data held by a memory are erased when electrical power is turned off or fails. **Non-volatile** forms of memory include magnetic and optical storage media such as DVDs, CD-ROMs, disks, and tapes, capacitive storage media such as Flash drives, and some programmable logic devices. Finally, the memories considered in this class are **random access memories (RAMs)**, which means that the time required to access an element in the memory is independent of the element being accessed. In contrast, **serial memories** such as magnetic tape require much less time to access data near the current location in the tape than data far away from the current location.

The figure on the left above shows a generic RAM structure. The memory contains 2^k elements of N bits each. A k-bit address input, $ADDR$, identifies the memory element of interest for any particular operation. The write enable input, WE, selects the operation to be performed: if WE is high, the operation is a write; if it is low, the operation is a read. Data to be written into an element are provided through N inputs at the top, and data read from an element appear on N outputs at the bottom. Finally, a **chip select** input, CS, functions as an enable control for the memory; when CS is low, the memory neither reads nor writes any location.

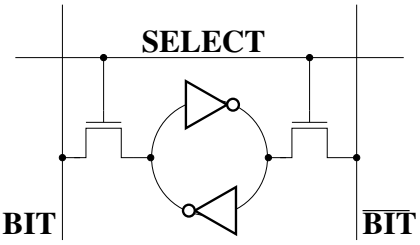
Random access memory further divides into two important types: **static RAM**, or **SRAM**, and **dynamic RAM**, or **DRAM**. SRAM employs active logic in the form of a two-inverter loop to maintain stored values.

DRAM uses a charged capacitor to store a bit; the charge drains over time and must be replaced, giving rise to the qualifier "dynamic." "Static" thus serves only to differentiate memories with active logic elements from those with capacitive elements. Both types are volatile, that is, both lose all data when the power supply is removed. We consider both SRAM and DRAM in this course, but the details of DRAM operation are beyond our scope.

3.6.2 Static Random Access Memory

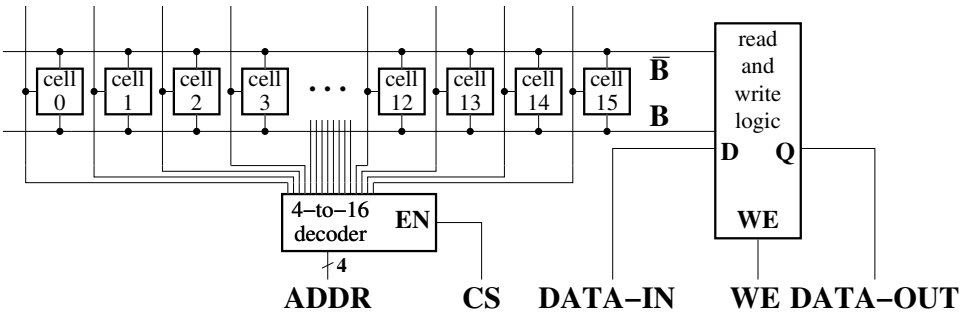
Static random access memory is used for high-speed applications such as processor caches and some embedded designs. As SRAM bit density—the number of bits in a given chip area—is significantly lower than DRAM bit density, most applications with less demanding speed requirements use DRAM. The main memory in most computers, for example, is DRAM, whereas the memory on the same chip as a processor is SRAM.[13] DRAM is also unavailable when recharging its capacitors, which can be a problem for applications with stringent real-time needs.

A diagram of an SRAM **cell** (a single bit) appears to the right. A dual-inverter loop stores the bit, and is connected to opposing BIT lines through transistors controlled by a $SELECT$ line. The cell works as follows. When $SELECT$ is high, the transistors connect the inverter loop to the bit lines. When writing a cell, the bit lines are held at opposite logic values, forcing the inverters to match the values on the lines and storing the value from the BIT input. When reading a cell, the bit lines are disconnected from other logic, allowing the inverters to drive the lines with their current outputs. The value stored previously is thus copied onto the BIT line as an output, and the opposite value is placed on the $\overline{BIT}$ line. When $SELECT$ is low, the transistors disconnect the inverters from the bit lines, and the cell holds its current value until $SELECT$ goes high again.



The actual operation of an SRAM cell is more complicated than we have described. For example, when writing a bit, the BIT lines can temporarily connect high voltage to ground (a short). The circuit must be designed carefully to minimize the power consumed during this process. When reading a bit, the BIT lines are pre-charged halfway between high-voltage and ground, and analog devices called sense amplifiers are used to detect the voltage changes on the BIT lines (driven by the inverter loop) as quickly as possible. These analog design issues are outside of the scope of our class.

A number of cells are combined into a **bit slice**, as shown to the right. The labels along the bottom of the figure are external inputs to the bit slice, and match the labels for the abstract memory discussed earlier. The bit slice in the figure can be thought of as a 16-address, 1-bit-addressable memory ($2^4 \times 1$b).

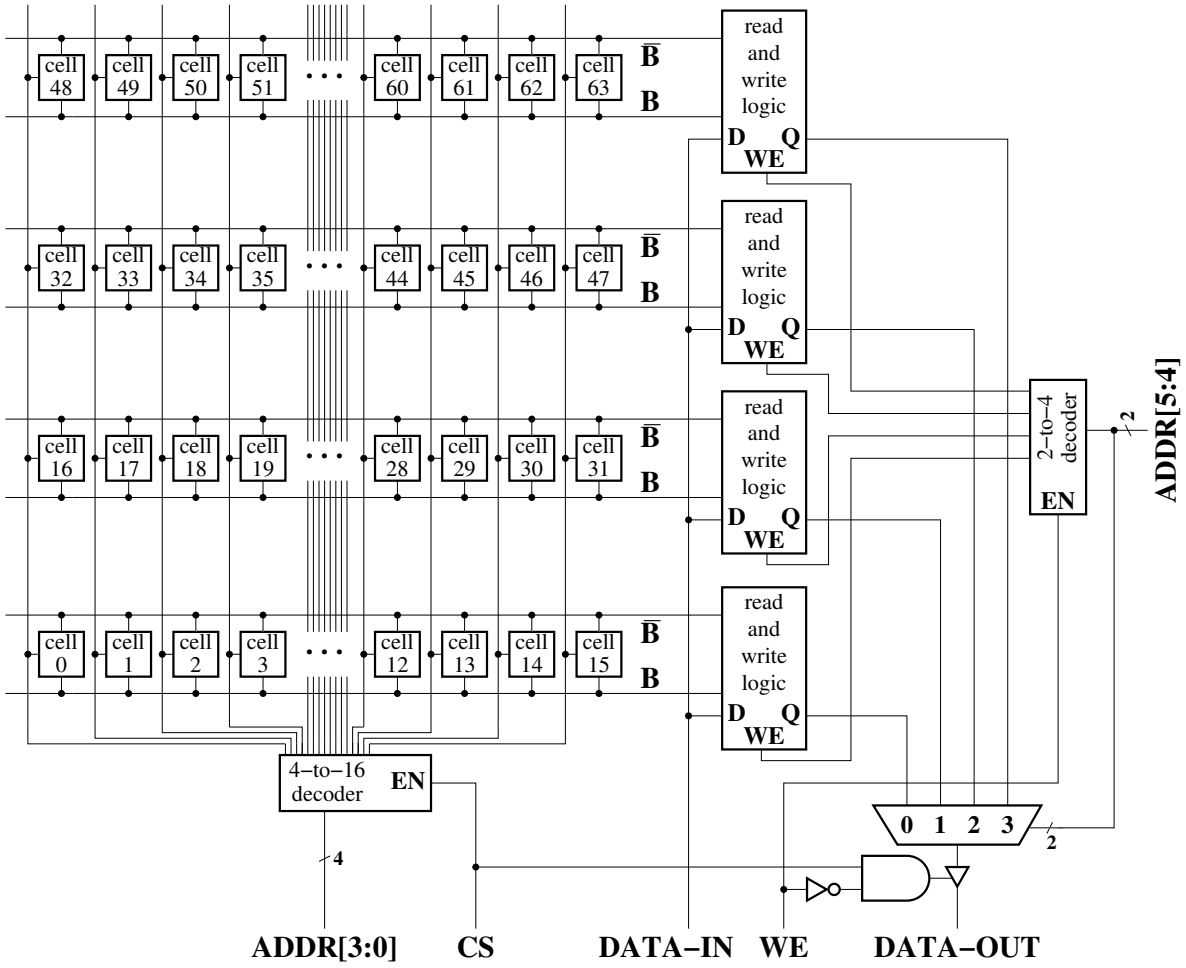


The cells in a bit slice share bit lines and analog read and write logic, which appears to the right in the figure. Based on the $ADDR$ input, a decoder sets one cell's $SELECT$ line high to enable a read or write operation to the cell. The chip select input CS drives the enable input of the decoder, so none of the memory cells is active when chip select is low ($CS = 0$), and exactly one of the memory cells is active when chip select is high ($CS = 1$). Actual bit slices can contain many more cells than are shown in the figure—more cells means less extra logic per cell, but slower memory, since longer wires have higher capacitance.

[13] Chips combining both DRAM and processor logic are available, and are used by some processor manufacturers (such as IBM). Research is underway to couple such logic types more efficiently by building 3D stacks of chips.

A read operation is performed as follows. We set $CS = 1$ and $WE = 0$, and place the address of the cell to be read on the $ADDR$ input. The decoder outputs a 1 on the appropriate cell's $SELECT$ line, and the read logic reads the bit from the cell and delivers it to its Q output, which is then available on the bit slice's $DATA\text{-}OUT$ output.

For a write operation, we set $CS = 1$ and $WE = 1$. We again place the address of the cell to be written on the $ADDR$ input and set the value of the bit slice's $DATA\text{-}IN$ input to the value to be written into the memory cell. When the decoder activates the cell's $SELECT$ line, the write logic writes the new value from its D input into the memory cell. Later reads from that cell then produce the new value.



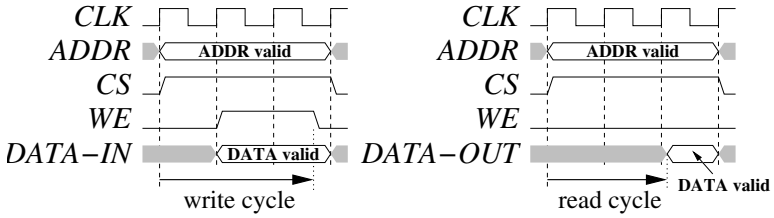
The outputs of the cell selection decoder can be used to control multiple bit slices, as shown in the figure above of a $2^6 \times 1$b memory. Selection between bit slices is then based on other bits from the address ($ADDR$). In the figure above, a 2-to-4 decoder is used to deliver write requests to one of four bit slices, and a 4-to-1 mux is used to choose the appropriate output bit for read requests.

The 4-to-16 decoder now activates one cell in each of the four bit slices. For a read operation, $WE = 0$, and the 2-to-4 decoder is not enabled, so it outputs all 0s. All four bit slices thus perform reads, and the desired result bit is forwarded to $DATA\text{-}OUT$ by the 4-to-1 mux. The tri-state buffer between the mux and $DATA\text{-}OUT$ is explained in a later section. For a write operation, exactly one of the bit slices has its WE input set to 1 by the 2-to-4 decoder. That bit slice writes the bit value delivered to all bit slices from $DATA\text{-}IN$. The other three bit slices perform reads, but their results are simply discarded.

The approach shown above, in which a cell is selected through a two-dimensional indexing scheme, is known as **coincident selection**. The qualifier "coincident" arises from the notion that the desired cell coincides with the intersection of the active row and column outputs from the decoders.

The benefit of coincident selection is easily calculated in terms of the number of gates required for the decoders. Decoder complexity is roughly equal to the number of outputs, as each output is a minterm and requires a unique gate to calculate it. Consider a 1M×8b RAM chip. The number of addresses is 2^{20}, and the total number of memory cells is 8,388,608 (2^{23}). One option is to use eight bit slices and a 20-to-1,048,576 decoder, or about 2^{20} gates. Alternatively, we can use 8,192 bit slices of 1,024 cells. For the second implementation, we need two 10-to-1024 decoders, or about 2^{11} gates. As chip area is roughly proportional to the number of gates, the savings are substantial. Other schemes are possible as well: if we want a more square chip area, we might choose to use 4,096 bit slices of 2,048 cells along with one 11-to-2048 decoder and one 9-to-512 decoder. This approach requires roughly 25% more decoder gates than our previous example, but is still far superior to the eight-bit-slice implementation.

Memories are typically unclocked devices. However, as you have seen, the circuits are highly structured, which enables engineers to cope with the complexity of sequential feedback design. Devices used to control memories are typically clocked, and the interaction between the two can be fairly complex.



Timing diagrams for reads and writes to SRAM are shown to the right. A write operation appears on the left. In the first cycle, the controller raises the chip select signal and places the memory address to be written on the address inputs. Once the memory has had time to set up the appropriate select lines internally, the WE input is raised, and data are placed on the data inputs. The delay, which is specified by the memory manufacturer, is necessary to avoid writing data to the incorrect element within the memory. The timing shown in the figure rounds this delay up to a single clock cycle, but the actual delay needed depends on the clock speed and the memory's specification. At some point after new data have been delivered to the memory, the write operation completes within the memory. The time from the application of the address until the (worst-case) completion of the write operation is called the **write cycle** of the memory, and is also specified by the memory manufacturer. Once the write cycle has passed, the controlling logic lowers WE, waits for the change to settle within the memory, then removes the address and lowers the chip select signal. The reason for the delay between these signal changes is the same: to avoid mistakenly overwriting another memory location.

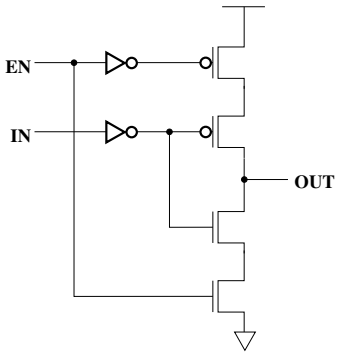
A read operation is quite similar. As shown on the right, the controlling logic places the address on the input lines and raises the chip select signal. No races need be considered, as read operations on SRAM do not affect the stored data. After a delay called the **read cycle**, the data can be read from the data outputs. The address can then be removed and the chip select signal lowered.

For both reads and writes, the number of cycles required for an operation depends on a combination of the clock cycle of the controller and the cycle time of the memory. For example, with a 25 nanosecond write cycle and a 10 nanosecond clock cycle, a write requires three cycles. In general, the number of cycles required is given by the formula $\lceil$memory cycle time/clock cycle time$\rceil$.

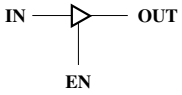
3.6.3 Tri-State Buffers and Combining Chips

Recall the buffer symbol—a triangle like an inverter, but with no inversion bubble—between the mux and the *DATA-OUT* signal of the $2^6 \times 1$b memory shown earlier. This **tri-state buffer** serves to disconnect the memory logic from the output line when the memory is not performing a read.

An implementation diagram for a tri-state buffer appears to the right along with the symbolic form and a truth table. The "Z" in the truth table output means high impedance (and is sometimes written "hi-Z"). In other words, there is effectively no electrical connection between the tri-state buffer and the output *OUT*.



tri–state buffer
(logic diagram)



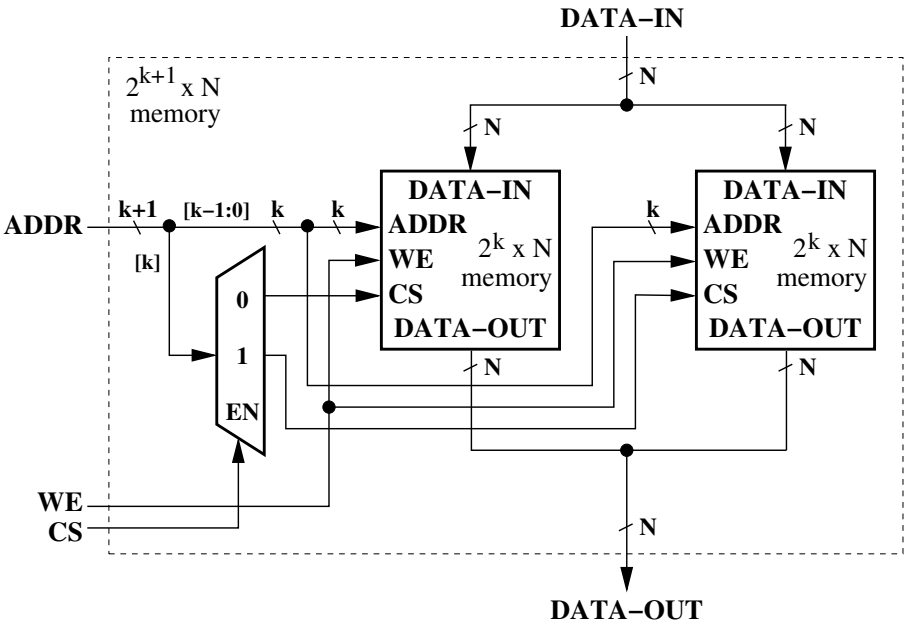
tri–state buffer
(symbolic form)

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

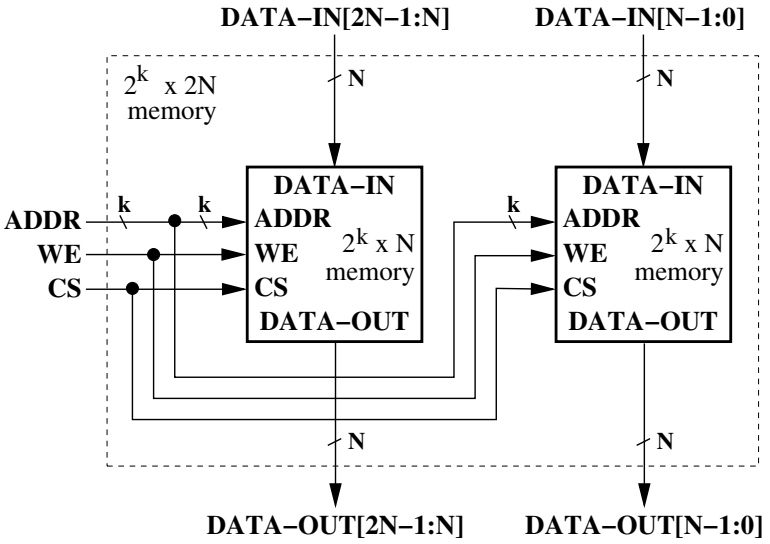
This logical disconnection is achieved by using the outer (upper and lower) pair of transistors in the logic diagram. When $EN = 0$, both transistors turn off, meaning that regardless of the value of IN, OUT is connected neither to high voltage nor to ground. When $EN = 1$, both transistors turn on, and the tri-state buffer acts as a pair of back-to-back inverters, copying the signal from IN to OUT, as shown in the truth table.

What benefit does this logical disconnection provide? So long as only one memory's chip select input is high at any time, the same output line can be shared by more than one memory without the need for additional multiplexers. Memory chips were often combined in this way to produce larger memories.

The figure to the right illustrates how larger memories can be constructed using multiple chips. In the case shown, two $2^k \times N$-bit memories are used to implement a $2^{k+1} \times N$-bit memory. One of the address bits—in the case shown, the most significant bit—is used to drive a decoder that determines which of the two chips is active ($CS = 1$). The decoder is enabled with the chip select signal for the larger memory, so neither chip is enabled when the external CS is low, as desired. The rest of the address bits, as well as the external data inputs and write enable signal, are simply delivered to both memories. The external data outputs are also connected to both memories. Ensuring that at most one chip select signal is high at any time guarantees that at most one of the two memory chips drives logic values on the data outputs.



Multiple chips can also be used to construct wider memories, as shown to the right. In the case shown, two $2^k \times N$-bit memories are used to implement a $2^k \times 2N$-bit memory. Both chips are either active or inactive at the same time, so the external address, write enable, and chip select inputs are routed to both chips. In contrast, the data inputs and outputs are separate: the left chip handles the high N bits of input on writes and produces the high N bits of output on reads, while the right chip handles the low N bits of input and produces the low N bits of output.

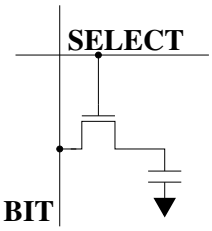


Historically, tri-state buffers were also used to reduce the number of pins needed on chips. Pins have long been a scarce resource, and the amount of data that can cross a chip's pins in a second (the product of the number of pins and the data rate per pin) has not grown nearly as rapidly as the number of transistors packed into a fixed area. By combining inputs and outputs, chip designers were able to halve the number of pins needed. For example, data inputs and outputs of memory were often combined into a single set of data wires, with bidirectional signals. When performing a read from a memory chip, the memory chip drove the data pins with the bits being read (tri-state buffers on the memory chip were enabled). When performing a write, other logic such as a processor wrote the value to be stored onto the data pins (tri-state buffers were not enabled).

3.6.4 Dynamic Random Access Memory*

Dynamic random access memory, or DRAM, is used for main memory in computers and for other applications in which size is more important than speed. While slower than SRAM, DRAM is denser (has more bits per chip area). A substantial part of DRAM density is due to transistor count: typical SRAM cells use six transistors (two for each inverter, and two more to connect the inverters to the bit lines), while DRAM cells use only a single transistor. However, memory designers have also made significant advances in further miniaturizing DRAM cells to improve density beyond the benefit available from simple transistor count.

A diagram of a DRAM cell appears to the right. DRAM storage is capacitive: a bit is stored by charging or not charging a capacitor. The capacitor is attached to a *BIT* line through a transistor controlled by a *SELECT* line. When *SELECT* is low, the capacitor is isolated and holds its charge. However, the transistor's resistance is finite, and some charge leaks out onto the bit line. Charge also leaks into the substrate on which the transistor is constructed. After some amount of time, all of the charge dissipates, and the bit is lost. To avoid such loss, the cell must be **refreshed** periodically by reading the contents and writing them back with active logic.

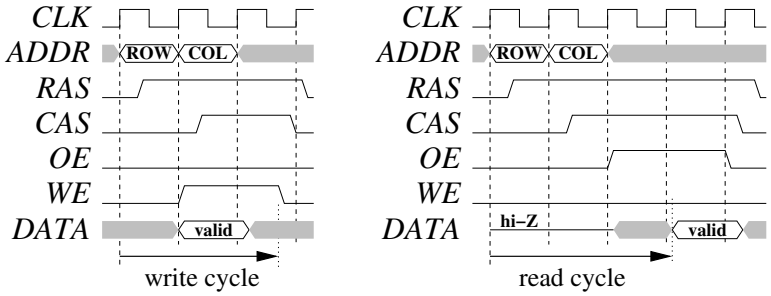


When the *SELECT* line is high during a write operation, logic driving the bit line forces charge onto the capacitor or removes all charge from it. For a read operation, the bit line is first brought to an intermediate voltage level (a voltage level between 0 and 1), then *SELECT* is raised, allowing the capacitor to either pull a small amount of charge from the bit line or to push a small amount of charge onto the bit line. The resulting change in voltage is then detected by a **sense amplifier** at the end of the bit line. A sense amp is analogous to a marble on a mountaintop: a small push causes the marble to roll rapidly downhill in the direction of the push. Similarly, a small change in voltage causes a sense amp's output to move rapidly to a logical 0 or 1, depending on the direction of the small change. As mentioned earlier, sense amplifiers also appear in SRAM implementations. While not technically necessary, as they are with DRAM, the use of a sense amp to react to small changes in voltage makes reads faster.

Each read operation on a DRAM cell brings the voltage on its capacitor closer to the intermediate voltage level, in effect destroying the data in the cell. DRAM is thus said to have **destructive reads**. To preserve data during a read, the bits must be written back into the cells after a read. For example, the output of the sense amplifiers can be used to drive the bit lines, rewriting the cells with the appropriate data.

At the chip level, typical DRAM inputs and outputs differ from those of SRAM. Due to the large size and high density of DRAM, addresses are split into row and column components and provided through a common set of pins. The DRAM stores the components in registers to support this approach. Additional inputs, known as the **row** and **column address strobes**—*RAS* and *CAS*, respectively—are used to indicate when address components are available. As you might guess from the structure of coincident selection, DRAM refresh occurs on a row-by-row basis (across bit slices—on columns rather than rows in the figures earlier in these notes, but the terminology of DRAM is a row). Raising the *SELECT* line for a row destructively reads the contents of all cells on that row, forcing the cells to be rewritten and effecting a refresh. The row is thus a natural basis for the refresh cycle. The DRAM data pins provide bidirectional signals for reading and writing elements of the DRAM. An **output enable** input, *OE*, controls tri-state buffers with the DRAM to determine whether or not the DRAM drives the data pins. The *WE* input, which controls the type of operation, is also present.

Timing diagrams for writes and reads on a historical DRAM implementation appear to the right. In both cases, the row component of the address is first applied to the address pins, then *RAS* is raised. In the next cycle of the controlling logic, the column component is applied to the address pins, and *CAS* is raised.



For a write, as shown on the left, the *WE* signal and the data can also be applied in the second cycle. The DRAM has internal timing and control logic that prevent races from overwriting an incorrect element (remember that the row and column addresses have to be stored in registers). The DRAM again specifies a write cycle, after which the operation is guaranteed to be complete. In order, the *WE*, *CAS*, and *RAS* signals are then lowered.

For a read operation, the output enable signal, *OE*, is raised after *CAS* is raised. The *DATA* pins, which should be floating (in other words, not driven by any logic), are then driven by the DRAM. After the read cycle, valid data appear on the *DATA* pins, and *OE*, *CAS*, and *RAS* are lowered in order after the data are read.

Modern DRAM chips are substantially more sophisticated than those discussed here, and many of the functions that used to be provided by external logic are now integrated onto the chips themselves. As an example of modern DRAMs, one can obtain the data sheet for Micron Semiconductor's 8Gb ($2^{31} \times 4$b, for example) DDR4 SDRAM, which is 366 pages long as of 11 May 2016.

The ability to synchronize to an external clock has become prevalent in the industry, leading to the somewhat confusing term SDRAM, which stands for **synchronous DRAM**. The memory structures themselves are still unclocked, but logic is provided on the chip to synchronize accesses to the external clock without the need for additional logic. The clock provided to the Micron chip just mentioned can be as fast as 1.6 GHz, and data can be transferred on both the rising and falling edges of the clock (hence the name DDR, or **double data rate**).

In addition to row and column components of the address, these chips further separate cells into **banks** and groups of banks. These allow a user to exploit parallelism by starting reads or writes to separate banks at the same time, thus improving the speed at which data can move in and out of the memory. For the $2^{31} \times 4$b version of the Micron chip, the cells are structured into 4 groups of 4 banks (16 banks total), each with 131,072 rows and 1,024 columns.

DRAM implementations provide interfaces for specifying refresh operations in addition to reads and writes. Managing refresh timing and execution is generally left to an external DRAM controller. For the Micron chip, refresh commands must be issued every 7.8 microseconds at normal temperatures. Each command refreshes about 2^{20} cells, so 8,192 commands refresh the whole chip in less than 64 milliseconds. Alternatively, the chip can handle refresh on-chip in order to maintain memory contents when the rest of the system is powered down.