ECE 290: Introduction to Computer Engineering **Fall 2000**
Notes from 25 and 30 October Lectures

Random Access Memories

I shall continue to avoid promising to provide these notes, although I have effectively promised to give you at least the full list of learning objectives in this format.

Learning Objectives

This section outlines what you should expect to know, and what I'll expect you to know, after reading these notes. As I indicated during lecture, the list in class was slightly abbreviated due to the limited space available on a slide; the full list is given here.

Terminology and Circuits

- definition of memory
- data units: bit, byte, word
- memory address
- memory operations
  - read
  - write
- types of memory
  - volatile and non-volatile
  - serial
  - random access (RAM)
- memory structures
  - cell
  - bit slice
  - coincident selection
- static RAM (SRAM)
  - physical and logical cell structure
  - typical "chip" inputs and outputs: data and address lines, chip select, read/$\overline{\text{write}}$
  - timing of read/write operations
- dynamic RAM (DRAM)
  - physical and logical cell structure
  - refresh
  - destructive reads
  - typical "chip" inputs and outputs: data and address lines, row and column address strobes (RAS, CAS), output enable
  - timing of read/write operations
  - DRAM controller
- bidirectional signals

Tradeoffs and Typical Values

- SRAM vs. DRAM
- typical refresh rates and styles
  - burst
  - distributed

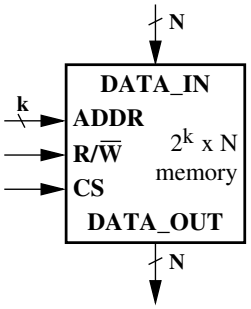
Skills

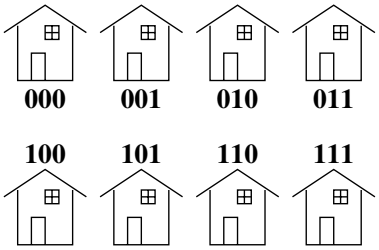
- Employ tri-state buffers to share input and output lines for a circuit.
- Construct larger memories out of smaller bit slices or chips and decoders.

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 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 strongly on the particular context. For example, when talking about PC's, a word usually means 16 bits. When talking about workstations or mainframes, it usually means 32 bits.

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 cell. 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 disks, tapes, and CD-ROM's, as well as some programmable logic devices, such as ROM's. Finally, the memories considered in this class are **random access memories (RAM's)**, 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.

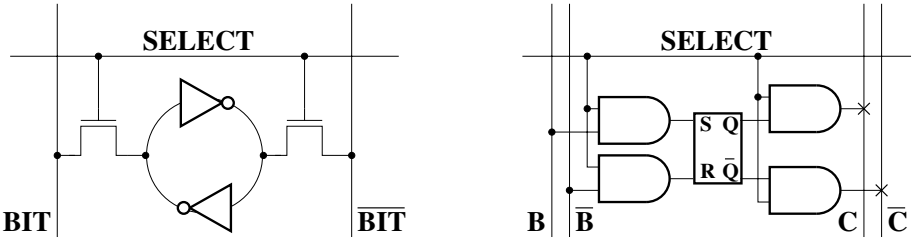
The figure on the left above shows a generic RAM structure. The memory contains 2^k elements of N bits each. A k-bit address input, $ADDR$, identifies the memory element of interest for any particular operation. The **read/write** input, $R/\overline{W}$, selects the operation to be performed: if $R/\overline{W}$ is high, the operation is a read; if it is low, the operation is a write. 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, *i.e.*, both lose all data when the power supply is removed. We study both SRAM and DRAM in some detail in this course.

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.* DRAM is also unavailable when recharging its capacitors, which can be a problem for applications with stringent real-time needs.

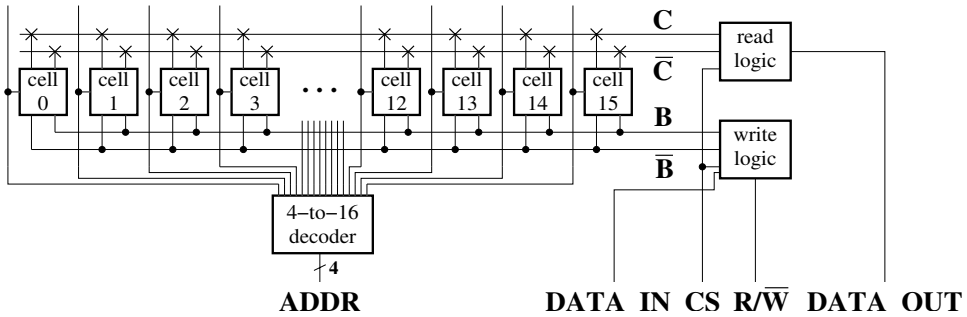
*Research is underway to explore the potential of chips that combine DRAM and processor logic. Until a few years ago, the processes used to produce the two types of devices did not allow a single chip to contain both.



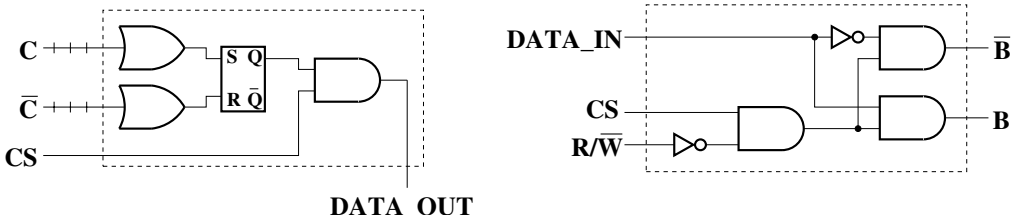
Two diagrams of an SRAM **cell** (a single bit) appear above. On the left is the physical implementation: a dual-inverter loop hooked to opposing BIT lines through transistors controlled by a $SELECT$ line. On the right is a logical implementation[†] modeled after that given by Mano & Kime.

The physical cell works as follows. When $SELECT$ is high, the transistors connect the inverter loop to the bit lines. When writing a cell, the 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 inverter loop to drive the lines to their current values. 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 effectively disconnect the inverters from the bit lines, and the cell holds its current value until $SELECT$ goes high again.

The logical cell retains the $SELECT$ line, replaces the inverter loop with an S-R latch, and splits the bit lines into bit write lines (B and $\bar{B}$) and bit read lines (C and $\bar{C}$). When $SELECT$ is low, all AND gates output 0, and the isolated cell holds its value. When $SELECT$ is high, the cell can be written by raising the B or $\bar{B}$ input signals, which set or reset the latch, as appropriate. Similarly, the latched value appears on C and $\bar{C}$. Recall that the $\times$ markers represent connections to many-input gates, which appear in the read logic described below.



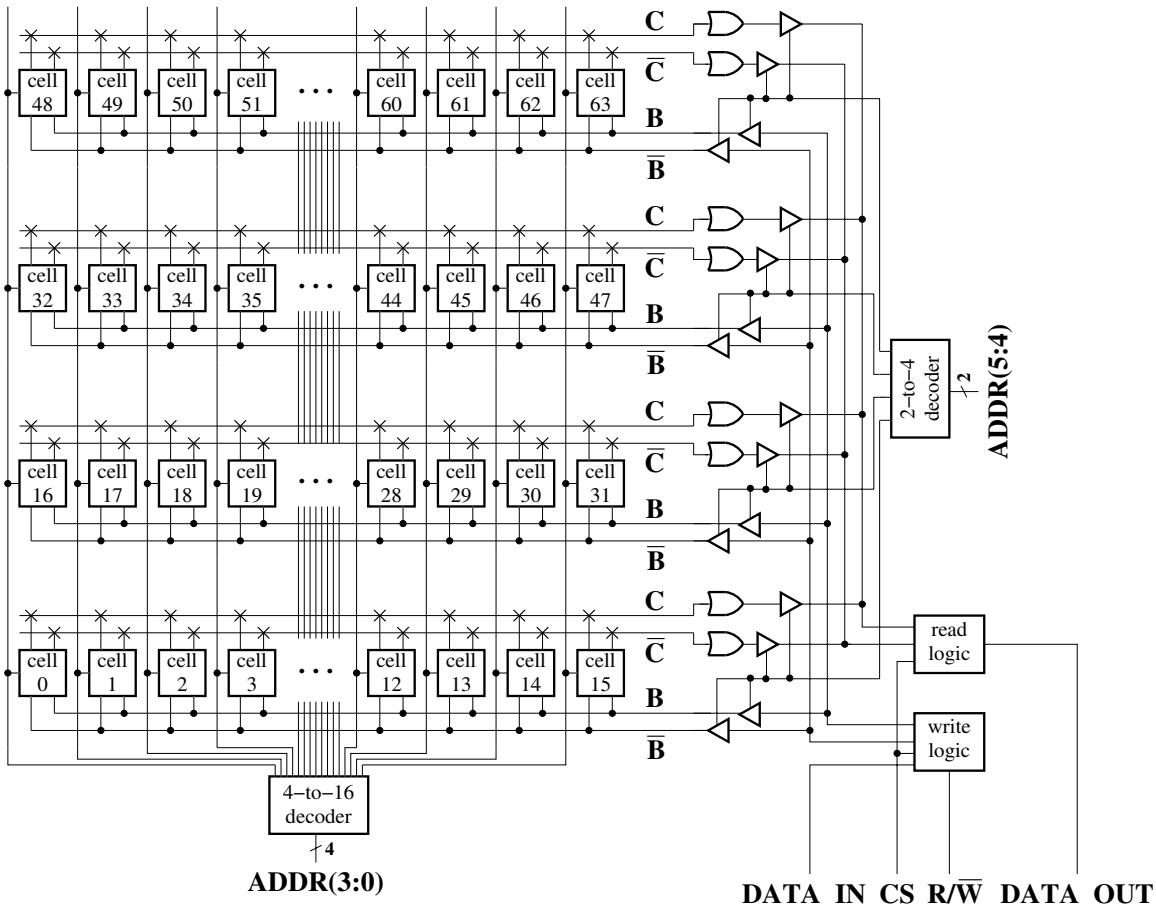
A number of cells are combined into a **bit slice**, as shown above. The cells share bit lines and read/write logic, which appears to the right in the figure. Based on the $ADDR$ input, a decoder sets one cell's $SELECT$ line high to enable a read or write operation to the cell. Details of the read and write logic are shown below, to the left and right, respectively.



The read logic requires only the bit read lines and the chip select signal as inputs. The read lines function logically as many-input OR gates, and the results of these lines are used to set or reset the S-R latch in the read logic. When CS is high, the value held in the latch is then placed on the $DATA\_OUT$ line.

The write logic requires two enable inputs, CS and $R/\overline{W}$, as well as a data input. When CS is high and a write operation is requested, *i.e.*, $R/\overline{W}$ is low, the output of the AND gate in the lower left of the diagram goes high, enabling the AND gates to the right of the diagram to place the data input on the bit write lines.

[†] "Logical implementation" here implies that the functional behavior of the circuit is equivalent to that of the real circuit. The real circuit is that shown on the left of the figure.

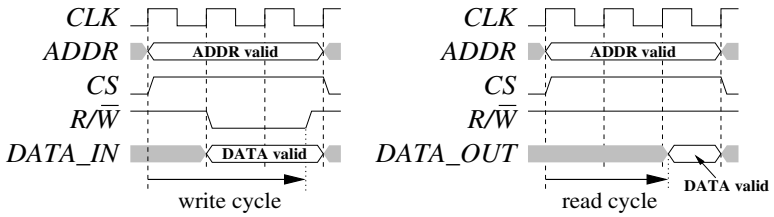


The outputs of the cell selection decoder can be used to control multiple bit slices, as shown above. Selection between bit slices is then based on other bits from the $ADDRESS$ input. In the figure above, a 2-to-4 decoder enables one of four sets of tri-state buffers that connect the bit read and write lines to the read and write logic. The many-input OR gates—a fictional construct of the logical representation—have been replicated for each bit slice. In a real implementation, the transistor-gated connections to the bit lines eliminate the need for the OR gates, and the extra logic amounts to only a pair of transistors per bit slice.

The approach shown above, in which one or more cells are 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 select lines.

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. Fanout trees for input terms and inverted terms add relatively few gates. Consider a 1M×8b RAM chip. The number of addresses is 2^{20}. One option is to use a single bit slice and a 20-to-1048576 decoder, or about 2^{20} gates. Alternatively, we can use 8,192 bit slices of 1,024 cells (remember that we must output eight bits). For this 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 50% more decoder gates than our previous example, but is still far superior to the single 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 at the top of the next page. 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.



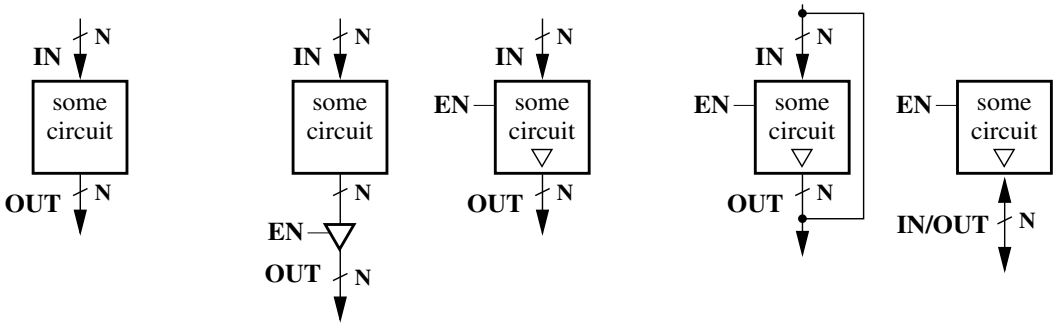
Once the memory has had time to set up the appropriate select lines internally, the $R/\overline{W}$ input is lowered and data are placed on the data inputs. The delay, which is specified by the memory manufacturer, is necessary to avoid writing data to the incorrect element within the memory. In the diagram, the delay is one cycle, but delay logic can be used to tune the timing to match the memory's specification, if desired. 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 manufacturer. One the write cycle has passed, the controlling logic raises $R/\overline{W}$, waits for the change to settle within the memory, then removes the address and lowers the chip select signal. The reason for the delay 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 $\lceil \text{memory cycle time} / \text{clock cycle time} \rceil$.

Bidirectional Signals

We have on several previous occasions discussed the utility of tri-state buffers in gating outputs and constructing multiplexers. With shift registers, we also considered the use of tri-state buffers to use the same lines for reading the register and parallel load of the register. In this section, we consider in general the application of tri-state buffers to reduce pin count and examine the symbols used to denote their presence.



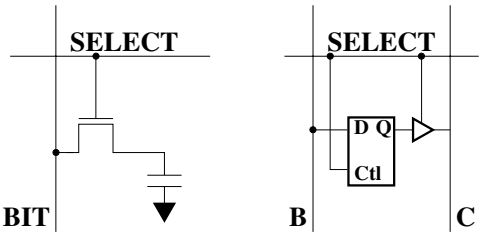
The figure above shows three groups of equivalent circuits. We begin with the generic circuit on the left, with N inputs and N outputs. The circuit may have additional inputs and outputs beyond those shown, but it will be convenient to restrict this discussion to an equal number of inputs and outputs. The second group in the figure extends the first by using an enable input, EN, to gate the circuit outputs with N tri-state buffers. The left member of the group adds the buffers externally, while the right member (third from the left in the overall figure) adds them implicitly, as indicated by the inverted triangle symbol near the OUT pins. This symbol is not meant to point towards the pins, but is rather always drawn in the orientation shown, regardless of output direction in a figure. The third group further extends the circuit by connecting its gated outputs to its inputs, either externally or internally (fourth and fifth from the left, respectively). The resulting connections are called **bidirectional signals**, as information can flow either into or out of the circuit. Bidirectional signals are important for memory devices, for which the number of logical inputs and outputs

can be quite large. Data inputs and outputs, for example, are typically combined into a single set of bidirectional signals. The arrowheads in the figure are not a standard part of the representation, but are sometimes provided to clarify the flow of information. The labels provide complete I/O information and allow you to identify bidirectional signals.

With bidirectional signals, and with all outputs gated by tri-state buffers, it is important to ensure that multiple circuits are not simultaneously allowed to drive a set of wires, as attempting to drive wires to different logic values creates a short circuit from high voltage to ground, which can easily destroy the system.

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 also is also denser (has more bits per chip area).



Two diagrams of a DRAM cell appear above. On the left is the physical implementation: a capacitor attached to a BIT line through a transistor controlled by a $SELECT$ line. On the right is a logical implementation modeled after that given by Mano & Kime.

The logical implementation employs a D latch to record the value on the bit write line, B, whenever the $SELECT$ line is high. The output of the latch is also placed on the bit read line, C, when $SELECT$ is high. Rather than many-input gates, a tri-state buffer controls the output gating to remind you that DRAM cells are read only when selected.

As illustrated by the physical cell structure, DRAM storage is capacitive, *i.e.*, each bits is stored by charging or not charging a capacitor. When $SELECT$ is low, the capacitor is isolated, and it holds its charge. However, the resistance across the transistor is finite, and some charge leaks out onto the bit line. Charge also leaks into the substrate on which the device 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.

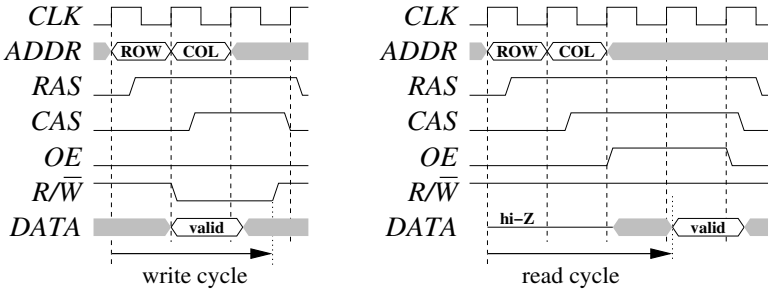
When the $SELECT$ line is high during a write operation, logic driving the bit line forces charge onto the capacitor or removes all charge from it. For a read operation, the bit line is first brought to an intermediate voltage level (a voltage level between 0 and 1), then $SELECT$ is raised, allowing 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. 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 data read must be written back into the cells. 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 many DRAM's, 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 avail-

[‡]The implementation of a sense amplifier lies outside the scope of this class, but you should understand their role in memory.

able. These control signals are also used to manage the DRAM refresh process (see Mano & Kime for details). As you might guess from the structure of coincident selection, DRAM refresh occurs on a row-by-row basis; 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 $R/\overline{W}$ input, which controls the type of operation, is also present.



Timing diagrams for DRAM writes and reads appear above. 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 $R/\overline{W}$ 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 $R/\overline{W}$ signal is then raised, the CAS signal lowered, and the RAS signal lowered. Other orders of signal removal have different meanings, such as initiation of a refresh.

For a read operation, the output enable signal, OE, is lowered after CAS is raised. The $DATA$ pins, which should be floating (*i.e.*, 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.

A typical DRAM implementation provides several approaches to managing refresh, but does not initiate any refreshes internally. Refresh requirements are specified, but managing the refresh itself is left to a **DRAM controller**. The duties of this controller also include mapping addresses into row and column components, managing timing for signals to and from the DRAM, and providing status indicators on the state of the DRAM.

As an example of refresh rates and requirements for modern DRAM's, I obtained a few specifications for a 16Mx4b EDO DRAM chip manufactured by Micron Semiconductor. The cells are structured into 4,096 rows, each of which must be refreshed every 64 milliseconds. Using a certain style of refresh (CAS-before-RAS, or CBR), the process of refreshing a single row takes roughly 100 nanoseconds. The most common approach to managing refresh, termed **distributed refresh**, cycles through rows one at a time over a period of the required refresh time, in this case 64 milliseconds. Row refreshes occur regularly within this period, or about every 16 microseconds. The refreshes keep the DRAM busy 0.64% of the time; at other times, it can be used for reads and writes. Alternatively, we might choose a **burst refresh** approach, in which we refresh all rows in a burst. A burst refresh requires roughly 410 microseconds for the DRAM under discussion, as all 4,096 rows must be refreshed, and each row requires about 100 nanoseconds. A delay of 410 microseconds is a long delay by processor standards, thus burst refresh is rarely used.

[§] In practice, RAS, CAS, and OE are active low signals, and are thus usually written and appear as $\overline{RAS}$, $\overline{CAS}$, and $\overline{OE}$.