CHAPTER

5

THE MEMORY SYSTEM

CHAPTER OBJECTIVES

In this chapter you will learn about:

- Basic memory circuits
- Organization of the main memory
- Cache memory concept, which shortens the effective memory access time
- Virtual memory mechanism, which increases the apparent size of the main memory
- Magnetic disks, optical disks, and magnetic tapes used for secondary storage

Programs and the data they operate on are held in the memory of the computer. In this chapter, we discuss how this vital part of the computer operates. By now, the reader appreciates that the execution speed of programs is highly dependent on the speed with which instructions and data can be transferred between the processor and the memory. It is also important to have a large memory to facilitate execution of programs that are large and deal with huge amounts of data.

Ideally, the memory would be fast, large, and inexpensive. Unfortunately, it is impossible to meet all three of these requirements simultaneously. Increased speed and size are achieved at increased cost. To solve this problem, much work has gone into developing clever structures that improve the apparent speed and size of the memory, yet keep the cost reasonable.

First, we describe the most common components and organizations used to implement the memory. Then we examine memory speed and discuss how the apparent speed of the memory can be increased by means of caches. Next, we present the virtual memory concept, which increases the apparent size of the memory. Finally, we discuss the secondary storage devices, which provide much larger storage capability.

5.1 SOME BASIC CONCEPTS

The maximum size of the memory that can be used in any computer is determined by the addressing scheme. For example, a 16-bit computer that generates 16-bit addresses is capable of addressing up to $2^{16} = 64K$ memory locations. Similarly, machines whose instructions generate 32-bit addresses can utilize a memory that contains up to $2^{32} = 4G$ (giga) memory locations, whereas machines with 40-bit addresses can access up to $2^{40} = 1T$ (tera) locations. The number of locations represents the size of the address space of the computer.

Most modern computers are byte addressable. Figure 2.7 shows the possible address assignments for a byte-addressable 32-bit computer. The big-endian arrangement is used in the 68000 processor. The little-endian arrangement is used in Intel processors. The ARM architecture can be configured to use either arrangement. As far as the memory structure is concerned, there is no substantial difference between the two schemes.

The memory is usually designed to store and retrieve data in word-length quantities. In fact, the number of bits actually stored or retrieved in one memory access is the most common definition of the word length of a computer. Consider, for example, a byte-addressable computer whose instructions generate 32-bit addresses. When a 32-bit address is sent from the processor to the memory unit, the high-order 30 bits determine which word will be accessed. If a byte quantity is specified, the low-order 2 bits of the address specify which byte location is involved. In a Read operation, other bytes may be fetched from the memory, but they are ignored by the processor. If the byte operation is a Write, however, the control circuitry of the memory must ensure that the contents of other bytes of the same word are not changed.

Modern implementations of computer memory are rather complex and difficult to understand on first encounter. To simplify our introduction to memory structures, we

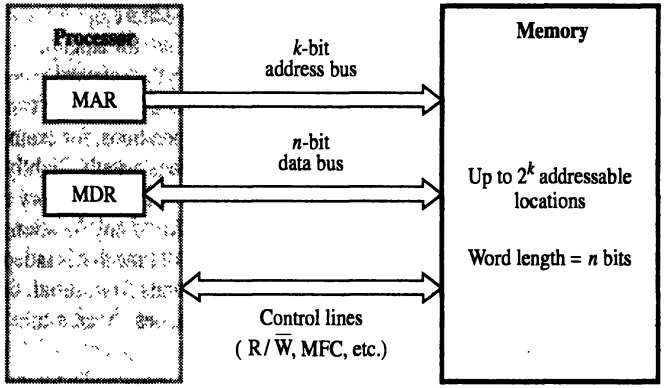

Figure 5.1 Connection of the memory to the processor.

will first present a traditional architecture. Then, in later sections, we will discuss the latest approaches.

From the system standpoint, we can view the memory unit as a black box. Data transfer between the memory and the processor takes place through the use of two processor registers, usually called MAR (memory address register) and MDR (memory data register), as introduced in Section 1.2. If MAR is k bits long and MDR is n bits long, then the memory unit may contain up to 2^k addressable locations. During a memory cycle, n bits of data are transferred between the memory and the processor. This transfer takes place over the processor bus, which has k address lines and n data lines. The bus also includes the control lines Read/$\overline{\text{Write}}$ (R/$\overline{\text{W}}$) and Memory Function Completed (MFC) for coordinating data transfers. Other control lines may be added to indicate the number of bytes to be transferred. The connection between the processor and the memory is shown schematically in Figure 5.1.

The processor reads data from the memory by loading the address of the required memory location into the MAR register and setting the R/$\overline{\text{W}}$ line to 1. The memory responds by placing the data from the addressed location onto the data lines, and confirms this action by asserting the MFC signal. Upon receipt of the MFC signal, the processor loads the data on the data lines into the MDR register.

The processor writes data into a memory location by loading the address of this location into MAR and loading the data into MDR. It indicates that a write operation is involved by setting the R/$\overline{\text{W}}$ line to 0.

If read or write operations involve consecutive address locations in the main memory, then a "block transfer" operation can be performed in which the only address sent to the memory is the one that identifies the first location. We will encounter a need for such block transfers in Section 5.5.

Memory accesses may be synchronized using a clock, or they may be controlled using special signals that control transfers on the bus, using the bus signaling schemes described in Section 4.5.1. Memory read and write operations are controlled as input and output bus transfers, respectively.

A useful measure of the speed of memory units is the time that elapses between the initiation of an operation and the completion of that operation, for example, the time between the Read and the MFC signals. This is referred to as the *memory access time.* Another important measure is the *memory cycle time,* which is the minimum time delay required between the initiation of two successive memory operations, for example, the time between two successive Read operations. The cycle time is usually slightly longer than the access time, depending on the implementation details of the memory unit.

A memory unit is called *random-access memory* (RAM) if any location can be accessed for a Read or Write operation in some fixed amount of time that is independent of the location's address. This distinguishes such memory units from serial, or partly serial, access storage devices such as magnetic disks and tapes. Access time on the latter devices depends on the address or position of the data.

The basic technology for implementing the memory uses semiconductor integrated circuits. The sections that follow present some basic facts about the internal structure and operation of such memories. We then discuss some of the techniques used to increase the effective speed and size of the memory.

The processor of a computer can usually process instructions and data faster than they can be fetched from a reasonably priced memory unit. The memory cycle time, then, is the bottleneck in the system. One way to reduce the memory access time is to use a *cache memory.* This is a small, fast memory that is inserted between the larger, slower main memory and the processor. It holds the currently active segments of a program and their data.

Virtual memory is another important concept related to memory organization. So far, we have assumed that the addresses generated by the processor directly specify physical locations in the memory. This may not always be the case. For reasons that will become apparent later in this chapter, data may be stored in physical memory locations that have addresses different from those specified by the program. The memory control circuitry translates the address specified by the program into an address that can be used to access the physical memory. In such a case, an address generated by the processor is referred to as a *virtual* or *logical address.* The virtual address space is mapped onto the physical memory where data are actually stored. The mapping function is implemented by a special memory control circuit, often called the *memory management unit.* This mapping function can be changed during program execution according to system requirements.

Virtual memory is used to increase the apparent size of the physical memory. Data are addressed in a virtual address space that can be as large as the addressing capability of the processor. But at any given time, only the active portion of this space is mapped onto locations in the physical memory. The remaining virtual addresses are mapped onto the bulk storage devices used, which are usually magnetic disks. As the active portion of the virtual address space changes during program execution, the memory management unit changes the mapping function and transfers data between the disk and the memory. Thus, during every memory cycle, an address-processing mechanism determines whether the addressed information is in the physical memory unit. If it is, then the proper word is accessed and execution proceeds. If it is not, a *page* of words containing the desired word is transferred from the disk to the memory, as explained in Section 5.7.1. This page displaces some page in the memory that is currently inactive.

Because of the time required to move pages between the disk and the memory, there is a speed degradation if pages are moved frequently. By judiciously choosing which page to replace in the memory, however, there may be reasonably long periods when the probability is high that the words accessed by the processor are in the physical memory unit.

This section has briefly introduced several organizational features of memory systems. These features have been developed to help provide a computer system with as large and as fast a memory as can be afforded in relation to the overall cost of the system. We do not expect the reader to grasp all the ideas or their implications now; more detail is given later. We introduce these terms together to establish that they are related; a study of their interrelationships is as important as a detailed study of their individual features.

5.2 SEMICONDUCTOR RAM MEMORIES

Semiconductor memories are available in a wide range of speeds. Their cycle times range from 100 ns to less than 10 ns. When first introduced in the late 1960s, they were much more expensive than the magnetic-core memories they replaced. Because of rapid advances in VLSI (Very Large Scale Integration) technology, the cost of semiconductor memories has dropped dramatically. As a result, they are now used almost exclusively in implementing memories. In this section, we discuss the main characteristics of semiconductor memories. We start by introducing the way that a number of memory cells are organized inside a chip.

5.2.1 INTERNAL ORGANIZATION OF MEMORY CHIPS

Memory cells are usually organized in the form of an array, in which each cell is capable of storing one bit of information. A possible organization is illustrated in Figure 5.2. Each row of cells constitutes a memory word, and all cells of a row are connected to a common line referred to as the *word line,* which is driven by the address decoder on the chip. The cells in each column are connected to a Sense/Write circuit by two *bit lines.* The Sense/Write circuits are connected to the data input/output lines of the chip. During a Read operation, these circuits sense, or read, the information stored in the cells selected by a word line and transmit this information to the output data lines. During a Write operation, the Sense/Write circuits receive input information and store it in the cells of the selected word.

Figure 5.2 is an example of a very small memory chip consisting of 16 words of 8 bits each. This is referred to as a 16 × 8 organization. The data input and the data output of each Sense/Write circuit are connected to a single bidirectional data line that can be connected to the data bus of a computer. Two control lines, $R/\overline{W}$ and CS, are provided in addition to address and data lines. The $R/\overline{W}$ (Read/$\overline{\text{Write}}$) input specifies the required operation, and the CS (Chip Select) input selects a given chip in a multichip memory system. This will be discussed in Section 5.2.4.

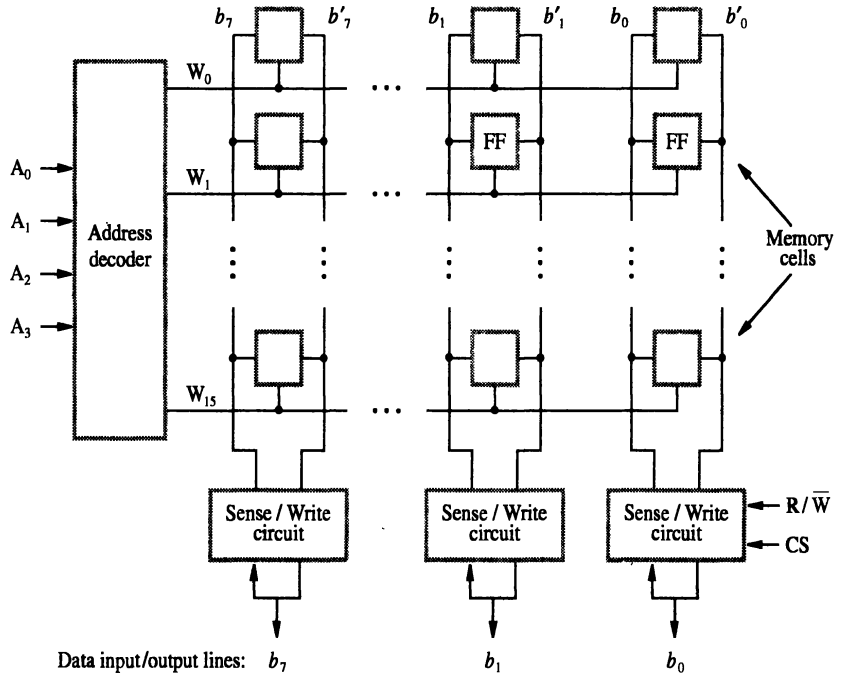

Figure 5.2 Organization of bit cells in a memory chip.

The memory circuit in Figure 5.2 stores 128 bits and requires 14 external connections for address, data, and control lines. Of course, it also needs two lines for power supply and ground connections. Consider now a slightly larger memory circuit, one that has 1K (1024) memory cells. This circuit can be organized as a 128 × 8 memory, requiring a total of 19 external connections. Alternatively, the same number of cells can be organized into a 1K × 1 format. In this case, a 10-bit address is needed, but there is only one data line, resulting in 15 external connections. Figure 5.3 shows such an organization. The required 10-bit address is divided into two groups of 5 bits each to form the row and column addresses for the cell array. A row address selects a row of 32 cells, all of which are accessed in parallel. However, according to the column address, only one of these cells is connected to the external data line by the output multiplexer and input demultiplexer.

Commercially available memory chips contain a much larger number of memory cells than the examples shown in Figures 5.2 and 5.3. We use small examples to make the figures easy to understand. Large chips have essentially the same organization as Figure 5.3 but use a larger memory cell array and have more external connections. For example, a 4M-bit chip may have a 512K × 8 organization, in which case 19 address and 8 data input/output pins are needed. Chips with a capacity of hundreds of megabits are now available.

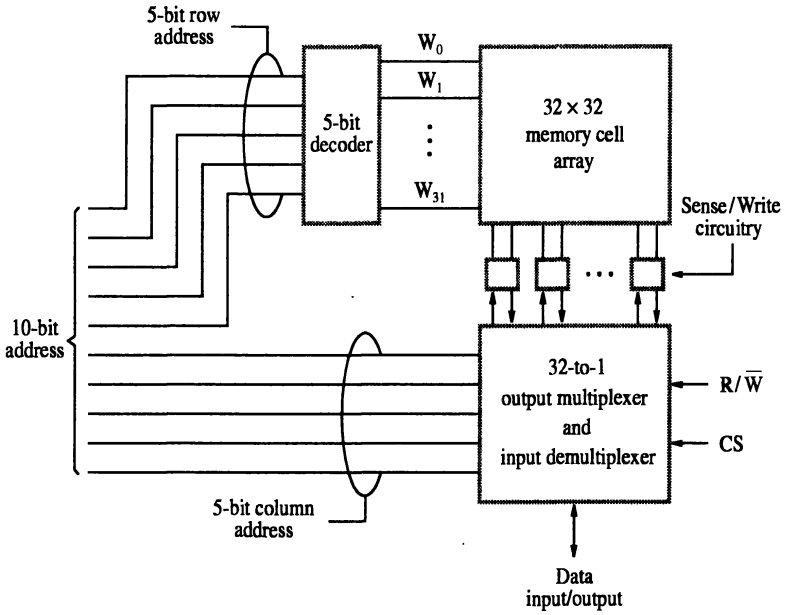

Figure 5.3 Organization of a 1K x 1 memory chip.

5.2.2 STATIC MEMORIES

Memories that consist of circuits capable of retaining their state as long as power is applied are known as *static memories*. Figure 5.4 illustrates how a *static RAM* (SRAM) cell may be implemented. Two inverters are cross-connected to form a latch. The latch is connected to two bit lines by transistors T_1 and T_2. These transistors act as switches that can be opened or closed under control of the word line. When the word line is at ground level, the transistors are turned off and the latch retains its state. For example, let us assume that the cell is in state 1 if the logic value at point X is 1 and at point Y is 0. This state is maintained as long as the signal on the word line is at ground level.

Read Operation

In order to read the state of the SRAM cell, the word line is activated to close switches T_1 and T_2. If the cell is in state 1, the signal on bit line b is high and the signal on bit line b' is low. The opposite is true if the cell is in state 0. Thus, b and b' are complements of each other. Sense/Write circuits at the end of the bit lines monitor the state of b and b' and set the output accordingly.

Write Operation

The state of the cell is set by placing the appropriate value on bit line b and its complement on b', and then activating the word line. This forces the cell into the corresponding state. The required signals on the bit lines are generated by the Sense/Write circuit.

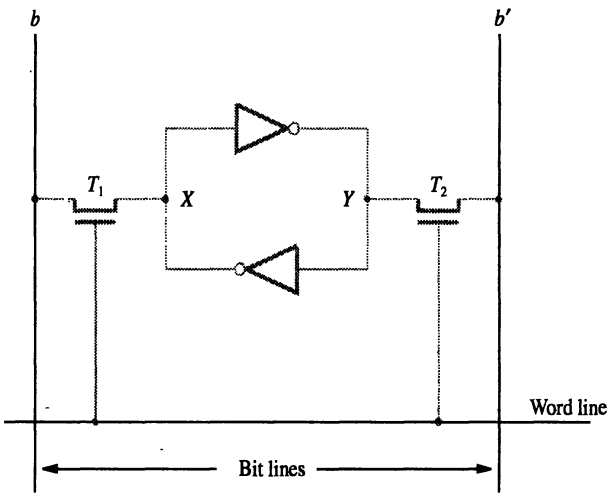

Figure 5.4 A static RAM cell.

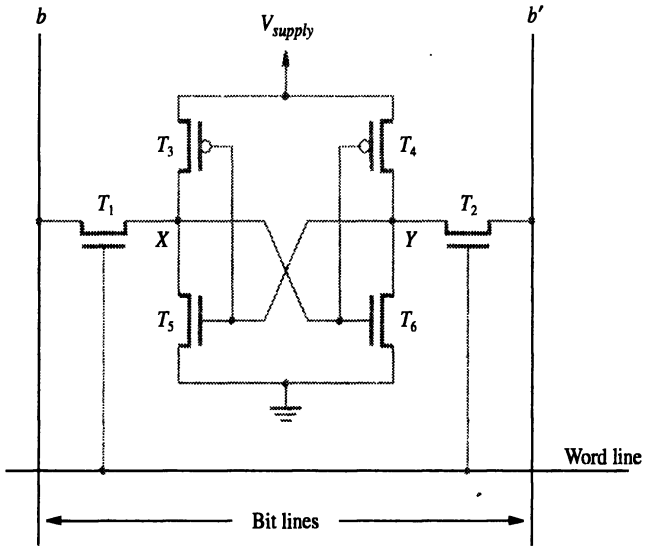

Figure 5.5 An example of a CMOS memory cell.

CMOS Cell

A CMOS realization of the cell in Figure 5.4 is given in Figure 5.5. Transistor pairs (T_3, T_5) and (T_4, T_6) form the inverters in the latch (see Appendix A). The state of the cell is read or written as just explained. For example, in state 1, the voltage at point X is maintained high by having transistors T_3 and T_6 on, while T_4 and T_5 are off. Thus,

if T_1 and T_2 are turned on (closed), bit lines b and b' will have high and low signals, respectively.

The power supply voltage, V_{supply}, is 5 V in older CMOS SRAMs or 3.3 V in new low-voltage versions. Note that continuous power is needed for the cell to retain its state. If power is interrupted, the cell's contents will be lost. When power is restored, the latch will settle into a stable state, but it will not necessarily be the same state the cell was in before the interruption. Hence, SRAMs are said to be *volatile* memories because their contents are lost when power is interrupted.

A major advantage of CMOS SRAMs is their very low power consumption because current flows in the cell only when the cell is being accessed. Otherwise, T_1, T_2, and one transistor in each inverter are turned off, ensuring that there is no active path between V_{supply} and ground.

Static RAMs can be accessed very quickly. Access times of just a few nanoseconds are found in commercially available chips. SRAMs are used in applications where speed is of critical concern.

5.2.3 ASYNCHRONOUS DRAMS

Static RAMs are fast, but they come at a high cost because their cells require several transistors. Less expensive RAMs can be implemented if simpler cells are used. However, such cells do not retain their state indefinitely; hence, they are called *dynamic RAMs* (DRAMs).

Information is stored in a dynamic memory cell in the form of a charge on a capacitor, and this charge can be maintained for only tens of milliseconds. Since the cell is required to store information for a much longer time, its contents must be periodically refreshed by restoring the capacitor charge to its full value.

An example of a dynamic memory cell that consists of a capacitor, C, and a transistor, T, is shown in Figure 5.6. In order to store information in this cell, transistor

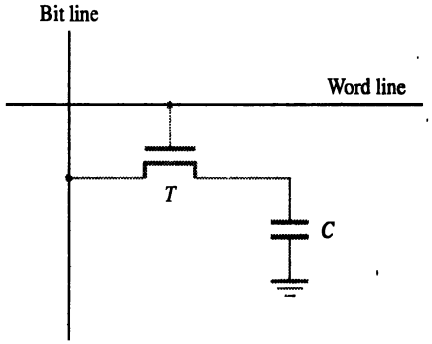

Figure 5.6 A single-transistor dynamic memory cell.

T is turned on and an appropriate voltage is applied to the bit line. This causes a known amount of charge to be stored in the capacitor.

After the transistor is turned off, the capacitor begins to discharge. This is caused by the capacitor's own leakage resistance and by the fact that the transistor continues to conduct a tiny amount of current, measured in picoamperes, after it is turned off. Hence, the information stored in the cell can be retrieved correctly only if it is read before the charge on the capacitor drops below some threshold value. During a Read operation, the transistor in a selected cell is turned on. A sense amplifier connected to the bit line detects whether the charge stored on the capacitor is above the threshold value. If so, it drives the bit line to a full voltage that represents logic value 1. This voltage recharges the capacitor to the full charge that corresponds to logic value 1. If the sense amplifier detects that the charge on the capacitor is below the threshold value, it pulls the bit line to ground level, which ensures that the capacitor will have no charge, representing logic value 0. Thus, reading the contents of the cell automatically refreshes its contents. All cells in a selected row are read at the same time, which refreshes the contents of the entire row. The detailed implementation of the sense amplifier circuit is beyond the scope of this book.

A 16-megabit DRAM chip, configured as 2M × 8, is shown in Figure 5.7. The cells are organized in the form of a 4K × 4K array. The 4096 cells in each row are divided into 512 groups of 8, so that a row can store 512 bytes of data. Therefore, 12 address bits are needed to select a row. Another 9 bits are needed to specify a group of 8 bits in the selected row. Thus, a 21-bit address is needed to access a byte in this memory. The high-order 12 bits and the low-order 9 bits of the address constitute

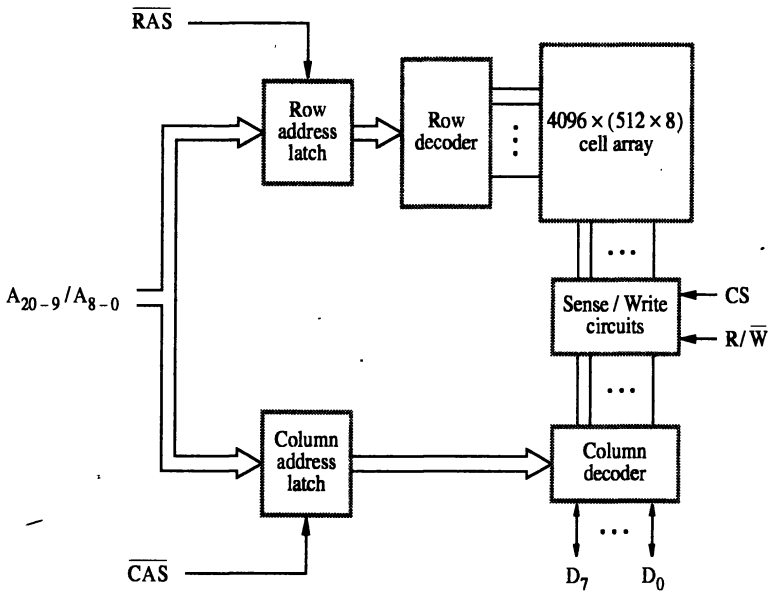

Figure 5.7 Internal organization of a 2M × 8 dynamic memory chip.

the row and column addresses of a byte, respectively. To reduce the number of pins needed for external connections, the row and column addresses are multiplexed on 12 pins. During a Read or a Write operation, the row address is applied first. It is loaded into the row address latch in response to a signal pulse on the Row Address Strobe (RAS) input of the chip. Then a Read operation is initiated, in which all cells on the selected row are read and refreshed. Shortly after the row address is loaded, the column address is applied to the address pins and loaded into the column address latch under control of the Column Address Strobe (CAS) signal. The information in this latch is decoded and the appropriate group of 8 Sense/Write circuits are selected. If the R/$\overline{W}$ control signal indicates a Read operation, the output values of the selected circuits are transferred to the data lines, D_{7-0}. For a Write operation, the information on the D_{7-0} lines is transferred to the selected circuits. This information is then used to overwrite the contents of the selected cells in the corresponding 8 columns. We should note that in commercial DRAM chips, the RAS and CAS control signals are active low so that they cause the latching of addresses when they change from high to low. To indicate this fact, these signals are shown on diagrams as $\overline{RAS}$ and $\overline{CAS}$.

Applying a row address causes all cells on the corresponding row to be read and refreshed during both Read and Write operations. To ensure that the contents of a DRAM are maintained, each row of cells must be accessed periodically. A *refresh circuit* usually performs this function automatically. Many dynamic memory chips incorporate a refresh facility within the chips themselves. In this case, the dynamic nature of these memory chips is almost invisible to the user.

In the DRAM described in this section, the timing of the memory device is controlled asynchronously. A specialized memory controller circuit provides the necessary control signals, RAS and CAS, that govern the timing. The processor must take into account the delay in the response of the memory. Such memories are referred to as *asynchronous DRAMs*.

Because of their high density and low cost, DRAMs are widely used in the memory units of computers. Available chips range in size from 1M to 256M bits, and even larger chips are being developed. To reduce the number of memory chips needed in a given computer, a DRAM chip is organized to read or write a number of bits in parallel, as indicated in Figure 5.7. To provide flexibility in designing memory systems, these chips are manufactured in different organizations. For example, a 64-Mbit chip may be organized as 16M × 4, 8M × 8, or 4M × 16.

Fast Page Mode

When the DRAM in Figure 5.7 is accessed, the contents of all 4096 cells in the selected row are sensed, but only 8 bits are placed on the data lines D_{7-0}. This byte is selected by the column address bits A_{8-0}. A simple modification can make it possible to access the other bytes in the same row without having to reselect the row. A latch can be added at the output of the sense amplifier in each column. The application of a row address will load the latches corresponding to all bits in the selected row. Then, it is only necessary to apply different column addresses to place the different bytes on the data lines.

The most useful arrangement is to transfer the bytes in sequential order, which is achieved by applying a consecutive sequence of column addresses under the control

of successive CAS signals. This scheme allows transferring a block of data at a much faster rate than can be achieved for transfers involving random addresses. The block transfer capability is referred to as the *fast page mode* feature. (Popular jargon refers to small groups of bytes as blocks, and larger groups as pages.)

The faster rate attainable in block transfers can be exploited in applications in which memory accesses follow regular patterns, such as in graphics terminals. This feature is also beneficial in general-purpose computers for transferring data blocks between the main memory and a cache, as we will explain in Section 5.5.

5.2.4 SYNCHRONOUS DRAMS

More recent developments in memory technology have resulted in DRAMs whose operation is directly synchronized with a clock signal. Such memories are known as *synchronous DRAMs* (SDRAMs). Figure 5.8 indicates the structure of an SDRAM. The cell array is the same as in asynchronous DRAMs. The address and data connections are buffered by means of registers. We should particularly note that the output of each

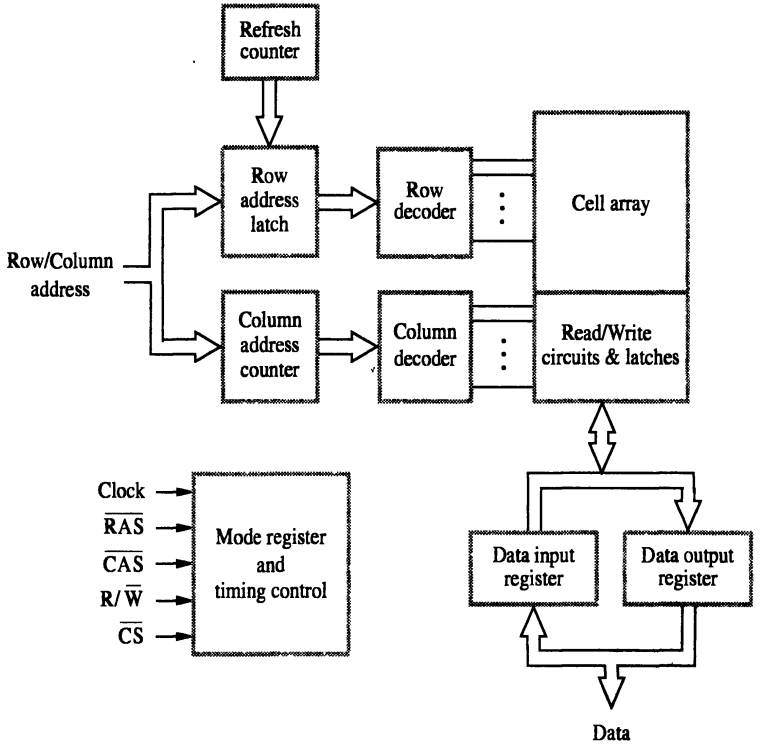

Figure 5.8 Synchronous DRAM.

sense amplifier is connected to a latch. A Read operation causes the contents of all cells in the selected row to be loaded into these latches. But, if an access is made for refreshing purposes only, it will not change the contents of these latches; it will merely refresh the contents of the cells. Data held in the latches that correspond to the selected column(s) are transferred into the data output register, thus becoming available on the data output pins.

SDRAMs have several different modes of operation, which can be selected by writing control information into a *mode* register. For example, burst operations of different lengths can be specified. The burst operations use the block transfer capability described above as the fast page mode feature. In SDRAMs, it is not necessary to provide externally generated pulses on the CAS line to select successive columns. The necessary control signals are provided internally using a column counter and the clock signal. New data can be placed on the data lines in each clock cycle. All actions are triggered by the rising edge of the clock.

Figure 5.9 shows a timing diagram for a typical burst read of length 4. First, the row address is latched under control of the $\overline{RAS}$ signal. The memory typically takes 2 or 3 clock cycles (we use 2 in the figure) to activate the selected row. Then, the column address is latched under control of the $\overline{CAS}$ signal. After a delay of one clock cycle, the first set of data bits is placed on the data lines. The SDRAM automatically increments the column address to access the next three sets of bits in the selected row, which are placed on the data lines in the next 3 clock cycles.

SDRAMs have built-in refresh circuitry. A part of this circuitry is a refresh counter, which provides the addresses of the rows that are selected for refreshing. In a typical SDRAM, each row must be refreshed at least every 64 ms.

Commercial SDRAMs can be used with clock speeds above 100 MHz. These chips are designed to meet the requirements of commercially available processors that are used

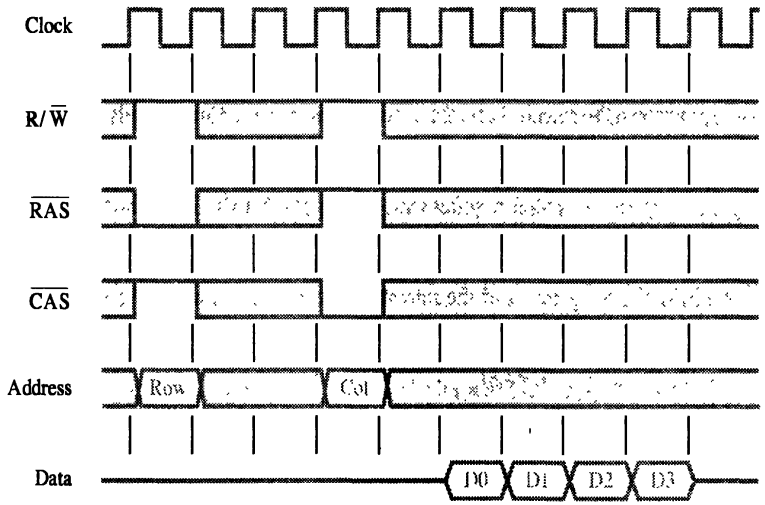

Figure 5.9 Burst read of length 4 in an SDRAM.

in large volume. For example, Intel has defined PC100 and PC133 bus specifications in which the system bus (to which the main memory is connected) is controlled by a 100 or 133 MHz clock, respectively. Therefore, major manufacturers of memory chips produce 100 and 133 MHz SDRAM chips.

Latency and Bandwidth

Transfers between the memory and the processor involve single words of data or small blocks of words (to or from the processor caches which are discussed in Section 5.5). Large blocks, constituting a page of data, are transferred between the memory and the disks, as described in Section 5.7. The speed and efficiency of these transfers have a large impact on the performance of a computer system. A good indication of the performance is given by two parameters: latency and bandwidth.

The term *memory latency* is used to refer to the amount of time it takes to transfer a word of data to or from the memory. In the case of reading or writing a single word of data, the latency provides a complete indication of memory performance. But, in the case of burst operations that transfer a block of data, the time needed to complete the operation depends also on the rate at which successive words can be transferred and on the size of the block. In block transfers, the term latency is used to denote the time it takes to transfer the first word of data. This time is usually substantially longer than the time needed to transfer each subsequent word of a block. For instance, in the timing diagram in Figure 5.9, the access cycle begins with the assertion of the $\overline{RAS}$ signal. The first word of data is transferred five clock cycles later. Thus, the latency is five clock cycles. If the clock rate is 100 MHz, then the latency is 50 ns. The remaining three words are transferred in consecutive clock cycles.

When transferring blocks of data, it is of interest to know how much time is needed to transfer an entire block. Since blocks can be variable in size, it is useful to define a performance measure in terms of the number of bits or bytes that can be transferred in one second. This measure is often referred to as the memory *bandwidth*. The bandwidth of a memory unit (consisting of one or more memory chips) depends on the speed of access to the stored data and on the number of bits that can be accessed in parallel. However, the effective bandwidth in a computer system (involving data transfers between the memory and the processor) is not determined solely by the speed of the memory; it also depends on the transfer capability of the links that connect the memory and the processor, typically the speed of the bus. Memory chips are usually designed to meet the speed requirements of popular buses. The bandwidth clearly depends on the speed of access and transmission along a single wire, as well as on the number of bits that can be transferred in parallel, namely the number of wires. Thus, the bandwidth is the product of the rate at which data are transferred (and accessed) and the width of the data bus.

Double-Data-Rate SDRAM

In the continuous quest for improved performance, a faster version of SDRAM has been developed. The standard SDRAM performs all actions on the rising edge of the clock signal. A similar memory device is available, which accesses the cell array in the same way, but transfers data on both edges of the clock. The latency of these devices is the same as for standard SDRAMs. But, since they transfer data on both edges of the

clock, their bandwidth is essentially doubled for long burst transfers. Such devices are known as *double-data-rate SDRAMs* (DDR SDRAMs).

To make it possible to access the data at a high enough rate, the cell array is organized in two banks. Each bank can be accessed separately. Consecutive words of a given block are stored in different banks. Such *interleaving* of words allows simultaneous access to two words that are transferred on successive edges of the clock. We will consider the concept of interleaving in more detail in Section 5.6.1.

DDR SDRAMs and standard SDRAMs are most efficiently used in applications where block transfers are prevalent. This is the case in general-purpose computers in which main memory transfers are primarily to and from processor caches, as we will see in Section 5.5. Block transfers are also done in high-quality video displays.

5.2.5 STRUCTURE OF LARGER MEMORIES

We have discussed the basic organization of memory circuits as they may be implemented on a single chip. Next, we should examine how memory chips may be connected to form a much larger memory.

Static Memory Systems

Consider a memory consisting of 2M (2,097,152) words of 32 bits each. Figure 5.10 shows how we can implement this memory using 512K × 8 static memory chips. Each column in the figure consists of four chips, which implement one byte position. Four of these sets provide the required 2M × 32 memory. Each chip has a control input called Chip Select. When this input is set to 1, it enables the chip to accept data from or to place data on its data lines. The data output for each chip is of the three-state type (see Section A.5.4). Only the selected chip places data on the data output line, while all other outputs are in the high-impedance state. Twenty one address bits are needed to select a 32-bit word in this memory. The high-order 2 bits of the address are decoded to determine which of the four Chip Select control signals should be activated, and the remaining 19 address bits are used to access specific byte locations inside each chip of the selected row. The R/$\overline{W}$ inputs of all chips are tied together to provide a common Read/$\overline{\text{Write}}$ control (not shown in the figure).

Dynamic Memory Systems

The organization of large dynamic memory systems is essentially the same as the memory shown in Figure 5.10. However, physical implementation is often done more conveniently in the form of *memory modules*.

Modern computers use very large memories; even a small personal computer is likely to have at least 32M bytes of memory. Typical workstations have at least 128M bytes of memory. A large memory leads to better performance because more of the programs and data used in processing can be held in the memory, thus reducing the frequency of accessing the information in secondary storage. However, if a large memory is built by placing DRAM chips directly on the main system printed-circuit board that contains the processor, often referred to as a *motherboard*, it will occupy an unacceptably large amount of space on the board. Also, it is awkward to provide for future

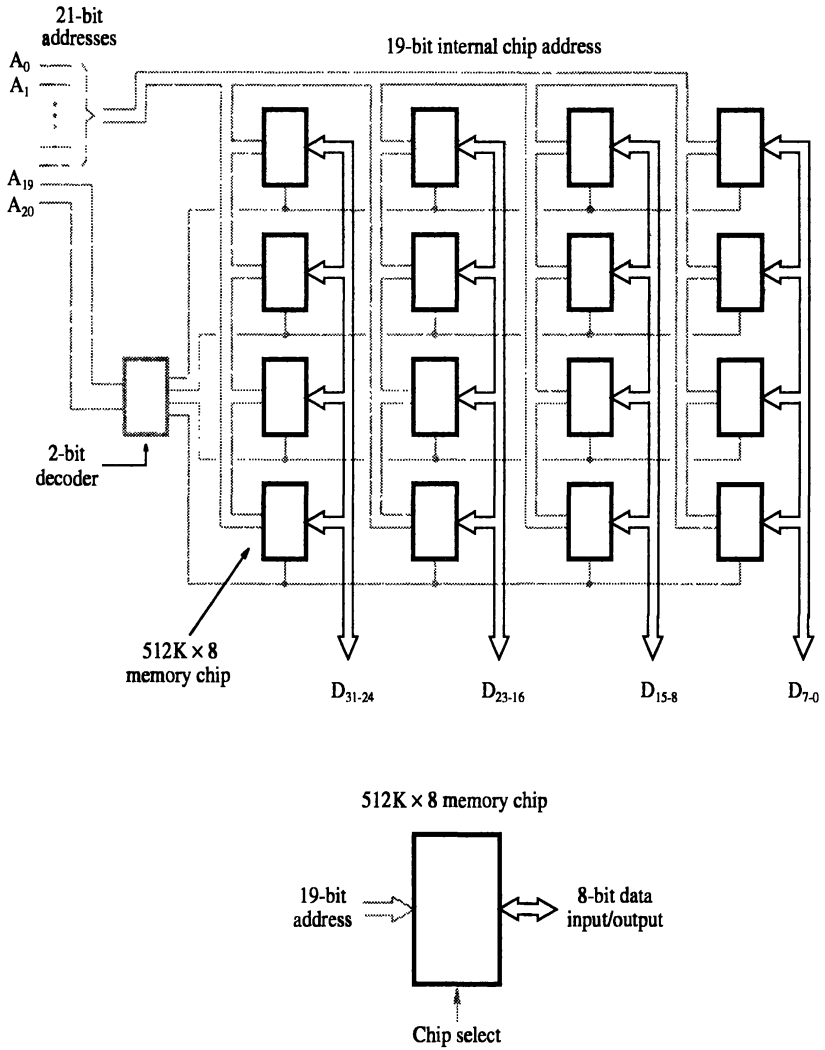

Figure 5.10 Organization of a 2M × 32 memory module using 512K × 8 static memory chips.

expansion of the memory, because space must be allocated and wiring provided for the maximum expected size. These packaging considerations have led to the development of larger memory units known as *SIMMs* (Single In-line Memory Modules) and *DIMMs* (Dual In-line Memory Modules). Such a module is an assembly of several memory chips on a separate small board that plugs vertically into a single socket on the motherboard. SIMMs and DIMMs of different sizes are designed to use the same size socket. For example, 4M × 32, 16M × 32, and 32M × 32 bit DIMMs all use the same

including the operating system software. When a computer is turned on, the operating system software has to be loaded from the disk into the memory. This requires execution of a program that "boots" the operating system. Since the boot program is quite large, most of it is stored on the disk. The processor must execute some instructions that load the boot program into the memory. If the entire memory consisted of only volatile memory chips, the processor would have no means of accessing these instructions. A practical solution is to provide a small amount of nonvolatile memory that holds the instructions whose execution results in loading the boot program from the disk.

Nonvolatile memory is used extensively in embedded systems, which are presented in Chapter 9. Such systems typically do not use disk storage devices. Their programs are stored in nonvolatile semiconductor memory devices.

Different types of nonvolatile memory have been developed. Generally, the contents of such memory can be read as if they were SRAM or DRAM memories. But, a special writing process is needed to place the information into this memory. Since its normal operation involves only reading of stored data, a memory of this type is called *read-only memory* (ROM).

5.3.1 ROM

Figure 5.12 shows a possible configuration for a ROM cell. A logic value 0 is stored in the cell if the transistor is connected to ground at point *P*; otherwise, a 1 is stored. The bit line is connected through a resistor to the power supply. To read the state of the cell, the word line is activated. Thus, the transistor switch is closed and the voltage on the bit line drops to near zero if there is a connection between the transistor and ground. If there is no connection to ground, the bit line remains at the high voltage, indicating a 1. A sense circuit at the end of the bit line generates the proper output value. Data are written into a ROM when it is manufactured.

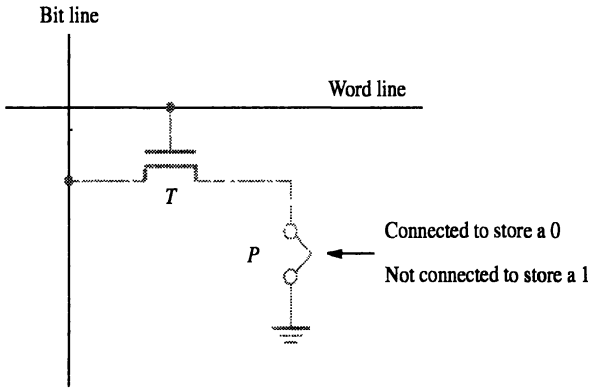

Figure 5.12 A ROM cell.

5.3.2 PROM

Some ROM designs allow the data to be loaded by the user, thus providing a *programmable ROM* (PROM). Programmability is achieved by inserting a fuse at point *P* in Figure 5.12. Before it is programmed, the memory contains all 0s. The user can insert 1s at the required locations by burning out the fuses at these locations using high-current pulses. Of course, this process is irreversible.

PROMs provide flexibility and convenience not available with ROMs. The latter are economically attractive for storing fixed programs and data when high volumes of ROMs are produced. However, the cost of preparing the masks needed for storing a particular information pattern in ROMs makes them very expensive when only a small number are required. In this case, PROMs provide a faster and considerably less expensive approach because they can be programmed directly by the user.

5.3.3 EPROM

Another type of ROM chip allows the stored data to be erased and new data to be loaded. Such an erasable, reprogrammable ROM is usually called an *EPROM*. It provides considerable flexibility during the development phase of digital systems. Since EPROMs are capable of retaining stored information for a long time, they can be used in place of ROMs while software is being developed. In this way, memory changes and updates can be easily made.

An EPROM cell has a structure similar to the ROM cell in Figure 5.12. In an EPROM cell, however, the connection to ground is always made at point *P* and a special transistor is used, which has the ability to function either as a normal transistor or as a disabled transistor that is always turned off. This transistor can be programmed to behave as a permanently open switch, by injecting charge into it that becomes trapped inside. Thus, an EPROM cell can be used to construct a memory in the same way as the previously discussed ROM cell.

The important advantage of EPROM chips is that their contents can be erased and reprogrammed. Erasure requires dissipating the charges trapped in the transistors of memory cells; this can be done by exposing the chip to ultraviolet light. For this reason, EPROM chips are mounted in packages that have transparent windows.

5.3.4 EEPROM

A significant disadvantage of EPROMs is that a chip must be physically removed from the circuit for reprogramming and that its entire contents are erased by the ultraviolet light. It is possible to implement another version of erasable PROMs that can be both programmed and erased electrically. Such chips, called EEPROMs, do not have to be removed for erasure. Moreover, it is possible to erase the cell contents selectively. The only disadvantage of EEPROMs is that different voltages are needed for erasing, writing, and reading the stored data.

5.3.5 FLASH MEMORY

An approach similar to EEPROM technology has more recently given rise to *flash memory* devices. A flash cell is based on a single transistor controlled by trapped charge, just like an EEPROM cell. While similar in some respects, there are also substantial differences between flash and EEPROM devices. In EEPROM it is possible to read and write the contents of a single cell. In a flash device it is possible to read the contents of a single cell, but it is only possible to write an entire block of cells. Prior to writing, the previous contents of the block are erased. Flash devices have greater density, which leads to higher capacity and a lower cost per bit. They require a single power supply voltage, and consume less power in their operation.

The low power consumption of flash memory makes it attractive for use in portable equipment that is battery driven. Typical applications include hand-held computers, cell phones, digital cameras, and MP3 music players. In hand-held computers and cell phones, flash memory holds the software needed to operate the equipment, thus obviating the need for a disk drive. In digital cameras, flash memory is used to store picture image data. In MP3 players, flash memory stores the data that represent sound. Cell phones, digital cameras, and MP3 players are good examples of embedded systems, which will be discussed in detail in Chapter 9.

Single flash chips do not provide sufficient storage capacity for the applications mentioned above. Larger memory modules consisting of a number of chips are needed. There are two popular choices for the implementation of such modules: flash cards and flash drives.

Flash Cards

One way of constructing a larger module is to mount flash chips on a small card. Such flash cards have a standard interface that makes them usable in a variety of products. A card is simply plugged into a conveniently accessible slot. Flash cards come in a variety of memory sizes. Typical sizes are 8, 32, and 64 Mbytes. A minute of music can be stored in about 1 Mbyte of memory, using the MP3 encoding format. Hence, a 64-MB flash card can store an hour of music.

Flash Drives

Larger flash memory modules have been developed to replace hard disk drives. These flash drives are designed to fully emulate the hard disks, to the point that they can be fitted into standard disk drive bays. However, the storage capacity of flash drives is significantly lower. Currently, the capacity of flash drives is less than one gigabyte. In contrast, hard disks can store many gigabytes.

The fact that flash drives are solid state electronic devices that have no movable parts provides some important advantages. They have shorter seek and access times, which results in faster response. (Seek and access times are discussed in the context of disks in Section 5.9.) They have lower power consumption, which makes them attractive for battery driven applications, and they are also insensitive to vibration.

The disadvantages of flash drives vis-a-vis hard disk drives are their smaller capacity and higher cost per bit. Disks provide an extremely low cost per bit. Another disadvantage is that the flash memory will deteriorate after it has been written a number of times. Fortunately, this number is high, typically at least one million times.

5.4 SPEED, SIZE, AND COST

We have already stated that an ideal memory would be fast, large, and inexpensive. From the discussion in Section 5.2, it is clear that a very fast memory can be implemented if SRAM chips are used. But these chips are expensive because their basic cells have six transistors, which precludes packing a very large number of cells onto a single chip. Thus, for cost reasons, it is impractical to build a large memory using SRAM chips. The alternative is to use Dynamic RAM chips, which have much simpler basic cells and thus are much less expensive. But such memories are significantly slower.

Although dynamic memory units in the range of hundreds of megabytes can be implemented at a reasonable cost, the affordable size is still small compared to the demands of large programs with voluminous data. A solution is provided by using secondary storage, mainly magnetic disks, to implement large memory spaces. Very large disks are available at a reasonable price, and they are used extensively in computer systems. However, they are much slower than the semiconductor memory units. So we conclude the following: A huge amount of cost-effective storage can be provided by magnetic disks. A large, yet affordable, main memory can be built with dynamic RAM technology. This leaves SRAMs to be used in smaller units where speed is of the essence, such as in cache memories.

All of these different types of memory units are employed effectively in a computer. The entire computer memory can be viewed as the hierarchy depicted in Figure 5.13. The fastest access is to data held in processor registers. Therefore, if we consider the registers to be part of the memory hierarchy, then the processor registers are at the top in terms of the speed of access. Of course, the registers provide only a minuscule portion of the required memory.

At the next level of the hierarchy is a relatively small amount of memory that can be implemented directly on the processor chip. This memory, called a *processor cache,* holds copies of instructions and data stored in a much larger memory that is provided externally. The cache memory concept was introduced in Figure 1.6 and is examined in detail in Section 5.5. There are often two levels of caches. A primary cache is always located on the processor chip. This cache is small because it competes for space on the processor chip, which must implement many other functions. The primary cache is referred to as *level 1* (L1) cache. A larger, secondary cache is placed between the primary cache and the rest of the memory. It is referred to as *level 2* (L2) cache. It is usually implemented using SRAM chips.

Including a primary cache on the processor chip and using a larger, off-chip, secondary cache is currently the most common way of designing computers. However, other arrangements can be found in practice. It is possible not to have a cache on the processor chip at all. Also, it is possible to have both L1 and L2 caches on the processor chip.

The next level in the hierarchy is called the *main memory*. This rather large memory is implemented using dynamic memory components, typically in the form of SIMMs,

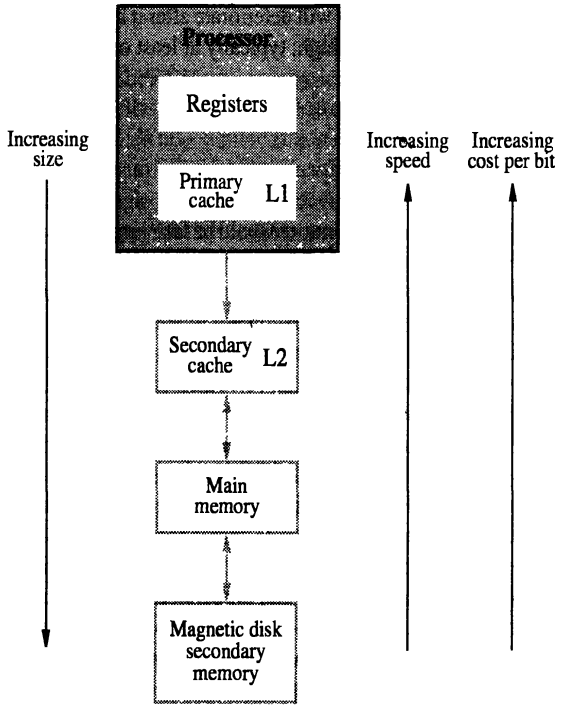

Figure 5.13 Memory hierarchy.

DIMMs, or RIMMs. The main memory is much larger but significantly slower than the cache memory. In a typical computer, the access time for the main memory is about ten times longer than the access time for the L1 cache.

Disk devices provide a huge amount of inexpensive storage. They are very slow compared to the semiconductor devices used to implement the main memory. We will discuss disk technology in Section 5.9.

During program execution, the speed of memory access is of utmost importance. The key to managing the operation of the hierarchical memory system in Figure 5.13 is to bring the instructions and data that will be used in the near future as close to the processor as possible. This can be done by using the mechanisms presented in the sections that follow. We begin with a detailed discussion of cache memories.

5.5 CACHE MEMORIES

The speed of the main memory is very low in comparison with the speed of modern processors. For good performance, the processor cannot spend much of its time waiting to access instructions and data in main memory. Hence, it is important to devise a scheme

that reduces the time needed to access the necessary information. Since the speed of the main memory unit is limited by electronic and packaging constraints, the solution must be sought in a different architectural arrangement. An efficient solution is to use a fast *cache memory* which essentially makes the main memory appear to the processor to be faster than it really is.

The effectiveness of the cache mechanism is based on a property of computer programs called locality of reference. Analysis of programs shows that most of their execution time is spent on routines in which many instructions are executed repeatedly. These instructions may constitute a simple loop, nested loops, or a few procedures that repeatedly call each other. The actual detailed pattern of instruction sequencing is not important — the point is that many instructions in localized areas of the program are executed repeatedly during some time period, and the remainder of the program is accessed relatively infrequently. This is referred to as *locality of reference*. It manifests itself in two ways: temporal and spatial. The first means that a recently executed instruction is likely to be executed again very soon. The spatial aspect means that instructions in close proximity to a recently executed instruction (with respect to the instructions' addresses) are also likely to be executed soon.

If the active segments of a program can be placed in a fast cache memory, then the total execution time can be reduced significantly. Conceptually, operation of a cache memory is very simple. The memory control circuitry is designed to take advantage of the property of locality of reference. The temporal aspect of the locality of reference suggests that whenever an information item (instruction or data) is first needed, this item should be brought into the cache where it will hopefully remain until it is needed again. The spatial aspect suggests that instead of fetching just one item from the main memory to the cache, it is useful to fetch several items that reside at adjacent addresses as well. We will use the term *block* to refer to a set of contiguous address locations of some size. Another term that is often used to refer to a cache block is *cache line*.

Consider the simple arrangement in Figure 5.14. When a Read request is received from the processor, the contents of a block of memory words containing the location specified are transferred into the cache one word at a time. Subsequently, when the program references any of the locations in this block, the desired contents are read directly from the cache. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of

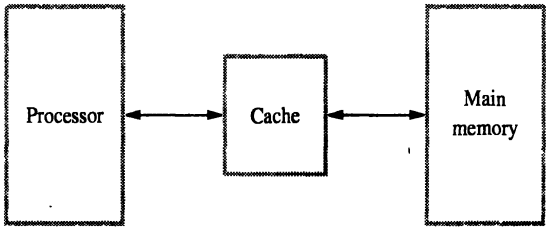

Figure 5.14 Use of a cache memory.

blocks in the main memory. The correspondence between the main memory blocks and those in the cache is specified by a mapping function. When the cache is full and a memory word (instruction or data) that is not in the cache is referenced, the cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word. The collection of rules for making this decision constitutes the *replacement algorithm.*

The processor does not need to know explicitly about the existence of the cache. It simply issues Read and Write requests using addresses that refer to locations in the memory. The cache control circuitry determines whether the requested word currently exists in the cache. If it does, the Read or Write operation is performed on the appropriate cache location. In this case, a *read* or *write hit* is said to have occurred. In a Read operation, the main memory is not involved. For a Write operation, the system can proceed in two ways. In the first technique, called the *write-through* protocol, the cache location and the main memory location are updated simultaneously. The second technique is to update only the cache location and to mark it as updated with an associated flag bit, often called the *dirty* or *modified* bit. The main memory location of the word is updated later, when the block containing this marked word is to be removed from the cache to make room for a new block. This technique is known as the *write-back,* or *copy-back,* protocol. The write-through protocol is simpler, but it results in unnecessary Write operations in the main memory when a given cache word is updated several times during its cache residency. Note that the write-back protocol may also result in unnecessary Write operations because when a cache block is written back to the memory all words of the block are written back, even if only a single word has been changed while the block was in the cache.

When the addressed word in a Read operation is not in the cache, a *read miss* occurs. The block of words that contains the requested word is copied from the main memory into the cache. After the entire block is loaded into the cache, the particular word requested is forwarded to the processor. Alternatively, this word may be sent to the processor as soon as it is read from the main memory. The latter approach, which is called *load-through,* or *early restart,* reduces the processor's waiting period somewhat, but at the expense of more complex circuitry.

During a Write operation, if the addressed word is not in the cache, a *write miss* occurs. Then, if the write-through protocol is used, the information is written directly into the main memory. In the case of the write-back protocol, the block containing the addressed word is first brought into the cache, and then the desired word in the cache is overwritten with the new information.

5.5.1 MAPPING FUNCTIONS

To discuss possible methods for specifying where memory blocks are placed in the cache, we use a specific small example. Consider a cache consisting of 128 blocks of 16 words each, for a total of 2048 (2K) words, and assume that the main memory is addressable by a 16-bit address. The main memory has 64K words, which we will view as 4K blocks of 16 words each. For simplicity, we will assume that consecutive addresses refer to consecutive words.

Direct Mapping

The simplest way to determine cache locations in which to store memory blocks is the *direct-mapping* technique. In this technique, block j of the main memory maps onto block j modulo 128 of the cache, as depicted in Figure 5.15. Thus, whenever one of the main memory blocks 0, 128, 256, ... is loaded in the cache, it is stored in cache block 0. Blocks 1, 129, 257, ... are stored in cache block 1, and so on. Since more than one memory block is mapped onto a given cache block position, contention may arise for that position even when the cache is not full. For example, instructions of a

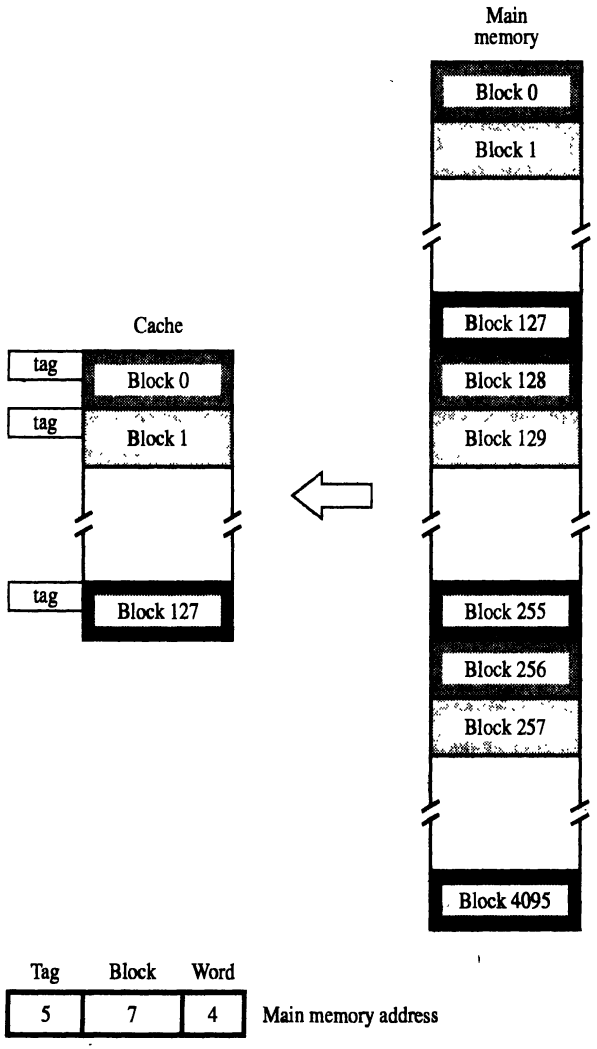

Figure 5.15 Direct-mapped cache.

program may start in block 1 and continue in block 129, possibly after a branch. As this program is executed, both of these blocks must be transferred to the block-1 position in the cache. Contention is resolved by allowing the new block to overwrite the currently resident block. In this case, the replacement algorithm is trivial.

Placement of a block in the cache is determined from the memory address. The memory address can be divided into three fields, as shown in Figure 5.15. The low-order 4 bits select one of 16 words in a block. When a new block enters the cache, the 7-bit cache block field determines the cache position in which this block must be stored. The high-order 5 bits of the memory address of the block are stored in 5 *tag* bits associated with its location in the cache. They identify which of the 32 blocks that are mapped into this cache position are currently resident in the cache. As execution proceeds, the 7-bit cache block field of each address generated by the processor points to a particular block location in the cache. The high-order 5 bits of the address are compared with the tag bits associated with that cache location. If they match, then the desired word is in that block of the cache. If there is no match, then the block containing the required word must first be read from the main memory and loaded into the cache. The direct-mapping technique is easy to implement, but it is not very flexible.

Associative Mapping

Figure 5.16 shows a much more flexible mapping method, in which a main memory block can be placed into any cache block position. In this case, 12 tag bits are required to identify a memory block when it is resident in the cache. The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to see if the desired block is present. This is called the *associative-mapping* technique. It gives complete freedom in choosing the cache location in which to place the memory block. Thus, the space in the cache can be used more efficiently. A new block that has to be brought into the cache has to replace (eject) an existing block only if the cache is full. In this case, we need an algorithm to select the block to be replaced. Many replacement algorithms are possible, as we discuss in Section 5.5.2. The cost of an associative cache is higher than the cost of a direct-mapped cache because of the need to search all 128 tag patterns to determine whether a given block is in the cache. A search of this kind is called an *associative search*. For performance reasons, the tags must be searched in parallel.

Set-Associative Mapping

A combination of the direct- and associative-mapping techniques can be used. Blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set. Hence, the contention problem of the direct method is eased by having a few choices for block placement. At the same time, the hardware cost is reduced by decreasing the size of the associative search. An example of this *set-associative-mapping* technique is shown in Figure 5.17 for a cache with two blocks per set. In this case, memory blocks 0, 64, 128, ..., 4032 map into cache set 0, and they can occupy either of the two block positions within this set. Having 64 sets means that the 6-bit set field of the address determines which set of the cache might contain the desired block. The tag field of the address must then be associatively compared to the tags of the two blocks of the set to check if the desired block is present. This two-way associative search is simple to implement.

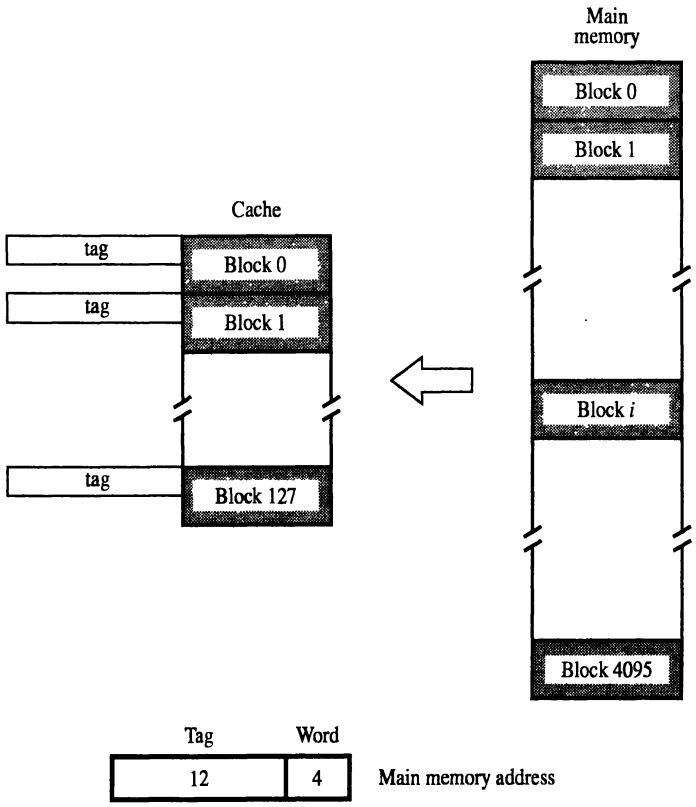

Figure 5.16 Associative-mapped cache.

The number of blocks per set is a parameter that can be selected to suit the requirements of a particular computer. For the main memory and cache sizes in Figure 5.17, four blocks per set can be accommodated by a 5-bit set field, eight blocks per set by a 4-bit set field, and so on. The extreme condition of 128 blocks per set requires no set bits and corresponds to the fully associative technique, with 12 tag bits. The other extreme of one block per set is the direct-mapping method. A cache that has k blocks per set is referred to as a k-way set-associative cache.

One more control bit, called the *valid bit,* must be provided for each block. This bit indicates whether the block contains valid data. It should not be confused with the modified, or dirty, bit mentioned earlier. The dirty bit, which indicates whether the block has been modified during its cache residency, is needed only in systems that do not use the write-through method. The valid bits are all set to 0 when power is initially applied to the system or when the main memory is loaded with new programs and data from the disk. Transfers from the disk to the main memory are carried out by a DMA mechanism. Normally, they bypass the cache for both cost and performance reasons. The valid bit of a particular cache block is set to 1 the first time this block is loaded

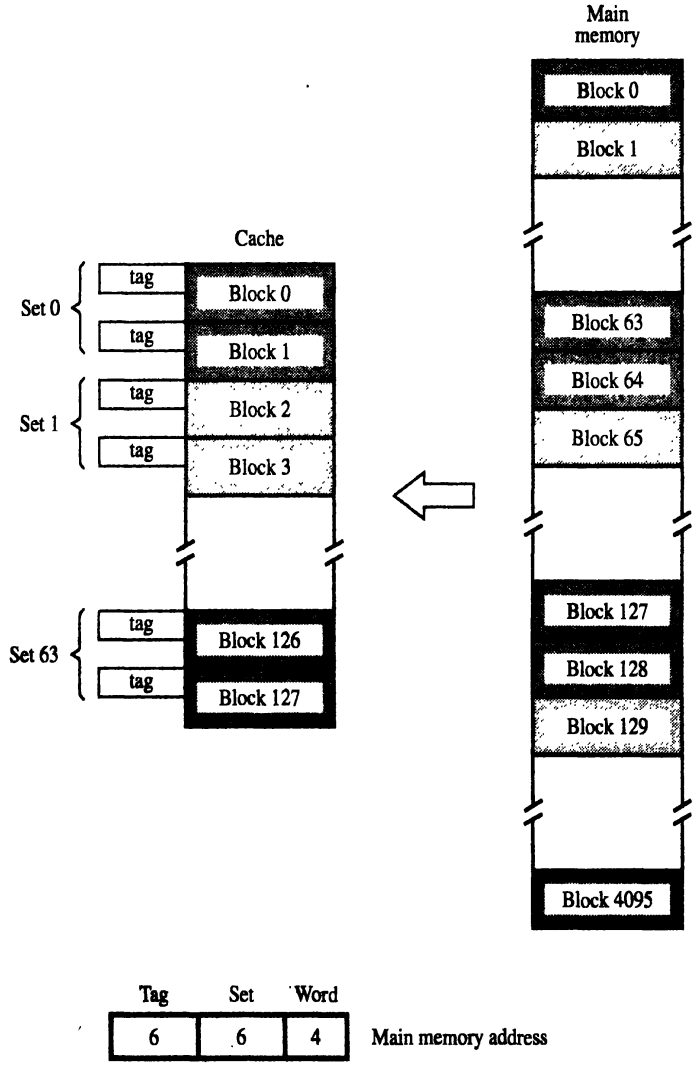

Figure 5.17 Set-associative-mapped cache with two blocks per set.

from the main memory. Whenever a main memory block is updated by a source that bypasses the cache, a check is made to determine whether the block being loaded is currently in the cache. If it is, its valid bit is cleared to 0. This ensures that *stale* data will not exist in the cache.

A similar difficulty arises when a DMA transfer is made from the main memory to the disk, and the cache uses the write-back protocol. In this case, the data in the memory might not reflect the changes that may have been made in the cached copy.

One solution to this problem is to *flush* the cache by forcing the dirty data to be written back to the memory before the DMA transfer takes place. The operating system can do this easily, and it does not affect performance greatly, because such disk transfers do not occur often. This need to ensure that two different entities (the processor and DMA subsystems in this case) use the same copies of data is referred to as a *cache-coherence* problem.

5.5.2 REPLACEMENT ALGORITHMS

In a direct-mapped cache, the position of each block is predetermined; hence, no replacement strategy exists. In associative and set-associative caches there exists some flexibility. When a new block is to be brought into the cache and all the positions that it may occupy are full, the cache controller must decide which of the old blocks to overwrite. This is an important issue because the decision can be a strong determining factor in system performance. In general, the objective is to keep blocks in the cache that are likely to be referenced in the near future. However, it is not easy to determine which blocks are about to be referenced. The property of locality of reference in programs gives a clue to a reasonable strategy. Because programs usually stay in localized areas for reasonable periods of time, there is a high probability that the blocks that have been referenced recently will be referenced again soon. Therefore, when a block is to be overwritten, it is sensible to overwrite the one that has gone the longest time without being referenced. This block is called the *least recently used* (LRU) block, and the technique is called the *LRU replacement algorithm*.

To use the LRU algorithm, the cache controller must track references to all blocks as computation proceeds. Suppose it is required to track the LRU block of a four-block set in a set-associative cache. A 2-bit counter can be used for each block. When a hit occurs, the counter of the block that is referenced is set to 0. Counters with values originally lower than the referenced one are incremented by one, and all others remain unchanged. When a *miss* occurs and the set is not full, the counter associated with the new block loaded from the main memory is set to 0, and the values of all other counters are increased by one. When a miss occurs and the set is full, the block with the counter value 3 is removed, the new block is put in its place, and its counter is set to 0. The other three block counters are incremented by one. It can be easily verified that the counter values of occupied blocks are always distinct.

The LRU algorithm has been used extensively. Although it performs well for many access patterns, it can lead to poor performance in some cases. For example, it produces disappointing results when accesses are made to sequential elements of an array that is slightly too large to fit into the cache (see Section 5.5.3 and Problem 5.12). Performance of the LRU algorithm can be improved by introducing a small amount of randomness in deciding which block to replace.

Several other replacement algorithms are also used in practice. An intuitively reasonable rule would be to remove the "oldest" block from a full set when a new block must be brought in. However, because this algorithm does not take into account the recent pattern of access to blocks in the cache, it is generally not as effective as the LRU