**FIGURE 7-23**

Construction of a Johnson counter

to a valid state. This difficulty can be corrected by modifying the circuit to avoid this undesirable condition. One correcting procedure is to disconnect the output from flip-flop  $B$  that goes to the  $D$  input of flip-flop  $C$ , and instead enable the input of flip-flop  $C$  by the function:

$$DC = (A + C)B$$

where  $DC$  is the flip-flop input function for the  $D$  input of flip-flop  $C$ .

Johnson counters can be constructed for any number of timing sequences. The number of flip-flops needed is one-half the number of timing signals. The number of decoding gates is equal to the number of timing signals and only 2-input gates are employed.

## 7-7 RANDOM-ACCESS MEMORY (RAM)

A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of the device. Memory cells can be accessed for information transfer to or from any desired random location and hence the name *random-access memory*, abbreviated RAM.

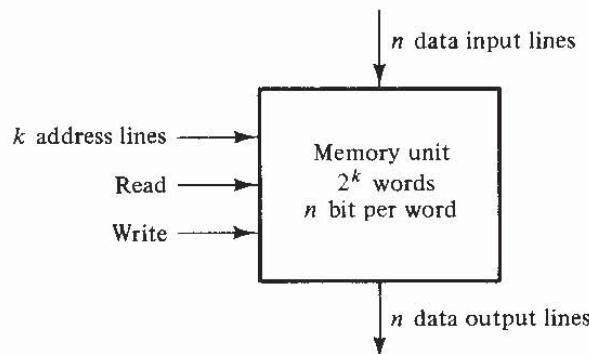
A memory unit stores binary information in groups of bits called *words*. A word in

memory is an entity of bits that move in and out of storage as a unit. A memory word is a group of 1's and 0's and may represent a number, an instruction, one or more alphanumeric characters, or any other binary-coded information. A group of eight bits is called a *byte*. Most computer memories use words that are multiples of 8 bits in length. Thus, a 16-bit word contains two bytes, and a 32-bit word is made up of four bytes. The capacity of a memory unit is usually stated as the total number of bytes that it can store.

The communication between a memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer. A block diagram of the memory unit is shown in Fig. 7-24. The  $n$  data input lines provide the information to be stored in memory and the  $n$  data output lines supply the information coming out of memory. The  $k$  address lines specify the particular word chosen among the many available. The two control inputs specify the direction of transfer desired: The write input causes binary data to be transferred into the memory, and the read input causes binary data to be transferred out of memory.

The memory unit is specified by the number of words it contains and the number of bits in each word. The address lines select one particular word. Each word in memory is assigned an identification number, called an address, starting from 0 and continuing with 1, 2, 3, up to  $2^k - 1$ , where  $k$  is the number of address lines. The selection of a specific word inside the memory is done by applying the  $k$ -bit binary address to the address lines. A decoder inside the memory accepts this address and opens the paths needed to select the word specified. Computer memories may range from 1024 words, requiring an address of 10 bits, to  $2^{32}$  words, requiring 32 address bits. It is customary to refer to the number of words (or bytes) in a memory with one of the letters K (kilo), M (mega), or G (giga). K is equal to  $2^{10}$ , M is equal to  $2^{20}$ , and G is equal to  $2^{30}$ . Thus,  $64K = 2^{16}$ ,  $2M = 2^{21}$ , and  $4G = 2^{32}$ .

Consider, for example, the memory unit with a capacity of 1K words of 16 bits each. Since  $1K = 1024 = 2^{10}$  and 16 bits constitute two bytes, we can say that the memory can accommodate  $2048 = 2K$  bytes. Figure 7-25 shows the possible content of the first three and the last three words of this memory. Each word contains 16 bits,



**FIGURE 7-24**  
Block diagram of a memory unit

Memory address		Memory content
Binary	decimal	
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
:	:	:
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

**FIGURE 7-25**  
Content of a  $1024 \times 16$  memory

which can be divided into two bytes. The words are recognized by their decimal address from 0 to 1023. The equivalent binary address consists of 10 bits. The first address is specified with ten 0's, and the last address is specified with ten 1's. This is because 1023 in binary is equal to 1111111111. A word in memory is selected by its binary address. When a word is read or written, the memory operates on all 16 bits as a single unit.

The  $1K \times 16$  memory of Fig. 7-25 has 10 bits in the address and 16 bits in each word. As another example, a  $64K \times 10$  memory will have 16 bits in the address (since  $64K = 2^{16}$ ) and each word will consist of 10 bits. The number of address bits needed in a memory is dependent on the total number of words that can be stored in the memory and is independent of the number of bits in each word. The number of bits in the address is determined from the relationship  $2^k = m$ , where  $m$  is the total number of words, and  $k$  is the number of address bits.

### Write and Read Operations

The two operations that a random-access memory can perform are the write and read operations. The write signal specifies a transfer-in operation and the read signal specifies a transfer-out operation. On accepting one of these control signals, the internal circuits inside the memory provide the desired function. The steps that must be taken for the purpose of transferring a new word to be stored into memory are as follows:

1. Transfer the binary address of the desired word to the address lines.
2. Transfer the data bits that must be stored in memory to the data input lines.
3. Activate the *write* input.

**TABLE 7-7**  
**Control Inputs to Memory Chip**

Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

The memory unit will then take the bits from the input data lines and store them in the word specified by the address lines.

The steps that must be taken for the purpose of transferring a stored word out of memory are as follows:

1. Transfer the binary address of the desired word to the address lines.
2. Activate the *read* input.

The memory unit will then take the bits from the word that has been selected by the address and apply them to the output data lines. The content of the selected word does not change after reading.

Commercial memory components available in integrated-circuit chips sometimes provide the two control inputs for reading and writing in a somewhat different configuration. Instead of having separate read and write inputs to control the two operations, some integrated circuits provide two other control inputs: one input selects the unit and the other determines the operation. The memory operations that result from these control inputs are specified in Table 7-7.

The memory enable (sometimes called the chip select) is used to enable the particular memory chip in a multichip implementation of a large memory. When the memory enable is inactive, the memory chip is not selected and no operation is performed. When the memory enable input is active, the read/write input determines the operation to be performed.

### Types of Memories

The mode of access of a memory system is determined by the type of components used. In a random-access memory, the word locations may be thought of as being separated in space, with each word occupying one particular location. In a sequential-access memory, the information stored in some medium is not immediately accessible, but is available only at certain intervals of time. A magnetic-tape unit is of this type. Each memory location passes the read and write heads in turn, but information is read out only when the requested word has been reached. The *access time* of a memory is the time required to select a word and either read or write it. In a random-access memory, the access time is always the same regardless of the particular location of the word. In a sequential-access memory, the time it takes to access a word depends on the position of the word with respect to the reading-head position and therefore, the access time is variable.

Integrated-circuit RAM units are available in two possible operating modes, *static* and *dynamic*. The static RAM consists essentially of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit. The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors. The capacitors are provided inside the chip by MOS transistors. The stored charge on the capacitors tends to discharge with time and the capacitors must be periodically recharged by *refreshing* the dynamic memory. Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. Dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip, but static RAM is easier to use and has shorter read and write cycles.

Memory units that lose the stored information when power is turned off are said to be *volatile*. Integrated-circuit RAMs, both static and dynamic, are of this category since the binary cells need external power to maintain the stored information. In contrast, a nonvolatile memory, such as magnetic disk, retains its stored information after removal of power. This is because the data stored on magnetic components is manifested by the direction of magnetization, which is retained after power is turned off. Another nonvolatile memory is the read-only memory (ROM) discussed in Section 5-7. A nonvolatile property is desirable in digital computers to store programs that are needed while the computer is in operation. Programs and data that cannot be altered are stored in ROM. Other large programs are maintained on magnetic disks. When power is turned on, the computer can use the programs from ROM. The other programs residing on disks can be transferred into the computer RAM as needed. Before turning the power off, the user transfers the binary information from the computer RAM into a disk if this information must be retained.

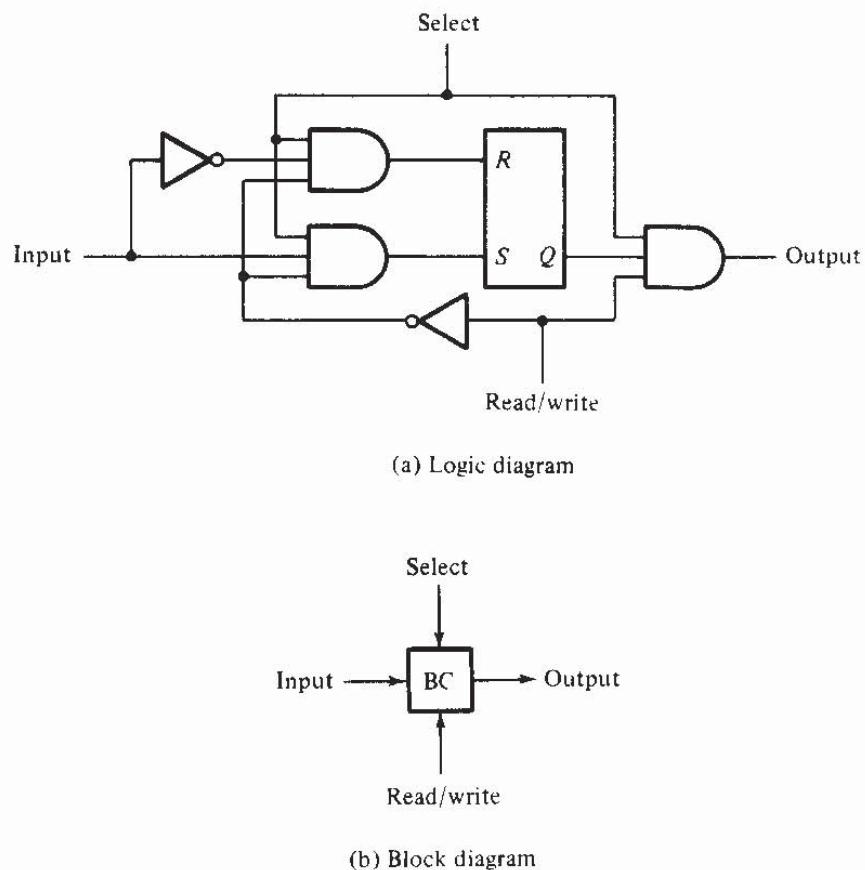
## 7-8 MEMORY DECODING

---

In addition to the storage components in a memory unit, there is a need for decoding circuits to select the memory word specified by the input address. In this section, we present the internal construction of a random-access memory and demonstrate the operation of the decoder. To be able to include the entire memory in one diagram, the memory unit presented here has a small capacity of 12 bits arranged in 4 words of 3 bits each. In addition to internal decoders, a memory unit may also need external decoders. This happens when integrated-circuit RAM chips are connected in a multichip memory configuration. The use of an external decoder to provide a large capacity memory will be demonstrated by means of an example.

### Internal Construction

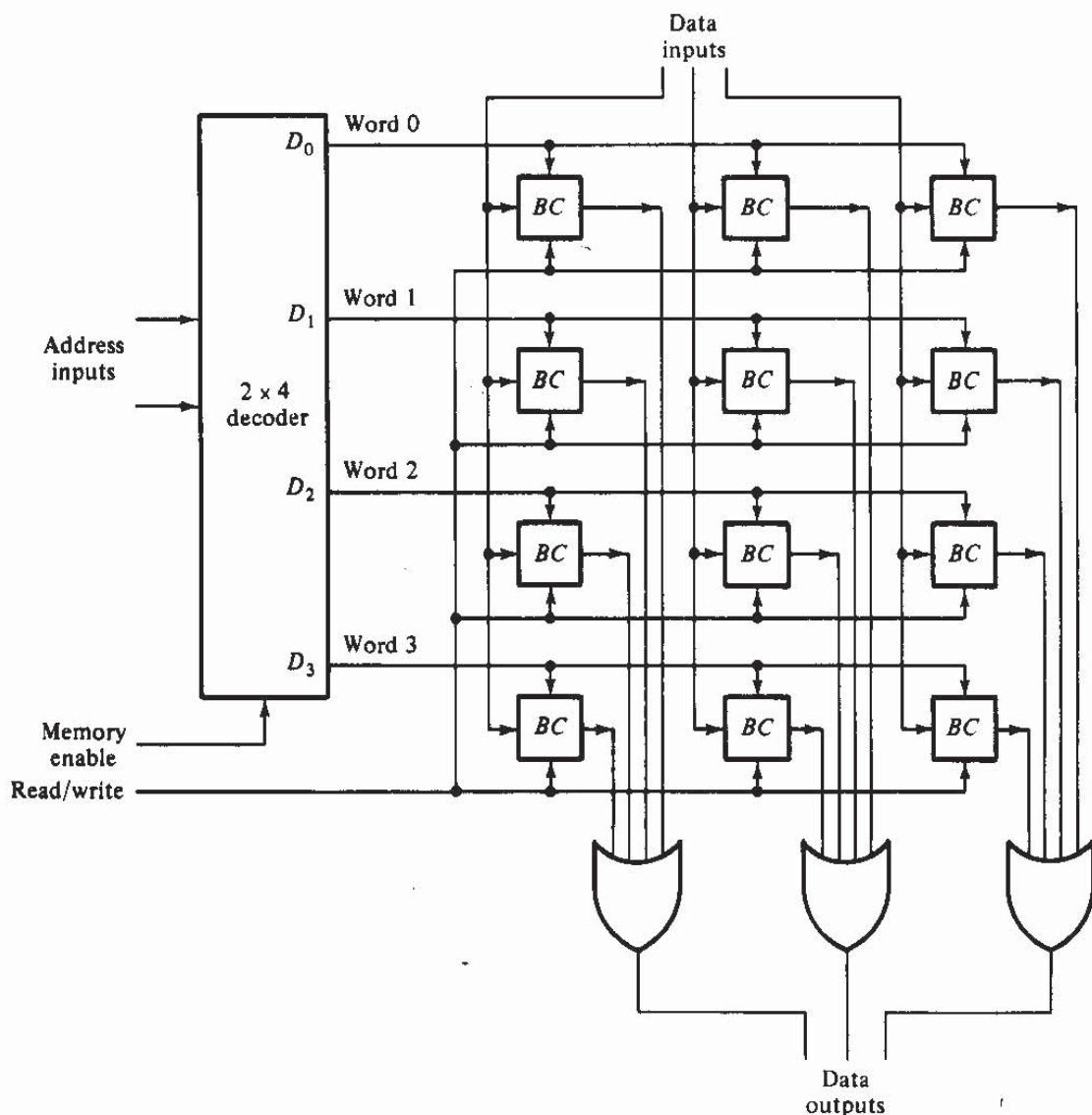
The internal construction of a random-access memory of  $m$  words with  $n$  bits per word consists of  $m \times n$  binary storage cells and associated decoding circuits for selecting individual words. The binary storage cell is the basic building block of a memory unit.

**FIGURE 7-26**

Memory cell

The equivalent logic of a binary cell that stores one bit of information is shown in Fig. 7-26. Although the cell is shown to include gates and a flip-flop, internally, it is constructed with two transistors having multiple inputs. A binary storage cell must be very small in order to be able to pack as many cells as possible in the area available in the integrated-circuit chip. The binary cell stores one bit in its internal flip-flop. It has three inputs and one output. The select input enables the cell for reading or writing and the read/write input determines the cell operation when it is selected. A 1 in the read/write input provides the read operation by forming a path from the flip-flop to the output terminal. A 0 in the read/write input provides the write operation by forming a path from the input terminal to the flip-flop. Note that the flip-flop operates without a clock and is similar to an SR latch (see Fig. 6-2).

The logical construction of a small RAM is shown in Fig. 7-27. It consists of 4 words of 3 bits each and has a total of 12 binary cells. Each block labeled BC represents the binary cell with its three inputs and one output, as specified in Fig. 7-26(b). A memory with four words needs two address lines. The two address inputs go through a  $2 \times 4$  decoder to select one of the four words. The decoder is enabled with the memory-enable input. When the memory enable is 0, all outputs of the decoder are 0

**FIGURE 7-27**Logical construction of a  $4 \times 3$  RAM

and none of the memory words are selected. With the memory enable at 1, one of the four words is selected, dictated by the value in the two address lines. Once a word has been selected, the read/write input determines the operation. During the read operation, the four bits of the selected word go through OR gates to the output terminals. During the write operation, the data available in the input lines are transferred into the four binary cells of the selected word. The binary cells that are not selected are disabled and their previous binary values remain unchanged. When the memory-enable input that goes into the decoder is equal to 0, none of the words are selected and the contents of all cells remain unchanged regardless of the value of the read/write input.

Commercial random-access memories may have a capacity of thousands of words and each word may range from 1 to 64 bits. The logical construction of a large capacity memory would be a direct extension of the configuration shown here. A memory with  $2^k$  words of  $n$  bits per word requires  $k$  address lines that go into a  $k \times 2^k$  decoder. Each one of the decoder outputs selects one word of  $n$  bits for reading or writing.

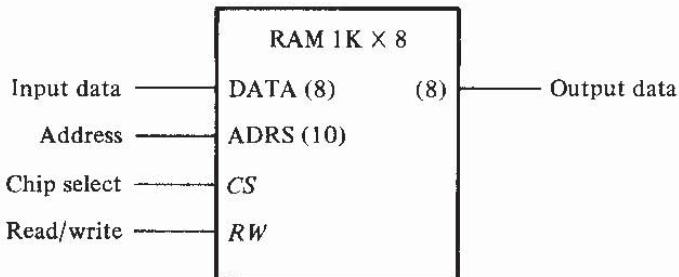
### Array of RAM Chips

Integrated-circuit RAM chips are available in a variety of sizes. If the memory unit needed for an application is larger than the capacity of one chip, it is necessary to combine a number of chips in an array to form the required memory size. The capacity of the memory depends on two parameters: the number of words and the number of bits per word. An increase in the number of words requires that we increase the address length. Every bit added to the length of the address doubles the number of words in memory. The increase in the number of bits per word requires that we increase the length of the data input and output lines, but the address length remains the same.

To demonstrate with an example, let us first introduce a typical RAM chip, as shown in Fig. 7-28. The capacity of the RAM is 1024 words of 8 bits each. It requires a 10-bit address and 8 input and output lines. These are shown in the block diagram by a single line and a number indicating the total number of inputs or outputs. The chip-select (CS) input selects the particular RAM chip and the read/write (RW) input specifies the read or write operation when the chip is selected.

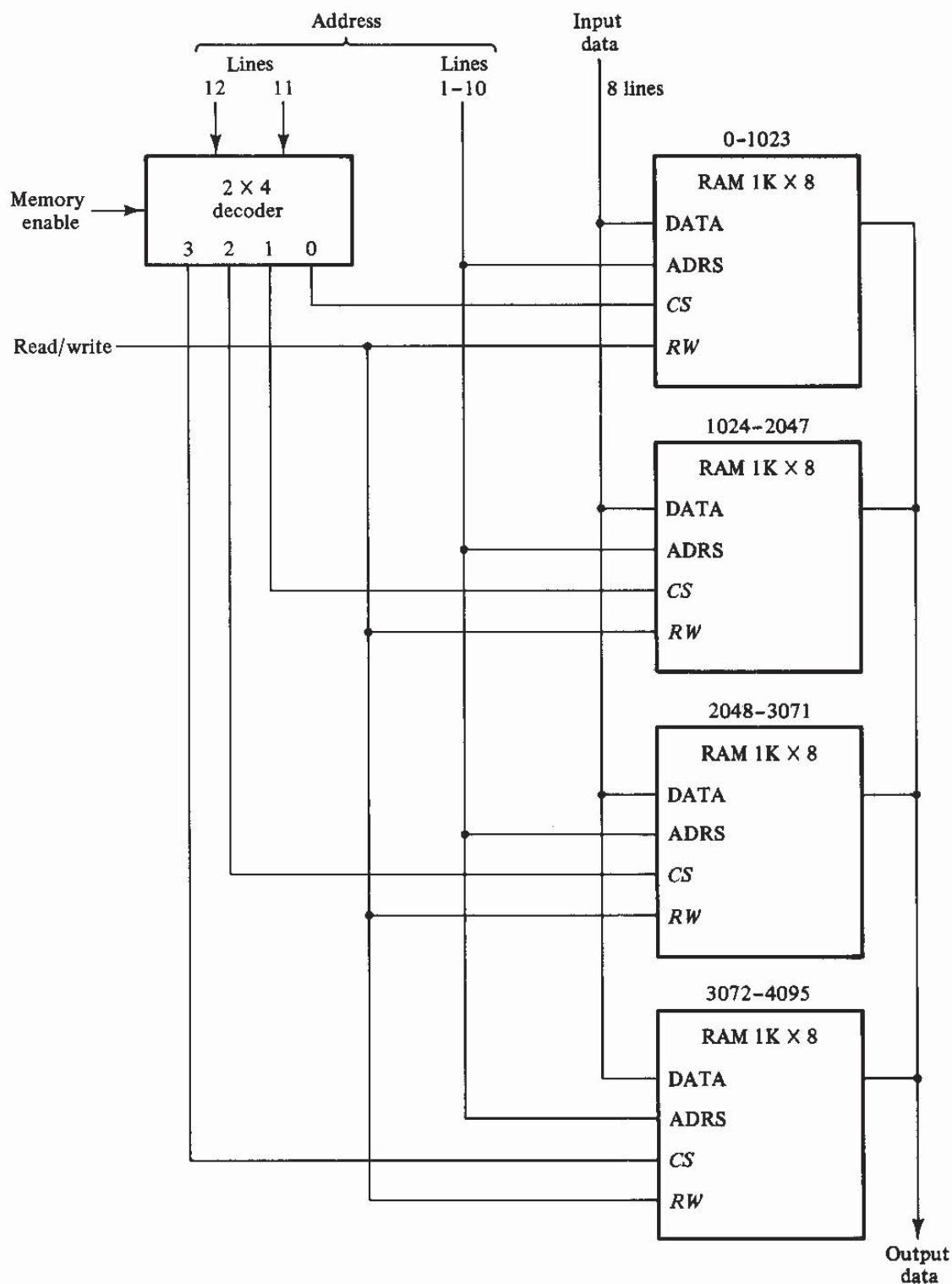
Suppose that we want to increase the number of words in the memory by using two or more RAM chips. Since every bit added to the address doubles the binary number that can be formed, it is natural to increase the number of words in factors of 2. For example, two RAM chips will double the number of words and add one bit to the composite address. Four RAM chips multiply the number of words by 4 and add two bits to the composite address.

Consider the possibility of constructing a  $4K \times 8$  RAM with four  $1K \times 8$  RAM chips. This is shown in Fig. 7-29. The 8 input data lines go to all the chips. The outputs must be ORed together to form the common 8 output data lines. (The OR gates are not shown in the diagram.) The 4K word memory requires a 12-bit address. The 10 least significant bits of the address are applied to the address inputs of all four chips.



**FIGURE 7-28**

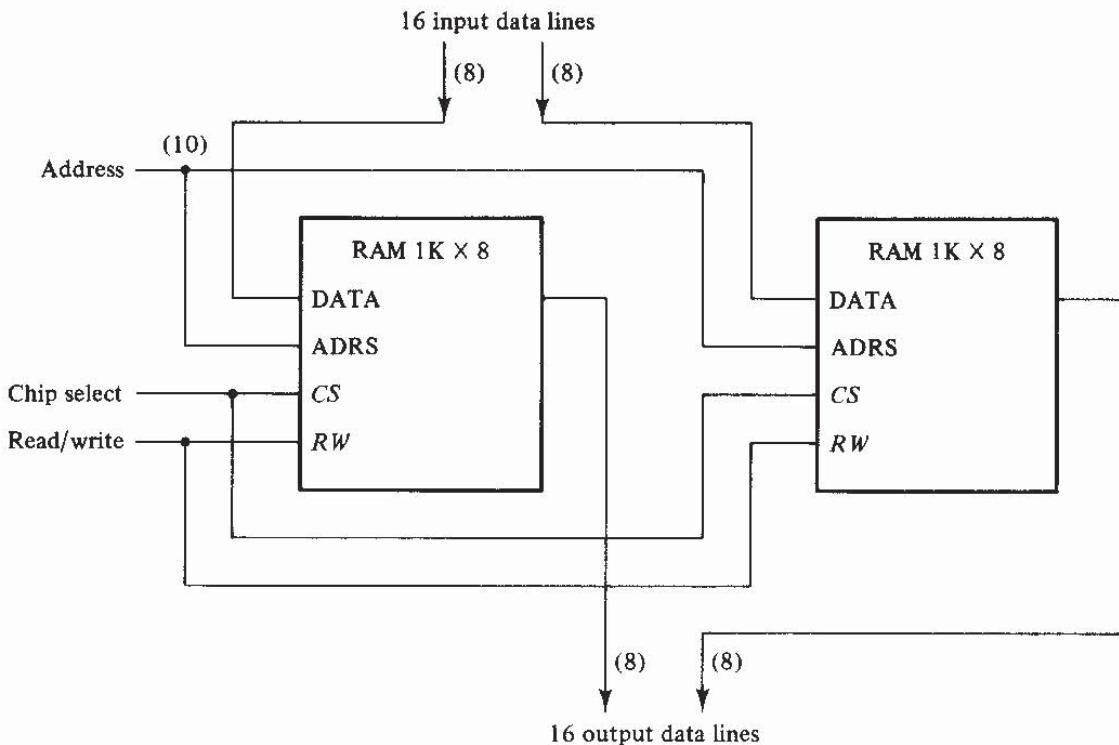
Block diagram of a  $1K \times 8$  RAM chip.

**FIGURE 7-29**Block diagram of a  $4K \times 8$  RAM.

The other two most significant bits are applied to a  $2 \times 4$  decoder. The four outputs of the decoder are applied to the CS inputs of each chip. The memory is disabled when the memory-enable input of the decoder is equal to 0. This causes all four outputs of the decoder to be in the 0 state and none of the chips are selected. When the decoder is enabled, address bits 12 and 11 determine the particular chip that is selected. If bits 12 and 11 are equal to 00, the first RAM chip is selected. The remaining ten address bits select a word within the chip in the range from 0 to 1023. The next 1024 words are selected from the second RAM chip with a 12-bit address that starts with 01 and follows by the ten bits from the common address lines. The address range for each chip is listed in decimal over its block diagram in Fig. 7-29.

It is also possible to combine two chips to form a composite memory containing the same number of words but with twice as many bits in each word. Figure 7-30 shows the interconnection of two  $1K \times 8$  chips to form a  $1K \times 16$  memory. The 16 input and output data lines are split between the two chips. Both receive the same 10-bit address and the common CS and RW control inputs.

The two techniques just described may be combined to assemble an array of identical chips into a large-capacity memory. The composite memory will have a number of bits per word that is a multiple of that for one chip. The total number of words will increase in factors of 2 times the word capacity of one chip. An external decoder is needed to



**FIGURE 7-30**

Block diagram of a  $1K \times 16$  RAM.

select the individual chips from the additional address bits of the composite memory.

To reduce the number of pins in the package, many RAM integrated circuits provide common terminals for the input data and output data. The common terminals are said to be *bidirectional*, which means that for the read operation, they act as outputs, and for the write operation, they act as inputs.

## 7-9 ERROR-CORRECTING CODE

---

The complexity level of a memory array may cause occasional errors in storing and retrieving the binary information. The reliability of a memory unit may be improved by employing error-detecting and correcting codes. The most common error-detection scheme is the parity bit. (See Section 4-9.) A parity bit is generated and stored along with the data word in memory. The parity of the word is checked after reading it from memory. The data word is accepted if the parity sense is correct. If the parity checked results in an inversion, an error is detected, but it cannot be corrected.

An error-correcting code generates multiple check bits that are stored with the data word in memory. Each check bit is a parity over a group of bits in the data word. When the word is read from memory, the associated parity bits are also read from memory and compared with a new set of check bits generated from the read data. If the check bits compare, it signifies that no error has occurred. If the check bits do not compare with the stored parity, they generate a unique pattern, called a *syndrome*, that can be used to identify the bit in error. A single error occurs when a bit changes in value from 1 to 0 or from 0 to 1 during the write or read operation. If the specific bit in error is identified, then the error can be corrected by complementing the erroneous bit.

### Hamming Code

One of the most common error-correcting codes used in random-access memories was devised by R. W. Hamming. In the Hamming code,  $k$  parity bits are added to an  $n$ -bit data word, forming a new word of  $n + k$  bits. The bit positions are numbered in sequence from 1 to  $n + k$ . Those positions numbered as a power of 2 are reserved for the parity bits. The remaining bits are the data bits. The code can be used with words of any length. Before giving the general characteristics of the code, we will illustrate its operation with a data word of eight bits.

Consider, for example, the 8-bit data word 11000100. We include four parity bits with the 8-bit word and arrange the 12 bits as follows:

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	$P_1$	$P_2$	1	$P_4$	1	0	0	$P_8$	0	1	0	0

The four parity bits,  $P_1$ ,  $P_2$ ,  $P_4$ , and  $P_8$ , are in positions 1, 2, 4, and 8, respectively. The eight bits of the data word are in the remaining positions. Each parity bit is calculated as follows:

minterms are associated with  $A'$  and the second half with  $A$ . By circling the minterms of the function and applying the rules for finding values for the multiplexer inputs, we obtain the implementation shown. ■

Let us now compare the multiplexer method with the decoder method for implementing combinational circuits. The decoder method requires an OR gate for each output function, but only one decoder is needed to generate all minterms. The multiplexer method uses smaller-size units but requires one multiplexer for each output function. It would seem reasonable to assume that combinational circuits with a small number of outputs should be implemented with multiplexers. Combinational circuits with many output functions would probably use fewer ICs with the decoder method.

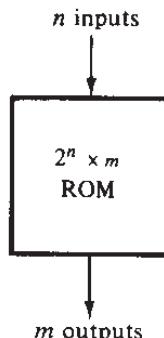
Although multiplexers and decoders may be used in the implementation of combinational circuits, it must be realized that decoders are mostly used for decoding binary information and multiplexers are mostly used to form a selected path between multiple sources and a single destination.

## 5-7 READ-ONLY MEMORY (ROM)

We saw in Section 5-5 that a decoder generates the  $2^n$  minterms of the  $n$  input variables. By inserting OR gates to sum the minterms of Boolean functions, we were able to generate any desired combinational circuit. A read-only memory (ROM) is a device that includes both the decoder and the OR gates within a single IC package. The connections between the outputs of the decoder and the inputs of the OR gates can be specified for each particular configuration. The ROM is used to implement complex combinational circuits within one IC package or as permanent storage for binary information.

A ROM is essentially a memory (or storage) device in which permanent binary information is stored. The binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern. ROMs come with special internal electronic fuses that can be “programmed” for a specific configuration. Once the pattern is established, it stays within the unit even when power is turned off and on again.

A block diagram of a ROM is shown in Fig. 5-21. It consists of  $n$  input lines and  $m$  output lines. Each bit combination of the input variables is called an *address*. Each bit combination that comes out of the output lines is called a *word*. The number of bits per word is equal to the number of output lines,  $m$ . An address is essentially a binary number that denotes one of the minterms of  $n$  variables. The number of distinct addresses possible with  $n$  input variables is  $2^n$ . An output word can be selected by a unique address, and since there are  $2^n$  distinct addresses in a ROM, there are  $2^n$  distinct words that are said to be stored in the unit. The word available on the output lines at any given time depends on the address value applied to the input lines. A ROM is charac-



**FIGURE 5-21**  
ROM block diagram

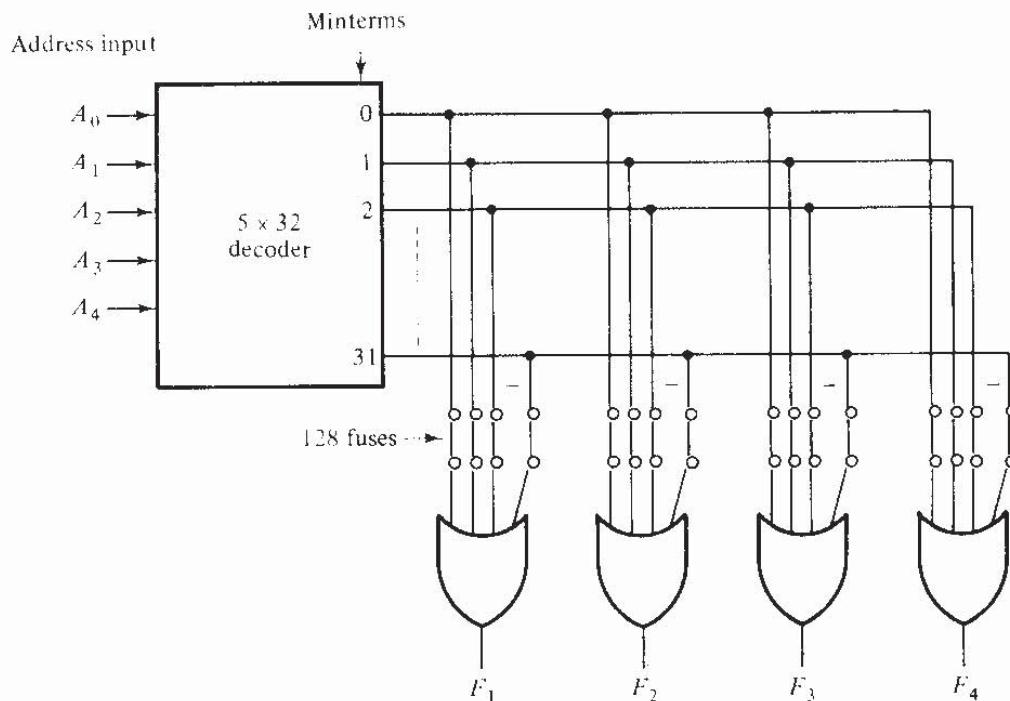
terized by the number of words  $2^n$  and the number of bits per word  $m$ . This terminology is used because of the similarity between the read-only memory and the random-access memory, which is presented in Section 7-7.

Consider a  $32 \times 8$  ROM. The unit consists of 32 words of 8 bits each. This means that there are eight output lines and that there are 32 distinct words stored in the unit, each of which may be applied to the output lines. The particular word selected that is presently available on the output lines is determined from the five input lines. There are only five inputs in a  $32 \times 8$  ROM because  $2^5 = 32$ , and with five variables, we can specify 32 addresses or minterms. For each address input, there is a unique selected word. Thus, if the input address is 00000, word number 0 is selected and it appears on the output lines. If the input address is 11111, word number 31 is selected and applied to the output lines. In between, there are 30 other addresses that can select the other 30 words.

The number of addressed words in a ROM is determined from the fact that  $n$  input lines are needed to specify  $2^n$  words. A ROM is sometimes specified by the total number of bits it contains, which is  $2^n \times m$ . For example, a 2048-bit ROM may be organized as 512 words of 4 bits each. This means that the unit has four output lines and nine input lines to specify  $2^9 = 512$  words. The total number of bits stored in the unit is  $512 \times 4 = 2048$ .

Internally, the ROM is a combinational circuit with AND gates connected as a decoder and a number of OR gates equal to the number of outputs in the unit. Figure 5-22 shows the internal logic construction of a  $32 \times 4$  ROM. The five input variables are decoded into 32 lines by means of 32 AND gates and 5 inverters. Each output of the decoder represents one of the minterms of a function of five variables. Each one of the 32 addresses selects one and only one output from the decoder. The address is a 5-bit number applied to the inputs, and the selected minterm out of the decoder is the one marked with the equivalent decimal number. The 32 outputs of the decoder are connected through fuses to each OR gate. Only four of these fuses are shown in the diagram, but actually each OR gate has 32 inputs and each input goes through a fuse that can be blown as desired.

The ROM is a two-level implementation in sum of minterms form. It does not have to be an AND-OR implementation, but it can be any other possible two-level minterm

**FIGURE 5-22**Logic construction of a  $32 \times 4$  ROM

implementation. The second level is usually a wired-logic connection (see Section 3-7) to facilitate the blowing of fuses.

ROMs have many important applications in the design of digital computer systems. Their use for implementing complex combinational circuits is just one of these applications. Other uses of ROMs are presented in other parts of the book in conjunction with their particular applications.

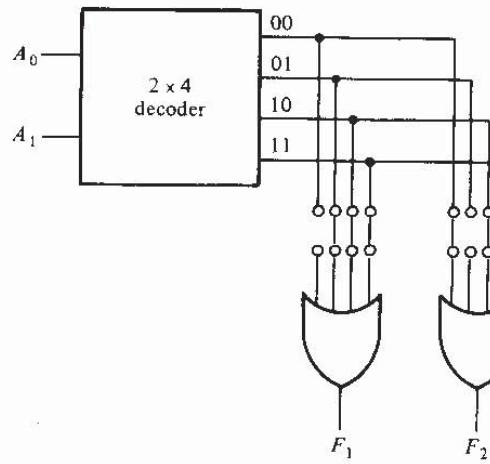
### Combinational Logic Implementation

From the logic diagram of the ROM, it is clear that each output provides the sum of all the minterms of the  $n$  input variables. Remember that any Boolean function can be expressed in sum of minterms form. By breaking the links of those minterms not included in the function, each ROM output can be made to represent the Boolean function of one of the output variables in the combinational circuit. For an  $n$ -input,  $m$ -output combinational circuit, we need a  $2^n \times m$  ROM. The blowing of the fuses is referred to as *programming* the ROM. The designer need only specify a ROM program table that gives the information for the required paths in the ROM. The actual programming is a hardware procedure that follows the specifications listed in the program table.

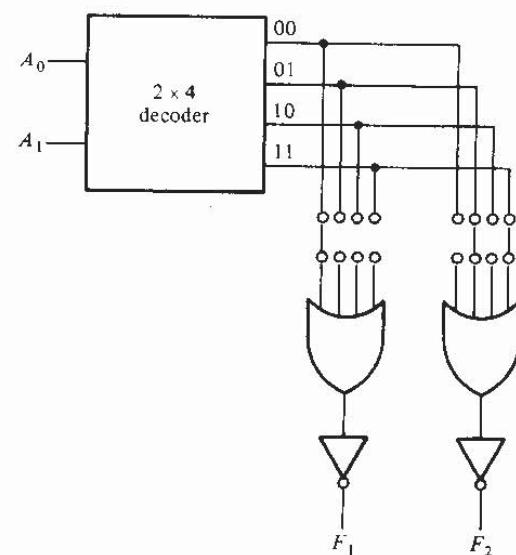
Let us clarify the process with a specific example. The truth table in Fig. 5-23(a) specifies a combinational circuit with two inputs and two outputs. The Boolean functions can be expressed in sum of minterms:

$A_1$	$A_0$	$F_1$	$F_2$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

(a) Truth table



(b) ROM with AND-OR gates



(c) ROM with AND-OR-INVERT gates

**FIGURE 5-23**Combinational-circuit implementation with a  $4 \times 2$  ROM

$$F_1(A_1, A_0) = \Sigma(1, 2, 3)$$

$$F_2(A_1, A_0) = \Sigma(0, 2)$$

When a combinational circuit is implemented by means of a ROM, the functions must be expressed in sum of minterms or, better yet, by a truth table. If the output functions are simplified, we find that the circuit needs only one OR gate and an inverter. Obviously, this is too simple a combinational circuit to be implemented with a ROM. The advantage of a ROM is in complex combinational circuits. This example merely demonstrates the procedure and should not be considered in a practical situation.

The ROM that implements the combinational circuit must have two inputs and two outputs; so its size must be  $4 \times 2$ . Figure 5-23(b) shows the internal construction of such a ROM. It is now necessary to determine which of the eight available fuses must be blown and which should be left intact. This can be easily done from the output functions listed in the truth table. Those minterms that specify an output of 0 should not have a path to the output through the OR gate. Thus, for this particular case, the truth table shows three 0's, and their corresponding fuses to the OR gates must be blown. It

is obvious that we must assume here that an open input to an OR gate behaves as a 0 input.

Some ROM units come with an inverter after each of the OR gates and, as a consequence, they are specified as having initially all 0's at their outputs. The programming procedure in such ROMs requires that we open the paths of the minterms (or addresses) that specify an output of 1 in the truth table. The output of the OR gate will then generate the complement of the function, but the inverter placed after the OR gate complements the function once more to provide the normal output. This is shown in the ROM of Fig. 5-23(c).

The previous example demonstrates the general procedure for implementing any combinational circuit with a ROM. From the number of inputs and outputs in the combinational circuit, we first determine the size of ROM required. Then we must obtain the programming truth table of the ROM; no other manipulation or simplification is required. The 0's (or 1's) in the output functions of the truth table directly specify those fuses that must be blown to provide the required combinational circuit in sum of minterms form.

In practice, when one designs a circuit by means of a ROM, it is not necessary to show the internal gate connections of fuses inside the unit, as was done in Fig. 5-23. This was shown there for demonstration purposes only. All the designer has to do is specify the particular ROM (or its designation number) and provide the ROM truth table, as in Fig. 5-23(a). The truth table gives all the information for programming the ROM. No internal logic diagram is needed to accompany the truth table.

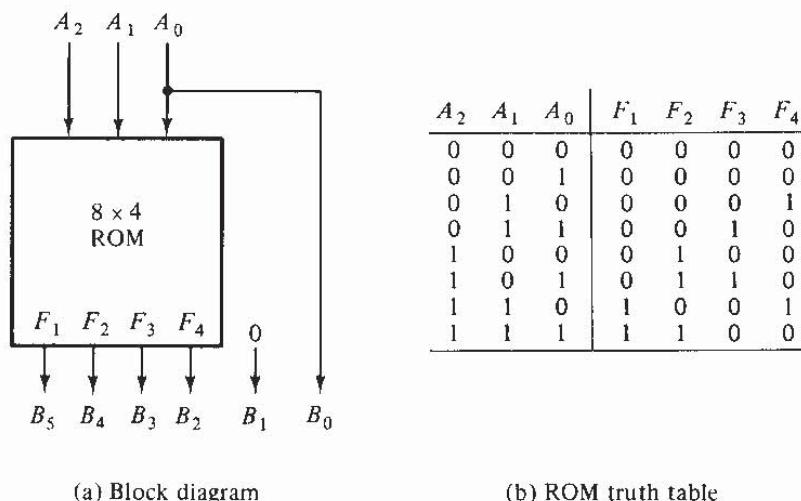
**Example  
5-3**

Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and generates an output binary number equal to the square of the input number.

The first step is to derive the truth table for the combinational circuit. In most cases, this is all that is needed. In some cases, we can fit a smaller truth table for the ROM by using certain properties in the truth table of the combinational circuit. Table 5-5 is the

**TABLE 5-5**  
**Truth Table for Circuit of Example 5-3**

Inputs			Outputs						Decimal
$A_2$	$A_1$	$A_0$	$B_5$	$B_4$	$B_3$	$B_2$	$B_1$	$B_0$	
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49



**FIGURE 5-24**  
ROM implementation of Example 5-3

truth table for the combinational circuit. Three inputs and six outputs are needed to accommodate all possible numbers. We note that output  $B_0$  is always equal to input  $A_0$ ; so there is no need to generate  $B_0$  with a ROM since it is equal to an input variable. Moreover, output  $B_1$  is always 0, so this output is always known. We actually need to generate only four outputs with the ROM; the other two are easily obtained. The minimum-size ROM needed must have three inputs and four outputs. Three inputs specify eight words, so the ROM size must be  $8 \times 4$ . The ROM implementation is shown in Fig. 5-24. The three inputs specify eight words of four bits each. The other two outputs of the combinational circuit are equal to 0 and  $A_0$ . The truth table in Fig. 5-24 specifies all the information needed for programming the ROM, and the block diagram shows the required connections. ■

### Types of ROMs

The required paths in a ROM may be programmed in two different ways. The first is called *mask programming* and is done by the manufacturer during the last fabrication process of the unit. The procedure for fabricating a ROM requires that the customer fill out the truth table the ROM is to satisfy. The truth table may be submitted on a special form provided by the manufacturer. More often, it is submitted in a computer input medium in the format specified on the data sheet of the particular ROM. The manufacturer makes the corresponding mask for the paths to produce the 1's and 0's according to the customer's truth table. This procedure is costly because the vendor charges the customer a special fee for custom masking a ROM. For this reason, mask programming is economical only if large quantities of the same ROM configuration are to be manufactured.

For small quantities, it is more economical to use a second type of ROM called a *programmable read-only memory*, or PROM. When ordered, PROM units contain all 0's (or all 1's) in every bit of the stored words. The fuses in the PROM are blown by application of current pulses through the output terminals. A blown fuse defines one binary state and an unbroken link represents the other state. This allows the user to program the unit in the laboratory to achieve the desired relationship between input addresses and stored words. Special units called *PROM programmers* are available commercially to facilitate this procedure. In any case, all procedures for programming ROMs are *hardware* procedures even though the word *programming* is used.

The hardware procedure for programming ROMs or PROMs is irreversible and, once programmed, the fixed pattern is permanent and cannot be altered. Once a bit pattern has been established, the unit must be discarded if the bit pattern is to be changed. A third type of unit available is called *erasable PROM*, or EPROM. EPROMs can be restructured to the initial value (all 0's or all 1's) even though they have been changed previously. When an EPROM is placed under a special ultraviolet light for a given period of time, the shortwave radiation discharges the internal gates that serve as contacts. After erasure, the ROM returns to its initial state and can be reprogrammed. Certain ROMs can be erased with electrical signals instead of ultraviolet light, and these are called *electrically erasable PROMs*, or EEPROMs.

The function of a ROM can be interpreted in two different ways. The first interpretation is of a unit that implements any combinational circuit. From this point of view, each output terminal is considered separately as the output of a Boolean function expressed in sum of minterms. The second interpretation considers the ROM to be a storage unit having a fixed pattern of bit strings called *words*. From this point of view, the inputs specify an *address* to a specific stored word, which is then applied to the outputs. For example, the ROM of Fig. 5-24 has three address lines, which specify eight stored words as given by the truth table. Each word is four bits long. This is the reason why the unit is given the name *read-only memory*. *Memory* is commonly used to designate a storage unit. *Read* is commonly used to signify that the contents of a word specified by an address in a storage unit is placed at the output terminals. Thus, a ROM is a memory unit with a fixed word pattern that can be read out upon application of a given address. The bit pattern in the ROM is permanent and cannot be changed during normal operation.

ROMs are widely used to implement complex combinational circuits directly from their truth tables. They are useful for converting from one binary code to another (such as ASCII to EBCDIC and vice versa), for arithmetic functions such as multipliers, for display of characters in a cathode-ray tube, and in many other applications requiring a large number of inputs and outputs. They are also employed in the design of control units of digital systems. As such, they are used to store fixed bit patterns that represent the sequence of control variables needed to enable the various operations in the system. A control unit that utilizes a ROM to store binary control information is called a *microprogrammed control unit*.