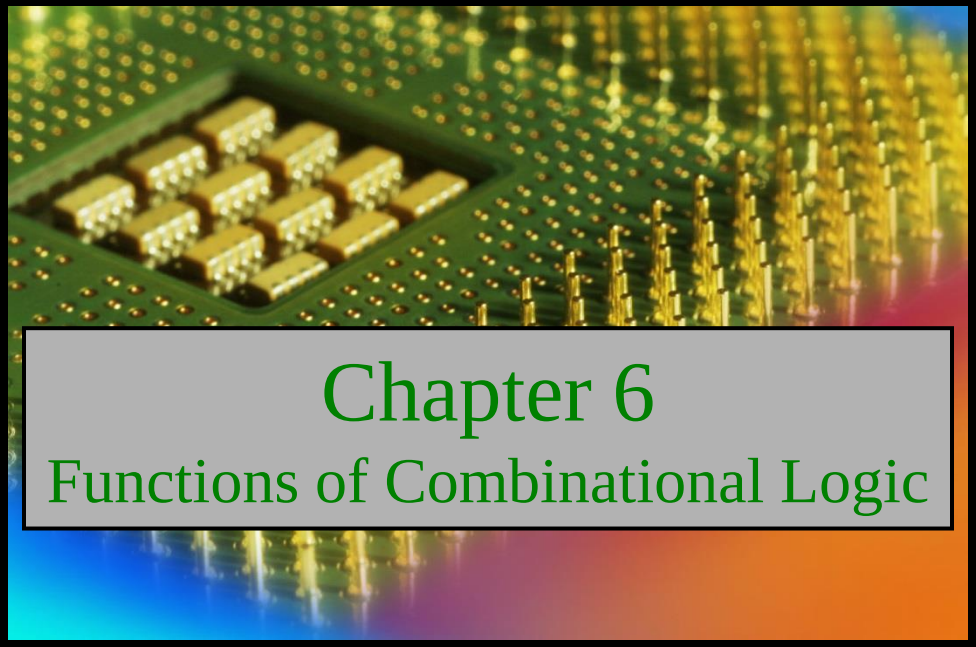
EE-227
Digital Logic Design

Spring-2019



Chapter 6
Functions of Combinational Logic

Course Instructor: Engr. Khalid Iqbal Soomro

Half and Full Adders

Adders are important in computers and also in other types of digital systems in which numerical data are processed.

An understanding of the basic adder operation is fundamental to the study of digital systems.

In this section, the half-adder and the full-adder are introduced.

Half Adder Logic

From the operation of the half-adder as stated in Table 6–1, expressions can be derived for the sum and the output carry as functions of the inputs. Notice that the output carry (C_{out}) is a 1 only when both A and B are 1s; therefore, C_{out} can be expressed as the AND of the input variables.

$$C_{out} = AB \qquad \text{Equation 6–1}$$

Now observe that the sum output ($\sum$) is a 1 only if the input variables, A and B, are not equal. The sum can therefore be expressed as the exclusive-OR of the input variables.

$$\sum = A \oplus B \qquad \text{Equation 6–2}$$

Inputs		Outputs	
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 6–1

Half Adder Logic

From Equations 6–1 and 6–2, the logic implementation required for the half-adder function can be developed.

The output carry is produced with an AND gate with *A* and *B* on the inputs, and the sum output is generated with an exclusive-OR gate, as shown in Figure 6–2.

Remember that the exclusive-OR can be implemented with AND gates, an OR gate, and inverters.

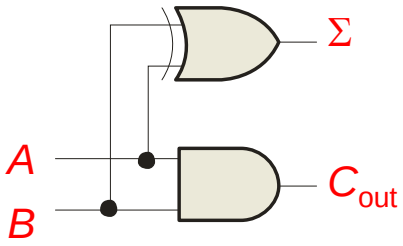


Figure 6–2

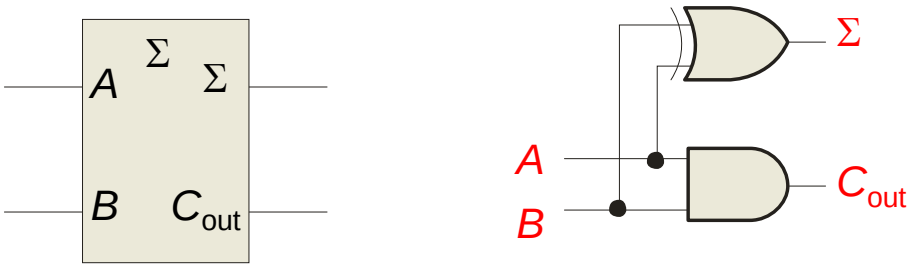
Half-Adder

Basic rules of binary addition are performed by a **half adder**, which has two binary inputs (*A* and *B*) and two binary outputs (Carry out and Sum).

The inputs and outputs can be summarized on a truth table.

Inputs		Outputs	
A	*B*	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The logic symbol and equivalent circuit are:



$$\Sigma = A \oplus B$$

$$C_{out} = AB$$

Figure 6–2

Full-Adder Logic

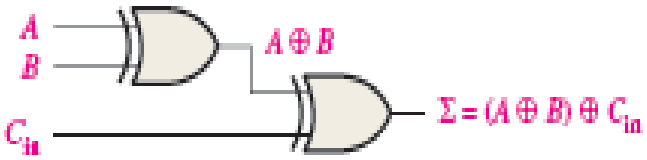
The full-adder must add the two input bits and the input carry. From the half-adder you know that the sum of the input bits *A* and *B* is the exclusive-OR of those two variables, *AB*.

For the input carry (C_{in}) to be added to the input bits, it must be exclusive-Ored with *AB*, yielding the equation for the sum output of the full-adder.

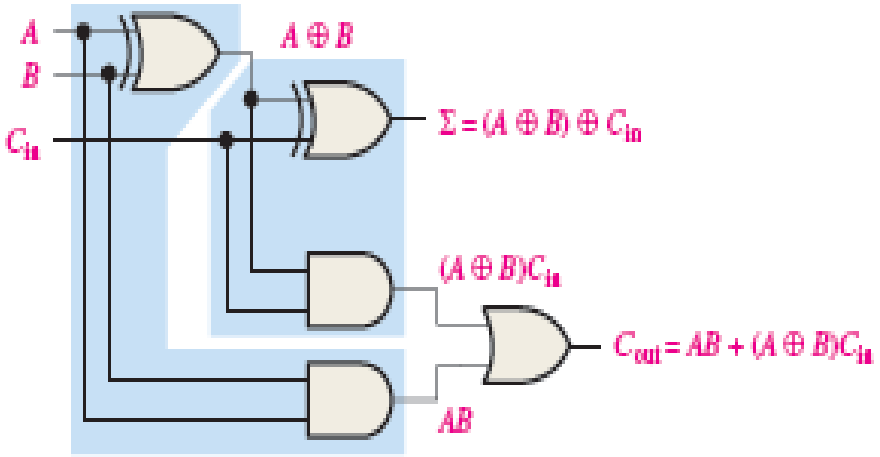
$$\sum = (A \oplus B) \oplus C_{in} \qquad \text{Equation 6–3}$$

This means that to implement the full-adder sum function, two 2-input exclusive-OR gates can be used. The first must generate the term $A \oplus B$, and the second has as its inputs the output of the first XOR gate and the input carry, as illustrated in Figure 6–4(a).

Full-Adder Logic



(a) Logic required to form the sum of three bits



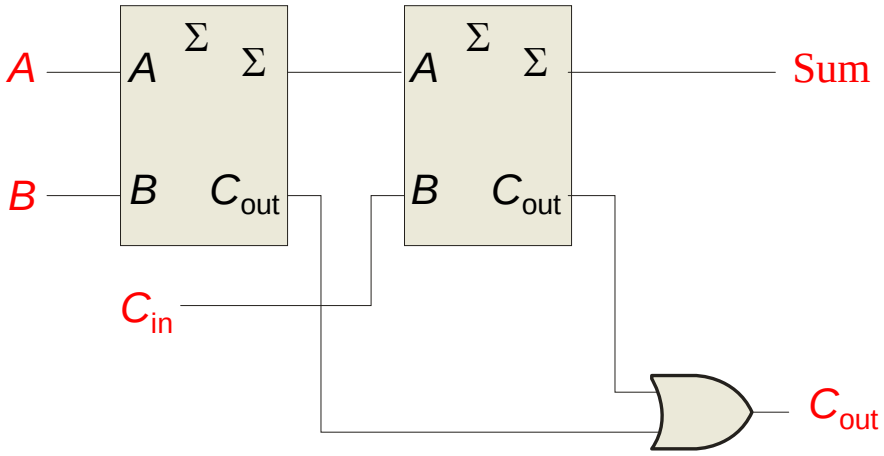
(b) Complete logic circuit for a full-adder (each half-adder is enclosed by a shaded area)

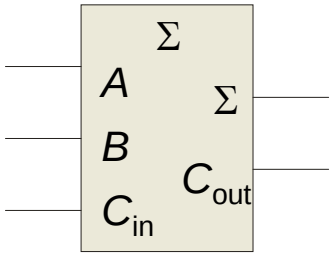
Full-Adder

By contrast, a **full adder** has three binary inputs (*A*, *B*, and Carry in) and two binary outputs (Carry out and Sum). The truth table summarizes the operation.

A full-adder can be constructed from two half adders as shown:

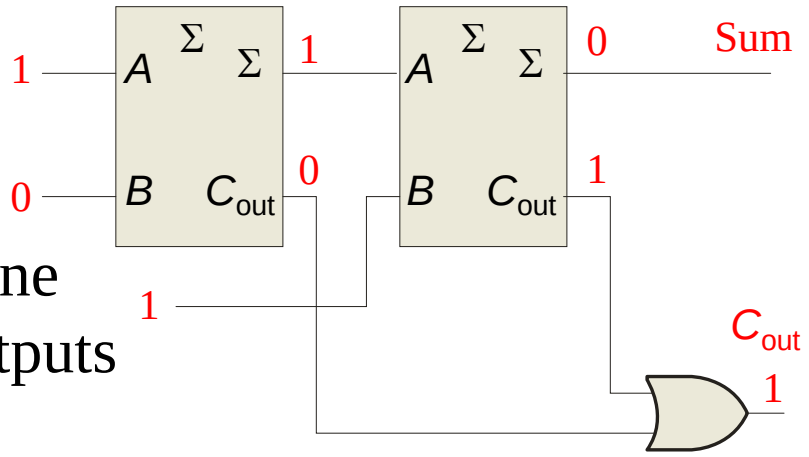
Inputs			Outputs	
A	*B*	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1





Symbol

Full-Adder



Example

For the given inputs, determine the intermediate and final outputs of the full adder.

Solution

The first half-adder has inputs of 1 and 0; therefore the Sum =1 and the Carry out = 0.

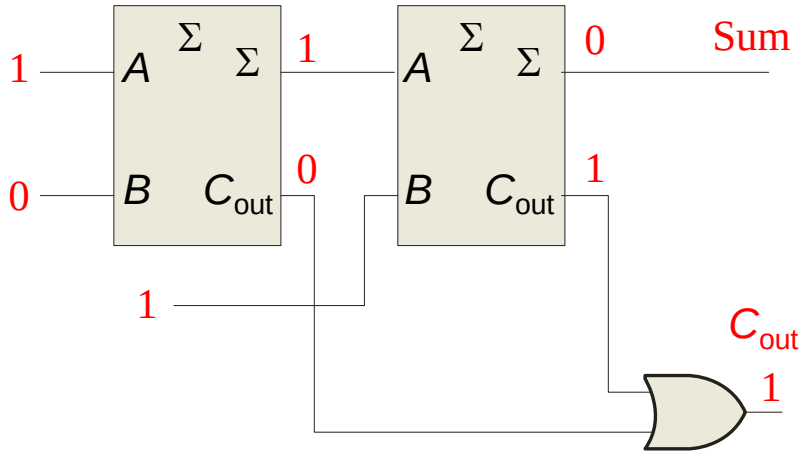
The second half-adder has inputs of 1 and 1; therefore the Sum = 0 and the Carry out = 1.

The OR gate has inputs of 1 and 0, therefore the final carry out = 1.

Full-Adder

Notice that the result from the previous example can be read directly on the truth table for a full adder.

Inputs			Outputs	
A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



EXAMPLE 6–1

For each of the three full-adders in Figure 6–6, determine the outputs for the inputs shown.

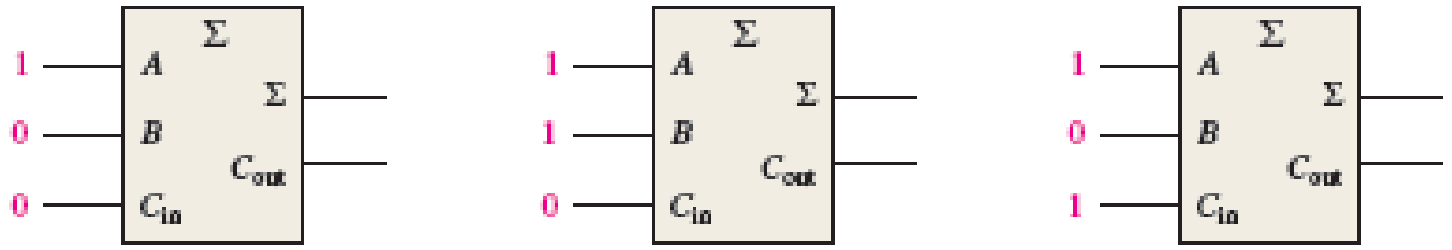


FIGURE 6–6

Solution

(a) The input bits are $A = 1$, $B = 0$, and $C_{in} = 0$.

$$1 + 0 + 0 = 1 \text{ with no carry}$$

Therefore, $\Sigma = \mathbf{1}$ and $C_{out} = \mathbf{0}$.

(b) The input bits are $A = 1$, $B = 1$, and $C_{in} = 0$.

$$1 + 1 + 0 = 0 \text{ with a carry of 1}$$

Therefore, $\Sigma = \mathbf{0}$ and $C_{out} = \mathbf{1}$.

(c) The input bits are $A = 1$, $B = 0$, and $C_{in} = 1$.

$$1 + 0 + 1 = 0 \text{ with a carry of 1}$$

Therefore, $\Sigma = \mathbf{0}$ and $C_{out} = \mathbf{1}$.

Half Subtractor

Subtracting a single-bit binary value Y from X (i.e. X -Y) produces a difference bit D and a borrow out bit B_{out}

This operation is called half subtraction and the circuit to realize this is called a half subtractor.

Half Subtractor Truth Table

Inputs		Outputs	
X	Y	D	B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

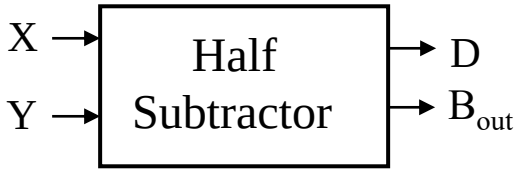
$D(X,Y) = \Sigma\,(1,2)$

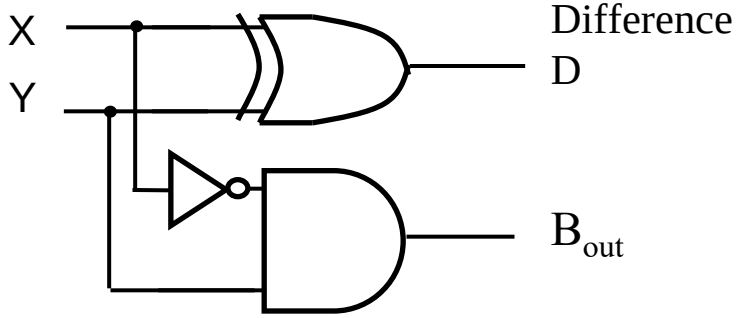
$D = X'Y + XY'$

$D = X \oplus Y$

$B_{out}(X, Y,) = \Sigma\,(1)$

$B_{out} = X'Y$





Full Subtractor

Subtracting two single-bit binary values, Y, B_{in} from a single-bit value X produces a difference bit D and a borrow out B_{out} bit. This is called full subtraction.

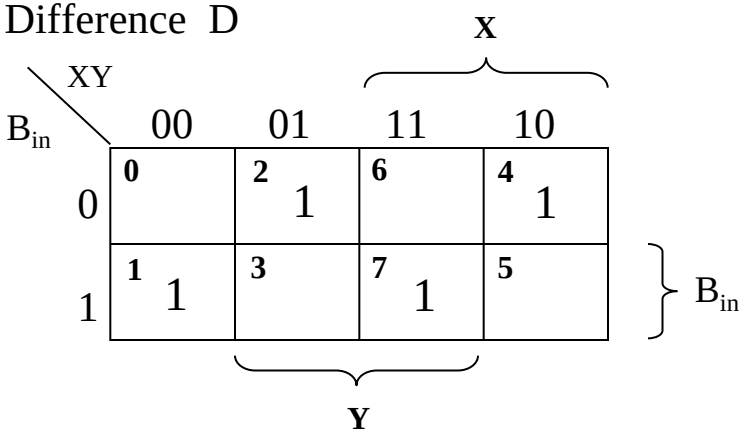
Full Subtractor Truth Table

Inputs			Outputs	
X	Y	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$D(X,Y,B_{in}) = D(1,2,4,7)$

$B_{out}(X,Y,B_{in}) = \Sigma(1,2,3,7)$

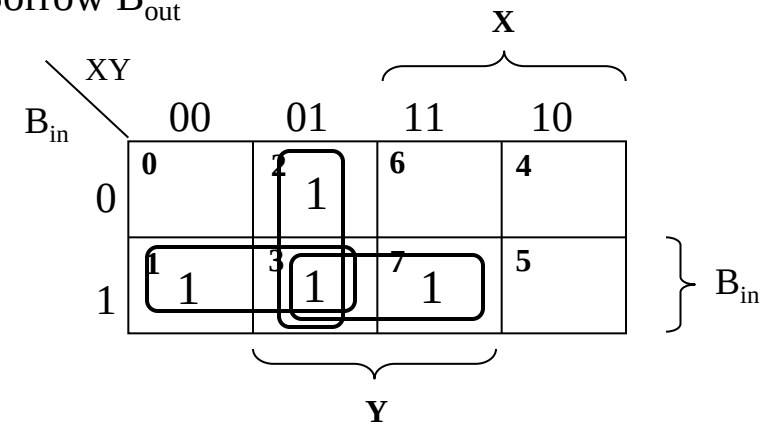
Difference D



$D = X'Y' B_{in} + X'Y (B_{in})' + XY'(B_{in})' + XYB_{in}$

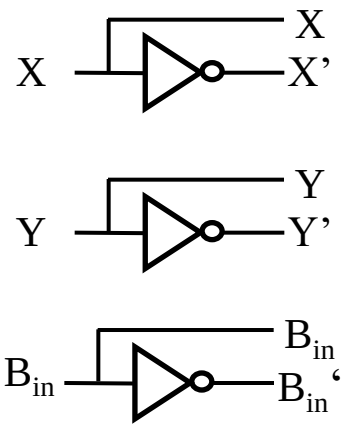
$D = X \oplus Y \oplus B_{in}$

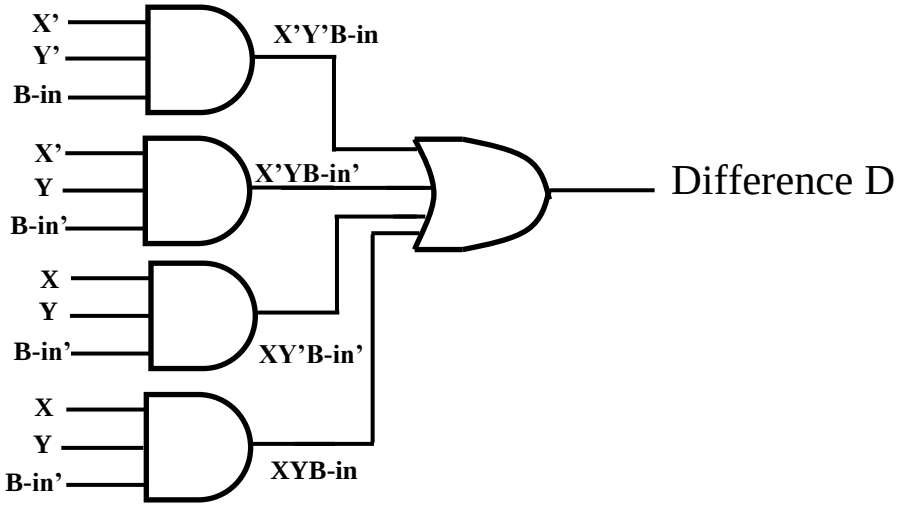
Borrow B_{out}

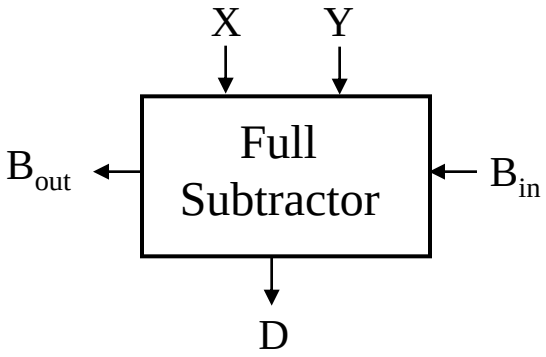


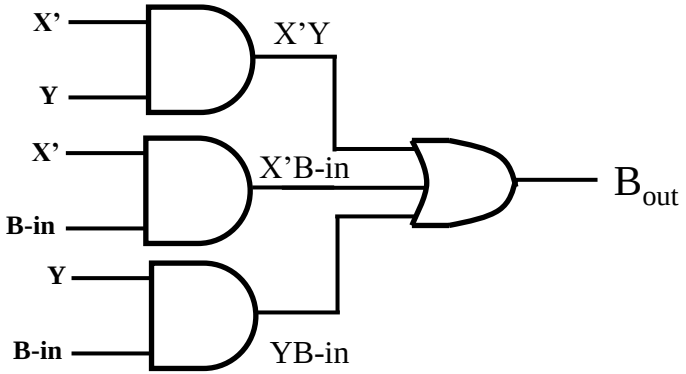
$B_{out} = X' B_{in} + X'Y + Y B_{in}$

Full Subtractor Circuit Using AND-OR

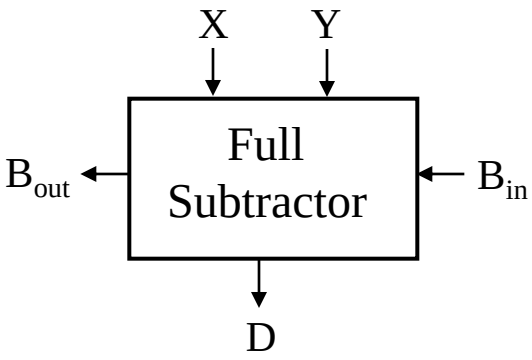


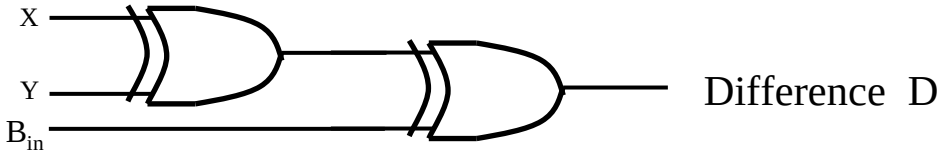


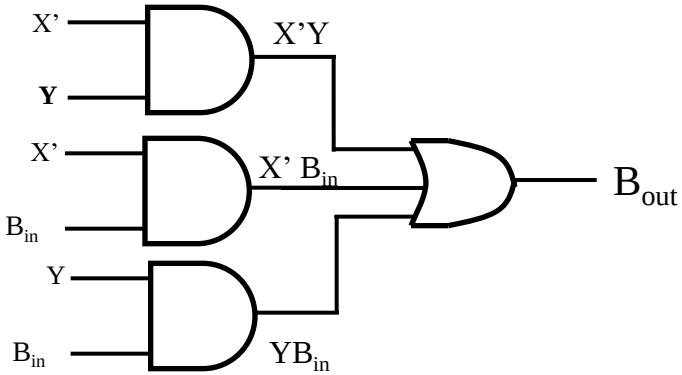




Full Subtractor Circuit Using XOR

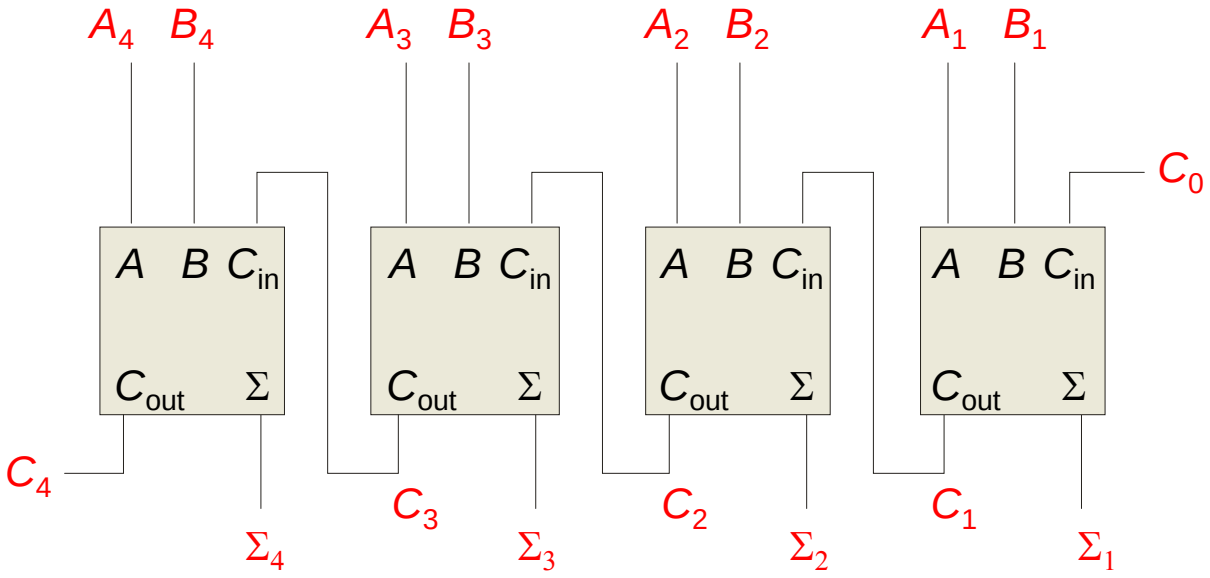






Parallel Adders

Full adders are combined into parallel adders that can add binary numbers with multiple bits. A 4-bit adder is shown.



The output carry (C_4) is not ready until it propagates through all of the full adders. This is called *ripple carry*, delaying the addition process.

EXAMPLE 6–2

Determine the sum generated by the 3-bit parallel adder in Figure 6–8 and show the intermediate carries when the binary numbers 101 and 011 are being added.

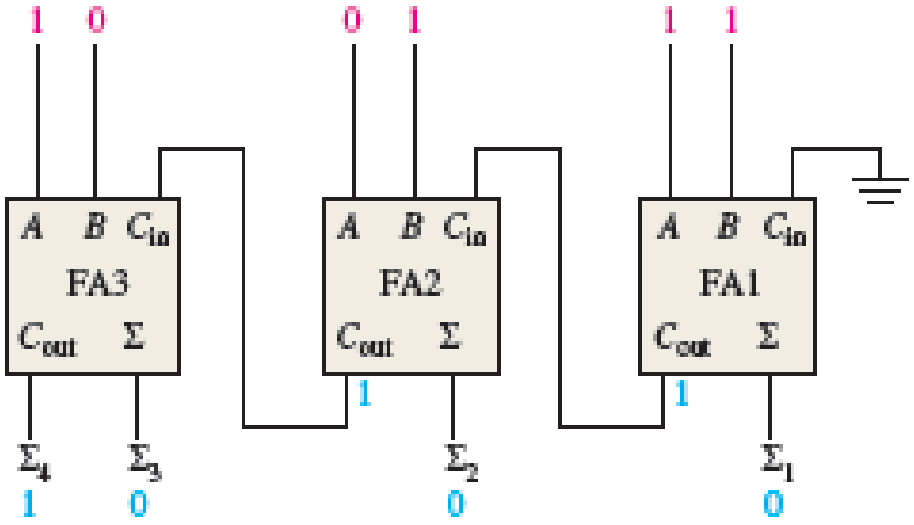


FIGURE 6–8

Solution

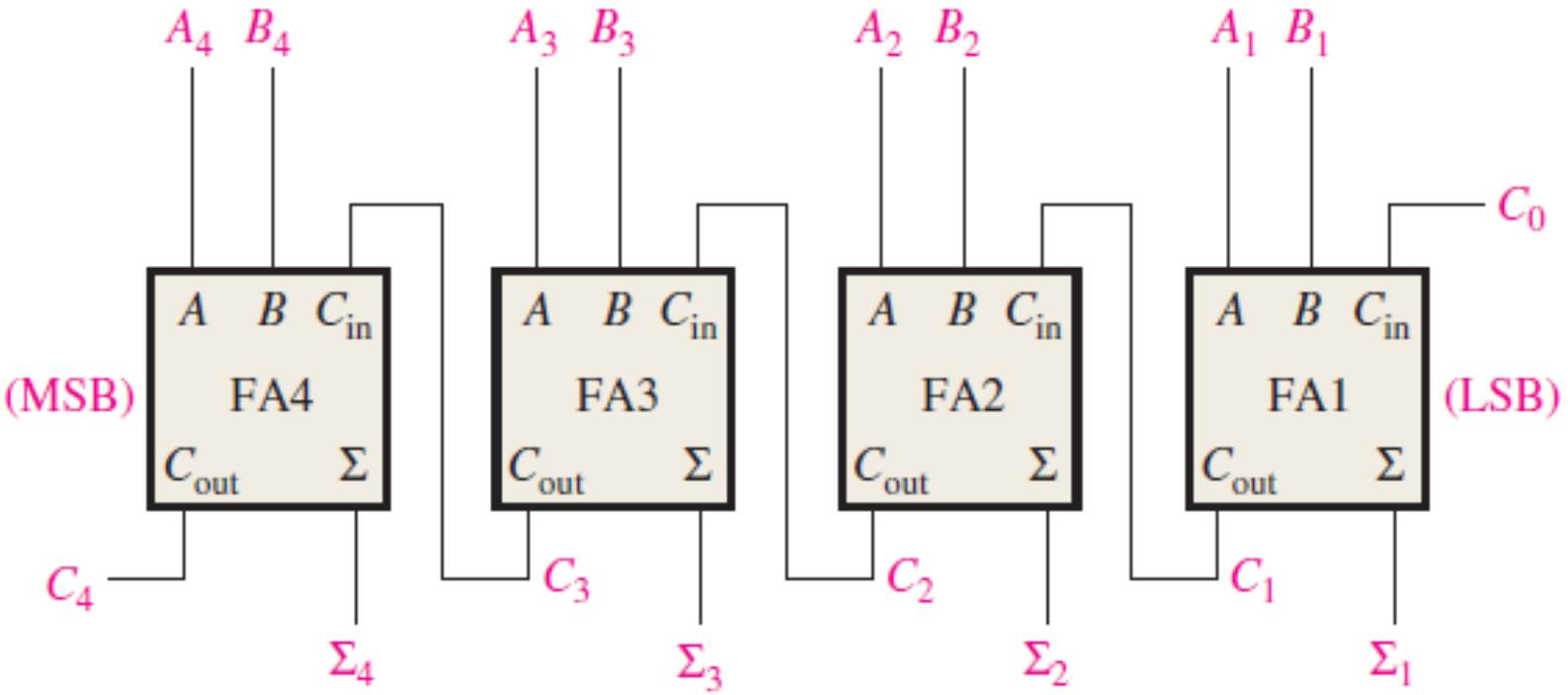
The LSBs of the two numbers are added in the right-most full-adder. The sum bits and the intermediate carries are indicated in blue in Figure 6–8.

Four-Bit Parallel Adders

A group of four bits is called a *nibble*. A basic 4-bit parallel adder is implemented with four full-adder stages as shown in Figure 6–9.

Again, the LSBs (A_1 and B_1) in each number being added go into the right-most full-adder; the higher-order bits are applied as shown to the successively higher-order adders, with the MSBs (A_4 and B_4) in each number being applied to the left-most full-adder.

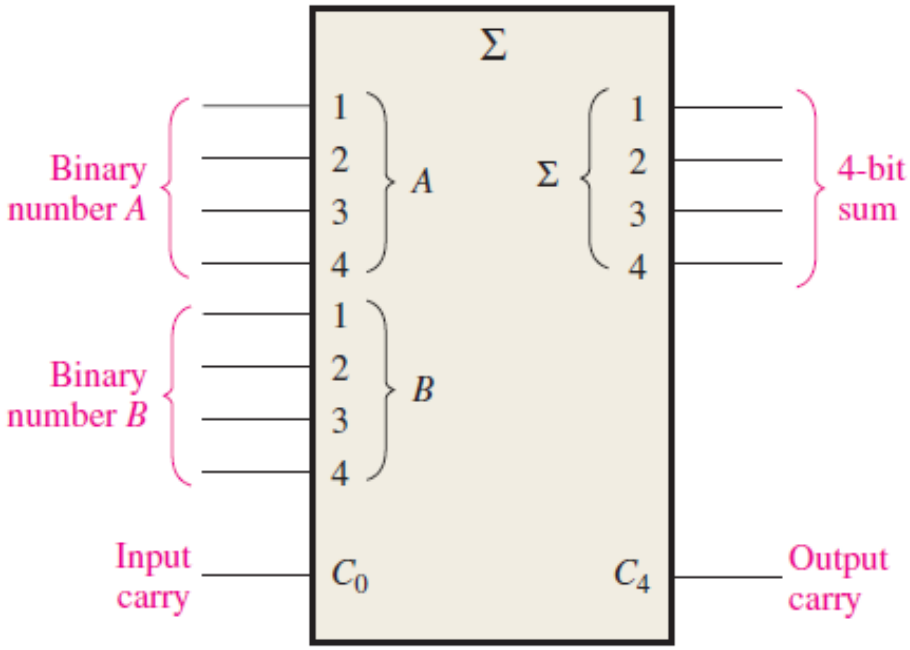
The carry output of each adder is connected to the carry input of the next higher-order adder as indicated. These are called *internal carries*.



(a) Block diagram

FIGURE 6–9 A 4-bit parallel adder.

In keeping with most manufacturers' data sheets, the input labeled C_0 is the input carry to the least significant bit adder; C_4, in the case of four bits, is the output carry of the most significant bit adder; and $\sum_1$ (LSB) through $\sum_4$ (MSB) are the sum outputs. The logic symbol is shown in Figure 6–9(b).



(b) Logic symbol

Truth Table for a 4-Bit Parallel Adder

Table 6–3 is the truth table for a 4-bit adder. On some data sheets, truth tables may be called *function tables* or *functional truth tables*.

The subscript *n* represents the adder bits and can be 1, 2, 3, or 4 for the 4-bit adder. C_{n-1} is the carry from the previous adder.

TABLE 6–3

Truth table for each stage of a 4-bit parallel adder.

C_{n-1}	A_n	B_n	Σ_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Carries C_1, C_2, and C_3 are generated internally. C_0 is an external carry input and C_4 is an output. Example 6–3 illustrates how to use Table 6–3.

EXAMPLE 6–3

Use the 4-bit parallel adder truth table (Table 6–3) to find the sum and output carry for the addition of the following two 4-bit numbers if the input carry (C_{n-1}) is 0:

$$A_4A_3A_2A_1 = 1100 \quad \text{and} \quad B_4B_3B_2B_1 = 1100$$

Solution

For $n = 1$: $A_1 = 0$, $B_1 = 0$, and $C_{n-1} = 0$. From the 1st row of the table,

$$\Sigma_1 = \mathbf{0} \quad \text{and} \quad C_1 = 0$$

For $n = 2$: $A_2 = 0$, $B_2 = 0$, and $C_{n-1} = 0$. From the 1st row of the table,

$$\Sigma_2 = \mathbf{0} \quad \text{and} \quad C_2 = 0$$

For $n = 3$: $A_3 = 1$, $B_3 = 1$, and $C_{n-1} = 0$. From the 4th row of the table,

$$\Sigma_3 = \mathbf{0} \quad \text{and} \quad C_3 = 1$$

For $n = 4$: $A_4 = 1$, $B_4 = 1$, and $C_{n-1} = 1$. From the last row of the table,

$$\Sigma_4 = \mathbf{1} \quad \text{and} \quad C_4 = \mathbf{1}$$

C_4 becomes the output carry; the sum of 1100 and 1100 is 11000.

TABLE 6–3

Truth table for each stage of a 4-bit parallel adder.

C_{n-1}	A_n	B_n	Σ_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The Ripple Carry Adder

A ripple carry adder is one in which the carry output of each full-adder is connected to the carry input of the next higher-order stage (a stage is one full-adder).

The sum and the output carry of any stage cannot be produced until the input carry occurs; this causes a time delay in the addition process, as illustrated in Figure 6–14.

The carry propagation delay for each full-adder is the time from the application of the input carry until the output carry occurs, assuming that the A and B inputs are already present.

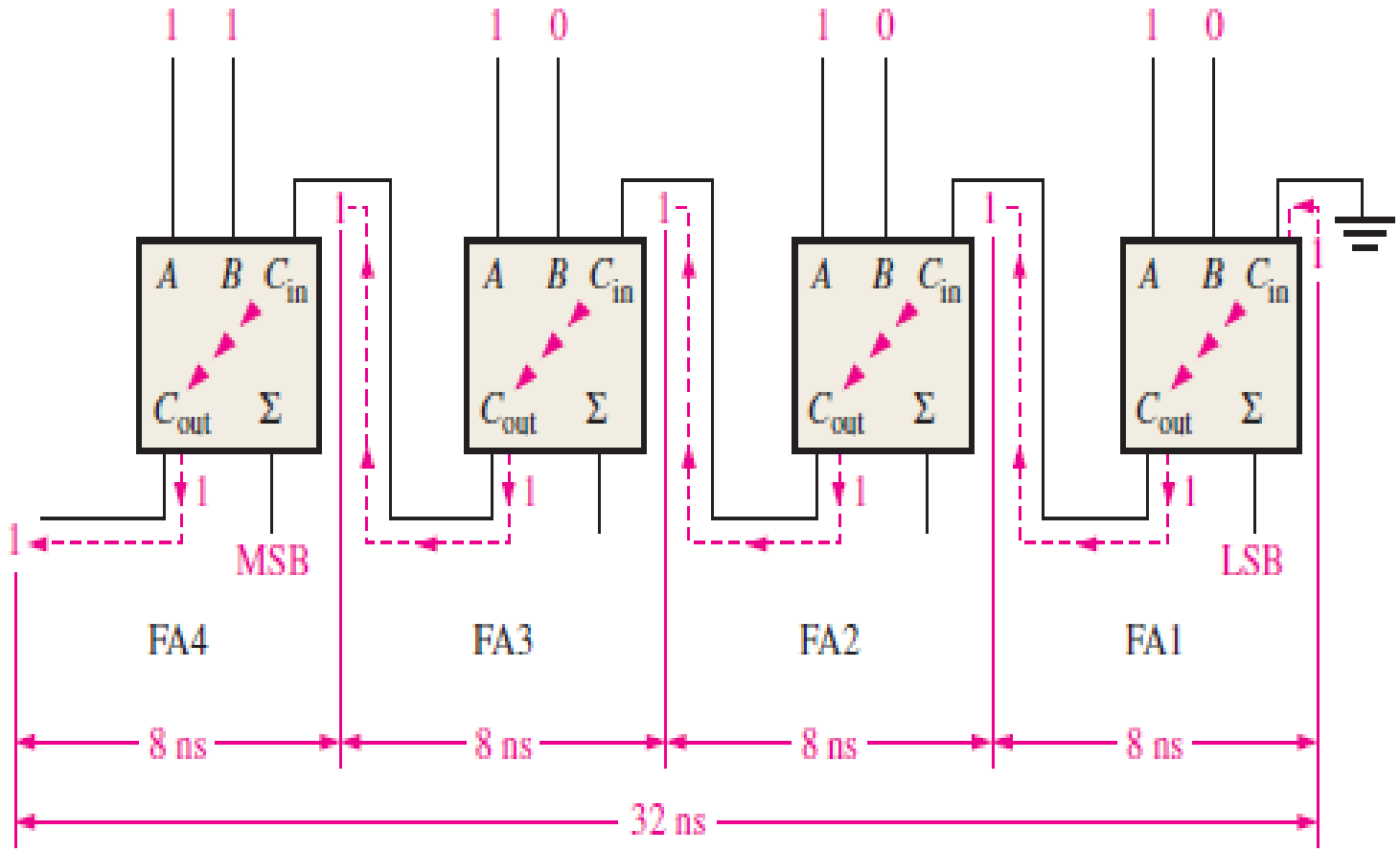


FIGURE 6–14 A 4-bit parallel ripple carry adder showing "worst-case" carry propagation delays.

The Look-Ahead Carry Adder

The speed with which an addition can be performed is limited by the time required for the carries to propagate, or ripple, through all the stages of a parallel adder.

One method of speeding up the addition process by eliminating this ripple carry delay is called *look-ahead carry addition*.

The look-ahead carry adder anticipates the output carry of each stage, and based on the inputs, produces the output carry by either *carry generation* or *carry propagation*.

Carry generation occurs when an output carry is produced (generated) internally by the full-adder. A carry is generated only when both input bits are 1s. The generated carry, Cg, is expressed as the AND function of the two input bits, A and B.

$$Cg = AB \qquad \text{Equation 6–5}$$

Carry propagation occurs when the input carry is rippled to become the output carry. An input carry may be propagated by the full-adder when either or both of the input bits are 1s. The propagated carry, Cp, is expressed as the OR function of the input bits.

$$Cp = A + B \qquad \text{Equation 6–6}$$

The conditions for carry generation and carry propagation are illustrated in Figure 6–15.The three arrowheads symbolize ripple (propagation).

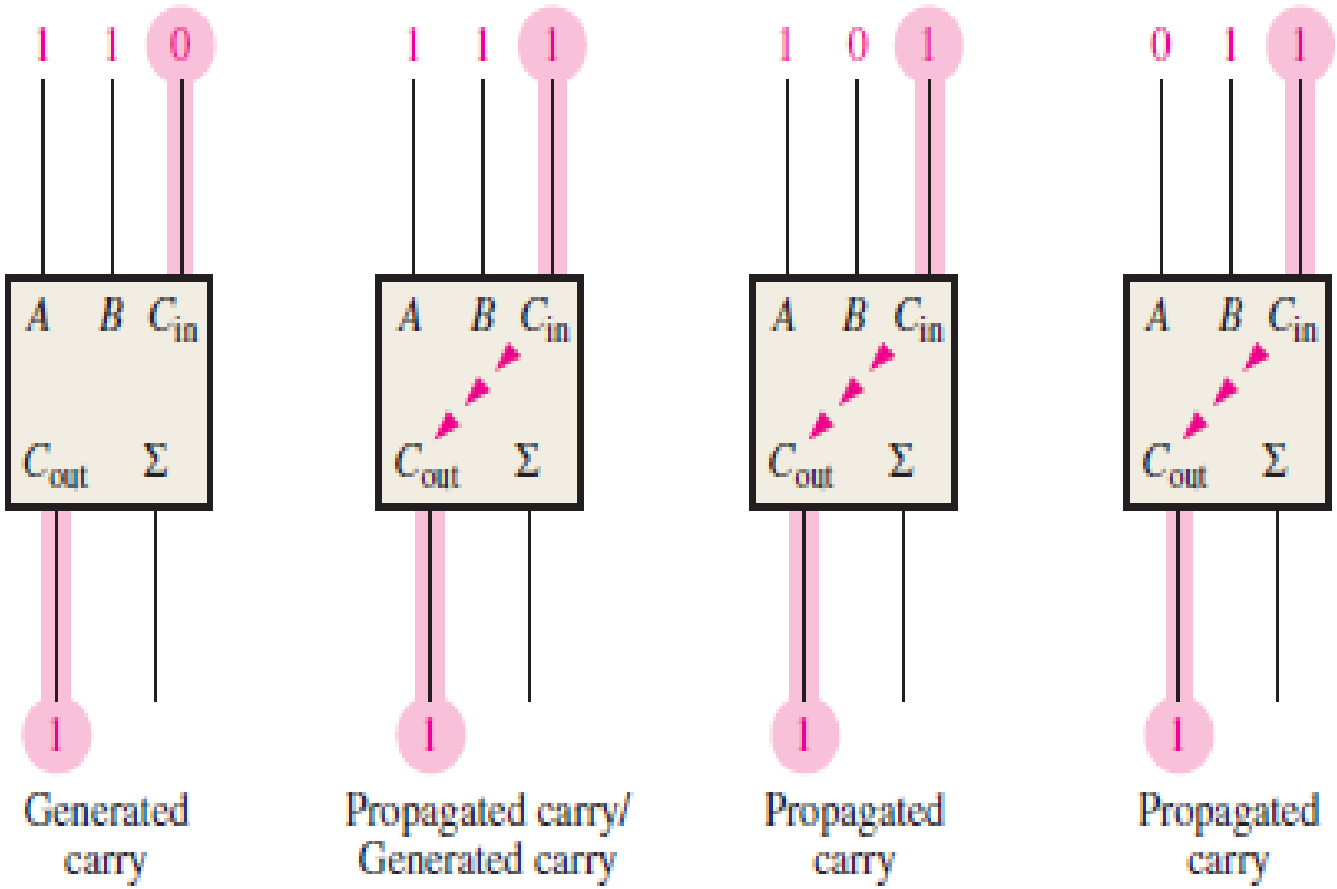


FIGURE 6–15 Illustration of conditions for carry generation and carry propagation.

The output carry of a full-adder can be expressed in terms of both the generated carry (*Cg*) and the propagated carry (*Cp*).

The output carry (C_{out}) is a 1 if the generated carry is a 1 OR if the propagated carry is a 1 AND the input carry (C_{in}) is a 1.

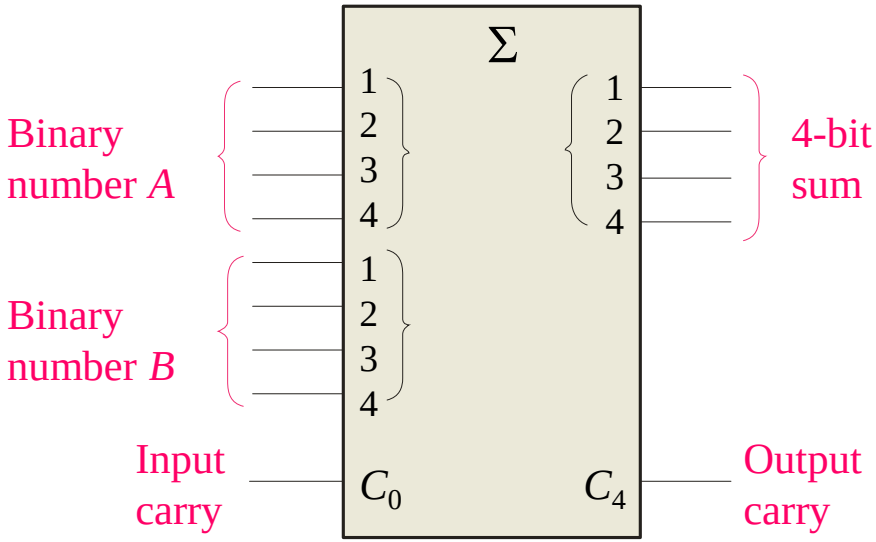
In other words, we get an output carry of 1 if it is generated by the full-adder (A = 1 AND B = 1) or if the adder propagates the input carry (A = 1 OR B = 1) AND C_{in} = 1.

This relationship is expressed as

$$\boldsymbol{C_{out} = Cg + CpC_{in}} \qquad \text{Equation 6–7}$$

Parallel Adders

The logic symbol for a 4-bit parallel adder is shown. This 4-bit adder includes a carry in (labeled (C_0) and a Carry out (labeled C_4).



The 74LS283 is an example. It features *look-ahead carry*, which adds logic to minimize the output carry delay. For the 74LS283, the maximum delay to the output carry is 17 ns.

Comparators

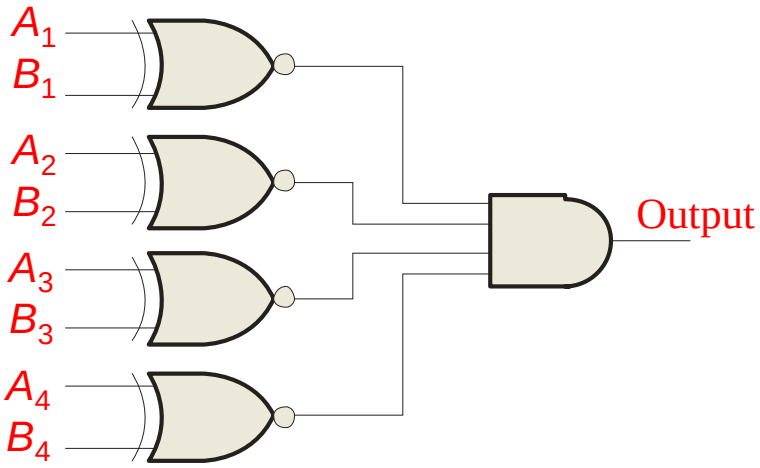
The function of a comparator is to compare the magnitudes of two binary numbers to determine the relationship between them. In the simplest form, a comparator can test for equality using XNOR gates.

Example

How could you test two 4-bit numbers for equality?

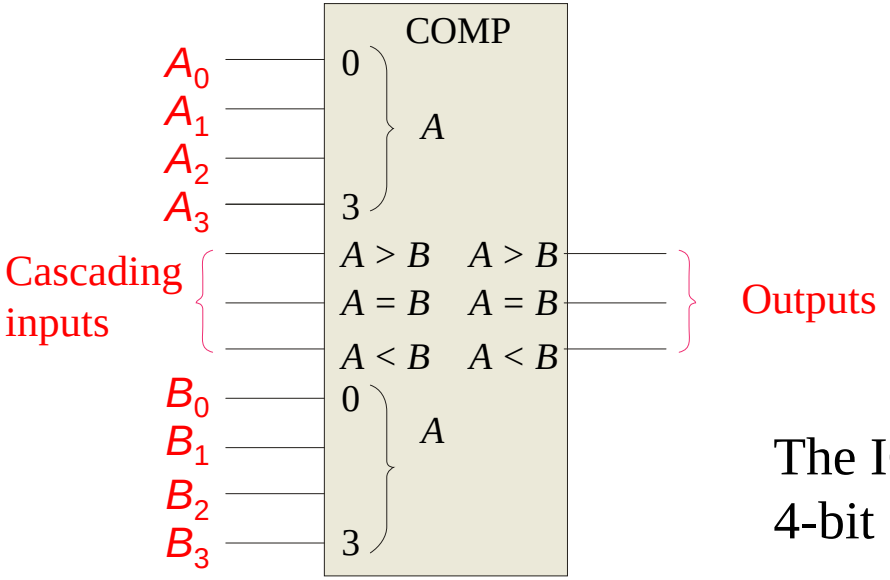
Solution

AND the outputs of four XNOR gates.



Comparators

IC comparators provide outputs to indicate which of the numbers is larger or if they are equal. The bits are numbered starting at 0, rather than 1 as in the case of adders. Cascading inputs are provided to expand the comparator to larger numbers.



The IC shown is the 4-bit 74LS85.

Comparators [Equality]

As you learned in Chapter 3, the exclusive-NOR gate can be used as a basic comparator because its output is a 0 if the two input bits are not equal and a 1 if the input bits are equal.

Fig 6–18 shows the exclusive-NOR gate as a 2-bit comparator.

FIGURE 6–18 Basic comparator operation.

In order to compare binary numbers containing two bits each, an additional exclusive-NOR gate is necessary. The two least significant bits (LSBs) of the two numbers are compared by gate G_1, and the two most significant bits (MSBs) are compared by gate G_2, as shown in Figure 6–19.

If the two numbers are equal, their corresponding bits are the same, and the output of each exclusive-NOR gate is a 1. If the corresponding sets of bits are not equal, a 0 occurs on that exclusive-NOR gate output.

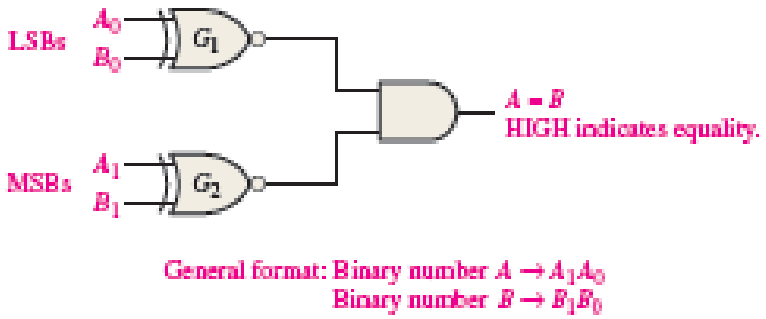


FIGURE 6–19 Logic diagram for equality comparison of two 2-bit numbers.

EXAMPLE 6–5

Apply each of the following sets of binary numbers to the comparator inputs in Figure 6–20, and determine the output by following the logic levels through the circuit.

(a) 10 and 10 (b) 11 and 10

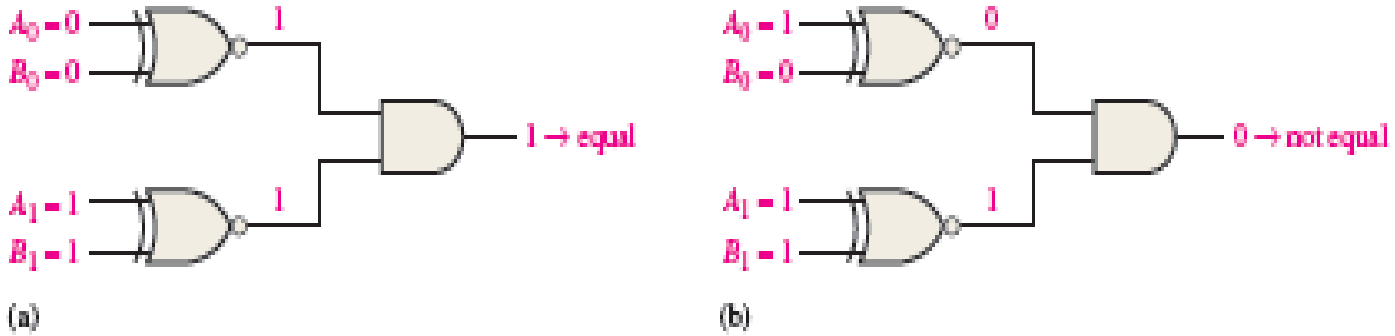


FIGURE 6–20

Solution

(a) The output is 1 for inputs 10 and 10, as shown in Figure 6–20(a).

(b) The output is 0 for inputs 11 and 10, as shown in Figure 6–20(b).

Comparators [Inequality]

In addition to the equality output, fixed-function comparators can provide additional outputs that indicate which of the two binary numbers being compared is larger. That is, there is an output that indicates when number A is greater than number B (A > B) and an output that indicates when number A is less than number B (A < B), as shown in the logic symbol for a 4-bit comparator in Figure 6–21.

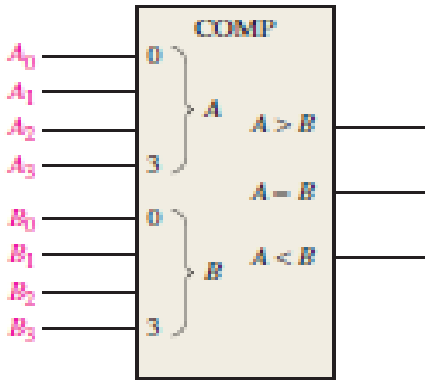


FIGURE 6–21 Logic symbol for a 4-bit comparator with inequality indication.

Comparators [Inequality]

To determine an inequality of binary numbers A and B, you first examine the highest order bit in each number. The following conditions are possible:

1. If $A_3 = 1$ and $B_3 = 0$, number A is greater than number B.
2. If $A_3 = 0$ and $B_3 = 1$, number A is less than number B.
3. If $A_3 = B_3$, then you must examine the next lower bit position for an inequality.

These three operations are valid for each bit position in the numbers. The general procedure used in a comparator is to check for an inequality in a bit position, starting with the highest-order bits (MSBs). When such an inequality is found, the relationship of the two numbers is established, & any other inequalities in lower-order bit positions is ignored because it is possible for an opposite indication to occur.

EXAMPLE 6–6

Determine the $A = B$, $A > B$, and $A < B$ outputs for the input numbers shown on the comparator in Figure 6–22.

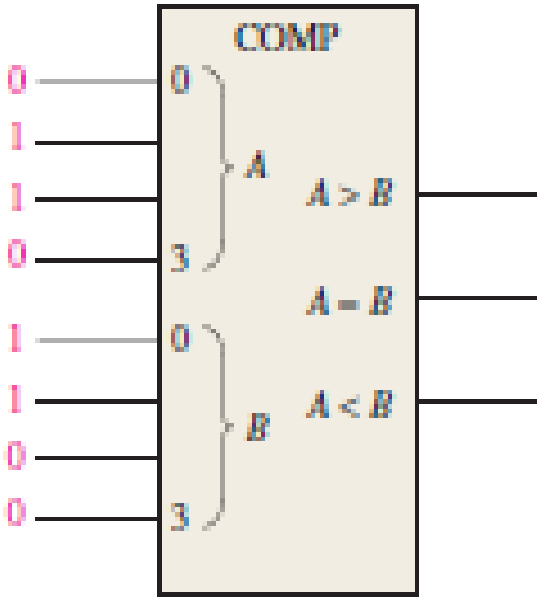


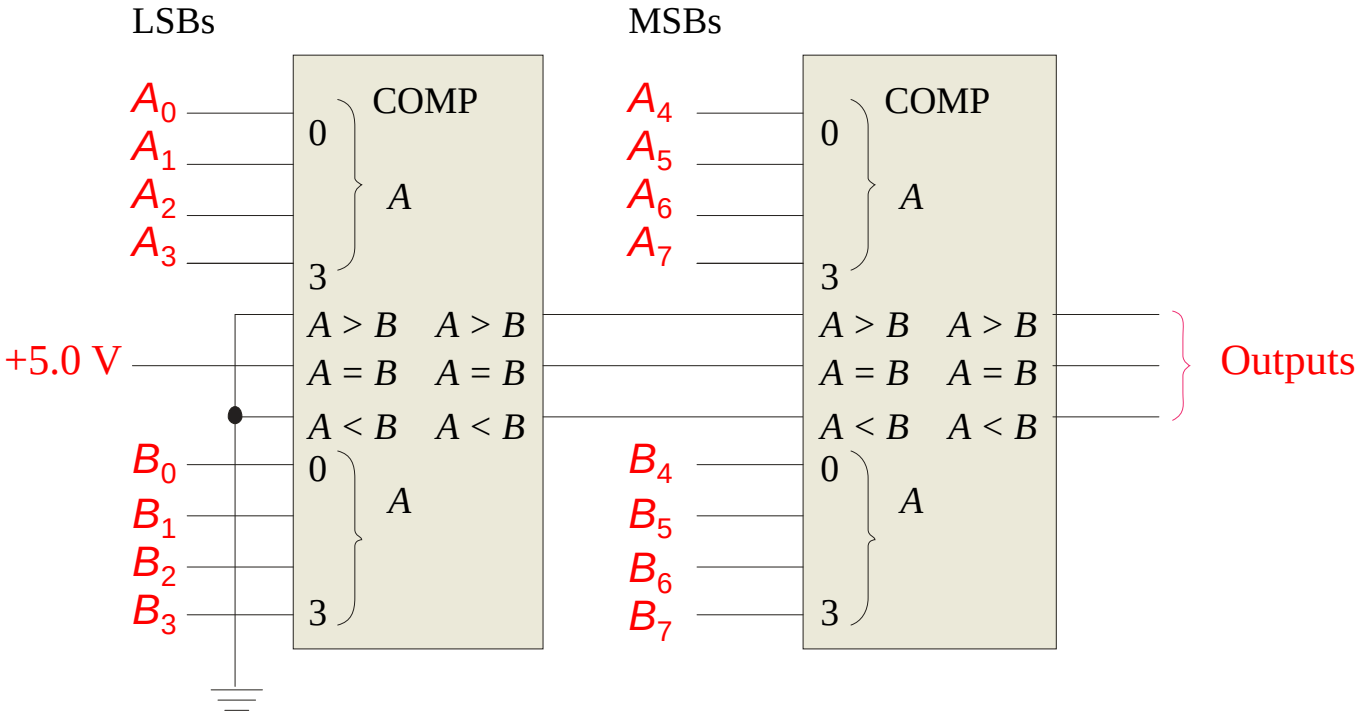
FIGURE 6–22

Solution

The number on the A inputs is 0110 and the number on the B inputs is 0011. The $A > B$ output is **HIGH** and the other outputs are **LOW**.

Comparators

IC comparators can be expanded using the cascading inputs as shown. The lowest order comparator has a HIGH on the $A = B$ input.



InfoNote

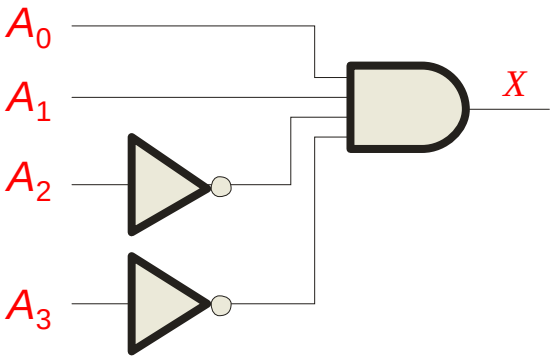
An instruction tells the processor what operation to perform. Instructions are in machine code (1s and 0s) and, in order for the processor to carry out an instruction, the instruction must be decoded.

Instruction decoding is one of the steps in instruction *pipelining*, which are as follows: Instruction is read from the memory (*instruction fetch*), instruction is decoded, operand(s) is (are) read from memory (*operand fetch*), instruction is executed, and result is written back to memory.

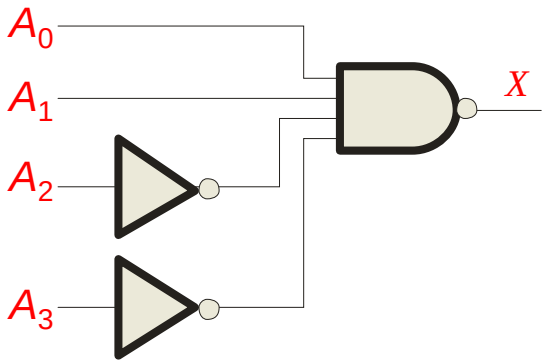
Basically, pipelining allows the next instruction to begin processing before the current one is completed.

Decoders

A **decoder** is a logic circuit that detects the presence of a specific combination of bits at its input. Two simple decoders that detect the presence of the binary code 0011 are shown. The first has an active HIGH output; the second has an active LOW output.



Active HIGH decoder for 0011

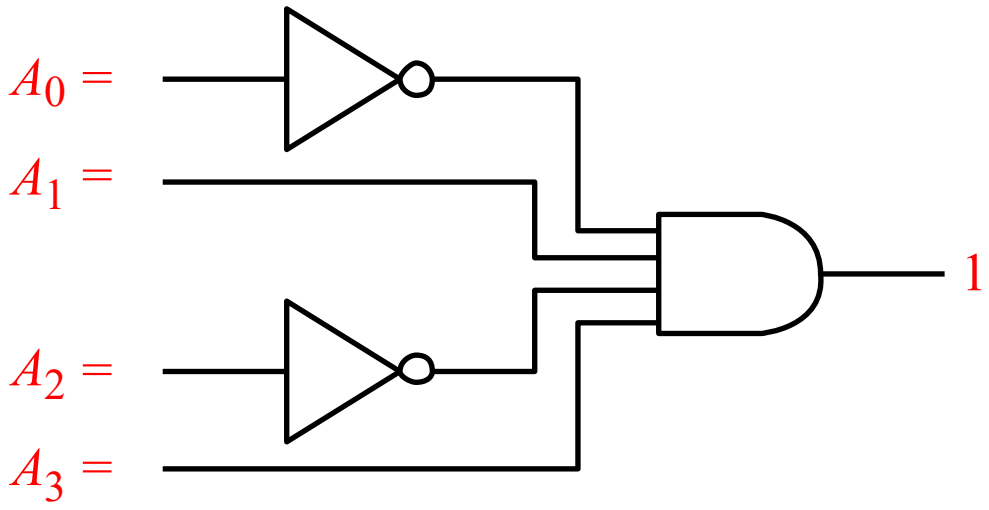


Active LOW decoder for 0011

Decoders

Question

Assume the output of the decoder shown is a logic 1. What are the inputs to the decoder?



The 4-Bit Decoder

In order to decode all possible combinations of four bits, sixteen decoding gates are required ($2^4 = 16$).

This type of decoder is commonly called either a *4-line-to-16-line* decoder because there are four inputs and sixteen outputs or a *1-of-16* decoder because for any given code on the inputs, one of the sixteen outputs is activated.

A list of the sixteen binary codes and their corresponding decoding functions is given in Table 6–4.

TABLE 6–4

Decoding functions and truth table for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs.

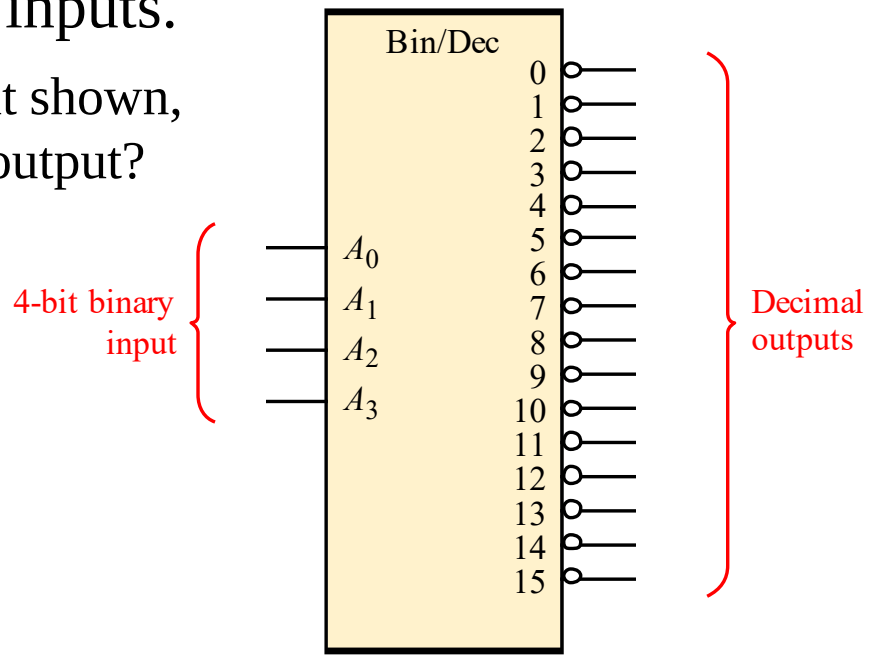
Decimal Digit	Binary Inputs A_3	A_2	A_1	A_0	Decoding Function	Outputs 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$A_3\overline{A}_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$A_3\overline{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	$A_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	$A_3A_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$A_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Decoders

IC decoders have multiple outputs to decode any combination of inputs. For example the binary-to-decimal decoder shown here has 16 outputs – one for each combination of binary inputs.

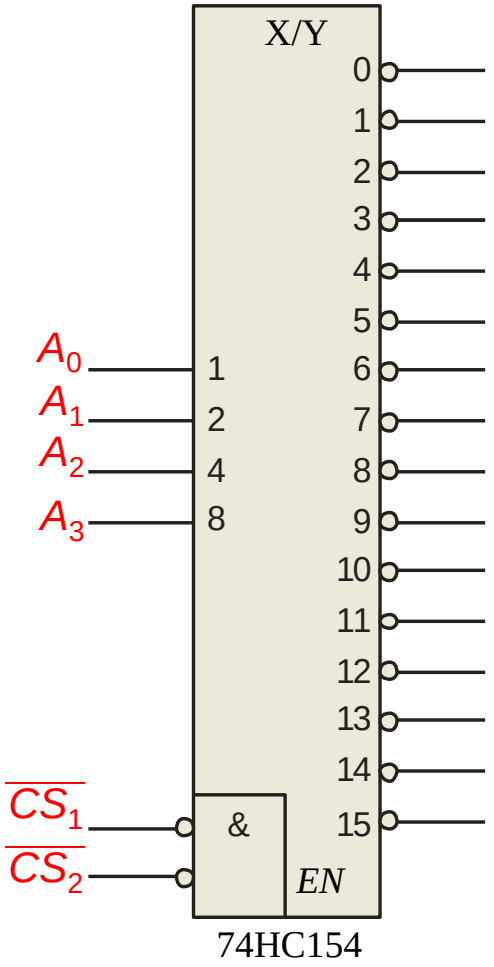
Question

For the input shown, what is the output?



Decoders

A specific integrated circuit decoder is the 74HC154 (shown as a 4-to-16 decoder). It includes two active LOW chip select lines CS_1 & CS_2which must be at the active level to enable the outputs. These lines can be used to expand the decoder to larger than 4-bit inputs.



EXAMPLE 6–9

A certain application requires that a 5-bit number be decoded. Use 74HC154 decoders to implement the logic. The binary number is represented by the format $A_4A_3A_2A_1A_0$.

Solution

Since the 74HC154 can handle only four bits, two decoders must be used to form a 5-bit expansion. The fifth bit, A_4, is connected to the chip select inputs, $\overline{CS}_1$ and $\overline{CS}_2$, of one decoder, and $\overline{A}_4$ is connected to the $\overline{CS}_1$ and $\overline{CS}_2$ inputs of the other decoder, as shown in Figure 6–30. When the decimal number is 15 or less, $A_4 = 0$, the low-order decoder is enabled, and the high-order decoder is disabled. When the decimal number is greater than 15, $A_4 = 1$ so $\overline{A}_4 = 0$, the high-order decoder is enabled, and the low-order decoder is disabled.

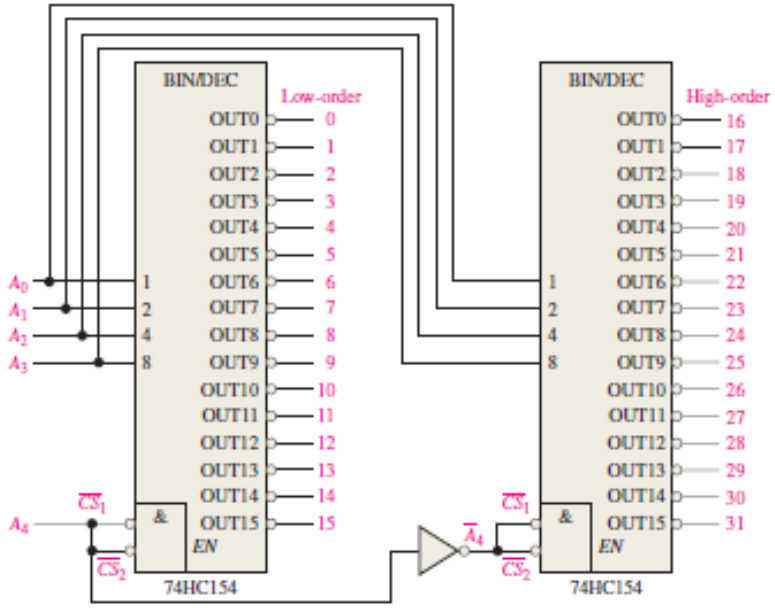
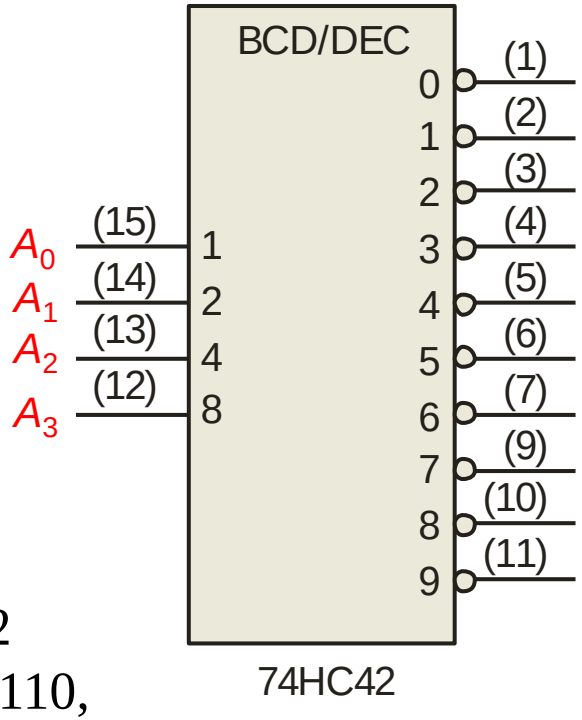


FIGURE 6–30 A 5-bit decoder using 74HC154s.

BCD-Decoders

The BCD-to-decimal decoder converts each BCD code (8421 code) into one of ten possible decimal digit indications.

It is frequently referred as a 4-line-to-10-line decoder or a 1-of-10 decoder.



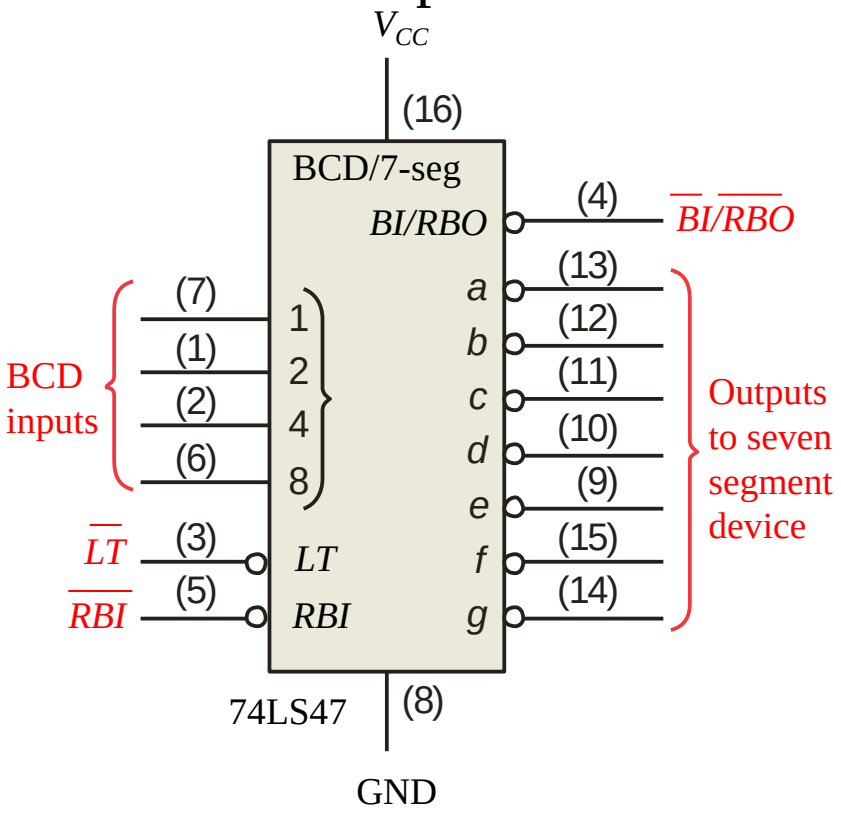
Example Assume the inputs to the 74HC42 decoder are the sequence 0101, 0110, 0011, and 0010. Describe the output.

Solution All lines are HIGH except for one active output, which is LOW. The active outputs are 5, 6, 3, and 2 in that order.

BCD Decoder/Driver

Another useful decoder is the 74LS47. This is a BCD-to-seven segment display with active LOW outputs.

The *a*-*g* outputs are designed for much higher current than most devices (hence the word driver in the name).



LT , RBI and BI/RBO

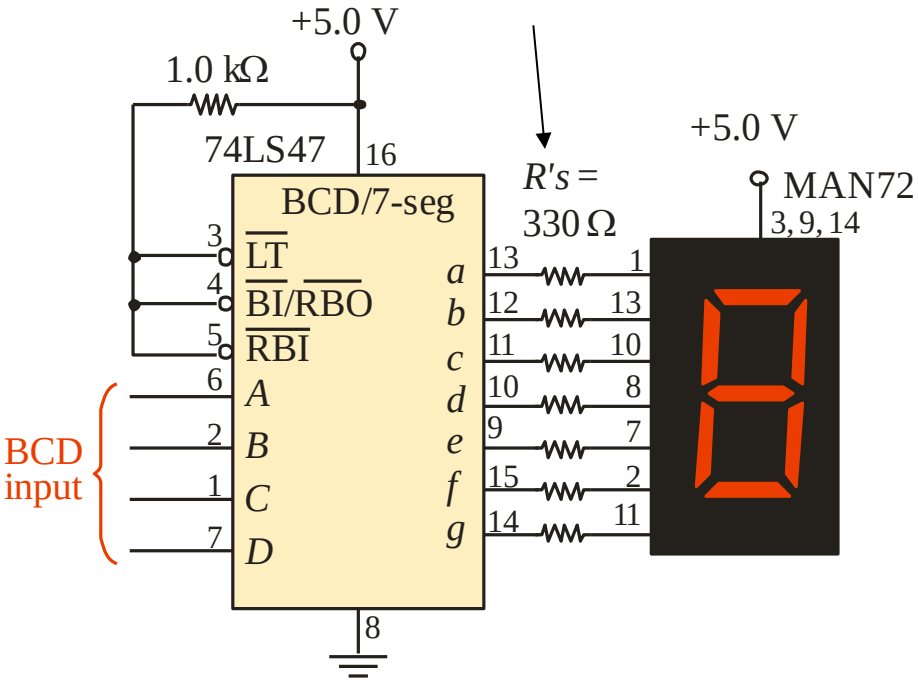
Lamp Test: When a LOW is applied to the LT input and the BI/RBO is HIGH, all of the seven segments in the display are turned on. Lamp test is used to verify that no segments are burned out.

Zero Suppression : Zero suppression is a feature used for multidigit displays to blank out unnecessary zeros. For example, in a 6-digit display the number 6.4 may be displayed as 006.400 if the zeros are not blanked out.

Blanking the zeros at the front of a number is called leading zero suppression and blanking the zeros at the back of the number is called trailing zero suppression. Keep in mind that only nonessential zeros are blanked. With zero suppression, the number 030.080 will be displayed as 30.08 (the essential zeros remain).

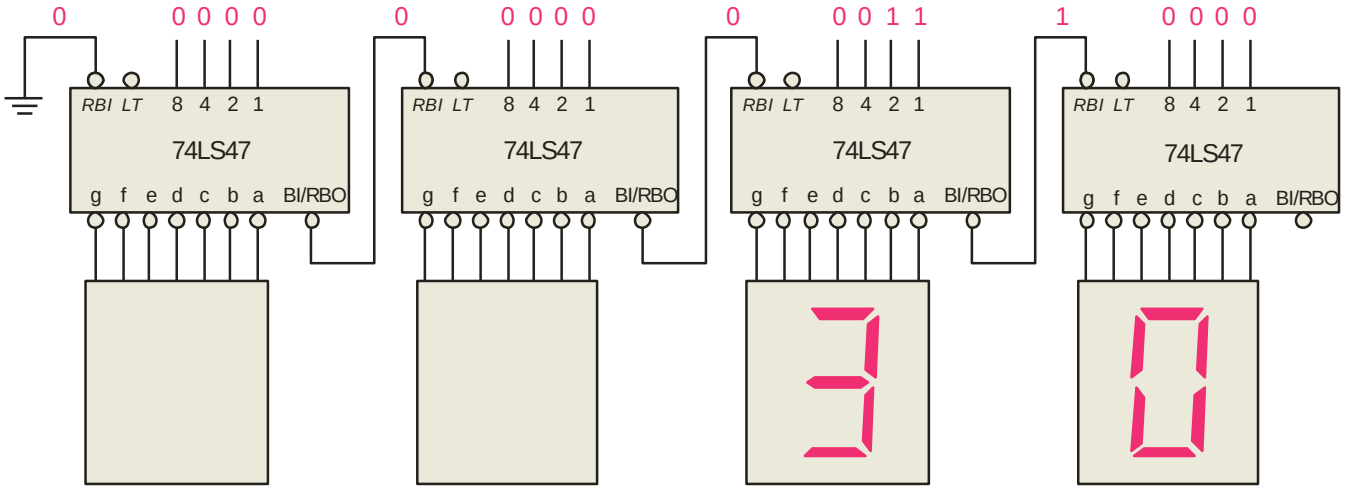
BCD Decoder/Driver

Here the 7447A is connected to an LED seven segment display. Notice the current limiting resistors, required to prevent overdriving the LED display.



BCD Decoder/Driver

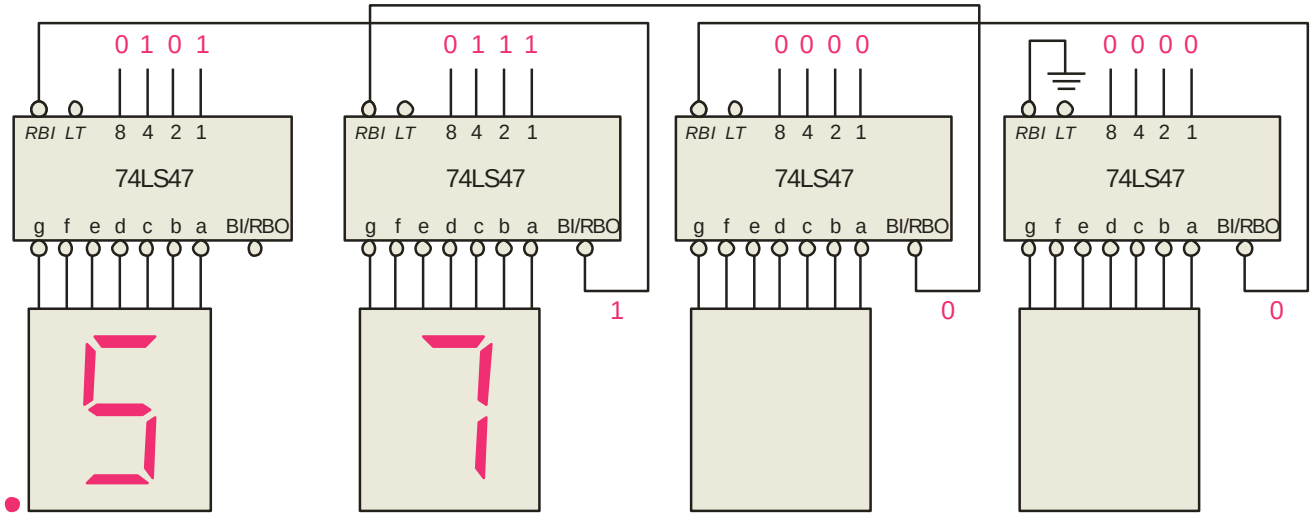
The 74LS47 features leading zero suppression, which blanks unnecessary leading zeros but keeps significant zeros as illustrated here. The $\overline{BI}/\overline{RBO}$ output is connected to the $\overline{RBI}$ input of the next decoder.



Depending on the display type, current limiting resistors may be required.

BCD Decoder/Driver

Trailing zero suppression blanks unnecessary trailing zeros to the right of the decimal point as illustrated here. The *RBI* input is connected to the *BI/RBO* output of the following decoder.



Encoders

An encoder is a combinational logic circuit that essentially performs a “reverse” decoder function.

An encoder accepts an active level on one of its inputs representing a digit, such as a decimal digit, and converts it to a coded output, such as BCD or binary.

The process of converting from familiar numbers to a coded format is called *encoding*.

InfoNote

An assembler can be thought of as a software encoder because it interprets the mnemonic instructions with which a program is written and carries out the applicable encoding to convert each mnemonic to a machine code instruction (series of 1s and 0s) that the processor can understand.

Examples of mnemonic instructions for a processor are ADD, MOV (move data), MUL (multiply), XOR, JMP (jump), and OUT (output to a port).

The Decimal-to-BCD Encoder

This type of encoder has ten inputs—one for each decimal digit—and four outputs corresponding to the BCD code, as shown in Figure 6–36. This is a basic 10-line-to-4-line encoder.

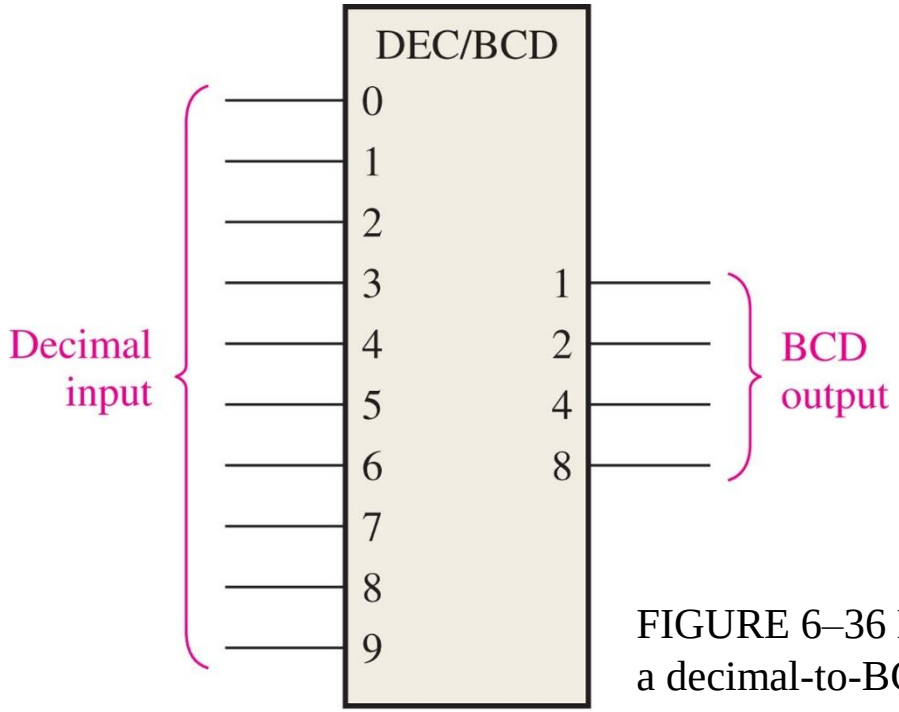


FIGURE 6–36 Logic symbol for a decimal-to-BCD encoder.

The BCD (8421) code is listed in Table 6–6. From this table you can determine the relationship between each BCD bit and the decimal digits in order to analyze the logic.

For instance, the most significant bit of the BCD code, A_3, is always a 1 for decimal digit 8 or 9.

An OR expression for bit A_3 in terms of the decimal digits can therefore be written as

$$A_3 = 8 + 9$$

TABLE 6–6

Decimal Digit	BCD Code A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

The Decimal-to-BCD Encoder

Bit A_2 is always a 1 for decimal digit 4, 5, 6 or 7 and can be expressed as an OR function as follows:

$$A_2 = 4 + 5 + 6 + 7$$

Bit A_1 is always a 1 for decimal digit 2, 3, 6, or 7 and can be expressed as:

$$A_1 = 2 + 3 + 6 + 7$$

Finally, A_0 is always a 1 for decimal digit 1, 3, 5, 7, or 9. The expression for A_0 is:

$$A_0 = 1 + 3 + 5 + 7 + 9$$

The Decimal-to-BCD Encoder

Now let's implement the logic circuitry required for encoding each decimal digit to a BCD code by using the logic expressions just developed.

It is simply a matter of ORing the appropriate decimal digit input lines to form each BCD output. The basic encoder logic resulting from these expressions is shown in Figure 6–37.

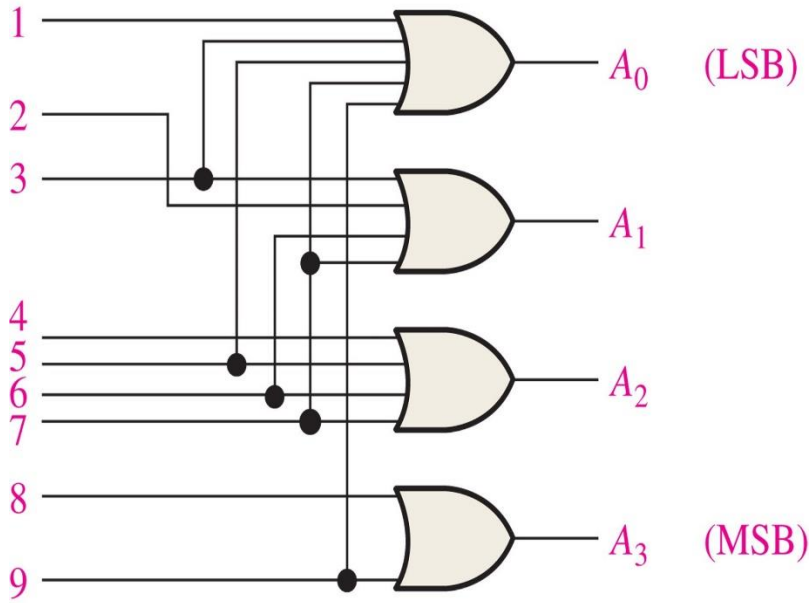


FIGURE 6–37 Basic logic diagram of a decimal-to-BCD encoder. A 0-digit input is not needed because the BCD outputs are all LOW when there are no HIGH inputs.

The Decimal-to-BCD Encoder

The basic operation of the circuit in Figure 6–37 is as follows:

When a HIGH appears on one of the decimal digit input lines, the appropriate levels occur on the four BCD output lines.

For instance, if input line 9 is HIGH (assuming all other input lines are LOW), this condition will produce a HIGH on outputs A_0 and A_3 and LOWs on outputs A_1 and A_2, which is the BCD code (1001) for decimal 9.

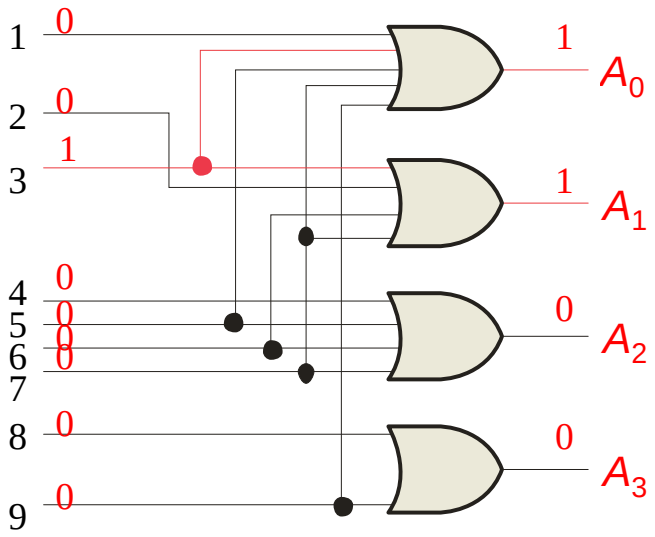
Encoders

Example

Show how the decimal-to-BCD encoder converts the decimal number 3 into a BCD 0011.

Solution

The top two OR gates have ones as indicated with the red lines. Thus the output is 0011.



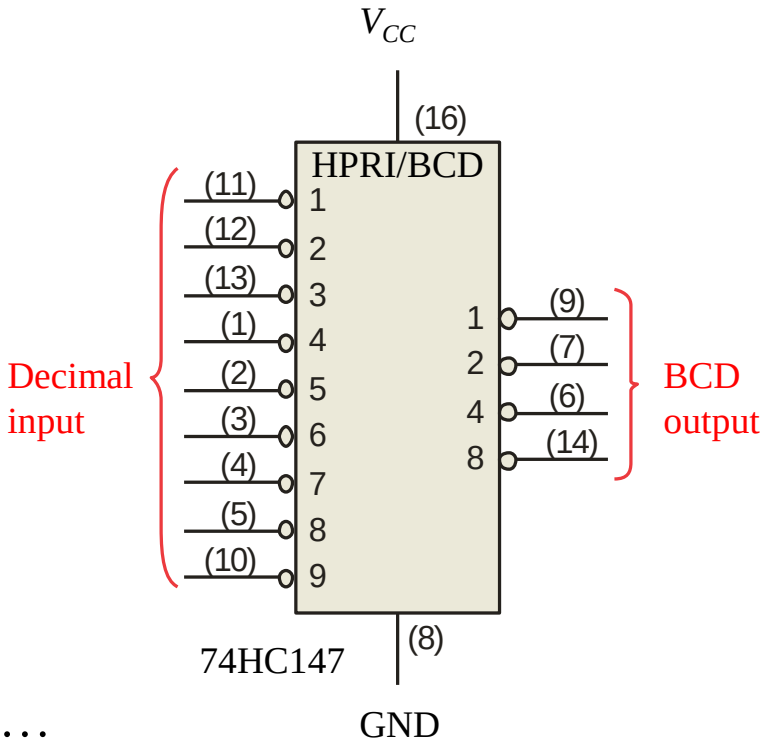
Encoders

The 74HC147 is an example of an IC encoder. It has ten active-LOW inputs and converts the active input to an active-LOW BCD output.

This device offers additional flexibility in that it is a **priority encoder**. This means that if more than one input is active, the one with the highest order decimal digit will be active.

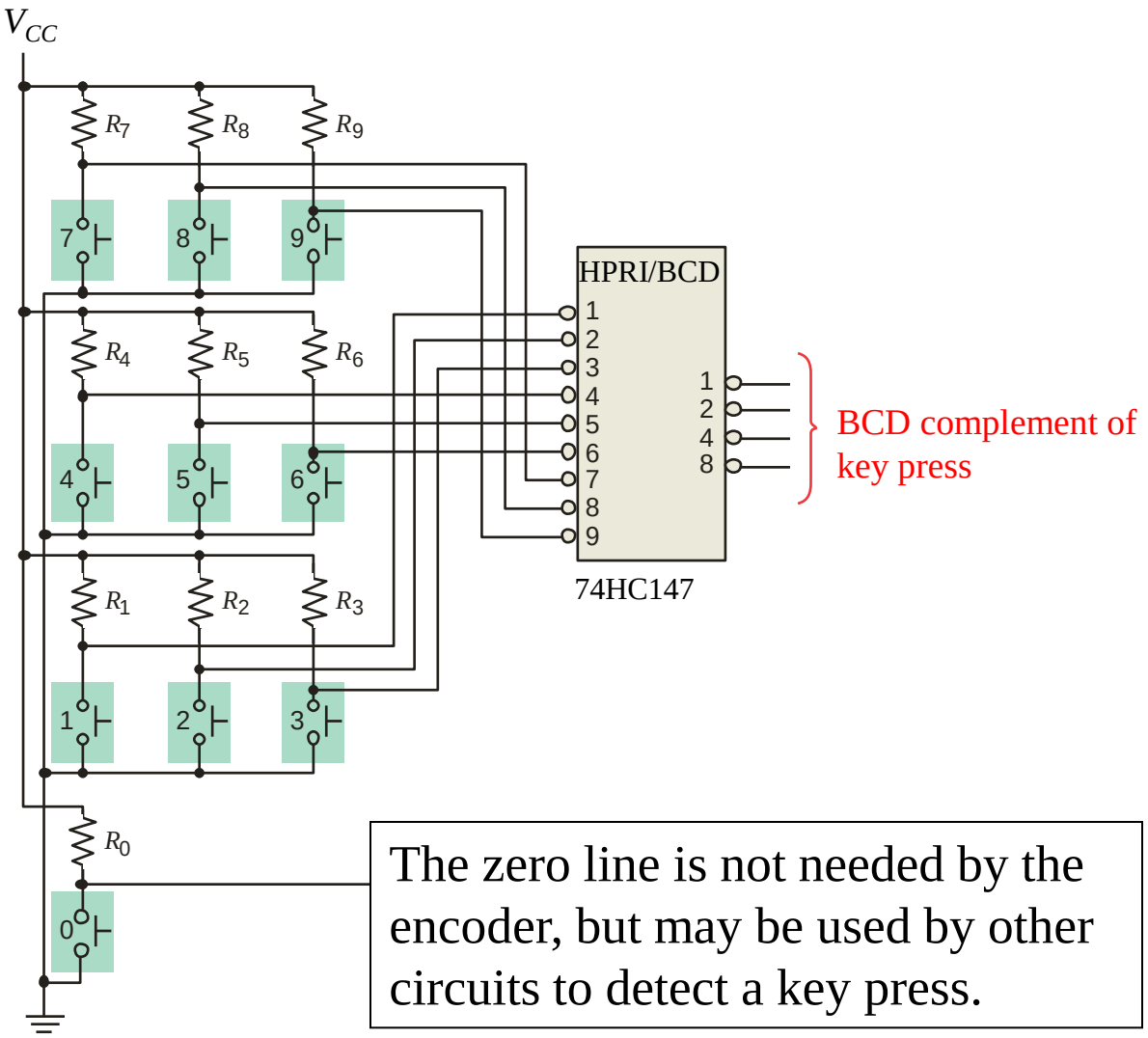
HPRI = highest value priority input

The next slide shows an application …



Encoders

Keyboard encoder



The zero line is not needed by the encoder, but may be used by other circuits to detect a key press.

Code converters

The binary numbers representing the weights of the BCD bits are summed to produce the total binary number.

Let's examine an 8-bit BCD code (one that represents a 2-digit decimal number) to understand the relationship between BCD and binary.

For instance, you already know that the decimal number 87 can be expressed in BCD as:

$$\underbrace{1000}_{8}\ \underbrace{0111}_{7}$$

Code converters

The left-most 4-bit group represents 80, and the right-most 4-bit group represents 7. That is, the left-most group has a weight of 10, and the right-most group has a weight of 1.

Within each group, the binary weight of each bit is as follows:

	Tens Digit				Units Digit			
Weight:	80	40	20	10	8	4	2	1
Bit designation:	B_3	B_2	B_1	B_0	A_3	A_2	A_1	A_0

The binary equivalent of each BCD bit is a binary number representing the weight of that bit within the total BCD number. This representation is given in Table 6–7.

Code converters

TABLE 6–7

Binary representations of BCD bit weights.

BCD Bit	BCD Weight	(MSB) Binary Representation						(LSB)
		64	32	16	8	4	2	1
A_0	1	0	0	0	0	0	0	1
A_1	2	0	0	0	0	0	1	0
A_2	4	0	0	0	0	1	0	0
A_3	8	0	0	0	1	0	0	0
B_0	10	0	0	0	1	0	1	0
B_1	20	0	0	1	0	1	0	0
B_2	40	0	1	0	1	0	0	0
B_3	80	1	0	1	0	0	0	0

If the binary representations for the weights of all the 1s in the BCD number are added, the result is the binary number that corresponds to the BCD number. Example 6–12 illustrates this.

EXAMPLE 6–12

Convert the BCD numbers 00100111 (decimal 27) and 10011000 (decimal 98) to binary.

Solution

Write the binary representations of the weights of all 1s appearing in the numbers, and then add them together.

80	40	20	10	8	4	2	1
0	0	1	0	0	1	1	1

0000001	1
0000010	2
0000100	4
+ 0010100	20
0011011	Binary number for decimal 27

80	40	20	10	8	4	2	1
1	0	0	1	1	0	0	0

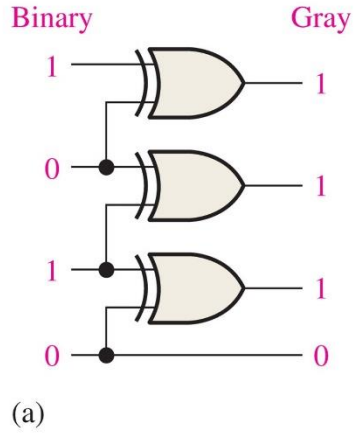
0001000	8
0001010	10
+ 1010000	80
1100010	Binary number for decimal 98

Code converters

The basic process for Gray-binary conversions was covered in Chapter 2. Exclusive-OR gates can be used for these conversions.

Figure 6–40 shows a 4-bit binary-to-Gray code converter, and Figure 6–41 illustrates a 4-bit Gray-to-binary converter.

Figure 6–40



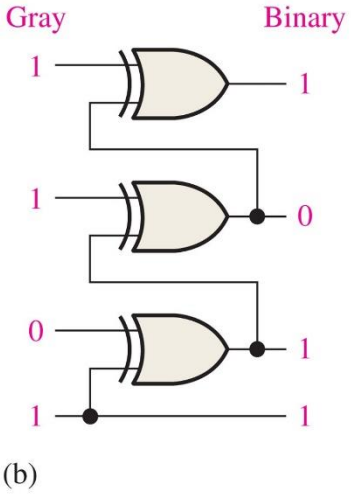
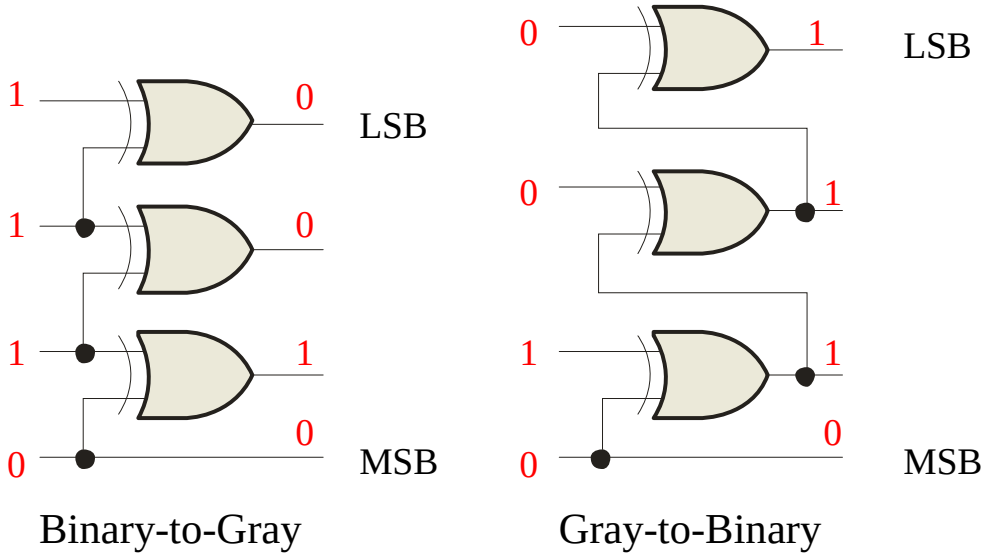


Figure 6–41

Code converters

Example Show the conversion of binary 0111 to Gray and back.

Solution



InfoNote

A bus is a multiple conductor pathway along which electrical signals are sent from one part of a computer to another. In computer networks, a shared bus is one that is connected to all the microprocessors in the system in order to exchange data.

A shared bus may contain memory and input/output devices that can be accessed by all the microprocessors in the system.

Access to the shared bus is controlled by a bus arbiter (a multiplexer of sorts) that allows only one microprocessor at a time to use the system's shared bus.

Multiplexers

A multiplexer (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.

The basic multiplexer has several data-input lines and a single output line. It also has data-select inputs, which permit digital data on any one of the inputs to be switched to the output line.

Multiplexers are also known as data selectors.

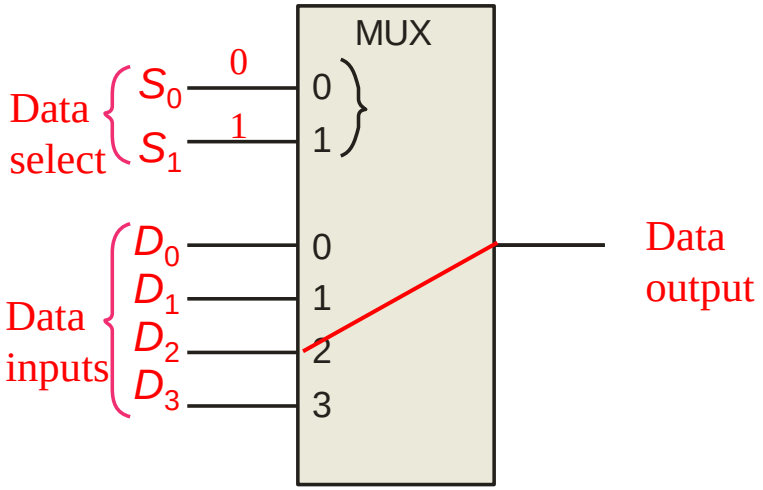
Multiplexers

A multiplexer (MUX) selects one data line from two or more input lines and routes data from the selected line to the output. The particular data line that is selected is determined by the select inputs.

Two select lines are shown here to choose any of the four data inputs.

Question

Which data line is selected if $S_1S_0 = 10$? D_2



MUX Implementation

Now let's look at the logic circuitry required to perform this MUX operation.

The data output is equal to the state of the *selected* data input. You can therefore, derive a logic expression for the output in terms of the data input and the select inputs.

The data output is equal to D_0 only if $S_1 = 0$ and $S_0 = 0$: $Y = D_0S_1S_0$.
The data output is equal to D_1 only if $S_1 = 0$ and $S_0 = 1$: $Y = D_1S_1S_0$.
The data output is equal to D_2 only if $S_1 = 1$ and $S_0 = 0$: $Y = D_2S_1S_0$.
The data output is equal to D_3 only if $S_1 = 1$ and $S_0 = 1$: $Y = D_3S_1S_0$.

When these terms are ORed, the total expression for the data output is:

$$Y = D_0\overline{S}_1\overline{S}_0 + D_1\overline{S}_1S_0 + D_2S_1\overline{S}_0 + D_3S_1S_0$$

The implementation of this equation requires four 3-input AND gates, a 4-input OR gate, and two inverters to generate the complements of S_1 and S_0, as shown in Figure 6–44.

Because data can be selected from any one of the input lines, this circuit is also referred to as a data selector.

TABLE 6–8

Data selection for a 1-of-4-multiplexer.

Data-Select Inputs		Input Selected
S_1	S_0	
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

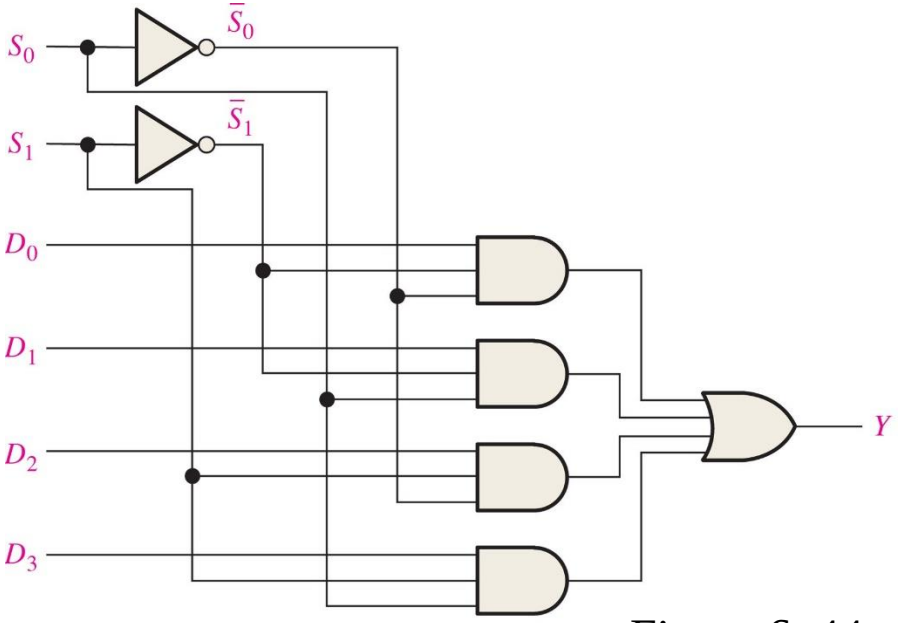


Figure 6–44

Demultiplexers

A Demultiplexer (DEMUX) basically reverses the multiplexing function.

It takes digital information from one line and distributes it to a given number of output lines.

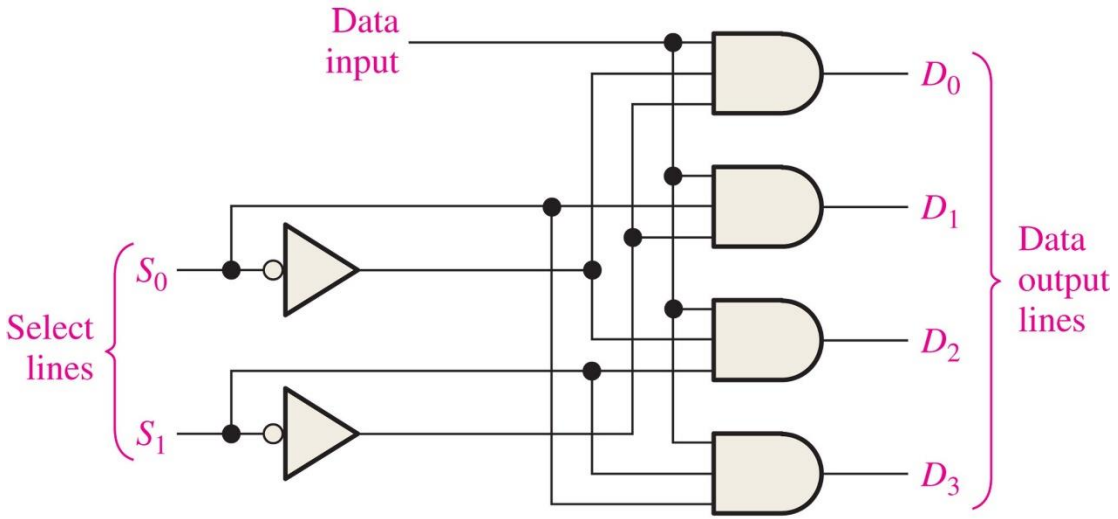
For this reason, the Demultiplexer is also known as a data distributor.

As you will learn, decoders can also be used as Demultiplexers.

Figure 6–52 shows a 1-line-to-4-line Demultiplexer (DEMUX) circuit. The data-input line goes to all of the AND gates.

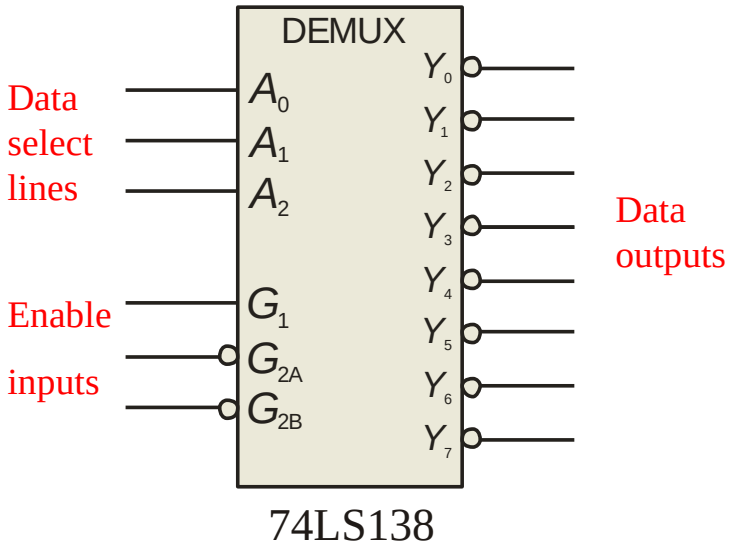
The two data-select lines enable only one gate at a time, and the data appearing on the data-input line will pass through the selected gate to the associated data-output line.

Figure 6–52



Demultiplexers

The 74LS138 was introduced previously as a decoder but can also serve as a DEMUX. When connected as a DEMUX, data is applied to one of the enable inputs, and routed to the selected output line depending on the select variables. Note that the outputs are active-LOW.



EXAMPLE 6–18

The serial data-input waveform (Data in) and data-select inputs (S_0 and S_1) are shown in Figure 6–53. Determine the data-output waveforms on D_0 through D_3 for the demultiplexer in Figure 6–52.

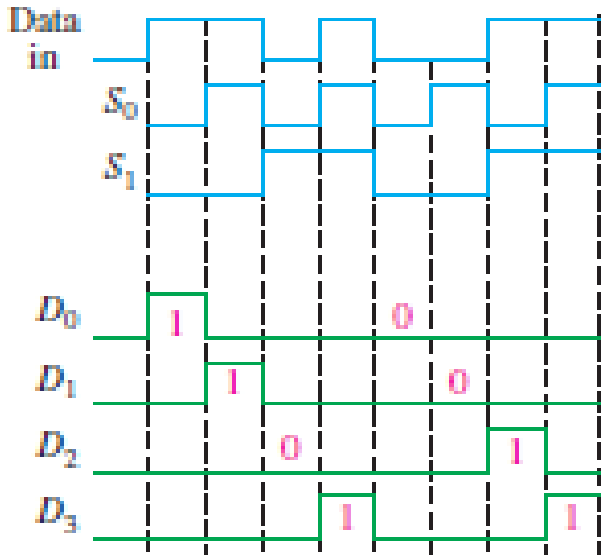
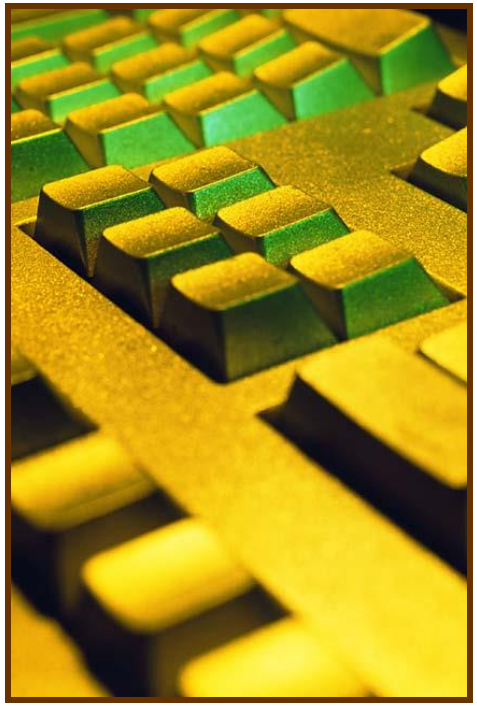


FIGURE 6–53

Solution

Notice that the select lines go through a binary sequence so that each successive input bit is routed to D_0, D_1, D_2, and D_3 in sequence, as shown by the output waveforms in Figure 6–53.

Parity Generators/Checkers



Parity is an error detection method that uses an extra bit appended to a group of bits to force them to be either odd or even. In even parity, the total number of ones is even; in odd parity the total number of ones is odd.

Example The ASCII letter S is 1010011. Show the parity bit for the letter S with odd and even parity.

Solution

S with odd parity = 11010011

S with even parity = 01010011

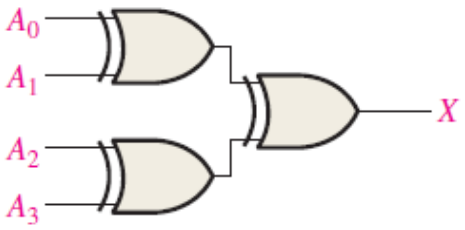
Parity Generators/Checkers

To determine if a given code has **even parity** or **odd parity,** all the bits in that code are summed. The modulo-2 sum of two bits can be generated by an exclusive-OR gate, as shown in Figure 6–55(a); the modulo-2 sum of four bits can be formed by three exclusive-OR gates connected as shown in Figure 6–55(b); and so on.

When the number of 1s on the inputs is even, the output *X* is 0 (LOW). When the number of 1s is odd, the output *X* is 1 (HIGH).

(a) Summing of two bits

Figure 6–52



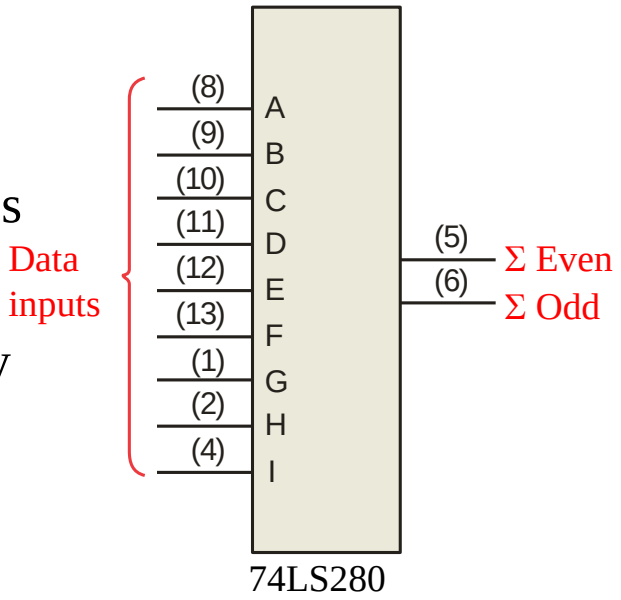
(b) Summing of four bits

Parity Generators/Checkers

The 74LS280 can be used to generate a parity bit or to check an incoming data stream for even or odd parity.

Checker: The 74LS280 can test codes with up to 9 bits. The even output will normally be HIGH if the data lines have even parity; otherwise it will be LOW. Likewise, the odd output will normally be HIGH if the data lines have odd parity; otherwise it will be LOW.

Generator: To generate <u>even</u> parity, the parity bit is taken from the <u>odd</u> parity output. To generate <u>odd</u> parity, the output is taken from the <u>even</u> parity output.



Selected Key Terms

Full-adder	A digital circuit that adds two bits and an input carry bit to produce a sum and an output carry.
Cascading	Connecting two or more similar devices in a manner that expands the capability of one device.
Ripple carry	A method of binary addition in which the output carry from each adder becomes the input carry of the next higher order adder.
Look-ahead carry	A method of binary addition whereby carries from the preceding adder stages are anticipated, thus eliminating carry propagation delays.

Selected Key Terms

Decoder	A digital circuit that converts coded information into a familiar or noncoded form.
Encoder	A digital circuit that converts information into a coded form.
Priority encoder	An encoder in which only the highest value input digit is encoded and any other active input is ignored.
Multiplexer (MUX)	A circuit that switches digital data from several input lines onto a single output line in a specified time sequence.
Demultiplexer (DEMUX)	A circuit that switches digital data from one input line onto a several output lines in a specified time sequence.

Quiz

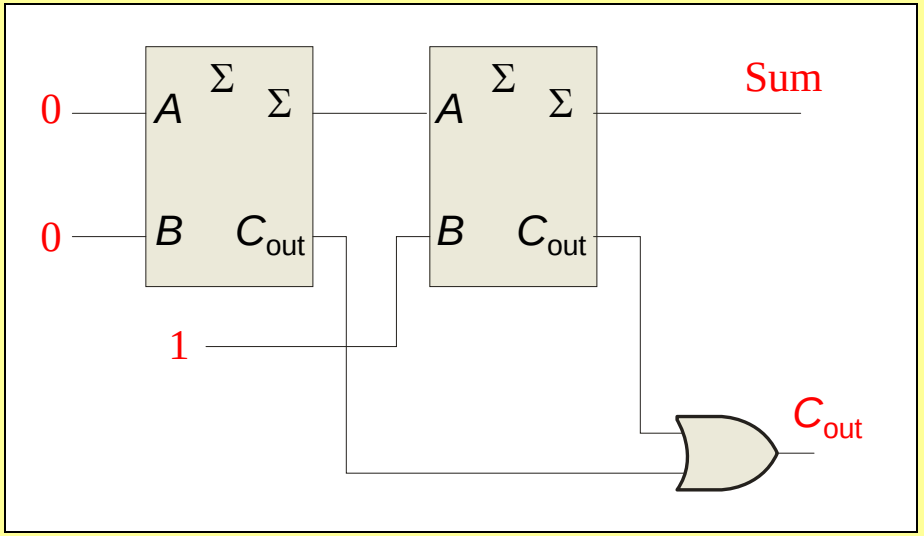
1. For the full-adder shown, assume the input bits are as shown with $A = 0$, $B = 0$, $C_{in} = 1$. The Sum and C_{out} will be

a. Sum = 0 $C_{out} = 0$

b. Sum = 0 $C_{out} = 1$

c. Sum = 1 $C_{out} = 0$

d. Sum = 1 $C_{out} = 1$



Quiz

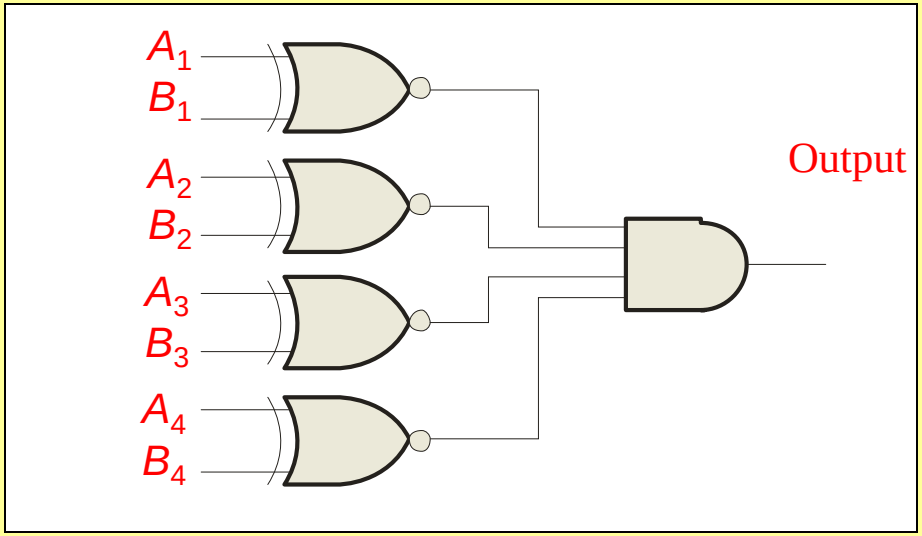
2. The output will be LOW if

a. $A < B$

b. $A > B$

c. both a and b are correct

d. $A = B$



Quiz

3. If you expand two 4-bit comparators to accept two 8-bit numbers, the output of the least significant comparator is

a. equal to the final output

b. connected to the cascading inputs of the most significant comparator

c. connected to the output of the most significant comparator

d. not used

Quiz

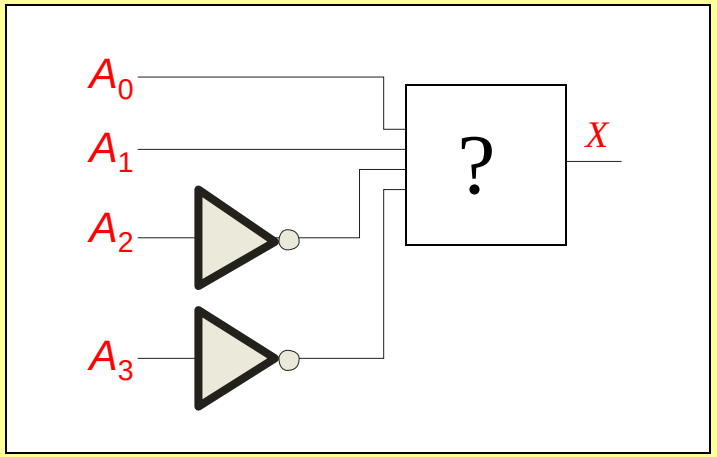
4. Assume you want to decode the binary number 0011 with an active-LOW decoder. The missing gate should be

a. an AND gate

b. an OR gate

c. a NAND gate

d. a NOR gate



Quiz

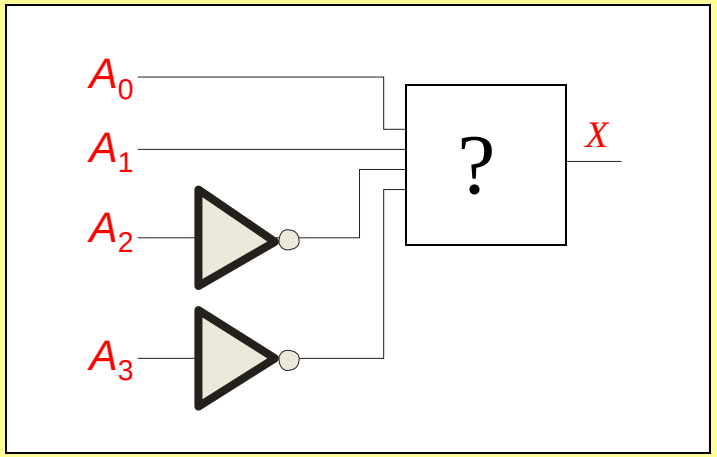
5. Assume you want to decode the binary number 0011 with an active-HIGH decoder. The missing gate should be

a. an AND gate

b. an OR gate

c. a NAND gate

d. a NOR gate



Quiz

6. The 74138 is a 3-to-8 decoder. Together, two of these ICs can be used to form one 4-to-16 decoder. To do this, connect

a. one decoder to the LSBs of the input; the other decoder to the MSBs of the input

b. all chip select lines to ground

c. all chip select lines to their active levels

d. one chip select line on each decoder to the input MSB

Quiz

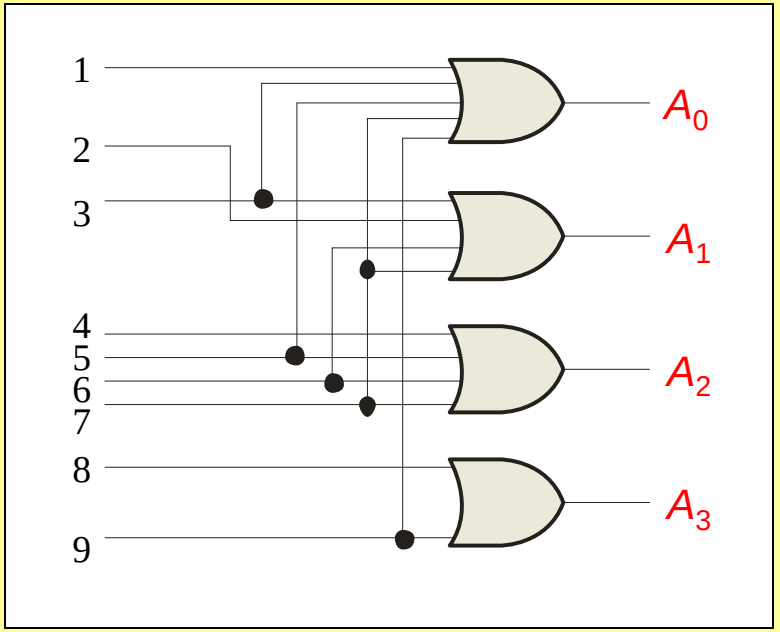
7. The decimal-to-binary encoder shown does not have a zero input. This is because

a. when zero is the input, all lines should be LOW

b. zero is not important

c. zero will produce illegal logic levels

d. another encoder is used for zero



Quiz

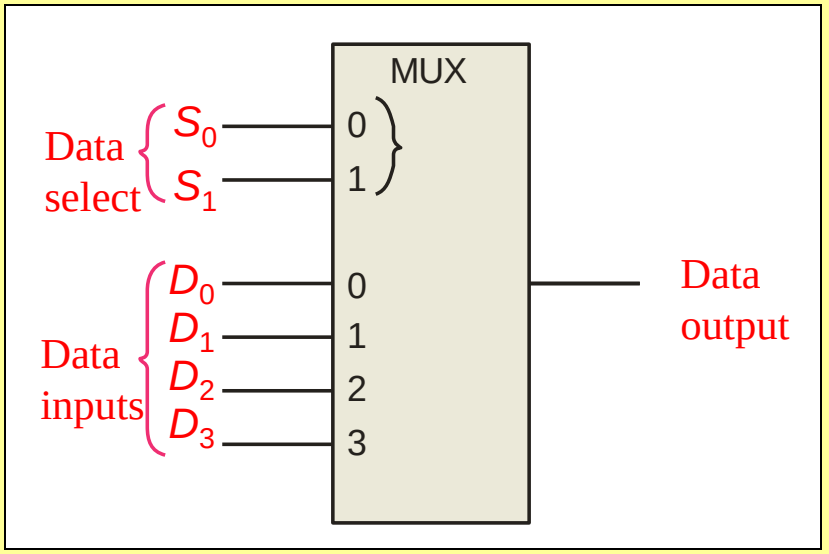
8. If the data select lines of the MUX are $S_1S_0 = 11$, the output will be

a. LOW

b. HIGH

c. equal to D_0

d. equal to D_3



Quiz

9. The 74138 decoder can also be used as

   a. an encoder

   b. a DEMUX

   c. a MUX

   d. none of the above

Quiz

10. The 74LS280 can generate even or odd parity. It can also be used as

a. an adder

b. a parity tester

c. a MUX

d. an encoder

Quiz

Answers:

1. c
2. c
3. b
4. c
5. a
6. d
7. a
8. d
9. b
10. b