

# **DIGITAL ABSTRACTION AND LOGIC GATES**

# Introduction

Value discretization forms the basis of the *digital abstraction*. The idea is to lump signal values that fall within some interval into a single value (see in Figure 9.1) where a voltage signal was discretized into two levels:

- voltage value between 0 volts and 2.5 volts is treated as a “0,” and
- a value between 2.5 volts and 5 volts as a “1.”

Correspondingly, to transmit the logical value “0” over a wire, we place the nominal voltage level of 1.25 on the wire. Similarly, to transmit the logical “1,” we place the nominal voltage level of 3.75 volts on the wire (see in Figure 9.1)

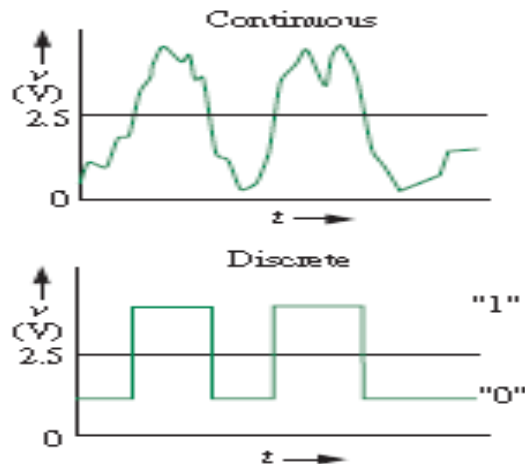


FIGURE 9.1 Value discretization into two levels.

# Introduction (continued)

Although the digital approach seems wasteful of signal dynamic range, it has a significant advantage over analog transmission in the presence of noise. Notice, this representation is immune to symmetric noise with a peak to peak value less than 2.5 V. To illustrate, consider the situation depicted in Figure 9.2 in which a sender desires to transmit a value  $A$  to a receiver.

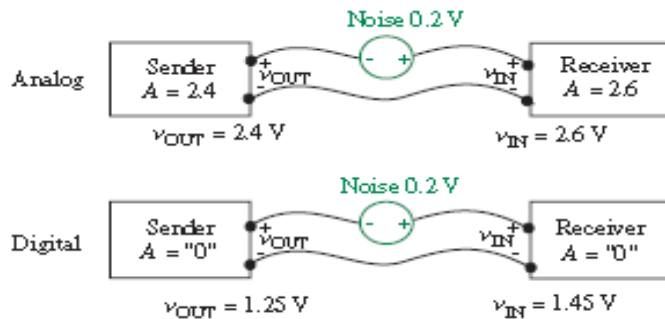
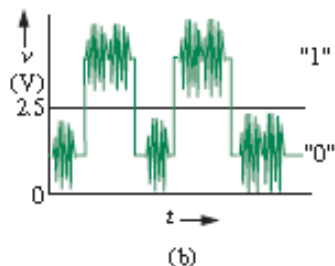
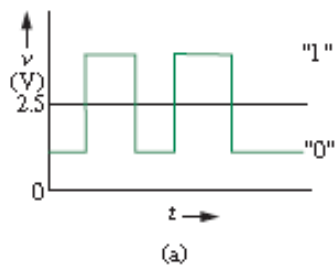


FIGURE 9.2 Signal transmission in the presence of noise. The noise is represented as a series voltage source.

In the digital case, suppose that the value  $A$  is a logical “0.” The sender transmits this value of  $A$  by representing it as a voltage level of 1.25 V on the wire, which is received as a voltage level of 1.45 by the receiver because of the series noise source. In this situation, since the received voltage falls below the 2.5-V threshold, the receiver interprets it correctly as a logical “0.” Thus, the sender and receiver were able to communicate without error in the digital case.

# Introduction (continued)

To illustrate further, consider the waveforms in Figure 9.3. Figure 9.3a shows a discretized signal waveform produced by a sender corresponding to a “0,” “1,” “0,” “1,” “0” sequence. Figure 9.3b shows the same signal with the superposition of some amount of noise, possibly during transmission through a noisy environment.



Two levels of signal precision are sufficient for many applications. As one example, logic computations involve signals that commonly take on one of two values: TRUE or FALSE.

FIGURE 9.3 Noise immunity for discretized signals: (a) a digital signal produced by a sender; (b) the signal received by a receiver following transmission through a noisy environment.

# Introduction (continued)

There are other applications that require more levels of precision:

A speech signal processing application might involve speech signals with 256 or more levels of precision.

One approach to achieving more precision is to use coding to create multi-digit numbers. When each digit takes on one of two values, the digit is called a *binary digit*, or bit.

Much as the familiar decimal system uses multiple digits to represent numbers other than 0 through 9, the binary system uses multiple bits to represent numbers other than 0 or 1.

Multi-bit signals are commonly transmitted by allocating multiple wires one for each bit, or occasionally, by time multiplexing multiple bits on a single wire.

# Introduction (continued)

The two-level representation is commonly known as the *binary representation*. Virtually all digital circuits use the binary representation because two-level circuits are much easier to build than multilevel circuits. The two levels in the binary representation are variously called (a) TRUE or FALSE, (b) ON or OFF, (c) 1 or 0, (d) HIGH or LOW.

Table 9.1 depicts these and several other physical realizations of the binary signals, TRUE and FALSE.

TABLE 9.1 Binary signal representation.  $v$  represents the value of some parameter.

TRUE	FALSE
0 V	5 V
5 V	0 V
2 V	0 V
0 V	1 V
ON	OFF
$0\text{ V} < v < 2.5\text{ V}$	$2.5\text{ V} < v < 5\text{ V}$
$0\text{ V} < v < 1\text{ V}$	$4\text{ V} < v < 5\text{ V}$
$0\text{ }\mu\text{A}$	$2\text{ }\mu\text{A}$

# Voltage Levels and the static discipline

The representations differed not only in the signal type (for example, current versus voltage), but also in the signal values (for example, 5 V versus 4 V to represent a logical 1).

Because we require that digital devices built by various manufacturers talk to each other, the devices must adhere to a common representation.

The representation must allow for large enough design margins so that devices can be built out of a wide range of technologies.

Furthermore, the representation should be such that the devices operate correctly even in the presence of some amount of noise.

The *static discipline* is a specification for digital devices. The static discipline requires devices to adhere to a common representation, and to guarantee that they interpret correctly inputs that are valid logical signals according to the common representation, and to produce outputs that are valid logical signals provided they receive valid logical inputs.

# Voltage Levels and the static discipline (continued)

One of the representations we saw earlier divided a voltage range into two intervals and associated a logic value with each, namely,

$$\text{Logic 0 : } 0.0 \text{ V} \leq V < 2.5 \text{ V.} \quad (9.1)$$

$$\text{Logic 1 : } 2.5 \text{ V} \leq V \leq 5.0 \text{ V.} \quad (9.2)$$

This simple representation is illustrated in Figure 9.4. According to this representation, if a receiver saw 2 V on a wire it would interpret it as a 0. Similarly, a receiver would interpret 4 V on a wire as a 1. Assume, for now, that values outside this range are invalid.

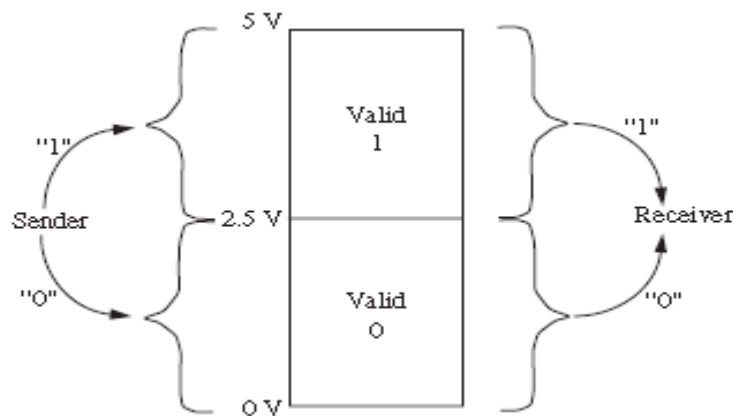


FIGURE 9.4 Senders and receivers use an agreed-upon mapping between voltage levels and logical signals so that they can communicate with each other.



# Voltage Levels and the static discipline (continued)

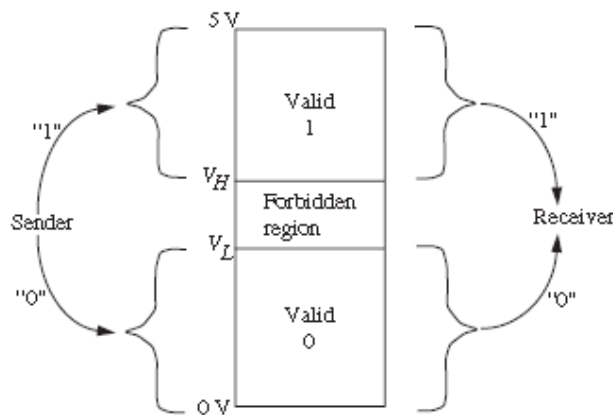
There is one problem, however. What does the receiver do if it sees a voltage level of 2.5 V on the wire? Does it interpret this signal value as a logical 0 or as a logical 1?

To eliminate such confusion, we further prescribe a *forbidden region* that separates the two valid regions. We further allow the behavior of the receiving device to be undefined if it sees a voltage in the forbidden region.

Thus, the correspondence between voltage levels and logic signals from the viewpoint of a receiver might look like:

$$\text{Logic 0 : } 0 \text{ V} \leq V \leq 2 \text{ V.} \quad (9.3)$$

$$\text{Logic 1 : } 3 \text{ V} \leq V \leq 5 \text{ V.} \quad (9.4)$$



This representation using a forbidden region is illustrated in Figure 9.5. In this representation, a receiver interprets signals above 3 V as a logical 1 and voltages below 2 V as a logical 0. Signal voltages between 2 V and 3 V are invalid.

FIGURE 9.5 A representation with forbidden regions showing the mapping of voltage levels and logical values.

# Voltage Levels and the static discipline (continued)

It often turns out that practical circuits are able to correctly interpret values outside the extremum points (below 0 volts for a logical 0 and above 5 V for a logical 1), within certain limits, of course. When devices can make this interpretation, our representation with the forbidden region allows senders to output any voltage value above  $V_H$  for a logical 1. Similarly, senders can output any value below  $V_L$  for a logical 0 (see Figure 9.6).

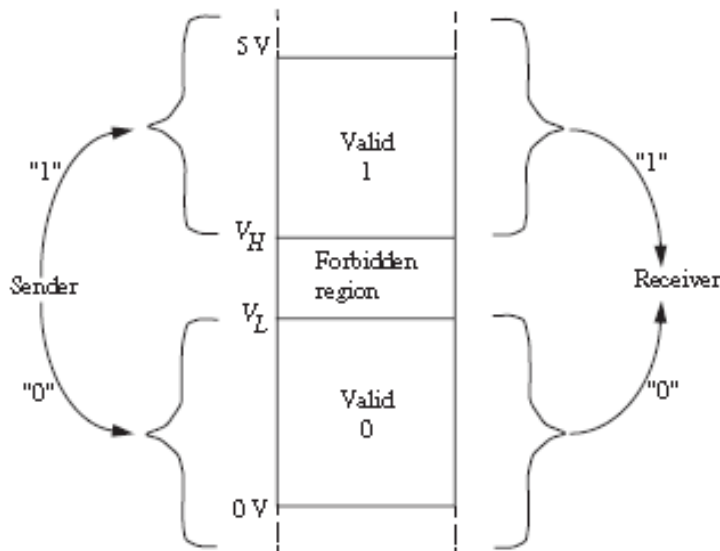


FIGURE 9.6 For many practical devices, a sender can output any voltage value above  $V_H$  for a logical 1, and any voltage value below  $V_L$  for a logical 0.

# Voltage Levels and the static discipline (continued)

There is one other problem with our representations illustrated in Figures 9.6 and 9.5: They do not offer any immunity to noise. As an illustration of the notion of noise margins, consider the two situations in Figure 9.7.

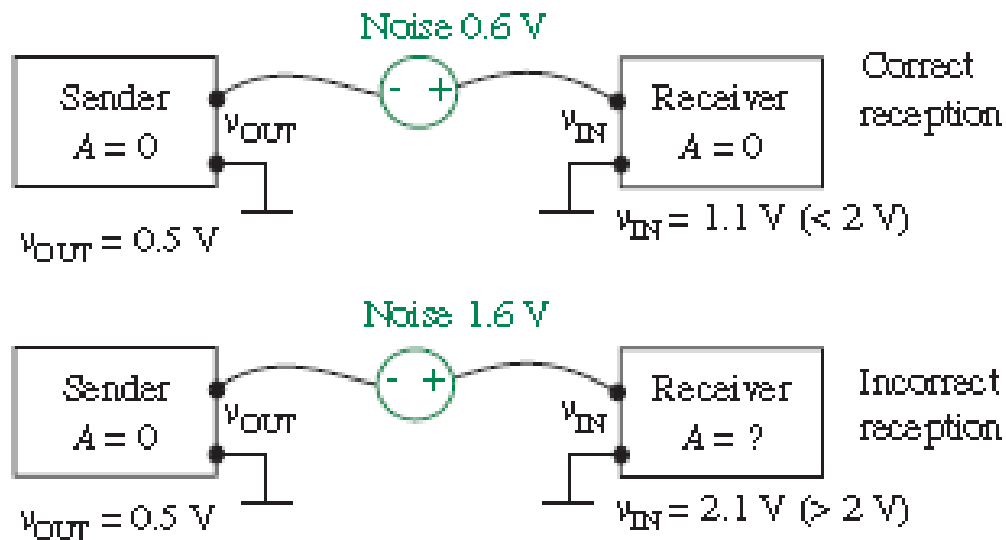


FIGURE 9.7 Noise margins and signal transmission.

# Voltage Levels and the static discipline (continued)

The tighter bounds on the voltage values for a sender compared to those for a receiver result in an asymmetry in input and output voltage thresholds.

This asymmetry is reflected in Figure 9.8, which shows the correspondence between valid voltage levels and logic signals that is in common use in digital circuits.

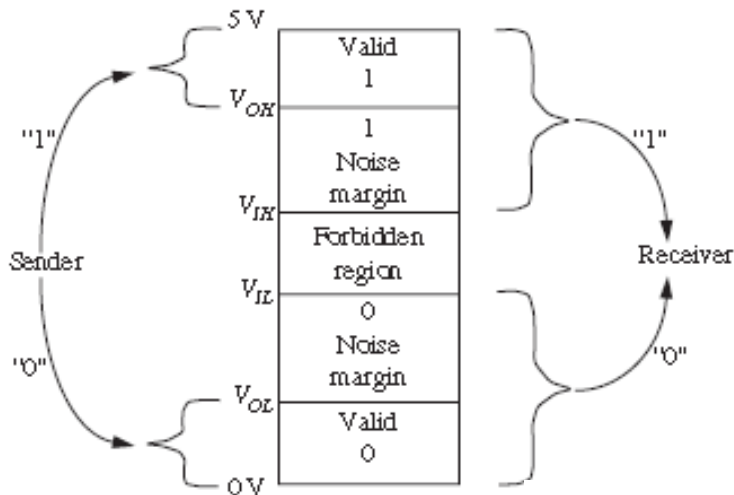


FIGURE 9.8 A mapping between voltage levels and logical signals that provides noise margins. For a logical high, senders must output values in the  $V_{OH}$  to 5-V range. For a logical low, senders must output values in the 0 V to  $V_{OL}$  range. Receivers must correspondingly interpret values greater than  $V_{IH}$  as a logical high, and output values lower than  $V_{IL}$  as a logical low.

To send a logical 0, the sender must produce an output voltage value that is less than  $V_{OL}$ . Correspondingly, the receiver must interpret input voltages below  $V_{IL}$  as a logical 0.

To allow for a reasonable noise margin,  $V_{IL}$  must be greater than  $V_{OL}$ .

# Voltage Levels and the static discipline (continued)

Similarly, to send a logical 1, the sender must produce an output voltage value that is greater than  $V_{OH}$ . Further, the receiver must interpret voltages above  $V_{IH}$  as a logical 1.

To allow for a reasonable noise margin,  $V_{OH}$  must be greater than  $V_{IH}$ . We can define both a noise margin for transmitting logical 1's and for transmitting logical 0's.

*Noise Margin:* The absolute value of the difference between the prescribed output voltage for a given logical value and the corresponding forbidden region voltage threshold for the receiver is called the *noise margin* for that logical value.

Figure 9.9a illustrates a scenario in which a sender outputs a 01010 sequence by producing the appropriate output voltage levels (between  $V_{OH}$  and 5 V for a logical 1, and between 0 V and  $V_{OL}$  for a logical 0).

Provided that the noise does not exceed the noise margins (voltages for a logical 0 do not exceed  $V_{IL}$  and voltages for a logical 1 do not fall below  $V_{IH}$ ), a receiver is able to correctly interpret the signal as illustrated in Figure 9.9b.

# Voltage Levels and the static discipline (continued)

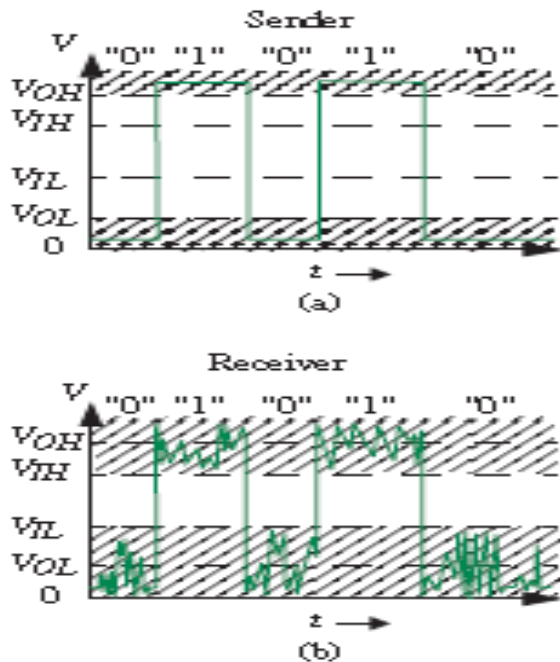


FIGURE 9.9 Senders must output voltages between  $V_{OH}$  and  $5\text{ V}$  to send a logical 1, and between  $0\text{ V}$  and  $V_{OL}$  for a logical 0. Correspondingly, receivers can interpret values greater than  $V_{IH}$  as a logical high, and values lower than  $V_{IL}$  as a logical low. The hashed regions are the valid ranges for senders and receivers.

As illustrated in Figure 9.8, the *noise margin for a logical 0* is given by

$$NM_0 = V_{IL} - V_{OL} \quad (9.5)$$

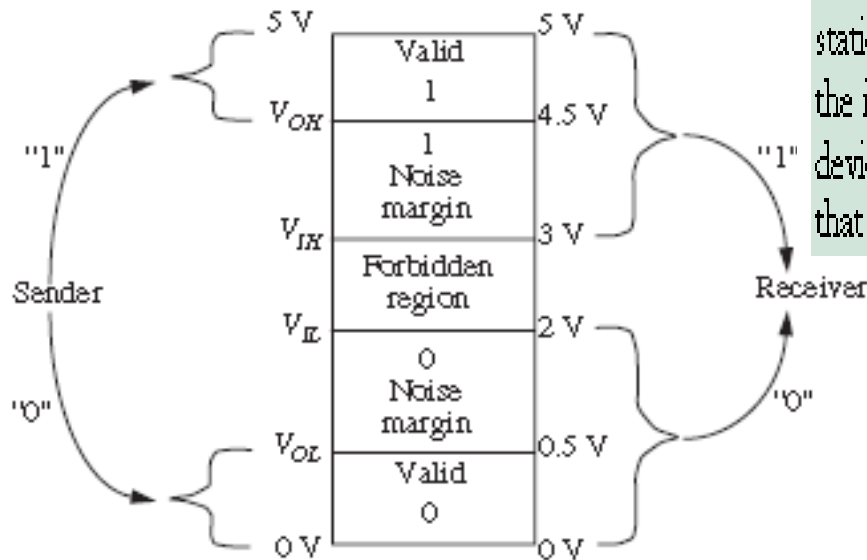
and the *noise margin for a logical 1* is given by

$$NM_1 = V_{OH} - V_{IH} \quad (9.6)$$

The region between  $V_{IL}$  and  $V_{IH}$  is the forbidden region.

# Voltage Levels and the static discipline (continued)

Relating the threshold voltage parameters to the numbers used in our example,  $V_{OH}$  corresponds to 4.5 V,  $V_{OL}$  corresponds to 0.5 V,  $V_{IH}$  corresponds to 3 V, and  $V_{IL}$  corresponds to 2 V. This mapping is illustrated in Figure 9.10.



*Static discipline* The *static discipline* is a specification for digital devices. The static discipline requires devices to interpret correctly voltages that fall within the input thresholds ( $V_{IL}$  and  $V_{IH}$ ). As long as valid inputs are provided to the devices, the discipline also requires the devices to produce valid output voltages that satisfy the output thresholds ( $V_{OL}$  and  $V_{OH}$ ).

FIGURE 9.10 An example of a mapping between voltage levels and logical values.

When designing logic devices, we are often interested in maximizing the noise margins to achieve maximum noise immunity. Referring to Figure 9.8, the 0 noise margin,  $NM0 = V_{IL} - V_{OL}$ , can be maximized by maximizing  $V_{IL}$  and minimizing  $V_{OL}$ . Similarly, the 1 noise margin,  $NM1 = V_{OH} - V_{IH}$ , can be maximized by maximizing  $V_{OH}$  and minimizing  $V_{IH}$ .

## Example 9.1 Observing a static discipline

The device company Yehaa Microelectronics, Inc. has developed a new process technology that is able to produce large quantities of a certain type of digital device known as an *adder* at a very low cost. For a logical 0, their adders produce a voltage level of 0.5 V at their outputs. Similarly, when outputting a logical 1, their adders produce the voltage level of 4.5 V. Furthermore, the Yehaa adders are able to interpret all signals between 0 V and 2 V at their inputs as a logical 0, and all signals between 3 V and 5 V as a logical 1.

Yehaa's sales team discovers that networking equipment company Disco Systems Inc. buys huge quantities of adder devices from a competitor Yikes Devices, Inc. Upon further research, the Yehaa sales team finds that the hardware systems in one of Disco's product lines operate under a static discipline with the following voltage thresholds:

$$V_{IL} = 2 \text{ V}, V_{IH} = 3.5 \text{ V}, V_{OL} = 1.5 \text{ V} \text{ and } V_{OH} = 4 \text{ V}.$$



## Voltage Levels and the static discipline (continued)

In other words, the mapping between voltage ranges and logical values in Disco's static discipline is as summarized in Figure 9.11.

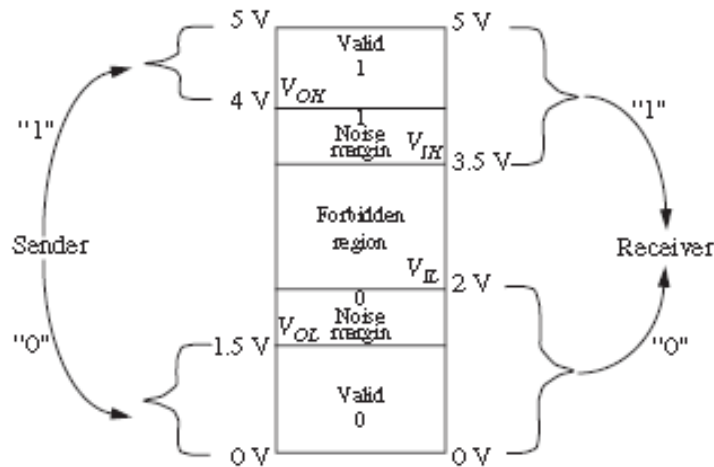


FIGURE 9.11 The mapping between voltage levels and logic values in the static discipline used by Disco Systems.

Yehaa's sales team wishes to sell their adders to Disco at a lower cost than those from Yikes, but first, Yehaa must determine whether their adders can safely replace the adders from Yikes. The sales team asks their development engineers to determine whether Yehaa's adders satisfy the static discipline under which Disco's system operates.

# Voltage Levels and the static discipline (continued)

The development team first looks at the output level for a logical 1 required by Disco's static discipline. The static discipline used by Disco requires devices to produce voltages between 4 V ( $V_{OH}$ ) and 5 V for a logical 1. As illustrated in Figure 9.12, Yehaa's devices produce a voltage level of 4.5 V for a logical 1, which falls within the required range, and so they satisfy the  $V_{OH}$  requirement.

Next, they look at the output voltage level for a logical 0. As depicted in Figure 9.12, Yehaa's devices produce a voltage level of 0.5 V for a logical 0, which falls within the 0 V to 1.5 V ( $V_{OL}$ ) range required for logical 0's by Disco's static discipline. Thus, Yehaa's devices satisfy the  $V_{OL}$  requirement.

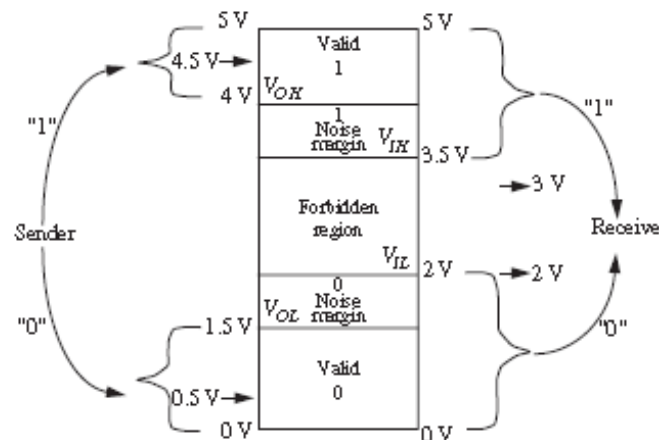


FIGURE 9.12 Comparing the voltages produced by Yehaa against the voltage levels required by the static discipline used by Disco Systems.

# Voltage Levels and the static discipline (continued)

The engineers now turn their attention to the input voltage levels required by Disco's static discipline. Yehaa's devices are able to interpret voltages as high as 2 V as a logical 0, so they can interpret any voltage between 0 V and 2 V ( $V_{IL}$ ) as a logical 0, just as required by Disco's static discipline. Thus, the Yehaa devices satisfy the  $V_{IL}$  requirement.

Similarly, Yehaa's devices are able to interpret voltages between 3.5 V ( $V_{IH}$ ) and 5 V as a logical 1, which again satisfies the  $V_{IH}$  requirement of Disco's static discipline. The fact that the Yehaa devices interpret certain voltages in Disco's forbidden region (specifically, those between 3 V and 3.5 V) as a logical 1 is irrelevant since devices are allowed arbitrary behavior for values in the forbidden region.

Thus, the development engineers are able to tell their sales team that Yehaa's adders satisfy Disco's static discipline and so they can be used as replacements for Disco's existing adders.

# Voltage Levels and the static discipline (continued)

## Example 9.2 Violating a static discipline

Yikes discovers that Disco is considering switching to Yehaa adders because Yehaa's devices are cheaper than Yikes devices. The Yikes sales team goes over their own product list and notices that they do carry a new adder that they can sell to Disco at an even lower cost than the Yehaa adders. Overjoyed, the sales team asks their own development engineers to check whether these new adders satisfy Disco's static discipline.

The new adders of Yikes have the following properties: For a logical 0, the new adders produce a voltage level of 1.7 V at their outputs. Similarly, when outputting a logical 1, their adders produce the voltage level of 4.5 V. The new Yikes adders interpret all signals between 0 V and 1.5 V at their inputs as a logical 0, and all signals between 4 V and 5 V as a logical 1. Their behavior for input signals in the 1.5 V to 4 V range is undefined.

Furthermore, recall that Disco's systems operate under a static discipline with the following voltage thresholds:  $V_{IL} = 2$  V,  $V_{IH} = 3.5$  V,  $V_{OL} = 1.5$  V, and  $V_{OH} = 4$  V.

The Yikes development team first looks at the output voltage levels required by Disco's static discipline. They observe that the 4.5-V output produced by their new adders falls within Disco's legitimate range for a logical 1 (between  $V_{OH} = 4$  V and 5 V), thus satisfying the  $V_{OH}$  requirement.

# Voltage Levels and the static discipline (continued)

Next, they turn their attention to the output voltage level required for a logical 0. To their disappointment, they discover that the 1.7-V output produced by their new adders for a logical 0 output is greater than the maximum value of  $V_{OL} = 1.5 \text{ V}$  allowed by Disco. Thus their new adders violate the  $V_{OL}$  requirement. At this point, the Yikes development team reluctantly concludes that their new adders cannot be sold to Disco.

As an exercise in futility, the development engineers further investigate the input voltage levels. Disco's static discipline requires that devices interpret any voltage between 0 V and 2 V ( $V_{IL}$ ) as a logical 0. The new adders from Yikes fail this test because they are unable to interpret signals above 1.5 V as a logical 0.

Next, the engineers investigate the input high voltage level, but quickly discover that the situation is even worse. Disco's systems require that all devices interpret voltages in the range 3.5 V ( $V_{IH}$ ) to 5 V as a logical 1. Unfortunately, their new adders can guarantee to interpret voltages only in the range 4 V to 5 V as a logical 1. Their behavior for input voltages in the range 3.5 V to 4 V is undefined. Since Disco's devices can legitimately produce voltages in this range for a logical 1, the new Yikes adders cannot co-exist with the existing Disco devices.

## Example 9.3 Noise margins

Recall that the hardware systems in one of Disco's product lines operate under a static discipline with the following voltage thresholds:

$$V_{IL} = 2 \text{ V}, V_{IH} = 3.5 \text{ V}, V_{OL} = 1.5 \text{ V}, \text{ and } V_{OH} = 4 \text{ V}.$$

Compute the noise margins.

From Equation 9.5, the noise margin for a logical 0 is given by

$$NM_0 = V_{IL} - V_{OL} = 2 \text{ V} - 1.5 \text{ V} = 0.5 \text{ V}.$$

Similarly, from Equation 9.6, the noise margin for a logical 1 is given by

$$NM_1 = V_{OH} - V_{IH} = 4 \text{ V} - 3.5 \text{ V} = 0.5 \text{ V}.$$

# Voltage Levels and the static discipline (continued)

## Example 9.4 A static discipline with improved noise margins

Disco Systems Inc. has been having intermittent faults in its systems. Their system architects figure out that because the static discipline they have adopted does not provide a sufficient noise margin, their systems are susceptible to noise. To improve the noise immunity of their systems, they decide to upgrade their systems to a new static discipline in which the output high voltage threshold is increased by 0.5 V, and the output low voltage threshold is decreased by 0.5 V. Both the input voltage thresholds remain unchanged. In other words, the improved static discipline has the following voltage thresholds:

$$V_{IL} = 2 \text{ V}, V_{IH} = 3.5 \text{ V}, V_{OL} = 1 \text{ V}, \text{ and } V_{OH} = 4.5 \text{ V}.$$

This choice affords their system a symmetric noise margin of 1 V. In other words, the noise margins for a logical 0 and a logical 1 are equal and are given by

$$NM_0 = 2 \text{ V} - 1 \text{ V} = 1 \text{ V}$$

and

$$NM_1 = 4.5 \text{ V} - 3.5 \text{ V} = 1 \text{ V}$$

# Voltage Levels and the static discipline (continued)

On hearing the upgrade announcement from Disco, the sales team of Yehaa claims that the adders they have sold Disco can be used under the upgraded static discipline. Let us determine whether this claim is true.

Recall that Yehaa's adders behave as follows: For a logical 0, Yehaa's adders produce a voltage level of 0.5 V at their outputs. Similarly, when outputting a logical 1, their adders produce the voltage level of 4.5 V. Yehaa adders are able to interpret all signals between 0 V and 2 V at their inputs as a logical 0, and all signals between 3 V and 5 V as a logical 1.

To operate under Disco's upgraded static discipline, we know that the adders must operate correctly with the tighter bounds on the output thresholds:

- ▶ When outputting a logical 1, the voltage their outputs produce must be at least  $V_{OH} = 4.5$  V. Since the Yehaa adders produce a 4.5 V output for a logical 1, they barely satisfy this condition.
- ▶ When outputting a logical 0, the voltage their outputs produce must be no greater than  $V_{OL} = 1$  V. Since the Yehaa adders produce a 0.5-V output for a logical 0, they satisfy this condition easily.

Thus, we have shown that the claim made by the Yehaa sales team is true.



# Voltage Levels and the static discipline (continued)

## Example 9.5 Comparing noise margins

Which of the two static disciplines shown below offers better noise margins?

Static discipline A has the voltage thresholds given by:

$$V_{\text{IL}} = 1.5 \text{ V}, V_{\text{IH}} = 3.5 \text{ V}, V_{\text{OL}} = 1 \text{ V}, \text{ and } V_{\text{OH}} = 4 \text{ V}.$$

Static discipline B has the voltage thresholds given by:

$$V_{\text{IL}} = 1.5 \text{ V}, V_{\text{IH}} = 3.5 \text{ V}, V_{\text{OL}} = 0.5 \text{ V}, \text{ and } V_{\text{OH}} = 4.5 \text{ V}.$$

For static discipline A:

$$\text{NM}_0 = 1.5 \text{ V} - 1 \text{ V} = 0.5 \text{ V}$$

and

$$\text{NM}_1 = 4 \text{ V} - 3.5 \text{ V} = 0.5 \text{ V}.$$

For static discipline B:

$$\text{NM}_0 = 1.5 \text{ V} - 0.5 \text{ V} = 1 \text{ V}$$

and

$$\text{NM}_1 = 4.5 \text{ V} - 3.5 \text{ V} = 1 \text{ V}.$$

Thus, the voltage thresholds of static discipline B offer a better noise margin.

# Boolean Logic

The binary representation has a natural correspondence to logic, and therefore digital circuits are commonly used to implement logic procedures. For example, consider the logical “if” statement:

*If X is TRUE AND Y is TRUE then Z is TRUE else Z is FALSE.*

We can represent this statement using a boolean equation as:

$$Z = X \text{ AND } Y.$$

In the previous equation,  $Z$  is true only when both  $X$  and  $Y$  are *TRUE* and *FALSE* otherwise. For brevity we often represent the *AND* function using the “.” symbol as:

$$Z = X \cdot Y.$$

Just as we represent the algebraic expression  $x \times y$  as  $xy$ , we often drop the *AND* symbol and write:

$$Z = X \cdot Y = XY.$$

The boolean equation for the statement:

*If (A is TRUE) OR (B is NOT TRUE) then (C is TRUE) else (C is FALSE)*

is

$$C = A + \bar{B}.$$

# Boolean Logic<sub>(continued)</sub>

The logic operators that we have seen thus far are summarized in Table 9.2.

OPERATOR	SYMBOL
AND	$\cdot$
OR	$+$
NOT	$\sim$

TABLE 9.2 Some logic operations and their symbols.

*Truth table* We often find it convenient to use a *truth table* representation of boolean functions. A truth table enumerates all possible input value combinations and the corresponding output values.

For example, the truth table representation for  $Z = X \cdot Y$  is shown in Table 9.3, that for  $Z = X + Y$  is shown in Table 9.4, that for  $Z = \bar{X}$  is shown in Table 9.5, and that for  $C = A + B$  is shown in Table 9.6.

TABLE 9.3 Truth table for  $Z = X \cdot Y$ .

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

TABLE 9.4 Truth table for  $Z = X + Y$ .

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

TABLE 9.5 Truth table for  $Z = \bar{X}$ .

X	Z
0	1
1	0

TABLE 9.6 Truth table for  $C = A + B$ .

A	B	C
0	0	1
0	1	0
1	0	1
1	1	1

# Boolean Logic<sub>(continued)</sub>

## Example 9.6 Motion detector logic

Let us write the boolean expression for a motion detector that operates as follows: The circuit must produce a signal  $L$  to turn on a set of lights when the signal  $M$  from a motion sensor is high, provided it is not daytime. Assume that a light sensor produces a signal  $D$  that is high during daytime.

Notice that  $L$  is nominally low. It must become high when  $M$  is high and  $D$  is low. Therefore, we can write

$$L = M\bar{D}.$$

# Boolean Logic<sub>(continued)</sub>

## Example 9.7 Truth table

TABLE 9.7 Truth Table for  $\overline{AB} + C + D$ .

A	B	C	D	OUTPUT
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

The truth table for the following logic expression is shown in Table 9.7.

$$\text{Output} = \overline{AB} + C + D.$$



# Combinational Gates

Yet another representation of boolean functions makes use of the *combinational gate* abstraction.

The digital gate notation for the boolean equation  $Z = X \text{ AND } Y$  is shown in Figure 9.13.

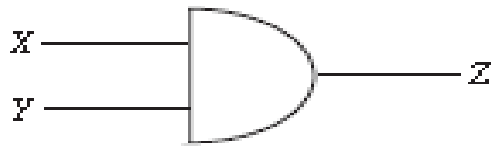


FIGURE 9.13 The *AND* gate

*Combinational gate abstraction* A combinational gate is an abstract representation of a circuit that satisfies two properties:

1. Its outputs are a function of its inputs alone.
2. It satisfies the static discipline.

# Combinational Gates<sub>(continued)</sub>

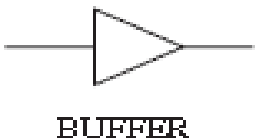
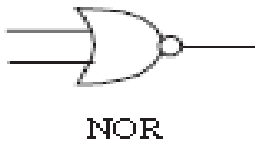
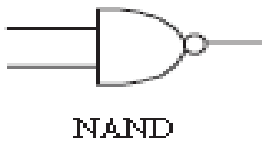
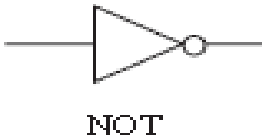
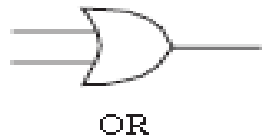


Figure 9.14 shows several useful gate symbols.

A truth table illustrating several of these functions is shown in Table 9.8. Each output column in the truth table corresponds to the given boolean function.

TABLE 9.8 Truth table for several two-input functions.

INPUTS		AND	OR	NAND	NOR
B	C	$B \cdot C$	$B + C$	$\overline{B \cdot C}$	$\overline{B + C}$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	1	0	0

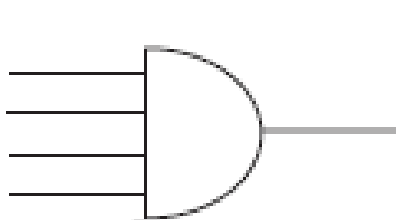
FIGURE 9.14 Gate symbols.

# Combinational Gates<sub>(continued)</sub>

Gates can have multiple inputs. For example, we can have a four-input *AND* gate that implements the function  $E = A \cdot B \cdot C \cdot D$  as shown in Figure 9.15.

As shown in Figure 9.16, we can combine digital gates using wires to implement digital circuits, thereby creating more complicated boolean functions.

Figure 9.17 shows a graphical view of the inputs and output for the digital circuit in Figure 9.16. Notice that the output continues to be valid even when the input signal is noisy.



Four-Input AND

FIGURE 9.15 A four-input *AND* gate.

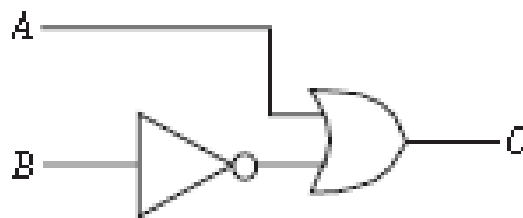


FIGURE 9.16 The gate-level digital circuit for  $C = A + \bar{B}$

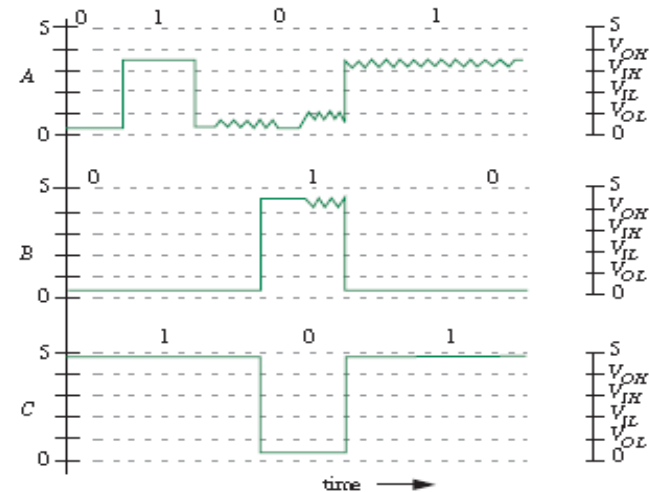


FIGURE 9.17 A noisy signal input to a digital circuit.



# Combinational Gates<sub>(continued)</sub>

## Example 9.8 Gate - level implementation

Let us implement the logic expression  $Output = \overline{AB + C + D}$  using gates. Notice that we have a choice in implementing this expression. We can use one two-input *AND* gate, a three input *OR* gate, and an inverter as shown in Figure 9.18. We can also replace the *OR* gate-inverter pair with a *NOR* gate. Alternatively, by rewriting the expression as

$$Output = \overline{((AB) + (C + D))}$$

we can implement the same circuit using an *AND* gate, an *OR* gate, and a *NOR* gate. We can also check each of the circuits against the truth table, and convince ourselves that they do work as desired.

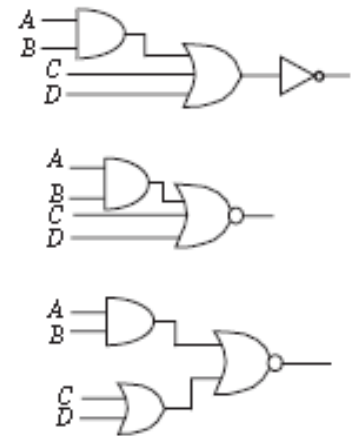


FIGURE 9.18  
Implementations of  
 $\overline{AB + C + D}$

# Combinational Gates<sub>(continued)</sub>

## Example 9.9 More gate - level implementations

Now, let us design a circuit for the expression:  $\overline{(A + B)CD}$ .

We can rewrite the expression as

$$\overline{(A + B)CD} = \overline{((A + B)(CD))}.$$

The corresponding gate-level implementation is shown in Figure 9.19.

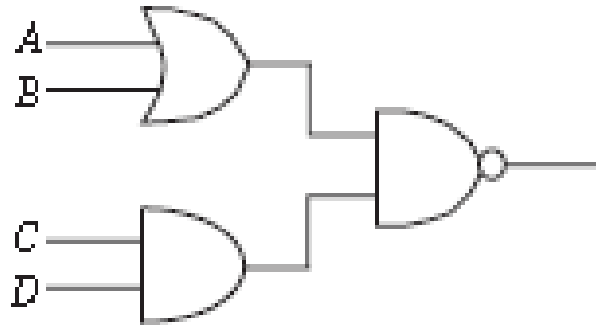


FIGURE 9.19 Implementation of  $\overline{(A + B)CD}$

# Combinational Gates<sub>(continued)</sub>

## Example 9.10 Yet another gate - level implementation

A circuit for the expression  $A + B\bar{B} + C$  is shown in Figure 9.20.

It requires three gates.

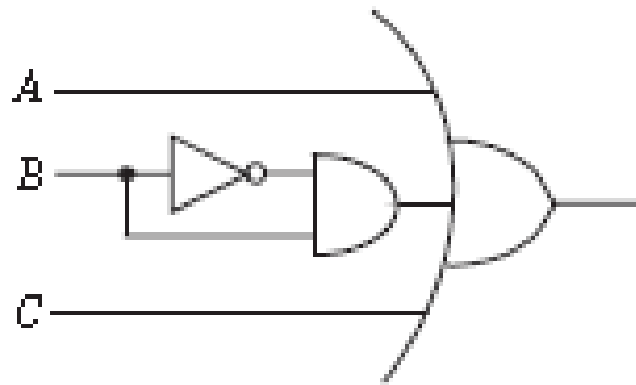


FIGURE 9.20 Implementation of  $A + B\bar{B} + C$ .

# Standard Sum-of-Products Representation

The previous two sections showed that logic expressions can be represented as truth tables or gate-level circuits.

In this section, we will show the equivalence of the representations by discussing how we can derive automatically a logic expression from a truth table.

*Sum-of-products* As the name implies, logic expressions in the sum-of-products form are represented using two levels of operations as a set of product (AND) terms, each comprising one or more variables in their true forms (for example,  $A$ ) or complement forms (e.g.,  $\overline{A}$ ), combined using the OR function.

For example, the logic expression  $AD + A\overline{B}C + \overline{A}B\overline{C}$  is in a sum-of-products representation containing the sum of three product terms. The first term contains two variables, while the latter two terms contain three variables each. The expression  $AB + C + \overline{D} + \overline{B}$  is also in a sum-of-products representation.

The expression  $\overline{AB + C}$ , however, is not in a sum of products representation, and neither is the expression  $(A + B)(B + \overline{C})$ .

# Standard Sum-of-Products Representation (continued)

We can write a sum-of-products expression from a truth table representation by first writing a product term for each row in the truth table with a 1 in its output column, and then summing these product terms.

Each product term comprises an *AND* function of all the input variables. A variable will appear in its true or complement form in a product term corresponding to a given row in the truth table depending on whether it appears as a 1 or a 0 in that row.

Thus, for example, a logic expression for the truth table in Table 9.4 is

$$Z = \overline{X}Y + X\overline{Y} + XY. \quad (9.7)$$

By construction, this expression is in a sum-of-products form. It has three product terms corresponding to the three 1's in the output column of the truth table.

## Example 9.11 Logic expression from a truth table

Write a logic expression corresponding to the truth table in Table 9.7.

There are three 1's in the output column of the truth table in Table 9.7, and so we expect to see three product terms. The product term corresponding to the 1 in the first row is  $\overline{A} \overline{B} \overline{C} \overline{D}$ .

Similarly, the next two products terms  $\overline{A} B \overline{C} \overline{D}$ , and  $A \overline{B} \overline{C} \overline{D}$ .

These three terms are combined with the *OR* function to yield the logic expression corresponding to the truth table as

$$\text{Output} = \overline{A} \overline{B} \overline{C} \overline{D} + \overline{A} B \overline{C} \overline{D} + A \overline{B} \overline{C} \overline{D}. \quad (9.8)$$



# Simplifying Logic Expressions

We are often interested in simplifying logic expressions to minimize their implementation cost.

For example, although the expression  $A + B\bar{B} + C$  appears to require three gates, simplification of the logic expression will result in a single gate.

Notice that the expression  $B\bar{B}$  always results in the answer 0 (a variable and its complement can never be TRUE at the same time). Furthermore, observe that  $A+0$  is always  $A$ . From these observations, we can simplify the expression as

$$A + B\bar{B} + C = A + 0 + C = A + C.$$

The reader can also verify that the expressions  $A + B\bar{B} + C$  and  $A + C$  are equivalent by developing the corresponding truth tables as illustrated in Table 9.9.

TABLE 9.9 Truth table for comparing the two expressions  $A + B\bar{B} + C$  and  $A + C$ .

A	B	C	$A + B\bar{B} + C$	$A + C$
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

# Simplifying Logic Expressions (continued)

The following primitive rules come in handy for simplifying logic expressions:

$$A \cdot \overline{A} = 0 \quad (9.9)$$

$$A \cdot A = A \quad (9.10)$$

$$A \cdot 0 = 0 \quad (9.11)$$

$$A \cdot 1 = A \quad (9.12)$$

$$A + \overline{A} = 1 \quad (9.13)$$

$$A + A = A \quad (9.14)$$

$$A + 0 = A \quad (9.15)$$

$$A + 1 = 1 \quad (9.16)$$

$$A + \overline{A}B = A + B \quad (9.17)$$

$$A(B + C) = AB + AC \quad (9.18)$$

$$AB = BA \quad (9.19)$$

$$A + B = B + A \quad (9.20)$$

$$(AB)C = A(BC) \quad (9.21)$$

$$(A + B) + C = A + (B + C) \quad (9.22)$$



# Simplifying Logic Expressions<sub>(continued)</sub>

You can verify these rules by comparing their truth tables. For example, the truth table comparing  $A + AB$  and  $A + B$  is shown in Table 9.10.

The following are another set of useful equalities called De Morgan's laws:

$$\overline{A \cdot B} = \overline{A} + \overline{B} \quad (9.23)$$

$$\overline{A + B} = \overline{A} \cdot \overline{B}. \quad (9.24)$$

TABLE 9.10 Truth table for comparing the two expressions  $A + \overline{A}B$  and  $A + B$ . Notice that for convenience we have added an extra column for the intermediate expression  $\overline{A}B$ .

A	B	$\overline{A}B$	$A + \overline{A}B$	$A + B$
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

# Simplifying Logic Expressions<sub>(continued)</sub>

De Morgan's laws can also be verified by developing truth tables as illustrated in Table 9.11.

Notice that the columns for  $\overline{A \cdot B}$  and  $\overline{A} + \overline{B}$  are identical. Similarly, observe that the columns for  $\overline{A + B}$  and  $\overline{A} \cdot \overline{B}$  are identical, thereby verifying De Morgan's laws.

TABLE 9.11 Truth table for verifying De Morgan's laws.

$A$	$B$	$\overline{A}$	$\overline{B}$	$A \cdot B$	$A + B$	$\overline{A \cdot B}$	$\overline{A + B}$	$\overline{A} \cdot \overline{B}$	$\overline{A} + \overline{B}$
0	0	1	1	0	0	1	1	1	1
0	1	1	0	0	1	1	0	0	1
1	0	0	1	0	1	1	0	0	1
1	1	0	0	1	1	0	0	0	0

# Simplifying Logic Expressions (continued)

De Morgan's laws can be expressed in terms of the gate notation as depicted in Figure 9.21. Consequently, the symbols on the right-hand side of the figure are often used in place of the corresponding *NAND* gate or the *NOR* gate.

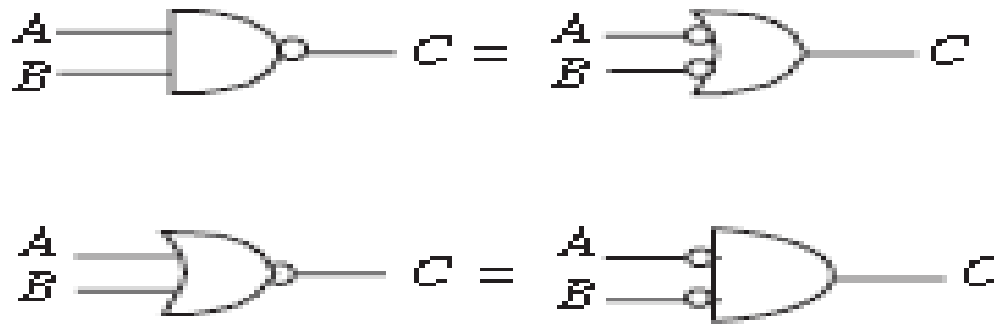


FIGURE 9.21 Gate equivalences implied by De Morgan's laws.

These rules can be used to simplify logic expressions to reduce the number of gates required to implement them. For example, a direct implementation of the logic expression  $A\bar{B}\bar{E} + BC + \bar{C}$  appears to take five 2-input gates as seen from Figure 9.22.

# Simplifying Logic Expressions (continued)

The implementation used in Figure 9.22 assumes that both TRUE and complement forms of each variable are available as inputs. In other words, for each variable  $X$ , we assume that both  $X$  and  $\bar{X}$  are available as inputs. Otherwise, we would need two additional inverters.

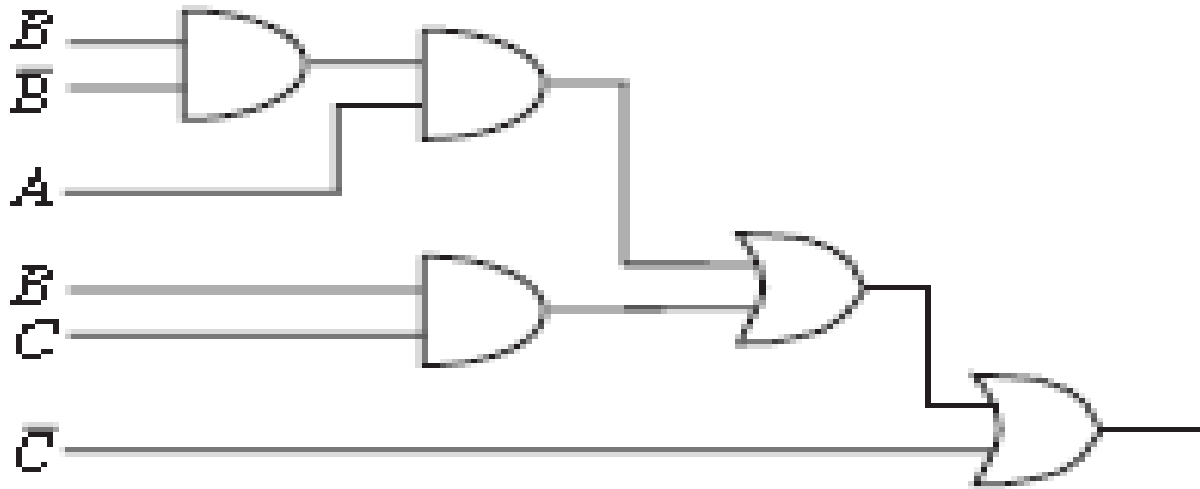


FIGURE 9.22 Direct implementation of  $AB\bar{B} + BC + \bar{C}$ .

# Simplifying Logic Expressions<sub>(continued)</sub>

To reduce the number of gates required for its implementation, we can simplify the expression  $AB\bar{E} + BC + \bar{C}$  as follows:

We first collect terms as shown below by applying the rule suggested by Equation 7.21:

$$AB\bar{E} + BC + \bar{C} = A(E\bar{E}) + BC + \bar{C}.$$

Then, we apply the simplification suggested by Equation 9.9 and obtain

$$A(E\bar{E}) + BC + \bar{C} = A0 + BC + \bar{C}.$$

Applying Equation 9.11 we get

$$A0 + BC + \bar{C} = 0 + BC + \bar{C}.$$

Grouping terms as suggested by Equation 9.22 we get

$$0 + BC + \bar{C} = (0 + BC) + \bar{C}.$$

Applying Equations 9.20 and 9.15 we get

$$(0 + BC) + \bar{C} = BC + \bar{C}.$$

Finally, applying Equation 9.17 after recognizing that both the OR operators are commutative (from Equations 9.19 and 9.20) we obtain the final simplified form

$$BC + \bar{C} = B + \bar{C}.$$

# Simplifying Logic Expressions (continued)

The implementation of  $B + \overline{C}$  takes just one gate and is shown in Figure 9.23. You might wish to work through some input values and verify that the circuits in Figures 9.22 and 9.23 are equivalent.

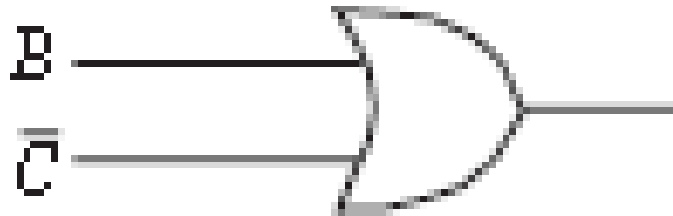


FIGURE 9.23 Implementation of  $B + \overline{C}$   
which results from simplifying  $AB\overline{B} + BC + \overline{C}$

The preceding rules can also be used to simplify logic expressions into a standard or canonic form. A standard form of representation makes it easy to compare the costs of competing implementations. One canonic form is the sum-of-products form.

# Simplifying Logic Expressions (continued)

Recall that logic expressions in this form are represented using two levels of operations as a set of product (*AND*) terms combined using the *OR* function. For example, the expression  $AB+C+D$  is in a sum-of-products representation. The expression  $\overline{AB}+C+D$ , however, is not. We can convert the latter expression into the sum-of-products form  $\overline{A}\overline{C}\overline{D} + \overline{B}\overline{C}\overline{D}$  using the equivalence rules as follows:

$$\overline{AB} + C + D = \overline{(AB) + (C + D)} \quad (9.25)$$

$$= \overline{(AB)} \overline{(C + D)} \quad (9.26)$$

$$= (\overline{A} + \overline{B})(\overline{C} \overline{D}) \quad (9.27)$$

$$= (\overline{A})(\overline{C} \overline{D}) + (\overline{B})(\overline{C} \overline{D}) \quad (9.28)$$

$$= \overline{A}\overline{C}\overline{D} + \overline{B}\overline{C}\overline{D} \quad (9.29)$$

We can also use the equalities to simplify expressions into their respective minimal forms. A commonly used form is called the *minimum sum-of-products form*. For example,  $A + A + \overline{A}C + D$  is a valid sum-of-products representation. Since  $A + A = A$ , and  $A + \overline{A}C = A + C$ , its minimum sum-of-products form is simply  $A + C + D$ .

It turns out that the expression in our previous example  $\overline{A}\overline{C}\overline{D} + \overline{B}\overline{C}\overline{D}$  is also the minimum sum-of-products representation.

# Simplifying Logic Expressions<sub>(continued)</sub>

## Example 9.12 Minimum sum - of - products form

Find the minimum sum-of-products representation for the boolean function  $A + \overline{A}C + B$ .

We first write the sum-of-products representation:

$$\begin{aligned}A + \overline{A}C + B &= A + (\overline{A} + \overline{C}) + B \\&= A + (A + \overline{C}) + B \\&= A + A + \overline{C} + B \\&= A + \overline{C} + B.\end{aligned}$$

Here,  $A + A + \overline{C} + B$  is in a sum-of-products form. The minimum sum-of-products form, however, is  $A + \overline{C} + B$ .



# Simplifying Logic Expressions (continued)

## Example 9.13 Simplifying a logic expression

Find the minimum sum-of-products representation for the boolean expression in Equation 9.7, namely

$$Z = \bar{X}Y + X\bar{Y} + XY$$

The following sequence of simplifications show that this expression for  $Z$  is equivalent to  $X + Y$ :

$$\begin{aligned} Z &= \bar{X}Y + X\bar{Y} + XY \\ &= \bar{X}Y + X(\bar{Y} + Y) \\ &= \bar{X}Y + X \cdot 1 \\ &= \bar{X}Y + X \\ &= Y + X \end{aligned}$$

# Simplifying Logic Expressions (continued)

## Example 9.14 Implementation using nors

It turns out that certain types of gates take up less room or are easier to build in certain technologies than other types of gates. We can make use of the equivalence rules to convert a circuit from one form to another.

Let us derive an implementation of the *AND* function based on two-input *NOR* gates. In other words, we wish to transform the expression  $Z = A \cdot B$  into one that uses only *NOR* operators. The following steps show how we can transform the *AND* expression into one that uses three *NOR* operations:

$$A \cdot B = (A + A) \cdot (B + B) \quad (9.30)$$

$$= \overline{\overline{(A + A) \cdot (B + B)}} \quad (9.31)$$

$$= \overline{(A + A) + (B + B)}. \quad (9.32)$$



# Number Systems

## Why Binary?

- Computers are made of a series of switches
- Each switch has two states: ON or OFF
- Each state can be represented by a number – 1 for “ON” and 0 for “OFF”

# Number Systems<sub>(continued)</sub>

## Understanding Placeholders

- ▶ Each numbering system has placeholders
- ▶ The possible values of each place holder depends on the maximum number of single-digit numbers available for that numbering system
- ▶ In the Base-10 world (aka Decimal), there are ten possible digits that each placeholder can take (0-9)

# Number Systems<sub>(continued)</sub>

## Base-10 Placeholders

Number:	4	3	6
Placeholder Name:	"Hundreds"	"Tens"	"Ones"
Value:	$100*4$	$10*3$	$1*6$
Exponential Expression:	$10^2*4$	$10^1*3$	$10^0*6$

# Number Systems<sub>(continued)</sub>

## Base-2 Placeholders

Number:	1	0	1
Placeholder Name:	"Fours"	"Twos"	"Ones"
$n_{10}$ Value:	$4*1$	$2*0$	$1*1$
$n_{10}$ Exponential Expression:	$2^{2*1}$	$2^{1*0}$	$2^{0*1}$

# Number Systems<sub>(continued)</sub>

## Converting Base-2 to Base-10

1 0 0 1 1<sub>2</sub>

ON/OFF       $\overline{\text{O}}$        $\overline{\text{O}}$        $\overline{\text{O}}$        $\overline{\text{O}}$

Exponent:      2<sup>4</sup>      ~~2<sup>3</sup>~~      ~~2<sup>2</sup>~~      2<sup>1</sup>      2<sup>0</sup>

Calculation:      16 + 0 + 0 + 2 + 1 =

**19<sub>10</sub>**

# Number Systems<sub>(continued)</sub>

## Converting Base-10 to Base-2

STEP ONE: Find the largest exponent of two that is less than or equal to the Base-10 number:

$$21_{10} =$$

**16**

**$2^4$**

**$2^3$**

**$2^2$**

**$2^1$**

**$2^0$**



# Number Systems<sub>(continued)</sub>

## Converting Base-10 to Base-2

STEP TWO: Calculate the value of each exponent and place that value above it:

$$21_{10} =$$

<b>16</b>	<b>8</b>	<b>4</b>	<b>2</b>	<b>1</b>
<b><math>2^4</math></b>	<b><math>2^3</math></b>	<b><math>2^2</math></b>	<b><math>2^1</math></b>	<b><math>2^0</math></b>

# Number Systems<sub>(continued)</sub>

## Converting Base-10 to Base-2

STEP THREE: Place a "1" above the left-most placeholder:

$$\begin{array}{rcccccc} 21_{10} = & 1 & & & & & \\ & 16 & 8 & 4 & 2 & 1 & \\ & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & \end{array}$$

# Number Systems<sub>(continued)</sub>

## Converting Base-10 to Base-2

STEP FOUR: Add left-most value to the next place-holder. Put a "0" above is the sum is greater than the number being converted, otherwise put a "1" above:

**$21_{10} =$**

**1 0**

**16**

~~**8**~~

**4**

**2**

**1**

**$2^4$**

~~**$2^3$**~~

**$2^2$**

**$2^1$**

**$2^0$**

# Number Systems<sub>(continued)</sub>

## Converting Base-10 to Base-2

STEP FIVE: Add subsequent placeholders. If the sum is greater than the number being converted, put a "0" above and disregard it. Otherwise, put a "1" above and including it in your running total.

$$\begin{array}{cccccc} \mathbf{21}_{10} = & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ & \mathbf{16} & \mathbf{8} & \mathbf{4} & \mathbf{2} & \mathbf{1} \\ & \mathbf{2^4} & \mathbf{2^3} & \mathbf{2^2} & \mathbf{2^1} & \mathbf{2^0} \end{array}$$

# Number Systems<sub>(continued)</sub>

## Converting Base-10 to Base-2

STEP SIX: The resulting 1s and 0s in the top row form the binary equivalent of the Base-10 number with which we started:

$$21_{10} = 10101_2$$

# Number Systems<sub>(continued)</sub>

## Introducing Octal

- ▶ Computer scientists are often looking for shortcuts to do things
- ▶ One of the ways in which we can represent binary numbers is to use their octal equivalents instead
- ▶ This is especially helpful when we have to do fairly complicated tasks using numbers

# Number Systems<sub>(continued)</sub>

## Introducing Octal

- ▶ The octal numbering system includes eight base digits (0-7)
- ▶ After 7, the next placeholder to the right begins with a “1”
- ▶ 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13 ...

# Number Systems<sub>(continued)</sub>

## Octal Placeholders

Number:	2	4	1
Placeholder Name:	"Sixty-Fours"	"Eights"	"Ones"
Value:	$64 * 2$	$8 * 4$	$1 * 1$
Exponential Expression:	$8^2 * 2$	$8^1 * 4$	$8^0 * 1$



# Number Systems<sub>(continued)</sub>

## Converting Base-2 to Base-8

**10001100101001<sub>2</sub>**

STEP ONE: Take the binary number and from right to left, group all placeholders in triplets. Add leading zeros, if necessary:

**010    001    100    101    001**

# Number Systems<sub>(continued)</sub>

## Converting Base-2 to Base-8

$$10001100101001_2 = 21451_8$$

STEP TWO: Convert each triplet to its single-digit octal equivalent. (***HINT:*** For each triplet, the octal conversion is the same as converting to a decimal number):

<b>010</b>	<b>001</b>	<b>100</b>	<b>101</b>	<b>001</b>
<b>2</b>	<b>1</b>	<b>4</b>	<b>5</b>	<b>1</b>

# Number Systems<sub>(continued)</sub>

## Converting Base-8 to Base-2

$$43520_8 = 100011101010000_2$$

STEP ONE: Take each octal digit and convert each digit to a binary triplet. Keep leading zeros:

<b>4</b>	<b>3</b>	<b>5</b>	<b>2</b>	<b>0</b>
<b>100</b>	<b>011</b>	<b>101</b>	<b>010</b>	<b>000</b>

# Number Systems<sub>(continued)</sub>

## Converting Base-8 to Base-10

**2374<sub>8</sub>**

STEP ONE: Multiply each octal digit by the exponential expression that represents its placeholder:

$$2 * 8^3 = 1024$$

$$3 * 8^2 = 192$$

$$7 * 8^1 = 56$$

$$4 * 8^0 = 4$$

# Number Systems<sub>(continued)</sub>

## Converting Base-8 to Base-10

$$2374_8 = 1276_{10}$$

STEP TWO: Add the products together.  
The sum represents the Base-10 equivalent:

$$1024 + 192 + 56 + 4 = 1276$$

# Number Systems<sub>(continued)</sub>

## Converting Base-10 to Base-8

**$4832_{10}$**

STEP ONE: Divide the Base-10 number by eight. DO NOT DIVIDE PAST THE DECIMAL POINT – INSTEAD INCLUDE A REMAINDER:  **$4832 / 8 = 604 \text{ r}0$**

# Number Systems<sub>(continued)</sub>

## Converting Base-10 to Base-8

**4832<sub>10</sub>**

STEP TWO: Divide the quotient of the previous expression by eight. Repeat the process until you have a quotient of 0:

$$4832 / 8 = 604 \quad \text{r0}$$

$$604 / 8 = 75 \quad \text{r4}$$

$$75 / 8 = 9 \quad \text{r3}$$

$$9 / 8 = 1 \quad \text{r1}$$

$$1 / 8 = 0 \quad \text{r1}$$


# Number Systems<sub>(continued)</sub>

## Converting Base-10 to Base-8

$$4832_{10} = 11340_8$$

STEP THREE: The octal equivalent can be found by looking at the remainders from the bottom up:

<b>4832</b>	<b>/ 8 =</b>	<b>604</b>	<b>r0</b>
<b>604</b>	<b>/ 8 =</b>	<b>75</b>	<b>r4</b>
<b>75</b>	<b>/ 8 =</b>	<b>9</b>	<b>r3</b>
<b>9</b>	<b>/ 8 =</b>	<b>1</b>	<b>r1</b>
<b>1</b>	<b>/ 8 =</b>	<b>0</b>	<b>r1</b>





# Number Systems<sub>(continued)</sub>

## Hexadecimal Numbering

- ▶ Sometimes, it is necessary to use a numbering system that has more than ten base digits
- ▶ One such numbering system, hexadecimal, is useful on the Web
- ▶ Hexadecimal number, a Base-16 numbering system, is used in specifying web colors

# Number Systems<sub>(continued)</sub>

## Hexadecimal Numbering

- There are new symbols for the Base-16 equivalents of the Base-10 numbers 10, 11, 12, 14 and 15. Examine:

<b>DEC</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>
<b>HEX</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>
<b>DEC</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>
<b>HEX</b>	<b>8</b>	<b>9</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>

# Number Systems<sub>(continued)</sub>

## Converting Base-10 to Base-16

**$214_{10}$**

STEP ONE: Divide the Base-10 number by sixteen. Do not divide past the decimal point:

$$214 / 16 = 13 \quad r6$$

# Number Systems (continued)

## Converting Base-10 to Base-16

**$214_{10}$**

STEP TWO: Convert both the quotient and the remainder to their hexadecimal equivalents:

$$\begin{array}{ccccc} \mathbf{214} & / & \mathbf{16} & = & \mathbf{13} & \mathbf{r6} \\ \text{Hex Equivalents} & & & & \mathbf{D} & \mathbf{6} \end{array}$$

# Number Systems (continued)

## Converting Base-10 to Base-16

$$214_{10} = D6_{16}$$

STEP THREE: The quotient represents the first digit of the hexadecimal equivalent and the remainder represents the second digit:

$$214 / 16 = 13 \quad r6$$

Hex Equivalents      D      6

# Number Systems<sub>(continued)</sub>

## Converting Fractional Numbers

Fractional number:

$$(N)_{b_1} = a_{-1}b_2^{-1} + a_{-2}b_2^{-2} + \cdots + a_{-p}b_2^{-p}$$

$$b_2 \cdot (N)_{b_1} = a_{-1} + a_{-2}b_2^{-1} + \cdots + a_{-p}b_2^{-p+1}$$

Example: Convert  $(0.3125)_{10}$  to base 8

$$0.3125 \cdot 8 = 2.5000 \text{ hence } a_{-1} = 2$$

$$0.5000 \cdot 8 = 4.0000 \text{ hence } a_{-2} = 4$$

Thus,  $(0.3125)_{10} = (0.24)_8$

# Number Systems<sub>(continued)</sub>

## Converting Fractional Numbers

**Example:** Convert  $(432.354)_{10}$  to binary

$Q_i$	$r_i$	
216	$0 = a_0$	$0.354 \cdot 2 = 0.708$ hence $a_{-1} = 0$
108	$0 = a_1$	$0.708 \cdot 2 = 1.416$ hence $a_{-2} = 1$
54	$0 = a_2$	$0.416 \cdot 2 = 0.832$ hence $a_{-3} = 0$
27	$0 = a_3$	$0.832 \cdot 2 = 1.664$ hence $a_{-4} = 1$
13	$1 = a_4$	$0.664 \cdot 2 = 1.328$ hence $a_{-5} = 1$
6	$1 = a_5$	$0.328 \cdot 2 = 0.656$ hence $a_{-6} = 0$
3	$0 = a_6$	$a_{-7} = 1$
1	$1 = a_7$	etc.
	$1 = a_8$	

Thus,  $(432.354)_{10} = (110110000.0101101...)_{2}$

# Number Systems<sub>(continued)</sub>

## Converting Fractional Numbers

Example: Convert  $(123.4)_8$  to binary

$$(123.4)_8 = (001\ 010\ 011.100)_2$$

Example: Convert  $(1010110.0101)_2$  to octal

$$(1010110.0101)_2 = (001\ 010\ 110.010\ 100)_2 = (126.24)_8$$



# Number Systems<sub>(continued)</sub>

## How To Represent Signed Numbers

- ▶ Plus and minus signs used for decimal numbers: 25 (or +25), -16, etc.
- ▶ For computers, it is desirable to represent everything as *bits*.
- ▶ Three types of signed binary number representations:
  1. signed magnitude,
  2. 1's complement, and
  3. 2's complement

# Number Systems<sub>(continued)</sub>

## 1. signed magnitude

- In each case: **left-most bit indicates sign: positive (0) or negative (1).**

Consider 1. ***signed magnitude***:

$$\begin{array}{c} \text{Sign bit} \nearrow \text{0} \text{0001100}_2 = 12_{10} \\ \nwarrow \text{Magnitude} \end{array}$$

$$\begin{array}{c} \text{Sign bit} \nearrow \text{1} \text{0001100}_2 = -12_{10} \\ \nwarrow \text{Magnitude} \end{array}$$

# Number Systems<sub>(continued)</sub>

## 2. One's Complement Representation

- ▶ The one's complement of a binary number involves inverting all bits.
- *To find negative of 1's complement number take the 1's complement of whole number including the sign bit.*

$$\begin{array}{c} \text{Sign bit} \nearrow 0 \quad 00001100_2 = 12_{10} \\ \quad \quad \quad \nwarrow \text{Magnitude} \end{array}$$

$$\begin{array}{c} \text{Sign bit} \nearrow 1 \quad 1110011_2 = -12_{10} \\ \quad \quad \quad \nwarrow \text{1's complement} \end{array}$$

# Number Systems<sub>(continued)</sub>

## 3. Two's Complement Representation

- The two's complement of a binary number involves inverting all bits **and adding 1**.
- To find the negative of a signed number take the 2's complement of the positive number including the sign bit.

$$\begin{array}{c} \text{Sign bit} \quad \text{Magnitude} \\ \swarrow \quad \nwarrow \\ 00001100_2 = 12_{10} \end{array}$$

$$\begin{array}{c} \text{Sign bit} \quad \text{2's complement} \\ \swarrow \quad \nwarrow \\ 11110100_2 = -12_{10} \end{array}$$

# Number Systems<sub>(continued)</sub>

## Floating-Point Representation

- A real number (floating-point number) contains two parts: a mantissa, significand, or fraction and an exponent
- Fig. 9.24 depicts both the 4-byte (**single precision**) and 8-byte (**double precision**) forms of real numbers
- The exponent is stored as a biased exponent
  - an exponent of  $2^3$  is represented as a biased exponent of  $127+3$  or  $130$  ( $82H$ ) in the single-precision form or as  $1023+3$  or  $1026$  ( $402H$ ) in the double-precision form

# Number Systems<sub>(continued)</sub>

## Floating-Point Representation

► Table 9.12 shows the single-precision form of this number and others.

**Table 9.12 Single-precision real numbers**

Decimal	Binary	Normalized	Sign	Biased Exponent	Mantissa
+12	1100	$1.1 \times 2^3$	0	10000010	10000000 00000000 00000000
-12	1100	$1.1 \times 2^3$	1	10000010	10000000 00000000 00000000
+100	1100100	$1.1001 \times 2^6$	0	10000101	10010000 00000000 00000000
-1.75	1.11	$1.11 \times 2^0$	1	01111111	11000000 00000000 00000000
+0.25	0.01	$1.0 \times 2^{-2}$	0	01111101	00000000 00000000 00000000
+0.0	0	0	0	00000000	00000000 00000000 00000000

# Number Systems<sub>(continued)</sub>

## Floating-Point Representation

### Example:

- Convert the following number; 37.75 into single-precision floating point numbers format to fit in 32 bit register.
  - **Convert the number from decimal into binary**
    - 100101.11
  - **Normalize all digits including the fraction to determine the exponent.**
    - $1.0010111 \times 2^5$

0 0 0 0 0 0 1 0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0



sign

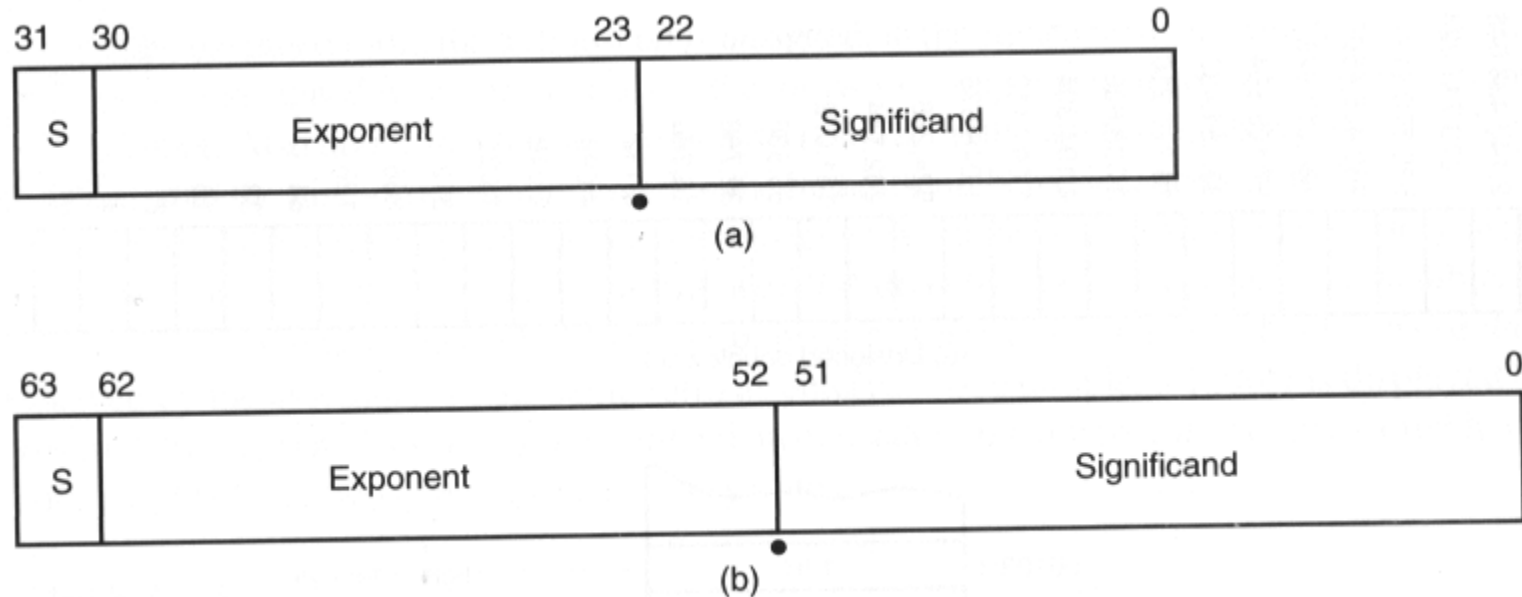


EXP



Significant

## Floating-Point Representation



**FIGURE 9.24** The floating-point numbers (a) single-precision using a bias of 7FH, and (b) double-precision using a bias of 3FFH