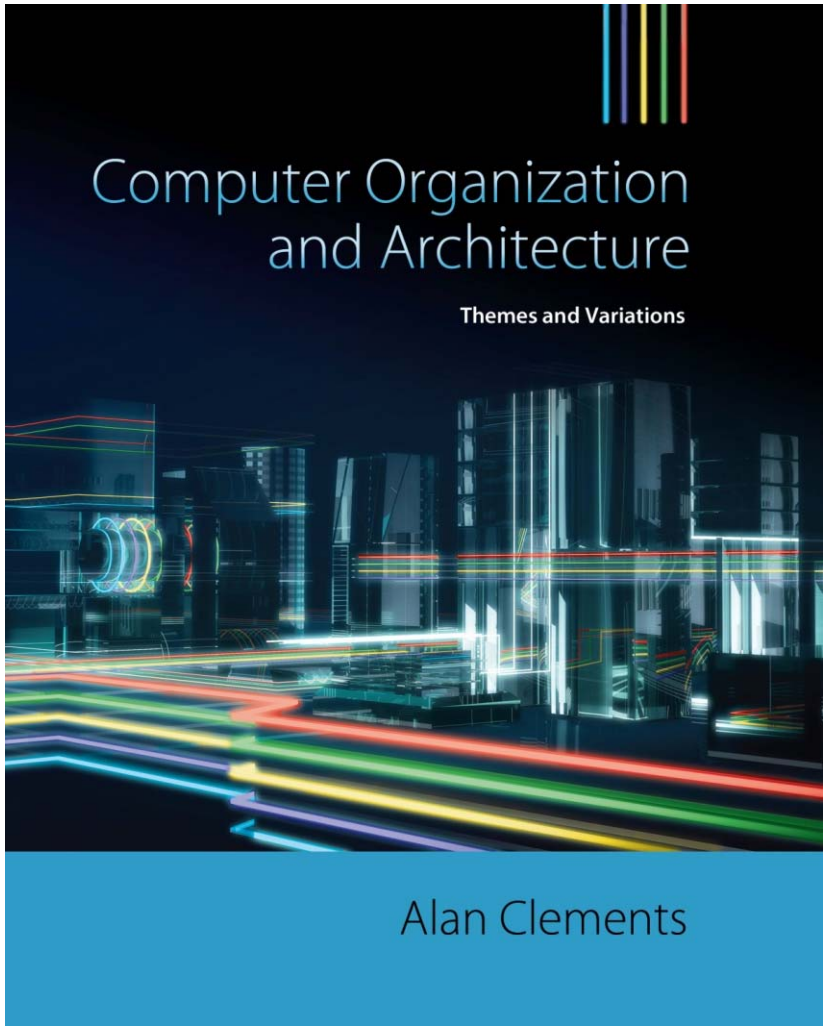
Part 4

Chapter 2

Computer Arithmetic and Digital Logic



CENGAGE Learning

Computer Logic

- Computers are constructed from two basic circuit elements — *gates* and *flip-flops*, known as *combinational* and *sequential* logic elements.

- A *combinational* logic element is a circuit whose output depends only on its current inputs,

  whereas

  A *sequential* logic element is a circuit whose output depends on its past history as well as its current inputs.

- A *sequential* element can *remember* its previous value (*memory element*).
- *Sequential* elements themselves can be made from simple combinational logic elements.
  - Hence, we can simply say that *computers can be constructed using just gates*

Logic Values

- A ***logic value*** can be either
  - the *logical 1* (also called the *true* or *high* state)
  - the *logical 0* (also called the *false* or *low* state)

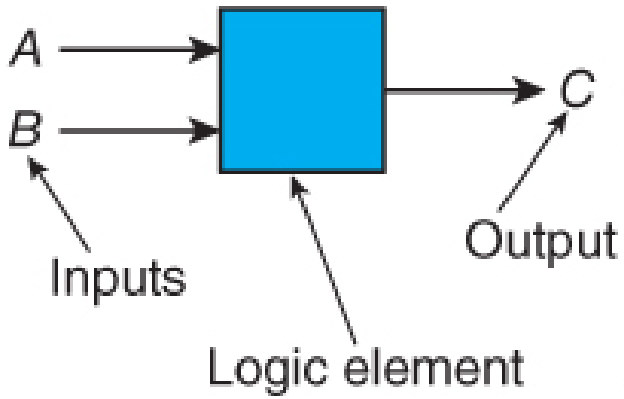
- Each logic state has an ***inverse*** or a ***complement*** that is the opposite of its current state.
  - The complement of a *true* or *1* state is a *false* or *0* state
  - The complement of a *false* or *0* state is a *true* or *1* state

Gates

- Figure 2.14 shows a black box of a gate with two *input* terminals, *A* and *B*, and a single *output* terminal *C*.
  - This gate takes the *two logic values* at its input terminals and
  - produces *a logic output value* that ***depends only on***
    - the *states of the inputs* and
    - the *nature of the logic element*.

FIGURE 2.13

The logic element



- Examples of gates include
  - **AND**, **OR**, **NOT** , **NAND**, **NOR** , **Exclusive OR** gates.

Gates

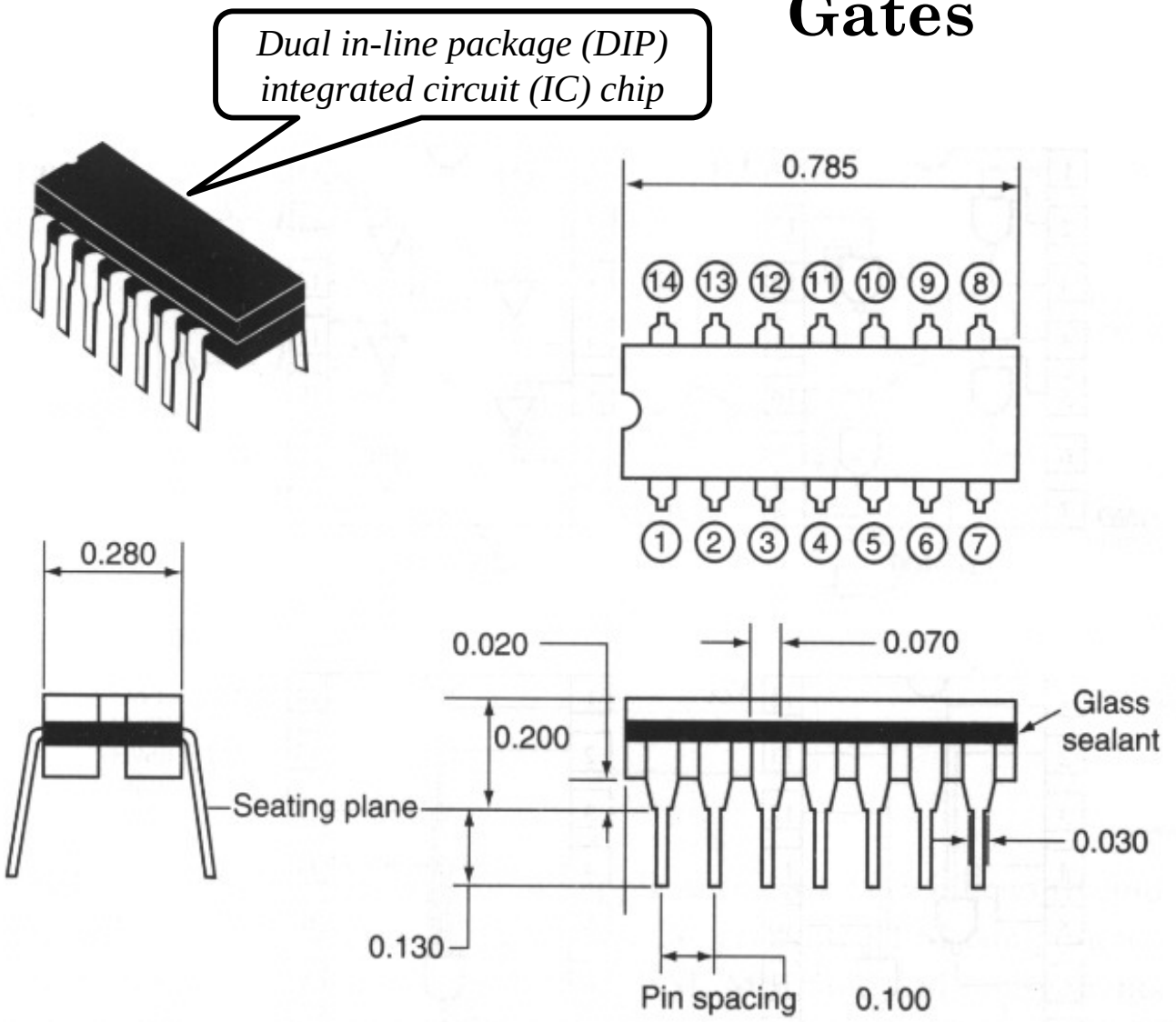
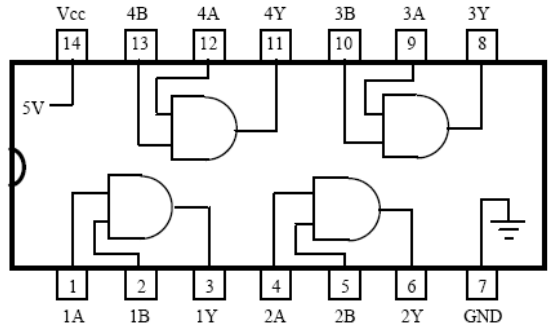
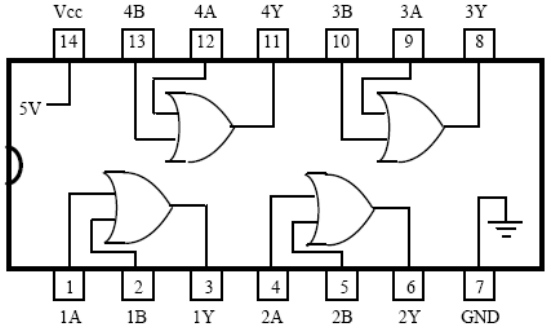


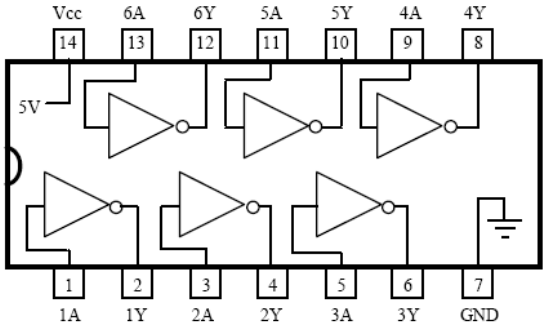
FIGURE 3.17
14-pin ceramic package (all linear dimensions are in inches).



7408: quad two input AND gates

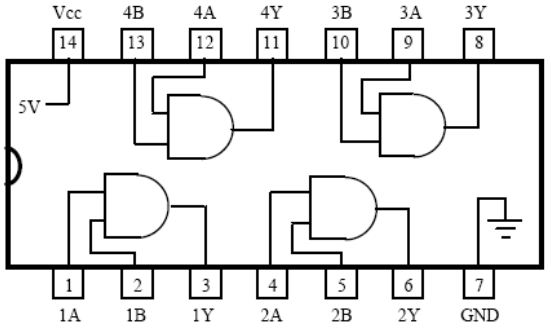


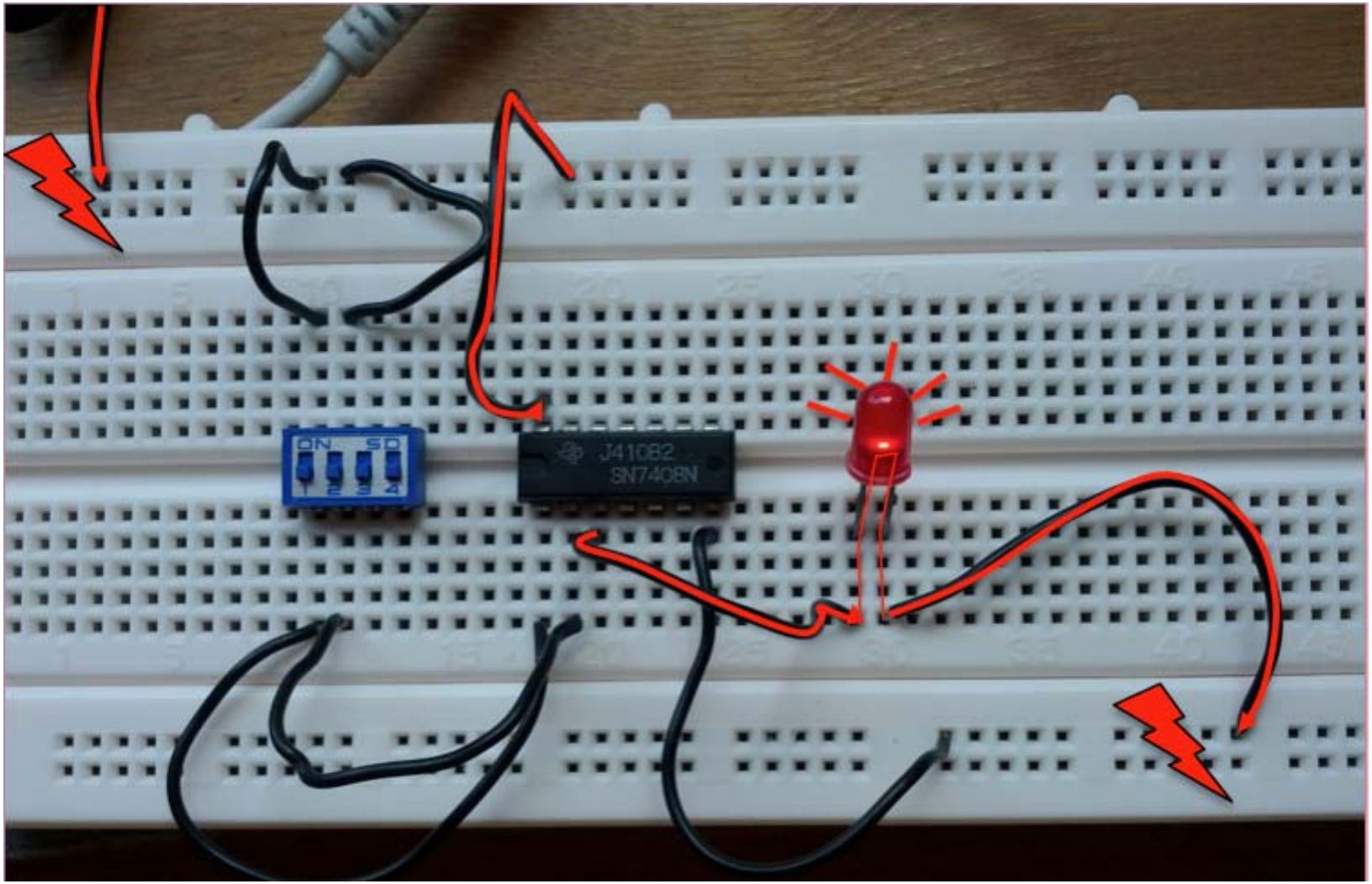
7432: quad two input OR gates



7404: hex inverter gates

Gates





The AND Gate

- The behavior of a gate is described by its *truth table* that *defines its output for each of the possible inputs*.
- Table 2.8a provides the truth table for the *two-input* **AND** gate.
  - If the two input are *A* and *B*, then the output C will be true (i.e., 1) if and only if both inputs *A* and *B* are true (i.e., 1) simultaneously.
- Table 2.8b gives the truth table for the *three input* **AND** gate.
  - If A, B, and C are the inputs, then the output D will be true (i.e., 1) if and only if all inputs are true (i.e., 1) simultaneously.
- The **AND** is represented by a "**.**"
  - the operation A **AND** B can be written as A **.** B

TABLE 2.8 Truth Table for the AND Gate

B	A	C = A·B
0	0	0
0	1	0
1	0	0
1	1	1

(a) Two-input AND gate

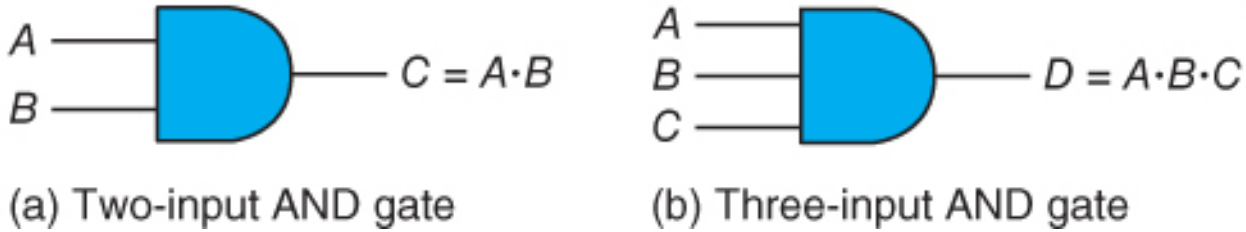
C	B	A	D = A·B·C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(b) Three-input AND gate

The AND Gate

- Figure 2.14 gives the symbols for 2-input and 3-input **AND** gates

FIGURE 2.14 The symbol for an AND gate



(a) Two-input AND gate (b) Three-input AND gate

The OR Gate

- The output of an **OR** gate is 1 if at least one of its inputs is 1.
- The only way to make the output of an **OR** gate go to a logical 0 is to set all its inputs to 0.
- The **OR** is represented by a "**+**"
  - the operation A **OR** B can be written as A + B

TABLE 2.9 Truth Table for the OR Gate

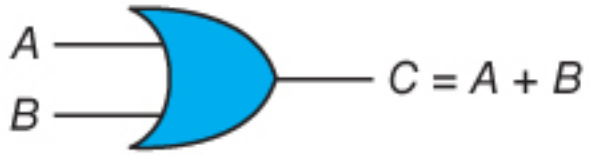
B	A	C = A + B
0	0	0
0	1	1
1	0	1
1	1	1

(a) Two-input OR gate

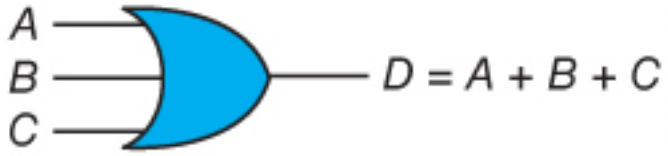
C	B	A	D = A + B + C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(b) Three-input OR gate

FIGURE 2.15 The symbol for an OR gate



(a) Two-input OR gate



(b) Three-input OR gate

The NOT gate or inverter

- The **NOT** is represented by
  - an "***overbar***", e.g., the operation **NOT** A is written as $\overline{A}$

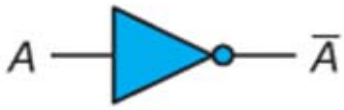
  or
  - a superscript c, e.g., the operation **NOT** A is written as A^c

  or
  - a tilde mark ~, e.g., the operation **NOT** A is written as ~ A

  or
  - a negation mark ¬, e.g., the operation **NOT** A is written as ¬ A

  or
  - an exclamation mark !, e.g., the operation **NOT** A is written as !A
- Note that, $\overline{\overline{A}} = A$

FIGURE 2.16 The symbol and truth table for an inverter



(a) Symbol for inverter

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

(b) Truth table of inverter

This bit is 1 not 0.
The book wrote it incorrectly as 0.

Comparing AND and OR Gates

TABLE 2.10 Truth Table for AND and OR Gates with Both Constant and Variable Inputs

AND		OR	
Constant	Variable	Constant	Variable
$0 \cdot 0 = 0$	$A \cdot 0 = 0$	$0 + 0 = 0$	$A + 0 = A$
$0 \cdot 1 = 0$	$A \cdot 1 = A$	$0 + 1 = 1$	$A + 1 = 1$
$1 \cdot 0 = 0$	$A \cdot \bar{A} = 0$	$1 + 0 = 1$	$A + \bar{A} = 1$
$1 \cdot 1 = 1$	$A \cdot A = A$	$1 + 1 = 1$	$A + A = A$

Derived Gates NOR, NAND, Exclusive OR

- **NOR**, **NAND** and **XOR** are gates that can be derived from basic gates.
  - a **NOR** gate is an **OR** followed by an **inverter**.
  - A **NAND** gate is an **AND** followed by and **inverter** and
  - An **XOR** gate is an **OR** gate whose output is true *only if an odd number of its input is true.*

This is B not C

TABLE 2.11 Truth Table for the NOR Gate, NAND Gate, and Exclusive OR Gates

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

(a) The NOR gate

A	B	$C = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

(b) The NAND gate

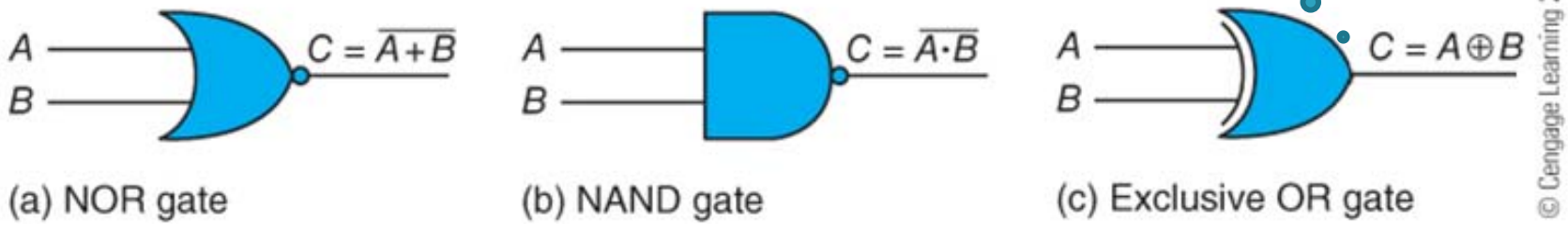
A	B	$C = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

(c) The XOR gate

- These gates (**NOR**, **NAND**, and **XOR**) are used extensively in digital circuits and have their own symbols.

There is no bubble here.
The book added it incorrectly.

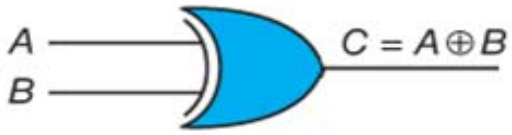
FIGURE 2.19 Three derived gates



(a) NOR gate (b) NAND gate (c) Exclusive OR gate

Exclusive OR

- The **Exclusive OR** function is written as **XOR** or **EOR**.
- The **Exclusive OR** is represented by $\oplus$ (e.g., $C = A \oplus B$).
- A two-input **XOR** gate can be constructed by *two* **inverters**, *two* **AND** gates and *one* **OR** gate, as shown in Figure 2.20. ($F = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$)



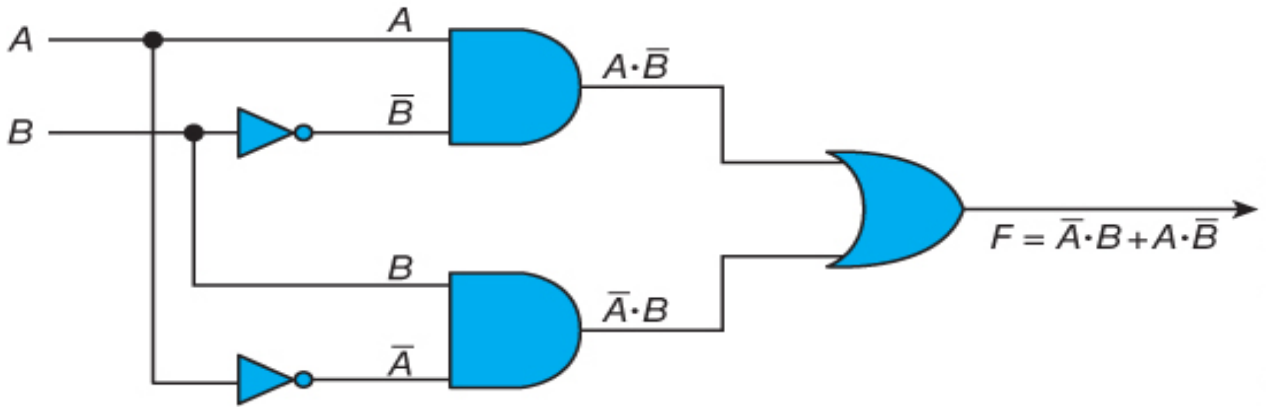
(c) Exclusive OR gate

Exclusive OR Gates

A	B	C = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

(c) The XOR gate

FIGURE 2.20 Constructing an XOR circuit from AND, OR, and NOT gates



Three Input Exclusive OR

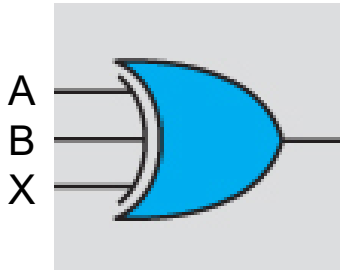
- A three-input **XOR** gate can be constructed with *two* **XOR** gates, each with two-inputs
- $C = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$

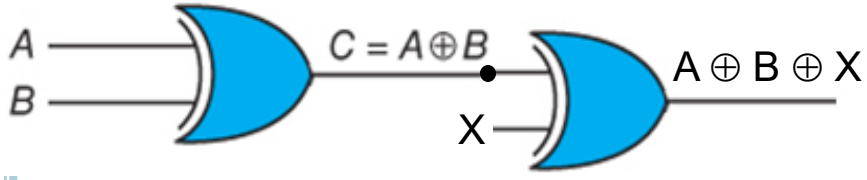
$$\overline{C} = \overline{(A \cdot \overline{B} + \overline{A} \cdot B)} = \overline{A} \cdot \overline{B} + A \cdot B$$

- $A \oplus B \oplus X = C \oplus X = C \cdot \overline{X} + \overline{C} \cdot X$

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

$$= A \cdot \overline{B} \cdot \overline{X} + \overline{A} \cdot B \cdot \overline{X} + \overline{A} \cdot \overline{B} \cdot X + A \cdot B \cdot X$$



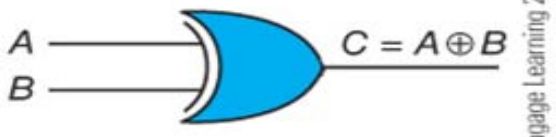


A	B	X	A ⊕ B ⊕ X
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

A	B	C = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Inversion Bubbles

- By **convention**, the triangle in inverters are often omitted from circuit diagrams and the *bubble notation is used*.
- *A small bubble is placed at a gate's input to indicate inversion.*
- In the circuit below, the *two* **AND** gates form the product of (**NOT** A) **AND** B and A **AND** (**NOT** B), i.e., $\overline{A} \cdot B + A \cdot \overline{B}$
- This circuit implements **XOR**



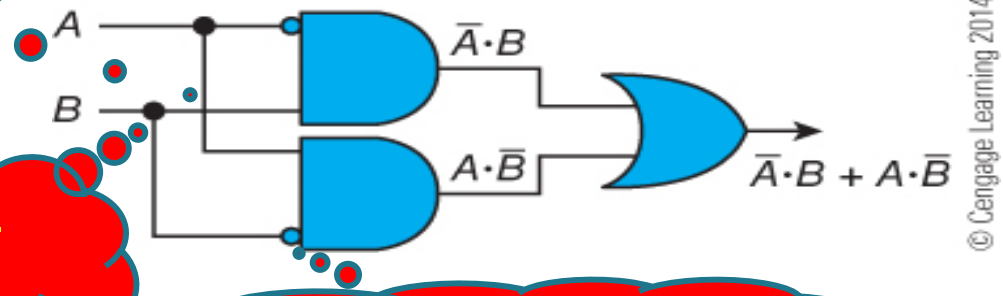
(c) Exclusive OR gate

Exclusive OR Gates

A	B	C = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

(c) The XOR gate

This intersection means not connected lines



This bubble means connected lines

This bubble means inversion

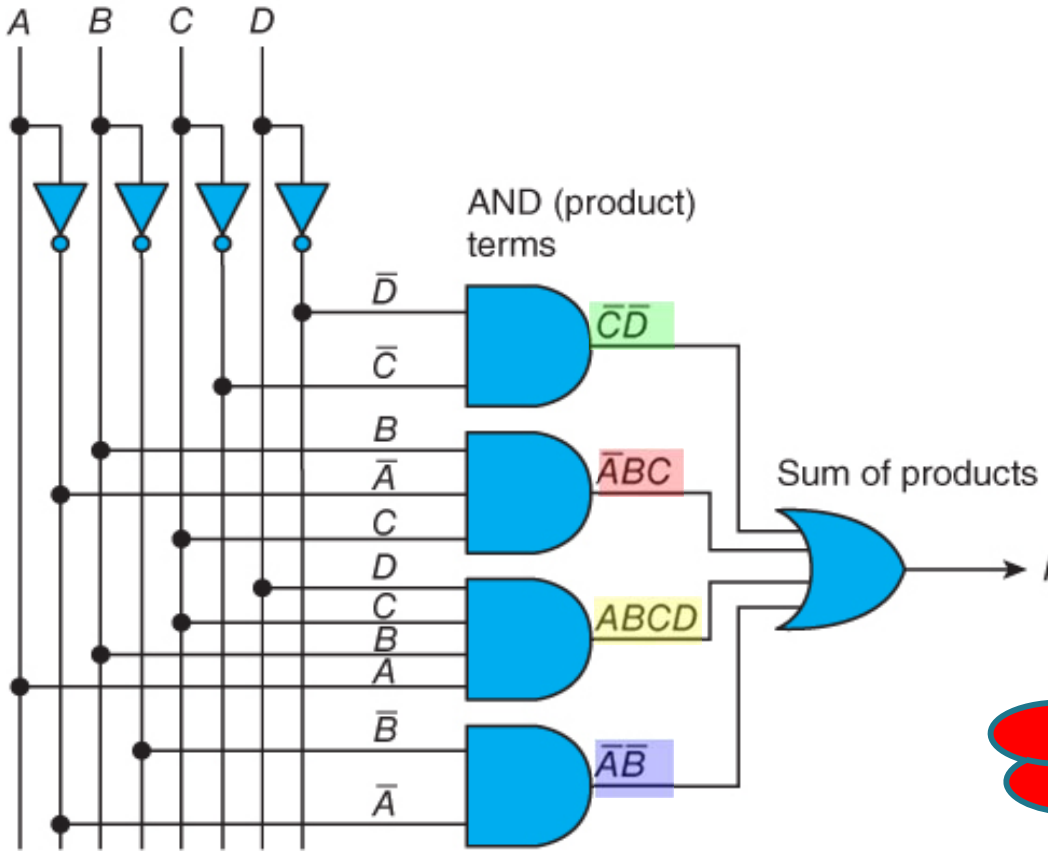
The *sum of products* truth table is identified by its 1's as output

Example of a Digital Circuit

- This is called a ***sum of products*** circuit.
- The output is the **OR** of **AND** terms
- Lines that cross each other ***without a black dot*** at their intersection are ***not connected*** together
- lines that ***meet at a dot*** are **connected**.

A	B	C	D	O/P
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

FIGURE 2.17 The generic AND-OR circuit



When any term of the above function ***equals 1***, the value of the entire function will be ***1***.

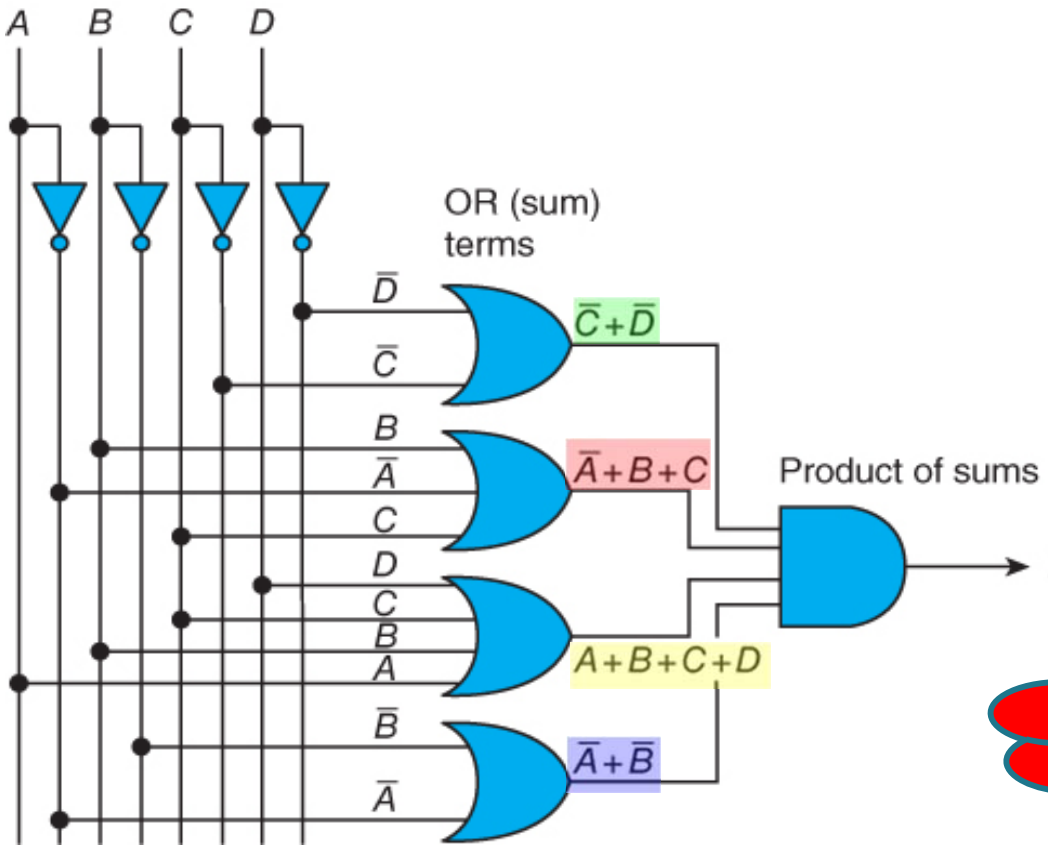
Example of a Digital Circuit

The *product of sums* truth table is identified by its 0's as output

- This is called a ***product of sums*** circuit.
- The output is the **AND** of **OR** terms

A	B	C	D	O/P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
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

FIGURE 2.18 The generic OR-AND circuit



When any term of the above function *equals 0*, the value of the entire function will be *0*.

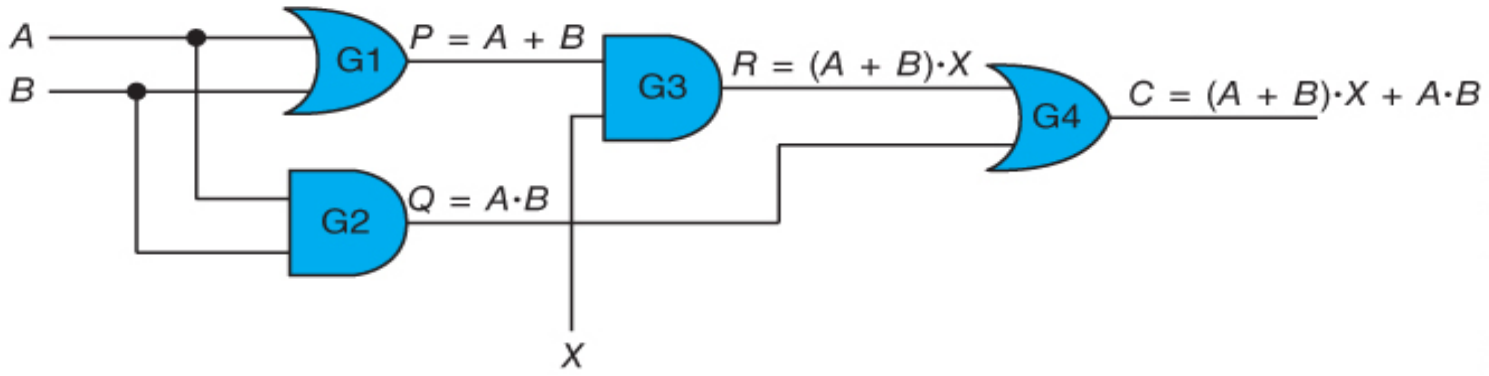
Boolean Algebra Follows Normal Algebraic Laws

- $X + Y = Y + X$ (Commutative law)
- $X \cdot Y = Y \cdot X$ (Commutative law)
- $X + (Y + Z) = (X + Y) + Z$ (Associative law)
- $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$ (Associative law)
- $X + Y \cdot Z = (X + Y).(X + Z)$ (Distributive law)
- $X \cdot (Y + Z) = X \cdot Y + X \cdot Z$ (Distributive law)
- $\overline{X + Y} = \bar{X} \cdot \bar{Y}$ (De Morgan's law)
- $\overline{X \cdot Y} = \bar{X} + \bar{Y}$ (De Morgan's law)
- $X + \bar{X} \cdot Y = X + Y$

More Example of a Digital Circuit

- Figure 2.21 describes a circuit with
  - four gates, labeled **G1**, **G2**, **G3** and **G4**.
  - three inputs A, B, and X, and
  - an output C.
  - It also has three intermediate logical values labeled P, Q, and R.
- We can treat a gate as a *processor* that operates on its inputs according to its logical function;
  - For example, the inputs to gate **G3** are P and X, and its output is P · X.
  - Because P = A + B, the output of **G3** is (A + B) · X.
  - Similarly the output of gate **G4** is R + Q,
  - Because R = (A + B) · X and Q = A · B, the output of gate **G4** is (A + B) · X + A · B.

FIGURE 2.21 A circuit with four gates



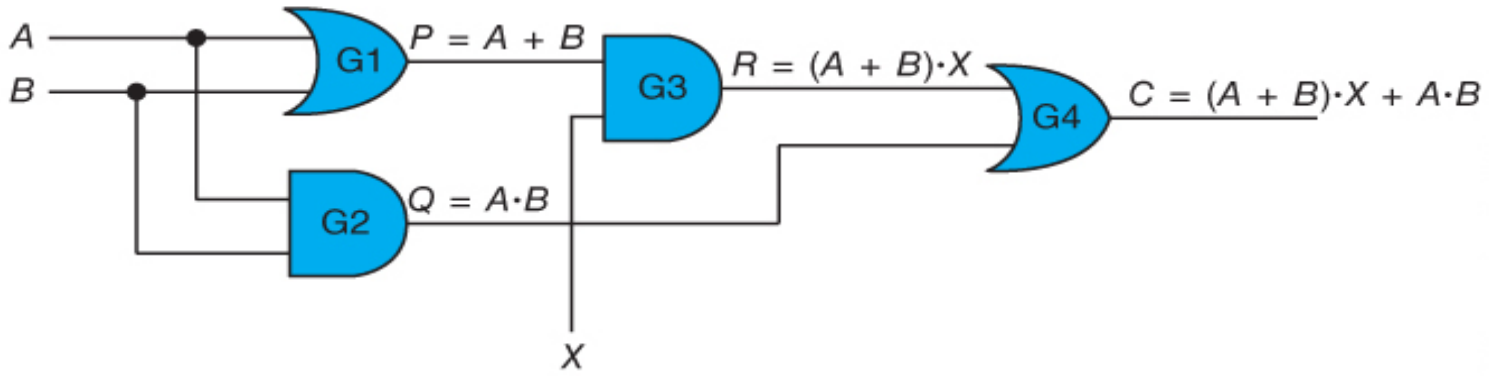
More Example of a Digital Circuit

- Table 2.12 gives the truth table for Figure 2.21.
- Note that the ***output corresponds to the carry out of a 3-bit adder***.

TABLE 2.12 Truth Table for Figure 2.21

Inputs			Intermediate Values			Output
X	A	B	$P = A + B$	$Q = A \cdot B$	$R = (A + B) \cdot X$	$C = Q + R$
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	1	0	0	0
0	1	1	1	1	0	1
1	0	0	0	0	0	0
1	0	1	1	0	1	1
1	1	0	1	0	1	1
1	1	1	1	1	1	1

FIGURE 2.21 A circuit with four gates



The Half-Adder and Full-Adder

- Table 2.13 gives the truth table of a *half-adder* that adds bit A to bit B to get a sum and a carry.

A single-bit full-adder is a logical circuit that performs an addition operation on three one-bit binary digits

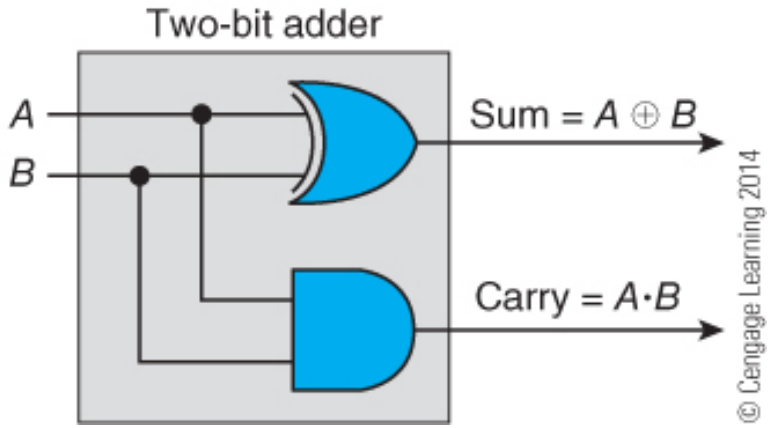
- Figure 2.22 shows the possible structure of a two-bit adder.
  - The carry bit is generated by **AND**ing the two inputs.

TABLE 2.13

Truth Table of a Half Adder

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

FIGURE 2.22 The two-bit adder (the half adder)



Full-Adder Circuit

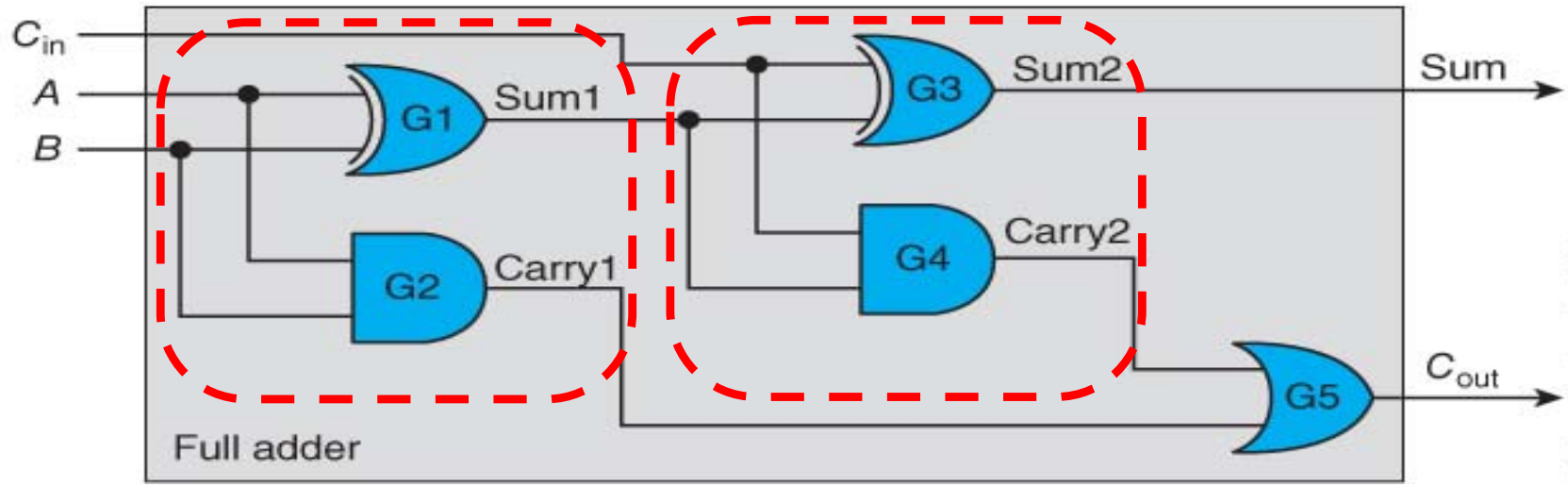
- Figure 2.3 gives the possible circuit of a *one-bit full-adder*.
  - Consists of *two half-adder* and a *one* ***OR*** gate

$$\text{Sum} = (A \oplus B) \oplus C_{in}$$
$$= (A \,.\, \overline{B} + \overline{A} \,.\, B) \,.\, \overline{C_{in}} + \overline{(A \,.\, \overline{B} + \overline{A} \,.\, B)} \,.\, C_{in}$$

$$C_{out} = A \,.\, B + (A \oplus B) \,.\, C_{in}$$

A	B	C_{in}	Sum	C_{out}
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

FIGURE 2.23 The full adder



Full-Adder Circuit

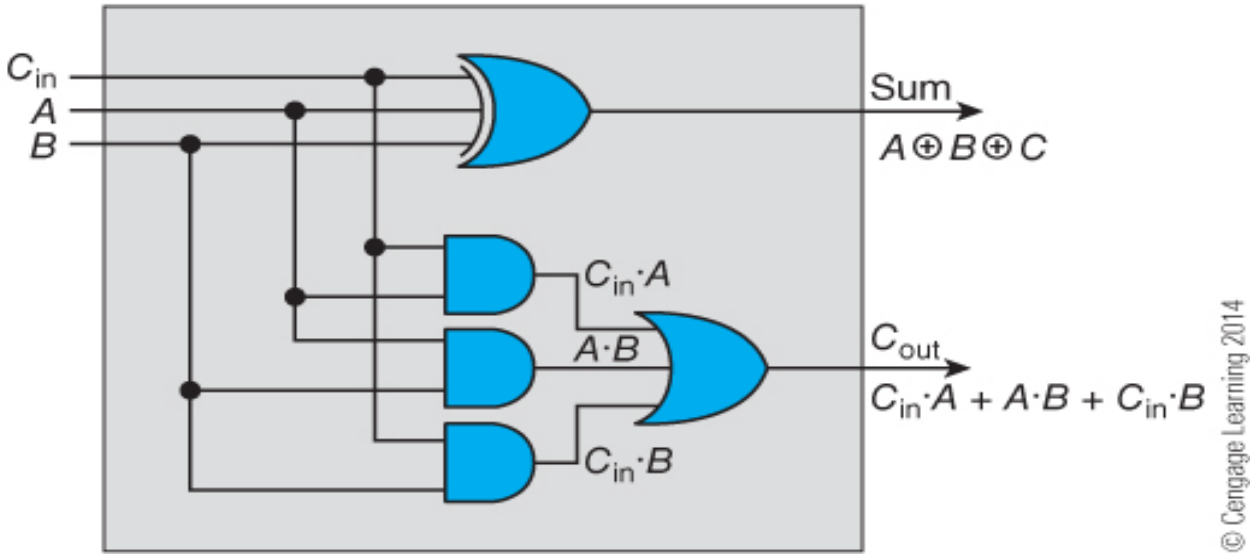
- Figure 2.3 gives an alternative circuit of a *one-bit full-adder*.

$$\text{Sum} = (A \oplus B) \oplus C_{in}$$
$$= (A \cdot \overline{B} + \overline{A} \cdot B) \cdot \overline{C_{in}} + \overline{(A \cdot \overline{B} + \overline{A} \cdot B)} \cdot C_{in}$$

$$C_{out} = C_{in} \cdot A + A \cdot B + C_{in} \cdot B$$

A	B	C_{in}	Sum	C_{out}
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

FIGURE 2.24 Alternative full adder circuit



Full-Adder Circuit

$\text{Sum} = (A \oplus B) \oplus C$

$= (A \,.\, \bar{B} + \bar{A} \,.\, B) \,.\, \bar{C} + \overline{(A \,.\, \bar{B} + \bar{A} \,.\, B)} \,.\, C$

Using De Morgan's law: $\overline{X + Y} = \bar{X} \,.\, \bar{Y}$

$= (A \,.\, \bar{B} + \bar{A} \,.\, B) \,.\, \bar{C} + (\overline{(A \,.\, \bar{B})} \,.\, \overline{(\bar{A} \,.\, B)}) \,.\, C$

Using De Morgan's law: $\overline{X \,.\, Y} = \bar{X} + \bar{Y}$

$= (A \,.\, \bar{B} + \bar{A} \,.\, B) \,.\, \bar{C} + ((\bar{A} + \bar{\bar{B}}) \,.\, (\bar{\bar{A}} + \bar{B})) \,.\, C$

Using property $\bar{\bar{A}} = A$

$= (A \,.\, \bar{B} + \bar{A} \,.\, B) \,.\, \bar{C} + ((\bar{A} + B) \,.\, (A + \bar{B})) \,.\, C$

Using Distributive law: $X \,.\, (Y + Z) = X \,.\, Y + X \,.\, Z$

$= (A \,.\, \bar{B} + \bar{A} \,.\, B) \,.\, \bar{C} + ((\bar{A} + B) \,.\, A + (\bar{A} + B) \,.\, \bar{B})) \,.\, C$

Using Commutative law: $X \,.\, Y = Y \,.\, X$

$= \bar{C} \,.\, (A \,.\, \bar{B} + \bar{A} \,.\, B) + (A \,.\, (\bar{A} + B) + \bar{B} \,.\, (\bar{A} + B)) \,.\, C$

Using Distributive law: $X \,.\, (Y + Z) = X \,.\, Y + X \,.\, Z$

$= (\bar{C} \,.\, A \,.\, \bar{B} + \bar{C} \,.\, \bar{A} \,.\, B) + ((A \,.\, \bar{A} + A \,.\, B) + (\bar{B} \,.\, \bar{A} + \bar{B} \,.\, B)) \,.\, C$

Using property $\bar{X} \,.\, X = 0$

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

Using inversion property: $X + 0 = X$

$= (\bar{C} \,.\, A \,.\, \bar{B} + \bar{C} \,.\, \bar{A} \,.\, B) + (A \,.\, B + \bar{B} \,.\, \bar{A}) \,.\, C$

Using Commutative law: $X \,.\, Y = Y \,.\, X$

$= (\bar{C} \,.\, A \,.\, \bar{B} + \bar{C} \,.\, \bar{A} \,.\, B) + C \,.\, (A \,.\, B + \bar{B} \,.\, \bar{A})$

Using Distributive law: $X \,.\, (Y + Z) = X \,.\, Y + X \,.\, Z$

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

Using Commutative law: $X \,.\, Y = Y \,.\, X$

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

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

Full-Adder Circuit

$C_{out} = (A + B)\cdot C + A\cdot B$ *From Figure 2.21*

$C_{out} = C\cdot A + A\cdot B + C\cdot B$ *From Figure 2.24*

$C_{out} = A.B + (A \oplus B).C$ *From Figure 2.23*

$C_{out} = A.B + (A.\overline{B} + \overline{A}.B).C$

Using Distributive law

$C_{out} = A.B + A.\overline{B}.C + \overline{A}.B.C$

Using Distributive law

$C_{out} = A.(B + \overline{B}.C) + \overline{A}.B.C$

Using $X + \overline{X}.Y = X + Y$

$C_{out} = A.(B + C) + \overline{A}.B.C$

Using Distributive law

$C_{out} = A.B + A.C + \overline{A}.B.C$

Using Distributive law

$C_{out} = A.B + (A+\overline{A}.B).C$

Using $X + \overline{X}.Y = X + Y$

$C_{out} = A.B + (A+ B).C$

Using Distributive law

$C_{out} = A.B + A.C+ B.C$

Using Commutative law

$C_{out} = C\cdot A + A\cdot B + C\cdot B$ *As in Figure 2.24*

Using Commutative law:

$C_{out} = A\cdot C + B\cdot C + A\cdot B$

Using Distributive law

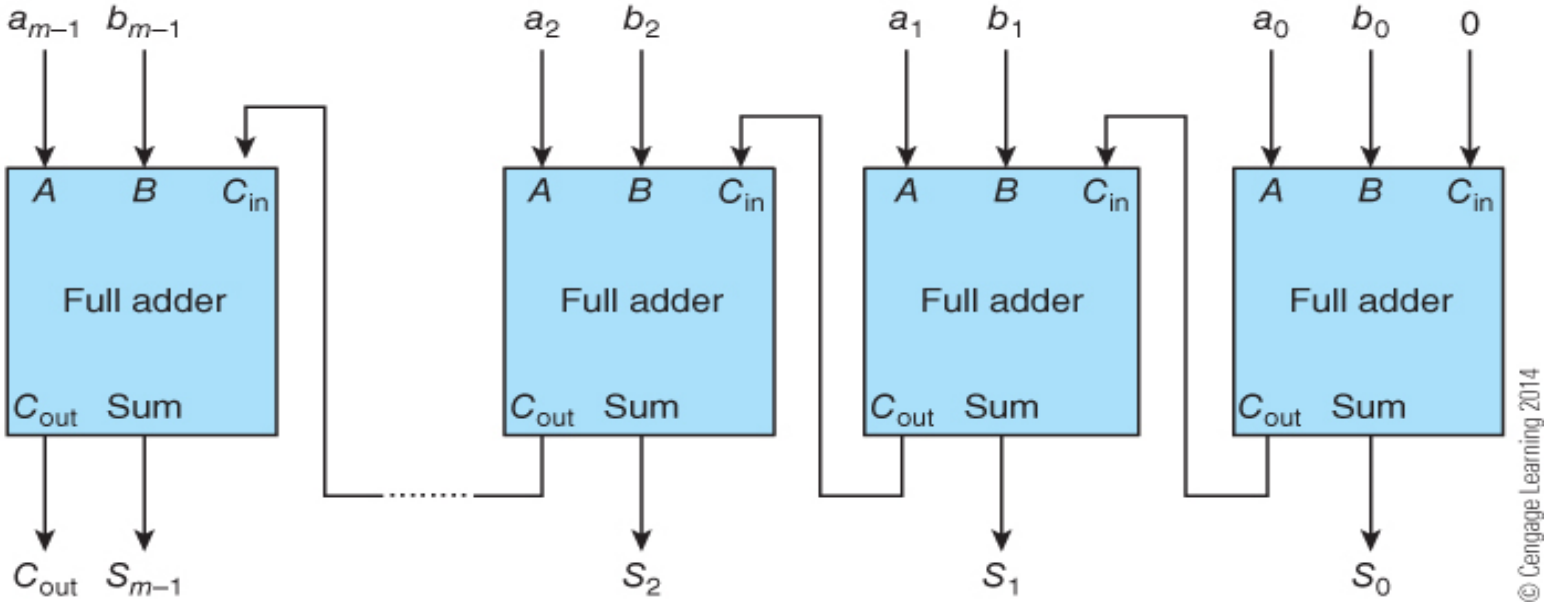
$C_{out} = (A + B)\cdot C + A\cdot B$ *As in Figure 2.21*

A	B	C	C_{out}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Full-Adder

- We need *m full-adder* circuits to add two *m*-bit words *in parallel* as Figure 2.25 demonstrates.
- The m_i *full-adder* adds bit a_i to bit b_i, together with a *carry-in* from the stage on its right, to produce a sum_i and a *carry-out* to the stage on its left.

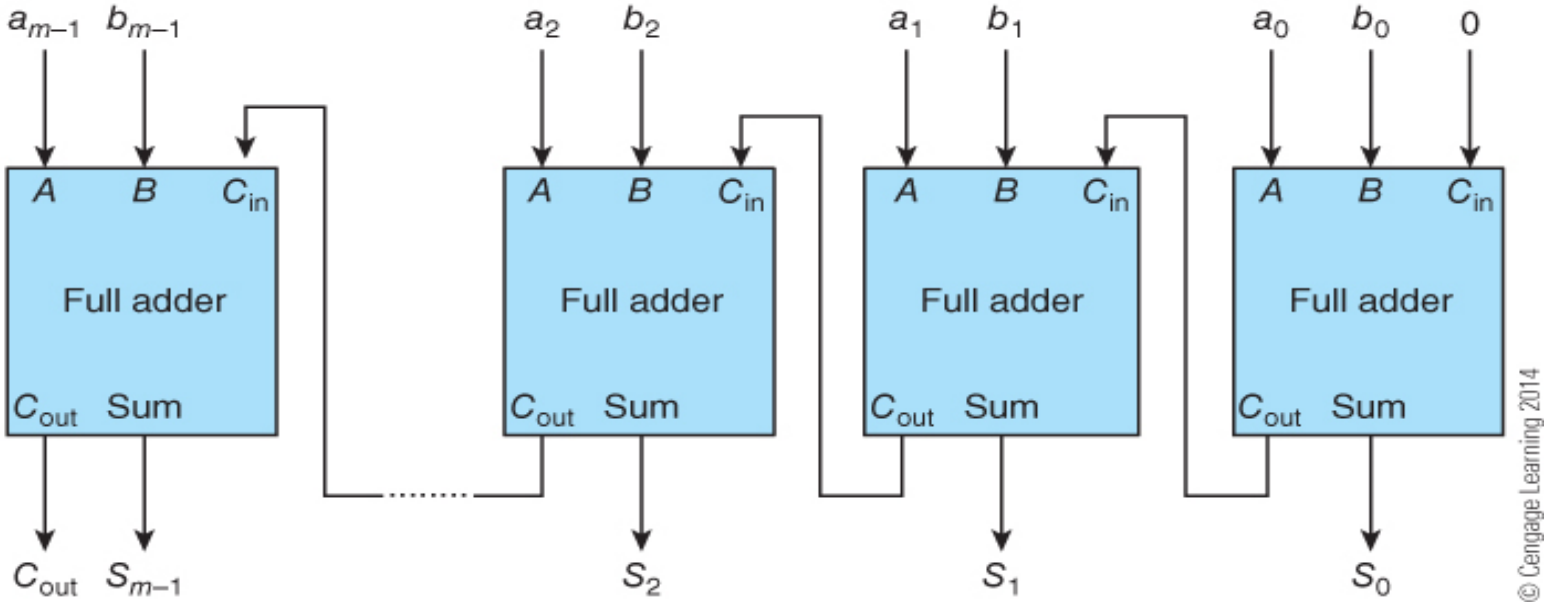
FIGURE 2.25 The parallel adder



Full-Adder

- This circuit is called a parallel-adder because all the bits of the two words to be added are presented to it at the same time.
- The circuit is *not truly parallel* because bit s_i cannot be correctly produced until the *carry-in*$_i$ bit has been calculated by the *previous stage*.
- This is a *ripple through* adder because addition is not complete until the carry bit has ***rippled*** through the circuit.
- *True parallel-adders* use high-speed *look-ahead carry* circuits to produce all carry bits at once, hence speeding up the addition operation.

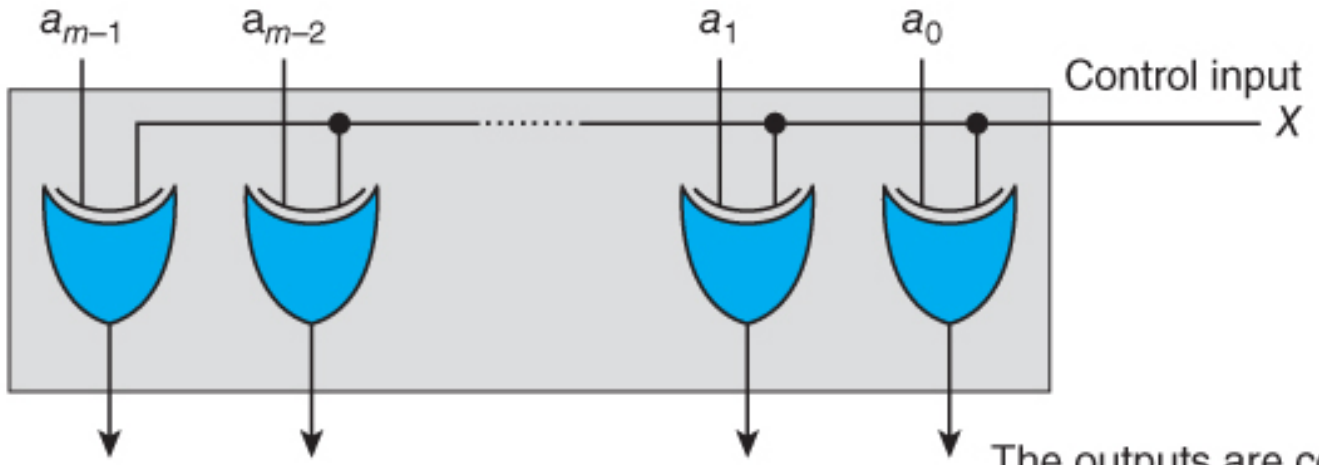
FIGURE 2.25 The parallel adder



Programmable Inverter

a	X	a ⊕ X
0	0	0
0	1	1
1	0	1
1	1	0

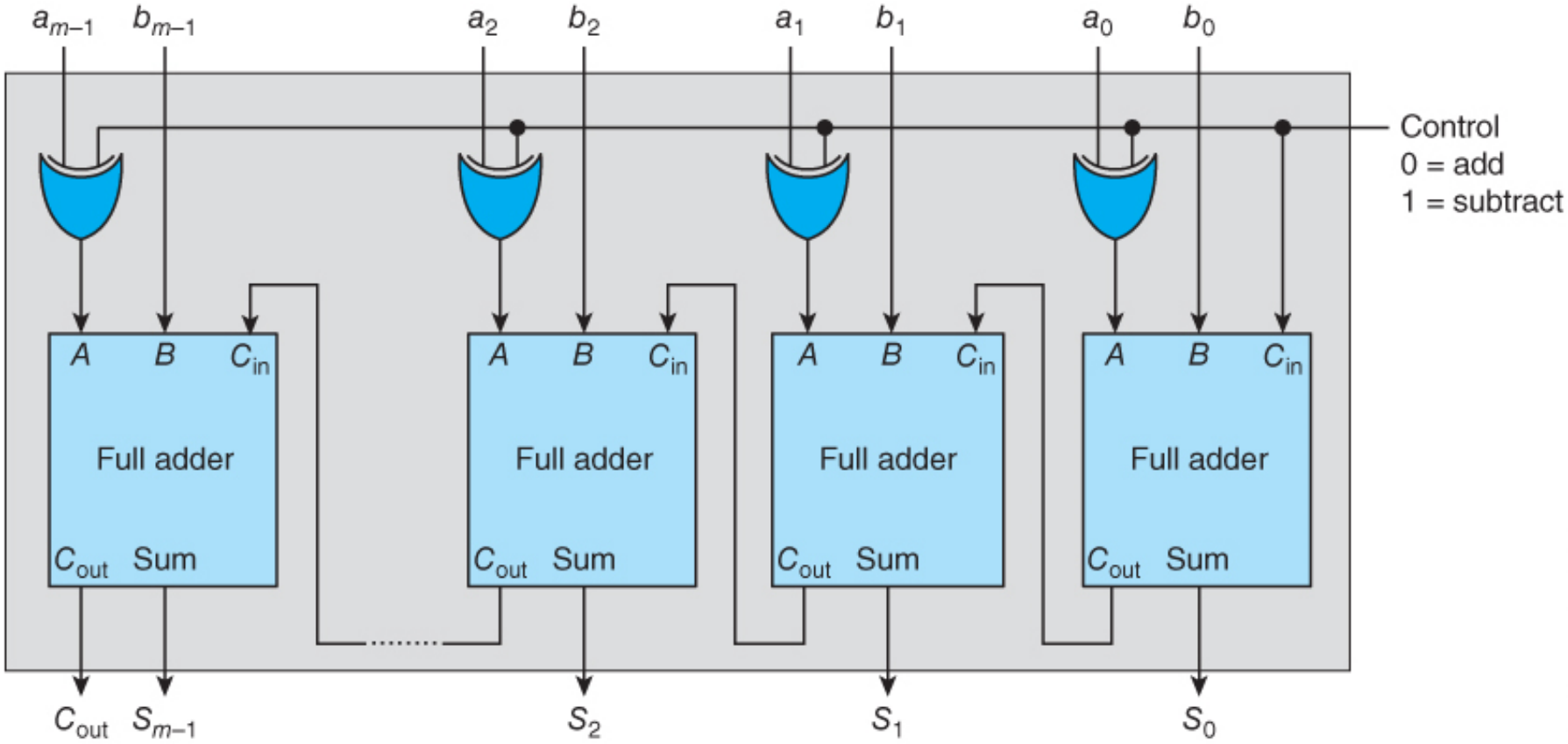
FIGURE 2.26 The programmable inverter



The outputs are copies of the respective inputs if $X = 0$, and the complements of the inputs if $X = 1$

Full-Adder/Subtractor

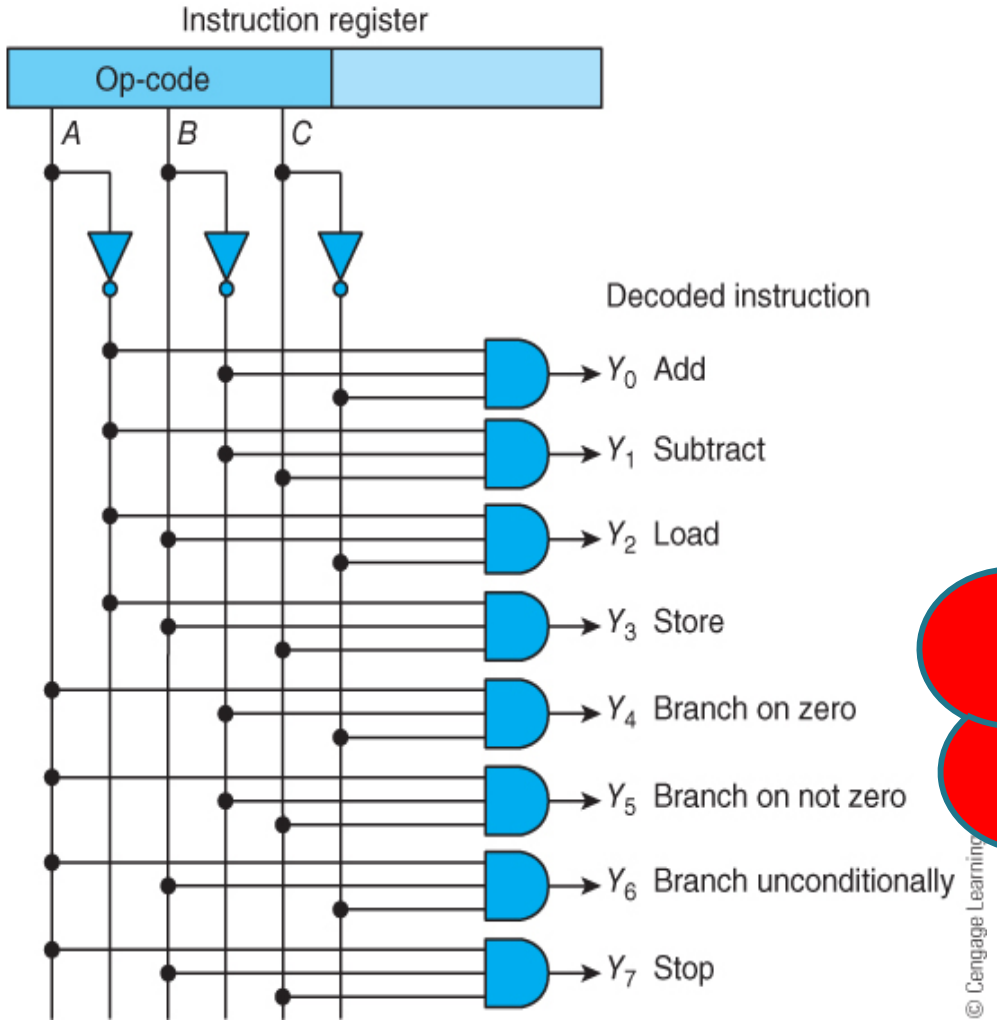
FIGURE 2.27 The adder/subtractor



The Decoder

- Figure 2.29 has ***three*** inputs A, B, and C, and ***eight*** outputs Y0 to Y7.
- The ***three*** inverters generate the complements of the inputs A, B, and C.
- Each of the ***eight*** ***AND*** gates is connected to ***three*** of the six lines .
  - each of the ***three*** variables appear in either its true or complemented form.

FIGURE 2.28 Application of a decoder



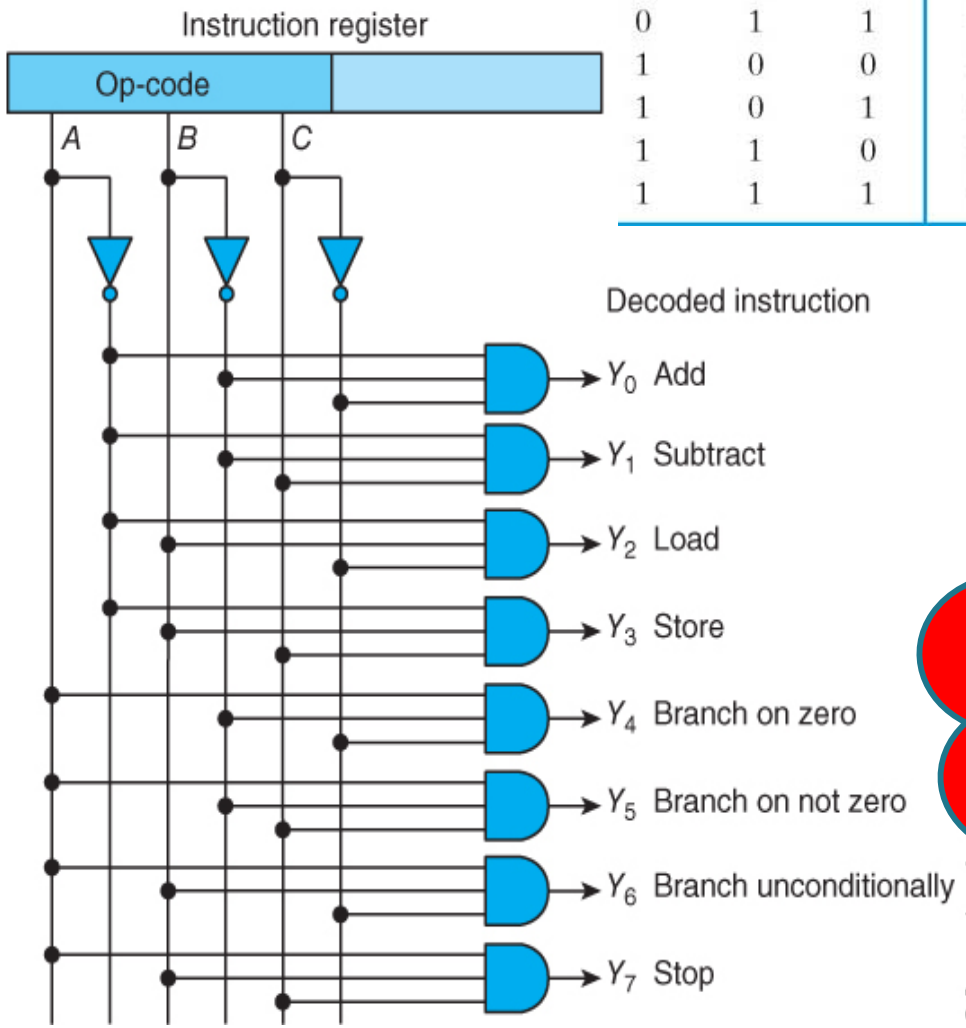
A ***decoder*** is combinational logic circuit that converts binary information from the n-bits coded input to a maximum of 2^n unique outputs.

The Decoder

TABLE 2.15 The Decoder

Inputs			Outputs							
A	B	C	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

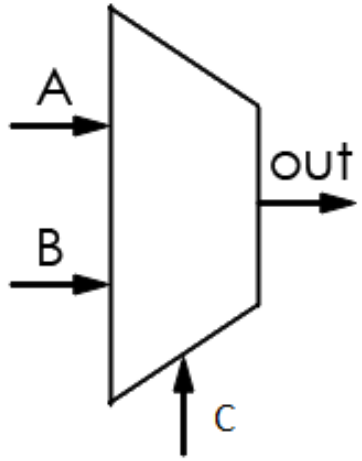
FIGURE 2.28 Application of a decoder



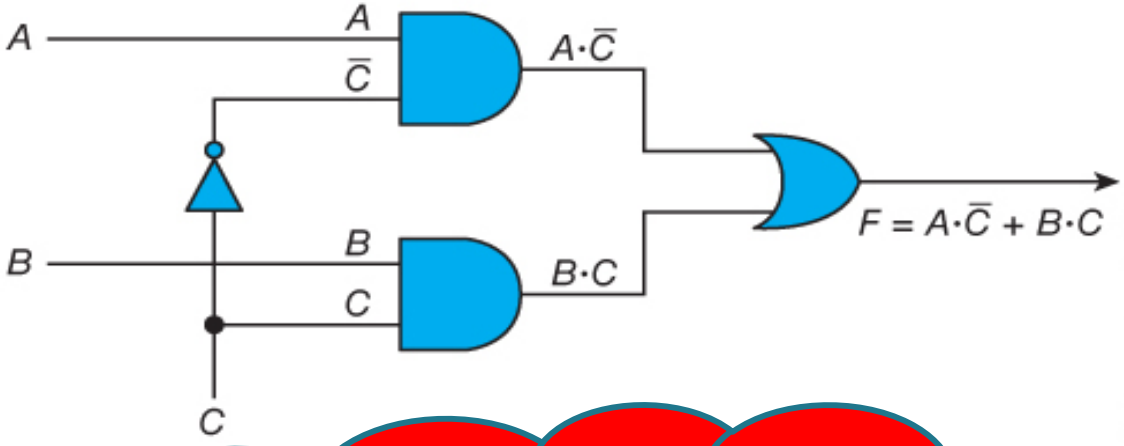
A ***decoder*** is combinational logic circuit that converts binary information from the n-bits coded input to a maximum of 2^n unique outputs.

The Multiplexer

- When C = 0, the output is A
- When C = 1, the output is B
- C works as a selector to select either A or B to go



Alternative representation of the two-input multiplexer



Truth table

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

A ***multiplexer*** is combinational logic circuit that has up to 2^n binary input lines and n select lines, where the n select lines are used to forward one of the input values to the output line.

The Multiplexer

- When C = 0, the output is A
- When C = 1, the output is B
- C works as a selector to select either A or B to go

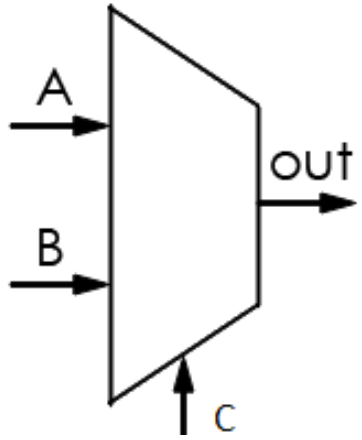
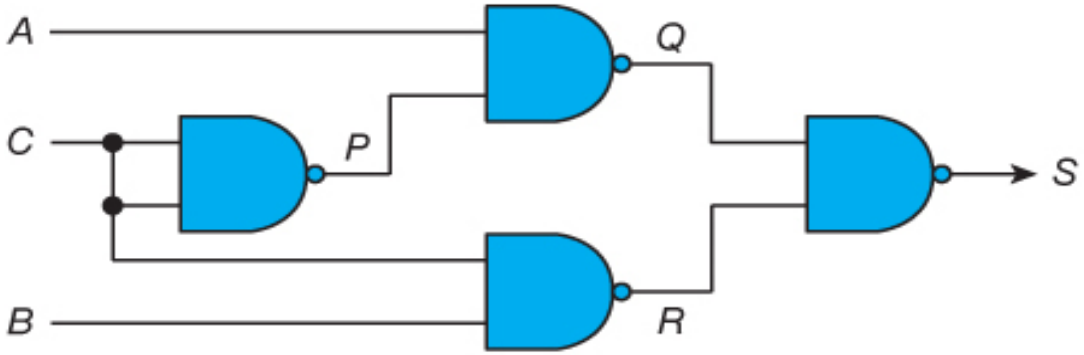


FIGURE 2.29 The two-input multiplexer and its truth table

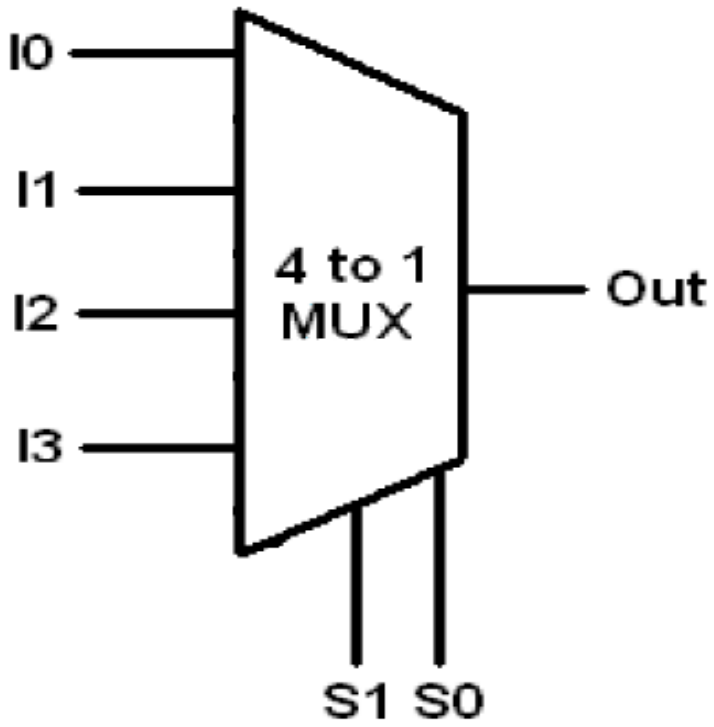


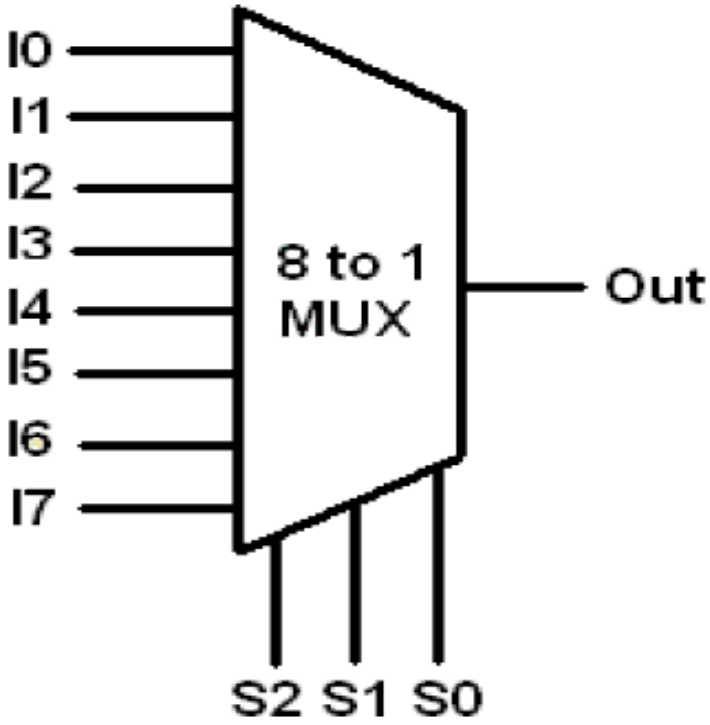
Truth table

C	A	B	P	Q	R	S
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1

A ***multiplexer*** is combinational logic circuit that has up to 2^n binary input lines and n select lines, where the n select lines are used to forward one of the input values to the output line.

The Multiplexer





A ***multiplexer*** is combinational logic circuit that has up to 2^n binary input lines and n select lines, where the n select lines are used to forward one of the input values to the output line.

One Bit of an ALU

- This diagram describes one-bit of a primitive ALU that can perform five operations on bits A and B (**XOR**, **AND**, **OR**, **NOT A** and **NOT B**).
- The function to be performed is determined by the *three-bit control signal* F2,F1,F0.
- The five functions are generated by the five gates on the left.
- On the right, five **AND** gates are used to gate the selected function to an **OR** gate to produce the output.