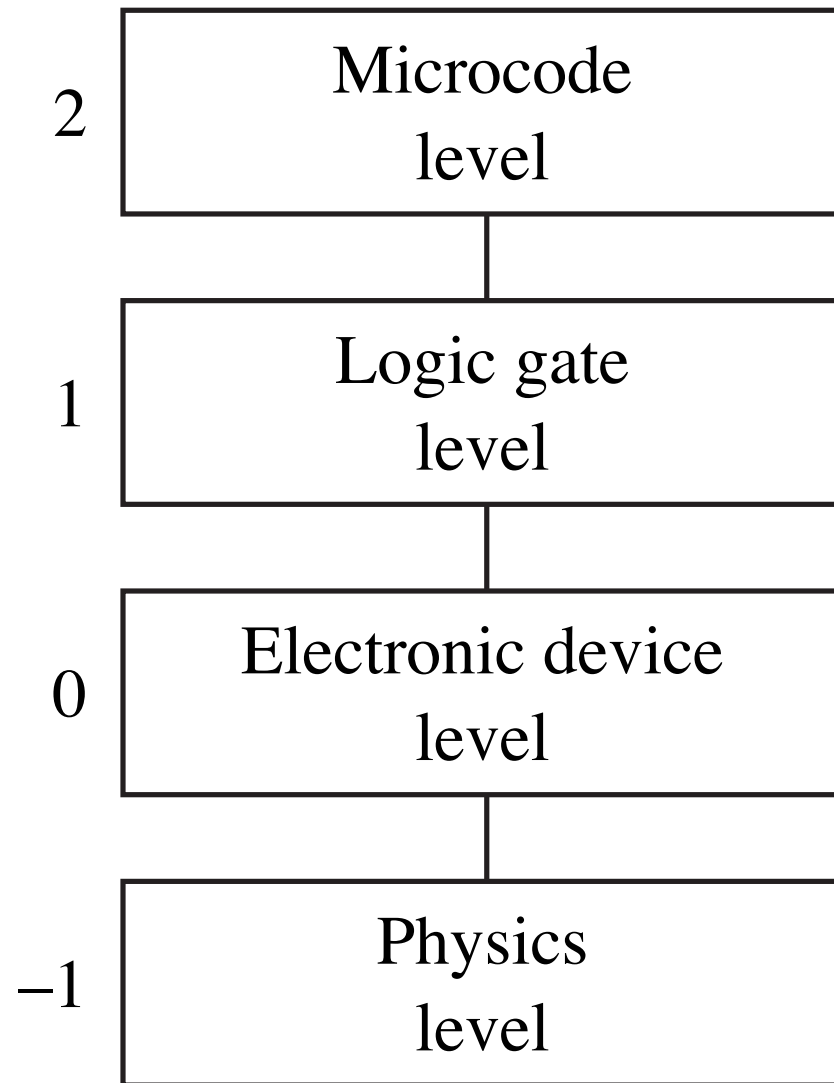
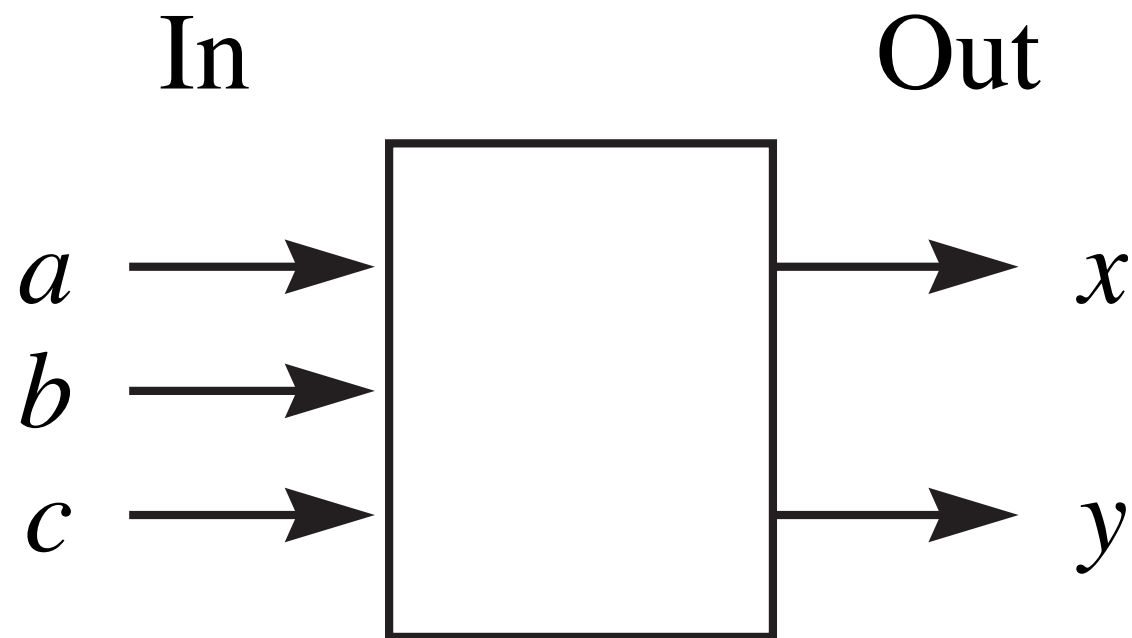


Combinational Circuits



Combinational circuit

- The output depends only on the input



Methods to describe a combinational circuit

- Truth table
- Boolean algebraic expression
- Logic diagram

Truth table

- Lists the output for every combination of the input

<i>a</i>	<i>b</i>	<i>c</i>	<i>x</i>	<i>y</i>
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>x</i>	<i>y</i>
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	1	0

Boolean algebra

- Three basic operations
 - ▶ Binary OR $+$
 - ▶ Binary AND \cdot
 - ▶ Unary Complement $'$

Ten properties of boolean algebra

- Commutative
- Associative
- Distributive
- Identity
- Complement

Commutative

$$x + y = y + x$$

$$x \cdot y = y \cdot x$$

Associative

$$(x + y) + z = x + (y + z)$$

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

Distributive

$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$

Identity

$$x + 0 = x$$

$$x \cdot 1 = x$$

Complement

$$x + (x') = 1$$

$$x \cdot (x') = 0$$

Precedence	Operator
Highest	Complement
	AND
Lowest	OR

Distributive

$$x + y \cdot z = (x + y) \cdot (x + z)$$

$$x \cdot (y + z) = x \cdot y + x \cdot z$$

Complement

$$x + x' = 1$$

$$x \cdot x' = 0$$

Associativity

$$(x + y) + z$$

$$x + y + z$$

Duality

- To obtain the dual expression
 - ▶ Exchange $+$ and \cdot
 - ▶ Exchange 1 and 0

Idempotent property

$$x + x = x$$

$$x \cdot x = x$$

Zero theorem

$$x + 1 = 1$$

$$x \cdot 0 = 0$$

Absorption property

$$x + x \cdot y = x$$

$$x \cdot (x + y) = x$$

Consensus theorem

$$x \cdot y + x' \cdot z + y \cdot z = x \cdot y + x' \cdot z$$

$$(x + y) \cdot (x' + z) \cdot (y + z) = (x + y) \cdot (x' + z)$$

De Morgan's law

$$(a \cdot b)' = a' + b'$$

$$(a + b)' = a' \cdot b'$$

Complement theorems

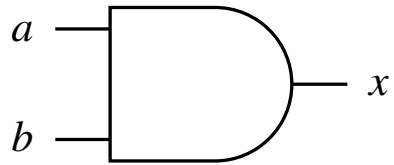
$$(x')' = x$$

$$1' = 0$$

$$0' = 1$$

Logic diagrams

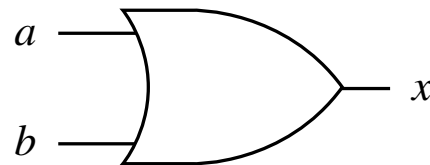
- An interconnection of logic gates
- Closely resembles the hardware
 - ▶ Gate symbol represents a group of transistors and other electronic components
 - ▶ Lines connecting gate symbols represent wires



$$x = a \cdot b$$

<i>a</i>	<i>b</i>	<i>x</i>
0	0	0
0	1	0
1	0	0
1	1	1

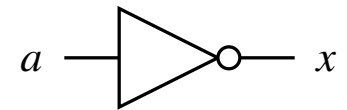
(a) AND gate.



$$x = a + b$$

<i>a</i>	<i>b</i>	<i>x</i>
0	0	0
0	1	1
1	0	1
1	1	1

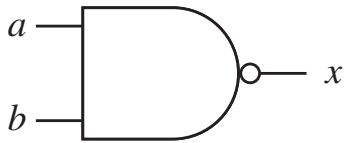
(b) OR gate.



$$x = a'$$

<i>a</i>	<i>x</i>
0	1
1	0

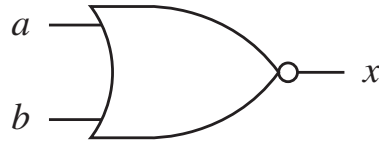
(c) Inverter.



$$x = (a \cdot b)'$$

<i>a</i>	<i>b</i>	<i>x</i>
0	0	1
0	1	1
1	0	1
1	1	0

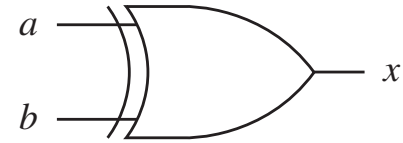
(a) NAND gate.



$$x = (a + b)'$$

<i>a</i>	<i>b</i>	<i>x</i>
0	0	1
0	1	0
1	0	0
1	1	0

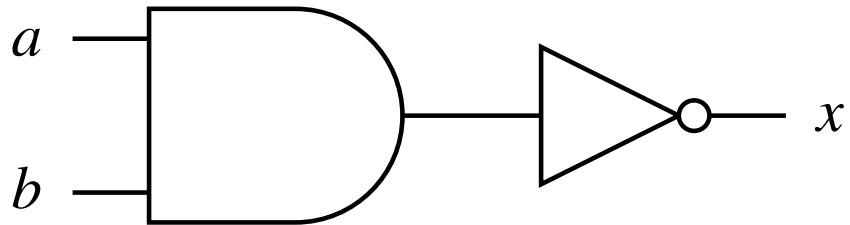
(b) NOR gate.



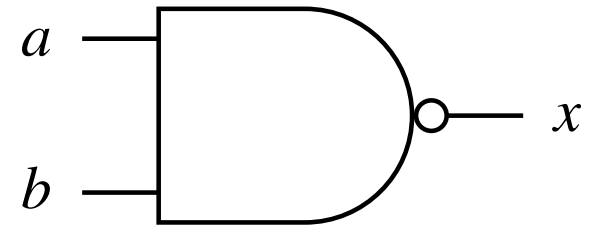
$$x = a \oplus b$$

<i>a</i>	<i>b</i>	<i>x</i>
0	0	0
0	1	1
1	0	1
1	1	0

(c) XOR gate.

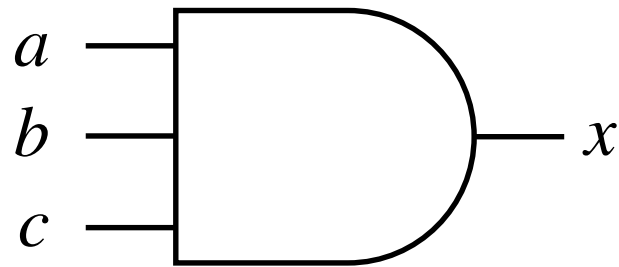


(a) AND inverter.



(b) NAND.

Precedence	Operator
Highest	Complement AND XOR
Lowest	OR

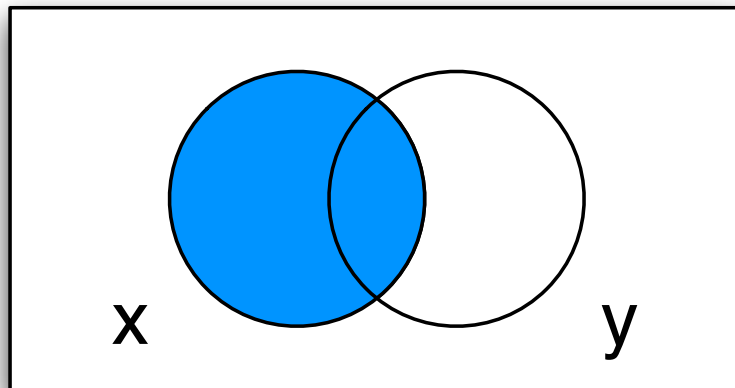


$$x = a \cdot b \cdot c$$

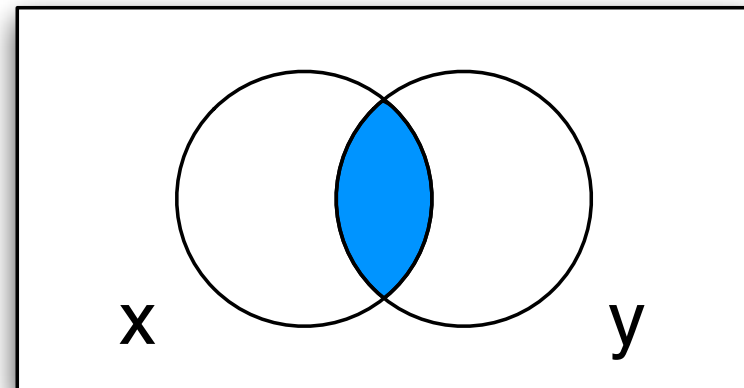
a	b	c	x
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Set theory representation

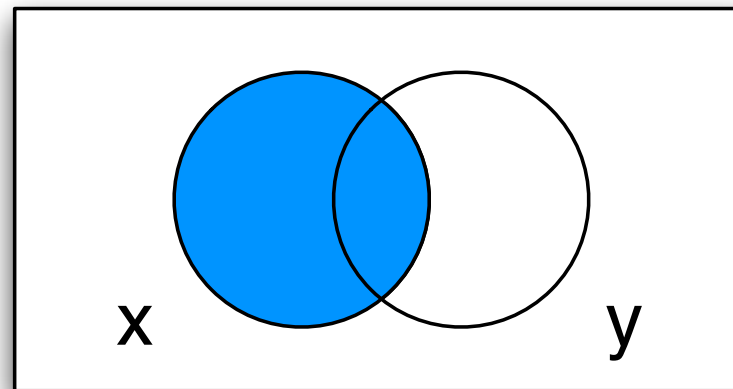
- OR gate is set union
- AND gate is set intersection
- Inverter is set complement



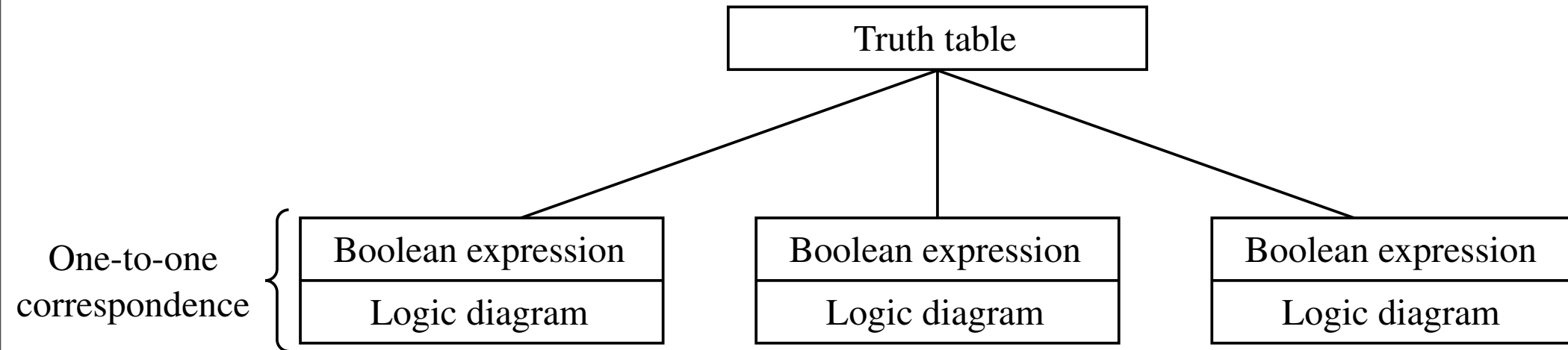
(a) x



(b) $x \cdot y$

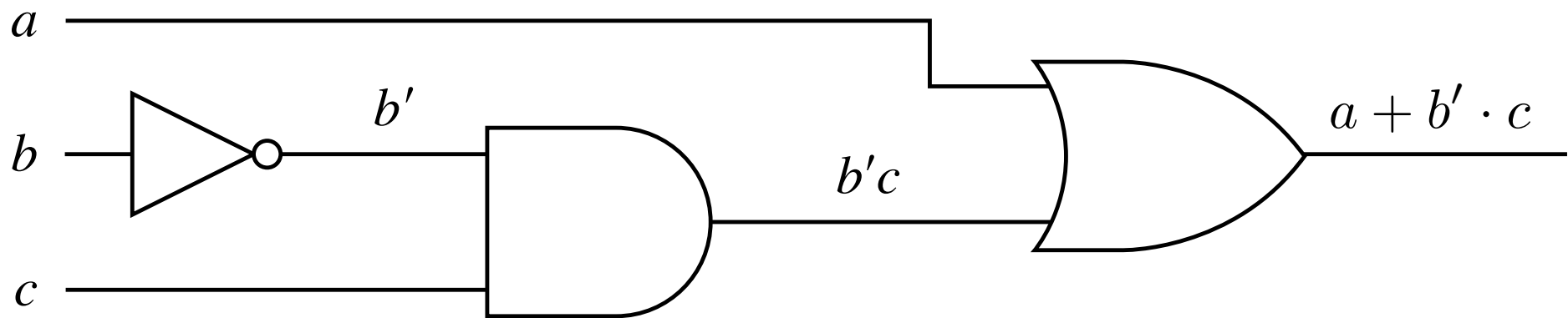


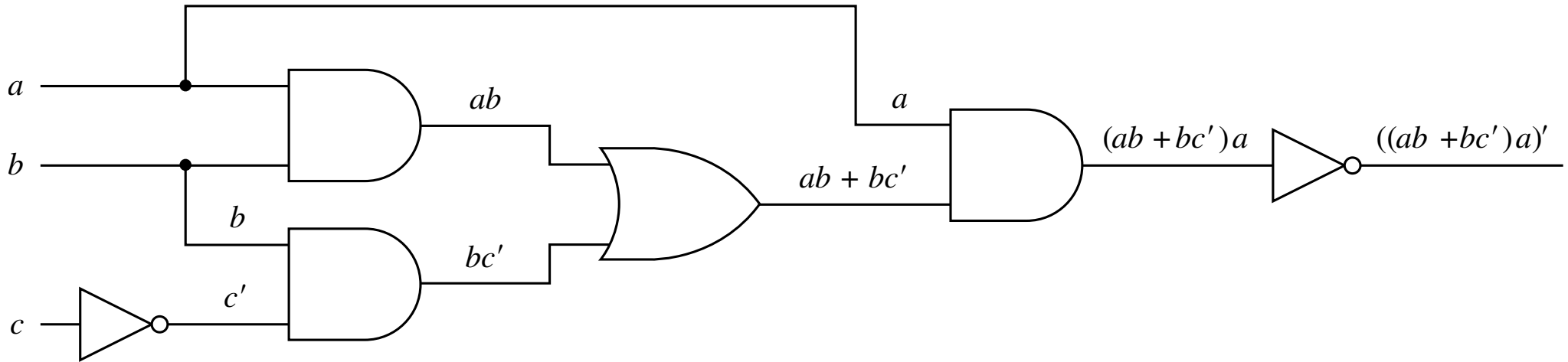
(c) $x + x \cdot y$



Boolean expressions and logic diagrams

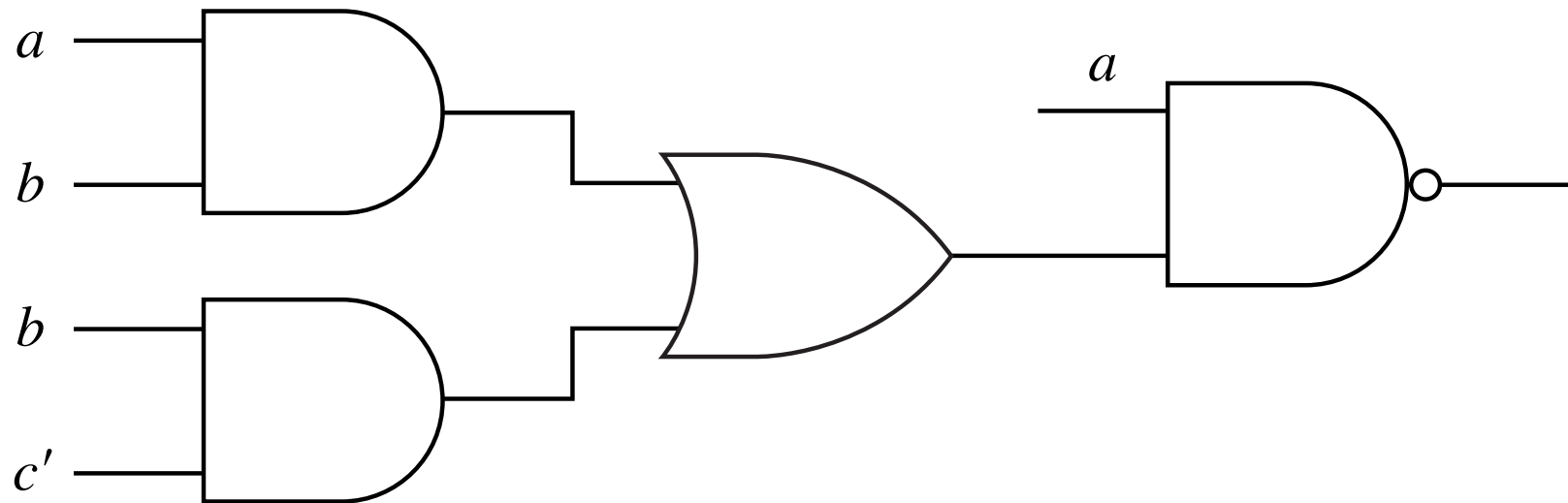
- AND gate corresponds to AND operation
- OR gate corresponds to OR operation
- Inverter corresponds to complement operation



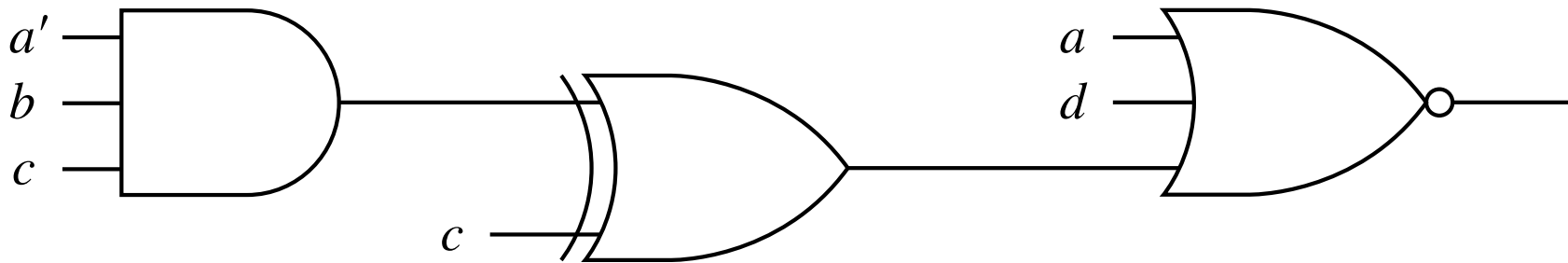


Abbreviated logic diagrams

- Any signal can be duplicated by a junction of two wires
- The complement of any variable can be produced by an inverter



$$(a'bc \oplus c + a + d)'$$



Truth tables and boolean expressions

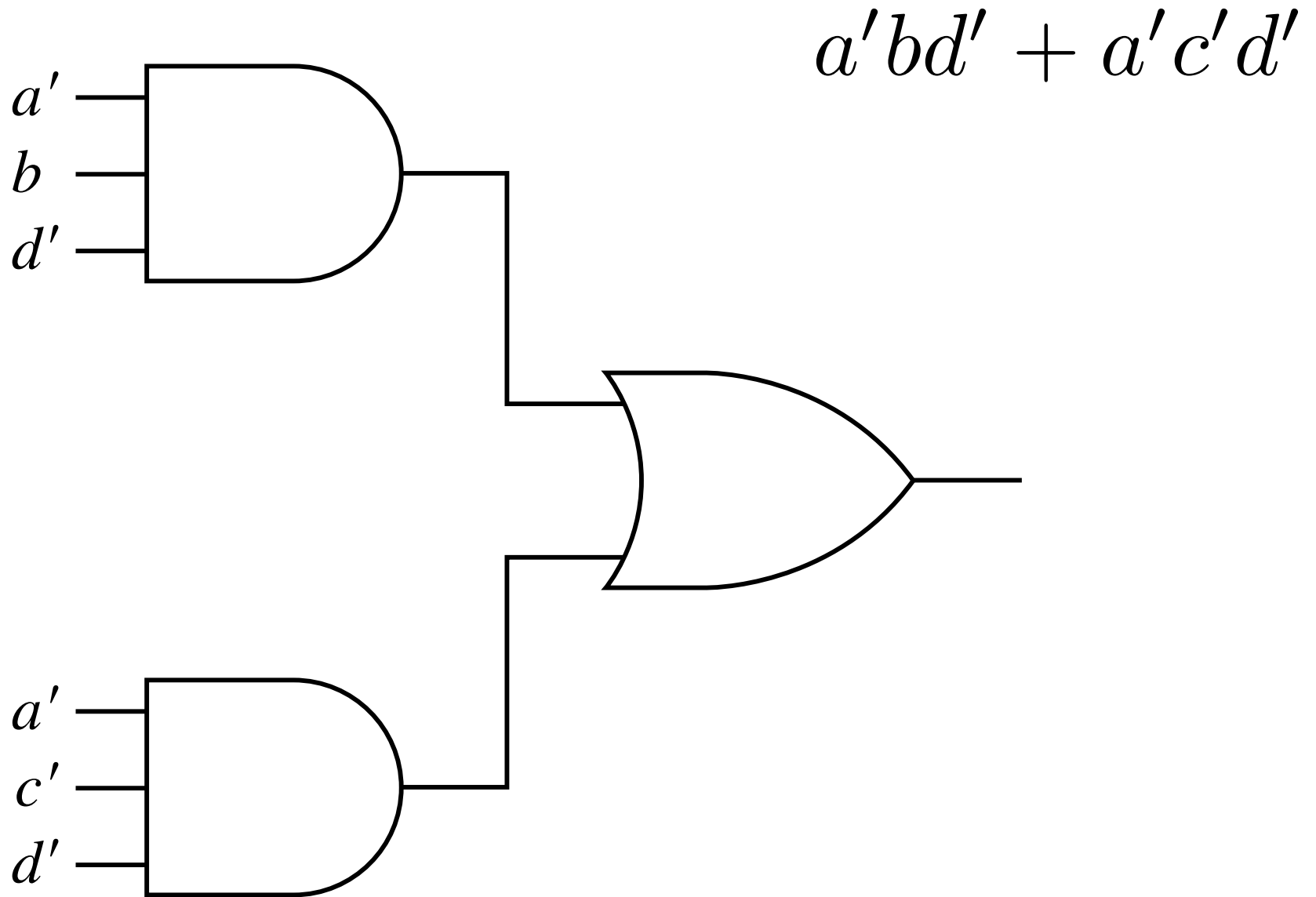
- Given a truth table, write a boolean expression without parentheses as an OR of several AND terms
- Each AND term corresponds to a 1 in the truth table

<i>a</i>	<i>b</i>	<i>c</i>	<i>x</i>
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

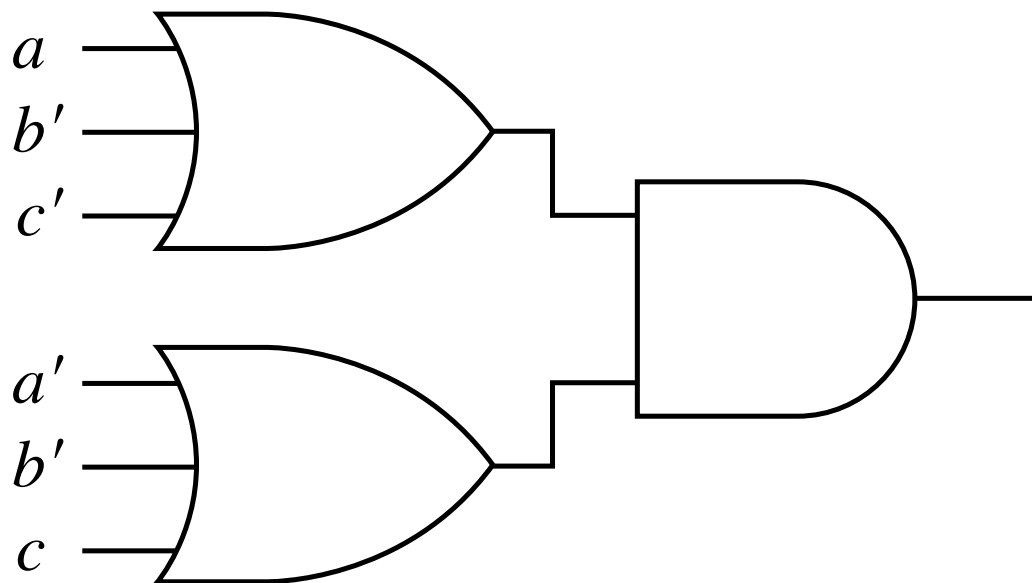
<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>x</i>
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Two-level circuits

- The *gate delay* is the time it takes for the output of a gate to respond to a change in its input
- Any combinational circuit can be transformed into an AND-OR circuit or an OR-AND circuit with at most two gate delays (not counting the gate delay of any inverters)



$$(a + b' + c')(a' + b' + c)$$

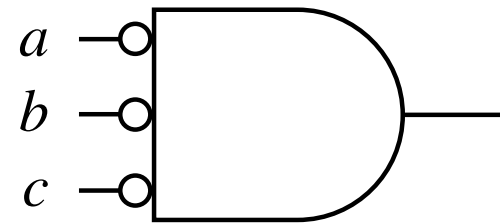
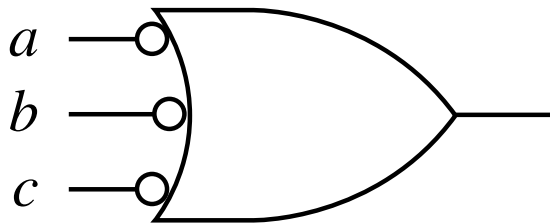
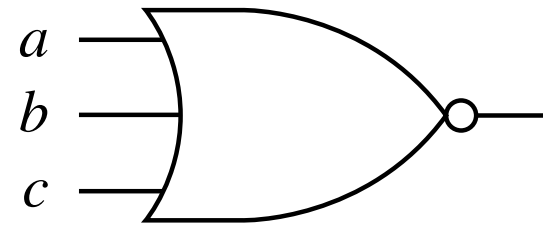
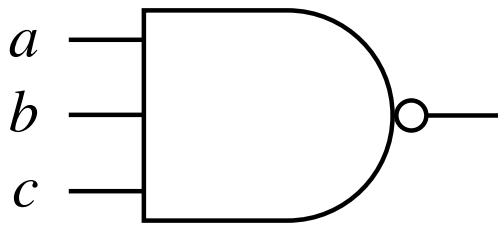


AND-OR versus OR-AND

- To transform any expression x to an equivalent OR-AND expression
 - ▶ Transform the complement of x to an AND-OR expression without parentheses using boolean algebra theorems
 - ▶ Use $x = (x')'$ and De Morgan's law

$$(abc)' = a' + b' + c'$$

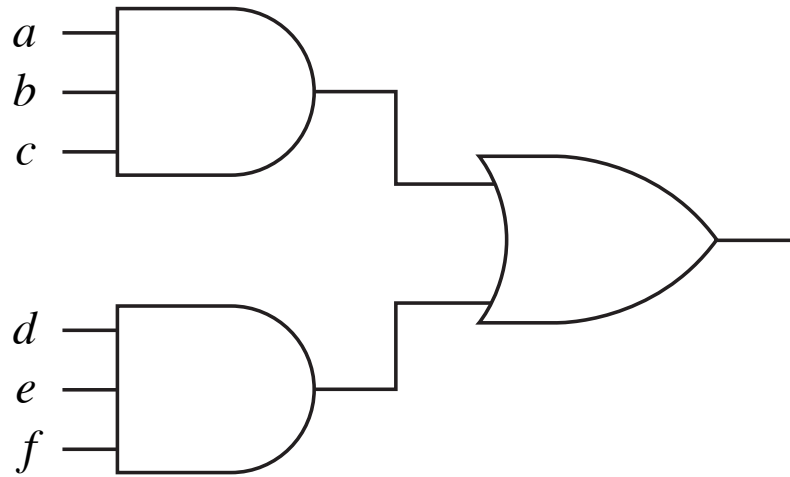
$$(a + b + c)' = a'b'c'$$



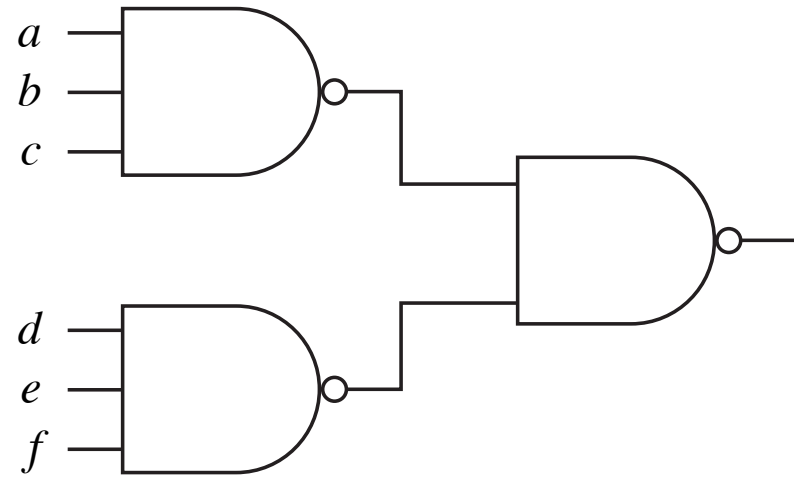
(a) A NAND gate as an inverted input OR gate.

(b) A NOR gate as an inverted input AND gate.

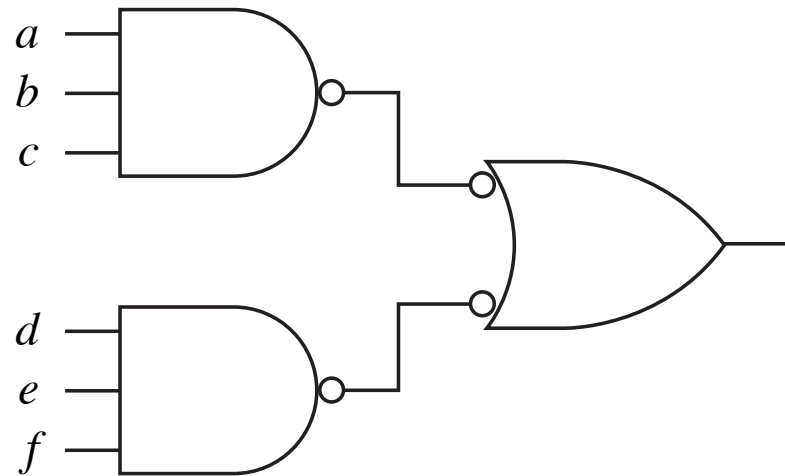
$$abc + def = [(abc)'(def)']'$$



(a) An AND-OR circuit.

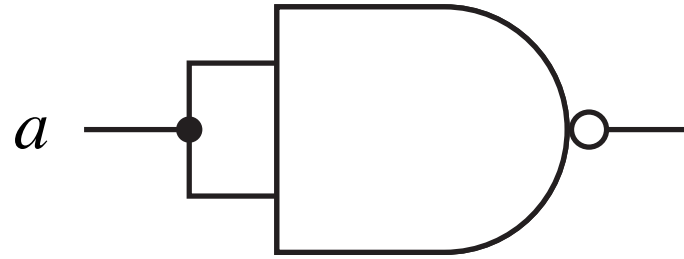


(b) The equivalent NAND-NAND circuit.

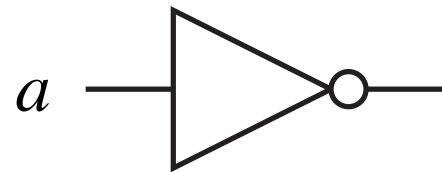
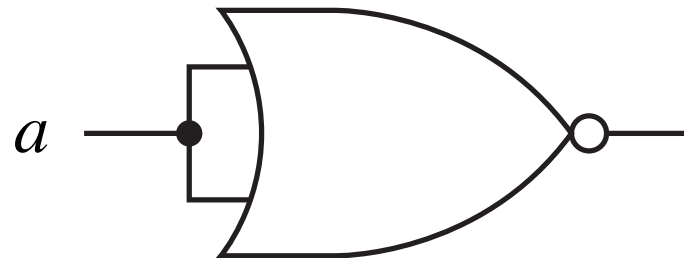


(c) The same NAND-NAND circuit as in part (b).

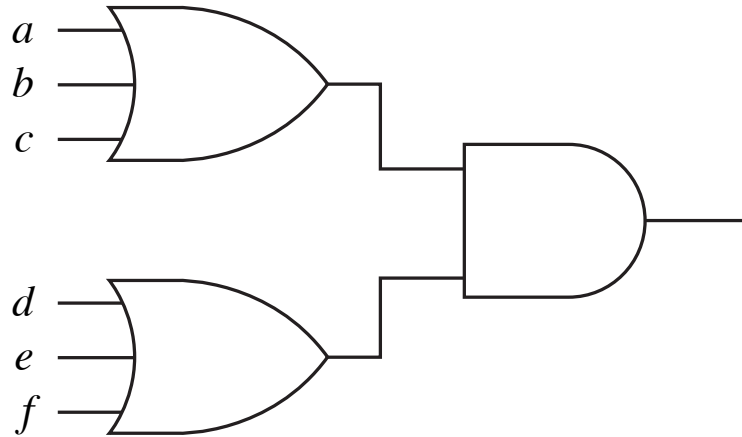
$$(a \cdot a)' = a'$$



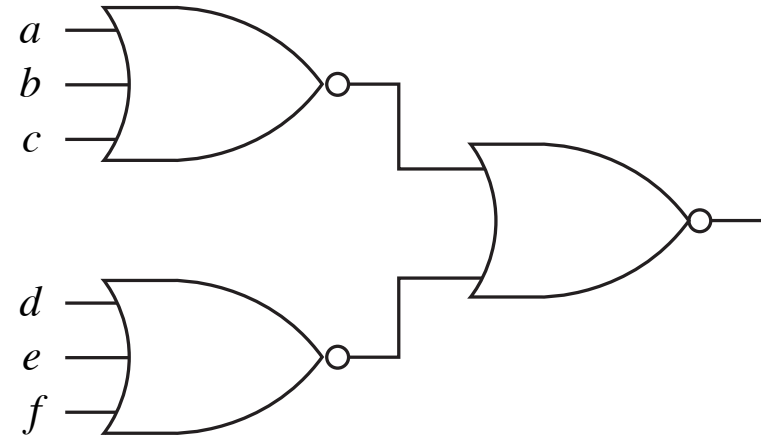
$$(a + a)' = a'$$



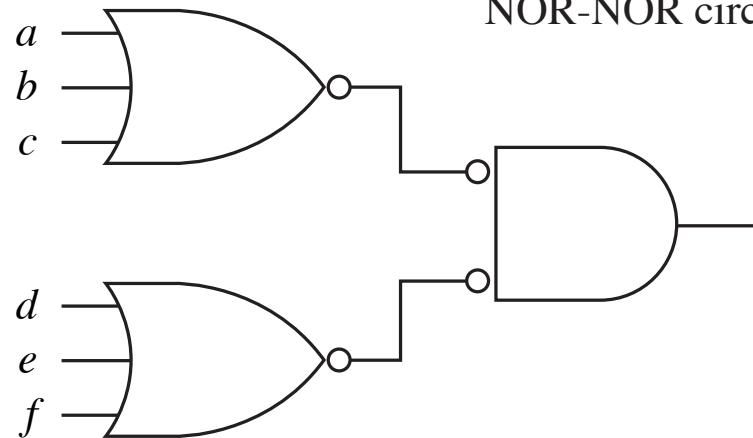
$$(a + b + c)(d + e + f) = [(a + b + c)' + (d + e + f)']'$$



(a) An OR-AND circuit.



(b) The equivalent NOR-NOR circuit.



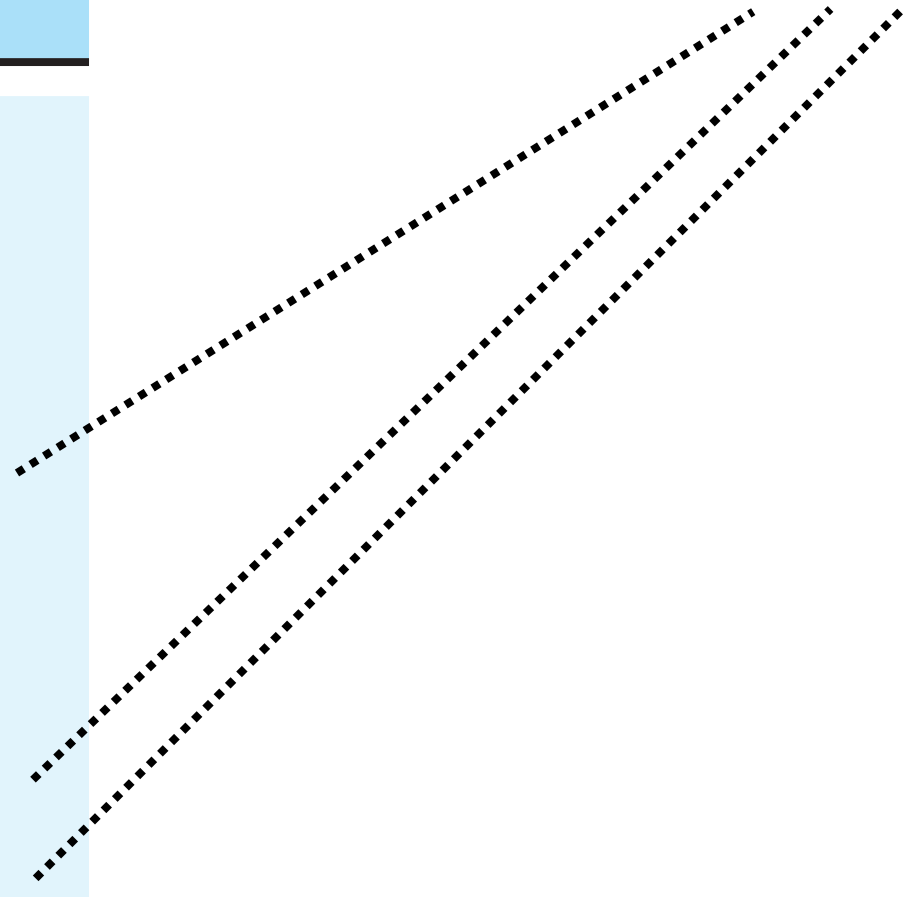
(c) The same NOR-NOR circuit as in part (b).

Canonical expressions

- A *minterm* is a term in an AND-OR expression in which all input variables occur exactly once
- A *canonical expression* is an OR of minterms in which no two identical minterms appear
- A canonical expression is directly related to a truth table because each minterm in the expression represents a 1 in the truth table

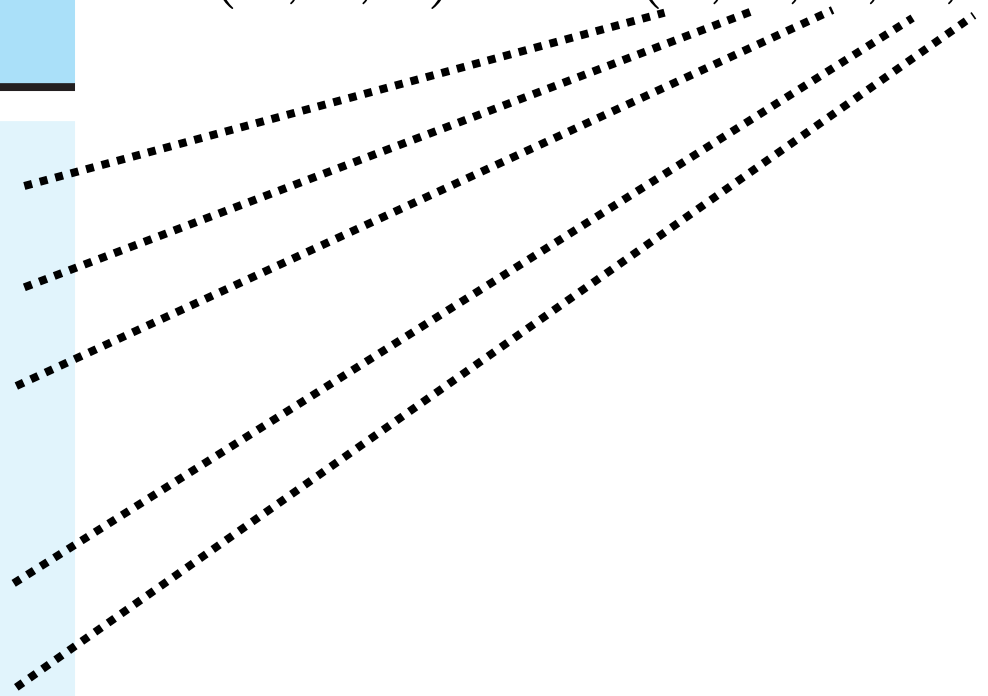
Row (dec)	<i>a</i>	<i>b</i>	<i>c</i>	<i>x</i>
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

$$x(a, b, c) = \Sigma(3, 6, 7)$$



Row (dec)	<i>a</i>	<i>b</i>	<i>c</i>	<i>x</i>
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

$$x(a, b, c) = \Pi(0, 1, 2, 4, 5)$$



Karnaugh maps

- The *distance* between two minterms is the number of places in which they differ
- Two minterms are *adjacent* if the distance between them is one
- A *Karnaugh map* is a truth table arranged so that adjacent cells represent adjacent minterms

		bc			
		00	01	11	10
a	0				
	1				

(a) The Karnaugh map.

bc			
00	01	<u>11</u>	<u>10</u>

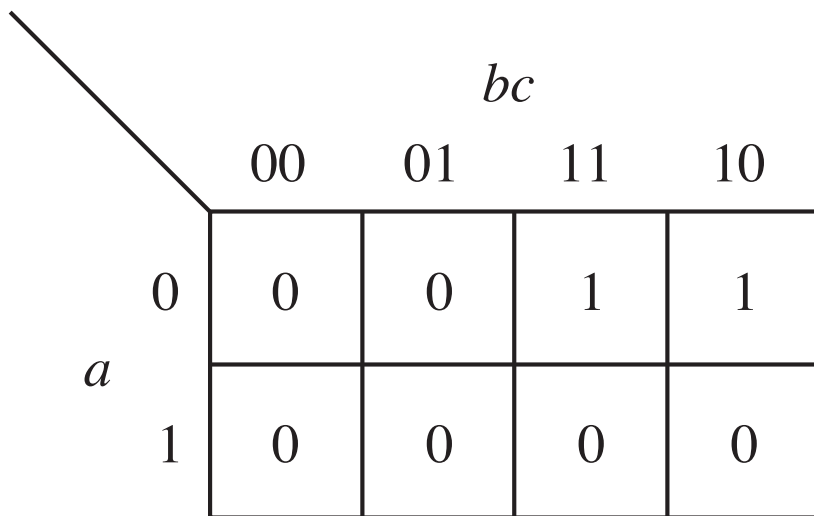
(b) The $b = 1$ region.

		bc			
		<u>00</u>	01	11	<u>10</u>

(c) The $c = 0$ region.

$$x(a, b, c) = a'bc + a'bc'$$

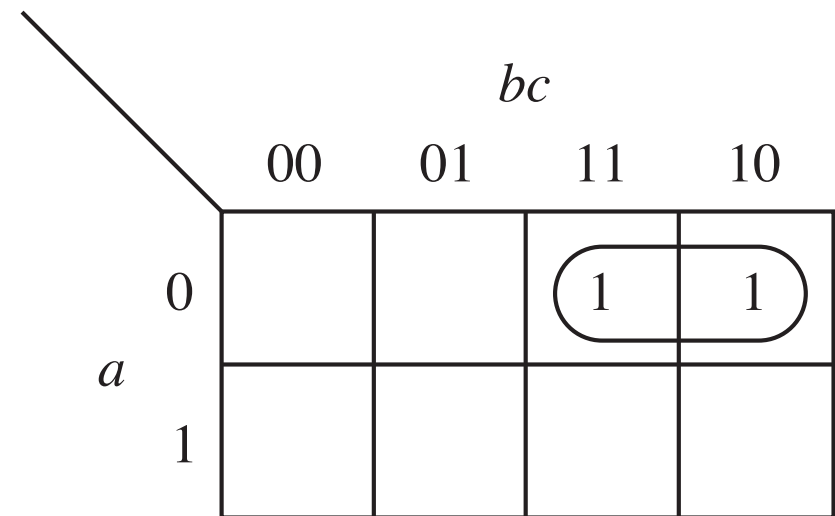
$$x(a, b, c) = a'b$$



A Karnaugh map for the function $x(a, b, c) = a'bc + a'bc'$. The vertical axis is labeled a with values 0 and 1. The horizontal axis is labeled bc with values 00, 01, 11, and 10. The map shows 1s in the cells where $a=0$ and bc is 11 or 10, and 0s elsewhere.

	bc	00	01	11	10
a 0		0	0	1	1
a 1		0	0	0	0

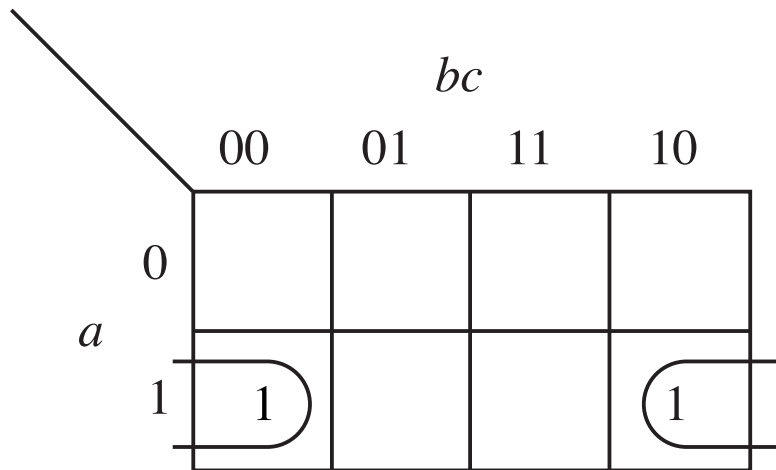
(a) The Karnaugh map.



The same Karnaugh map as in (a), but with a minimization circle drawn around the two 1s in the $a=0$ row, indicating that the function simplifies to $a'b$.

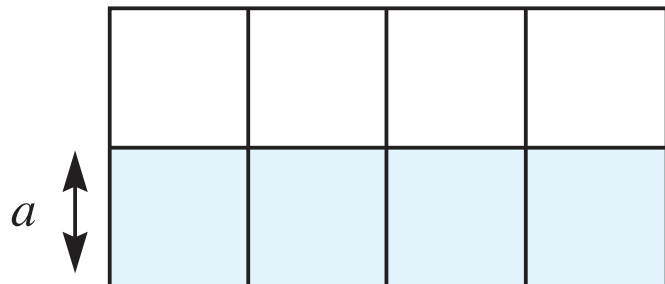
	bc	00	01	11	10
a 0				1	1
a 1					

(b) The minimization.

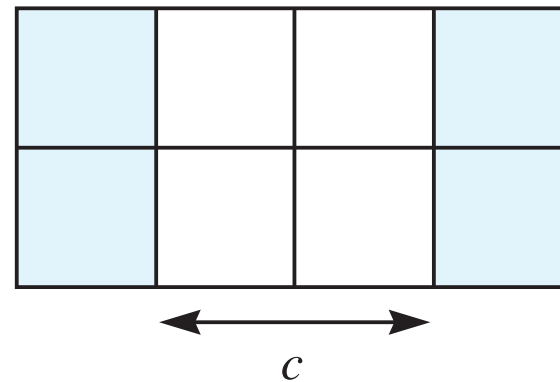


$$\begin{aligned}
 x(a, b, c) &= ab'c' + abc' \\
 &= ac'
 \end{aligned}$$

(a) The Karnaugh map.



(b) Region *a*.



(c) Region *c'*.

		bc			
		00	01	11	10
a	0			1	
	1			1	1

(a) $a'bc + abc = bc$

			<i>b</i>	
			1	
<i>a</i>			1	1
			<i>c</i>	

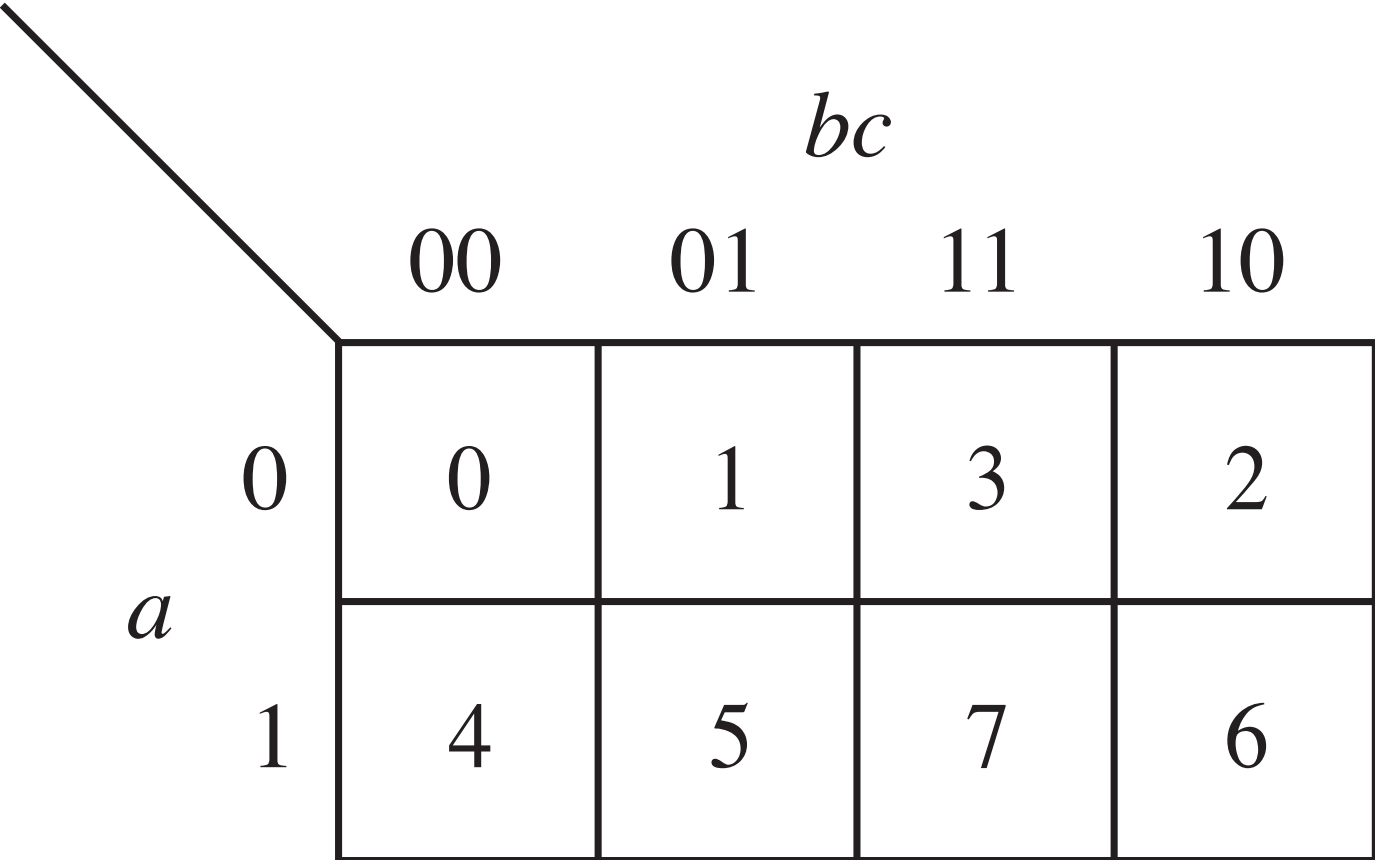
(b) $abc + abc' = ab$

			1	
			1	1

(c) $x = bc + ab$

$$x(a, b, c) = a'bc + abc + abc'$$

$$= bc + ab$$

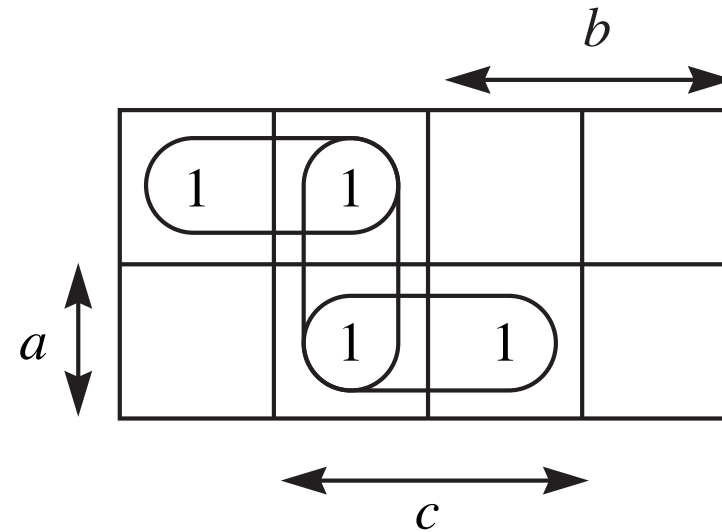


		bc			
		00	01	11	10
a	0	0	1	3	2
	1	4	5	7	6

a

	<i>bc</i>			
	00	01	11	10
0	1	1		
1		1	1	

(a) A bad strategy.



(b) The result of the bad strategy.

1	1		
	1	1	

(c) The correct minimization

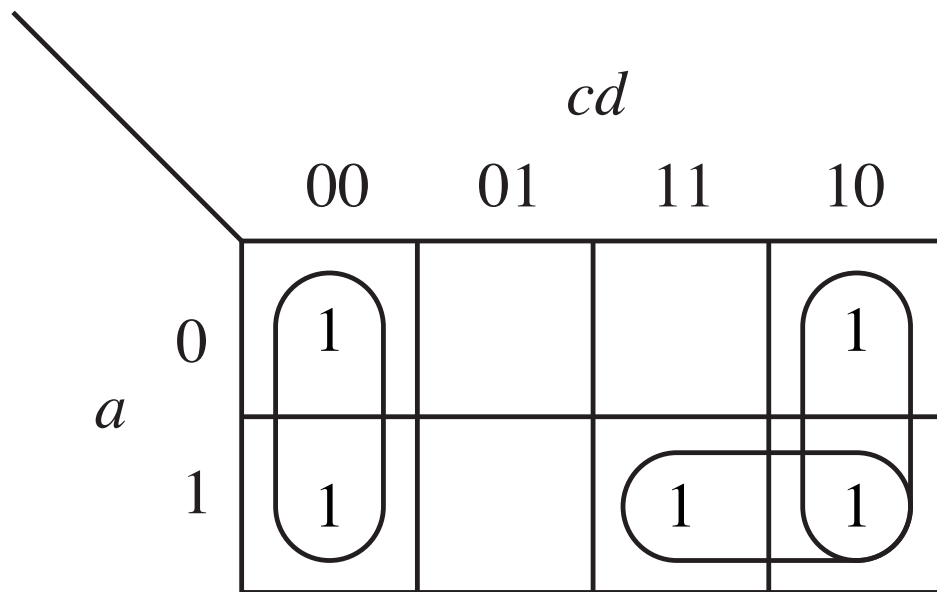
$$\begin{aligned}
 x(a, b, c) &= \Sigma(0, 1, 5, 7) \\
 &= a'b' + ac
 \end{aligned}$$

$$x(a, b, c) = \Sigma(0, 2, 4, 6, 7)$$

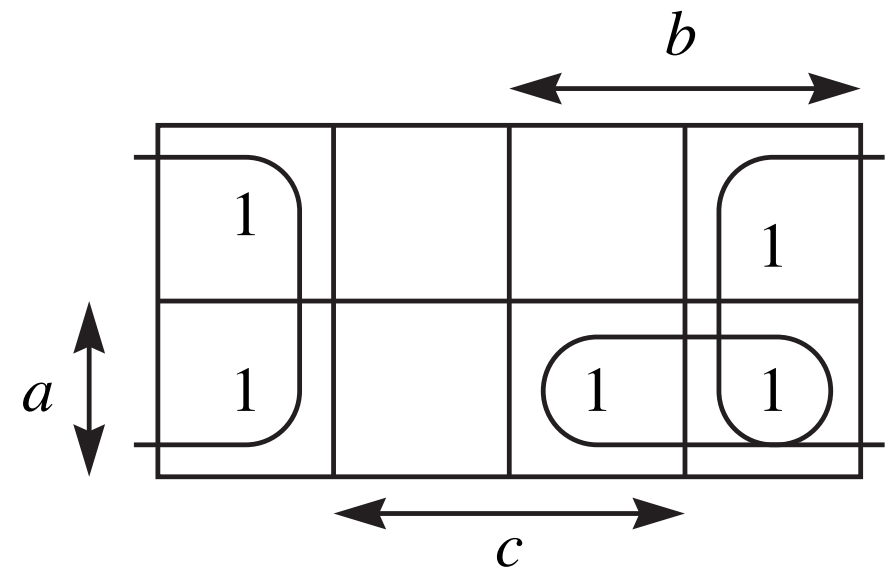
$$= b'c' + bc' + ab$$

$$x(a, b, c) = \Sigma(0, 2, 4, 6, 7)$$

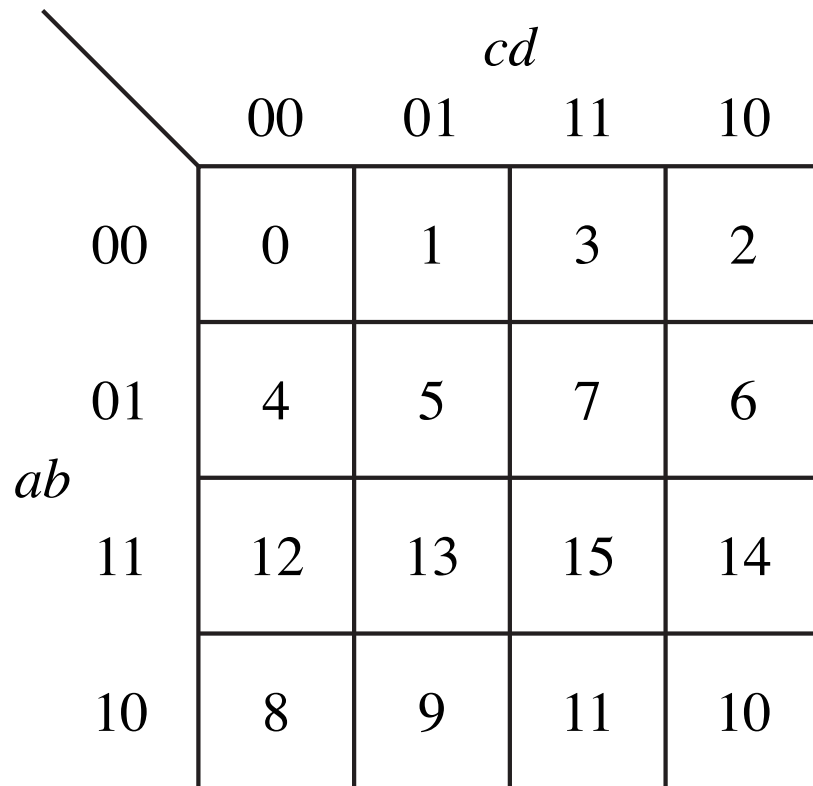
$$= c' + ab$$



(a) An incorrect minimization.

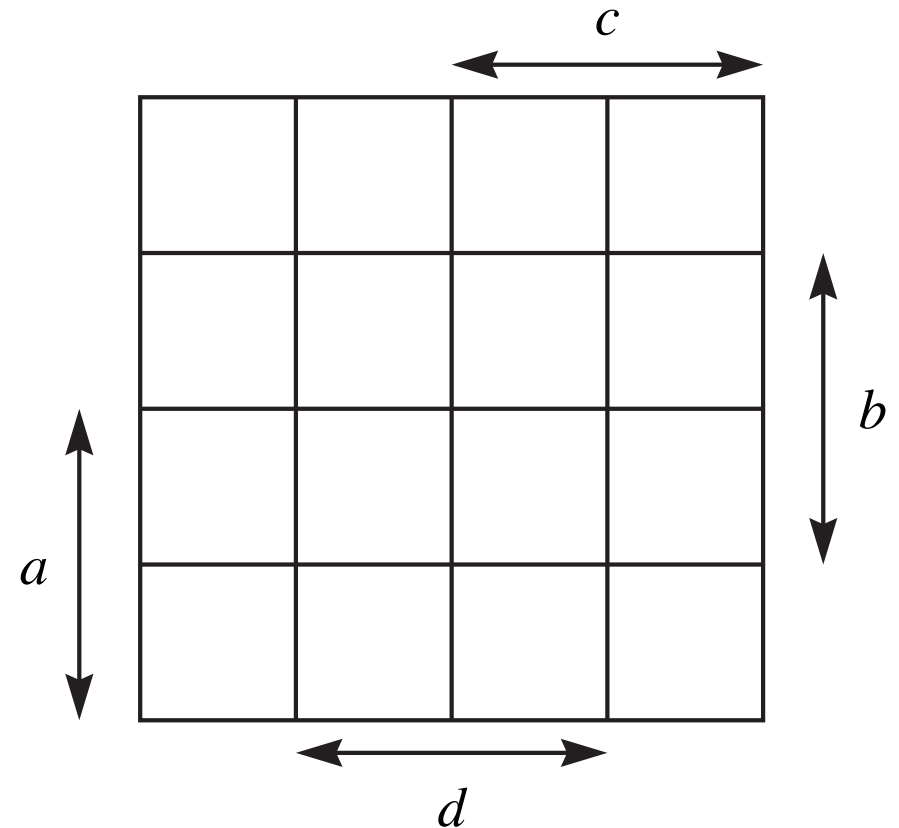


(b) The correct minimization.



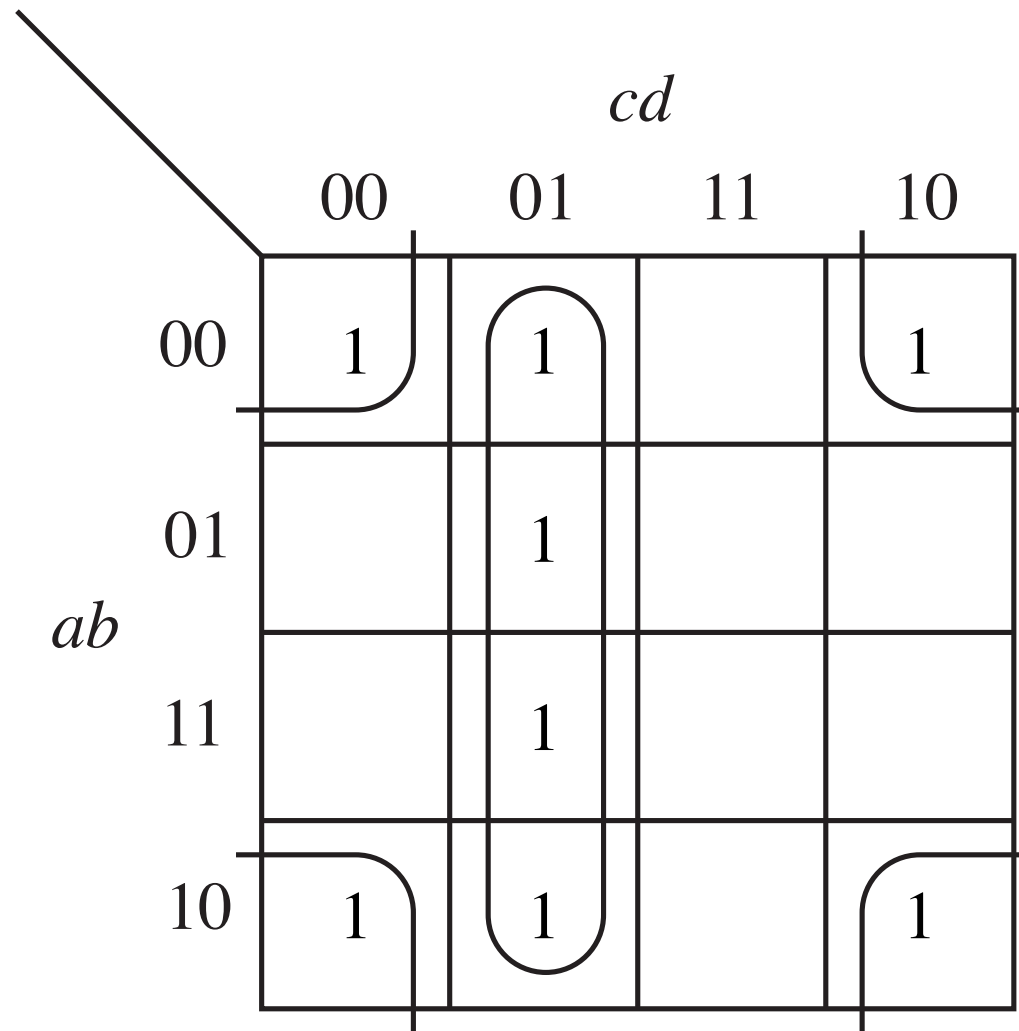
		<i>cd</i>			
		00	01	11	10
<i>ab</i>	00	0	1	3	2
	01	4	5	7	6
	11	12	13	15	14
	10	8	9	11	10

(a) Decimal labels for the minterms in the Karnaugh map.

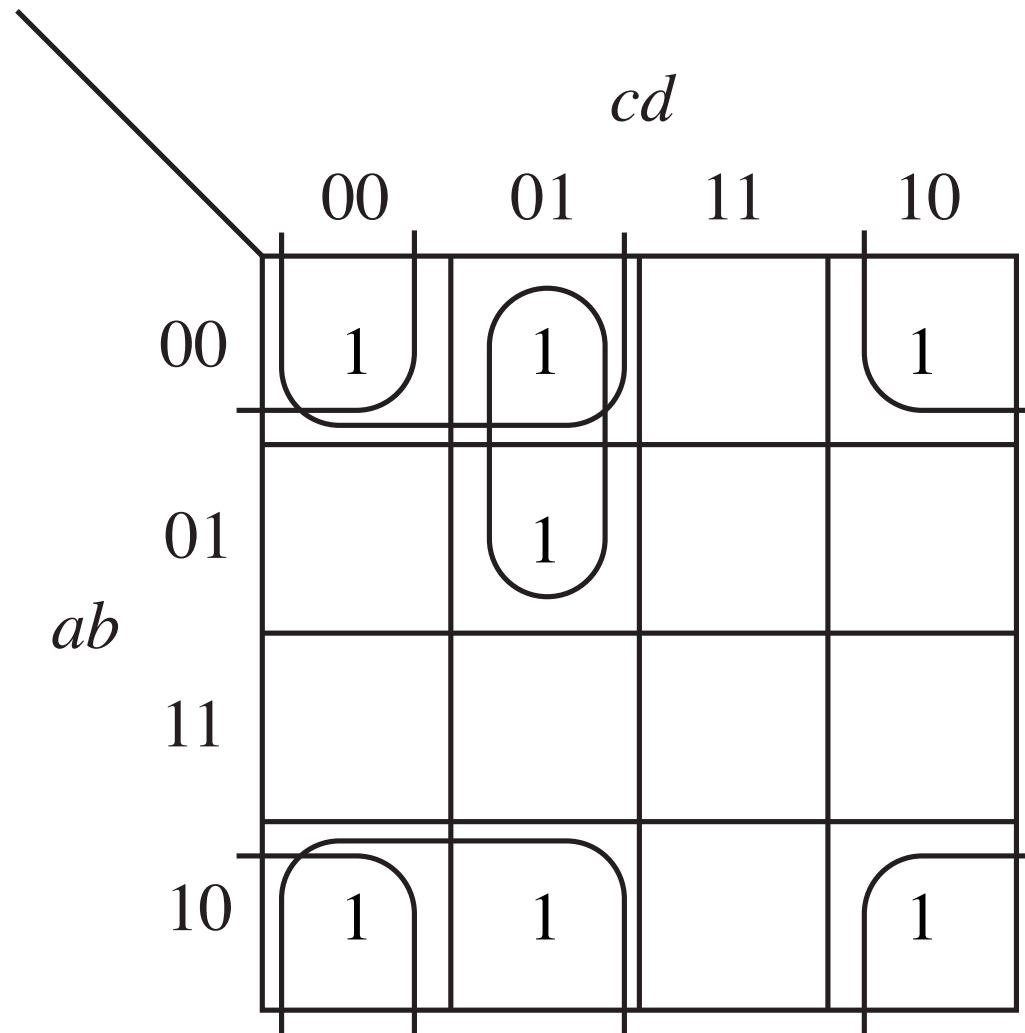


(b) The regions where the variables are 1.

$$x(a, b, c, d) = c'd + b'd'$$

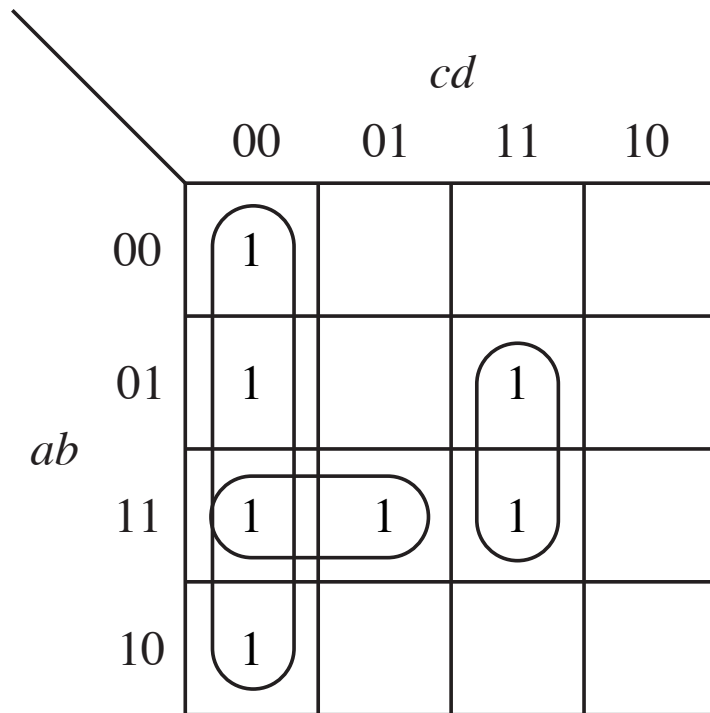


$$x(a, b, c, d) = a'c'd + b'c' + b'd'$$

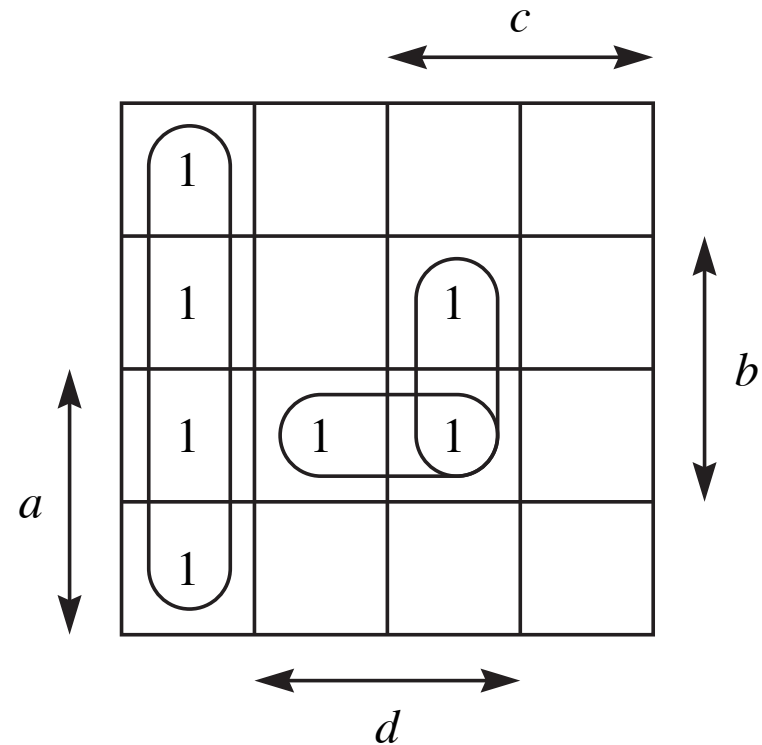


$$x(a, b, c, d) = c'd' + bcd + abc'$$

$$x(a, b, c, d) = c'd' + bcd + abd$$

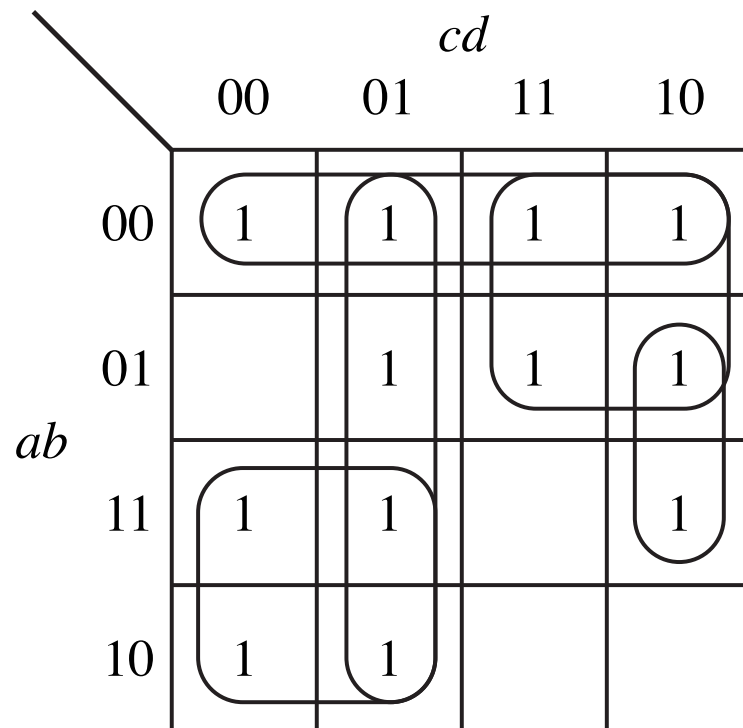


(a) One possible minimization.



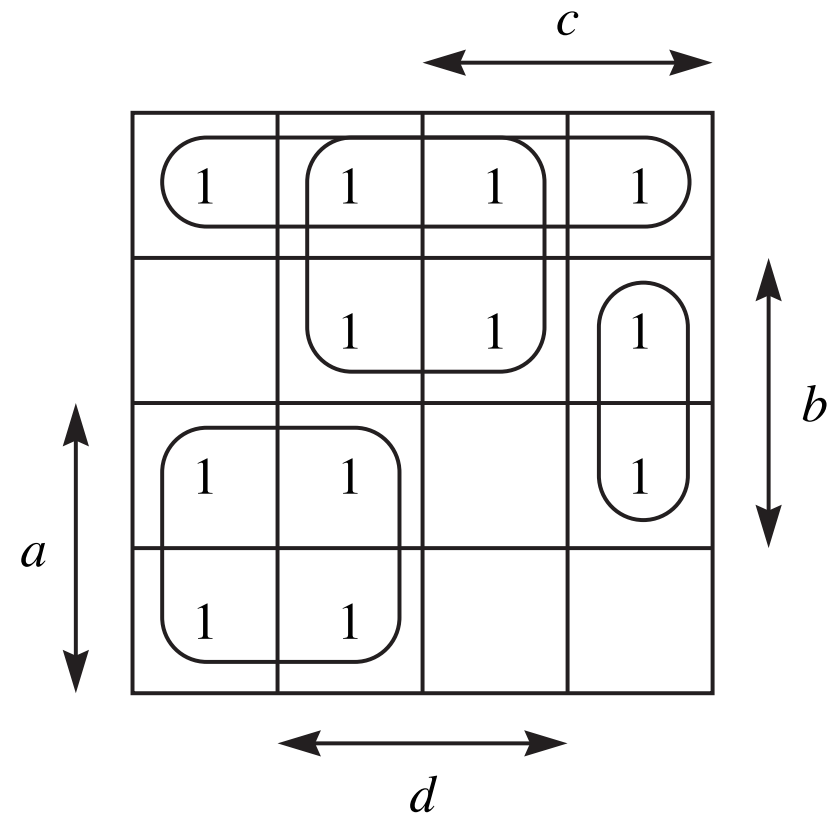
(b) A different minimization.

$$ac' + a'c + c'd + a'b' + bcd'$$



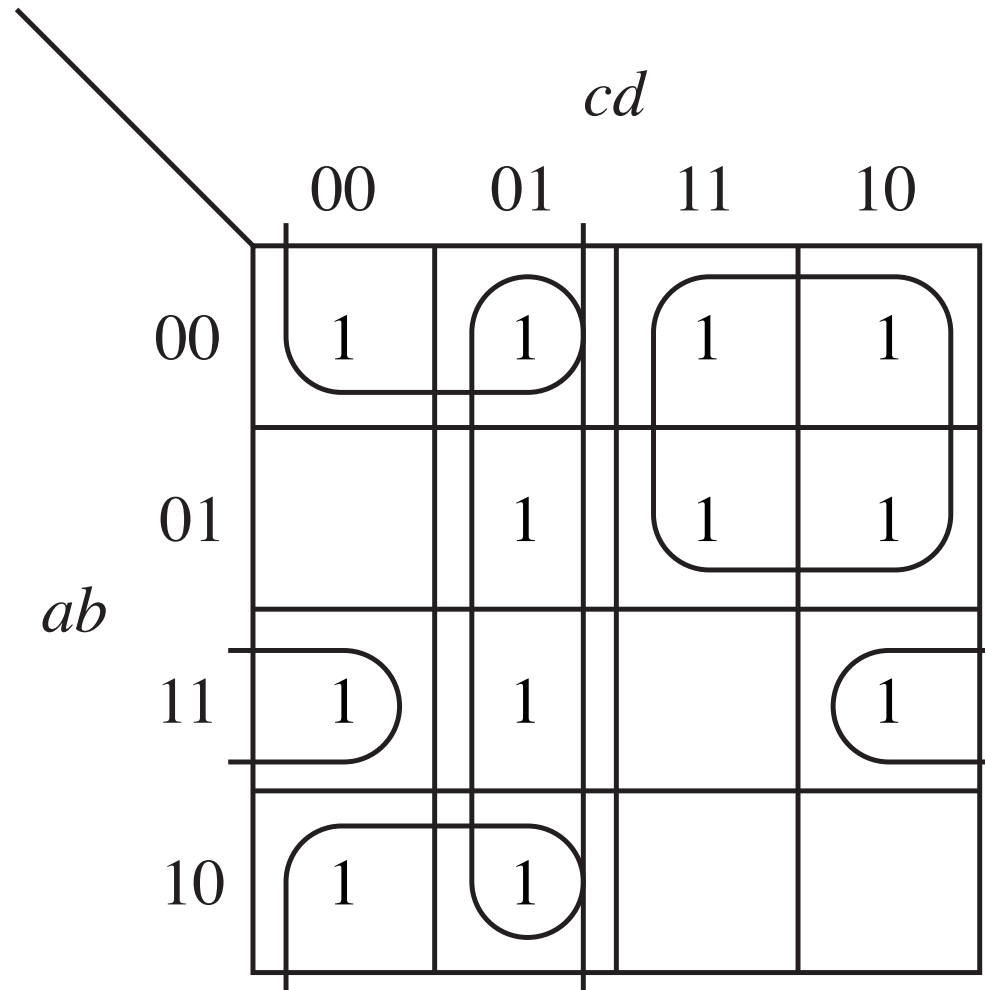
(a) A plausible but incorrect minimization.

$$ac' + a'd + a'b' + bcd'$$



(b) A correct minimization.

$$a'c + b'c' + c'd + abd'$$



Dual Karnaugh maps

- To minimize a function in an OR-AND expression minimize the complement of the function in the AND-OR expression
- Use $x = (x')'$ and De Morgan's law

		1	
		1	1

$$x = bc + ab$$

		<i>bc</i>			
		00	01	11	10
<i>a</i>	0	1	1		1
	1	1	1		

$$x' = b' + a'c'$$

$$x = (x')'$$

$$= (b' + a'c')'$$

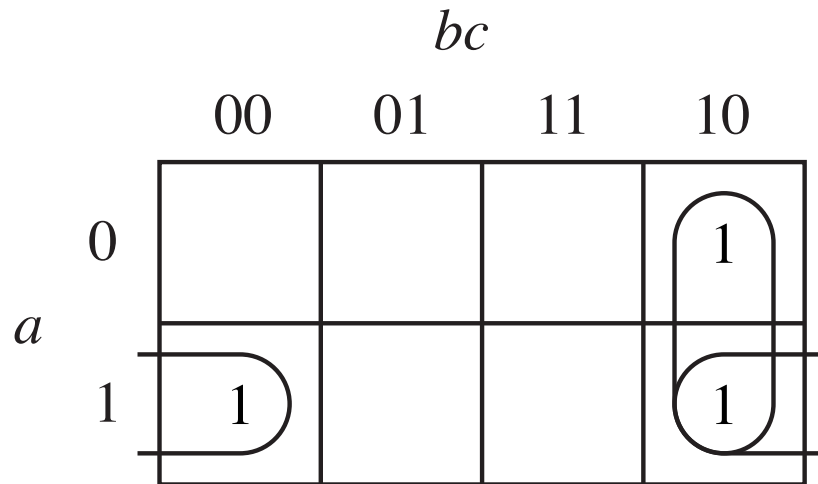
$$= b(a + c)$$

Don't-care conditions

- If an input combination is never expected to be present, you can choose to make it 0 or 1, whichever will better minimize the circuit
- A don't care condition is shown as an X in a Karnaugh map

$$x(a, b, c) = \Sigma(2, 4, 6)$$

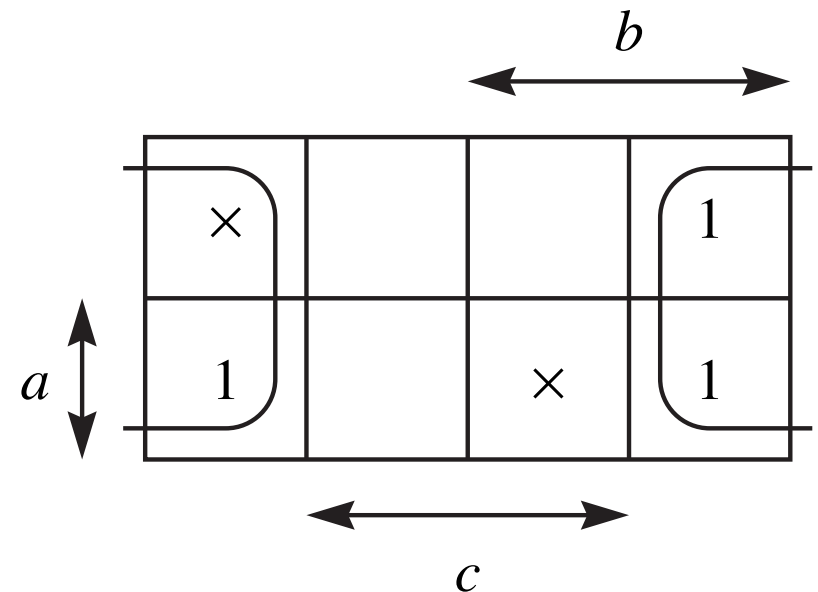
$$= bc' + ac'$$



(a) Minimizing a function without don't-care conditions.

$$x(a, b, c) = \Sigma(2, 4, 6) + d(0, 7)$$

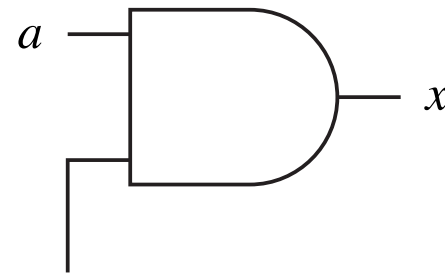
$$= c'$$



(b) Minimizing the same function with don't-care conditions.

Enable lines

- An enable line to a combinational device turns the device on or off
 - ▶ If enable = 0 the output is 0 regardless of any other inputs
 - ▶ If enable = 1 the device performs its function with the output depending on the other inputs



Enable

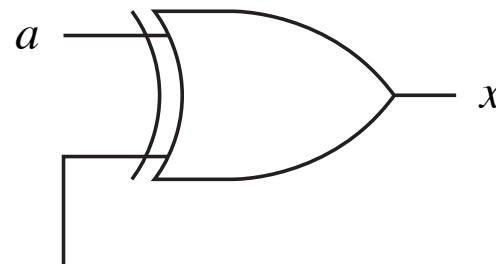
(a) Logic diagram of enable gate.

Enable = 1	
<i>a</i>	<i>x</i>
0	0
1	1

(b) Truth table with the device turned on.

Enable = 0	
<i>a</i>	<i>x</i>
0	0
1	0

(c) Truth table with the device turned off.



Invert

(a) Logic diagram of the selective inverter.

Invert = 1	
a	x
0	1
1	0

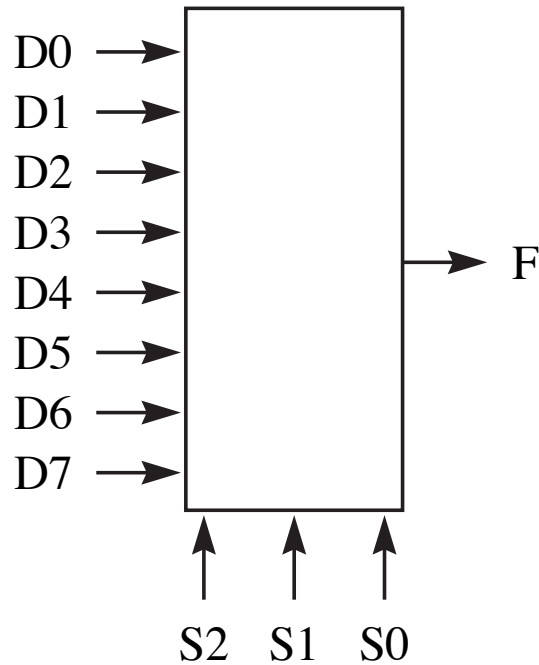
(b) Truth table with the inverter turned on.

Invert = 0	
a	x
0	0
1	1

(c) Truth table with the inverter turned off.

Multiplexer

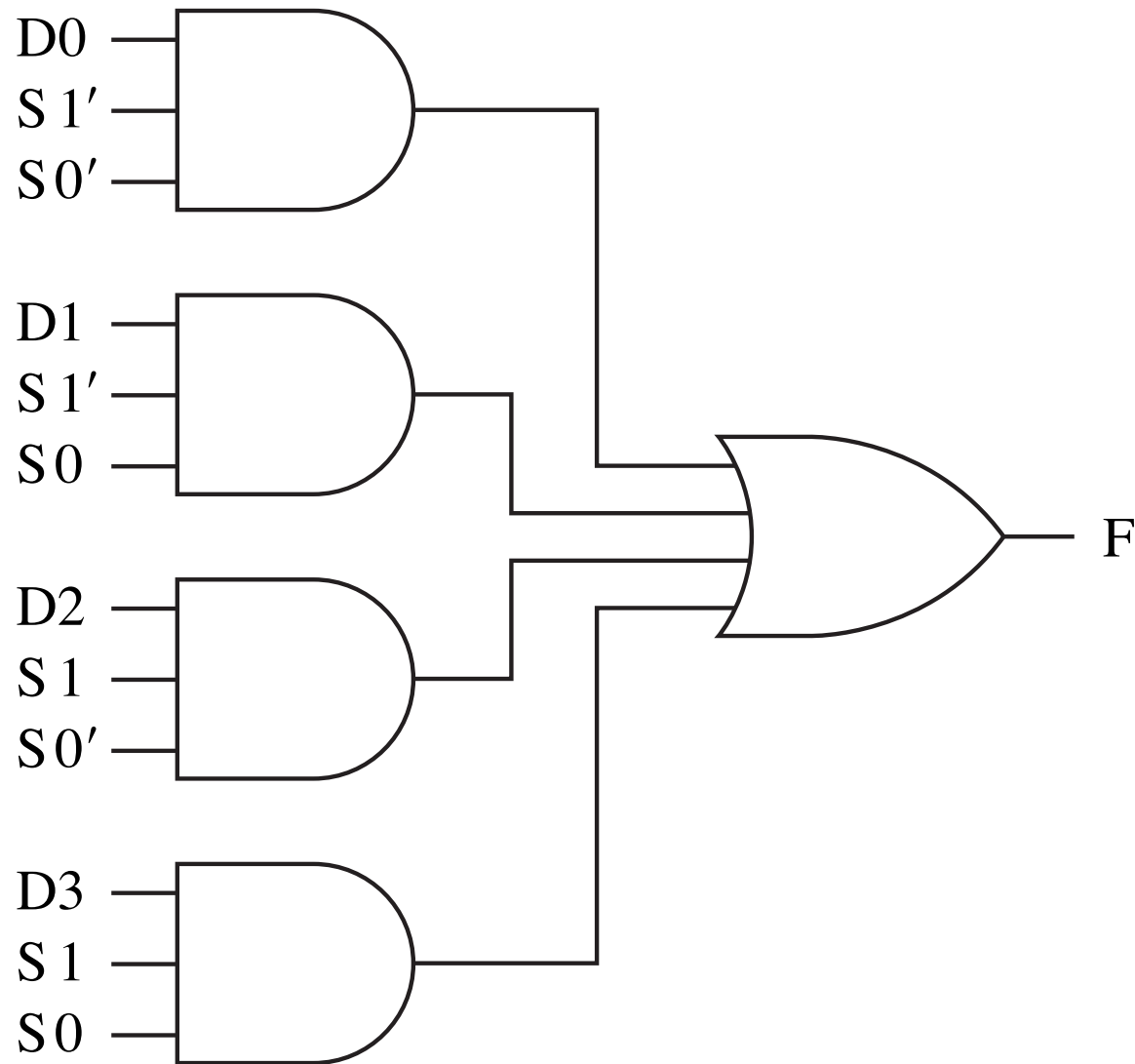
- A multiplexer selects one of several data inputs to be routed to a single data output
- Control lines determine the particular data input to be passed through



(a) Block diagram.

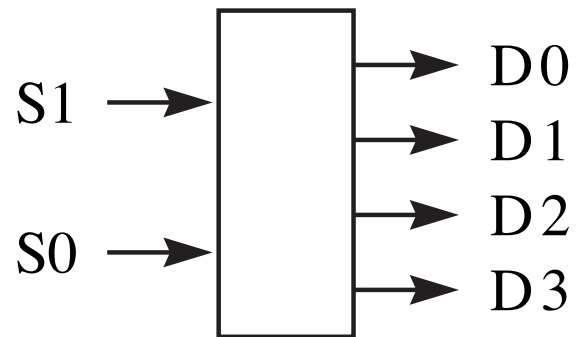
S2	S1	S0	F
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7

(b) Truth table.



Binary decoder

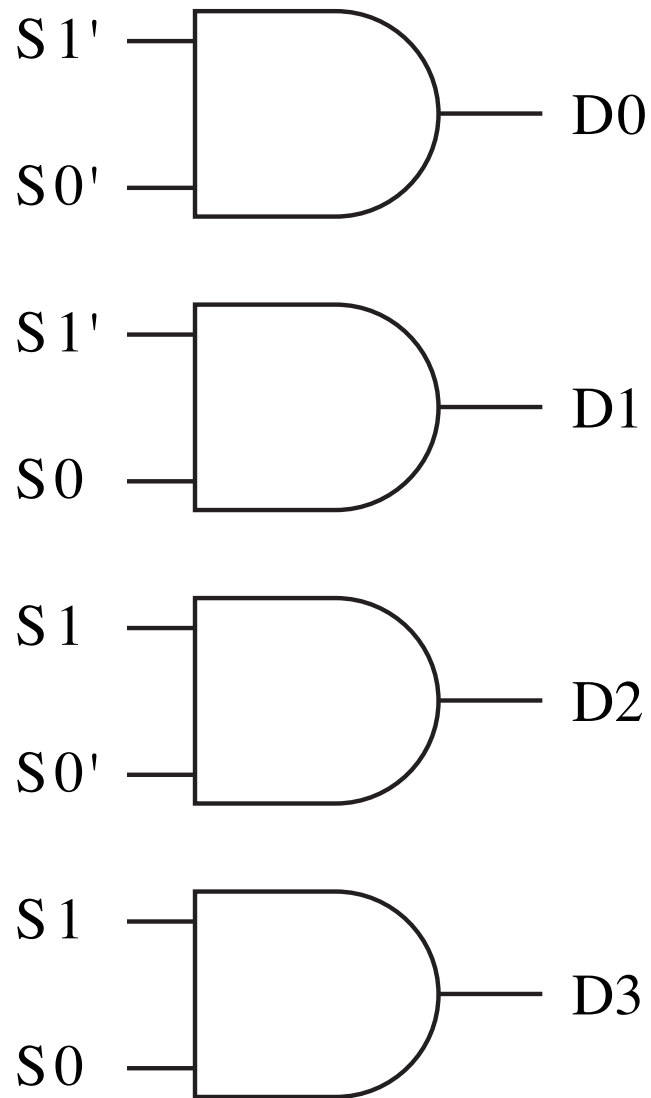
- A decoder takes a binary number as input and sets one of the data output lines to 1 and the rest to 0
- The data line that is set to 1 depends on the value of the binary number that is input

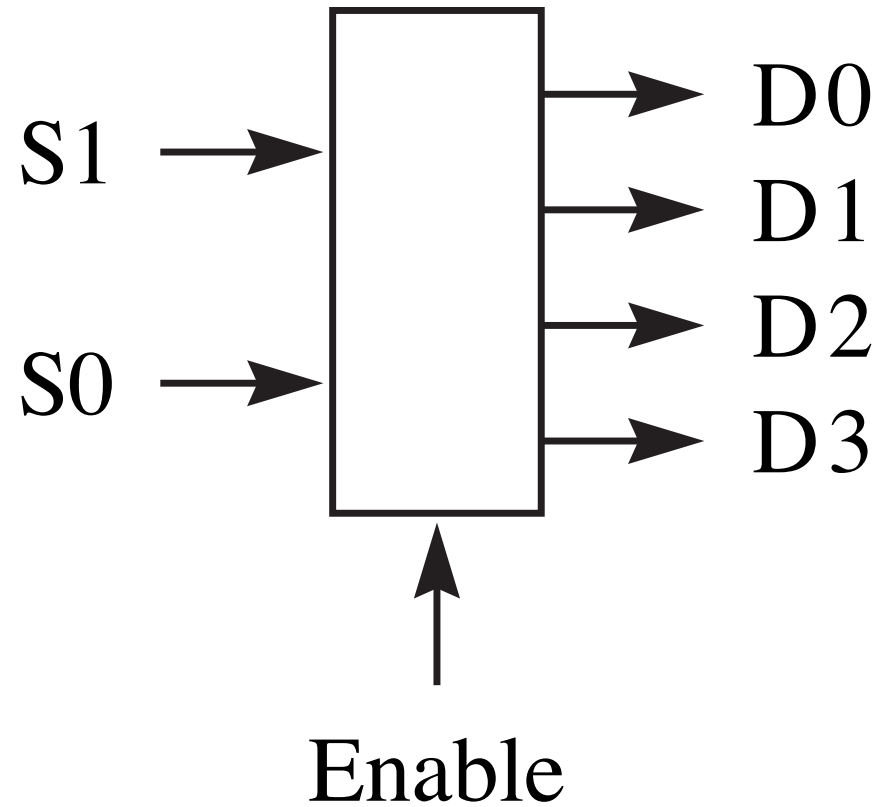


(a) Block diagram.

S1	S0	D0	D1	D2	D3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

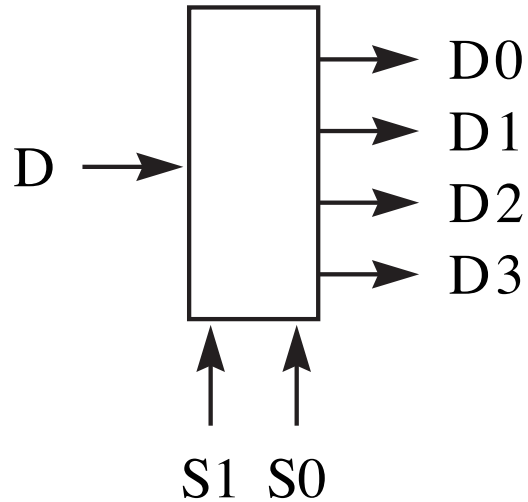
(b) Truth table.





Demultiplexer

- A demultiplexer routes a single input value to one of several output lines
- Control lines determine the data output line to which the input gets routed



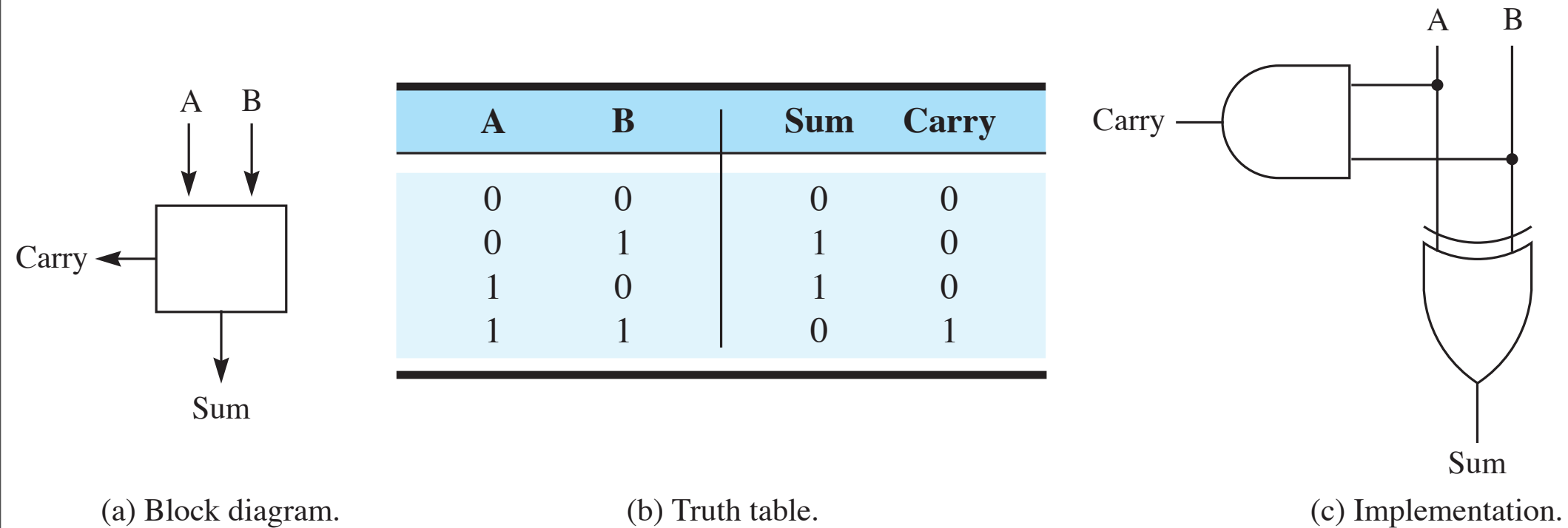
(a) Block diagram.

S1	S0	D0	D1	D2	D3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

(b) Truth table.

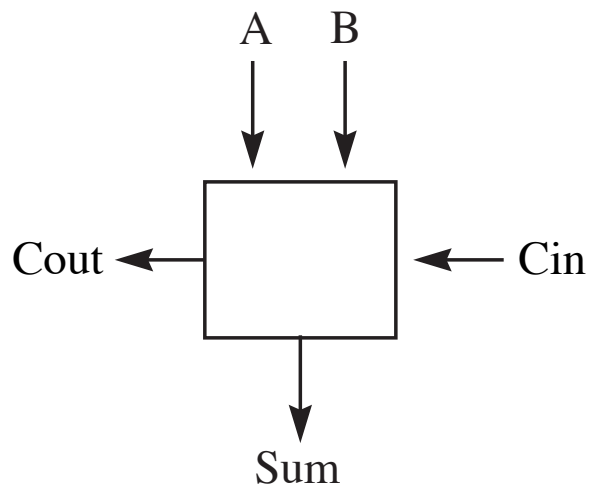
Half adder

- The half adder adds the right-most two bits of a binary number
- Inputs: The two bits
- Outputs: The sum bit and the carry bit



Full adder

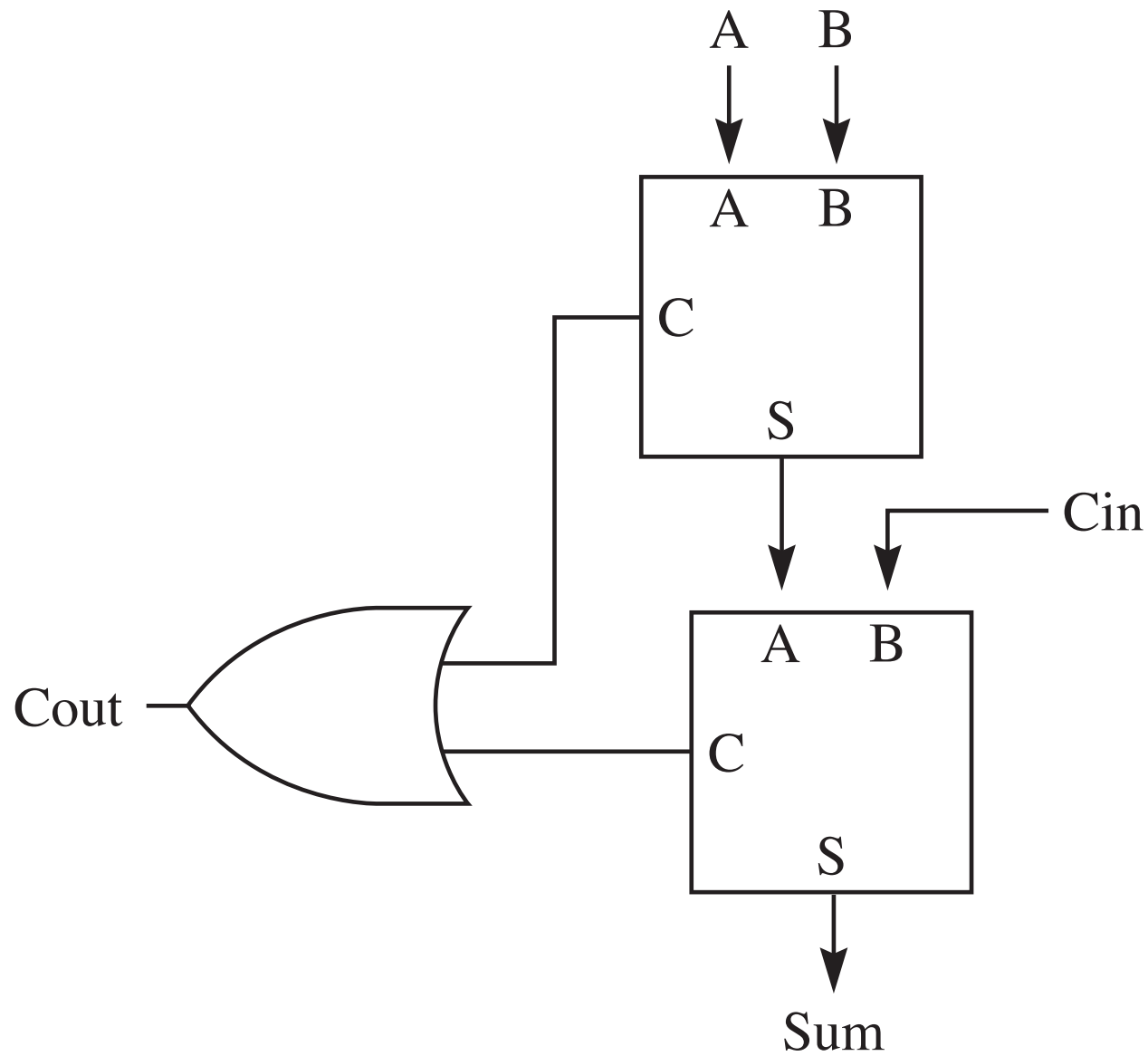
- The full adder adds one column of a binary number
- Inputs: The two bits for that column and the carry bit from the previous column
- Outputs: The sum bit and the carry bit for the next column



(a) Block diagram.

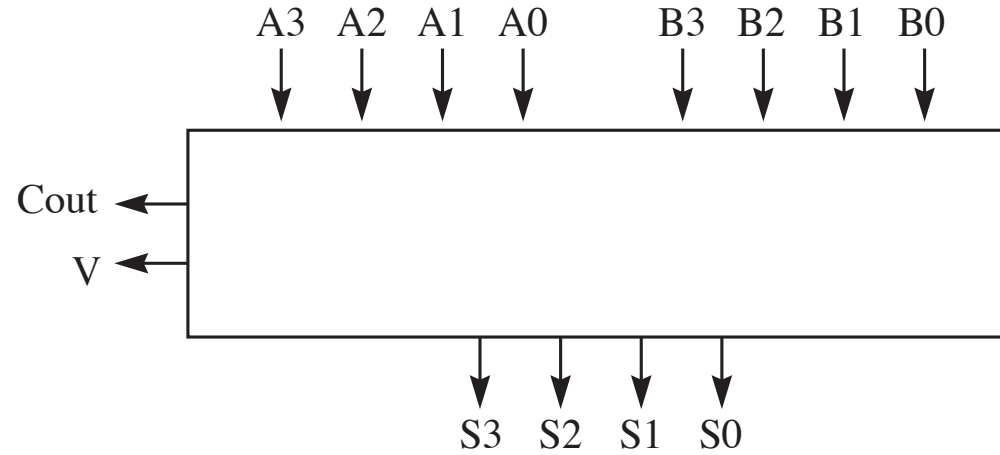
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(b) Truth table.

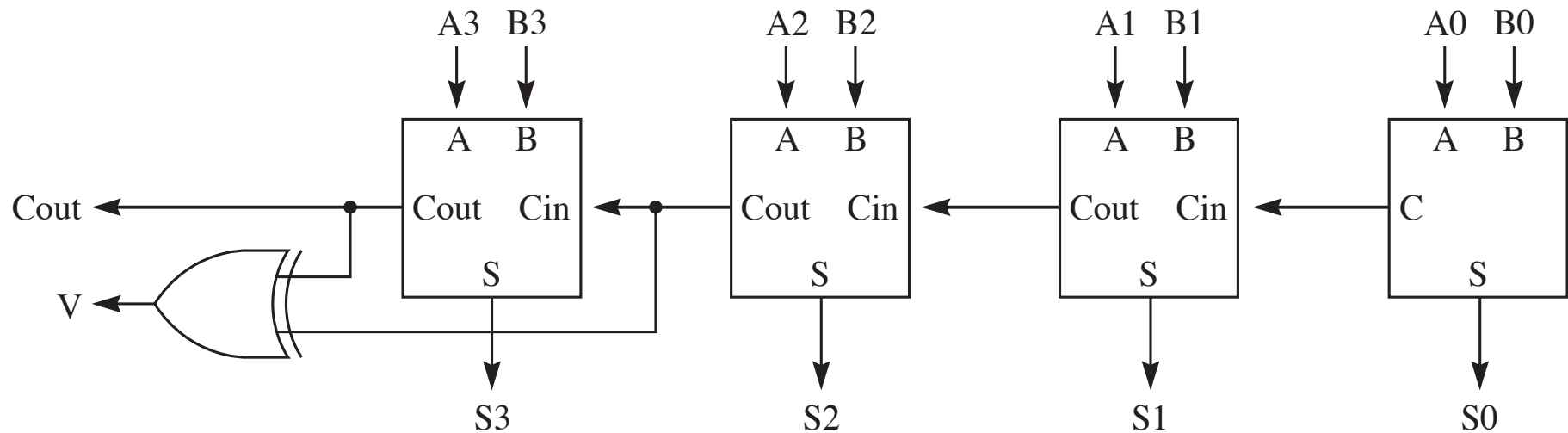


Ripple-carry adder

- The ripple-carry adder adds two n -bit binary numbers
- Inputs: The two n -bit binary numbers to be added
- Outputs: The n -bit sum, the C bit for the carry out, and the V bit for signed integer overflow



(a) Block diagram.



(b) Implementation. © 2010 Jones and Bartlett Publishers, LLC (www.jbpub.com)

Computing the V bit

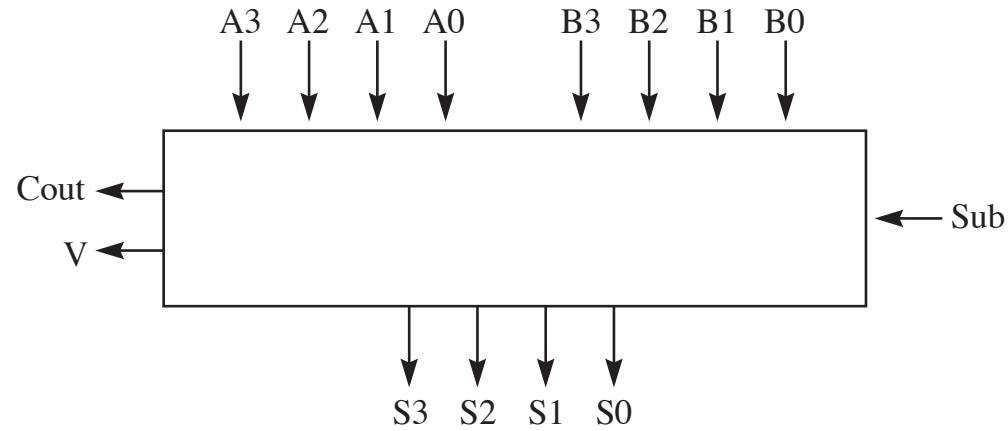
- You can only get an overflow in one of two cases
 - ▶ A and B are both positive, and the result is negative
 - ▶ A and B are both negative, and the result is positive

Adder/subtractor

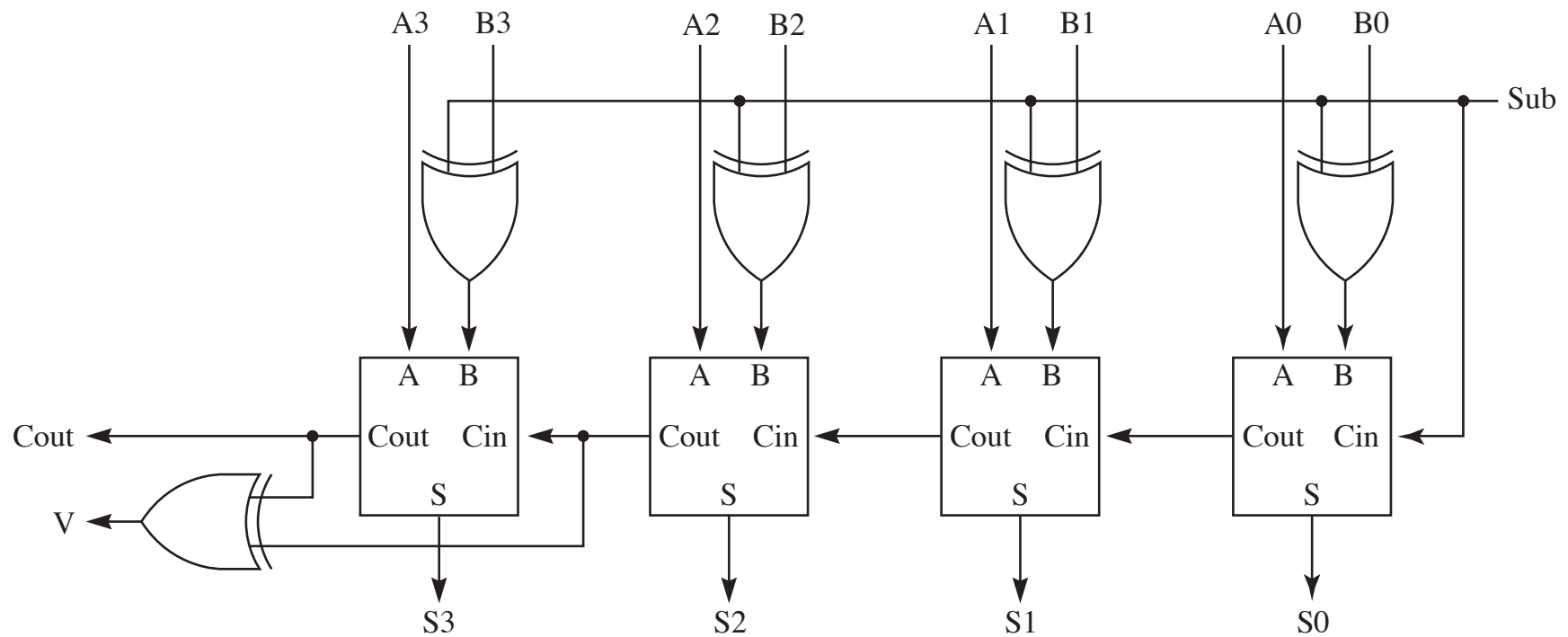
- Based on the relation

$$\text{NEG } x = 1 + \text{NOT } x$$

- XOR gates act as selective inverters
- $A - B = A + (-B)$

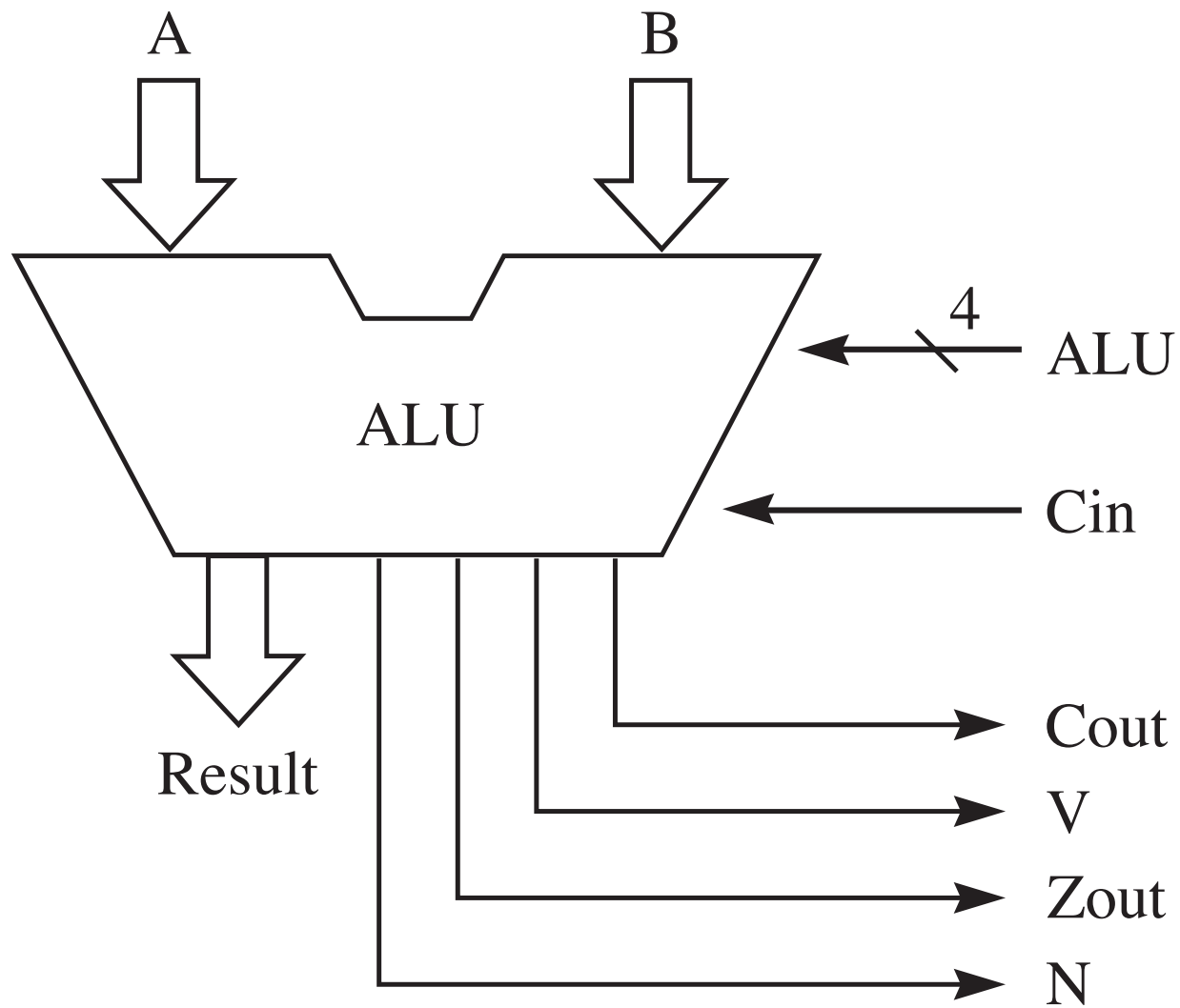


(a) Block diagram.



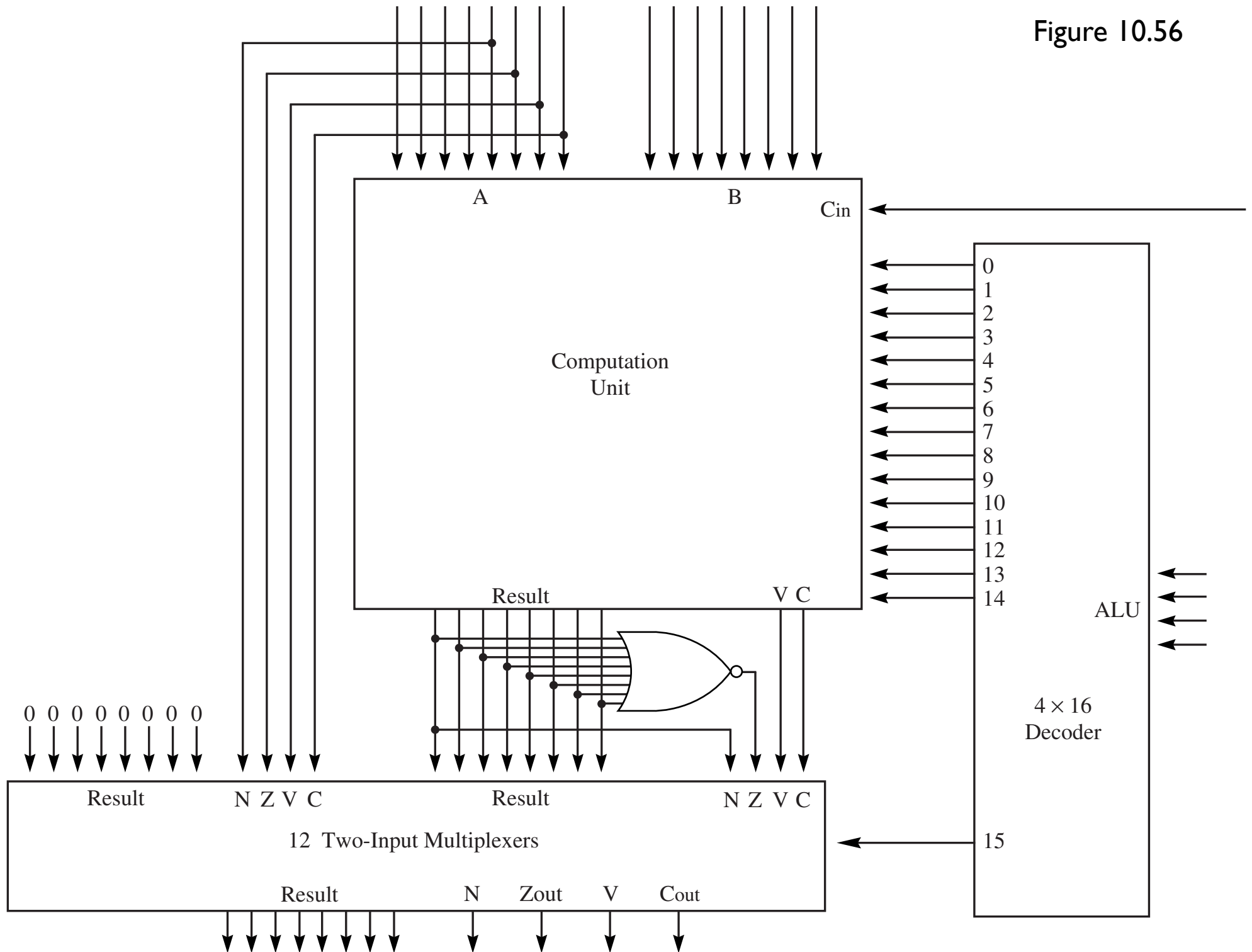
Arithmetic Logic Unit (ALU)

- Performs 16 different functions
- Inputs: Two n -bit binary numbers, four control lines that determine which function will be executed, and one carry input line
- Outputs: The n -bit result, the NZVC bits



ALU control			Status bits			
(bin)	(dec)	Result	N	Zout	V	Cout
0000	0	A	N	Z	0	0
0001	1	A plus B	N	Z	V	C
0010	2	A plus B plus Cin	N	Z	V	C
0011	3	A plus \bar{B} plus 1	N	Z	V	C
0100	4	A plus \bar{B} plus Cin	N	Z	V	C
0101	5	$A \cdot B$	N	Z	0	0
0110	6	$\overline{A \cdot B}$	N	Z	0	0
0111	7	$A + B$	N	Z	0	0
1000	8	$\overline{A + B}$	N	Z	0	0
1001	9	$A \oplus B$	N	Z	0	0
1010	10	\bar{A}	N	Z	0	0
1011	11	ASL A	N	Z	V	C
1100	12	ROL A	N	Z	0	C
1101	13	ASR A	N	Z	0	C
1110	14	ROR A	N	Z	0	C
1111	15	0	A<4>	A<5>	A<6>	A<7>

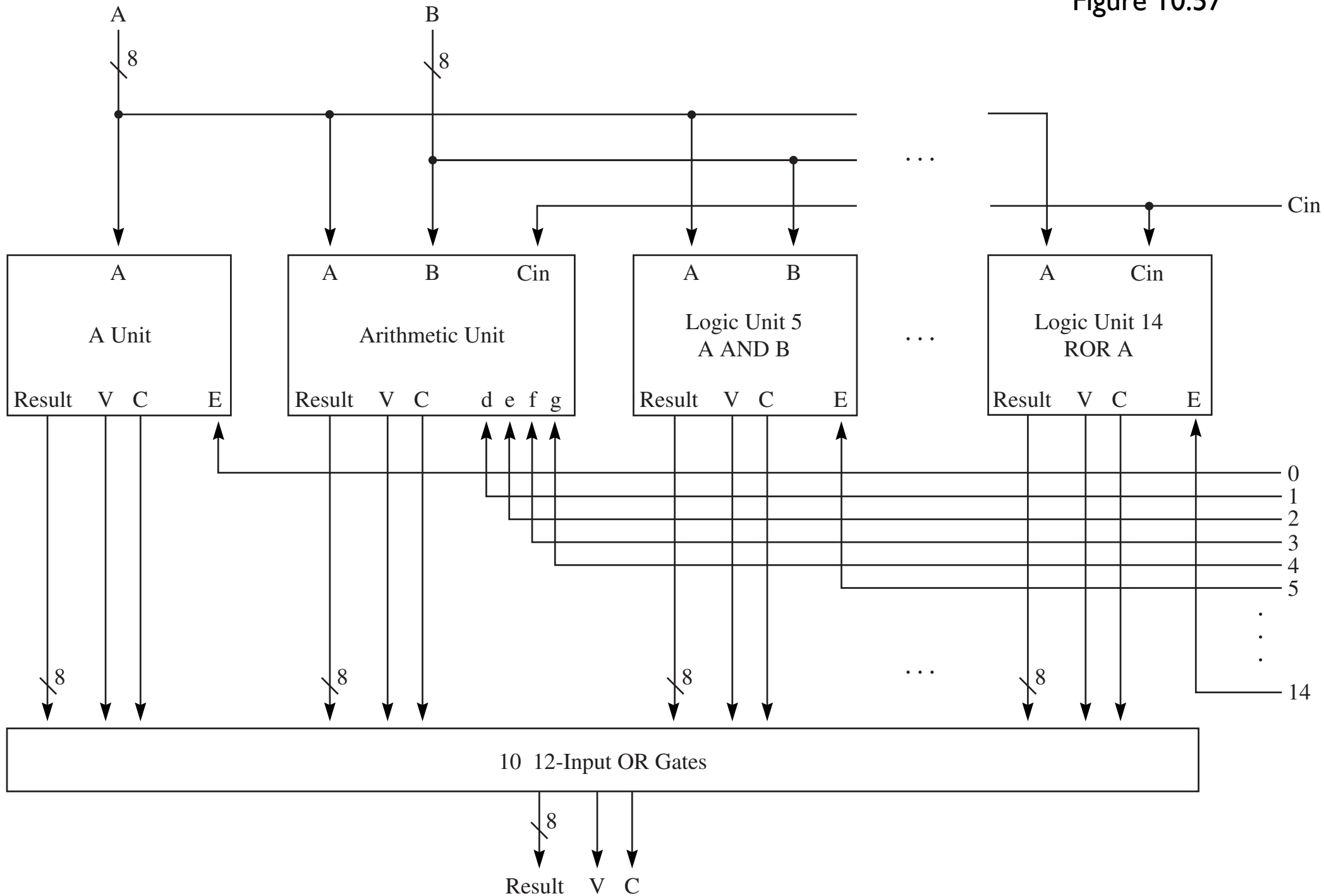
Figure 10.56

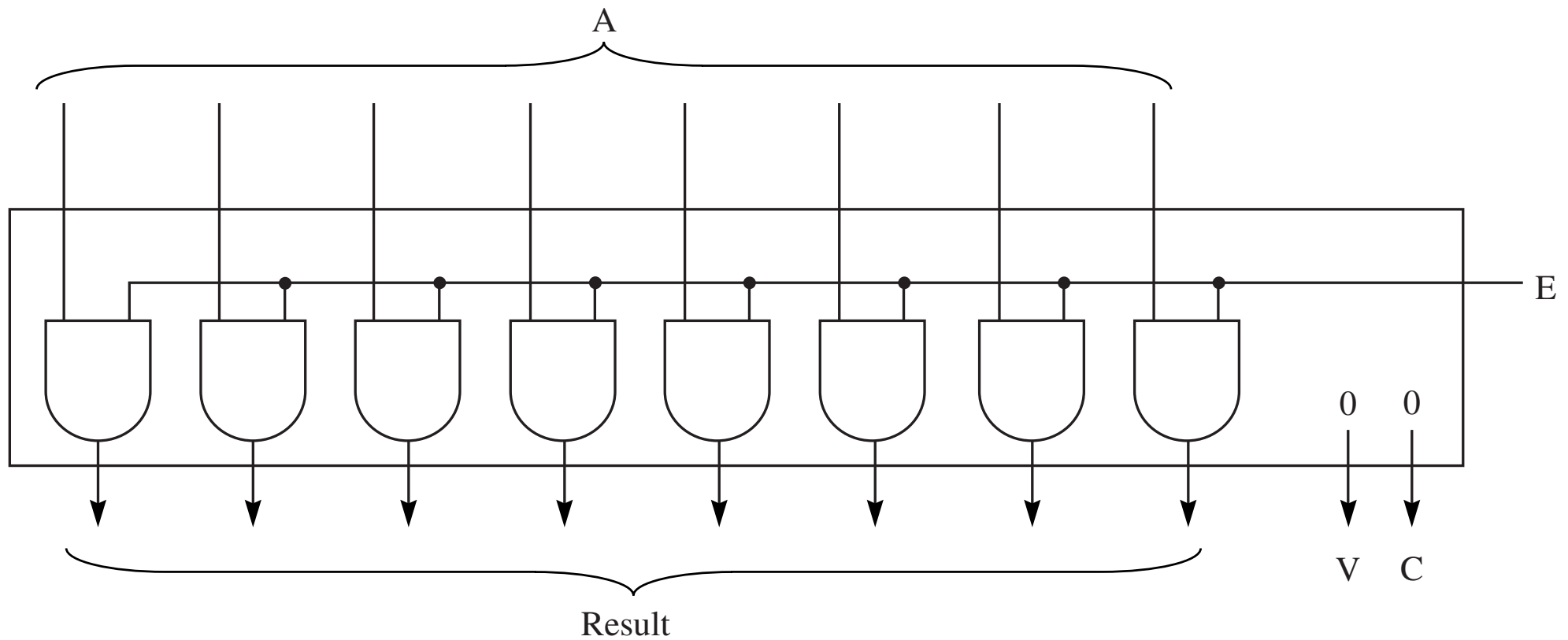


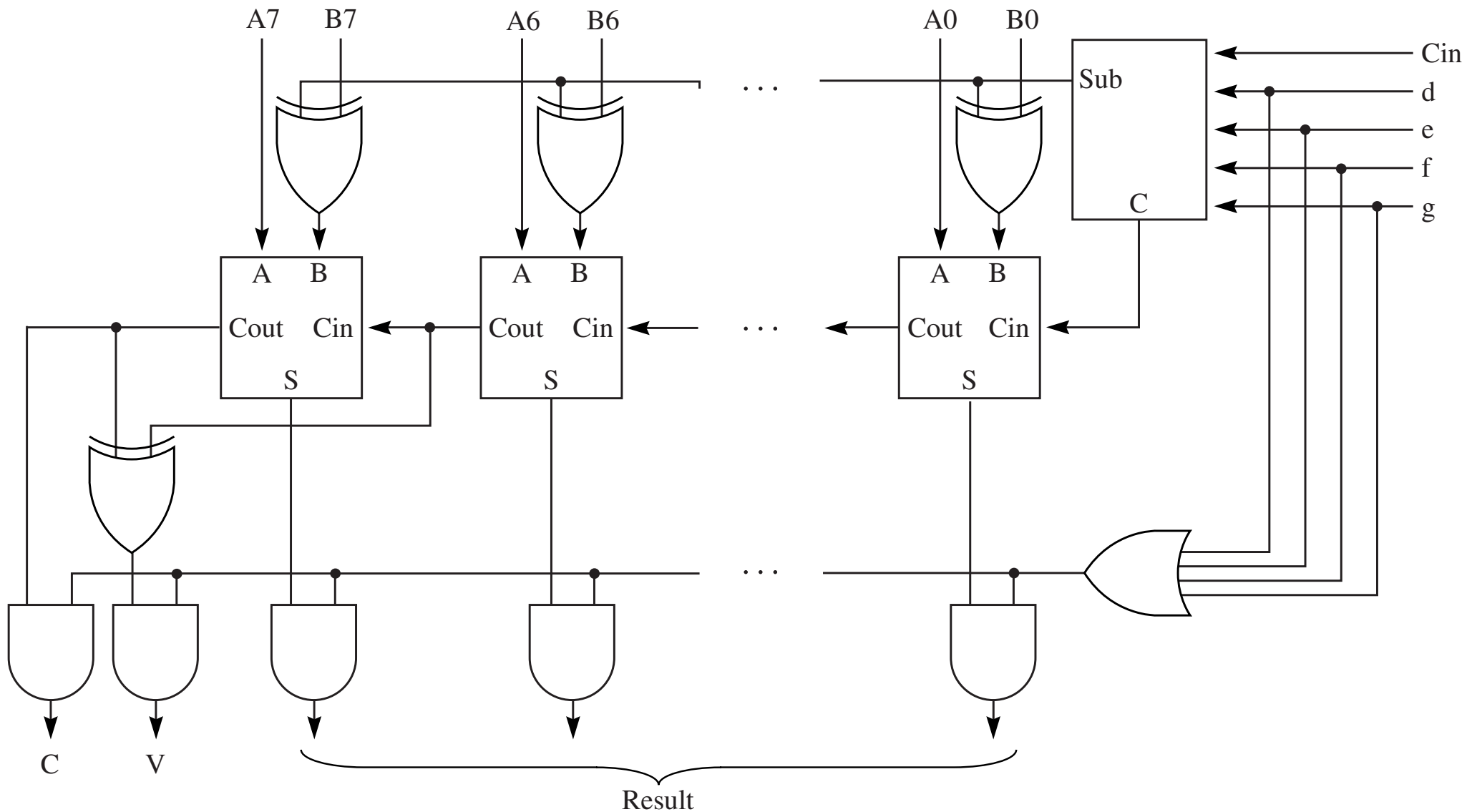
The multiplexer of Figure 10.56

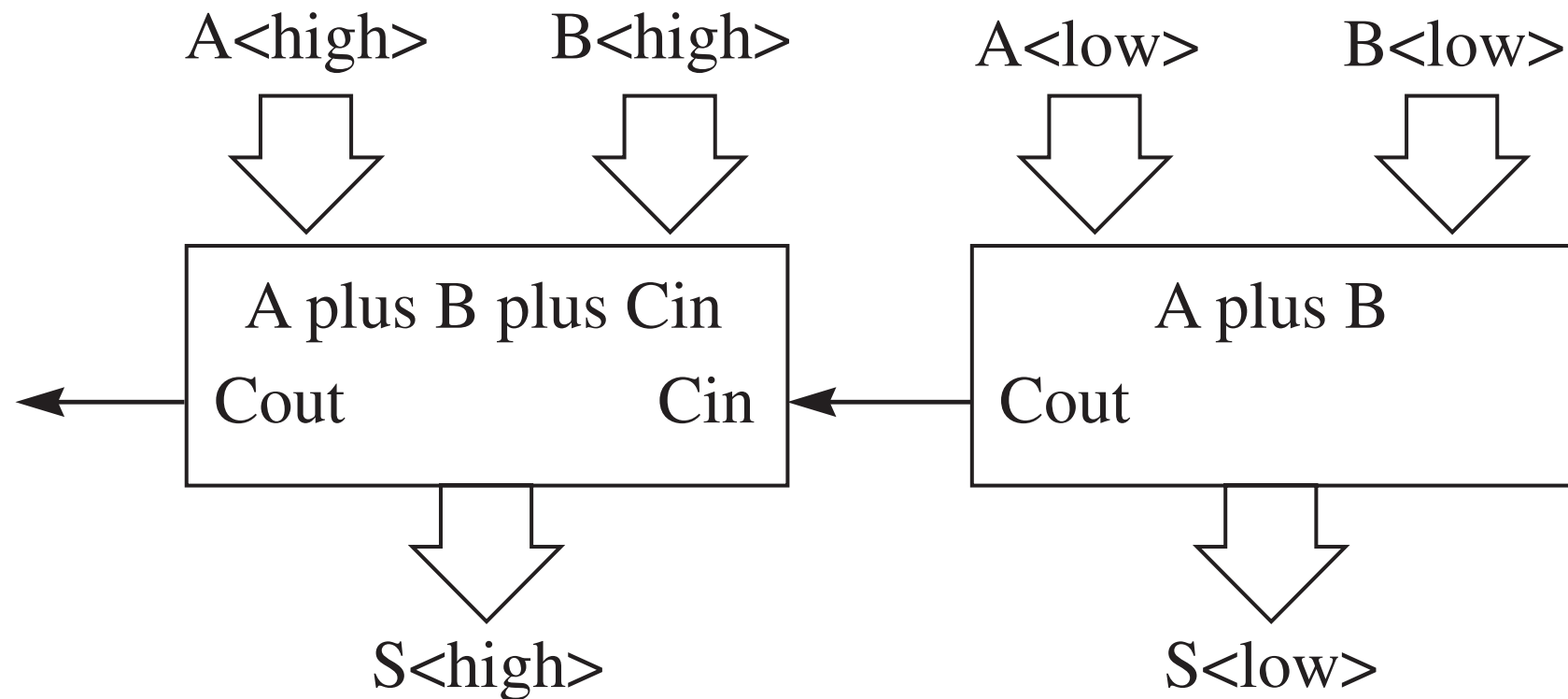
- If line 15 is 1, Result and NZVC from the *left* are routed to the output
- If line 15 is 0, Result and NZVC from the *right* are routed to the output

Figure 10.57

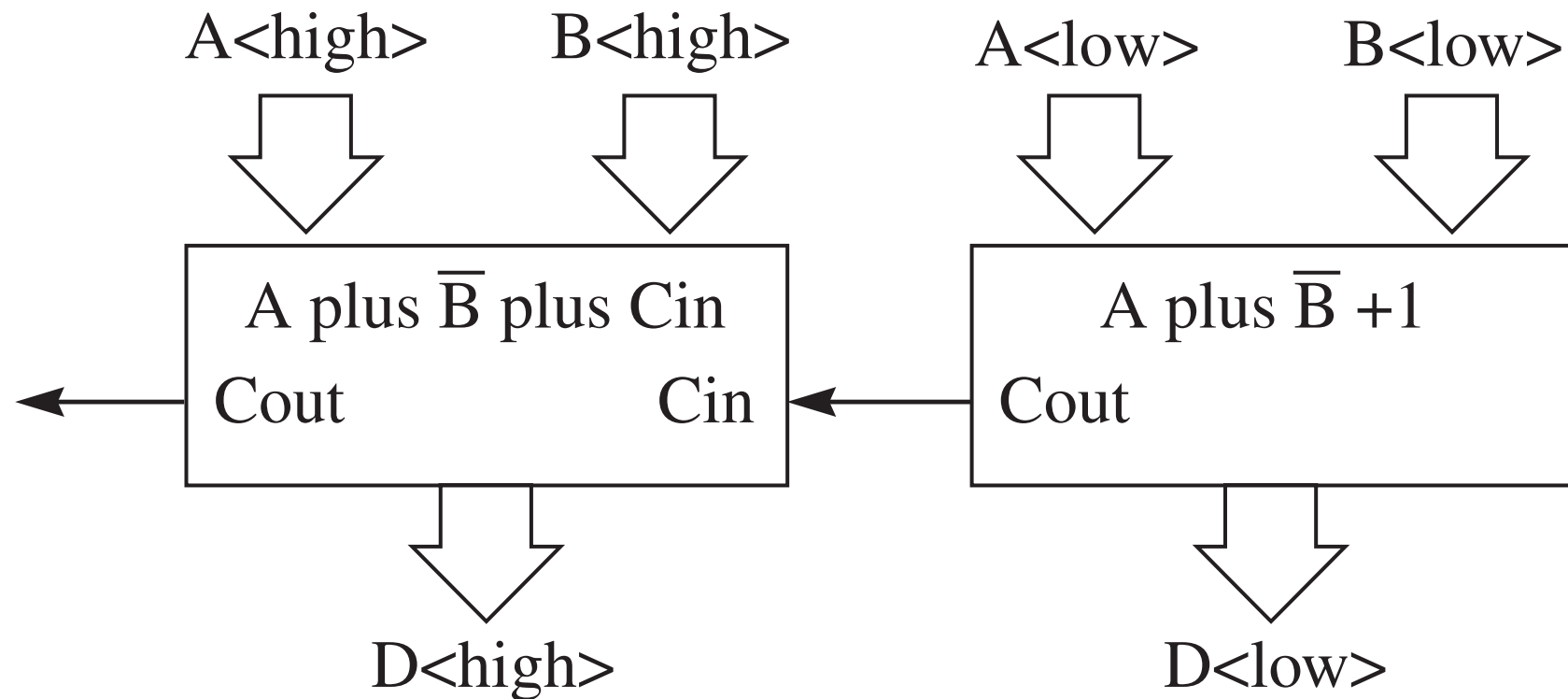






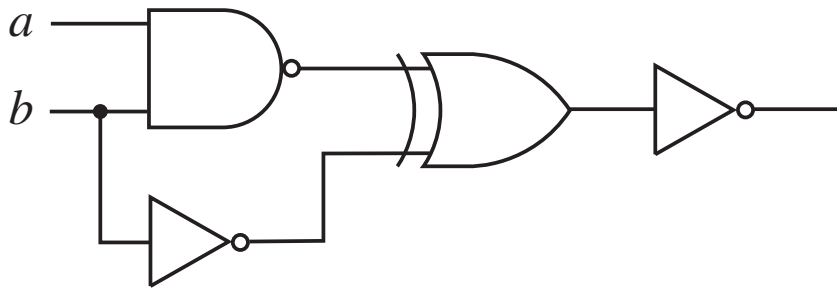


(a) 16-bit addition.

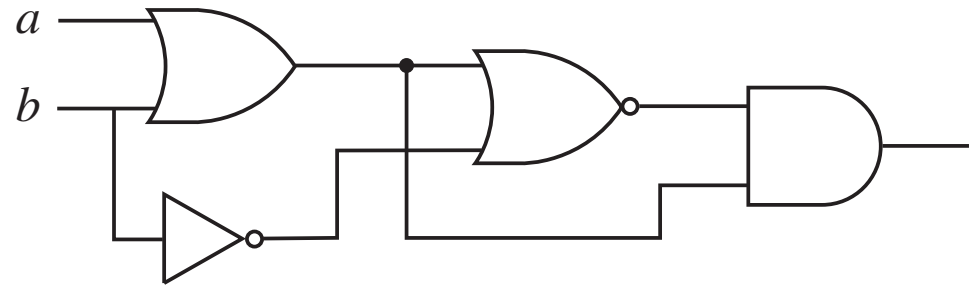


(b) 16-bit subtraction.

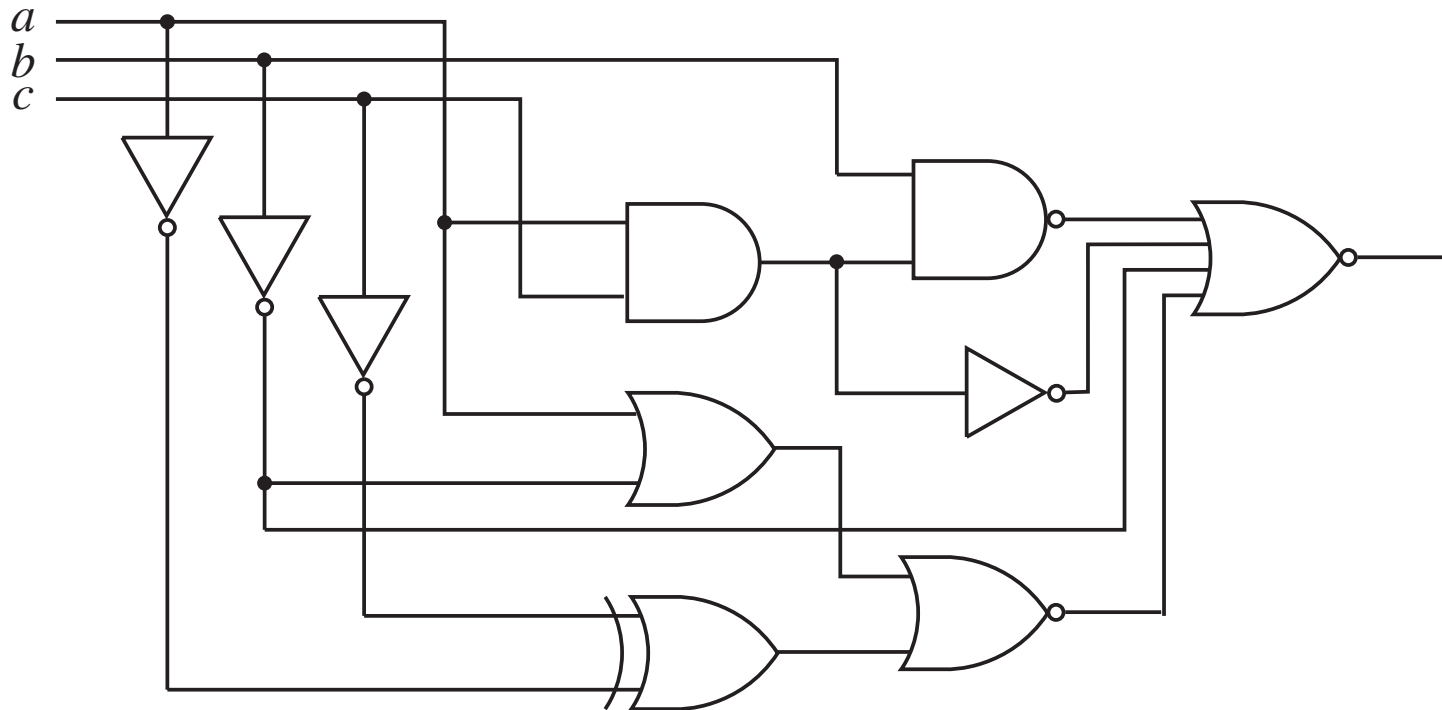
Function	d	e	f	g	Sub	C
A plus B	1	0	0	0	0	0
A plus B plus Cin	0	1	0	0	0	Cin
A plus \overline{B} plus 1	0	0	1	0	1	1
A plus \overline{B} plus Cin	0	0	0	1	1	Cin



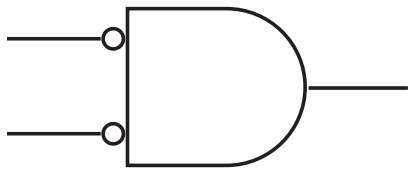
(a)



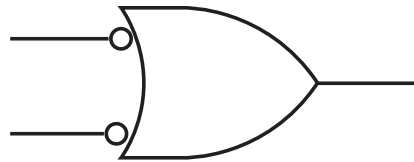
(b)



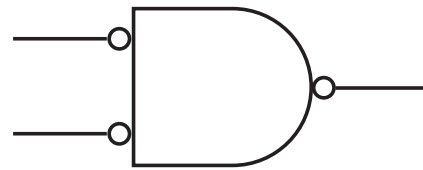
(c)



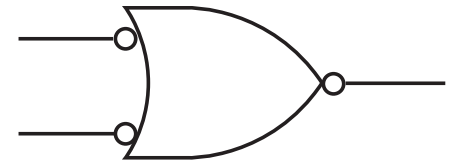
(a)



(b)



(c)



(d)

