Circuitos Elétricos e Sistemas Digitais

2018-2019 - 1.º Semestre

Sistemas Digitais

Introdução aos sistemas digitais

Bibliografia:

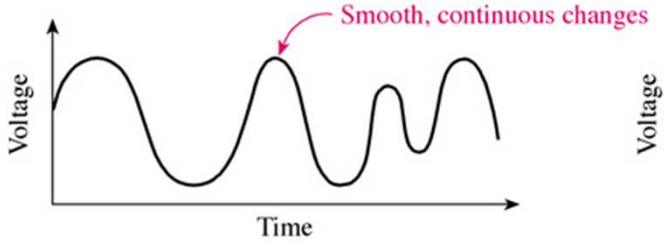
1) Logic and Computer Design Fundamentals, Morris, Kime, 4th Edition, Pearson Education Limited.

2) Digital Fundamentals - Global Edition, Floyd,11th Edition, Pearson Education, 2015.

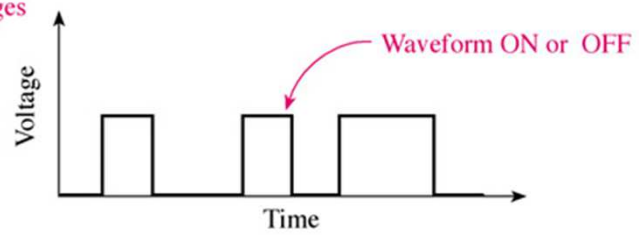
3) Digital Fundamentals: A Systems Approach, Floyd, Pearson Education

4) Sistemas Digitais: Fundamentos e Aplicações, 9ª edição, Floyd, Bookman

Sinais/Sistemas Digitais vs. Analógicos



(a)



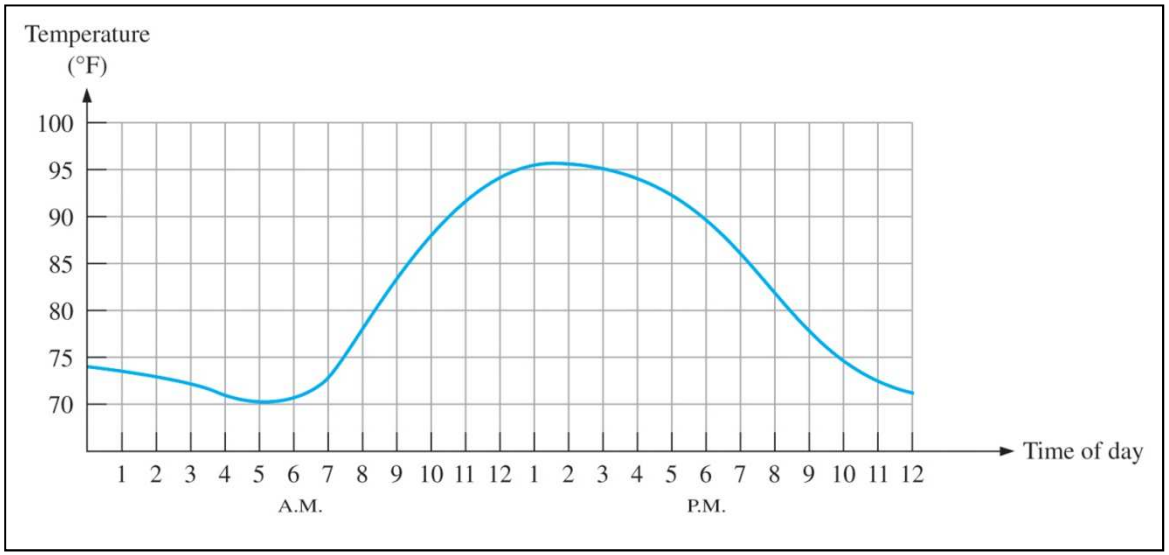
(b)

(c)

(d)

Quantidades analógicas

Most natural quantities that we see are **analog** and vary continuously. Analog systems can generally handle higher power than digital systems.



Digital systems can process, store, and transmit data more efficiently but can only assign discrete values to each point.

Sinais analógicos

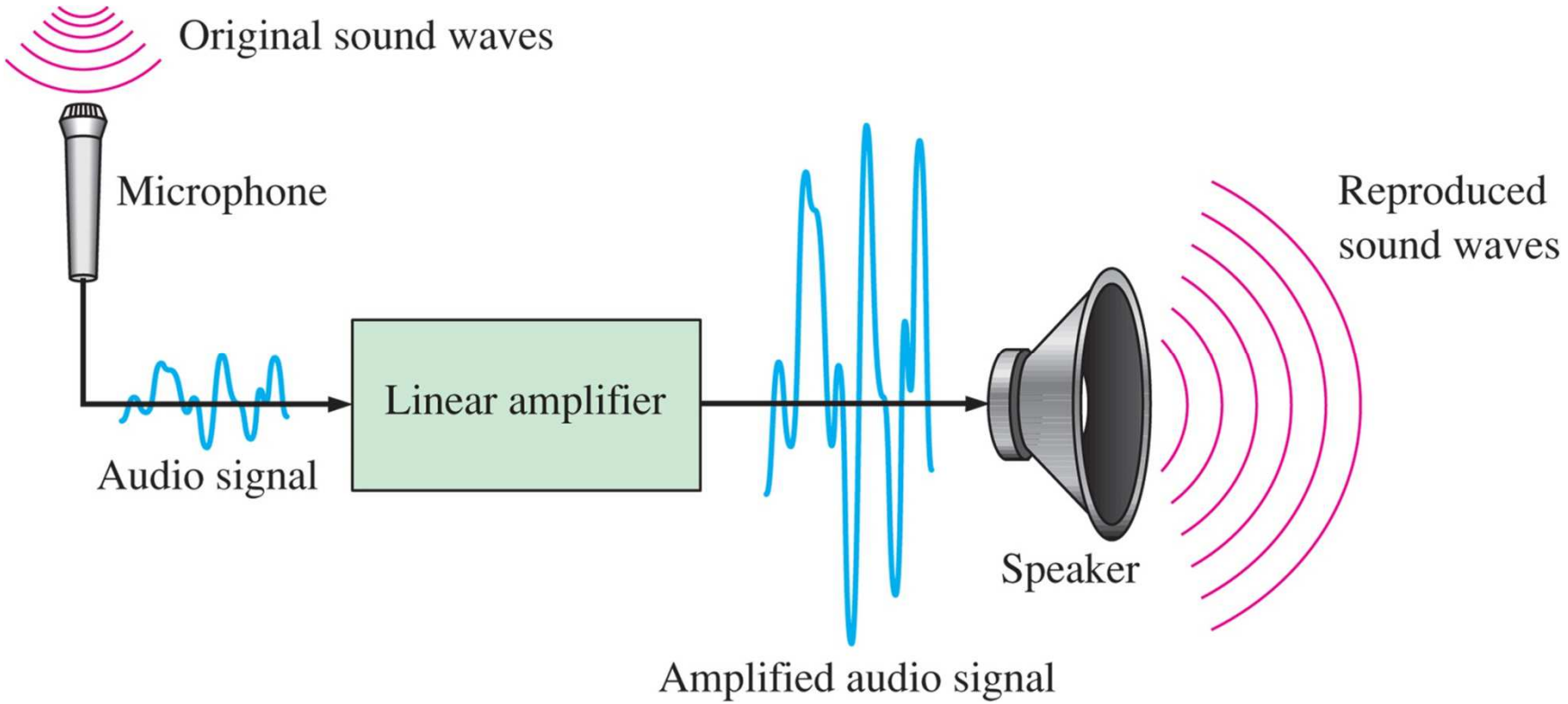
A waveform that continually varies in a certain manner is classified as an **analog signal**.

Examples:

- Sine waves
- Audio waves
- Amplitude modulated (AM) signals
- Frequency modulated (FM) signals

Sistema analógico

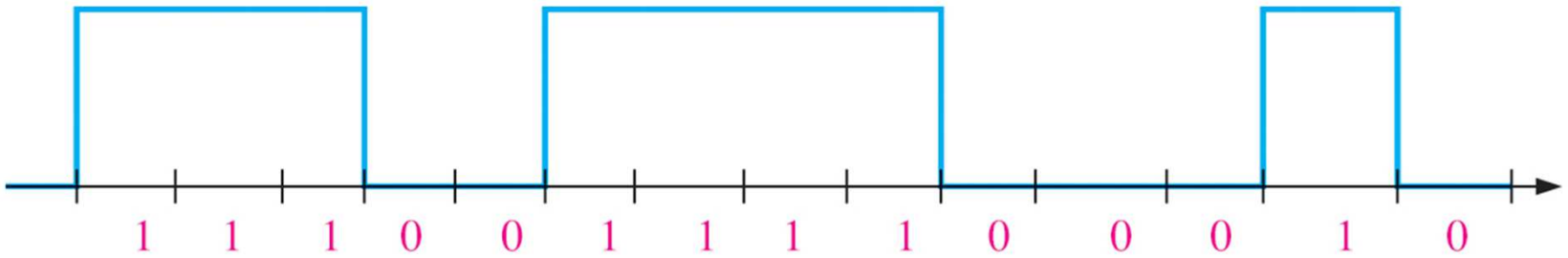
A basic audio public address system.



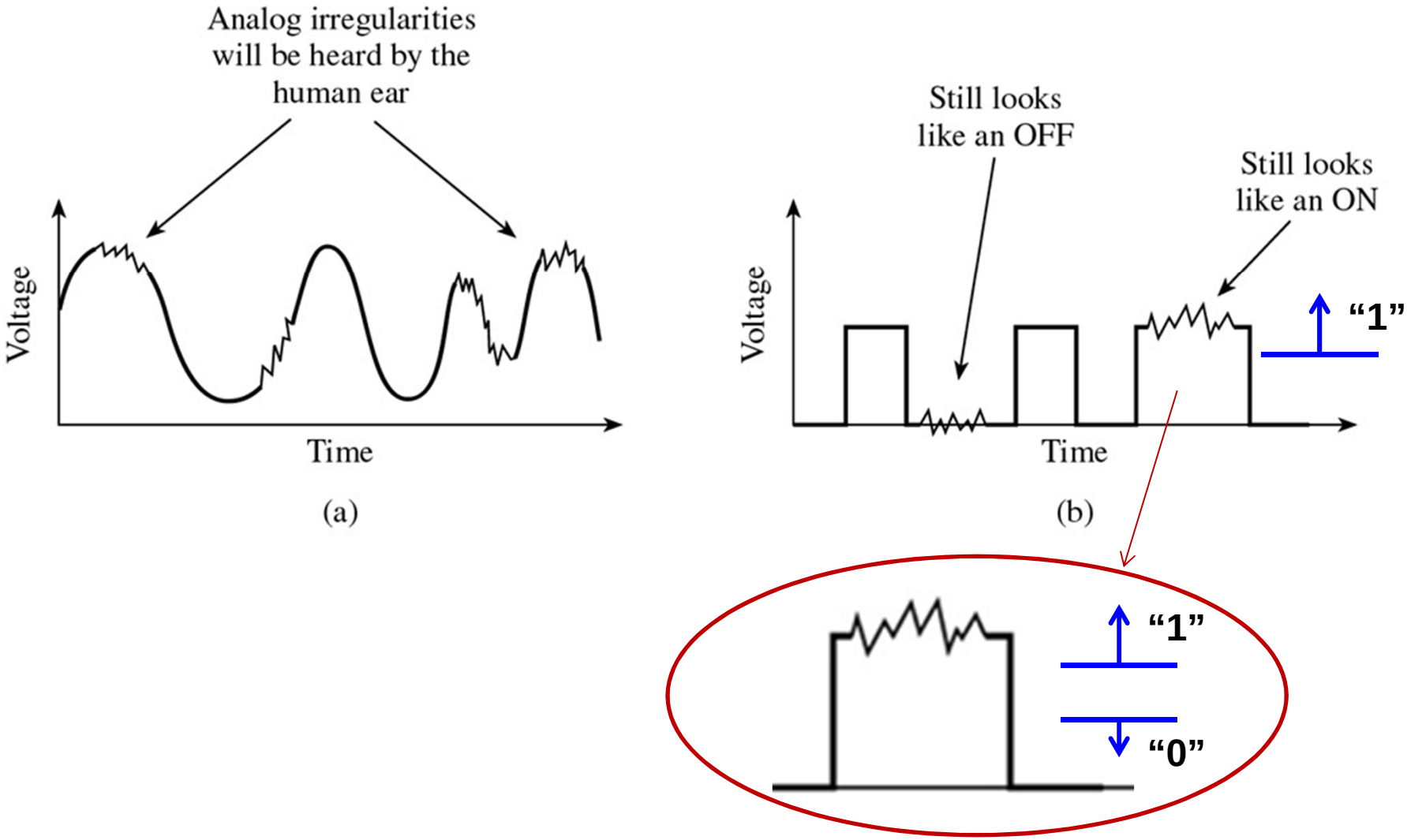
Sinais digitais

A waveform that represents a sequence of discrete values (1's and 0's) is called a **digital signal**.

Bit streams are found in telecommunications, computers, and other data system applications.



Vantagens dos sinais digitais:
Os sistemas digitais são + imunes ao ruído analógico

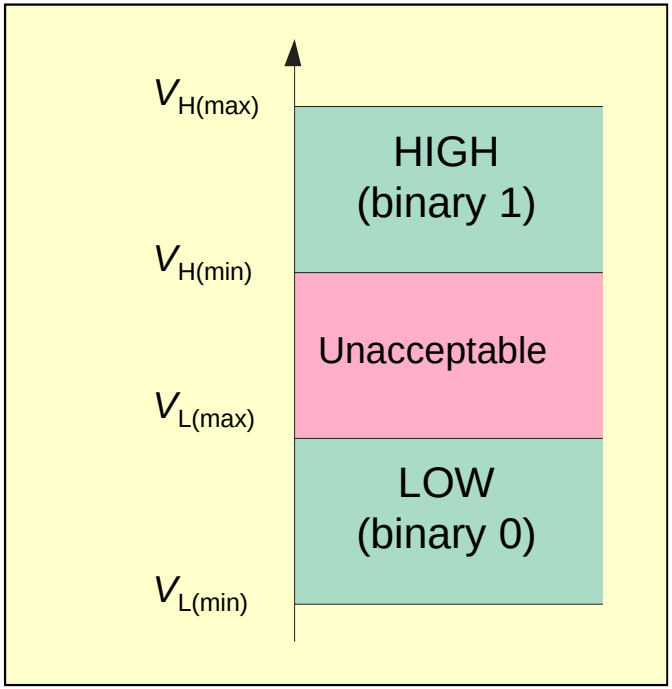


Dígitos binários e níveis lógicos

Binary Digits and Logic Levels

Digital electronics uses circuits that have two states, which are represented by two different voltage levels called HIGH and LOW. The voltages represent numbers in the binary system.

*In binary, a single number is called a **bit** (for **b**inary dig**it**). A bit can have the value of either a 0 or a 1, depending on if the voltage is HIGH or LOW.*



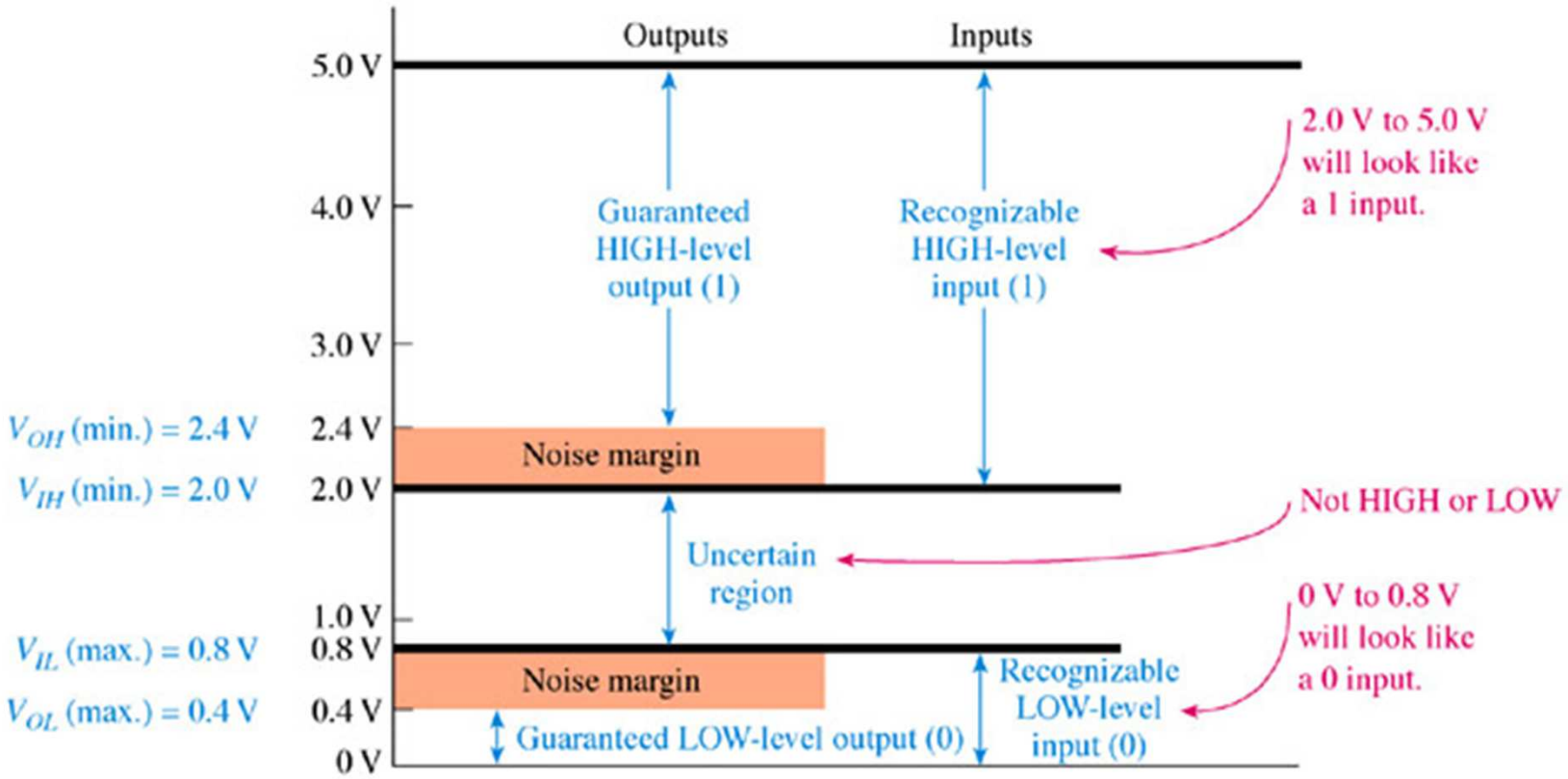
Uma palavra “binária”/código binário é formado por uma sequência de zeros (0) e uns (1). Exemplo:

8 em binário é representada por 1000

Níveis de tensão e de corrente na tecnologia TTL

TTL Voltage and Current Ratings

- Input/Output Voltages (graphical representation)



TTL - Transistor-Transistor Logic - Uma família de circuitos digitais

TTL Voltage and Current Ratings

- Input/Output Voltages and noise margin

TABLE 9–1 Standard 74XX Series Voltage Levels

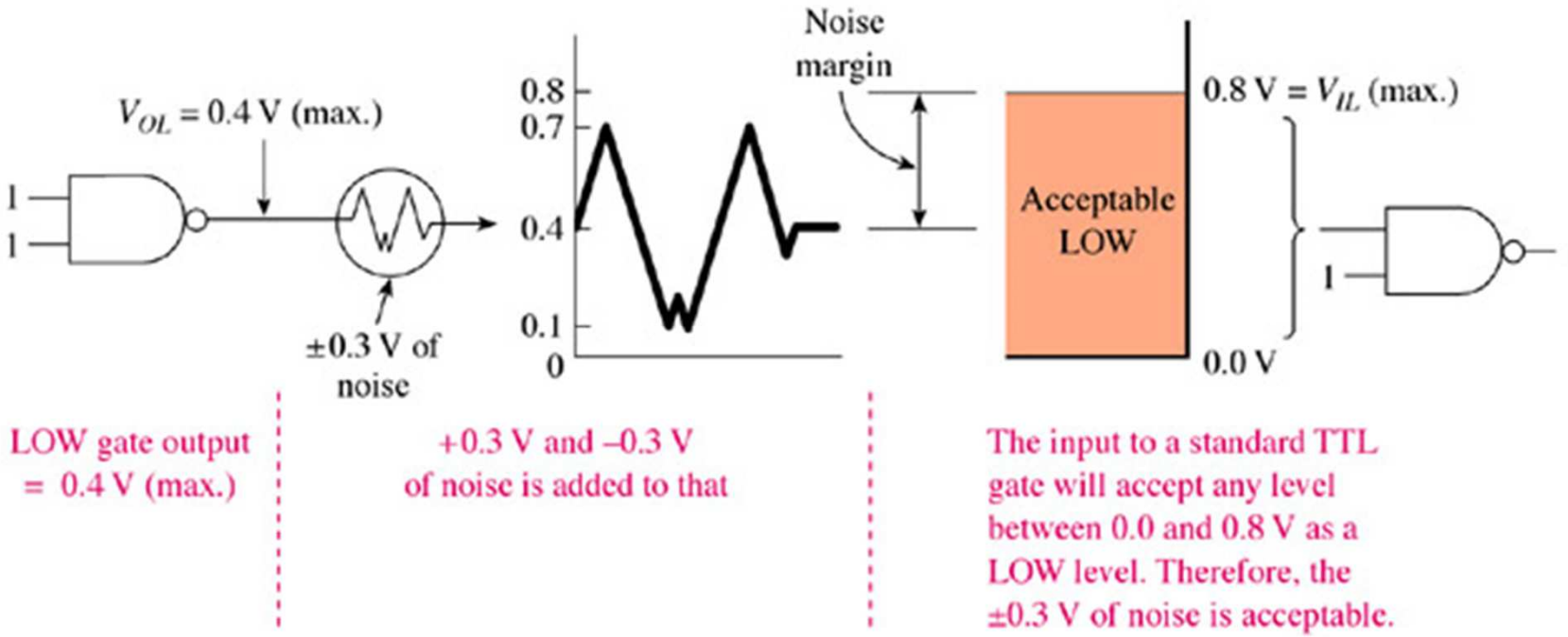
Parameter	Minimum	Typical	Maximum	
V_{OL}		0.2 V	0.4 V	Noise margin = 0.4 V
V_{IL}			0.8 V	
V_{OH}	2.4 V	3.4 V		Noise margin = 0.4 V
V_{IH}	2.0 V			

Noise margin (HIGH) = V_{OH} (min) − V_{IH} (min)
Noise margin (LOW) = V_{IL} (max) − V_{OL} (max)

TTL - Transistor-Transistor Logic - Uma família de circuitos digitais

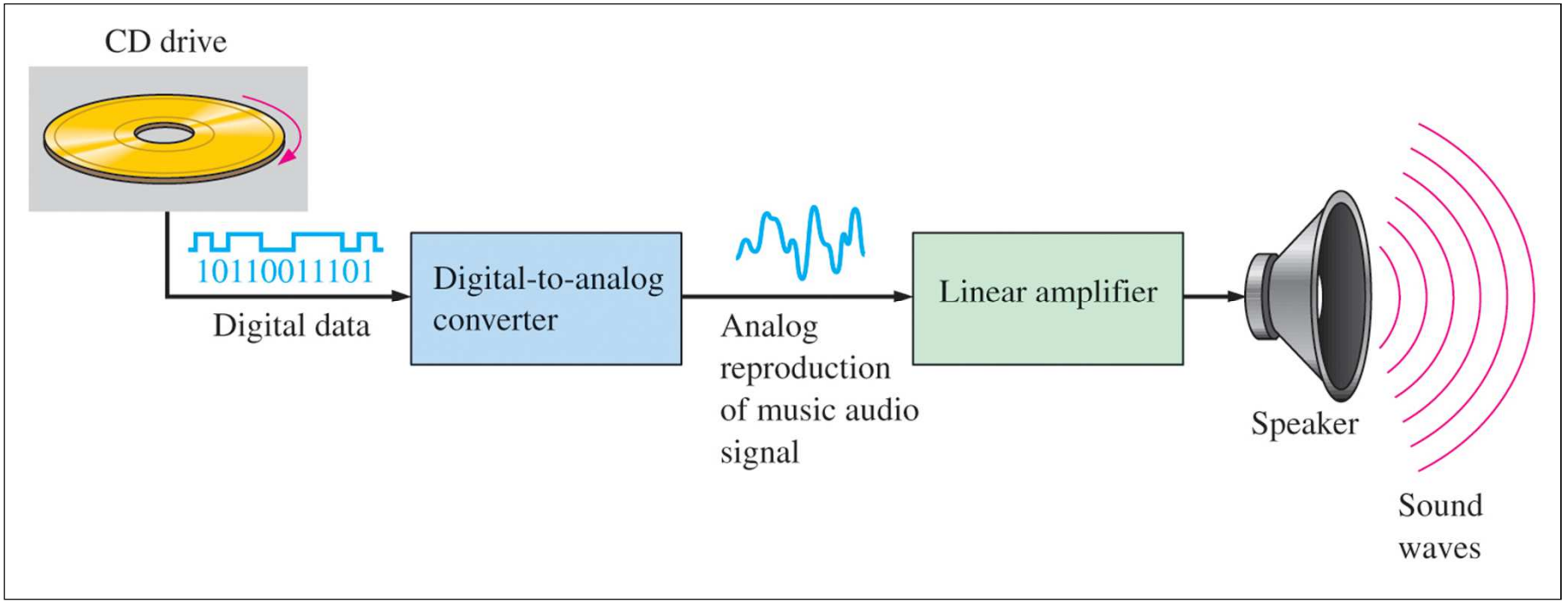
Margens de ruído
Noise Margin

Noise margin: The difference between high level voltages and low level voltages



Analog and Digital Systems

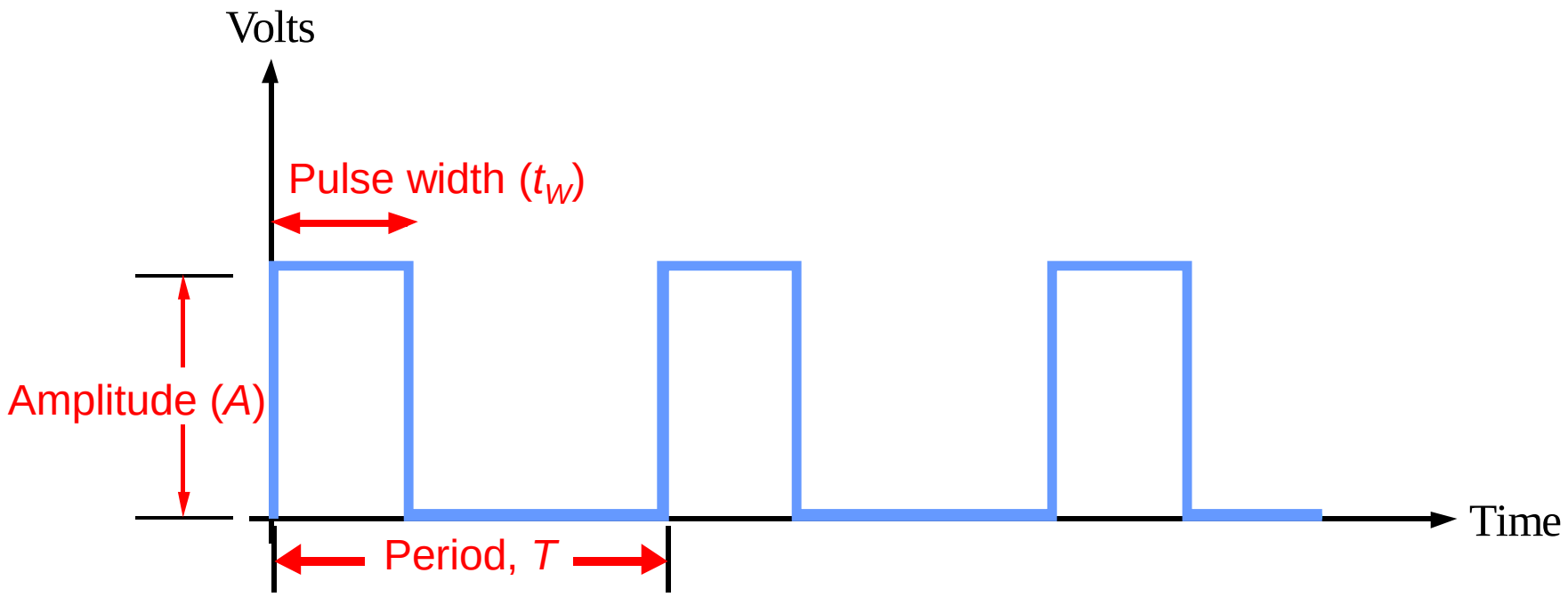
Many systems use a mix of analog and digital circuits to utilize the advantages of each. A typical CD player accepts digital data from the CD drive and converts it to an analog signal for amplification.



Formas de onda digitais: trem de pulsos

In addition to frequency and period, repetitive pulse waveforms are described by the amplitude (A), pulse width (t_W) and duty cycle.

Duty cycle is the ratio of t_W to T.



Periodic Pulse Waveforms

Periodic pulse waveforms are composed of pulses that repeats in a fixed interval called the **period**. The **frequency** is the rate it repeats and is measured in hertz.

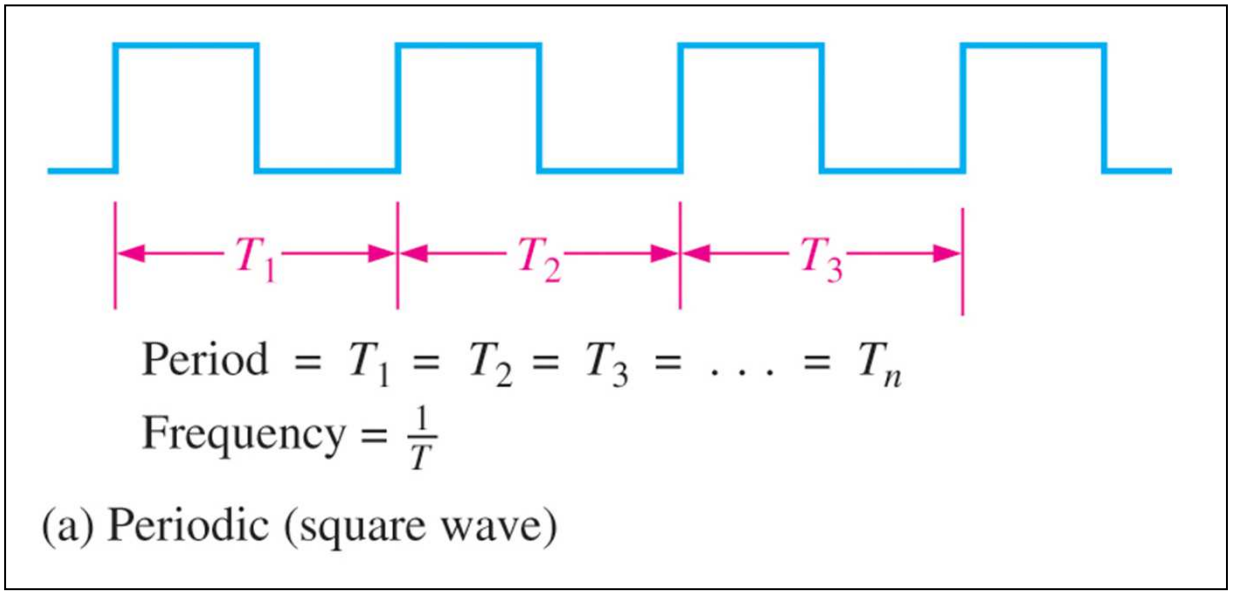
$$f = \frac{1}{T} \qquad T = \frac{1}{f}$$

The **clock** is a basic timing signal that is an example of a periodic wave.

What is the period of a repetitive wave if f = 3.2 GHz?

$$T = \frac{1}{f} = \frac{1}{3.2\,\text{GHz}} = \textbf{313 ps}$$

Periodic and Nonperiodic Pulse Waveforms



(a) Periodic (square wave)

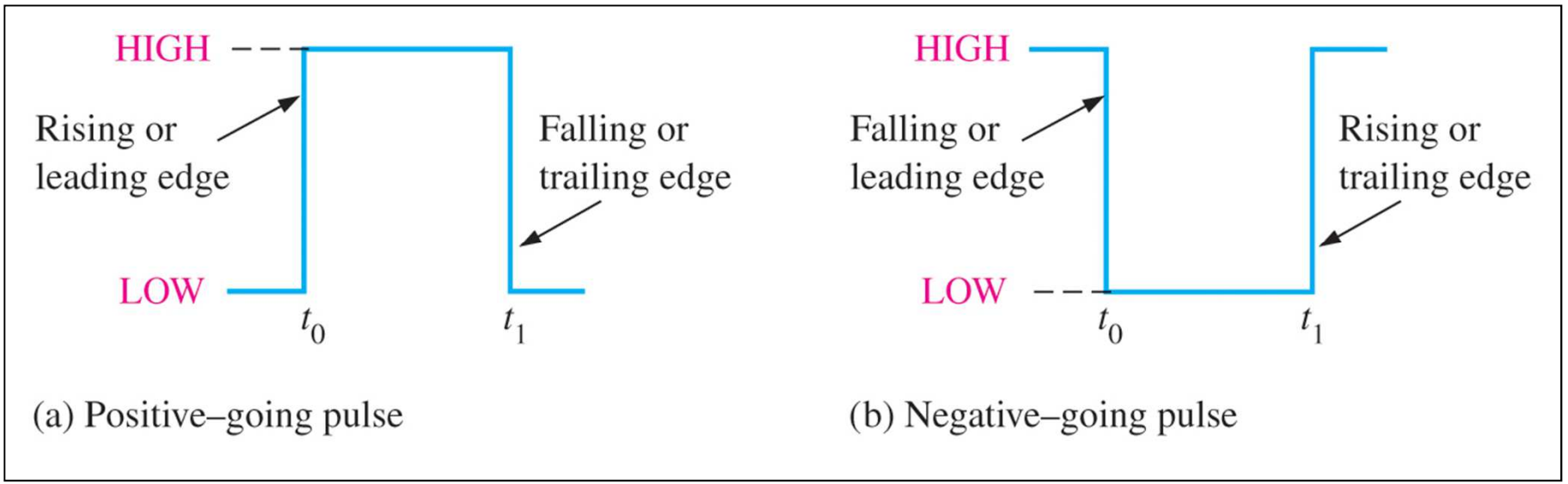
(b) Nonperiodic

Digital Waveforms

Digital waveforms change between the LOW and HIGH levels.

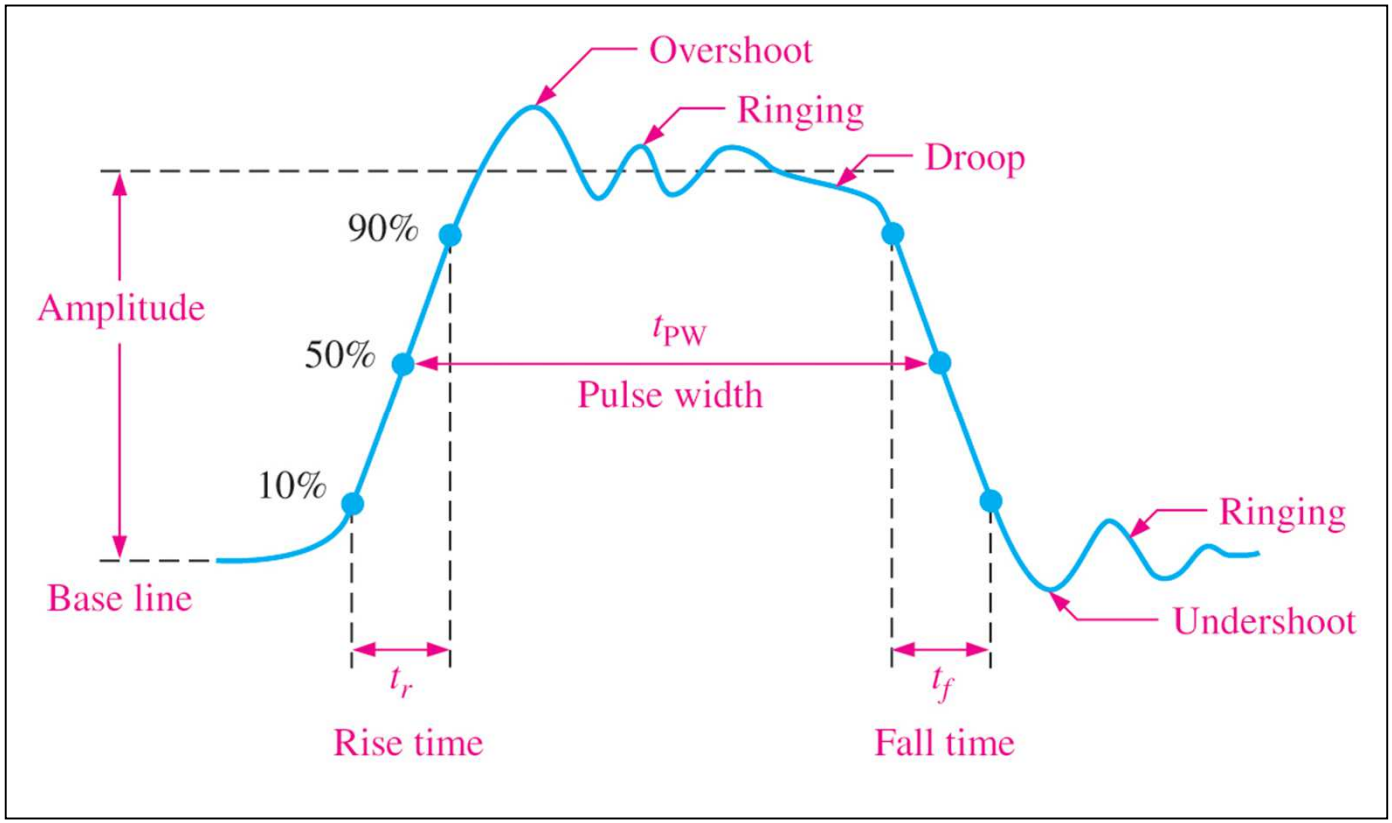
A **positive-going pulse** is one that goes from a normally LOW level to a HIGH level and then back again.

A **negative-going pulse** is one that goes from a normally HIGH level to LOW level and then back again.



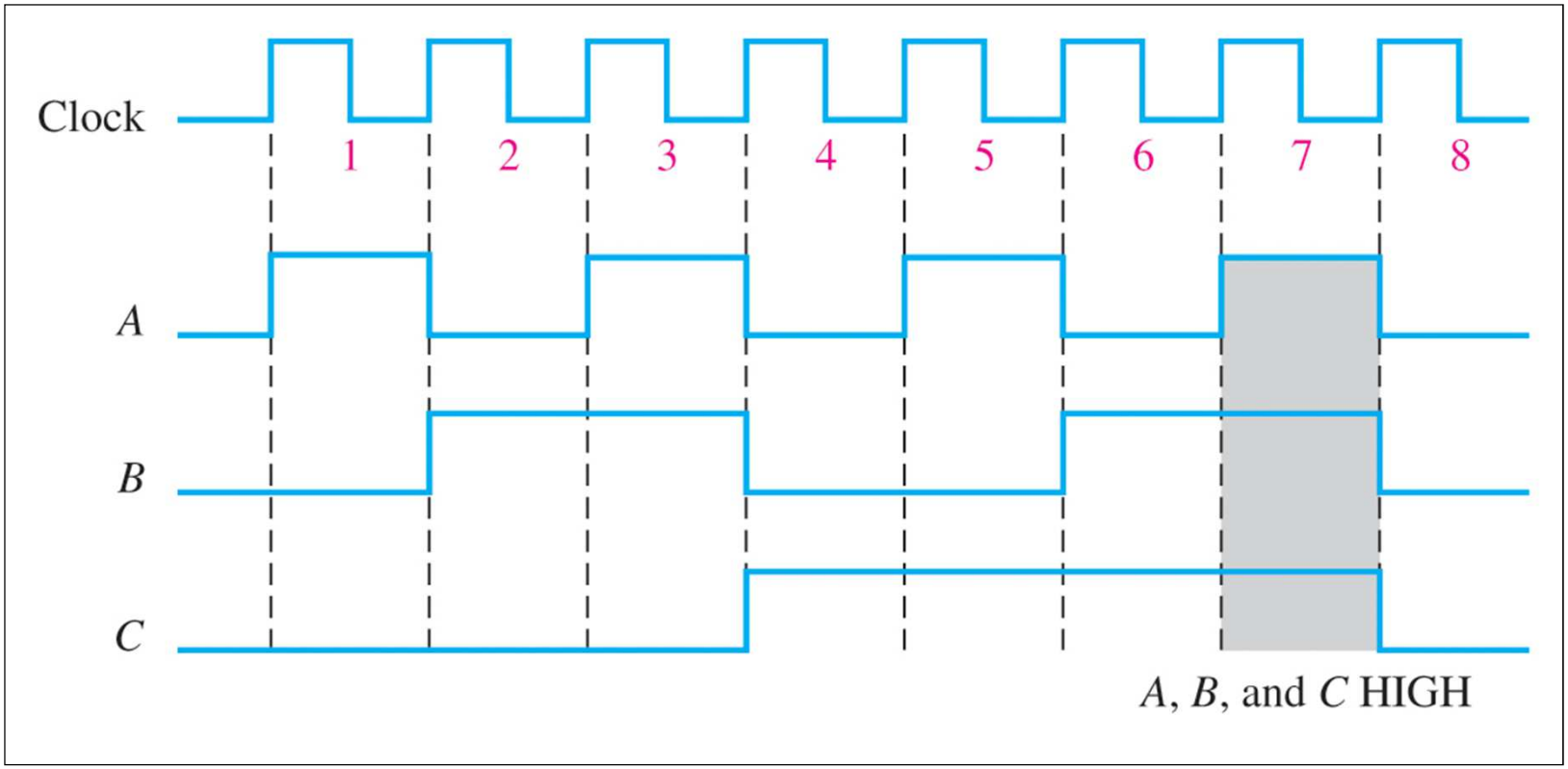
Pulse Definitions

Actual pulses are not ideal but are described by the rise time, fall time, amplitude, and other characteristics.



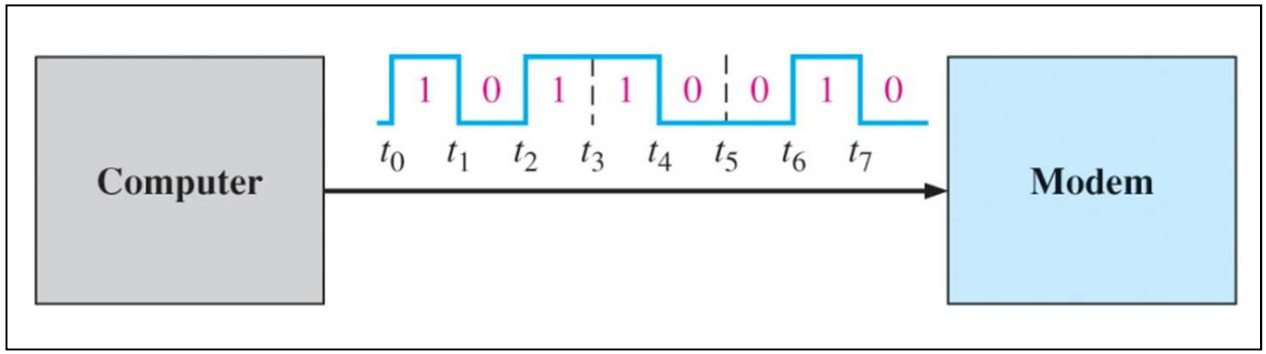
Diagramas temporais/temporização
Timing Diagrams

A timing diagram is used to show the relationship between two or more digital waveforms,



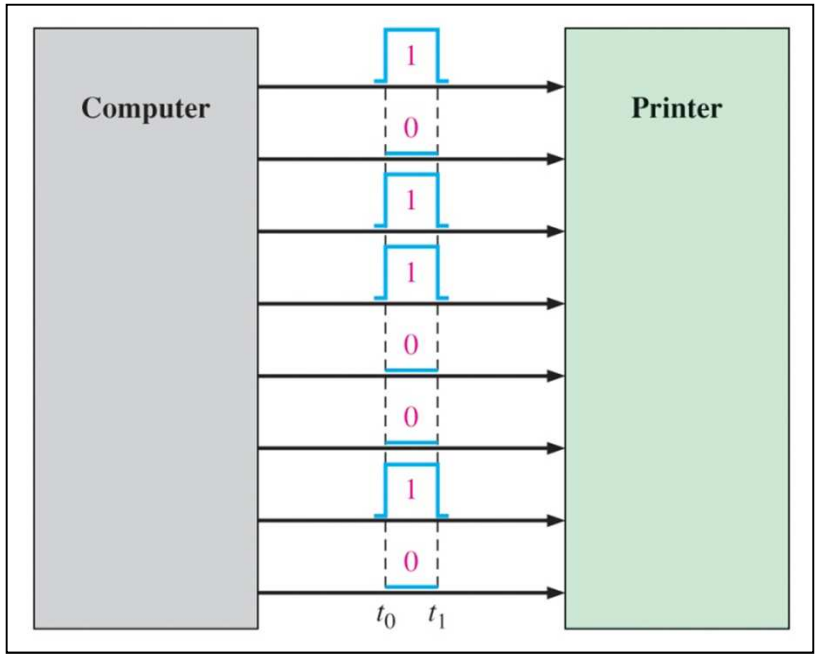
Transferência de informação binária

Data can be transmitted by either serial transfer or parallel transfer.



Serial data transfer

Parallel data transfer



Exemplos de sinais digitais

Modulação por Código de Pulso (PCM)

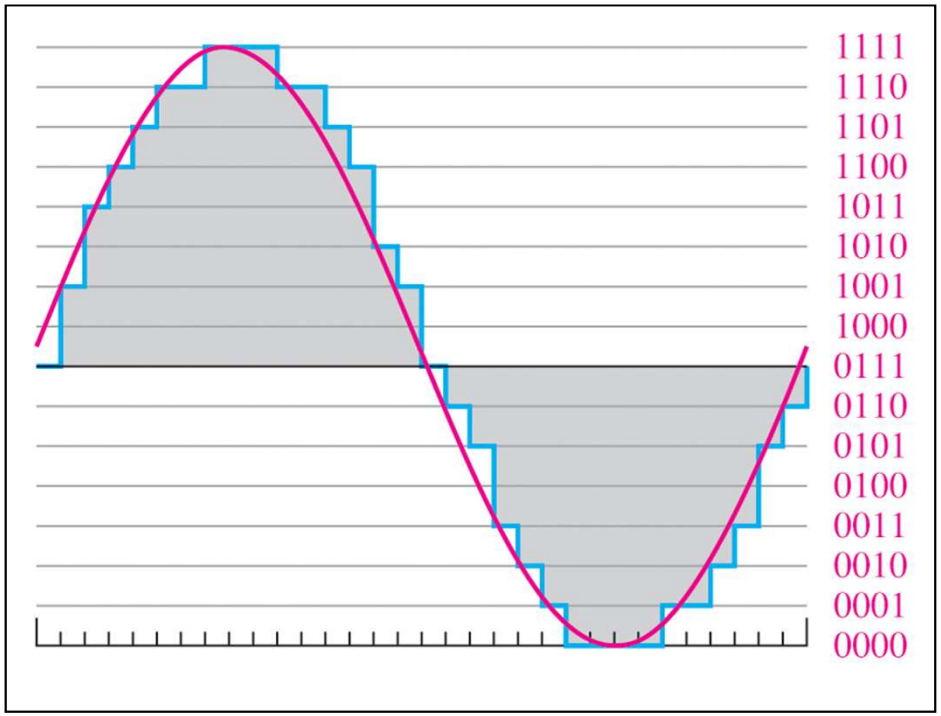
Pulse-code Modulation (PCM)

Modulação por Código de Pulso (**PCM**) ou em inglês Pulse-code modulation (PCM) é um método usado para representar digitalmente amostras de sinais analógicos.

PCM uses a sequence of digital codes to represent a sampled analog signal.

Sampling produces the "stair-step" voltage shown.

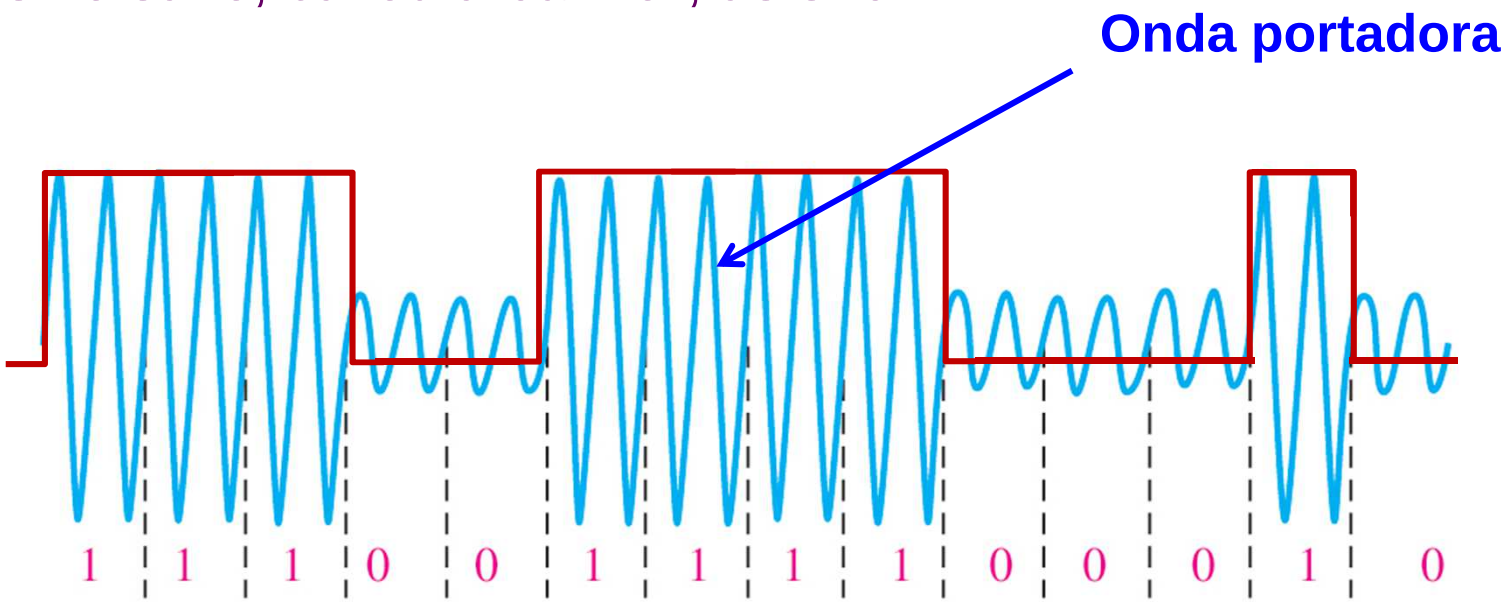
The higher the sampling rate, the more accurate the digitized waveform.



Exemplos de sinais digitais
Modulação digital

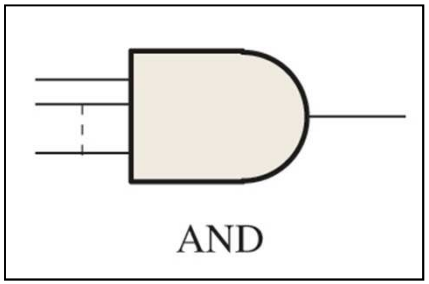
Digital modulation provides a means of transmitting data from one system to another.

One approach is to use digital data to modulate the amplitude of a sine save, called a **carrier**, as shown.

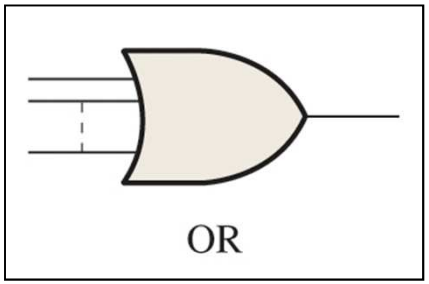


Basic Logic Functions

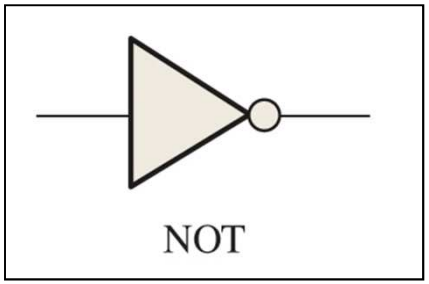
AND Produces a high output only if all inputs are high.



OR Produces a high input if one or more inputs are high.

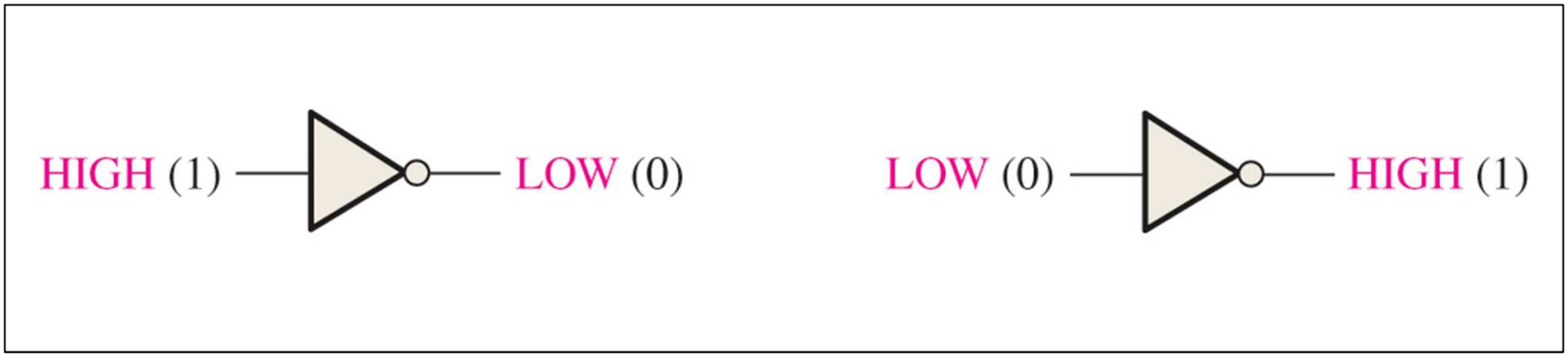


NOT Changes one logic level to the other logic level.



Inversor – operação negação
The NOT Operation

Changes one logic level to the other logic level.



The NOT operation is performed by a circuit called an **inverter**.

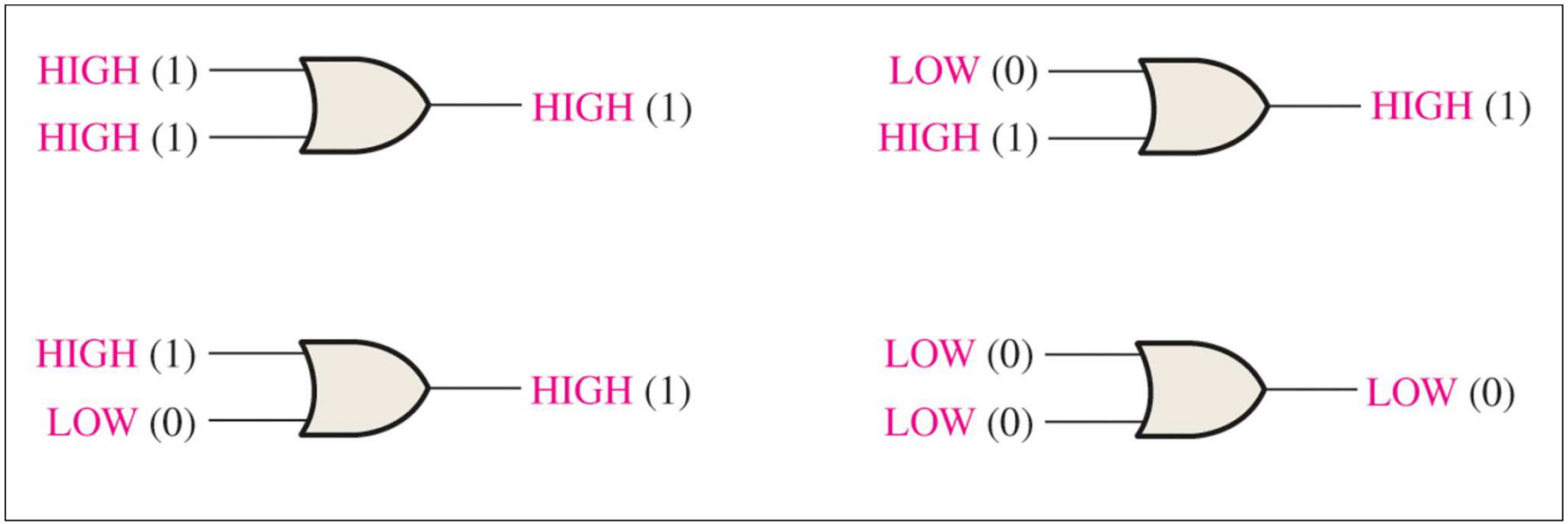
The AND Operation

Produces a high output only when ALL inputs are high.

The AND operation is performed by a circuit called an **AND gate**.

The OR Operation

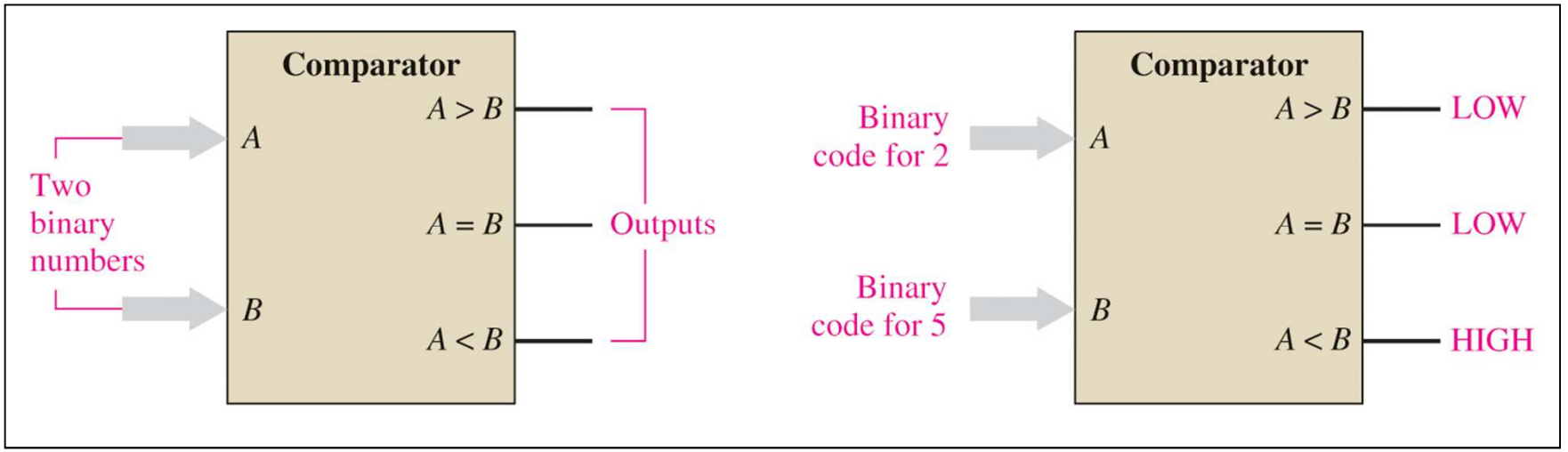
Produces a high output when one or more inputs are high.



The OR operation is performed by a circuit called an **OR gate**.

The Comparison Function

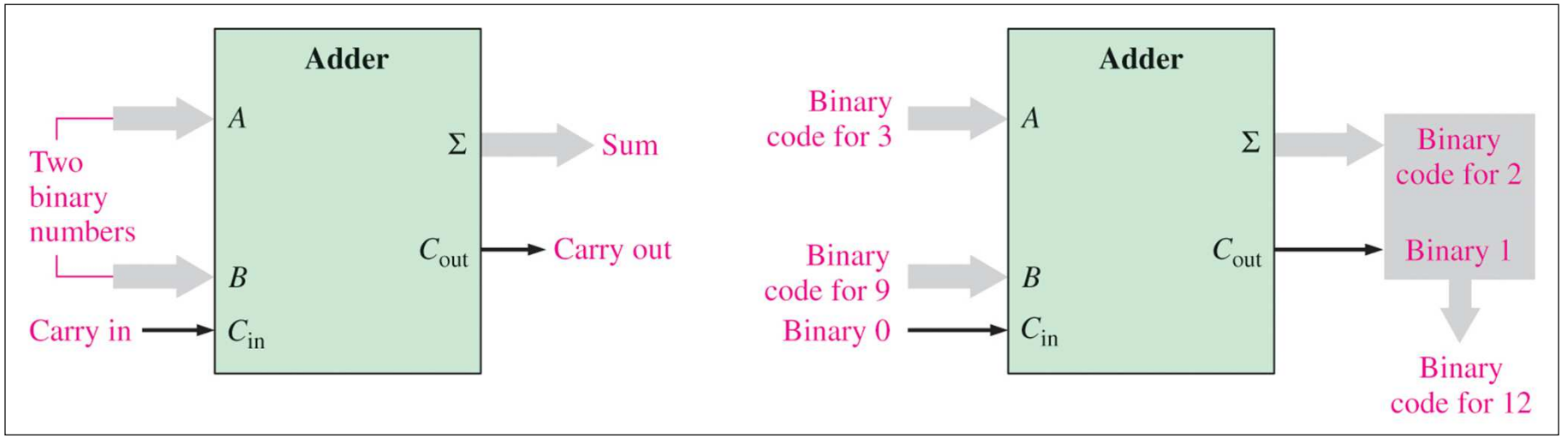
And, **or**, and **not** elements can be combined to form various logic functions. The **comparison function** indicates whether a binary value is greater than, equal to, or less than, another.



The comparison function is performed by a circuit called a **comparator**.

The Arithmetic Functions

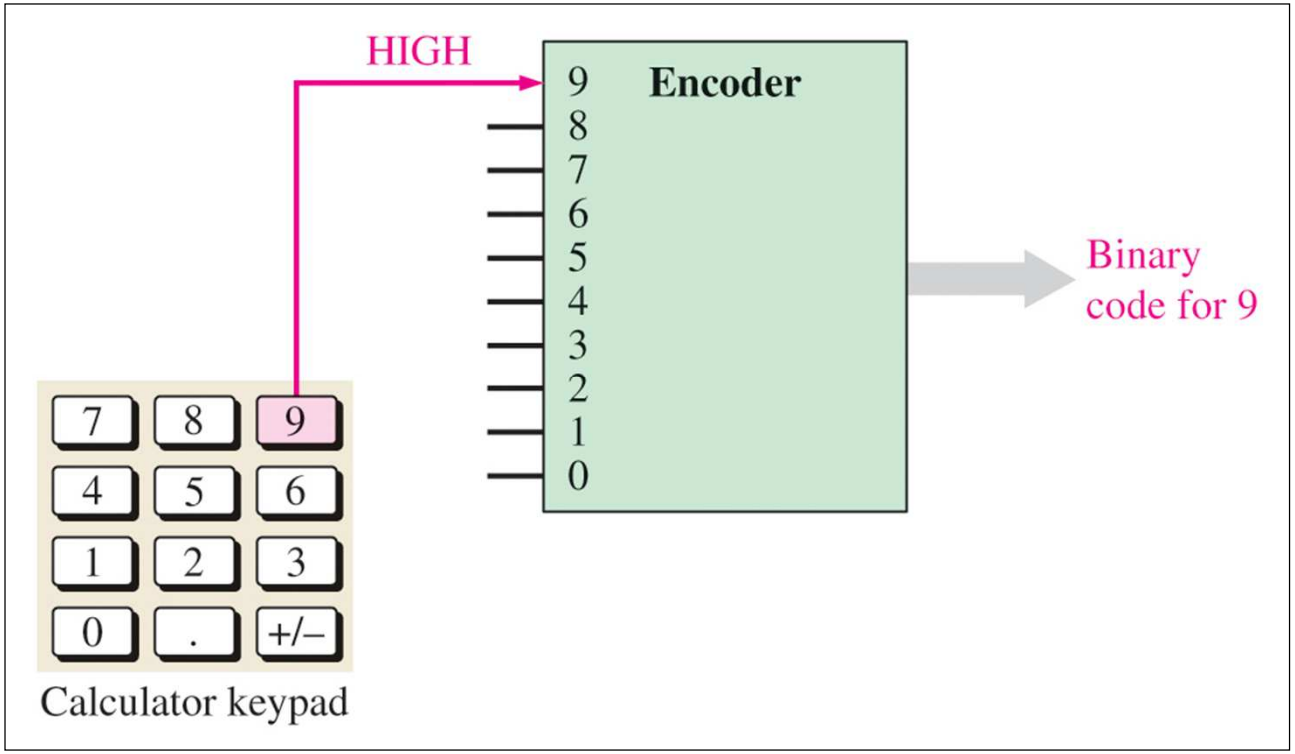
The **arithmetic functions** include addition, subtraction, multiplication, and division.



Addition is performed by an **adder** and subtraction by a **subtracter**. Multiplication and division are performed using circuits that are similar to adders and subtracters.

The Encoder Function

Code: A set of bits arranged in a unique pattern that represents specific information.

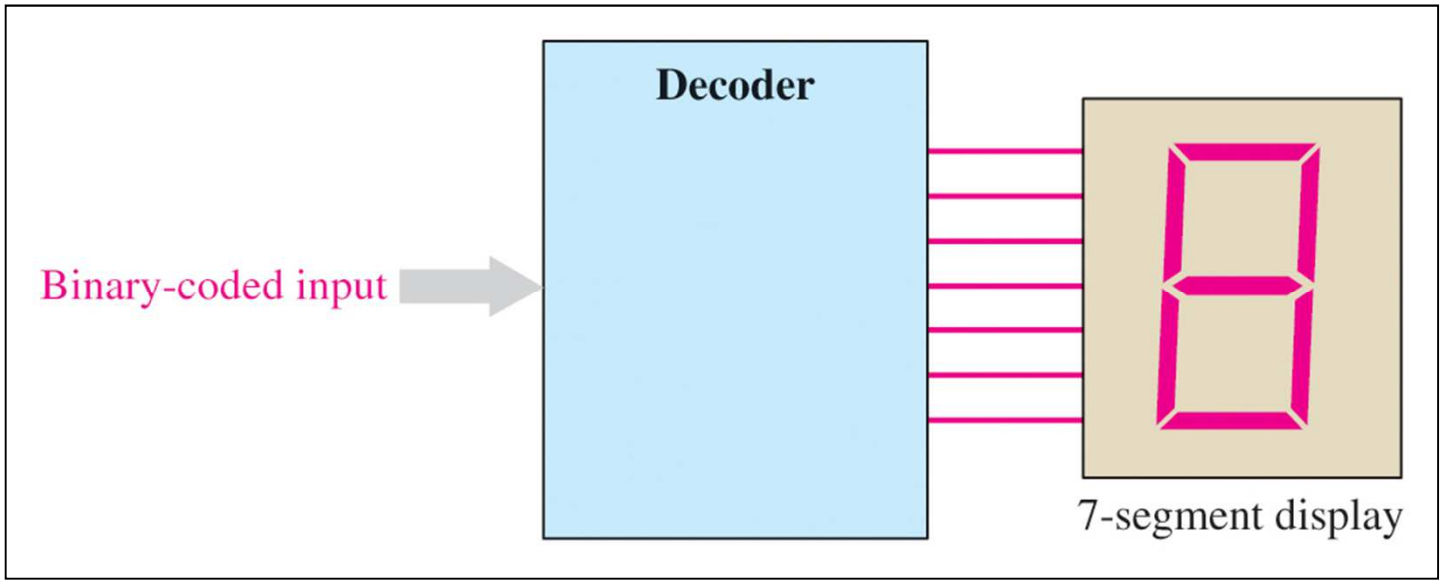


Encoder: A circuit that converts information into a coded form.

The Decoder Function

Decoder: A circuit that converts a binary code into a non-binary code of some kind.

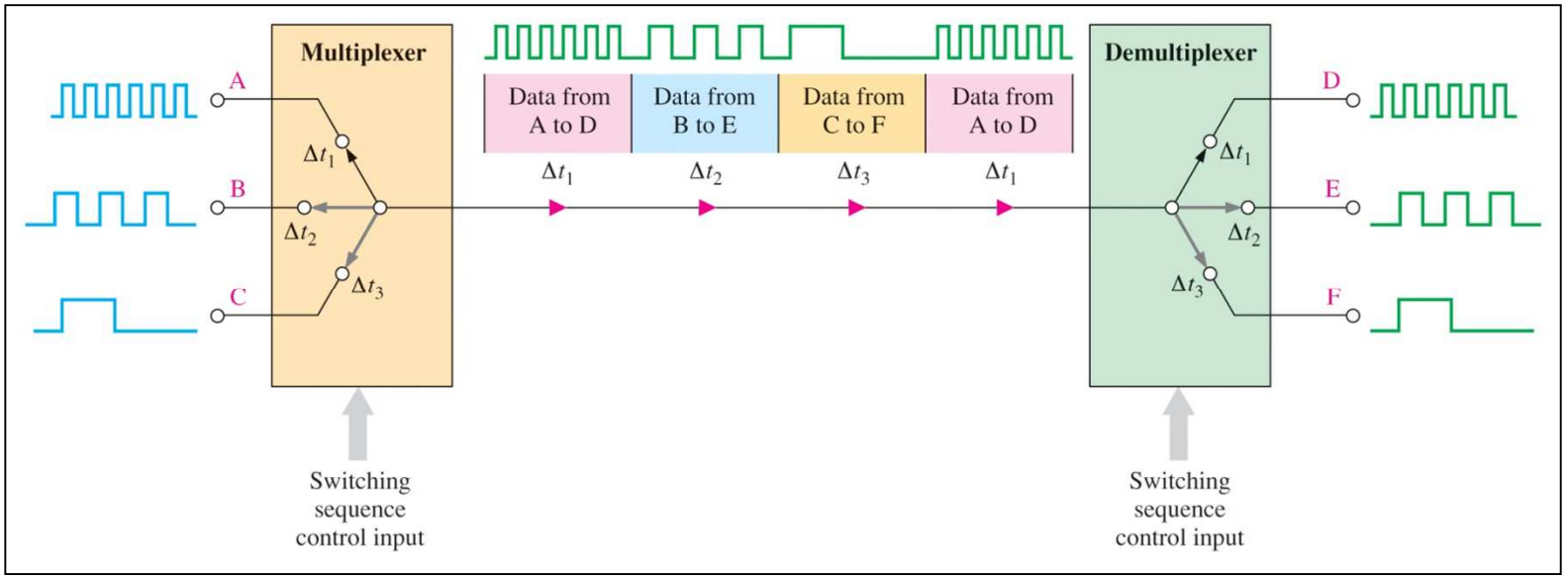
Example: The decoder below converts a binary coded input to a form that will light the segments in the 7-segment display required to display a specific character.



Função de seleção de dados

The Data Selection Function

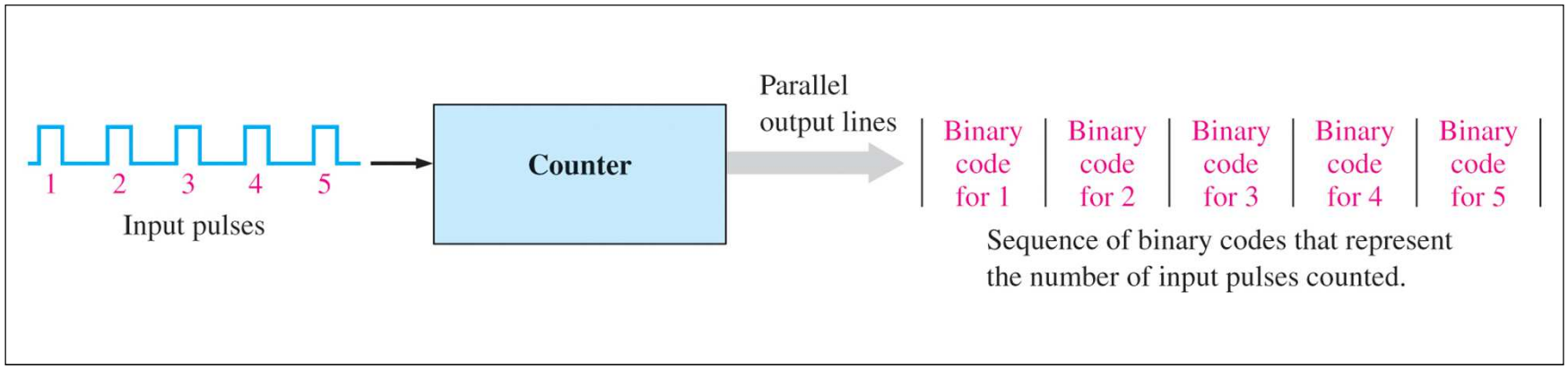
Multiplexer: A circuit that connects one of several data inputs to a single data output line; also called a **mux**



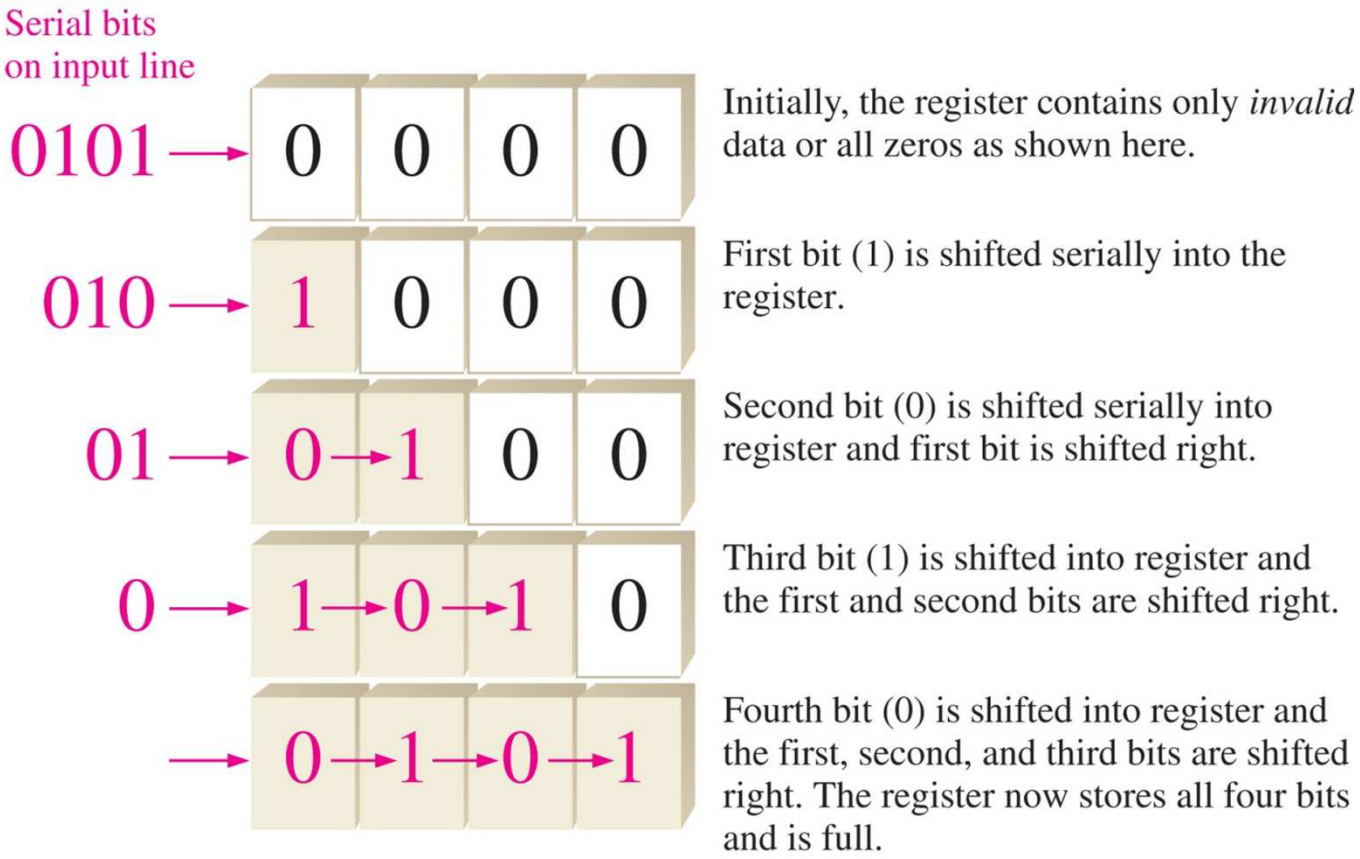
Demultiplexer: A circuit that connects one data input line to one of several data output lines; also called a **demux**

Função de contagem
The Counting Function

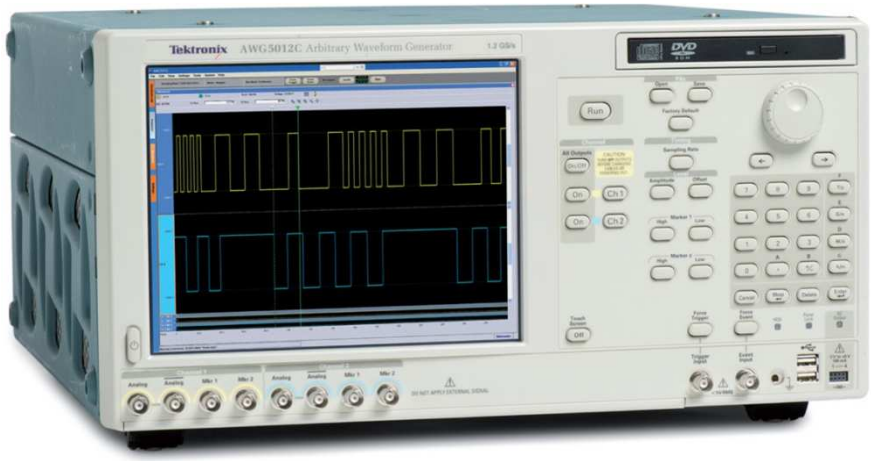
Counter: A sequential device; a **state machine** that has a unique internal sequence of states. Counters are used to count events or to generate output sequences represented by changing levels or pulses.



Função de armazenamento
(register/registradores)



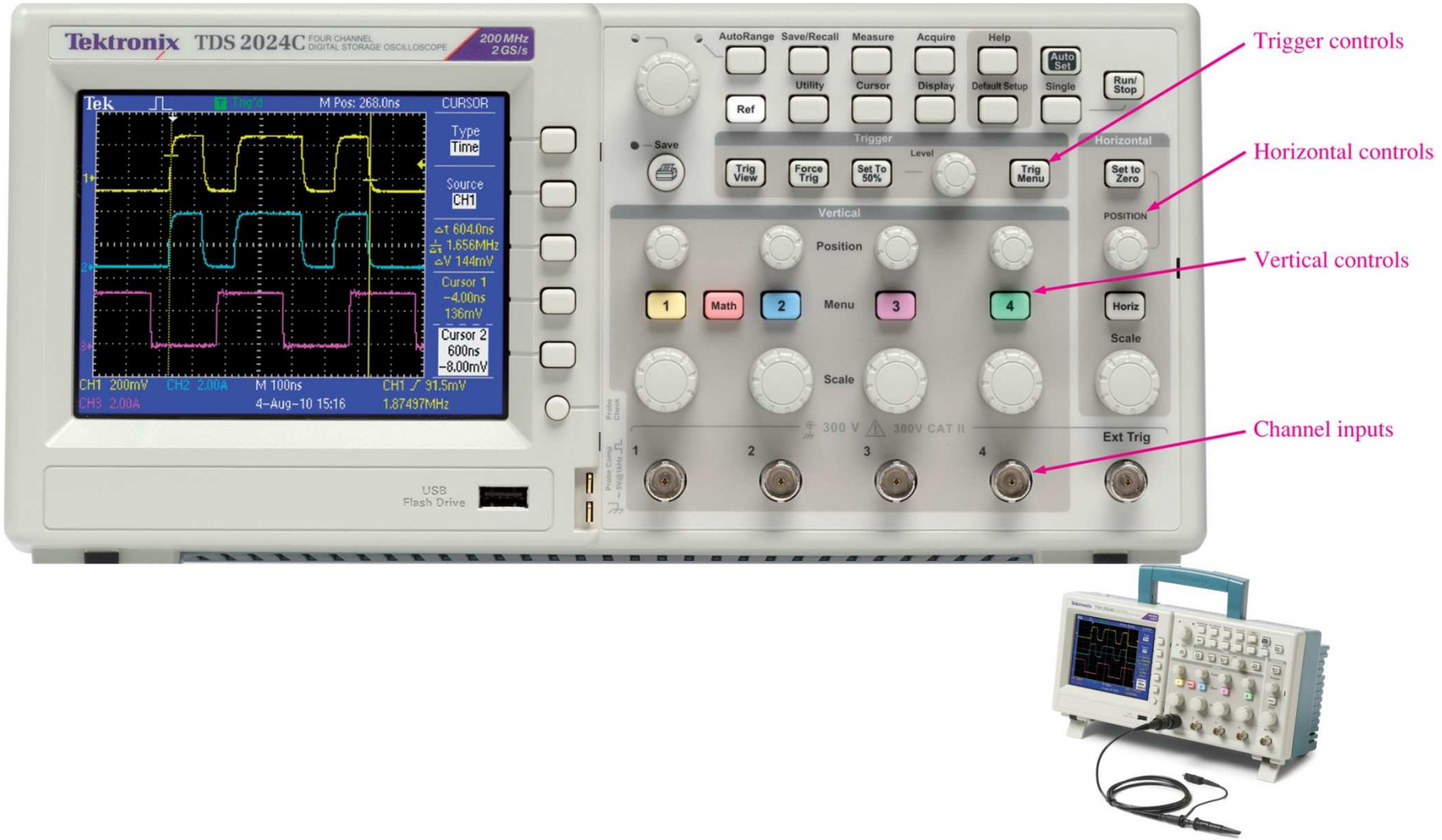
Geradores de funções arbitrárias



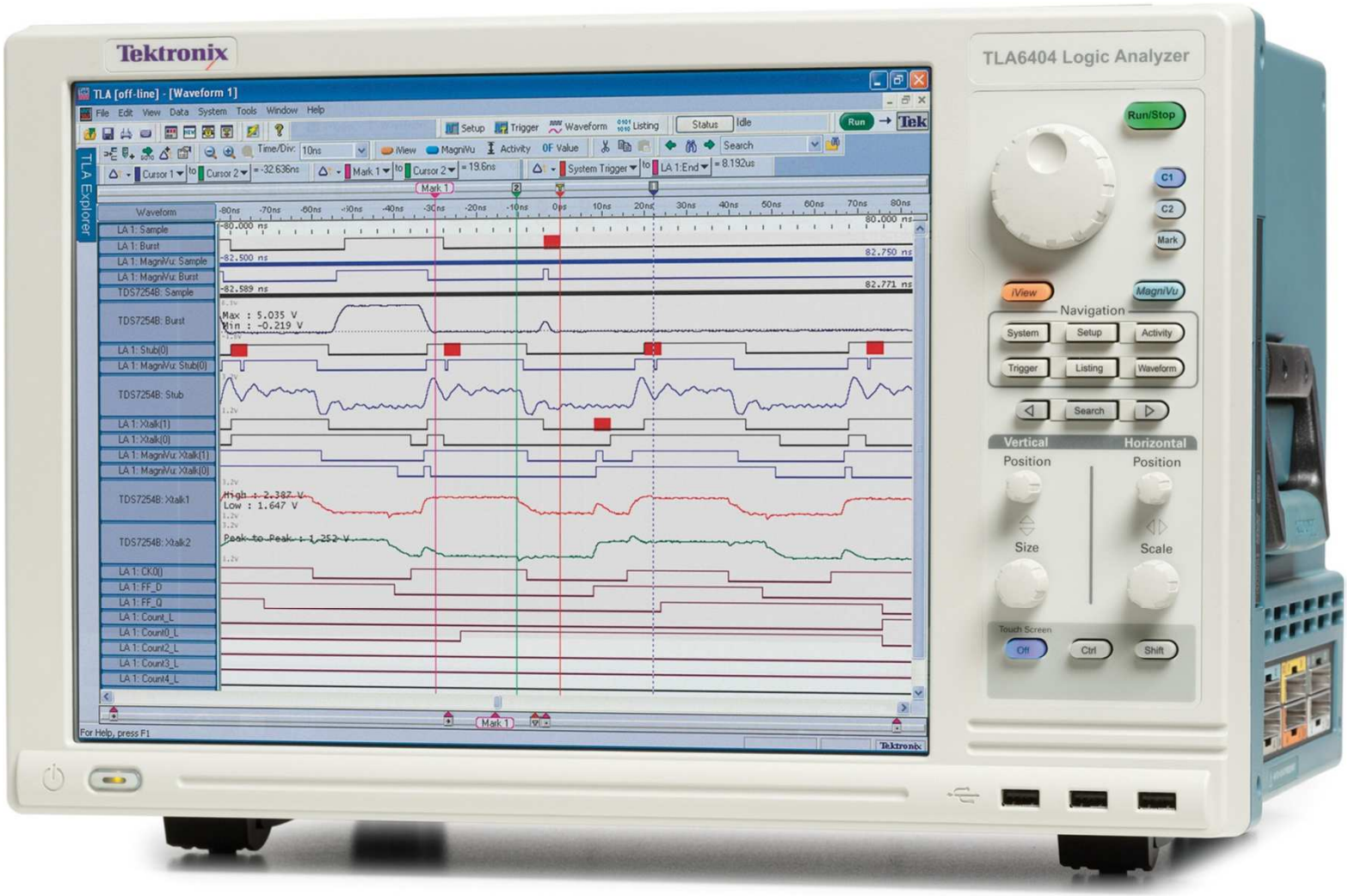
(a) Arbitrary waveform generator

(b) Function generator

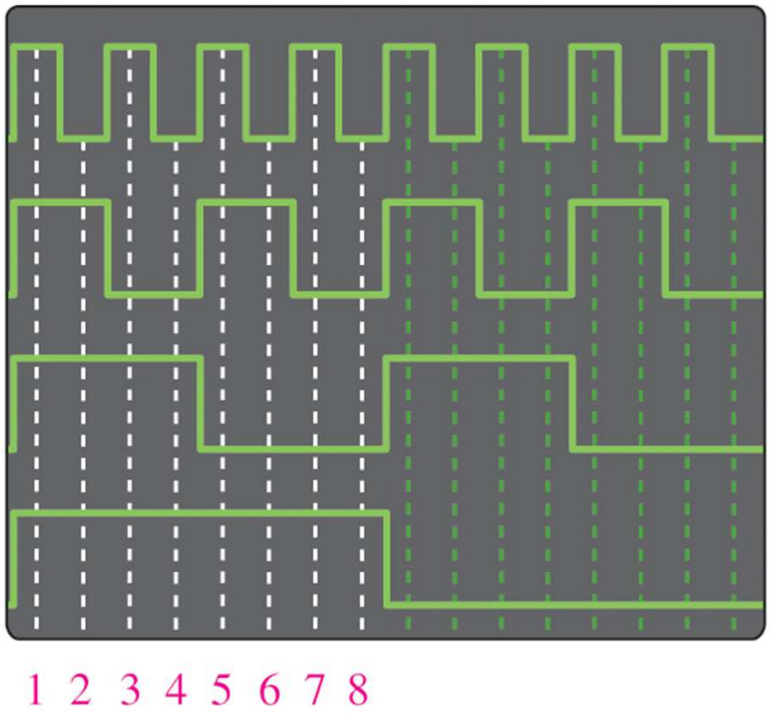
Typical digital oscilloscope with voltage probe. Used with permission from Tektronix, Inc.



Typical logic analyzer.
Used with permission from Tektronix, Inc.



Logic Analyzer Display Modes



(a) Waveform display

Sample	Binary	Hex	Time
1	1111	F	1 ns
2	1110	E	10 ns
3	1101	D	20 ns
4	1100	C	30 ns
5	1011	B	40 ns
6	1010	A	50 ns
7	1001	9	60 ns
8	1000	8	70 ns

(b) Listing display

Circuitos Elétricos e Sistemas Digitais

2018-2019 - 1.º Semestre

Sistemas Digitais

Number Systems, Operations, and Codes

The Decimal Number System

The decimal number system has only 10 symbols that represent quantity: 0 through 9. If you want to represent a value greater than 9, you must use more than one digit.

The column weights of decimal numbers are powers of ten that increase from right to left beginning with $10^0 = 1$:

$$\ldots 10^5 \;\; 10^4 \;\; 10^3 \;\; 10^2 \;\; 10^1 \;\; 10^0 .$$

For fractional decimal numbers, the column weights are negative powers of ten that decrease from left to right:

$$\ldots \; 10^2 \;\; 10^1 \;\; 10^0 . \; 10^{-1} \;\; 10^{-2} \;\; 10^{-3} \; \ldots$$

The Decimal Number System

Decimal numbers can be expressed as the sum of the products of each digit times the column value for that digit. Thus, the number 9240 can be expressed as

$$(9 \times 10^3) + (2 \times 10^2) + (4 \times 10^1) + (0 \times 10^0)$$

or

$$(9 \times 1{,}000) + (2 \times 100) + (4 \times 10) + (0 \times 1)$$

Express the number 480.52 as the sum of values of each digit.

$$480.52 = \mathbf{(4 \times 10^2) + (8 \times 10^1) + (0 \times 10^0) + (5 \times 10^{-1}) + (2 \times 10^{-2})}$$

The Binary Number System

The binary number system is used in digital systems. Binary has only two symbols that are used to represent quantities: 0 and 1.

The column weights of binary numbers are **powers of two** that increase from right to left beginning with $2^0 = 1$:

$$\ldots\ 2^5\ \ 2^4\ \ 2^3\ \ 2^2\ \ 2^1\ \ 2^0.$$

For fractional binary numbers, the column weights are negative powers of two that decrease from left to right:

$$\ldots\ 2^2\ \ 2^1\ \ 2^0.\ 2^{-1}\ \ 2^{-2}\ \ 2^{-3}\ \ 2^{-4}\ \ldots$$

The Binary Number System

A binary counting sequence for numbers from zero to fifteen is shown.

Notice the pattern of zeros and ones in each column.

Digital counters frequently have this same pattern of digits:

TABLE 2–1

DECIMAL NUMBER	BINARY NUMBER			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Pesos binários
Binary Weights

The positional weights for binary numbers are assigned as shown below.

TABLE 2–2 • Binary weights.

POSITIVE POWERS OF TWO (WHOLE NUMBERS)									NEGATIVE POWERS OF TWO (FRACTIONAL NUMBER)					
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
256	128	64	32	16	8	4	2	1	1/2 0.5	1/4 0.25	1/8 0.125	1/16 0.625	1/32 0.03125	1/64 0.015625

Binary-to-Decimal Conversion

The decimal equivalent of a binary number can be determined by adding the column values of all of the bits that are 1 and discarding all of the bits that are 0.

Convert the binary number 100101.01 to decimal.

Start by writing the column weights; then add the weights that correspond to each 1 in the number.

2^5	2^4	2^3	2^2	2^1	2^0.	2^{-1}	2^{-2}	
32	16	8	4	2	1	0.5	0.25	
1	0	0	1	0	1.	0	1	
32			+4		+1		+0.25 =	**37.25**

Decimal-to-Binary Conversions

You can convert a decimal whole number to binary by reversing the procedure. Write the decimal weight of each column and place 1's in the columns that sum to the decimal number.

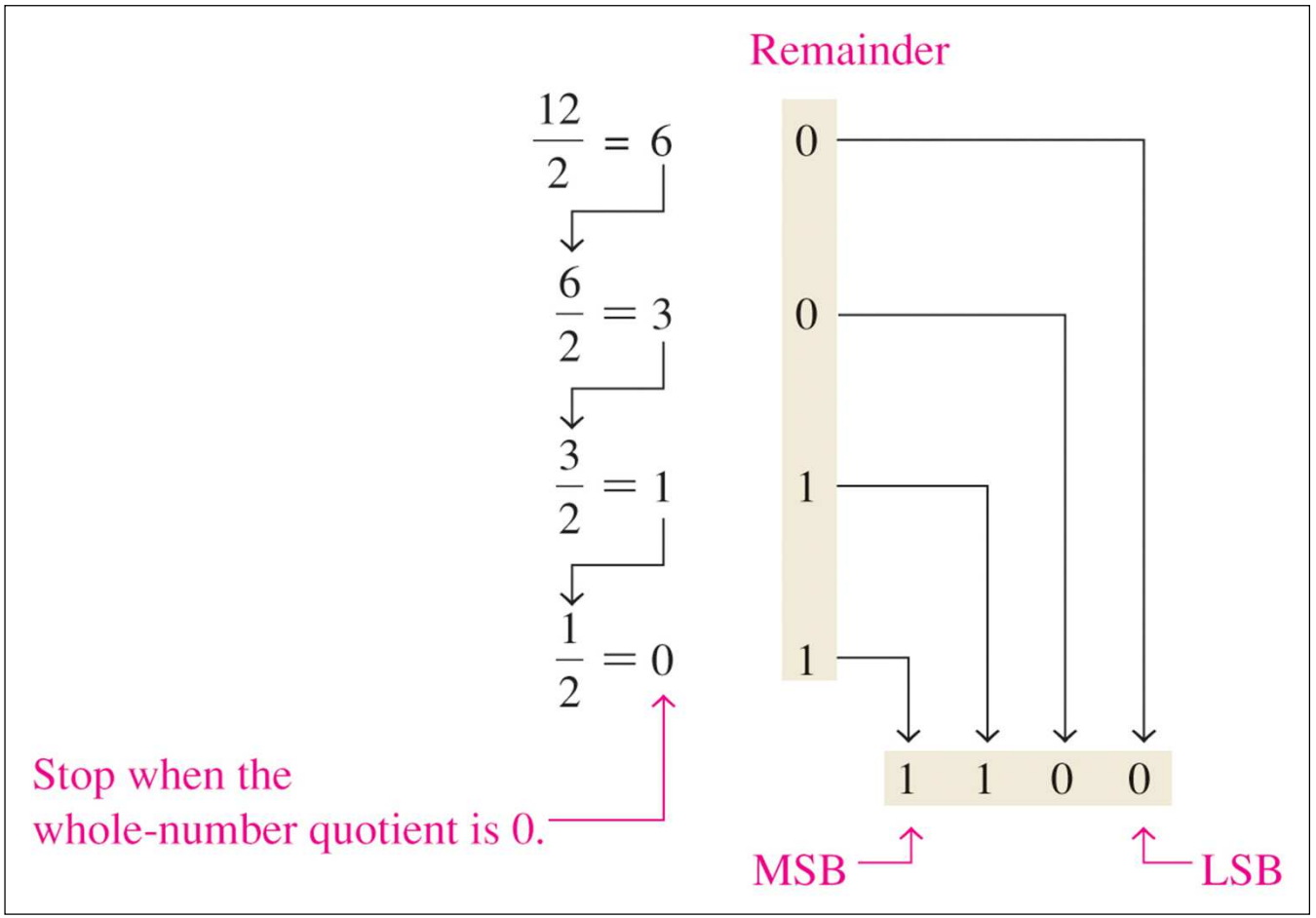
Convert the decimal number 49 to binary.

The column weights double in each position to the right. Write down column weights until the last number is larger than the one you want to convert.

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64	32	16	8	4	2	1
0	1	1	0	0	0	1

Decimal-to-Binary Conversions

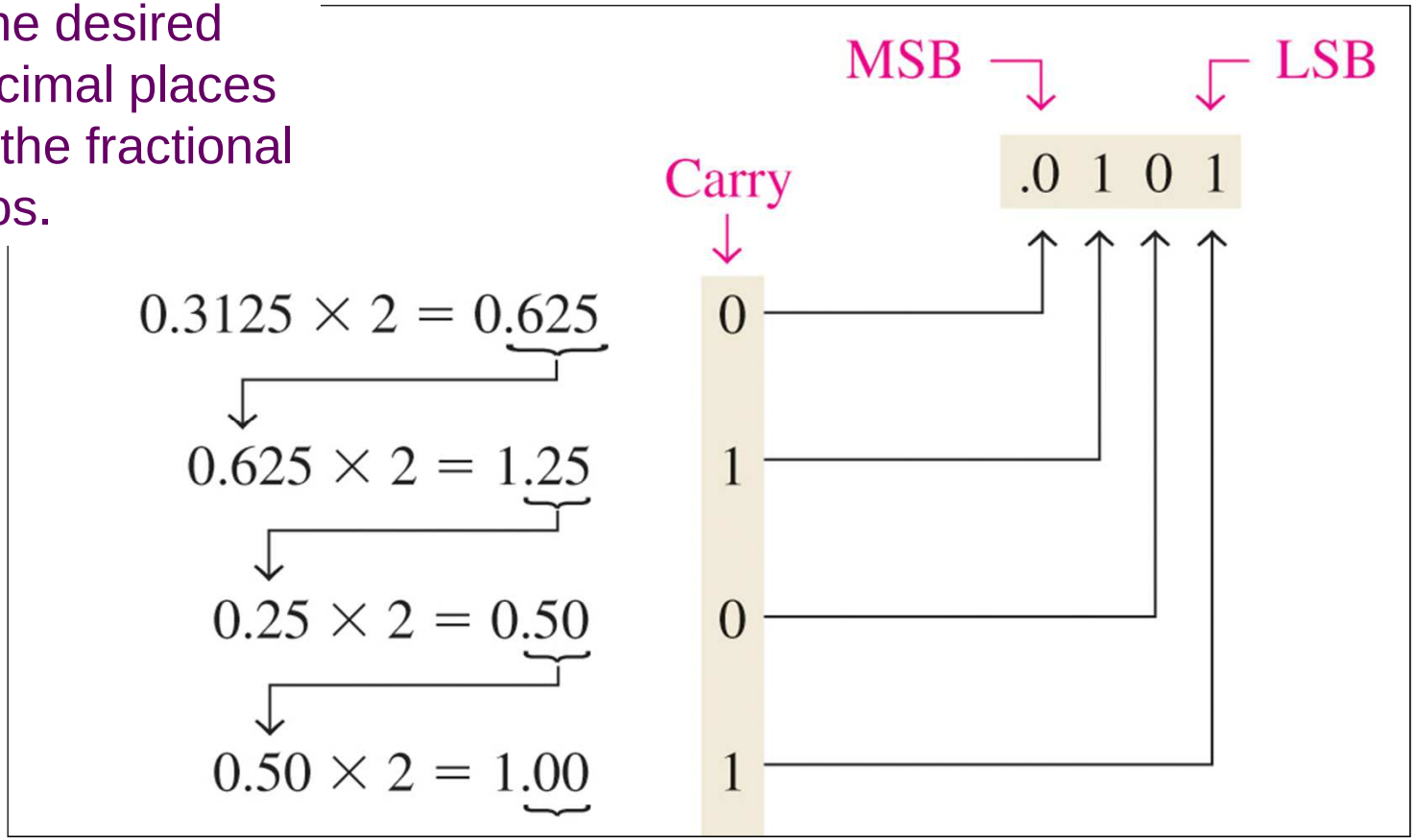
You can also convert a decimal number to binary using the “repeated division-by-2” method. This method is used here to convert 12 to binary.



Decimal-to-Binary Conversions

You can convert a decimal fraction to binary by repeatedly *multiplying* the fractional results of successive multiplications by 2. The carries form the binary number.

Continue to the desired number of decimal places or stop when the fractional part is all zeros.



Binary Addition

The rules for binary addition are

0 + 0 = 0	Sum = 0, carry = 0
0 + 1 = 0	Sum = 1, carry = 0
1 + 0 = 0	Sum = 1, carry = 0
1 + 1 = 10	Sum = 0, carry = 1

O bit de Carry *corresponde* ao decimal "vai-um". O carry indica se devemos ou não somar 1 à próxima coluna à esquerda durante o cálculo.

When an input carry = 1 due to a previous result, the rules are

1 + 0 + 0 = 01	Sum = 1, carry = 0
1 + 0 + 1 = 10	Sum = 0, carry = 1
1 + 1 + 0 = 10	Sum = 0, carry = 1
1 + 1 + 1 = 10	Sum = 1, carry = 1

Binary Subtraction

The rules for binary subtraction are

$$0 - 0 = 0$$
$$1 - 1 = 0$$
$$1 - 0 = 1$$
$$10 - 1 = 1 \text{ with a borrow of 1}$$

Subtract the binary number 00111 from 10101 and show the equivalent decimal subtraction.

$$\begin{array}{rr} {\scriptstyle 1\,1\,1} & \\ 10101 & 21 \\ 00111 & 7 \\ \hline 01110 & = 14 \end{array}$$

Binary Multiplication

The rules for binary multiplication are:

$0 \times 0 = 0$ $1 \times 0 = 0$ $0 \times 1 = 0$ $1 \times 1 = 1$

Perform the following binary multiplications:

(a) 11×11 **(b)** 101×111

SOLUTION

(a)

	11	3
	× 11	× 3
Partial products	11	9
	+11	
	1001	

(b)

	111	7
	× 101	× 5
Partial products	111	35
	000	
	+111	
	100011	

Binary Division

Binary division is performed in the same manner as decimal division, as demonstrated below.

Perform the following binary divisions:

(a) $110 \div 11$ **(b)** $110 \div 10$

SOLUTION

(a)

$$\begin{array}{r} \mathbf{10} \\ 11\overline{)110} \\ \underline{11} \\ 000 \end{array} \qquad \begin{array}{r} 2 \\ 3\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

(b)

$$\begin{array}{r} \mathbf{11} \\ 10\overline{)110} \\ \underline{10} \\ 10 \\ \underline{10} \\ 00 \end{array} \qquad \begin{array}{r} 3 \\ 2\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

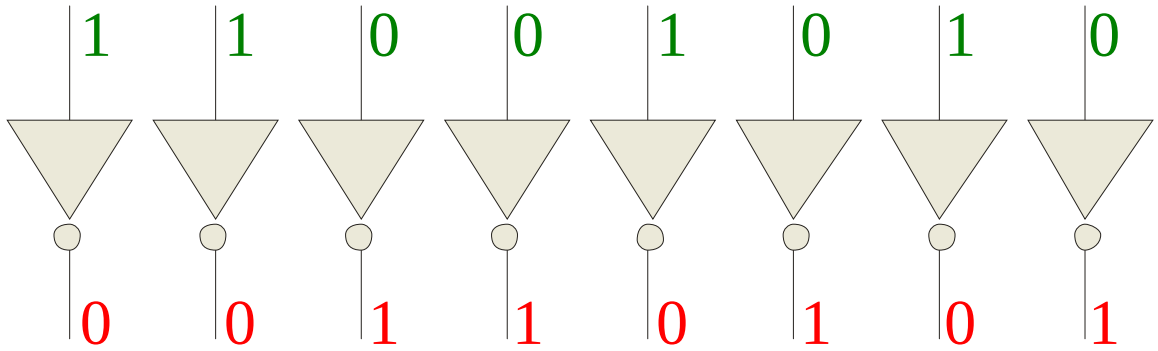
Complemento de 1

1’s Complement

The 1’s complement of a binary number is the number that is formed by reversing (inverting) all the digits. To form the 1’s complement, change all 0’s to 1’s and all 1’s to 0’s.

For example, the 1’s complement of 11001010 is: 00110101

In digital circuits, the 1’s complement is formed by using inverters:



Complemento de 2

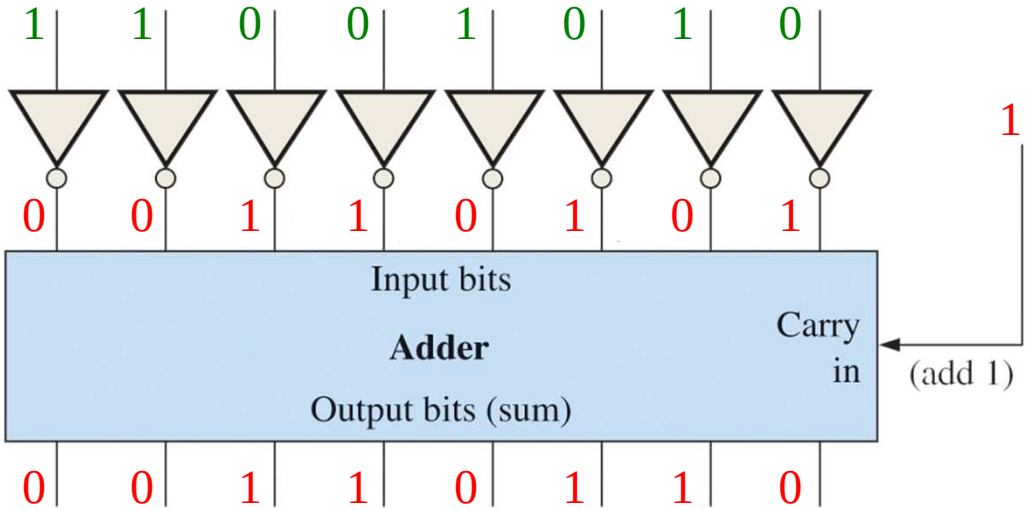
2's Complement

The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement.

Recall that the 1's complement of 11001010 is

For the 2's complement, add 1:

$$\begin{array}{r} 00110101 \text{ (1's complement)} \\ +1 \\ \hline 00110110 \text{ (2's complement)} \end{array}$$

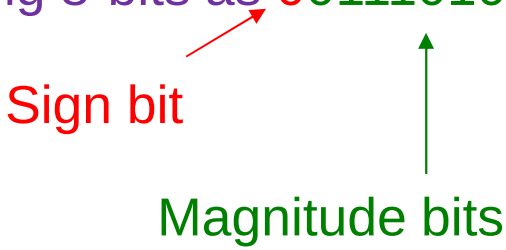


Números binários com sinal
Signed Binary Numbers

There are several ways to represent signed binary numbers. In all cases, the MSB in a signed number is the **sign bit**, that tells you if the number is positive or negative.

Computers use a modified 2's complement for signed numbers. Positive numbers are stored in *true* form (with a 0 for the sign bit) and negative numbers are stored in *complement* form (with a 1 for the sign bit).

For example, the positive number 58 is written using 8-bits as 00111010 (true form).



Signed Binary Numbers

Negative numbers are written as the 2's complement of the corresponding positive number.

The number –58 is written as:

–58 = 11000110 (complement form)

Sign bit

Magnitude bits

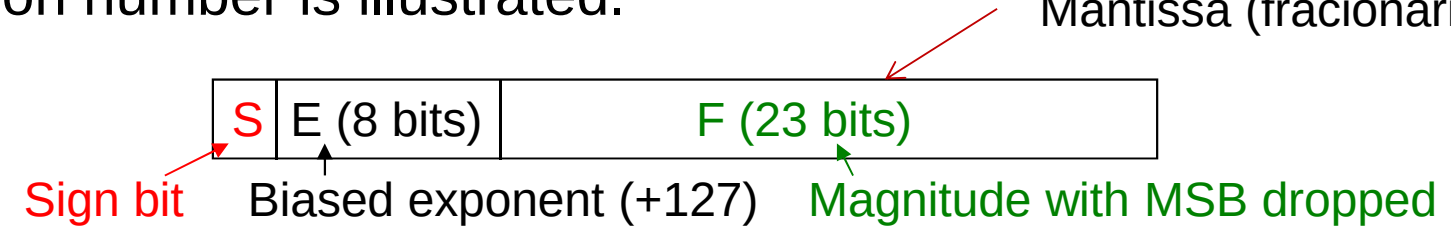
An easy way to read a signed number that uses this notation is to assign the sign bit a column weight of –128 (for an 8-bit number). Then add the column weights for the 1's.

Assuming that the sign bit = –128, show that 11000110 = –58 as a 2's complement signed number:

Column weights:	-128	64	32	16	8	4	2	1.	
	1	1	0	0	0	1	1	0	
	-128	+64				+4	+2		= –58

Números em ponto (vírgula) flutuante
Floating Point Numbers

Floating point notation is capable of representing very large or small numbers by using a form of scientific notation. A 32-bit single precision number is illustrated.



Express the speed of light, *c*, in single precision floating point notation. (*c* = 0.2998 x 10^9 m/s)

In binary, *c* = 0001 0001 1101 1110 1001 0101 1100 0000_2.

In scientific notation, *c* = 1.001 1101 1110 1001 0101 1100 0000 x 2^{28}.

S = 0 because the number is positive. E = 28 + 127 = 155_{10} = 1001 1011_2. F is the next 23 bits after the first 1 is dropped.

In floating point notation, *c* =

0	10011011	001 1101 1110 1001 0101 1100

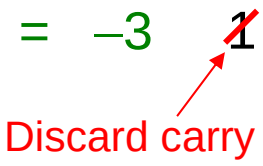
Arithmetic Operations With Signed Numbers

Using the signed number notation with negative numbers in 2's complement form simplifies addition and subtraction of signed numbers.

Rules for **addition**: Add the two signed numbers. Discard any final carries. The result is in signed form.

Examples:

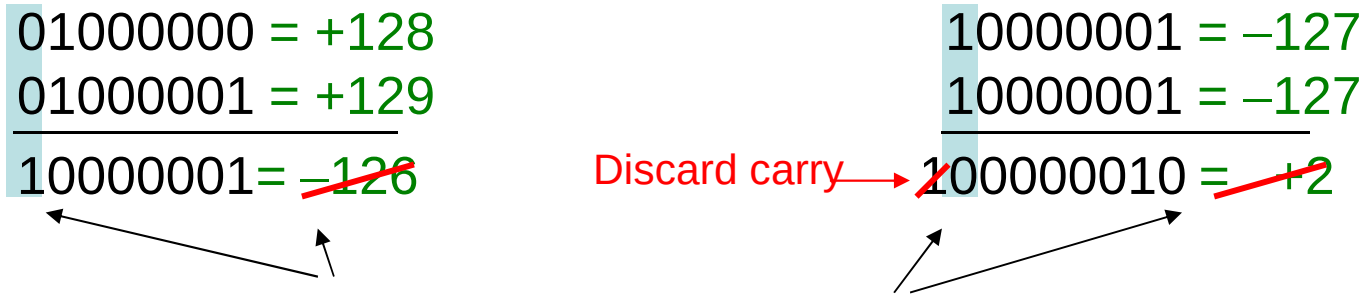
00011110 = +30	00001110 = +14	11111111 = –1
00001111 = +15	11101111 = –17	11111000 = –8
00101101 = +45	11111101 = –3	111110111 = –9



Arithmetic Operations With Signed Numbers

Note that if the number of bits required for the answer is exceeded, **overflow will occur**. This occurs only if both numbers have the same sign. The overflow is indicated by an incorrect sign bit.

Two examples are:



Wrong! The answer is incorrect and the sign bit has changed.

Arithmetic Operations With Signed Numbers

Rules for **subtraction**: 2's complement the subtrahend and add the numbers. Discard any final carries. The result is in signed form.

Repeat the examples done previously, but subtract:

00011110 (+30)	00001110 (+14)	11111111 (–1)
– 00001111 –(+15)	– 11101111 –(–17)	– 11111000 –(–8)

2's complement subtrahend and add:

00011110 = +30	00001110 = +14	11111111 = –1
11110001 = –15	00010001 = +17	00001000 = +8
~~1~~00001111 = +15	00011111 = +31	~~1~~00000111 = +7

Discard carry

Discard carry

Hexadecimal Numbers

Hexadecimal uses sixteen characters to represent numbers: the numbers 0 through 9 and the letters A through F.

Large binary number can easily be converted to hexadecimal by dividing it into 4-bit groups and converting each into its equivalent hexadecimal character.

Express $1001\ 0110\ 0000\ 1110_2$ in hexadecimal:

Group the binary number by 4-bits starting from the right.

Thus, 1001011000001110 = 960E

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Hexadecimal Numbers

Hexadecimal is a weighted number system. The column weights are powers of 16, which increase from right to left.

Column weights	16^3	16^2	16^1	16^0
	4096	256	16	1

Express $1A2F_{16}$ in decimal.

Start by writing the column weights:

4096	256	16	1
1	A	2	F_{16}

$1(4096) + 10(256) + 2(16) + 15(1) = \mathbf{6703_{10}}$

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Números octais
Octal Numbers

Octal uses the symbols 0 through 7 to represent numbers; 8 and 9 are not used in octal.

Binary number can easily be converted to octal by grouping bits 3 at a time and writing the equivalent octal character for each group.

Express $1\ 001\ 011\ 000\ 001\ 110_2$ in octal:

Group the binary number by 3-bits starting from the right. Thus, 1001011000001110 = **113016_8**

Decimal	Octal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111

Octal Numbers

Octal is also a weighted number system. The column weights are powers of 8, which increase from right to left.

Column weights	8^3	8^2	8^1	8^0
	512	64	8	1

Express 3702_8 in decimal.

Start by writing the column weights:

512	64	8	1
3	7	0	2

$3(512) + 7(64) + 0(8) + 2(1) = \mathbf{1986_{10}}$

Decimal	Octal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111

Decimal Codificado em Binário

BCD

Binary coded decimal (BCD) is a weighted code that is commonly used in digital systems, such as clock displays, where it is necessary to show decimal numbers.

The table illustrates the difference between straight binary and BCD.

BCD represents each decimal digit with a 4-bit code.

Notice that the codes 1010 through 1111 are not used in BCD.

Decimal	Binary	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	00010000
11	1011	00010001
12	1100	00010010
13	1101	00010011
14	1110	00010100
15	1111	00010101

Decimal Codificado em Binário
BCD

You can think of BCD in terms of column weights in groups of four bits. For an 8-bit BCD number, the column weights are:

80 40 20 10 8 4 2 1.

Example: What are the column weights for the BCD number 1000 0011 0101 1001?

8000 4000 2000 1000 800 400 200 100 80 40 20 10 8 4 2 1

Note that you can add the column weights where there is a 1 to obtain the equivalent decimal number. In this case:

$8000 + 200 + 100 + 40 + 10 + 8 + 1 = \mathbf{8359}_{10}$

Códigos binários

n.º de bits	Gama de variação	Tipos pré-definidos
N	$0..2^N-1$	-
8	0…255	byte
16	0…65535	Word / palavra

Códigos digitais: Código Gray

Gray Code

Gray code is an unweighted code that has a single bit change between one code word and the next in a sequence.

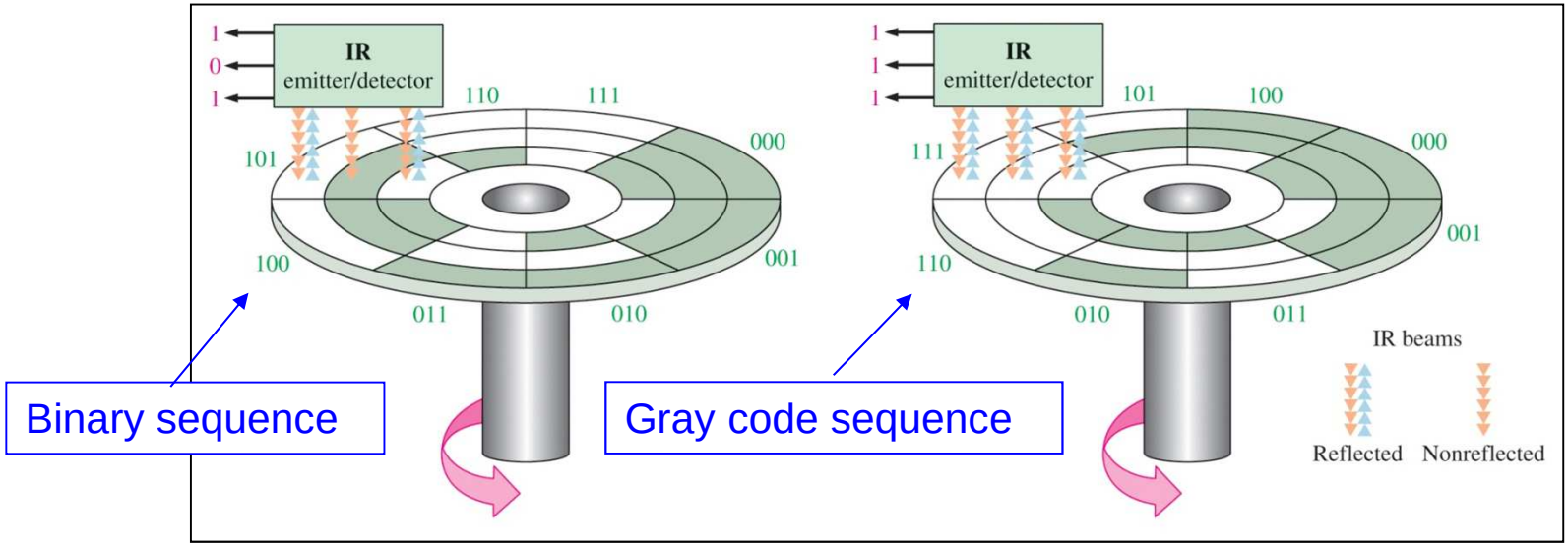
Gray code is used to avoid problems in systems where an error can occur if more than one bit changes at a time.

Decimal	Binary	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	**0011**	**0010**
4	**0100**	**0110**
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Exemplo de Aplicação do código Gray

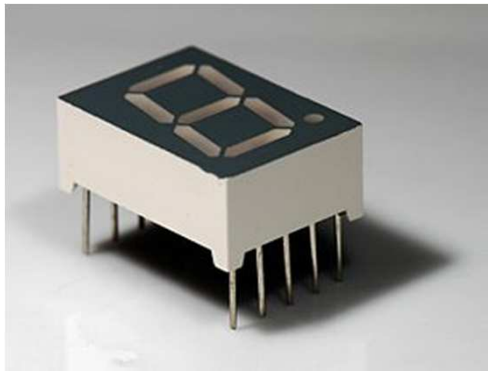
Gray Code: Shaft Encoder

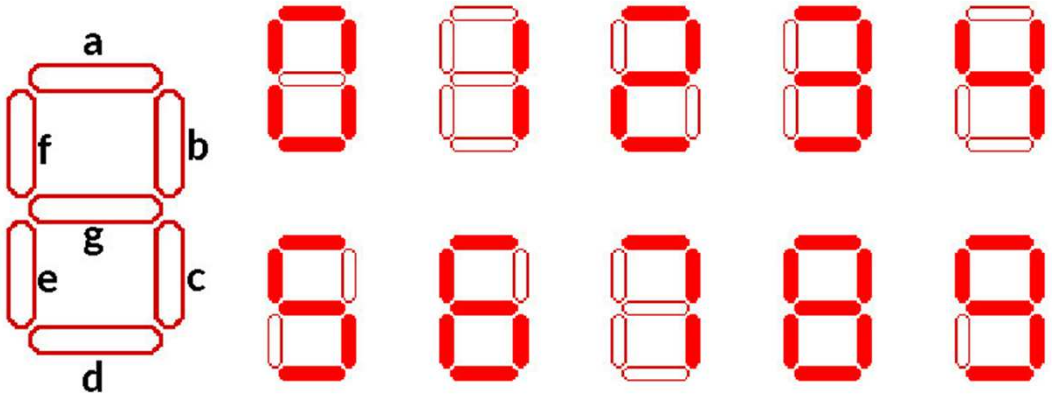
A shaft encoder is a typical application. Three IR emitter/detectors are used to encode the position of the shaft. The encoder on the left uses binary and can have three bits change together, creating a potential error. The encoder on the right uses gray code and only 1-bit changes, eliminating potential errors.



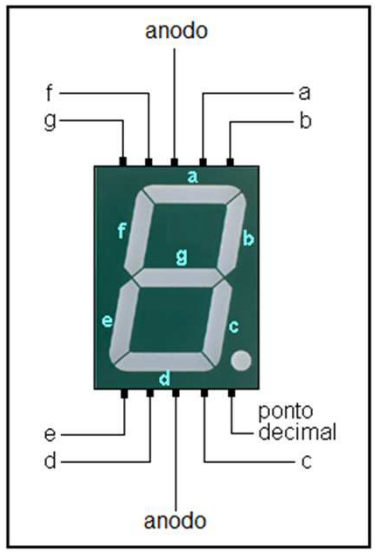
A 7-Segment Display Multiplexer

Um “display” de sete segmentos é composto de sete LEDs, os quais podem ser ligados ou desligados individualmente, e podem ser combinados para produzir representações simplificadas de algarismos árabes. Frequentemente, os sete segmentos são dispostos de forma oblíqua ou itálica, o que melhora a legibilidade. Os sete segmentos são dispostos num retângulo com dois segmentos verticais em cada lado e um segmento horizontal em cima e em baixo. O sétimo segmento bisseta o retângulo horizontalmente. Os segmentos de um “display” de sete segmentos são definidos pelas letras de **a** a **g**, conforme indicado à direita, onde o ponto decimal opcional DP (um "oitavo segmento") é usado para a exibição de números não-inteiros.





A 7-Segment Display Multiplexer



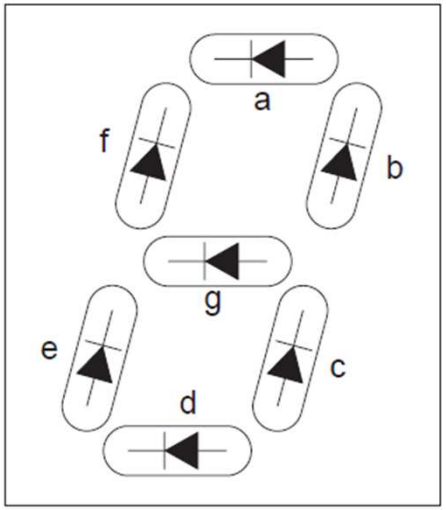
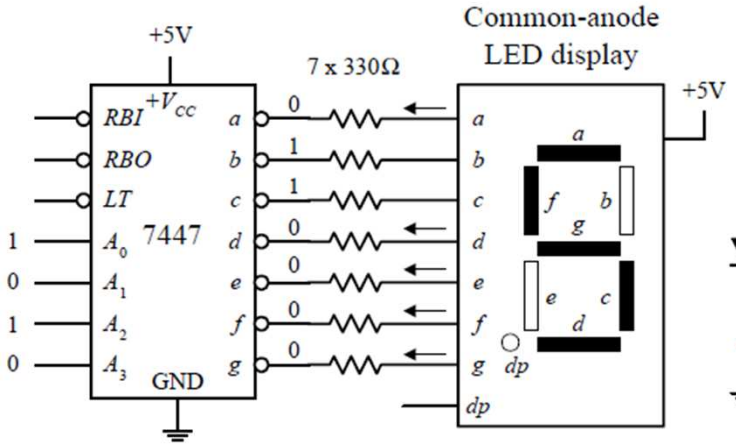


Figure 10-1. Seven-Segment Display Uses Seven LED Bars

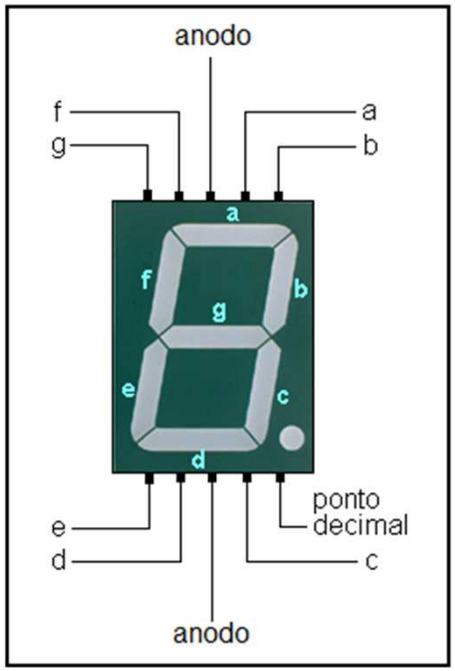
7447 BCD-to-7-segment decoder/LED driver IC

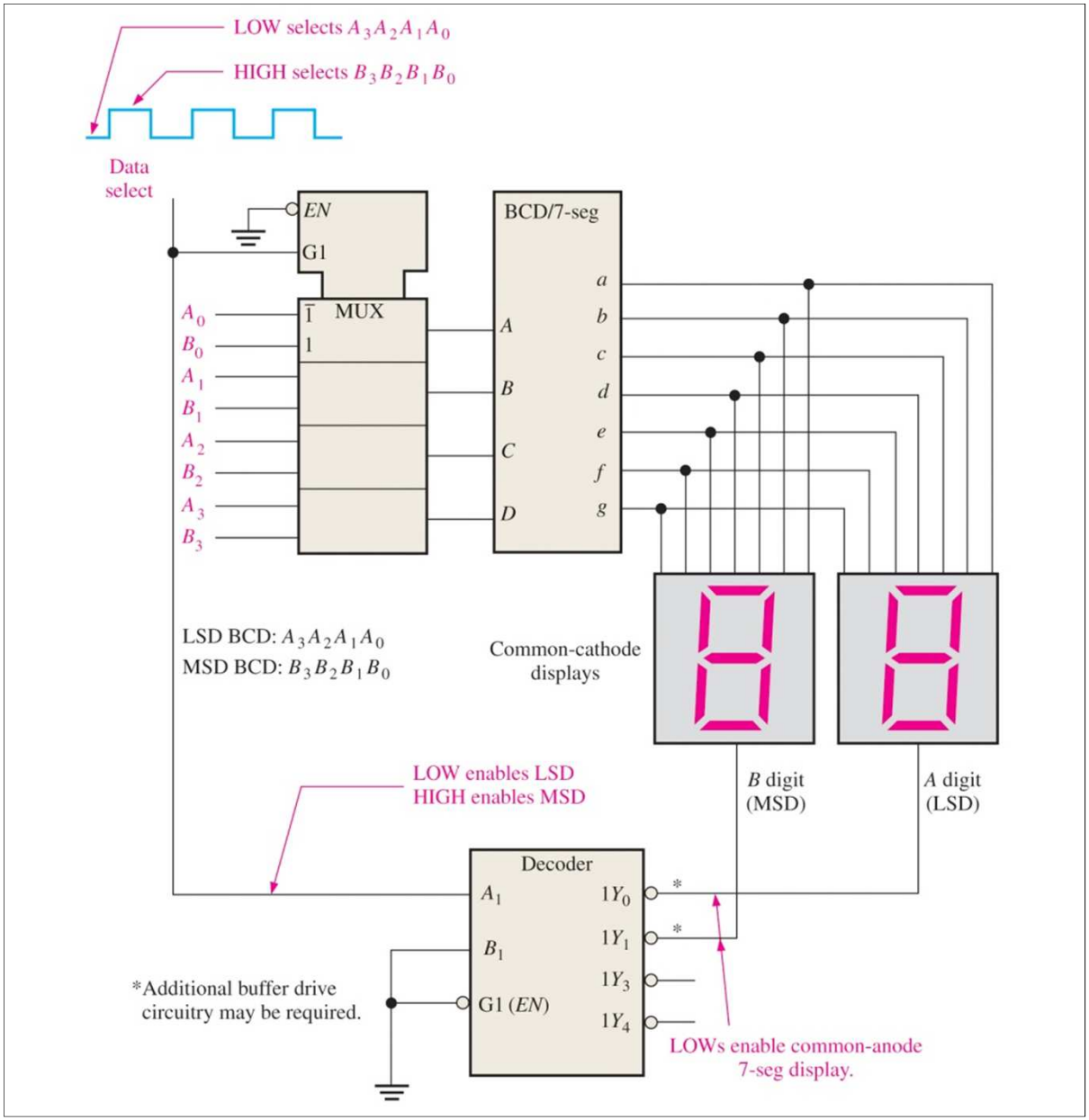


Inputs $A_3 A_2 A_1 A_0$	Digit	7447 outputs $\overline{a}\ \overline{b}\ \overline{c}\ \overline{d}\ \overline{e}\ \overline{f}\ \overline{g}$	Lit LED segment $a\ b\ c\ d\ e\ f\ g$
0 0 0 0	0	0 0 0 0 0 0 1	✹ ✹ ✹ ✹ ✹ ✹ ○
0 0 0 1	1	1 0 0 1 1 1 1	○ ✹ ✹ ○ ○ ○ ○
0 0 1 0	2	0 0 1 0 0 1 0	✹ ✹ ○ ✹ ✹ ○ ✹
0 0 1 1	3	0 0 0 0 1 1 0	✹ ✹ ✹ ✹ ○ ○ ✹
0 1 0 0	4	1 0 0 1 1 0 0	○ ✹ ✹ ○ ○ ✹ ✹
0 1 0 1	5	0 1 0 0 1 0 0	✹ ○ ✹ ✹ ○ ✹ ✹
0 1 1 0	6	0 1 0 0 0 0 0	✹ ○ ✹ ✹ ✹ ✹ ✹
0 1 1 1	7	0 0 0 1 1 1 1	✹ ✹ ✹ ○ ○ ○ ○
1 0 0 0	8	0 0 0 0 0 0 0	✹ ✹ ✹ ✹ ✹ ✹ ✹
1 0 0 1	9	0 0 0 1 1 0 0	✹ ✹ ✹ ○ ○ ✹ ✹

1 = HIGH voltage level, 0 = LOW voltage level,
✹ = LED segment ON, ○ = LED segment OFF

A 7-Segment Display Multiplexer





Códigos alfanuméricos: código ASCII

ASCII is a code for alphanumeric characters and control characters. In its original form, ASCII encoded 128 characters and symbols using 7-bits. The first 32 characters are control characters, that are based on obsolete teletype requirements, so these characters are generally assigned to other functions in modern usage.

In 1981, IBM introduced extended ASCII, which is an 8-bit code and increased the character set to 256. Other extended sets (such as Unicode) have been introduced to handle characters in languages other than English.

Código ASCII

TABLE 2-7

American Standard Code for Information Interchange (ASCII).

Control Characters				Graphic Symbols											
Name	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex
NUL	0	0000000	00	space	32	0100000	20	@	64	1000000	40	′	96	1100000	60
SOH	1	0000001	01	!	33	0100001	21	A	65	1000001	41	a	97	1100001	61
STX	2	0000010	02	”	34	0100010	22	B	66	1000010	42	b	98	1100010	62
ETX	3	0000011	03	#	35	0100011	23	C	67	1000011	43	c	99	1100011	63
EOT	4	0000100	04	$	36	0100100	24	D	68	1000100	44	d	100	1100100	64
ENQ	5	0000101	05	%	37	0100101	25	E	69	1000101	45	e	101	1100101	65
ACK	6	0000110	06	&	38	0100110	26	F	70	1000110	46	f	102	1100110	66
BEL	7	0000111	07	’	39	0100111	27	G	71	1000111	47	g	103	1100111	67
BS	8	0001000	08	(	40	0101000	28	H	72	1001000	48	h	104	1101000	68
HT	9	0001001	09	)	41	0101001	29	I	73	1001001	49	i	105	1101001	69
LF	10	0001010	0A	*	42	0101010	2A	J	74	1001010	4A	j	106	1101010	6A
VT	11	0001011	0B	+	43	0101011	2B	K	75	1001011	4B	k	107	1101011	6B
FF	12	0001100	0C	,	44	0101100	2C	L	76	1001100	4C	l	108	1101100	6C
CR	13	0001101	0D	−	45	0101101	2D	M	77	1001101	4D	m	109	1101101	6D
SO	14	0001110	0E	.	46	0101110	2E	N	78	1001110	4E	n	110	1101110	6E
SI	15	0001111	0F	/	47	0101111	2F	O	79	1001111	4F	o	111	1101111	6F
DLE	16	0010000	10	0	48	0110000	30	P	80	1010000	50	p	112	1110000	70
DC1	17	0010001	11	1	49	0110001	31	Q	81	1010001	51	q	113	1110001	71
DC2	18	0010010	12	2	50	0110010	32	R	82	1010010	52	r	114	1110010	72
DC3	19	0010011	13	3	51	0110011	33	S	83	1010011	53	s	115	1110011	73
DC4	20	0010100	14	4	52	0110100	34	T	84	1010100	54	t	116	1110100	74
NAK	21	0010101	15	5	53	0110101	35	U	85	1010101	55	u	117	1110101	75
SYN	22	0010110	16	6	54	0110110	36	V	86	1010110	56	v	118	1110110	76
ETB	23	0010111	17	7	55	0110111	37	W	87	1010111	57	w	119	1110111	77
CAN	24	0011000	18	8	56	0111000	38	X	88	1011000	58	x	120	1111000	78
EM	25	0011001	19	9	57	0111001	39	Y	89	1011001	59	y	121	1111001	79
SUB	26	0011010	1A	:	58	0111010	3A	Z	90	1011010	5A	z	122	1111010	7A
ESC	27	0011011	1B	;	59	0111011	3B	[	91	1011011	5B	{	123	1111011	7B
FS	28	0011100	1C	<	60	0111100	3C	\	92	1011100	5C	|	124	1111100	7C
GS	29	0011101	1D	=	61	0111101	3D	]	93	1011101	5D	}	125	1111101	7D
RS	30	0011110	1E	>	62	0111110	3E	^	94	1011110	5E	~	126	1111110	7E
US	31	0011111	1F	?	63	0111111	3F	_	95	1011111	5F	Del	127	1111111	7F

Unicode

Códigos de deteção e correção de erros:
Método da paridade para a deteção de erros
Parity Method for Error Detection

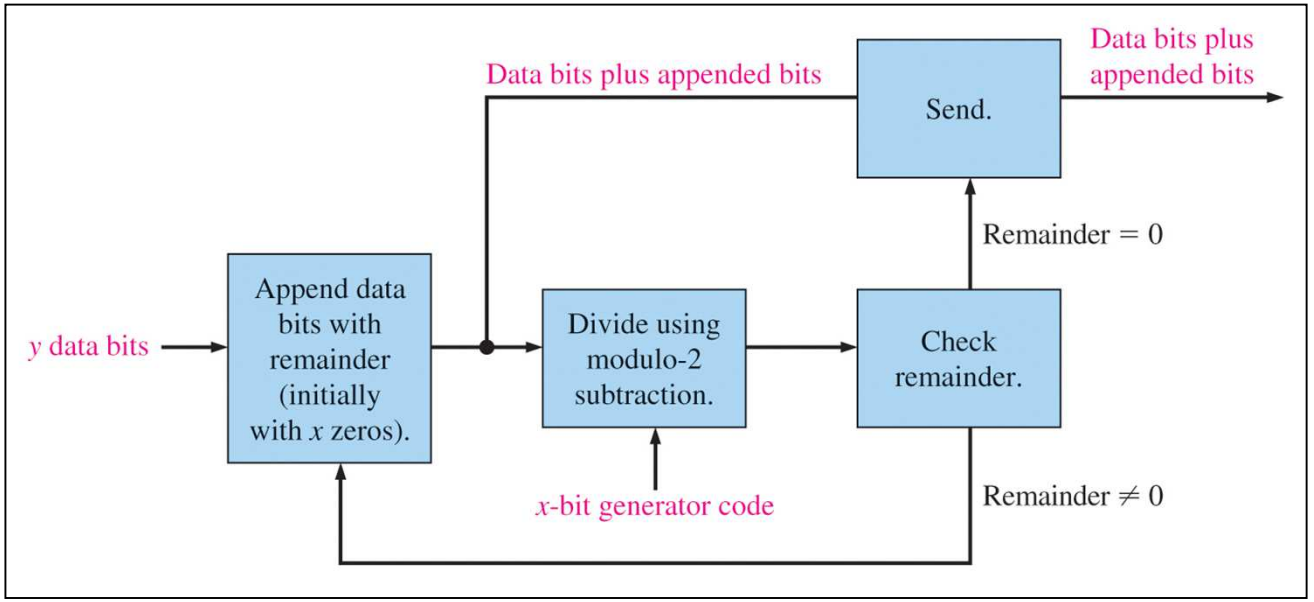
The parity method is a method of error detection for simple transmission errors involving one bit (or an odd number of bits). A **parity bit** is an "extra" bit attached to a group of bits to force the number of 1's to be either even (*even parity*) or odd (*odd parity*).

The ASCII character for "a" is 1100001 and for "A" is 1000001. What is the correct bit to append to make both of these have **odd parity**?

The ASCII "a" has an **odd number** of bits that are equal to 1; therefore the parity bit is 0. The ASCII "A" has an **even number** of bits that are equal to 1; therefore the parity bit is 1.

Códigos de deteção e correção de erros:
Cyclic Redundancy Check

The cyclic redundancy check (CRC) is an error detection method that can detect multiple errors in larger blocks of data. At the sending end, a **checksum** is appended to a block of data. At the receiving end, the checksum is generated and compared to the sent checksum. If the check sums are the same, no error is detected.



Códigos de deteção e correção de erros:
O Código de correção de erro Hamming

O **código de Hamming** foi desenvolvido por Richard Hamming, e é utilizado no processamento de sinal e nas telecomunicações. A sua utilização permite a transferência e armazenamento de dados de forma segura e eficiente.

Nas telecomunicações os códigos de Hamming utilizados são generalizações do Hamming (7,4). Estes podem detetar erros até dois bits e corrigir até um bit.

Em contraste, o código de paridade não pode corrigir erros, e pode detetar apenas um número impar de erros.

Devido à sua simplicidade os códigos Hamming, são amplamente utilizados na memória dos computadores (ECC). Neste contexto, é frequente utilizar um código de Hamming estendido com um bit de paridade extra.

Circuitos Elétricos e Sistemas Digitais

2018-2019 - 1.º Semestre

Sistemas Digitais

Portas lógicas e combinação de portas lógicas

Logic Gates and Gate Combinations

Eletrónica analógica vs eletrónica digital

Os circuitos/componentes analisados até aqui incluem-se na designação Electrónica Analógica. Os circuitos/componentes que estudamos agora pertencem ao ramo da Electrónica Digital.

A grande diferença reside, essencialmente, no seguinte aspecto: enquanto na Electrónica Analógica as sinais (quer de entrada, quer de saída) podem variar de um modo contínuo dentro de limites relativamente largos, em Electrónica Digital os sinais (quer as entradas, quer as saídas) apenas podem pertencer a duas gamas de valores.

Em circuitos digitais, as tensões assumem um número limitado de valores. Os sistemas digitais mais comuns empregam dois valores e são referidos como sistemas binários.

Circuitos digitais que operam com sinais de entrada binários e produzem sinais de saída também binários.

É costume designar esses dois intervalos de tensão por **um** e **zero**, símbolos **1** e **0**, ou alto (high) e baixo (low), ou ainda por verdadeiro e falso. Os circuitos digitais são aplicados, quase universalmente, em sistemas de comunicação, controlo, instrumentação, e, claro, em computação.

A complexidade de um circuito digital vai desde de um número pequeno de portas lógicas até computadores completos (um microprocessador) ou memórias de milhões de bits.

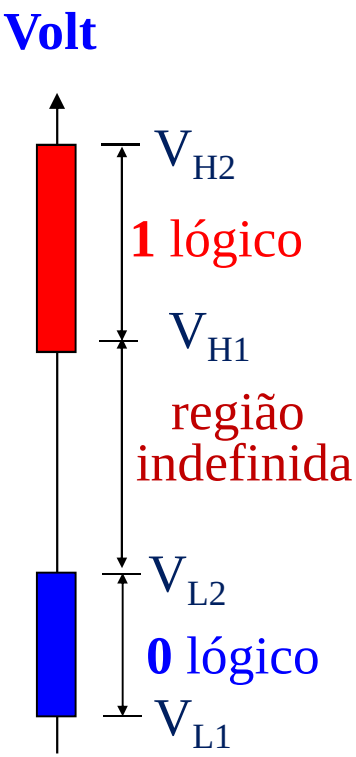
Lógica Positiva

Em circuitos binários, dois valores distintos de tensão podem representar os dois valores das variáveis binárias. Contudo, em virtude das inevitáveis tolerâncias dos componentes e efeito do ruído, que alteram por vezes os níveis de tensão, dois intervalos distintos de tensão são usualmente definidos.

Como mostra a figura abaixo, se o valor do sinal de tensão está compreendido no intervalo [V_{L1}, V_{L2}], o sinal é interpretado (pelo circuito digital) como um **0** lógico.

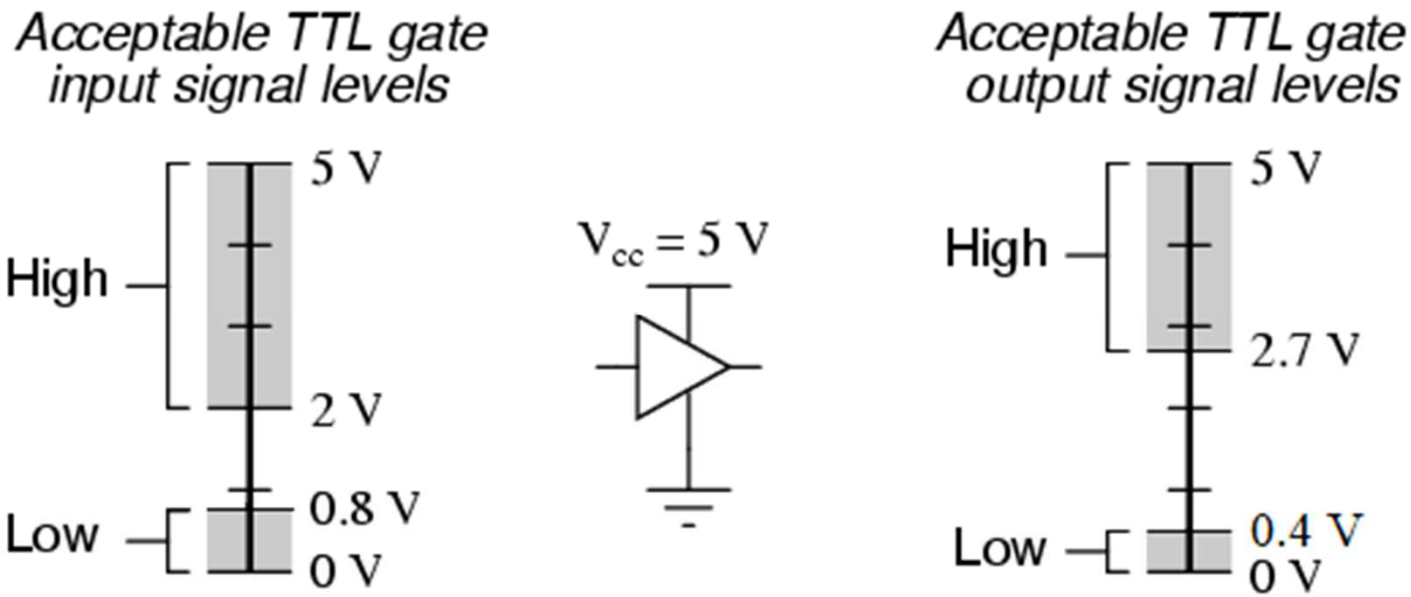
Se, por outro lado, o sinal pertence ao intervalo [V_{H1}, V_{H2}], é interpretado como **1** lógico. As duas regiões de tensão são separadas por uma região à qual não é suposto os sinais pertencerem.

Esta banda proibida representa a zona indefinida ou excluída. Uma vez que as tensões correspondentes ao 1 lógico são superiores aquelas que representam o 0 lógico, diz que os sistemas assim implementados usam **lógica positiva**. Claro que poderíamos inverter as definições e obteríamos sistemas de lógica negativa.



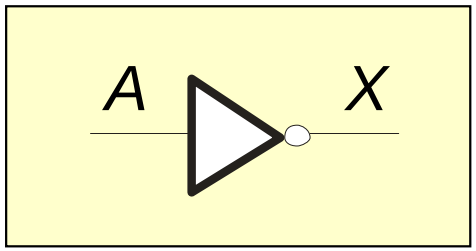
Níveis de tensão para portas lógicas TTL e CMOS

Aqui será usada a lógica positiva, e os vocábulos “alto” e “baixo” serão equivalentes a **1** e **0**, respetivamente. O intervalo de valores de tensão correspondente ao valor lógico 1 é [0, 0.9] V. O valor lógico 1 refere-se a tensões compreendidas entre 2.5 e 5 V. Tensões entre 0.9V e 2,5 V são proibidas, i.e., os circuitos não "sabem" como interpretá-las. **Nota: Quando se diz que uma tensão de entrada é zero, está-se a admitir que há uma ligação à massa, e não uma entrada flutuante**.



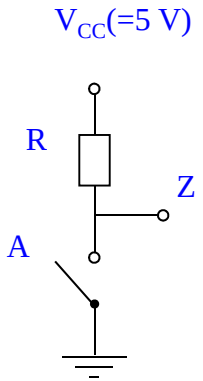
Inversor ou porta Não
The Inverter

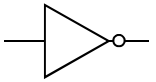
The inverter performs the Boolean **NOT** operation. When the input is LOW, the output is HIGH; when the input is HIGH, the output is LOW.



Input	Output
A	*X*
LOW (0)	HIGH (1)
HIGH (1)	LOW (0)

Função Não ou Inversora
Sejam um interruptor e uma resistência em série com a saída aos terminais do interruptor



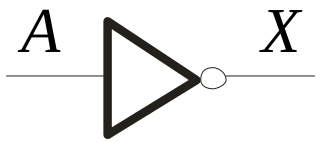


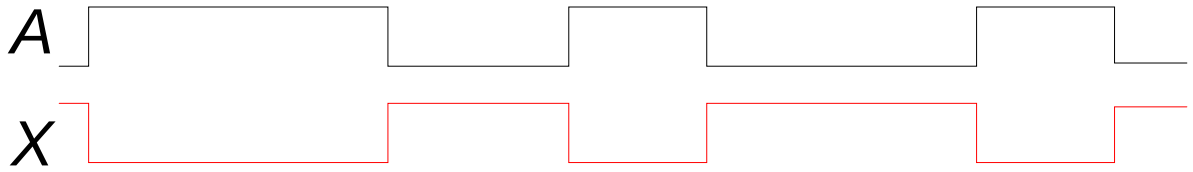
A	V_Z	*Z*
0	5	1
1	0	1

The **NOT** operation (complement) is shown with an over-bar. Thus, the Boolean expression for an inverter is $X = \overline{A}$.

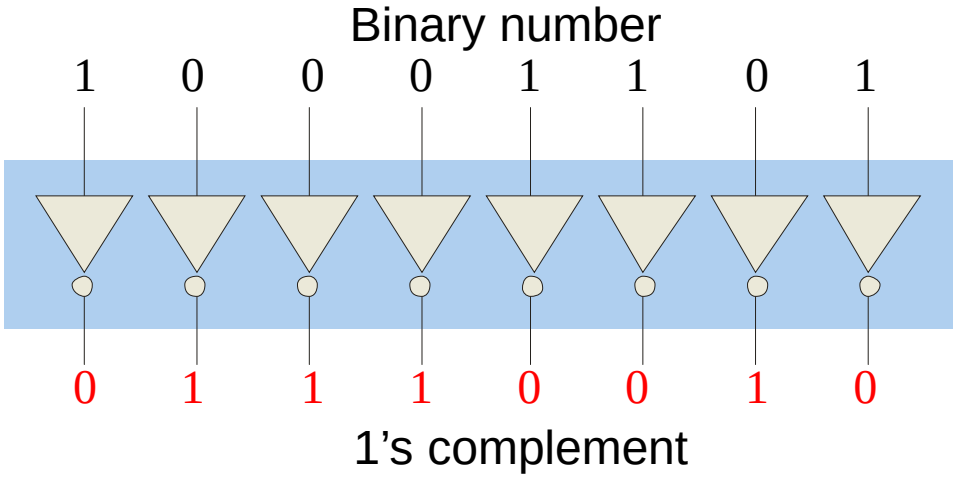
An Inverter Application

Example waveforms:





A group of inverters can be used to form the 1's complement of a binary number:



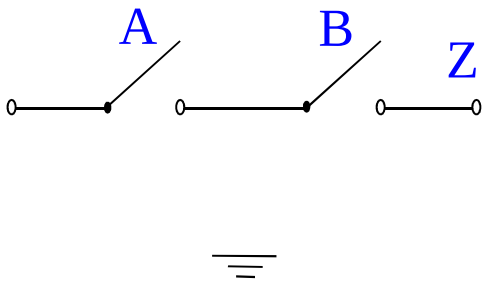
Porta E

The AND Gate

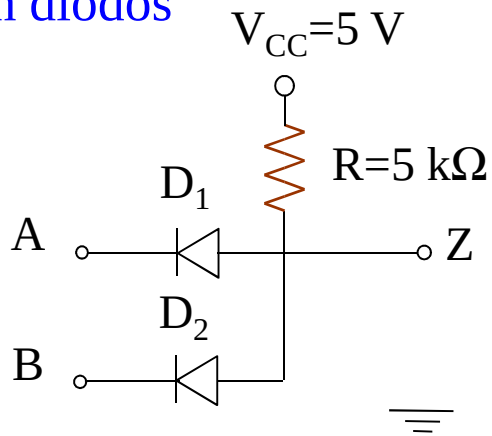
The **AND gate** produces a HIGH output when all inputs are HIGH; otherwise, the output is LOW. For a 2-input gate, the truth table is

Inputs		Output
A	*B*	*X*
0	0	0
0	1	0
1	0	0
1	1	1

Função E com dois interruptores em série



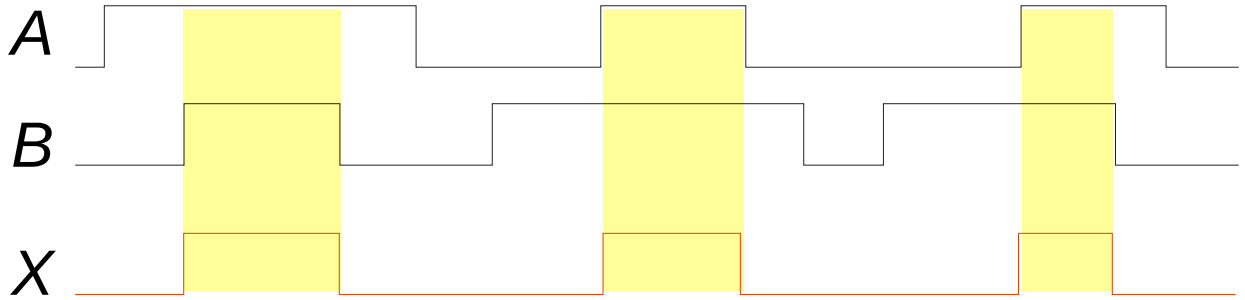
Função E implementada com díodos



The **AND** operation is usually shown with a dot between the variables but it may be implied (no dot). Thus, the AND operation is written as ***X = A·B*** or ***X = AB***.

The AND Gate

Example waveforms:



The AND operation is used in computer programming as a selective mask. If you want to retain certain bits of a binary number but reset the other bits to 0, you could set a mask with 1's in the position of the retained bits.

If the binary number 10100011 is ANDed with the mask 00001111, what is the result?

00000011

Porta OU

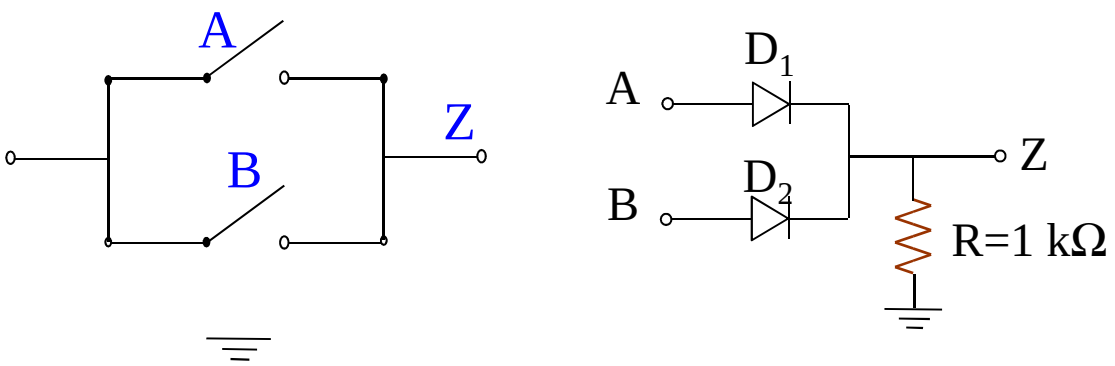
The OR Gate

The **OR gate** produces a HIGH output if any input is HIGH; if all inputs are LOW, the output is LOW. For a 2-input gate, the truth table is

Inputs		Output
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	1

Função OU:

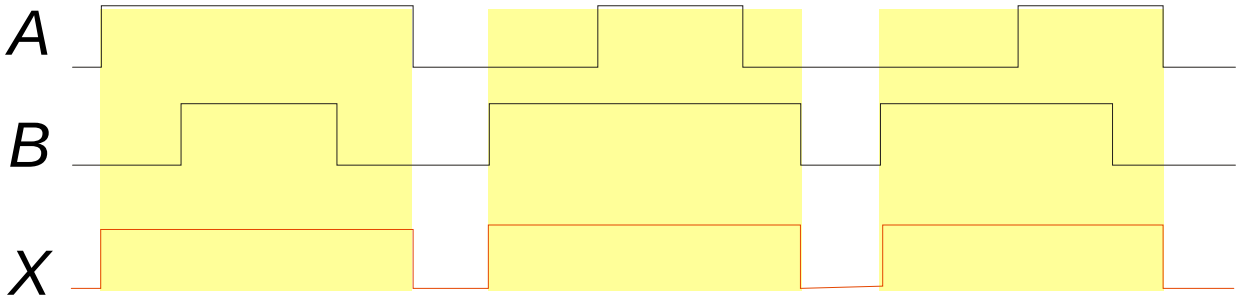
Sejam dois interruptores/díodos em paralelo



The **OR** operation is shown with a plus sign (+) between the variables. Thus, the OR operation is written as ***X = A + B***.

The OR Gate

Example waveforms:



The OR operation can be used in computer programming to set certain bits of a binary number to 1.

ASCII letters have a 1 in the bit 5 position for lower case letters and a 0 in this position for capitals. (Bit positions are numbered from right to left starting with 0.) What will be the result if you OR an ASCII letter with the 8-bit mask 00100000?

The resulting letter will be lower case.

Porta Não-E

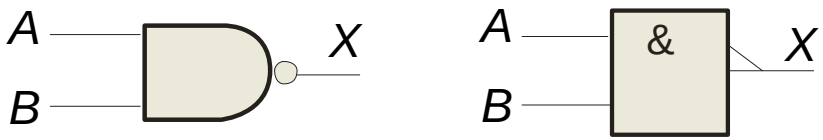
The NAND Gate

The **NAND gate** produces a LOW output when all inputs are HIGH; otherwise, the output is HIGH. For a 2-input gate, the truth table is

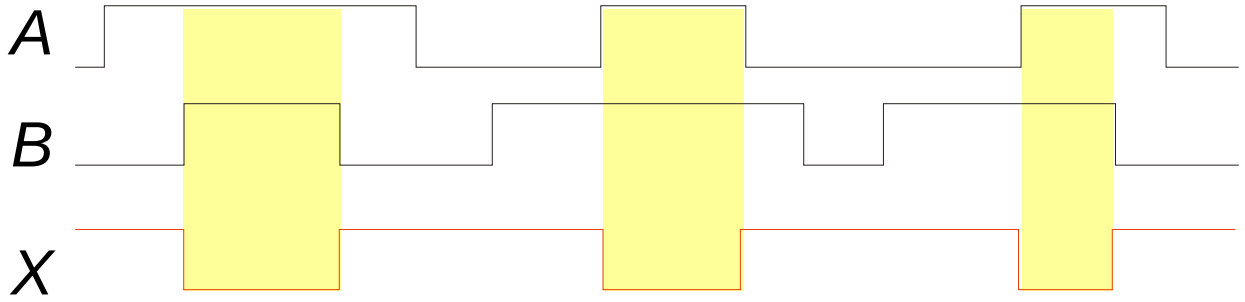
Inputs		Output
A	*B*	*X*
0	0	1
0	1	1
1	0	1
1	1	0

The **NAND** operation is shown with a dot between the variables and an over-bar covering them. Thus, the NAND operation is written as $X = \overline{A \cdot B}$ (Or, $X = \overline{AB}$.)

The NAND Gate

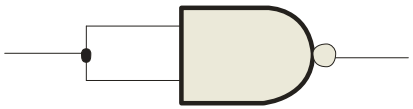


Example waveforms:



The NAND gate is referred to as a “universal” gate because all other basic gates can be constructed from NAND gates.

How would you connect a 2-input NAND gate to form a basic inverter?



Porta Não-Ou

The NOR Gate

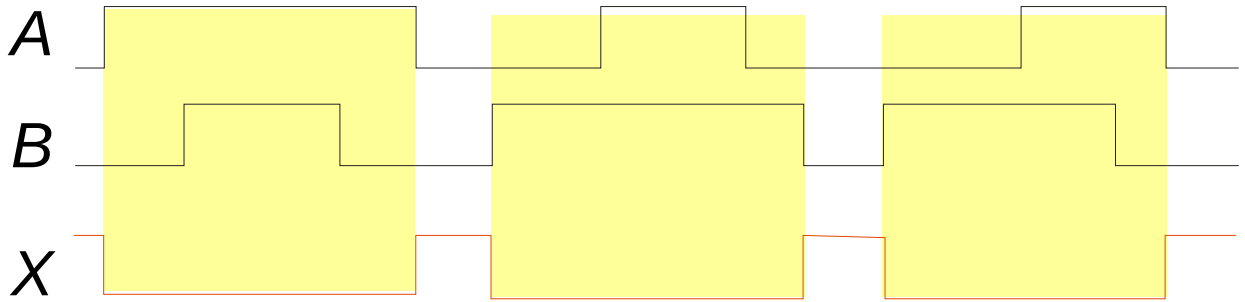
The **NOR gate** produces a LOW output if any input is HIGH; if all inputs are HIGH, the output is LOW. For a 2-input gate, the truth table is

Inputs		Output
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	0

The **NOR** operation is shown with a plus sign (+) between the variables and an overbar covering them. Thus, the NOR operation is written as $X = \overline{A + B}$.

The NOR Gate

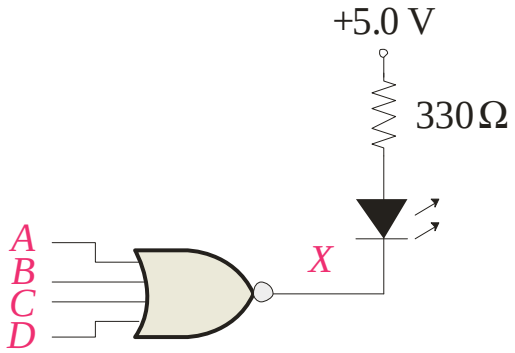
Example waveforms:



The NOR operation will produce a LOW if any input is HIGH.

When is the LED is ON for the circuit shown?

The LED will be on when any of the four inputs are HIGH.



Porta OU-Exclusivo

The XOR Gate

The **XOR gate** produces a HIGH output only when both inputs are at opposite logic levels. The truth table is

Inputs		Output
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	0

The **XOR** operation is written as $X = A\overline{B} + \overline{A}B$. Alternatively, it can be written with a **circled plus sign** between the variables as

$X = A \oplus B$.

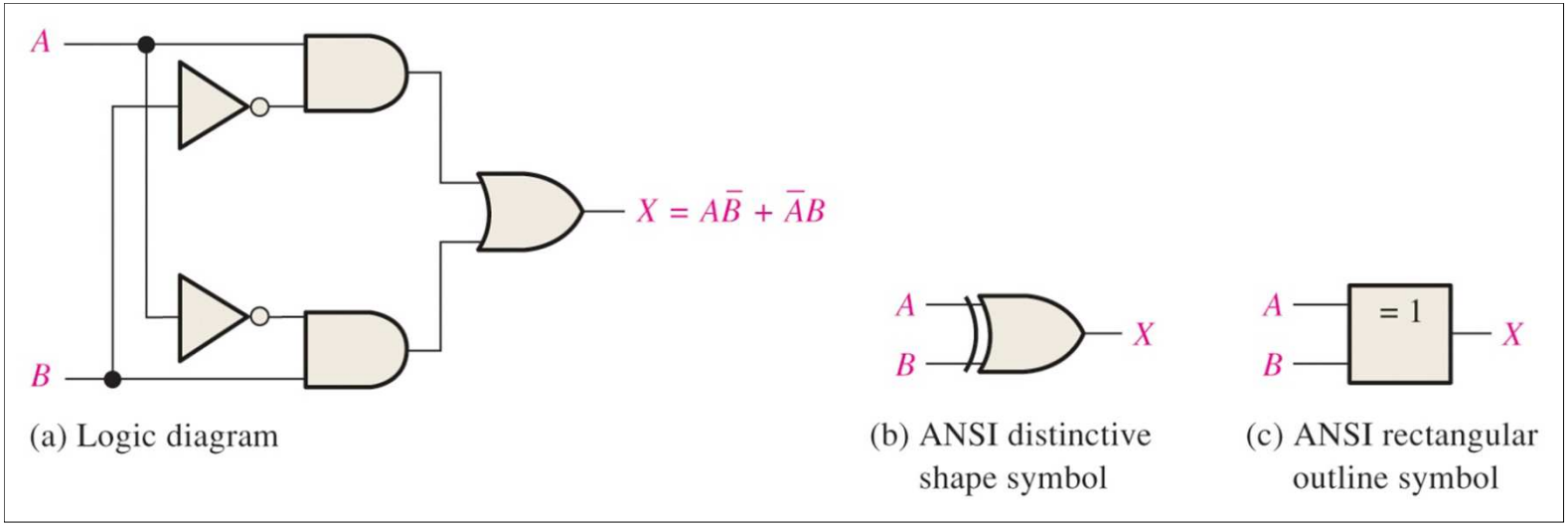
Exclusive-OR Logic

The truth table for an exclusive-OR gate is shown (right). Note that the output is HIGH whenever A and B are unequal.

The Boolean expression is $X = \bar{A}B + A\bar{B}$

TABLE 4–2 • Truth table for an exclusive-OR.

A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	0



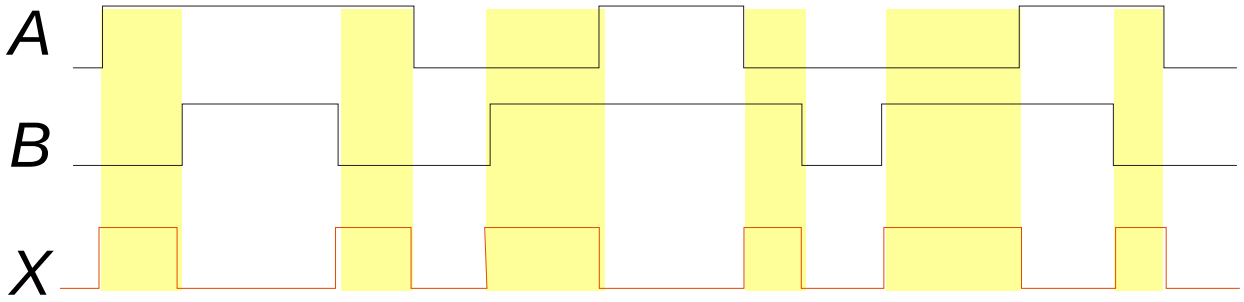
(a) Logic diagram

(b) ANSI distinctive shape symbol

(c) ANSI rectangular outline symbol

The XOR Gate

Example waveforms:

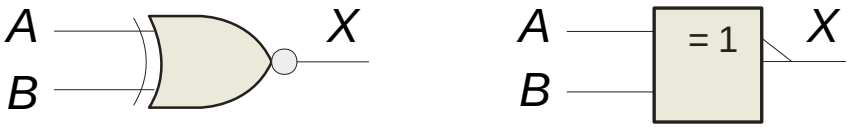


Notice that the XOR gate will produce a HIGH only when exactly one input is HIGH.

If the *A* and *B* waveforms are both inverted for the above waveforms, how is the output affected?

There is no change in the output.

Porta Não-OU-Exclusivo
The XNOR Gate

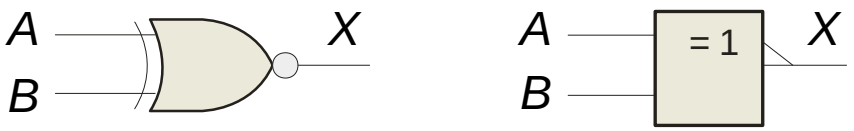


The **XNOR gate** produces a HIGH output only when both inputs are at the same logic level. The truth table is

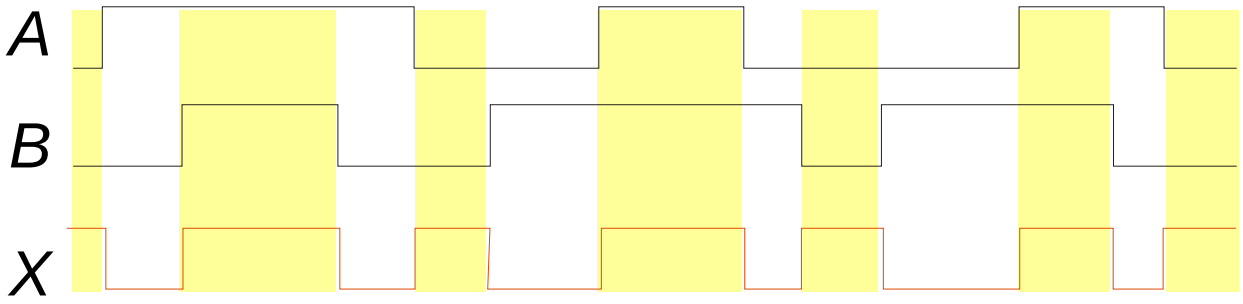
Inputs		Output
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	1

The **XNOR** operation shown as $X = AB + \overline{A}\,\overline{B}$. Alternatively, the XNOR operation can be shown with a **circled dot** between the variables. Thus, it can be shown as $X = A \odot B$.

The XNOR Gate



Example waveforms:



Notice that the XNOR gate will produce a HIGH when both inputs are the same. This makes it useful for comparison functions.

If the *A* waveform is inverted but *B* remains the same, how is the output affected?

The output will be inverted.

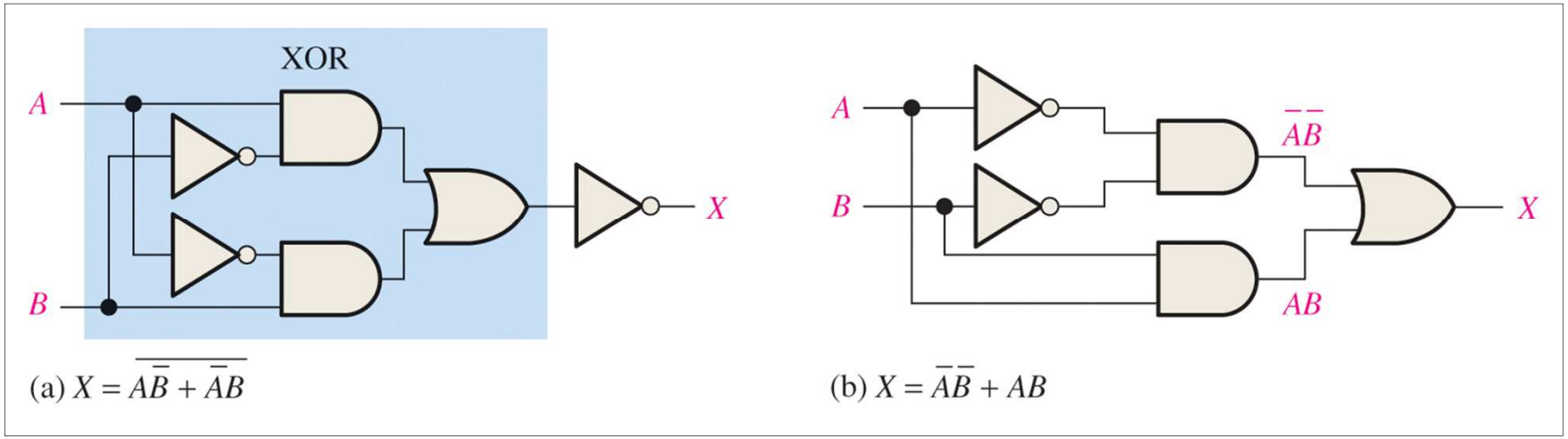
Exclusive-NOR Logic

The truth table for an exclusive-NOR gate is shown (right). Note that the output is HIGH whenever A and B are equal.

TABLE 3–16 • Truth table for an exclusive-NOR gate.

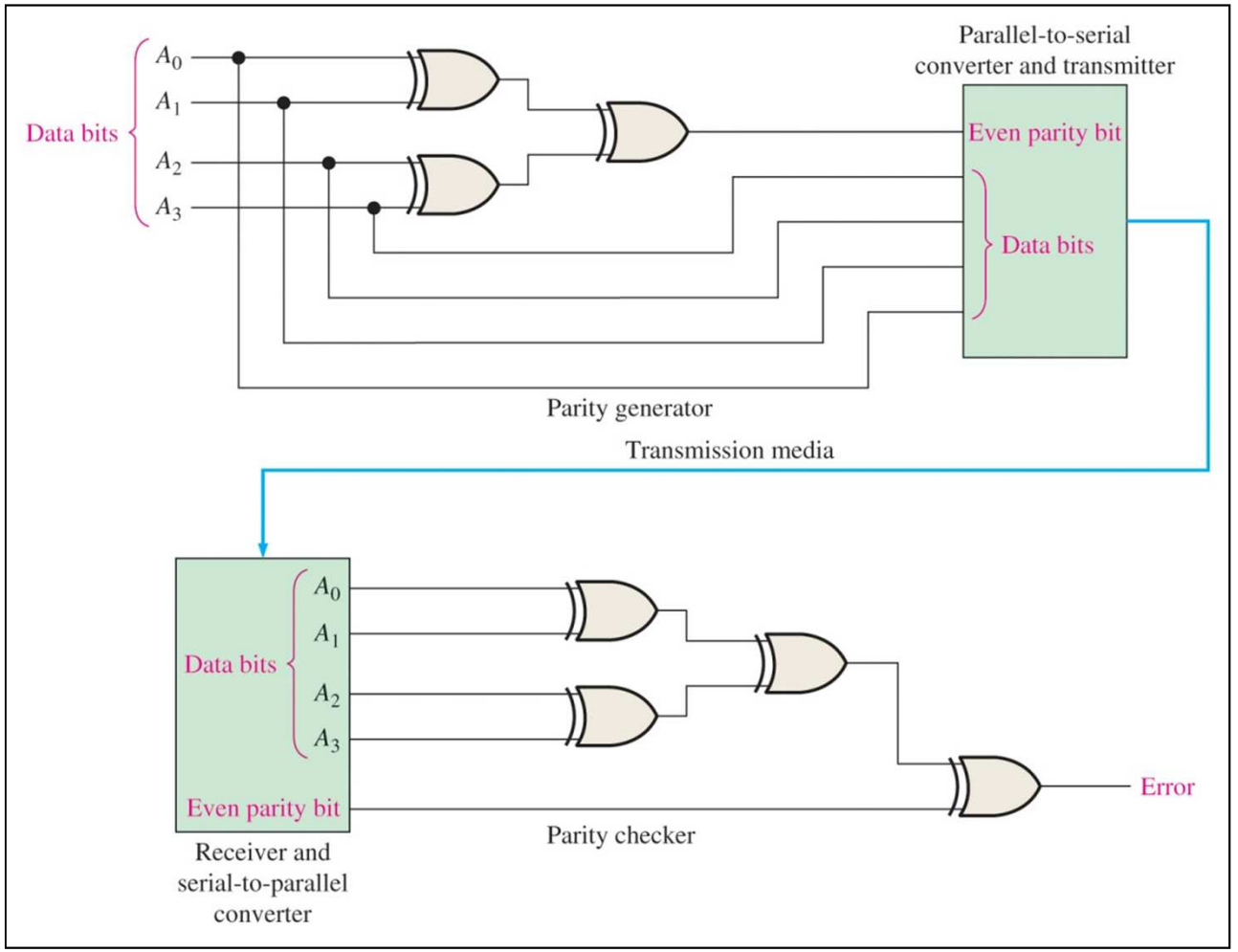
INPUTS		OUTPUT
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	1

The Boolean expression is $X = \overline{A}\overline{B} + AB$



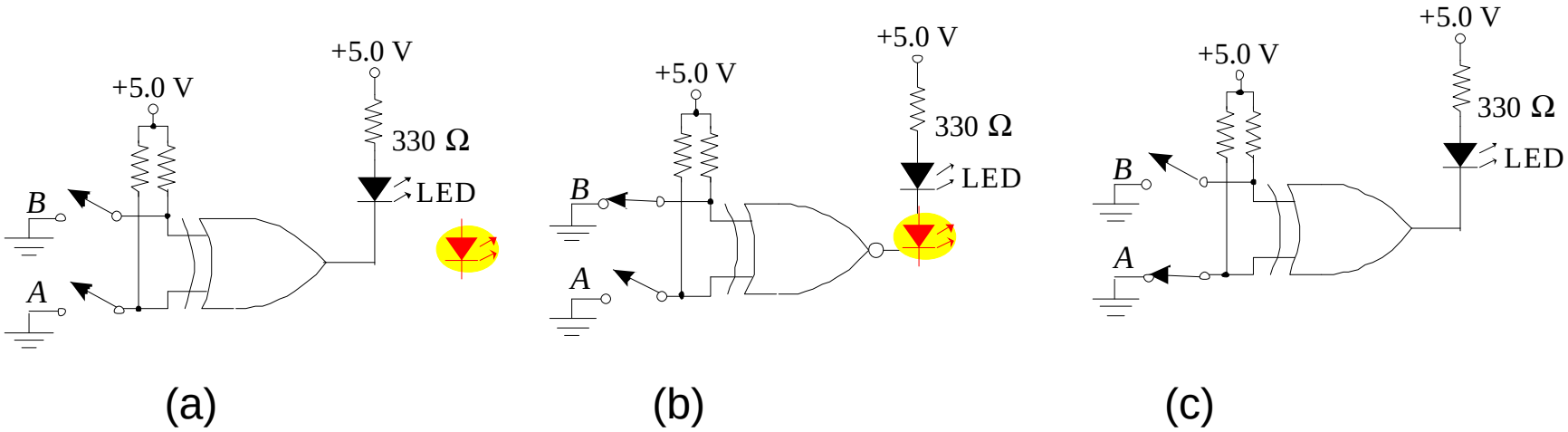
An Exclusive-NOR Logic Application

Data transmission with error detection



Exercício

For each circuit, determine if the LED should be on or off.



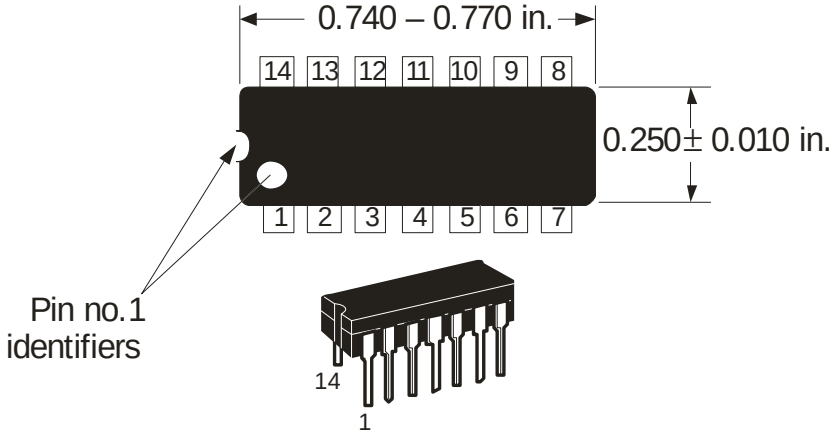
Circuit (a): XOR, inputs agree, output is LOW, LED is ON.

Circuit (b): XNOR, inputs disagree, output is LOW, LED is ON.

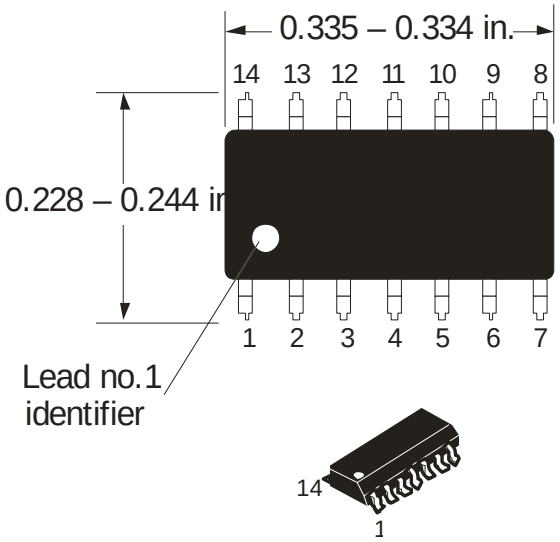
Circuit (c): XOR, inputs disagree, output is HIGH, LED is OFF.

Famílias lógicas TTL e CMOS

Two major fixed function logic families are TTL and CMOS. A third technology is BiCMOS, which combines the first two. Packaging for fixed function logic is shown.



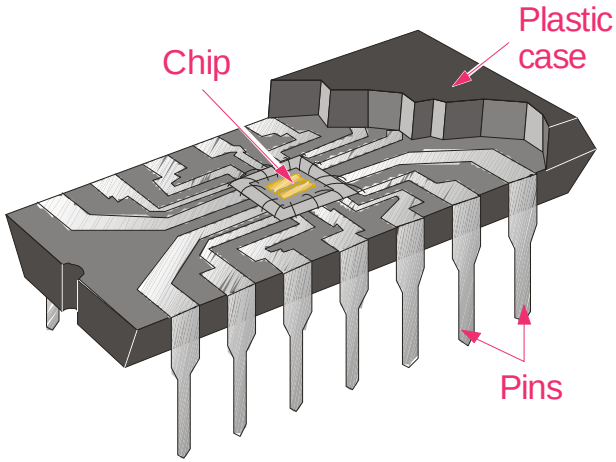
DIP package



SOIC package

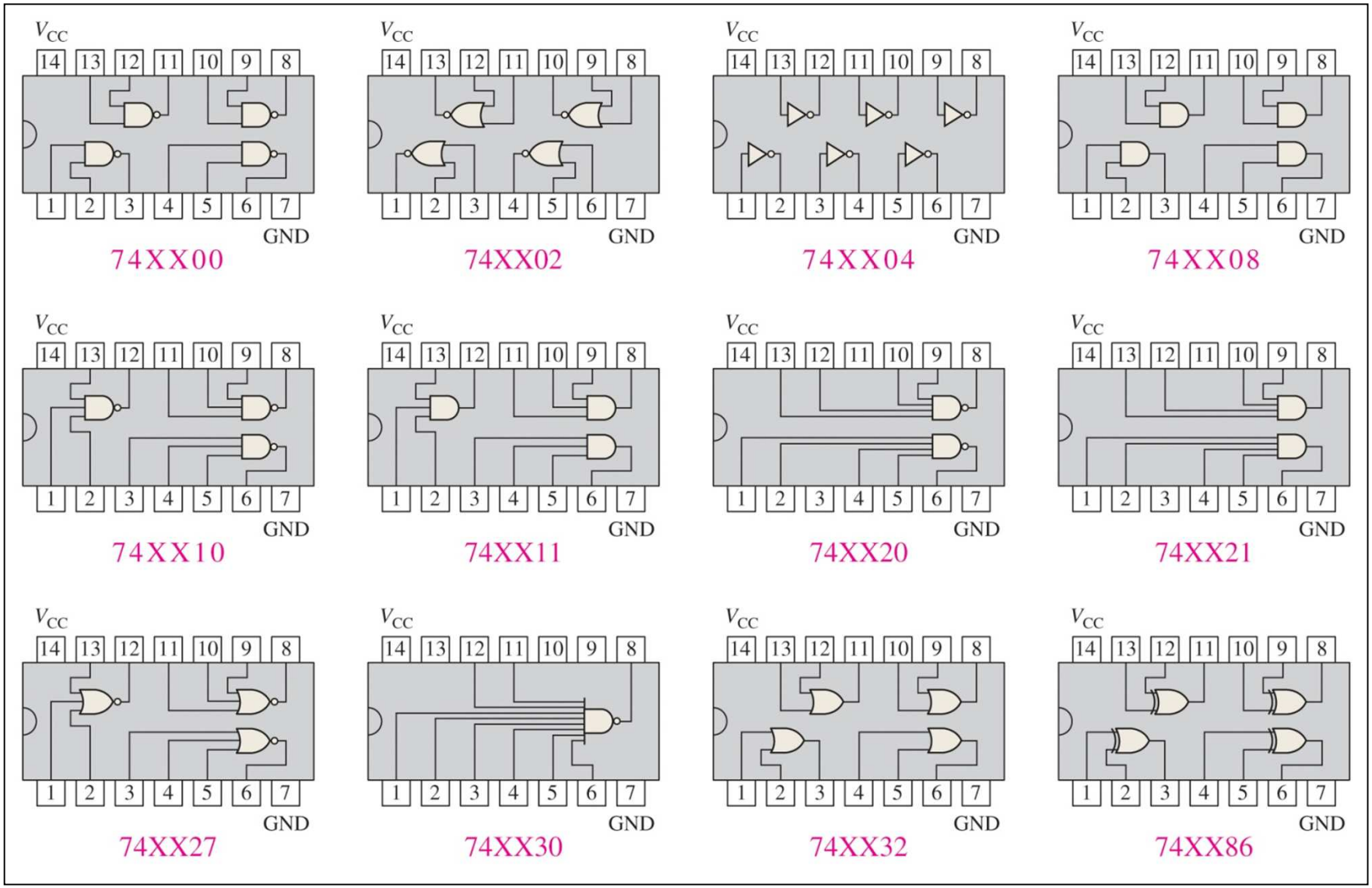
Circuitos integrados

Cutaway view of DIP (Dual-In-line Pins) chip:



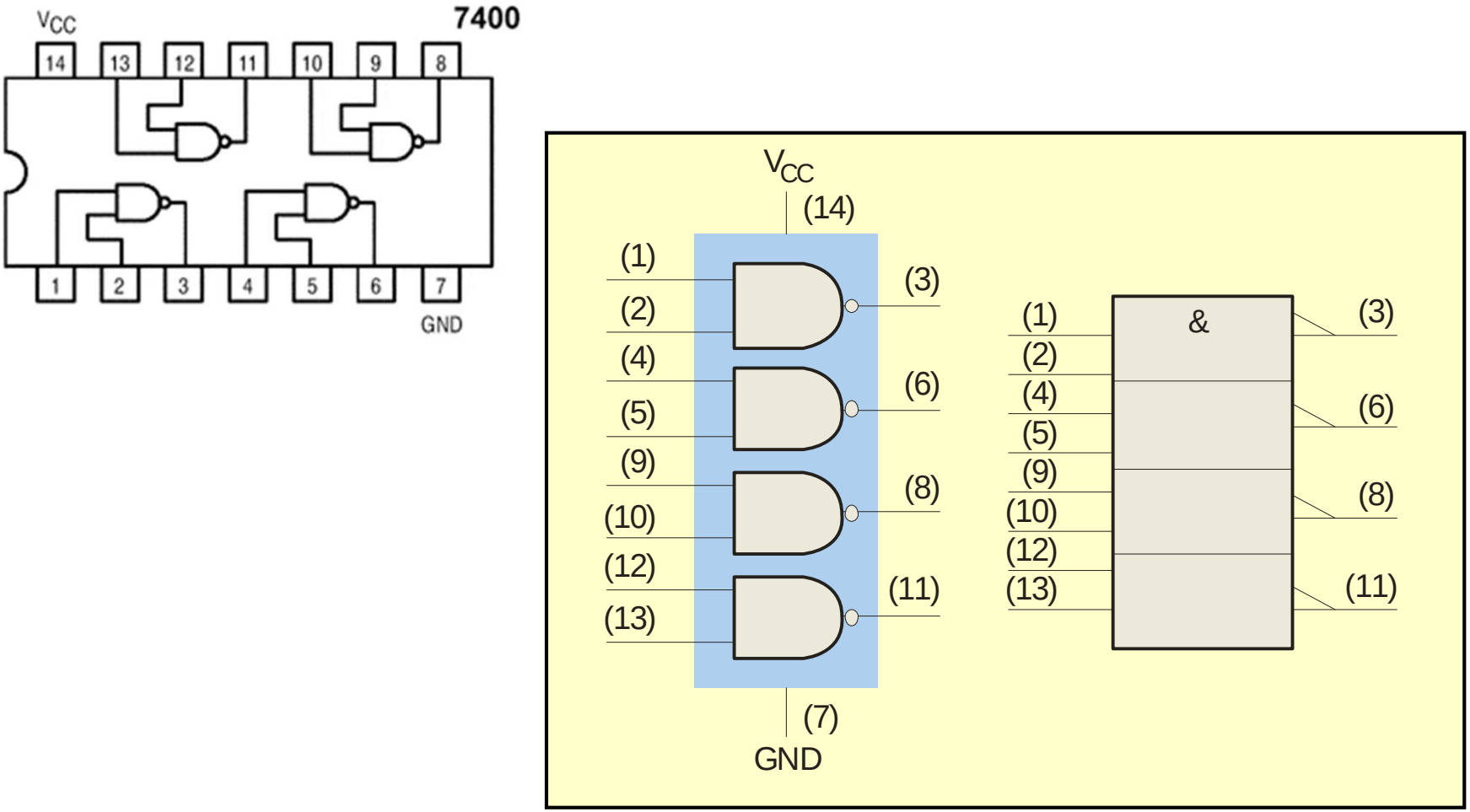
The TTL series, available as DIPs are popular for laboratory experiments with logic.

Integrados com funções lógicas pré-definidas
Fixed Function Logic ICs



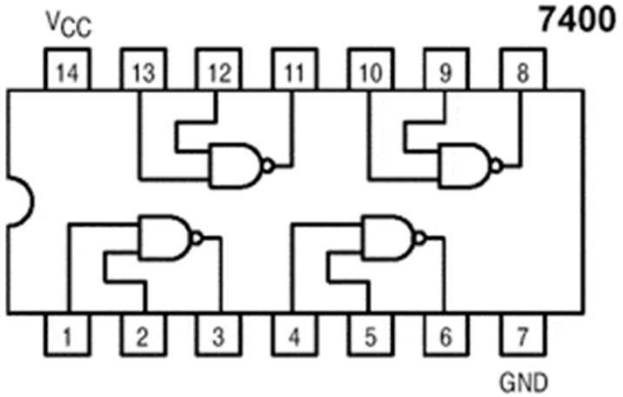
Exemplo 74XX00

Logic symbols show the gates and associated pin numbers.



Folha de dados do componente 74XX00

Data sheets include limits and conditions set by the manufacturer as well as DC and AC characteristics. For example, some maximum ratings for a 74HC00A are:



MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{CC}	DC Supply Voltage (Referenced to GND)	–0.5 to + 7.0 V	V
V_{in}	DC InputVoltage (Referenced to GND)	–0.5 to V_{CC} +0.5 V	V
V_{out}	DC Output Voltage (Referenced to GND)	– 0.5 to V_{CC} +0.5 V	V
I_{in}	DC Input Current, per pin	±20	mA
I_{out}	DC Output Current, per pin	±25	mA
I_{CC}	DC Supply Current, V_{CC} and GND pins	±50	mA
P_D	Power Dissipation in Still Air, Plastic or Ceramic DIP † SOIC Package † TSSOP Package †	750 500 450	mW
T_{stg}	Storage Temperature	–65 to + 150	℃
T_L	Lead Temperature, 1 mm from Case for 10 Seconds Plastic DIP, SOIC, or TSSOP Package Ceramic DIP	 260 300	℃

Variáveis Booleanas

Boolean Variables

A **variable** is a symbol used to represent an action, a condition, or data. Each variable has a value of 1 or 0.

The **complement** represents the inverse of a variable; indicated by an over-bar. Thus, the complement of *A* is $\overline{A}$.

A **literal** is a variable or its complement.

Adição Booleana
Boolean Addition

Addition is equivalent to the OR operation.

The sum term is 1 if one or more if the literals are 1. The sum term is zero only if each literal is 0.

Determine the values of *A, B,* and *C* that make the sum term of the expression

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

Each literal must equal 0; therefore $A = 1$, $B = 0$ and $C = 1$.

Boolean Multiplication

In Boolean algebra, multiplication is equivalent to the AND operation.

The product of literals forms a product term. The product term (AND gate output) equals 1 only when each literal equals 1.

What are the values of the *A*, *B* and *C* if the product term of $A \cdot \overline{B} \cdot C = 1$?

$$A \bullet \overline{B} \bullet \overline{C} = 1$$

Each literal must = 1; therefore *A* = 1, *B* = 0 and *C* = 0.

Propriedade comutativa
Commutative Laws

The **commutative laws** apply to both addition and multiplication. For addition, the commutative law states:

In terms of the result, the order in which variables are ORed makes no difference.

$$A + B = B + A$$

For multiplication, the commutative law states:

In terms of the result, the order in which variables are ANDed makes no difference.

$$AB = BA$$

Propriedade associativa
Associative Laws

The **associative laws** also apply to both addition and multiplication. For addition, the associative law states:

When ORing more than two variables, the result is the same regardless of the grouping of the variables.

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

For multiplication, the associative law states:

When ANDing more than two variables, the result is the same regardless of the grouping of the variables.

$$A(BC) = (AB)C$$

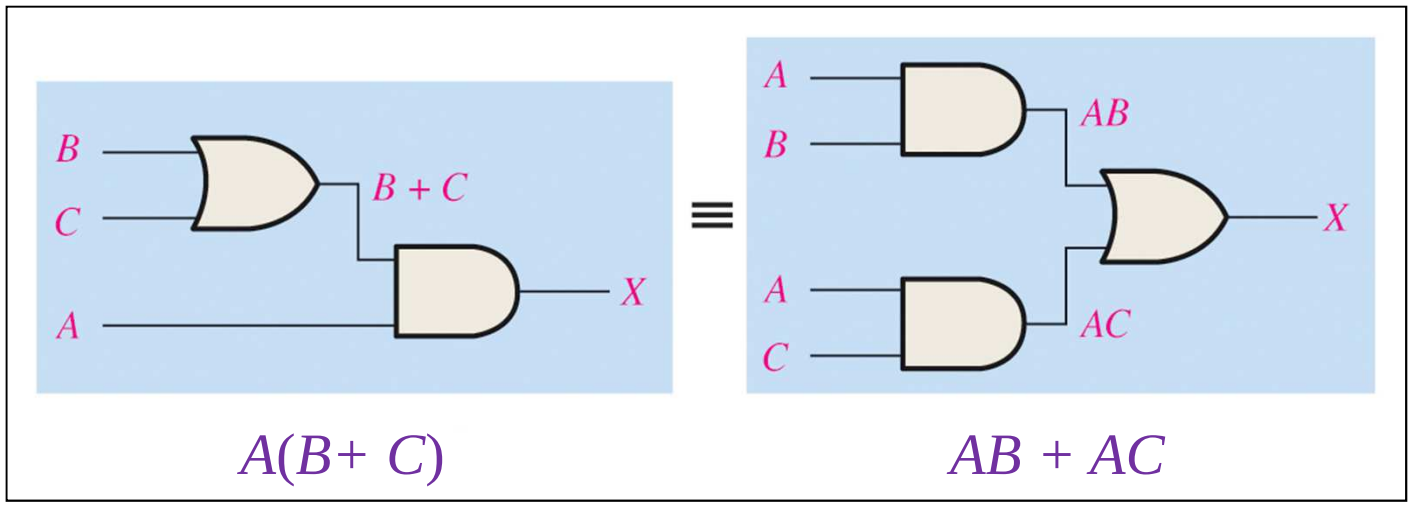
Propriedade distributiva

Distributive Law

The **distributive law** is the *factoring law*. A common variable can be factored from an expression just as in ordinary algebra. That is:

$$AB + AC = A(B + C)$$

The distributive law can be illustrated with equivalent circuits:



Regras da álgebra Booleana

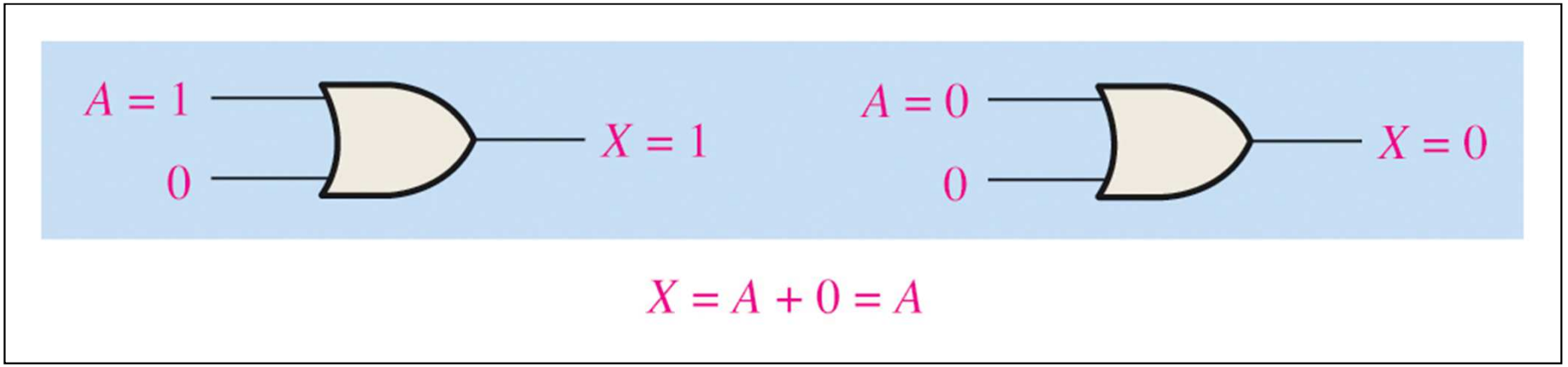
Rules of Boolean Algebra

1. $A + 0 = A$
2. $A + 1 = 1$
3. $A \cdot 0 = 0$
4. $A \cdot 1 = A$
5. $A + A = A$
6. $A + \overline{A} = 1$
7. $A \cdot A = A$
8. $A \cdot \overline{A} = 0$
9. $\overline{\overline{A}} = A$
10. $A + AB = A$
11. $A + \overline{A}B = A + B$
12. $(A + B)(A + C) = A + BC$

Rules of Boolean Algebra

Rule 1: $A + 0 = A$

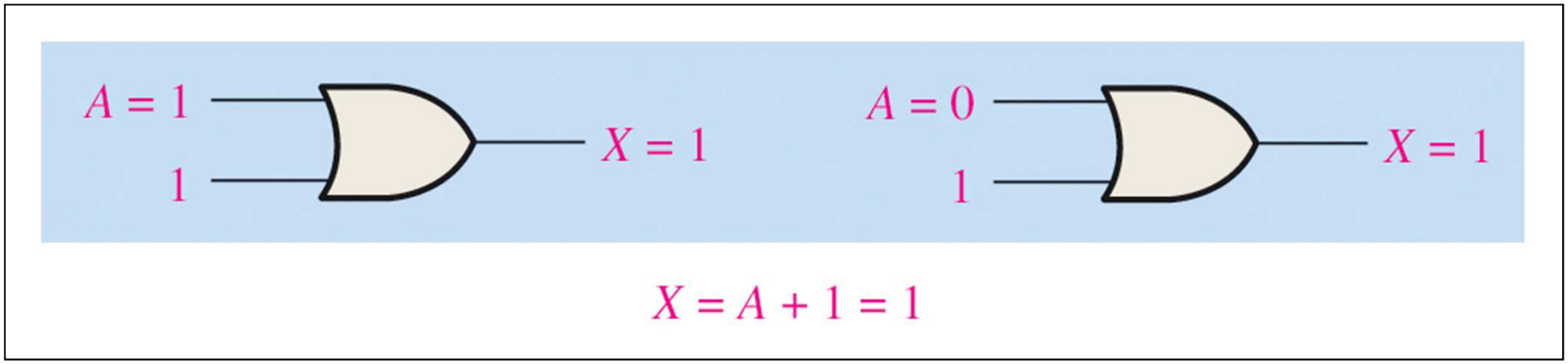
When $A = 1$, the input causes the output to go to $X = 1$.
When $A = 0$, the 0 inputs cause the output to go to $X = 0$.
In either case, the value of X equals the value of A.



Rules of Boolean Algebra

Rule 2: $A + 1 = 1$

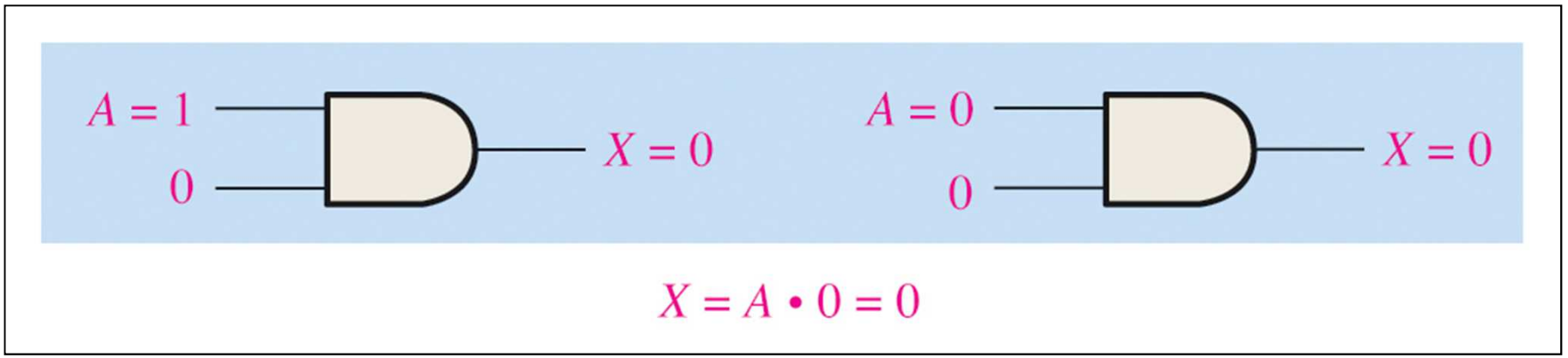
When $A = 1$, the inputs cause the output to go to $X = 1$.
When $A = 0$, the 1 input caused the output to go to $X = 1$.
In either case, the value of X equals one (1).



Rules of Boolean Algebra

Rule 3: $A \bullet 0 = 0$

When either input to an AND gate equals 0, the output from the gate has a value of $X = 0$, regardless of the value at the other input.



Rules of Boolean Algebra

Rule 4: $A \cdot 1 = A$

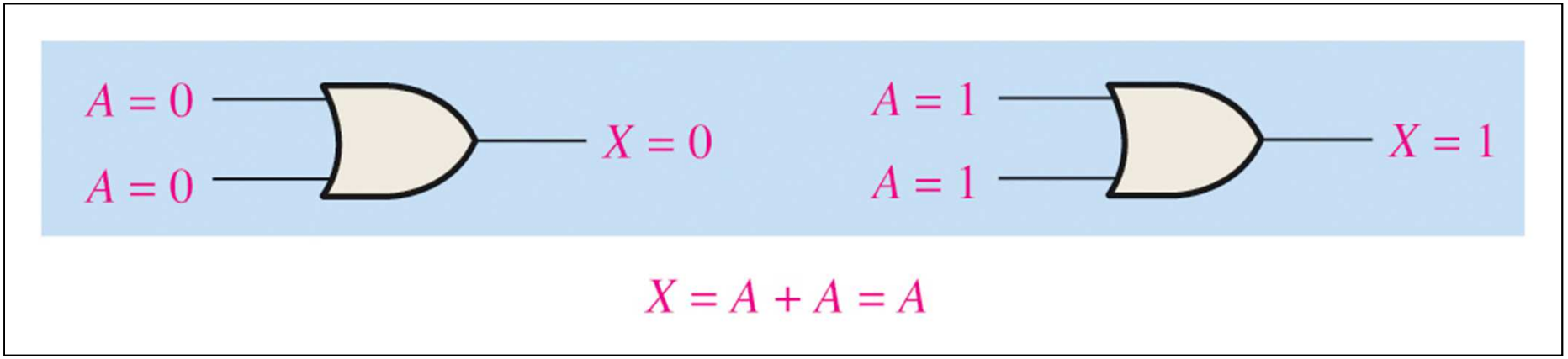
When one input to an AND gate equals 1, the output from the gate has a value of $X = A$. As shown, $X = 1$ when $A = 1$ and $X = 0$ when $A = 0$.

$X = A \cdot 1 = A$

Rules of Boolean Algebra

Rule 5: $A + A = A$

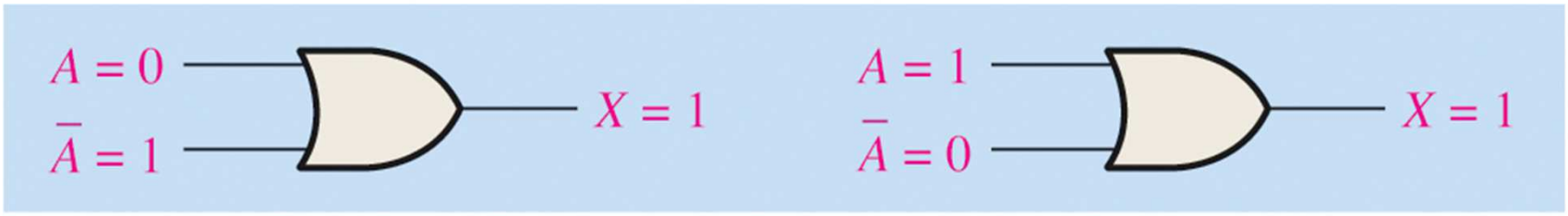
When the inputs to an OR gate are equal, the output equals the value at the inputs. When both inputs equal 1, the gate output is $X = 1$. When both inputs equal 0, the gate output is $X = 0$.



Rules of Boolean Algebra

Rule 6: $A + \overline{A} = 1$

When the inputs to an OR gate are unequal (complements), one of the two always equals 1. When either input equals 1, the gate output is $X = 1$. Therefore, the output from the OR gate equals 1 whenever the inputs are unequal (complementary).

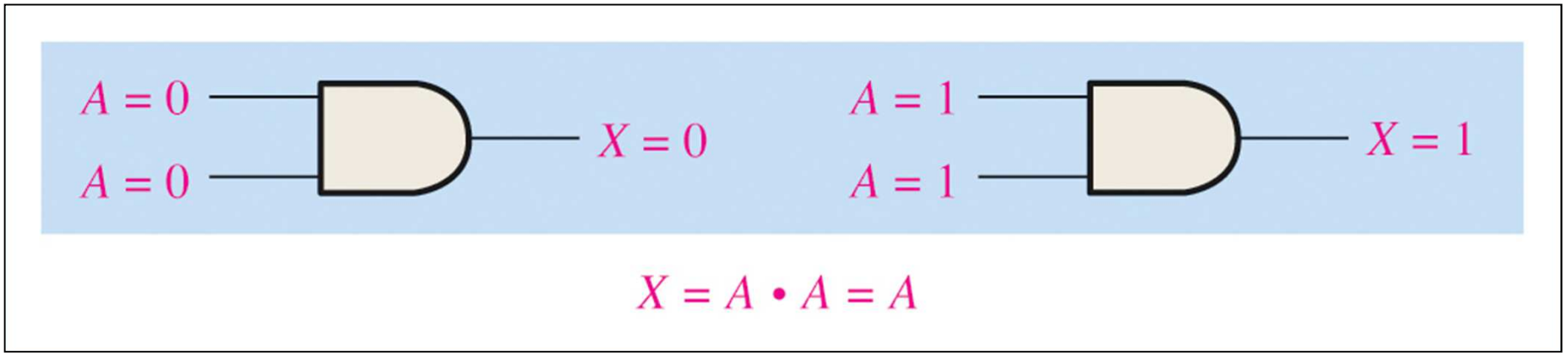


$$X = A + \overline{A} = 1$$

Rules of Boolean Algebra

Rule 7: $A \cdot A = A$

When the inputs to an AND gate are equal, the gate output also equals that value. Thus, $X = 1$ when both inputs equal 1 and $X = 0$ when both inputs equal 0.



Rules of Boolean Algebra

Rule 8: $A \cdot \overline{A} = 0$

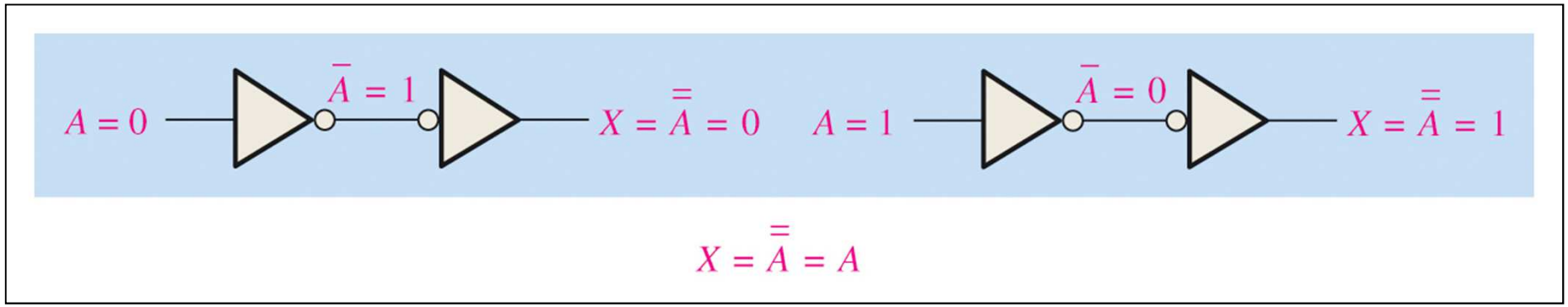
When the inputs to an AND gate are unequal (complements), one of the two always equals 0. When either input equals 0, the gate output is $X = 0$. Therefore, the output from the AND gate equals 0 whenever the inputs are unequal (complementary).

$$X = A \cdot \overline{A} = 0$$

Rules of Boolean Algebra

Rule 9: $\overline{\overline{A}} = A$

When a value is inverted, it is the complement of the original value. When inverted a second time, it returns to its original value. Thus, A = 0 inverted twice equals 0 and A = 1 inverted twice equals 1.



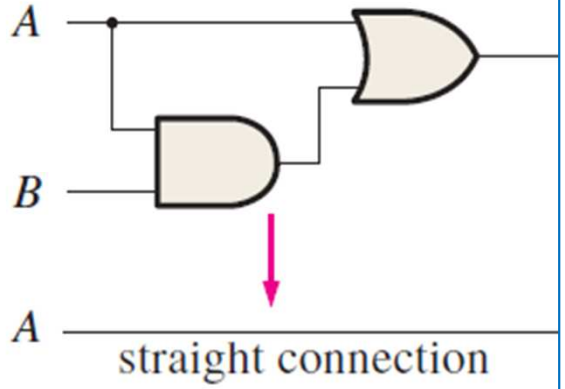
Rules of Boolean Algebra

Rule 10: $A + AB = A$

The circuit and truth table (below) can be used to demonstrate this rule. The truth table shows the outputs from the circuit for every possible combination of A and B. In each case, the output from the OR gate equals the value of A. Thus, $A + AB$ always equals the value of A.

A	B	AB	$A + AB$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

equal



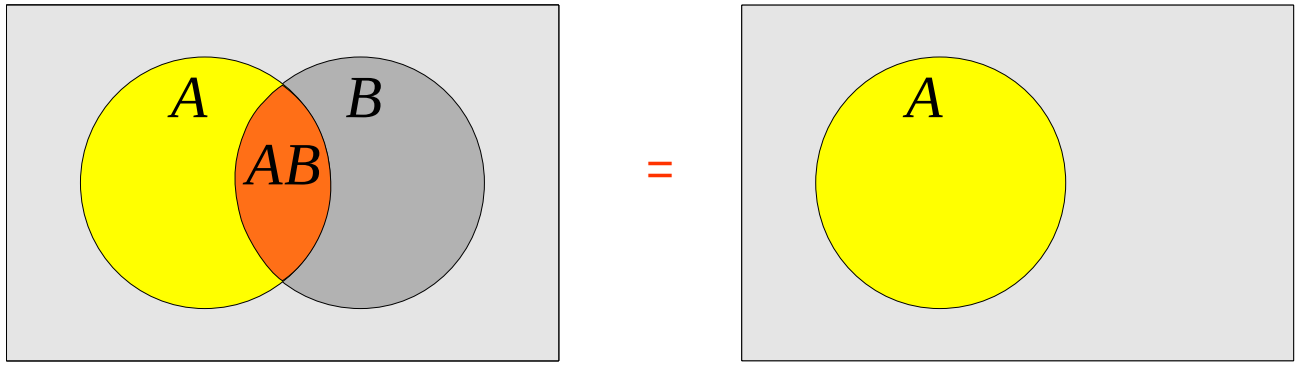
Rules of Boolean Algebra

Rule 10: $A + AB = A$

Rules of Boolean algebra can be illustrated with *Venn* diagrams. The variable ***A*** is shown as an area.

The rule $A + AB = A$ can be illustrated easily with a diagram. Add an overlapping area to represent the variable *B*.

The overlap region between A and B represents ***AB***.



The diagram visually shows that $A + AB = A$. Other rules can be illustrated with the diagrams as well.

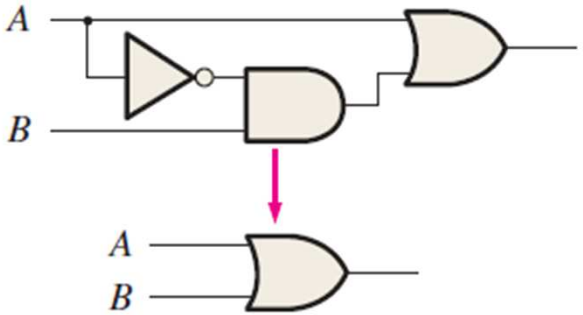
Rules of Boolean Algebra

Rule 11: $A + \overline{A}B = A + B$

The circuits and truth table (below) demonstrate this rule. The truth table shows the outputs from the two circuits are equal for every possible combination of A and B. As such, the two functions ($A + \overline{A}B$) and ($A + B$) are equal.

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

equal



Rules of Boolean Algebra

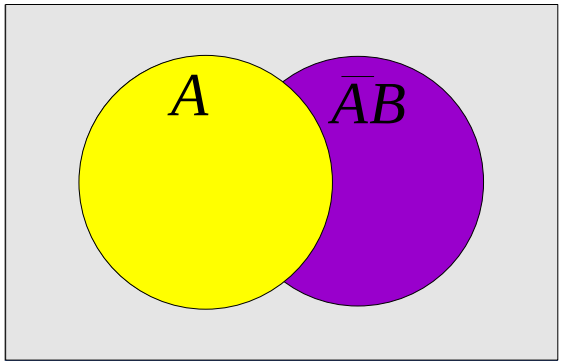
Rule 11: $A + \overline{A}B = A + B$

Illustrate the rule $A + \overline{A}B = A + B$ with a Venn diagram.

This time, $\overline{A}$ is represented by the blue area and B again by the red circle.

The intersection represents $\overline{A}B$.

Notice that $A + \overline{A}B = A + B$



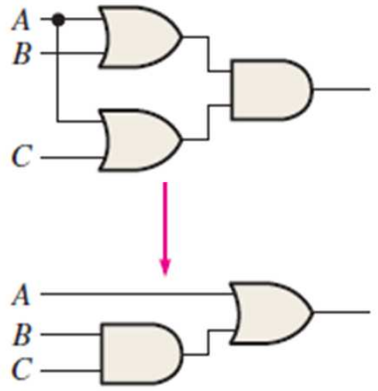
Rules of Boolean Algebra

Rule 12: $(A + B)(A + C) = A + BC$

The circuits and truth table (below) demonstrate this rule. The truth table shows the outputs from the two circuits are equal for every possible combination of *A* and *B*. As such, the two functions $(A + B)(A + C)$ and $(A + BC)$ are equal.

A	B	C	$A + B$	$A + C$	$(A + B)(A + C)$	BC	$A + BC$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal



Rules of Boolean Algebra

Rule 12: $(A + B)(A + C) = A + BC$

Three areas represent the variables *A*, *B*, and *C*.

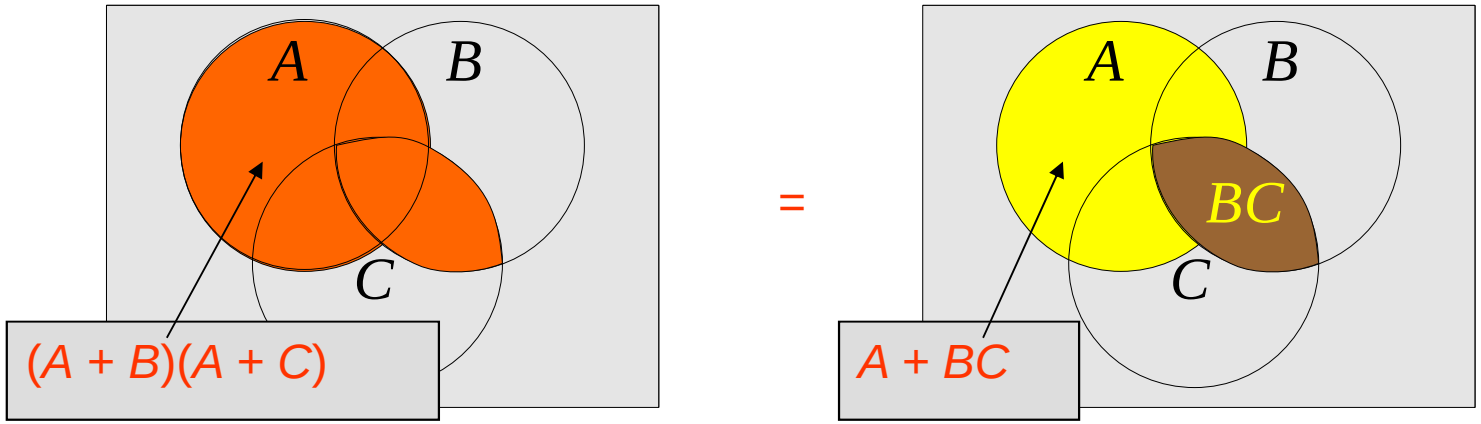
The area representing $A + B$ is shown in yellow.

The area representing $A + C$ is shown in red.

The overlap of red and yellow is shown in orange.

The overlapping area between *B* and C represents *BC*.

ORing with *A* gives the same area as before.

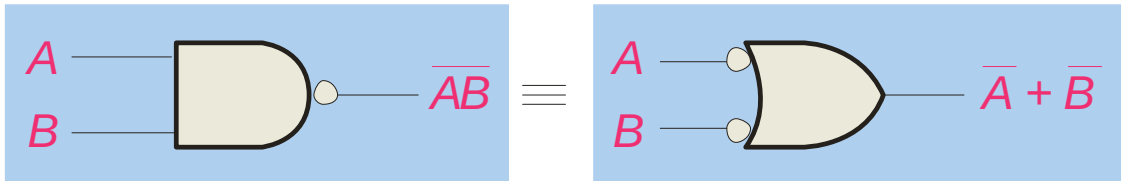


DeMorgan's 1st Theorem

The complement of a product of variables is equal to the sum of the complemented variables.

$$\overline{AB} = \overline{A} + \overline{B}$$

Applying DeMorgan's first theorem to gates:



NAND Negative-OR

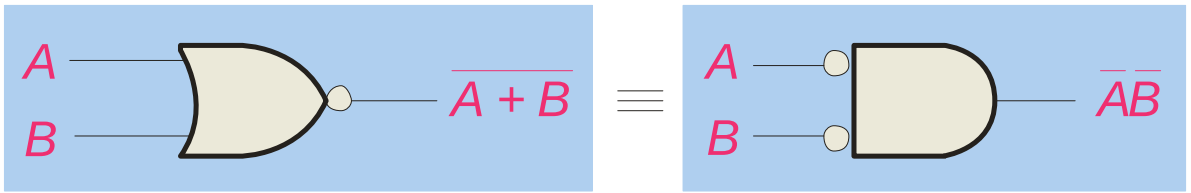
Inputs		Output	
A	B	$\overline{AB}$	$\overline{A} + \overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

DeMorgan's 2nd Theorem

The complement of a sum of variables is equal to the product of the complemented variables.

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

Applying DeMorgan's second theorem to gates:



NOR Negative-AND

Inputs		Output	
A	B	$\overline{A + B}$	$\overline{A}\,\overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Exercício: DeMorgan's Theorem

Apply DeMorgan's theorem to remove the overbar covering both terms from the expression

$$X = \overline{\overline{C} + D}.$$

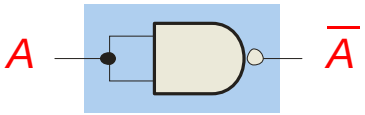
To apply DeMorgan's theorem to the expression, you can break the overbar covering both terms and change the sign between the terms. This results in

$$X = \overline{\overline{C}} \cdot \overline{D}.$$

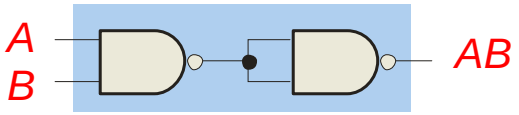
Deleting the double bar gives $X = C \cdot \overline{D}$.

NAND Universal Gates

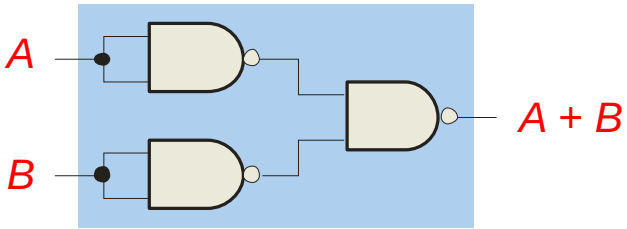
NAND gates are sometimes called **universal** gates because they can be used to produce the other basic Boolean functions.



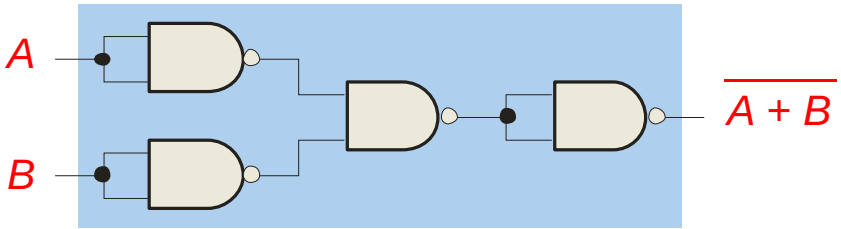
Inverter



AND gate



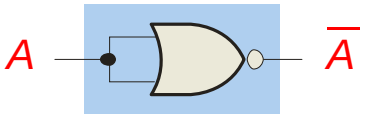
OR gate



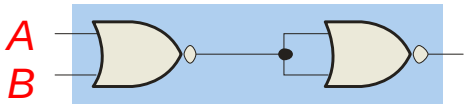
NOR gate

NOR Universal Gates

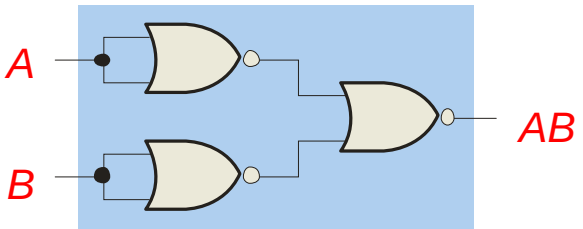
NOR gates are also **universal** gates and can form all of the basic gates.



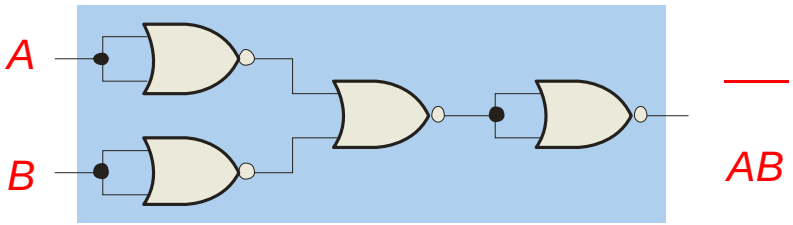
Inverter



OR gate



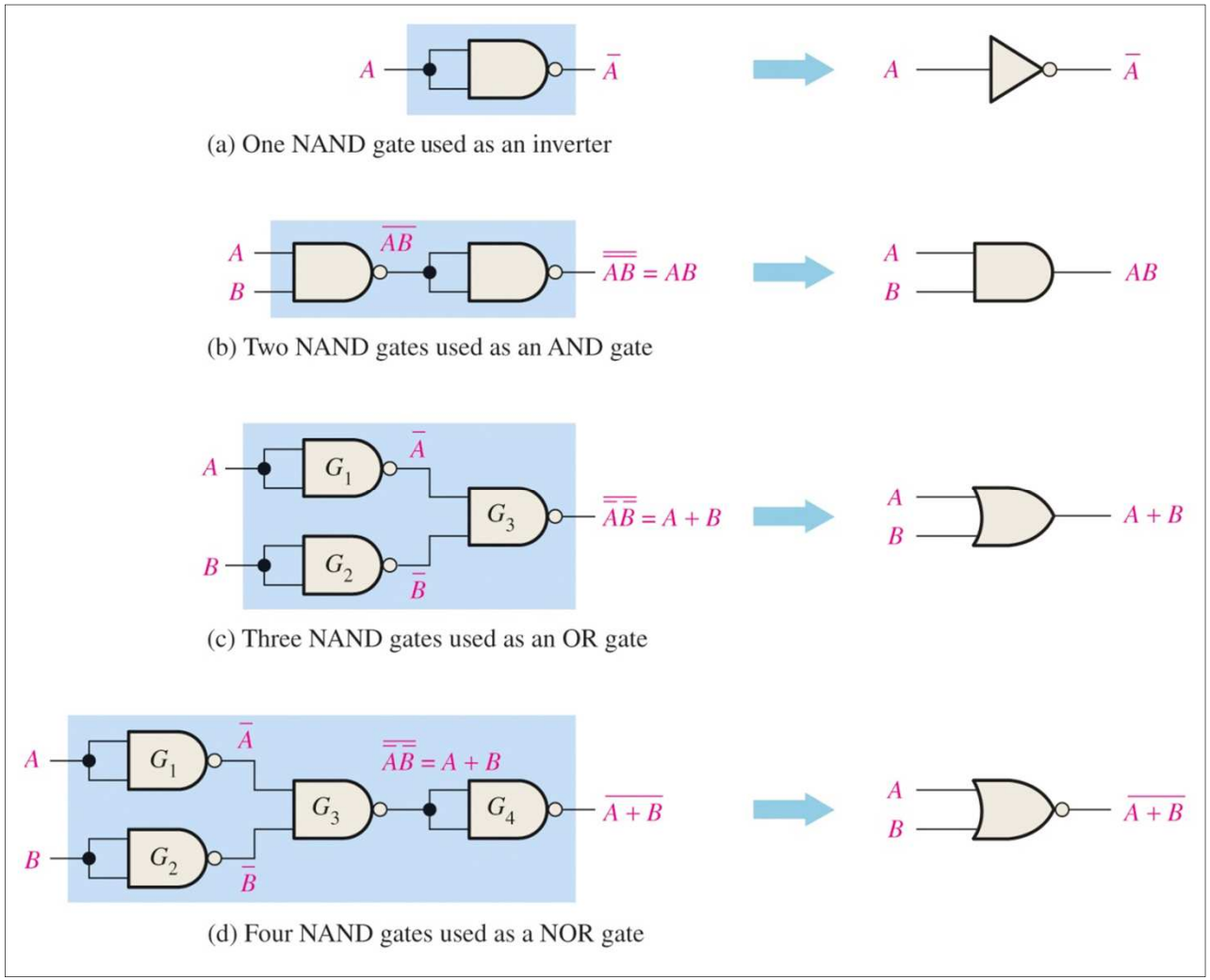
AND gate



NAND gate

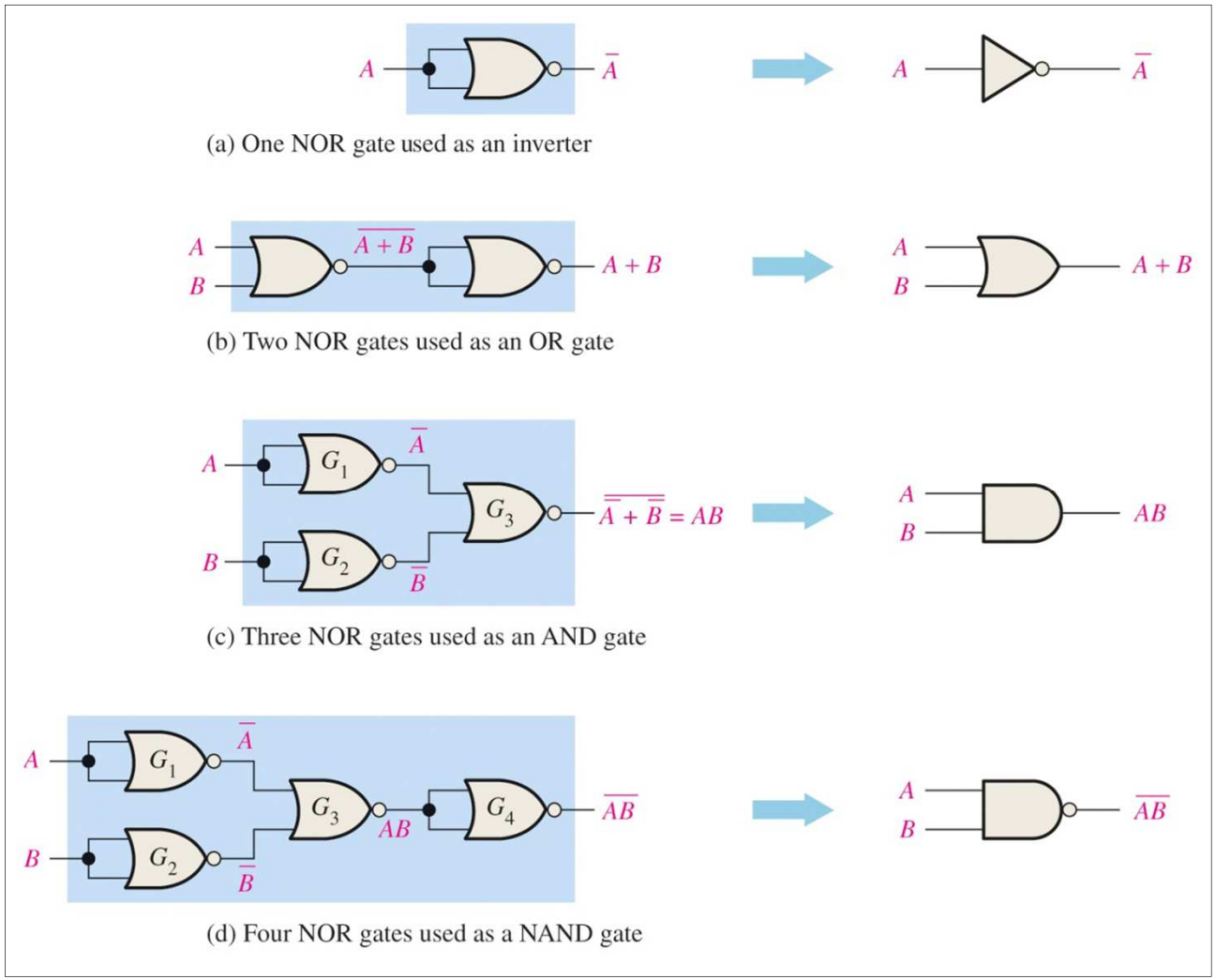
NAND Universal Gates

NAND gates are sometimes called **universal gates** because they can be used to produce the other basic Boolean functions.



NOR Universal Gates

NOR gates are also called **universal gates** because they can be used to produce the other basic Boolean functions.



Circuitos Elétricos e Sistemas Digitais

2018-2019 - 1.º Semestre

Sistemas Digitais

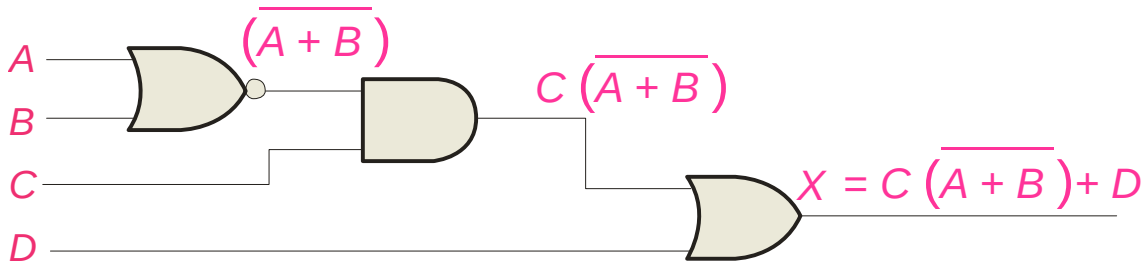
Combinational Logic

Boolean Analysis of Logic Circuits

Combinational logic circuits can be analyzed by writing the expression for each gate and combining the expressions according to the rules for Boolean algebra.

Apply Boolean algebra to derive the expression for *X*.

Write the expression for each gate:



Applying DeMorgan's theorem and the distribution law:

$$X = C\,(\overline{A}\ \overline{B}) + D = \overline{A}\,\overline{B}\,C + D$$

Soma de produtos e produto de somas

Boolean expressions can be written in the **sum-of-products** form (**SOP**) or in the **product-of-sums** form (**POS**). These forms can simplify the implementation of combinational logic.

In both forms, an overbar cannot extend over more than one variable.

An expression is in SOP form when two or more product terms are summed as in the following examples:

$\overline{A}\,\overline{B}\,\overline{C} + A\,B$ $\qquad$ $A\,B\,\overline{C} + \overline{C}\,\overline{D}$ $\qquad$ $C\,D + \overline{E}$

An expression is in POS form when two or more sum terms are multiplied as in the following examples:

$(A + B)(\overline{A} + C)$ $\qquad$ $(A + B + \overline{C})(B + D)$ $\qquad$ $(\overline{A} + B)C$

Forma “standard” do Soma de Produtos
[Sum-of-Products form (SOP)]

SOP Standard form

In **SOP standard form**, every variable in the domain must appear in each term.

The **SOP standard form is useful for constructing truth tables.**

You can expand a nonstandard term to standard form by multiplying the term by a term consisting of the sum of the missing variable and its complement.

Convert $X = \overline{A}\,\overline{B} + A\,B\,C$ to standard form.

The first term does not include the variable C. Therefore, multiply it by the $(C + \overline{C})$, which = 1:

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

Forma “standard” do Produto de Somas
[Product-of-Sums form (POS)]

In **POS standard form**, every variable in the domain must appear in each sum term of the expression.

You can expand a nonstandard POS expression to standard form by adding the product of the missing variable and its complement and applying rule 12, which states that $(A + B)(A + C) = A + BC$.

Convert $X = (\overline{A} + \overline{B})(A + B + C)$ to standard form.

The first sum term does not include the variable C.
Therefore, add $C\,\overline{C}$ and expand the result by rule 12.

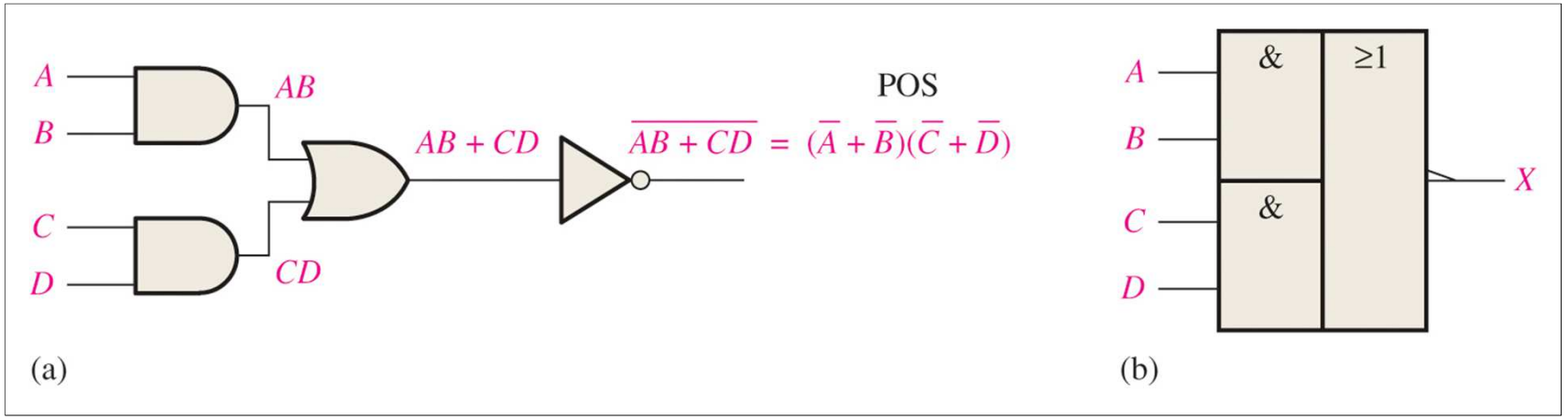
$$X = (\overline{A} + \overline{B} + C\,\overline{C})(A + B + C)$$

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

AND-OR-Invert Logic

When the output of a SOP form is inverted, the circuit is called an AND-OR-Invert circuit (AOI).

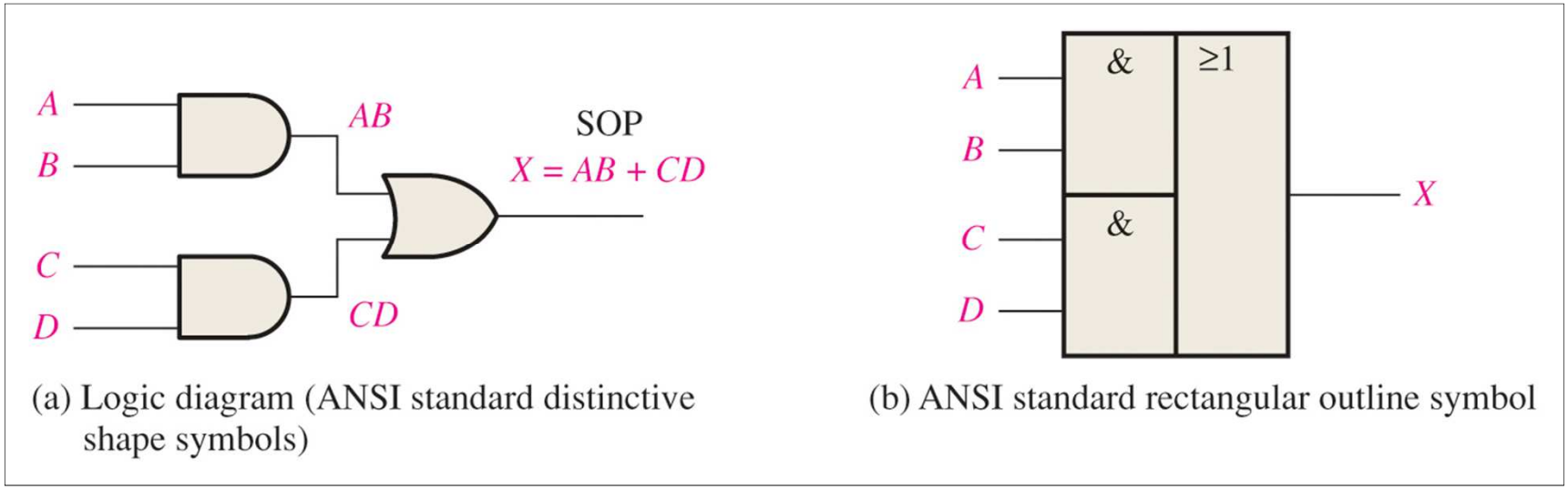
The AOI configuration lends itself to product-of-sums (POS) implementation.



The output from an AND-OR-Invert logic circuit is LOW whenever A and B are HIGH, or C and D are HIGH.

Combinational Logic Circuits

In Sum-of-Products (SOP) form, basic combinational circuits can be directly implemented with AND-OR combinations if the necessary complement terms are available.



(a) Logic diagram (ANSI standard distinctive shape symbols)

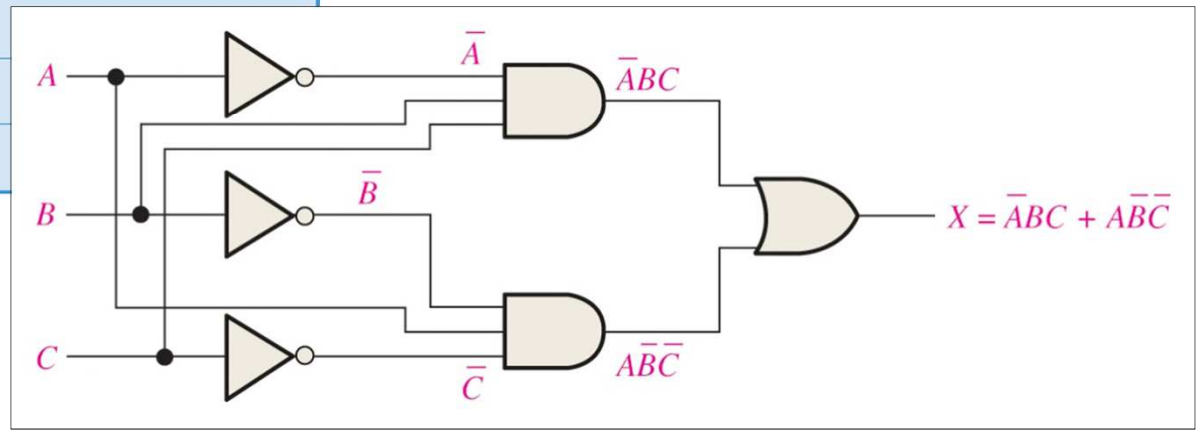
(b) ANSI standard rectangular outline symbol

Implementing Combinational Logic

Implementing a SOP expression is done by first forming the AND terms; then the terms are ORed together.

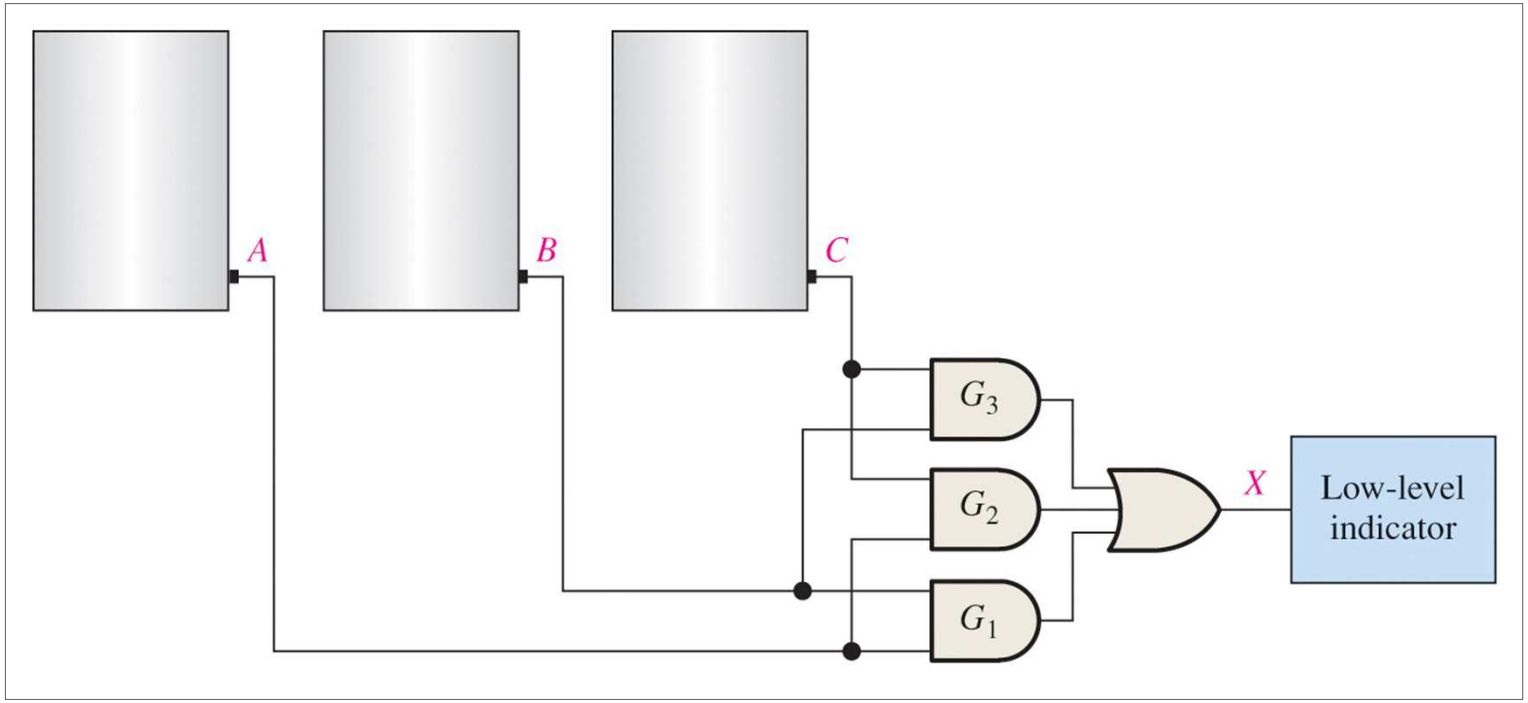
TABLE 4–3

INPUTS A	B	C	OUTPUT X	PRODUCT TERM
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	0	



Combinational Logic Circuit Application

A storage tank monitor



Síntese de Funções lógicas

Expansão em Soma-de-Produtos e Produto-de-Somas a partir da tabela de verdade. Considerar a tabela.

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

Procedimento para obter a expansão em **Soma-de-Produtos**:

i) fazer a lista dos valores binários das variáveis de entrada para as quais a saída é 1;

ii) converter cada valor binário da saída num termo-produto substituindo cada 1 pela varável correspondente e cada 0 pelo complemento da variável correspondente. Exemplo:

$$Y = \bar{A} \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C}$$

Procedimento para obter a expansão em Produto-de-Somas:

i) Listar os valores binários das variáveis de entrada para as quais a saída é 0;

ii) converter cada valor binário da saída num termo-soma o substituindo cada 1 pela complemento da varável correspondente e cada 0 pela variável correspondente. Exemplo:

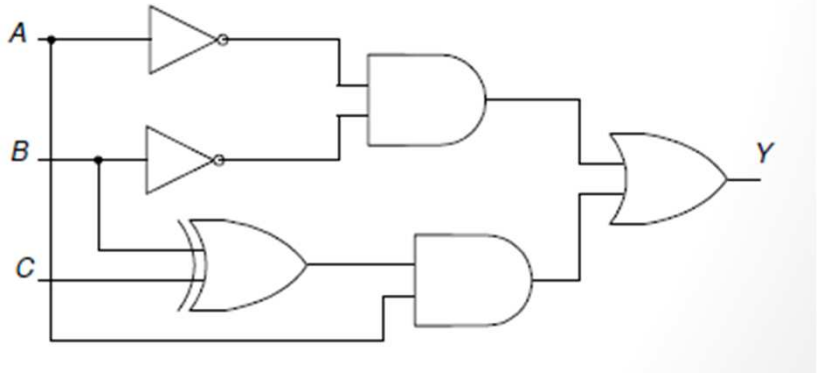
$$Y = (A + \overline{B} + C) \cdot (A + \overline{B} + \overline{C}) \cdot (\overline{A} + B + C) \cdot (\overline{A} + \overline{B} + \overline{C})$$

Simplificação Algébrica de Funções Lógicas

Para se obter a forma simplificado do circuito é necessário fazer-se simplificação das expressões obtidas. Considere-se o exemplo de uma soma-de-produtos:

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

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

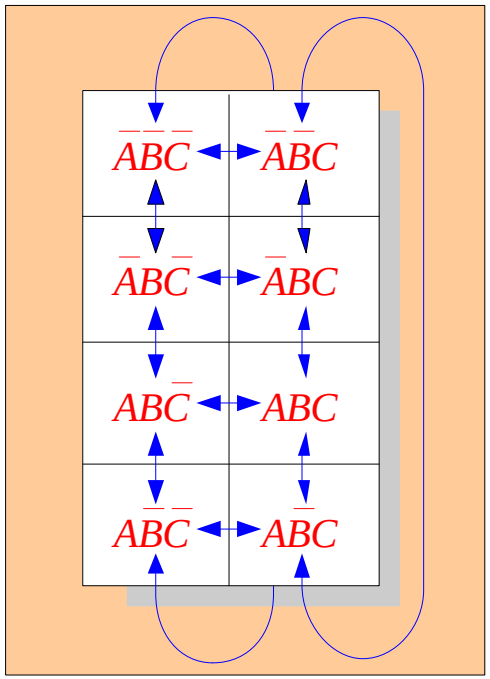


Mapas de Karnaugh

O mapa de Karnaugh de um circuito contém a mesma informação que a tabela de verdade desse circuito.

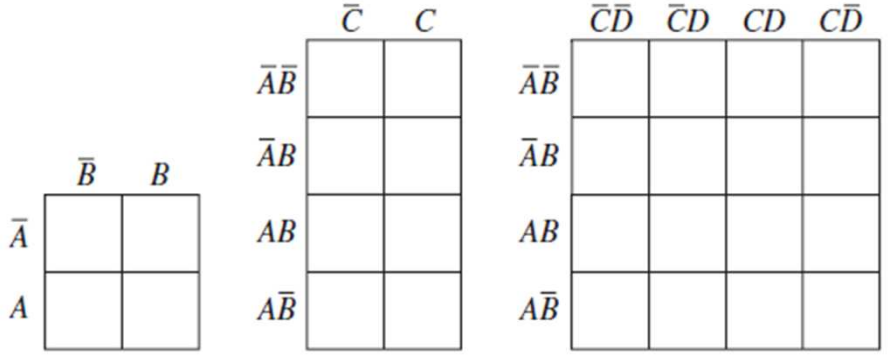
Os mapas são constituídos por um conjunto e células cujas coordenadas (linha e coluna) são os valores das variáveis (“1” ou “0”) que a função toma para a combinação de valores das variáveis correspondentes à linha e à coluna. Cada célula corresponde a uma linha da tabela de verdade.

As coordenadas são representadas em código Gray.



The map shown is for three variables labeled *A, B,* and *C*. Each cell represents one possible product term.

Each cell differs from an adjacent cell by only one variable.

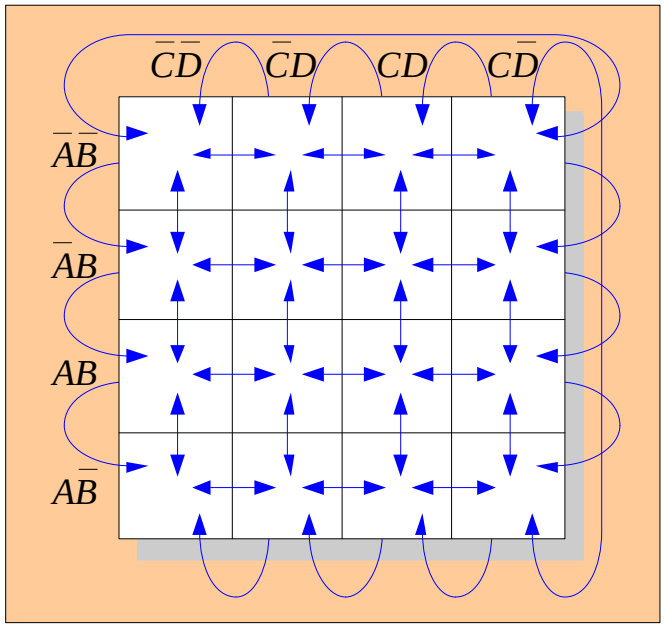


Two-, three-, and four-variable Karnaugh maps.

Mapas de Karnaugh

The Karnaugh map (K-map) is a tool for simplifying combinational logic with 3 or 4 variables. For 3 variables, 8 cells are required (2^3).

A 4-variable map has an adjacent cell on each of its four boundaries as shown.



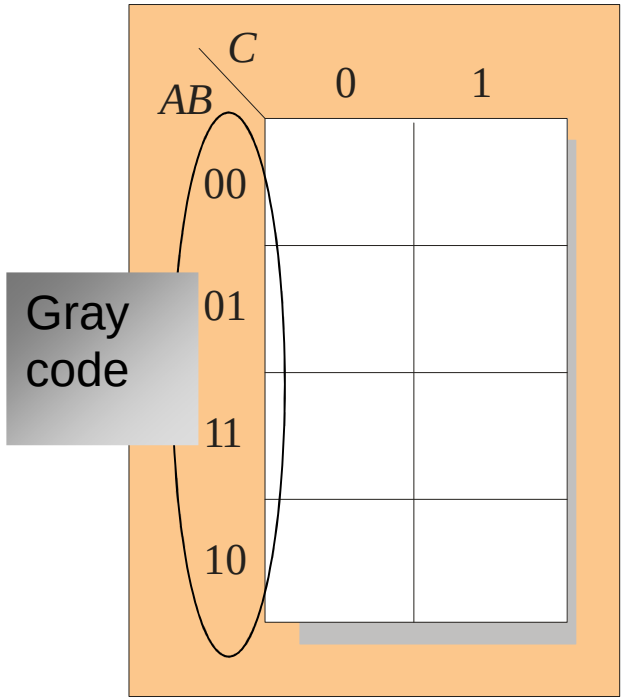
Each cell is different only by one variable from an adjacent cell.

Grouping follows the rules given in the text.

The following slide shows an example of reading a four variable map using binary numbers for the variables…

Mapas de Karnaugh

Cells are usually labeled using 0's and 1's to represent the variable and its complement.



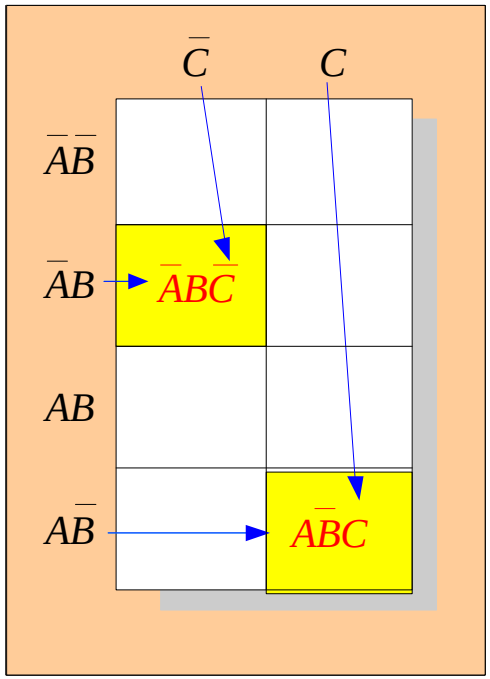
The numbers are entered in gray code, to force adjacent cells to be different by only one variable.

Ones are read as the true variable and zeros are read as the complemented variable.

Alternatively, cells can be labeled with the variable letters. This makes it simple to read, but it takes more time preparing the map.

Read the terms for the yellow cells.

The cells are $\overline{A}B\overline{C}$ and $A\overline{B}C$.



Mapas de Karnaugh

Metodologia:

- Todos os “1” devem ser lidos pelo menos uma vez
- Grupos de “1” em potencias de 2, e retangulares formam uma leitura
- O grupo deve ser o maior possível
- Deve-se ter o menor número possível de leituras
- Leitura das variáveis que se mantem constantes
- A leitura deve iniciar-se pelos “1” mais isolados
- Os “1” Com mais de uma opção de leitura são deixados para o final.

- Na prática é corrente escreverem-se apenas os “1”, omitindo os “0”
- Os mapas de Karnaugh de 5 ou de 6 varáveis podem ser obtidos por sobreposição ou espelhamento de mapas de 4 varáveis.

Códigos digitais: Código Gray

Gray Code

Gray code is an unweighted code that has a single bit change between one code word and the next in a sequence.

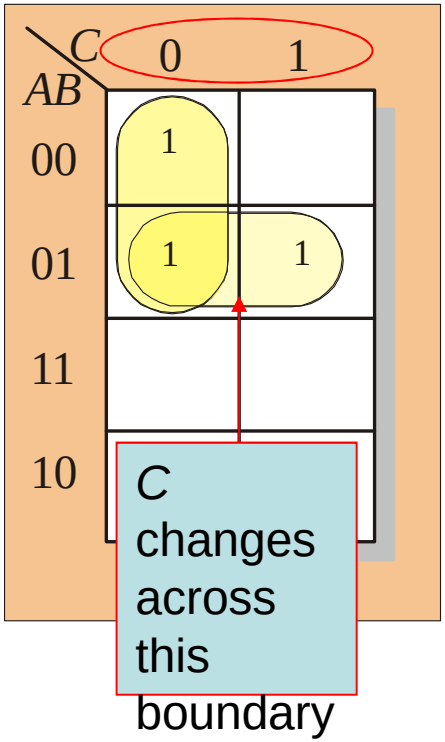
Gray code is used to avoid problems in systems where an error can occur if more than one bit changes at a time.

Decimal	Binary	Gray code
0	00	**00**
1	01	**01**
2	10	**11**
3	**11**	**10**

Mapas de Karnaugh - Exemplo de aplicação

K-maps can simplify combinational logic by grouping cells and eliminating variables that change.

Group the 1's on the map and read the minimum logic.

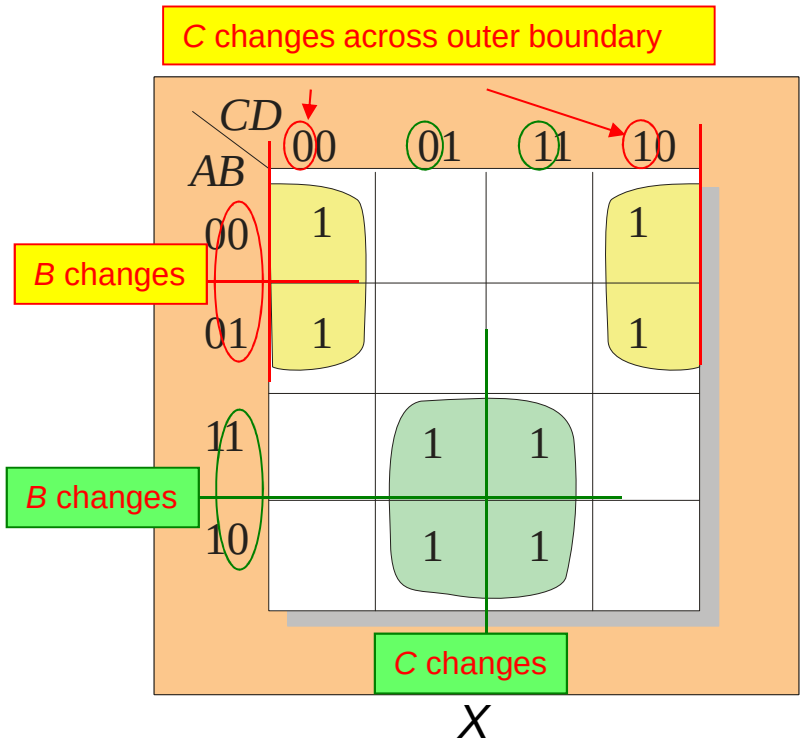


1. Group the 1's into two overlapping groups as indicated.
2. Read each group by eliminating any variable that changes across a boundary.
3. The vertical group is read $\overline{A}\,\overline{C}$.
4. The horizontal group is read $\overline{A}B$.

$$X = \overline{A}\,\overline{C} + \overline{A}B$$

Mapas de Karnaugh - Exemplo de aplicação

Group the 1's on the map and read the minimum logic.

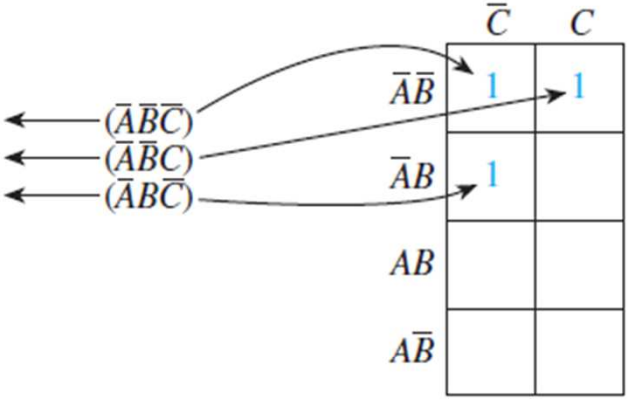


1. Group the 1's into two separate groups as indicated.
2. Read each group by eliminating any variable that changes across a boundary.
3. The upper (yellow) group is read as $\overline{AD}$.
4. The lower (green) group is read as AD.

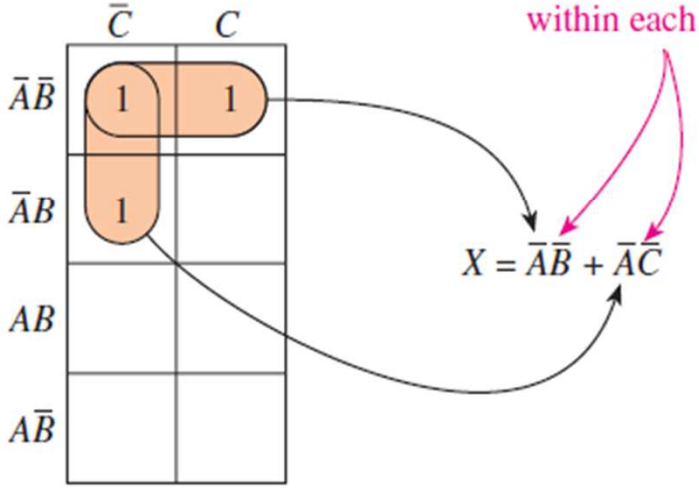
$$X = \overline{A}\,\overline{D} + AD$$

Mapas de Karnaugh - Exemplo de aplicação

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



Truth table and Karnaugh map of $X = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}B\overline{C}$.



Encircling adjacent cells in a Karnaugh map.

Mapas de Karnaugh - Exemplo de aplicação

Simplify the following equation using the Karnaugh mapping procedure:

$$X = \overline{A}B\overline{C}D + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}D + AB\overline{C}D + AB\overline{C}\,\overline{D} + ABCD$$

Solution: Because there are four different variables in the equation, we need a 16-cell map ($2^4 = 16$), as shown in Figure 5–90.

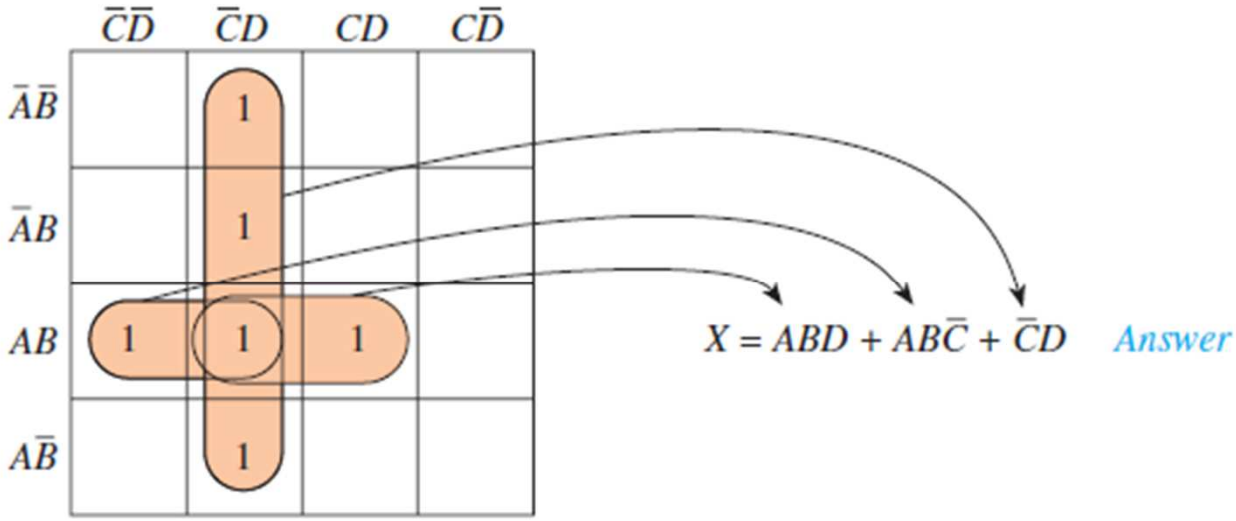


Figure 5–90 Solution to Example 5–29.

Exercício: Mapas de Karnaugh

Simplify the following equation using the Karnaugh mapping procedure:

$$X = B\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}D + \overline{A}BCD + ABCD$$

Solution: Notice in Figure 5–91 that the $B\overline{C}\,\overline{D}$ term in the original equation fills in *two* cells: $AB\overline{C}\,\overline{D} + \overline{A}B\overline{C}\,\overline{D}$. Also notice in Figure 5–91 that we could have encircled four cells and then two cells, but that would not have given us the simplest final equation. By encircling four cells and then another four cells, we are sure to get the simplest final equation. (Always encircle the largest number of cells possible, even if some of the cells have already been encircled in another group.)

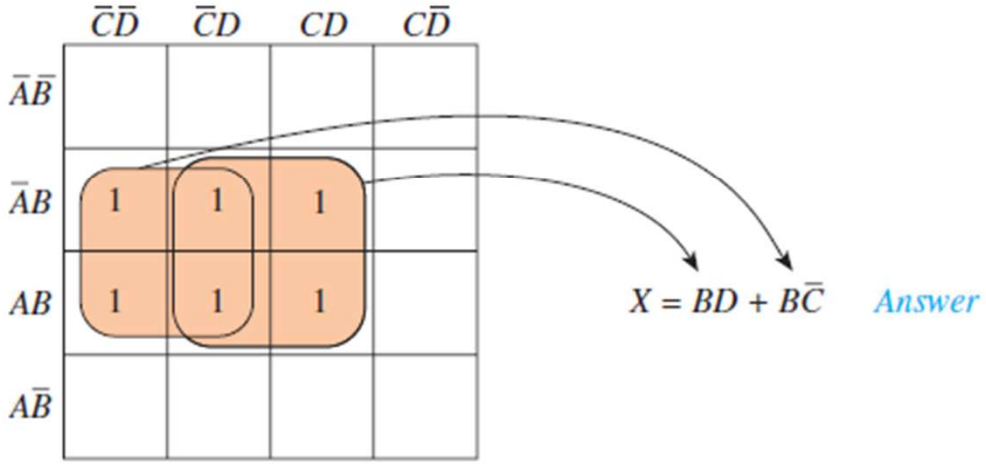


Figure 5–91 Solution to Example 5–30.

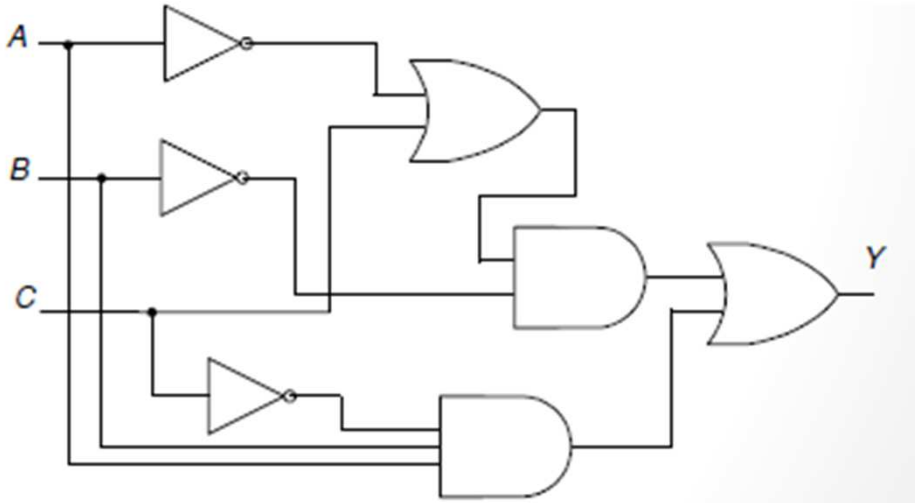
Mapa de Karnaugh - Exemplo de aplicação

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

A \ BC	00	01	11	10
0	1	1	0	0
1	0	1	0	1

$$Y = \overline{A} \cdot \overline{B} + \overline{B} \cdot C + A \cdot B \cdot \overline{C} =$$

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



Mapa de Karnaugh - Exemplo de aplicação

A \ BC	00	01	11	10
0	1	1		
1			1	1

$$Y = \overline{A} \cdot \overline{B} + A \cdot B = \overline{A \oplus B}$$

A \ BC	00	01	11	10
0		1	1	
1			1	1

$$Y = \overline{A} \cdot C + A \cdot B$$

A \ BC	00	01	11	10
0	1		1	1
1	1		1	1

$$Y = B + \overline{C}$$

AB \ CD	00	01	11	10
00	1		1	1
01		1	1	1
11				1
10	1			1

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

AB \ CD	00	01	11	10
00		1		
01		1	1	
11	1		1	1
10	1			1

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

Circuitos Elétricos e Sistemas Digitais

2018-2019 - 1.º Semestre

Sistemas Digitais

Functions of Combinational Logic

Somadores
"Meio-somador" / Half-Adder

Um meio-somador soma dois bits e produz um resultado (soma) e um carry de saída.

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

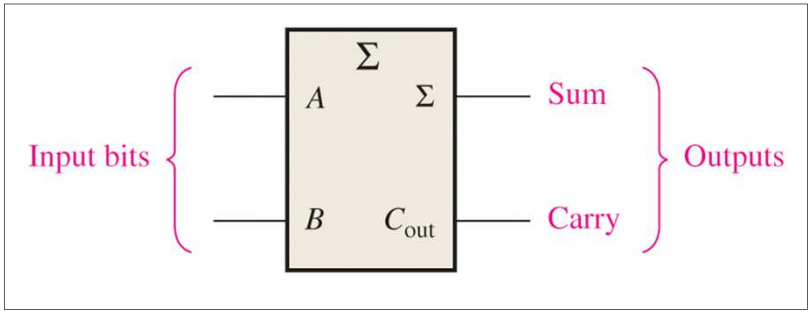
TABLE 5–1 • Half-adder truth table.

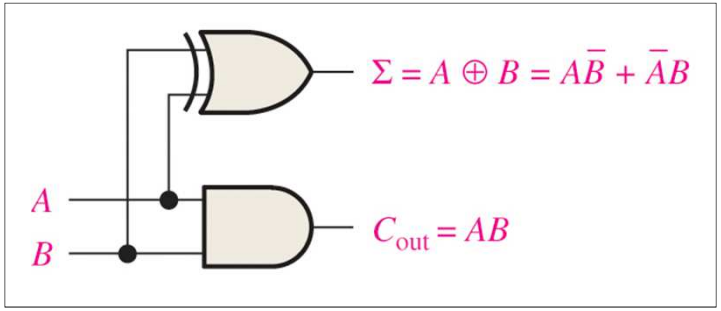
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

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

The logic symbol and equivalent circuit are:





Somadores

Somador Completo / Full-Adders

O somador-completo aceita dois bits de entrada e um carry de entrada, e gera uma saída de soma e um carry de saída.

A **full adder** accepts three binary inputs (*A*, *B,* and Carry-in) and provides two binary outputs (Carry-out and Sum). The truth table summarizes the operation.

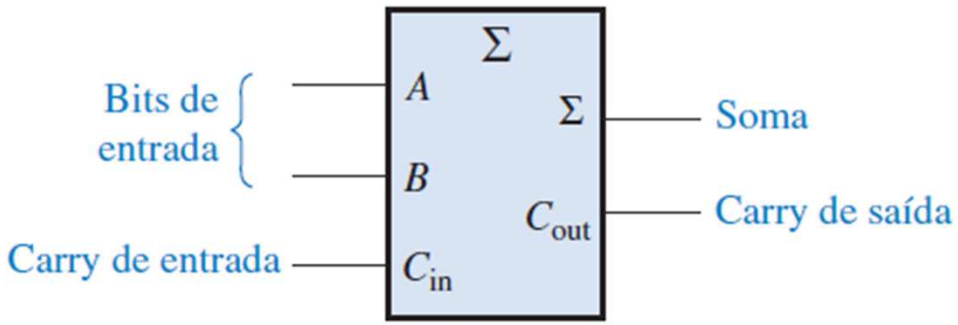


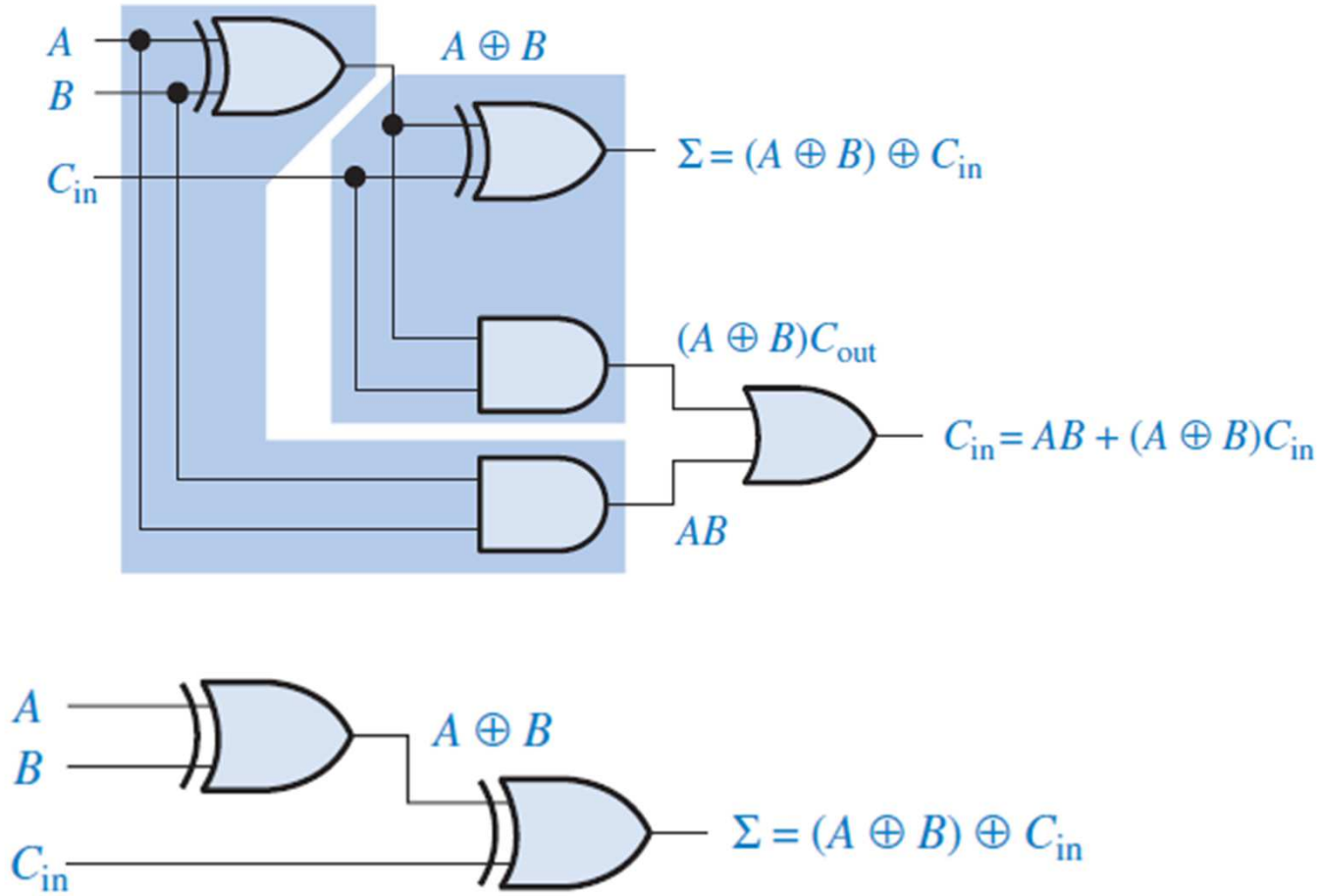
TABLE 5–2 • Full-adder truth table.

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

C_{in} = input carry, sometimes designated as *CI*
C_{out} = output carry, sometimes designated as *CO*
Σ = sum
A and *B* = input variables (operands)

Somador completo / Full-Adder

Circuito lógico para um somador completo



Lógica necessária para construir um somador de três bits.

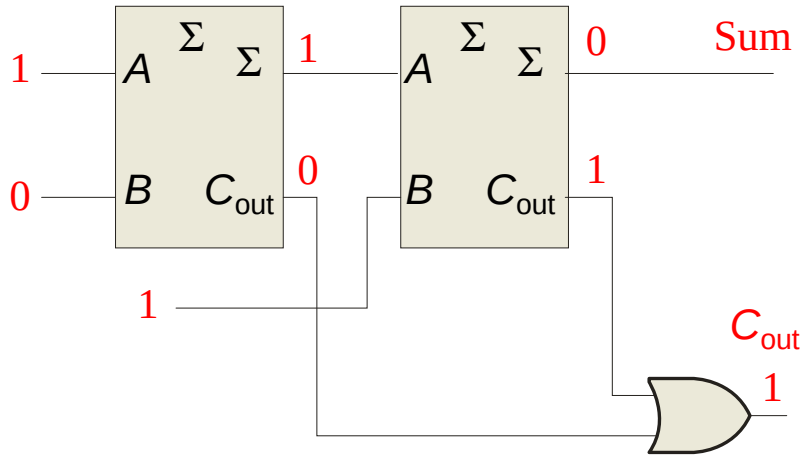
Somador completo / Full-Adder

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

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

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

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



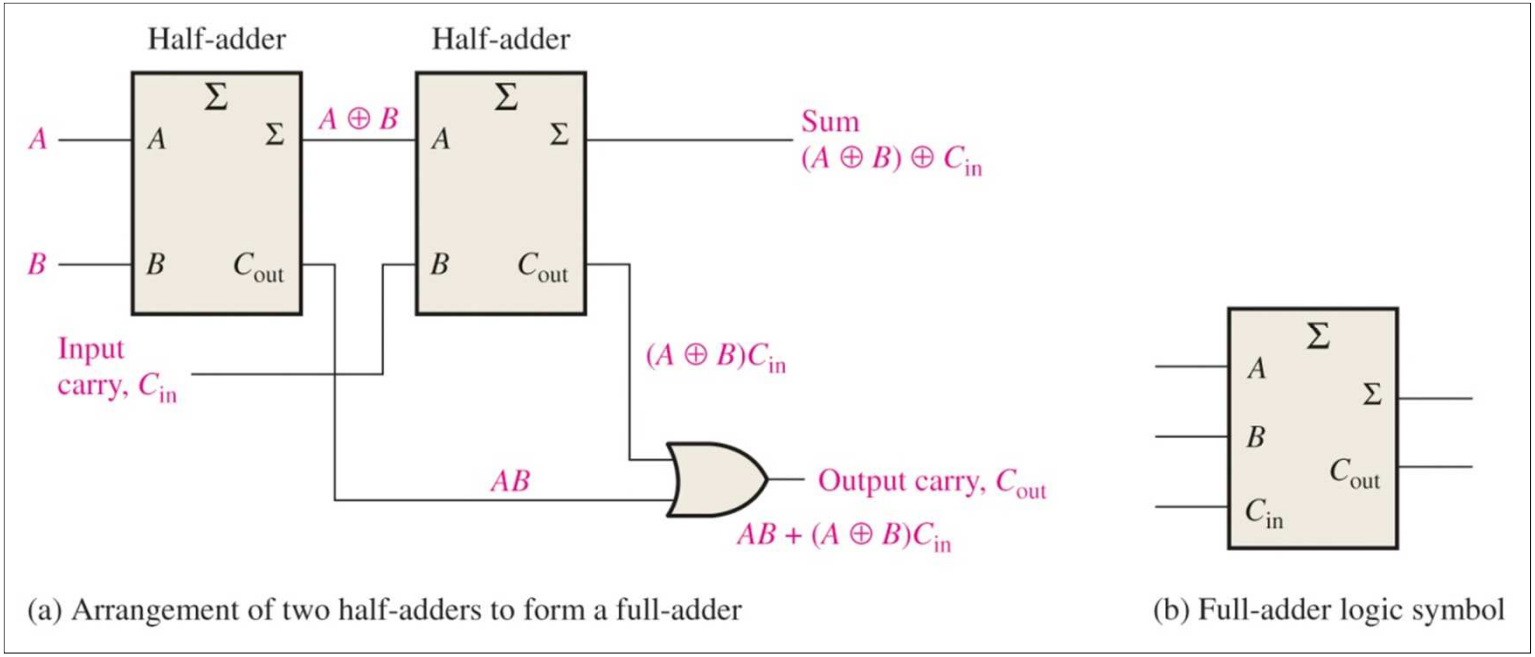
Somadores

Somador Completo / Full-Adders

O somador-completo aceita dois bits de entrada e um carry de entrada, e gera uma saída de soma e um carry de saída.

A **full adder** accepts three binary inputs (*A*, *B,* and Carry-in) and provides two binary outputs (Carry-out and Sum). The truth table summarizes the operation.

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

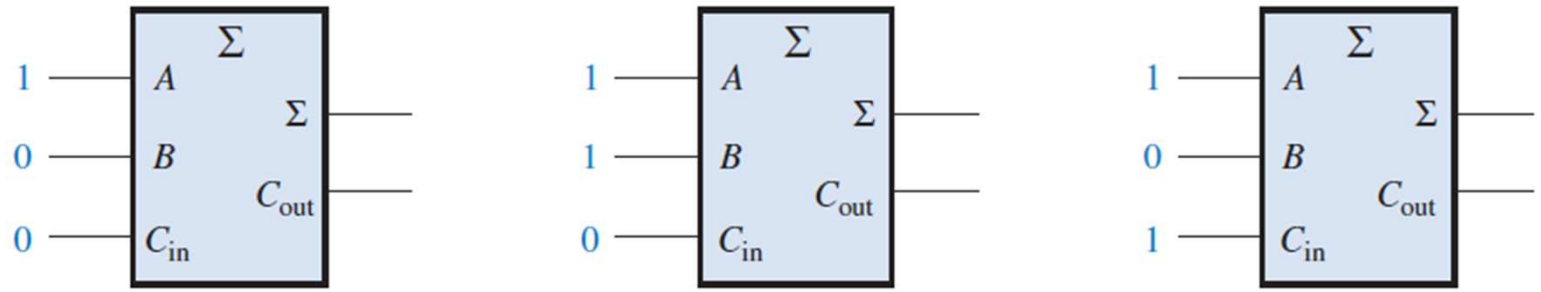


(a) Arrangement of two half-adders to form a full-adder

(b) Full-adder logic symbol

Exemplo 1

Para cada um dos três somadores-completos na Figura 6–6, determine as saídas para as entradas mostradas.



▲ **FIGURA 6–6**

(a) Os bits de entrada são $A = 1$, $B = 0$ e $C_{in} = 0$.

$1 + 0 + 0 = 1$ sem carry

Portanto, $\sum = \mathbf{1}$ e $C_{out} = \mathbf{0}$.

(b) Os bits de entrada são $A = 1$, $B = 1$ e $C_{in} = 0$.

$1 + 1 + 0 = 0$ com carry de 1

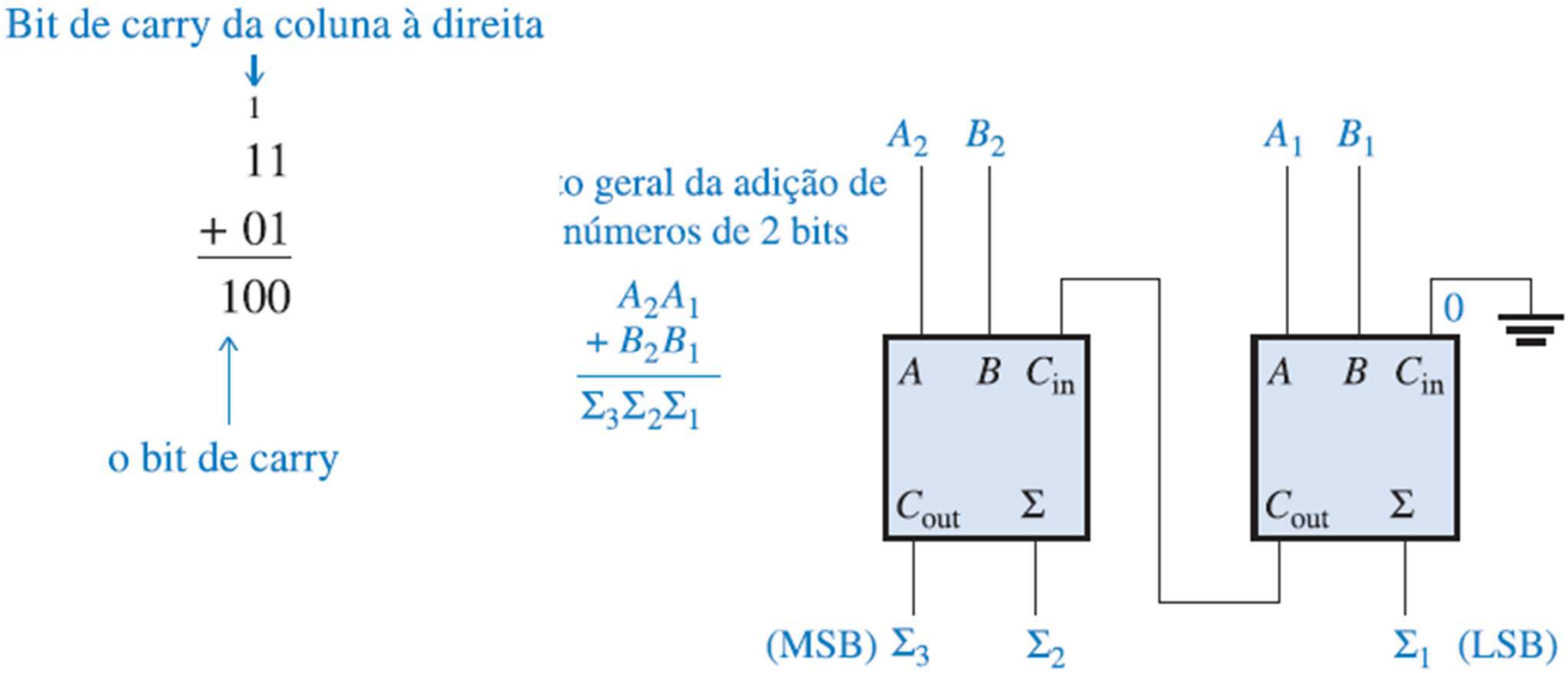
Portanto, $\sum = \mathbf{0}$ e $C_{out} = \mathbf{1}$.

(c) Os bits de entrada são $A = 1$, $B = 0$ e $C_{in} = 1$.

$1 + 0 + 1 = 0$ com carry de 1

Portanto, $\sum = \mathbf{0}$ e $C_{out} = \mathbf{1}$.

Somadores Binários Paralelos / Parallel Adders

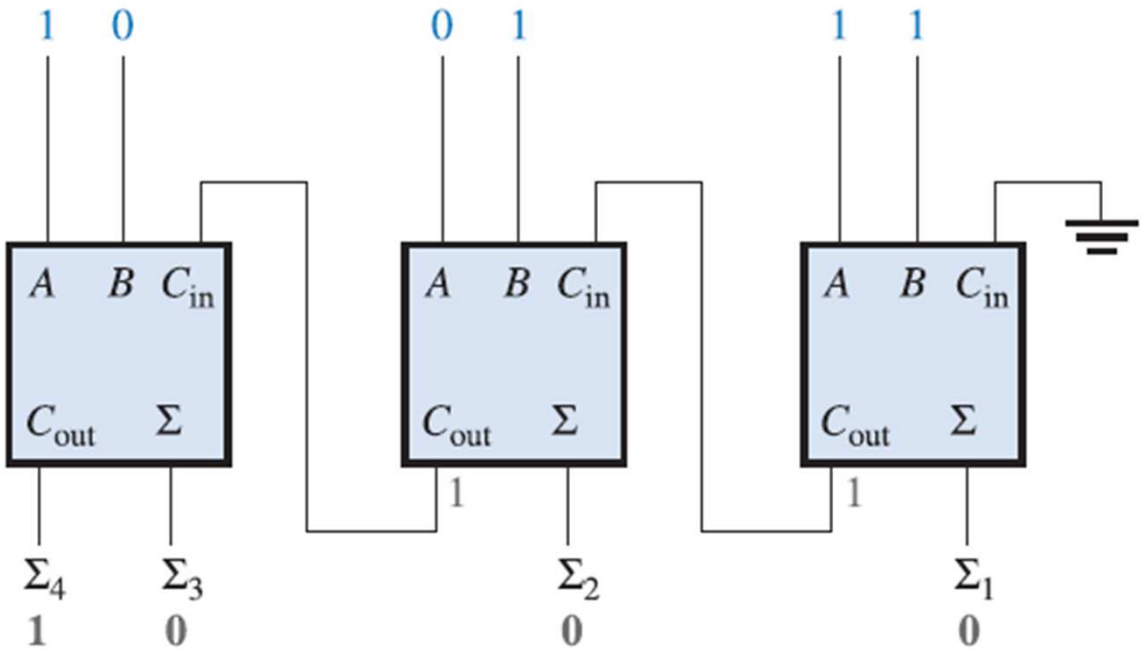


Na representação de um número binário ou outro código ponderado **aqui** *o LSB é o bit mais à direita numa representação horizontal e o bit mais alto numa representação vertical, a menos que seja especificado algo em contrário.*

Somadores Binários Paralelos / Parallel Adders

Exemplo:

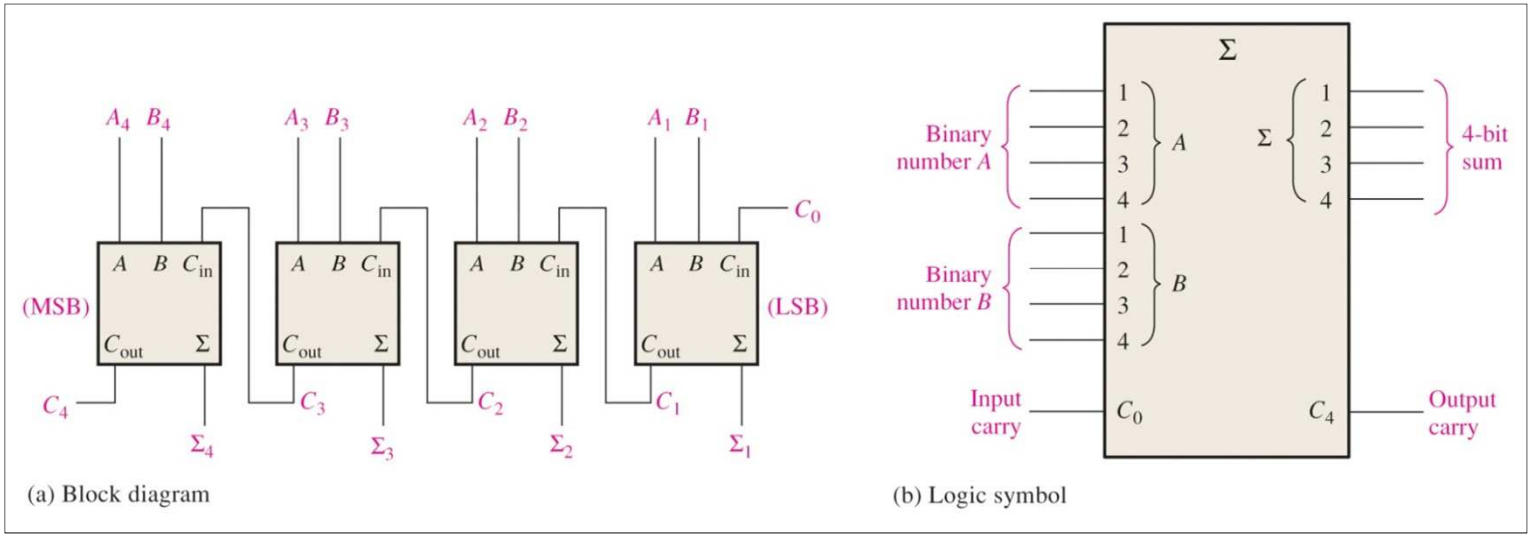
Determine a soma gerada pelo somador paralelo de 3 bits visto na Figura 6–8 e mostre os carries intermediários quando os números binários 101 e 011 são somados.



Os LSBs dos dois números são somados no somador-completo mais à direita.
Os bits do resultado e os carries intermediários são indicados em cinza na Figura 6–8.

Somadores Binários Paralelos / Parallel Adders

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



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

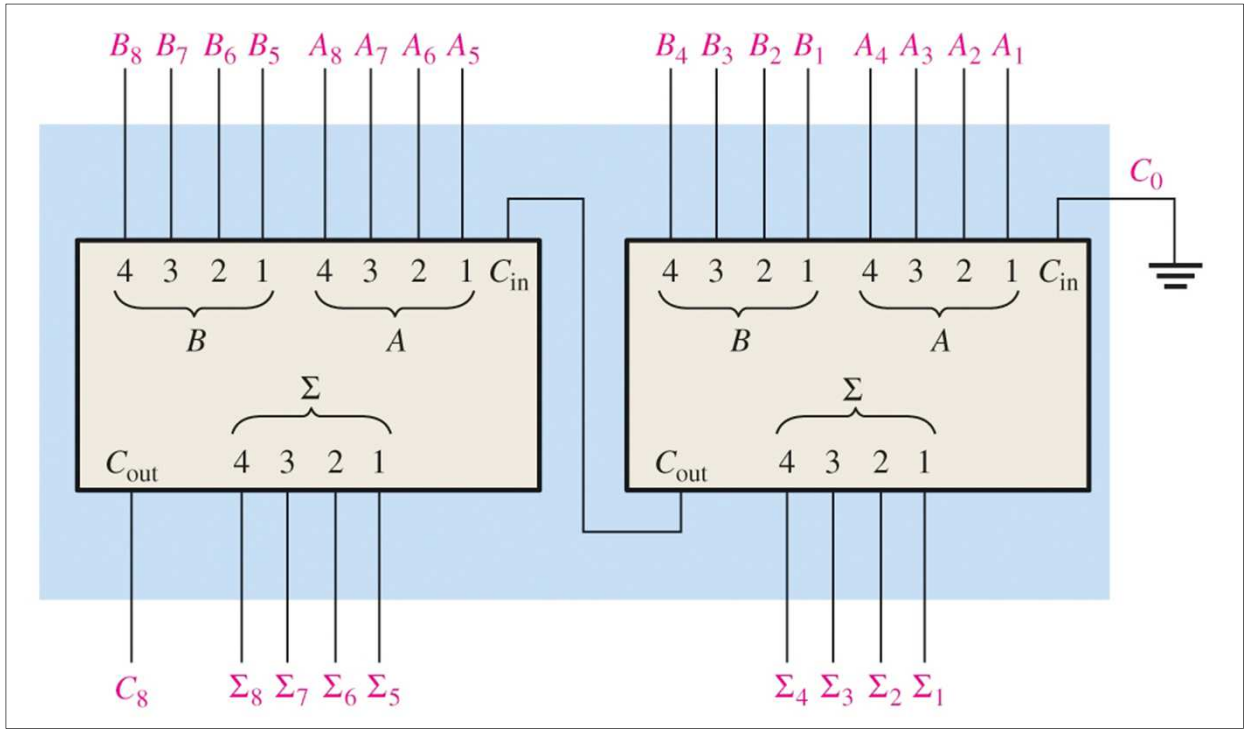
TABLE 6–3

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

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

Expansão de um somador / Adder Expansion

Two four-bit adders can be **cascaded** to form an 8-bit adder as shown.



The carry-in (C_0) pin on the lower-order adder is grounded and the carry-out pin is connected to the C_0 pin of the higher-order adder.

Comparadores / Comparators
Detetor de igualdade

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

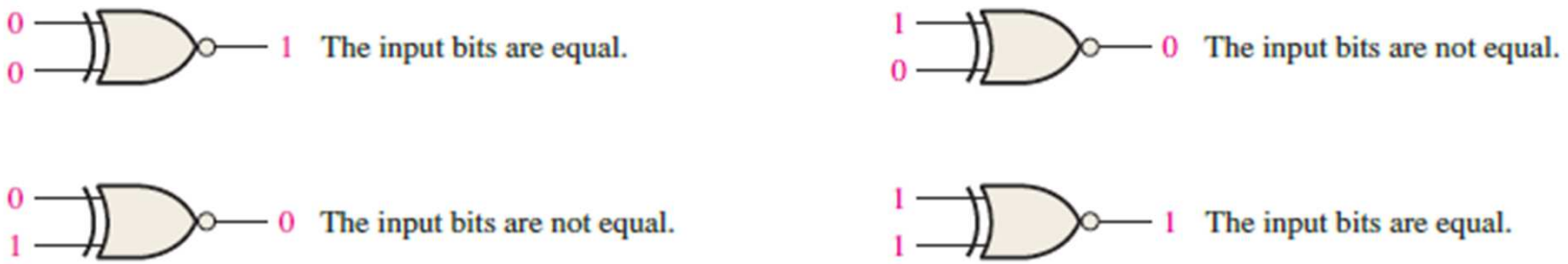


FIGURE 6–18 Basic comparator operation.

Comparadores / Comparators

Detetor de igualdade

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

How could you test two 2-bit numbers for equality?

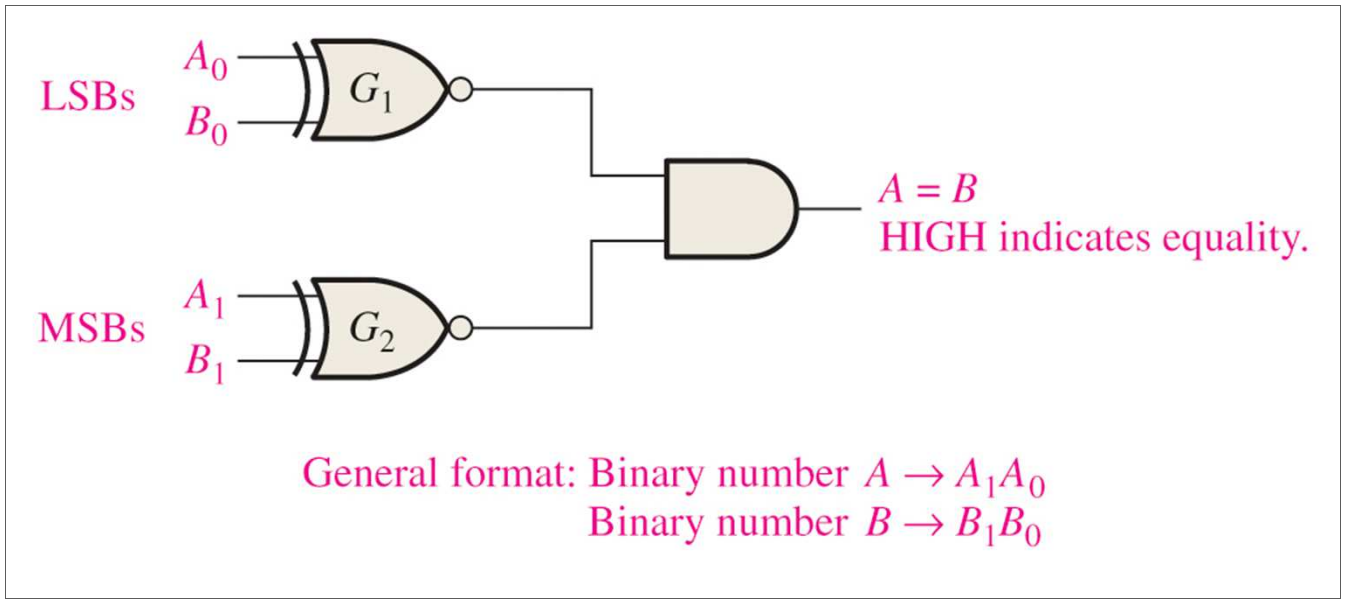


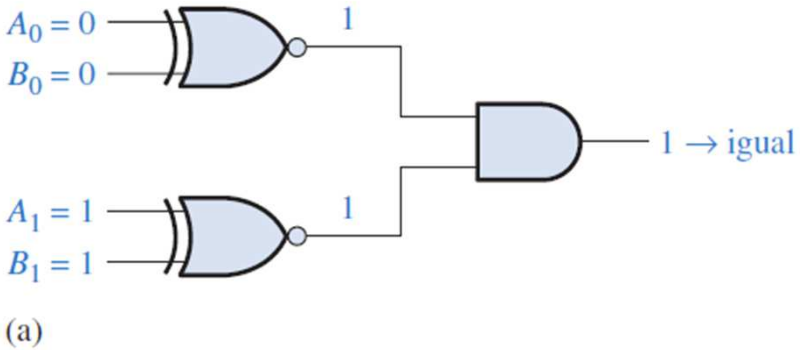
Diagrama lógico para comparação de igualdade de dois números de 2 bits.

AND the outputs of two XNOR gates

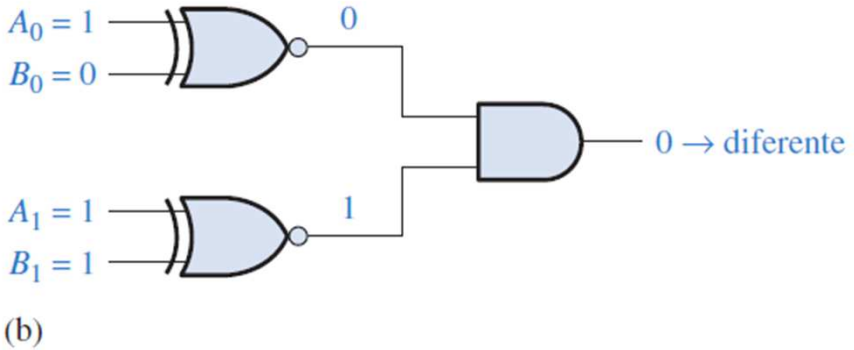
Comparadores / Comparators
Exemplo: Igualdade

Considerando os seguintes conjuntos de números binários e o circuito comparador mostrado na Figura 6–21, determine a saída do circuito para cada conjunto.

(a) 10 e 10 **(b)** 11 e 10



(a)



(b)

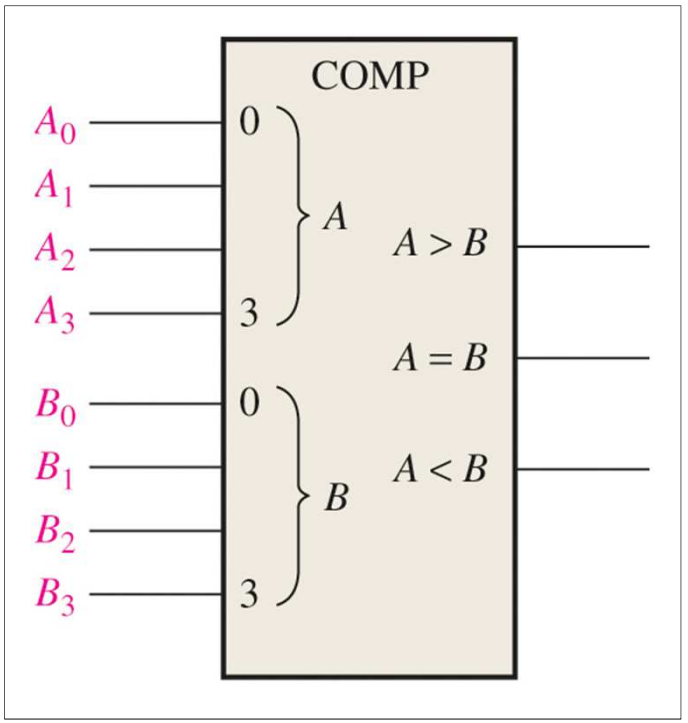
(a) A saída é nível **1** para as entradas 10 e 10,

(b) A saída é nível **0** para as entradas 11 e 10,

Comparadores / Comparators

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

Símbolo lógico para um comparador de 4 bits com indicador de desigualdade

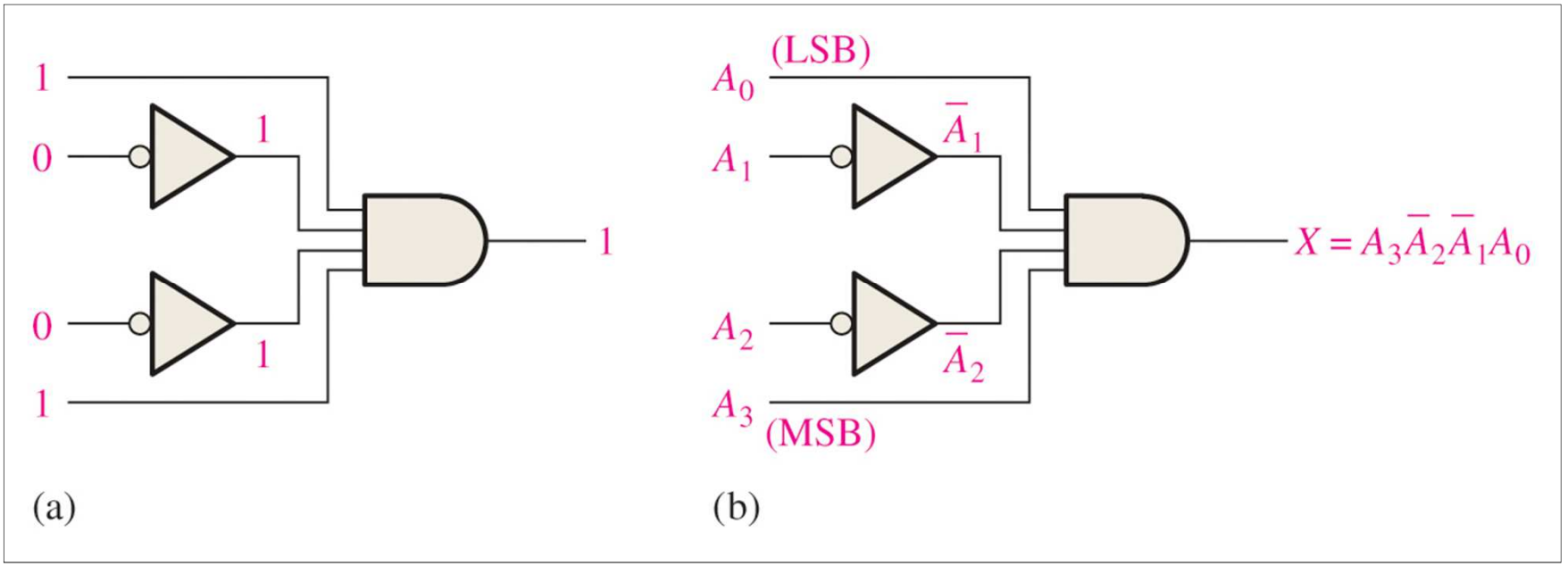


Cascading inputs are provided to expand the comparator to larger numbers.

Descodificadores /Decoders

A **decoder** is a logic circuit that detects the presence of a specific combination of bits at its input.

A simple decoder that detects the presence of the binary code 1001 is shown.



The circuit on the left has an active HIGH output for the inputs shown; the circuit on the right shows the logic expressions for the various gate outputs.

Descodificadores /Decoders

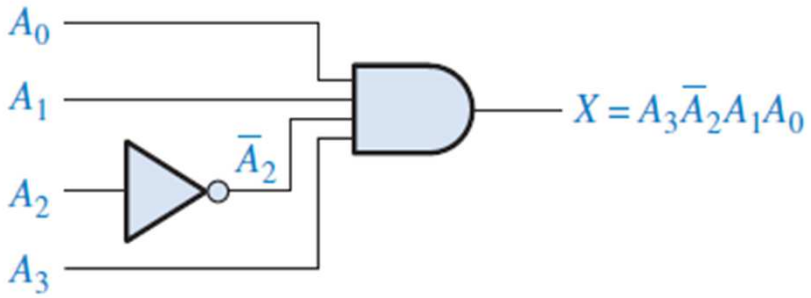
Exemplo:

Determine a lógica necessária para decodificar o número binário 1011 produzindo um nível ALTO na saída.

A função de decodificação pode ser obtida complementando apenas as variáveis que aparecem como 0 no número binário desejado, como a seguir:

$$X = A_3\overline{A}_2A_1A_0 \qquad (1011)$$

Essa função pode ser implementada conectando as variáveis verdadeiras (não-complementadas) A_0, A_1 e A_3 diretamente nas entradas de uma porta AND e invertendo a variável A_2 antes de aplicá-la na entrada da porta AND. A lógica de decodificação é mostrada na Figura 6–27.



▶ **FIGURA 6–27**

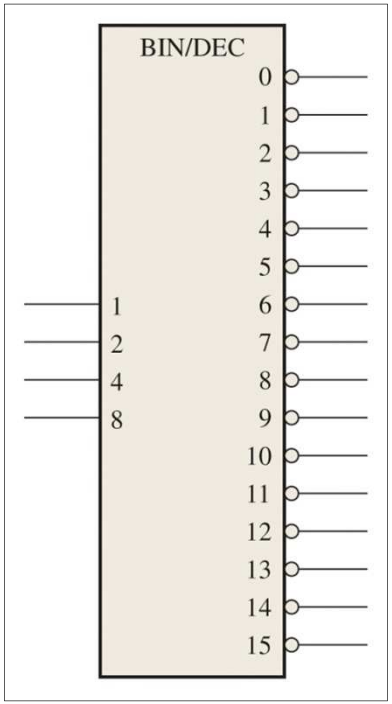
Lógica de decodificação para gerar uma saída de nível ALTO quando 1011 estiver nas entradas.

Descodificadores /Decoders

Descodificador de 4 bits

IC decoders have multiple outputs to decode any combination of inputs. For example the binary-to-decimal decoder shown here has 16 outputs – one for each combination of binary inputs. The first 8 lines of the circuit truth table are shown.

DECIMAL DIGIT	BINARY INPUTS A_3	A_2	A_1	A_0	DECODING FUNCTION	OUTPUTS 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1



Descodificadores /Decoders

Descodificador de 4 bits

tabela completa

TABLE 6–4

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

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

Descodificadores /Decoders

Descodificador de BCD para Decimal

BCD Decoder/Driver

BCD decoding functions.

Decimal Digit	BCD Code A_3	A_2	A_1	A_0	Decoding Function
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$

BCD (Binary-Coded Decimal): decimal codificado em binário

Descodificadores /Decoders

Descodificador de BCD para Decimal

Exemplo: CI 74HC42

O CI 74HC42 é um decodificador de BCD para decimal. O símbolo lógico é mostrado na Figura 6–32. Se as formas de onda de entrada vistas na Figura 6–33(a) são aplicadas nas entradas do CI 74HC42, mostre as formas de onda de saída.

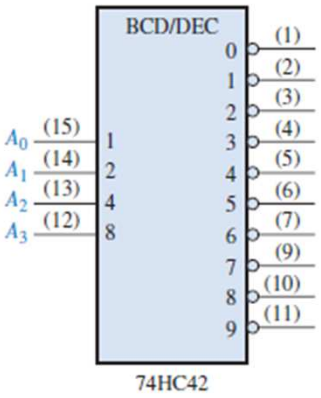
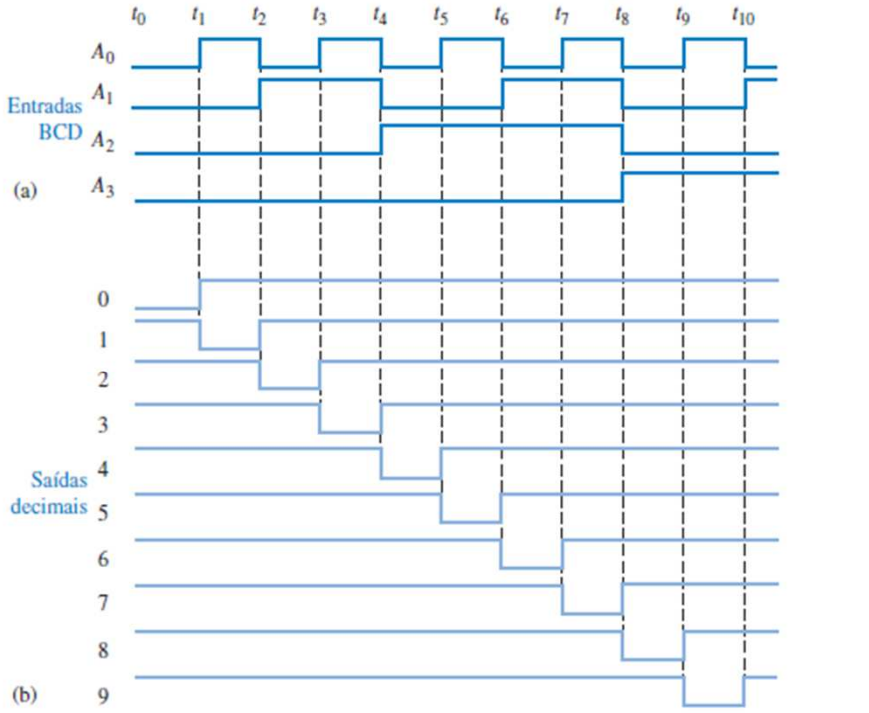


FIGURA 6–32

ecodificador de BCD para decimal
'4HC42).



▲ FIGURA 6–33

Solução As formas de onda de saída são mostradas na Figura 6–33(b). Como podemos ver, as entradas são uma seqüência BCD para os dígitos de 0 a 9. As formas de onda de saída no diagrama de temporização indicam essa seqüência BCD nas saídas de valores decimais.

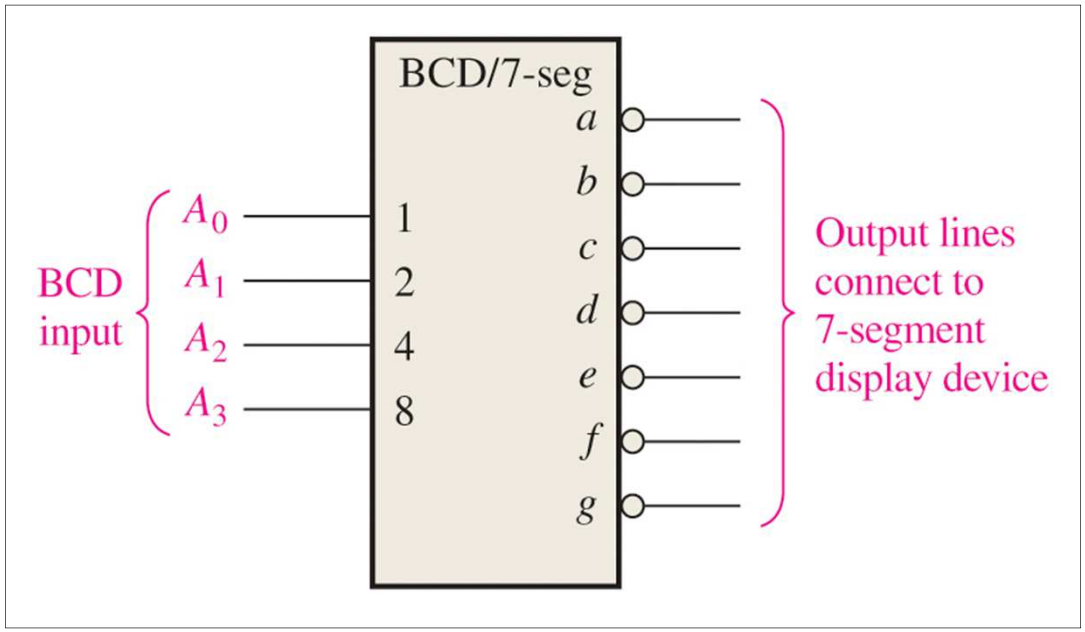
Descodificadores /Decoders

Descodificador de BCD para Decimal

BCD Decoder/Driver

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

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

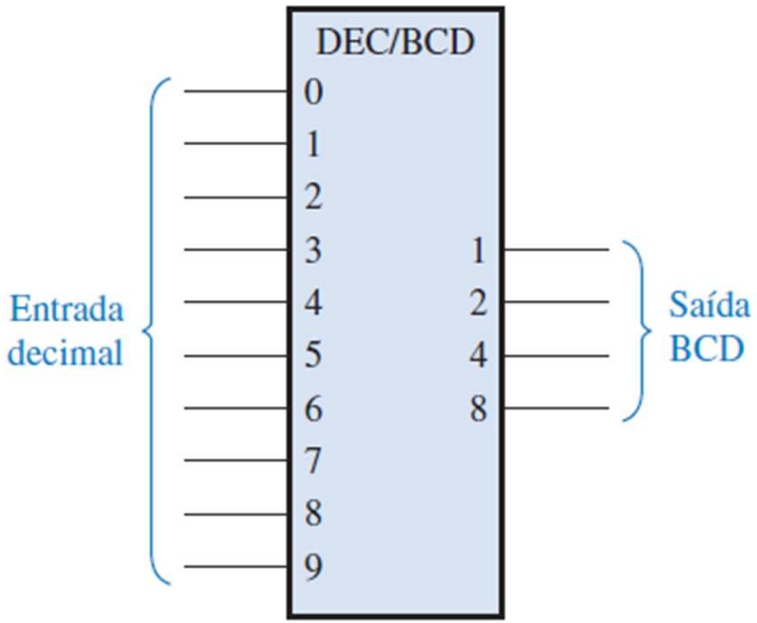


BCD (Binary-Coded Decimal): decimal codificado em binário

Codificadores / Encoders

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

Exemplo: Codificador de decimal para BCD

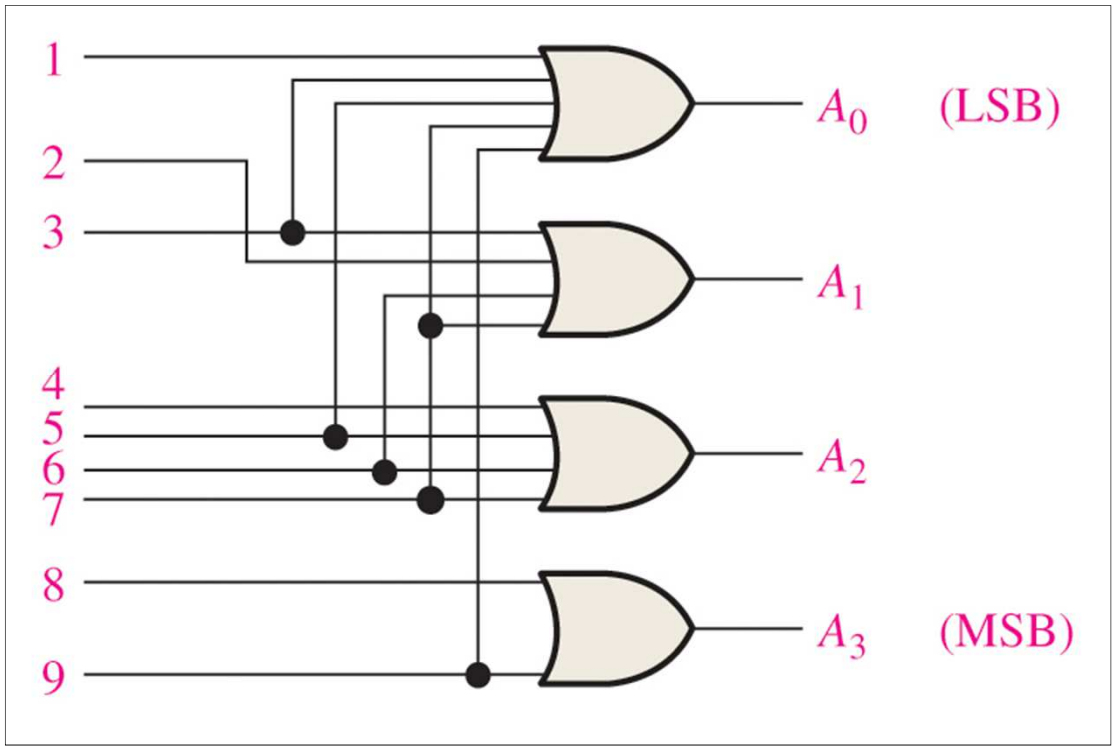


DÍGITO DECIMAL	CÓDIGO BCD A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Codificadores / Encoders

Codificador de decimal para BCD

Circuito lógico necessário para a codificação de cada digito decimal para o código BCD

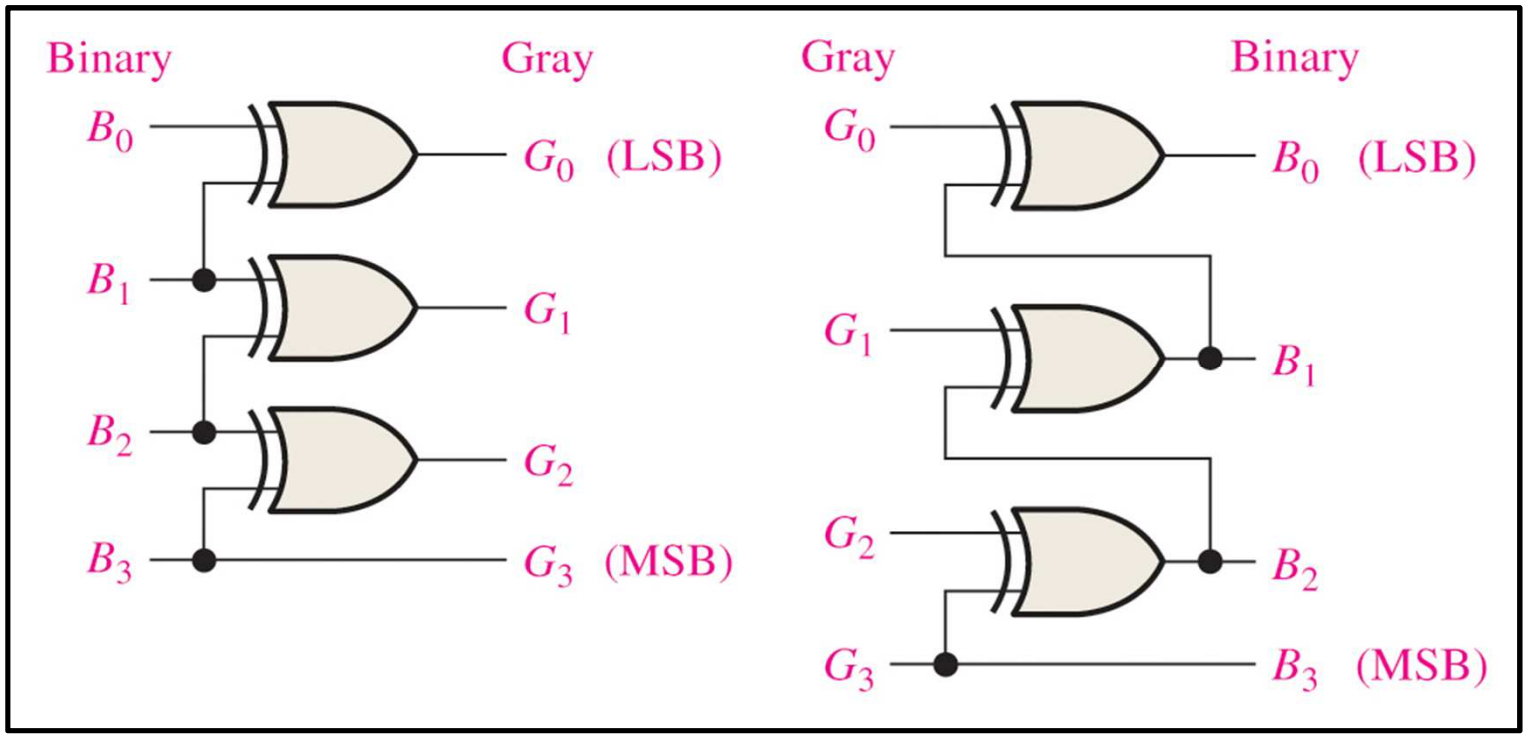


The decimal to BCD is an encoder with an input for each of the ten decimal digits and four outputs that represent the BCD code for the active digit. The basic logic diagram is shown. There is no zero input because the outputs are all LOW when the input is zero.

Conversores de códigos / Code Converters

There are various code converters that change one code to another.

One examples is the four bit binary-to-Gray converter / the Gray-to-binary converter.



Conversores de códigos / Code Converters

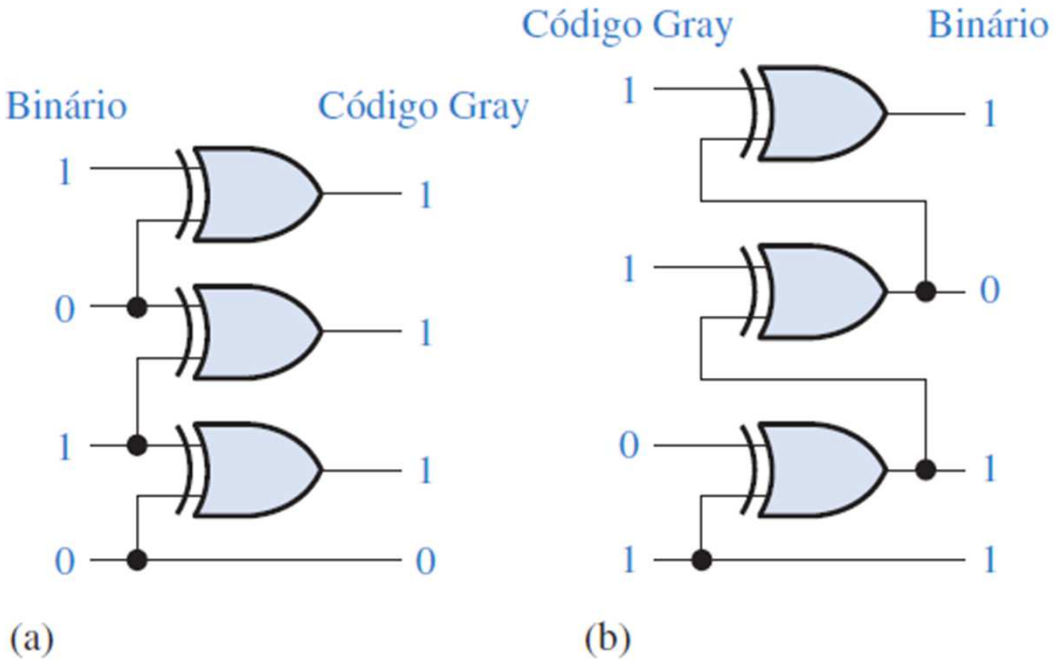
Conversão binário para código Gray e vice-versa

Converta o número binário 0101 para código Gray usando portas EX-OR.

Converta o código Gray 1011 para binário usando portas EX-OR.

(a) 0101_2 é 0111 em código Gray.

(b) 1011 em código Gray é 1101_2.



Conversores de códigos / Code Converters

Conversão de BCD para binário

Os números binários que representam os pesos dos bits BCD são somados para produzir o número binário total.

BIT BCD	PESO EM BCD	(MSB) REPRESENTAÇÃO BINÁRIA						(LSB)
		64	32	16	8	4	2	1
A_0	1	0	0	0	0	0	0	1
A_1	2	0	0	0	0	0	1	0
A_2	4	0	0	0	0	1	0	0
A_3	8	0	0	0	1	0	0	0
B_0	10	0	0	0	1	0	1	0
B_1	20	0	0	1	0	1	0	0
B_2	40	0	1	0	1	0	0	0
B_3	80	1	0	1	0	0	0	0

Conversores de códigos / Code Converters

Conversão de BCD para binário

Exemplo: conversor de BCD para binário

Converta os números BCD 00100111 (decimal 27) e 100110000 (decimal 98) para binário.

Escreva as representações binárias dos pesos de todos os 1s que aparecem nos números para, em seguida, somá-los.

80	40	20	10	8	4	2	1		
0	0	1	0	0	1	1	1		
								0000001	1
								0000010	2
								0000100	4
								+0010100	20
								0011011	Número binário para o decimal 27

80	40	20	10	8	4	2	1		
1	0	0	1	1	0	0	0		
								0001000	8
								0001010	10
								+1010000	80
								1100010	Número binário para o decimal 98

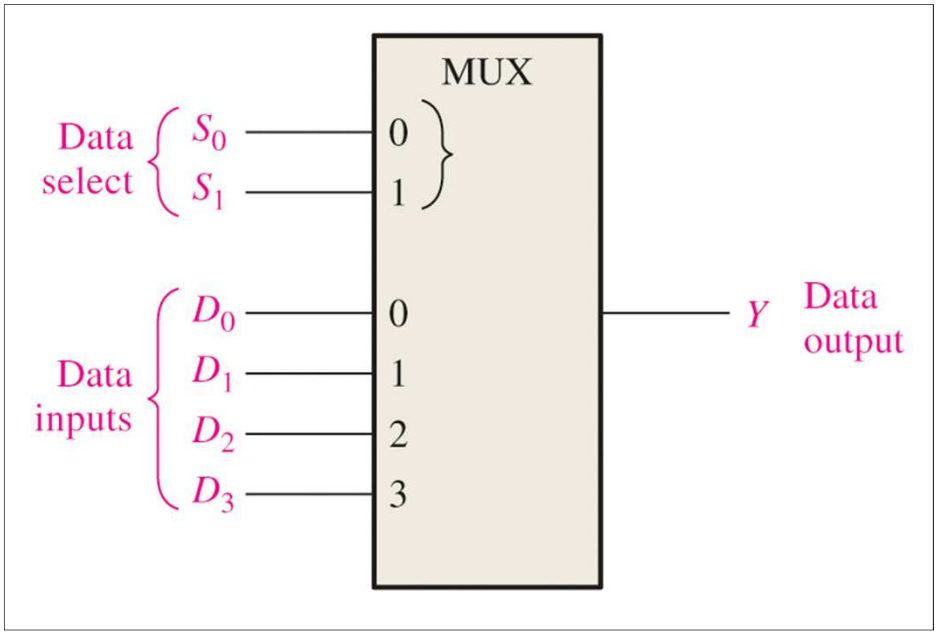
Multiplexadores / Multiplexers

(também referidos como slectores de dados)

A multiplexer (MUX) selects one of several data (*D*) inputs and routes data from that input to the output. The data line that is selected is determined by the select (*S*) inputs.

Exemplo: seletor/multiplexador de dados de 4 linhas para 1 linha

The multiplexer shown has two select (S) inputs that are used to select one of four data (D) inputs.



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

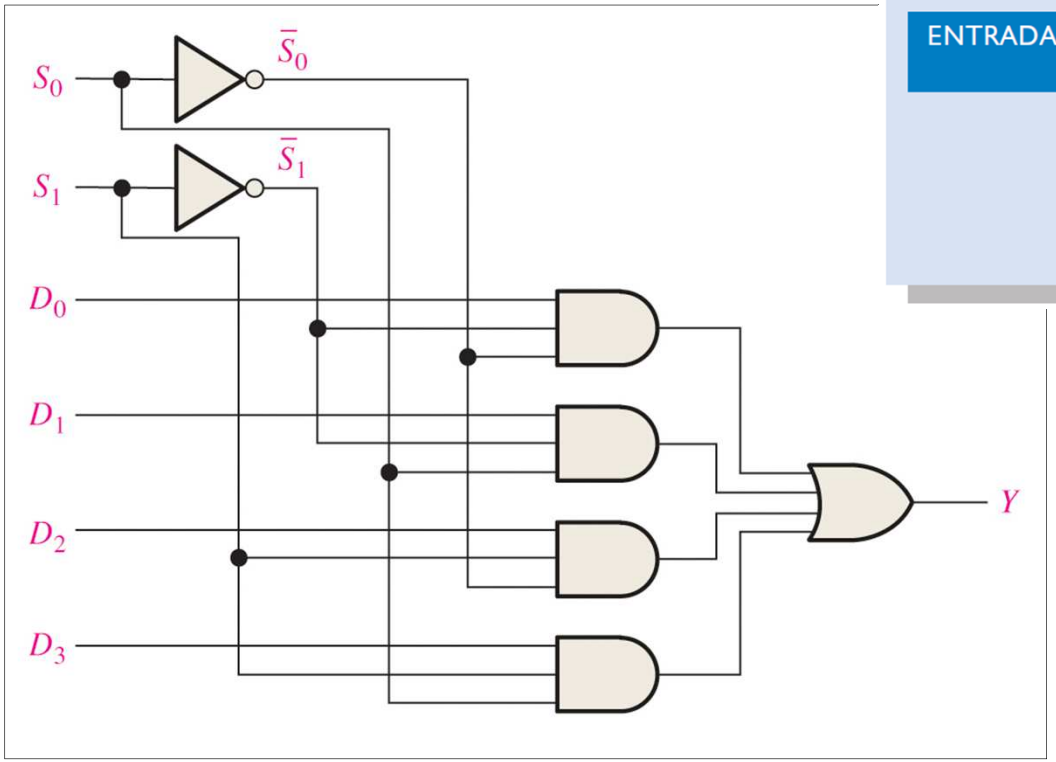
Data-Select Inputs		
S_1	S_0	Input Selected
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Which data line is selected if S_1S_0 = 10?

The select input (10) connects data line 2 to the output.

Multiplexadores / Multiplexers

Diagrama lógico para um multiplexador de 4 (entradas) para 1 (saída)



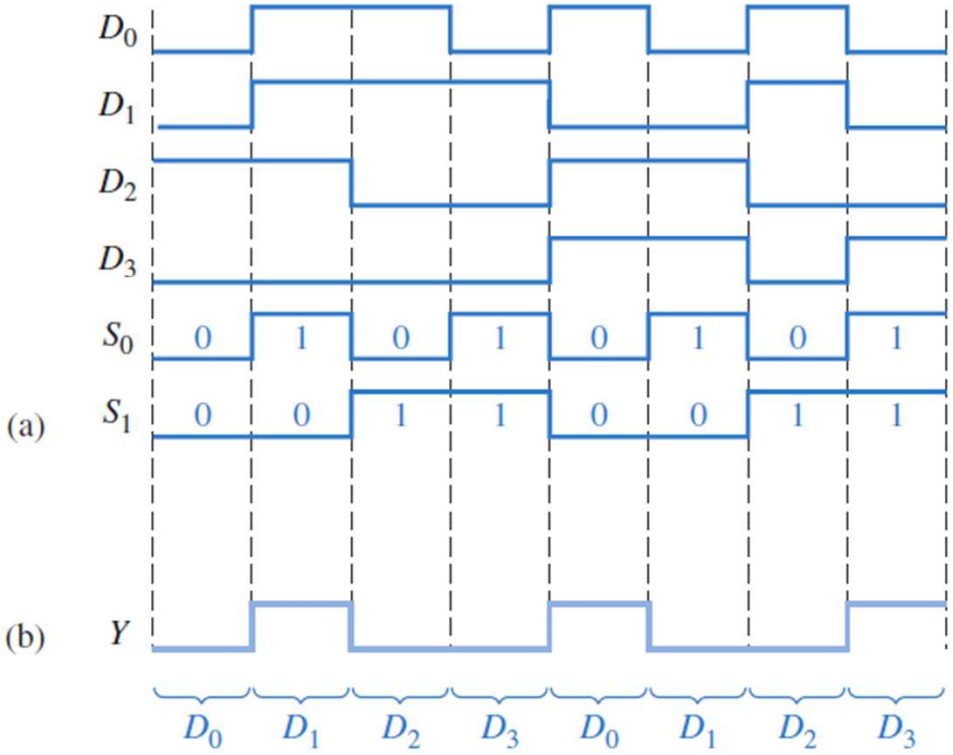
ENTRADAS DE SELEÇÃO DE DADOS S_1	S_0	ENTRADA SELECIONADA
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

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

Multiplexadores / Multiplexers

Exemplo: seletor/multiplexador de dados de 4 para 1

As formas de onda da entrada de dados e das entradas de seleção de dados vistas na Figura 6–48(a) são aplicadas no multiplexador mostrado na Figura 6–47. Determine a forma de onda de saída em relação às entradas.

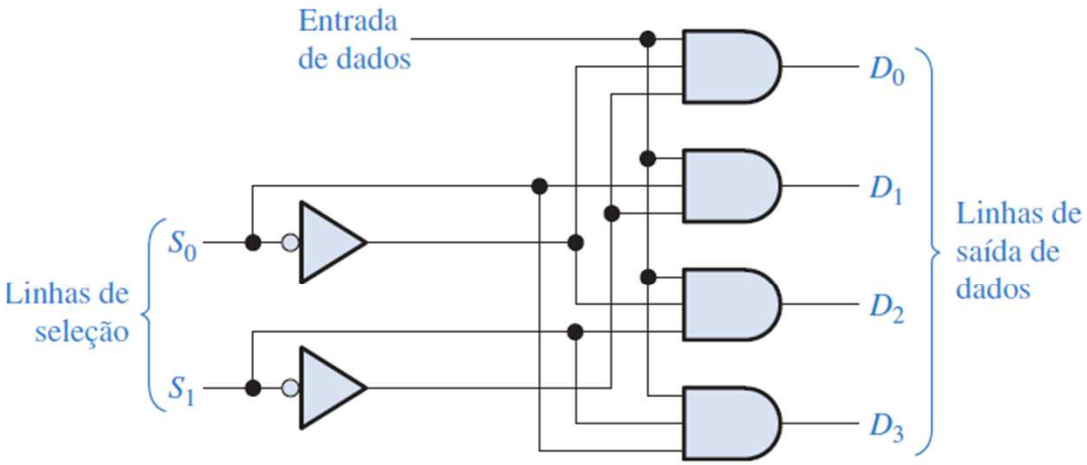


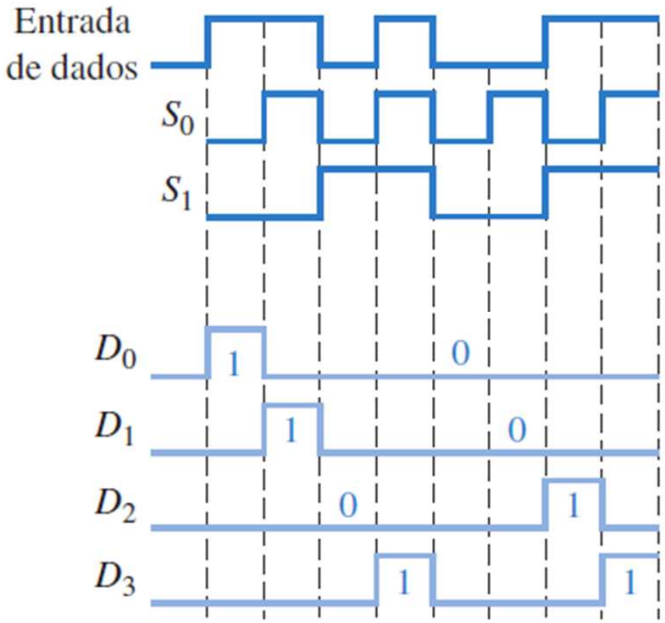
Os estados binários das entradas de seleção de dados durante cada intervalo determina qual dado de entrada é selecionado. Observe que as entradas de seleção de dados passam pela seqüência binária repetitiva: 00, 01, 10, 11, 00, 01, 10, 11 e assim por diante. A forma de onda de saída resultante é mostrada na Figura 6–48(b).

Desmultiplexadores / Demultiplexers

Um **demultiplexador (DEMUX)** basicamente inverte a função da multiplexação. Ele recebe informações digitais a partir de uma linha e as distribui para um determinado número de linhas de saída. Por essa razão, o demultiplexador também é conhecido como distribuidor de dados. Conforme estudaremos, os decodificadores também podem ser usados como demultiplexadores.

Exemplo: demultiplexador de dados de 1 linha para 4 linhas





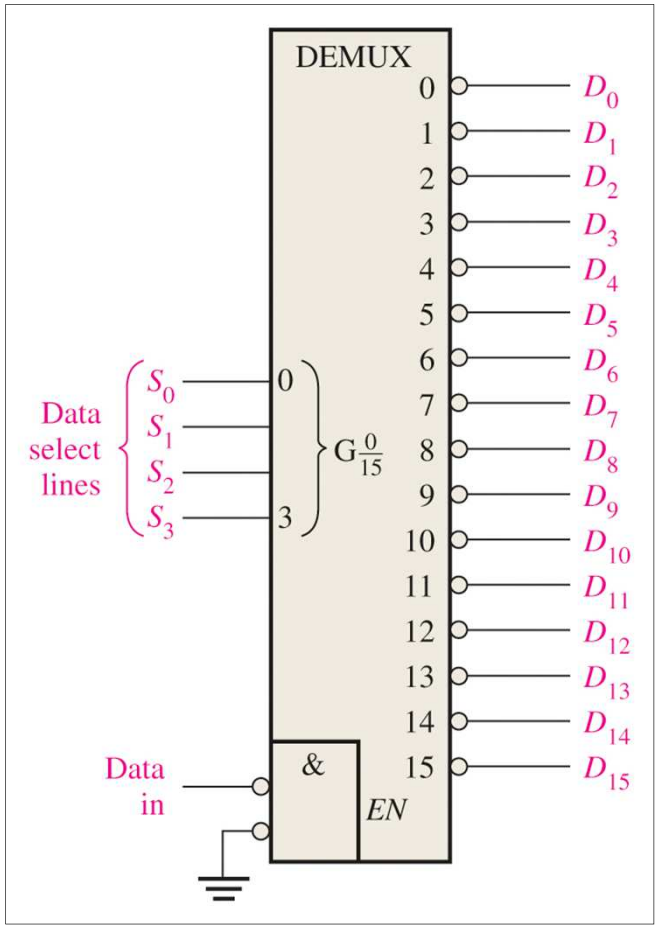
Observe que as linhas de seleção seguem uma seqüência binária de forma que cada bit sucessivo de entrada é direcionado para D_0, D_1, D_2 e D_3 na seqüência, conforme mostra as formas de onda vistas na Figura 6–56.

Desmultiplexadores / Demultiplexers

A demultiplexer (DEMUX) performs the opposite function from a MUX. It switches data from one input line to two or more data lines depending on the select inputs.

Data is applied to one of the data input pin, and routed to the selected output line depending on the select variables. Note that the outputs are active-LOW.

Exemplo de um descodificador usado como desmultiplexador: uma linha de entrada e 16 linhas de saída



Circuitos Elétricos e Sistemas Digitais

2018-2019 - 1.º Semestre

Sistemas Digitais

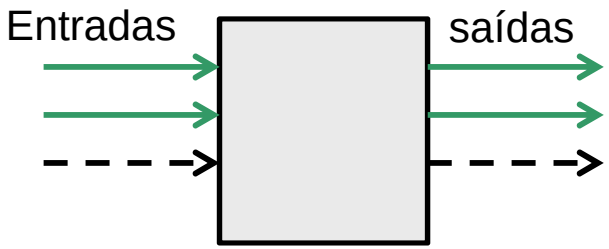
Introdução à Lógica Sequencial

Introdução à lógica sequencial

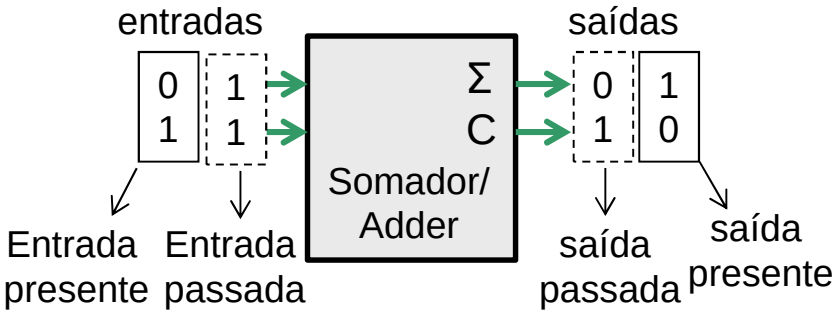
- Nos circuitos **lógicos combinacionais** a saída é função apenas das entradas, i.e., o valor da saída só depende dos valores das entradas, e obedecem às regras da lógica booleana. São descritos por tabelas de verdade.
- Nos **circuitos lógicos sequenciais**, a saída depende dos valores das entradas atuais e da saída passada (saída que resultou das entradas anteriores), isto é, depende também do histórico das entradas. Isto quer dizer que, ao contrário da lógica combinacional, a lógica sequencial tem **memória**. São descritos por tabelas de estados ou diagramas de estado.
- A lógica sequencial é usada para construir **máquinas de estado** finitas, um bloco básico na construção de todos circuitos digitais, e também de circuitos de memória e outros dispositivos.
- Quase todos circuitos digitais práticos são uma mistura de circuitos lógicos combinacionais e circuitos lógicos sequenciais.
- Os circuitos lógicos sequenciais podem ser divididos em dois tipos:
- Circuitos **sequenciais** síncronos: o estado do dispositivo muda apenas de acordo com os sinais de relógio.
- Circuitos **sequenciais assíncronos**: o estado do dispositivo pode mudar a qualquer momento de acordo com as alterações dos valores das entradas.

Lógica combinacional vs lógica sequencial

Lógica combinacional

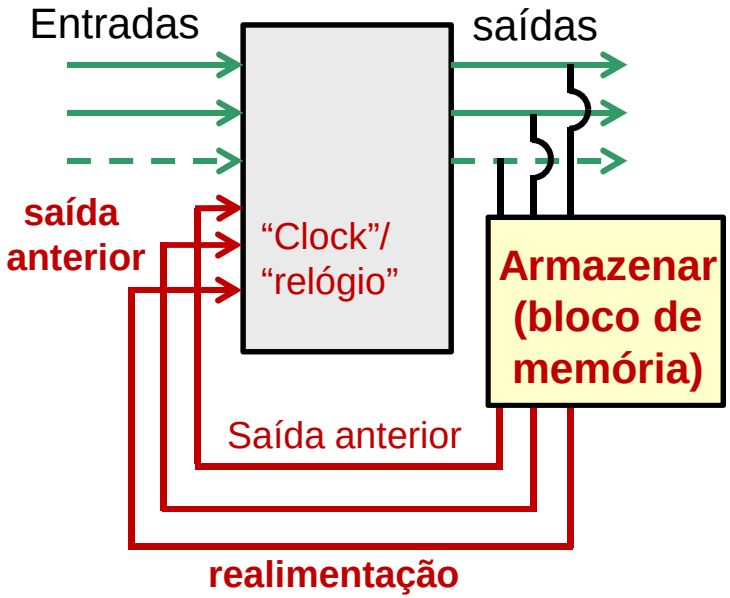


Exemplo de um circuito combinacional



A saída atual depende apenas das entradas atuais.

Lógica sequencial



A saída atual será dependente das entradas atuais e da saída anterior.

Embora as portas lógicas, por si só, não tenham capacidade de armazenar de informação (bits), pode-se criar um elemento de memória com portas lógicas aplicando o conceito de realimentação.

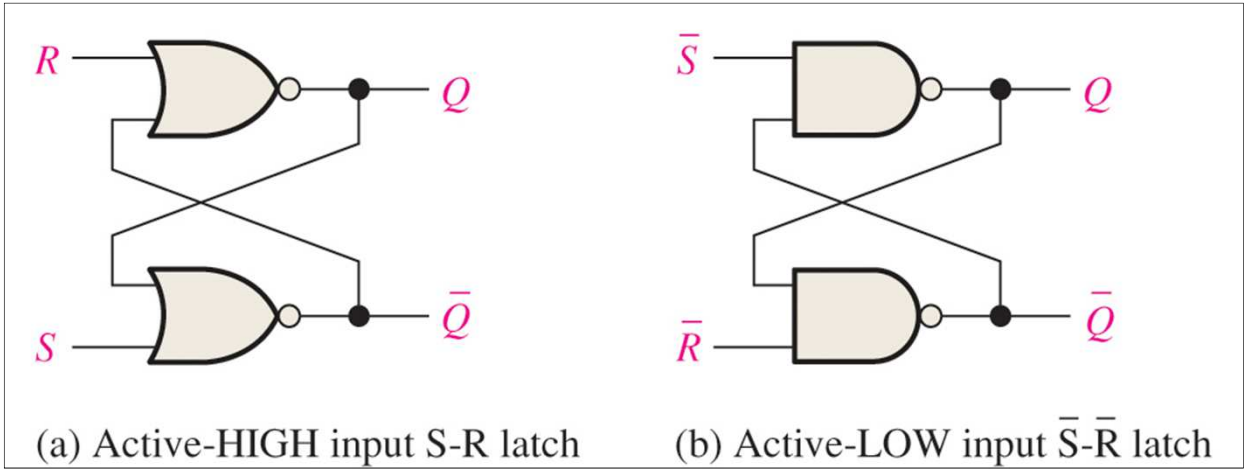
Realização da Função Memória

Se combinarmos duas portas lógicas idênticas com duas entradas (por exemplo portas NAND ou NOR), em que a saída de uma seja Q e a saída da outra seja a inversa da anterior, e fizermos com que a saída de uma porta vá alimentar uma das entrada outra, realiza-se a função memória. O elemento de memória obtido chama-se **trinco** (**latch** em inglês).

Se forem usadas portas NOR (NAND), a entrada livre da porta que produz Q chama-se RESET (SET), e a entrada da porta que produz a inversa de Q designa-se por SET (RESET).

Dependo do tipo de porta (NAND ou NOR) usada, as entradas do estão em repouso se forem ALTO (NAND) ou BAIXO (NOR). Sempre que se deseja alterar a saída Q (e portanto a sua inversa) **ativa-se** uma das entradas: passando-a de ALTO (BAIXO) para BAIXO(ALTO).

A figura da esquerda representa trinco/latch implementado com portas NOR (entradas ativadas com um ALTO); a figura da direita representa o trinco/latch implementado com portas NAND (entradas ativadas com um BAIXO).



(a) Active-HIGH input S-R latch (b) Active-LOW input $\overline{S}$-$\overline{R}$ latch

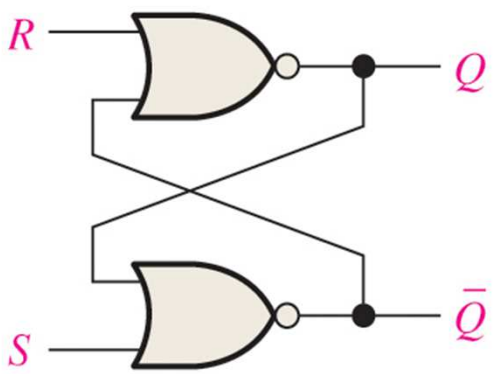
Dizer que as entradas são ativas a ALTO/BAIXO significa que é este valor lógico (ALTO/BAIXO) que impõe o resultado da porta NOR/NAND.

Trincos/Latches S-R

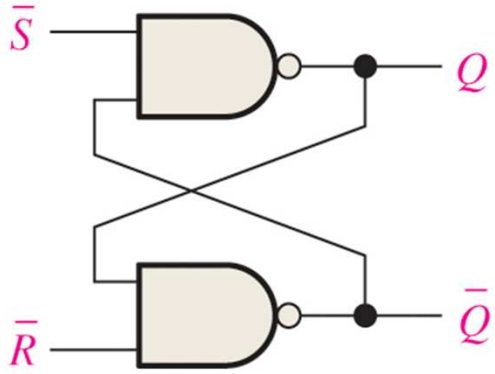
Um trinco/latch é um circuito lógico multivibrador com dois estados estáveis (biestável), que pode funcionar como elemento de armazenamento temporário (tem um bloco de memória). É a forma básica de memória: é capaz de **armazenar um bit**.

Um latch S-R (Set-Reset) com entrada ativa no nível ALTO é formado com duas portas NOR (latch NOR S-R), com acoplamento cruzado. O acoplamento cruzado produz uma realimentação regenerativa. Este latch responde a entradas ativas ALTO.

Um latch $\overline{S} - \overline{R}$ com entrada ativa no nível BAIXO é formado com duas portas NAND (latch NAND $\overline{S} - \overline{R}$), com acoplamento cruzado. Este latch responde a entradas ativas BAIXO.



(a) Active-HIGH input S-R latch



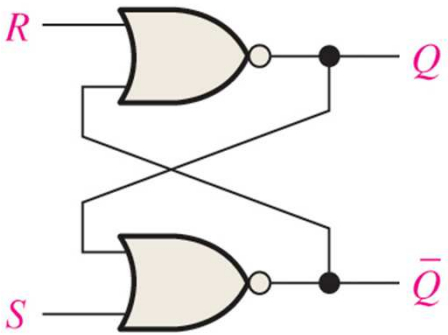
(b) Active-LOW input $\overline{S}$-$\overline{R}$ latch

Princípio de operação do trinco/latch S-R

Pode-se dizer que um latch é capaz de “trancar” o “0” e o “1”. Por isso o trinco/”latch” é o elemento básico de armazenamento de informação.

O latch S-R é implementado com portas NOR. O latch S-R com entrada ativa ALTO está num estado estável (“trancado”) quando as duas entradas são BAIXO. R representa o “Reset”: a saída é “0”. S representa o “Set”: a saída é “1”.

Assuma que o latch está inicialmente em BAIXO/RESET (Q=0) e que as entradas estão ambas inativas, isto é R=0 e S=0.
Para colocar o latch no estado “Q=1” (ALTO) aplica-se à entrada S um sinal ALTO. A saída inversa passa portanto para BAIXO.

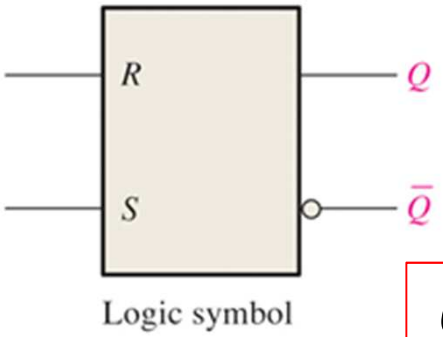


S	R	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	$\mathbf{Q_n}$	$\mathbf{\overline{Q_n}}$
0	1	0	1
1	0	1	0
1	1	NU	

NU: não usado

O estado 00 corresponde a **memória.**

Se a latch estiver inicialmente em ALTO/SET), Q=1, e as entradas ambas inativas, para a colocar em BAIXO (Q=0), R deve passar a ALTO, o que faz com que não-Q passe a ALTO.

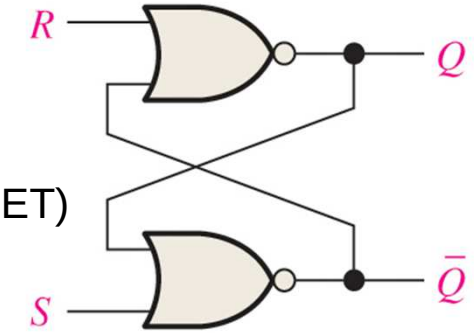


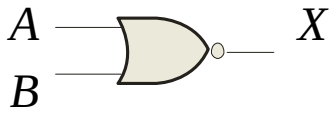
Logic symbol

$$Q_{n+1} = S + \bar{R} \cdot Q_n$$

Princípio de operação do trinco/latch S-R

***Presente* 1:** Q=0, R=0 e S=0
Ação: colocar S=1 (SET) e manter R=0
Resultado:
S=1 e Q=0 $\rightarrow \bar{Q} = 0$;
$\mathrm{R} = 0$ e $\bar{Q} = 0 \rightarrow$ Q= 1 - mudança de estado (SET)





T.V. NOR

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

***Presente* 2:** Q=1, R=0 e S=1
Ação: colocar S=0 e manter R=0
Resultado:
S=0 e Q=1 $\rightarrow \bar{Q} = 0$;
$\mathrm{R} = 0$ e $\bar{Q} = 0 \rightarrow$ Q= 1 - sem mudança de estado

***Presente* 3:** Q=1, R=0 e S=0
Ação: colocar R=1 (RESET) e manter S=0
Resultado:
R=1 e $\bar{Q} = 0 \rightarrow$ Q= 0;
$S = 0$ e $Q = 0 \rightarrow \bar{Q} = 1$ - mudança de estado (RESET)

S	R	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	$\mathbf{Q_n}$	$\overline{\mathbf{Q_n}}$ **(Memória)**
0	1	0	1
1	0	1	0
1	1	NU	

NU: não usado

***Presente* 4:** Q=0, R=1 e S=0
Ação: colocar S=1 e manter R=1
Resultado: S=1 e Q= 0 $\rightarrow \bar{Q} = 0$;
$\mathrm{R} = 1$ e $\bar{Q} = 0 \rightarrow$ Q = 0 (não é possível) – estado indeterminado

$$Q_{n+1} = S + \bar{R} \cdot Q_n$$

Princípio de operação do trinco/latch $\overline{S}$-$\overline{R}$

O latch $\overline{S} - \overline{R}$ é implementado com portas NAND.

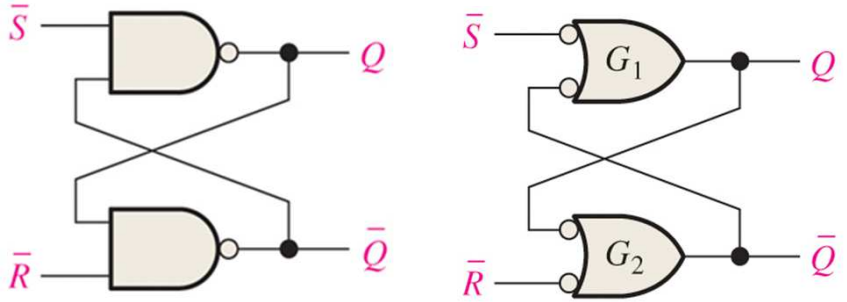
O latch $\overline{S} - \overline{R}$ com entrada ativa BAIXO está num estado estável ("trancado") quando as duas entradas são ALTO.

Considere o latch $\overline{S} - \overline{R}$: assuma que o latch está inicialmente RESET (Q=0) e que as entradas estão no nível inativo (1).

Para colocar o latch em SET (Q=1), um sinal BAIXO é aplicado à entrada $\overline{S}$, mantendo $\overline{R}$ em ALTO.

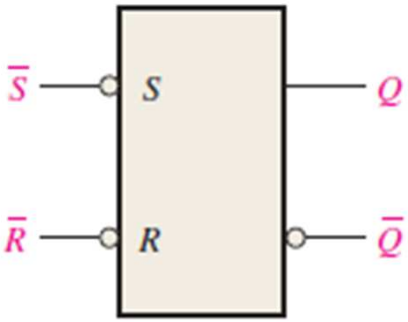
Para fazer o RESET (Q=0), aplica-se à entrada $\overline{R}$ um sinal BAIXO, mantendo $\overline{S}$ em ALTO.

Se, por outro lado, assumirmos que as entradas e a saída estão em ALTO, Q=1 (SET), tem-se $\overline{Q} = 0$. Se agora $\overline{R}$ passar a BAIXO, $\overline{Q}$ fica ALTO, e Q passa a BAIXO (RESET). Para que passe a SET $\overline{S}$ tem de comutar para o valor BAIXO.



Este circuitos são equivalentes

$\overline{S}$	$\overline{R}$	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	NU	
0	1	1	0
1	0	0	1
1	1	$\mathbf{Q_n}$	$\mathbf{\overline{Q_n}}$
		(Memoria)	



$Q_{n+1} =$

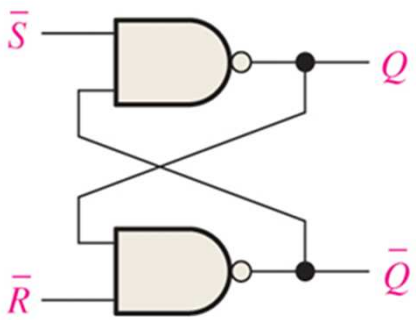
Princípio de operação do trinco/latch $\bar{S}$-$\bar{R}$

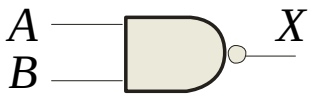
***Presente* 1:** Q=0, $\bar{S} = 1$ e $\bar{R}$=1
Ação: colocar $\bar{S} = 0$ (SET) e manter $\bar{R}$=1
Resultado:
$\bar{S} = 0$ e $\bar{Q} = 1 \rightarrow$ Q= 1
$\bar{R}$=1 e Q=1 $\rightarrow \bar{Q} = 0$ - mudança de estado

Presente 2: Q=1, $\bar{S} = 1$ e $\bar{R}$=1
Ação: colocar $\bar{S} = 0$ e manter $\bar{R}$=1
Resultado:
$\bar{R}$=1 e Q=1 $\rightarrow \bar{Q} = 0$;
$\bar{S} = 0$ e $\bar{Q} = 0 \rightarrow$ Q= 1 - sem mudança de estado

Presente 3: Q=1, $\bar{S} = 1$ e $\bar{R}$=1
Ação: colocar $\bar{R}$=0 (RESET) e manter $\bar{S} = 1$
Resultado:
$\bar{R}$=0 e Q= 1 $\rightarrow \bar{Q} = 1$;
$\bar{S} = 1$ e $\bar{Q} = 1 \rightarrow$ Q= 0 - mudança de estado

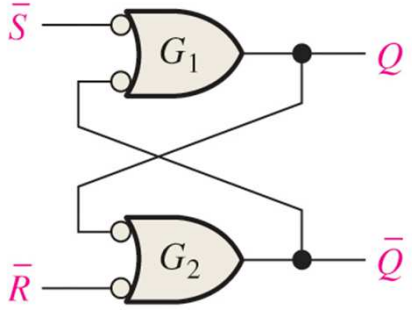
Presente 4: Q=0, $\bar{R}$=1 e $\bar{S} = 1$
Ação: colocar $\bar{R} = 0$ e manter $\bar{S}$
Resultado $\bar{R} = 0$ e $Q = 0 \rightarrow \bar{Q} = 1$
$\bar{S}$=1 e $\overline{Q} = 1 \rightarrow$ Q = 0 - mantém estado





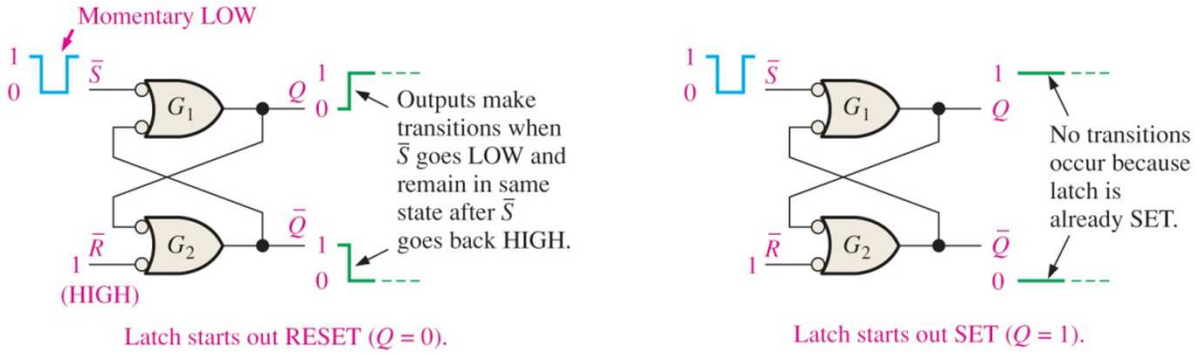
T.V. NAND

Inputs		Output
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

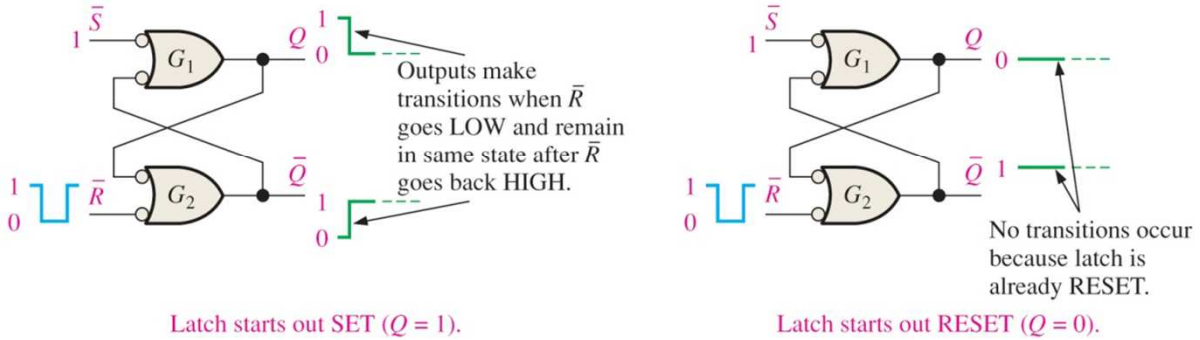


$\bar{S}$	$\bar{R}$	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	NU	
0	1	1	0
1	0	0	1
1	1	Q_n	$\overline{Q_n}$
		(Memoria)	

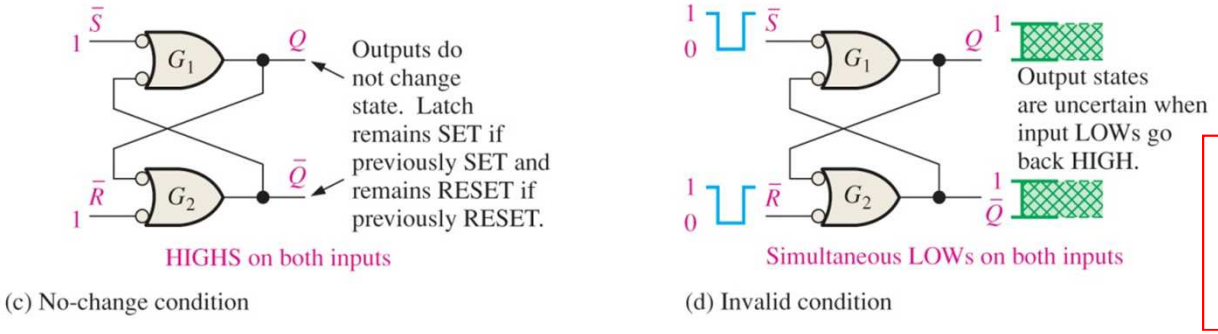
Resumo da dinâmica do trinco/latch $\bar{S}$-$\bar{R}$



(a) Two possibilities for the SET operation



(b) Two possibilities for the RESET operation



(c) No-change condition

(d) Invalid condition

Never apply an active set and reset at the same time (invalid).

Exercício

If the $\overline{S}$ and $\overline{R}$ waveforms in Figure 7–5(a) are applied to the inputs of the latch in Figure 7–4(b), determine the waveform that will be observed on the Q output. Assume that Q is initially LOW.

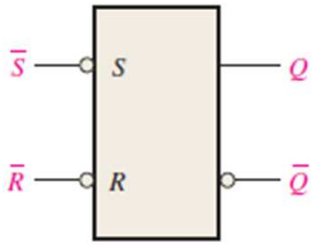


FIGURE 7–4

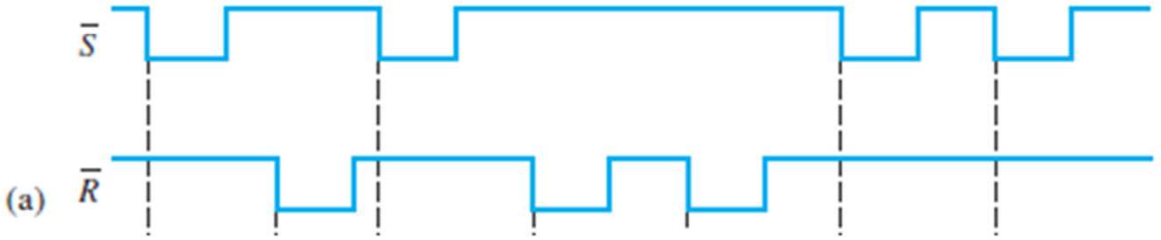
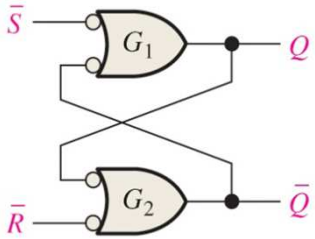


FIGURE 7–5

Solution



T.V. NAND

Inputs		Output
A	*B*	*X*
0	0	1
0	1	1
1	0	1
1	1	0

$\overline{S}$	$\overline{R}$	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	NU	
0	1	1	0
1	0	0	1
1	1	Q_n (Memoria)	$\overline{Q_n}$

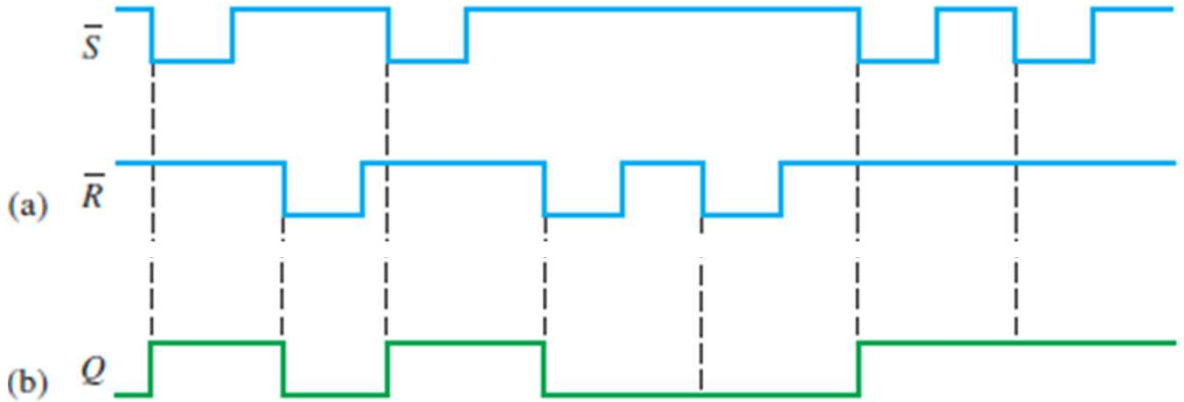
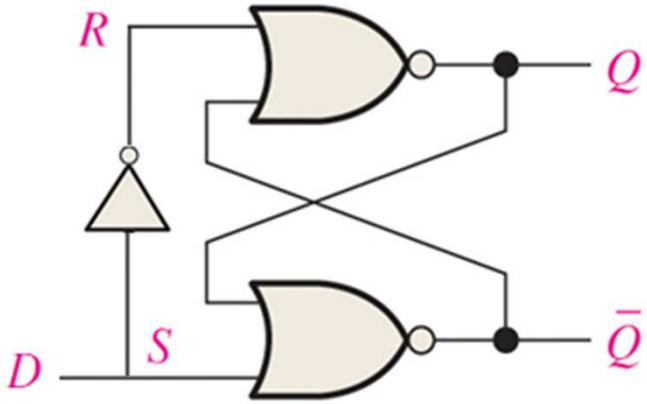


FIGURE 7–5

Bistável Tipo D

Se impusermos no latch S-R que **R e S são sempre diferentes** obtém-se o **latch D:**



S	R	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	**Q_n**	**$\overline{Q_n}$ (Memória)**
0	**1**	**0**	**1**
1	**0**	**1**	**0**
1	1	NU	

NU: não usado /

D	$\overline{D}$	Q_{n+1}	$\overline{Q_{n+1}}$
0	**1**	**0**	**1**
1	**0**	**1**	**0**

$$Q_{n+1} = D$$

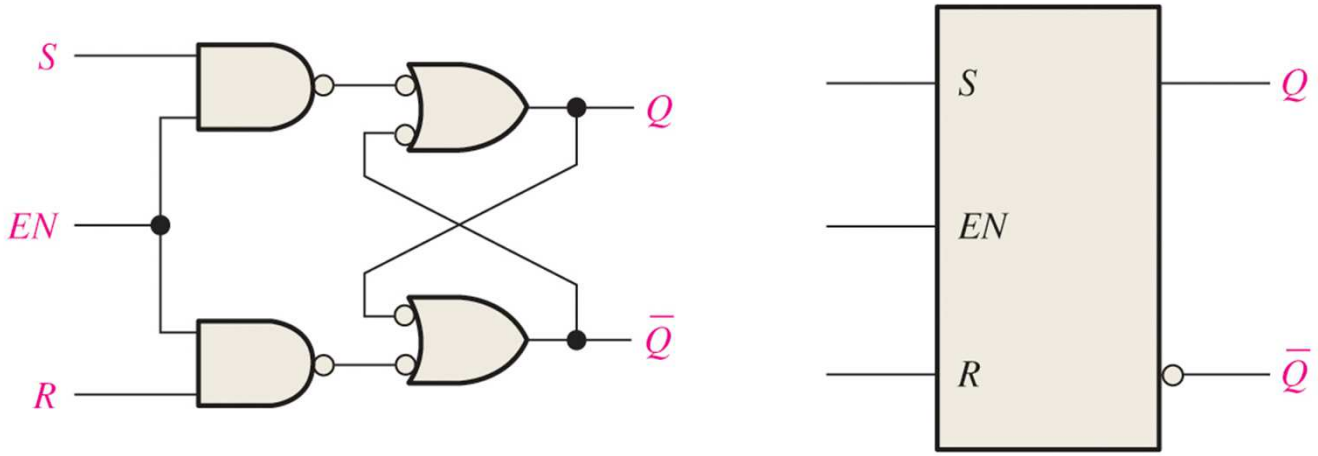
Latch S-R Controlado/Sincronizado

Gated S-R Latch

A gated latch is a variation on the basic latch.

The gated latch has an additional input, called enable (*EN*) that must be HIGH in order for the latch to respond to the *S* and *R* inputs.

EN/G: entrada de habilitação/ativação



(a) Logic diagram (b) Logic symbol

As entradas S e R **controlam** o estado para o qual o latch irá comutar **quando** um nível ALTO é aplicado à entrada EN.
O latch não mudará de estado até que EN seja nível ALTO. Neste circuito o estado inválido ocorre quando S e R forem simultaneamente nível ALTO.

Exercício

Determine the *Q* output waveform if the inputs shown in Figure 7–9(a) are applied to a gated S-R latch that is initially RESET.

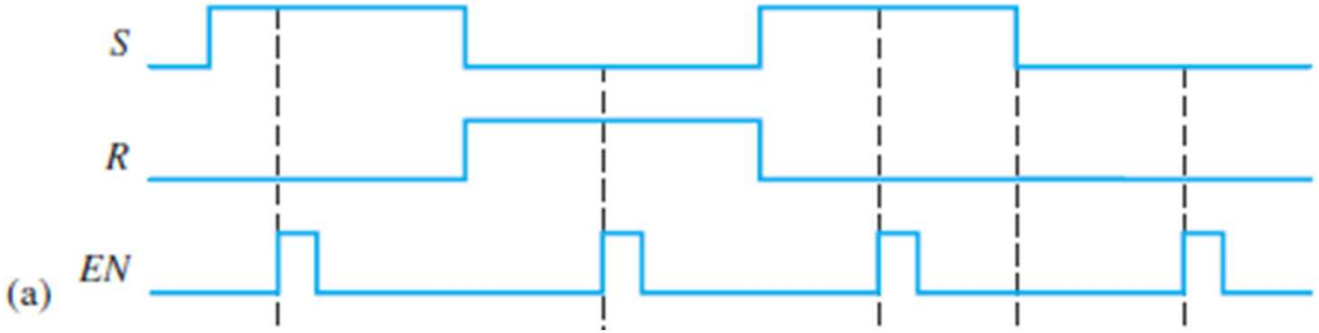


FIGURE 7–9

Solution

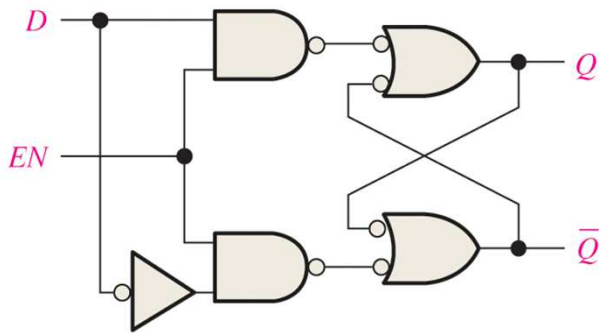
The *Q* waveform is shown in Figure 7–9(b). When *S* is HIGH and *R* is LOW, a HIGH on the *EN* input sets the latch. When *S* is LOW and *R* is HIGH, a HIGH on the *EN* input resets the latch. When both *S* and *R* are LOW, the *Q* output does not change from its present state.

FIGURE 7–9

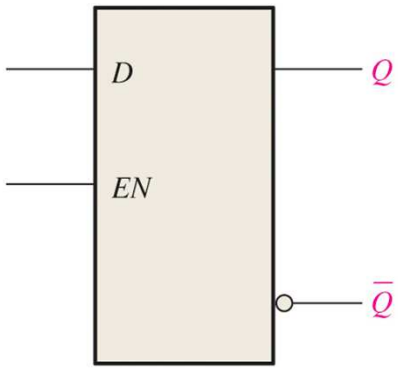
Trinco D Controlado / Latch D Controlado

Gated D Latch

The gated *D* latch is an variation of the *S-R* latch that combines the *S* and *R* inputs into a single *D* input.



(a) Logic diagram



(b) Logic symbol

D	$\overline{D}$	Q_{n+1}	$\overline{Q_{n+1}}$
0	**1**	**0**	**1**
1	**0**	**1**	**0**
1	1	NU	

NU: não usado

$$Q_{n+1} = D \cdot EN + Q_n \cdot \overline{EN}$$

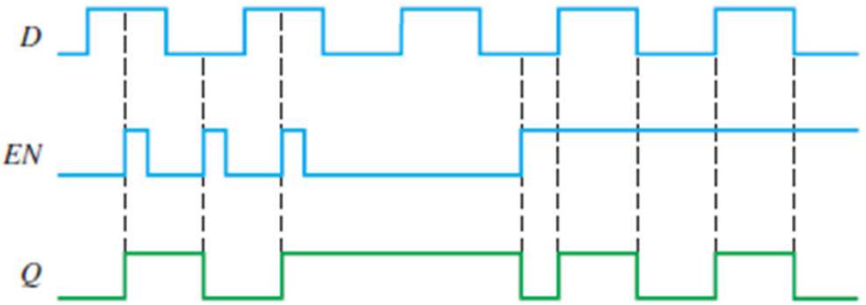
A simple rule for the *D* latch is: *Q* follows *D* when the EN is active.

Exercício

Determine the *Q* output waveform if the inputs shown in Figure 7–11(a) are applied to a gated D latch, which is initially RESET.

Solution

The *Q* waveform is shown in Figure 7–11(b).
When *D* is HIGH and *EN* is HIGH, *Q* goes HIGH.
When *D* is LOW and *EN* is HIGH, *Q* goes LOW.
When *EN* is LOW, the state of the latch is not affected by the *D* input.

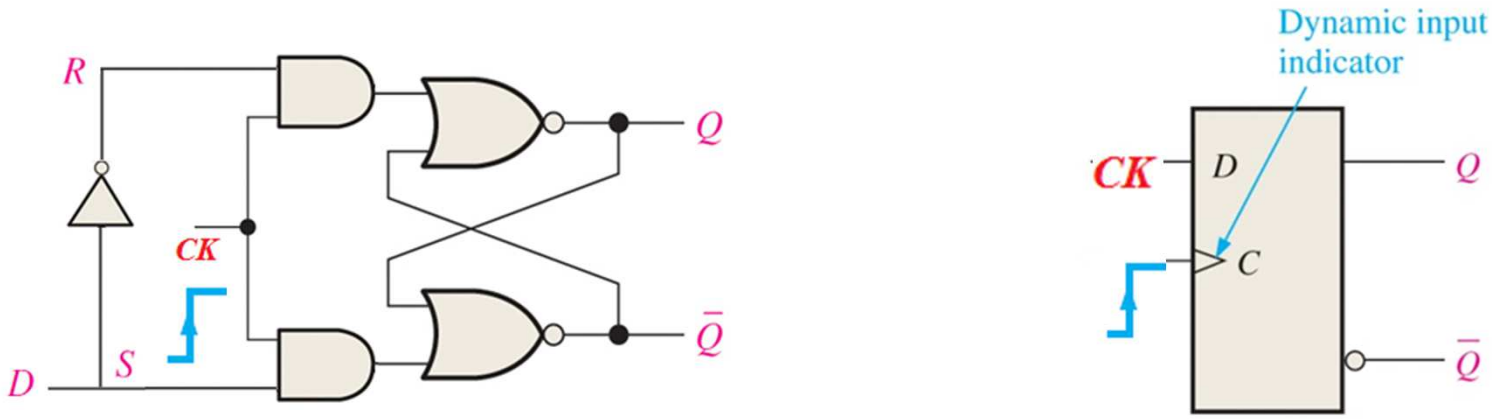


Latch D ativado por flanco - Flip-flop/Báscula D

Edge-Triggered Flip-flops

Quando um biestável é ativado para transição de estado (ou ao flanco/borda) passa a designar-se **por báscula (ou flip-flop**). Assim, um flip-flop difere de um latch na maneiro como muda de estado. Isto é, um flip-flop é um circuito controlado pelo sinal de relógio, em que **a mudança de estado** (isto é, **a entrada de um novo bit**) é determinada pelas transições/ flanco/borda ascendentes ou descentes do sinal de relógio**. A transição ascendente também se designa positiva (transição de 0 para 1): a transição descendente é também designada negativa (transição de 1 de para 0).**

Um flip-flop (báscula) é sensível à transição de estado (ou ao flanco/à borda) de um sinal de controlo (o sinal de relógio ou clock, CK). Existem básculas ativas ao flanco/borda ascendente (transições BAIXO (0) → ALTO (1) ou ao flanco/borda descendente (transição 1 para 0).



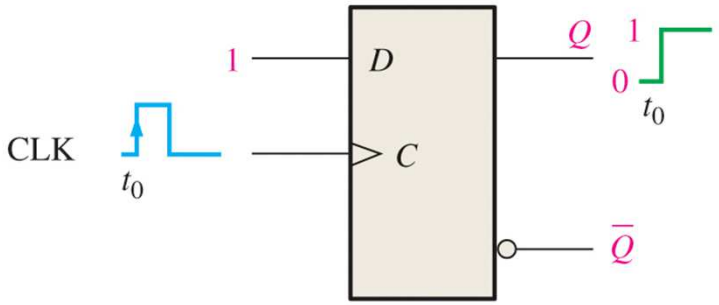
Circuito e símbolo de um flip-flop do tipo D ativo ao flanco ascendente (borda de subida – borda positiva) do pulso de relógio ou entrada de disparo (clock, C ou CK).

Flip-flops/Básculas do tipo D

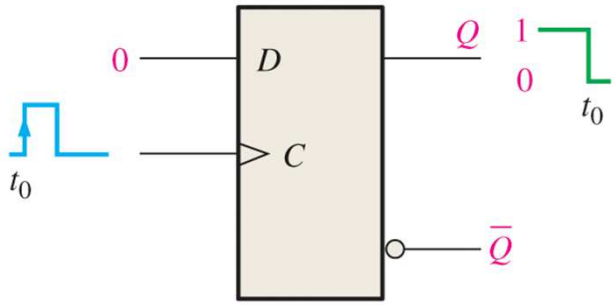
Edge-Triggered D Flip-flops

O flip-flop D é usado quando se pretende armazenar um único bit (1 ou 0). Conforme vimos, a adição de um inversor a um flip-flop S-R cria um flip-flop D básico, que tem apenas uma entrada D, além do clock. No estado SET o flip-flop armazena um nível 1 e no estado RESET armazena um nível 0.

O bit de dados da entrada D é transferido para a saída Q em resposta à flanco/borda do sinal de relógio.



(a) $D = 1$; flip-flop SETS on positive clock edge. (If already SET, it remains SET.)



(b) $D = 0$; flip-flop RESETS on positive clock edge. (If already RESET, it remains RESET.)

Truth table for a positive edge-triggered D flip-flop.

Inputs		Outputs		
D	CLK	Q	$\overline{Q}$	Comments
0	↑	0	1	RESET
1	↑	1	0	SET

↑ = clock transition LOW to HIGH

Uma vez “ativado”, a saída Q iguala o valor da entrada D até que um novo sinal de ativação (flanco do sinal de relógio) chegue.

Exercício

Determine the Q and $\overline{Q}$ output waveforms of the flip-flop in Figure 7–15 for the D and CLK inputs in Figure 7–16(a). Assume that the positive edge-triggered flip-flop is initially RESET.

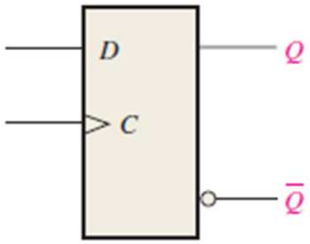


FIGURE 7–15

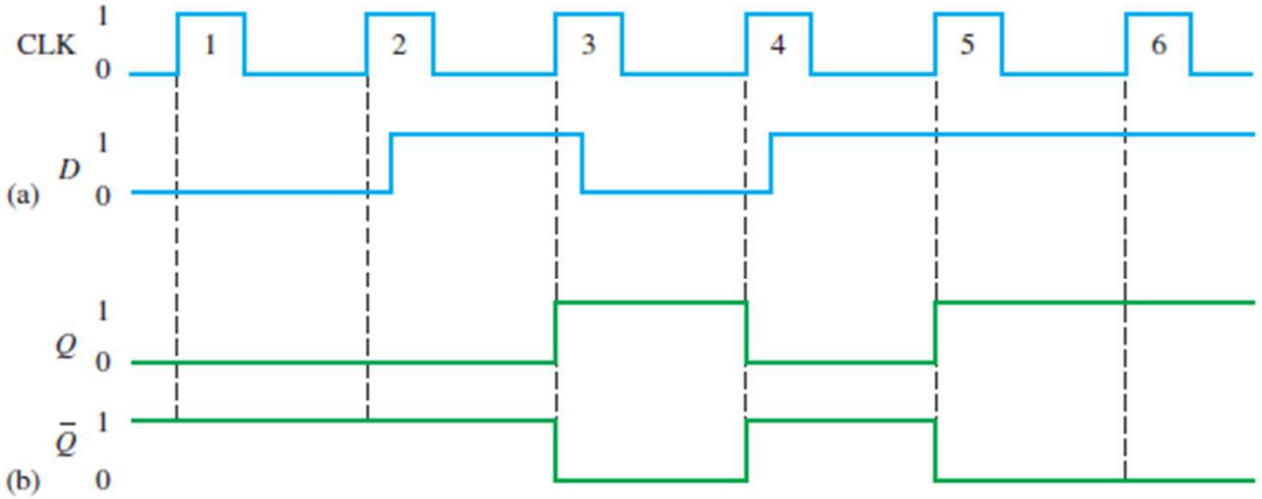


FIGURE 7–16

Solution

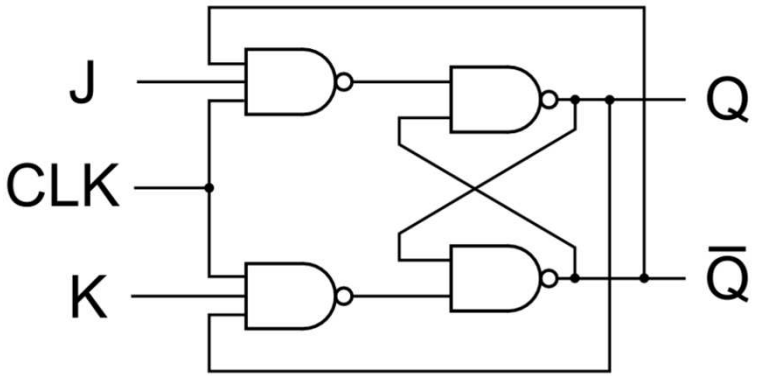
1. At clock pulse 1, D is LOW, so Q remains LOW (RESET).
2. At clock pulse 2, D is LOW, so Q remains LOW (RESET).
3. At clock pulse 3, D is HIGH, so Q goes HIGH (SET).
4. At clock pulse 4, D is LOW, so Q goes LOW (RESET).
5. At clock pulse 5, D is HIGH, so Q goes HIGH (SET).
6. At clock pulse 6, D is HIGH, so Q remains HIGH (SET).

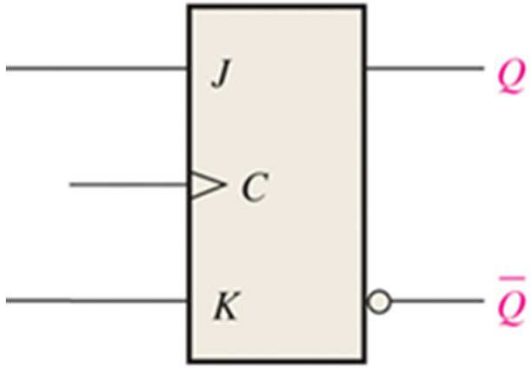
Once Q is determined, $\overline{Q}$ is easily found since it is simply the complement of Q. The resulting waveforms for Q and $\overline{Q}$ are shown in Figure 7–16(b) for the input waveforms in part (a).

Báscula JK / Flip-flop J K

No flip-flop JK a combinação RS="11" passa a ser possível porque flip-flop tira partido da lógica combinacional. O funcionamento do flip-flop J-K de disparo de borda/flanco é idêntico ao do flip-flop S-R nas condições de SET, RESET e repouso (memória). Porém o flip-flop J-K aprimora o funcionamento do flip-flop R-S interpretando a condição S = R = 1 como um comando de inversão, e deixa de haver o estado inválido como do flip-flop S-R. Neste caso, os valores das entradas J e K determinam, em conjunto com o disparo de flanco do pulso de relógio, o estado de saída.

Especificamente, a combinação J = 1, K = 0 corresponde ao “comando” para ativar (SET) a saída do flip-flop; a combinação J = 0, K = 1 ao “comando” para desativar (RESET) a saída do flip-flop; e a combinação J = K = 1 provoca a inversão das saídas do flip-flop, isto é, troca o sinal de saída Q pelo seu complemento.





$$Q_{n+1} = J \cdot \overline{Q_n} + \overline{K} \cdot Q_n$$

Flip-flop J-K

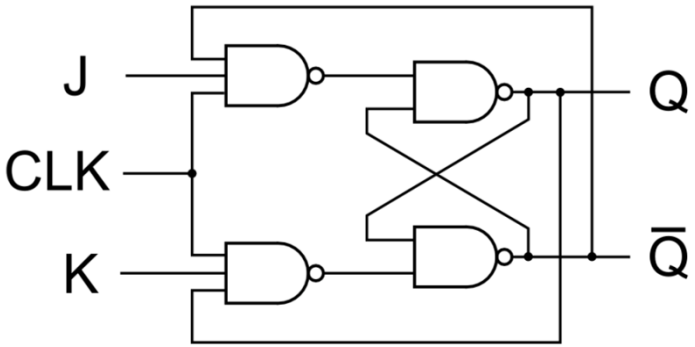
Consideremos o caso em que o flip-flop está resetado (Q=0), com a entrada J ALTA e a entrada K BAIXA.

Quando um pulso de relógio "chega" à entrada CLK, a borda de subida (ascendente) ativa o flip-flop, que passa para o estado SET (Q=1), uma vez que J é ALTO e Não-Q era ALTO até à chegada do pulso.

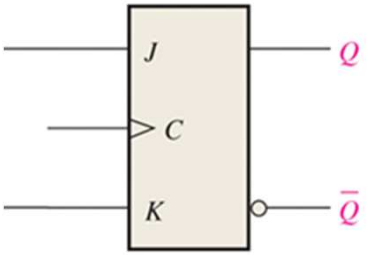
Se agora fizermos J BAIXO e K ALTO, a próxima borda ascendente do pulso a chegar faz com que o flip-flop mude para o estado RESET (K é ALTO e Q é ALTO até à chegada do pulso; a chegada do pulso comuta Não-Q para ALTO).

Se ambas as entradas forem BAIXO não há mudança de estado.

Se forem ambas ALTO há mudança (comutação/alteração) de estado (para o estado oposto) a cada chegada do pulso ascendente do sinal de relógio.



K	J	CK	Q_{n+1}	$\overline{Q_{n+1}}$	
0	0	↑	Q_n	$\overline{Q_n}$	Repouso
0	1	↑	1	0	RESET
1	0	↑	0	1	SET
1	1	↑	$\overline{Q_n}$	Q_n	Toggle*



***alterna**

$$Q_{n+1} = J \cdot \overline{Q_n} + \overline{K} \cdot Q_n$$

Flip-flip Tipo T (Toggle/Comutação)

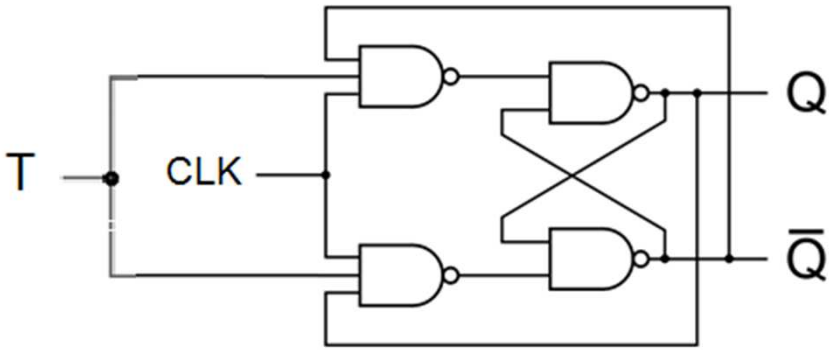
T (Toggle) Flip-flops

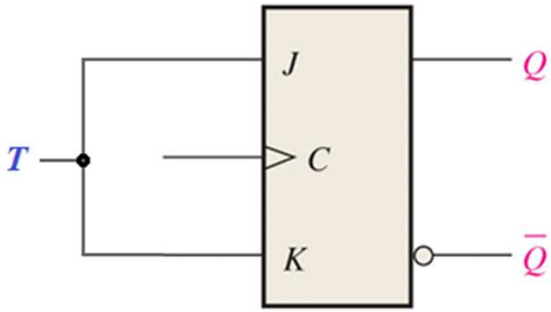
Como vimos, no biestável JK a combinação RS="11" é possível se tirarmos partido da lógica combinacional. Neste caso os valores das entradas J e K determinam univocamente o seu estado de saída, em conjunto com o pulso de relógio.

Assim, se ambas as entradas forem BAIXO não há mudança de estado. Se foram ambas forem ALTO há mudança (**comutação/alterna**) de estado (para o estado oposto) a cada transição de relógio sucessiva.

Um flip-flop J-K conetado para o modo ***toggle*** (**alterna**) é denominado **flip-flop tipo T**.

A saída muda de estado sempre que o biestável é habilitado pelo sinal de relógio.

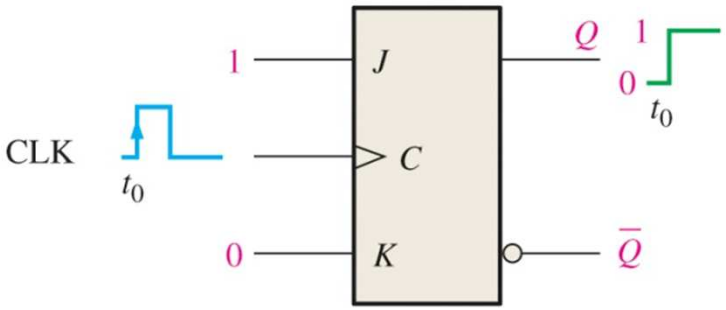




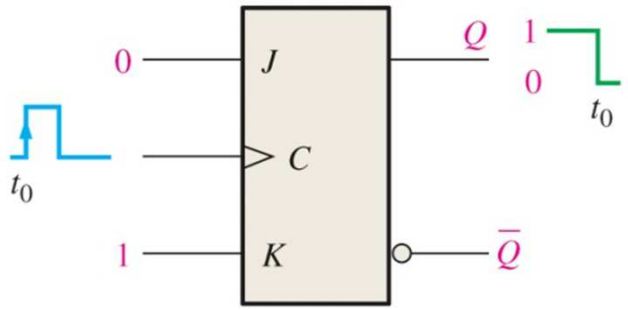
$$Q_{n+1} = T \cdot \overline{Q_n} + \overline{T} \cdot Q_n$$

Combinações da entradas J e K no flip-flop J-K
J-K Flip-flops

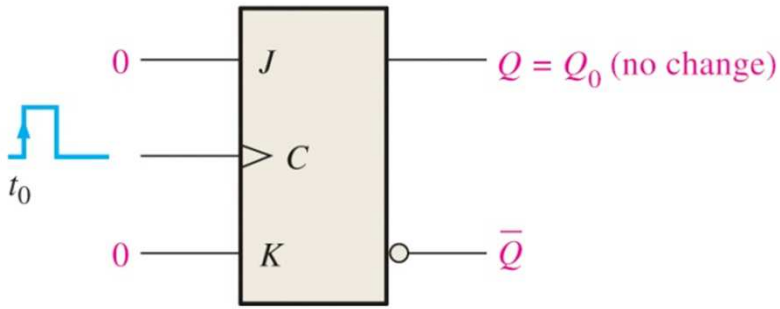
The values at the J and K inputs to a J-K flip-flop determine its output state. The results of the four possible input combinations of J and K are shown.



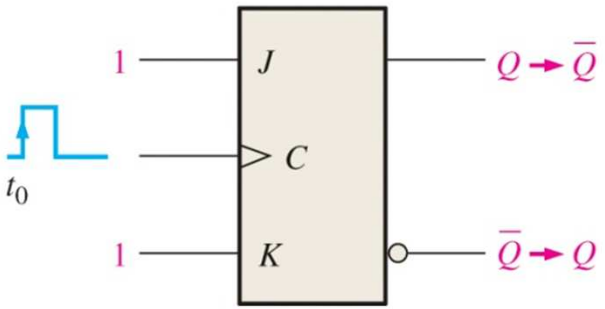
(a) $J = 1$, $K = 0$; flip-flop SETS on positive clock edge. (If already SET, it remains SET.)



(b) $J = 0$, $K = 1$; flip-flop RESETS on positive clock edge. (If already RESET, it remains RESET.)



(c) $J = 0$, $K = 0$; flip-flop does not change. (If SET, it remains SET; if RESET, it remains RESET.)



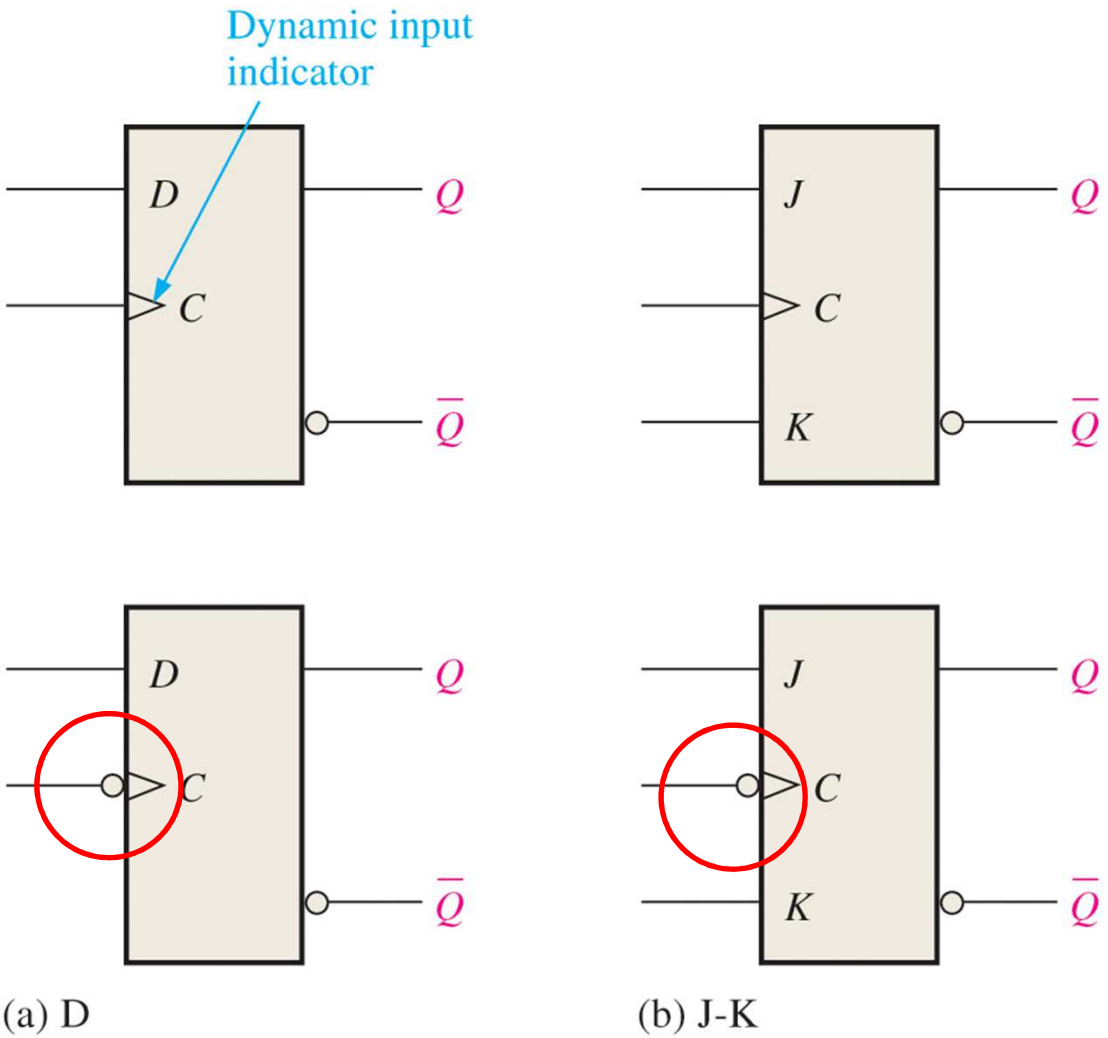
(d) $J = 1$, $K = 1$; flip-flop changes on each clock pulse (toggle).

Flip-flops/Básculas ativados por flanco positivo/negativo

Edge-Triggered Flip-flops

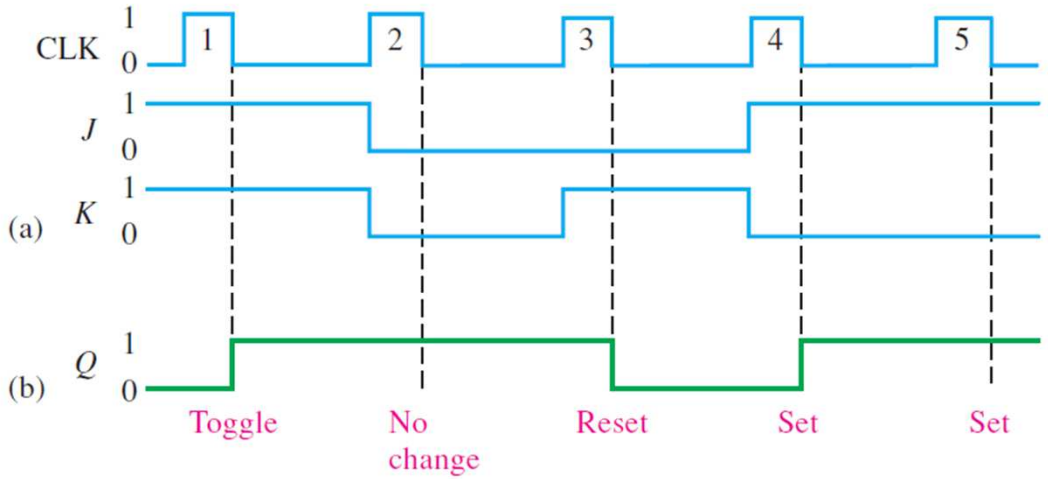
The output from an edge-triggered flip-flop changes on the **positive-going** or **negative-going** edge of its clock signal (C).

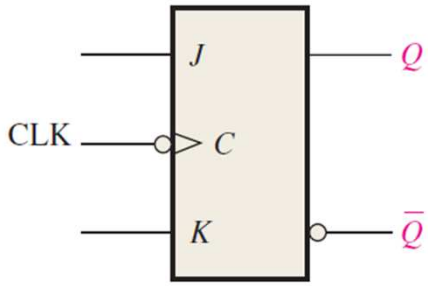
A **bubble on the clock (C)** input indicates that it is a **negative-edge triggered** flip-flop.



Exercício

The waveforms in Figure 7–18(a) are applied to the J, K, and clock inputs as indicated. Determine the Q output, assuming that the flip-flop is initially RESET.





$$Q_{n+1} = J \cdot \overline{Q_n} + \overline{K} \cdot Q_n$$

FIGURE 7–18

Solution

Since this is a negative edge-triggered flip-flop, as indicated by the "bubble" at the clock input, the Q output will change only on the negative-going edge of the clock pulse.

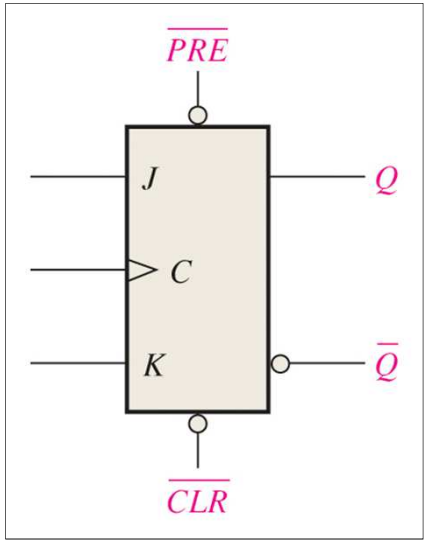
1. At the first clock pulse, both J and K are HIGH; and because this is a toggle condition, Q goes HIGH.
2. At clock pulse 2, a no-change condition exists on the inputs, keeping Q at a HIGH level.
3. When clock pulse 3 occurs, J is LOW and K is HIGH, resulting in a RESET condition; Q goes LOW.
4. At clock pulse 4, J is HIGH and K is LOW, resulting in a SET condition; Q goes HIGH.
5. A SET condition still exists on J and K when clock pulse 5 occurs, so Q will remain HIGH.

The resulting Q waveform is indicated in Figure 7–18(b).

Entradas Assíncronas de “Preset” e “Clear”
Flip-flop Asynchronous Inputs

As entradas dos flip-flops até agora consideradas, as entradas S-R, D e J-K, são denominadas entradas **síncronas** porque os dados nessas entradas são transferidos para a saída do flip-flop apenas na borda de disparo do pulso de clock; ou seja, **os dados são transferidos de forma sincronizada com o clock**.

A maioria dos flip-flops em circuitos integrados tem também **entradas assíncronas**, que afetam o estado do flip-flop independente do clock, normalmente denominadas **preset** (PRE) e **clear** (CLR), ou “**seta direto”** (S_D) e “reseta direto” (R_D) por alguns fabricantes.



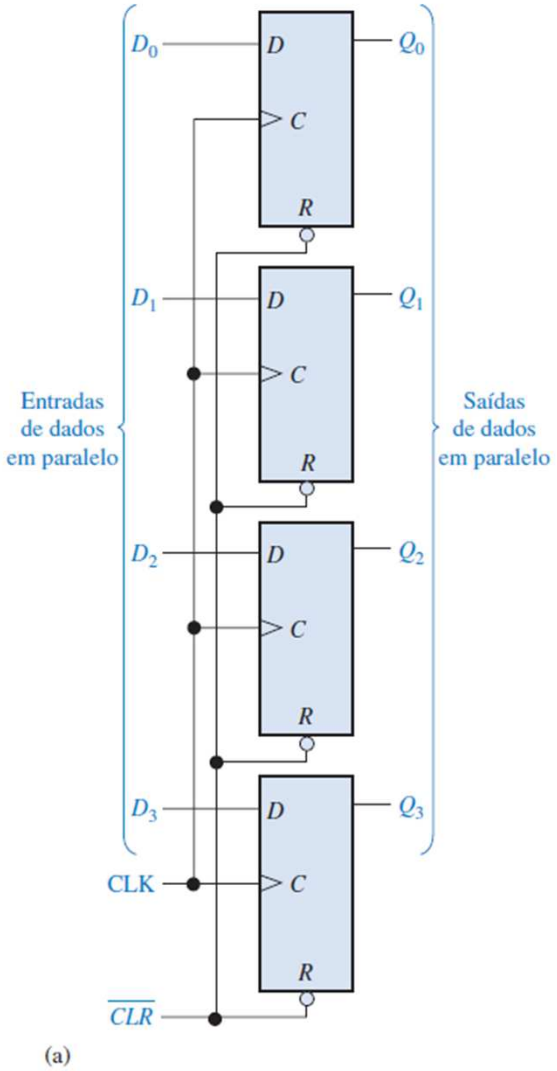
Um nível ativo na entrada **preset irá *setar*** o flip-flop e um nível ativo na **entrada clear irá *resetar* o flip-flop**.

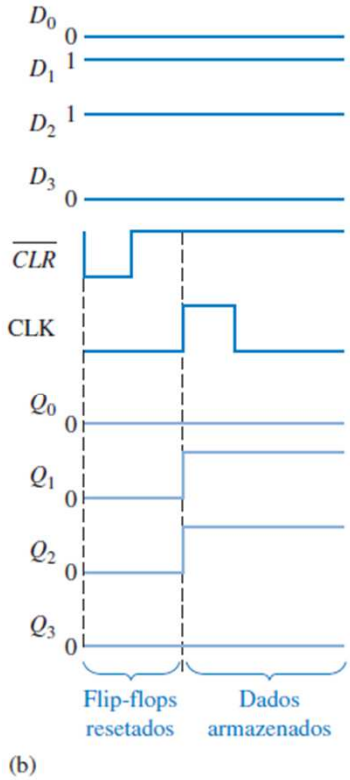
O símbolo lógico para um flip-flop J-K com entradas *preset* e *clear* é mostrado na figura ao lado, para o caso de entradas ativas em nível BAIXO, conforme indicado pelos pequenos círculos.

As entradas de *preset* (PRE) e *clear* (CLR) têm que ser mantidas em nível ALTO para a operação síncrona.

Aplicações de flip-flops

Armazenamento de Dados em Paralelo





Uma necessidade comum em sistemas digitais é armazenar diversos bits de dados em linhas em paralelo simultaneamente. Esta operação é ilustrada na Figura (a) usando quatro flip-flops. Cada uma das quatro linhas paralelas de dados é conectada na entrada D do flip-flop. As entradas de clock dos flip-flops são conectadas juntas, de forma que os flip-flops são disparados pelo mesmo pulso de clock.

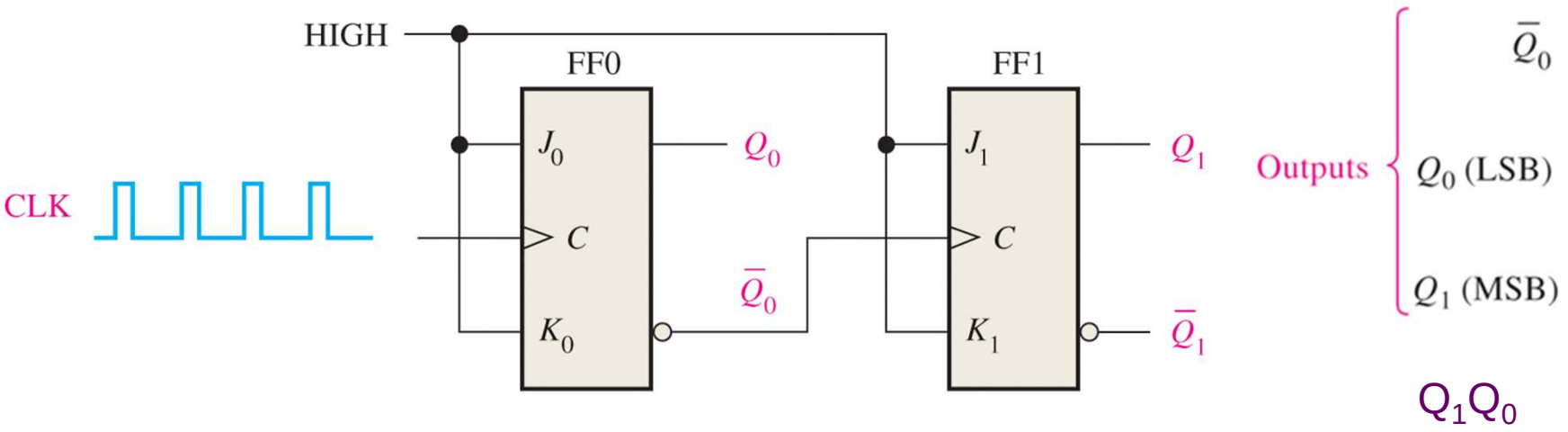
Neste exemplo são usados flip-flops disparados por borda positiva, assim os dados nas entradas D são armazenados simultaneamente pelos flip-flops na borda positiva do clock, conforme indicado no diagrama de temporização visto na Figura (b). Além disso, as entradas assíncronas de reset (R) são conectadas numa linha comum, a qual reseta todos os flip-flops.

Contador assíncrono de 2 bits
A 2-bit Asynchronous Counter

Um contador é assíncrono se os respetivos flip-flops não forem ativadas em simultâneo, isto é, não forem ativadas pelo mesmo sinal. Um contador assíncrono pode funcionar também como divisor de frequência.

Num contador assíncrono, o relógio é aplicado apenas ao primeiro "andar". Os andares subsequentes usando os sinais de relógio dos andares anteriores.

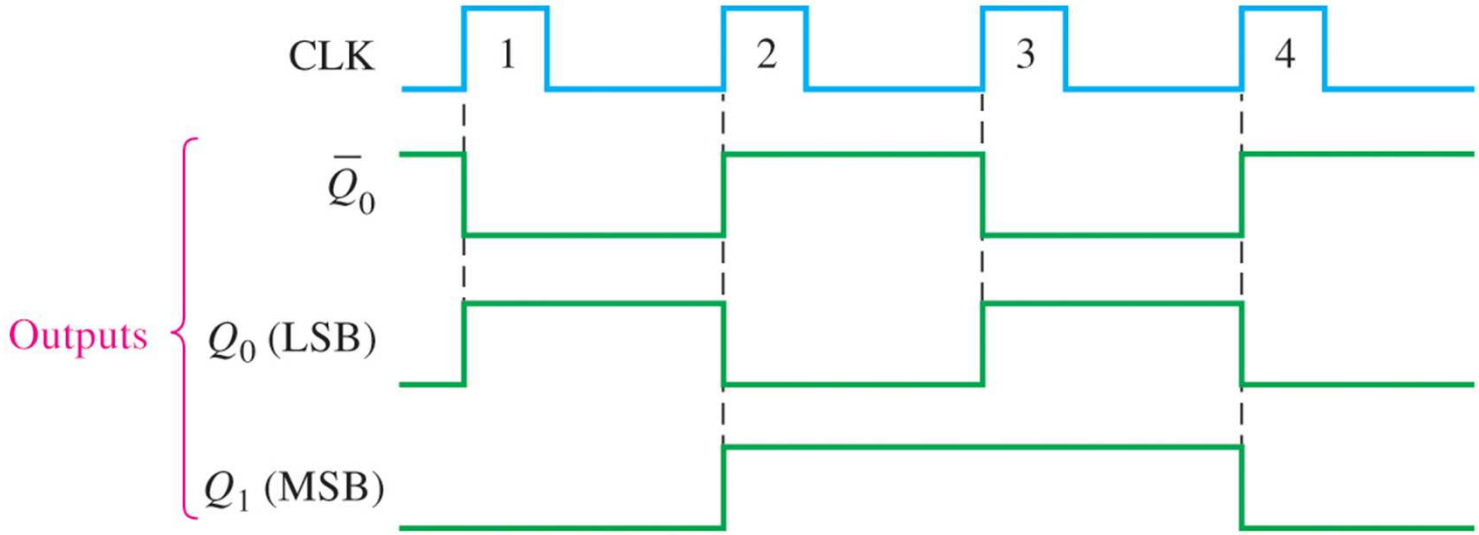
O esquema abaixo representa um contador assíncrono de 2 bit típico implementado com flip-flops T ($Q_{n+1} = T \cdot \overline{Q_n} + \overline{T} \cdot Q_n$).

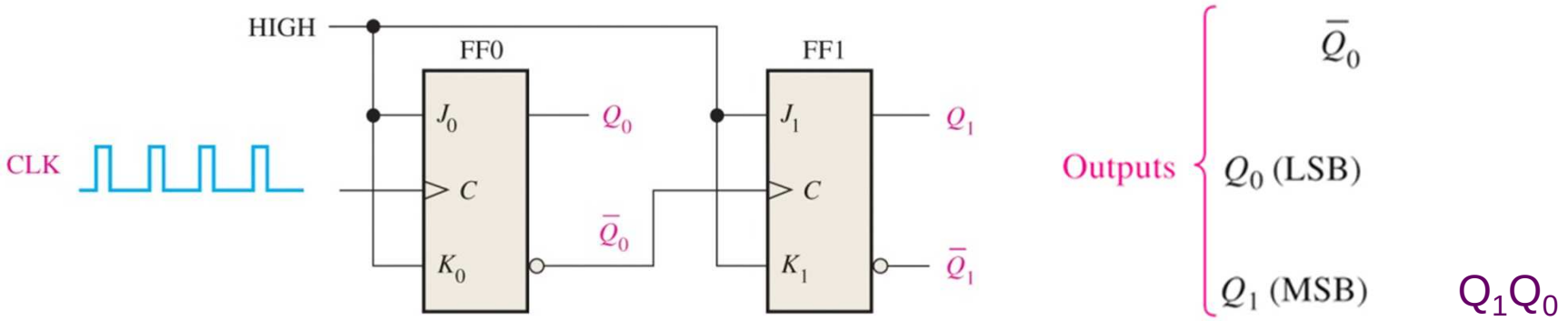


Contador assíncrono de 2 bits

A 2-bit Asynchronous Counter Timing Diagram

Diagrama de tempo do contador assíncrono de 2 bit implementado com flip-flops T.



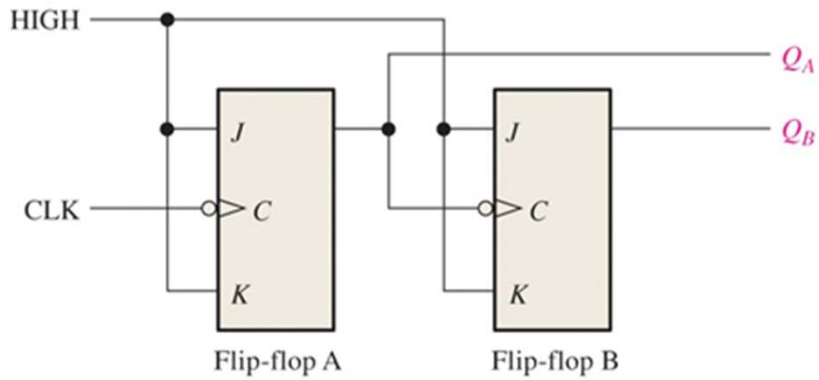


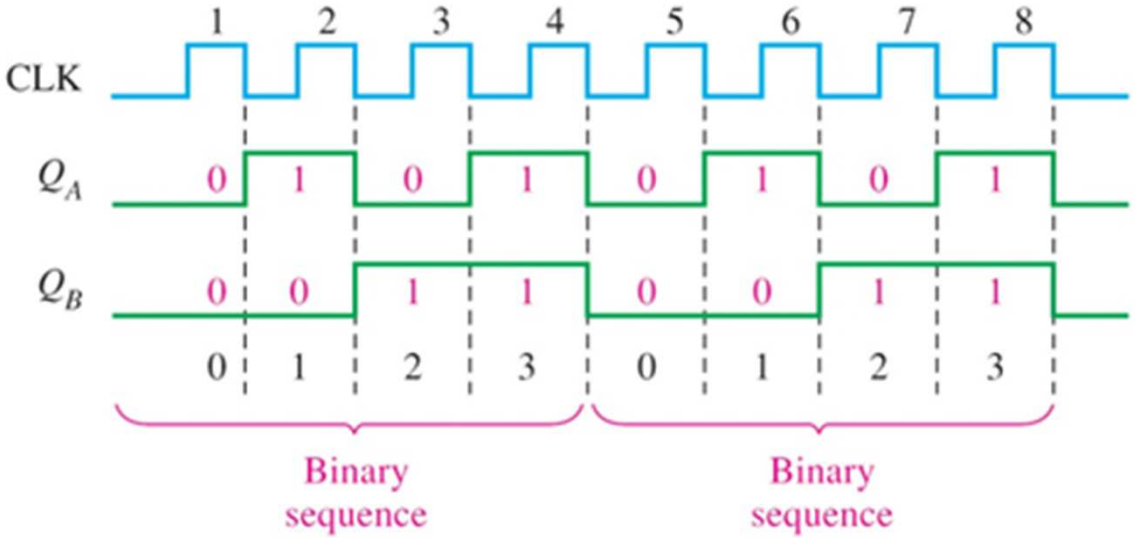
Q_1Q_0

Análise do Contador Assíncrono de 2 bits
Flip-Flop Counters

No exemplo que se segue os flip-flops são do tipo T ($Q_{n+1} = T \cdot \overline{Q_n} + \bar{T} \cdot Q_n$) disparados pela borda negativa. Os dois flip-flops estão inicialmente resetados.

O flip-flop A comuta na transição negativa de cada pulso de clock. A saída Q_A do flip-flop A é o clock do flip-flop B, assim cada vez que Q_A faz uma transição de nível ALTO para nível BAIXO, o flip-flop B muda de estado (toggle). As formas de onda resultante de Q_A e Q_B são mostradas na figura.

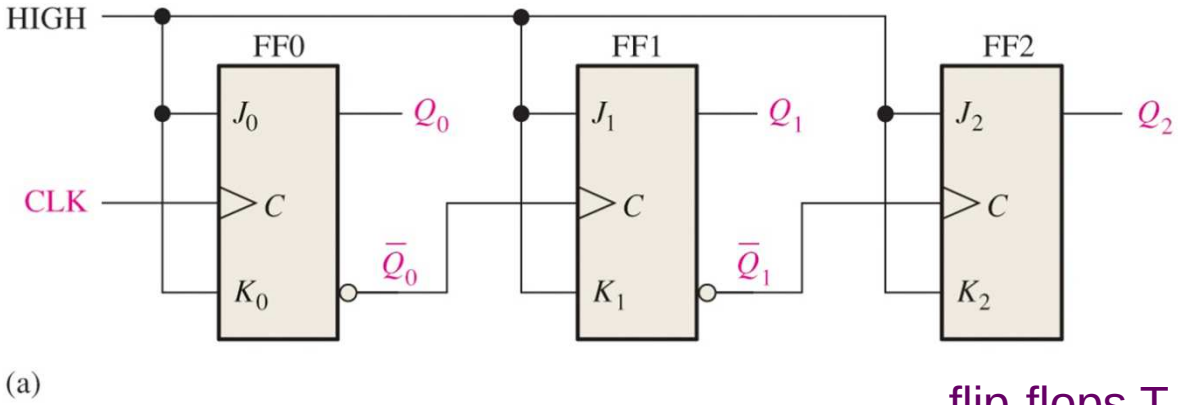




As saídas Q_A e Q_B indicam o número de pulsos de relógio recebidos:

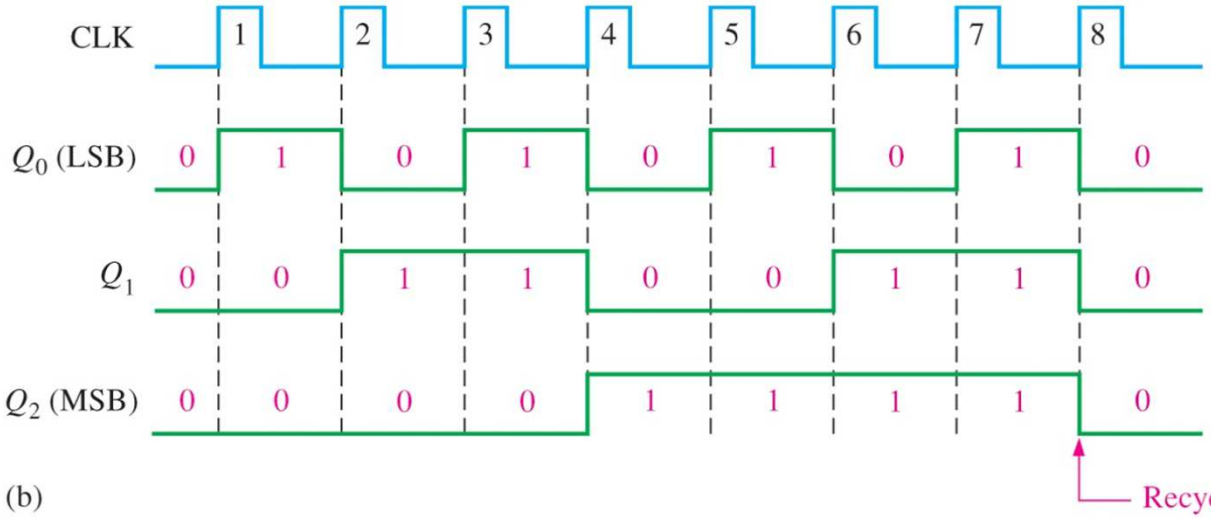
$Q_B Q_A$.

Contador assíncrono de 3 bits
A 3-bit Asynchronous Counter



(a)

flip-flops T ($Q_{n+1} = T \cdot \overline{Q_n} + \overline{T} \cdot Q_n$).



(b)

As saídas Q_0, Q_1 e Q_2 indicam o número de pulsos de relógio recebidos:

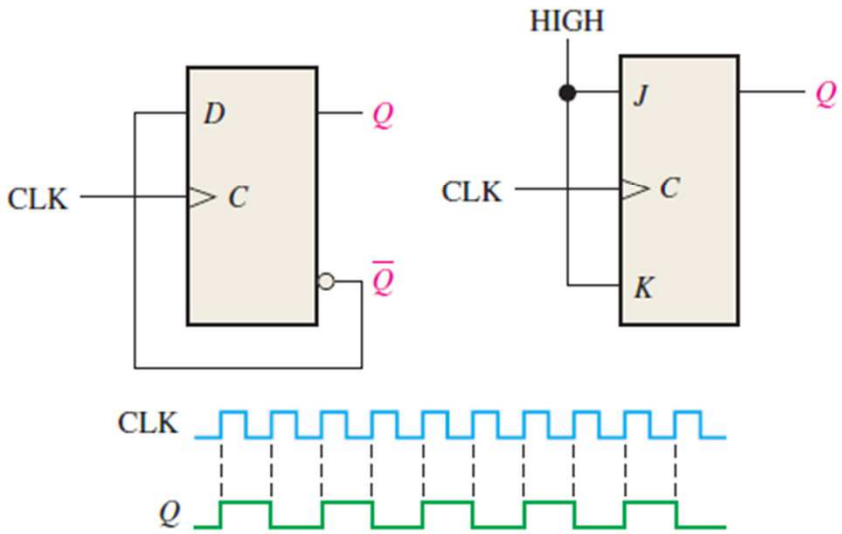
$Q_2Q_1Q_0$.

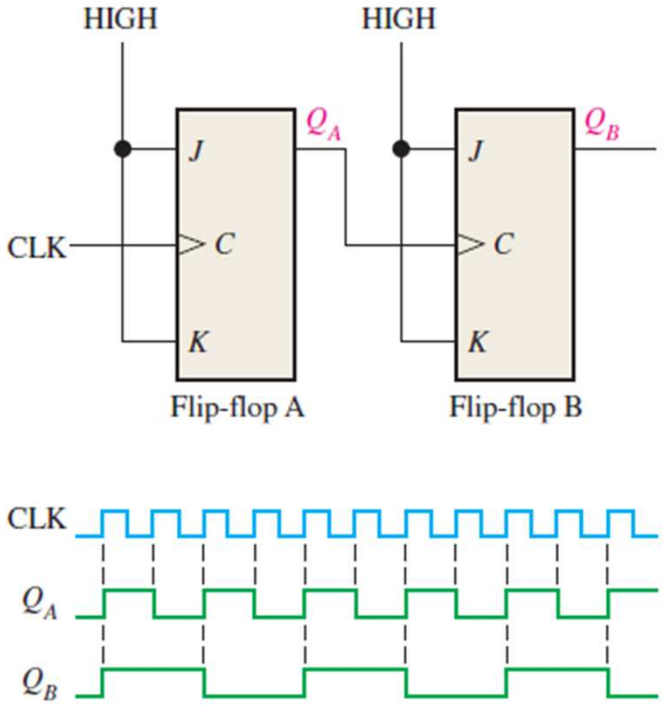
Divisores de Frequência por 2 e por 4
Frequency Dividers

Um contador assíncrono pode funcionar também como divisor de frequência. Os contadores assíncronos abaixo permitem contar até 1 (decimal, o e 1) e até 3 (em decimal, 0, 1, 2 e 3) são também designados como **divisores de frequência por 2 e por 4**.

Flip-flops T: $Q_{n+1} = T \cdot \overline{Q_n} + \bar{T} \cdot Q_n$

Fip-flops D: $Q_{n+1} = D$

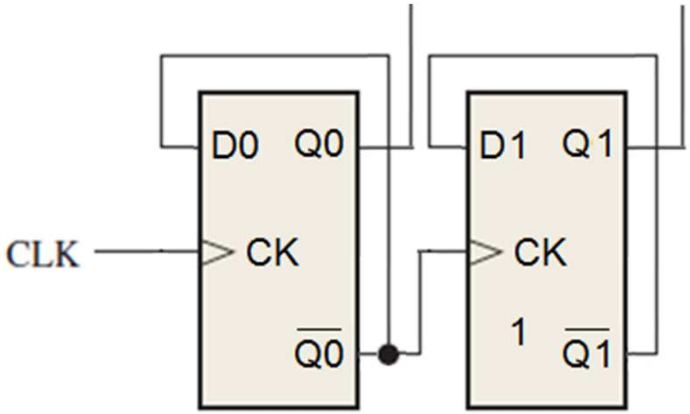


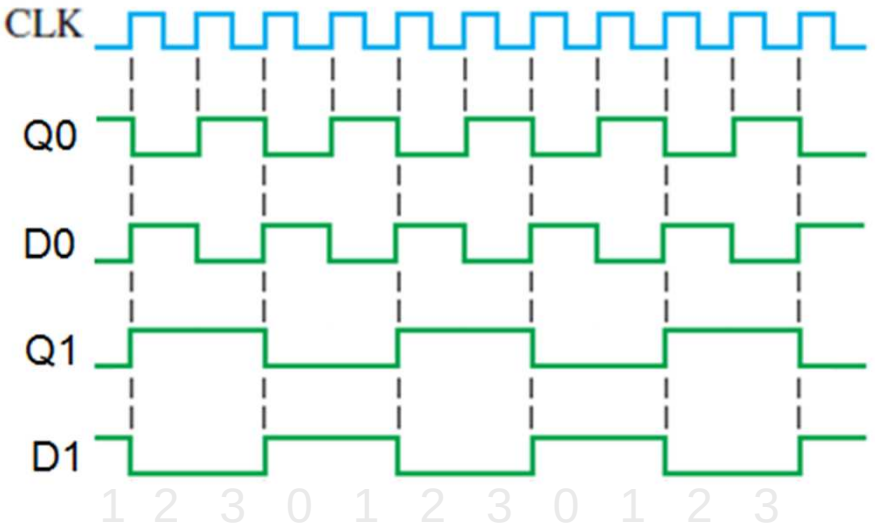


Os flip-flops D e J-K da esquerda estão ligados para funcionar como divisores de 2. Os flip-flops J-K da direita são colocados em cascata para formar um divisor por 4.

Divisor de Frequência por 4 com flip-flops D

O diagrama abaixo corresponde a um contador **assíncrono de 2 bits**, formado **por 2 flip-flop do tipo D** ativados à transição positiva do sinal de relógio. É também indicado o respetivo diagrama temporal.





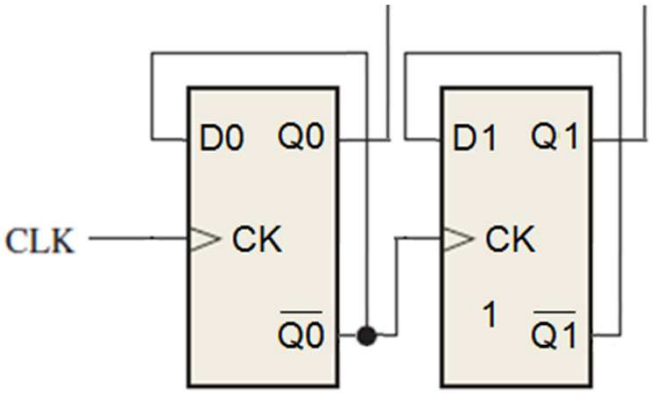
$Q_1Q_0 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00 \rightarrow 01$

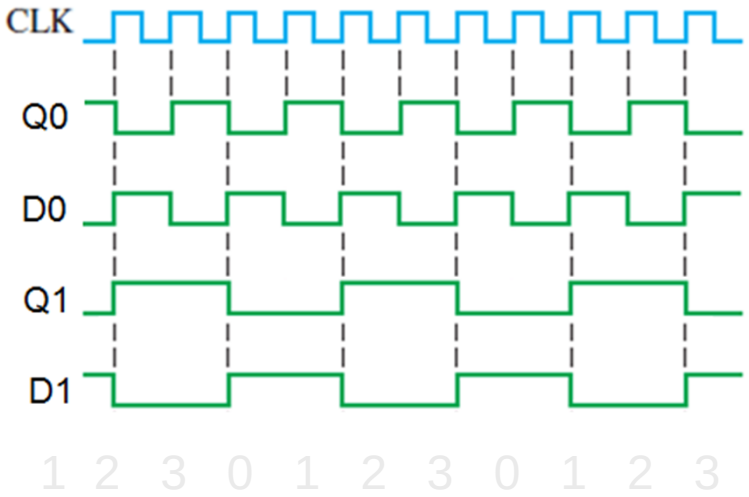
Este contador assíncrono de 2 bits permite contar até 3 (em decimal) é também um **divisor de frequência por 2 e por 4**: a saída de Q0 corresponde a um sinal com o **dobro** do período do sinal de relógio CK; a saída Q1 corresponde a um sinal cujo período é o **quadruplo** do período do sinal de relógio.

Contadores de Módulo

O módulo de um contador corresponde ao número de estado que o contador percorre (“conta”) em um ciclo completo de contagem. O módulo de um contador é um valor x que o contador pode contar. Um contador de módulo igual a 4 pode contar até 4; um contador de módulo 8 pode contar até 8 (7 decimal). (Ter presente que um contador também faz divisão de frequência).

O diagrama abaixo corresponde a um contador **assíncrono de 2 bits**, formado **por 2 flip-flop do tipo D** ativados à transição positiva do sinal de relógio.





Este contador assíncrono de 2 bits permite contar até 4 (0, 1, 2 e 3, em decimal). Portanto é um contador de módulo 4. Este contador atua também como um divisor de frequência por 2 e um divisor de frequência por 4.

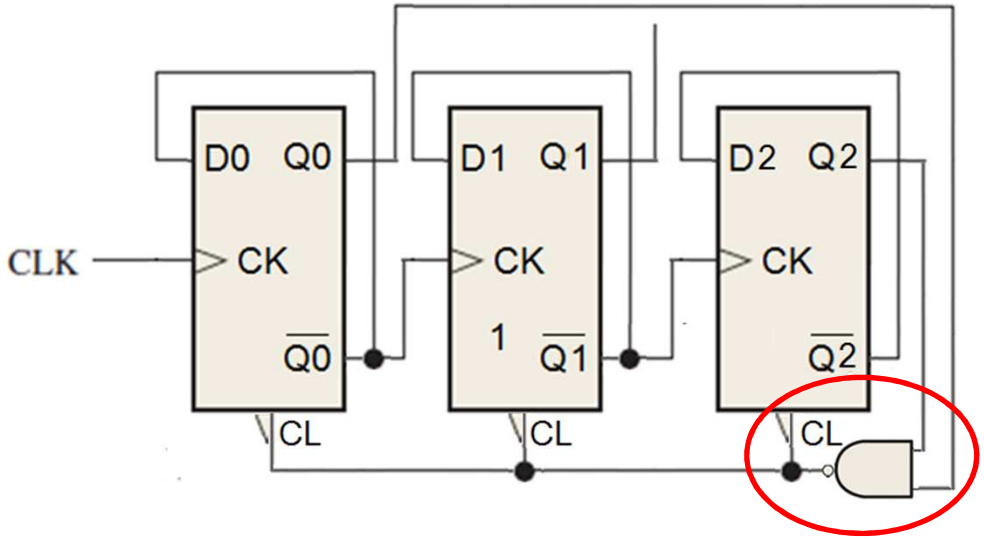
Contador de Módulo 6

O módulo de um contador é um valor x que o contador pode contar. Um contador de módulo igual a 6 pode contar até 6 [000 (0) … 101 (5)].

O diagrama abaixo corresponde a um **contador assíncrono de 3 bit**, formado por 3 flip-flop do tipo D ativados à transição positiva do sinal de relógio.

Este contador poderá contar até 8. Contudo pretende-se que ele conta apenas até 6. Neste caso a contagem é inicializada à 7ª sequencia, por forma a que contagem termine após se obterem as primeiras 6 sequências.

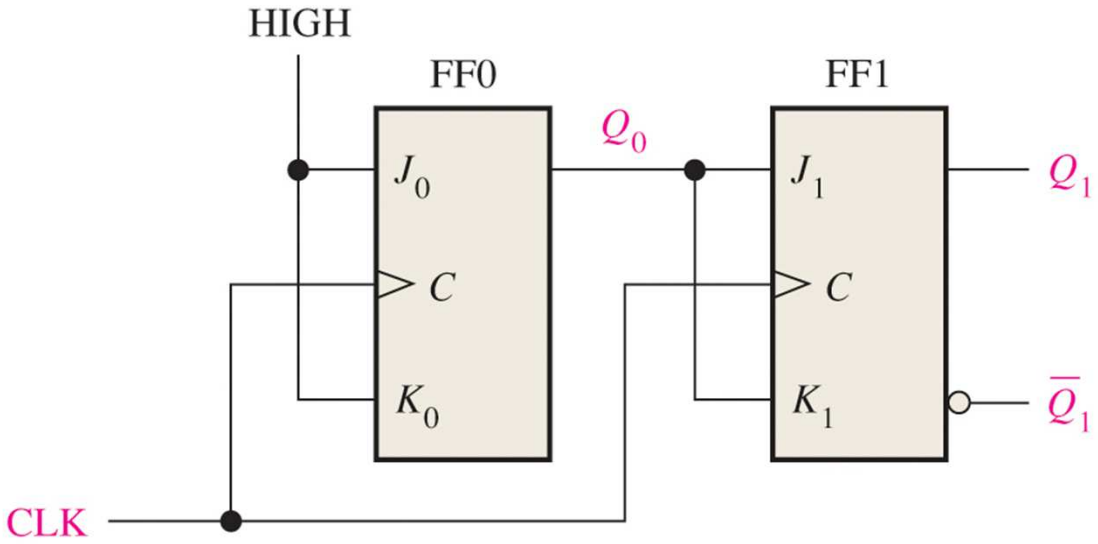
A sequência '101', isto é, quando Q_0=1, Q_1=0, e Q_2=1 permite reiniciar a contagem, e, portanto, inicializar o respetivo flip-flop com valor zero (clear, CL='0'). O sinal de inicialização ou de clear – é aplicado às entradas CL dos flip-flops.



$$Q_2Q_1Q_0 \rightarrow 000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100 \rightarrow 101 \rightarrow 000$$

Contador de 2 bit síncrono
A 2-bit Synchronous Counter

Um contador é síncrono quando os respetivos flip-flops (básculas) são ativados ao mesmo tempo, usando um sinal de relógio comum. Nestes contadores é eliminada a desvantagem dos diferentes tempos de propagação dos sinais “através” dos flip-flops. Contudo, e por isso, requerem mais elementos para controlar as mudanças de estados.

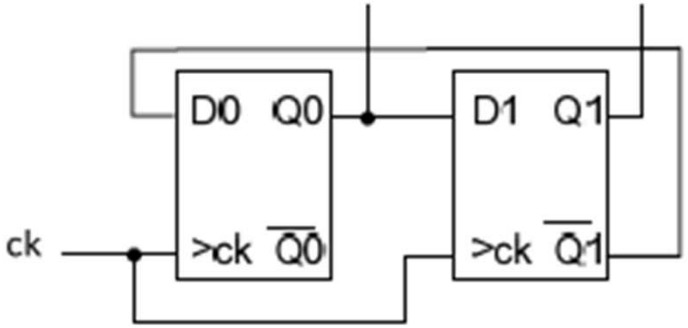


Este contador síncrono de 2 bit tem a mesma sequência de contagem do contador assíncrono de 3 bit anterior.

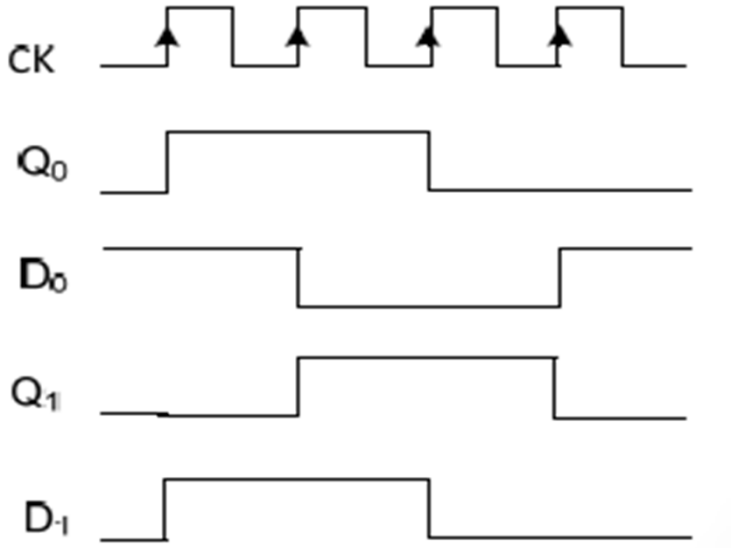
Flip-flops T: $Q_{n+1} = T \cdot \overline{Q_n} + \bar{T} \cdot Q_n$.

Contador de 2 bit síncrono com flip-flops D

Um contador de código Gray de 2 bit pode ser implementado com 2 flip-flops do tipo D sensíveis ao flanco ascendente do sinal de relógio. Novamente não é considerado o tempo de propagação nos flip-flops.



$D_0 = \overline{Q_1} \qquad D_1 = Q_0$

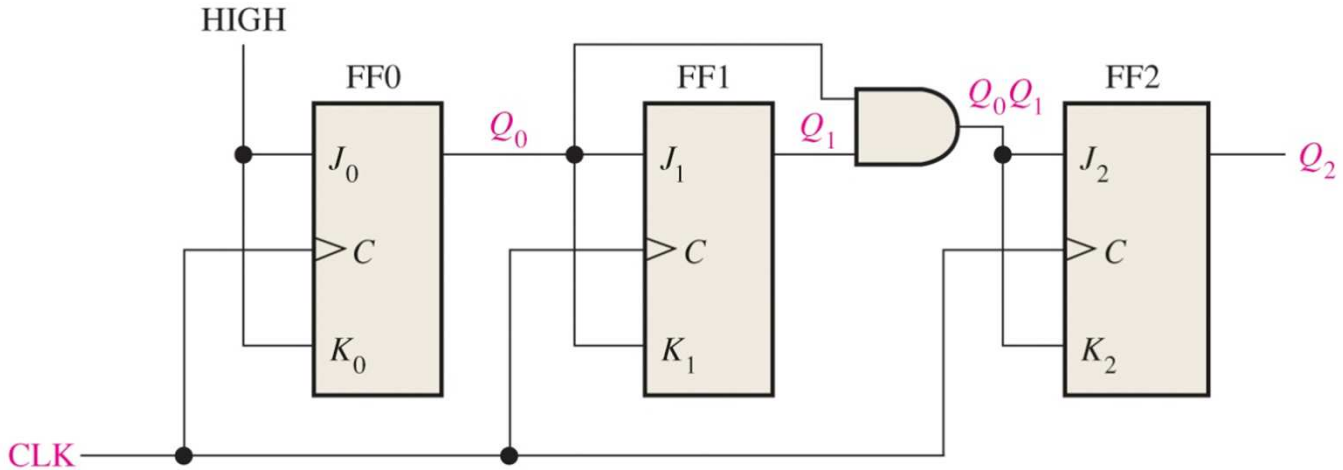


$Q_1Q_0 \rightarrow '00' \rightarrow '01' \rightarrow '11' \rightarrow '10' \rightarrow '00'$

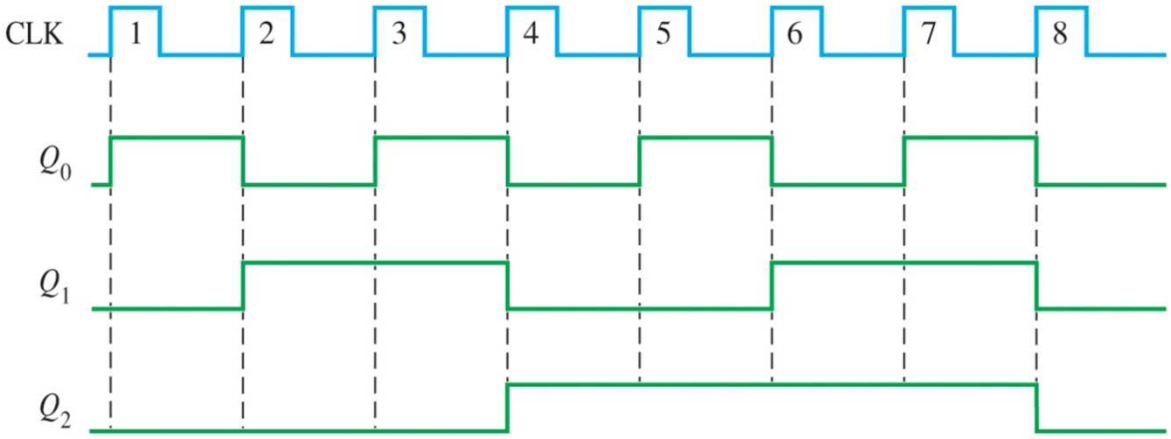
Flip-flops D: $Q_{n+1} = D$.

Contador binário de 3 bit síncrono

A 3-bit Synchronous Binary Counter

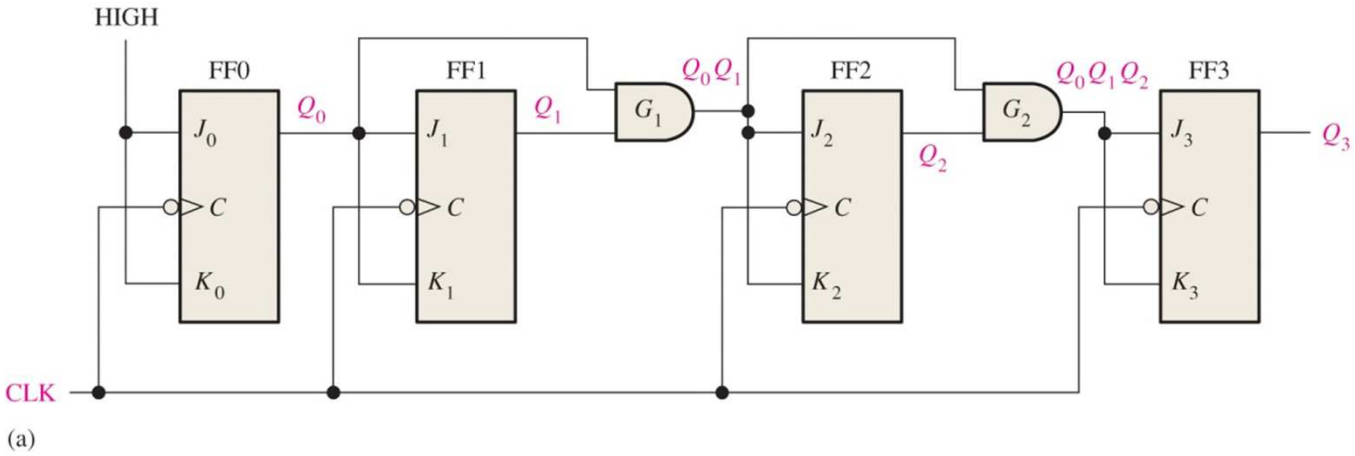


The 3-bit binary counter has an AND gate, unlike the 2-bit counter just described. The output from the AND gate provides the J and K inputs to FF2.

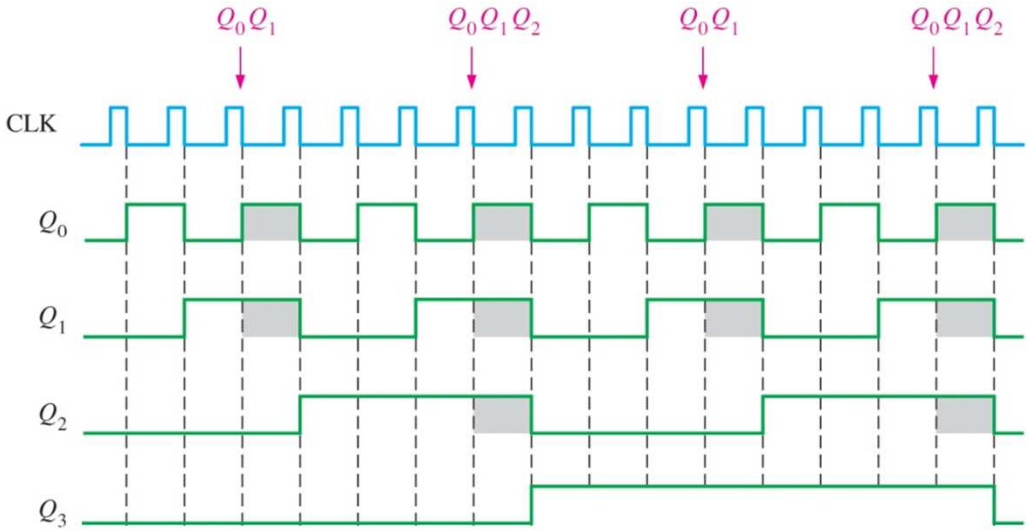


Contador binário síncrono de 4 bit

A 4-bit Synchronous Binary Counter



(a)

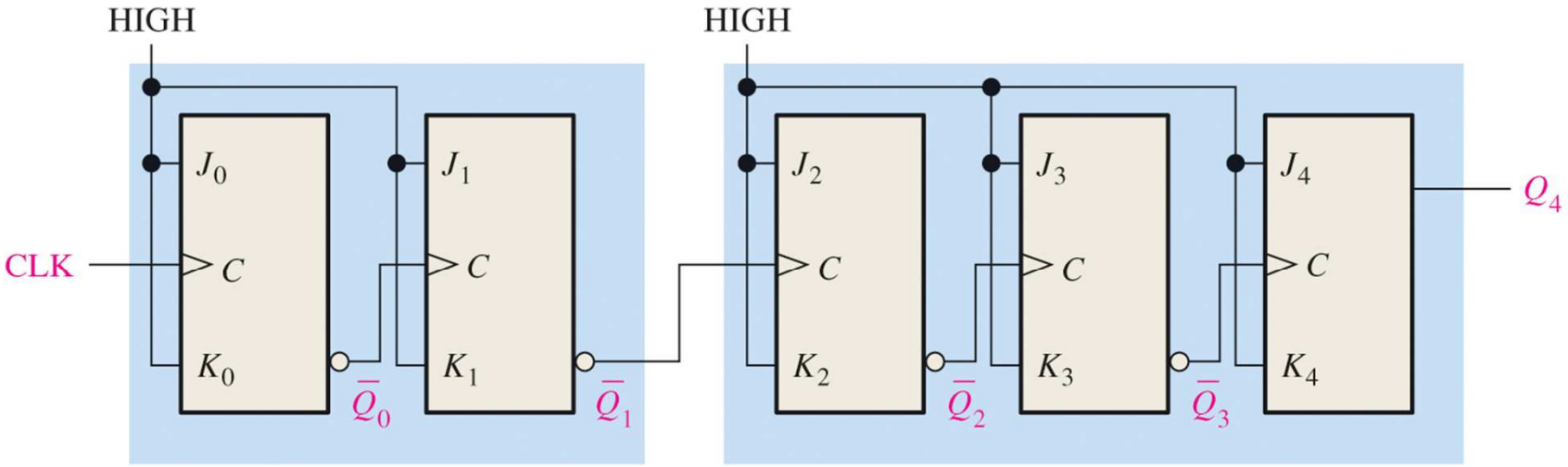


(b)

Contadores de módulo elevado

Contadores em cascata
Cascaded Counters

Counters can be cascaded to produce various divide-by (or *modulus*) values. In this case, a modulus-4 counter is cascaded with a modulus-8 counter. The resulting counter modulus equals the product of the two counters: **Modulus 32.**

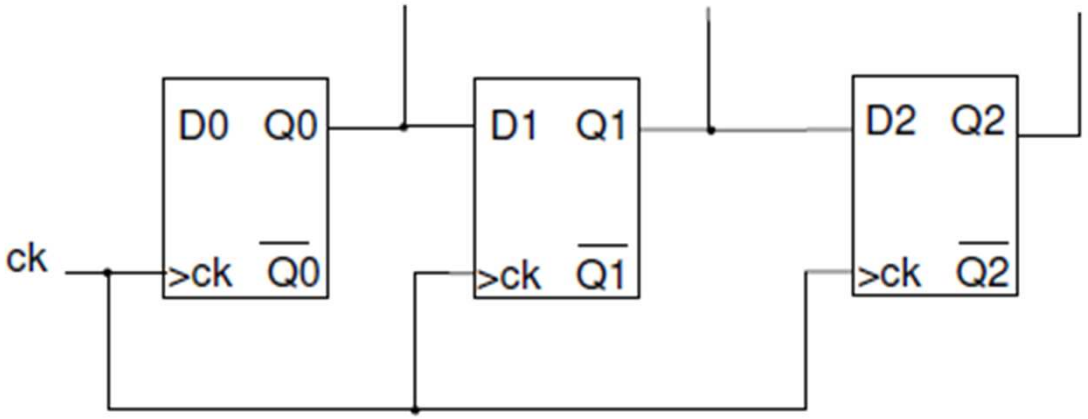


$Q_4Q_3Q_2Q_1Q_0$

Registo de Deslocamento

Num registo de deslocamento, a informação que se encontra associada à saída de flip-flop é transferida, no próximo flanco, ao flip-flop seguinte.

Registo de deslocamento de 3 bit implementado com flip-flop do tipo D:

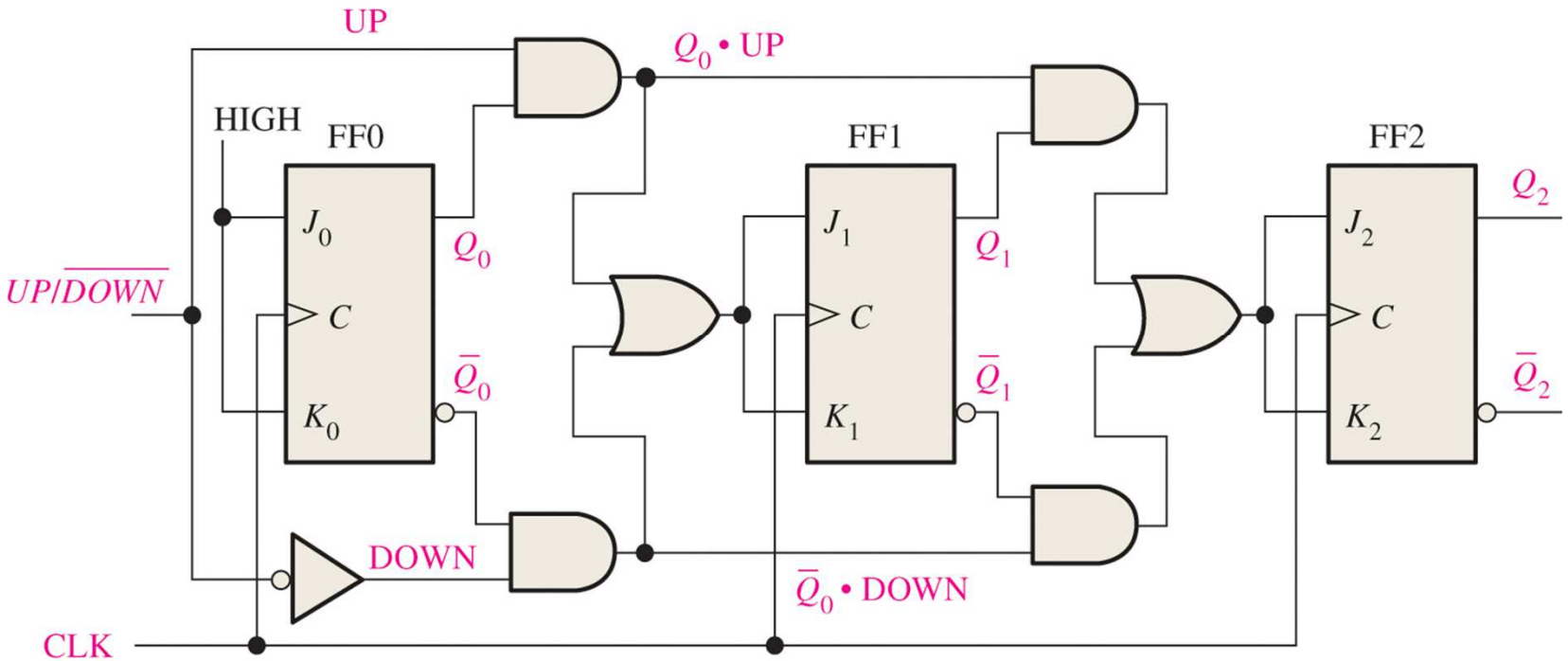


D_0 → variável exterior, para n≥1 tem-se:

$$D_{n+1} = Q_n$$

Contador síncrono Ascendente/Descendente
Up/Down Synchronous Counters

An up/down counter is capable of progressing in either direction depending on a control input.



Contador descodificador
Counter Decoding

Decoding is the detection of a binary number and can be done with an AND gate.

The output from the AND gate in the circuit shown goes high when the counter output equals six (110).