2.7 Binary Arithmetic

Binary arithmetic is essential in all types of digital systems. To understand these systems, you must know the basics of binary addition, subtraction, multiplication, and division.

2.7.1 Binary Addition

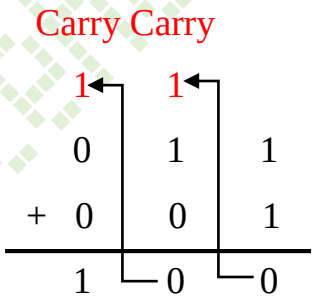
The four basic rules for adding binary digits (bits) are as follows:

0 + 0 = 0 Sum of 0 with a carry of 0
0 + 1 = 1 Sum of 1 with a carry of 0
1 + 0 = 1 Sum of 1 with a carry of 0
1 + 1 = 10 Sum of 0 with a carry of 1

Notice that the first three rules result in a single bit and in the fourth rule the addition of two 1s yields a binary two (10). When binary numbers are added, the last condition creates a sum of 0 in a given column and a carry of 1 over to the next column to the left, as illustrated in the following examples:

Example: Add 11 + 1

Sol.



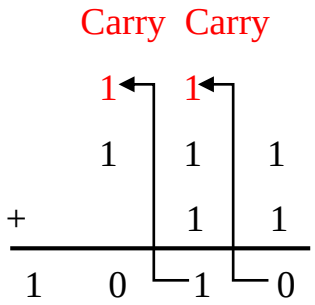
In the right column, 1 + 1 = 0 with a carry of 1 to the next column to the left. In the middle column, 1 + 1 + 0 = 0 with a carry of 1 to the next column to the left. In the left column, 1 + 0 + 0 = 1.

Carry bits

1 + 0 + 0 = 0 1 Sum of 1 with a carry of 0
1 + 1 + 0 = 1 0 Sum of 0 with a carry of 1
1 + 0 + 1 = 1 0 Sum of 0 with a carry of 1
1 + 1 + 1 = 1 1 Sum of 1 with a carry of 1

Example: Add 111 + 11

Sol.



2.7.2 Binary Subtraction

The four basic rules for subtracting bits are as follows:

0 - 0 = 0

1 - 1 = 0

1 - 0 = 1

10 - 1 = 1 0 - 1 with a borrow of 1

When subtracting numbers, you sometimes have to borrow from the next column to the left. A borrow is required in binary only when you try to subtract a 1 from a 0. In this case, when a 1 is borrowed from the next column to the left, a 10 is created in the column being subtracted, and the last of the four basic rules just listed must be applied.

Example: Subtract 011_2 from 101_2.

Sol.

Left column:

When a 1 is borrowed,

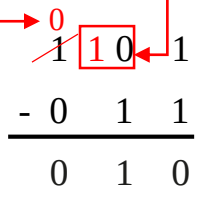
a 0 is left, so 0 - 0 = 0.

Middle column:

Borrow 1 from next column

to the left, making a 10 in

this column, then 10 - 1 = 1.



2.7.3 Binary Multiplication

The four basic rules for multiplying bits are as follows:

$$0 \times 0 = 0, \quad 0 \times 1 = 0, \quad 1 \times 0 = 0, \quad 1 \times 1 = 1$$

Multiplication is performed with binary numbers in the same manner as with decimal numbers. It involves forming partial products, shifting each successive partial product left one place, and then adding all the partial products.

Example**:** Perform the following binary multiplications:

(a) $11_2 \times 11_2$ (b) $101_2 \times 111_2$

Sol.

(a)

$$\begin{array}{r} 11 \\ \times\ 11 \\ \hline 11 \\ +11 \\ \hline \mathbf{1001} \end{array} \qquad \begin{array}{r} 3 \\ \times\ 3 \\ \hline 9 \end{array}$$

Partial products

(b)

$$\begin{array}{r} 111 \\ \times\ 101 \\ \hline 111 \\ 000 \\ +111 \\ \hline \mathbf{100011} \end{array} \qquad \begin{array}{r} 7 \\ \times\ 5 \\ \hline 35 \end{array}$$

Partial products

2.7.4 Binary Division

Division in binary follows the same procedure as division in decimal

Example**:** Perform the following binary divisions: (a) $110_2 \div 11_2$ (b) $110_2 \div 10_2$

Sol.

(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)

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

2.8 Complements of Binary Numbers

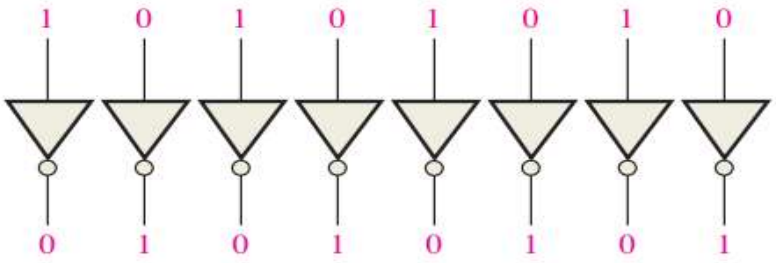
The 1's complement and the 2's complement of a binary number are important because they permit the representation of negative numbers. The method of 2's complement arithmetic is commonly used in computers to handle negative numbers.

2.8.1 Finding the 1's Complement

The 1's complement of a binary number is found by changing all 1s to 0s and all 0s to 1s, as illustrated below:

1 0 1 1 0 0 1 0	Binary number
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
0 1 0 0 1 1 0 1	1's complement

The simplest way to obtain the 1's complement of a binary number with a digital circuit is to use parallel inverters (NOT circuits), as shown in Figure below for an 8-bit binary number.



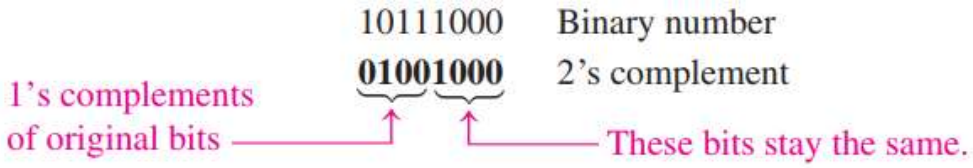
Example of inverters used to obtain the 1's complement of a binary number.

2.8.2 Finding the 2's Complement

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

Example**:** Find the 2's complement of 10111000 using the alternative method.

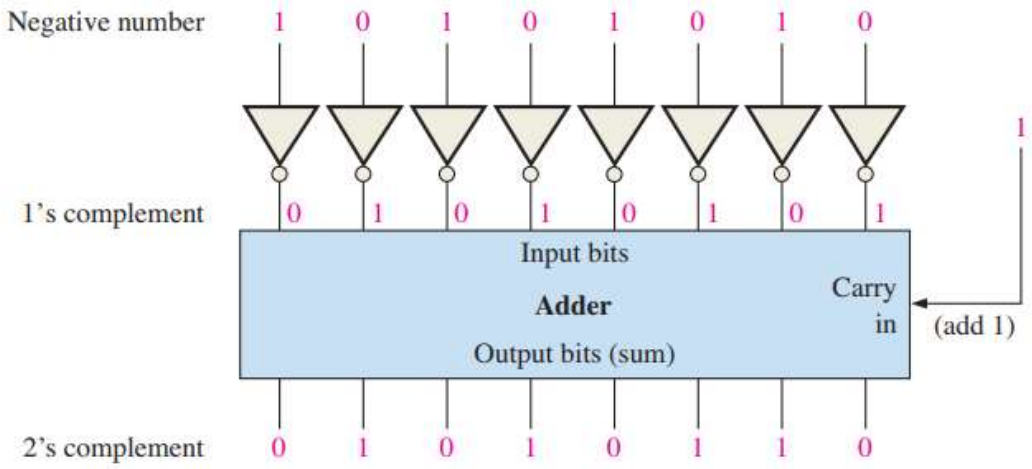
Sol.



Related Problem:

Find the 2's complement of 11000000.

The 2's complement of a negative binary number can be realized using inverters and an adder, as indicated in Figure below. This illustrates how an 8-bit number can be converted to its 2's complement by first inverting each bit (taking the 1's complement) and then adding 1 to the 1's complement with an adder circuit.



Example of obtaining the 2's complement of a negative binary number.

To convert from a 1's or 2's complement back to the true (uncomplemented) binary form, use the same two procedures described previously. To go from the 1's complement back to true binary, reverse all the bits. To go from the 2's complement form back to true binary, take the 1's complement of the 2's complement number and add 1 to the least significant bit.

2.8.3 1's Complement Form

Positive numbers in 1's complement form are represented the same way as the positive sign-magnitude numbers. Negative numbers, however, are the 1's complements of the corresponding positive numbers. For example, using eight bits, the decimal number 225 is expressed as the 1's complement of +25 (00011001) as
11100110

In the 1's complement form, a negative number is the 1's complement of the corresponding positive number.

Example**:** Find the 1's complement of 00010011.

Sol.

Change each bit in a number to get the 1's complement. The 1's complement of a binary number is found by changing all 1s to 0s and all 0s to 1s, as illustrated below:

0	0	0	1	0	0	1	1	Binary number
↓	↓	↓	↓	↓	↓	↓	↓	
1	1	1	0	1	1	0	0	1's complement

2.8.4 2's Complement Form

Positive numbers in 2's complement form are represented the same way as in the signmagnitude and 1's complement forms. Negative numbers are the 2's complements of the corresponding positive numbers. Again, using eight bits, let's take decimal number 225 and express it as the 2's complement of +25 (00011001). Inverting each bit and adding 1, you get

$$-25 = 11100111$$

In the 2's complement form, a negative number is the 2's complement of the corresponding positive number.

Example**:** Represent −2 in 2's complement.

Sol.

2	=	0	0	1	0
		1	1	0	1
+					1
		1	1	1	0

1's complement

results in 2's complement

= -2

2.9 Binary Coded Decimal (BCD)

Binary coded decimal (BCD) is a way to express each of the decimal digits with a binary code. There are only ten code groups in the BCD system, so it is very easy to convert between decimal and BCD. Because we like to read and write in decimal, the BCD code provides an excellent interface to binary systems. Examples of such interfaces are keypad inputs and digital readouts.

2.9.1 The 8421 BCD Code

The 8421 code is a type of BCD (binary coded decimal) code. Binary coded decimal means that each decimal digit, 0 through 9, is represented by a binary code of four bits. The designation 8421 indicates the binary weights of the four bits (2^3, 2^2, 2^1, 2^0). The ease of conversion between 8421 code numbers and the familiar decimal numbers is the main advantage of this code. All you have to remember are the ten binary combinations that represent the ten decimal digits as shown in Table below. The 8421 code is the predominant BCD code, and when we refer to BCD, we always mean the 8421 code unless otherwise stated.

In BCD, 4 bits represent each decimal digit.

Decimal/BCD conversion.

Decimal Digit	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

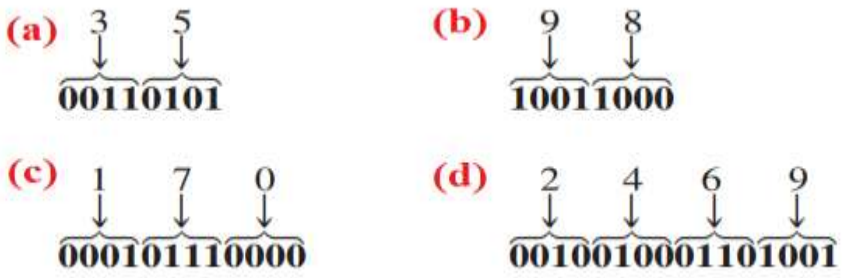
2.9.1.1 Invalid Codes

You should realize that, with four bits, sixteen numbers (0000 through 1111) can be represented but that, in the 8421 code, only ten of these are used. The six code combinations that are not used—1010, 1011, 1100, 1101, 1110, and 1111—are invalid in the 8421 BCD code.
To express any decimal number in BCD, simply replace each decimal digit with the appropriate 4-bit code, as shown by example below.

Example**:** Convert each of the following decimal numbers to BCD:

(a) 35 (b) 98 (c) 170 (d) 2469

Sol.

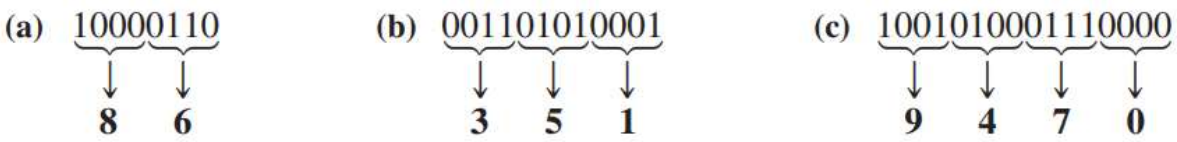


It is equally easy to determine a decimal number from a BCD number. Start at the right-most bit and break the code into groups of four bits. Then write the decimal digit represented by each 4-bit group.

Example**:** Convert each of the following BCD codes to decimal:

(a) 10000110 (b) 001101010001 (c) 1001010001110000

Sol.



2.9.1.2 Applications

Digital clocks, digital thermometers, digital meters, and other devices with seven-segment displays typically use BCD code to simplify the displaying of decimal numbers. BCD is not as efficient as straight binary for calculations, but it is particularly useful if only limited processing is required, such as in a digital thermometer.

2.9.2 BCD Addition

BCD is a numerical code and can be used in arithmetic operations. Addition is the most important operation because the other three operations (subtraction, multiplication, and division) can be accomplished by the use of addition. Here is how to add two BCD numbers:

Step 1: Add the two BCD numbers.
Step 2: If a 4-bit sum is equal to or less than 9, it is a valid BCD number.
Step 3: If a 4-bit sum is greater than 9, or if a carry out of the 4-bit group is generated, it is an invalid result. Add 6 (0110) to the 4-bit sum in order to skip the six invalid states and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next 4-bit group.

Example**:** Add the following BCD numbers:

(a) 0011 + 0100 (b) 00100011 + 00010101

(c) 10000110 + 00010011 (d) 010001010000 + 010000010111

Sol.

(a)
$$\begin{array}{r} 0011 \\ +\,0100 \\ \hline \mathbf{0111} \end{array} \qquad \begin{array}{r} 3 \\ +\,4 \\ \hline 7 \end{array}$$

(b)
$$\begin{array}{rr} 0010 & 0011 \\ +\,0001 & 0101 \\ \hline \mathbf{0011} & \mathbf{1000} \end{array} \qquad \begin{array}{r} 23 \\ +\,15 \\ \hline 38 \end{array}$$

(c)
$$\begin{array}{rr} 1000 & 0110 \\ +\,0001 & 0011 \\ \hline \mathbf{1001} & \mathbf{1001} \end{array} \qquad \begin{array}{r} 86 \\ +\,13 \\ \hline \mathbf{99} \end{array}$$

(d)
$$\begin{array}{rrr} 0100 & 0101 & 0000 \\ +\,0100 & 0001 & 0111 \\ \hline \mathbf{1000} & \mathbf{0110} & \mathbf{0111} \end{array} \qquad \begin{array}{r} 450 \\ +\,417 \\ \hline \mathbf{867} \end{array}$$

Note that in each case the sum in any 4-bit column does not exceed 9, and the results are valid BCD numbers.

Example**:** Add the following BCD numbers:
(a) 1001 + 0100 (b) 1001 + 1001

(c) 00010110 + 00010101 (d) 01100111 + 01010011

Sol.

(a)

1001			9
+ 0100			+4
1101		Invalid BCD number (>9)	13
+ 0110		Add 6	
0001	**0011**	Valid BCD number	
↓	↓		
1	3		

(b)

1001			9
+ 1001			+ 9
1 0010		Invalid because of carry	18
+ 0110		Add 6	
0001	**1000**	Valid BCD number	
↓	↓		
1	8		

(c)

0001	0110		16
+ 0001	0101		+ 15
0010	1011	Right group is invalid (>9), left group is valid.	31
	+ 0110	Add 6 to invalid code. Add carry, 0001, to next group.	
0011	**0001**	Valid BCD number	
↓	↓		
3	1		

(d)

	0110	0111		67
	+ 0101	0011		+ 53
	1011	1010	Both groups are invalid (>9)	120
	+ 0110	+ 0110	Add 6 to both groups	
0001	**0010**	**0000**	Valid BCD number	
↓	↓	↓		
1	2	0		

Advantages of BCD Codes	Disadvantages of BCD Codes
1. BCD coding is similar to the binary equivalent of decimal numbers 0–9. 2. BCD has no limitation for number size. 3. It is easier to convert decimal numbers from or to BCD than to binary form.	1. Each decimal number requires four bits to be represented in the BCD code. 2. Arithmetic operations in BCD or weighted codes are much complicated as it deals with more number of bits and also it has different set of rules. 3. BCD is less efficient than binary.

2.10 Digital Codes

Many specialized codes are used in digital systems. You have just learned about the BCD code; now let's look at a few others. Some codes are strictly numeric, like BCD, and others are alphanumeric; that is, they are used to represent numbers, letters, symbols, and instructions. The codes introduced in this section are the Gray code, the ASCII code, and the Unicode.

2.10.1 The Gray Code

The Gray code is unweighted and is not an arithmetic code; that is, there are no specific weights assigned to the bit positions. The important feature of the Gray code is that it exhibits only a single bit change from one code word to the next in sequence. This property is important in many applications, such as shaft position encoders, where error susceptibility increases with the number of bit changes between adjacent numbers in a sequence. The table below is a listing of the 4-bit Gray code for decimal numbers 0 through 15. Binary numbers are shown in the table for reference. Like binary numbers, the Gray code can have any number of bits. Notice the single-bit change between successive Gray code words. For instance, in going from decimal 3 to decimal 4, the Gray code changes from 0010 to 0110, while the binary code changes from 0011 to 0100, a change of three bits. The only bit change in the Gray code is in the third bit from the right: the other bits remain the same.

Four-bit Gray code.

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

The single bit change characteristic of the Gray code minimizes the chance for error.

2.10.1.1 Binary-to-Gray Code Conversion

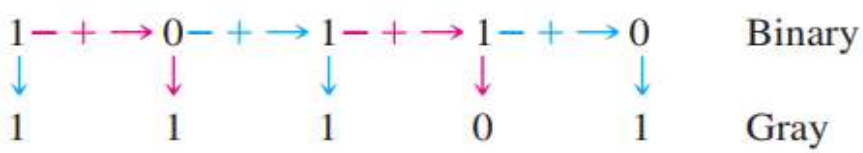
Conversion between binary code and Gray code is sometimes useful. The following rules explain how to convert from a binary number to a Gray code word:

1. The most significant bit (left-most) in the Gray code is the same as the corresponding MSB in the binary number.

2. Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.

Example**:** **T**he conversion of the binary number 10110 to Gray code is as

Sol.



The Gray code is 11101.

2.10.1.2 Gray-to-Binary Code Conversion

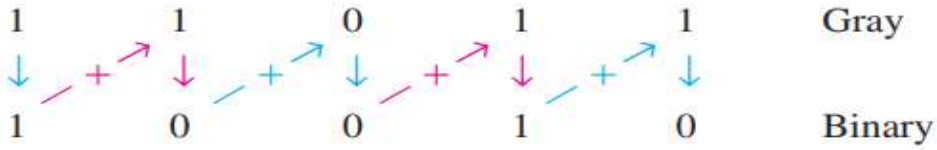
To convert from Gray code to binary, use a similar method; however, there are some differences. The following rules apply:

1. The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.

2. Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.

Example**:** **T**he conversion of the Gray code word 11011 to binary is as follows:

Sol.



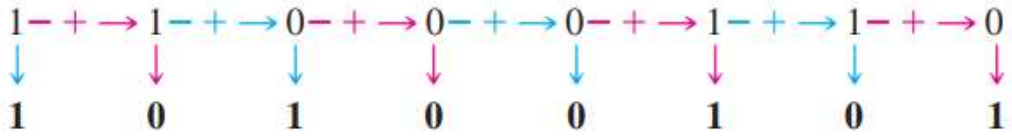
The binary number is 10010.

Example**:** (a) Convert the binary number 11000110 to Gray code.

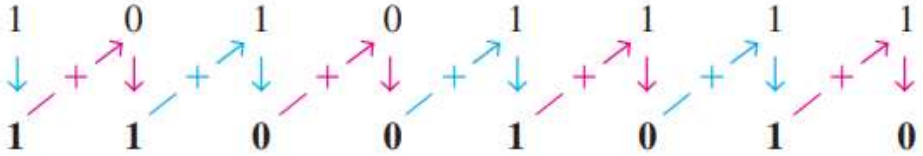
(b) Convert the Gray code 10101111 to binary.

Sol.

(a) Binary to Gray code:



(b) Gray code to binary:



2.10.2 Alphanumeric codes

In order to communicate, you need not only numbers but also letters and other symbols. Alphanumeric codes are codes that represent numbers and alphabetic characters (letters). At a minimum, an alphanumeric code must represent 10 decimal digits and 26 letters of the alphabet, for a total of 36 items. This number requires six bits in each code combination because five bits are insufficient ($2^5 = 32$). There are 64 total combinations of six bits, so there are 28 unused code combinations.

We need spaces, periods, colons, semicolons, question marks, etc. We also need instructions to tell the receiving system what to do with the information. With codes that are six bits long, we can handle decimal numbers, the alphabet, and 28 other symbols.

2.10.3 ASCII

ASCII is the abbreviation for American Standard Code for Information Interchange. Pronounced "askee," ASCII is a universally accepted alphanumeric code used in most computers and other electronic equipment. Most computer keyboards are standardized with ASCII. When you enter a letter, a number, or control command, the corresponding ASCII code goes into the computer.

ASCII has 128 characters and symbols represented by a 7-bit binary code. Actually, ASCII can be considered an 8-bit code with the MSB always 0. This 8-bit code is 00 through 7F in hexadecimal. The first 32 ASCII characters are nongraphic commands that are never printed or displayed and are used only for control purposes. Examples of the control characters are "null," "line feed," "start of the text," and "escape." The other characters are graphic symbols that can be printed or displayed and include the letters of the alphabet (lowercase and uppercase), the ten decimal digits, punctuation signs, and other commonly used symbols.

Info Note: A computer keyboard has a dedicated microprocessor that constantly scans keyboard circuits to detect when a key has been pressed and released. A unique scan code is produced by computer software representing that particular key. The scan code is then converted to an alphanumeric code (ASCII) for use by the computer.

2.10.3.1 The ASCII Control Characters

The first 32 codes in the (ASCII Table) below represent the control characters. These are used to allow devices such as a computer and printer to communicate with each other when passing information and data. The control key function allows a control character to be entered directly from an ASCII keyboard by pressing the control key (CTRL) and the corresponding symbol.

ASCII Table

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

Example Use ASCII table to determine the binary ASCII codes that are entered from the computer's keyboard when the following C language program statement is typed in. Also express each code in hexadecimal**.** *If (x > 5)*

Sol.

The ASCII code for each symbol is found in ASCII table.

Symbol	Binary	Hexadecimal
i	1101001	69_{16}
f	1100110	66_{16}
Space	0100000	20_{16}
(	0101000	28_{16}
x	1111000	78_{16}
>	0111110	$3E_{16}$
5	0110101	35_{16}
)	0101001	29_{16}