NUMBER SYSTEM

COMMON NUMBER SYSTEM

SYSTEM	BASE	SYMBOLS	EXAMPLE
Decimal	10	0,1,2,3,4,5,6,7,8,9	$(9)_{10}$
Binary	2	0,1	$(111)_2$
Octal	8	0,1,2,3,4,5,6,7	$(5)_8$
Hexa Decimal	16	0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F	$(2A)_{16}$

CASE I: DECIMAL TO BASE-R CONVERSION

1. DECIMAL TO BASE-2 CONVERSION

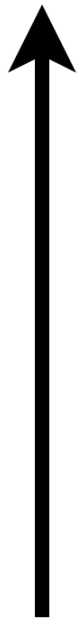
2. DECIMAL TO BASE-8 CONVERSION

3. DECIMAL TO BASE-16 CONVERSION

1. DECIMAL TO BINARY CONVERSION

$$(7)_{10}$$

2	7	1
2	3	1
2	1	1
	0	



$$\therefore (7)_{10} = (111)_2$$

CLASS WORK

--- CONVERT $(41)_{10}$ TO BINARY

$$(41)_{10}$$

2	41	1
2	20	0
2	10	0
2	5	1
2	2	0
2	1	1
	0	

$$\therefore (41)_{10} = (101001)_2$$

--- CONVERT Fractional Decimal To Binary

$$(0.6875)_{10}$$

DECIMAL TO BINARY CONVERSION

Fractional Decmal	Operation	Product	Fraction Part	Integer Part
0.6875	**Multiply X 2**	**1.3750**	**0.3750**	**1**
0.3750	**Multiply X 2**	**0.7500**	**0.7500**	**0**
0.7500	**Multiply X 2**	**1.5000**	**0.5000**	**1**
0.5000	**Multiply X 2**	**1.0000**	**0.000**	**1**

$$\therefore (0.6875)_{10} = (1011)_{2}$$

CONVERT $(41.625)_{10}$

STEP 1: INTEGER PART

2	41	1
2	20	0
2	10	0
2	5	1
2	2	0
2	1	1
	0	

$\therefore (41)_{10} = (101001)_2$

STEP 2: FRACTIONAL PART

Fractional Decmal	Operation	Product	Fraction Part	Integer Part
0.625	Multiply X 2	1.250	0.250	1
0.250	Multiply X 2	0.500	0.500	0
0.500	Multiply X 2	1.000	0.000	1

$\therefore (0.625)_{10} = (101)_2$

Hence, Combining Step 1 and Step 2

$$(41.625)_{10} = (101001.101)_2$$

2. DECIMAL TO OCTAL CONVERSION

$(XXX)_{10}$  $(XXX)_{8}$

$(153.45)_{10}$  $(?)_{8}$

$(153.45)_{10}$

DECIMAL TO OCTAL CONVERSION

STEP 1: INTEGER PART

8	**153**	1
8	19	3
8	2	2
	0	

$$\therefore (153)_{10} = (231)_8$$

STEP 2: FRACTIONAL PART

Fractional Decimal	Operation	Product	Fraction Part	Integer Part
0.45	Multiply X 8	3.60	0.60	3
0.60	Multiply X 8	4.80	0.80	4
0.80	Multiply X 8	6.40	0.40	6

$$\therefore (0.45)_{10} = (346)_8$$

$(153.45)_{10}$

DECIMAL TO OCTAL CONVERSION

STEP 1: INTEGER PART

8	**153**	1
8	19	3
8	2	2
	0	

$\therefore (153)_{10} = (231)_8$

STEP 2: FRACTIONAL PART

Fractional Decimal	Operation	Product	Fraction Part	Integer Part
0.45	Multiply X 8	3.60	0.60	3
0.60	Multiply X 8	4.80	0.80	4
0.80	Multiply X 8	6.40	0.40	6

$\therefore (0.45)_{10} = (346)_8$

Hence, Combining Step 1 and Step 2

$(153.45)_{10} = (231.346)_8$

3. DECIMAL TO HEXA-DECIMAL CONVERSION

$(XXX)_{10}$  $(XXX)_{16}$

$(1459.43)_{10}$  $(?)_{16}$

DECIMAL TO HEXA-DECIMALCONVERSION

$(1459.43)_{10}$

STEP 1: INTEGER PART

16	**1459**	3
16	91	11 (B)
16	5	5
	0	

$\therefore (1459)_{10} = (5B3)_{16}$

STEP 2: FRACTIONAL PART

Fractional Decimal	Operation	Product	Fraction Part	Integer Part
0.43	Multiply X 16	6.80	0.80	6
0.88	Multiply X 16	14.08	0.08	**E**
0.08	Multiply X 16	1.28	0.28	**1**

. . . . Never Ending Loop

$\therefore (0.43)_{10} = (6E1)_{16}$

Hence, Combining Step 1 and Step 2

$(1459.43)_{10} = (5B3.6E1)_{16}$

CASE I: DECIMAL TO BASE-R CONVERSION SUMMARY

1. Separate the number into **integer** and **fraction** parts if radix point is given.
2. Divide **“Decimal Integer part”** by base **R** repeatedly until quotient becomes **zero** and store remainders at each step.
3. Multiply **“Decimal Fraction part”** successively by **R** and accumulate the integer digits so obtained.
4. Combine both accumulated results and **parenthesize** the whole result with subscript **R**

CASE II: BASE-R TO DECIMAL CONVERSION

1. BASE-2 TO DECIMAL CONVERSION

2. BASE-8 TO DECIMAL CONVERSION

3. BASE-16 TO DECIMAL CONVERSION

1. BINARY TO DECIMAL CONVERSION

$(XXX)_2$ → $(XXX)_{10}$

$(1101.01)_2$ → $(XXX)_{10}$

$$(1101.01)_2$$

$$= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 . \; 0 \times 2^{-1} + 1 \times 2^{-2}$$

$$= 8 + 4 + 0 + 1 . \; 0 + \frac{1}{4}$$

$$= 13.25$$

$$\therefore (1101.01)_2 = (13.25)_{10}$$

1. OCTAL TO DECIMAL CONVERSION

$(XXX)_8$  $(XXX)_{10}$

$(17.6)_8$  $(XXX)_{10}$

$$(17.6)_8$$

$$= 1 \times 8^1 + 7 \times 8^0 . 6 \times 8^{-1}$$

$$= 8 + 7 . 6 \times \frac{1}{8}$$

$$= 15 . 75$$

$$\therefore (17.6)_8 = (15.75)_{10}$$

1. HEXA-DECIMAL TO DECIMAL CONVERSION

$(XXX)_{16}$ → $(XXX)_{10}$

$(A8.D)_{16}$ → $(XXX)_{10}$

$$(A8.D)_{16}$$

$$= A \times 16^{1} + 8 \times 16^{0} . D \times 16^{-1}$$

$$= 10 \times 16 + 8 . 13 \times \frac{1}{16}$$

$$= 160 + 8 . 8125$$

$$= 168.8125$$

$$(A8.D)_{16}$$

$$= A \times 16^{1} + 8 \times 16^{0} . D \times 16^{-1}$$

$$= 10 \times 16 + 8 . 13 \times \frac{1}{16}$$

$$= 160 + 8 . 8125$$

$$= 168.8125$$

$$\therefore (A8.D)_{16} = (168.8125)_{10}$$

CASE III: 2^k TO 2^r CONVERSION

1. 2^1 TO 2^3 CONVERSION
2. 2^1 TO 2^4 CONVERSION
3. 2^3 TO 2^1 CONVERSION
4. 2^4 TO 2^1 CONVERSION
5. 2^4 TO 2^3 CONVERSION
6. 2^3 TO 2^4 CONVERSION

Case III -> 1. 2^1 TO 2^3 CONVERSION

$(\ XXXXXX\ )_2$ ⟶ $(\ XXXXXX\ )_8$

$(\ 11011.110\ )_2$ ⟶ $(\ XXXXXX\ )_8$

Case III -> 1. 2^1 TO 2^3 CONVERSION

$$(11011.110)_2$$

Case III -> 1. 2^1 TO 2^3 CONVERSION

$$(011011.110)_2$$

$$= (33.6)_8$$

$$\therefore (11011.110)_2 = (33.6)_8$$

BINARY TO OCTAL CONVERSION

3-bit Binary Number	Octal Number
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

All Octal Numbers Can be represented by 3 bits

Group The Binary Numbers 3 at a Time

Case III -> 1. 2^1 TO 2^3 CONVERSION

Class Work

BINARY TO OCTAL CONVERSION

3-bit Binary Number	Octal Number
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

All Octal Numbers Can be represented by 3 bits

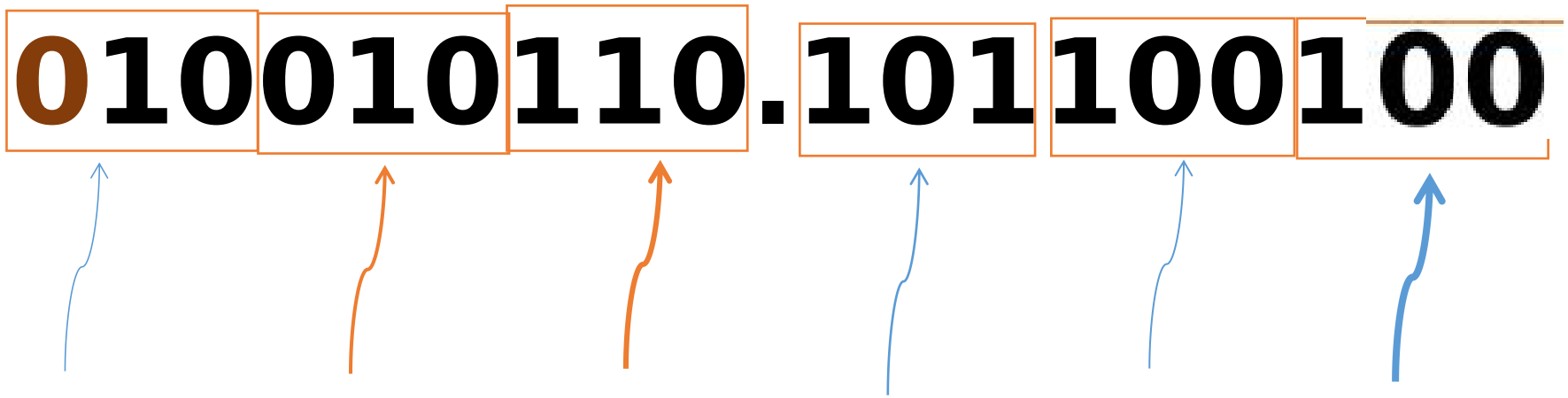
$$(1001110.1011001)_2$$

Case III -> 1. 2^1 TO 2^3 CONVERSION

BINARY TO OCTAL CONVERSION

3-bit Binary Number	Octal Number
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

All Octal Numbers Can be represented by 3 bits



$= (226.544)_8$

$\therefore (010010110.101100100)_2 = (226.544)_8$

Case III -> 2. 2^1 TO 2^4 CONVERSION

$(XXXXXX)_2 \longrightarrow (XXXXXX)_{16}$

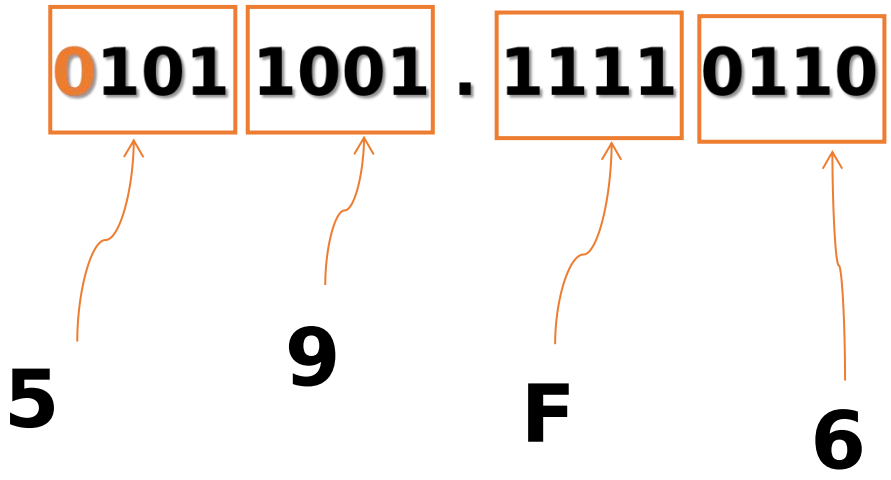
$(1011001.11110110)_2 \longrightarrow (XXXXXX)_{16}$

Case III -> 2. 2^1 TO 2^4 Conversion

$$(1011001.11110110)_2$$

Case III -> 2. 2^1 TO 2^4 Conversion

$$(1011001.11110110)_2$$



$$= 59.F6$$

$$\therefore (1011001.11110110)_2 = (59.F6)_{16}$$

BINARY TO HEXA DECIMAL CONVERSION

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

All Hexa Decimal Numbers Can be represented by 4 bits

Case III -> 2. 2^1 TO 2^4 Conversion

CLASS WORK

$$(1010001011110.001111)_2$$

1010 0010 1110.0011 1100

A 2 E .3 C

$$\therefore (1010001011110.001111)_2 = (A2E.3C)_{16}$$

BINARY TO HEXA DECIMAL CONVERSION

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

All Hexa Decimal Numbers Can be represented by 4 bits

Case III -> 3. 2^3 TO 2^1 CONVERSION

$(XXXXXX)_8$ → $(XXXXXX)_2$

$(237.46)_8$ → $(XXXXXX)_2$

Case III -> 3. 2^3 TO 2^1 CONVERSION

$$(237.46)_8$$

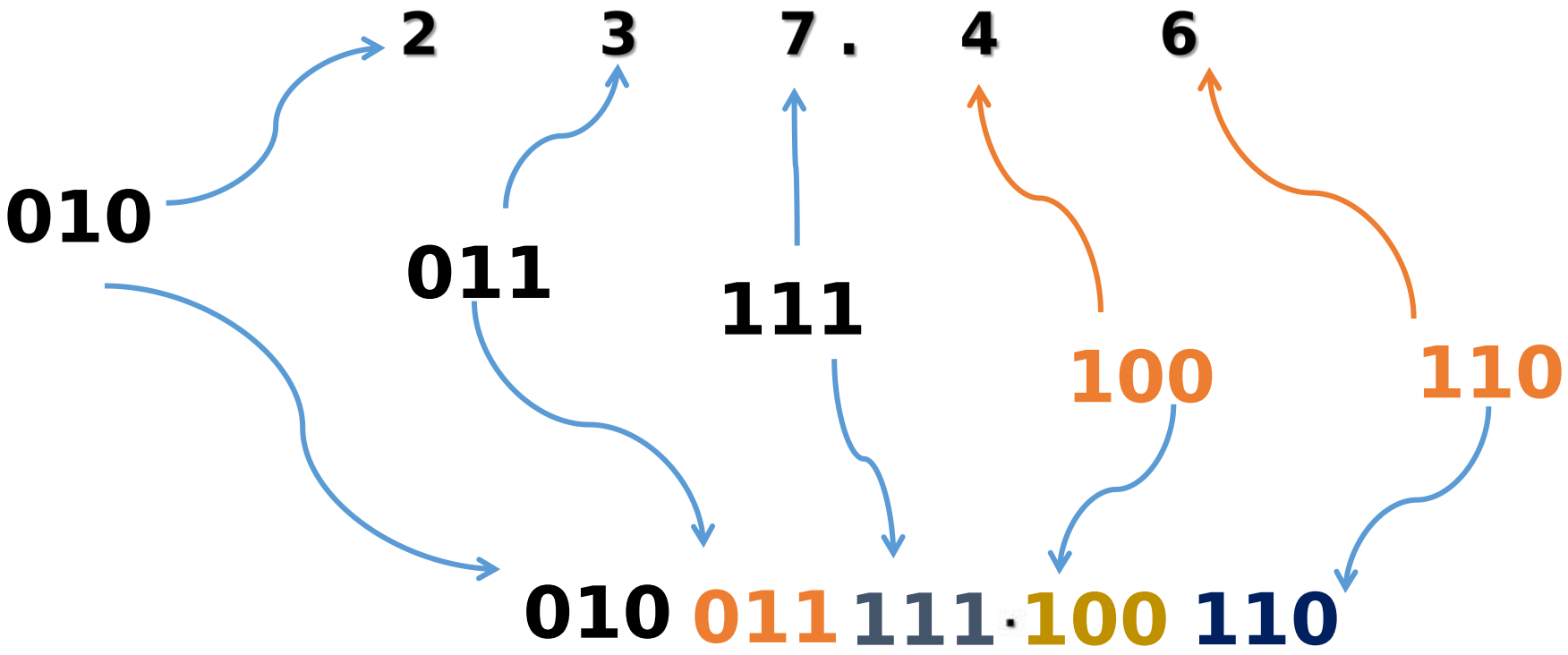
OCTAL TO BINARY CONVERSION

3-bit Binary Number	Octal Number
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

All Octal Numbers Can be represented by 3 bits

Case III -> 3. 2^3 TO 2^1 CONVERSION

$(237.46)_8$



$$\therefore (237.46)_8 = (010011111.100110)_2$$

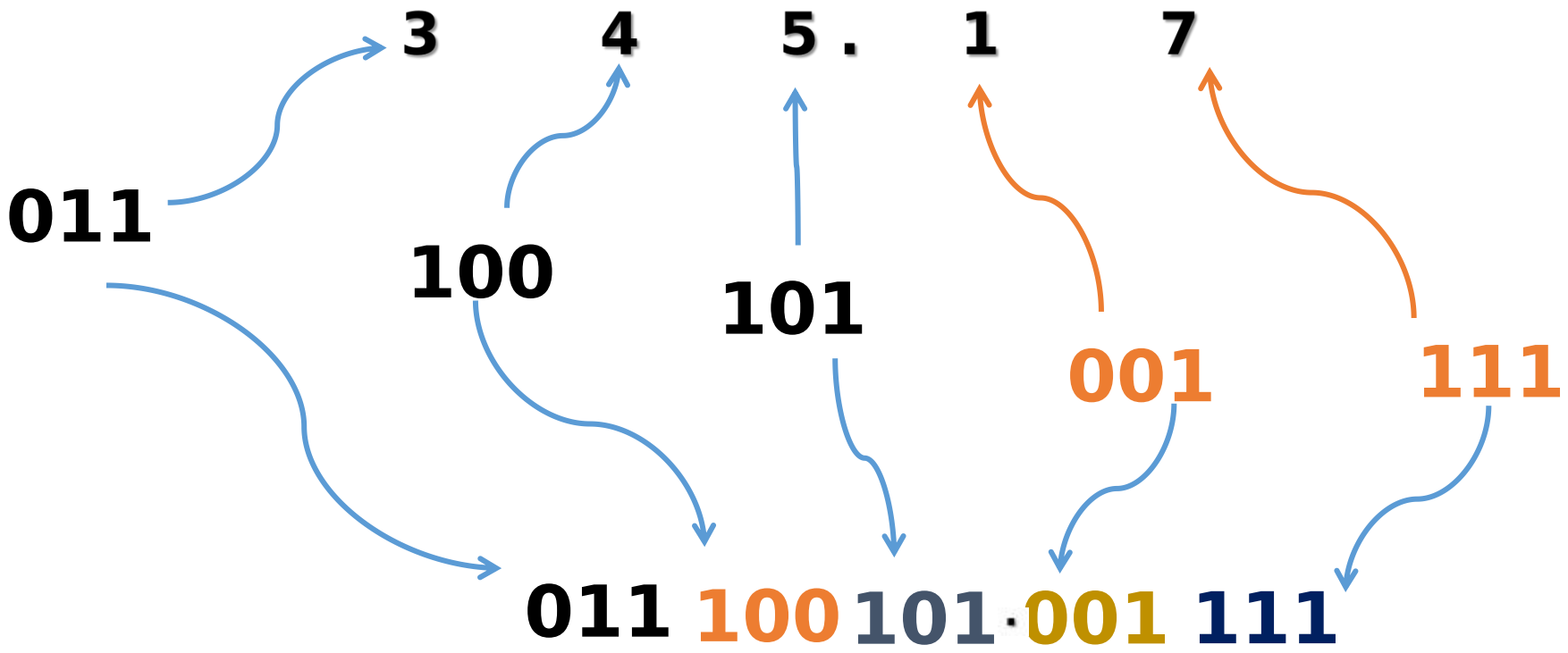
OCTAL TO BINARY CONVERSION

Case III -> 3. 2^3 TO 2^1 CONVERSION

Class Work: $(345.17)_8$

3-bit Binary Number	Octal Number
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

All Octal Numbers Can be represented by 3 bits



$$\therefore (345.17)_8 = (011100101.001111)_2$$

Case III -> 4. 2^4 TO 2^1 CONVERSION

$(XXXXXX)_{16}$ → $(XXXXXX)_2$

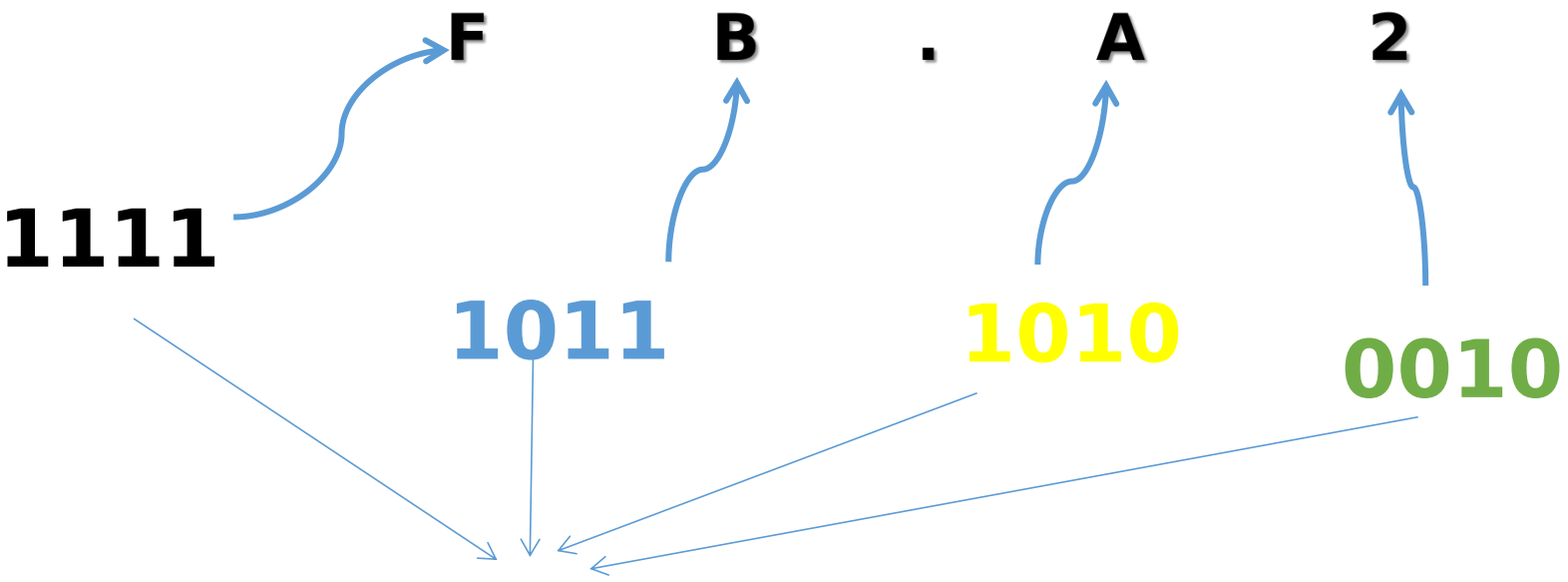
$(59.F6)_{16}$ → $(XXXXXX)_2$

Case III -> 4. 2^4 TO 2^1 CONVERSION

$(59.F6)_{16}$

Case III -> 4. 2^4 TO 2^1 CONVERSION

CLASS WORK : $(FB.A2)_{16}$



$\therefore (FB.A2)_{16} = (11111011.10100010)_2$

HEXA DECIMAL TO BINARY CONVERSION

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

All Hexa Decimal Numbers Can be represented by 4 bits

Case III -> 5. 2^3 TO 2^4 CONVERSION

$(XXXXXX)_8$ ⟶ $(XXXXXX)_{16}$

$(27.23)_8$ ⟶ $(XXXXXX)_{16}$

Step 1: Convert every octal digit to 3 binary digits

Step 2: convert every 4 binary digits to 1 hex digit

Case III -> 5. 2^3 TO 2^4 CONVERSION

$$(27.23)_8$$

Step 1: Convert every octal digit to 3 binary digits

2 7 . 2 3

010 111 010 011

= 010111.010011

Step 2: Convert every 4 binary digits to 1 hex digit

010111.010011

0001 0111.0100 1100

1 7 4 C

= 17.4C

$$\therefore (27.23)_8 = (17.4C)_{16}$$

Case III -> 6. 2^4 TO 2^3 CONVERSION

$(XXXXXX)_{16}$ → $(XXXXXX)_8$

$(17.4C)_{16}$ → $(XXXXXX)_8$

Step 1: Convert each hex digit to 4 binary digits
Step 2: convert each 3 binary digits to octal digits

Case III -> 6. 2^4 TO 2^3 CONVERSION

$(17.4C)_{16}$ ⟶ $(XXXXXX)_8$

Step 1: Convert each hex digit to 4 binary digits

$$(17.4C)_{16}$$

1	7	.	4	C
↑	↑		↑	↑
0001	0111		0100	1100

$$= 00010111.01001100$$

HEXA DECIMAL TO OCTAL CONVERSION

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

Case III -> 6. 2^4 TO 2^3 CONVERSION

$(17.4C)_{16}$ ⟶ $(XXXXXX)_8$

Step 1: Convert each hex digit to 4 binary digits

$$(17.4C)_{16}$$

1	7	.	4	C
0001	0111		0100	1100

$= 00010111.01001100$

Step 2: Convert each 3 binary digits to octal digits

$$00010111.01001100$$

$$000\ 010\ 111.010\ 011\ 000$$

0	2	7	2	3	0

$\therefore (17.4C)_{16} = (27.230)_{16}$

BINARY ARITHMETIC

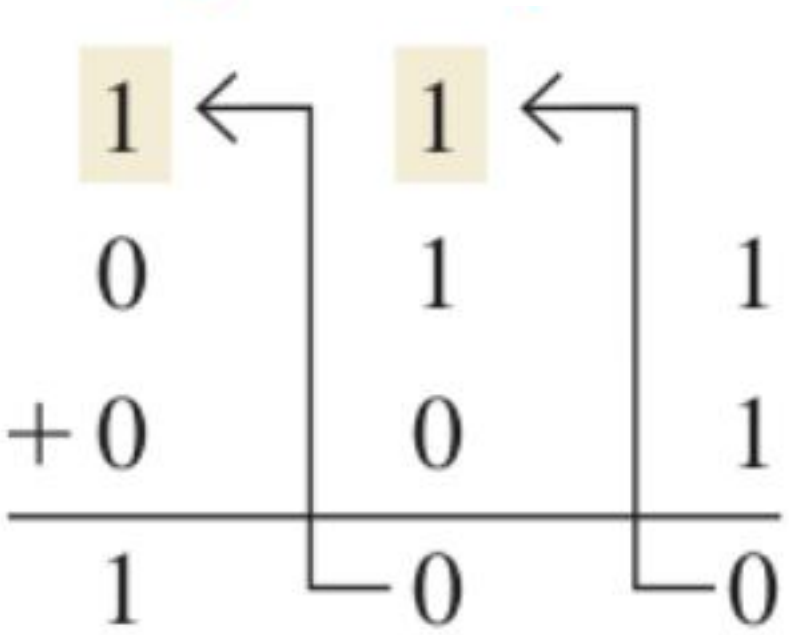
Binary arithmetic involves mathematical operations, such as addition, subtraction, multiplication, and division, performed on binary numbers, which use only the digits 0 and 1. In binary addition, the sum of two digits can be 0, 1, or 10 (carry). Subtraction involves borrowing, and multiplication and division follow similar principles to decimal arithmetic. Binary arithmetic is fundamental in computer science and digital electronics, where data is represented in binary format.

- Binary Addition
- Binary Subtraction
- Binary Multiplication
- Binary Division

Binary Addition

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0 (carry 1 to the next significant bit)

Binary Addition



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.

When there is a carry of 1, you have a situation in which three bits are being added (a bit in each of the two numbers and a carry bit).

BINARY ADDITION

Add the following binary numbers:

(a) 11 + 11

(b) 100 + 10

(c) 111 + 11

(d) 110 + 100

BINARY ADDITION

A + B	Sum	Carry
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

(a)

$$\begin{array}{r} 11 \\ +\ 11 \\ \hline \mathbf{110} \end{array}$$

(b)

$$\begin{array}{r} 100 \\ +\ 10 \\ \hline \mathbf{110} \end{array}$$

BINARY ADDITION

A + B	Sum	Carry
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

(c)

$$\begin{array}{r} {\scriptstyle 1\ 1} \\ 111 \\ +\ 11 \\ \hline \mathbf{1010} \end{array}$$

(d)

$$\begin{array}{r} {\scriptstyle 1} \\ 110 \\ +\ 100 \\ \hline \mathbf{1010} \end{array}$$

Binary Addition , Example 2

10001001 +10010101

Binary Addition

BINARY ARITHMETIC

A + B	Sum	Carry
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

10001001 +10010101

1

1 0 0 0 1 0 0 1

\+ 1 0 0 1 0 1 0 1

1 0 0 0 1 1 1 1 0

BINARY SUBSTRACTION

The four basic rules for subtracting bits are as follows:

$$0 - 0 = 0$$

$$1 - 1 = 0$$

$$1 - 0 = 1$$

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

BINARY SUBSTRACTION

- 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

BINARY SUBSTRACTION

Perform the following binary subtractions:

(a) 11 − 01 **(b)** 11 − 10

Input A	Input B	Subtract (S) A-B	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

BINARY SUBSTRACTION

Perform the following binary subtractions:

(a) 11 − 01 **(b)** 11 − 10

(a)

$$\begin{array}{r} 11 \\ -01 \\ \hline \mathbf{10} \end{array}$$

Input A	Input B	Subtract (S) A-B	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

BINARY SUBSTRACTION

Perform the following binary subtractions:

(a) 11 − 01 (b) 11 − 10

(b)

$$\begin{array}{r} 11 \\ -10 \\ \hline \mathbf{01} \end{array}$$

Input A	Input B	Subtract (S) A-B	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

No borrows were required in this example.

BINARY SUBSTRACTION

Subtract 011 from 101.

$$\begin{array}{r} 0 \\ 101 \\ -011 \\ \hline \mathbf{010} \end{array} \qquad \begin{array}{r} \\ 5 \\ -3 \\ \hline 2 \end{array}$$

Input A	Input B	Subtract (S) A-B	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

BINARY SUBSTRACTION

Subtract 011 from 101.

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 $10 - 1 = 1$.

$$\begin{array}{r} {\scriptstyle 0\ \ 1^{1}\ \ } \\ 1\,\not{1}\,01 \\ -\,0\ 1\ 1 \\ \hline 0\ 1\ 0 \end{array}$$

Right column:
$1 - 1 = 0$

BINARY MULTIPLICATION

The four basic rules for multiplying bits are as follows:

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

BINARY MULTIPLICATION

The four basic rules for multiplying bits are as follows:

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.

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

BINARY MULTIPLICATION

Perform the following binary multiplications:

(a) 11×11 (b) 101×111

A + B	Sum	Carry
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

(a)

$$\begin{array}{r} 11 \\ \times\ 11 \\ \hline {}_{1}11 \\ +11 \\ \hline \mathbf{1001} \end{array}$$

CARRY

Partial products

$$\begin{array}{r} 3 \\ \times\ 3 \\ \hline 9 \end{array}$$

BINARY MULTIPLICATION

Perform the following binary multiplications:

(a) 11×11 (b) 101×111

A + B	Sum	Carry
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

(b)

CARRY

$$\begin{array}{r} 111 \\ \times\ 101 \\ \hline 1\,111 \\ 1\,000\_ \\ +\,111\_\_ \\ \hline \mathbf{100011} \end{array} \qquad \begin{array}{r} 7 \\ \times\ 5 \\ \hline 35 \end{array}$$

Partial products

BINARY DIVISION

Division in binary follows the same procedure as division in decimal

Rules for Binary Division

Dividend	Divisor	Result
0	1	0
1	1	1
Division by zero is meaningless		

BINARY DIVISION

Perform the following binary divisions:

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

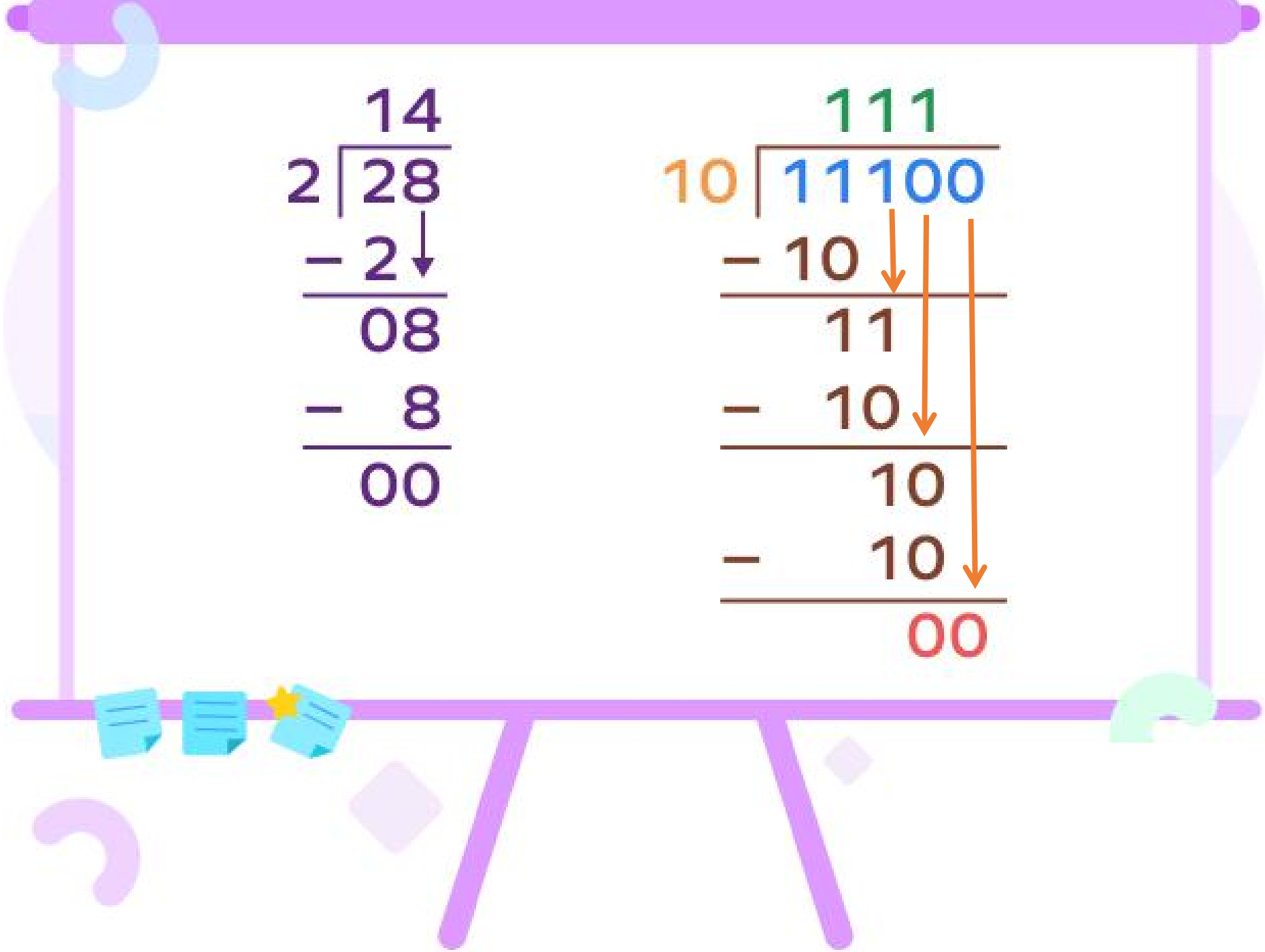
(a)

$$\begin{array}{r} \mathbf{10} \\ 11\overline{)110} \\ \underline{11} \\ 000 \end{array}$$

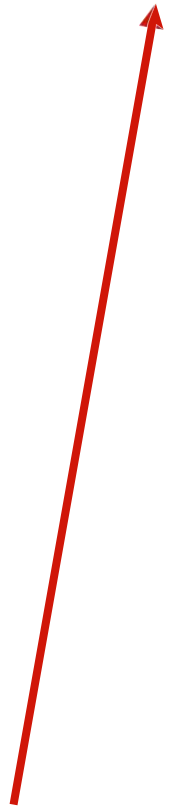
(b)

$$\begin{array}{r} 2 \\ 3\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

BINARY DIVISION



n	2^n
0	$2^0=1$
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
4	$2^4=16$
5	$2^5=32$
6	$2^6=64$
7	$2^7=128$



n	2^n	
8	$2^8=256$	
9	$2^9=512$	
10	$2^{10}=1024$	**Kilo**
11	$2^{11}=2048$	
12	$2^{12}=4096$	
20	$2^{20}=1M$	**Mega**
30	$2^{30}=1G$	**Giga**
40	$2^{40}=1T$	**Tera**

Representation of Signed Numbers

Digital systems, such as the computer, must be able to handle both positive and negative numbers.

A signed binary number consists of both sign and magnitude information.

The sign indicates whether a number is positive or negative, and the magnitude is the value of the number.

Representation of Signed Numbers

There are three forms in which signed integer (whole) numbers can be represented in binary:

sign-magnitude

1’s complement

2’s complement

Out of these, the 2’s complement is the most important and the sign-magnitude is the least used.

Noninteger and very large or small numbers can be expressed in floating-point format.

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.

Complements of Binary Numbers

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

1 0 1 1 0 0 1 0 Binary number

0 1 0 0 1 1 0 1 1’s complement

Complements of Binary Numbers

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.

i.e 2’s complement = (1’s complement) + 1

Note* : Add **1** to the 1’s complement to get the 2’s complement.

Complements of Binary Numbers

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.

Find the 2's complement of 10110010.

1	0	1	1	0	0	1	0	**Binary Number**
0	1	0	0	1	1	0	1	**1's Complement**
						+	1	

Complements of Binary Numbers

Finding the 2's Complement

Class Work : 2's complement of 11001011

1	1	0	0	1	0	1	1	Binary Number
0	0	1	1	0	1	0	0	1's Complement
						+	1	
0	0	1	1	0	1	0	1	→ 2's complement

Complements of Binary Numbers

Finding the 2's Complement

Alternative Method:

10111000

10111000 Binary number

1. Start at the right with the **LSB** and write the bits as they are up to and including the first 1.

2. Take the 1's complements of the remaining bits.

Complements of Binary Numbers

Finding the 2's Complement

Alternative Method:

10111000

10111000 Binary number

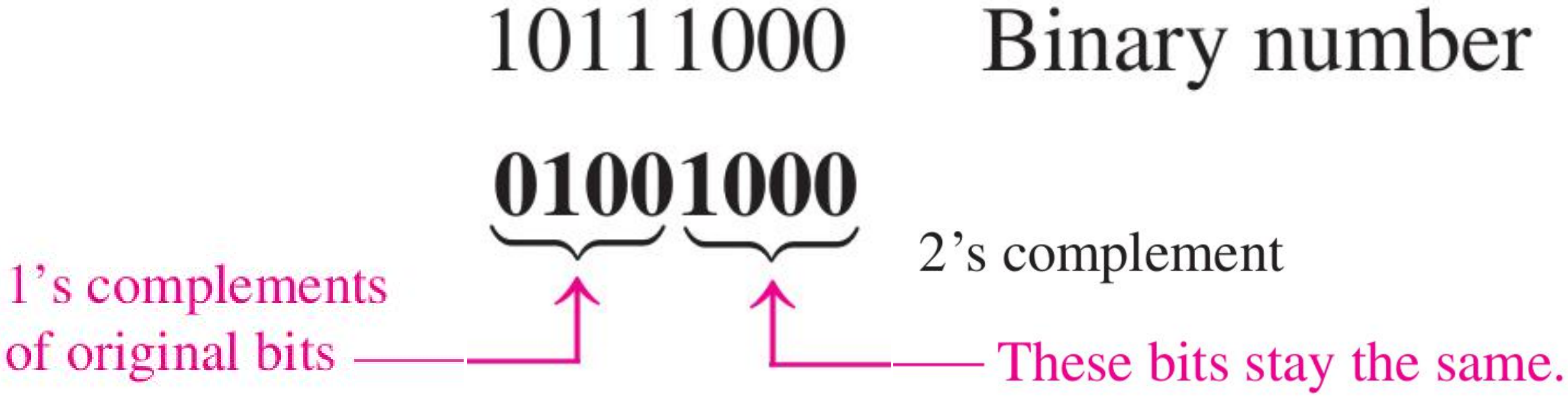
Start at the right with the **LSB**
and write the bits as they are up to and including the first 1.

Complements of Binary Numbers

Finding the 2's Complement

Alternative Method:

10111000



Assignments

1. Determine the 1's complement of each binary number:

(a) 00011010 **(b)** 11110111 **(c)** 10001101

2. Determine the 2's complement of each binary number:

(a) 00010110 **(b)** 11111100 **(c)** 10010001

Representation of Signed Numbers

The Sign Bit

The left-most bit in a signed binary number is the sign bit, which tells you whether the number is positive or negative.

A **0** sign bit indicates a **positive** number

A **1** sign bit indicates a **negative** number

Representation of Signed Numbers

Sign-Magnitude Form

When a signed binary number is represented in sign-magnitude, the left-most bit is the sign bit and the remaining bits are the magnitude bits.

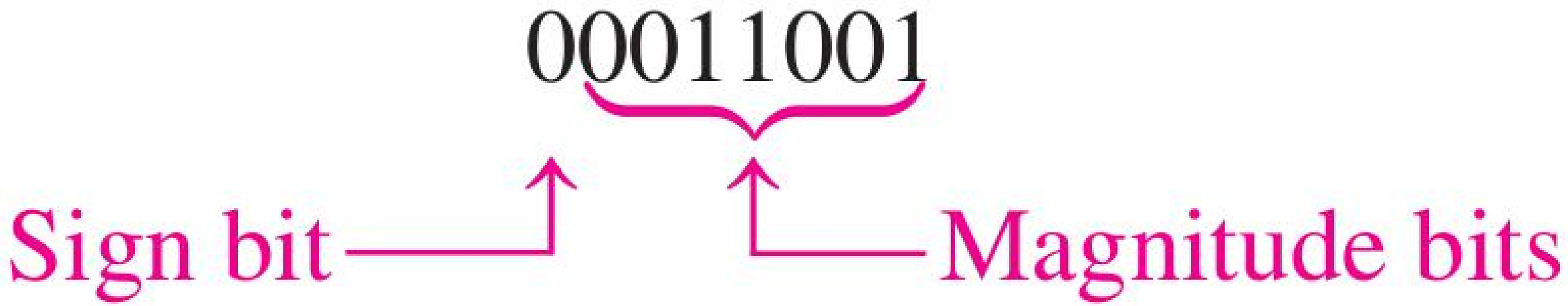
The magnitude bits are in true (uncomplemented) binary for both positive and negative numbers.

For example, the decimal number
+ 25 is expressed as an 8-bit signed binary number using the sign-magnitude form as

Representation of Signed Numbers

Sign-Magnitude Form

For example, the decimal number
+ 25 is expressed as an 8-bit signed binary number using the sign-magnitude form as



Representation of Signed Numbers

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 -25 is expressed as the 1’s complement of +25 .

+25 (00011001)

11100110

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

Representation of Signed Numbers

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 -25 and express it as the 2’s complement of +25

+25 (00011001)

11100110 → 1’s Complement

+1

11100111 → 2’s Complement

Representation of Signed Numbers

Express the decimal number −39 as an 8-bit number in the sign-magnitude, 1's complement, and 2's complement forms.

Solution

First, write the 8-bit number for +39.

00100111

In the *1's complement form*, −39 is produced by taking the 1's complement of +39

11011000

Representation of Signed Numbers

Express the decimal number −39 as an 8-bit number in the sign-magnitude, 1's complement, and 2's complement forms.

Solution

First, write the 8-bit number for +39.

$$00100111$$

In the *2's complement form,* −39 is produced by taking the 2's complement of +39

$$\begin{array}{rl} 11011000 & \text{1's complement} \\ + \quad\quad 1 & \\ \hline \mathbf{11011001} & \text{2's complement} \end{array}$$

Representation of Signed Numbers

Assignment

Express +19 and −19 as 8-bit numbers in sign-magnitude, 1's complement, and 2's complement.

Representation of Signed Numbers

The Decimal Value of Signed Numbers

Sign-Magnitude

Decimal values of positive and negative numbers in the sign-magnitude form are determined by summing the weights in all the magnitude bit positions where there are 1s and ignoring those positions where there are zeros.

The sign is determined by examination of the sign bit.

The Decimal Value of Signed Numbers

Sign-Magnitude

Determine the decimal value of this signed binary number expressed in sign-magnitude: 10010101.

Solution

The seven magnitude bits and their powers-of-two weights are as follows:

2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	0	1	0	1

Summing the weights where there are 1s,

$$16 + 4 + 1 = 21$$

The sign bit is 1; therefore, the decimal number is $\mathbf{-21}$.

ASSIGNMENT

Related Problem

Determine the decimal value of the sign-magnitude number 01110111.

Representation of Signed Numbers

The Decimal Value of Signed Numbers

1’s Complement

Decimal values of positive numbers in the 1’s complement form are determined by summing the weights in all bit positions where there are **1s** and ignoring those positions where there are zeros.

Decimal values of negative numbers are determined by assigning a negative value to the weight of the sign bit, summing all the weights where there are **1s**, and adding **1** to the result.

The Decimal Value of Signed Numbers

1's Complement

Determine the decimal values of the signed binary numbers expressed in 1's complement:

(a) 00010111 **(b)** 11101000

Solution

(a) The bits and their powers-of-two weights for the positive number are as

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	1	0	1	1	1

Summing the weights where there are 1s,

$$16 + 4 + 2 + 1 = \mathbf{+23}$$

The Decimal Value of Signed Numbers

1's Complement

Determine the decimal values of the signed binary numbers expressed in 1's complement:

(a) 00010111 **(b)** 11101000

Solution

(b) The bits and their powers-of-two weights for the negative number are as Notice that the negative sign bit has a weight of -2^7 or -128.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	0	0

Summing the weights where there are 1s,

$$-128 + 64 + 32 + 8 = -24$$

Adding 1 to the result, the final decimal number is

$$-24 + 1 = \mathbf{-23}$$

ASSIGNMENT

Related Problem

Determine the decimal value of the 1’s complement number 11101011.

Representation of Signed Numbers

The Decimal Value of Signed Numbers

2’s Complement

Decimal values of **positive and negative** numbers in the 2’s complement form are determined by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros.

The weight of the sign bit in a negative number is given a negative value.

The Decimal Value of Signed Numbers

2's Complement

Determine the decimal values of the signed binary numbers expressed in 2's complement:

(a) 01010110 **(b)** 10101010

Solution

(a) The bits and their powers-of-two weights for the positive number are as follows:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	1	0	1	1	0

Summing the weights where there are 1s,

$$64 + 16 + 4 + 2 = \mathbf{+86}$$

The Decimal Value of Signed Numbers

2's Complement

Determine the decimal values of the signed binary numbers expressed in 2's complement:

(a) 01010110 **(b)** 10101010

Solution

(b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of $-2^7 = -128$.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	1	0	1	0

Summing the weights where there are 1s,

$$-128 + 32 + 8 + 2 = \mathbf{-86}$$

ASSIGNMENT

Related Problem

Determine the decimal value of the 2’s complement number 11010111.

The Decimal Value of Signed Numbers

From these examples, you can see why the 2's complement form is preferred for representing signed integer numbers:

To convert to decimal, it simply requires a summation of weights regardless of whether the number is positive or negative.

The 1's complement system requires adding 1 to the summation of weights for negative numbers but not for positive numbers.

Also, the 1's complement form is generally not used because two representations of zero (00000000 or 11111111) are possible.

The Decimal Value of Signed Numbers

Now, when discussing the 1's complement representation, the issue arises with the representation of zero.

In a typical binary system, zero is represented as all zeros. However, in 1's complement, there are two possible representations for zero:

All zeros: This is the standard representation of zero in binary, where all the bits are set to 0 (e.g., 00000000).

All ones: In 1's complement, you can represent zero by having all bits set to 1 as well (e.g., 11111111).

The Decimal Value of Signed Numbers

The presence of two representations for zero can lead to confusion and inconsistency.

Depending on the context and the specific implementation, it might not be clear which representation is intended.

This ambiguity makes 1's complement less convenient and more error-prone compared to other binary representations, such as 2's complement.

Floating-Point Numbers

A floating-point number (also known as a real number) consists of two parts plus a sign.

The **mantissa** is the part of a floating-point number that represents the magnitude of the number and is between 0 and 1.

The **exponent** is the part of a floating-point number that represents the number of places that the decimal point (or binary point) is to be moved.

Floating-Point Numbers

A decimal example will be helpful in understanding the basic concept of floating-point numbers.

Let’s consider a decimal number which, in integer form, is 241,506,800.

The mantissa is .2415068 and the exponent is 9.

Floating-Point Numbers

A decimal example will be helpful in understanding the basic concept of floating-point numbers.

Let's consider a decimal number which, in integer form, is 241,506,800.

The mantissa is .2415068 and the exponent is 9.

When the integer is expressed as a floating-point number, it is normalized by moving the decimal point to the left of all the digits so that the mantissa is a fractional number and the exponent is the power of ten.

The Decimal Value of Signed Numbers

Floating-Point Numbers

A decimal example will be helpful in understanding the basic concept of floating-point numbers.

Let’s consider a decimal number which, in integer form, is 241,506,800.

The mantissa is .2415068 and the exponent is 9.

When the integer is expressed as a floating-point number,it is normalized by moving the decimal point to the left of all the digits so that the mantissa is a fractional number and the exponent is the power of ten.

The floatingpoint number is written as

$$0.2415068 \times 10^{9}$$

Single-Precision Floating-Point Binary Numbers

In the standard format for a single-precision binary number, the sign bit (S) is the left-most bit, the exponent (E) includes the next eight bits, and the mantissa or fractional part (F) includes the remaining 23 bits, as shown next.

Single-Precision Floating-Point Binary Numbers

To illustrate how a binary number is expressed in floating-point format, let's use **101101001000 1** as an example.

First, it can be expressed as 1 plus a fractional binary number by moving the binary point 12 places to the left and then multiplying by the appropriate power of two.

$$101101001000 1$$

$$= 1.011010010001 \times 2^{12}$$

Single-Precision Floating-Point Binary Numbers

$$1011010010001$$

$$= 1.011010010001 \times 2^{12}$$

Assuming that this is a positive number, the sign bit (S) is 0.

The exponent, 12, is expressed as a biased exponent by adding it to 127 (12 + 127 = 139).

The biased exponent (E) is expressed as the binary number 10001011.

The mantissa is the fractional part (F) of the binary number, **.011010010001**.

Because there is always a 1 to the left
of the binary point in the power-of-two expression,it is not included in the mantissa.

The complete floating- point number is :

S	E	F
0	10001011	01101001000100000000000

Representation of BCD or 8421

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,vso 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.

Representation of BCD or 8421

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

Representation of BCD

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

Invalid Codes

1011 1101 1111

1010

1100

1110

Representation of BCD

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

Convert each of the following decimal numbers to BCD:

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

Representation of BCD

Decimal/BCD conversion.

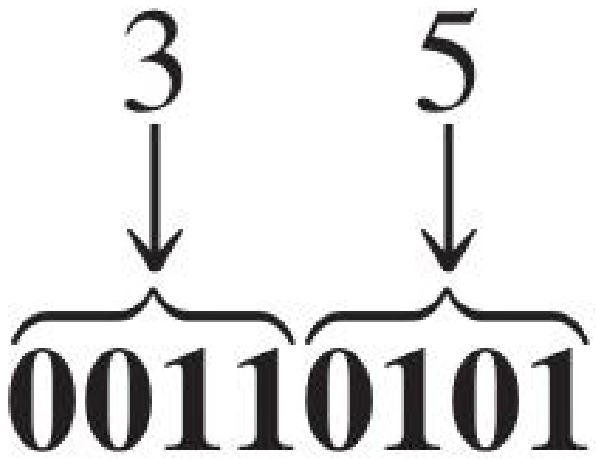
Decimal Digit	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Convert each of the following decimal numbers to BCD:

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

Solution

(a)



Representation of BCD

Decimal/BCD conversion.

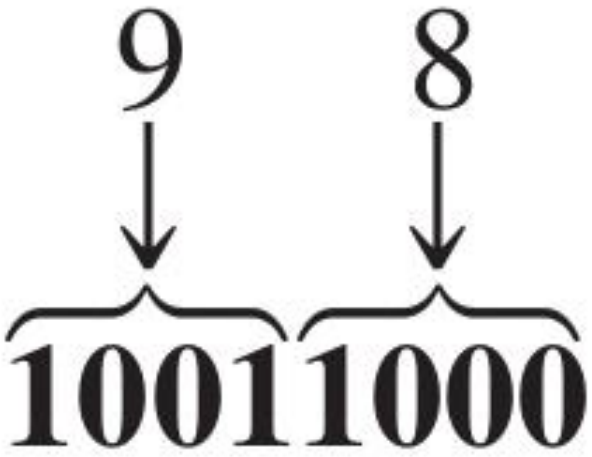
Decimal Digit	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Convert each of the following decimal numbers to BCD:

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

Solution

(b) 9 8



Representation of BCD

Decimal/BCD conversion.

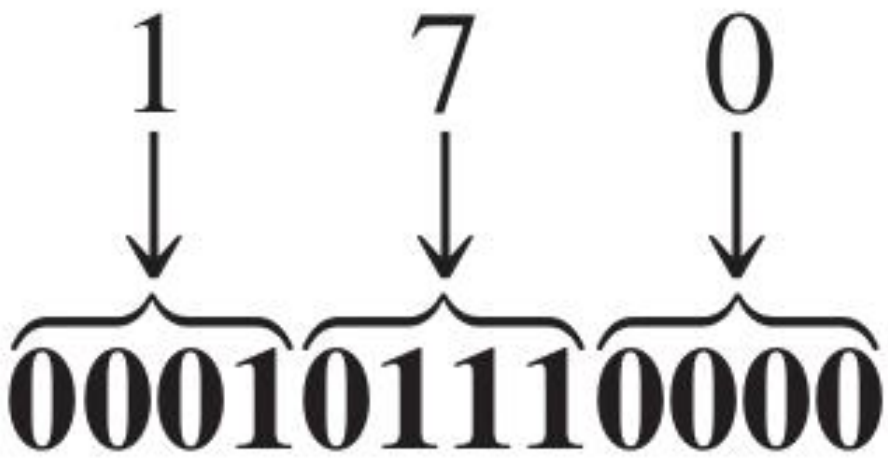
Decimal Digit	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Convert each of the following decimal numbers to BCD:

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

Solution

(c)



Representation of BCD

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

Convert each of the following decimal numbers to BCD:

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

Solution

(d) 2 4 6 9

0010 0100 0110 1001

Representation of BCD

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

Related Problem

Convert the decimal number 9673 to BCD.

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.

Representation of BCD

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

Convert each of the following BCD codes to decimal:

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

Solution

(a) 10000110

Representation of BCD

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

Convert each of the following BCD codes to decimal:

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

Solution

(a) $\underbrace{1000}_{8}\underbrace{0110}_{6}$

Representation of BCD

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

Convert each of the following BCD codes to decimal:

(a) 10000110 (b) 001101010001 (c) 100101000111000

Solution

(b) 0011 0101 0001

↓ ↓ ↓

3 5 1

Representation of BCD

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

Convert each of the following BCD codes to decimal:

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

Solution

(c) 1001 0100 0111 0000

↓ ↓ ↓ ↓

9 4 7 0

Representation of BCD

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

Convert the BCD code

1000001000100111110110

to decimal.

Assignment

ASCII

Pronounced “askee,”

American Standard Code for Information Interchange

ASCII is a universally accepted alphanumeric code used in most computers and other electronic equipment.

Most computer keyboards are standardized with the **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.

ASCII

Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char
32	space	52	4	72	H	92	\	112	p
33	!	53	5	73	I	93	]	113	q
34	"	54	6	74	J	94	^	114	r
35	#	55	7	75	K	95	_	115	s
36	$	56	8	76	L	96	`	116	t
37	%	57	9	77	M	97	a	117	u
38	&	58	:	78	N	98	b	118	v
39	'	59	;	79	O	99	c	119	w
40	(	60	<	80	P	100	d	120	x
41	)	61	=	81	Q	101	e	121	y
42	*	62	>	82	R	102	f	122	z
43	+	63	?	83	S	103	g	123	{
44	,	64	@	84	T	104	h	124	|
45	-	65	A	85	U	105	i	125	}
46	.	66	B	86	V	106	j	126	~
47	/	67	C	87	W	107	k		
48	0	68	D	88	X	108	l		
49	1	69	E	89	Y	109	m		
50	2	70	F	90	Z	110	n		
51	3	71	G	91	[	111	o		

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.

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

The Gray Code

Binary-to-Gray Code Conversion

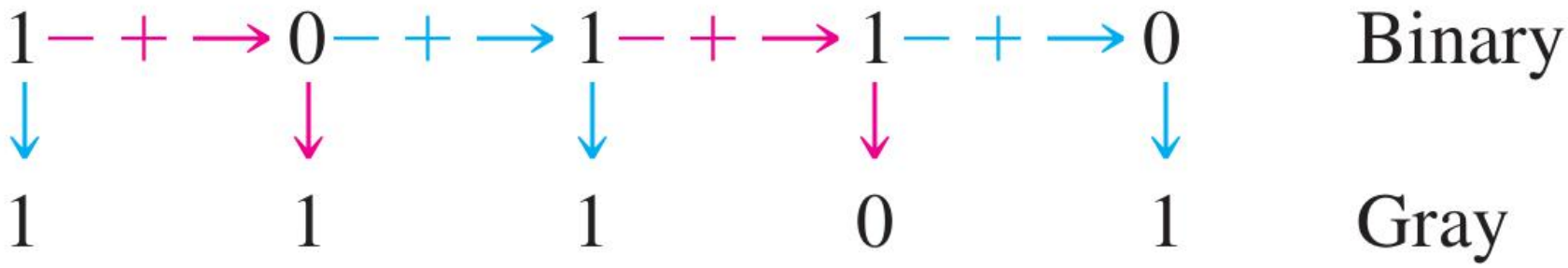
The following rules explain how to convert from a binary number to a Gray code word:

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

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

Binary-to-Gray Code Conversion

For example, the conversion of the binary number **10110** to Gray code is as follows:



The Gray code is 11101.

Gray-to-Binary Code Conversion

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

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

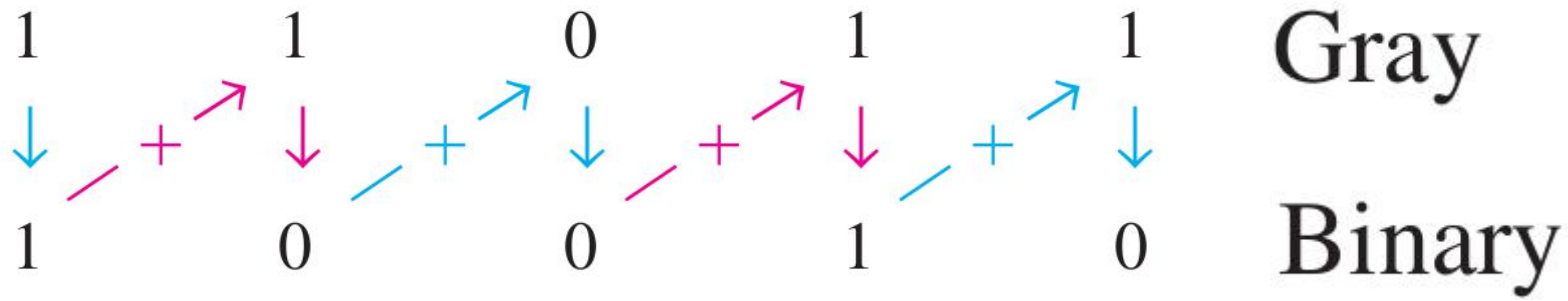
Discard carries.

Gray-to-Binary Code Conversion

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

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

For example, the conversion of the Gray code word **11011** to binary is as follows:



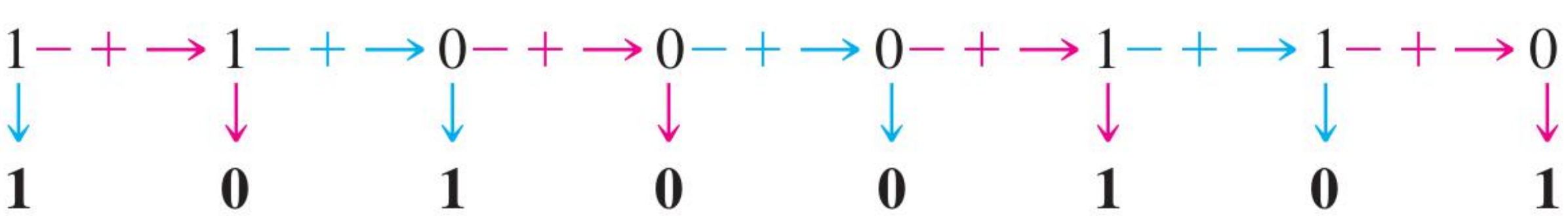
The binary number is **10010**.

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

(b) Convert the Gray code 10101111 to binary.

Solution

(a) Binary to Gray code:

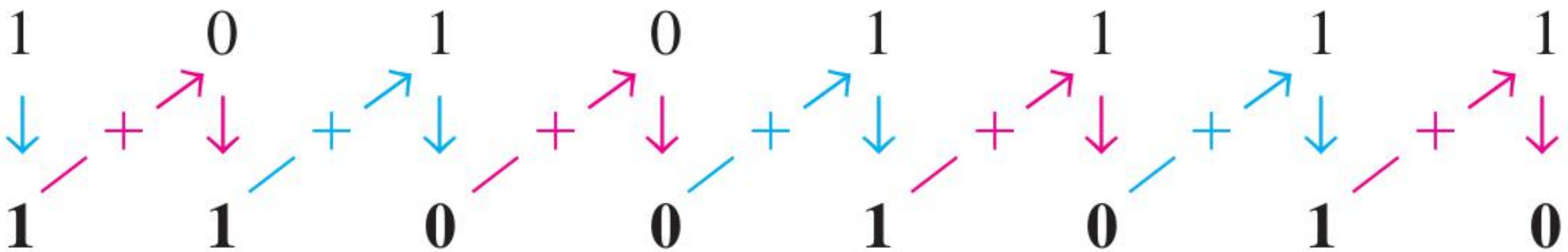


(a) Convert the binary number 11000110 to Gray code.

(b) Convert the Gray code 10101111 to binary.

Solution

(b) Gray code to binary:



Related Problem

(a) Convert *binary* **101101** to *Gray* code.

(b) Convert *Gray* code **100111** to *binary*.

ASSIGNMENT

Error Codes

Parity Method for Error Detection

Many systems use a parity bit as a means for bit error detection.

Any group of bits contain either an even or an odd number of 1s.

A parity bit is attached to a group of bits to make the total number of 1s in a group always even or always odd. An even parity bit makes the total number of 1s even, and an odd parity bit makes the total odd.

A given system operates with even or odd **parity**, but not both. For instance, if a system operates with even parity, a check is made on each group of bits received to make sure the total number of 1s in that group is even. If there is an odd number of 1s, an error has occurred.

Error Codes

Parity Method for Error Detection

The parity bit can be attached to the code at either the beginning or the end, depending on system design. Notice that the total number of 1s, including the parity bit, is always even for even parity and always odd for odd parity.

The BCD code with parity bits.

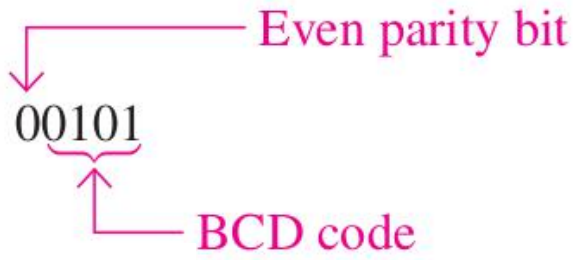
Even Parity		Odd Parity	
P	BCD	*P*	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

Error Codes

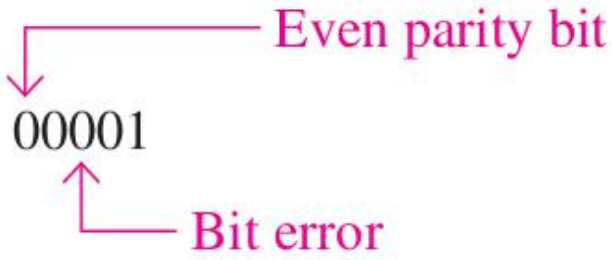
Parity Method for Error Detection

Detecting an Error

A parity bit provides for the detection of a single bit error (or any odd number of errors, which is very unlikely) but cannot check for two errors in one group. For instance, let's assume that we wish to transmit the BCD code 0101. (Parity can be used with any number of bits; we are using four for illustration.) The total code transmitted, including the even parity bit, is



Now let's assume that an error occurs in the third bit from the left (the 1 becomes a 0).



When this code is received, the parity check circuitry determines that there is only a single 1 (odd number), when there should be an even number of 1s. Because an even number of 1s does not appear in the code when it is received, an error is indicated.

Assign the proper even parity bit to the following code groups:

(a) 1010 **(b)** 111000 **(c)** 101101

(d) 1000111001001 **(e)** 101101011111

Solution

Make the parity bit either 1 or 0 as necessary to make the total number of 1s even. The parity bit will be the left-most bit (color).

(a) **0**1010 **(b)** **1**111000 **(c)** **0**101101

(d) **0**1000111001001 **(e)** **1**101101011111

Error Codes

Parity Method for Error Detection

Detecting an Error

An odd parity system receives the following code groups:

10110,
11010,
110011,
110101110100,
and **1100010101010**.
Determine which groups, if any, are in error.

Solution

Since odd parity is required, any group with an even number of 1s is incorrect. The following groups are in error: **110011** and **1100010101010**.

Excess-3 Code

The **excess-3** representation of a decimal digit **d** can be obtained by adding 3 to its value.

The **excess-3** code is an unweighted code because its value is obtained by adding three to the corresponding binary value.