1. Number Systems and codes

Number Systems

Decimal number: A number system relates quantities and symbols. We are familiar with the decimal number system which uses 10 digits.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

The decimal number system is also known as base-10 system since there are 10 digits. The base or radix of the system tells us how many digits the number system uses. The base is indicated by a subscript to the number. The number 278 stands for the following representation. The number 278 can be expanded as

$$278_{10} = (2 \times 10^2) + (7 \times 10^1) + (8 \times 10^0)$$

The individual digits in the number represent the coefficient (2, 7, 8 in an expansion) of the number in powers of 10. The right most digit is the coefficient of zeroth power of 10 and the next is the coefficient of first power of 10 and so on.

Binary number : Modern computers do not process decimal numbers. Instead, they work with binary which uses only numerals 0 and 1. In binary number system, the base is 2 and the numerals 0 and 1 are used to represent a number. ie., 0 and 1 are the binary digits.

In the binary system, the individual digits represent the coefficient of powers of 2. For example, the decimal number 13 is expanded as

$$13 = 8 + 4 + 0 + 1$$

$$13 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$1 \quad 1 \quad 0 \quad 1$$

The coefficients in the expansion are 1101

$\therefore 13_{10} = 1101_2$

The number 13 in decimal system is equivalent to 1101 in the binary number system. One can see that $14_{10} = 1110_2$ and $12_{10} = 1100_2$.

Why binary numbers are used in computers?

The working of almost all digital computers and systems are based on binary (two-state) operation. For instance, a switch can be open (0) or closed (1). Therefore a switch is one example of a natural binary device. A magnetic core is another example of two - state operation (clock-wise magnetisation and counter-clock wise magnetisation). A punched card is also an example of two-state concept. A hole in the card is one posibility and no hole is the other. A transistor can be in cut-off state (off) or in saturated state (on).

Thus, switches, punched cards, magnetic tape, transistors and all other digital components are based on binary operation. The binary number system is less complicated than the decimal system because it is composed of only two digits. This is why it is convenient to use binary numbers when analysing or designing digital circuits.

Conversion of Decimal Number into Binary Number

(Using double dabble method)

We progressively divide the given decimal number by 2, till a quatient 0 is reached. The reminder of each dividion is written on the right. The reminders taken from down to up, give the binary number.

For example , to convert the decimal number 29 into binary number:

2	29		↑
2	14	- 1	↑
2	7	- 0	↑ Read
2	3	- 1	↑ up
2	1	- 1	↑
	0	- 1	↑

As the qutient 0 is reached, the conversion is finished. Now the reminders are read upward to get the binary equivalent. Thus,

decimal number 29 = 11101 binary number

A binary digit (a one or a zero) is called a bit. A group of bits having a significance is a byte or code. A byte is sometimes called a character and a group of one or more characters is a word. When binary number has 4 bits, it is called a nibble. A binary number with a string of 8 bits is known as a byte.

bit = x

nibble = xxxx

byte = xxxxxxxx

Equivalent numbers in decimal and binary notation

Decimal notation	Binary notation	Decimal notation	Binary notation
0	00000	11	01011
1	00001	12	01100
2	00010	13	01101
3	00011	14	01110
4	00100	15	01111
5	00101	16	10000
6	00110	17	10001
7	00111	18	10010
8	01000	19	10011
9	01001	20	10100
10	01010	21	10101

Whenever a binary number has all 1s (consisting of only 1s), we can find its decimal equivalent by using the formula:

Decimal = $(2^n - 1)$ where n is the number of bits in the binary number. For example, 1111 has four bits.

Decimal equivalent = $2^4 - 1 = 16 - 1 = 15$

Again, the binary 1111 1111 has eight digits.

The decimal equivalent = $2^8 - 1 = 256 - 1 = 255$

A decimal fraction may be converted into binary as follows:-

Consider the decimal fraction 0.85. This is multiplied by 2 and the carry is recorded in integer position.

0.85 × 2 = 1.7 = 0.7	with a carry of	1	
0.7 × 2 = 1.4 = 0.4	with a carry of	1	
0.4 × 2 = 0.8 = 0.8	with a carry of	0	Read down
0.8 × 2 = 1.6 = 0.6	with a carry of	1	
0.6 × 2 = 1.2 = 0.2	with a carry of	1	
0.2 × 2 = 0.4 = 0.4	with a carry of	0	

Reading the carry downwards gives the binary fraction as 0.110110

Thus, decimal fraction 0.85 = 0.110110 binary.

In this case, we have stopped the conversion process after getting six binary digits. Due to this, the answer is only approximate. If more accuracy is needed, we continue multiplying by 2 until we have as many digits as necessary for the application.

Example

1. Convert decimal 23.6 to binary number.

First we split the decimal 23.6 into an integer 23 and a fraction 0.6. Then, we apply the double dabble method to each part. Decimal 23 = binary 10111.

2	23	
2	11	– 1
2	5	– 1
2	2	– 1
2	1	– 0
	0	– 1

Read up

For 0.6, we proceed as below:

$0.6 \times 2 = 1.2 = 0.2$ with a carry of 1
$0.2 \times 2 = 0.4 = 0.4$ " 0
$0.4 \times 2 = 0.8 = 0.8$ " 0
$0.8 \times 2 = 1.6 = 0.6$ " 1
$0.6 \times 2 = 1.2 = 0.2$ " 1

Read down

Thus, the decimal 23.6 = 10111.10011 binary.

2. *Convert the decimal number 37. 31 into 10-bit binary .*

$[37]_{10} = 1 \times 2^5 + 1 \times 2^2 + 1 \times 2^0$

$= [100101]_2$

$0.31 \times 2 = 0.62 - 0$
$0.62 \times 2 = 1.24 - 1$
$0.24 \times 2 = 0.48 - 0$
$0.48 \times 2 = 0.96 - 0$

$\therefore [0.31]_{10} = [0.0100]_2$

$\therefore [37.31]_{10} = [100101.0100]_2$.

Ans : $[100101.0100]_2$.

Problem

1. Convert a 32-bit binary number that has all 1 s into decimal equivalent number [*Ans:* 4, 294, 967, 295]

2. Convert $(6.215)_{10}$ into 10-bit binary. [*Ans:* 110.0011011_2]

Binary - Decimal Conversion

We can write any given binary number into its equivalent decimal number. The given binary number is expressed in terms of the weight corresponding to each binary digit position. The least significant digit (the one on the right most) has a weight of 1. The second position from the right has a weight of 2; the next one has weight of 4 and then 8,16,32 and so on. These weights are in the ascending powers of 2. The sum of the individual weights is the decimal equivalent of the given binary number. For example, given the binary 10011, what is the decimal equivalent?

$$\begin{aligned} 10011 &= (1\times 2^4)+(0\times 2^3)+(0\times 2^2)\,(1\times 2^1)+(1\times 2^0) \\ &= (1\times 16)+(0\times 8)+(0\times 4)+(1\times 2)+(1\times 1) \\ &= 16+0+0+2+1 \\ &= 19 \end{aligned}$$

binary 10011 = decimal 19.

The above procedure is streamlined as follows:

the binary is	1	0	0	1	1
the weights are	16	8	4	2	1
Scoring off for zero	16			2	1

Then, 16 + 2 + 1 = 19

Example : Convert binary 10101 into the equivalent decimal

The given binary is	1	0	1	0	1
	16	8	4	2	1
Scoring off for zero	16		4		1

Then, $16 + 4 + 1 = 21$

binary 10101 = decimal 21.

Binary fraction to decimal : Let the given binary fraction be 0.1011. To find the decimal equivalent, the weights of the binary digit positions to the right of the binary point are written as

$\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$ and so on.

In powers of 2, the weights are $\frac{1}{2^1}, \frac{1}{2^2}, \frac{1}{2^3}, \frac{1}{2^4}$ and so on.

i.e., $2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}$, and so on

In decimal form, these are

0.5, 0.25, 0.125, 0.0625 etc.

Thus, the given fraction is 0. 1 0 1 1

↓ ↓ ↓ ↓

0.5 0.25 0.125 0.0625

The decimal equivalent is $0.5 + 0 + 0.125 + 0.0625 = 0.6875$.

Example : *Find decimal equivalent of* 101.1101.

For mixed numbers like this, we handle integer part and fractional part separately.

1	0	1	.	1	1	0	1
2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}
2^2		2^0	.	$\frac{1}{2}$	$\frac{1}{4}$		$\frac{1}{16}$
4		+1	.	0.5 + 0.25		+ 0.0625	
		5	.	8125			

binary 101.1101 = 5.8125 decimal

Binary addition

Addition means combining of two or more physical quantities. For example $3 + 2 = 5$ represents combining of three things with two things to obtain a total of five things.

Computer circuits do not handle decimal numbers. They process binary numbers. Binary addition is the key to binary subtraction, multiplication and division. The most basic cases of binary addition are

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 10$ One plus one equals one zero (zero carry one)

Example

1. *Add binary 10 (one zero) with binary 10 (one zero)*

$$\begin{array}{r} 1\ 0 \\ +\ 1\ 0 \\ \hline 1\ 0\ 0 \end{array}$$

Adding the first column on the right, zero plus zero is zero. In the second column, one plus one is zero; placing zero in the same column and carry 1 to the next higher column, we get 100 (one zero zero)

2. *Find* $1 + 1 + 1$ *in binary.*

First add $1 + 1$ as 10. Now add the remaining 1 with this result.

i.e.,
$$\begin{array}{r} 1\ 0 \\ +\ 1 \\ \hline 1\ 1 \end{array}$$ (one one)

3. *Find* 1 + 1 + 1 + 1 *in binary*

1 + 1 = 10

10 + 1 = 11

$$\begin{array}{r} 1\ 1\ + \\ 1 \\ \hline 1\ 0\ 0 \end{array}$$

One plus one equals 10 giving 0 in the first column and carry one to the second column. The carry one and the one in the second column gives 10, giving 0 in the second column and 1 in the third column.

4. *Add the following binary numbers* 1010 *and* 1011.

$$\begin{array}{r} 1\ 0\ 1\ 0 \\ +\ 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 1\ 0\ 1 \end{array}$$

using column-by-column addition

$\therefore 1010 + 1011 = 10101$

5. *Add* 1111 *and* 1011 *in binary form*

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ 1\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 1\ 0 \end{array}$$

6. *Find the result of binary addition* 101 + 110 +111

$$\begin{array}{r} 1\ 0\ 1 \\ 1\ 1\ 0 \\ 1\ 1\ 1 \\ \hline 1\ 0\ 0\ 1\ 0 \end{array}$$

Alternately, we first add 101 and 110 and then add the result obtained with 111.

$$\begin{array}{r} 1\ 0\ 1 \\ +\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 1 \\ 1\ 1\ 1 \\ \hline 1\ 0\ 0\ 1\ 0 \\ \hline \end{array}$$

Binary subtraction

For binary subtraction the following rules can be followed.

$0 - 0 = 0$

$1 - 0 = 1$

$1 - 1 = 0$

$10 - 1 = 1$ (one zero minus one equals one)

• • − • = •

For large binary numbers, subtract column by column, borrowing from the next higher column, if necessary.

Example

1. *Subtract* 1010 *from* 1111 *in binary*

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ 1\ 0\ 1\ 0 \\ \hline 0\ 1\ 0\ 1 \\ \hline \end{array}$$

Ans : 1 0 1

2. *Substarct* 110 *from* 1001 *in binary*

$$\begin{array}{r} 1\ 0\ 0\ 1 \\ -\ 0\ 1\ 1\ 0 \\ \hline 0\ 0\ 1\ 1 \\ \hline \end{array}$$

First column : $1 - 0 = 1$

second column : 10 (after borrowing form left)

think that it is $2 - 1 = 1$

third column : $1 - 1 = 0$

fourth column : $0 - 0 = 0$

3. *Subtract* 111 *from* 100 *in binary*

$$\begin{array}{r} 1\ 0\ 0 \\ -\ 1\ 1\ 1 \\ \hline -\ 0\ 1\ 1 \end{array}$$

We subtract smaller numbers from the larger numbers and prefix the sign of the larger number in the result.

i.e., $\begin{array}{r} 1\ 1\ 1 \\ -\ 1\ 0\ 0 \\ \hline -\ 0\ 1\ 1 \end{array}$

Since the larger number has negative sign, the result is negative.

$100 - 111 = -011$

4. *Subract one from* 100_2.

$$\begin{array}{r} 1\ 0\ 0 \\ -\ \ \ 1 \\ \hline 1\ 1_2 \end{array}$$

Ans : 11_2

1's and 2's complement (with unsigned binary numbers)

Complement of one (1) is zero (0). Complement of zero (0) is one (1).1's complement of a binary number is the number that results when we change each zero (0) to one (1) and each one (1) to zero (0).

A = 0101 has a 1's complement $\overline{A}$ = 1010

A = 1001 has a 1's complement $\overline{A}$ = 0110

The 2's Complement is the binary number that results when we add 1 to the 1's complement. 2's complement = 1's complement +1

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

where A' = 2's complement; $\overline{A}$ = 1's complement.

1's complement of −01101 = 10010.

Adding this with 11011,

$$\begin{array}{r} 1\ 1\ 0\ 1\ 1 \\ +\ 1\ 0\ 0\ 1\ 0 \\ \hline 1\ 0\ 1\ 1\ 0\ 1 \\ \hookrightarrow\ 1 \\ \hline 0\ 1\ 1\ 1\ 0 \end{array}$$

0 1 1 1 0 is the answer

2. *Use 1's complement method to perform M − N with M* = 1000100 *and N* = 1010100

$$\begin{array}{rcl} M & = & 1\ 0\ 0\ 0\ 1\ 0\ 0 + \\ \overline{N} & = & 0\ 1\ 0\ 1\ 0\ 1\ 1 \\ \hline & & 1\ 1\ 0\ 1\ 1\ 1\ 1 \end{array}$$

with no carry. ∴ The result is negative. To get the result, we have to take 1's complement of the above. The 1's complement of the above binary is 0010000.

∴ Answer : −10000

Hexadecimal Numbers

Hexadecimal numbers are used in micro-processiors for programming, analysing or trouble shooting. Any one working with micro computers saves lot of time and energy by using hexadecimal numbers. The hexadecimal number system has a base of 16. It is customery to use 0 to 9 and the letters A to F. Hence the list of hexadecimal numbers is

0 1 2 3 4 5 6 7 8 9 A B C D E F (16 bits)

Here A represents numeral 10, B represents 11 ... and F represents 15. After reaching F, two digit combinations are formed such as 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21 so on.

Example: Write down the next number to the following hexadecimal numbers:

835C, A47F, BFFF, 3F, 9F

Number	Next Number
835C	835D
A47F	A480
BFFF	C000
3F	40
9F	A0

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
:	:
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11

Fig. 2. Decimal - Hexadecimal

Hexadecimal - to - Binary conversion

Here we convert each hexadecimal digit to its 4-bit equivalent binary. For example, 9AF in hexadecimal can be converted into binary form as follows :

9 A F
↓ ↓ ↓
1001 1010 1111

9AF (hexadecimal) = 1001 1010 1111 binary.

Similarly one can verify that C5E2 will read as

C 5 E 2
↓ ↓ ↓ ↓
1100 0101 1110 0010

Hexadecimal C5E2 = 1100 0101 1110 0010 binary.

Example : Find the binary number for the hexadecimal number D2E.8

D	2	E	.	8
↓	↓	↓	.	↓
1101	0010	1110	.	1000

Binary-to Hexadecimal conversion

Given the binary number, we group the binary into four bit nibbles: Starting at the binary point, convert each group (nibble) into its hexa deci mal equivalent adding necessary zeros at extreme ends. The assembly of codes reached gives the hexa decimal equivalent of the given binary number. For example,

binary	1000	1100	1010	1110
	↓	↓	↓	↓
	8	C	A	E

∴ The hexadecimal equivalent is 8C AE

Similarly, one can verify that

Binary 1110 1000 1101 0110 is equivalent to hexadecimal E8D6.

Hexadecimal-to-Decimal conversion

Given a hexa decimal number, we use powers of 16 for the weight of each digit. By summing the products of the hexadecimal digits and their weights, we get the decimal equivalent. The weights for the digit position in a hexa decimal number are as follows.

$$\leftarrow 16^3 \quad 16^2 \quad 16^1 \quad 16^0 \quad . \quad 16^{-1} \quad 16^{-2} \quad 16^{-3} \rightarrow$$

$$\uparrow$$

hexadecimal point.

Example : Convert hexadecimal F8E6.39 into the equivalent decimal.

$$F8E6 = F(16^3) + 8(16^2) + E(16)^1 + 6(16)^0$$
$$= 15(16^3) + 8(16^2) + 14(16^1) + 6(16^0)$$
$$= 61,440 + 2048 + 224 + 6$$
$$= 63,718$$
$$0.39 = 3(16^{-1}) + 9(16^{-2})$$
$$= \frac{3}{16} + \frac{9}{256} = 0.1876 + 0.0352$$
$$= 0.2227$$

hexadecimal F8E6.39 = 63,718 . 2227 decimal

Similarly, one can verify that hexa decimal 3C8 = 968 decimal.

Decimal to Hexadecimal change

The decimal number is divided successively by 16, writing down the remainders till the quotient 0 is reached. The remainders are converted to hex notations and read in the reverse order (that is, from bottom to top)

Example : Convert decimal 72905 to hexa decimal system

			remainder	hexadecimal notations	
16	72905				
16	4556	–	9	9	↑
16	284	–	12	C	
16	17	–	12	C	Read up
16	1	–	01	1	
	0	–	01	1	

Reading the remainders from bottom to top, the result is 11 CC9.

Decimal 72905 = 11CC9 hexadecimal.

Similarly one can verify that decimal 2479 is equivalent to hexadecimal 9 AF.

Example

1. Subtract $[3A8]_{16}$ from $[1273]_{16}$

$[1273]_{16} = 1 \times 16^3 + 2 \times 16^2 + 7 \times 16^1 + 3 \times 16^0$

$= 4096 + 512 + 112 + 3 = [4723]_{10}$

$[3A8]_{16} = 3 \times 16^2 + 10 \times 16^1 + 8 \times 16^0 = [936]_{10}$

$[4723]_{10} - [936]_{10} = [3787]_{10}$

Convert $[3787]_{10}$ in to hexa decimal number.

$[3787]_{10} = E \times 16^2 + C \times 16^1 + B \times 16^0 = [ECB]_{16}$.

[*Ans* : ECB_{16}]

Octal numbers

The octal number system has a base of eight (8), meaning it has eight basic symbols. They are 0,1,2,3,4,5,6,7 (There is no 8 or 9 symbol).

The octal numbers run as follows:

0	1	2	3	4	5	6	7,
10	11	12	13	14	15	16	17,
20	21	22	23	24	25	26	27,
...	...	...	...	...	...	...	37,
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	77
100	101	...	...	...	...	...	...

To remember octal numbers, think of the decimal numbers and cancel any number with a digit greater than 7. The numbers that remain are octal numbers.

In the octal system, the individual digit represents the coefficient of power of 8. The position value of each digit is in ascending powers of 8 for integers and in decending powers of 8 for fractions as shown below:

etc ←	8^3	8^2	8^1	8^0	8^{-1}	8^{-2}	8^{-3}	→ etc

To convert octal to decimal, multiply each octal digit by its weight and add the resulting products. For example, octal number 23 becomes

$$2 \times (8^1) + 3\,(8^0) = 16 + 3 = 19 \text{ decimal}$$

$$\therefore \text{ octal } 23 = \text{decimal } 19.$$

Again, consider octal 257. It is expressed in powers of 8 as

$$2\,(8^2) + 5\,(8^1) + 7\,(8^0) = 128 + 40 + 7 = 175$$

$$\text{octal } 257 = 175 \text{ decimal.}$$

Example: Convert octal 312.45 *into its decimal equivalent,*

$$312 \,.\, 45$$

$$3 \times (8^2) + 1 \times (8^1) + 2 \times (8^0) \,.\, 4\,(8^{-1}) + 5\,(8^{-2})$$

$$192 \quad + \quad 8 \quad + \quad 2 \quad . \quad \frac{4}{8} \quad + \quad \frac{5}{64}$$

$$202 \;.\; 578$$

$$\therefore \text{ octal } 312.45 = 202.578 \text{ decimal.}$$

Decimal to octal conversions

For this, a method known as "octal dabble" is followed. In this method we progressively divide the given decimal number by 8 (the base of octal system). Write down the reminders after each division. These reminders taken in reverse order from bottom to top forms the

required octal number. As an example, convert decimal 197 to its equivalent octal number.

8	197	
8	24	.....5 ↑
8	3	.....0 |
	0	.....3 |

decimal 197 = 305 octal

Like this, one can verify that decimal 175 is equivalent to 257 octal.

Given a decimal fraction 0.23, how to convert it into octal. Multiply each digit by 8, writing the carry into the integer position. The carries, taken in forward order from top to bottom gives the required octal fraction.

$0.23 \times 8 = 1.84 = 0.84$ with a carry 1 ↓

$0.84 \times 8 = 6.72 = 0.72$ with a carry 6 ↓

$0.72 \times 8 = 5.76 = 0.76$ with a carry 5 ↓ etc.

$\therefore$ 0.23 decimal = 0.165 octal

We have terminated after three places. For more accuracy, we would continue multiplying to get more octal digits.

Octal-to-binary conversion

For this we change each octal digit to its binary equivalent. For example to convert octal 23 to its binary equivalent, we write binary equivalent of 2 and 3 as

2	3
↓	↓
010	011

octal 23 = 010 011 binary

Often, a space is left between groups of 3 bits. This makes it easier to read the binary number.

Example 1: Convert octal 34.562 into binary equivalent.

$$\begin{array}{ccccccccc} 3 & & 4 & . & 5 & & 6 & & 2 \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ 011 & & 100 & . & 101 & & 110 & & 010 \end{array}$$

octal 34.562 = 011 100 . 101 110 010 binary

Binary to - octal conversion

Here we start with the given binary number. We write down the binary number into groups of 3 bits, starting at the binary point. Convert each group of 3 into its octal equivlent.

Example : Convert binary 1011. 01101 to octal.

$$\begin{array}{lccccccc} 1011.01101 \rightarrow & 001 & 011 & . & 011 & & 010 \\ & \downarrow & \downarrow & & \downarrow & & \downarrow \\ & 1 & 3 & . & 3 & & 2 \end{array}$$

(Note that we can add zeros to complete the outer side groups)

$$\therefore 1011 . 01101_2 = 13. 32_8$$

Advantages of converting octal-to-binary and vice versa in digital work

(1) It is easier to handle input and output data of a digital computer in octal form, (i.e; it needs less circuitry to work with octal numbers).

(2) The numbers of octal system are one third as compared to binary numbers. Hence octal number system is often used for programming in assembly language of a digital computer.

Hexa-decimal to Octal conversion and vice verse

To convert a hexa-decimal number to an octal number, the following steps may be followed.

1. Convert the given hexa decimal number to its binary equivalent.
2. Form groups of three bits, starting from the Least-Significant bit (LSB).
3. Write the equivalent Octal number for each group of three bits.

For example, to convert $[47]_{16}$ to the octal number.

$$[47]_{16} = [0100\ 0111]_2$$
$$= [01\ 000\ 111]_2$$
$$= [107]_8$$

Thus $[47]_{16} = [107]_8$

Next, we convert an octal number to the equivalent hexadecimal number. The steps are as follows.

(i) Convert the given octal number to the equivalent binary number.

(ii) Form groups of *four* bits, starting from the LSB.

(iii) Write the equivalent hexadicimal number for each of the four bits.

For example, convert $[32]_8$ into its hexadecimal equivalent number.

$$[32]_8 = [011\ 010]_2$$
$$= [01\ 1010]_2$$
$$\therefore \quad [32]_8 = [1\ A]_{16}$$

Problem

1. Convert the following hexadecimal number to octal number.

(a) $[381\ B]_{16}$ (b) $[AB2]_{16}$ (c) $[ABC]_{16}$ (d) $[2A0]_{16}$

[*Ans* : (a) $[34033]_8$ (b) $[5662]_8$ (c) $[5274G]_8$ (d) $[1240]_8$]

2. Convert the following octal numbers into hexadecimal.

(a) $[137]_8$ (b) $[775]_8$ (c) $[1275]_8$ (d) $[4555]_8$

[*Ans*: (a) $[5F16]_{16}$ (b) $[1FD]_{16}$ (c) $[2BD]_{16}$ (d) $[96D]_{16}$]

Codes

Discrete informations are generally in the form of numbers, letters (alphabets) or special characters. When a computer is used to handle the information the discrete informations are fed into a digital computer and the message is retrieved from the computer by using some kind of symbolic representation of information. The binary digit 0 and 1 are used to coin the symbolic representation but they are arranged according to a set of rules.

When numbers, letters or words are represented by a special group of symbols, it is called coding and the special group of symbols is called a code.

Excess -3 code

Excess-3 code is a digital code that is obtained by adding 3 to each decimal digit and then converting the result to four-bit binary.

For example, the excess-3 code for decimal 2 is found by adding 3 to 2. The result is 5 and the binary equivalent of 5 is 0101.

$$\begin{array}{r} 2 \\ +\ 3 \\ \hline 5 \end{array} \rightarrow 0101$$

∴ The excess -3 code for 2 is 0101.

One can verify that the excess -3 code for 9 is 1100.

To convert any decimal number to the excess-3 form, add 3 to each decimal digit and then convert the sum to BCD number. As an example, to convert 12 add 3 to each decimal digit.

1	2
+ 3	+ 3
4	5

Next convert the sum to BCD form

4	5
↓	↓
0100	0101

So, the excess - 3 code for 12 is 0100 0101.

As another example, let us convert 430 into an excess-3 number.

4	3	0
+3	+3	+3
7	6	3
↓	↓	↓
0111	0110	0011

∴ excess-3 code for 430 is 0111 0110 0011

Problem : Convert each of the following decimal nuimber to excess-3 code : (a) 13 (b) 59 (c) 35 (d) 87

Example : What is the excess -3 code for binary 0111?

The given number is 0111 in binary form. We add 3 (binary 0011) to it.

	0	1	1	1
+	0	0	1	1
1	0	1	0	

The excess-3 code for 0111 is 1010

For example, the excess -3 code for decimal numbers from 0 to 9 are as follows:

Decimal	BCD	Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

The special feature of excess-3 code is that it is self-complementing. i.e.; the 1's complement of an excess -3 number is the excess -3 code for the 9's complement of the corresponding decimal number.

For example, the excess -3 code for decimal 4 is 0111

The 1's complement of this is 1000 ... ... (1)

The 9's complement of 4 is $(9-4)=5$

The excess -3 code for 5 is 1000 ... ... (2)

From (1) and (2), we see that 1's complement of an excess -3 number is equal to the excess-3 code for the 9's complement of the corresponding decimal number.

Advantages

The self complementing property of excess -3 code is useful for some of the arithmetic operations (like addition and subtraction) in computers. Moreover, because of this property, no special complement - generator circuit is needed. Another advantage of excess -3 code is that at east one 1 is present in all states, thus providing an error-detection (checking) facility.

Binary Coded Decimals (BCD)

A BCD code is one in which the digits of a decimal number are encoded one at a time-into group of binary digits. In the 8421 code we use 4 bit in a binary group. Whenever we refer to BCD code, we mean the 8421 code. The four bits represent the decimal digits form 0 to 9. The designation 8421 indicates the binary weights of the four bits $2^3,2^2,2^1,2^0$.

Given a decimal number, the 8 4 2 1 code expresses each decimal digits by its 4- bit binary equivalent. For example, the decimal number 429 is changed to its equivalent as follows:

4	2	9
↓	↓	↓
0100	0010	1001

In the BCD code, 0100 0010 1001 stands for the decimal number 429. Similarly one can verify that the decimal 8963 can be encoded as

1000 1001 0110 0011 in the BCD form.

Decimal 7 corresponds to 0111 in binary since the place values in the binary from left are 8,4,2,1 respectively. i.e.the weight in the BCD code are 8 4 2 1.

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

One can see that the largest 4- bit group in the 8421 code is 1001 (corresponding to the largest decimal 9) and the lowest 4-bit group is 0000. Thus, the BCD digits are from 0000 to 1001.All bcombinations above this (1010 to 1111) cannot exist in the BCD code because the highest decimal digit, which can be coded in BCD form is 9.

Application of BCD code BCD numbers are useful whenever decimal information is transferred into or out of a digital system. For example, the circuits in the pocket calculator can process BCD numbers because we enter decimal numbers through the key board and see decimal answers on the LED or liquid crystal display. Other example of BCD systems are electronic counters, digital volt meters and digital clocks.

The main advantage of the 8421 code is that it is easy to convert to and from decimal numbers.

The disadvantages of 8421 code is that the rules for binary addition do not apply to the entire 8421 number, but only to the individual 4-bit groups. For example, adding 12 and 9 binary form is easy. But if we try it in the 8421 code, we get an unacceptable answer as

12	0001	0010
+ 9	+	1001
	0001	1011

We are unable to decode 0001 1011 because there is no such number in the 8421 code. Therefore, addition of 8421 numbers is not so simple as for binary numbers.

The BCD code can be converted into decimal numbers as follows: Starting at the decimal point of the binary number, we break the given BCD into group of 4 bits. Then we write the decimal bit represented by each 4-bit group.

BCD	0101	0111	1000
	↓	↓	↓
decimal	5	7	8

578 is the decimal equivalent of the BCD 0101 0111 1000

Again, given the binary 1100000.0111. Its BCD is

	0110	0000	.	0111
	↓	↓		↓
decimal	6	0	.	7

Example

Convert the BCD number 1001 0011 *into binary.*

Given BCD number is 1001 0011

First we convert this into decimal equivalent. The decimal number is 93. This is converted into equivalent binary, by double - dabble method. The answer is $(101\ 1101)_2$.

Problem

Convert the following BCD into equivalent binary : 0110 0111

[*Ans* : $(100011)_2$]

Gray code

Gray code is a class of codes, in which only one bit in the code group changes when moving from one step to the next successive number. Thus, adjacent code numbers differ only by one bit in the Gray code and so it is called unit - distance code.

This code is not good for arithmatic operation. This is a big disadvantage.

Decimal numbers	Binary code	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
....	....	....

(MSB same in both cases)

Conversion of a binary number to gray code :

Step 1: Keep the first bit (MSB) of the Gray code same as the first bit of the binary number.

Step 2: The second bit of the Gray code equals the exclusive -OR of the first and second bits of the binary number. i.e.,it will be 1 if these binary bits are different and 0 if they are the same.

Step:3 The third Gray code bit equals the exclusive-OR of the second and third bits of the binary numbers and so on.

Example : Convert binary number $[1\ 0\ 1\ 1\ 0]_2$ *to Gray code.*

Step 1 : Binary 1 0 1 1 0
↓
Gray 1

Step 2 : Binary 1 [0 1] 1 0
↓
Gray 1 1 1 1 0

Step 3 : Binary 1 [0 1] 1 0
↓
Gray 1 1 1 1 0

Similarly for the next digit

Binary 1 0 [1 1] 0
↓
Gray 1 1 1 0 0

Binary 1 0 1 [1 0]
↓
Gray 1 1 1 0 1

Thus, Binary $[1\ 0\ 1\ 1\ 0]_2$ = Gray 1 1 1 0 1

Problem : Convert binary $[1\ 0\ 1\ 0\ 1\ 1\ 0]_2$ into Gray code

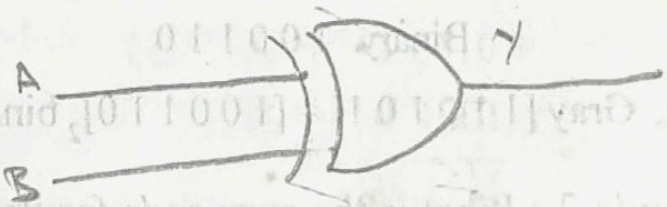
[*Ans:* 1 1 1 1 1 0 1]

Exclusive-OR

Truth Table $Y = A \oplus B$

Input A	Input B	Input A
0	0	0
0	1	1
1	0	1
1	1	0

Circuit diagram.



Conversion from gray code to binary

(i) The first binary bit (MSB) is the same as that of the first Gray code bit.

(ii) If the second Gray bit is zero, the second binary bit is the *same* as that of the first binary; if the second gray bit is 1 the second binary bit is the inverse of the first binary bit.

(iii) Step (ii) is repeated for each successive bit.

Example : Convert Gray code 1 1 0 1 0 1 into binary form

Step 1 :	Gray	1 1 0 1 0 1
	Binary	1
Step 2 :	Gray	1 1 0 1 0 1
	Binary	1 0
Step 3 :	Gray	1 1 0 1 0 1
	Binary	1 0 0
Step 4 :	Gray	1 1 0 1 0 1
	Binary	1 0 0 1
Step 5 :	Gray	1 1 0 1 0 1
	Binary	1 0 0 1 1
Step 6 :	Gray	1 1 0 1 0 1
	Binary	1 0 0 1 1 0

Thus, Gray $[1\ 1\ 0\ 1\ 0\ 1] = [1\ 0\ 0\ 1\ 1\ 0]_2$ binary

Example 2 : What is the gray code for decimal 8 ?

Decimal 8 is expressed first in binary form.

$8 = [1000]_2$

Binary	1 0 0 0
Gray	1 1 0 0

Sign - magnitude numbers

The decimal number + 7 is represented as + 0111 in binary form. The decimal number – 6 is represented as – 0110 in binary form. However computer uses a sign bit 0 to indicate positive (+ve) number and a sign bit 1 to indicate a negative (–ve) number. For example,

+ 7 is written as	0	0111
– 6 is written as	1	0110

This is sign-magnitude form of the binary numbers. The sign bit 0 or 1 is adjacent to the most significiant bit (left most) of the given number.

Examples of sign magnitutde numbers are given below:

+ 7	... ...		0	0 1 1 1
− 6	... ...		1	0 1 1 0
− 16	... ...	1	0 0 0 1	0 0 0 0
+ 25	... ...	0	0 0 0 1	1 0 0 1
− 128	... ...	1	1 0 0 0	0 0 0 0

Sign magnitude numbers are used in analog to digital conversions.

Boolean algebra

Boolean algebra provides a simple mathematical technique for the design, simplification and analysis of switching systems. This mathematical technique helps the understanding of switching circuits, which are very complex for visual analysis. This technique also gives a method of representing different switches and logic functions in mathematical form.

Boolean algebra will be useful in the following ways:

1. The logic functions may be expressed as algebraic equations, offering more compactness and clarity than the truth tables of logic gates.
2. An algebraic equation can be obtained from a truth table by using the Boolean algebra.
3. Boolean equation may be simplified, leading to an understanding of implementation of a desired logic function.

Boolean algebra requires the use of a set of special symbols given below:

1. Symbol for constants : 0 and 1
2. Symbol for variables : *A, B, C, X, Y, Z, W, Q* etc.
3. Symbol for functions : AND function, •
   : OR function, +
   : NOT function (inversion), – (bar)
4. Symbol for equality : =

Boolean postulates

The Boolean postulates originate from the three basic logic functions of AND, OR and NOT, whose truth tables are shown below:

AND		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT	
A	Y
0	1
1	0

The postulates are expressed as algebraic equations, using the appropriate function symbols as given below:

Postulates

	Algebraic equation	
(1)	$0 \cdot 0 = 0$	derived from the AND function
(2)	$0 \cdot 1 = 0$	
(3)	$1 \cdot 0 = 0$	
(4)	$1 \cdot 1 = 1$	
(5)	$0 + 0 = 0$	the OR function
(6)	$0 + 1 = 1$	
(7)	$1 + 0 = 1$	
(8)	$1 + 1 = 1$	
(9)	$\overline{0} = 1$	the NOT function
(10)	$\overline{1} = 0$	

Derivation of Boolean Properties from the Postulates

(1) $A \cdot 0 = 0$

Proof:

Case 1 : $A = 0$; then $A \cdot 0 = 0 \cdot 0 = 0$ by postulate 1

Case 2 : If $A=1$, then $A \cdot 0 = 1 \cdot 0 = 0$ by postulate 3.

Therefore $A \cdot 0 = 0$ for all A.

(2) $0 \cdot A = 0$

Proof : (left as an exercise for the student)

(3) $A.1 = A$

Proof:

Case 1 : If $A = 0$, then $A.1 = 0.1 = 0 = A$

Case 2 : If $A = 1$, then $A.1 = 1.1 = 1$. Therefore $A.1 = A$ for all A.

(4) $1.\ A = A$

Proof: (left as an exercise for the student)

(5) $A + 0 = A$

Proof:

Case 1: If $A = 0$, $A + 0 = 0 + 0 = 0 = A$

Case 2: If $A = 1$, $A + 0 = 1 + 0 = 1$. Then $A + 0 = A$ for all A

(6) $0 + A = A$

(7) $A + 1 = 1$

(8) $1 + A = 1$

(9) $A \cdot A = A$

Proof:

Case 1: If $A = 0$, $A \cdot A = 0 \cdot 0 = 0 = A$

Case 2: If $A = 1$, $A \cdot A = 1 \cdot 1 = 1$. Then $A \cdot A = A$ for all A.

(10) $A \cdot \overline{A} = 0$

Proof:

Case 1: $A = 0$, $A\ \overline{A} = \overline{0} \cdot 0 = 0 \cdot 1 = 0$

Case 2: If $A = \overline{1}$, $A.\ \overline{A} = 1 \cdot \overline{1} = 1.0 = 0$. Then $A \cdot \overline{A} = 0$ for all A.

(11) $A + A = A$

(12) $A + \overline{A} = 1$

(13) $\bar{\bar{A}} = A$ ..... double complementation

(14) $A.B = B.A$ .....commutative law

(15) $A + B = B + A$

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

(17) $A + (B.C) = (A+B).(A+C)$ .....distributive law.

Proof:

Case 1: $A = 0$ Then,

left side $= A + (B.C) = B.C$

right side $= (0 + B).(0 + C) = B.C$

Case 2: $A = 1$ Then,

left side $= 1 + B.C = 1$

right side $= (1 + B).(1 + C) = 1.1 = 1$

Therefore $A + (B.C) = (A + B).(A + C)$ for all A, B, C.

(18) $A.(B.C) = (A.B).C$

(19) $A + (B + C) = (A + B) + C$ .....Associative law

(20) $A + (A.B) = A.(1 + B) = A.1 = A$.....Absorption theorem

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

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

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

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

(25) $\overline{A.B} = \bar{A} + \bar{B}$ ..... De Morgaa's Law

De Morgan's Laws

De Morgan's laws are helpful to see how an expression using AND operation can be converted into equivalent expression using OR operation and vice versa.

Any logic expression can be inverted (or complemented) using simply by inverting term so that A becomes $\bar{A}$; B becomes $\bar{B}$ and every AND becomes OR and vice-versa. This is stated mathematically by De Morgan's laws as follows:

Law 1: $\overline{A+B} = \bar{A} \cdot \bar{B}$

i.e. The complement of sum equals the product of complements.

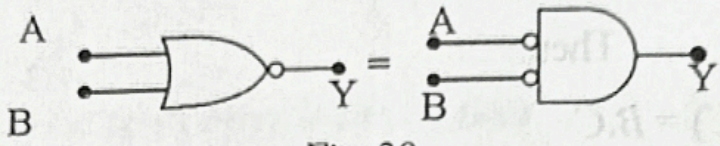


Fig. 39

The law reveals that a NOR gate $(\overline{A+B})$ is equivalent to bubbled AND gate $(\bar{A} \cdot \bar{B})$

Law II : $\overline{A \cdot B} = \bar{A} + \bar{B}$

ie., The complement of product equals the sum of complements.

The law reveals that a NAND gate $(\overline{A \cdot B})$ is equilvalent to bubbled OR gate $(\bar{A} + \bar{B})$.

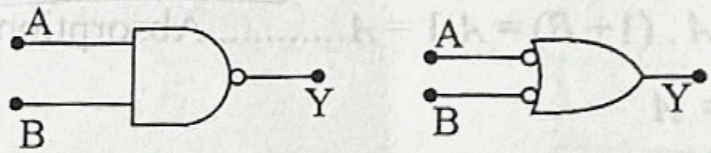


Fig. 40

To prove the law I: $\overline{A+B} = \bar{A} \cdot \bar{B}$ by induction

We have to show that the left side is equal to the right side for all posible values of A and B. There are four possible cases:

(1) $A = 0$; $B = 0$

Left: $\overline{A+B} = \overline{0+0} = \bar{0} = 1$

Right: $\bar{A} \cdot \bar{B} = \bar{0} \cdot \bar{0} = 1.1 = 1$

(2) $A = 0;\ B = 1$

Left: $\overline{A+B} = \overline{0+1} = \overline{1} = 0$

Right: $\overline{A} \cdot \overline{B} = \overline{0} \cdot \overline{1} = 1.0 = 0$

(3) $A = 1;\ B = 0$

Left : $\overline{A+B} = \overline{1+0} = \overline{1} = 0$

Right: $\overline{A} \cdot \overline{B} = \overline{1} \cdot \overline{0} = 0.1 = 0$

(4) $A = 1;\ B = 1$

Left : $\overline{A+B} = \overline{1+1} = \overline{1} = 0$

Right : $\overline{A} \cdot \overline{B} = \overline{1} \cdot \overline{1} = 0.0 = 0$

In all the above cases. the left side equals the right side, proving De Morgan's Law 1.

To prove the law II: $\overline{A \cdot B} = \overline{A} + \overline{B}$ using truth table:

A	B	$A \cdot B$	$\overline{A \cdot B}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

In all four cases $\overline{A \cdot B} = \overline{A} + \overline{B}$, thus proving the law.II

According to law II, $\overline{A \cdot B} = \overline{A} + \overline{B}$

Taking complements, $\overline{\overline{A \cdot B}} = \overline{\overline{A} + \overline{B}}$

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

The meaning of the expression in the *left* side is: "If and only if all input A and B are high (1), the output is high (1)".

This is equivalent to the statement on the *right* side "If atleast one input is low (0), then the output is low (0)". This is the meaning conveyed by De Morgan's law

Applications

1. To show that the positive logic AND gate becomes an OR gate when negative logic is considered

Consider an AND gate circuit, with positive logic. Let A, B, C be the inputs to the AND gate.

The out put is $Y = A . B . C$ ... ... (1)

Now, if negative logic is used, each variable will be complemented and in addition, the output will be complemented. The inputs are $\bar{A}$, $\bar{B}$, $\bar{C}$ and output is $\bar{Y}$. Hence, $\bar{Y} = \bar{A} . \bar{B} . \bar{C}$

The complemented output $\bar{\bar{Y}} = \overline{\bar{A} . \bar{B} . \bar{C}}$

$$\therefore Y = \overline{\bar{A} . \bar{B} . \bar{C}}$$

Using De Morgan's law $Y = \bar{\bar{A}} + \bar{\bar{B}} + \bar{\bar{C}}$

$$\therefore Y = A + B + C \quad \text{... ... ...(2)}$$

This is the OR output, under negative logic. Equation (2) is derived from equation (1). Hence the result. In a similar way, we can show that if the original gate had been an AND gate negative logic, it would become an OR gate if positive logic is used. Thus, a given electronic circuit might be considered to be either an AND gate or an OR gate depending upon whether / or not positive logic is used.

As another example, consider a NAND gate using positive logic. Its output will be $Y = \overline{A\,B\,C}$

Now, if negative logic is used, the output becomes $\bar{Y} = \overline{\bar{A}\ \bar{B}\,\bar{C}}$

The complemented output is $\bar{\bar{Y}} = \overline{\bar{A}\ \bar{B}\,\bar{C}}$

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

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

using De Morgan's law. This is just the output of a NOR gate under negative logic. Therefore, the positive logic NAND gate has become a NOR gate under negative logic.

2. Implementing the basic logic gates using NAND gates only (the NAND gate as Universal building block)

The NAND gate can be regarded as universal building block because it can be connected to other NAND gates to generate any logic function. We show here how the NOT, the AND, the OR, the NOR and the XOR logic gates can be fabricated with the use of NAND gates only.

(i) NOT *gate using* NAND

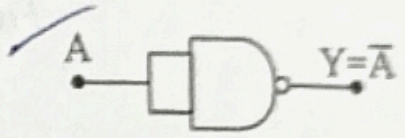


Fig. 41 *NOT using NAND.*

In a NAND gate, let the inputs be A and B.
The output $Y = \overline{A \,.\, B}$

If the inputs are the same, each equal to A, (the two input terminals are tied together)

$$Y = \overline{A \,.\, A}, \quad \text{i.e., } Y = \bar{A}$$

For the input A the output is $\bar{A}$. Thus, the function of the NOT gate is performed.

(ii) AND *gate using* NAND *gates.*

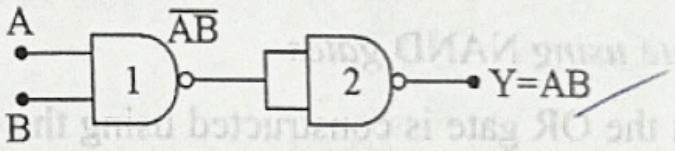


Fig. 42 *AND using NAND*

The inputs A and B are fed to NAND gate (1) and its output is $\overline{AB}$. This is used to feed two equal inputs, $\overline{AB}$ each to the NAND gate.

2) The second NAND gate works as a NOT gate.

The output $Y = \overline{\overline{AB}}$; i.e., $Y = A\,B$

Thus, the function of an AND gate is carried out by the use of two NAND gates.

iii) OR *gate using* NAND *gates*

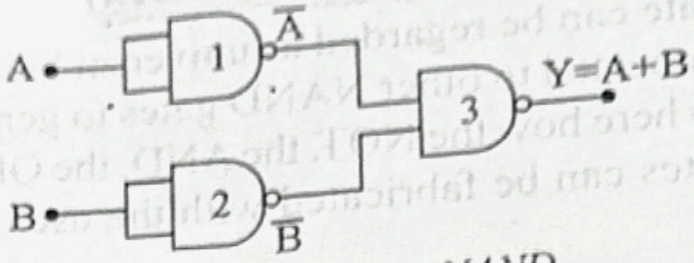


Fig. 43 OR using NAND

The input A is fed to the NAND gate (1) which acts as an invertor (NOT gate). The output is $\overline{A}$. Similarly, the input B is fed to the NAND (2), which act as an invertor, to get the output $\overline{B}$. *Now*, $\overline{A}$ and $\overline{B}$ are fed as inputs to the NAND (3). After the NAND operation, the output becomes

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

Using De Morgon's law,

$$Y = \overline{\overline{A}} + \overline{\overline{B}} \qquad \text{i.e; } Y = A + B$$

Thus, the OR operation is carried out with three NAND gates only.

(iv) NOR *gate using* NAND *gates*

First the OR gate is constructed using the NAND gates 1, 2 and 3. If A and B are the inputs, we know that the output of NAND 3 is

$$Y' = A + B$$

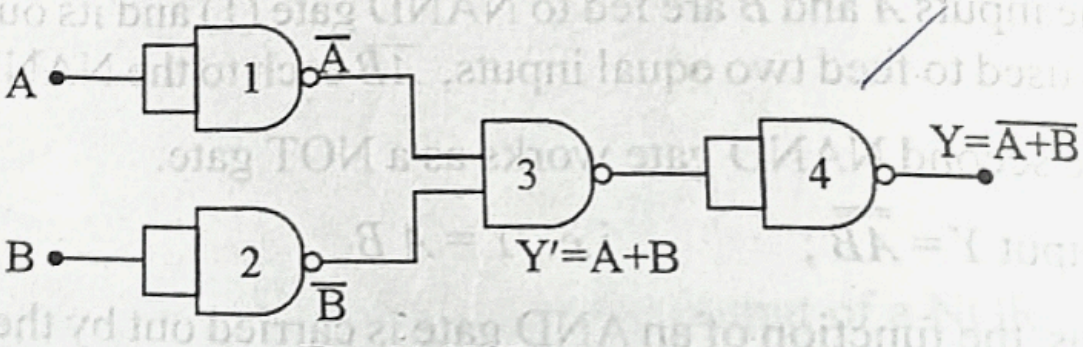


Fig. 44 NOR using NAND

Using another NAND gate to act as an inverter, Y can be inverted to obtain $Y = \overline{A + B}$. The complete circuit arrangement is shown in the figure. Thus the NOR operation can be carried out by the use of four NAND gates.

(v) Ex - OR gate using NAND *gates*

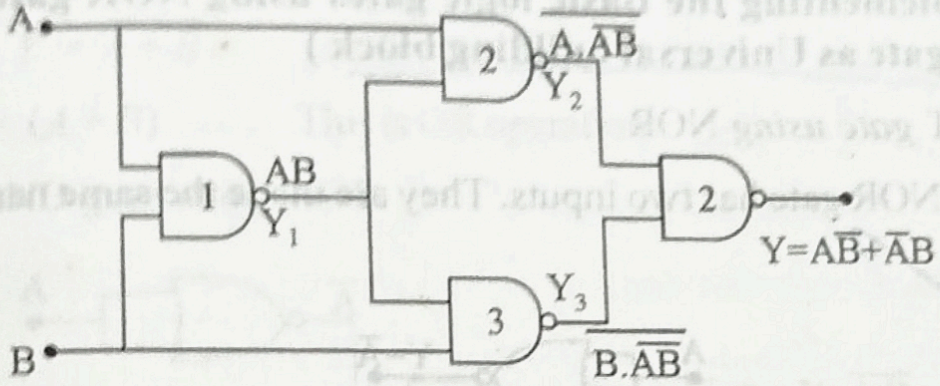


Fig. 45 XOR using NAND

Let A and B be the inputs fed to the NAND gate 1.

The output of NAND 1 is $Y_1 = \overline{A \,.\, B}$

The inputs to the NAND 2 are A, Y_1

The output of NAND 2 is $Y_2 = \overline{A \,.\, Y_1}$

$$Y_2 = \overline{A \,.\, (\overline{A\,B})} = A \,.\, \overline{(\bar{A} + \bar{B})}$$

$$= \overline{A\,\bar{A} + A\,\bar{B}}$$

$$= \overline{A \,.\, \bar{B}} \quad \text{since } A \,.\, \bar{A} = 0$$

The inputs to the NAND 3 are B, Y_1

The output of NAND 3 is $Y_3 = \overline{B \,.\, \bar{A}}$

The input to the NAND 4 are Y_2 and Y_3

The output of NAND 4 is $Y = \overline{Y_2 \,.\, Y_3}$

$$= \overline{\overline{A\,\bar{B}} \,.\, \overline{B\,\bar{A}}}$$

$$= \overline{\overline{A\,\bar{B}} + \overline{B\,\bar{A}}}$$

$$Y = A\,\bar{B} + B\,\bar{A}$$

$$Y = A \oplus B$$

Thus the EX-OR operation can be carried out by the use of four NAND gates

3. Implementing the Basic logic gates using NOR gates only (NOR gate as Universal building block)

(i) NOT *gate using* NOR

The NOR gate has two inputs. They are made the same namely A

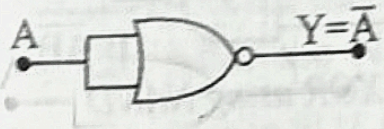


Fig. 46 NOT using NOR

The output $Y = \overline{A + A}$, i.e., $Y = \bar{A}$. This is NOT operation.

(ii) AND *gate using* NOR

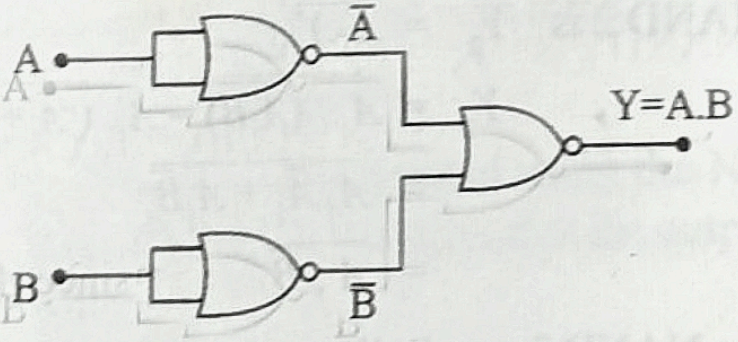


Fig. 47 AND using NOR

Given two inputs A and B, first $\bar{A}$ and $\bar{B}$ are produced using two NOR gates as before.

The final output $Y = \overline{\bar{A} + \bar{B}} = \bar{\bar{A}} . \bar{\bar{B}}$

$Y = A . B$. This is AND operation

(iii) OR *gate using* NOR

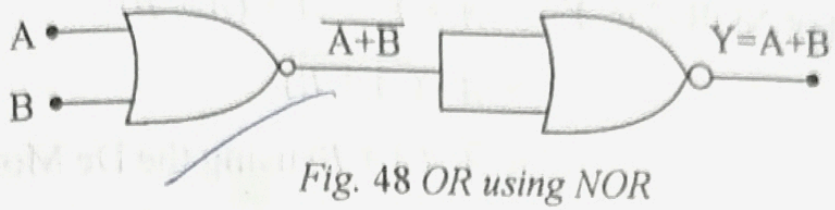


Fig. 48 *OR using NOR*

Output $Y = \overline{\overline{A+B}}$

$Y = (A+B)$ This is OR operation.

(iv) NAND *gate using* NOR

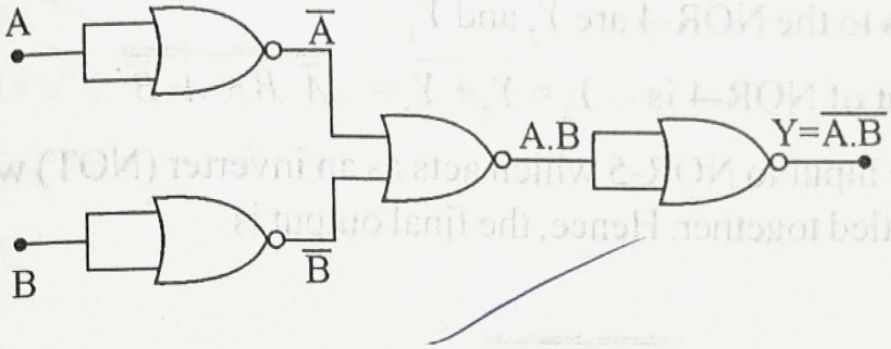


Fig. 49 *NAND using NOR*

Output $Y = \overline{A \,.\, B}$ This is NAND operation.

(v) Ex-OR *gate using* NOR

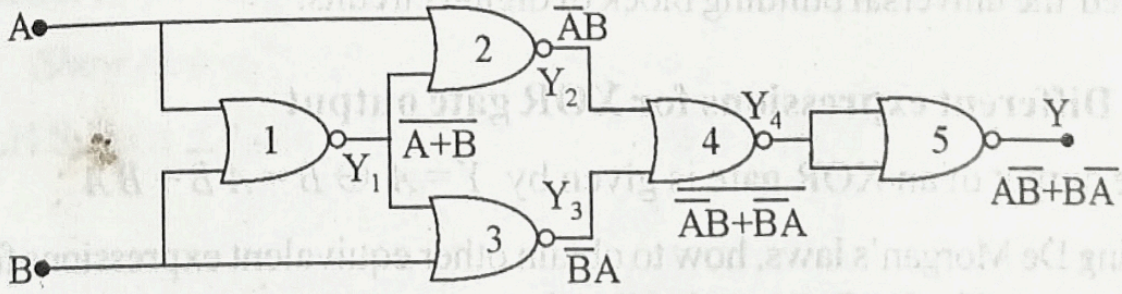


Fig. 50 *XOR using NOR*

Let A and B be the inputs fed to the NOR gate 1.

The output of NOR-1 is $Y_1 = \overline{A+B}$