Performing Binary Logical Operation in Decimal Arithmetic

YumnamKirani Singh

C-DAC Silchar,
Grond Floor, IIPC Building, NIT Silchar Campus,
Yumnam.singh@cdac.in

Abstract*:*Proposed here are the new ways of performing binary logical operations using decimal arithmetic. Usually, binary logical operations such as AND, OR, XOR etc are performed in binary arithmetic which is time consuming and quite erroneous when performed manually. We are familiar with decimal arithmetic, so if we can perform binary logical operations in decimal arithmetic, it will be easier for us and at the same time less error prone and help us understanding the logical operations in a better way. These logical operations are basic building block of digital devices and systems in the modern digital world. Understanding different ways of performing these operations may altogether bring a change in how the digital devices are designed. A deeper study has been made on these operations and a new logical operation named YOR has also been introduced.

***Keywords*: Binary Logical Operations, OR, AND, XOR, YOR, Logical tables, Logical image patterns**

(Article history: Received: 17th March 2019 and accepted 9th June 2019)

I. INTRODUCTION

We know in the digital world, we need to process the data in binary forms. Quite often, in addition to basic arithmetic operations like addition, subtraction etc we also need to perform logical operations such as AND, OR, XOR etc. In addition to using these operations designing digital circuits, these operations are also used in computations [2,3,5]. Out of these operators, XOR operations is popularly used in encryption [1,4].These binary logical operations are the basic building blocks of the digital devices and systems in the modern digital world. These operations are so far done in binary form. So, when we need to XOR any two numbers, (say 120 and135), we first convert the two decimal numbers into binary numbers (1111000 and 10000111). We then perform the operation in binary and then the resulted binary number will be converted back into decimal. If we can directly perform the XOR operation in decimal, then at least the conversion steps would be saved and at the same time we can easily perform it manually. That is main motivation behind the writing of this paper. We want to develop a way of performing binary logical operation using decimal arithmetic. To understand the binary logical operation and their effects on decimal numbers we study the logical operations separately. We derive the recursive relation to generate tables of desired size in decimal. Some observable properties of each logical operation are also given. The way of performing each logical operation in decimal arithmetic with examples is also given.

II. BINARY LOGICAL OPERATIONS

Binary logic operators are used mainly in digital circuit design, communication and low-level programming languages. There is also an Algebra known as Boolean Algebra, specifically dealing with these logical operators. These binary logical operators are studied using binary arithmetic. In this section we discuss these binary logical operators in terms of decimal arithmetic and explore some their new properties.

A. ANDing Operation

The binary logic for AND operation is that if two binary bits are 1, it outputs 1 otherwise 0. This when put in the form of a table is commonly known as truth table. When two decimal numbers are to be ANDed, the two numbers are converted into binary and perform the AND operation bit by bit. The resulting bits are then converted back into decimal. Performing AND operation in binary is erroneous for larger numbers. We will find a way to perform the ANDing operation in decimal arithmetic. For that, we need to perform systematic study of the ANDing operation on decimal arithmetic. Table-1 shows the ANDing of numbers 0 to 15 and their corresponding results.

There is a pattern in the table. If we divide the table into four equal parts, we see that the first, second and the third parts are equal. The fourth part is obtained from any of the first three parts by adding 8. Similarly, if we divide the any of the four equal parts into four equal parts, we see similar arrangement of patterns. For example, if we divide the first part into four equal parts, the first three parts are equal. Adding 4 to any of the three parts gives the fourth part. This regularity of pattern in logical AND operation can be expressed in a recursive form as

$$M_0 = 0$$

$$M_k = \begin{bmatrix} M_{k-1} & M_{k-1} \\ M_{k-1} & 2^k + M_{k-1} \end{bmatrix}$$

Where k is any positive integer.

When $k = 1$, it gives the truth table of the binary AND operation.

Table-I: Logic table of AND operation in decimal

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

From the Table-I, we can observe the following properties.

1. ANDing of any number with 0 is 0
2. ANDing of any number with itself is the number itself.
3. ANDing of any number with 1 is 0 or 1 according as the number is even or odd.
4. ANDing of any two numbers x and y is 0 if $x+y=2^k-1$
5. ANDing of any number with 2^k is 0 or 2^k according as the quotient of the number when divided by 2^k is even or odd.
6. ANDing of any number with 2^k-1 is the modulus of 2^k

Using the recursive relation of AND logic, we can easily get a AND matrix of size $2^k \times 2^k$

The image pattern of the AND matrix of size 256x256 is shown in Fig. 1.

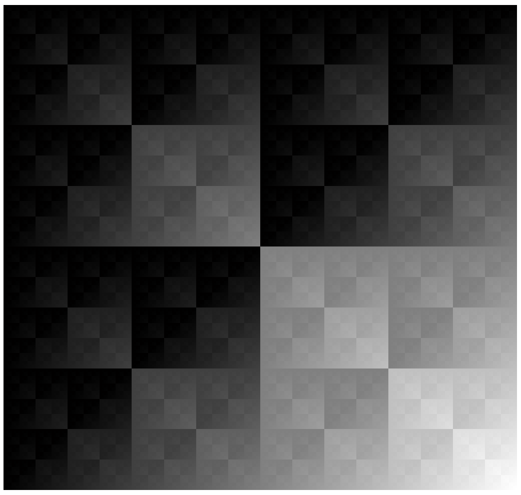
Fig. 1: Image of AND logic table of size 256x256

Performing AND binary operation in decimal:

When two decimal numbers are to be ANDed, express the numbers as sum of powers of 2. Then, take only the common and add them.

Example-A.1: ANDing of 6 and 5, say AND(6,5)
Express 6 and 5 as sum of powers of 2. So,
6= 4+2, 5= 4+1
Note here that 1 is also power of 2, i.e $2^0=1$
Find the powers of 2 common to both and add.
Since 4 is common, AND(5,6)=4
Binary AND operation: Binary of 6 =110
Binary of 5 =101
ANDing 110 and 101=100, which is 4 in decimal.

Example-A.2:ANDing of 27 and 13
Express 25 and 13 as sum of powers of 2.
27=16+8+2+1, 13=8+4+1
Find the powers of 2 common to both and add.
Since 8+1 is common to both, AND(27,13)=8+1=9

B. ORing Operation

The binary logic for OR operation is that when two input bits are 0, the output is 0 otherwise 1. That is, if any of the input bits is 1, the output is 1. We can extend it to decimal number. That is, unless all bits of inputs are all zeros, the result is a positive integer. The OR logic table in decimal for integers 0 to 15 is shown in Table-2. It can be seen that, except the first element, which corresponds to the ORing of 0 with 0, all elements are positive. If we closely observe and analyze the table, we could see some kind of regular number patterns. If we divide the matrix into 4 equal parts, we see that except the first part, the remaining three parts are equal. This is true if, we subdivide any part into four equal parts.

Table-II: Logic table of OR operation in decimal

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

The following properties can be observed from Table-II.

1. ORing of any number with 0 is the number.
2. ORing of any number with 1 is the number or the number plus 1 according as the number is odd or even.
3. ORing of any number with itself is the number.
4. ORing of any two numbers x and y is 2^k-1 if $x+y=2^k-1$
5. ORing of any number with 2^k-1 is $2^k(q+1)-1$, where q is the quotient when the number is divided by 2^k-1
6. ORing of any number with 2^k is 2^k+r, where r is the remainder when the number is divided by 2^k.

The regularity of pattern in logical OR operation can be expressed recursively as

$M_0=0$

$$M_k=\begin{bmatrix} M_{k-1} & 2^k+M_{k-1} \\ 2^k+M_{k-1} & 2^k+M_{k-1} \end{bmatrix}$$

where k is any positive integer.

When $k=1$, it gives the truth table of the binary OR operation.

Using the recursive relation of OR logic, we can easily get a OR matrix of size $2^k\times 2^k$

The image pattern of the OR logic table of size 256x256 is shown in Fig. 2.

Performing OR binary operation in decimal:

Express the two decimal number as sum of powers of 2. Add the various powers of 2 in the two numbers only once.

Example-B.1: ORing of 41 and 25, i.e., OR(41,25)
Express the two numbers as a sum of powers of 2
41=32 + 8 + 1, 25=16 +8 + 1
Adding various powers of 2 (of 41 and 25) only once, we get
32 + 16 + 8 +1= 57
So, OR(41,25)=57

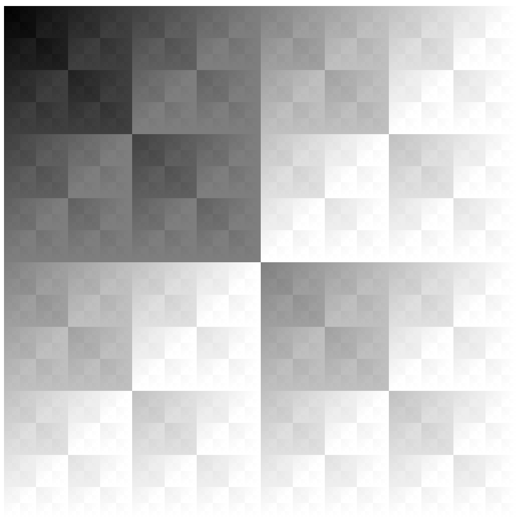

Fig. 2: Image of OR logic table of size 256x256

Example-B.2:ORing of 13 and 27
Express 13 and 27 as a sum of power of 2.
13=8+4+1, 27=16+8+2+1
Adding various powers of 2 (in 13 and 27) only once.
16+8+4+2+1=31
So, OR(13,27)=31

Another way to find the OR(x,y) is
OR(x,y)=(x+y) –AND(x,y)

Example-B.3:OR(13,27)
OR(13,27)=(13+27)-AND(13,27)
=40 –9 =31,
Same as in Example-B.2.

C. *XORing Operations*

The logic behind the binary XORing operation is that when the two input bits are the same the output bit is 0 and otherwise the output bit is 1. This is useful to test whether the two bits are equal or not. This can be extended to test whether two decimal numbers are same or not. XORing of two decimal numbers yields 0 is the two numbers are the same otherwise a positive number. Table-3 gives the result of XORing of any two numbers between 0 and 15. We can clearly see that the diagonal elements are 0 and off-diagonal elements are 15. The diagonal elements are result of XORing of a decimal number with itself. The off-diagonal elements are XORing of a two decimal numbers whose sum is equal to 15 i.e, $2^k - 1$ where $k = 4$ is the square root of the size of the table. If we closely observe and analyze the table, we could see some kind of regular number patterns. If we divide the matrix into 4 equal parts, we see that the diagonal parts are equal.
This regularity of the pattern in XOR logical operation can be expressed recursively as

$$M_0 = 0$$

$$M_k = \begin{bmatrix} M_{k-1} & 2^k + M_{k-1} \\ 2^k + M_{k-1} & M_{k-1} \end{bmatrix}$$

where k is positive integer.
When $k = 1$, it gives the truth table of the binary XOR operation.
The following properties can be observed from Table-3.

1. XORing of any number with 0 is the number itself.
2. XORing of any number with 1 is the number plus 1 or minus 1 according as the number is even or odd.
3. XORing of any number with itself is 0
4. XORing of any two numbers x and y is $2^k - 1$ if $x + y = 2^k - 1$
5. XORing of any two numbers x and y is 1 if $|x - y| = 1$
6. XORing of any two numbers x and y is $2^k - 2$ if $x + y = 2^k - 2$ or 2^k
7. XORing of any number with 2^{k-1} is $2^k(q+1) - (r+1)$, where q and r are the quotient and remainder when x is divided by 2^k.
8. XORing of any number x with 2^k is $2^k(q + (-1)^q) + r$, where q and r are the quotient and remainder when x is divided by 2^k.
9. XOR logic is reversible. That is, if XOR(x,y)=z, then XOR(z,y)=x and XOR(x,z)=y. E.g., XOR(7,3)=4, XOR(4,3)=7, XOR(7,4)=3.

The XOR table exhibits repetitive patterns. The image pattern of XOR table of size 256x256 for k=8 is shown in Fig. 3.

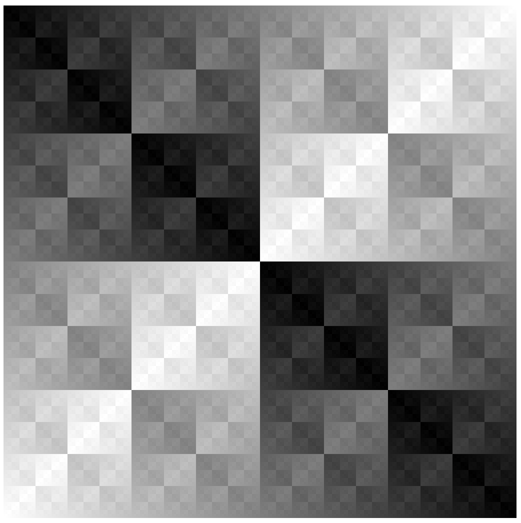
Fig. 3: Image of XOR logic table of size 256x256

Table-IIILogic table of XOR operation in decimal

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

Performing XOR binary operation in decimal:
Express the two numbers as sum of powers of 2. Take only those elements, which are not common to both and add.

Example-C.1: XORing of 13 and 27, i.e., XOR(13,27)
Express 13 and 27 as sums of powers of 2.
13=8+4+1
27=16 + 8+2+1
Find the powers of 2, which are uncommon and add.
16+4+2=22
So, XOR (13,27)=22

Example-C.2: XOR (11,14)
11= 8+2+1
14=8+4+2
Uncommon powers of 2 in 11 and 14 are 1 and 4
So, XOR (11,14)=1+4=5

As OR operation gives the sum of the factors of powers of 2, which are common and uncommon, AND operation gives the sum of power of 2 which are common, then XOR which gives the sum of powers of 2,which are uncommon to both inputs can be derived from OR and AND operation as

XOR(x,y)=OR(x,y)-AND(x,y)

Example C.3: XOR (13,27)
OR(13, 27)=31, AND(13, 27)=9
So, XOR(13,27)= OR(13, 27)- AND(13,27)=31-9=22

Another way to find XOR(x,y) is using the following relation
XOR(x,y)=(x+y)-2*AND(x,y),
where * denotes multiplication.
So, XOR(13,27)=(13+27)-2* AND(13,27)
=40 –2*9=40-18=22
Also, we can write XOR operation as
XOR(x,y)=2*OR(x,y) – (x+y)

So, XOR(13,27)=2*OR(13,27)-(13+27)
=2*31-40=22

D. YORing Operation

YORing logic is similar to the XORing logic in the sense it can be used to compare the equality of two input bits or numbers. If the two input bits are equal it gives 1, otherwise zero. This can be extended for number inputs. If the two numbers are equal it gives 1, otherwise it gives other non-negative numbers. Table-4 shows the YOR logic table for numbers between 0 and 15. The table has regular pattern similar to the XOR table. If we divide the table into four equal parts, the diagonal parts are equal. Similar is the case if we subdivide any of the parts into four equal parts. This regularity of pattern in YOR logic can be expressed recursively as.

$M_0 = 0$

$$M_k = \begin{bmatrix} 2^k + M_{k-1} & M_{k-1} \\ M_{k-1} & 2^k + M_{k-1} \end{bmatrix}$$

where k is positive integer.
When $k=1$, it gives the truth table of the binary YOR operation.
The following properties can be observed from Table-4.

1. YORing of any number with 0 is the number plus 1 or minus 1 according as the number is even or odd
2. YORing of any number with 1 is the number itself
3. YORing of a number with itself is 1
4. YORing of any two numbers x and y is $2^k - 2$ if $x + y = 2^k - 1$
5. YORing of any two numbers x and y is 0 if $|x - y| = 1$
6. YORing of any two numbers x and y is $2^k - 1$ if $x + y = 2^k - 2$

Table-4: (16×16) YOR logic in decimal

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

7. YORing of any number x with 2^{k-1} is $2^k(q+1)-(r+1)+(-1)^{r_1}$, where q and r are the quotient and remainder when x is divided by 2^k and r_1 is the remainder when $x+1$ is divided by 2.
8. YORing of any number x with 2^k is $2^k(q+(-1)^q)+r+(-1)^{r_1}$, where q and r are the quotient and remainder when x is divided by 2^k and r_1 is the remainder when x is divided by 2.
9. YOR logic is reversible. That is, if YOR(x,y)=z, then YOR(z,y)=x and YOR(x,z)=y. E.g., YOR(7,5)=3, YOR(3,5)=7, YOR(7,3)=5.

The YOR logic, similar to XOR logic exhibits a regular geometric pattern. Following Figure-4 shows the geometric pattern, we obtained from YOR logic Table of size 256x256.

Performing YOR operation in decimal:

Find the XOR of the two numbers. Add 1 if the two numbers are odd or even. If one is odd and the other even, subtract 1. That is, if both inputs are even or odd, yor(x,y)=xor(x,y) +1 else yor(x,y)=xor(x,y)-1. This can also be expressed as
YOR(x,y)=XOR(x,y) +(-1)$^{(x+y\ \%2)}$

Example-D.1:YORing of YOR(13,27)
13=8+4+1, 27=16+8+2+1
XOR(13,27)=16+4+2=22
We have to add 1 to XOR since both the two inputs are odd i.e., the sum of the two input is even so remainder of 2 is 0.
So,YOR(13,27)=22 + 1=23

Example-D.2:YOR(11,14)
11=8+2+1, 14=8+4+2
XOR(11,14)=4+1=5
Since one of the input is even and the other odd.
YOR (11,14)=XOR(11,14)-1 =5-1=4

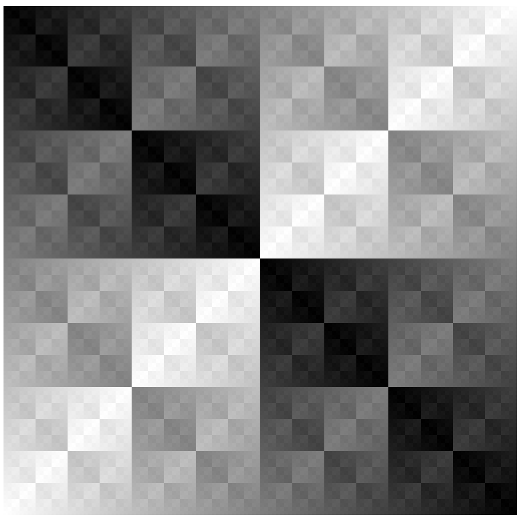

Fig. 4: Image of YOR logic table of size 256x256

III. Discussions

We have described the ways of performing binary logical operation in decimal arithmetic. Out of the four logical operations, the XOR and YOR logics are reversible logic, which are useful for data encryption purpose. Examples of performing of the logical binary operation in decimal are also given. Performing a logical operation using other logical operations is also given. For example, performing OR operation using AND operation and vice versa. Also, XORing can be obtained from the AND and OR operations. We also use the term (x +y) in obtaining one logical operation from others. The term x+y signifies the addition of OR and AND operation.
That is,
(x+y)=OR(x,y) +AND(x,y)
XOR(x,y)=OR(x,y) – AND(x,y)
So, from these, two expressions we can write OR and AND operations in terms of XOR as
OR(x,y)=[(x+y)+XOR(x,y)]/2
AND(x,y)=[(x+y)-XOR(x,y)]/2

So, we can also think of XOR as universal logic operation from which we can get all the remaining three logical operations, AND, OR and YOR. As finding AND or XOR is easier in decimal arithmetic, we can use either of them to perform other logical operations.
Besides these four logical operations, there are other logical operation NOR (NOT OR), NAND (NOT NAND), XNOR (NOT XOR), YNOR (NOT YOR) by applying inverse logic called NOT. The NOT logic operation corresponds to complementing the bits in its operand. It takes single operand and finds the sum of missing power when the operand is expressed as sum of powers of 2. Consider the number 25, which can be express as$(2^4+2^3+2^0)$. In the expression of 25, 2^2+2^1 is missing. So, NOT(25) is 6. Alternatively, the NOT operation can be performed by using the following relation
NOT(x)= 2^k -1-x,
where k=floor(log2(x))+1.
For example, NOT(12)=$2^4-1-12=3$
So, we can compute NOR(41,25)=NOT (OR(41,25)) =NOT(47)=$2^6-1-47=16$
i.e., NOR(41,25)=16.
In this way, we can compute all logical inverse binary logical operations in decimal.

Conclusions

The binary logical operations are studied and described their computations in decimal arithmetic. Several properties of the logical operations in decimal are also derived. Generalized recursive relations for the logical operations that can be used to generate a logical matrix or table in decimal are also given. Several ways of performing binary logical operations in decimal arithmetic are also described to enable a reader understand binary operation clearly. This will help the teachers and students in understanding and performing the binary logical operations in an easier way.

References

[1] Z.A. Kissel, “Ofuscation of standard XOR algorithm,” Crossroads, vol. 11(3), pp. 6-6, 2005.

[2] M.Morries Mano. “Digital Logic and Computer Design”, Prentice Hall, 6rd Ed. 2016.

[3] M. Moris Mano, Michel D. Ciletti,. “Digital Design – with an introduction Verilog HDL”, Pearson, 5th Ed, 2014.

[4] Abdelfatah A.Tamimi, Ayman M. Abdalla, “An Image Encryption Algorithm with XOR and S-BOX” Int’l Conf. IP, Comp.Vision and Pattern Recognition (IPCV’15), pp. 167-169.

[5] John F.Wakerly, Digital Design-Principles & Practices. Prentice Hall INC, 4th Ed., 2012.

AUTHOR PROFILE

YumnamKirani Singhcompleted Master's Degree in Electronics Science from Guwahati University in 1997 and got Ph. D. degree from Indian Statistical Institute, Kolkata in 2006. Served as a lecturer in Electronics in Shri Shankaracharaya College of Engineering & Technology from Jan 2005 to May 2006. Joined CDAC Kolkata in May 2006 and worked there before coming to CDAC Silchar, in March 2014. Developed Bino's Model of Multiplication, ISITRA, YKSK Transforms and several other image binarization and edge detection techniques. Interested in working in the application and research areas of Signal Processing, Image Processing, Pattern Recognition and Information Security. Also published several papers in national and international journals and conferences