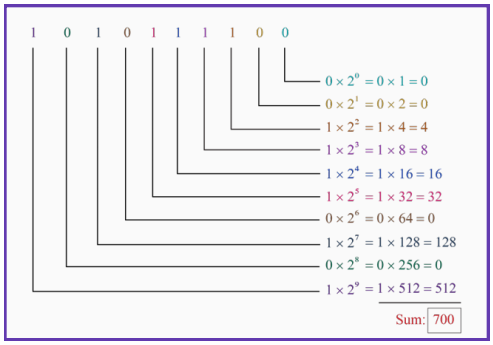
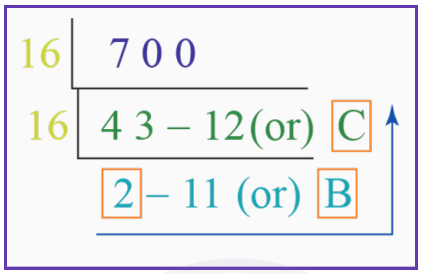


Thus, 10101111002 = 70010 → (1)

Step 2: Convert the above number (which is in the decimal system), into the required number system (hexadecimal).

Here, we have to convert 70010 into the hexadecimal system using the above-mentioned process. It should be noted that in the hexadecimal system, the numbers 11 and 12 are written as B and C respectively.



Thus, 70010 = 2BC16 → (2)
From the equations (1) and (2), 10101111002 = 2BC16

Bit manipulation:

This can be defined as the process of applying logical operations on sequence of bits where bits is the smallest form of data which is used to store information in a computer.
Broadly bits are either 0 or 1. Combination of these two digits is the foundation of bit manipulation.
Bit manipulation is used because it nearly consumes constant time for operations and is very efficient on all systems.
The most useful bits operator that we will cover in the lecture are as follows:

- AND (&)
- OR (|)
- NOT (!)
- XOR (^)
- Left shift (<<)
- Right shift (>>)

1. AND operator:
Symbol : &
This is a binary operator that operates on two operands. The numbers are converted in their binary format and corresponding bits of both the operands are operated.
This results in 1 if both the bits are 1. Otherwise if any of the bit in both operands is 0 the resultant of this AND

operator is 0.

The table for AND operator is shown below:

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

For example : let a = 5 and b = 7
Let's perform a&b
a = 101
b = 111
———————
101 which is equivalent to 5 in decimal.

2. OR operator:
Symbol : |
This is a binary operator that operates on two operands. The numbers are converted in their binary format and corresponding bits of both the operands are operated.
This results in 0 if both the bits are 0. Otherwise if any of the bits in both operands is 1 the resultant of this OR operator is 1.

The table for OR operator is shown below:

Input A	Input B	OR Operator A | B
0	0	0
0	1	1
1	0	1
1	1	1

For example : let a = 5 and b = 7
Let's perform a|b
a = 101
b = 111
———————
111 which is equivalent to 7 in decimal.

3. NOT operator:
Symbol : !
This is a unary operator which negates the effect of the input bit. If the bit inputted is 0 it is converted to 1, if the

bit inputted is 1 it is converted to 0.

x	x'
0	1
1	0

Example : let a = 4,
Let's find out !a
4 in binary would be 100, its complement / not would be to convert every 0 to 1 and every 1 to 0.
!a = 011 = 3 in decimal.

4. XOR operator:
Symbol : ^
This is a binary operator that operates on two operands. The numbers are converted in their binary format and corresponding bits of both the operands are operated.
This results in 0 if both the bits are same i.e either both the bits are 0 or both the bits are 1. Otherwise if the bits in both operands are different the resultant of this XOR operator is 1.

The table for XOR operator is shown below:

A	B	A **XOR** B
0	0	0
0	1	1
1	0	1
1	1	0

For example : let a = 5 and b = 7
Let's perform a|b
a = 101
b = 111
———————
010 which is equivalent to 2 in decimal.

5. Left shift operator:
Symbol : <<
This is a binary operator that works on two operands and left shifts the bits of the first operand, the second operand decides the number of places to shift.
For example, let num = 1100110010 and we want to left shift it by 1 bit.
The num after shift will be 11001100100
If we want to left shift it by 2 bits. Then num would be 110011001000
Observation: 1100110010 = 818 in decimal
And after left shift by 1 bit, 11001100100 = 1636 in decimal = 818 *(2^1)
After left shift by 2 bits, 110011001000 = 3272 = 818 * (2^2)

From here we can conclude, shifting an integer “x” with an integer “y” denoted as ‘(x<<y)’ is equivalent to multiplying x with 2^y (2 raised to power y).

For example : 4 << 3 = 4 * 2^3 = 4 * 8 = 32

Note: In an arithmetic shift, the sign bit is extended to preserve the signedness of the number as to prevent overflow

6. Right shift operator:
Symbol : >>
This is a binary operator that works on two operands and right shifts the bits of the first operand, the second operand decides the number of places to shift.
For example, let num = 1100110010 and we want to right shift it by 1 bit.
The num after shift will be 110011001
If we want to right shift it by 2 bits. Then num would be 11001100
Observation: 1100110010 = 818 in decimal
And after right shift by 1 bit, 0110011001 = 409 in decimal = 818 *(2^(-1))
After left shift by 2 bits, 0011001100 = 204= 818 * (2^(-2))

From here we can conclude, shifting an integer “x” with an integer “y” denoted as ‘(x>>y)’ is equivalent to multiplying x with 1/(2^y)(2 raised to power negative y).

Another point of observation here will be to continue right shifting each time on the right end we are getting the bit of the number and one bit from the number is removed. That indicates, if we want to get each bit of the number one by one we can continually right shift till the number does not become 0.

For example : 40 << 3 = 40 * 1/(2^3) = 40 * 1/8 = 5

We will solve some problems using bits manipulation in assignment.

Next Class teasers:

- Introduction to Recursion
- Factorial & Fibonacci problems
- Power of number using bits
- Count number of stairs