Bit Manipulation

Bit manipulation is basically the act of algorithmically manipulating bits (smallest unit of data in a computer).
A bit represents a logical state with one of two possible values ‘0’ or ‘1’, which in other representations can be referred to as ‘true’ or ‘false’ / ‘on’ or ‘off’.
Since bits represent logical states efficiently, this makes the binary system suitable for electronic circuits that use logic gates and this is why binary is used in all modern computers.

Decimal and Binary number systems

- **Decimal number system:**

  We usually represent numbers in decimal notation(base 10) that provides ten unique digits - 0,1,2,3,4,5,6,7,8,9. To form any number, we can combine these digits to form a sequence such that each decimal digit represents a value multiplied by a power of 10 in this sequence.
  For example:
  244 = 200 + 40 + 4 = 2*10^2 + 4*10^1 + 4*10^0.

- **Binary number system:**

  The binary system works in a similar manner to that of the decimal number system, the only difference lies in the fact that binary notation(base 2) provides two digits - 0 and 1. A binary number is defined as a sequence of 1s and 0s like ‘010100’, ‘101’, ‘0’,’1111’,’000111’, etc. In a binary number, each digit is referred to as a bit, where each such bit represents a power of decimal 2.
  Similar to decimal notation we can convert a given binary sequence to its decimal equivalent.
  For example:
  (base 2)1100: (Base 10) 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0 = 8 + 4 + 0 + 0 = 12.
  A bit is said to be **set** if it is ‘1’ and **unset** if the bit is ‘0’.

Bitwise Operators

Bitwise operators are used to performing bitwise operations. A bitwise operation operates on a bit string(string consisting of 0s and 1s only) or a bit array(array of 0s and 1s only) at the level of its individual bits. Bit operations are usually fast and are used for calculations and comparisons at the processor level.

1. **Bitwise AND(&)**
   A **binary operator** that takes bit representation of its operands and the resulting bit is 1 if both the bits in bits patterns are 1 otherwise, the resulting bit is 0.
   For example:
   X = 5 $(101)_2$ and Y = 3 $(011)_2$
   X & Y = $(101)_2$ & $(011)_2$ = $(001)_2$ = 1

2. **Bitwise OR(|)**
   A **binary operator** that takes bit representation of its operands and the resulting bit is 0 if both the bits in bits patterns are 0 otherwise, the resulting bit is 1.
   For example:
   X = 5 $(101)_2$ and Y = 3 $(011)_2$
   X | Y = $(101)_2$ | $(011)_2$ = $(111)_2$ = 7

3. **Bitwise NOT(~)**
   A **unary operator** that takes bit representation of its operand and just flips the bits of the bit pattern, i.e if the ith bit is 0 then it will be changed to 1 and vice versa.
   Bitwise NOT is the **1's complement** of a number.
   For example:
   X = 5 $(101)_2$
   ~X = $(010)_2$ = 2

4. **Bitwise XOR(^)**
   A **binary operator** that takes bit representations of its operands and the resulting bit is 0 if the bits in bit patterns are the same i.e the bits in both the patterns at a given position is either 1 or 0 otherwise, the resulting bit is 1 if the bits in bit patterns are different.
   For example:
   X = 5 $(101)_2$ and Y = 3 $(011)_2$
   X ^ Y = $(101)_2$ ^ $(011)_2$ = $(110)_2$ = 6

5. **Left shift operator(<<)**
   A **binary operator** that left shifts the bits of the first operand, with the second operand deciding the number of places to shift left, and append that many 0s at the end. We discard the k left most bits.
   For example:
   X = 5 $(101)_2$
   X << 2 = $(00101)_2$ << 4 = $(10100)_2$ = 20
   In other words, left shift by k places is equivalent to multiplying the bit pattern by 2^k, as shown in the above example - 5 << 2 = 5 * 2^2 = 20.
   So, A << x = A * 2^x that is why (1<< n) is equal to power(2,n) or 2^n.

6. **Right shift operator(>>)**
   A **binary operator** that right shifts the bits of the first operand, with the second operand deciding the number of places to shift right. We discard the k rightmost bits.
   For example:
   X = 5 $(101)_2$
   X >> 2 = $(101)_2$ = $(001)_2$ = 1
   In other words, the right shift by k places is equivalent to dividing the bit pattern by 2^k (integer division), as shown in the above example - 5 >> 2 = 5 / (2^2) = 1.
   So, A >> x = A / (2^x) (integer division).

X	Y	X & Y	X | Y	X ^ Y	~X
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0