

- Binary representation of Negative Numbers
 - Negative numbers are represented in 2's complement form.
 - Range of numbers: $[-2^{m-1} \text{ to } 2^m - 1]$
 - Steps to get 2's complement: (1) invert all bits (2) Add 1
 - Most formula = $2^m - x$

Why 2's complement form? → other option → signed bit → 1's complement

- We have only one representation of zero.
- The arithmetic operations are easier to perform. Actually 2's complement form is derived from the idea of 0-x.
- The leading bit is always 1. ($2^{m-1} \rightarrow$ leading bit always 0) for two numbers

Note: 1) The first bit in a signed representation is the sign of number, 0 for non-negative numbers and 1 for negative numbers, and the remaining ($m-1$) bits contain the magnitude of the number.

2) Two's complement is used for negative numbers.

3) In an unsigned representation, only non-negative numbers can be used, but the upper bound for values is higher.

Conversion between two representations:

A signed $-x$ equals to an unsigned $2^m - x$.

If a number is larger than the upper bound of bit representation, the number will overflow.

In a signed representation, next no after 2^{m-1} is -2^{m-1} .

In an unsigned representation, next no after 2^{m-1} is 0.

② Bitwise Operators in Python

<code>print(bin(18))</code>	→ 0b10010	Special prefixes for all bases: → for integer decimal representation → for binary representation → for octal → for hexadecimal
<code>print(bin(12))</code>	→ 0b1100	
<code>print(int("0b10010", 2))</code> → 18		

Bitwise AND (&)

LeftShift operator (<<)

RightShift operator (>>)

Bitwise Not (~)

operator	operations	Result
XOR	$X \wedge 0s$	X
XOR	$X \wedge 1s$	$\sim X$
XOR	$X \wedge X$	0

operator	operations	Result
AND	$X \& 0s$	0
AND	$X \& 1s$	X
AND	$X \& X$	X

Here, 0s or 1s mean a sequence of 0s or 1s.

eg: $X=6 \Rightarrow 0110$	eg: $X=7 \Rightarrow 0111$	eg: $X=5 \Rightarrow 0101$
$\begin{array}{r} 0110 \\ 0110 \\ + 0000 \\ \hline 0110 \\ \underline{\underline{X}} \end{array}$	$\begin{array}{r} 0111 \\ 0111 \\ + 0000 \\ \hline 0111 \\ \underline{\underline{X}} \end{array}$	$\begin{array}{r} 0101 \\ 0101 \\ + 0000 \\ \hline 0101 \\ \underline{\underline{X}} \end{array}$

operator	operations	Result
OR	$X 0s$	0
OR	$X 1s$	1s
OR	$X X$	X

③ Get Bit

- Find the bit at a particular position of given number N.
- Find bitwise AND of given number and $2^{k-1} (1 << (k-1))$
- If value is 1 then bit is set else.

$$(num \& (1 << k-1)) != 0 \text{ or } ((num >> (k-1)) \& 1) \neq 0$$

at k^{th} position

Set Bit

- Method is used to set the bit at a particular position (say K).
- Idea is update the value of given number N to the bitwise OR of N and $2^{K-1} (1 << (k-1))$.
- If value is 1 then bit is set.

$$(num | (1 << k-1))$$

Clear Bit

- Method used to clear bit at k^{th} position of a number N.
- Update the given number N to bitwise AND of N and complement of 2^{k-1} i.e. $\sim (1 << (k-1))$.
- mask = $\sim (1 << (k-1))$
- return num & mask

Toggling a bit at k^{th} position

- use XOR because $0 \wedge 0 = 1 \wedge 0 = 0$ and $0 \wedge 1 = 1 \wedge 0 = 1$
- If two bits are different then XOR operator returns a set bit (1) else it returns an unset bit (0).

$$\text{num} = \text{num} \wedge (1 << (k-1))$$

Inverting every bit of a number / 1's complement

- Using NOT operator \Rightarrow return $(\sim \text{num})$

2's complement of a number

- 1's complement + 1 or using \sim operator

$$(\sim \text{num} + 1) \text{ or } (\sim \text{num})$$

Stripping of lowest set bit (Brian Kernighan's)

$$X = X \& (X-1)$$

- $(X-1)$ Inverts all bits till it encounters the lowest set '1' and also inverts the lowest set '1'.
- After adding X and $(X-1)$, we get lowest set bit stripped.

Getting the lowest set bit

$$\text{num} = \text{num} \& (\sim \text{num})$$

- If we AND original number X with 2's complement which is $\sim X$, we get lowest set bit.

$$\text{eg: } X = 12 \Rightarrow 1100$$

$\begin{array}{r} 1100 \\ 0100 \\ \hline 0100 \end{array}$

1's comp $\Rightarrow 0011 \Rightarrow$ 2's comp $\Rightarrow 0100$

Divide by 2 using right shift operator

$$\text{num} \gg 1$$

Multiply by 2 using left shift

$$\text{num} \ll 1$$

Bit Ticks

- Clearing all bits from Lsb to k^{th} position
 - mask = $\sim (1 << (i-1)) - 1$
 - $X \&= \text{mask}$

Logic: to clear all bits from Lsb to k^{th} bit, we have to AND x with mask having Lsb to k^{th} bit 0.

→ to obtain such mask, first left shift 1 i times.

→ Now, if we minus 1, all bits from 0 to $i-1$ becomes 1 and remaining bits become 0.

→ Now, simply taking complement of mask to get all first i bits to 0 and remaining to 1.

- Clearing all bits from k^{th} to l^{th} bit

$$\text{mask} = (1 << l) - 1$$

$X \&= \text{mask}$

- Multiply / Divide by 2 \Rightarrow left shift / right shift by 1

- Uppercase to lowercase alphabet

Notes: If we set 5th bit of uppercase character, then it will be converted to lowercase character.

$$\begin{array}{l} A \rightarrow 01000001 \\ \vdots \\ Z \rightarrow 01011010 \end{array}$$

mask = 00100000 represent space (' ')

Character end with mask will convert to lowercase

$$ch = ch \& 1;$$

- Lowercase to uppercase alphabet

mask = 1101111 represent underscore ('_')

$$ch = ch \& \sim 1;$$

- Count set bits

Brian Kernighan's Algorithm

→ Subtracting 1 from decimal number flips all the bits after the rightmost set bit (which is 1) including rightmost set bit.

→ So if we subtract 1, and do it bitwise and with itself ($n \& (n-1)$), we get rightmost bit.

→ If we do $n \& (n-1)$ in a loop and count the number of times the loop executes, we get set bit counts.

- Compute XOR from 1 to n

If $n/2^i$ is 0 : return n

If $n/2^i$ is 1 : return 1

If $n/2^i$ is 2 : return $n+1$

If $n/2^i$ is 3 : return 0

- Count of numbers (n) smaller than equal to n such that $n+x = n^2x$

mathematical trick: $\text{count} = \text{pow}(2, \text{count of zero bits})$

We know $\rightarrow n+x = (n^2)^2 + (n^2)$

→ total values of i such that $n^2 = 0$.

→ we can count zero bits in the binary representation of n.

→ i must consist all set-bits of n.

→ If 2^m bit is set in n, then k^m bit in i must be 0.

else k^m bit can be 0 or 1.

→ Hence total combinations are $2^{\text{count of zero bits in n}}$

- How to check if a number is power of 2?

If a no. is power of 2, then bitwise AND of N and $N-1$ will be 0.

So, just check these two conditions, (1) If N is not 0.

(2) $N \& (N-1) == 0$.

between n and $(\text{not}(X \& (X-1)))$

- Find XOR of all subsets of a set

→ ans is 0 if given set has more than one element.

→ for set with single element, answer is value of single element.

- Quick Way to swap two numbers

$$\begin{array}{l} a^1 = b \\ b^1 = a \\ a^1 = b \end{array} \quad \begin{array}{l} a = a^1 b \\ b = b^1 (a^1 b) = a \\ a = (a^1 b)^1 (a) = b \end{array}$$

- Find most significant set bit

$$k = \text{int}(\text{math.log}(n/2))$$

return $1 < k$

- Check if a number has bits in an alternate pattern

→ compute bitwise XOR of n and $n \gg 1</$