

BINARY

Bit Manipulation in C

- **Operators**

& AND **|** OR **^** XOR **~** NOT **<<** LEFT SHIFT **>>** RIGHT SHIFT

- **Common Operations**

set bit: `num |= (1 << pos)`

clear bit: `num &= ~(1 << pos)`

toggle bit: `num ^= (1 << pos)`

check bit: `(num & (1 << pos)) != 0`

extract bit: `(num >> pos) & 1`

extract a range of bits: `(num >> pos) & ((1 << length) - 1)`

- **Example**

```
void copyBit(int *dst, int src, int srcPos, int dstPos) {  
    int bit = (src >> srcPos) & 1; // extract bit  
    *dst &= ~(1 << dstPos); // clear destination bit  
    *dst |= (bit << dstPos); // set destination bit  
}
```

Binary

- In C++, **std::bitset** represents a fixed-size sequence of N bits

- Example:

```
std::bitset<8> bitmap;
```

```
bitmap.reset(1)
```

```
bitmap.set(1)
```

```
if (bitmap.test(1)) { // true
```

```
...
```

- **reset** : set bit to false
- **set** : set a specific bit
- **test** : check a specific bit
- **count** : return the number of bits set to true
- **flip** : toggle the value of the bits (if true, set to false and vice-versa)

Problem – 371. Sum of Two Integers

Medium



LeetCode

leetcode.com/problems/sum-of-two-integers

Problem

- Sum two integer numbers **a** and **b**
- You can't use **+** or **-**

Solution – 371. Sum of Two Integers

Medium



LeetCode

leetcode.com/problems/sum-of-two-integers

Solution

- **Example:**

$a = 101$ and $b = 110$

expected result = $101 + 110 = 1011$

- **Find the position where carry occurs using and operation:**

$101 \& 110 = 100$

- **Sum without carry using XOR:**

$101 \wedge 110 = 011$

- **Shift left the carry:**

$100 \ll 1 = 1000$

- **Add the previous sum to the carry**

$011 \wedge 1000 = 1011$

- **Check the carry from the previous operation:**

$011 \wedge 1000 = 0000$

- **Now if it is zero, the result is 1011. Otherwise, repeat the process**

Code – 371. Sum of Two Integers

Medium



LeetCode

leetcode.com/problems/sum-of-two-integers

Code Time: **O(1)** Space: **O(1)**

```
int getSum(int a, int b) {
    while (b != 0) {
        // find which bits produce a carry
        // e.g. 101 & 111 = 101
        int carry = static_cast<unsigned>(a & b);
        // adds without carry (XOR)
        // 101 ^ 111 = 010
        a = a ^ b;
        // move carry to the left
        // b = 110
        b = carry << 1;
    }
    return a;
}
```

Problem – 191. Number of 1 Bits

Easy



LeetCode

leetcode.com/problems/number-of-1-bits

Problem

- You are given a positive integer n
- Return the number of set bits
- **Example:**

$n = 11$

output: 3

Binary is 1011, hence three set bits "1"

Solution – 191. Number of 1 Bits

Easy



LeetCode

leetcode.com/problems/number-of-1-bits

Solution

- Use Brian Kernighan's Bit Counting Algorithm
- removes the rightmost 1-bit from n in each iteration
- Example:

$n = 12$ (1100)

First iteration: $n=1100$, $n-1=1011$, so $1100 \& 1011 = 1000$ (removed rightmost 1)

Second iteration: $n=1000$, $n-1=0111$, so $1000 \& 0111 = 0000$ (removed last 1)

Result: 2 iterations = 2 ones counted

Code – 191. Number of 1 Bits

Easy



LeetCode

leetcode.com/problems/number-of-1-bits

Code

```
// Brian Kernighan's algorithm
int hammingWeight(int n) {
    // return std::popcount(n);
    int count = 0;
    while (n) {
        n &= (n - 1);
        ++count;
    }
    return count;
}
```

Problem – 338. Counting Bits

Easy



LeetCode

leetcode.com/problems/counting-bits

Problem

- You are given an integer n representing n different numbers starting from 0
- Return an array with the number of 1 for each integer

- **Example:**

$n = 3$

$0 \rightarrow 0$ (zero '1's)

$1 \rightarrow 1$ (one '1')

$2 \rightarrow 10$ (one '1')

$3 \rightarrow 11$ (two '1's')

Output: $[0, 1, 1, 2]$

Solution – 338. Counting Bits

Easy



LeetCode

leetcode.com/problems/counting-bits

Solution

- One solution is to code **Brian Kernighan's algorithm** and then call it **multiple times; or**
- **Use previously calculated results:** the bit count of any number i equals the bit count of $i/2$ plus 1 if the last bit is 1
- `res[i] = res[i >> 1] + (i & 1)`
 - `i >> 1` removes the rightmost bit (divides by 2)
 - `i & 1` checks if the rightmost bit is 1

Code – 338. Counting Bits

Easy



LeetCode

leetcode.com/problems/counting-bits

Code Time: $O(n)$ Space: $O(n)$

```
vector<int> countBits(int n) {  
    vector<int> res(n + 1, 0);  
    for (int i = 1; i <= n; ++i) {  
        // i >> 1 = i is divided by 2 (i with the last bit removed)  
        // i & 1 is 1 if the last bit is set  
        // number of set bits in i = number of set bits in i / 2 + 1 if last bit is 1.  
        res[i] = res[i >> 1] + (i & 1);  
    }  
    return res;  
}
```

Problem – 268. Missing Number

Easy



LeetCode

<https://leetcode.com/problems/missing-number>

Problem

- You are given an **array of nums** with **n** distinct numbers
- Return only the number in the range that is missing from the array

- **Example:**

nums = [3, 0, 1]

Output: 2

from the sequence 0,1,x,3 the missing number x is 2

Problem – 268. Missing Number

Easy



LeetCode

<https://leetcode.com/problems/missing-number>

Solution

- ...

Problem – 268. Missing Number

Easy



LeetCode

<https://leetcode.com/problems/missing-number>

Code Time: $O(-)$ Space: $O(-)$

■ ...

Problem – 190. Reverse Bits

Easy



LeetCode

leetcode.com/problems/reverse-bits

Problem

■ ...

Problem – 190. Reverse Bits

Easy



LeetCode

leetcode.com/problems/reverse-bits

Solution

- ...

Problem – 190. Reverse Bits

Easy



LeetCode

leetcode.com/problems/reverse-bits

Code Time: $O(-)$ Space: $O(-)$

■ ...

Negabinary

- Non-standard positional numeral system that uses base of -2
- Allow representing negative numbers in binary
- Example:

1101_{-2}

$$(-2)^3 + (-2)^2 + 0 + (-2)^0 = -8 + 4 + 0 + 1 = -3$$

Summing Negabinary

- Add as a regular binary number, but with **negative carry**

$$0 + 0 = 0$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \quad \text{with a negative carry 1}$$

$$\mathbf{1} + 1 = 0 \quad (\text{subtract})$$

$$\mathbf{1} + 0 = 1 \quad \text{with a positive carry 1}$$

Negabinary

Example 1

$$\begin{array}{r} 11\ 1 \\ 1011 \\ + 1110 \\ \hline = 110001 \end{array}$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ with negative carry } 1$$

$$1 + 1 = 0$$

$$1 + 1 = 0 \text{ with negative carry } 1$$

$$1 + 0 = 1 \text{ with positive carry } 1$$

$$1 + 0 = 1$$

red 1 = negative carry

green 1 = regular carry

Example 2

$$\begin{array}{r} 1111 \\ 101010 \\ + 101100 \\ \hline = 11110110 \end{array}$$

Reference

<https://math.stackexchange.com/questions/3251605/how-to-add-negabinary-numbers>

Problem 1073 – Adding Two Negabinary Numbers

Medium

<https://leetcode.com/problems/adding-two-negabinary-numbers>

Given two numbers `arr1` and `arr2` in base -2, return the result of adding them together.

Each number is given in *array format*: as an array of 0s and 1s, from most significant bit to least significant bit. For example, `arr = [1,1,0,1]` represents the number $(-2)^3 + (-2)^2 + (-2)^0 = -3$. A number `arr` in array, format is also guaranteed to have no leading zeros: either `arr == [0]` or `arr[0] == 1`.

Return the result of adding `arr1` and `arr2` in the same format: as an array of 0s and 1s with no leading zeros.

Example 1

Input: `arr1 = [1,1,1,1,1]`, `arr2 = [1,0,1]`

Output: `[1,0,0,0,0]`

Explanation: `arr1` represents 11, `arr2` represents 5, the output represents 16.

Example 2

Input: `arr1 = [0]`, `arr2 = [0]`

Output: `[0]`

Example 3

Input: `arr1 = [0]`, `arr2 = [1]`

Output: `[1]`

Solution 1073 – Adding Two Negabinary Numbers

Medium

<https://leetcode.com/problems/adding-two-negabinary-numbers>