**231. Power of Two**

**Solution**

**Overview**

We're not going to discuss here an obvious O(log⁡N)\mathcal{O}(\log N)O(log*N*) time solution

class Solution {

public boolean isPowerOfTwo(int n) {

if (n == 0) return false;

while (n % 2 == 0) n /= 2;

return n == 1;

}

}

Instead, the problem will be solved in O(1)\mathcal{O}(1)O(1) time with the help of bitwise operators. The idea is to discuss such bitwise tricks as

* How to get / isolate the rightmost 1-bit : x & (-x).
* How to turn off (= set to 0) the rightmost 1-bit : x & (x - 1).

These tricks are often used as something obvious in more complex bit-manipulation solutions, like for [N Queens problem](https://leetcode.com/articles/n-queens-ii/), and it's important to recognize them to understand what is going on.

**Intuition**

The idea behind both solutions will be the same: a power of two in binary representation is one 1-bit, followed by some zeros:

1=(00000001)21 = (0000 0001)\_21=(00000001)2​

2=(00000010)22 = (0000 0010)\_22=(00000010)2​

4=(00000100)24 = (0000 0100)\_24=(00000100)2​

8=(00001000)28 = (0000 1000)\_28=(00001000)2​

A number which is not a power of two, has more than one 1-bit in its binary representation:

3=(00000011)23 = (0000 0011)\_23=(00000011)2​

5=(00000101)25 = (0000 0101)\_25=(00000101)2​

6=(00000110)26 = (0000 0110)\_26=(00000110)2​

7=(00000111)27 = (0000 0111)\_27=(00000111)2​

The only exception is 0, which should be treated separately.

**Approach 1: Bitwise Operators : Get the Rightmost 1-bit**

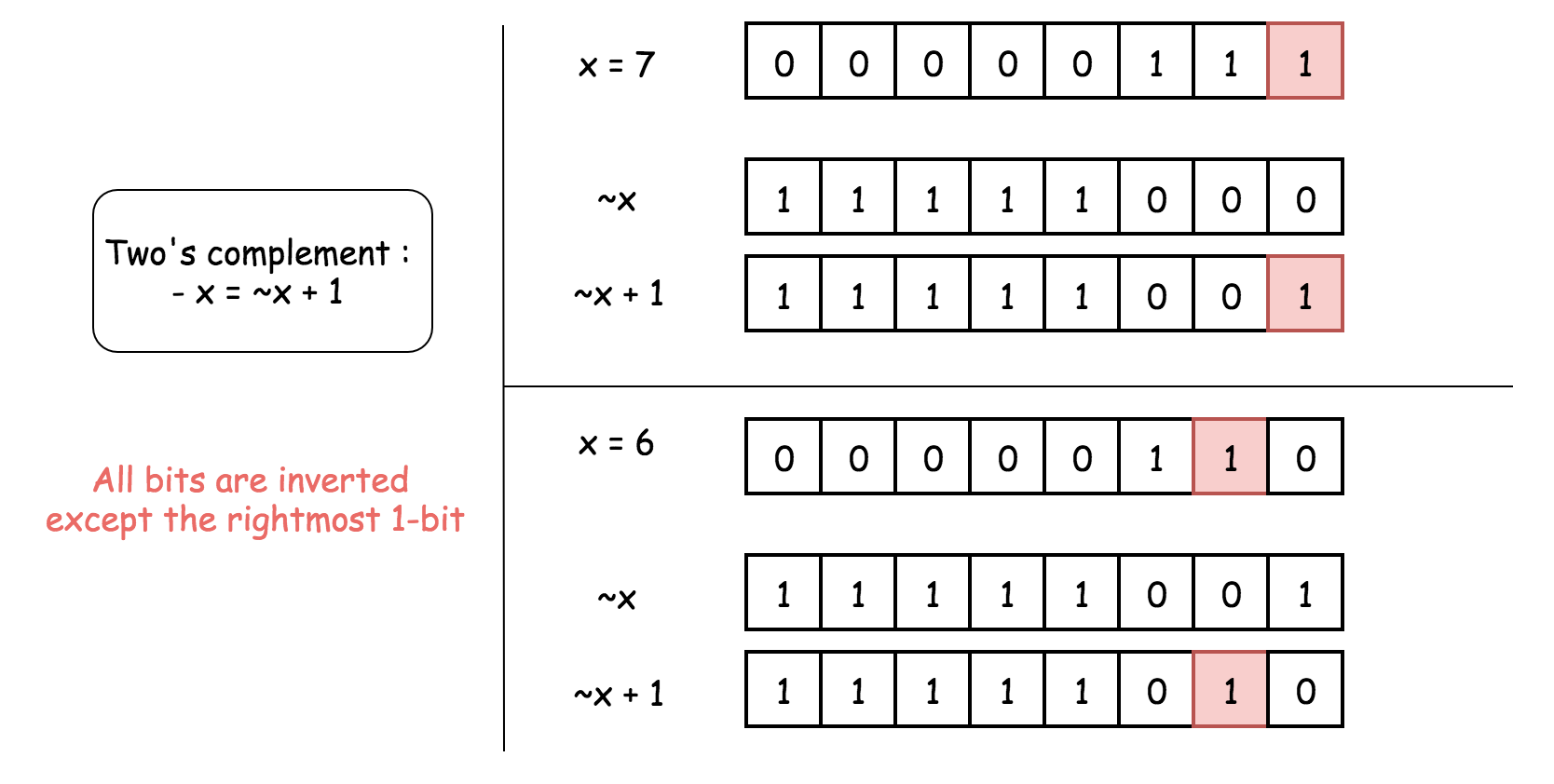
**Get/Isolate the Rightmost 1-bit**

Let's first discuss why x & (-x) is a way to keep the rightmost 1-bit and to set all the other bits to 0.

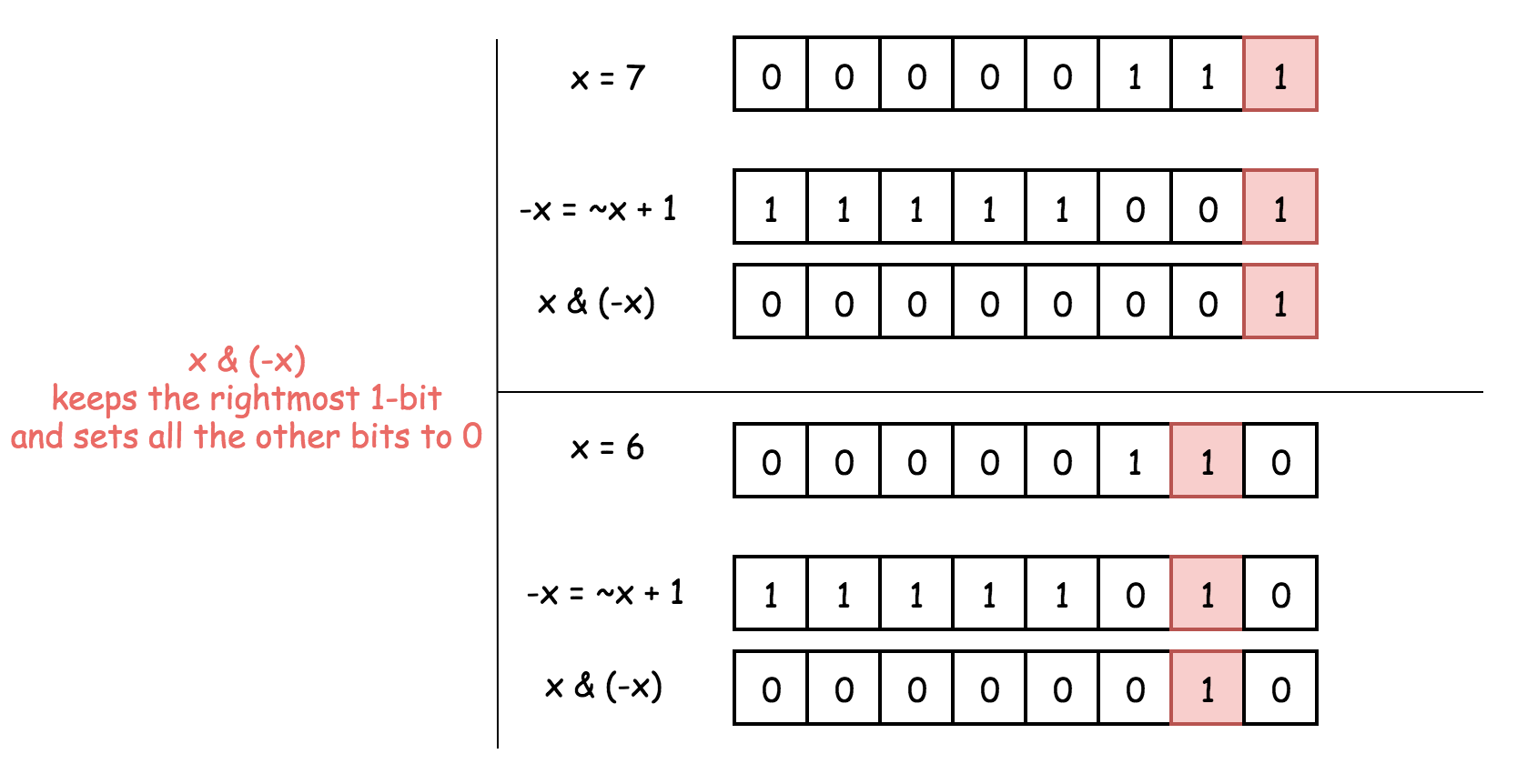
Basically, that works because of [two's complement](https://en.wikipedia.org/wiki/Two%27s_complement). In two's complement notation −x-x−*x* is the same as ¬x+1\lnot x + 1¬*x*+1. In other words, to compute −x-x−*x* one has to revert all bits in xx*x* and then to add 1 to the result.

Adding 1 to ¬x\lnot x¬*x* in binary representation means to carry that 1-bit till the rightmost 0-bit in ¬x\lnot x¬*x* and to set all the lower bits to zero. Note, that the rightmost 0-bit in ¬x\lnot x¬*x* corresponds to the rightmost 1-bit in xx*x*.

In summary, −x-x−*x* is the same as ¬x+1\lnot x + 1¬*x*+1. This operation reverts all bits of x except the rightmost 1-bit.



Hence, x and -x have just one bit in common - the rightmost 1-bit. That means that x & (-x) would keep that rightmost 1-bit and set all the other bits to 0.

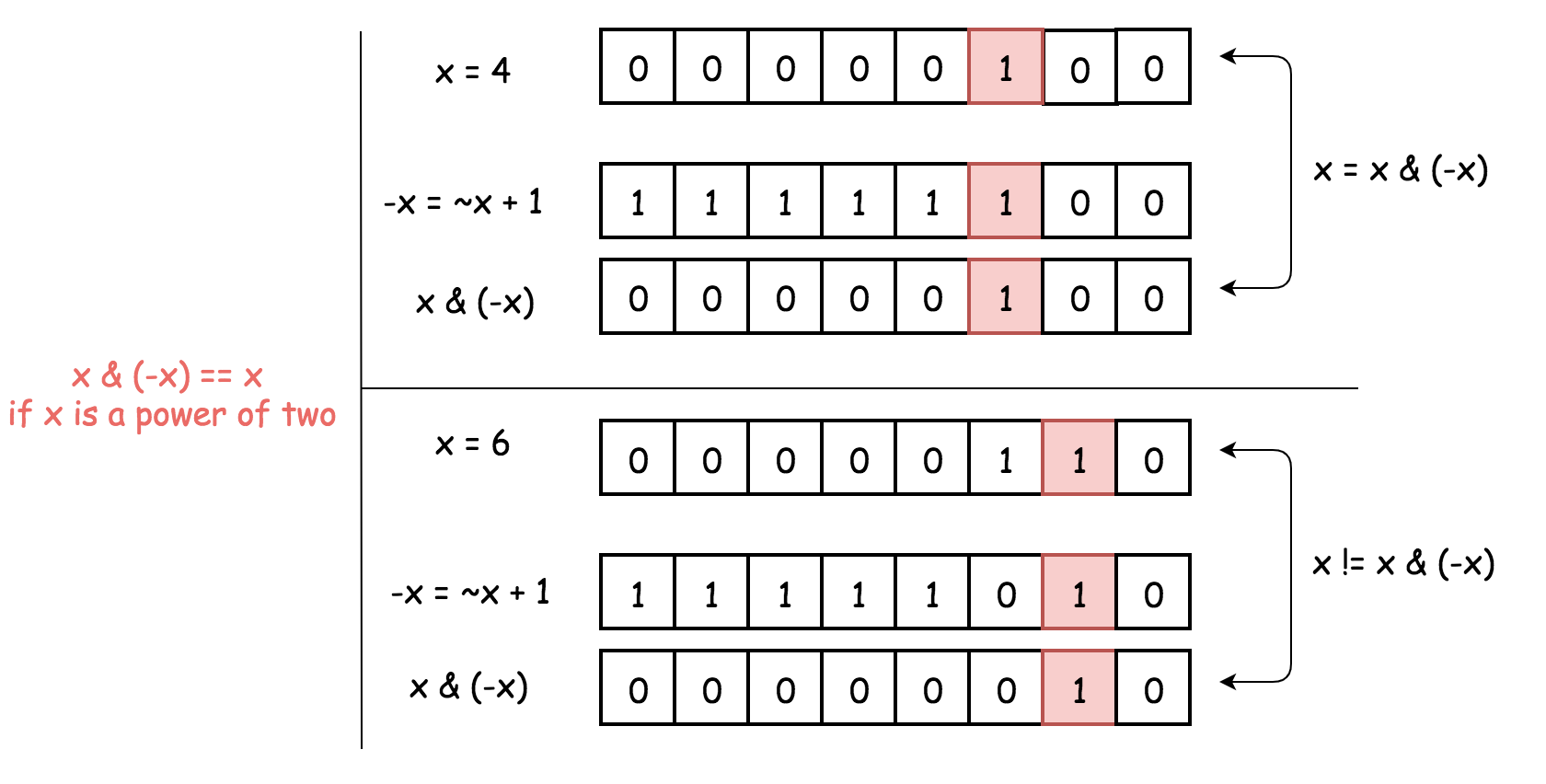


**Detect Power of Two**

So let's do x & (-x) to keep the rightmost 1-bit and to set all the others bits to zero. As discussed above, for the power of two it would result in x itself, since a power of two contains just one 1-bit.

Other numbers have more than 1-bit in their binary representation and hence for them x & (-x) would not be equal to x itself.

Hence a number is a power of two if x & (-x) == x.



**Implementation**

class Solution {

public:

bool isPowerOfTwo(int n) {

if (n == 0) return false;

long x = n;

return (x & (-x)) == x;

}

};

**Complexity Analysis**

* Time complexity : O(1)\mathcal{O}(1)O(1).
* Space complexity : O(1)\mathcal{O}(1)O(1).

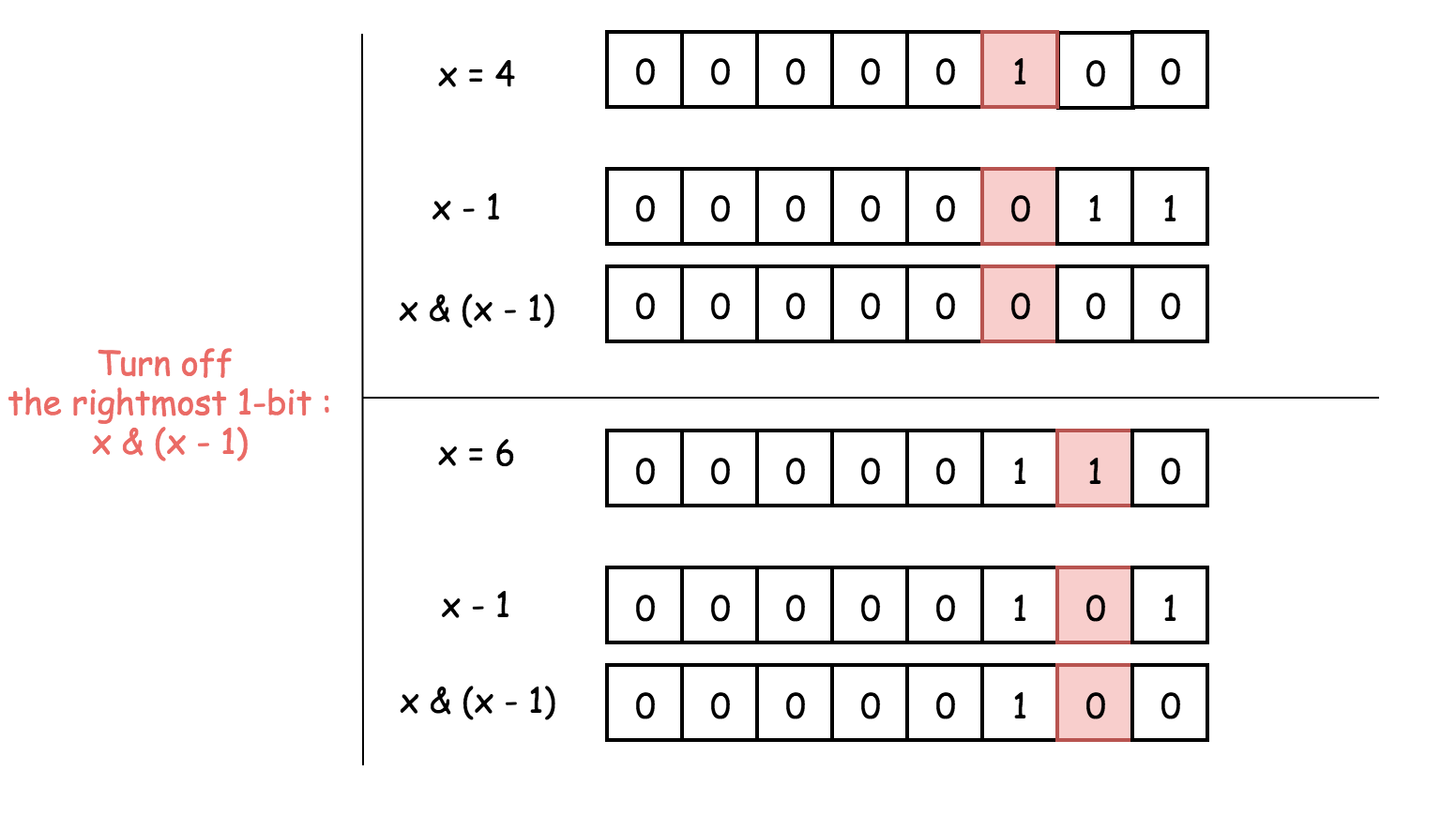
**Approach 2: Bitwise operators : Turn off the Rightmost 1-bit**

**Turn off the Rightmost 1-bit**

Let's first discuss why x & (x - 1) is a way to set the rightmost 1-bit to zero.

To subtract 1 means to change the rightmost 1-bit to 0 and to set all the lower bits to 1.

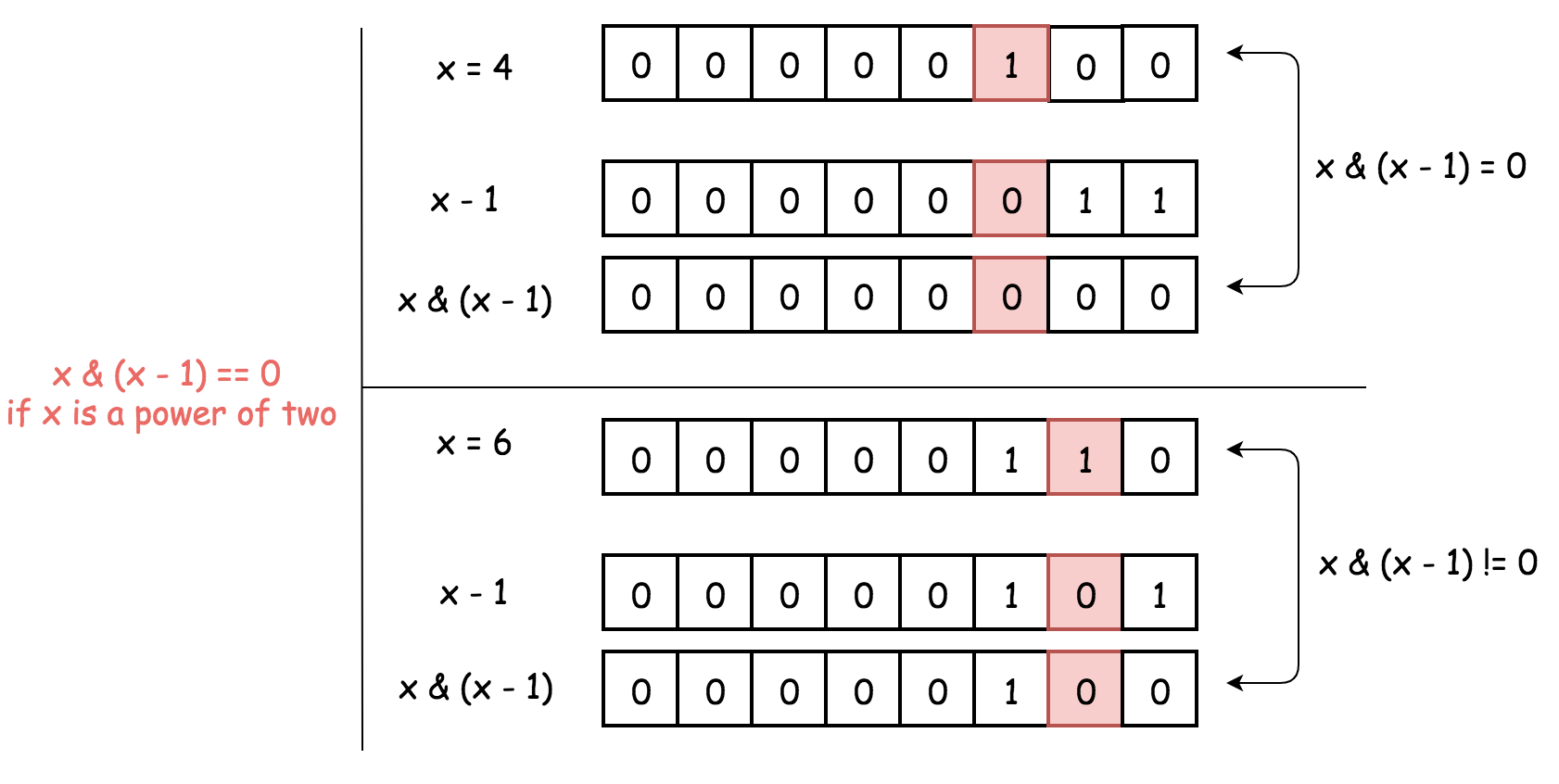
Now AND operator: the rightmost 1-bit will be turned off because 1 & 0 = 0, and all the lower bits as well.



**Detect Power of Two**

The solution is straightforward:

1. Power of two has just one 1-bit.
2. x & (x - 1) sets this 1-bit to zero, and hence one has to verify if the result is zero x & (x - 1) == 0.



**Implementation**

class Solution {

public:

bool isPowerOfTwo(int n) {

if (n == 0) return false;

long x = n;

return (x & (-x)) == x;

}

};

**Complexity Analysis**

* Time complexity : O(1)\mathcal{O}(1)O(1).
* Space complexity : O(1)\mathcal{O}(1)O(1).