Bit Manipulation Interview Questions

Brush up on your bitwise operations

Under the hood, numbers are just bits set to 0 or 1. Try some of these common and trickier questions involving bit operations.

Test Bit

Given a number, write a function that tests if its i^{th} bit is set.

Python 3.6 ▾

```
def test_bit_set(number, index):
    '''
    Returns True if number has the index'th bit set
    and False otherwise.
    '''
```

> We'll say that the bits are numbered from the least significant bit (on the right) to the most significant bit (on the left).
>
> So, the binary number `0000 0001` has the 0^{th} bit set and all the rest of its bits are *clear* (not set).

Answer

We can test if the value has a specific bit set using a left shift with an and .

First, we'll create a mask by taking `1` and shifting it left until the set bit is at the index we want to test.

```
1 << 0  →  0000 0001   # for the 0th bit
1 << 1  →  0000 0010   # for the 1st bit
1 << 2  →  0000 0100   # for the 2nd bit
...
1 << 7  →  1000 0000   # for the 7th bit
```

Then, we'll `&` the shifted `1` with the value we're testing. If the result is zero, then the bit isn't set; otherwise, it is.

```
& 0101 1101
  0010 0000
-----------
  0000 0000
```

```
& 0101 1101
  0100 0000
-----------
  0100 0000
```

Here's an implementation in code:

Python 3.6 ▾

```
def test_bit_set(number, index):
    '''
    Returns True if number has the index'th bit set
    and False otherwise.
    '''
    mask = 1 << index
    return number & mask != 0
```

> You could squish this into a one-liner if you wanted. We tend to prefer clarity over brevity though. :)

Set Bit

Given a number, write a function that sets its i^{th} bit to 1.

Python 3.6 ▾

```
def set_bit(number, index):
    '''
    Set the index'th bit of number to 1, and return
    the result.
    '''
```

Answer

We can set a specific bit using a left shift with an or .

First, we'll make a mask by taking a 1 and shifting it left until the set bit is at the index we want to set.

```
1 << 0  →  0000 0001   # for the 0th bit
1 << 1  →  0000 0010   # for the 1st bit
1 << 2  →  0000 0100   # for the 2nd bit
...
1 << 7  →  1000 0000   # for the 7th bit
```

Then, we'll | the shifted 1 with the value. This sets the bit to 1, leaving all the other bits unchanged.

```
| 0101 1101
  0010 0000
-----------
  0111 1101
```

Here's an implementation in code:

Python 3.6 ▾

```
def set_bit(number, index):
    '''
    Set the index'th bit of number to 1, and return
    the result.
    '''
    mask = 1 << index
    return number | mask
```

> Again, this could be a one-liner if you wanted.

Clear Bit

Given a number, write a function that clears its i^{th} bit by setting it to `0`.

Python 3.6 ▾

```
def clear_bit(number, index):
    '''
    Set the index'th bit of number to 0, and return
    the result.
    '''
```

Answer

We can clear a specific bit set using a left shift , a not , and an and .

First, we'll make our mask by taking `1`, shifting it left until the set bit is at the index we want to clear, and not'ing the result. This makes a mask where every bit is set *except* for the one we want to clear.

```
~(1 << 0)  →  1111 1110   # for the 0th bit
~(1 << 1)  →  1111 1101   # for the 1st bit
~(1 << 2)  →  1111 1011   # for the 2nd bit
...
~(1 << 7)  →  0111 1111   # for the 7th bit
```

Then, we'll `&` the shifted `1` with the value we're testing. This clears the bit that we left as `0` and leaves all the other bits unchanged.

```
& 0101 1101
  1011 1111
-----------
  0001 1101
```

Here's an implementation in code:

Python 3.6 ▾

```
def clear_bit(number, index):
    '''
    Set the index'th bit of number to 0, and return
    the result.
    '''
    mask = ~(1 << index)
    return number & mask
```

Toggle Bit

Given a number, write a function that toggles its i^{th} bit. (If the bit is `1`, set it to `0`. If it's `0`, set it to `1`.

Python 3.6 ▾

```python
def toggle_bit(number, index):
    '''
    Toggle the index'th bit of number. (If it's 0, set it to
    1; if it's 1, set it to 0.)
    '''
```

Answer

We can set a specific bit using a left shift↴ with an exclusive or↴ .

First, we'll take `1` and shift it left until the set bit is at the index we want to set.

```
1 << 0  →  0000 0001   # for the 0th bit
1 << 1  →  0000 0010   # for the 1st bit
1 << 2  →  0000 0100   # for the 2nd bit
...
1 << 7  →  1000 0000   # for the 7th bit
```

Then, we'll `^` the shifted `1` with the value. If the bit was a `1`, then the `^` with a `1` sets it to zero. If the bit was a `0`, then the `^` with a `1` sets it to one. All the other bits are xor'd with zero, leaving them unchanged.

```
^ 0101 1101
  0010 0000
-----------
  0111 1101
```

```
^ 0101 1101
  0100 0000
-----------
  0001 1101
```

Here's an implementation in code:

Python 3.6 ▾

```
def toggle_bit(number, index):
    '''
    Toggle the index'th bit of number. (If it's 0, set it to
    1; if it's 1, set it to 0.)
    '''
    mask = 1 << index
    return number ^ mask
```

Single Bit Set

Given a number, write a function that determines if the number has exactly one bit set.

Python 3.6 ▾

```
def single_bit_set(number):
    '''
    Return True if number has exactly one bit set to 1; False
    if it has any other number of bits set to 1.
    '''
```

> Sometimes, you'll hear this problem framed in terms of powers of two: "Write a function that determines if a number is a power of two."
>
> All powers of two have exactly one bit set, so these questions are identical.

Answer

We can determine if a number has exactly one bit set with an and .

First, we'll take the number and subtract 1.

```
0100 0000 - 0000 0001   →   0011 1111
0000 1000 - 0000 0001   →   0000 0111
0101 0111 - 0000 0001   →   0101 0110
1110 1010 - 0000 0001   →   1110 1001
```

Notice how the subtraction clears the least-significant set bit and sets all the lower bits to `1`. Everything to the left of the least-significant set bit is unchanged.

```
                                  --- Least Significant Set Bit
                                 /
Original Number:   0  1  0  1    1  0  0  0

Original Number:   0  1  0  1    0  1  1  1
   Minus One       \--------/    |  \-----/
                   Unchanged     |  Set to 1
                                 |
                                 \--- Least significant set bit is cleared
```

Look what happens when we `&` `number` with `number - 1`:

```
    0  1  0  1    1  0  0  0     Original number
&   0  1  0  1    0  1  1  1     Original number minus one
-----------------------------
    0  1  0  1    0  0  0  0
    \--------/    |  \-----/
         |        |     |
         |    Set to 0  |
         |              |
         |--------------|
          Unchanged from
          original number
```

This *clears* the least-significant set bit and leaves the rest of the number unchanged.

If there is exactly one bit set, then the result of the `&` will be zero.

If there are multiple bits set, then the `&` will only clear the *lowest* bit, leaving the other bits set.

Here's how we'd write this in code:

Python 3.6 ▾

```
def single_bit_set(number):
    '''
    Return True if number has exactly one bit set to 1; False
    if it has any other number of bits set to 1.
    '''

    # Special case for zero
    if number == 0:
        return False
    return number & (number - 1) == 0
```

What's next?

Check out our mock coding interview questions (/next).

They mimic a real interview by offering hints when you're stuck or you're missing an optimization.

Try some questions now ➡

Want more coding interview help?

Check out **interviewcake.com** for more advice, guides, and practice questions.