

Finish [signing up](#) for your accountMake your voice heard. [Take the 2020 Developer Survey now.](#)

What USEFUL bitwise operator code tricks should a developer know about? [closed]

Asked 10 years, 4 months ago Active 2 months ago Viewed 30k times



63



108



Closed. This question is [opinion-based](#). It is not currently accepting answers.



Want to improve this question? Update the question so it can be answered with facts and citations by [editing this post](#).

Closed 2 years ago.

I must say I have never had cause to use bitwise operators, but I am sure there are some operations that I have performed that would have been more efficiently done with them. How have "shifting" and "OR-ing" helped you solve a problem more efficiently?

[language-agnostic](#)[bit-manipulation](#)[bit](#)

edited Mar 30 '11 at 6:39

[ming_codes](#)

2,542 19 24

asked Oct 7 '09 at 17:44

[non sequitor](#)

15.3k 8 40 59

Would you mind to change your accepted answer to choose CS's answer? – [Xam](#) Sep 12 '18 at 23:21

@Xam - CS's answer came in almost 4 yrs after Martin's and it was instructive to me at the time I needed it. So on principle I won't change it, but CS and Mohasin both benefit from the upvotes that make their answers more popular than Martin's. – [non sequitor](#) Sep 22 '18 at 19:51

11 Answers



42



See the famous [Bit Twiddling Hacks](#)

Most of the multiply/divide ones are unnecessary - the compiler will do that automatically and you will just confuse people.

But there are a bunch of, 'check/set/toggle bit N' type hacks that are very useful if you work with hardware or communications protocols.

edited Oct 8 '09 at 21:47

answered Oct 7 '09 at 18:38

[Martin Beckett](#)

86k 20 170 246

Finish [signing up](#) for your account

x

135 Convert letter to **lowercase**:

+100



- **OR** by space => $(x \mid ' ')$
- Result is always lowercase even if letter is already lowercase
- eg. $('a' \mid ' ') \Rightarrow 'a'$; $('A' \mid ' ') \Rightarrow 'a'$

Convert letter to **uppercase**:

- **AND** by underline => $(x \& '_')$
- Result is always uppercase even if letter is already uppercase
- eg. $('a' \& '_') \Rightarrow 'A'$; $('A' \& '_') \Rightarrow 'A'$

Invert letter's case:

- **XOR** by space => $(x \wedge ' ')$
- eg. $('a' \wedge ' ') \Rightarrow 'A'$; $('A' \wedge ' ') \Rightarrow 'a'$

Letter's **position** in alphabet:

- **AND** by $\text{chr}(31) / \text{binary}('11111') / (\text{hex}('1F')) \Rightarrow (x \& "\backslash x1F")$
- Result is in 1..26 range, letter case is not important
- eg. $('a' \& "\backslash x1F") \Rightarrow 1$; $('B' \& "\backslash x1F") \Rightarrow 2$

Get letter's **position** in alphabet (for **Uppercase** letters only):

- **AND** by $?$ => $(x \& '?')$ *or* **XOR** by $@$ => $(x \wedge '@')$
- eg. $('C' \& '?') \Rightarrow 3$; $('Z' \wedge '@') \Rightarrow 26$

Get letter's **position** in alphabet (for **lowercase** letters only):

- **XOR** by backtick/ $\text{chr}(96) / \text{binary}('1100000') / \text{hex}('60') \Rightarrow (x \wedge '`')$
- eg. $('d' \wedge '`') \Rightarrow 4$; $('x' \wedge '`') \Rightarrow 25$

Note: using anything other than the english letters will produce garbage results

answered Apr 23 '13 at 17:51



CSφ

9,293 9 35 59

3 How did you know i would be interested :) – [Baba](#) Apr 23 '13 at 18:41

@Ka: Does this works in javascript too? I tried these in `firebug's console` but I always got `0` . – [Razort4x](#) May 6 '13 at 7:01

6 @Razort4x it works in JS via [fromCharCode](#) and [charCodeAt](#). eg.
`String.fromCharCode("a".charCodeAt(0) & 95);` – [CSφ](#) May 7 '13 at 10:13



57

- Bitwise operations on integers(int)

Finish [signing up](#) for your account

```
int maxInt = ~(1 << 31);  
int maxInt = (1 << 31) - 1;  
int maxInt = (1 << -1) - 1;
```

Get the minimum integer

```
int minInt = 1 << 31;  
int minInt = 1 << -1;
```

Get the maximum long

```
long maxLong = ((long)1 << 127) - 1;
```

Multiplied by 2

```
n << 1; // n*2
```

Divided by 2

```
n >> 1; // n/2
```

Multiplied by the m-th power of 2

```
n << m;
```

Divided by the m-th power of 2

```
n >> m;
```

Check odd number

```
(n & 1) == 1;
```

Exchange two values

```
a ^= b;  
b ^= a;  
a ^= b;
```

Get absolute value

```
(n ^ (n >> 31)) - (n >> 31);
```

Get the max of two values

```
b & ((a-b) >> 31) | a & (~(a-b) >> 31);
```

Get the min of two values

Finish [signing up](#) for your account**Check whether both have the same sign**

```
(x ^ y) >= 0;
```

Calculate 2^n

```
2 << (n-1);
```

Whether is factorial of 2

```
n > 0 ? (n & (n - 1)) == 0 : false;
```

Modulo 2^n against m

```
m & (n - 1);
```

Get the average

```
(x + y) >> 1;  
((x ^ y) >> 1) + (x & y);
```

Get the m-th bit of n (from low to high)

```
(n >> (m-1)) & 1;
```

Set the m-th bit of n to 0 (from low to high)

```
n & ~(1 << (m-1));
```

n + 1

```
~n
```

n - 1

```
~-n
```

Get the contrast number

```
~n + 1;  
(n ^ -1) + 1;
```

if (x==a) x=b; if (x==b) x=a;

```
x = a ^ b ^ x;
```

edited Mar 17 '15 at 18:32

answered Nov 25 '14 at 10:10

Finish [signing up](#) for your account

x

Math.floor() === x >> 0 and Math.ceil() === y | 1 as well – [newshorts](#) Jun 12 '17 at 23:12

From what I know, Bitwise operators are for integers and characters only and not for real valued types. You use Math.floor or Math.ceil with real valued numbers not integers. – [Shashank Avusali](#) Jul 18 '17 at 15:19

what's the point of doing `if (x==a) x=b; if (x==b) x=a;` ? it's just equivalent to `if (x == b) x = a;` . And the term for *contrast number* is the negated value or the two's complement, which could be easier done with `-n` – [phuclv](#) Aug 18 '18 at 4:44

@phuclv I think these operations are very useful when you are doing operations in low-level languages. Instead of writing complex 'if-else' and branching logic in low-level language, it becomes easy to implement the logic this way. – [BraveNinja](#) Dec 14 '18 at 3:19

@BraveNinja there's no complex if-else here. Only a single compare then jump is needed, or no jump at all if the architecture has conditional move. Moreover it's not quite a *useful* trick since it may actually be slower than normal assignments due to dependencies – [phuclv](#) Dec 14 '18 at 5:34

There's only three that I've ever used with any frequency:

12

1. Set a bit: `a |= 1 << bit;`
2. Clear a bit: `a &= ~(1 << bit);`
3. Test that a bit is set: `a & (1 << bit);`



edited Dec 3 '19 at 10:45

[vsync](#)

65.2k

40

200

255

answered Oct 8 '09 at 22:13

[Scott](#)

1,093

7

16

[Matters Computational: Ideas, Algorithms, Source Code, by Jorg Arndt \(PDF\)](#). This book contains tons of stuff, I found it via a link at <http://www.hackersdelight.org/>

6

Average without overflow

A routine for the computation of the average $(x + y)/2$ of two arguments x and y is



```
static inline ulong average(ulong x, ulong y)
// Return floor( (x+y)/2 )
// Use: x+y == ((x&y)<<1) + (x^y)
// that is: sum == carries + sum_without_carries
{
    return (x & y) + ((x ^ y) >> 1);
}
```

answered Dec 14 '11 at 8:52

[u0b34a0f6ae](#)

37.2k

11

81

97

Thanks for the book Link ! – [Debashish](#) Dec 26 '16 at 23:12

Finish [signing up](#) for your account

x

2

- See which integer values appear more frequently in the collection
- [Use short bit-sequences to represent the values which appear more frequently](#). (and longer bit-sequences to represent the values which appear less frequently)
- Concatenate the bits-sequences: so for example, the first 3 bits in the resulting bit stream might represent one integer, then the next 9 bits another integer, etc.

answered Oct 7 '09 at 18:30

[ChrisW](#)

49.5k

10

94

191

1

I used bitwise operators to efficiently implement distance calculations for [bitstrings](#). In my application bitstrings were used to represent positions in a discretised space (an [octree](#), if you're interested, encoded with [Morton ordering](#)). The distance calculations were needed to know whether points on the grid fell within a particular radius.

answered Oct 7 '09 at 18:35

[ire_and_curses](#)

60.8k

22

106

133

1

1) Divide/Multiply by a power of 2

```
foo >>= x; (divide by power of 2)
```

```
foo <<= x; (multiply by power of 2)
```

2) Swap

```
x ^= y;
y = x ^ y;
x ^= y;
```

answered Oct 7 '09 at 17:51

[Taylor Leese](#)

42.8k

24

100

138

It'd be interesting to see benchmarks demonstrating whether those are actually faster than the normal way on modern compilers. – [sepp2k](#) Oct 7 '09 at 18:04

I'd be pretty confident the shift is faster. The swap is more about not needing additional memory than being faster. – [Taylor Leese](#) Oct 7 '09 at 18:16

10 @Taylor: Most modern compilers will use a shift when it's the fastest way, without you having to manually code it. – [Ken White](#) Oct 7 '09 at 18:33

1

Counting set bits, finding lowest/highest set bit, finding nth-from-top/bottom set bit and others can be useful, and it's worth looking at the [bit-twiddling hacks](#) site.

Finish [signing up](#) for your account

standard library functions - a lot of them are better handled using specialise CPU instructions on some platforms.

answered Oct 7 '09 at 18:37

[Steve314](#)**24.9k** 10 53 110

0



While multiplying/dividing by shifting seems nifty, the only thing I needed once in a while was compressing booleans into bits. For that you need bitwise AND/OR, and probably bit shifting/inversion.

answered Oct 7 '09 at 17:57

[sbi](#)**192k** 44 225 414

0



I wanted a function to round numbers to the next highest power of two, so I visited the Bit Twiddling website that's been brought up several times and came up with this:

```
i--;
i |= i >> 1;
i |= i >> 2;
i |= i >> 4;
i |= i >> 8;
i |= i >> 16;
i++;
```

I use it on a `size_t` type. It probably won't play well on signed types. If you're worried about portability to platforms with different sized types, sprinkle your code with `#if SIZE_MAX >= (number)` directives in appropriate places.

answered Oct 8 '09 at 22:03

[Chris Lutz](#)**61.2k** 14 116 175