

# Package ‘bitops’

July 2, 2014

**Version** 1.0-6

**Date** 2013-08-17

**Author** S original by Steve Dutky <sdutky@terpalum.umd.edu> initial R  
port and extensions by Martin Maechler; revised and modified by Steve Dutky

**Maintainer** Martin Maechler <maechler@stat.math.ethz.ch>

**Title** Bitwise Operations

**Description** Functions for bitwise operations on integer vectors.

**License** GPL (>= 2)

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2013-08-17 21:10:34

## R topics documented:

bitAnd . . . . .	2
bitFlip . . . . .	3
bitShiftL . . . . .	4
cksum . . . . .	4
<b>Index</b>	<b>6</b>

---

`bitAnd`*Bitwise And, Or and Xor Operations*

---

**Description**

Bitwise operations, ‘and’ (&), ‘or’ (|), and ‘Xor’ (xor).

**Usage**

```
bitAnd(a, b)
bitOr (a, b)
bitXor(a, b)
```

**Arguments**

a,b                    numeric vectors of compatible length.

**Details**

The bitwise operations are applied to the arguments cast as 32 bit (unsigned long) integers. NA is returned wherever the magnitude of the arguments is not less than  $2^31$ , or, where either of the arguments is not finite.

**Value**

numeric vector of maximum length of a or b.

**Author(s)**

Steve Dutky

**See Also**

[bitFlip](#), [bitShiftL](#); further, [cksum](#).

**Examples**

```
bitAnd(15,7) == 7
bitOr(15,7) == 15
bitXor(15,7) == 8
bitOr(-1,0) == 4294967295
```

---

bitFlip	<i>Binary Flip (Not) Operator</i>
---------	-----------------------------------

---

### Description

The binary flip (not) operator, `bitFlip(a, w)`, “flips every bit” of `a` up to the `w`-th bit.

### Usage

```
bitFlip(a, bitWidth=32)
```

### Arguments

<code>a</code>	numeric vector.
<code>bitWidth</code>	scalar integer between 0 and 32.

### Value

binary numeric vector of the same length as `a` masked with  $(2^{\text{bitWidth}})-1$ . `NA` is returned for any value of `a` that is not finite or whose magnitude is greater or equal to  $2^{32}$ .

### Author(s)

Steve Dutky

### See Also

[bitShiftL](#), [bitXor](#), etc.

### Examples

```
stopifnot(  
  bitFlip(-1) == 0,  
  bitFlip(0) == 2^32 - 1,  
  bitFlip(0, bitWidth=8) == 255  
)
```

`bitShiftL`*Bitwise Shift Operator (to the Left or Right)*

---

**Description**

Shifting integers bitwise to the left or to the right.

**Usage**

```
bitShiftL(a, b)
bitShiftR(a, b)
```

**Arguments**

a                    numeric vector (integer valued), to be shifted.  
b                    integer vector

**Value**

numeric vector of the maximum length as a or b containing the value of a shifted to the left or right by b bits. NA is returned wherever the value of a or b is not finite, or, wherever the magnitude of a is greater than or equal to  $2^{32}$ .

**See Also**

[bitFlip](#), [bitXor](#), etc.

**Examples**

```
bitShiftR(-1,1) == 2147483647
bitShiftL(2147483647,1) == 4294967294
bitShiftL(-1,1) == 4294967294
```

---

`cksum`*Compute Check Sum*

---

**Description**

Return a cyclic redundancy checksum for each element in the argument.

**Usage**

```
cksum(a)
```

**Arguments**

a                    coerced to character vector

**Details**

[NA](#)'s appearing in the argument are returned as [NA](#)'s.

The default calculation is identical to that given in pseudo-code in the ACM article (in the References).

**Value**

numeric vector of the same length as `a`.

**Author(s)**

Steve Dutky <[sdutky@terpalum.umd.edu](mailto:sdutky@terpalum.umd.edu)>

**References**

Fashioned from `cksum(1)` UNIX command line utility, i.e., `man cksum`.

Dilip V. Sarwate (1988) Computation of Cyclic Redundancy Checks Via Table Lookup, *Communications of the ACM* **31**, 8, 1008–1013.

**See Also**

[bitShiftL](#), [bitAnd](#), etc.

**Examples**

```
b <- "I would rather have a bottle in front of me than frontal lobotomy\n"
stopifnot(cksum(b) == 1342168430)
(bv <- strsplit(b, " ")[[1]])
cksum(bv) # now a vector of length 13
```

# Index

## \*Topic **arith**

- bitAnd, 2
- bitFlip, 3
- bitShiftL, 4
- cksum, 4

## \*Topic **utilities**

- bitAnd, 2
- bitFlip, 3
- bitShiftL, 4
- cksum, 4

&, 2

- bitAnd, 2, 5
- bitFlip, 2, 3, 4
- bitOr (bitAnd), 2
- bitShiftL, 2, 3, 4, 5
- bitShiftR (bitShiftL), 4
- bitXor, 3, 4
- bitXor (bitAnd), 2

cksum, 2, 4

NA, 3, 5

xor, 2