

# Package ‘bitops’

October 12, 2022

**Version** 1.0-7

**Date** 2021-04-13

**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)

**URL** <https://github.com/mmaechler/R-bitops>

**BugReports** <https://github.com/mmaechler/R-bitops/issues>

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2021-04-24 11:40:08 UTC

## R topics documented:

bitAnd . . . . .	2
bitFlip . . . . .	3
bitShiftL . . . . .	4
cksum . . . . .	5
<b>Index</b>	<b>7</b>

---

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

---

**Description**

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

**Usage**

```
bitAnd(a, b)
a %% b
bitOr (a, b)
a %|% b
bitXor(a, b)
a %^% b
```

**Arguments**

`a, b` numeric vectors of compatible length, each treated as 32 bit “strings”.

**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.

For bitwise ‘not’ (`!` in R), use `bitFlip()`.

**Value**

non-negative integer valued numeric vector of maximum length of a or b.

**Author(s)**

Steve Dutky; idea for operators: Dan L Robertson

**See Also**

`bitFlip`, `bitShiftL`; further, `cksum`.

**Examples**

```
bitAnd(15,7) == 7 ; identical(15 %% 7, bitAnd(15, 7))
bitOr(15,7) == 15 ; identical(15 %|% 7, bitOr (15, 7))
bitXor(15,7) == 8 ; identical(15 %^% 7, bitXor(15,7))
bitOr(-1,0) == 4294967295 ; identical(-1 %|% 0, bitOr(-1,0))
```

---

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

---

**Description**

The binary flip (‘not’, R’s !) 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  $2^{32}$ .

**Note**

`bitFlip(a, w)` is an “involution”, i.e. it is its own inverse – *when* `a` is in  $\{0, 1, \dots, 2^{32}-1\}$ . Notably, negative values `a` are equivalent to their values in the above range, see also `bitUnique()` in the ‘Examples’.

**Author(s)**

Steve Dutky

**See Also**

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

**Examples**

```
bitFlip(0:5)
##
bitUnique <- function(x) bitFlip(bitFlip(x)) # "identity" when x in 0:(2^32-1)
bitUnique( 0:16 ) # identical (well, double precision)
bitUnique(-(1:16)) # 4294967295 ...
stopifnot(
  identical(bitUnique(-(1:16)), 2^32 -(1:16)),
  bitFlip(-1) == 0,
  bitFlip(0 ) == 2^32 - 1,
  bitFlip(0, bitWidth=8) == 255
)
```

bitShiftL

*Bitwise Shift Operator (to the Left or Right)***Description**

These functions shift integers bitwise to the left or to the right, returning *unsigned integers*, i.e., values in  $0, 1, \dots, 2^{32} - 1$ .

**Usage**

```
bitShiftL(a, b)
a %<<% b
bitShiftR(a, b)
a %>>% b
```

**Arguments**

**a** numeric vector (integer valued), to be shifted.  
**b** integer (valued) vector. Internally, only  $b \% 32$  is used, e.g,  $b = 32$  is equivalent to  $b = 0$ , i.e., *no* shift. This corresponds to *cyclic* rotation (to the left or right).

**Value**

non-negative integer valued numeric vector of maximum length of **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**

```
bitShiftL(0:4, 1) # 0 2 4 6 8
bitShiftL(0:3, 2) # 0 4 8 12

stopifnot(exprs = {
  identical(bitShiftL(0:4, 1), 0:4 %<<% 1)
  identical(bitShiftR(0:3, 2), 0:3 %>>% 2)
})

bitShiftR(0:7, 1) # 0 0 1 1 2 2 3 3 <==> N %/% 2
bitShiftR(0:7, 2) # 0 0 0 0 1 1 1 1 <==> N %/% 4
## all outputs are "unsigned integer" :
stopifnot( bitShiftL(-1, 0) == 2^32 - 1 ,
           bitShiftL(-7, 0) == 4294967289 ,
           bitShiftL(-7, 0) == bitShiftR(-7, 0))
```

```

bitShiftR(-1,1) == 2147483647
bitShiftL(2147483647,1) == 4294967294 # <==> * 2
bitShiftL( -1,      1) == 4294967294

bitShiftL(47, 32) # is 47

## 5 Christmas trees ( bitShiftL *rotates* to the left)
t(outer(1:5, 0:40, bitShiftL))

N <- as.numeric( rpois(1000, 100) )
stopifnot(identical(bitShiftL(N,0),  N),
           identical(bitShiftL(N,1), 2*N),
           identical(bitShiftL(N,2), 4*N),
           ## right shift:
           identical(bitShiftR(N,2), N %% 4),
           identical(bitShiftR(N,4), N %% 16))

```

---

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

## \* arith

bitAnd, 2  
bitFlip, 3  
bitShiftL, 4  
cksum, 5

## \* utilities

bitAnd, 2  
bitFlip, 3  
bitShiftL, 4  
cksum, 5

%<<% (bitShiftL), 4

%>>% (bitShiftL), 4

%&% (bitAnd), 2

%^% (bitAnd), 2

&, 2

bitAnd, 2, 6

bitFlip, 2, 3, 4

bitOr (bitAnd), 2

bitShiftL, 2, 3, 4, 6

bitShiftR (bitShiftL), 4

bitXor, 3, 4

bitXor (bitAnd), 2

cksum, 2, 5

NA, 3, 5

xor, 2