Logical Operators

MATLAB offers three types of logical operators and functions:

- Element−wise −− operate on corresponding elements of logical arrays.
- Bit−wise −− operate on corresponding bits of integer values or arrays.
- Short−circuit −− operate on scalar, logical expressions.

The values returned by MATLAB logical operators and functions, with the exception of bit−wise functions, are of type `logical` and are suitable for use with logical indexing.

Element−Wise Operators and Functions

The following logical operators and functions perform element−wise logical operations on their inputs to produce a like−sized output array. The examples shown in the following table use vector inputs `A` and `B`, where

```
A = [0 1 1 0 1];
B = [1 1 0 0 1];
```

Operator	Description	Example
`&`	Returns `1` for every element location that is `true` (nonzero) in both arrays, and `0` for all other elements.	`A & B = 01001`
`|`	Returns `1` for every element location that is `true` (nonzero) in either one or the other, or both arrays, and `0` for all other elements.	`A | B = 11101`
`~`	Complements each element of the input array, `A`.	`~A = 10010`
`xor`	Returns `1` for every element location that is `true` (nonzero) in only one array, and `0` for all other elements.	`xor(A,B)=10100`

For operators and functions that take two array operands, (`&`, `|`, and `xor`), both arrays must have equal dimensions, with each dimension being the same size. The one exception to this is where one operand is a scalar and the other is not. In this case, MATLAB tests the scalar against every element of the other operand.

Note MATLAB converts any finite nonzero, numeric values used as inputs to logical expressions to logical `1`, or `true`.

Operator Overloading. You can overload the `&`, `|`, and `~` operators to make their behavior dependent upon the data type on which they are being used. Each of these operators has a representative function that is called whenever that operator is used. These are shown in the table below.

Logical Operation	Equivalent Function
`A & B`	`and(A, B)`
`A | B`	`or(A, B)`
`~A`	`not(A)`

Other Array Functions. Two other MATLAB functions that operate logically on arrays, but not in an element−wise fashion, are `any` and `all`. These functions show whether *any* or *all* elements of a vector, or a vector within a matrix or an array, are nonzero.

When used on a matrix, `any` and `all` operate on the columns of the matrix. When used on an N−dimensional array, they operate on the first nonsingleton dimension of the array. Or, you can specify an additional `dimension` input to operate on a specific dimension of the array.

The examples shown in the following table use array input `A`, where

```
A = [0   1   2;
     0  -3   8;
     0   5   0];
```

Function	Description	Example
`any(A)`	Returns `1` for a vector where *any* element of the vector is `true` (nonzero), and `0` if no elements are `true`.	`any(A)` `ans =` `0 1 1`
`all(A)`	Returns `1` for a vector where *all* elements of the vector are `true` (nonzero), and `0` if all elements are not `true`.	`all(A)` `ans =` `0 1 0`

Note The `all` and `any` functions ignore any `NaN` values in the input arrays.

Logical Expressions Using the find Function. The `find` function determines the indices of array elements that meet a given logical condition. The function is useful for creating masks and index matrices. In its most general form, `find` returns a single vector of indices. This vector can be used to index into arrays of any size or shape.

For example,

```
A = magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

i = find(A > 8);
A(i) = 100
A =
   100     2     3   100
     5   100   100     8
   100     7     6   100
     4   100   100     1
```

Note An alternative to using `find` in this context is to index into the matrix using the logical expression itself. See the example below.

The last two statements of the previous example can be replaced with this one statement:

```
A(A > 8) = 100;
```

You can also use `find` to obtain both the row and column indices of a rectangular matrix for the array values that meet the logical condition:

```
A = magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

[row, col] = find(A > 12)
row =
     1
     4
     4
     1
col =
     1
     2
     3
     4
```

Bit−Wise Functions

The following functions perform bit−wise logical operations on nonnegative integer inputs. Inputs may be scalar or in arrays. If in arrays, these functions produce a like−sized output array.

The examples shown in the following table use scalar inputs`A` and `B`, where

```
A = 28;                 % binary 11100
B = 21;                 % binary 10101
```

Function	Description	Example
`bitand`	Returns the bit−wise AND of two nonnegative integer arguments.	`bitand(A,B) = 20 (binary 10100)`

bitor	Returns the bit−wise OR of two nonnegative integer arguments.	`bitor(A,B) = 29 (binary 11101)`
bitcmp	Returns the bit−wise complement as an n−bit number, where n is the second input argument to `bitcmp`.	`bitcmp(A,5) = 3 (binary 00011)`
bitxor	Returns the bit−wise exclusive OR of two nonnegative integer arguments.	`bitxor(A,B) = 9 (binary 01001)`

Short−Circuit Operators

The following operators perform AND and OR operations on logical expressions containing scalar values. They are *short−circuit* operators in that they evaluate their second operand only when the result is not fully determined by the first operand.

Operator	Description
&&	Returns logical `1` (`true`) if both inputs evaluate to `true`, and logical `0` (`false`) if they do not.
||	Returns logical `1` (`true`) if either input, or both, evaluate to `true`, and logical `0` (`false`) if they do not.

The statement shown here performs an AND of two logical terms, `A` and `B`:

```
A && B
```

If `A` equals zero, then the entire expression will evaluate to logical `0` (`false`), regardless of the value of `B`. Under these circumstances, there is no need to evaluate `B` because the result is already known. In this case, MATLAB short−circuits the statement by evaluating only the first term.

A similar case is when you OR two terms and the first term is `true`. Again, regardless of the value of `B`, the statement will evaluate to `true`. There is no need to evaluate the second term, and MATLAB does not do so.

Advantage of Short−Circuiting. You can use the short−circuit operators to evaluate an expression only when certain conditions are satisfied. For example, you want to execute an M−file function only if the M−file resides on the current MATLAB path.

Short−circuiting keeps the following code from generating an error when the file, `myfun.m`, cannot be found:

```
comp = (exist('myfun.m') == 2) && (myfun(x) >= y)
```

Similarly, this statement avoids divide−by−zero errors when `b` equals zero:

```
x = (b ~= 0) && (a/b > 18.5)
```

You can also use the && and || operators in if and while statements to take advantage of their short−circuiting behavior:

```
if (nargin >= 3) && (ischar(varargin{3}))
```

Relational Operators Operator Precedence

© 1994−2005 The MathWorks, Inc. • Terms of Use • Patents • Trademarks