```
(dpb x (byte 0 y) z) → z
```

for all valid values of *x*, *y*, and *z*.

Historically, the name "dpb" comes from a DEC PDP-10 assembly language instruction meaning "deposit byte."

ldb *Accessor*

Syntax:

ldb *bytespec integer* → *byte*

(**setf** (**ldb** *bytespec place*) *new-byte*)

Pronunciation:

[ˈ**lidib**] or [ˈ**lidəb**] or [ˈ**el**ˈ**dē**ˈ**bē**]

Arguments and Values:

bytespec—a *byte specifier*.

integer—an *integer*.

byte, *new-byte*—a non-negative *integer*.

Description:

ldb extracts and returns the *byte* of *integer* specified by *bytespec*.

ldb returns an *integer* in which the bits with weights $2^{(s-1)}$ through 2^0 are the same as those in *integer* with weights $2^{(p+s-1)}$ through 2^p, and all other bits zero; s is `(byte-size bytespec)` and p is `(byte-position bytespec)`.

setf may be used with **ldb** to modify a byte within the *integer* that is stored in a given *place*. The order of evaluation, when an **ldb** form is supplied to **setf**, is exactly left-to-right. The effect is to perform a **dpb** operation and then store the result back into the *place*.

Examples:

```
(ldb (byte 2 1) 10) → 1
(setq a (list 8)) → (8)
(setf (ldb (byte 2 1) (car a)) 1) → 1
a → (10)
```

See Also:

byte, **byte-position**, **byte-size**, **dpb**

Notes:

```
(logbitp j (ldb (byte s p) n))
   ≡ (and (< j s) (logbitp (+ j p) n))
```

In general,

```
(ldb (byte 0 x) y) → 0
```

for all valid values of *x* and *y*.

Historically, the name "ldb" comes from a DEC PDP-10 assembly language instruction meaning "load byte."

ldb-test *Function*

Syntax:

ldb-test *bytespec integer* → *generalized-boolean*

Arguments and Values:

bytespec—a *byte specifier*.

integer—an *integer*.

generalized-boolean—a *generalized boolean*.

Description:

Returns *true* if any of the bits of the byte in *integer* specified by *bytespec* is non-zero; otherwise returns *false*.

Examples:

```
(ldb-test (byte 4 1) 16) → true
(ldb-test (byte 3 1) 16) → false
(ldb-test (byte 3 2) 16) → true
```

See Also:

byte, **ldb**, **zerop**

Notes:

```
(ldb-test bytespec n) ≡
(not (zerop (ldb bytespec n))) ≡
(logtest (ldb bytespec -1) n)
```

mask-field *Accessor*

Syntax:

mask-field *bytespec integer* → *masked-integer*

(**setf** (**mask-field** *bytespec place*) *new-masked-integer*)

Arguments and Values:

bytespec—a *byte specifier*.

integer—an *integer*.

masked-integer, *new-masked-integer*—a non-negative *integer*.

Description:

mask-field performs a "mask" operation on *integer*. It returns an *integer* that has the same bits as *integer* in the *byte* specified by *bytespec*, but that has zero-bits everywhere else.

setf may be used with **mask-field** to modify a byte within the *integer* that is stored in a given *place*. The effect is to perform a **deposit-field** operation and then store the result back into the *place*.

Examples:

```
(mask-field (byte 1 5) -1) → 32
(setq a 15) → 15
(mask-field (byte 2 0) a) → 3
a → 15
(setf (mask-field (byte 2 0) a) 1) → 1
a → 13
```

See Also:

byte, **ldb**

Notes:

```
(ldb bs (mask-field bs n)) ≡ (ldb bs n)
(logbitp j (mask-field (byte s p) n))
  ≡ (and (>= j p) (< j s) (logbitp j n))
(mask-field bs n) ≡ (logand n (dpb -1 bs 0))
```

most-positive-fixnum, most-negative-fixnum *Constant Variable*

Constant Value:

implementation-dependent.

Description:

most-positive-fixnum is that *fixnum* closest in value to positive infinity provided by the implementation, and greater than or equal to both 2^{15} - 1 and **array-dimension-limit**.

most-negative-fixnum is that *fixnum* closest in value to negative infinity provided by the implementation, and less than or equal to -2^{15}.

decode-float, scale-float, float-radix, float-sign, float-digits, float-precision, integer-decode-float *Function*

Syntax:

decode-float *float* → *significand*, *exponent*, *sign*

scale-float *float integer* → *scaled-float*

float-radix *float* → *float-radix*

float-sign *float-1* `&optional` *float-2* → *signed-float*

float-digits *float* → *digits1*

float-precision *float* → *digits2*

integer-decode-float *float* → *significand*, *exponent*, *integer-sign*

Arguments and Values:

digits1—a non-negative *integer*.

digits2—a non-negative *integer*.

exponent—an *integer*.

float—a *float*.

float-1—a *float*.

float-2—a *float*.

decode-float, scale-float, float-radix, float-sign, ...

float-radix—an *integer*.

integer—a non-negative *integer*.

integer-sign—the *integer* `-1`, or the *integer* `1`.

scaled-float—a *float*.

sign—A *float* of the same *type* as *float* but numerically equal to `1.0` or `-1.0`.

signed-float—a *float*.

significand—a *float*.

Description:

decode-float computes three values that characterize *float*. The first value is of the same *type* as *float* and represents the significand. The second value represents the exponent to which the radix (notated in this description by *b*) must be raised to obtain the value that, when multiplied with the first result, produces the absolute value of *float*. If *float* is zero, any *integer* value may be returned, provided that the identity shown for **scale-float** holds. The third value is of the same *type* as *float* and is 1.0 if *float* is greater than or equal to zero or -1.0 otherwise.

decode-float divides *float* by an integral power of *b* so as to bring its value between $1/b$ (inclusive) and 1 (exclusive), and returns the quotient as the first value. If *float* is zero, however, the result equals the absolute value of *float* (that is, if there is a negative zero, its significand is considered to be a positive zero).

scale-float returns `(*` *float* `(expt (float` *b* *float*`)` *integer*`))`, where *b* is the radix of the floating-point representation. *float* is not necessarily between $1/b$ and 1.

float-radix returns the radix of *float*.

float-sign returns a number `z` such that `z` and *float-1* have the same sign and also such that `z` and *float-2* have the same absolute value. If *float-2* is not supplied, its value is `(float 1` *float-1*`)`. If an implementation has distinct representations for negative zero and positive zero, then `(float-sign -0.0)` → `-1.0`.

float-digits returns the number of radix *b* digits used in the representation of *float* (including any implicit digits, such as a "hidden bit").

float-precision returns the number of significant radix *b* digits present in *float*; if *float* is a *float* zero, then the result is an *integer* zero.

For *normalized floats*, the results of **float-digits** and **float-precision** are the same, but the precision is less than the number of representation digits for a *denormalized* or zero number.

integer-decode-float computes three values that characterize *float* - the significand scaled so as to be an *integer*, and the same last two values that are returned by **decode-float**. If *float* is zero, **integer-decode-float** returns zero as the first value. The second value bears the same relationship

decode-float, scale-float, float-radix, float-sign, ...

to the first value as for **decode-float**:

```
(multiple-value-bind (signif expon sign)
                     (integer-decode-float f)
  (scale-float (float signif f) expon)) ≡ (abs f)
```

Examples:

```
;; Note that since the purpose of this functionality is to expose
;; details of the implementation, all of these examples are necessarily
;; very implementation-dependent.  Results may vary widely.
;; Values shown here are chosen consistently from one particular implementation.
(decode-float .5) → 0.5, 0, 1.0
(decode-float 1.0) → 0.5, 1, 1.0
(scale-float 1.0 1) → 2.0
(scale-float 10.01 -2) → 2.5025
(scale-float 23.0 0) → 23.0
(float-radix 1.0) → 2
(float-sign 5.0) → 1.0
(float-sign -5.0) → -1.0
(float-sign 0.0) → 1.0
(float-sign 1.0 0.0) → 0.0
(float-sign 1.0 -10.0) → 10.0
(float-sign -1.0 10.0) → -10.0
(float-digits 1.0) → 24
(float-precision 1.0) → 24
(float-precision least-positive-single-float) → 1
(integer-decode-float 1.0) → 8388608, -23, 1
```

Affected By:

The implementation's representation for *floats*.

Exceptional Situations:

The functions **decode-float**, **float-radix**, **float-digits**, **float-precision**, and **integer-decode-float** should signal an error if their only argument is not a *float*.

The *function* **scale-float** should signal an error if its first argument is not a *float* or if its second argument is not an *integer*.

The *function* **float-sign** should signal an error if its first argument is not a *float* or if its second argument is supplied but is not a *float*.

Notes:

The product of the first result of **decode-float** or **integer-decode-float**, of the radix raised to the power of the second result, and of the third result is exactly equal to the value of *float*.

```
(multiple-value-bind (signif expon sign)
```

```
                      (decode-float f)
   (scale-float signif expon))
≡ (abs f)
```

and

```
 (multiple-value-bind (signif expon sign)
                      (decode-float f)
   (* (scale-float signif expon) sign))
≡ f
```

float *Function*

Syntax:

float *number* &optional *prototype* → *float*

Arguments and Values:

number—a *real*.

prototype—a *float*.

float—a *float*.

Description:

float converts a *real* number to a *float*.

If a *prototype* is supplied, a *float* is returned that is mathematically equal to *number* but has the same format as *prototype*.

If *prototype* is not supplied, then if the *number* is already a *float*, it is returned; otherwise, a *float* is returned that is mathematically equal to *number* but is a *single float*.

Examples:

```
 (float 0) → 0.0
 (float 1 .5) → 1.0
 (float 1.0) → 1.0
 (float 1/2) → 0.5
→ 1.0d0
or→ 1.0
 (eql (float 1.0 1.0d0) 1.0d0) → true
```

See Also:

coerce

floatp *Function*

Syntax:

floatp *object*

generalized-boolean

Arguments and Values:

object—an *object*.

generalized-boolean—a *generalized boolean*.

Description:

Returns *true* if *object* is of *type* **float**; otherwise, returns *false*.

Examples:

```
(floatp 1.2d2) → true
(floatp 1.212) → true
(floatp 1.2s2) → true
(floatp (expt 2 130)) → false
```

Notes:

```
(floatp object) ≡ (typep object 'float)
```

most-positive-short-float, least-positive-short-float, least-positive-normalized-short-float, most-positive-double-float, least-positive-double-float, least-positive-normalized-double-float, most-positive-long-float, least-positive-long-float, least-positive-normalized-long-float, most-positive-single-float, least-positive-single-float, least-positive-normalized-single-float, most-negative-short-float, least-negative-short-float, least-negative-normalized-short-float, most-negative-single-float, least-negative-single-float, least-negative-normalized-single-float, most-negative-double-float, least-negative-double-float, least-negative-normalized-double-float, most-negative-long-float, least-negative-long-float, least-negative-normalized-long-float *Constant Variable*

Constant Value:

implementation-dependent.

Description:

These *constant variables* provide a way for programs to examine the *implementation-defined* limits for the various float formats.

Of these *variables*, each which has "`-normalized`" in its *name* must have a *value* which is a *normalized float*, and each which does not have "`-normalized`" in its name may have a *value* which is either a *normalized float* or a *denormalized float*, as appropriate.

Of these *variables*, each which has "`short-float`" in its name must have a *value* which is a *short float*, each which has "`single-float`" in its name must have a *value* which is a *single float*, each which has "`double-float`" in its name must have a *value* which is a *double float*, and each which has "`long-float`" in its name must have a *value* which is a *long float*.

- **most-positive-short-float**, **most-positive-single-float**, **most-positive-double-float**, **most-positive-long-float**

  Each of these *constant variables* has as its *value* the positive *float* of the largest magnitude (closest in value to, but not equal to, positive infinity) for the float format implied by its name.

- **least-positive-short-float**, **least-positive-normalized-short-float**, **least-positive-single-float**, **least-positive-normalized-single-float**, **least-positive-double-float**, **least-positive-normalized-double-float**, **least-positive-long-float**, **least-positive-normalized-long-float**

  Each of these *constant variables* has as its *value* the smallest positive (nonzero) *float* for the float format implied by its name.

- **least-negative-short-float**, **least-negative-normalized-short-float**, **least-negative-single-float**, **least-negative-normalized-single-float**, **least-negative-double-float**, **least-negative-normalized-double-float**, **least-negative-long-float**, **least-negative-normalized-long-float**

  Each of these *constant variables* has as its *value* the negative (nonzero) *float* of the smallest magnitude for the float format implied by its name. (If an implementation supports minus zero as a *different object* from positive zero, this value must not be minus zero.)

- **most-negative-short-float**, **most-negative-single-float**, **most-negative-double-float**, **most-negative-long-float**

  Each of these *constant variables* has as its *value* the negative *float* of the largest magnitude (closest in value to, but not equal to, negative infinity) for the float format implied by its name.

Notes:

short-float-epsilon, short-float-negative-epsilon, single-float-epsilon, single-float-negative-epsilon, double-float-epsilon, double-float-negative-epsilon, long-float-epsilon, long-float-negative-epsilon *Constant Variable*

Constant Value:

implementation-dependent.

Description:

The value of each of the constants **short-float-epsilon**, **single-float-epsilon**, **double-float-epsilon**, and **long-float-epsilon** is the smallest positive *float* ϵ of the given format, such that the following expression is *true* when evaluated:

```
(not (= (float 1 ϵ) (+ (float 1 ϵ) ϵ)))
```

The value of each of the constants **short-float-negative-epsilon**, **single-float-negative-epsilon**, **double-float-negative-epsilon**, and **long-float-negative-epsilon** is the smallest positive *float* ϵ of the given format, such that the following expression is *true* when evaluated:

```
(not (= (float 1 ϵ) (- (float 1 ϵ) ϵ)))
```

arithmetic-error *Condition Type*

Class Precedence List:

arithmetic-error, **error**, **serious-condition**, **condition**, **t**

Description:

The *type* **arithmetic-error** consists of error conditions that occur during arithmetic operations. The operation and operands are initialized with the initialization arguments named `:operation` and `:operands` to **make-condition**, and are *accessed* by the functions **arithmetic-error-operation** and **arithmetic-error-operands**.

See Also:

arithmetic-error-operation, **arithmetic-error-operands**

arithmetic-error-operands, arithmetic-error-operation *Function*

Syntax:

arithmetic-error-operands *condition* → *operands*

arithmetic-error-operation *condition* → *operation*

Arguments and Values:

condition—a *condition* of *type* **arithmetic-error**.

operands—a *list*.

operation—a *function designator*.

Description:

arithmetic-error-operands returns a *list* of the operands which were used in the offending call to the operation that signaled the *condition*.

arithmetic-error-operation returns a *list* of the offending operation in the offending call that signaled the *condition*.

See Also:

arithmetic-error, Chapter 9 (Conditions)

Notes:

division-by-zero *Condition Type*

Class Precedence List:

division-by-zero, **arithmetic-error**, **error**, **serious-condition**, **condition**, **t**

Description:

The *type* **division-by-zero** consists of error conditions that occur because of division by zero.

floating-point-invalid-operation *Condition Type*

Class Precedence List:

floating-point-invalid-operation, **arithmetic-error**, **error**, **serious-condition**, **condition**, **t**

Description:

The *type* **floating-point-invalid-operation** consists of error conditions that occur because of certain floating point traps.

It is *implementation-dependent* whether floating point traps occur, and whether or how they may be enabled or disabled. Therefore, conforming code may establish handlers for this condition, but must not depend on its being *signaled*.

floating-point-inexact *Condition Type*

Class Precedence List:

floating-point-inexact, **arithmetic-error**, **error**, **serious-condition**, **condition**, **t**

Description:

The *type* **floating-point-inexact** consists of error conditions that occur because of certain floating point traps.

It is *implementation-dependent* whether floating point traps occur, and whether or how they may be enabled or disabled. Therefore, conforming code may establish handlers for this condition, but must not depend on its being *signaled*.

floating-point-overflow *Condition Type*

Class Precedence List:

floating-point-overflow, **arithmetic-error**, **error**, **serious-condition**, **condition**, **t**

Description:

The *type* **floating-point-overflow** consists of error conditions that occur because of floating-point overflow.

floating-point-underflow *Condition Type*

Class Precedence List:

floating-point-underflow, **arithmetic-error**, **error**, **serious-condition**, **condition**, **t**

Description:

The *type* **floating-point-underflow** consists of error conditions that occur because of floating-point underflow.