Chapter 17

Intrinsic Functions

An *intrinsic function* (or *intrinsic subroutine*) is a function (or subroutine) that is predefined within the language. CM Fortran supports all the intrinsic functions specified in the Fortran 77 standard, most of the additional functions (and several intrinsic subroutines) specified by the Fortran 90 standard, and a number of functions that were included in early drafts of the Fortran 90 standard, but were later removed.

Intrinsic functions are categorized into the following groups:

- inquiry functions
- elemental functions
- transformational functions

An *inquiry function* returns information about certain characteristics of its argument. Such functions do not depend on the *value* of their argument, so the argument may be undefined. An *elemental function* is a function with only scalar arguments that can be called with array-valued arguments, provided all array arguments are the same shape; the function is applied to corresponding elements of the array arguments as if it had been called repeatedly with a series of scalar arguments. (An *elemental subroutine* is defined similarly. See Section 12.5.2.2 for information on elemental function references.) A *transformational function* accepts array arguments and usually returns an array result, in many cases of a different shape from the input arguments.

The following sections list the intrinsic functions, grouped into general categories. The name of each function and its arguments is listed, with optional arguments indicated in italic brackets. In these sections, the elemental functions are indicated by a comment `!E`, the transformational functions by `!T`, and the inquiry functions by `!I`. Section 17.17 lists the functions alphabetically and describes each function in detail.

NOTE: CM Fortran defines more intrinsic functions than does Fortran 77. This means that a Fortran 77 program may have a different interpretation from a CM Fortran program if

Copyright © 1989–1994 Thinking Machines Corporation

it invokes a procedure having the same name as one of the new intrinsic functions (or the intrinsic subroutine), unless that procedure is specified in an EXTERNAL statement.

Much of the text for this chapter is derived from the *ANSI Fortran 8x Standard (X3.9–198X, X3J3 / S8.108)*.

17.1 Numeric Functions

The functions in this group are all elemental functions that either perform type conversions or simple numeric operations.

The following numeric functions convert arguments from one type to another.

```
ABS( A )                                    !E
AIMAG( Z )                                  !E
CMPLX( X [, Y ] [, MOLD ] )                 !E
DBLE( A )                                   !E
DCMPLX( X [, Y ] )                          !E
DPROD( X, Y )                               !E
INT( A [, KIND ] )                          !E
NINT( A [, KIND ] )                         !E
REAL( A [, MOLD ] )                         !E
```

NOTE: For Fortran 90 compatibility, the functions CMPLX and REAL accept the keyword KIND instead of the equivalent MOLD when they are compiled with the -f90syntax switch.

```
CMPLX( X [, Y ] [, KIND ] )                 !E
REAL( A [, KIND ] )                         !E
```

The following numeric functions perform simple numeric operations and do not convert arguments from one type to another.

```
AINT( A )                                   !E
ANINT( A )                                  !E
CEILING( A )                                !E
CONJG( Z )                                  !E
```


Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

```
DIM( X, Y )                                    !E
FLOOR( A )                                     !E
MAX( A1, A2 [, A3]... )                        !E
MIN( A1, A2 [, A3]... )                        !E
MOD( A, P )                                    !E
MODULO( A, P )                                 !E
SIGN( A, B )                                   !E
```

NOTE: For Fortran 90 compatibility, the functions `AINT` and `ANINT` accept an optional `KIND` argument when they are compiled with the `-f90syntax` switch.

```
AINT( A [, KIND] )                             !E
ANINT( A [, KIND] )                            !E
```

17.2 Mathematical Functions

The following elemental functions evaluate elementary mathematical functions.

```
ACOS( X )                                      !E
ASIN( X )                                      !E
ATAN( X )                                      !E
ATAN2( Y, X )                                  !E
COS( X )                                       !E
COSH( X )                                      !E
EXP( X )                                       !E
LOG( X )                                       !E
LOG10( X )                                     !E
SIN( X )                                       !E
SINH( X )                                      !E
SQRT( X )                                      !E
TAN( X )                                       !E
TANH( X )                                      !E
```


Copyright © 1989–1994 Thinking Machines Corporation

17.3 Character Functions

The functions in this group perform character operations. All functions except LEN are elemental functions, but none may be used with array arguments.

The following functions convert from a single character to an integer, or the reverse.

```
CHAR( I )                                   !E*
ICHAR( C )                                  !E*
```

The following functions perform lexicographic comparison between character strings. Similar functions are provided by the relational operators.

```
LGE( STRING_A, STRING_B )                   !E*
LGT( STRING_A, STRING_B )                   !E*
LLE( STRING_A, STRING_B )                   !E*
LLT( STRING_A, STRING_B )                   !E*
```

The following function finds the location of a substring in a string.

```
INDEX( STRING, SUBSTRING )                  !E*
```

The inquiry function LEN returns the length of a character entity. The value of a character variable passed as an argument to this function need not be defined. The value of LEN is computed at compile time when possible.

```
LEN( STRING )                               !I
```

17.4 Kind Functions

The following functions determine the kind of a type or return the smallest kind needed to represent certain numbers.

```
KIND( X )                                   !I
SELECTED_INT_KIND( R )                      !T
SELECTED_REAL_KIND( [ P ] [, R ] )          !T
```


Copyright © 1989–1994 Thinking Machines Corporation

17.5 Numeric Inquiry Functions

These functions return information about numbers of the same type and kind type as the argument X.

```
BIT_SIZE( I )                                   !I
DIGITS( X )                                     !I
EPSILON( X )                                    !I
HUGE( X )                                       !I
MAXEXPONENT( X )                                !I
MINEXPONENT( X )                                !I
PRECISION( X )                                  !I
RADIX( X )                                      !I
RANGE( X )                                      !I
TINY( X )                                       !I
```

17.6 Bit-Level Functions

The following elemental functions perform bit manipulation operations on integer values.

```
BTEST( I, POS )                                 !E
IAND( I, J )                                    !E
IBCLR( I, POS )                                 !E
IBITS( I, POS, LEN )                            !E
IBSET( I, POS )                                 !E
IEOR( I, J )                                    !E
IOR( I, J )                                     !E
ISHFT( I, SHIFT )                               !E
ISHFTC( I, SHIFT, SIZE )                        !E
NOT( I )                                        !E
```

The following functions return information about integer or logical values.

```
LEADZ( X )                                      !E
POPCNT( X )                                     !E
POPPAR( X )                                     !E
```


Copyright © 1989–1994 Thinking Machines Corporation

17.7 Floating-Point Manipulation Functions

These functions return information about the system's numerical capabilities when arguments are represented as model numbers, as described in Section 17.15.

```
EXPONENT( X )                                   !E
FRACTION( X )                                   !E
NEAREST( X, S )                                 !E
RRSPACING( X )                                  !E
SCALE( X, I )                                   !E
SET_EXPONENT( X, I )                            !E
SPACING( X )                                    !E
```

17.8 Vector and Matrix Multiplication Functions

The matrix multiplication function MATMUL operates on two matrix arguments (either of which can be a vector), returning the product under matrix multiplication. The DOT_PRODUCT function returns the scalar product of two vectors. On logical matrices and vectors, MATMUL performs Boolean matrix multiplication. For logical vectors, DOT_PRODUCT returns the Boolean scalar product.

```
DOT_PRODUCT( VECTOR_A, VECTOR_B )               !T
MATMUL( MATRIX_A, MATRIX_B )                    !T
```

17.9 Array Reduction Functions

These functions perform numerical, logical, and counting operations on arrays. They may be applied to a whole array to give a scalar result, or they may be applied over a given dimension to yield an array of results (the rank of the result is the rank of the original array reduced by one). By using a logical mask that is conformable with the given array, the computation may be confined to any subset of the array (e.g., the positive elements).

```
ALL( MASK [, DIM ] )                            !T
ANY( MASK [, DIM ] )                            !T
COUNT( MASK [, DIM ] )                          !T
MAXVAL( [, DIM ] [, MASK ] )                    !T
```


Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

```
MINVAL( ARRAY [, DIM ] [, MASK ] )                !T
PRODUCT( ARRAY [, DIM ] [, MASK ] )               !T
SUM( ARRAY [, DIM ] [, MASK ] )                   !T
```

17.10 Array Inquiry Functions

These functions return information about particular characteristics of an array or vector. These functions can be applied to array arguments whose values have not yet been defined (assigned to), since they do not depend on the value of their arguments. The value of an array inquiry function is computed at compile time for some cases.

```
ALLOCATED( ARRAY )                                !I
ASSOCIATED( POINTER )                             !I
LBOUND( ARRAY [, DIM ] )                          !I
RANK( SOURCE )                                    !I
SHAPE( SOURCE )                                   !I
SIZE( ARRAY [, DIM ] )                            !I
UBOUND( ARRAY [, DIM ] )                          !I
```

17.11 Array Construction Functions

The functions MERGE, SPREAD, RESHAPE, PACK, and UNPACK construct new arrays from the elements of existing arrays. MERGE combines two conformable arrays into one array by an element-wise choice based on a logical mask. SPREAD constructs an array from several copies of an actual argument (by adding an extra dimension, as in forming a book from copies of one page). RESHAPE produces an array with the same elements and a different shape. PACK and UNPACK respectively gather and scatter the elements of a one-dimensional array from and to positions in another array where the positions are specified by a logical mask. REPLICATE constructs an array from several copies of an actual argument by increasing the size of one of the dimensions. DIAGONAL constructs a diagonal matrix.

```
DIAGONAL( ARRAY [, FILL ] )                       !T
MERGE( TSOURCE, FSOURCE, MASK )                   !E
PACK( ARRAY, MASK [, VECTOR ] )                   !T
```


Copyright © 1989–1994 Thinking Machines Corporation

```
REPLICATE( ARRAY, DIM, NCOPIES )                    !T
RESHAPE( MOLD, SOURCE [, PAD ] [, ORDER ] )         !T
SPREAD( SOURCE, DIM, NCOPIES )                      !T
UNPACK( VECTOR, MASK, FIELD )                       !T
```

NOTE: For Fortran 90 compatibility, the function RESHAPE changes in two ways when a program is compiled with the -f90syntax switch: the keyword SHAPE is used instead of the equivalent MOLD, and the argument order is changed.

```
RESHAPE( SOURCE, SHAPE [, PAD ] [, ORDER ] )        !T
```

17.12 Array Manipulation Functions

The shift functions leave the shape of an array unaltered but shift the positions of the elements parallel to a specified dimension of the array. The TRANSPOSE function performs the matrix transpose operation on a two-dimensional array.

```
CSHIFT( ARRAY, DIM, SHIFT )                         !T
EOSHIFT( ARRAY, DIM, SHIFT [, BOUNDARY ] )          !T
TRANSPOSE( MATRIX )                                 !T
```

NOTE: For Fortran 90 compatibility, the order of arguments to the functions CSHIFT and EOSHIFT change when a program is compiled with the -f90syntax switch, and the DIM argument becomes optional.

```
CSHIFT( ARRAY, SHIFT [, DIM ] )                     !T
EOSHIFT( ARRAY, SHIFT [, BOUNDARY ] [, DIM ] )      !T
```

17.13 Array Location Functions

The functions MAXLOC and MINLOC return the location (subscripts) of an element of an array with the maximum or minimum value, respectively, under the control of a logical mask. The functions FIRSTLOC and LASTLOC locate the leading and trailing edges, respectively, of the set of true elements of a logical mask. PROJECT extracts the elements

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

that lie along an edge of an array. (PROJECT also may be classified as a reduction function — see Section 17.9.)

```
FIRSTLOC( MASK [, DIM ] )                              !T
LASTLOC( MASK [, DIM ] )                               !T
MAXLOC( ARRAY [, MASK ] )                              !T
MINLOC( ARRAY [, MASK ] )                              !T
PROJECT( ARRAY, MASK, FIELD [, DIM ] )                 !T
```

17.14 Subroutines

These subroutines perform assorted functions, including returning date and system time information, handling random number generation, and copying bit fields.

```
DATE_AND_TIME( [ DATE ] [, TIME ] [, ZONE ] [, VALUES ] )
MVBITS( FROM, FROMPOS, LEN, TO, TOPOS )
RANDOM_NUMBER( HARVEST )
RANDOM_SEED( [ SIZE ] [, PUT ] [, GET ] )
SYSTEM_CLOCK( COUNT, COUNT_RATE, COUNT_MAX )
```

17.15 Models for Integer and Real Data

The representation of integer and real data can be described by two numerical models. These models are the basis for the information returned by the floating-point manipulation functions.

The model set for integer i is defined by:

$$i = s \cdot \sum_{k=1}^{q} w_k \, 2^{k-1}$$

where q is a positive integer, each w_k is 0 or 1, and s is +1 or –1.

Copyright © 1989–1994 Thinking Machines Corporation

The model set for real x is defined by:

$$x = \begin{cases} 0 \; or \\ s \cdot b^e \cdot \sum_{k=1}^{p} f_k \, 2^{-k} \end{cases}$$

where p is an integer exceeding one; each f_k is either zero or 1, except f_1 which is 1; s is +1 or –1; and e is an integer that lies in the range e_{max} to e_{min}, inclusive. For $x = 0$, its exponent e and digits f_k are defined to be zero. The integer parameter q determines the set of model integers, and the integer parameters p, e_{min}, and e_{max} determine the set of model floating-point numbers. The parameters of the integer and real models are available for each integer and real data type implemented by the processor. The parameters characterize the set of available numbers in the definition of the model. The numeric manipulation and inquiry functions provide values related to the parameters and other constants related to them.

17.16 Specific Names for Generic Intrinsic Functions

Some intrinsic functions may be referenced with either a *generic name* or a *specific name*. Generic function names can be used with more than one type of argument. The compiler scans the arguments and generates a call to a specific function name. Only specific names can appear as actual arguments to an external procedure, and the arguments must be scalar. If an intrinsic function does not have a specific name, it cannot be used as an actual argument. Also, those specific intrinsic function names listed below that are marked with an asterisk must not be used as an actual argument. (Note that the list below only includes those intrinsic functions that have specific names.)

Generic Name	Specific Name	Argument Type	Function Type
`ABS ( A )`	`IABS ( A )`	integer	integer
`ABS ( A )`	`ABS ( A )`	real	real
`ABS ( A )`	`DABS ( A )`	double precision	double precision
`ABS ( Z )`	`CABS ( Z )`	complex	real
`ABS ( Z )`	`CDABS ( Z )`	double complex	double precision
`ACOS ( X )`	`ACOS ( X )`	real	real
`ACOS ( X )`	`DACOS ( X )`	double precision	double precision

Copyright © 1989–1994 Thinking Machines Corporation

Generic Name	Specific Name	Argument Type	Function Type
AIMAG(Z)	AIMAG(Z)	complex	real
AIMAG(Z)	DIMAG(Z)	double complex	double precision
AINT(A)	AINT(A)	real	real
AINT(A)	DINT(A)	double precision	double precision
ANINT(A)	ANINT(A)	real	real
ANINT(A)	DNINT(A)	double precision	double precision
ASIN(X)	ASIN(X)	real	real
ASIN(X)	DASIN(X)	double precision	double precision
ATAN(X)	ATAN(X)	real	real
ATAN(X)	DATAN(X)	double precision	double precision
ATAN2(Y, X)	ATAN2(Y, X)	real	real
ATAN2(Y, X)	DATAN2(Y, X)	double precision	double precision
CHAR(I)	*CHAR(I)	integer	character
CONJG(Z)	CONJG(Z)	complex	complex
CONJG(Z)	DCONJG(Z)	double complex	double complex
COS(X)	COS(X)	real	real
COS(X)	DCOS(X)	double precision	double precision
COS(Z)	CCOS(Z)	complex	complex
COS(Z)	CDCOS(Z)	double complex	double complex
COSH(X)	COSH(X)	real	real
COSH(X)	DCOSH(X)	double precision	double precision
DIM(X, Y)	IDIM(X, Y)	integer	integer
DIM(X, Y)	DIM(X, Y)	real	real
DIM(X, Y)	DDIM(X, Y)	double precision	double precision
DPROD(X, Y)	DPROD(X, Y)	real	double precision
EXP(X)	EXP(X)	real	real
EXP(X)	DEXP(X)	double precision	double precision
EXP(Z)	CEXP(Z)	complex	complex
EXP(Z)	CDEXP(Z)	double complex	double complex
ICHAR(C)	ICHAR(C)	character	integer
INDEX(S, T)	INDEX(S, T)	character	integer
INT(A)	*IFIX(A)	real	integer
INT(A)	*INT(A)	real	integer
INT(A)	*IDINT(A)	double precision	integer
LEN(S)	LEN(S)	character	integer
LGE(S, T)	LGE(S, T)	character	logical
LGT(S, T)	LGT(S, T)	character	logical

Copyright © 1989–1994 Thinking Machines Corporation

Generic Name	Specific Name	Argument Type	Function Type
LLE(S,T)	LLE(S,T)	character	logical
LLT(S,T)	LLT(S,T)	character	logical
LOG(X)	ALOG(X)	real	real
LOG(X)	DLOG(X)	double precision	double precision
LOG(Z)	CLOG(Z)	complex	complex
LOG(Z)	CDLOG(Z)	double complex	double complex
LOG10(X)	ALOG10(X)	real	real
LOG10(X)	DLOG10(X)	double precision	double precision
MAX(A1,A2,A3,..)	*MAX0(A1,A2,A3,...)	integer	integer
MAX(A1,A2,A3,..)	*AMAX1(A1,A2,A3,...)	real	real
MAX(A1,A2,A3,...)	*DMAX1(A1,A2,A3,...)	double precision	double precision
MAX(A1,A2,A3,...)	*AMAX0(A1,A2,A3,...)	integer	real
MAX(A1,A2,A3,...)	*MAX1(A1,A2,A3,...)	real	integer
MIN(A1,A2,A3)	*MIN0(A1,A2,A3,...)	integer	integer
MIN(A1,A2,A3,...)	*AMIN1(A1,A2,A3,...)	real	real
MIN(A1,A2,A3,...)	*DMIN1(A1,A2,A3,...)	double precision	double precision
MIN(A1,A2,A3,...)	*AMIN0(A1,A2,A3,...)	integer	real
MIN(A1,A2,A3,...)	*MIN1(A1,A2,A3,...)	real	integer
MOD(A,P)	MOD(A,P)	integer	integer
MOD(A,P)	AMOD(A,P)	real	real
MOD(A,P)	DMOD(A,P)	double precision	double precision
NINT(A)	NINT(A)	real	integer
NINT(A)	IDNINT(A)	double precision	integer
REAL(A)	*REAL(A)	integer	real
REAL(A)	*FLOAT(A)	integer	real
REAL(A)	*SNGL(A)	double precision	real
SIGN(A,B)	ISIGN(A,B)	integer	integer
SIGN(A,B)	SIGN(A,B)	real	real
SIGN(A,B)	DSIGN(A,B)	double precision	double precision
SIN(X)	SIN(X)	real	real
SIN(X)	DSIN(X)	double precision	double precision
SIN(Z)	CSIN(Z)	complex	complex
SIN(Z)	CDSIN(Z)	double complex	double complex
SINH(X)	SINH(X)	real	real
SINH(X)	DSINH(X)	double precision	double precision
SQRT(X)	SQRT(X)	real	real
SQRT(X)	DSQRT(X)	double precision	double precision
SQRT(Z)	CSQRT(Z)	complex	complex

Copyright © 1989–1994 Thinking Machines Corporation

Generic Name	Specific Name	Argument Type	Function Type
SQRT (Z)	CDSQRT (Z)	double complex	double complex
TAN (X)	TAN (X)	real	real
TAN (X)	DTAN (X)	double precision	double precision
TANH (X)	TANH (X)	real	real
TANH (X)	DTANH (X)	double precision	double precision

17.17 Specifications of the Intrinsic Functions

This section provides detailed specifications of all the intrinsic functions in alphabetical order.

Throughout the examples in this section, arrays of rank 2 or 3 are written using the matrix notation commonly used in mathematics, as in:

$$A = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix} \qquad B = \begin{bmatrix} \begin{bmatrix} 111 & 121 & 131 & 141 \\ 211 & 221 & 231 & 241 \\ 311 & 321 & 331 & 341 \end{bmatrix} \\ \begin{bmatrix} 112 & 122 & 132 & 142 \\ 212 & 222 & 232 & 242 \\ 312 & 322 & 332 & 342 \end{bmatrix} \end{bmatrix}$$

The elements of array A in array element order are: 11, 21, 31, 12, ..., 33. The elements of array B in array element order are: 111, 211, 311, 121, ..., 341, 112, 212, 312, 122, ..., 342.

For compactness (and consistency with the rest of this manual), arrays of rank one are usually written as array constructors, as in C = [10,20,30] as an alternative to matrix notation $C = \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}$.

Also for compactness, the abbreviations *T* and *F* are used in place of the logical constants .TRUE. and .FALSE., respectively. Within matrix brackets and array constructors representing logical arrays, the character period (.) is used for .FALSE..

Copyright © 1989–1994 Thinking Machines Corporation

In giving examples, the symbols → and ⇒ mean "evaluates exactly to" and "evaluates approximately to." For example:

```
SUM(A) ⇒ 1.5732    ! SUM(A) has the approximate value 1.5732
SUM(B) → π         ! SUM(B) has the exact value pi
```

As in syntax rules, italic brackets *[ ]* designate optional arguments in a function specification.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

ABS(A)

Description

Computes the absolute value of a number.

Kind

Elemental function.

Argument

`A` An expression of numeric type.

Result Type

The same as `A` if `A` is integer or real; the result is real with the precision of `A` if `A` is complex.

Result Value

If `A` is of type integer or real, the value of the result is $|A|$. If `A` is of type complex with value `(x,y)`, the result is an approximation to $\sqrt{x^2 + y^2}$.

Examples

```
ABS(-3) → 3
ABS(2.917d2) → 291.7d0
ABS( (-3,-4) ) → 5.
ABS( (-3,4d0) ) → 5.d0
```

$$\texttt{ABS(}\begin{bmatrix} -1 & 3 \\ 2 & -5 \end{bmatrix}\texttt{)} \rightarrow \begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

ACOS(X)

Description

Computes the arccosine (inverse cosine) trigonometric function of a value in radians.

Kind

Elemental function.

Argument

X	An expression of type real. Its value must satisfy the inequality $\lvert X \rvert \leq 1$.

Result Type

Same as X.

Result Value

The result is an approximation to the acos(X) trigonometric function, expressed in radians. It lies in the range $0 \leq \text{ACOS(X)} \leq \pi$.

Examples

ACOS(0.) $\rightarrow \pi/2 \Rightarrow$ 1.570796

ACOS(1.d0) $\rightarrow$ 0.d0

ACOS(-1.) $\rightarrow \pi \Rightarrow$ 3.141593

ACOS(COS(π)) $\rightarrow \pi$

$$\text{ACOS}\left(\begin{bmatrix} -1.0 & -.5 \\ 0.5 & .75 \end{bmatrix}\right) \Rightarrow \begin{bmatrix} 3.14159 & 2.09440 \\ 1.04720 & 0.72273 \end{bmatrix}$$

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

AIMAG(Z)

Description

Returns the imaginary part of a complex number.

Kind

Elemental function.

Argument

`Z` An expression of type complex.

Result Type

Real with the precision of `Z`.

Result Value

If `Z` has the value `(`*x*`,`*y*`)`, the result has value *y*.

Examples

```
AIMAG((-3,2)) → 2.
AIMAG((-2,-9d0)) → -9.d0
AIMAG([(-3,2) (0,-1)]) → [2. -1.]
      [(-1,0) (-5,7)]    [0.  7.]
```


Copyright © 1989–1994 Thinking Machines Corporation

AINT(A)

Description

Truncates a real value to a whole number.

Compatibility Note

For Fortran 90 compatibility, AINT accepts the optional KIND argument when a program is compiled with the -f90syntax switch.

Invoked with:	**AINT is interpreted as:**
-f90syntax	AINT(A *[, KIND]*)
-nof90syntax (default)	AINT(A)

KIND is a scalar integer constant expression indicating the length of the result type. If KIND is present, the kind type is that specified by KIND; otherwise, the kind type is that of A.

Kind

Elemental function.

Arguments

A An expression of type real.

Result Type

Same type as A.

Result Value

The result is the whole number nearest to A, truncated toward zero. If $|A| < 1$, the result is zero. If $|A| \geq 1$, the result is the whole number whose magnitude is the largest integer that does not exceed the magnitude of A and whose sign is the same as that of A.

Examples Using -nof90syntax (Default)

```
AINT(-3.2d0) → -3.d0

AINT(2.91738d2) → 291.d0
```


Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

$$\texttt{AINT}\left(\begin{bmatrix}-3.2 & 0.0 & -0.7\\ -1.0 & -5.7 & 0.7\end{bmatrix}\right) \rightarrow \begin{bmatrix}-3.0 & 0.0 & -0.0\\ -1.0 & -5.0 & 0.0\end{bmatrix}$$

Examples Using −f90syntax

```
AINT(-3.2d0) → -3.d0

AINT(2.91738, 64) → 2_64

AINT(12.34, KIND=64) → 12_64
```


Copyright © 1989–1994 Thinking Machines Corporation

ALL(MASK [, DIM])

Description

Determines whether all elements are true in a logical array, or whether all elements are true in each of the complete rank-one sections parallel to a given dimension of an array of rank two or greater.

Kind

Transformational (reduction) function.

Arguments

`MASK` An array of type logical.

`DIM` (optional)
A scalar *constant* expression of type integer, with a value in the range $1 \le \texttt{DIM} \le n$, where n is the rank of `MASK`.

Result Type

The result is of type logical. It is a scalar if `DIM` is absent or if `MASK` has rank one; otherwise, it is an array with shape equal to that of `MASK` with dimension `DIM` removed.

Result Value

Case 1a `ALL( MASK, DIM = 1 )`
`MASK` has rank `1`, `DIM` is present

or

Case 1b `ALL( MASK )`
`MASK` has any rank, `DIM` is absent:

The result is scalar. The result is true if all elements of `MASK` are true, or if `MASK` is a zero-sized array.

Case 2 `ALL( MASK, DIM )`
`MASK` has rank $n > 1$, `DIM` is present:

For a `MASK` of rank n with shape

$$[\ d_1, d_2, \ldots, d_{\texttt{DIM}-1}, d_{\texttt{DIM}}, d_{\texttt{DIM}+1}, \ldots, d_n\ ]$$

the result is an array of rank $n-1$ with shape

$$[\ d_1, d_2, \ldots, d_{\texttt{DIM}-1}, d_{\texttt{DIM}+1}, \ldots, d_n\ ].$$

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

The result is formed by applying ALL to each of the complete rank-one sections of MASK that lie parallel to dimension DIM. Thus, the value of element

R(s_1, s_2, ..., s_{DIM-1}, s_{DIM+1}, ..., s_n)

of a result array R is equal to

ALL($MASK_{DIM}$)

where $MASK_{DIM}$ is the array section

MASK(s_1, s_2, ..., s_{DIM-1}, : , s_{DIM+1}, ..., s_n) .

If only dimension DIM of MASK is zero-sized, the result is an array with all elements equal to false. If any dimension of MASK other than DIM is zero-sized, the result is a zero-sized array.

Examples

Case 1a MASK has rank 1, DIM is present:

ALL([*T*,.,*T*]) → *F*

ALL([*T*,.,*T*], DIM = 1) → *F*

ALL(MASK = [*T*,.,*T*,*T*], DIM = 1) → *F*

Case 1b MASK has any rank, DIM is absent:

ALL($\begin{bmatrix} T & T & T \\ T & T & T \end{bmatrix}$) → *T*

Case 2 MASK has rank $n > 1$, DIM is present:

If B = $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, and C = $\begin{bmatrix} 0 & 3 & 5 \\ 7 & 4 & 8 \end{bmatrix}$, then

ALL(B .EQ. C, DIM = 1) → [.,*T*,.]

and

ALL(B .EQ. C, DIM = 2) → [.,.]

Copyright © 1989–1994 Thinking Machines Corporation

ALLOCATED(ARRAY)

Description

Indicates whether an allocatable array is currently allocated.

Kind

Inquiry function.

Argument

`ARRAY` An allocatable array of any type.

Result Type

A scalar of type logical.

Result Value

The result has the value `.TRUE.` if `ARRAY` is currently allocated and has the value `.FALSE.` if `ARRAY` is not currently allocated. The result is undefined if the allocation status of the array is undefined.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

ANINT(A)

Description

Rounds a real value to a whole number.

Compatibility Note

For Fortran 90 compatibility, ANINT accepts the optional KIND argument when a program is compiled with the -f90syntax switch.

Invoked with:	**ANINT is interpreted as:**
-f90syntax	ANINT(A *[*, KIND*]*)
-nof90syntax (default)	ANINT(A)

KIND is a scalar integer constant expression indicating the length of the result type. If KIND is present, the kind type is that specified by KIND; otherwise, the kind type is that of A.

Kind

Elemental function.

Argument

A An expression of type real.

Result Type

Real (with the same precision as A).

Result Value

The result is the whole number nearest to A, rounded away from zero. If A > 0, the result is equal to AINT(A+0.5); if A ≤ 0, the result is equal to AINT(A-0.5).

Examples Using -nof90syntax (Default)

```
ANINT(-3.2d0) → -3.d0

ANINT(2.91738d2) → 292.d0
```


Copyright © 1989–1994 Thinking Machines Corporation

$$\text{ANINT}\left(\begin{bmatrix} -3.2 & 0.0 & -0.7 \\ -1.0 & -5.7 & 0.7 \end{bmatrix}\right) \rightarrow \begin{bmatrix} -3.0 & 0.0 & -1.0 \\ -1.0 & -6.0 & 1.0 \end{bmatrix}$$

Examples Using −f90syntax

ANINT(-3.2d0) → -3.d0

ANINT(2.91738, 64) → 3_64

ANINT(12.34, KIND=64) → 12_64

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

ANY(MASK *[*, DIM *]*)

Description

Determines whether any elements are true in a logical array, or whether any elements are true in each of the complete rank-one sections parallel to a given dimension of an array of rank two or greater.

Kind

Transformational (reduction) function.

Arguments

`MASK` An array of type logical.

`DIM` (optional)
A scalar *constant* expression of type integer, with a value in the range $1 \leq \texttt{DIM} \leq n$, where n is the rank of `MASK`.

Result Type

The result is of type logical. It is a scalar if `DIM` is absent or if `MASK` has rank one; otherwise, it is an array with shape equal to that of `MASK` with dimension `DIM` removed.

Result Value

Case 1a `ANY ( MASK, DIM = 1 )`
`MASK` has rank `1`, `DIM` is present

or

Case 1b `ANY ( MASK )`
`MASK` has any rank, `DIM` is absent:

The result is scalar. The result is true if any elements of `MASK` are true, or if `MASK` is a zero-sized array.

Case 2 `ANY ( MASK, DIM )`
`MASK` has rank $n > 1$, `DIM` is present:

For a `MASK` of rank n with shape

$$[\ d_1, d_2, \ldots, d_{\texttt{DIM}-1}, d_{\texttt{DIM}}, d_{\texttt{DIM}+1}, \ldots, d_n\ ]$$

the result is an array of rank $n-1$ with shape

$$[\ d_1, d_2, \ldots, d_{\texttt{DIM}-1}, d_{\texttt{DIM}+1}, \ldots, d_n\ ].$$

Copyright © 1989–1994 Thinking Machines Corporation

The result is formed by applying ANY to each of the complete rank-one sections of MASK that lie parallel to dimension DIM. Thus, the value of element

R(s_1, s_2, ..., s_{DIM-1}, s_{DIM+1}, ..., s_n)

of a result array R is equal to

ANY($MASK_{DIM}$)

where $MASK_{DIM}$ is the array section

MASK(s_1, s_2, ..., s_{DIM-1}, : , s_{DIM+1}, ..., s_n)

If only dimension DIM of MASK is zero-sized, the result is an array with all elements equal to false. If any dimension of MASK other than DIM is zero-sized, the result is a zero-sized array.

Examples

Case 1a MASK has rank 1, DIM is present:

ANY([*T*, ., *T*]) → *T*

ANY([*T*, ., *T*], DIM = 1) → *T*

ANY(MASK = [*T*, ., *T*, *T*], DIM = 1) → *T*

Case 1b MASK has any rank, DIM is absent:

ANY($\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$) → *F*

Case 2 MASK has rank $n > 1$, DIM is present:

If B = $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, and C = $\begin{bmatrix} 0 & 4 & 5 \\ 7 & 3 & 8 \end{bmatrix}$, then

ANY(B .EQ. C, DIM = 1) → [., ., *T*]

and

ANY(B .EQ. C, DIM = 2) → [*T*, .]

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

ASIN(X)

Description

Computes the arcsine (inverse sine) trigonometric function.

Kind

Elemental function.

Argument

X — An expression of type real. Its value must satisfy the inequality $|X| \leq 1$.

Result Type

Same as X.

Result Value

The result is an approximation to the `arcsin(X)` trigonometric function, expressed in radians. It lies in the range $-\pi/2 \leq$ `ASIN(X)` $\leq \pi/2$.

Examples

`ASIN(0.d0)` $\rightarrow$ `0.d0`

`ASIN(1.)` $\rightarrow \pi/2 \Rightarrow$ `1.570796`

`ASIN(-1.)` $\rightarrow -\pi/2 \Rightarrow$ `-1.570796`

`ASIN(SIN(`π`))` $\rightarrow$ `0.`

$$\texttt{ASIN}\left(\begin{bmatrix} 0.0 & 0.5 \\ -.5 & 1.0 \end{bmatrix}\right) \Rightarrow \begin{bmatrix} 0.00000 & 0.52359 \\ -0.52359 & 1.57080 \end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

ASSOCIATED(POINTER)

Description

Returns the association status of the argument `POINTER`.

Kind

Inquiry function.

Argument

`POINTER` An array pointer of any type.

Result Type

A scalar of type logical.

Result Value

The result has the value `.TRUE.` if `POINTER` is currently associated with a target, and `.FALSE.` otherwise.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

ATAN(X)

Description

Computes the arctangent (inverse tangent) trigonometric function.

Kind

Elemental function.

Argument

X An expression of type real.

Result Type

Same as X.

Result Value

The result is an approximation to the `arctan(X)` trigonometric function, expressed in radians. It lies in the range $-\pi/2 \leq$ `ATAN(X)` $\leq \pi/2$.

Examples

```
ATAN(1.5574077) ⇒ 1.
ATAN(0.d0) → 0.d0
ATAN(1.) ⇒ 0.7853982
ATAN(-1.) ⇒ -0.7853982
ATAN(TAN(0.5)) → 0.5
ATAN([ 0.0  0.5]) ⇒ [ 0.00000 0.46365]
     [-.5   1.0]    [-0.46565 0.78539]
```


Copyright © 1989–1994 Thinking Machines Corporation

ATAN2(Y, X)

Description

Computes the arctangent (inverse tangent) trigonometric function. The result is the principal value of the argument of the nonzero complex number (X, Y)

Kind

Elemental function.

Arguments

Y	An expression of type real.
X	An expression of the same type as Y. If Y has the value zero, X must not have the value zero.

Result Type

Same as X.

Result Value

The result is an approximation to the principal value of the argument of the complex number CMPLX(X,Y), expressed in radians. It lies in the range

$-\pi < \texttt{ATAN2(Y,Z)} \leq \pi$

and is equal to an approximation of the arctan(Y/X) trigonometric function if $X \neq 0$. The function has the following characteristics:

if Y > 0, the result is positive

if Y < 0, the result is negative

if Y = 0, the result is 0 if X > 0 and π if X < 0

if X = 0, the result is π/2 if Y > 0 and −π/2 if Y < 0.

Examples

```
ATAN2( 1., 2. ) ⇒ 0.4636476
ATAN2( 1.d0, -2.d0 ) ⇒ 2.677945d0
```


Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

$$\texttt{ATAN2(Y=}\begin{bmatrix}+1. & 0.\\ +1. & 0.\\ -1. & +1.\\ -1. & -1.\end{bmatrix}\texttt{, X=}\begin{bmatrix}+1. & +1.\\ -1. & -1.\\ +1. & 0.\\ -1. & 0.\end{bmatrix}\texttt{)} \rightarrow \begin{bmatrix}+0.78540 & 0.00000\\ 2.35619 & 3.14159\\ -0.78540 & 1.57080\\ -2.35619 & -1.57080\end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

BIT_SIZE(I)

Description

Returns the number of bits in the model representing numbers of the same kind type as the argument.

Kind

Inquiry function.

Argument

`I` A scalar or array of type `INTEGER`.

Result Type

Scalar integer of the same kind type as `I`.

Result Value

The result is one of the following values, depending on the kind type of the argument.

Data Type	Value
`INTEGER*4`	32
`INTEGER*8`	64

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

BTEST(I, POS)

Description

Tests whether a given bit of an integer value is set.

Kind

Elemental function.

Arguments

`I`	An expression of type integer.
`POS`	An expression of type integer. If `INTEGER*4`, the value is in the range $0 \le$ `POS` ≤ 31; if `INTEGER*8`, the value is in the range $0 \le$ `POS` ≤ 63.

Result Type

Logical.

Result Value

The result has the value true if bit `POS` of `I` has the value `1` and has the value false if bit `POS` of `I` has the value `0`.

The low-order (right-most) bit of an integer value has position `0`. If the type is `INTEGER*4`, the high-order (left-most) bit has position `31`; if the type is `INTEGER*8`, the high-order bit has position `63`.

Examples

`BTEST( 8, 3 )` $\rightarrow$ *T*

`BTEST( 1+4+16+512, POS = [0:10] )` $\rightarrow$ `[`*T*`,.,`*T*`,.,`*T*`,.,.,.,.,`*T*`,.]`

$$\texttt{BTEST(}\begin{bmatrix}0 & 3 & 6\\ 1 & 4 & 7\\ 2 & 5 & 8\end{bmatrix}\texttt{, POS = 2)} \rightarrow \begin{bmatrix}. & . & T\\ . & T & T\\ . & T & .\end{bmatrix}$$

$$\texttt{BTEST(1+4+8, POS =}\begin{bmatrix}0 & 3 & 6\\ 1 & 4 & 7\\ 2 & 5 & 8\end{bmatrix}\texttt{)} \rightarrow \begin{bmatrix}T & T & .\\ . & . & .\\ T & . & .\end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

CEILING(A)

Description

Returns the least integer greater than or equal to `A`.

Kind

Elemental function.

Argument

`A` A scalar or array of type `REAL`.

Result Type

A scalar or array of type `INTEGER*4`.

Result Value

The result is equal to the least integer greater than or equal to `A`. The result is undefined if the processor cannot represent this value as an `INTEGER*4`.

Examples

```
CEILING(3.7) → 4

CEILING(-3.7) → -3
```


Copyright © 1989–1994 Thinking Machines Corporation

CHAR(I)

Description

Returns the character in a given position of the ASCII collating sequence. (The function is the inverse of the function ICHAR.)

Kind

Elemental function (restricted to scalar arguments).

Argument

I	A scalar integer expression with a value in the range 0 to 127. (The 128 possible values indicate one of the characters in the ASCII character set.)

Result Type

Character of length one.

Description

The result is the character in position I of the ASCII collating sequence.

It is always the case that

```
CHAR( ICHAR(C) )
```

has the value C for any ASCII character C. Also,

```
ICHAR( CHAR(I) )
```

has the value I for $0 \le I \le 127$.

Examples

```
CHAR(65) → "A"
CHAR(66) → "B"
CHAR(90) → "Z"
CHAR(122) → "z"
CHAR(49) → "1"
CHAR(50) → "2"
CHAR(40) → "("
```


Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

CMPLX(X *[*, Y *]* *[*, MOLD *]*)†

Description

Converts a number or a pair of numbers to a value of type complex.

†Compatibility Note

For Fortran 90 compatibility, `CMPLX` accepts the keyword `KIND` for the third argument when a program is compiled with the `-f90syntax` switch.

Invoked with:	**`CMPLX` is interpreted as:**
`-f90syntax`	`CMPLX(X [, Y] [, KIND])`
`-nof90syntax` (default)	`CMPLX(X [, Y] [, MOLD])`

Kind

Elemental function.

Arguments

`X` An expression of numeric type.

`Y` (optional)
An expression of type integer or real. `Y` must not be present if `X` is complex.

`MOLD` (optional)
An expression of type real.

Result Type

The result is of type complex. If `MOLD` is absent or is present with single precision, the result has single precision; if `MOLD` is present with double precision, the result has double precision.

Result Value

Case 1 `CMPLX( X [, Y ] [, MOLD] )`
`MOLD` is absent, or present with single precision:

If `Y` is present, the result is the single-precision complex value whose real part is `REAL(X)` and whose imaginary part is `REAL(Y)`.

Copyright © 1989–1994 Thinking Machines Corporation

If Y is absent and X is of type integer or real, the result is the single-precision complex value whose real part is REAL(X) and whose imaginary part is zero. If Y is absent and X is of type complex, the result is X (converted to single-precision if necessary).

Case 2 CMPLX(X *[*, Y *]*, MOLD)
MOLD is present with double precision:

If Y is present, the result is the double-precision complex value whose real part is DBLE(X) and whose imaginary part is DBLE(Y).

If Y is absent and X is of type integer or real, the result is the double-precision complex value whose real part is DBLE(X) and whose imaginary part is zero. If Y is absent and X is of type complex, the result is X (widened to double-precision if necessary).

Examples

```
CMPLX( 1, 2 ) → (1.,2.)
CMPLX( 1., 2. ) → (1.,2.)
CMPLX( 1d0, 2d0, 0e0 ) → (1.,2.)
CMPLX( 1, 2, 0d0 ) → (1.,2.d0)
CMPLX( (1, 2) ) → (1.,2.)
CMPLX( (1, 2d0) ) → (1.,2.)
```

$$\texttt{CMPLX}\left(\begin{bmatrix}-1 & 3\\ -5 & 2\end{bmatrix},\begin{bmatrix}-2 & 0\\ -4 & 3\end{bmatrix}\right) \rightarrow \begin{bmatrix}(-1,-2) & (3,0)\\ (-5,-4) & (2,3)\end{bmatrix}$$

$$\texttt{CMPLX}\left(\begin{bmatrix}-1 & 3\\ -5 & 2\end{bmatrix}\right) \rightarrow \begin{bmatrix}(-1,0) & (3,0)\\ (-5,0) & (2,0)\end{bmatrix}$$

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

CONJG(Z)

Description

Computes the conjugate of a complex number.

Kind

Elemental function.

Argument

Z	An expression of type complex.

Result Type

Same type as Z.

Result Value

If Z has the value (*x*,*y*), the result is (*x*,−*y*).

Examples

```
CONJG((-3,2)) → (-3,-2)

CONJG((-2,-9)) → (-2,9)
```

$$\texttt{CONJG(}\begin{bmatrix}\texttt{(-3,2)} & \texttt{(0,-1)} & \texttt{(0,.7)}\\ \texttt{(-1,0)} & \texttt{(-5,7)} & \texttt{(0,-.7)}\end{bmatrix}\texttt{)} \rightarrow \begin{bmatrix}\texttt{(-3,-2)} & \texttt{(0,1)} & \texttt{(0,-.7)}\\ \texttt{(-1,0)} & \texttt{(-5,-7)} & \texttt{(0,.7)}\end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

COS(X)

Description

Computes the cosine trigonometric function.

Kind

Elemental function.

Argument

X	An expression of type real or complex.

Result Type

Same as X.

Result Value

The result has a value equal to the `cos(X)` trigonometric function. If X is of type real, it is regarded as a value in radians. For complex values of X with the form `(`*a*`,`*b*`)`, the result is calculated using the identity

```
COS(X) = CMPLX( COS(a)*COSH(b), -SIN(a)*SINH(b), MOLD = X )
```

The real part *a* is regarded as a value in radians.

Examples

```
COS(0.d0) → 1.d0
COS(π/2) ⇒ 0.
COS(ACOS(0.25)) → 0.25
COS((π,0)) → (-1,0)
COS((π,1)) ⇒ (-1.5430806,0.)
```

$$\mathtt{COS}\left(\begin{bmatrix} 0.52359 & 1.57080 \\ 1.04720 & 2.35619 \end{bmatrix}\right) \Rightarrow \begin{bmatrix} 0.86603 & 0.00000 \\ 0.50000 & -0.70711 \end{bmatrix}$$

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

COSH(X)

Description

Computes the hyperbolic cosine function.

Kind

Elemental function.

Argument

X An expression of type real.

Result Type

Same as X.

Result Value

The result is an approximation to the `cosh(X)` mathematical function.

Examples

```
COSH(0.) → 1.
```

$$\texttt{COSH}\left(\begin{bmatrix}0.5 & 2.0 & 5.0\\ 1.0 & 3.0 & 9.0\end{bmatrix}\right) \Rightarrow \begin{bmatrix}1.12763 & 3.76220 & 74.2100\\ 1.54308 & 10.0677 & 4051.54\end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

COUNT(MASK [, DIM])

Description

Counts the number of true elements in a logical array, or counts the number of true elements in each of the complete rank-one sections parallel to a given dimension of an array of rank two or greater.

Kind

Transformational (reduction) function.

Arguments

`MASK` An array of type logical.

`DIM` (optional)
A scalar *constant* expression of type integer, with a value in the range $1 \leq \texttt{DIM} \leq n$, where n is the rank of `MASK`.

Result Type

The result is of type `INTEGER*4`. It is a scalar if `DIM` is absent or if `MASK` has rank one; otherwise, it is an array with shape equal to that of `MASK` with dimension `DIM` removed.

Result Value

Case 1a `COUNT( MASK, DIM = 1 )`
`MASK` has rank `1`, `DIM` is present

or

Case 1b `COUNT( MASK )`
`MASK` has any rank, `DIM` is absent:

The result is scalar. The result is the total number of true elements in `MASK`. If `MASK` has size zero, the result is zero.

Case 2 `COUNT( MASK, DIM )`
`MASK` has rank $n > 1$, `DIM` is present:

For a `MASK` of rank n with shape

$[\ d_1, d_2, \ldots, d_{\texttt{DIM}-1}, d_{\texttt{DIM}}, d_{\texttt{DIM}+1}, \ldots, d_n\ ]$

the result is an array of rank $n-1$ with shape

$[\ d_1, d_2, \ldots, d_{\texttt{DIM}-1}, d_{\texttt{DIM}+1}, \ldots, d_n\ ]$.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

The result is formed by applying COUNT to each of the complete rank-one sections along dimension DIM of MASK. Thus, the value of element

R(s_1, s_2, ..., s_{DIM-1}, s_{DIM+1}, ..., s_n)

of a result array R is equal to

COUNT($MASK_{DIM}$)

where $MASK_{DIM}$ is the array section

MASK(s_1, s_2, ..., s_{DIM-1}, : , s_{DIM+1}, ..., s_n)

If only dimension DIM of MASK is zero-sized, the result is an array with all elements equal to zero. If any dimension of MASK other than DIM is zero-sized, the result is a zero-sized array.

Examples

Case 1a MASK has rank 1, DIM is present:

COUNT([*T*, ., *T*]) → 2

COUNT([*T*, ., *T*], DIM = 1) → 2

COUNT(MASK = [*T*, ., *T*, *T*], DIM = 1) → 3

Case 1b MASK has any rank, DIM is absent:

COUNT($\begin{bmatrix} . & T & . \\ T & . & T \end{bmatrix}$) → 3

Case 2 MASK has rank $n > 1$, DIM is present:

If B = $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, and C = $\begin{bmatrix} 0 & 3 & 5 \\ 7 & 4 & 8 \end{bmatrix}$, then

COUNT(B .EQ. C, DIM = 1) → [0,2,1]

and

COUNT(B .EQ. C, DIM = 2) → [2,1]

Copyright © 1989–1994 Thinking Machines Corporation

CSHIFT(ARRAY, DIM, SHIFT)

Description

Performs a circular shift of an array of rank one, or performs circular shifts of each of the complete rank-one sections parallel to a given dimension of an array of rank two or greater. Elements shifted out at one end of a section are shifted in at the other end. Different sections may be shifted by different amounts and in different directions.

Compatibility Note

`CSHIFT` becomes compatible with Fortran 90 when a program is compiled with the `-f90syntax` switch. When the function is invoked without argument keywords, CM Fortran interprets the argument order as:

Invoked with:	**`CSHIFT` is interpreted as:**
`-f90syntax`	`CSHIFT(ARRAY, SHIFT [,DIM])`
`-nof90syntax` (default)	`CSHIFT(ARRAY, DIM, SHIFT)`

The use of keywords allows the arguments to appear in any order, regardless of the compilation switch.

When the `-f90syntax` switch is used, the `DIM` argument is optional. If `DIM` is absent, its value is treated as 1.

Kind

Transformational function.

Arguments

`ARRAY` An array of numeric or logical type.

`DIM` A scalar of type integer, with a value in the range $1 \leq$ `DIM` $\leq n$, where n is the rank of `ARRAY`.

`SHIFT` An integer expression. `SHIFT` must be a scalar if `ARRAY` has rank one; otherwise, it may be a scalar or an array. If `SHIFT` is an array, it must be of shape equal to that of `ARRAY` with dimension `DIM` removed.

Result Type

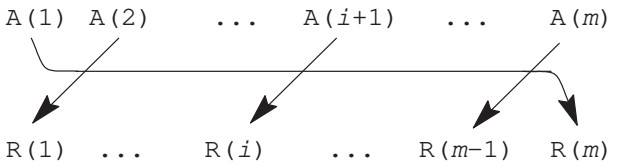
The result has the type and shape of `ARRAY`.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Result Value

Case 1a `CSHIFT( ARRAY, DIM = 1, SHIFT )`
`ARRAY` has rank one, `SHIFT` is a nonnegative scalar value:

If `SHIFT` is `1` and `ARRAY` has size *m*, the result is defined by the diagram

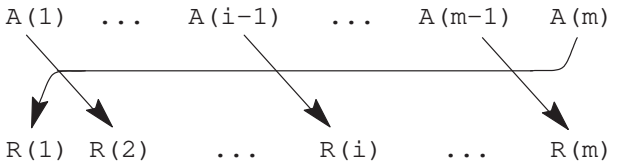


where the `A(`*i*`)` and the `R(`*i*`)` represent elements of `ARRAY` and the result, respectively, and the arrows indicate the movement of data.

If `SHIFT` is greater than `1`, the result is equal to `SHIFT` repeated applications of `CSHIFT` to `ARRAY`, with `SHIFT = 1`. If `SHIFT` is equal to zero, the result is the original array.

Case 1b `CSHIFT( ARRAY, DIM = 1, SHIFT )`
`ARRAY` has rank one, `SHIFT` is a negative scalar value:

If `SHIFT` is `−1` and `ARRAY` has size *m*, the result is defined by the diagram



where the `A(`*i*`)` and the `R(`*i*`)` represent elements of `ARRAY` and the result, respectively, and the arrows indicate the movement of data.

If `SHIFT` is less than `−1`, the result is equivalent to `|SHIFT|` repeated applications of `CSHIFT` to `ARRAY`, with `SHIFT = −1`.

Copyright © 1989–1994 Thinking Machines Corporation

Case 2 `CSHIFT( ARRAY, DIM, SHIFT )`
`ARRAY` has rank n>1, `SHIFT` is a scalar or an array:

The result is formed by performing a circular shift of each of the complete rank-one sections of `ARRAY` parallel to dimension `DIM`. The value of section

$R(s_1, s_2, \ldots, s_{DIM-1}, :, s_{DIM+1}, \ldots, s_n)$

of a result array `R` is equal to

$CSHIFT(ARRAY_{DIM}, DIM = 1, SHIFT_{VAL})$

where $ARRAY_{DIM}$ is the array section

$ARRAY(s_1, s_2, \ldots, s_{DIM-1}, :, s_{DIM+1}, \ldots, s_n)$

and where $SHIFT_{VAL}$ is the scalar `SHIFT`, or is the array element

$SHIFT(s_1, s_2, \ldots, s_{DIM-1}, s_{DIM+1}, \ldots, s_n)$

if `SHIFT` is an array.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Examples

Case 1 `ARRAY` has rank one, `SHIFT` is a scalar:

If `V = [1,2,3,4,5,6]`, the effect of shifting `V` circularly to the left by two positions is achieved by

```
CSHIFT( V, DIM = 1, SHIFT = 2 ) → [3,4,5,6,1,2]
```

while

```
CSHIFT( V, DIM = 1, SHIFT = -1 ) → [6,1,2,3,4,5]
```

achieves a circular shift to the right by one position.

Case 2 `ARRAY` has rank *n*>1, `SHIFT` is a scalar or an array:

The rows of a rank-two array may all be shifted by the same amount or by different amounts.

If $\mathtt{A} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$, then

$$\mathtt{CSHIFT(\ A,\ DIM = 2,\ SHIFT = -1\ )} \rightarrow \begin{bmatrix} 4 & 1 & 2 & 3 \\ 4 & 1 & 2 & 3 \\ 4 & 1 & 2 & 3 \end{bmatrix}$$

$$\mathtt{CSHIFT(\ A,\ DIM = 2,\ SHIFT = [-1,1,0]\ )} \rightarrow \begin{bmatrix} 4 & 1 & 2 & 3 \\ 2 & 3 & 4 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

If $\mathtt{C} = \begin{bmatrix} \begin{bmatrix} 111 & 121 & 131 & 141 \\ 211 & 221 & 231 & 241 \\ 311 & 321 & 331 & 341 \end{bmatrix} \\ \begin{bmatrix} 112 & 122 & 132 & 142 \\ 212 & 222 & 232 & 242 \\ 312 & 322 & 332 & 342 \end{bmatrix} \end{bmatrix}$ and $\mathtt{S} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & -1 \\ 3 & -1 \end{bmatrix}$, then

$$\mathtt{CSHIFT(\ C,\ DIM = 1,\ SHIFT = S\ )} \rightarrow \begin{bmatrix} \begin{bmatrix} 111 & 221 & 331 & 141 \\ 211 & 321 & 131 & 241 \\ 311 & 121 & 231 & 341 \end{bmatrix} \\ \begin{bmatrix} 212 & 222 & 332 & 342 \\ 312 & 322 & 132 & 142 \\ 112 & 122 & 232 & 242 \end{bmatrix} \end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

DATE_AND_TIME(*[*DATE*] [*, TIME*] [*, ZONE*] [*, VALUES*]*)

Description

Returns integer data from the date and real-time clock.

Kind

Subroutine.

Arguments

`DATE` (optional)

A scalar of type `CHARACTER`, with a length of at least 8, in order to contain the complete value. It is an `INTENT(OUT)` argument. Its leftmost 8 characters are set to a value of the form CCYYMMDD, where CC is the century, YY the year within the century, MM the month within the year, and DD the day within the month. If there is no data available, `DATE` is blank.

`TIME` (optional)

A scalar of type `CHARACTER`, with a length of at least 10 in order to contain the complete value. It is an `INTENT(OUT)` argument. Its leftmost 10 characters are set to a value of the form HHMMSS.SSS, where HH is the hour of the day, MM is the minutes of the hour, and SS.SSS is the seconds and milliseconds of the minute. If there is no clock available, `TIME` is blank.

`ZONE` (optional)

A scalar of type `CHARACTER`, with a length of at least 5 in order to contain the complete value. It is an `INTENT(OUT)` argument. Its leftmost 5 characters are set to a value of the form +/–HHMM, where HH and MM are the time difference with respect to Coordinated Universal Time (UTC), also known as Greenwich Mean Time, in hours and minutes, respectively. If there is no clock available, `ZONE` is blank.

`VALUES` (optional)

Rank one `INTEGER*4`. It is an `INTENT(OUT)` argument. Its size must be at least 8. The values returned in `VALUES` are as follows:

`VALUES(1)`	the year (for example, 1994)
`VALUES(2)`	the month of the year
`VALUES(3)`	the day of the month
`VALUES(4)`	the time difference with respect to Coordinated Universal Time (UTC) in minutes
`VALUES(5)`	the hour of the day, in the range of 0 to 23

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

VALUES(6)	the minutes of the hour, in the range 0 to 59
VALUES(7)	the seconds of the minute, in the range 0 to 59
VALUES(8)	the milliseconds of the second, in the range 0 to 999

If there is no date or clock available, a given VALUES value is –HUGE (0).

Example

```
 INTEGER DATE-TIME (8)
 CHARACTER (LEN=10) BIG-BEN (3)
 CALL DATE_AND_TIME (BIG-BEN(1), BIG-BEN(2), BIG-BEN(3),
&                    DATE-TIME)
```

If called in Geneva, Switzerland on 1985 April 12 at 15:27:35.5, this code would assign the value 19850412 to BIG-BEN(1), the value 152735.500 to BIG-BEN(2), and the value +0100 to BIG-BEN(3), and the following values to DATE-TIME: 1985, 4, 12, 60, 15, 27, 35, 500.

Copyright © 1989–1994 Thinking Machines Corporation

DBLE(A)

Description

Converts a numeric value to a double-precision real value.

Kind

Elemental function.

Argument

A	An expression of numeric type.

Result Type

Double-precision real.

Result Value

Case 1	If A is of type integer or single-precision real, the result is A converted to double precision.
Case 2	If A is of type double-precision real, the result is A.
Case 3	If A is of type complex, the result is the real part of A, converted (if necessary) to double precision.

Examples

```
DBLE(175.d0) → 175.d0
DBLE(57) → 57.d0
DBLE(1.) → 1.d0
DBLE((1,2))→ 1.d0
DBLE((1,2d0)) → 1.d0
```

$$\texttt{DBLE}\left(\begin{bmatrix} -1 & 3 \\ -5 & 2 \end{bmatrix}\right) \rightarrow \begin{bmatrix} -1.d0 & 3.d0 \\ -5.d0 & 2.d0 \end{bmatrix}$$

$$\texttt{DBLE}\left(\begin{bmatrix} (-1,-2) & (3,0) \\ (-5,-4) & (2,3) \end{bmatrix}\right) \rightarrow \begin{bmatrix} -1.d0 & 3.d0 \\ -5.d0 & 2.d0 \end{bmatrix}$$

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

DCMPLX(X *[*, Y *]*)

Description

Converts a number to double-precision complex type.

Kind

Elemental function.

Arguments

`X` An expression of numeric type.

`Y` (optional)
An expression of type integer or real. `Y` must not be present if `X` is of type complex.

Result Type

The result is of type double-precision complex.

Result Value

If `Y` is present, the result is the double-precision complex value whose real part is `DBLE(X)` and whose imaginary part is `DBLE(Y)`.

If `Y` is absent and `X` is of type integer or real, the result is the double-precision complex value whose real part is `DBLE(X)` and whose imaginary part is zero. If `Y` is absent and `X` is of type complex, the result is `X` (converted to double precision if necessary).

NOTE: A reference to `DCMPLX(X,Y)` is equivalent to `CMPLX(X,Y,0d0)`.

Examples

```
DCMPLX( 1, 2 ) → (1.,2.d0)
DCMPLX( 1., 2. ) → (1.,2.d0)
DCMPLX( 1+1 ) → (2.,0.d0)
DCMPLX( (1,2) ) → (1.,2.d0)
```

$$\texttt{DCMPLX}\left(\begin{bmatrix} -1 & 3 \\ -5 & 2 \end{bmatrix}\right) \rightarrow \begin{bmatrix} (-1,0.d0) & (3,0.d0) \\ (-5,0.d0) & (2,0.d0) \end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

DIAGONAL(ARRAY [, FILL])

Description

Creates a diagonal matrix from a rank-one array.

Kind

Transformational function.

Arguments

ARRAY A rank-one array of numeric or logical type.

FILL (optional)

A scalar of the same type as ARRAY. It may be omitted, in which case it is as if it were present with the value shown in the table below.

Type of ARRAY	Value of BOUNDARY
INTEGER	0
REAL	0.
DOUBLE PRECISION	0.d0
COMPLEX	(0., 0.)
DOUBLE COMPLEX	(0., 0d0)
LOGICAL	.FALSE.

Result Type

The result has the same type as ARRAY, with rank two and shape [*n*, *n*], where *n* is the size of ARRAY.

Result Value

Element (*i*,*i*) of the result has value ARRAY(*i*) for $1 \le i \le n$; all other elements have the value FILL.

Examples

DIAGONAL([3[*T*]]) $\rightarrow \begin{bmatrix} T & . & . \\ . & T & . \\ . & . & T \end{bmatrix}$

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

```
DIAGONAL( [11.,22.,33.], FILL = -1. ) → [ 11.  -1.  -1. ]
                                        [ -1.  22.  -1. ]
                                        [ -1.  -1.  33. ]
```


Copyright © 1989–1994 Thinking Machines Corporation

DIGITS(X)

Description

Returns the number of significant digits used to represent numbers of the same type and kind type as the argument.

Kind

Inquiry function.

Argument

`X` A scalar or array of type `INTEGER` or `REAL`.

Result Type

Scalar of type `INTEGER*4`.

Result Value

The result is one of the following values, depending on the kind type of the argument.

Data Type	Value
`INTEGER*4`	31
`INTEGER*8`	63
`REAL*4`	24
`REAL*8`	53

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

DIM(X, Y)

Description

Returns the difference between two numbers if that difference is positive, otherwise returns zero.

Kind

Elemental function.

Arguments

X	An expression of type integer or real.
Y	An expression of the same type as X.

Result Type

Same type as X.

Result Value

The value of the result is X−Y if X is greater than Y, and zero otherwise.

Examples

```
DIM(-3., 2.) → 0.
```

$$\texttt{DIM}\left(\begin{bmatrix}-3 & 0 & 7\\ -1 & 5 & 4\end{bmatrix}, 2\right) \rightarrow \begin{bmatrix}0 & 0 & 5\\ 0 & 3 & 2\end{bmatrix}$$

For any values X and Y of the same type (either integer or real),

```
DIM(X,Y) = MAX(X-Y,0)
```


Copyright © 1989–1994 Thinking Machines Corporation

DOT_PRODUCT(VECTOR_A, VECTOR_B)

Description

Performs dot-product multiplication of numeric or logical vectors.

Compatibility Note

For back compatibility, CM Fortran also accepts the intrinsic function DOTPRODUCT, which is equivalent to DOT_PRODUCT.

Kind

Transformational function.

Arguments

VECTOR_A
: A rank-one array of numeric or logical type.

VECTOR_B
: A rank-one array of numeric type if VECTOR_A is of numeric type, or of type logical if VECTOR_A is of type logical. It must be of the same size as VECTOR_A.

Result Type

The result is scalar. If the arguments are of type logical, the result is of type logical. If the arguments are of numeric type, the type of the result is the type of VECTOR_A * VECTOR_B, according to Table 2, page 86.

Result Value

Case 1 VECTOR_A is of type integer or real:

The result has the value

```
SUM( VECTOR_A * VECTOR_B ).
```

If the vectors have size zero, the result has the value zero.

Copyright © 1989–1994 Thinking Machines Corporation

Case 2 VECTOR_A is of type complex:

The result has the value

```
SUM( CONJG(VECTOR_A) * VECTOR_B ) .
```

If the vectors have size zero, the result has the value zero.

Case 3 VECTOR_A is of type logical:

The result has the value

```
ANY( VECTOR_A .AND. VECTOR_B ) .
```

If the vectors have size zero, the result has the value .FALSE.

Examples

```
DOT_PRODUCT( [1,2,3], [2.,3.,4.] ) → 20.
```

If A = [1,2,3] and B = [(2,0),(3,0),(-2,1)], then

```
DOT_PRODUCT( A, B ) → (2,3)
```

and

```
DOT_PRODUCT( B, A ) → (2,-3)
```

```
DOT_PRODUCT( [T,.,T,.], [.,T,.,T] ) → F
```

```
DOT_PRODUCT( [T,.,.,T], [.,T,.,T] ) → T
```


Copyright © 1989–1994 Thinking Machines Corporation

DPROD(X, Y)

Description

Computes the double-precision real product of two single-precision real numbers.

Kind

Elemental function.

Arguments

`X`	An expression of type single-precision real.
`Y`	An expression of type single-precision real.

Result Type

Double-precision real.

Result Value

The value of the result is `DBLE ( X * Y )`.

Examples

```
DPROD(101.,-2.) → -202.d0
DPROD( 2., 3. ) → 6.d0
```

```
DPROD( [ -4.  7. ]  , 3. ) ⇒ [ -12.00000095d0   21.0000017d0 ]
       [  3.  .5 ]           [  -9.00000072d0    1.50000012d0 ]
```


Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

EOSHIFT(ARRAY, DIM, SHIFT *[*, BOUNDARY *]*)

Description

Performs an end-off shift of an array of rank one, or performs end-off shifts of each of the complete rank-one sections parallel to a given dimension of an array of rank two or greater. Elements are shifted out at one end of a section and copies of a boundary value are shifted in at the other end. Different sections may have different boundary values, and may be shifted by different amounts and in different directions.

Compatibility Note

`EOSHIFT` becomes compatible with Fortran 90 when a program is compiled with the `-f90syntax` switch. When the function is invoked without argument keywords, CM Fortran interprets the argument order as:

Invoked with:	**`EOSHIFT` is interpreted as:**
`-f90syntax`	`EOSHIFT(ARRAY, SHIFT [,BOUNDARY] [,DIM])`
`-nof90syntax` (default)	`EOSHIFT(ARRAY, DIM, SHIFT [,BOUNDARY])`

The use of keywords allows the arguments to appear in any order, regardless of the compilation switch.

When the `-f90syntax` switch is used, the `DIM` argument is optional. If `DIM` is absent, its value is treated as 1.

Kind

Transformational function.

Copyright © 1989–1994 Thinking Machines Corporation

Arguments

ARRAY An array of numeric or logical type.

DIM A scalar of type integer, with a value in the range $1 \leq \texttt{DIM} \leq n$, where n is the rank of ARRAY.

SHIFT An integer expression. SHIFT must be a scalar if ARRAY has rank one; otherwise, it may be a scalar or an array. If SHIFT is an array, it must be of shape equal to that of ARRAY with dimension DIM removed.

BOUNDARY (optional)
An expression of the same type as ARRAY. BOUNDARY must be scalar if ARRAY has rank one; otherwise, it may be either a scalar or an array with shape equal to that of ARRAY with dimension DIM removed. BOUNDARY may be omitted, in which case it is as if it were present with the scalar value shown in the table below.

Type of ARRAY	Value of BOUNDARY
INTEGER*4	0
INTEGER*8	0_64
REAL	0.
DOUBLE PRECISION	0.d0
COMPLEX	(0., 0.)
DOUBLE COMPLEX	(0., 0d0)
LOGICAL	.FALSE.

Result Type

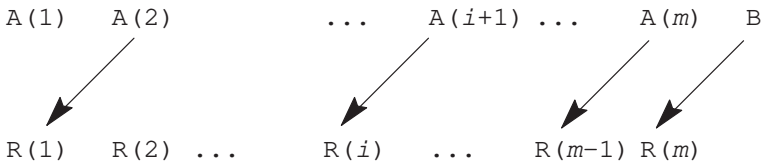
The result has the type and shape of ARRAY.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Result Value

Case 1a `EOSHIFT( ARRAY, DIM = 1, SHIFT [, BOUNDARY ] )`
`ARRAY` has rank one, `SHIFT` is a nonnegative scalar value:

If `SHIFT` is `1` and `ARRAY` has size *m*, the result is defined by the diagram

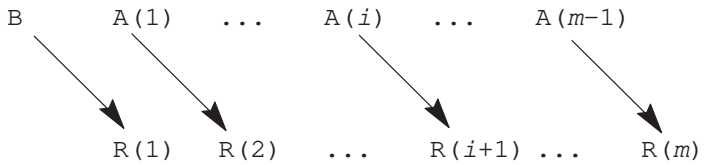


where the `A(i)` and the `R(i)` represent elements of `ARRAY` and the result, respectively, and `B` is `BOUNDARY`, or the appropriate default value from the table above if `BOUNDARY` is not present. The arrows indicate the movement of data.

If `SHIFT` is greater than `1`, the result is equal to `SHIFT` repeated applications of `EOSHIFT` to `ARRAY`, with `SHIFT = 1`. If `SHIFT` is equal to zero, the result is the original array.

Case 1b `EOSHIFT( ARRAY, DIM = 1, SHIFT [, BOUNDARY ] )`
`ARRAY` has rank one, `SHIFT` is a negative scalar value:

If `SHIFT` is `−1` and `ARRAY` has size *m*, the result is defined by the diagram



where the `A(i)` and the `R(i)` represent elements of `ARRAY` and the result, respectively, and `B` is `BOUNDARY`, or the appropriate default value from the table above if `BOUNDARY` is not present. The arrows indicate the movement of data.

If `SHIFT` is less than `−1`, the result is equivalent to `|SHIFT|` repeated applications of `EOSHIFT` to `ARRAY`, with `SHIFT = −1`.

Copyright © 1989–1994 Thinking Machines Corporation

Case 2 `EOSHIFT( ARRAY, DIM, SHIFT [, BOUNDARY ] )`
`ARRAY` has rank n>1, `SHIFT` is a scalar or an array:

The result is formed by performing an end-off shift of each of the complete rank-one sections of `ARRAY` parallel to dimension `DIM`. The value of section

$R(s_1, s_2, \ldots, s_{DIM-1}, :, s_{DIM+1}, \ldots, s_n)$

of a result array `R` is equal to

$EOSHIFT(ARRAY_{DIM}, DIM=1, SHIFT_{VAL}\ [, BOUNDARY_{VAL}\ ])$

where $ARRAY_{DIM}$ is the array section

$ARRAY(s_1, s_2, \ldots, s_{DIM-1}, :, s_{DIM+1}, \ldots, s_n)$

and where $SHIFT_{VAL}$ is the scalar `SHIFT` or is the array element

$SHIFT(s_1, s_2, \ldots, s_{DIM-1}, s_{DIM+1}, \ldots, s_n)$

if `SHIFT` is an array. Similarly, $BOUNDARY_{VAL}$ is the scalar `BOUNDARY` or is the array element

$BOUNDARY(s_1, s_2, \ldots, s_{DIM-1}, s_{DIM+1}, \ldots, s_n)$.

if `BOUNDARY` is an array.

Examples

Case 1 `ARRAY` has rank one, `SHIFT` is a scalar:

If `V = [1,2,3,4,5,6]`, the effect of shifting `V` end-off to the left by `3` positions is achieved by

```
EOSHIFT( V, DIM = 1, SHIFT = 3 ) → [4,5,6,0,0,0]
```

while the reference

```
EOSHIFT( V, DIM = 1, SHIFT = -2, BOUNDARY = 99 ) →
    [99,99,1,2,3,4]
```

achieves an end-off shift to the right by `2` positions with a boundary of `99`.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Case 2 ARRAY has rank *n*>1, SHIFT is a scalar or an array:

The rows of a matrix may all be shifted by the same amount or by different amounts and the boundary elements can be the same or different.

If $A = \begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \end{bmatrix}$, then

```
EOSHIFT( A, DIM = 1, SHIFT = 1, BOUNDARY = [1:4])→
```

$$\begin{bmatrix} 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```
EOSHIFT( A, DIM = 2, SHIFT = −1, BOUNDARY = [1:3]) →
```

$$\begin{bmatrix} 1 & 11 & 12 & 13 \\ 2 & 21 & 22 & 23 \\ 3 & 31 & 32 & 33 \end{bmatrix}$$

```
EOSHIFT( A, DIM = 1, SHIFT = [−2,−1,0,1] ) →
```

$$\begin{bmatrix} 0 & 0 & 13 & 24 \\ 0 & 12 & 23 & 34 \\ 11 & 22 & 33 & 0 \end{bmatrix}$$

```
EOSHIFT( A, DIM = 2, SHIFT = [−1,0,+1] ) →
```

$$\begin{bmatrix} 0 & 11 & 12 & 13 \\ 21 & 22 & 23 & 24 \\ 32 & 33 & 34 & 0 \end{bmatrix}$$

If $A = \begin{bmatrix} \begin{bmatrix} 111 & 121 & 131 & 141 \\ 211 & 221 & 231 & 241 \\ 311 & 321 & 331 & 341 \end{bmatrix} \\ \begin{bmatrix} 112 & 122 & 132 & 142 \\ 212 & 222 & 232 & 242 \\ 312 & 322 & 332 & 342 \end{bmatrix} \end{bmatrix}$, $S = \begin{bmatrix} -1 & 1 \\ 0 & 2 \\ 1 & 3 \end{bmatrix}$, and $B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$, then

```
EOSHIFT( A, DIM = 2, SHIFT = S, BOUNDARY = B ) →
```

$$\begin{bmatrix} \begin{bmatrix} 1 & 111 & 121 & 131 \\ 211 & 221 & 231 & 241 \\ 321 & 331 & 341 & 3 \end{bmatrix} \\ \begin{bmatrix} 122 & 132 & 142 & 4 \\ 232 & 242 & 5 & 5 \\ 342 & 6 & 6 & 6 \end{bmatrix} \end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

EPSILON(X)

Description

Returns a positive number that is almost negligible compared to unity in the model representing numbers of the same type and kind type as the argument.

Kind

Inquiry function.

Argument

`X` A scalar or array of type `REAL`.

Result Type

Scalar of the same type and kind type as `X`.

Result Value

The result has the value 2^{1-p} where p is as defined in Section 17.15 for the model representing numbers of the same type and kind type as `X`.

The result is one of the following values, depending on the kind type of the argument.

Data Type	Value
`REAL*4`	2^{-23}
`REAL*8`	2^{-52}

Copyright © 1989–1994 Thinking Machines Corporation

EXP(X)

Description

Computes the exponential function.

Kind

Elemental function.

Argument

X An expression of type real or complex.

Result Type

Same as X.

Result Value

The result is an approximation to the exponential function e^X. For complex values of X with the form (*a*,*b*), the result is calculated using the identity

```
EXP(X) = EXP(a) * CMPLX( COS(b), SIN(b), MOLD = X )
```

The imaginary part *b* is regarded as a value in radians.

Examples

```
EXP(0.) → 1.
EXP(1.d0) ⇒ 2.718281828459045d0
EXP((1,π)) ⇒ (-2.718282,0.)
```

$$\texttt{EXP}\left(\begin{bmatrix}0.00000 & 0.69315\\ 1.00000 & 1.57080\end{bmatrix}\right) \Rightarrow \begin{bmatrix}1.00000 & 2.00000\\ 2.71828 & 4.81048\end{bmatrix}$$

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

EXPONENT(X)

Description

Returns the exponent part of the argument when it is represented as a model number.

Kind

Elemental function.

Argument

`X` A scalar or array of type `REAL`.

Result Type

Scalar of type `INTEGER*4`.

Result Value

The result has a value equal to the exponent *e* of the model representation defined in Section 17.15 for the value of `X`, provided `X` is nonzero and *e* is within the range for integers. `EXPONENT(X)` has the value zero if `X` is zero.

In other words, `EXPONENT(X)` is the least power to which 2 is raised, to yield a value greater than or equal to `X`.

Examples

```
EXPONENT(1.0) → 1
EXPONENT(4.1) ⇒ 3
```


Copyright © 1989–1994 Thinking Machines Corporation

FIRSTLOC(MASK *[*, DIM *]*)

Description

Locates the first true element (in array element order) of a logical mask array, or locates the first true element of each of the complete rank-one sections parallel to a given dimension of an array of rank two or greater, effectively locating the "leading edge" of the set of true elements along the given dimension. The result is a logical mask that may be used to select elements of the original array.

Kind

Transformational function.

Arguments

`MASK` An array of type logical.

`DIM` (optional)
A scalar expression of type integer, with a value in the range $1 \leq$ `DIM` $\leq n$, where n is the rank of `MASK`.

Result Type

The result is an array of type logical with the same shape as `MASK`.

Result Value

Case 1 `FIRSTLOC ( MASK )`
`DIM` is absent:

The result has at most one true element. If `MASK` is all false, the result is all false. If `MASK` contains one or more true elements, the result has a single true element and it is in the position corresponding to the first true element (in array element order) of `MASK`.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Case 2 `FIRSTLOC( MASK, DIM )`
`DIM` is present:

The result is formed by applying `FIRSTLOC` to each of the rank-one array sections of `MASK` that lie parallel to dimension `DIM`. Thus the value of section

`R(`s_1`,`s_2`,...,`s_{DIM-1}`,:,`s_{DIM+1}`,...,`s_n`)`

of a result array `R` is equal to

`FIRSTLOC( MASK`$_{DIM}$ `)`

where `MASK`$_{DIM}$ is array section

`MASK(`s_1`,`s_2`,...,`s_{DIM-1}`,:,`s_{DIM+1}`,...,`s_n`)`.

Examples

Case 1 `DIM` is absent:

$$\texttt{FIRSTLOC(}\begin{bmatrix} . & . & T & . \\ . & T & T & . \\ . & T & . & T \end{bmatrix}\texttt{)} \rightarrow \begin{bmatrix} . & . & . & . \\ . & T & . & . \\ . & . & . & . \end{bmatrix}$$

`FIRSTLOC( [.,.,.,`T`,`T`,.,`T`] )` $\rightarrow$ `[.,.,.,`T`,.,.,.]`

Case 2 `DIM` is present:

The "top edge" of a two-dimensional mask is obtained by

$$\texttt{FIRSTLOC(}\begin{bmatrix} . & . & T & . \\ . & T & T & . \\ . & T & . & T \end{bmatrix}\texttt{, DIM = 1)} \rightarrow \begin{bmatrix} . & . & T & . \\ . & T & . & . \\ . & . & . & T \end{bmatrix}$$

while the "left edge" is obtained by

$$\texttt{FIRSTLOC(}\begin{bmatrix} . & . & T & . \\ . & T & T & . \\ . & T & . & T \end{bmatrix}\texttt{, DIM = 2)} \rightarrow \begin{bmatrix} . & . & T & . \\ . & T & . & . \\ . & T & . & . \end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

FLOOR(A)

Description

Returns the greatest integer less than or equal to `A`.

Kind

Elemental function.

Argument

`A` A scalar or array of type `REAL`.

Result Type

A scalar or array of type `INTEGER*4`.

Result Value

The result is equal to the greatest integer less than or equal to `A`. The result is undefined if the processor cannot represent this value as an `INTEGER*4`.

Examples

```
FLOOR(3.7) → 3

FLOOR(-3.7) → -4
```


Copyright © 1989–1994 Thinking Machines Corporation

FRACTION(X)

Description

Returns the fractional part of the argument when it is represented as a model number.

Kind

Elemental function.

Argument

X A scalar or array of type REAL.

Result Type

Same as X.

Result Value

The result has the value $X \times 2^{-e}$, where e is as defined in Section 17.15 for the model representation of X. If X has the value zero, the result has the value zero.

FRACTION(X) $\times$ $2^{\text{EXPONENT(X)}}$ = X.

Example

```
FRACTION(3.0) → 0.75
```


Copyright © 1989–1994 Thinking Machines Corporation

HUGE(X)

Description

Returns the largest number in the model representing numbers of the same type and kind type as the argument.

Kind

Inquiry function.

Argument

`X` A scalar or array of type `INTEGER` or `REAL`.

Result Type

Scalar of the same type and kind type as `X`.

Result Value

The result has the value 2^q-1 if `X` is of type integer and $(1-2^{-p})2^{e_{max}}$ if `X` is of type real, where q, p, and e_{max} are as defined in Section 17.15 for the model representing numbers of the same type and kind type as `X`.

The result is one of the following values, depending on the kind type of the argument.

Data Type	Value
`INTEGER*4`	$2^{31}-1$
`INTEGER*8`	$2^{63}-1$
`REAL*4`	$2^{128}(1-2^{-24})$
`REAL*8`	$2^{1024}(1-2^{-53})$

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

IAND(I, J)

Description

Performs the bitwise logical conjunction of two integer values.

Kind

Elemental function.

Arguments

`I`	An expression of type integer.
`J`	An expression of type integer.

Result Type

Integer.

Result Value

The result is the value obtained by combining `I` and `J` bit-by-bit according to the following truth table:

I	J	IAND(I,J)
1	1	1
1	0	0
0	1	0
0	0	0

The low-order (right-most) bit of an integer value has position `0`. If the type is `INTEGER*4`, the high-order (left-most) bit has position `31`; if the type is `INTEGER*8`, the high-order (left-most) bit has position `64`.

Examples

```
IAND( 8, 3 ) → 0
IAND( 1+4+16+512, [0:10] ) → [0,1,0,1,4,5,4,5,0,1,0]
```

$$\texttt{IAND(}\begin{bmatrix}0 & 3 & 6\\ 1 & 4 & 7\\ 2 & 5 & 8\end{bmatrix}\texttt{, 1+4+8)} \rightarrow \begin{bmatrix}0 & 1 & 4\\ 1 & 4 & 5\\ 0 & 5 & 8\end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

IBCLR(I, POS)

Description

Clears to zero a given bit of an integer value.

Kind

Elemental function.

Arguments

`I`	An expression of type integer.
`POS`	An expression of type integer. If `INTEGER*4`, the value is in the range $0 \le$ `POS` ≤ 31; if `INTEGER*8`, the value is in the range $0 \le$ `POS` ≤ 63.

Result Type

Integer.

Result Value

The result has the value of `I`, except that bit `POS` of the sequence of bits representing the result is cleared to zero.

The low-order (right-most) bit of an integer value has position `0`. If the type is `INTEGER*4`, the high-order (left-most) bit has position `31`; if the type is `INTEGER*8`, the high-order (left-most) bit has position `64`.

Examples

```
IBCLR(1025, POS = 10 ) → 1
IBCLR(1025, POS = 0 ) → 1024
IBCLR(-1, POS = 0 ) → -2
```

$$\texttt{IBCLR(}\begin{bmatrix}100011_2\\100111_2\\101011_2\\-1\end{bmatrix}\texttt{, POS =}\begin{bmatrix}1\\2\\2\\31\end{bmatrix}\texttt{)}\rightarrow\begin{bmatrix}100001_2\\100011_2\\101011_2\\2147483647\end{bmatrix}$$

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Copyright © 1989–1994 Thinking Machines Corporation

IBITS(I, POS, LEN)

Description

Extracts a given subfield of bits from an integer value.

Kind

Elemental function.

Arguments

`I`	An expression of type integer.
`POS`	An expression of type `INTEGER*4` with a nonnegative value satisfying the inequality `POS + LEN ≤ 32`, or `INTEGER*8` with a nonnegative value satisfying the inequality `POS + LEN ≤ 64`.
`LEN`	An expression of type `INTEGER*4` with a positive value satisfying the inequality `POS + LEN ≤ 32`, or `INTEGER*8` with a positive value satisfying the inequality `POS + LEN ≤ 64`.

Result Type

Integer.

Result Value

The subfield of `I` of length `LEN` bits beginning at bit `POS` is extracted; the result is the extracted subfield, right-justified, with high-order bits cleared to zero.

The low-order (right-most) bit of an integer value has position `0`. If the type is `INTEGER*4`, the high-order (left-most) bit has position `31`; if the type is `INTEGER*8`, the high-order (left-most) bit has position `64`.

Examples

`IBITS(`$2^{10}+2^9$`, POS = 8, LEN = 2 )` $\rightarrow$ `2`

`IBITS(`$2^{10}+2^9$`, POS = 8, LEN = 3 )` $\rightarrow$ `6`

`IBITS(-1, POS = 1, LEN = 2 )` $\rightarrow$ `3`

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

$$\texttt{IBITS(}\begin{bmatrix}101011_2\\100111_2\\101011_2\\-1\end{bmatrix}\texttt{, POS =}\begin{bmatrix}0\\2\\3\\30\end{bmatrix}\texttt{, LEN =}\begin{bmatrix}2\\4\\3\\2\end{bmatrix}\texttt{)}\rightarrow\begin{bmatrix}0011_2\\1001_2\\0101_2\\0011_2\end{bmatrix}$$

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Copyright © 1989–1994 Thinking Machines Corporation

IBSET(I, POS)

Description

Sets to one a given bit of an integer value.

Kind

Elemental function.

Arguments

`I`	An expression of type integer.
`POS`	An expression of type integer. If `INTEGER*4`, the value is in the range $0 \le POS \le 31$; if `INTEGER*8`, the value is in the range $0 \le POS \le 63$.

Result Type

Integer.

Result Value

The result has the value of `I`, except that bit `POS` of the sequence of bits representing the result is set to one.

The low-order (right-most) bit of an integer value has position `0`. If the type is `INTEGER*4`, the high-order (left-most) bit has position `31`; if the type is `INTEGER*8`, the high-order (left-most) bit has position `63`.

Examples

```
IBSET(1000, POS = 2 ) → 1004
IBSET(1024, POS = 0 ) → 1025
IBSET(0, POS = 31 ) → -2147483648
```

$$\texttt{IBSET(}\begin{bmatrix}100011_2\\100111_2\\101011_2\\0\end{bmatrix}\texttt{, POS =}\begin{bmatrix}4\\1\\2\\31\end{bmatrix}\texttt{)}\rightarrow\begin{bmatrix}110011_2\\100111_2\\101111_2\\2^{31}\end{bmatrix}$$

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

ICHAR(C)

Description

Returns the position of a character in the ASCII collating sequence. (The function is the inverse of the function CHAR.)

Kind

Elemental function (restricted to scalar arguments).

Argument

C	A scalar expression of type character, with length one. (It must be an ASCII character.)

Result Type

A scalar of type integer.

Result Value

The result is a value in the range 0 to 127 indicating the position of C in the ASCII collating sequence. For any ASCII characters C and D, the relation

```
C .LE. D
```

is true if and only if

```
ICHAR(C) .LE. ICHAR(D)
```

is true; the relation

```
C .EQ. D
```

is true if and only if

```
ICHAR(C) .EQ. ICHAR(D)
```

is true.

Examples

```
ICHAR("A") → 65          ICHAR("1") → 49
ICHAR("B") → 66          ICHAR("2") → 50
ICHAR("Z") → 90          ICHAR("(") → 40
ICHAR("z") → 122
```


Copyright © 1989–1994 Thinking Machines Corporation

IEOR(I, J)

Description

Performs the bitwise logical exclusive disjunction of two integer values.

Kind

Elemental function.

Arguments

`I`	An expression of type integer.
`J`	An expression of type integer.

Result Type

Integer.

Result Value

The result is the value obtained by combining `I` and `J` bit-by-bit according to the following truth table:

I	J	IEOR(I,J)
1	1	0
1	0	1
0	1	1
0	0	0

The low-order (right-most) bit of an integer value has position `0`. If the type is `INTEGER*4`, the high-order (left-most) bit has position `31`; if the type is `INTEGER*8`, the high-order (left-most) bit has position `63`.

Examples

```
IEOR( 8, 3 ) → 11
IEOR( 1+4+16, [0:10] ) → [21,20,23,22,17,16,19,18,29,28,31]
```

$$\texttt{IEOR(}\begin{bmatrix}0 & 3 & 6\\ 1 & 4 & 7\\ 2 & 5 & 8\end{bmatrix}\texttt{, 1+4+8)} \rightarrow \begin{bmatrix}13 & 14 & 11\\ 12 & 9 & 10\\ 15 & 8 & 5\end{bmatrix}$$

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Copyright © 1989–1994 Thinking Machines Corporation

INDEX(STRING, SUBSTRING)

Description

Returns the starting position of a substring within a string.

Kind

Elemental function (restricted to scalar arguments).

Arguments

`STRING`	A scalar expression of type character.
`SUBSTRING`	A scalar expression of type character.

Result Type

Integer.

Result Value

The result is the minimum value of `I` such that

```
STRING( I : I + LEN(SUBSTRING) - 1 ) .EQ. SUBSTRING
```

or zero if there is no such value of `I`. Zero is returned if

```
LEN(STRING) < LEN(SUBSTRING)
```

Examples

```
INDEX( "CM Fortran", "M" ) → 2
INDEX( "CM Fortran", "o" ) → 5
INDEX( "CM Fortran", "rtr" ) → 6
INDEX( "CM Fortran", "RTR" ) → 0
INDEX( "CM Fortran", "CM Fortran" ) → 1
INDEX( "CM Fortran", LINE ) → 0,
   if LINE = "Once upon a midnight dreary"
```


Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

INT(A *[*, KIND *]*)

Description

Converts a number to integer type.

Compatibility Note

When a program is not compiled with the `-f90syntax` switch, it is necessary to use the `KIND=` keyword if the `KIND` argument is used. If the `-f90syntax` switch is used, no keyword is necessary.

Kind

Elemental function.

Arguments

`A` An expression of numeric type.

`KIND` (optional)
A scalar integer constant expression indicating the length of the result type. When providing this argument, it is necessary to use the keyword `KIND=`, unless the code will be compiled with the `-f90syntax` switch.

Result Type

Integer. If `KIND` is present, the kind type parameter is `32` for an `INTEGER*4` result or `64` for an `INTEGER*8` result; otherwise, the kind type parameter is `32`.

Result Value

Case 1 `A` is of type integer:

The result is `A`.

Case 2 `A` is of type real:

The result is the integer nearest to `A`, truncated toward zero. If $|A| < 1$, the result is zero. If $|A| \geq 1$, the result is the integer whose magnitude is the largest integer that does not exceed the magnitude of `A` and whose sign is the same as that of `A`.

Copyright © 1989–1994 Thinking Machines Corporation

Case 3 `A` is of type complex:

The result is obtained by applying the Case 2 rule to the real part of `A`.

The result is undefined if the processor cannot represent the result in the specified integer type.

Examples

```
INT(-6) → -6
INT(0.999) → 0
INT(-3.7) → -3
INT((5,6)) → 5
INT((7,8d0)) → 7
INT(-4, KIND=64) → -4_64
INT(0.3, KIND=64) → 0_64
INT((2,8), KIND=64) → 2_64
```

$$\texttt{INT}\left(\begin{bmatrix} 0.6 & 27.9 \\ -0.4 & 5.2 \end{bmatrix}\right) \Rightarrow \begin{bmatrix} 0 & 27 \\ 0 & 5 \end{bmatrix}$$

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

IOR(I, J)

Description

Performs the bitwise logical inclusive disjunction of two integer values.

Kind

Elemental function.

Arguments

`I`	An expression of type integer.
`J`	An expression of type integer.

Result Type

Integer.

Result Value

The result is the value obtained by combining `I` and `J` bit-by-bit according to the following truth table:

I	J	IOR(I,J)
1	1	1
1	0	1
0	1	1
0	0	0

The low-order (right-most) bit of an integer value has position `0`. If the type is `INTEGER*4`, the high-order (left-most) bit has position `31`; if the type is `INTEGER*8`, the high-order (left-most) bit has position `63`.

Examples

```
IOR( 8, 3 ) → 11
IOR( 1+4+16, [0:10] ) → [21,21,23,23,21,21,23,23,29,29,31]
```

$$\texttt{IOR(}\begin{bmatrix}0 & 3 & 6\\ 1 & 4 & 7\\ 2 & 5 & 8\end{bmatrix}\texttt{, 1+4+8)} \rightarrow \begin{bmatrix}13 & 15 & 15\\ 13 & 13 & 15\\ 15 & 13 & 13\end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

ISHFT(I, SHIFT)

Description

Performs a bitwise logical shift of an integer value.

Kind

Elemental function.

Arguments

`I`	An expression of type integer.
`SHIFT`	An expression of type integer. If `INTEGER*4`, the value is in the range $-32 \le$ `SHIFT` ≤ 32; if `INTEGER*8`, the value is in the range $-64 \le$ `SHIFT` ≤ 64.

Result Type

Integer.

Result Value

The result is the value obtained by shifting the bits of `I` by $|$`SHIFT`$|$ bit positions. If `SHIFT` is positive, the shift is to the left; if `SHIFT` is negative, the shift is to the right; and if `SHIFT` is zero, no shift is performed. Bits shifted out are lost, and zeros are shifted in from the opposite end.

The low-order (right-most) bit of an integer value has position `0`. If the type is `INTEGER*4`, the high-order (left-most) bit has position `31`; if the type is `INTEGER*8`, the high-order (left-most) bit has position `63`.

Examples

`ISHFT( 1000, SHIFT = 2 )` $\rightarrow$ `4000`

`ISHFT(` 2^{12}`, SHIFT = -2 )` $\rightarrow 2^{10}$

$$\texttt{ISHFT(}\begin{bmatrix}100011_2\\100111_2\\101111_2\\100011_2\end{bmatrix}\texttt{, SHIFT =}\begin{bmatrix}1\\2\\3\\0\end{bmatrix}\texttt{)}\rightarrow\begin{bmatrix}001000110_2\\010011100_2\\101111000_2\\000100011_2\end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

$$\texttt{ISHFT}\left(\begin{bmatrix}100011_2\\100111_2\\101111_2\\100011_2\end{bmatrix}, \texttt{SHIFT} = \begin{bmatrix}-1\\-2\\-3\\31\end{bmatrix}\right) \rightarrow \begin{bmatrix}00010001_2\\00001001_2\\00000101_2\\2^{31}\end{bmatrix}$$

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

ISHFTC(I, SHIFT, SIZE)

Description

Performs a bitwise circular shift of a low-order subfield of an integer value, leaving high-order bits unchanged.

Kind

Elemental function.

Arguments

`I`	An expression of type integer.
`SHIFT`	An expression of type integer, with a value in the range $-\texttt{SIZE} \le \texttt{SHIFT} \le \texttt{SIZE}$.
`SIZE`	An expression of type integer, with a value in the range $1 \le \texttt{SIZE} \le 32$.

Result Type

Integer.

Result Value

The result is the value obtained by shifting the `SIZE` low-order bits of `I` by $|\texttt{SHIFT}|$ bit positions. If `SHIFT` is positive, the shift is to the left; if `SHIFT` is negative, the shift is to the right; and if `SHIFT` is zero, no shift is performed. No bits are lost, and the unshifted bits are unaltered.

The low-order (right-most) bit of an integer value has position `0`. If the type is `INTEGER*4`, the high-order (left-most) bit has position `31`; if the type is `INTEGER*8`, the high-order (left-most) bit has position `63`.

Examples

$$\texttt{ISHFTC(1025, SHIFT = -2, SIZE = 16)} \rightarrow 16640$$

$$\texttt{ISHFTC(} \begin{bmatrix} 100011_2 \\ 100111_2 \\ 101111_2 \\ 100011_2 \end{bmatrix} \texttt{, SHIFT =} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 0 \end{bmatrix} \texttt{, SIZE = 5)} \rightarrow \begin{bmatrix} 100110_2 \\ 111100_2 \\ 111011_2 \\ 100011_2 \end{bmatrix}$$

Copyright © 1989–1994 Thinking Machines Corporation

$$\texttt{ISHFTC(}\begin{bmatrix}00110110_2\\01100110_2\\11000110_2\\11000110_2\end{bmatrix}\texttt{, SHIFT =}\begin{bmatrix}-1\\-2\\-3\\-5\end{bmatrix}\texttt{, SIZE =}\begin{bmatrix}2\\4\\5\\6\end{bmatrix}\texttt{)}\rightarrow\begin{bmatrix}00110101_2\\01101001_2\\11011000_2\\11001100_2\end{bmatrix}$$

Note

The `INTEGER*8` type is supported only on CM-5 systems with vector units.

Copyright © 1989–1994 Thinking Machines Corporation