Compiler Library/XL Reference Manual

HP 3000 MPE/iX Computer Systems

Edition 2

Manufacturing Part Number: 32650-90029
E1088

U.S.A. October 1988

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).

Acknowledgments

UNIX is a registered trademark of The Open Group.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Print History

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

First Edition.....November 1987.....31510A.00.00

Second Edition.....October 1988.....31510A.00.01

Preface

The *Compiler Library/XL Reference Manual* is the reference to mathematical
and utility functions and procedures available on the MPE XL operating system.

This manual assumes the user has a working knowledge of the language or languages to be used and of the MPE XL operating system.

This manual contains the following:

Chapter 1	describes the fomat of the function and procedures descriptions, describes conventions, compares data types, shows the internal representation of the data types, and summarizes the functions and procedures in a series of tables.
Chapter 2	describes the mathematical functions and procedures in alphabetical order.
Chapter 3	describes the utility procedures in alphabetical order.
Chapter 4	discusses the packed-decimal procedures in alphabetical order.
Appendix A	describes the error routine and lists the library error messages, possible causes for the errors, and a user action.

Additional Documentation

More information on the languages and operating system can be found in the following manuals:

* *HP FORTRAN 77/XL Reference Manual* (31501-90010)
* *HP COBOL II Reference Manual* (31500-90001)
* *HP COBOL II/XL Reference Manual Supplement* (31500-90005)
* *HP Pascal Reference Manual* (31502-90001)
* *Trap Handling Programmers' Guide* (32650-90026)

Conventions

The following conventions are used in this manual:

1. All numbers are decimal values unless otherwise noted.
2. A dollar sign ($) prefix indicates a hexadecimal numer (Chapter 4 only).
3. In all examples, a blank space is represented by a Å.
4. Mathematical notation in the test includes the following definitions:

   A value *x* in the range (*a,b*) means *a<x<b*

   A value *x* in the range [*a,b*] means *a<=x<=b*

   A value *x* in the range (*a,b*] means *a<x<=b*

   A value *x* in the range [*a,b*) means *a<b*

Chapter 1 Introduction

The Compiler Library/XL functions and procedures perform input/output, internal data conversion, mathematical functions, error-reporting functions and packed-decimal operations. This chapter describes the format of the function and procedure descriptions, compares data types, shows the internal representation of the data types, and summarizes the functions and procedures in a series of tables.

Format of the Functions Description

The functions in Chapter 2, "Mathematical Functions and Procedures" and Chapter 3, "Utility Procedures" are described in a standard format; the following items are included in the format, when applicable:

Declaration

Contains the parts of the function or procedure declaration that define the requirements for actual parameters (arguments) included in a function call, procedure call, or calling sequence.

Accuracy

Describes the function accuracy, using the following notation:

x = true value of the argument(s)

y = computed value of the argument(s)

f = true value of the result

g = computed value of the result

|x-y| = absolute error in the argument(s)

|x-y/x| = relative error in the argument(s)

|f-g| = absolute error in the result(s)

|f-g/f| = relative error in the result(s)

Attributes

Parameters:	Describes the type and range of value(s) allowed by the function.
Result:	Describes the type and range of value(s).
HP FORTRAN 77/XL:	Where applicable, names the corresponding HP FORTRAN 77/XL intrinsic or external function.
Error:	Briefly describes the error conditions.

Comments

When needed, comments are supplied.

Special Values

When needed, special values are supplied.

Compiler Library Data Types

Table 1-1 below shows the corresponding data types between HP Pascal, HP FORTRAN 77/XL, HP COBOL II/XL and HP C/XL.

NOTE Many of the examples shown in this document are written in Pascal and are described in terms of Pascal and Pascal data types.

Table 1-1. Corresponding Data Types

HP Pascal	HP FORTRAN 77/XL	HP COBOL II/XL	HP C/XL
LONGREAL	DOUBLE PRECISION or REAL*8	Not applicable	double
SHORTINT	INTEGER*2	PIC S9 through PIC S9999 COMP	short
REAL	REAL or REAL*4	Not applicable	float
INTEGER	INTEGER or INTEGER*4	PIC S9(5) through PIC S9(9) COMP	int
TYPE complex=RECORD realpart:REAL; imagpart:REAL: END;	COMPLEX or COMPLEX*8	Not applicable	typedef struct{ float realpart; float imagpart; }COMPLEX;
complex_matx= ARRAY [1..N,1..N] of complex	COMPLEX complex_matx (N,N)	Not applicable	COMPLEX complex_matx [N][N];
longreal_matx= ARRAY [1..N,1..N] of LONGREAL	DOUBLE PRECISION longreal_matx (N,N)	Not applicable	double longreal_matx [N][N];
real_matx= ARRAY [1..N,1..N] of REAL	REAL real_matx (N,N)	Not applicable	float real_matx [N][N];

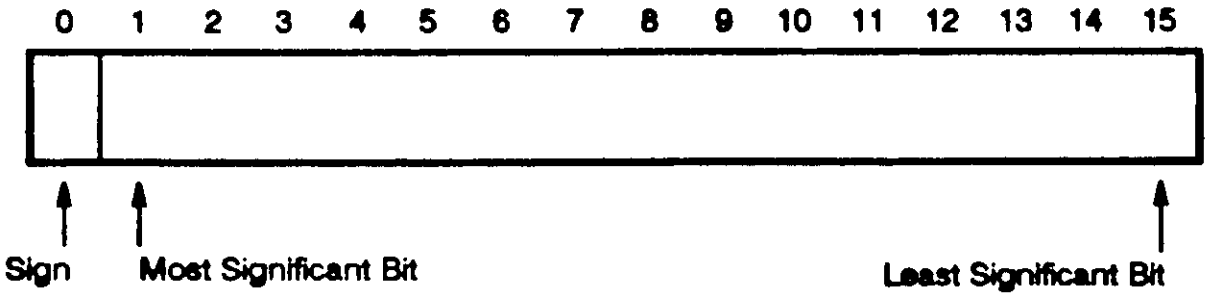
Internal Representation of Data

This section shows how different data types are internally represented.

Shortint

Range: [-32768,32767] in two bytes

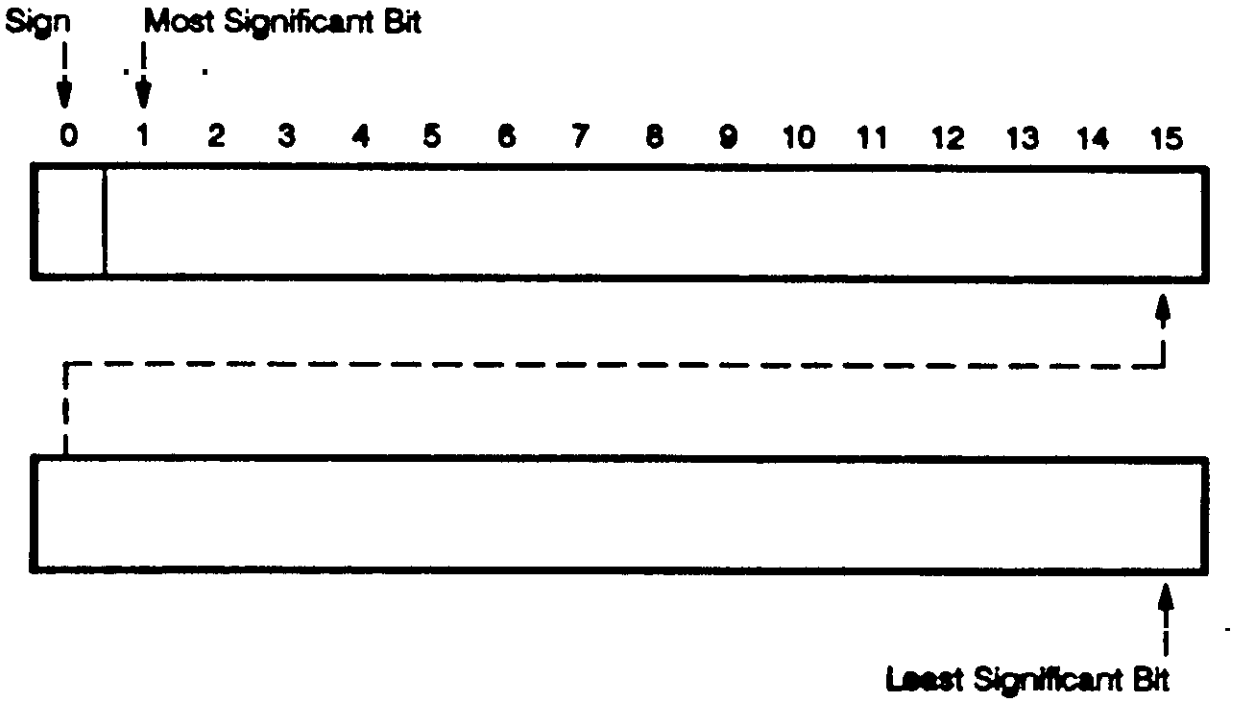
Representation:



Integer

Range: [-2147483648,2147483647] in four bytes (concatenated)

Representation:

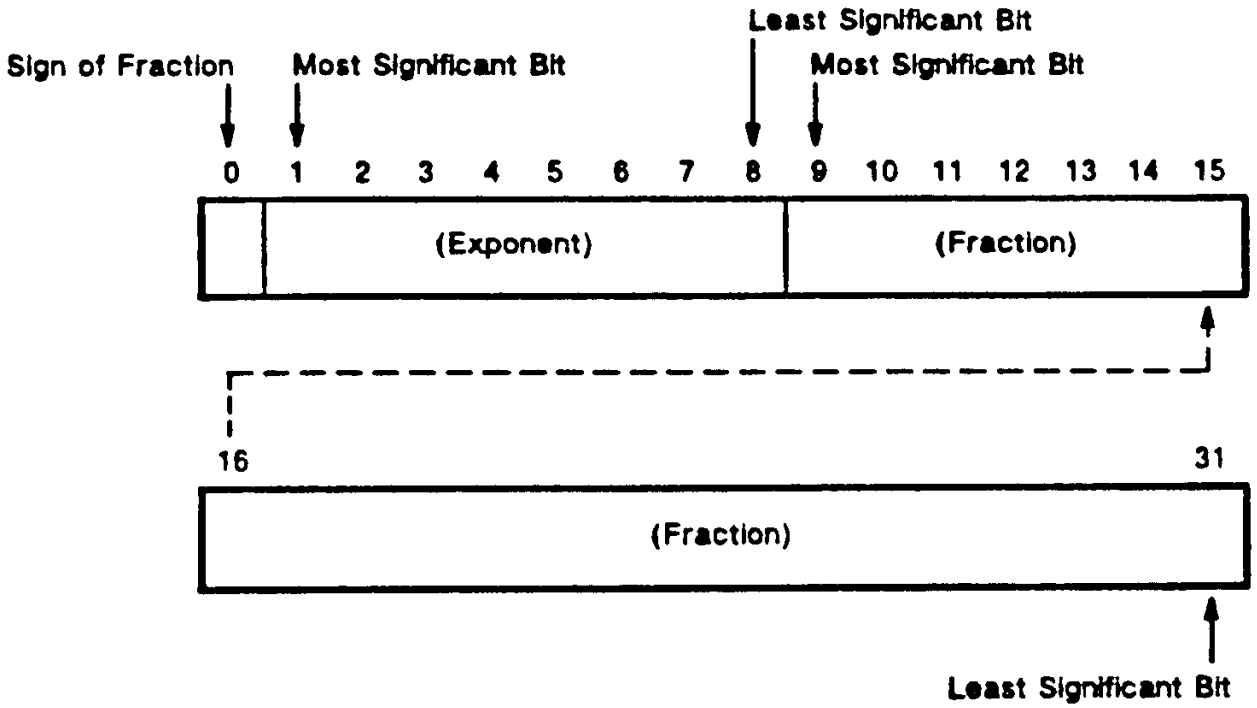


Real

Range:

[-3.402823 x 1038 , -1.401298 x 10-45] and
0.0 and
1.401298 x 10-45 , 3.402823 x 1038] in four bytes (concatenated)

Representation:



For real numbers stored in normalized form, the exponent portion has a range of [1,254] and the fraction portion has a range of [0,223 -1]. The sign bit indicates the sign of the fraction portion (0 indicates a positive value and 1 indicates a negative value).

Real numbers in normalized form have an implied value of 1.0 to the left of the fraction's most significant bit. Thus, a normalized decimal value equals

(-1)Sign * 2Exponent-127 * (1.0+*Fraction* * 2-23)

and has a range of

(-1)Sign * [1.175494 x 10-38 ,3.402823 x 1038]

However, the IEEE Standard P754 allows decimal numbers whose absolute values are smaller than 1.175494 x 10-38 . This so-called denormalized number with *Exponent* equal to zero has the value

(-1)Sign * 2-126 * (0.0 + *Fraction* * 2-23)

and has a range of

(-1)Sign * [1.401298 x 10-45 ,1.175494 x 10-38)

When the significant bit, the exponent, and the fraction are zero, the decimal value is equal to 0.0.

NaN (not a number) can be encoded in the real number format as follows:

When all the exponent bits are set to 1 and at least one of the fraction bits is non-zero, there is a NaN regardless of the sign bit. For example:

Exponent = 255; fraction <> 0

Infinity can also be encoded in the real number format as follows:

When all exponent bits are set to 1 and all of the fraction bits are set to 0, there is an Infinity: (-1)Sign Infinity. For example:

Exponent = 255; fraction = 0

Longreal

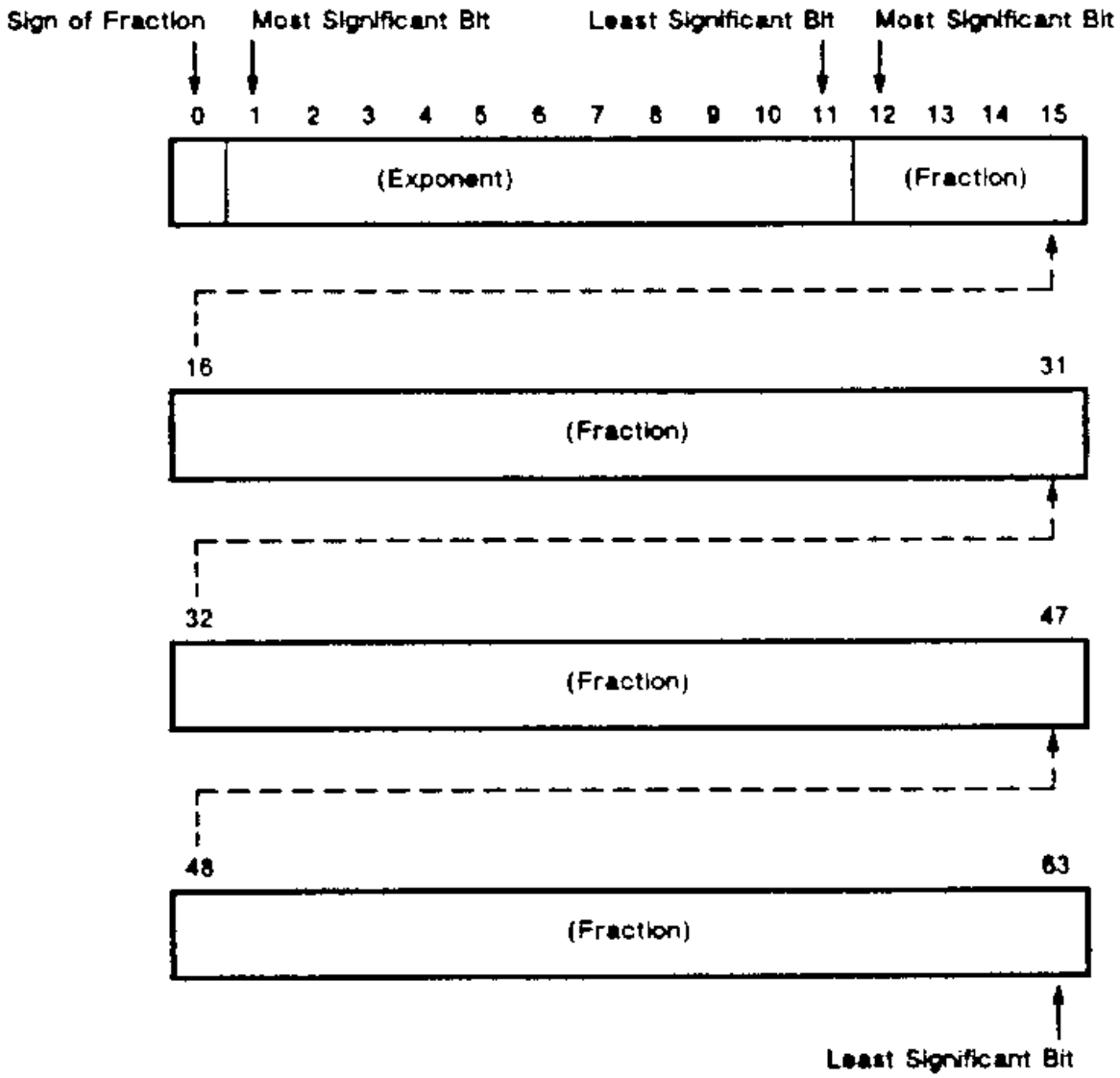
Range:

[-1.797693134862316 x 10308 ,-4.940656458412465 x 10-324] and

0.0 and

[4.940656458412465 x 10-324 ,1.797693134862316 x 10308] in eight bytes (concatenated)

Representation:



For longreal numbers stored in normalized form, the exponent portion has a range of [1,2046] and the fraction portion has a range of [0,252 -1]. The sign bit indicates the sign of the fraction portion (0 indicates a positive value and 1 indicates a negative value).

Longreal numbers in normalized form have an implied value of 1.0 to the left of the fraction's most significant bit. Thus, a normalized decimal value equals the following:

(-1)Sign * 2Exponent-1023 * (1.0+*Fraction* * 2-52)

and has a range of

(-1)Sign [2.225073858507201 x 10-308 ,1.797693134862316 x 10308]

However, the IEEE Standard P754 allows decimal numbers whose absolute values are smaller than 2.225073858507201 x 10-308 . Denormalized numbers with *Exponent* equal to zero have the value

(-1)Sign * 2-1022 *(0.0 + *Fraction* * 2-52)

and has a range of

(-1)Sign * [4.940656458412465 x 10-324 ,2.225073858507201 x 10-308]

When the significant bit, the exponent, and the fraction are zero, the decimal value is equal to 0.0.

NaN (not a number) can be encoded in the longreal number format, as follows:

When all the exponent bits are set to 1 and at least one of the fraction bits is non-zero, there is a NaN regardless of the sign bit. For example:

Exponent = 2047; fraction <> 0

Infinity can also be encoded in the longreal number format as follows:

When all exponent bits are set to 1 and all of the fraction bits are set to 0, there is an Infinity: (-1)Sign Infinity.

For example:

Exponent = 2047; fraction = 0

NOTE The procedures described in Chapter 4, "Packed-Decimal Procedures," use packed-decimal and external-decimal numbers.

Functions and Procedures

Table 1-2 through Table 1-11 summarize the functions and procedures in the Compiler Library. Chapter 2 describes mathematical functions. Chapter 3 describes the utility procedures. Chapter 4 describes the packed-decimal procedures. Appendix A describes an error message handling routine and the Compiler Library/XL error messages.

Table 1-2. Absolute Value Functions

Function	Description
CABS or CABS'	Calculates the absolute value of a complex number.
DABS or DABS'	Calculates the absolute value of a longreal number.
DSIGN or DSIGN'	Calculates the absolute value of one longreal number and gives it the sign of a second longreal number.
ISIGN or ISIGN'	Calculates the absolute value of a shortint number and gives it the sign of a second shortint number.
JSIGN or JSIGN'	Calculates the absolute value of one integer number and gives it the sign of a second integer number.
SIGN or SIGN'	Calculates the absolute value of one real number and gives it the sign of a second real number.

Table 1-3. Number Conversion Functions

Function	Description
AINT or AINT'	Truncates a real number to an integer number in real representation.
AMOD or AMOD'	Calculates a real number modulus a second real number.
DDINT or DDINT'	Truncates a longreal number to an integer number in longreal representation.
DFIX or DFIX'	Truncates a longreal number to an integer number.
DFLOAT or DFLOAT'	Converts an integer number to a longreal number.
DMOD	Calculates a longreal number modulus a second longreal number.
IFIX or IFIX'	Truncates a real number to a shortint number.
INT or INT'	Truncates a real number to a shortint number.

Table 1-4. Exponent, Root, and Logarithm Functions

Function	Description
ALOG or ALOG'	Calculates the natural logarithm of a positive real number.
ALOG10	Calculates the base 10 logarithm of a positive real number.
CEXP or CEXP'	Calculates *ex* , where *x* is a complex number.
CLOG or CLOG'	Calculates the natural logarithm of a complex number.
CSQRT or CSQRT'	Calculates the square root of a complex number.
DEXP or DEXP'	Calculates *ex* , where *x* is a longreal number.
DLOG or DLOG'	Calculates the natural logarithm of a positive longreal number.
DLOG10	Calculates the base 10 logarithm of a positive longreal number.
DSQRT or DSQRT'	Calculates the square root of a longreal number.
EXP or EXP'	Calculates *ex* , where *x* is a real number.
SQRT or SQRT'	Calculates the square root of a real number.

Table 1-5. Trigonometry Functions

Function	Description
ATAN or ATAN'	Calculates the arctangent of a real number.
ATAN2 or ATAN2'	Calculates the arctangent of the quotient of two real numbers.
CCOS or CCOS'	Calculates the cosine of a complex number.
CCOSH or CCOSH'	Calculates the hyperbolic cosine of a complex number.
COS or COS'	Calculates the cosine of a real number in radians.
COSH or COSH'	Calculates the hyperbolic cosine of a real number.
CSIN or CSIN'	Calculates the sine of a complex number.
CSINH or CSINH'	Calculates the hyperbolic sine of a complex number.
CTAN or CTAN'	Calculates the tangent of a complex number.
CTANH or CTANH'	Calculates the hyperbolic tangent of a complex number.
DATAN or DATAN'	Calculates the arctangent of a longreal number.
DATAN2	Calculates the arctangent of the quotient of two longreal numbers.
DCOS or DCOS'	Calculates the cosine of a longreal number in radians.
DCOSH or DCOSH'	Calculates the hyperbolic cosine of a longreal number.
DSIN or DSIN'	Calculates the sine of a longreal number in radians.
DSINH or DSINH'	Calculates the hyperbolic sine of a longreal number.
DTAN or DTAN'	Calculates the tangent of a longreal number in radians.

DTANH or DTANH'	Calculates the hyperbolic tangent of a longreal number.
SIN or SIN'	Calculates the sine of a real number in radians.
SINH or SINH'	Calculates the hyperbolic tangent of a real number.
TAN or TAN'	Calculates the tangent of a real number in radians.
TANH or TANH'	Calculates the hyperbolic tangent of a real number.

Table 1-6. Matrix Functions

Function	Description
CINVERT	Inverts a square matrix of complex numbers.
DINVERT	Inverts a square matrix of longreal numbers.
INVERT	Inverts a square matrix of real numbers.

Table 1-7. Random Number Functions

Function	Description
RAND or RAND'	Generates the next element of a sequence of pseudo-random numbers.
RAND1 or RAND1'	Generates a random number for use as a starting point for RAND.

Table 1-8. Integer Arithmetic Functions

Function	Description
DADD	Calculates the sum of two integer numbers.
DCMP	Compares two integer numbers.
DDIV or DDIV'	Calculates the quotient of one integer number divided by another integer number.
DMPY or DMPY'	Calculates the product of two integer numbers.
DNEG	Negates an integer number.
DREM or DREM'	Calculates the remainder of one integer number divided by another integer number.
DSUB	Calculates the difference between two integer numbers.

Table 1-9. Intrinsic Function

Function	Description
XLIBTRAP	Specifies a user-defined function to process library errors.

Table 1-10. Utility Procedures

Procedure	Description
EXTIN' or HPEXTIN	Converts a byte array of ASCII numeric data to an internal representation.
INEXT' or HPINEXT	Converts an internal representation of a number to a byte array for ASCII numeric data.

Table 1-11. Packed-Decimal Procedures

Procedure	Description
HPPACADDD	Adds two packed-decimal numbers.
HPPACCMPD	Compares two packed-decimal operands and sets a comparison code.
HPPACCVAD	Converts from ASCII to packed-decimal
HPPACCVBD	Converts a binary number to packed-decimal.
HPPACCVDA	Converts a packed-decimal number to ASCII.
HPPACCVDB	Converts a packed-decimal number to binary.
HPPACLONGDIVD or HPPACDIVD	Calculates the quotient, or the quotient and the remainder, of a packed-decimal dividend and a packed-decimal divisor.
HPPACMPYD	Multiplies two packed-decimal numbers
HPPACNSLD	Performs a packed-decimal normalizing left shift.
HPPACSLD	Performs a packed-decimal left shift.
HPPACSRD	Performs a packed-decimal right shift.
HPPACSUBD	Subtracts one packed-decimal number from another.

You can use the HP Pascal intrinsic statement to declare the compiler library procedure or function. For HP Pascal, for example, the external declaration could be declared:

```
FUNCTION SIN (VAR x:REAL): REAL;EXTERNAL;
```

or the external declaration could more easily be declared as:

```
FUNCTION SIN:REAL;INTRINSIC;
```

You can use the HP FORTRAN 77 system intrinsic statement to declare the compiler library procedure or function.

For HP FORTRAN 77/XL, for example, the external declaration could be

declared by:

```
$ALIAS EXTIN = 'EXTIN"' (%REF, %REF, %VAL, %VAL, %VAL, %VAL, %REF, %REF)
```

or the intrinsic statement could be used:

```
SYSTEM INTRINSIC HPEXTIN
```

In HP C/XL, the intrinsic pragma is used to declare an intrinsic. For example, instead of using the following declaration:

```
extern void HPPACCVDA( );
```

you could write:

```
#pragma intrinsic HPPACCVDA
```

The function names ending with a prime (EXTIN', for example) can be called by HP Pascal or HP FORTRAN 77/XL programs using the ALIAS command. See the *HP Pascal Reference Manual* or the *HP FORTRAN 77/XL Reference Manual* for a detailed description of the ALIAS command. For example, specify the following statement in your HP Pascal program:

```
PROCEDURE EXTIN $ALIAS 'EXTIN"'$;INTRINSIC;
```

Or specify the following statement in your HP FORTRAN 77/XL program:

```
$ALIAS EXTIN = 'EXTIN"' (%REF, %REF, %VAL, %VAL, %VAL, %VAL, %REF, %REF)
```

Chapter 2 Mathematical Functions and Procedures

This chapter describes the mathematical functions and procedures in alphabetical order.

NOTE The function names ending with a prime (EXTIN', for example) can be called by HP Pascal programs using the ALIAS command with the EXTERNAL or INTRINSIC statements. For example, here is a declaration that uses ALIAS with EXTERNAL:

```
PROCEDURE EXTIN $ALIAS 'EXTIN"' $(
        ANYVAR charstring : bytearray;
           VAR stringlen  : SHORTINT;
               decplaces  : SHORTINT;
               datatype   : SHORTINT;
               scale      : SHORTINT;
               delimiters : SHORTINT;
        ANYVAR result     : INTEGER;
           VAR error      : SHORTINT
             )
    EXTERNAL;
```

The next example illustrates the preferred approach of using ALIAS with INTRINSIC:

```
PROCEDURE EXTIN $ALIAS 'EXTIN"'; INTRINSIC;
```

In HP FORTRAN 77/XL, you must declare routine names with a prime, as follows:

```
$ALIAS EXTIN = 'EXTIN"'(%REF, %REF, %VAL, %VAL, %VAL,
%VAL, %REF, %REF)"
```

AINT (or AINT')

AINT truncates a real number to an integer number in real representation.

Declaration

```
FUNCTION AINT(x:REAL):REAL; EXTERNAL;
```

Attributes

Parameter:	A real number.
Result:	A real number.
HP FORTRAN 77/XL:	Intrinsic function: Y=AINT(X).
Error:	If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "AINT(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

ALOG (or ALOG') or ALOG10

ALOG calculates the natural logarithm of a positive real number; ALOG10 calculates the base 10 logarithm of a positive real number.

Declaration

```
FUNCTION ALOG(VAR x:REAL):REAL; EXTERNAL;
                or

FUNCTION ALOG10(VAR x:REAL):REAL; EXTERNAL;
```

Accuracy

$$\text{When} \quad \frac{|x-y|}{|x|} \simeq \epsilon ,$$

$$\text{maximum} \quad \frac{|f-g|}{|f|} \simeq \frac{\epsilon}{|\ln(x)|}$$

Attributes

Parameter:	A positive real number.
Result:	A real number (ALOG10 = ALOG*log10 (*e*)).
HP FORTRAN 77/XL:	Intrinsic function ALOG(X) or ALOG10(X).
Error:	If the argument is a NaN or if it is negative, there are two possible actions. If the INVALID trap is enabled, the message "ALOG(X): X< 0.0 OR X=NaN" or "ALOG10(X): X< 0.0 OR X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned. See Appendix A, "Compiler Library Messages" for more details.

Special Values

```
alog (+Infinity) = +Infinity

alog (0) = -Infinity
```

AMOD (or AMOD')

AMOD calculates a first real number modulus a second real number.

Declaration

```
FUNCTION AMOD(x,y:REAL):REAL; EXTERNAL;
```

Attributes

Parameters:	Both arguments are real numbers.
Result:	A real number.
HP FORTRAN 77/XL:	Intrinsic function: Z=AMOD(X,Y).
Error:	If either of the arguments is a NaN (or the first argument is infinity, or the second

argument is zero) there are two possible actions. If the INVALID trap is enabled, the message "AMOD(X,Y): ANY OF X AND Y=NaN OR X=INFINITY OR Y=0.0" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

ATAN (or ATAN')

ATAN calculates the arctangent of a real number.

Declaration

```
FUNCTION ATAN(VAR x:REAL):REAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \in$ **, maximum** $|f-g| \simeq \frac{\in}{1+x^2}$

Attributes

Parameter:	A real number.
Result:	A representable real number in the range [-pi/2,pi/2]. See Chapter 1 for details on the internal representation of real numbers.
HP FORTRAN 77/XL:	Intrinsic function: Y=ATAN(X).
Error:	If the argument is a NaN, there are two cases. If the INVALID trap is enabled, the message "ATAN(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

Special Values

atan (+Infinity) = pi/2

atan (-Infinity) = -pi/2

ATAN2 (or ATAN2')

ATAN2 calculates the arctangent of the quotient of two real numbers.

Declaration

```
FUNCTION ATAN2(VAR x,y:REAL):REAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \in$ **, maximum** $|f-g| \simeq \frac{\in}{1+w^2}$

Where $w = \frac{x}{y}$

Attributes

Parameters: Real numbers; both must not be zero.

Result: A representable real number in one of the following ranges:

	$y \geq 0$	$y < 0$
$x \geq 0$	[0, pi/2]	[pi/2, pi]
$x < 0$	[−pi/2, 0)	(−pi, −pi/2)

See Chapter 1 for details on the internal representation of real numbers.

HP FORTRAN 77/XL: Intrinsic function: Z=ATAN2(X,Y)

Error: If either of the arguments is a NaN or if both arguments are zero (or Infinity), there are two possible actions. If the INVALID trap is enabled, the message "ATAN2(X,Y): X=Y=0.0 OR X=Y=INFINITY OR ANY OF X AND Y=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

See Appendix A for more details.

CABS (or CABS')

CABS calculates the absolute value of a complex number.

Declaration

```
TYPE complex=RECORD
              realpart:REAL;
              imagpart:REAL;
             END;

FUNCTION CABS(VAR x:complex):REAL; EXTERNAL;
```

Accuracy

Depends on the accuracy of SQRT.

Attributes

Parameter: Any complex number representable in two real numbers: one for *a* and one for *b*.

Result: A non-negative real number.

HP FORTRAN 77/XL: Intrinsic function: Y=CABS(X).

Error: If either part of the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "CABS(X): ANY PART OF X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

If *a* and/or *b* are near the overflow threshold (*a* and/or *b* approximately equals to 1038), the message "CABS OVERFLOW" occurs. See Appendix A for more details.

CCOS (or CCOS')

CCOS calculates the cosine of a complex number.

Declaration

Complex numbers in HP FORTRAN 77/XL programs are represented as an ordered pair (a 2-element array) of real numbers: one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy eight bytes.

```
TYPE complex=RECORD
              realpart:REAL;
              imagpart:REAL;
             END;

FUNCTION CCOS(VAR x:complex):complex; EXTERNAL;
```

Accuracy

Depends on the accuracy of SIN, COS, and EXP.

Attributes

Parameter: A complex number.

Result: A complex number.

HP FORTRAN 77/XL: Intrinsic function: Y=CCOS(X).

Error: See the SIN, COS, and EXP error descriptions.

CCOSH (or CCOSH')

CCOSH calculates the hyperbolic cosine of a complex number.

Declaration

Complex numbers in HP FORTRAN 77/XL programs are represented as an ordered pair of real numbers: one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy eight bytes.

```
TYPE complex=RECORD
              realpart:REAL;
              imagpart:REAL;
             END;

FUNCTION CCOSH(VAR x:complex):complex; EXTERNAL;
```

Accuracy

Depends on the accuracy of SIN, COS, COSH, and SINH.

Attributes

Parameter:	A complex number.
Result:	A complex number.
HP FORTRAN 77/XL:	Callable as a system intrinsic: Y=CCOSH(X).
Error:	See the SIN, COS, COSH, and SINH error descriptions.

CEXP (or CEXP')

CEXP calculates e^x, where x is a complex number.

Declaration

Complex numbers in HP FORTRAN 77/XL programs are represented as an ordered pair of real numbers: one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy eight bytes.

```
TYPE complex=RECORD
               realpart:REAL;
               imagpart:REAL;
             END;

FUNCTION CEXP(VAR x:complex):complex; EXTERNAL;
```

Accuracy

Depends on the accuracy of SIN, COS, and EXP.

Attributes

Parameter:	Any complex number representable in two representable real numbers: one for the real part *a* and one for the imaginary part b; *a* must be in the range [-87.3366, 88.7228]. See Chapter 1 for details on the internal representation of real numbers.
Result:	A complex number.
HP FORTRAN 77/XL:	Intrinsic function: Y=CEXP(X).
Error:	See the SIN, COS, and EXP error descriptions.

CINVERT

CINVERT inverts a square matrix containing complex elements represented as an ordered pair of real elements: one for the real part *a* and one for the imaginary part *b*. The resulting inverse is stored over the input matrix.

Declaration

```
TYPE complex=RECORD
               realpart:REAL;
               imagpart:REAL;
             END;
     complex_matx=ARRAY[1..N, 1..N] OF complex;

PROCEDURE CINVERT (VAR N:SHORTINT; ANYVAR A:complex_matx;
                   VAR D:SHORTINT); EXTERNAL;
```

Attributes

Parameters:	N is an integer for the order of the matrix; A

	is a real identifier of the matrix for D, an integer identifier.
Result:	CINVERT replaces the original matrix, and D equals 1 if the matrix is non-singular or 0 if the matrix is singular and no inverse exists.
HP FORTRAN 77/XL:	Callable as a system intrinsic: CALL CINVERT(N, A, D) or as a system intrinsic statement.
Error:	None.

CLOG (or CLOG')

CLOG calculates the natural logarithm of a complex number.

Declaration

Complex numbers in HP FORTRAN 77/XL programs are represented as an ordered pair of real numbers: one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy eight bytes.

```
TYPE complex=RECORD
              realpart:REAL;
              imagpart:REAL;
             END;

FUNCTION CLOG(VAR x:complex):complex; EXTERNAL;
```

Accuracy

Depends on the accuracy of ALOG, SQRT, and ATAN2.

Attributes

Parameter:	Any nonzero complex number representable in two real numbers; both parts must not be zero.
Result:	A complex number.
HP FORTRAN 77/XL:	Intrinsic function: Y=CLOG(X).
Error:	See the ALOG, SQRT, and ATAN2 error descriptions.

COS (or COS')

COS calculates the cosine of a real number in radians.

Declaration

```
FUNCTION COS(VAR x:REAL):REAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \in$, maximum $|f-g| \simeq \in \sin x$

Attributes

Parameter:	A real number in radians.
Result:	A representable real number in the range [-1.0, 1.0]. See Chapter 1 for details on the internal representation of real numbers.
HP FORTRAN 77/XL:	Intrinsic function: Y=COS(X).
Error:	If the argument is a NaN or an Infinity, there are two cases. If the INVALID trap is enabled, the message "COS(X): X=NaN OR INFINITY" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

COSH (or COSH')

COSH calculates the hyperbolic cosine of a real number.

Declaration

```
FUNCTION COSH(VAR x:REAL):REAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \epsilon$, maximum $|f-g| \simeq \epsilon \sinh x$

Attributes

Parameter:	A real number.
Result:	A real number.
HP FORTRAN 77/XL:	Intrinsic function: Y=COSH(X).
Error:	If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "COSH(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned. If ABS(X) is >= 88.7196, the result cannot be represented and the message "COSH(X) OVERFLOW" occurs. See Appendix A for more details.

CSIN (or CSIN')

CSIN calculates the sine of a complex number.

Declaration

Complex numbers in HP FORTRAN 77/XL programs are represented as an ordered pair of real numbers: one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy eight bytes.

```
TYPE complex=RECORD
               realpart:REAL;
               imagpart:REAL;
             END;

FUNCTION CSIN(VAR x:complex):complex; EXTERNAL;
```

Accuracy

Depends on the accuracy of SIN, COS, and EXP.

Attributes

Parameter:	A complex number.
Result:	A complex number.
HP FORTRAN 77/XL:	Intrinsic function: Y=CSIN(X).
Error:	See the SIN, COS, and EXP error descriptions.

CSINH (or CSINH')

CSINH calculates the hyperbolic sine of a complex number.

Declaration

Complex numbers in HP FORTRAN 77/XL programs are represented as an ordered pair of real numbers: one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy eight bytes.

```
TYPE complex=RECORD
              realpart:REAL;
              imagpart:REAL;
             END;

FUNCTION CSINH(VAR x:complex):complex; EXTERNAL;
```

Accuracy

Depends on the accuracy of SIN, COS, SINH, and COSH.

Attributes

Parameter:	A complex number.
Result:	A complex number.
HP FORTRAN 77/XL:	Callable as a system intrinsic: Y=CSINH(X).
Error:	See the SIN, COS, SINH, and COSH error descriptions.

CSQRT (or CSQRT')

CSQRT calculates the square root of a complex number.

Declaration

Complex numbers in HP FORTRAN 77/XL programs are represented as an ordered pair of real numbers: one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy eight bytes.

```
TYPE complex=RECORD
              realpart:REAL;
              imagpart:REAL;
             END;

FUNCTION CSQRT(VAR x:complex):complex; EXTERNAL;
```

Accuracy

Depends on the accuracy of SQRT.

Attributes

Parameter:	Any complex number representable in two real numbers: one for *a* and one for *b*.
Result:	A complex number.
HP FORTRAN 77/XL:	Intrinsic function: Y=CSQRT(X).
Error:	If any part of the argument is a NaN, there are two cases. If the INVALID trap is enabled, the message "COMPLEX SQRT: ANY PART OF THE ARGUMENT = NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned to both the real and imaginary part.

CTAN (or CTAN')

CTAN calculates the tangent of a complex number.

Declaration

Complex numbers in HP FORTRAN 77/XL programs are represented as an ordered pair of real numbers: one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy eight bytes.

```
 TYPE complex=RECORD
               realpart:REAL;
               imagpart:REAL;
              END;

FUNCTION CTAN(VAR x:complex):complex; EXTERNAL;
```

Accuracy

Depends on the accuracy of SIN, COS, SINH, and COSH.

Attributes

Parameter:	A complex number.
Result:	A complex number.
HP FORTRAN 77/XL:	Intrinsic function: Y=CTAN(X).
Error:	See SIN, COS, SINH, and COSH error descriptions.

CTANH (or CTANH')

CTANH calculates the hyperbolic tangent of a complex number.

Declaration

Complex numbers in HP FORTRAN 77/XL programs are represented as an ordered pair of real numbers: one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy eight bytes.

```
TYPE complex=RECORD
               realpart:REAL;
               imagpart:REAL;
              END;

FUNCTION CTANH(VAR x:complex):complex; EXTERNAL;
```

Accuracy

Depends on the accuracy of SIN, COS, COSH, and SINH.

Attributes

Parameter:	A complex number.
Result:	A complex number.
HP FORTRAN 77/XL:	Callable as a system intrinsic: Y=CTANH(X).
Error:	See SIN, COS, SINH, and COSH error descriptions.

DABS (or DABS')

DABS calculates the absolute value of a longreal number.

Declaration

```
FUNCTION DABS(x:LONGREAL):LONGREAL; EXTERNAL;
```

Attributes

Parameter:	A longreal number.
Result:	A longreal number.
HP FORTRAN 77/XL:	Intrinsic function: Y=DABS(X).
Error:	If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "DABS(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

DADD

DADD calculates the sum of two integer numbers.

Declaration

```
FUNCTION DADD(VAR d1,d2:INTEGER):INTEGER; EXTERNAL;
```

Attributes

Parameters:	Both arguments are integer numbers.
Result:	An integer number.
HP FORTRAN 77/XL:	Callable as a system intrinsic: I=DADD(J,K)
Error:	If the result cannot be represented by an integer number, an arithmetic trap INTEGER OVERFLOW occurs if traps are enabled.

DATAN (or DATAN')

DATAN calculates the arctangent of a longreal number.

Declaration

```
FUNCTION DATAN(VAR x:LONGREAL):LONGREAL; EXTERNAL;
```

Accuracy

$$\text{When} \quad |x-y| \simeq \epsilon \text{ , maximum } |f-g| \simeq \frac{\epsilon}{(1+x^2)}$$

Attributes

Parameter:	A longreal number.
Result:	A representable longreal number in the range [-pi/2, pi/2]. See Chapter 1 for details on the internal representation of longreal numbers.
HP FORTRAN 77/XL:	Intrinsic function: Y=DATAN(X).
Error:	If the argument is a NaN, there are two cases. If the INVALID trap is enabled, the message "DATAN(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

Special Values

datan (+Infinity) = pi/2

datan (-Infinity) = -pi/2

DATAN2

DATAN2 calculates the arctangent of the quotient of two longreal numbers.

Declaration

```
FUNCTION DATAN2(VAR x,y:LONGREAL):LONGREAL; EXTERNAL;
```

Accuracy

$$\text{When} \quad |x-y| \simeq \epsilon \text{ , maximum } |f-g| \simeq \frac{\epsilon}{(1+w^2)}$$

$$\text{When} \quad w = \frac{x}{y}$$

Attributes

Parameters:	Longreal numbers; both must not be zero.
Result:	A representable longreal number in one of the following ranges:

	$y \geq 0$	$y < 0$
$x \geq 0$	[0, pi/2]	[pi/2, pi]
$x < 0$	[-pi/2, 0)	(-pi, -pi/2)

See Chapter 1 for details on the internal representation of longreal numbers.

HP FORTRAN 77/XL: Intrinsic function: Z=DATAN2(X,Y)

Error: If either of the arguments is a NaN or if both arguments are zero or infinity, there are two possible actions. If the INVALID trap is enabled, the message "DATAN2(X,Y): X=Y=0.0 OR X=Y=INFINITY OR ANY OF X AND Y=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

See Appendix A for more details.

DCMP

DCMP compares two integer numbers.

Declaration

```
FUNCTION DCMP(VAR d1,d2:INTEGER):INTEGER; EXTERNAL;
```

Attributes

Parameters: Both arguments are integer numbers.

Result:

```
-1 if D1 < D2
0 if D1 = D2
1 if D1 > D2
```

HP FORTRAN 77/XL: Callable as a system intrinsic: I = DCMP(J,K). For example, the arithmetic IF statement

```
IF(I) 10,20,30
```

directs the program as follows:

```
go to statement 10 if X < Y
go to statement 20 if X = Y
go to statement 30 if X > Y
```

Error: None.

DCOS (or DCOS')

DCOS calculates the cosine of a longreal number in radians.

Declaration

```
FUNCTION DCOS(VAR x:LONGREAL):LONGREAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \epsilon$, maximum $|f-g| \simeq \epsilon \sin x$

Attributes

Parameter: A longreal number in radians.

Result:	A representable longreal number in the range [-1.0, 1.0]. See Chapter 1 for details on the internal representation of longreal numbers.
HP FORTRAN 77/XL:	Intrinsic function: Y=DCOS(X).
Error:	If the argument is a NaN or an Infinity, there are two cases. If the INVALID trap is enabled, the message "DCOS(X): X=NaN OR INFINITY" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned. When DABS (X) >= 2.6378256 x 107 , DCOS (X) completely loses significance. See Appendix A for more details.

DCOSH (or DCOSH')

DCOSH calculates the hyperbolic cosine of a longreal number.

Declaration

```
FUNCTION DCOSH(VAR x:LONGREAL):LONGREAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \epsilon$, maximum $|f-g| \simeq \epsilon \sinh x$

Attributes

Parameter:	A longreal number.
Result:	A longreal number.
HP FORTRAN 77/XL:	Intrinsic function: Y=DCOSH(X).
Error:	If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "DCOSH(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned. If DABS(X) >= 709.7795518826419, the result cannot be represented and the message "DCOSH(X) OVERFLOW" occurs. See Appendix A for more details.

DDINT (or DDINT')

DDINT truncates a longreal number to an integer number in longreal representation.

Declaration

```
FUNCTION DDINT(x:LONGREAL):LONGREAL; EXTERNAL;
```

Attributes

Parameter:	A longreal number.
Result:	A longreal number.
HP FORTRAN 77/XL:	Intrinsic function: Y=DDINT(X).

Error: If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "DINT(X) OR DDINT(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

DDIV (or DDIV')

DDIV calculates the quotient of one integer number divided by another integer number. See the description of the DREM procedure for the remainder.

Declaration

```
FUNCTION DDIV(VAR d1,d2:INTEGER):INTEGER; EXTERNAL;
```

Attributes

Parameters: Both arguments are integer numbers.

Result: An integer number (the quotient only).

HP FORTRAN 77/XL: Callable as a system intrinsic: I = DDIV(J,K)

Error: If the parameter D2 equals zero, the arithmetic trap INTEGER DIVIDE BY ZERO occurs if traps are enabled.

DEXP (or DEXP')

DEXP calculates e^x, where x is a longreal number.

Declaration

```
FUNCTION DEXP(VAR x:LONGREAL):LONGREAL; EXTERNAL;
```

Accuracy

$$\textbf{When } |x-y| \simeq \in \textbf{, maximum } \frac{|f-g|}{|f|} \simeq \in$$

Attributes

Parameter: A representable longreal number in the range [-708.396418532264, 709.782712893384]. See Chapter 1 for details on the internal representation of real numbers.

Result: A representable positive longreal number.

HP FORTRAN 77/XL: Intrinsic function: Y=DEXP(X).

Error: If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "DEXP(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

If the argument is >= 709.78271289338397 the message "DEXP(X) OVERFLOW" occurs and the result is infinity. If the argument is <= -708.39641853226408 the message "DEXP(X) UNDERFLOW" occurs and the result is set to zero.

Special Values

dexp (+Infinity) = +Infinity

dexp (-Infinity) = 0

DFIX (or DFIX')

DFIX truncates a longreal number to an integer number.

Declaration

```
FUNCTION DFIX(x:LONGREAL):INTEGER; EXTERNAL;
```

Attributes

Parameter:	A longreal number.
Result:	An integer number.
HP FORTRAN 77/XL:	Callable as a system intrinsic: I = DFIX(X)
Error:	If traps are enabled and if the truncated longreal number cannot be represented in the two words of the integer, the arithmetic trap INTEGER OVERFLOW occurs.

DFLOAT (or DFLOAT')

DFLOAT converts an integer number to a longreal number.

Declaration

```
FUNCTION DFLOAT(x:INTEGER):LONGREAL; EXTERNAL;
```

Attributes

Parameter:	An integer number.
Result:	A longreal number.
HP FORTRAN 77/XL:	Intrinsic function: I = DFLOAT(J)
Error:	None.

DINVERT

DINVERT inverts a square matrix containing longreal numbers; the resulting inverse is stored over the input matrix.

Declaration

```
TYPE longreal_matx=ARRAY[1..N, 1..N] OF longreal;

PROCEDURE DINVERT(n:SHORTINT; ANYVAR a:LONGREAL_matx;
                  VAR d:SHORTINT); EXTERNAL;
```

Attributes

Parameters:	n is an integer for the order of the matrix; a is a longreal identifier of the matrix for d, an integer identifier.
Result:	DINVERT replaces the input matrix. d is equal

to 1 if the matrix is non-singular or 0 if the matrix is singular and no inverse exists.

HP FORTRAN 77/XL: Callable as a system intrinsic:

```
CALL DINVERT(n, a, d)
```

Error: None.

DLOG (or DLOG') or DLOG10

DLOG calculates the natural logarithm of a positive longreal number; DLOG10 calculates the base 10 logarithm of a positive longreal number.

Declaration

```
FUNCTION DLOG(VAR x:LONGREAL):LONGREAL; EXTERNAL;
              or
 FUNCTION DLOG10(VAR x:LONGREAL):LONGREAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \frac{\in}{|x|}$, **maximum** $|f-g| \sim \frac{\in}{|1n(x)|}$

Attributes

Parameter: A representable positive longreal number. See Chapter 1 for details on the internal representation of longreal numbers.

Result: A longreal number (DLOG10 = DLOG * log10 (*e*)).

HP FORTRAN 77/XL: Intrinsic function: Y=DLOG(X) or Y=DLOG10(X).

Error: If the argument is a NaN or a negative number, there are two possible actions. If the INVALID trap is enabled, the message "DLOG(X): X< 0.0 OR X=NaN" or "DLOG10(X): X <0.0 OR X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

Special Values

dlog (+Infinity) = +Infinity

dlog (0) = -Infinity

DMOD

DMOD calculates a longreal number modulus a second longreal number.

Declaration

```
FUNCTION DMOD(VAR x,y:LONGREAL):LONGREAL; EXTERNAL;
```

Attributes

Parameters: Both arguments are longreal numbers.

Result: A longreal number.

HP FORTRAN 77/XL: Intrinsic function: Z=DMOD(X,Y).

Error: If either of the arguments is a NaN (or the first argument is infinity or the second argument is zero), there are two possible actions. If the INVALID trap is enabled, the message "DMOD(X,Y): ANY OF X AND Y=NaN OR X=INFINITY OR Y=0.0" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

DMPY (or DMPY')

DMPY calculates the product of two integer numbers.

Declaration

```
FUNCTION DMPY(VAR d1,d2:INTEGER):INTEGER; EXTERNAL;
```

Attributes

Parameters: Both arguments are integer numbers.

Result: An integer number.

HP FORTRAN 77/XL: Callable as a system intrinsic: I = DMPY(J,K)

Error: If traps are enabled and if the result cannot be represented by an integer number, the arithmetic trap INTEGER OVERFLOW occurs.

DNEG

DNEG negates an integer number.

Declaration

```
FUNCTION DNEG(VAR d:INTEGER):INTEGER; EXTERNAL;
```

Attributes

Parameter: An integer number.

Result: The integer number with the opposite sign.

HP FORTRAN 77/XL: Callable as a system intrinsic: I = DNEG(J)

Error: None.

DREM (or DREM')

DREM calculates the remainder of one integer number divided by another integer number. See the DDIV procedure for the quotient.

Declaration

```
FUNCTION DREM(VAR d1,d2:INTEGER):INTEGER; EXTERNAL;
```

Attributes

Parameters: Both arguments are integer numbers.

Result: An integer number (the remainder only).

HP FORTRAN 77/XL: Callable as a system intrinsic: I=DREM(J,K)

Error: If traps are enabled and if the parameter d2 equals zero, the arithmetic trap INTEGER DIVIDE

BY ZERO occurs.

DSIGN (or DSIGN')

DSIGN calculates the absolute value of the first longreal number and gives the number the sign of the second longreal number.

Declaration

```
FUNCTION DSIGN(x,y:LONGREAL):LONGREAL; EXTERNAL;
```

Attributes

Parameters:	Both arguments are longreal numbers; if the second number is zero, the sign is assumed to be positive.
Result:	A longreal number.
HP FORTRAN 77/XL:	Intrinsic function: Z=DSIGN(X,Y).
Error:	If either of the arguments is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "SIGN FUNCTION: ANY ARGUMENT=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

DSIN (or DSIN')

DSIN calculates the sine of a longreal number in radians.

Declaration

```
FUNCTION DSIN(VAR x:LONGREAL):LONGREAL; EXTERNAL;
```

Accuracy

$$\text{When} \quad |x-y| \simeq \epsilon \text{ , maximum } |f-g| \simeq \epsilon \cos x$$

Attributes

Parameter:	A longreal number in radians.
Result:	A representable longreal number in the range [-1.0, 1.0]. See Chapter 1 for details on the internal representation of longreal numbers.
HP FORTRAN 77/XL:	Intrinsic function: Y=DSIN(X).
Error:	If the argument is a NaN or an Infinity, there are two cases. If the INVALID trap is enabled, the message "DSIN(X): X=NaN OR INFINITY" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned. When ABS (X) > 2.6378256 x 107 , DSIN (X) completely loses significance. See Appendix A for more details.

DSINH (or DSINH')

DSINH calculates the hyperbolic sine of a longreal number.

Declaration

```
FUNCTION DSINH(VAR x:LONGREAL):LONGREAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \epsilon$, maximum $|f-g| \simeq \epsilon \cosh x$

Attributes

Parameter:	A longreal number.
Result:	A longreal number.
HP FORTRAN 77/XL:	Intrinsic function: Y=DSINH(X).
Error:	If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "DSINH(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned. If DSINH(X) >= 709.7795518826419, the result cannot be represented and the message "DSINH(X) OVERFLOW" occurs. See Appendix A for more details.

DSQRT (or DSQRT')

DSQRT calculates the square root of a longreal number.

Declaration

```
FUNCTION DSQRT(VAR x:LONGREAL):LONGREAL; EXTERNAL;
```

Accuracy

When $\frac{|x-y|}{|x|} \simeq \epsilon$, maximum $\frac{|f-g|}{|f|} \simeq \frac{1}{2} \epsilon$

Attributes

Parameter:	A non-negative longreal number.
Result:	A non-negative longreal number.
HP FORTRAN 77/XL:	Intrinsic function: Y=DSQRT(X).
Error:	If the argument is a NaN or if it is negative, there are two possible actions. If the INVALID trap is enabled, the message "DSQRT(X): X<0.0 OR X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

DSUB

DSUB calculates the difference between two integer numbers.

Declaration

```
FUNCTION DSUB(VAR d1,d2:INTEGER):INTEGER; EXTERNAL;
```

Attributes

Parameters:	An integer number.
Result:	An integer number.
HP FORTRAN 77/XL:	Callable as a system intrinsic: I = DSUB(J,K)
Error:	If traps are enabled and if the result cannot be represented by an integer, the arithmetic trap INTEGER OVERFLOW occurs.

DTAN (or DTAN')

DTAN calculates the tangent of a longreal number in radians.

Declaration

```
FUNCTION DTAN(VAR x:LONGREAL):LONGREAL; EXTERNAL;
```

Accuracy

$$\text{When } |x-y| \simeq \in \text{ , maximum } |f-g| \simeq \in \sec^2 x$$

Attributes

Parameter:	A longreal number in radians.
Result:	A longreal number.
HP FORTRAN 77/XL:	Intrinsic function: Y=DTAN(X).
Error:	If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "DTAN(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned. When DABS(X) > 149078413.0, DTAN(X) completely loses significance. See Appendix A for more details.

DTANH (or DTANH')

DTANH calculates the hyperbolic tangent of a longreal number.

Declaration

```
FUNCTION DTANH(VAR x:LONGREAL):LONGREAL; EXTERNAL;
```

Accuracy

$$\text{When } |x-y| \simeq \epsilon \text{ , maximum } |f-g| \simeq \epsilon \operatorname{sech}^2 x$$

Attributes

Parameter:	A longreal number.
Result:	A representable longreal number in the range [0.0, 1.0]. See Chapter 1 for details on the internal representation of longreal numbers.
HP FORTRAN 77/XL:	Intrinsic function: Y=DTANH(X).
Error:	If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "DTANH(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

EXP (or EXP')

EXP calculates *ex* , where *x* is a real number.

Declaration

```
FUNCTION EXP(VAR x:REAL):REAL; EXTERNAL;
```

Accuracy

$$\text{When} \quad |x-y| \simeq \epsilon \text{ , maximum } \frac{|f-g|}{|f|} \simeq \epsilon$$

Attributes

Parameter:	A representable real number in the range [-87.3366, 88.7228]. See Chapter 1 for details on the internal representation of real numbers.
Result:	A representable positive real number.
HP FORTRAN 77/XL:	Intrinsic function: Y=EXP(X).
Error:	If the argument is a NaN, there are two cases. If the INVALID trap is enabled, the message "EXP(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned. If the argument is >= 88.722839, the message "EXP(X) OVERFLOW" occurs and the result is infinity. If the argument is <= -87.336548, the message "EXP(X) UNDERFLOW" occurs and the result is set to zero.

Special Values

exp (+Infinity) = +Infinity

exp (-Infinity) = 0

IFIX (or IFIX')

IFIX truncates a real number to a shortint number.

Declaration

```
FUNCTION IFIX(x:REAL):SHORTINT; EXTERNAL;
```

Attributes

Parameter:	A representable real number in the range [-32768.0, 32767.0]. See Chapter 1 for details on the internal representation of real numbers.
Result:	An integer number.
HP FORTRAN 77/XL:	Intrinsic function: I=IFIX(X).
Error:	If traps are enabled and if the real number is outside the range stated, the arithmetic trap INTEGER OVERFLOW occurs.

INT (or INT')

INT truncates a real number to a shortint number.

Declaration

```
FUNCTION INT(x:REAL):SHORTINT; EXTERNAL;
```

Attributes

Parameter:	A representable real number in the range [-32768.0, 32767.0]. See Chapter 1 for details on the internal representation of real numbers.
Result:	An integer number.
HP FORTRAN 77/XL:	Intrinsic function: I=INT(X).
Error:	If traps are enabled and if the real number is outside the range stated, the arithmetic trap INTEGER OVERFLOW occurs.

INVERT

INVERT inverts a square matrix containing real numbers; the resulting inverse is stored over the input matrix.

Declaration

```
TYPE real_matx=ARRAY[1..N, 1..N] OF real;

PROCEDURE INVERT(n:SHORTINT; ANYVAR a:REAL_matx;
                 VAR d:SHORTINT); EXTERNAL;
```

Attributes

Parameters:	For n, an integer for the order of the matrix; for a, a real identifier of the matrix; for d, an integer identifier.
Result:	INVERT replaces the original matrix, and d is equal to one if the matrix is non-singular or zero if the matrix is singular and no inverse exists.
HP FORTRAN 77/XL:	Callable as a system intrinsic: `CALL INVERT(n, a, d)`
Error:	None.

ISIGN (or ISIGN')

ISIGN calculates the absolute value of a shortint number and gives the

number the sign of the second shortint number.

Declaration

```
FUNCTION ISIGN(j,k:SHORTINT):SHORTINT; EXTERNAL;
```

Attributes

Parameters:	Both arguments are shortint numbers; if the second number is zero, the sign is assumed to be positive.
Result:	An integer number.
HP FORTRAN 77/XL:	Intrinsic function: L=ISIGN(J,K)
Error:	None.

JSIGN (or JSIGN')

JSIGN calculates the absolute value of the first integer number and gives the number the sign of the second integer number.

Declaration

```
FUNCTION JSIGN(j,k:INTEGER):INTEGER; EXTERNAL;
```

Attributes

Parameters:	Both arguments are integer numbers; if the second number is zero, the sign is assumed to be positive.
Result:	An integer number.
HP FORTRAN 77/XL:	Intrinsic function: L=JSIGN(J,K).
Error:	None.

RAND (or RAND')

RAND generates the next element of a sequence of pseudo-random numbers.

Declaration

```
FUNCTION RAND(VAR x:INTEGER):REAL; EXTERNAL;
```

Attributes

Parameter:	A integer number.
Results:	A representable real number uniformly distributed in the range (0.0, 1.0) returned as the value of the routine, and a 32-bit quantity replacing the original value of the parameter. See Chapter 1 for details on the internal representation of real numbers.
HP FORTRAN 77/XL:	Callable as a system intrinsic: Y = RAND(I)

Comments

The parameter value for the initial call to RAND completely determines a sequence of pseudo-random numbers. Each time RAND returns a new value to the calling program, it also sets a new 32-bit value in place of the parameter. To continue the pseudo-random sequence thus initiated, that 32-bit value must be used as the parameter in the next call to RAND.

RAND1 (or RAND1')

RAND1 generates a random number, that can be used as the starting point for the RAND procedure.

Declaration

```
FUNCTION RAND1: INTEGER; EXTERNAL;
```

Attributes

Parameter:	None.
Result:	A 32-bit integer quantity.
HP FORTRAN 77/XL:	Callable as a system intrinsic:I=RAND1( )
Error:	None.

Comments

This random value is derived from the 31-bit logical quantity changed every millisecond by the MPE XL system timer.

SIGN (or SIGN')

SIGN calculates the absolute value of the first real number and gives the number the sign of the second real number.

Declaration

```
FUNCTION SIGN(x,y:REAL):REAL; EXTERNAL;
```

Attributes

Parameters:	Both arguments are real numbers; if the second number is zero, the sign is assumed to be positive.
Result:	A real number.
HP FORTRAN 77/XL:	Intrinsic function: Z=SIGN(X,Y).
Error:	If either of the arguments is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "SIGN FUNCTION: ANY ARGUMENT=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

SIN (or SIN')

SIN calculates the sine of a real number in radians.

Declaration

```
FUNCTION SIN(VAR x:REAL):REAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \in$ **, maximum** $|f-g| \simeq \in \cos x$

Attributes

Parameter:	A real number in radians.
Result:	A representable real number in the range [-1.0, 1.0]. See Chapter 1 for details on the internal representation of real numbers.
HP FORTRAN 77/XL:	Intrinsic function: Y=SIN(X).
Error:	If the argument is a NaN or an Infinity, there are two possible actions. If the INVALID trap is enabled, the message "SIN(X): X=NaN OR INFINITY" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

SINH (or SINH')

SINH calculates the hyperbolic sine of a real number.

Declaration

```
FUNCTION SINH(VAR x:REAL):REAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \in$, maximum $|f-g| \simeq \in \cosh x$

Attributes

Parameter:	A real number.
Result:	A real number.
HP FORTRAN 77/XL:	Intrinsic function: Y=SINH(X).
Error:	If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "SINH(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned. If ABS(X) >= 88.7196, the result cannot be represented and the message "SINH(X) OVERFLOW" occurs. See Appendix A for more details.

SQRT (or SQRT')

SQRT calculates the square root of a real number.

Declaration

```
FUNCTION SQRT(VAR x:REAL):REAL; EXTERNAL;
```

Accuracy

When $\frac{|x-y|}{|x|} \simeq \epsilon$, maximum $\frac{|f-g|}{|f|} \simeq \frac{1}{2} \epsilon$

Attributes

Parameter:	A non-negative real number.
Result:	A non-negative real number.
HP FORTRAN 77/XL:	Intrinsic function: Y=SQRT(X).
Error:	If the argument is a NaN or if it is negative, there are two possible actions. If the INVALID trap is enabled, the message "SQRT(X): X<0.0 OR X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

TAN (or TAN')

TAN calculates the tangent of a real number in radians.

Declaration

```
FUNCTION TAN(VAR x:REAL):REAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \in$, maximum $|f-g| \simeq \in \sec^2 x$

Attributes

Parameter:	A real number in radians.
Result:	A real number.
HP FORTRAN 77/XL:	Intrinsic function: Y=TAN(X)
Error:	If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "TAN(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned. When ABS(X) > 6433.0, TAN(X) completely loses significance. See Appendix A for more details.

TANH (or TANH')

TANH calculates the hyperbolic tangent of a real number.

Declaration

```
FUNCTION TANH(VAR x:REAL):REAL; EXTERNAL;
```

Accuracy

When $|x-y| \simeq \in$, maximum $|f-g| \simeq \in \text{sech}^2 x$

Attributes

Parameter:	A real number.
Result:	A representable real number in the range [0.0, 1.0]. See Chapter 1 for details on the internal representation of real numbers.
HP FORTRAN 77/XL:	Intrinsic function: Y=TANH(X).
Error:	If the argument is a NaN, there are two possible actions. If the INVALID trap is enabled, the message "TANH(X): X=NaN" occurs. Otherwise, the INVALID flag is set. In either case, a quiet NaN is returned.

Chapter 3 Utility Procedures

This chapter describes the utility procedures EXTIN'/HPEXTIN and INEXT'/HPINEXT.

EXTIN'/HPEXTIN

This utility procedure is callable by the names EXTIN' and HPEXTIN.

NOTE EXTIN' is provided for compatibility with MPE V. In new program development, it is advisable to call HPEXTIN for ease of use. The declaration is the same for both utility names.

EXTIN'/HPEXTIN converts a byte array containing an input string of ASCII digits into one of these internal representations:

* SHORTINT
* REAL
* INTEGER
* LONGREAL

See the "Comments" section for more details.

Declaration

```
TYPE bytearray = PACKED ARRAY [1..N] OF CHAR;
PROCEDURE HPEXTIN  (
          ANYVAR charstring : bytearray;
             VAR stringlen  : SHORTINT;
                 decplaces  : SHORTINT;
                 datatype   : SHORTINT;
                 scale      : SHORTINT;
                 delimiters : SHORTINT;
          ANYVAR result     : INTEGER;
             VAR error      : SHORTINT
               )
OPTION UNCHECKABLE_ANYVAR;
EXTERNAL;
```

Attributes

Parameters (Input):	charstring =	The byte array to be converted, passed by reference.
	stringlen =	Upon entry, the field width of the ASCII input string including all special characters. (See Comment 1 for more details.)
	decplaces =	The number of digits *d* to be interpreted as fraction digits (multiply the integer field by 10-d if the input string does *not* include a decimal point). See Comment 1 for more details. If a decimal point is included in the input string, this

parameter has no effect. If decplaces is less than zero, the procedure assumes decplaces is zero. If datatype is 0 or -1, this parameter is ignored.

datatype = The internal representation desired:

0 = SHORTINT
1 = REAL
-1 = INTEGER
-2 = LONGREAL

scale = The scale factor (see Comment 2 for more details). This factor is ignored if datatype=0 (SHORTINT) or datatype=-1 (INTEGER).

delimiters = Imbedded blanks, dollar signs, and commas in the input are treated differently, depending upon whether the value is even or odd.

Even: $ and/or commas and/or imbedded blanks are delimiters.

Odd : Imbedded blanks are treated as zeros; a $ and/or a comma to the left of every third digit to the left of the decimal point are allowed.

Parameters (Output): result= 32-bit pointer (address) to the first word of result storage according to the type specified.

error = Error indicator (see Comments 5 and 6 for more details).

2 = No integer or fraction value was detected (see Comment 6).

1 = An illegal character was detected (see Comment 5).

0 = Valid result; no error.

-1 = The number is larger than the largest representable value and an illegal character was detected.

-2 = The resulting number is greater than the largest representable value of datatype. See Chapter 1 for details on representable values.

-3 = The number is less

		than the smallest representable value and an illegal character was detected.
	-4 =	The resulting number is less than the smallest representable value of datatype. See Chapter 1 for details on representable values.
	W =	Upon exit, the number of string characters used to compute the result (see Comment 5).

Results: See "Parameters (Output)" above.

HP FORTRAN 77/XL: Use the $ALIAS directive to call EXTIN' ; use the system intrinsic statement to call HPEXTIN.

Errors: See "Parameters (Output)" above.

Comments

1. The external form of the input is a string of ASCII digits that can include integer, fraction, and exponent subfields:

```
     Integer           Fraction       Exponent
     field             field          field
        |                 |              |
        V                 V              V
     |<---------->| |<----------->|  |<---------->|
      +n...n   .    n...n          E+ee
                ^
                |
          Decimal point
```

   The exponent field input can be any of these forms:

```
     +e      +ee      Ee      Eee       De      Dee
     -e     -ee      E+e     E+ee      D+e      D+ee
     +eee   -eee     E-e     E-ee     D-e     D-ee
```

   where e is an exponent value digit.

2. The scale parameter has no effect if the input string includes an exponent field. Otherwise, a scale of *n* sets the result to the input string value * 10-n . For example,

STRING Array	scale	RESULT
4398.76	3	4.39876
543.21	-3	543210.

3. The type of the result is independent of the input string format. For example, the input 4398.76 can be converted to integer form. The conversion rules are stated in Table 3-1.

Table 3-1. Rules for Data Type Conversion

Data Type	Value of datatype	Result
SHORTINT	0	Truncates a fractional input.
REAL	1	Accepts a fractional input.
INTEGER	-1	Truncates a fractional input.
LONGREAL	-2	Accepts a fractional input.

4. Leading blanks in the input string are ignored; if the parameter delimiters is odd, trailing blanks are treated as zeros.

5. If error is set to an odd value, an illegal character was input; if error is odd and negative, an illegal character and illegal value was detected. The parameter intlvalue is computed from the input string characters that preceded the delimiting digit or illegal character. Parameter stringlen can be used as an index into charstring to locate that delimiter or illegal character. The following are examples of illegal character inputs:

```
+1.345A  (A is illegal)
 7543CUP  (C, U, and P are illegal)
```

6. If error is set to 2, no integer or fraction value was detected and a result cannot be computed. The following are examples of non-value inputs:

```
+.E5     (the exponent E5 has no base)
 -.A      (no base, no exponent)
```

7. When case is significant, the names for the EXTIN'/HPEXTIN procedure are in lower case.

NOTE Dollar signs ($) and commas for monetary or numeration form in the input are ignored, but they must be counted in parameter stringlen, and must be in the proper places (for example, $ must lead a number, and commas should appear every third digit).

INEXT'/HPINEXT

INEXT' and HPINEXT are two names for the same utility procedure. INEXT'/HPINEXT converts a number in storage (in one of four internal representations) to a byte array for an output string of ASCII digits. The internal representations are:

* SHORTINT
* REAL
* INTEGER
* LONGREAL

See the "Comments" section for more details.

Declaration

```
TYPE bytearray = PACKED ARRAY [1..N] OF CHAR;
PROCEDURE HPINEXT  (
          ANYVAR value      : INTEGER;
                 datatype   : SHORTINT;
                 fieldwidth : SHORTINT;
                 decplaces  : SHORTINT;
                 kind       : SHORTINT;
                 scale      : SHORTINT;
          ANYVAR charstring : bytearray;
             VAR error      : SHORTINT
                )
OPTION UNCHECKABLE_ANYVAR;
EXTERNAL;
```

Attributes

Parameters (Input):

value = 32-bit pointer (address) to the first word of the internal representation to be converted.

datatype = The type of internal representation:

0	=	SHORTINT
1	=	REAL
-1	=	INTEGER
-2	=	LONGREAL

fieldwidth = Field width *w* of the ASCII string, including all special characters. See the kind parameter description for more details.

Set fieldwidth to at least decplaces+6 to allow for special characters when kind equals 3 (G*w.d* format) or 2 (D*w.d* format) or 1 (E*w.d* format). If a positive scale factor is also used, set fieldwidth to at least decplaces+7.

decplaces = The number of fractional digits *d* in the ASCII string. If decplaces is equal to zero, no fractional digits are included in the output, even though a decimal point is included. If fieldwidth <= zero or decplaces < zero, an error is implied and error is set to -1.

kind = The kind of conversion desired. See Table 3-2 for a list of possible values of kind.

Table 3-2. Values of kind and Associated Formats

Value of kind	Format	Example
3	G*w.d*	See "Comments" for more details.
2	D*w.d*	0.12345D+04
1	E*w.d*	0.12345E+04
0	I*w*	1234
-1	N*w.d*	1,234.5
-2	M*w.d*	$1,234.5
-3	F*w.d*	1234.5

scale = The scale factor (see "Comments" for more details).

Parameters (Output):

charstring = Pointer to the first byte array for the ASCII string output. The result occupies the first fieldwidth characters (bytes) in this array.

error = Is -1 if fieldwidth is too small for the result in the specified kind. If decplaces < zero, or if fieldwidth <= zero, the byte array is filled with asterisks (*). The error parameter is zero if the result is valid.

Results: See "Parameters (Output)" above for details.

HP FORTRAN 77/XL: Use the $ALIAS directive to call INEXT' ; use the system intrinsic statement to call HPINEXT.

Errors: See "Parameters (Output)" above for details.

Comments

1. The result charstring is an array of ASCII digits; charstring can also include the sign character (-), a decimal point, and an exponent field for kind = 1 (for the E*w.d* format) or kind = 2 (for the D*w.d* format). Generally, the exponent field includes the letter E or D followed by a signed two-digit integer. If the absolute value of the exponent is greater than 99, the exponent field appears as a signed three-digit integer (without a preceding E). Alternatively, charstring can include a sign character (-), a dollar sign for kind = -2 (for the M*w.d* format) and/or commas for kind = -2 or -3 (for the M*w.d* or N*w.d* formats). (*w* equals the parameter fieldwidth and *d* equals the parameter decplaces.)

2. To use kind = 3 (G*w.d* format), set decplaces to the number of significant digits and set fieldwidth to decplaces+6 to allow for special characters. Then, kind = 3 is used as kind = -3 or 1 (F*w.d* or E*w.d* format), according to the absolute value of the internal representation of value intlvalue, as shown in Table 3-3.

Table 3-3. Gw.d Formatting Rules

Value	Format Used
intlvalue < 0.1	E*w.d*
0.1 <= intlvalue < 1	F(*w* -4).*d* plus 4 spaces
1 < intlvalue < 101	F(*w* -4).(*d* -1) plus 4 spaces
101 <= intlvalue < 102	F(*w* -4).(*d* -2) plus 4 spaces
102 <= intlvalue < 103	F(*w* -4).(*d* -3) plus 4 spaces
.	.
.	.
10(d-1) <= intlvalue < 10d	F(*w* -4).0 plus 4 spaces
10d <= intlvalue	Ew.d

In general, if the number of integer digits in intlvalue is greater than decplaces or equal to zero, kind = 1 is used (the E*w.d* format).

Examples

The following examples show how the G*w.d* format is used.

If you specify the format G12.6 and the value of intlvalue is 1234.5, the equivalent F*w.d* format is calculated as follows: because 103 < intlvalue < 104 , the format used is F(*w*- 4).(*d*- 4) plus four spaces. Therefore, in this example, the equivalent F*w.d* format is F8.2 plus four spaces; the value stored is Å1234.50ÅÅÅÅ.

If you specify the format G13.7 and the value of intlvalue is 123456.7, the equivalent F*w.d* format is F(*w-4*).(*d-6*). Therefore, in this example, the format used is F9.1 plus four spaces; the value stored is Å123456.7ÅÅÅÅ.

Finally, if you specify the format G9.2 and the value of intlvalue is 123.4, the equivalent E*w.d* format is E9.2; the value stored is ÅÅ.12E+03.

3. The scale parameter does not affect kind = 0. When kind = 1 or 2, the result string uses these factors:

   * The internal representation value fraction is multiplied by 10s (where *s* is scale).
   * The internal representation value exponent is reduced by scale.
   * When scale is <= zero, the charstring fraction has -scale leading zeros, followed by decplaces + scale significant digits.
   * When scale is < zero, charstring has scale significant digits to the left of the decimal point and (decplaces - scale) + 1 significant digits to the right of the decimal point.
   * The least significant digit in charstring is rounded.

   For example, if intlvalue = 1234.5, kind = 1, fieldwidth = 11, and

decplaces = 3, the following is true:

```
If scale = 0,    charstring = ÅÅÅ.123E+04
If scale = -2,   charstring = ÅÅÅ.001E+06
If scale = 2,    charstring = ÅÅ12.35E+02
```

When kind = -3, -2, or -1, the result charstring is the internal representation intlvalue multiplied by 10s (where *s* is scale); the result is then converted.

For example, if intlvalue = 1234.5, kind = -3, fieldwidth = 11, and decplaces = 3, the following is true:

```
If scale = 0,  charstring = ÅÅÅ1234.500
If scale = -2, charstring = ÅÅÅÅÅ12.345
If scale = 2,  charstring = Å123450.000
```

When kind = -3, the following is true:

* If kind = 3 (G*w.d*) is used as kind = -3(F*w.d*), scale has no effect.

* If kind = 3 is used as kind = 1 (E*w.d*),scale affects charstring as described for kind = 2 or 1.

4. When case significant, the names for the INEXT'/HPINEXT are in lower case.

Chapter 4 Packed-Decimal Procedures

This chapter describes the packed-decimal procedures that perform the following basic operations:

* Arithmetic operations
* Comparisons
* Left and right shifts
* Conversions to and from binary
* Conversions to and from external decimal

These procedures facilitate migration to MPE XL of applications that perform packed-decimal operations using ASSEMBLE statements in the SPL/V programming language. Packed-decimal operations on SPL/V are typically performed by pushing several parameters onto the stack and then executing, via an ASSEMBLE statement, the appropriate decimal instruction.

Programs written in SPL/V can be run on 900 Series HP 3000 computer systems in Compatibility Mode. However, it is often desirable to convert them to languages for which Native Mode programs can be generated.

The procedures described in this chapter provide packed-decimal operations for programs executing in Native Mode. There is a procedure corresponding to each of the 11 packed-decimal instructions in the HP 3000 Series V instruction set. In addition, there is a procedure corresponding to each of the two packed-decimal division routines provided in the MPE V/E Compiler Library.

The packed-decimal procedures are also suitable for use in new programs being developed in Native Mode. Because the sizes of the operands are passed as parameters, these procedures can be used in applications where the field sizes are not known at compile-time, for example, general-purpose database applications and report writers.

NOTE These procedures execute entirely in Native Mode and take advantage of the architecture of the 900 Series HP 3000 computers as much as possible. However, because of their generality and the fact that they are external procedures, they are not as efficient as the code that can be generated by a compiler.

Therefore, in applications where the speed of packed-decimal operations is a primary concern, you may wish to consider a language that directly supports the packed-decimal data type, such as HP COBOL II/XL.

Packed-decimal procedures can be declared as intrinsics and can be called from high level languages, such as HP C/XL, HP Pascal/XL and HP FORTRAN 77/XL.

Most of the procedures use the packed-decimal format; two procedures (HPPACCVAD and HPPACCVDA) use external-decimal number format; and two procedures (HPPACCVBD and HPPACCVDB) use binary numbers.

[

Packed-Decimal Numbers

The format of a packed-decimal number is shown in Figure 4-1.

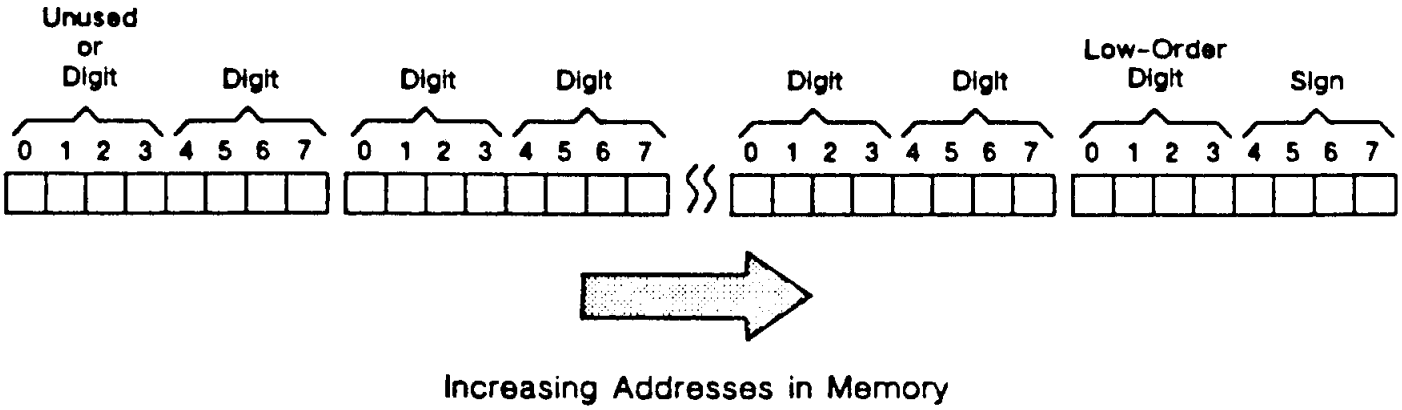


Figure 4-1. Packed-Decimal Format

The left-most byte in the illustration contains the high-order digit. If this digit is in bit postions 4 through 7, positions 0 through 3 of the same byte are ignored. (A digit count specifies the number of digits to be recognized.) A packed-decimal number has the following characteristics:

* Each decimal digit is represented in binary coded decimal (BCD) form by four bits.

* The sign is represented by four bits.

* In storage, the four sign bits may be in the following bit positions of a 32-bit word: (4-7), (12-15), (20-23), or (28-31). Expressed in different terms, the sign is always in positions (4-7) of an 8-bit byte; the byte is byte 0, byte 1, byte 2, or byte 3 of a 32-bit word.

* Bytes to the left of the byte containing the high-order digit are not part of the number field and may have any contents, even though they may be part of the same 32-bit word. Likewise, bytes to the right of the byte containing the sign bits are not part of the field and may have any contents, even though they may be part of the same 32-bit word. Bytes that are outside of the field will not be accessed or modified.

* Succeeding 4-bit groups to the left of the sign (Figure 4-1) can contain successively higher-order digits.

* There are no unused bits between the sign and the high-order digit.

* The length may be up to 28 digits.

* The storage address is the address of the byte that contains the high-order digit.

* When a packed-decimal number is source data for a decimal arithmetic procedure, sign bits 1101 are recognized as minus. All other bit combinations are recognized as plus, except that HPPACCVDA recognizes 1111 as designating an unsigned number.

* When a packed-decimal number is the result of a decimal arithmetic procedure, sign bits 1100 indicate plus and 1101 indicate minus. There are no unsigned result operands except for HPPACCVAD, which furnishes 1111 to indicate an unsigned number.

* A leading nonsignificant packed-decimal digit is not modified by any procedure other than HPPACCVAD, which inserts a zero.

External-Decimal Numbers

Figure 4-2 shows the format of external-decimal numbers.

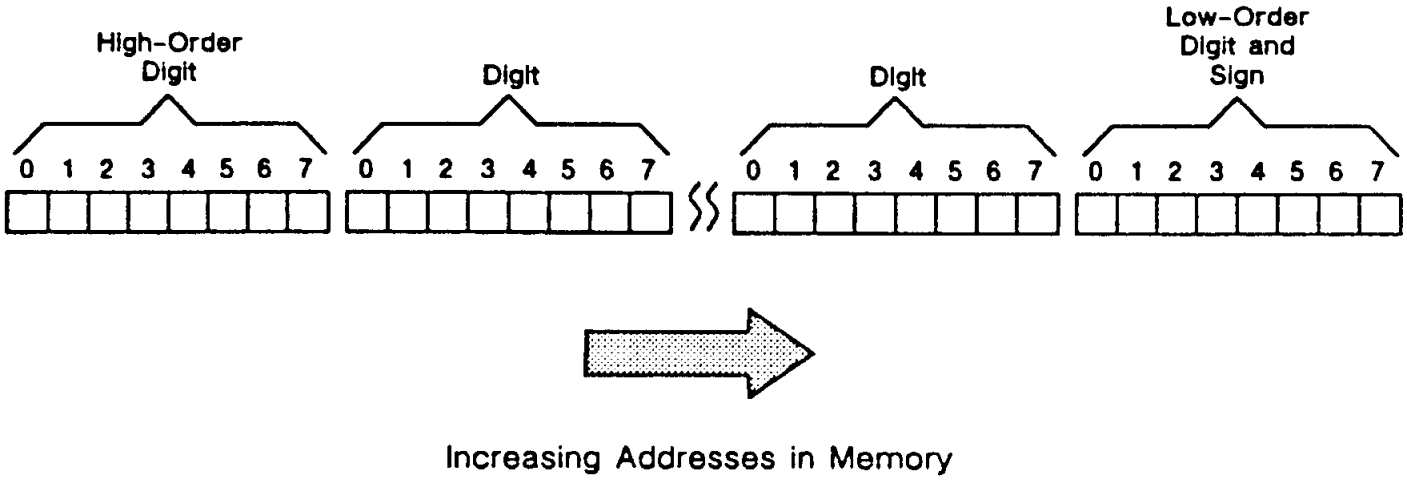


Figure 4-2. External-decimal Format

External-decimal numbers have the following characteristics:

* Each digit is expressed by eight bits.
* An indication of the sign of the number is included in the representation of the low-order digit.
* In storage, the low-order digit may be any byte of a 32-bit word.
* Bytes to the left of the byte containing the high-order digit are not part of the number field and may have any contents, even though they may be part of the same 32-bit word. Likewise, bytes to the right of the byte containing the sign bits are not part of the field and may have any contents, even though they may be part of the same 32-bit word. Bytes that are outside of the field will not be accessed or modified.
* Subsequent bytes to the left of the low-order digit (Figure 4-2) contain successively higher-order digits.
* There are no unused bytes between the low-order digit and the high-order digit.
* The length may be up to 28 digits.
* The storage address is the address of the byte that contains the high-order digit.

Table 4-1 shows the low-order digit for positive, negative, and unsigned numbers. The dollar sign ($) prefix indicates a hexadecimal number.

Table 4-1. Low-Order Digits of External-Decimal Numbers

Low-Order Digit, Decimal Number	Low-Order Digit, External-Decimal Number		
	Unsigned	Positive	Negative
0	$30	$7b {	$7d }
1	$31	$41 A	$4a J
2	$32	$42 B	$4b K
3	$33	$43 C	$4c L
4	$34	$44 D	$4d M
5	$35	$45 E	$4e N
6	$36	$46 F	$4f O
7	$37	$47 G	$50 P
8	$38	$48 H	$51 Q
9	$39	$49 I	$52 R

In the table, the letters A through R, and the braces, are the ASCII equivalents of the hexadecimal numbers shown.

Digits other than the low-order digit conform with the "Unsigned" column of Table 4-1.

Binary Numbers

The binary numbers used as parameters to HPPACCVBD and HPPACCVDB have the following characteristics.

* The number is stored in twos-complement format.
* The length is a multiple of 16 bits.
* The length may be up to 96 bits.
* In storage, the high-order bit must occupy bit 0 or bit 15 of a 32-bit word. In other words, the number must be aligned on an even-byte boundary.
* The storage address is the address of the byte that contains the high-order bit.

NOTE For compatibility with the HP 3000 Series V instruction set, the length of a binary number is given in terms of 16-bit elements, sometimes referred to as "words." To avoid confusion with the 32-bit words of the Series 900 computer systems, the term "16-bit word" is used in this chapter when referring to the 16-bit elements of a binary number.

Trap Handling

When an error is detected by a packed-decimal procedure, the procedure stores information about the operation being performed, the parameters, and the nature of the error, and then causes a trap by executing a reserved instruction. Depending on the state of the MPE XL trap subsystem, the trap may be ignored, the process may be aborted, a user-written trap handler may be invoked, or a Pascal/XL recover block may be executed.

If a trap handler is invoked and returns control to the packed-decimal procedure, the packed-decimal procedure will immediately return to its caller.

Under MPE XL, the ARITRAP intrinsic is used to enable or disable arithmetic traps collectively. HPENBLTRAP is used to enable or disable arithmetic traps selectively and XARITRAP is used to arm or disarm a user-written trap handler. HPENBLTRAP and XARITRAP use a mask to specify individual trap conditions. Bit (7:1) of the mask designates a "3000 Mode Packed-decimal error."

The system trap handler reacts as shown in the following table:

condition	**action**
Trap disabled	Ignore trap, continue execution of process.
Trap enabled, but no trap handler armed	TRY statement active: do an ESCAPE with *errnum* =7,*subsys* =200. TRY statement not active: display error message and abort process.
Trap enabled and trap handler armed	Invoke user-written trap handler.

When control is transferred to a user-written trap handler that has been armed with XARITRAP, the trap handler is passed a pointer to a record containing useful information. For the format and contents of the information record, as well as a complete description of trap handling under MPE XL, refer to the *Trap Handling Programmer's Guide.*

Parameters

The following section lists and describes the parameters used by the packed-decimal library procedures.

operand2 Typically, the operand that receives the result of the operation. The exception is HPPACCMPD, which does not change either operand.

op2digs The length of *operand2* in digits.

operand1 The other operand. As shown in the calling sequences, *operand2* is given before *operand1*, except in the case of HPPACCMPD.

op1digs The length of *operand1* in digits.

source The sending field in a conversion to or from external-decimal or binary.

sourcedigs The length of the source field in digits.

sourcewords The length of the source field in 16-bit "words."

target The receiving field in a conversion.

targetdigs The length of the target field in digits.

comparison_code an integer to which the procedure will always return a value of 0, 1, or 2.

NOTE *Comparison_code* corresponds to the condition code of the status register on HP 3000 Series V systems.

The value returned is not meaningful when an operand length of zero is specified, or when an error is detected. If a result is truncated because the target digit count is too small, *comparison_code* is set in accordance with the truncated result.

All the procedures except HPPACCMPD set *comparison_code* as follows:.

comparison_code = 0 if result > 0
= 1 if result < 0
= 2 if result = 0

HPPACCMPD sets *comparison_code* as follows:
comparison_code = 0 if operand1 > operand2
= 1 if operand1 < operand2
= 2 if operand1 = operand2

pacstatus An integer to which the procedure will always return an indication of the success or failure of the operation. When no error is detected, *pacstatus* is set to 0. When an error is detected, *pacstatus* is set to a code indicating the type of error that occurred. By default, detection of an error will result in the process being aborted with a 3000-mode packed-decimal trap. Therefore, a program will normally examine this parameter only in cases where default trap-handling is not in effect.

The state of *pacstatus* (zero or nonzero) corresponds to the overflow bit of the status register on HP 3000 Series V systems.

The following values are possible for this parameter:

0 No error
1 Decimal overflow
2 Invalid ASCII digit
3 Invalid packed-decimal digit
4 Invalid source word count
5 Invalid decimal operand length
6 Decimal divide-by-zero

carry An integer to which the procedure will always return a value of 0 or 1. This value corresponds to the carry bit of the status register on HP 3000 Series/V systems.

sign_control For HPPACCVDA, a parameter that controls the sign of the result when the source field is signed.

shift_amt For the shift operations, the number of digits to be shifted left or right. For HPPACNSLD, this parameter is passed by reference; for the other shift operations, it

is passed by value.

NOTE The packed-decimal procedures do not alter the condition code associated with the process.

Procedure Declarations

The procedure declarations in this chapter assume the following Pascal directives and declarations.

```
$STANDARD_LEVEL 'EXT_MODCAL'$
$UPPERCASE ON$
$TYPE
   ascii_type    = $ALIGNMENT 1$ RECORD END;
   binary_type   = $ALIGNMENT 2$ RECORD END;
   decimal_type  = $ALIGNMENT 1$ RECORD END;
```

In the listings of possible traps, the numbers in parentheses are the trap subcodes supplied to the user when a trap handler has been armed, as explained under "Trap Handling" (above). If traps have been disabled, these are the codes returned to the caller through the *pacstatus* parameter.

HPPACADDD

The HPPACADDD procedure adds two operands and stores the result.

Declaration

```
PROCEDURE HPPACADDD (
          ANYVAR  operand2        :decimal_type;
                  op2digs         :INTEGER;
          ANYVAR  operand1        :decimal_type;
                  op1digs         :INTEGER;
          VAR     comparison_code :INTEGER;
          VAR     pacstatus       :INTEGER)
     OPTION
          DEFAULT_PARMS (
                  comparison_code :=NIL,
                  pacstatus       :=NIL)
          UNCHECKABLE_ANYVAR;
     EXTERNAL;
```

Comments

Operand1 and *operand2* are added and the sum is stored into the *operand2* field. A decimal overflow occurs if all significant digits of the result do not fit in the *operand2* field. This results in a trap, and the left-truncated result is stored in the *operand2* field.

Possible Traps

Decimal overflow (1)
Invalid packed-decimal digit (3)
Invalid decimal operand length (5)

HPPACCMPD

HPPACCMPD compares two decimal operands and sets a condition code.

```
PROCEDURE HPPACCMPD (
          ANYVAR  operand1        :decimal_type;
                  op1digs         :INTEGER;
          ANYVAR  operand2        :decimal_type;
```

```
                    op2digs         :INTEGER;
               VAR  comparison_code :INTEGER;
               VAR  pacstatus       :INTEGER)
          OPTION
               DEFAULT_PARMS (
                      comparison_code  :=NIL,
                      pacstatus        :=NIL)
               UNCHECKABLE_ANYVAR;
          EXTERNAL;
```

Comments

Operand1 is compared to *operand2* and *comparison_code* is set. The operands remain unchanged at their original addresses.

Possible Traps

Invalid packed-decimal digit (3)
Invalid decimal operand length (5)

HPPACCVAD

HPPACCVAD performs an ASCII-to-decimal conversion.

Declaration

```
     PROCEDURE HPPACCVAD (
               ANYVAR target          : decimal_type;
                      targetdigs      : INTEGER;
               ANYVAR source          : ascii_type;
                      sourcedigs      : INTEGER;
               VAR    comparison_code : INTEGER;
               VAR    pacstatus       : INTEGER)
        OPTION
               DEFAULT_PARMS (
                      comparison_code := NIL,
                      pacstatus       := NIL)
               UNCHECKABLE_ANYVAR;
        EXTERNAL;
```

Comments

* Source digits in external-decimal are converted to packed-decimal digits. Source digits, except for the rightmost in the field, must be leading blanks or unsigned digits. The rightmost digit indicates the sign.

* Leading blanks are converted to packed-decimal zeros.

* Blanks between digits are illegal.

* An all-blank field converts to an unsigned zero target field.

* An unsigned external-decimal operand produces an unsigned packed-decimal result.

* If the number of target digits is less than the number of source digits, the source is converted until the target is filled, producing a left-truncated result. In this case, the remaining source digits are not examined for validity.

* If the source digit count is less than the target digit count, left zero fill is placed in the target field.

* If either the source or target digit count is zero, no conversion is performed.

Possible Traps

Invalid ASCII digit (2)
Invalid decimal operand length (5)

HPPACCVBD

HPPACCVBD converts a binary number to packed-decimal.

Declaration

```
      PROCEDURE HPPACCVBD (
                ANYVAR  target           :decimal_type;
                        targetdigs       :INTEGER;
                ANYVAR  source           :binary_type;
                        sourcewords      :INTEGER;
                VAR     comparison_code  :INTEGER;
                VAR     pacstatus        :INTEGER)
           OPTION
                DEFAULT_PARMS (
                        comparison_code  :=NIL,
                        pacstatus        :=NIL)
                UNCHECKABLE_ANYVAR;
           EXTERNAL;
```

Comments

* The source field is converted to packed-decimal and stored in the target field. The source is considered to be a two s-complement binary number; its length is specified in sixteen-bit "words."

* If the "word" count is not in the range 0<=n<=6, a trap occurs.

* If the target digit count is not in the range 0<=n<=28, a trap occurs.

* After the binary source is converted, leading zeros are stored until the target field is filled.

* If the number of digits generated is greater than the target digit count, the partial result is stored and a decimal overflow trap occurs.

* If either the target digit count or the source word count is zero, no conversion is performed.

Possible Traps

Decimal overflow (1)
Invalid source word count (4)
Invalid decimal operand length (5)

HPPACCVDA

HPPACCVDA converts a packed-decimal number to ASCII.

Declaration

```
      PROCEDURE HPPACCVDA (
                ANYVAR  target           :ascii_type;
                        targetdigs       :INTEGER;
                ANYVAR  source           :decimal_type;
                        sign_control     :INTEGER;
                VAR     comparison_code  :INTEGER;
                VAR     pacstatus        :INTEGER)
           OPTION
                DEFAULT_PARMS (
                        sign_control     :=0,
```

```
                    comparison_code :=NIL,
                    pacstatus       :=NIL)
            UNCHECKABLE_ANYVAR;
      EXTERNAL;
```

Comments

* The source packed-decimal digits are converted to fill the target field. An unsigned source operand produces an unsigned external-decimal result. When the source is signed, the sign of the result is determined by the two low-order bits of *sign_control* as described below.

  sign_control = 0 Target sign is the same as source.

  sign_control = 1 Target is negative if source is negative. Otherwise, it is unsigned.

  sign_control = 2 Target is unsigned.

  sign_control = 3 Target is unsigned.

* *Comparison_code* is set in accordance with the stored result. An unsigned result is considered nonnegative, so only 0 or 2 can be returned in *comparison_code* if *sign_control* is 2 or 3.

* If the target count is zero, no conversion is performed.

Possible Traps

Invalid packed-decimal digit (3)
Invalid decimal operand length (5)

HPPACCVDB

HPPACCVDB converts a packed-decimal number to binary.

Declaration

```
   PROCEDURE HPPACCVDB (
             ANYVAR  target          :binary_type;
             ANYVAR  source          :decimal_type;
                     sourcedigs      :INTEGER;
             VAR     comparison_code :INTEGER;
             VAR     pacstatus       :INTEGER)
      OPTION
             DEFAULT_PARMS (
                     comparison_code :=NIL,
                     pacstatus       :=NIL)
             UNCHECKABLE_ANYVAR;
      EXTERNAL;
```

Comments

The number of decimal digits specified in the source digit count are converted to a twos-complement binary number and stored in the target field. The length of the target in sixteen-bit "words" is based on the length of the source, as follows.

Source Digit Count	Target "Words"
1 to 4	1
5 to 9	2
10 to 18	4
19 to 28	6

If the source digit count is zero, no conversion is performed.

Possible Traps

Invalid packed-decimal digit (3)
Invalid decimal operand length (5)

HPPACCVDB

HPPACCVDB converts a packed-decimal number to binary.

Declaration

```
      PROCEDURE HPPACCVDB (
                ANYVAR  target          :binary_type;
                ANYVAR  source          :decimal_type;
                        sourcedigs      :INTEGER;
                VAR     comparison_code :INTEGER;
                VAR     pacstatus       :INTEGER)
           OPTION
                DEFAULT_PARMS (
                        comparison_code :=NIL,
                        pacstatus       :=NIL)
                UNCHECKABLE_ANYVAR;
           EXTERNAL;
```

Comments

The number of decimal digits specified in the source digit count are converted to a twos-complement binary number and stored in the target field. The length of the target in sixteen-bit "words" is based on the length of the source, as follows.

Source Digit Count	Target "Words"
1 to 4	1
5 to 9	2
10 to 18	4
19 to 28	6

If the source digit count is zero, no conversion is performed.

Possible Traps

Invalid packed-decimal digit (3)
Invalid decimal operand length (5)

Special Conventions of HPPACLONGDIVD

HPPACLONGDIVD includes the following conventions for compatibility with the MPE V E procedure, LONGDIVD.

* Operand lengths are not validated. Results are undefined if *operand1* is longer than 28 digits, or if *operand2* is longer than 36 digits.

* When *operand2* is longer than 28 digits, only the first 28 digits (and the sign) are passed by the caller. The remaining digits are assumed to be zeros, and no space needs to be allocated for them by the caller. In effect, HPPACLONGDIVD will supply the necessary zeros by multiplying its local copy of *operand2* by the appropriate power of ten.

* When using HPPACLONGDIVD, the caller is responsible for ensuring that the quotient does not exceed 28 digits. If it does, results are undefined.

Possible Traps

Invalid packed-decimal digit (3)
Invalid decimal operand length (5)left arrow*For HPPACDIVD only*
Decimal divide-by-zero (6)

Special Conventions of HPPACLONGDIVD

HPPACLONGDIVD includes the following conventions for compatibility with the MPE V E procedure, LONGDIVD.

* Operand lengths are not validated. Results are undefined if *operand1* is longer than 28 digits, or if *operand2* is longer than 36 digits.

* When *operand2* is longer than 28 digits, only the first 28 digits (and the sign) are passed by the caller. The remaining digits are assumed to be zeros, and no space needs to be allocated for them by the caller. In effect, HPPACLONGDIVD will supply the necessary zeros by multiplying its local copy of *operand2* by the appropriate power of ten.

* When using HPPACLONGDIVD, the caller is responsible for ensuring that

the quotient does not exceed 28 digits. If it does, results are undefined.

Possible Traps

Invalid packed-decimal digit (3)
Invalid decimal operand length (5)left arrow*For HPPACDIVD only*
Decimal divide-by-zero (6)

HPPACMPYD

HPPACMPYD performs multiplication of two decimal numbers.

Declaration

```
      PROCEDURE HPPACMPYD (
                ANYVAR  operand2         :decimal_type;
                        op2digs          :INTEGER
                ANYVAR  operand1         :decimal_type
                        op1digs          :INTEGER;
                        shift_amt        :INTEGER;
                VAR     comparison_code  :INTEGER;
                VAR     pacstatus        :INTEGER)
            OPTION
                DEFAULT_PARMS (
                        comparison_code  :=NIL,
                        pacstatus        :=NIL)
                UNCHECKABLE_ANYVAR;
            EXTERNAL;
```

Comments

Operand1 and *operand2* are multiplied and the product is stored into the *operand2* field. If all significant digits of the result do not fit into the *operand2* field, a decimal overflow occurs. The result stored in this case will be left-truncated unless the actual result is greater than 28 digits. If the actual result is over 28 digits, nothing will be stored.

Possible Traps

Decimal overflow (1)
Invalid packed-decimal digit (3)
Invalid decimal operand length (5)

HPPACNSLD

HPPACNSLD performs a decimal normalizing left shift.

Declaration

```
      PROCEDURE HPPACNSLD (
                ANYVAR  operand2        :decimal_type;
                        op2digs         :INTEGER;
                ANYVAR  operand1        :decimal_type;
                        op1digs         :INTEGER;
                VAR     shift_amt       :INTEGER;
                VAR     comparison_code :INTEGER;
                VAR     pacstatus       :INTEGER;
                VAR     carry           :INTEGER)
            OPTION
                DEFAULT_PARMS (
                        comparison_code :=NIL,
                        pacstatus       :=NIL,
                        carry           :=NIL)
                UNCHECKABLE_ANYVAR;
            EXTERNAL;
```

Comments

* *Operand1* is moved to the *operand2* field with its digits offset by the shift amount in the low-order five bits of *shift_amt*.

* Leading or trailing digits in the result field that are not supplied by the source operand will be zeros.

* If the shift amount is large enough that significant digits of *operand1* would be shifted out of the *operand2* field, the effective shift amount is reduced so *operand1* is left-justified in the *operand2* field. In addition, a number equal to the difference between the specified and actual shifts amounts is left in *shift_amt*, and *carry* is set to one.

* If the length of the *operand2* field is such that significant digits would be lost even with a shift amount of zero, a decimal overflow trap occurs, and no data movement occurs.

Possible Traps

Decimal overflow (1)
Invalid packed-decimal digit (3)
Invalid decimal operand length (5)

HPPACSLD

HPPACSLD performs a decimal left shift.

Declaration

```
      PROCEDURE HPPACSLD (
                ANYVAR  operand2        :decimal_type;
                        op2digs         :INTEGER;
                ANYVAR  operand1        :decimal_type;
                        op1digs         :INTEGER;
                        shift_amt       :INTEGER;
                VAR     comparison_code :INTEGER;
                VAR     pacstatus       :INTEGER;
                VAR     carry           :INTEGER)
           OPTION
                DEFAULT_PARMS (
                        comparison_code :=NIL,
                        pacstatus       :=NIL,
                        carry           :=NIL)
                UNCHECKABLE_ANYVAR;
           EXTERNAL;
```

Comments

* *Operand1* is moved to the *operand2* field with its digits offset to the left of its sign by the shift amount in the low-order five bits of *shift_amt*.

* Leading or trailing digits in the result field that are not supplied by the source operand will be zeros.

* Digits shifted out of the *operand2* field are lost, and *carry* is set to one to indicate that significant digits were lost.

Possible Traps

Invalid packed-decimal digit (3)
Invalid decimal operand length (5)

HPPACSRD

HPPACSRD performs a decimal right shift.

Declaration

```
      PROCEDURE HPPACSRD (
                ANYVAR  operand2        :decimal_type;
                        op2digs         :INTEGER;
                ANYVAR  operand1        :decimal_type;
                        op1digs         :INTEGER;
                        shift_amt       :INTEGER;
                VAR     comparison_code :INTEGER;
                VAR     pacstatus       :INTEGER;
           OPTION
                DEFAULT_PARMS (
                        comparison_code :=NIL,
                        pacstatus       :=NIL)
                UNCHECKABLE_ANYVAR;
           EXTERNAL;
```

Comments

* *Operand1* is moved to the *operand2* field, with its digits offset to the right relative to its sign, by the shift amount in the low-order five bits of *shift_amt*.

* Digits shifted into the sign are lost.

* Zeros are shifted in from the left and high order zeros are inserted to fill the *operand2* field, if necessary.

Possible Traps

Invalid packed-decimal digit (3)
Invalid decimal operand length (5)

HPPACSUBD

HPPACSUBD performs decimal subtraction.

Declaration

```
     PROCEDURE HPPACSUBD (
                ANYVAR  operand2        :decimal_type;
                        op2digs         :INTEGER;
                ANYVAR  operand1        :decimal_type;
                        op1digs         :INTEGER;
                VAR     comparison_code :INTEGER;
                VAR     pacstatus       :INTEGER;
           OPTION
                DEFAULT_PARMS (
                        comparison_code :=NIL,
                        pacstatus       :=NIL)
                UNCHECKABLE_ANYVAR;
           EXTERNAL;
```

Comments

* *Operand1* is subtracted from *operand2* and the difference is stored into the *operand2* field.

* A decimal overflow occurs if the result digits do not fit in the *operand2* field. The left-truncated result is stored in *operand2* and a trap occurs.

Possible Traps

Decimal overflow (1)
Invalid packed-decimal digit (3)
Invalid decimal operand length (5)

Sample Program

This section contains a sample program to demonstrate how to call the packed-decimal library routines. The program takes three large numbers (U.S. federal government annual appropriations for the years 1977 through 1979), converts them from ASCII to packed-decimal, adds them, converts the sum to ASCII, and prints the answer.

There are four versions of the program. The first example is written in HP C/XL, the second in HP Pascal/XL, the third in HP FORTRAN 77/XL, and the fourth in SPL/V, using ASSEMBLE statements. The SPL/V example is included to show how the procedure calls would have been done on an MPE V based HP 3000. This may be helpful to users converting SPL code to C, Pascal, or FORTRAN.

Program Output

All four versions of the program produce the following output:

```
1977   046655980996406
1978   050778229148999
1979   056396083378825
Total  153830293524230
```

HP C/XL Example

```
/*
 *    This program calculates total appropriations by the
 *    United States federal government for the years 1977-1979
 */
#pragma intrinsic HPPACADDD
#pragma intrinsic HPPACCVAD
#pragma intrinsic HPPACCVBD
#pragma intrinsic HPPACCVDA

typedef struct {char c [15]} ascii_num_15;   /* 15-digit numbers in ASCII   */

typedef struct {char c [8]} packed_num_15;   /* 8-character field holds     */
                                             /*   15 digits and sign        */

static ascii_num_15 yearly [3] =       /* Annual appropriations, in cents   */
        {"046655980996406",
         "050778229148999",
         "056396083378825"};

main ()
{
   short int     zero=0;
   int           year;
   packed_num_15 packed_sum, packed_temp;
   ascii_num_15  ascii_sum;

   HPPACCVBD (&packed_sum, 15, &zero, 1);       /* Initialize sum           */

   for (year=1977; year<=1979; year++)
      {
      printf ("%d   %.15s\n",                   /* Echo annual data         */
              year, &yearly [year-1977]);
      HPPACCVAD (&packed_temp, 15,              /* Convert ASCII to decimal */
                 &yearly [year-1977], 15);
      HPPACADDD (&packed_sum, 15,               /* Add to sum               */
                 &packed_temp, 15);
      }

   HPPACCVDA (&ascii_sum, 15,                   /* Convert sum to ASCII,    */
              &packed_sum, 1);                  /*   suppressing plus sign  */
   printf ("Total  %.15s\n", &ascii_sum);
}
```

HP Pascal/XL Example

```
{
      This program calculates total appropriations by the
      United States federal government for the years 1977-1979
}

$notes off$
$standard_level 'ext_modcal'$
program appropriations (output);

type
   ascii_num_15  =                      { 15-digit numbers in ASCII          }
      packed array [1..15] of char;
   nibble        = 0..15;               { Half a byte, holds a digit or sign }
   packed_num_15 =                      { Holds 15 digits and sign           }
      crunched record
         digits : crunched array [1..15]
                     of nibble;
         sign   : nibble;
      end;
   year_type     = 1977..1979;
   yearly_table  = array [year_type]
                       of ascii_num_15;

const
   yearly = yearly_table                 { Annual appropriations, in cents   }
               ['046655980996406',
                '050778229148999',
                '056396083378825'];

var
   zero                   : shortint;
   year                   : year_type;
   packed_sum, packed_temp : packed_num_15;
   ascii_sum, ascii_temp   : ascii_num_15;

procedure HPPACADDD; intrinsic;
procedure HPPACCVAD; intrinsic;
procedure HPPACCVBD; intrinsic;
procedure HPPACCVDA; intrinsic;

begin

   zero := 0;
   HPPACCVBD (packed_sum, 15, zero, 1);            { Initialize sum         }

   for year := 1977 to 1979 do
      begin
      writeln (year:1, '   ', yearly [year]);      { Echo annual data       }
      ascii_temp := yearly [year];                 { Can't pass const as var }
      HPPACCVAD (packed_temp, 15, ascii_temp, 15); { Convert ASCII to decimal}
      HPPACADDD (packed_sum, 15, packed_temp, 15); { Add to sum             }
      end;

   HPPACCVDA (ascii_sum, 15,                       { Convert sum to ASCI,   }
              packed_sum, 1);                      {   suppressing plus     }
   writeln ('Total  ', ascii_sum);

end.
```

HP FORTRAN 77/XL Example

```
C
C     This program calculates total appropriations by the
C     United States federal government for the years 1977-1979
C
$STANDARD_LEVEL SYSTEM
```

```
      SYSTEM INTRINSIC HPPACADDD, HPPACCVAD, HPPACCVBD, HPPACCVDA

      INTEGER*2 ZERO
      INTEGER YEAR

      CHARACTER*15 YEARLY (1977:1979)   ! 15-digit numbers in ASCII
      CHARACTER*15 ASCII_SUM

      CHARACTER*8 PACKED_SUM            ! 8-character field holds
      CHARACTER*8 PACKED_TEMP           !   15 digits and sign

      DATA YEARLY/                      ! Annual appropriations, in cents
     1            '046655980996406',
     2            '050778229148999',
     3            '056396083378825'/
      DATA ZERO/0/

      CALL HPPACCVBD (PACKED_SUM, 15,           ! Initialize sum
     >                ZERO, 1)

      DO YEAR = 1977, 1979
         PRINT *, YEAR, '   ', YEARLY (YEAR)    ! Echo annual data
         CALL HPPACCVAD (PACKED_TEMP, 15,       ! Convert ASCII to decimal
     >                   YEARLY (YEAR), 15)
         CALL HPPACADDD (PACKED_SUM, 15,        ! Add to sum
     >                   PACKED_TEMP, 15)
      END DO

      CALL HPPACCVDA (ASCII_SUM, 15,            ! Convert sum to ASCII,
     >                PACKED_SUM, 1)            !   suppressing plus sign
      PRINT *, 'Total  ', ASCII_SUM

      STOP
      END
```

SPL/V Example

```
<<
 *    This program calculates total appropriations by the
 *    United States federal government for the years 1977-1979
>>
BEGIN
BYTE ARRAY YEARLY (0:44) :=  << Annual appropriations, in cents    >>
         "046655980996406",
         "050778229148999",
         "056396083378825";

INTEGER ZERO:=0, YEAR;

BYTE ARRAY PACKED'SUM (0:7);     << 8-character field holds        >>
BYTE ARRAY PACKED'TEMP (0:7);    <<   15 digits and sign           >>

BYTE ARRAY ASCII'SUM (0:14);     << 15-digit number in ASCII       >>

ARRAY PRINT'BUFFER'W (0:10);
BYTE ARRAY PRINT'BUFFER'B (*) = PRINT'BUFFER'W;

INTRINSIC ASCII, PRINT;

<< Initialize sum >>
TOS := @PACKED'SUM;  TOS := 15;
TOS := @ZERO;        TOS := 1;
ASSEMBLE (CVBD);

FOR YEAR := 1977 UNTIL 1979 DO
   BEGIN

   << Echo annual data >>
```

```
   ASCII (YEAR, 10, PRINT'BUFFER'B);
   MOVE PRINT'BUFFER'B (4) := "   ";
   MOVE PRINT'BUFFER'B (7) := YEARLY (15 * (YEAR-1977)), (15);
   PRINT (PRINT'BUFFER'W, -22, %40);

   << Convert ASCII to decimal >>
   TOS := @PACKED'TEMP;                TOS := 15;
   TOS := @YEARLY (15 * (YEAR-1977));  TOS := 15;
   ASSEMBLE (CVAD);

   << Add to sum >>
   TOS := @PACKED'SUM;  TOS := 15;
   TOS := @PACKED'TEMP; TOS := 15;
   ASSEMBLE (ADDD);

   END;

<< Convert sum to ASCII, suppressing plus sign >>
TOS := @ASCII'SUM;  TOS := 15;
TOS := @PACKED'SUM;
ASSEMBLE (CVDA ABS);

MOVE PRINT'BUFFER'B := "Total  ";
MOVE PRINT'BUFFER'B (7) := ASCII'SUM, (15);
PRINT (PRINT'BUFFER'W, -22, %40);
END.
```

Appendix A Compiler Library Error Messages

This appendix describes Compiler Library error handling, the intrinsic function XLIBTRAP, and lists error messages.

Many of the Compiler Library procedures detect errors in processed data. When error detection occurs, the procedures call an error routine that reports the error conditions and aborts your program.

The error message format is as follows:

```
procedure name: message
```

For example,

```
**** ALOG(X): X < 0.0 OR X=NaN (MATHERR26)
```

The same number may correspond to more than one error message. This appendix uses the letters **A**, **B**, or **C** to differentiate between these messages, as shown in the example below:

```
19     MESSAGE     A. DTAN(X): X=NaN OR INFINITY
                   B. DTAN(X): TOTAL LOSS OF PRECISION
```

XLIBTRAP

Using the HP FORTRAN 77/XL compiler, you can override the process described above by using the ON statement. For more details, see the *HP FORTRAN 77/XL Reference Manual*. You can also override the normal error functions and specify your own error procedure by using the intrinsic function XLIBTRAP, which has the following format:

```
PROCEDURE xlibtrap;intrinsic;
```

or

```
PROCEDURE xlibtrap (plabel:INTEGER; VAR oldplabel: INTEGER);
EXTERNAL;
```

where

plabel = external label of the user-written error procedure to disarm the library trap mechanisms. If plabel equals zero, control is *not* passed to a user error procedure.

oldplabel = original plabel; returned to permit the user to return to the previous conditions.

Execution proceeds as follows:

1. A library procedure finds an error and calls the library error handler.
2. The library error handler checks for a user-written error procedure or checks for an indication to ignore the trap. To ignore the trap, use the following HP FORTRAN 77/XL statements:

```
ON EXTERNAL ERROR IGNORE
ON INTERNAL ERROR IGNORE
```

(The HP FORTRAN 77/XL ON statement has additional options that make it easier to use than XLIBTRAP. See the *HP FORTRAN 77/XL Reference Manual* for more details.)

3. If a user-written error procedure has not been specified by using XLIBTRAP or with the HP FORTRAN 77/XL ON statement and the trap is not to be ignored, the appropriate error report is produced and the current program is aborted. The library trap mechanism is disarmed when the user-written procedure is called. If the compiler finds an indication to ignore the trap, control is immediately returned to the library procedure.

4. When the user-written error procedure returns control to the library error handler, the library trap mechanism is rearmed.

5. As defined below, a user-written error procedure can either set a result or a set a flag. The QUIT parameter in the example below directs the library error handler to abort the current program or to return control to the procedure that found the error.

A user-written error procedure (named error in this example) could be declared as follows:

```
PROGRAM libtrap_test(output);
```

A Pascal Example Using XLIBTRAP

The following example shows a user-written HP Pascal error procedure to be called whenever an error is detected.

```
{This program tests the ability to invoke library trap handling from Pascal.}

   TYPE

      word_int = integer;

      plabel = word_int;

      stack_array = ARRAY [1 .. 4] of word_int;

   VAR
      Nan_var,    {Holds the value -1.}
      x,y: REAL;  {Used to hold results returned from the library routines.}
      oldplabel : plabel;
      result : real;
      took_trap : Boolean;

   {Declare the sqrt library routine.}
   FUNCTION sqrt : REAL; INTRINSIC;

   {Declare the alog library routine.}
   FUNCTION alog : REAL; INTRINSIC;

   {Declare XLIBTRAP.}
   PROCEDURE xlibtrap; intrinsic;

   {Provide the error routine.}

   PROCEDURE error (VAR marker: stack_array;
                    VAR errornum,
                    VAR quit: word_int);

     {Declare constants.}
     CONST
        sqrt_errornum = 10;
        sqrt_default  = -999.0;

        continue      = 0;
```

```
            abort         = 1;

         {This procedure returns a result of -999 and continues
         the execution of the program if the SQRT routine has
         the error.  Otherwise, the program aborts.}
         BEGIN
            IF errornum = sqrt_errornum THEN BEGIN
               result := sqrt_default;
               took_trap := True;
               quit := continue;
            END
            ELSE quit := abort;
         END;  {error}

         {Start the main body of the program.}
         BEGIN

            {Set up the -1 value.}
            x := -1.0;
            Nan_var := sqrt(x);
            took_trap := False;

            {Arm the trap mechanism by calling XLIBTRAP.}
            xlibtrap(waddress(error), oldplabel);

            writeln ('Calling sqrt with NaN');
            {Call sqrt with a negative value.}
            x := sqrt(Nan_var);
            if took_trap
               then begin
                  x := result;
                  took_trap := False;
               end;

            writeln('X should be -999.0.  X = ', x);

            writeln('This should be the last line before an abort.');

            {Call alog with a negative value.}
            y := alog(Nan_var);

            writeln('This line should NOT be printed');

         END.
```

where

result =	the result of the calculation that produced the error. The library procedure determines the size and type of the value returned to result. If you know that only a subset of the possible result types occurs in your program, the extensive variant RECORD structure shown above is not needed. You can change the value returned to result in the library procedure and have the change reflected in the code if optimization is *not* being used.
errornum =	the error number. (See the "Error Messages" section in this appendix for a list of the error messages.)
quit =	a flag set by the user-written procedure error. If quit equals zero, the trap returns to your program without printing an error message. If quit does not equal zero, your program is aborted.

A FORTRAN Example Using XLIBTRAP

The libtrap_test program in the above example can also be written FORTRAN 77/XL using the ON statement:

```
            PROGRAM libtrap_test

            IMPLICIT NONE

            REAL x,y,NaN
            INTEGER INaN
            EQUIVALENCE (NaN, INaN)
            DATA INaN / Z'7FF0' /  ! bit pattern for single- precision NaN

            ON EXTERNAL ERROR CALL ERROR  ! Set up the trap.

               PRINT *, 'Calling sqrt with', NaN
               x=sqrt (NaN)
               PRINT *, 'x should be -999.0. x=',X
               PRINT *, 'This should be the last line before an abort.'
               y=alog (NaN)
               PRINT *, 'This line should NOT be printed.'
            END

            SUBROUTINE ERROR (errornum, result, operand 1, operand 2)
            IMPLICIT NONE

               INTEGER*2 errornum
               REAL result, sqrt_default, operand1, operand2
               INTEGER sqrt_errornum, abort

               !Declare the constants.

               PARAMETER (sqrt_default= -999.0)
               PARAMETER (sqrt_errornum=10, abort=0)

            print *, 'In ERROR, params =', errornum, result, operand1

               !Test for SQRT

            IF (errornum .eq. sqrt_errornum) THEN

            !If so, set the default of -999.0 and continue
               result=sqrt_default
            ELSE

            !  Abort if it is not SQRT.
               print *, 'Errornum set to ABORT.'
               errornum = abort
            ENDIF
            END
```

Error Messages

1	MESSAGE	ATAN2(X,Y): X=Y=0.0 OR X=Y=INFINITY OR ANY OF X AND Y=NaN
	CAUSE	Both of the arguments are zero or infinity, or any of the arguments is a NaN.
	ACTION	Change the arguments so both of them are not zero or infinity, and neither is a NaN.
	PROCEDURE	**ATAN2** or **ATAN2'**
3	MESSAGE	DATAN2(X,Y): X=Y=0.0 OR X=Y=INFINITY OR ANY OF X AND Y=NaN

CAUSE Both of the arguments are zero or infinity, or any of the arguments is a NaN.

ACTION Change the arguments so both of them are not zero or infinity, and neither of the arguments is a NaN.

PROCEDURE **DATAN2** or **DATAN2'**

5 MESSAGE **A.** EXP(X): X=NaN
B. EXP(X) OVERFLOW
C. EXP(X) UNDERFLOW

CAUSE **A.** The argument is a NaN.
B. The argument is greater than 88.7228.
C. The argument is less than -87.3365.

ACTION **A.** Change the argument so it is not a NaN.
B. Change the argument so it is less than or equal to 88.7228.
C. Change the argument so it is greater than or equal to -87.3365.

PROCEDURE **EXP** or **EXP'**

6 MESSAGE **A.** DEXP(X) OVERFLOW
B. DEXP(X) UNDERFLOW
C. DEXP(X): X = NaN

CAUSE **A.** The argument is greater than 709.7827128933838.
B. The argument is less than -708.3964185322639.
C. The argument is a NaN.

ACTION **A.** Change the argument to less than or equal to 709.7827128933838.
B. Change the argument to greater than or equal to ...-708.3964185322639.
C. Change the argument so it is not a NaN.

PROCEDURE **DEXP** or **DEXP'**

7 MESSAGE ALOG(X): X< 0.0 OR X=NaN

CAUSE The argument is less than zero or is a NaN.

ACTION Change the argument so it is greater than or equal to zero or is not a NaN.

PROCEDURE **ALOG** or **ALOG'**

8 MESSAGE DLOG(X): X< 0.0 OR X=NaN

CAUSE The argument is less than zero or is a NaN.

ACTION Change the argument so it is greater than or equal to zero and not a NaN.

PROCEDURE **DLOG** or **DLOG'**

9 MESSAGE CABS(X): ANY PART OF X=NaN

CAUSE Either or both parts of the complex argument is a NaN.

ACTION Change the argument so neither part is a NaN.

PROCEDURE **CABS** or **CABS'**

10 MESSAGE SQRT(X): X< 0.0 OR X=NaN

CAUSE The argument is less than zero or is a NaN.

ACTION Change the argument to be greater than or equal to zero, or not a NaN.

PROCEDURE **SQRT** or **SQRT'**

11 MESSAGE DSQRT(X): X< 0.0 OR X=NaN

CAUSE The argument is less than zero or is a NaN.

ACTION Change the argument so it is greater than or equal to zero, or not a NaN.

PROCEDURE **DSQRT** or **DSQRT'**

14 MESSAGE SIN(X): X=NaN OR INFINITY

CAUSE The argument is a NaN or infinity.

ACTION Change the argument so it is not a NaN or an infinity.

PROCEDURE **SIN** or **SIN'**

15 MESSAGE **A.** DSIN(X): X=NaN OR INFINITY
B. DSIN(X): TOTAL LOSS OF PRECISION

CAUSE **A.** The argument is a NaN or an infinity.
B. The absolute value of the argument is greater than or equal to
...2.6378256 x 107 .

ACTION **A.** Change the argument so it is not a NaN or an infinity.
B. Change the absolute value of the argument so it is less than
...2.6378256 x 107 .

PROCEDURE **DSIN** or **DSIN'**

16 MESSAGE COS(X): X=NaN OR INFINITY

CAUSE The argument is a NaN or infinity.

ACTION Change the argument so it is not a NaN or an infinity.

PROCEDURE **COS** or **COS'**

17 MESSAGE **A.** DCOS(X): X=NaN OR INFINITY
B. DCOS(X): TOTAL LOSS OF PRECISION

CAUSE **A.** The argument is a NaN or infinity.
B. The absolute value of the argument is greater than or equal to
...2.637856 x 107 .

ACTION **A.** Change the argument so it is not a NaN or an infinity.
B. Change the absolute value of the argument so it is less than or
...equal to 2.637856 x 107 .

PROCEDURE **DCOS** or **DCOS'**

18 MESSAGE **A.** TAN(X): X=NaN OR INFINITY
B. TAN(X): TOTAL LOSS OF PRECISION

CAUSE **A.** The argument is a NaN or infinity.
B. The absolute value of the argument is greater than 6433.0

ACTION **A.** Change the argument so it is not a NaN or an infinity.
B. Change the argument so its absolute value is less than or equal to
...6433.0.

PROCEDURE **TAN** or **TAN'**

19 MESSAGE **A.** DTAN(X): X=NaN OR INFINITY
B. DTAN(X): TOTAL LOSS OF PRECISION

CAUSE **A.** The argument is a NaN or infinity.
B. The absolute value of the argument is greater than 149078413.0.

ACTION **A.** Change the argument so it is not a NaN or an infinity.
B. Change the absolute value of the argument so it is less than
...149078413.0.

PROCEDURE **DTAN** or **DTAN'**

20 MESSAGE ALOG10(X), X< 0.0 OR X=NaN

CAUSE The argument is less than zero or is a NaN.

ACTION Change the argument so it is greater than or equal to zero or is not a NaN.

PROCEDURE **ALOG10**

21 MESSAGE DLOG10(X), X < 0.0 OR X=NaN

CAUSE The argument is less than zero or is a NaN.

ACTION Change the argument so it is greater than or equal

to zero and not a NaN.

PROCEDURE **DLOG10**

22 MESSAGE **A.** SINH(X): X=NaN
B. SINH(X) OVERFLOW

CAUSE **A.** The argument is a NaN.
B. The absolute value of the argument is greater than 88.7196.

ACTION **A.** Change the argument so it is not a NaN.
B. Change the argument so its absolute value is less than or equal to 88.7196.

PROCEDURE **SINH** or **SINH'**

23 MESSAGE **A.** DSINH(X): X=NaN
B. DSINH(X) OVERFLOW

CAUSE **A.** The argument is a NaN.
B. The absolute value of the argument is greater than 709.7795518826419.

ACTION **A.** Change the argument so it is not a NaN.
B. Change the argument so its absolute value is less than or equal to 709.7795518826419.

PROCEDURE **DSINH** or **DSINH'**

24 MESSAGE **A.** COSH(X): X=NaN
B. COSH(X) OVERFLOW

CAUSE **A.** The argument is a NaN.
B. The absolute value of the argument is greater than 88.7196.

ACTION **A.** Change the argument so it is not a NaN.
B. Change the argument so its absolute value is less than or equal to 88.7196.

PROCEDURE **COSH** or **COSH'**

25 MESSAGE **A.** DCOSH(X): X=NaN
B. DCOSH(X) OVERFLOW

CAUSE **A.** The argument is a NaN.
B. The absolute value of the argument is greater than 709.7795518826419.

ACTION **A.** Change the argument so it is not a NaN.
B. Change the argument so its absolute value is less than or equal to 709.7795518826419.

PROCEDURE **DCOSH** or **DCOSH'**

26 MESSAGE ASIN(X): |X| > 1.0 OR X=NaN

CAUSE The absolute value of the argument is greater than 1.0 or the argument is a NaN.

ACTION Change the argument so its absolute value is not greater than zero and the argument is not a NaN.

PROCEDURE **ASIN** or **ASIN'**

27 MESSAGE DASIN(X): |X| > 1.0 OR X=NaN

CAUSE The absolute value of the argument is greater than 1.0 or the argument is a NaN.

ACTION Change the argument so its absolute value is not greater than zero and the argument is not a NaN.

PROCEDURE **DASIN** or **DASIN'**

28 MESSAGE ACOS(X): |X| > 1.0 OR X=NaN

CAUSE The absolute value of the argument is greater than 1.0 or the argument is a NaN.

ACTION Change the argument so its absolute value is not greater than zero and the argument is not a NaN.

PROCEDURE **ACOS** or **ACOS'**

29 MESSAGE DACOS(X): |X| > 1.0 or X=NaN

CAUSE The absolute value of the argument is greater than 1.0 or the argument is a NaN.

ACTION Change the argument so its absolute value is not greater than zero and the argument is not a NaN.

PROCEDURE **DACOS** or **DACOS'**

30 MESSAGE DATAN(X): X=NaN

CAUSE The argument is a NaN.

ACTION Change the argument so it is not a NaN.

PROCEDURE **DATAN** or **DATAN'**

31 MESSAGE ATAN(X): X=NaN

CAUSE The argument is a NaN.

ACTION Change the argument so it is not a NaN.

PROCEDURE **ATAN** or **ATAN'**

34 MESSAGE DABS(X) X=NaN

CAUSE The argument is a NaN.
ACTION Change the argument so it is not a NaN.

PROCEDURE **DABS** or **DABS'**

36 MESSAGE TANH(X): X=NaN

CAUSE The argument is a NaN.

ACTION Change the argument so it is not a NaN.

PROCEDURE **TANH** or **TANH'**

37 MESSAGE DMOD(X,Y): ANY OF X AND Y=NaN OR X=INFINITY OR Y=0.0

CAUSE Either or both of the arguments are NaN, or X=infinity, or Y=0.0.

ACTION Change the arguments so neither of them is a NaN, X is not infinity, and Y is not zero.

PROCEDURE **DMOD**

38 MESSAGE AMOD(X,Y): ANY OF X AND Y=NaN OR X=INFINITY OR Y=0.0

CAUSE Either or both arguments are NaN or X=infinity, or Y=0.0.

ACTION Change the arguments so neither of them is a NaN, X is not infinity, and Y is not zero.

PROCEDURE **AMOD** or **AMOD'**

39 MESSAGE DTANH(X): X=NaN

CAUSE The argument is a NaN.

ACTION Change the argument so it is not a NaN.

PROCEDURE **DTANH** or **DTANH'**

41 MESSAGE AINT(X): X=NaN

CAUSE The argument is a NaN.
ACTION Change the argument so it is not a NaN.

PROCEDURE **AINT** or **AINT'**

42 MESSAGE DDINT(X) OR DDINT(X): X=NaN

CAUSE The argument is a NaN.

ACTION Change the argument so it is not a NaN.

PROCEDURE **DDINT** or **DDINT'**

47 MESSAGE SIGN FUNCTION: ANY ARGUMENT = NaN

CAUSE Either or both of the arguments are NaN.

ACTION Change the arguments so neither of them is a NaN.

PROCEDURE **SIGN** or **SIGN'**