

File created: 16-May-90 13:16:23 {DSK}<usr>local>lde>lispcore>sources>CMLFLOAT.;2

changes to: (IL:VARS IL:CMLFLOATCOMS)

previous date: 7-Feb-88 15:16:05 {DSK}<usr>local>lde>lispcore>sources>CMLFLOAT.;1

Read Table: XCL

Package: LISP

Format: XCSS

; Copyright (c) 1986, 1987, 1988, 1990 by Venue & Xerox Corporation. All rights reserved.

(IL:RPAQQ **IL:CMLFLOATCOMS**

(

::: CMLFLOAT -- Covering sections 12.5-12.5.3 irrational, transcendental, exponential, logarithmic, trigonometric, and hyperbolic functions. Section  
::: 12.10, implementation parameters.

```
(IL:DECLARE\ IL:DONTCOPY IL:DOEVAL@COMPILE ; To generate unboxed opcodes
  (IL:FILES IL:UNBOXEDOPS) ; To get constants from lfloat

  (IL:FILES (IL:LOADCOMP
    IL:LLFLOAT))
(IL:COMS
  ;; Section 12.10, implementation parameters.
  ;; %FLOAT allows us to recreate FLOATPs in a way that is independent of the ordinary reading and printing FLOATPs to files
  ;; which involves loss of the last couple bits of accuracy due to rounding effects.
  (IL:FUNCTIONS %FLOAT)
  (IL:VARIABLES MOST-POSITIVE-FIXNUM MOST-NEGATIVE-FIXNUM)
  (IL:VARIABLES MOST-POSITIVE-SINGLE-FLOAT LEAST-POSITIVE-SINGLE-FLOAT
    LEAST-NEGATIVE-SINGLE-FLOAT MOST-NEGATIVE-SINGLE-FLOAT)
  (IL:VARIABLES MOST-POSITIVE-SHORT-FLOAT LEAST-POSITIVE-SHORT-FLOAT LEAST-NEGATIVE-SHORT-FLOAT
    MOST-NEGATIVE-SHORT-FLOAT MOST-POSITIVE-DOUBLE-FLOAT LEAST-POSITIVE-DOUBLE-FLOAT
    LEAST-NEGATIVE-DOUBLE-FLOAT MOST-NEGATIVE-DOUBLE-FLOAT MOST-POSITIVE-LONG-FLOAT
    LEAST-POSITIVE-LONG-FLOAT LEAST-NEGATIVE-LONG-FLOAT MOST-NEGATIVE-LONG-FLOAT)
  ;; EPSILON is the smallest positive floating point number such that (NOT (= (FLOAT 1 EPSILON) (+ (FLOAT 1 EPSILON)
  ;; EPSILON)))
  (IL:VARIABLES SINGLE-FLOAT-EPSILON)
  (IL:VARIABLES SHORT-FLOAT-EPSILON DOUBLE-FLOAT-EPSILON LONG-FLOAT-EPSILON)
  ;; NEGATIVE-EPSILON is the smallest negative floating point number such that (NOT (= (FLOAT 1 NEGATIVE-EPSILON) (-
  ;; (FLOAT 1 NEGATIVE-EPSILON) NEGATIVE-EPSILON)))
  (IL:VARIABLES SINGLE-FLOAT-NEGATIVE-EPSILON)
  (IL:VARIABLES SHORT-FLOAT-NEGATIVE-EPSILON DOUBLE-FLOAT-NEGATIVE-EPSILON
    LONG-FLOAT-NEGATIVE-EPSILON)
  (IL:VARIABLES PI))
(IL:COMS
  ;; Internal constants
  (IL:DECLARE\ IL:DONTCOPY IL:DOEVAL@COMPILE
    (IL:VARIABLES %E %2PI %PI %2PI/3 %PI/2 %-PI/2 %PI/3 %PI/4 %-PI/4 %PI/6 %2/PI)))
(IL:COMS
  ;; Utility macros
  (IL:DECLARE\ IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FUNCTIONS %FLOAT-UNBOX %GET-TABLE-ENTRY
    %POLYEVAL %UFTRUNCATE %UMAKE-FLOAT)))
;; Unpack floating point functions
(IL:COMS (IL:FUNCTIONS DECODE-FLOAT SCALE-FLOAT FLOAT-RADIX FLOAT-SIGN FLOAT-DIGITS FLOAT-PRECISION
  INTEGER-DECODE-FLOAT))
(IL:COMS
  ;; Exp (e to the power x)
  (IL:COMS (IL:DECLARE\ IL:DONTCOPY IL:DOEVAL@COMPILE (IL:VARIABLES %LOG-BASE2-E))
    (IL:VARIABLES %EXP-POLY %EXP-TABLE))
  (IL:FUNCTIONS %EXP-FLOAT)
  (IL:FUNCTIONS EXP))
(IL:COMS
  ;; Expt (x to the power y)
  (IL:FUNCTIONS %EXPT-INTEG %EXPT-FLOAT-INTEG)
  (IL:FUNCTIONS EXPT))
(IL:COMS
  ;; Log (log base e)
  (IL:COMS (IL:DECLARE\ IL:DONTCOPY IL:DOEVAL@COMPILE (IL:VARIABLES %LOG2 %SQRT2))
    (IL:VARIABLES %LOG-PPOLY %LOG-QPOLY))
  (IL:FUNCTIONS %LOG-FLOAT)
  (IL:FUNCTIONS LOG))
(IL:COMS
  ;; Sqrt
  (IL:FUNCTIONS %SQRT-FLOAT %SQRT-COMPLEX)
```

```

        (IL:FUNCTIONS SQRT))
(IL:COMS
  ;; Sin and Cos
  (IL:COMS (IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE (IL:VARIABLES %SIN-EPSILON))
    (IL:VARIABLES %SIN-PPOLY %SIN-QPOLY))
  (IL:FUNCTIONS %SIN-FLOAT)
  (IL:FUNCTIONS SIN COS))
(IL:COMS
  ;; Tan
  (IL:COMS (IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE (IL:VARIABLES %TAN-EPSILON))
    (IL:VARIABLES %TAN-PPOLY %TAN-QPOLY))
  (IL:FUNCTIONS %TAN-FLOAT)
  (IL:FUNCTIONS TAN))
(IL:COMS
  ;; Asin and Acos
  (IL:COMS (IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE (IL:VARIABLES %ASIN-EPSILON))
    (IL:VARIABLES %ASIN-PPOLY %ASIN-QPOLY))
  (IL:FUNCTIONS %ASIN-FLOAT)
  (IL:FUNCTIONS ASIN ACOS))
(IL:COMS
  ;; Atan
  (IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE (IL:VARIABLES %SQRT3 %2-SQRT3 %INV-2-SQRT3))
  (IL:FUNCTIONS %ATAN-FLOAT)
  (IL:FUNCTIONS ATAN))
(IL:COMS
  ;; Cis (exp (i x))
  (IL:FUNCTIONS CIS))
(IL:COMS
  ;; Sinh, Cosh Tanh
  (IL:FUNCTIONS SINH COSH TANH))
(IL:COMS
  ;; Asinh Acosh Atanh
  (IL:FUNCTIONS ASINH ACOSH ATANH))
(IL:COMS
  ;; rational and rationalize
  (IL:FUNCTIONS %RATIONAL-FLOAT %RATIONALIZE-FLOAT))
(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE (IL:LOCALVARS . T))
(IL:PROP (IL:MAKEFILE-ENVIRONMENT IL:FILETYPE)
  IL:CMLFLOAT))

```

;;; CMLFLOAT -- Covering sections 12.5-12.5.3 irrational, transcendental, exponential, logarithmic, trigonometric, and hyperbolic functions. Section 12.10, implementation parameters.

```

(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE
(IL:FILESLOAD IL:UNBOXEDOPS)
(IL:FILESLOAD (IL:LOADCOMP)
  IL:LLFLOAT)
)

```

;; Section 12.10, implementation parameters.

;; %FLOAT allows us to recreate FLOATPs in a way that is independent of the ordinary reading and printing FLOATPs to files which involves loss of the last couple bits of accuracy due to rounding effects.

```

(DEFUN %FLOAT (HIWORD LOWORD)
  (IL:\FLOATBOX (IL:\VAG2 HIWORD LOWORD)))

```

```

(DEFCONSTANT MOST-POSITIVE-FIXNUM 65535)

```

```

(DEFCONSTANT MOST-NEGATIVE-FIXNUM -65536)

```

```

(DEFCONSTANT MOST-POSITIVE-SINGLE-FLOAT (%FLOAT 32639 65535))

```

```

(DEFCONSTANT LEAST-POSITIVE-SINGLE-FLOAT (%FLOAT 0 1))

```

```

(DEFCONSTANT LEAST-NEGATIVE-SINGLE-FLOAT (%FLOAT 32768 1))

```

```

(DEFCONSTANT MOST-NEGATIVE-SINGLE-FLOAT (%FLOAT 65407 65535))

```

(DEFCONSTANT **MOST-POSITIVE-SHORT-FLOAT** MOST-POSITIVE-SINGLE-FLOAT)

(DEFCONSTANT **LEAST-POSITIVE-SHORT-FLOAT** LEAST-POSITIVE-SINGLE-FLOAT)

(DEFCONSTANT **LEAST-NEGATIVE-SHORT-FLOAT** LEAST-NEGATIVE-SINGLE-FLOAT)

(DEFCONSTANT **MOST-NEGATIVE-SHORT-FLOAT** MOST-NEGATIVE-SINGLE-FLOAT)

(DEFCONSTANT **MOST-POSITIVE-DOUBLE-FLOAT** MOST-POSITIVE-SINGLE-FLOAT)

(DEFCONSTANT **LEAST-POSITIVE-DOUBLE-FLOAT** LEAST-POSITIVE-SINGLE-FLOAT)

(DEFCONSTANT **LEAST-NEGATIVE-DOUBLE-FLOAT** LEAST-NEGATIVE-SINGLE-FLOAT)

(DEFCONSTANT **MOST-NEGATIVE-DOUBLE-FLOAT** MOST-NEGATIVE-SINGLE-FLOAT)

(DEFCONSTANT **MOST-POSITIVE-LONG-FLOAT** MOST-POSITIVE-SINGLE-FLOAT)

(DEFCONSTANT **LEAST-POSITIVE-LONG-FLOAT** LEAST-POSITIVE-SINGLE-FLOAT)

(DEFCONSTANT **LEAST-NEGATIVE-LONG-FLOAT** LEAST-NEGATIVE-SINGLE-FLOAT)

(DEFCONSTANT **MOST-NEGATIVE-LONG-FLOAT** MOST-NEGATIVE-SINGLE-FLOAT)

:: EPSILON is the smallest positive floating point number such that (NOT (= (FLOAT 1 EPSILON) (+ (FLOAT 1 EPSILON) EPSILON)))

(DEFCONSTANT **SINGLE-FLOAT-EPSILON** (%FLOAT (ASH 103 7) 1))

(DEFCONSTANT **SHORT-FLOAT-EPSILON** SINGLE-FLOAT-EPSILON)

(DEFCONSTANT **DOUBLE-FLOAT-EPSILON** SINGLE-FLOAT-EPSILON)

(DEFCONSTANT **LONG-FLOAT-EPSILON** SINGLE-FLOAT-EPSILON)

:: NEGATIVE-EPSILON is the smallest negative floating point number such that (NOT (= (FLOAT 1 NEGATIVE-EPSILON) (- (FLOAT 1 NEGATIVE-EPSILON) NEGATIVE-EPSILON)))

(DEFCONSTANT **SINGLE-FLOAT-NEGATIVE-EPSILON** (%FLOAT 13184 0))

(DEFCONSTANT **SHORT-FLOAT-NEGATIVE-EPSILON** SINGLE-FLOAT-NEGATIVE-EPSILON)

(DEFCONSTANT **DOUBLE-FLOAT-NEGATIVE-EPSILON** SINGLE-FLOAT-NEGATIVE-EPSILON)

(DEFCONSTANT **LONG-FLOAT-NEGATIVE-EPSILON** SINGLE-FLOAT-NEGATIVE-EPSILON)

(DEFCONSTANT **PI** (%FLOAT 16457 4059))

:: Internal constants

(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE

(DEFCONSTANT **%E** (%FLOAT 16429 63572))

(DEFCONSTANT **%2PI** (%FLOAT 16585 4059))

(DEFCONSTANT **%PI** (%FLOAT 16457 4059))

(DEFCONSTANT **%2PI/3** (%FLOAT 16390 2706))

```

{MEDLEY}<sources>CMLFLOAT.;1

(DEFCONSTANT %PI/2 (%FLOAT 16329 4059))

(DEFCONSTANT %-PI/2 (%FLOAT 49097 4059))

(DEFCONSTANT %PI/3 (%FLOAT 16262 2706))

(DEFCONSTANT %PI/4 (%FLOAT 16201 4059))

(DEFCONSTANT %-PI/4 (%FLOAT 48969 4059))

(DEFCONSTANT %PI/6 (%FLOAT 16134 2706))

(DEFCONSTANT %2/PI (%FLOAT 16162 63875))
)

;; Utility macros

(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE

(DEFMACRO %FLOAT-UNBOX (FLOAT SIGN EXP HI LO &OPTIONAL DONTSHIFT)
  ;; If dontshift is T -- the floatp fields are simply unpacked (with the hiddenbit restored -- and exp set to 1 for denormalized numbers). If dontshift is NIL
  ;; -- exp, hi and lo are fiddled so the high bit of hi is on.
  \ (LET ((FLONUM (FLOAT ,FLOAT)))
        (SETQ ,SIGN (IL:|fetch| (IL:FLOATP IL:SIGNBIT) IL:|of| FLONUM))
        (SETQ ,EXP (IL:|fetch| (IL:FLOATP IL:EXPONENT) IL:|of| FLONUM))
        (SETQ ,HI (IL:|fetch| (IL:FLOATP IL:HIFRACTION) IL:|of| FLONUM))
        (SETQ ,LO (IL:|fetch| (IL:FLOATP IL:LOFRACTION) IL:|of| FLONUM))
        (IF (EQ ,EXP IL:\MAX.EXPONENT) ; might want to check for NaN's here if EXP = \MAX.EXPONENT
            (ERROR "Not a number: ~s" FLONUM))
        (IF (EQ 0 ,EXP)
            (WHEN (NOT (AND (EQ 0 ,HI)
                          (EQ 0 ,LO))) ; Denormalized number
                (SETQ ,EXP 1)
                ,@(IF (NULL DONTSHIFT)
                    \ ((LOOP (IF (NOT (EQ 0 (LOGAND ,HI IL:\HIDDENBIT)))
                                (RETURN NIL))
                       (IL:.LLSH1. ,HI ,LO)
                       (SETQ ,EXP (1- ,EXP)))))) ; Restore the hidden bit
                (SETQ ,HI (+ ,HI IL:\HIDDENBIT)))
            ,@(IF (NULL DONTSHIFT)
                \ ((IL:.LLSH8. ,HI ,LO)))
            NIL))

(DEFMACRO %GET-TABLE-ENTRY (ARRAY INDEX)
  \ (IL:\GETBASEFLOATP (IL:|fetch| (IL:ONED-ARRAY IL:BASE) IL:|of| ,ARRAY)
    (IL:LLSH ,INDEX 1))

(DEFMACRO %POLYEVAL (X COEFFS DEGREE)
  \ (IL:\FLOATBOX ((IL:OPCODES IL:UBFLOAT3 0)
                  (IL:\FLOATUNBOX ,X)
                  (IL:|fetch| (IL:ONED-ARRAY IL:BASE) IL:|of| ,COEFFS)
                  ,DEGREE))

(DEFMACRO %UFTRUNCATE (INT REM FLOAT &OPTIONAL DIVISOR)
  ;; As in truncate. Assumes FLOAT and DIVISOR are unboxed floatp's.
  (IF DIVISOR
    \ (LET ((FFLOAT ,FLOAT)
            (FDIVISOR ,DIVISOR))
        (DECLARE (TYPE FLOAT FFLOAT FDIVISOR))
        (SETQ ,INT (IL:UFIX (IL:FQUOTIENT FFLOAT FDIVISOR)))
        (SETQ ,REM (- FFLOAT (* FDIVISOR (FLOAT ,INT))))
        NIL)
    \ (LET ((FFLOAT ,FLOAT))
        (DECLARE (TYPE FLOAT FFLOAT))
        (SETQ ,INT (IL:UFIX FFLOAT))
        (SETQ ,REM (- FFLOAT (FLOAT ,INT)))
        NIL))

(DEFMACRO %UMAKE-FLOAT (SIGN EXP HI LOW)
  ;; as in \makefloat -- but produces an unboxed number
  \ (IL:\FLOATBOX ((IL:OPENLAMBDA (SIGN EXP HI LO)
                          (IL:.LRSH8. HI LO)
                          (SETQ HI (+ (ASH EXP 7)

```

```

                (LOGAND 127 HI)))
        (IF (EQ SIGN 1)
            (SETQ HI (LOGIOR IL:\\SIGNBIT HI)))
        (IL:\\VAG2 HI LO))
    ,SIGN
    ,EXP
    ,HI
    ,LOW)))
)

```

:: Unpack floating point functions

```

(DEFUN DECODE-FLOAT (FLOAT)
  (SETQ FLOAT (FLOAT FLOAT))
  (IF (= FLOAT 0.0)
      (VALUES 0.0 0 1.0)
      (LET (SIGN EXP HI LO)
          (%FLOAT-UNBOX FLOAT SIGN EXP HI LO)
          (VALUES (IL:\\MAKEFLOAT 0 (1- IL:\\EXPONENT.BIAS)
                    HI LO)
                  (- EXP (1- IL:\\EXPONENT.BIAS))
                  (IF (EQ SIGN 0)
                      1.0
                      -1.0))))))

```

```

(DEFUN SCALE-FLOAT (FLOAT INTEGER &OPTIONAL OLD-BOX)
  (SETQ FLOAT (FLOAT FLOAT))
  (IF (= FLOAT 0.0)
      0.0
      (LET (SIGN EXP HI LO)
          (%FLOAT-UNBOX FLOAT SIGN EXP HI LO)
          (IL:\\MAKEFLOAT SIGN (+ EXP INTEGER)
                        HI LO NIL OLD-BOX))))

```

```

(DEFUN FLOAT-RADIX (FLOAT)
  2)

```

```

(DEFUN FLOAT-SIGN (FLOAT1 &OPTIONAL FLOAT2 OLD-BOX)
  :: Old-box is a floatp box to reuse (may be eq to float2)
  (IF (FLOATP FLOAT1)
      (IF (NULL FLOAT2)
          (IF (MINUSP FLOAT1)
              -1.0
              1.0)
          (IF (FLOATP FLOAT2)
              (IF (EQ (MINUSP FLOAT1)
                      (MINUSP FLOAT2))
                  FLOAT2
                  (IF (FLOATP OLD-BOX)
                      (LET ((NEW-SIGN-BIT (IF (EQ 0 (IL:FETCH (IL:FLOATP IL:SIGNBIT) IL:OF FLOAT2))
                                              1
                                              0)))
                          :: Now smash the old-box
                          (IL:\\PUTBASEFLOATP OLD-BOX 0 FLOAT2)
                          (IL:replace (IL:FLOATP IL:SIGNBIT) IL:of OLD-BOX IL:with NEW-SIGN-BIT)
                          OLD-BOX)
                          (- FLOAT2)))
                      (%NOT-FLOAT-ERROR FLOAT2)))
              (%NOT-FLOAT-ERROR FLOAT1)))

```

```

(DEFUN FLOAT-DIGITS (FLOAT)
  (IF (FLOATP FLOAT)
      24
      (%NOT-FLOAT-ERROR FLOAT)))

```

```

(DEFUN FLOAT-PRECISION (FLOAT)
  (IF (FLOATP FLOAT)
      (IF (= FLOAT 0.0)
          0
          (LET (SIGN EXP HI LO)
              (%FLOAT-UNBOX FLOAT SIGN EXP HI LO T)
              (IF (< HI IL:\\HIDDENBIT) ; Denormalized number
                  (IF (EQ HI 0)
                      (INTEGER-LENGTH LO)
                      (+ 16 (INTEGER-LENGTH HI)))
                  24))) ; Normalized number
          (%NOT-FLOAT-ERROR FLOAT)))

```

(DEFUN **INTEGER-DECODE-FLOAT** (FLOAT)

;; As in decode-float -- but returns integers

```
(SETQ FLOAT (FLOAT FLOAT))
(IF (= FLOAT 0.0)
  (VALUES 0 0 1)
  (LET (SIGN EXP HI LO)
    (%FLOAT-UNBOX FLOAT SIGN EXP HI LO T)
    (VALUES (+ (ASH HI 16)
              LO)
            (- EXP (+ IL:\EXPONENT.BIAS 23))
            (IF (EQ SIGN 0)
                1
                -1))))))
```

;; Exp (e to the power x)

(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE

(DEFCONSTANT **%LOG-BASE2-E** (%FLOAT 16312 43579))  
)

(XCL:DEFGLOBALVAR **%EXP-POLY**

;; %EXP-POLY contains P and Q coefficients of Hart et al EXPB 1103 rational approximation to (EXPT 2 X) in interval (0 .125).

```
(MAKE-ARRAY 6 :ELEMENT-TYPE 'SINGLE-FLOAT :INITIAL-CONTENTS (LIST (%FLOAT 15549 17659)
                                                                    (%FLOAT 16256 0)
                                                                    (%FLOAT 16801 38273)
                                                                    (%FLOAT 17257 7717)
                                                                    (%FLOAT 17597 11739)
                                                                    (%FLOAT 17800 30401))))
```

(XCL:DEFGLOBALVAR **%EXP-TABLE**

;; %EXP-TABLE contains values of powers (EXPT 2 (/ N 8)) .

```
(MAKE-ARRAY 8 :ELEMENT-TYPE 'SINGLE-FLOAT :INITIAL-CONTENTS (LIST (%FLOAT 16256 0)
                                                                    (%FLOAT 16267 38338)
                                                                    (%FLOAT 16280 14320)
                                                                    (%FLOAT 16293 65239)
                                                                    (%FLOAT 16309 1267)
                                                                    (%FLOAT 16325 26410)
                                                                    (%FLOAT 16343 17661)
                                                                    (%FLOAT 16362 49351))))
```

(DEFUN **%EXP-FLOAT** (X)

;; (CL:EXP X) for float X calculated via EXPB 1103 rational approximation of Hart et al.

```
(LET ((FX (FLOAT X))
      R M N ANSWER RECIPFLG)
  (DECLARE (TYPE FLOAT FX R))
  ;; First, arrange X to be in interval (0 infinity) via identity (CL:EXP (minus X)) = (/ 1.0 (CL:EXP X))
  (WHEN (IL:UFLESSP FX 0.0)
    (SETQ FX (IL:UFMINUS FX))
    (SETQ RECIPFLG T))
  ;; Next, the problem of (CL:EXP X) is converted into a problem (EXPT 2 Y) where Y = (* %LOG-BASE2-E X).
  ;; Then range reduction is accomplished via (EXPT 2 Y) = (* (EXPT 2 M) (EXPT 2 (/ N 8)) (EXPT 2 R)) where M and N are integers and R is a
  ;; float in the interval (0.0 .125).
  ;; After M, N, R are determined, (EXPT 2 M) is effected by scaling, (EXPT 2 (/ N 8)) is found by table lookup, and (EXPT 2 R) is calculated by
  ;; rational approximation EXPB 1103 of Hart et al.
  (%UFTRUNCATE M R (* %LOG-BASE2-E FX))
  (%UFTRUNCATE N R 0.125)
  (SETQ FX (IL:FTIMES (%GET-TABLE-ENTRY %EXP-TABLE N)
                    (IL:FQUOTIENT (%POLYEVAL R %EXP-POLY 5)
                                   (%POLYEVAL (IL:UFMINUS R)
                                               %EXP-POLY 5))))
  (COND
    (RECIPFLG (SETQ ANSWER (SETQ FX (IL:FQUOTIENT 1.0 FX)))
              (SCALE-FLOAT ANSWER (- M)
                            ANSWER))
    (T (SETQ ANSWER FX)
       (SCALE-FLOAT ANSWER M ANSWER))))))
```

(DEFUN **EXP** (NUMBER)

```
(TYPECASE NUMBER
  (COMPLEX (LET ((EXP (%EXP-FLOAT (COMPLEX-REALPART NUMBER)))
                (Y (COMPLEX-IMAGPART NUMBER)))
              (COMPLEX (* EXP (COS Y))
                       (* EXP (SIN Y)))))
  (NUMBER (%EXP-FLOAT NUMBER)))
```



```

(FLOAT (%EXPT-FLOAT-INTEGER BASE-NUMBER POWER-NUMBER))
(COMPLEX (* (%EXPT-FLOAT-INTEGER (%COMPLEX-ABS BASE-NUMBER)
POWER-NUMBER)
(CIS (* POWER-NUMBER (PHASE BASE-NUMBER))))))
(OTHERWISE (%NOT-NUMBER-ERROR BASE-NUMBER))))
(NUMBER (IF (= BASE-NUMBER 0)
BASE-NUMBER
(EXP (* POWER-NUMBER (LOG BASE-NUMBER))))))
(OTHERWISE (%NOT-NUMBER-ERROR POWER-NUMBER))))

```

:: Log (log base e)

```
(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE
```

```
(DEFCONSTANT %LOG2 (%FLOAT 16177 29208))
```

```
(DEFCONSTANT %SQRT2 (%FLOAT 16309 1267))
)
```

(XCL:DEFGLOBALVAR %LOG-PPOLY

:: %LOG-PPOLY and %LOG-QPOLY contain P and Q coefficients of Hart et al LOGE 2707 rational approximation to (LOG X) in interval ((SQRT .5) (SQRT 2))

```
(MAKE-ARRAY 5 :ELEMENT-TYPE 'SINGLE-FLOAT :INITIAL-CONTENTS (LIST (%FLOAT 16042 22803)
(%FLOAT 49484 23590)
(%FLOAT 17044 17982)
(%FLOAT 49926 37153)
(%FLOAT 17046 5367))))
```

(XCL:DEFGLOBALVAR %LOG-QPOLY

:: %LOG-PPOLY and %LOG-QPOLY contain P and Q coefficients of Hart et al LOGE 2707 rational approximation to (LOG X) in interval ((SQRT .5) (SQRT 2))

```
(MAKE-ARRAY 5 :ELEMENT-TYPE 'SINGLE-FLOAT :INITIAL-CONTENTS (LIST (%FLOAT 16256 0)
(%FLOAT 49512 9103)
(%FLOAT 16992 42274)
(%FLOAT 49823 38048)
(%FLOAT 16918 5367))))
```

(DEFUN %LOG-FLOAT (X)

:: (LOG X) for positive float X.

```
(IF (<= (SETQ X (FLOAT X))
0.0)
(ERROR "Log of zero: ~s" X))
```

:: Range reduce to an R in interval ((SQRT 0.5) (SQRT 2)) via identity (LOG X) = (+ (LOG R) (\* %LOG-2 EXP)) for a suitable integer EXP. exp is found from the exponent field of the IEEE floating point number representation of x.

```
(LET (R EXP ANSWER)
(DECLARE (TYPE FLOAT R))
(LET (SIGN HI LO)
(%FLOAT-UNBOX X SIGN EXP HI LO)
(SETQ EXP (- EXP IL:\EXONENT.BIAS))
(SETQ R (%UMAKE-FLOAT SIGN IL:\EXONENT.BIAS HI LO))
NIL)
(WHEN (IL:UFGREATERP R %SQRT2)
(SETQ EXP (1+ EXP))
(SETQ R (IL:FQUOTIENT R 2.0)))
:: (LOG R) is calculated by rational approximation LOGE 2707 of Hart et al.
(LET* ((Z (IL:FQUOTIENT (1- R)
(1+ R)))
(Z2 (* Z Z)))
(DECLARE (TYPE FLOAT Z Z2))
(SETQ ANSWER (SETQ R (+ (* Z (IL:FQUOTIENT (%POLYEVAL Z2 %LOG-PPOLY 4)
(%POLYEVAL Z2 %LOG-QPOLY 4)))
(* %LOG2 EXP))))))
ANSWER))
```

(DEFUN LOG (NUMBER &OPTIONAL BASE)

```
(IF BASE
(IL:QUOTIENT (LOG NUMBER)
(LOG BASE))
(TYPECASE NUMBER
((OR FLOAT RATIONAL) (IF (MINUSP NUMBER)
(COMPLEX (%LOG-FLOAT (- NUMBER)
PI)
(%LOG-FLOAT NUMBER))))
(COMPLEX (COMPLEX (%LOG-FLOAT (%COMPLEX-ABS NUMBER)
(PHASE NUMBER)))
(OTHERWISE (%NOT-NUMBER-ERROR NUMBER))))))
```



:: Sqrt

```
(DEFUN %SQRT-FLOAT (X)
  ;; (SQRT X) for nonnegative float X
  (SETQ X (FLOAT X))
  (IF (<= X 0.0)
    0.0
    (LET ((FX X)
          (V)
          (DECLARE (TYPE FLOAT FX V)))
      (LET (SIGN EXP HI LO)
          (%FLOAT-UNBOX X SIGN EXP HI LO)
          ;; First guess
          (SETQ V (%UMAKE-FLOAT 0 (+ (ASH EXP -1)
                                     (IL:CONSTANT (1+ (ASH IL:\\EXPONENT.BIAS -1))))
                HI LO))
          NIL)
      ;; Four step newton-raphson
      (DOTIMES (I 4 V)
        (SETQ V (* 0.5 (+ V (IL:FQUOTIENT FX V))))))))
```

```
(DEFUN %SQRT-COMPLEX (Z)
  ;; (SQRT X) for complex X.
  (LET ((R (FLOAT (COMPLEX-REALPART Z)))
        (I (FLOAT (COMPLEX-IMAGPART Z)))
        (ABS (SQRT (ABS Z)))
        (PHASE (IL:FQUOTIENT (PHASE Z)
                              2.0)))
      C D E ANSWER)
    (DECLARE (TYPE FLOAT ABS R I))
    ;; Newton's method.
    (LET ((C (* ABS (COS PHASE)))
          (D (* ABS (SIN PHASE)))
          E)
        (DECLARE (TYPE FLOAT C D E))
        (DOTIMES (K 4 (COMPLEX C D))
          (SETQ E (+ (* C C)
                    (* D D)))
          (SETQ C (* 0.5 (+ C (IL:FQUOTIENT (+ (* R C)
                                              (* I D))
                                             E))))
          (SETQ D (* 0.5 (+ D (IL:FQUOTIENT (- (* I C)
                                              (* R D))
                                             E))))))
```

```
(DEFUN SQRT (NUMBER)
  (IF (= NUMBER 0)
    0.0
    (TYPECASE NUMBER
      (COMPLEX (%SQRT-COMPLEX NUMBER))
      (NUMBER (IF (MINUSP NUMBER)
                  (COMPLEX 0 (SQRT (- NUMBER)))
                  (%SQRT-FLOAT NUMBER)))
      (OTHERWISE (%NOT-NUMBER-ERROR NUMBER))))
```

; Negative real axis maps into positive imaginary axis.

:: Sin and Cos

```
(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE
```

(DEFCONSTANT %SIN-EPSILON

```
;; %SIN-EPSILON is sufficiently small that (SIN X) = X for X in interval (0 %SIN-EPSILON). It suffices to take %SIN-EPSILON a little bit smaller than
;; (SQRT (* 6 SINGLE-FLOAT-EPSILON)) which we get by the Taylor series expansion (SIN X) = (+ X (/ (EXPT X 3) 6) ...) (The relative error caused
;; by omitting (/ (EXPT X 3) 6) isn't observable.) Comparison against %SIN-EPSILON is used to avoid POLYEVAL microcode underflow when
;; computing SIN.
(%FLOAT 14720 0))
```

(XCL:DEFGLOBALVAR %SIN-PPOLY

```
;; %SIN-PPOLY and %SIN-QPOLY contain adapted P and Q coefficients of Hart et al SIN 3374 rational approximation to (SIN X) in interval (0 (/ PI
;; 2)). The coefficients for %SIN-PPOLY and %SIN-QPOLY have been computed from Hart using extended precision routines and the relations
;; %SIN-PPOLY = (REVERSE (for I from 0 as ENTRY in PS collect (/ (* (EXPT (/ 2 PI) (1+ (* 2 I))) ENTRY) Q0))) and %SIN-QPOLY = (REVERSE
;; (for I from 0 as ENTRY in QS collect (/ (* (EXPT (/ 2 PI) (* 2 I)) ENTRY) Q0)))
(MAKE-ARRAY 6 :ELEMENT-TYPE 'SINGLE-FLOAT :INITIAL-CONTENTS (LIST (%FLOAT 45236 25611))
```

(%FLOAT 13589 26148)
(%FLOAT 47286 34797)
(%FLOAT 15295 3306)
(%FLOAT 48666 34805)
(%FLOAT 16256 0))

(XCL:DEFGLOBALVAR %SIN-QPOLY

:: %SIN-PPOLY and %SIN-QPOLY contain adapted P and Q coefficients of Hart et al SIN 3374 rational approximation to (SIN X) in interval (0 (/ PI
:: 2)). The coefficients for %SIN-PPOLY and %SIN-QPOLY have been computed from Hart using extended precision routines and the relations
:: %SIN-PPOLY = (REVERSE (for I from 0 as ENTRY in PS collect (/ (\* (EXPT (/ 2 PI) (1+ (\* 2 I))) ENTRY) Q0))) and %SIN-QPOLY = (REVERSE
:: (for I from 0 as ENTRY in QS collect (/ (\* (EXPT (/ 2 PI) (\* 2 I)) ENTRY) Q0))) \*

(MAKE-ARRAY 6 :ELEMENT-TYPE 'SINGLE-FLOAT :INITIAL-CONTENTS (LIST (%FLOAT 11384 52865)
(%FLOAT 12553 9550)
(%FLOAT 13604 38385)
(%FLOAT 14593 18841)
(%FLOAT 15489 5549)
(%FLOAT 16256 0))))

(DEFUN %SIN-FLOAT (X COS-FLG)

:: SIN of a FLOAT X calculated via SIN 3374 rational approximation of Hart et al.

(LET ((THETA (FLOAT X))
(SIGN 1.0)
SIGN)

(DECLARE (TYPE FLOAT THETA SIGN))

:: If this function is called by COS then use (COS X) = (SIN (-- %PI/2 X)) = (SIN (+ %PI/2 X)). Case out on sign of X for improved numerical
:: stability. Avoids unnecessary rounding and promotes symmetric properties. (COS X) = (COS (-- X)) is guaranteed by this strategy.

(IF COS-FLG
(IF (IL:UFGREATERP THETA 0.0)
(SETQ THETA (- %PI/2 THETA))
(SETQ THETA (+ %PI/2 THETA))))

:: First range reduce to (0 infinity) by (SIN (minus X)) = (minus (SIN X)) This strategy guarantees (SIN (minus X)) = (minus (SIN X))

(WHEN (IL:UFLESSP THETA 0.0)
(SETQ SIGN -1.0)
(SETQ THETA (IL:UFMINUS THETA))))

:: Next range reduce to interval (0 %2PI) by (SIN X) = (SIN (MOD X %2PI))

(IF (IL:UFGEQ THETA %2PI)
(SETQ THETA (- THETA (\* %2PI (FLOAT (IL:UFIX (IL:FQUOTIENT THETA %2PI)))))))

:: Next range reduce to interval (0 PI) by (SIN (+ X PI)) = (minus (SIN X))

(WHEN (IL:UFGREATERP THETA PI)
(SETQ THETA (- THETA PI))
(SETQ SIGN (IL:UFMINUS SIGN))))

:: Next range reduce to interval (0 %PI/2) by (SIN (+ X %PI/2)) = (SIN (minus %PI/2 X))

(IF (IL:UFGREATERP THETA %PI/2)
(SETQ THETA (- PI THETA)))
(IF (IL:UFLESSP THETA %SIN-EPSILON)

:: If R is in the interval (0 %SIN-EPSILON) then (SIN R) = R to the precision that we can offer. Return R because (1) it is desirable that
:: (SIN R) = R exactly for small R and (2) microcode POLYEVAL will underflow on sufficiently small positive R

(SETQ THETA (\* SIGN THETA))

:: Now use SIN 3374 rational approximation of Harris et al. which works on interval (0 %PI/2)

(LET ((R2 (\* THETA THETA)))
(DECLARE (TYPE FLOAT R2))
(SETQ THETA (\* SIGN THETA (IL:FQUOTIENT (%POLYEVAL R2 %SIN-PPOLY 5)
(%POLYEVAL R2 %SIN-QPOLY 5))))))

(DEFUN SIN (RADIANS)

(TYPECASE RADIANS
(COMPLEX (LET ((X (COMPLEX-REALPART RADIANS))
(Y (COMPLEX-IMAGPART RADIANS)))
(COMPLEX (\* (SIN X)
(COSH Y))
(\* (COS X)
(SINH Y))))))
(NUMBER (%SIN-FLOAT RADIANS NIL))
(OTHERWISE (%NOT-NUMBER-ERROR RADIANS))))

(DEFUN COS (RADIANS)

(TYPECASE RADIANS
(COMPLEX (LET ((X (COMPLEX-REALPART RADIANS))
(Y (COMPLEX-IMAGPART RADIANS)))
(COMPLEX (\* (COS X)
(COSH Y))
(- (\* (SIN X)
(SINH Y))))))
(NUMBER (%SIN-FLOAT RADIANS T))

(OTHERWISE (%NOT-NUMBER-ERROR RADIANS))))

:: Tan

(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE

(DEFCONSTANT %TAN-EPSILON

:: %TAN-EPSILON is sufficiently small that (TAN X) = X for X in interval (0 %TAN-EPSILON). It suffices to take %TAN-EPSILON a little bit smaller than (SQRT (\* 3 SINGLE-FLOAT-EPSILON)) which we get by the Taylor series expansion (TAN X) = (+ X (/ (EXPT X 3) 3) ...) (The relative error caused by omitting (/ (EXPT X 3) 3) isn't observable.) Comparison against %TAN-EPSILON is used to avoid POLYEVAL microcode underflow when computing TAN. (%FLOAT 14720 0))

(XCL:DEFGLOBALVAR %TAN-PPOLY

:: %TAN-PPOLY and %TAN-QPOLY contain adapted P and Q coefficients of Hart et al TAN 4288 rational approximation to (TAN X) in interval (-PI/4 PI/4). The coefficients for %TAN-PPOLY and %TAN-QPOLY have been computed from Hart using extended precision routines and the relations %TAN-PPOLY = (REVERSE (for I from 0 as ENTRY in PS collect (/ (\* (EXPT (/ 4 PI) (1+ (\* 2 I))) ENTRY) Q0))) and %TAN-QPOLY = (REVERSE (for I from 0 as ENTRY in QS collect (/ (\* (EXPT (/ 4 PI) (\* 2 I)) ENTRY) Q0))) (MAKE-ARRAY 5 :ELEMENT-TYPE 'SINGLE-FLOAT :INITIAL-CONTENTS (LIST (%FLOAT 13237 21090) (%FLOAT 47141 15825) (%FLOAT 15246 8785) (%FLOAT 48655 48761) (%FLOAT 16256 0))))

(XCL:DEFGLOBALVAR %TAN-QPOLY

:: %TAN-PPOLY and %TAN-QPOLY contain adapted P and Q coefficients of Hart et al TAN 4288 rational approximation to (TAN X) in interval (-PI/4 PI/4). The coefficients for %TAN-PPOLY and %TAN-QPOLY have been computed from Hart using extended precision routines and the relations %TAN-PPOLY = (REVERSE (for I from 0 as ENTRY in PS collect (/ (\* (EXPT (/ 4 PI) (1+ (\* 2 I))) ENTRY) Q0))) and %TAN-QPOLY = (REVERSE (for I from 0 as ENTRY in QS collect (/ (\* (EXPT (/ 4 PI) (\* 2 I)) ENTRY) Q0))) (MAKE-ARRAY 6 :ELEMENT-TYPE 'SINGLE-FLOAT :INITIAL-CONTENTS (LIST (%FLOAT 45267 36947) (%FLOAT 13848 46875) (%FLOAT 47612 53738) (%FLOAT 15596 52854) (%FLOAT 48882 35303) (%FLOAT 16256 0))))

(DEFUN %TAN-FLOAT (X)

:: TAN of a FLOAT X calculated via TAN 4288 rational approximation of Hart et al. (LET ((FX (FLOAT X)) (SIGN 1.0) RECIPFLG) (DECLARE (TYPE FLOAT FX SIGN)) :: First range reduce to (0 infinity) by (TAN (minus X)) = (minus (TAN X)) (WHEN (IL:UFLESSP FX 0.0) (SETQ SIGN -1.0) (SETQ FX (IL:UFMINUS FX))) :: Next range reduce to (0 PI) (IF (IL:UFGEQ FX PI) (SETQ FX (- FX (\* PI (FLOAT (IL:UFIX (IL:FQUOTIENT FX PI))))))) :: Next, range reduce to (-PI/4 PI/4) using (TAN X) = (TAN (minus X PI)) to get into interval (-PI/2 PI/2) and then (TAN X) = (/ (TAN (minus PI/2 X))) to get into interval (-PI/4 PI/4) (COND ((IL:UFGREATERP FX %PI/2) (SETQ FX (- FX PI)) (WHEN (IL:UFLESSP FX %-PI/4) (SETQ RECIPFLG T) (SETQ FX (- %-PI/2 FX)))) (T (WHEN (IL:UFGREATERP FX %PI/4) (SETQ RECIPFLG T) (SETQ FX (- %PI/2 FX)))))) (COND ((IL:UFLESSP (IL:UFABS FX) %TAN-EPSILON) :: If R is in the interval (0 %TAN-EPSILON) then (TAN R) = R to the precision that we can offer. Return R because (1) it is desirable that (TAN R) = R exactly for small R and (2) microcode POLYEVAL will underflow on sufficiently small positive R. (SETQ FX (\* SIGN FX)) (IF RECIPFLG (SETQ FX (IL:FQUOTIENT 1.0 FX)) FX)) (T :: Now use TAN 4288 rational approximation of Hart et al. which works on interval (0 %PI/4) (LET ((R2 (\* FX FX)) (DECLARE (TYPE FLOAT R2)) (SETQ FX (\* SIGN FX (IL:FQUOTIENT (%POLYEVAL R2 %TAN-PPOLY 4)

```

                                (%POLYEVAL R2 %TAN-QPOLY 5)))
(IF RECIPFLG
 (SETQ FX (IL:FQUOTIENT 1.0 FX))
 FX))))))

```

```

(DEFUN TAN (RADIANS)
 (TYPECASE RADIANS
 (COMPLEX (LET* ((X (* 2.0 (COMPLEX-REALPART RADIANS)))
 (Y (* 2.0 (COMPLEX-IMAGPART RADIANS)))
 (DENOM (+ (COS X)
 (COSH Y))))
 (COMPLEX (IL:QUOTIENT (SIN X)
 DENOM)
 (IL:QUOTIENT (SINH Y)
 DENOM))))
 (NUMBER (%TAN-FLOAT RADIANS))
 (OTHERWISE (%NOT-NUMBER-ERROR RADIANS))))

```

:: Asin and Acos

```
(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE
```

(DEFCONSTANT %ASIN-EPSILON

:: %ASIN-EPSILON is sufficiently small that (ASIN X) = X for X in interval (0 %ASIN-EPSILON). It suffices to take %ASIN-EPSILON a little bit smaller than (\* 2 SINGLE-FLOAT-EPSILON) which we get by the Taylor series expansion (ASIN X) = (+ X (/ (EXPT X 3) 6) ...) (The relative error caused by omitting (/ (EXPT X 3) 6) isn't observable.) Comparison against %ASIN-EPSILON is used to avoid POLYEVAL microcode underflow when computing SIN.

```
(%FLOAT 14720 0))
```

(XCL:DEFGLOBALVAR %ASIN-PPOLY

:: %ASIN-PPOLY and %ASIN-QPOLY contain P and Q coefficients of Hart et al ARCSN 4671 rational approximation to (ASIN X) in interval (0 (SQRT .5)).

```
(MAKE-ARRAY 7 :ELEMENT-TYPE 'SINGLE-FLOAT :INITIAL-CONTENTS (LIST (%FLOAT 16007 50045)
(%FLOAT 49549 8020)
(%FLOAT 17236 15848)
(%FLOAT 50285 63464)
(%FLOAT 17650 31235)
(%FLOAT 50403 62852)
(%FLOAT 17440 39471))))

```

(XCL:DEFGLOBALVAR %ASIN-QPOLY

:: %ASIN-PPOLY and %ASIN-QPOLY contain P and Q coefficients of Hart et al ARCSN 4671 rational approximation to (ASIN X) in interval (0 (SQRT .5)).

```
(MAKE-ARRAY 7 :ELEMENT-TYPE 'SINGLE-FLOAT :INITIAL-CONTENTS (LIST (%FLOAT 16256 0)
(%FLOAT 49672 25817)
(%FLOAT 17308 55260)
(%FLOAT 50326 38098)
(%FLOAT 17674 22210)
(%FLOAT 50417 22451)
(%FLOAT 17440 39471))))

```

(DEFUN %ASIN-FLOAT (X ACOS-FLG)

:: (ASIN X) for float X calculated via ARCSN 4671 rational approximation of Hart et al.

```
(IF (OR (< X -1.0)
(> X 1.0))
(ERROR "Arg not in range: ~s" X))
(LET ((FX (FLOAT X))
NEGATIVE REDUCED)
(DECLARE (TYPE FLOAT FX))

```

:: Range reduce to (0 1) via identity (ASIN (minus X)) = (minus (ASIN X))

```
(WHEN (IL:UFLESSP FX 0.0)
 (SETQ NEGATIVE T)
 (SETQ FX (IL:UFMINUS FX)))

```

:: Range reduce to (0 0.5) via identity (ASIN X) = (minus %PI/2 (\* 2.0 (ASIN (SQRT (\* 0.5 (minus 1.0 R)))))) Avoids numerical instability calculating (ASIN X) for X near one. SIN is horizontally flat near %PI/2 so calculating (ASIN X) by rational approximation wouldn't work well for X near (SIN %PI/2) = 1

```
(WHEN (IL:UFGREATERP FX 0.5)
 (SETQ REDUCED T)
 (SETQ FX (SQRT (SETQ FX (* 0.5 (- 1.0 FX))))))

```

:: R is now in range (0 0.5) Use ARCSN 4671 rational approximation to calculate (ASIN R)

```
(IF (IL:UFGREATERP FX %ASIN-EPSILON)

```

:: If R is in the interval (0 %SIN-EPSILON) then (ASIN R) = R to the precision that we can offer.

```
(LET ((R2 (* FX FX))

```

```
(DECLARE (TYPE FLOAT R2))
(SETQ FX (* FX (IL:QUOTIENT (%POLYEVAL R2 %ASIN-PPOLY 6)
(%POLYEVAL R2 %ASIN-QPOLY 6))))
NIL))
(IF REDUCED
 (SETQ FX (- %PI/2 (* 2.0 FX))))
(IF NEGATIVE
 (SETQ FX (IL:UFMINUS FX)))
;; In case we want (ACOS X) then use identity (ACOS X) = (minus %PI/2 (ASIN X))
(IF ACOS-FLG
 (SETQ FX (- %PI/2 FX)))
FX))
```

```
(DEFUN ASIN (NUMBER)
 (TYPECASE NUMBER
 (COMPLEX (LET ((Z (LOG (+ (COMPLEX (- (COMPLEX-IMAGPART NUMBER))
(COMPLEX-REALPART NUMBER))
(SQRT (- 1 (* NUMBER NUMBER))))))
 (COMPLEX (COMPLEX-IMAGPART Z)
(- (COMPLEX-REALPART Z))))))
 (NUMBER (%ASIN-FLOAT NUMBER NIL))
 (OTHERWISE (%NOT-NUMBER-ERROR NUMBER))))
```

```
(DEFUN ACOS (RADIANS)
 (TYPECASE RADIANS
 (COMPLEX (LET ((Z (SQRT (- 1 (* RADIANS RADIANS))))
 (SETQ Z (LOG (+ RADIANS (COMPLEX (- (COMPLEX-IMAGPART Z))
(COMPLEX-REALPART Z))))))
 (COMPLEX (COMPLEX-IMAGPART Z)
(- (COMPLEX-REALPART Z))))))
 (NUMBER (%ASIN-FLOAT RADIANS T))
 (OTHERWISE (%NOT-NUMBER-ERROR RADIANS))))
```

;; Atan

```
(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE
```

```
(DEFCONSTANT %SQRT3 (%FLOAT 16349 46039))
```

```
(DEFCONSTANT %2-SQRT3 (%FLOAT 16009 12451))
```

```
(DEFCONSTANT %INV-2-SQRT3 (%FLOAT 16494 55788))
)
```

```
(DEFUN %ATAN-FLOAT (Y &OPTIONAL X)
 (LET ((FY (FLOAT Y))
 FX FARG)
 (DECLARE (TYPE FLOAT FY FX FARG))
 ;; Compute farg
 (COND
 ((NULL X)
 (IF (= Y 0.0)
 (RETURN-FROM %ATAN-FLOAT 0.0)
 (SETQ FARG FY)))
 (T ;; Don't use unboxed version of =, because it doesn't return t on comparison of 0.0 and -0.0
 (SETQ FX (FLOAT X))
 (COND
 ((= X 0.0)
 (IF (= Y 0.0)
 (ERROR "Both args to atan are 0.0")
 (RETURN-FROM %ATAN-FLOAT (IF (> Y 0.0)
%PI/2
(- %PI/2))))))
 ((= Y 0.0)
 (RETURN-FROM %ATAN-FLOAT (IF (> X 0.0)
0.0
PI)))
 ((> Y 0.0)
 (IF (> X 0.0)
 (SETQ FARG (IL:FQUOTIENT FY FX))
 (SETQ FARG (IL:FQUOTIENT (IL:UFMINUS FY)
FX))))
 ((> X 0.0)
 (SETQ FARG (IL:FQUOTIENT FY (IL:UFMINUS FX))))
 (T (SETQ FARG (IL:FQUOTIENT FY FX))))))
 ;; Compute result
 (LET ((CONSTANT 0.0)
```

```

(CONSTANT-FLAG T)
NEGATE-FLAG ADD-FLAG)
(DECLARE (TYPE FLOAT CONSTANT))
;; (ATAN (minus X)) = (minus (ATAN X))
(WHEN (IL:UFLESSP FARG 0.0)
  (SETQ NEGATE-FLAG T)
  (SETQ FARG (IL:UFMINUS FARG)))
;; Range reduce to (0, 2-sqrt(3))
(COND
  ((IL:UFGEQ FARG %INV-2-SQRT3)
   ;; (ATAN X) = (minus %PI/2 (ATAN (/ X)))
   (SETQ CONSTANT %PI/2)
   (SETQ FARG (IL:FQUOTIENT 1.0 FARG)))
  ((IL:UFGEQ FARG 1.0)
   (SETQ CONSTANT %PI/3)
   (SETQ FARG (IL:FQUOTIENT (- %SQRT3 FARG)
                             (+ 1.0 (* FARG %SQRT3)))))
  ((IL:UFGEQ FARG %2-SQRT3)
   (SETQ ADD-FLAG T)
   (SETQ CONSTANT %PI/6)
   (SETQ FARG (IL:FQUOTIENT (- (* FARG %SQRT3)
                               1.0)
                             (+ %SQRT3 FARG)))))
  (T (SETQ CONSTANT-FLAG NIL)))
;; Power series expansion cons'ed up on the fly
(LET ((SQR (IL:UFMINUS (* FARG FARG)))
      (INT 1.0)
      (POW FARG)
      (OLD 0.0))
  (DECLARE (TYPE FLOAT SQR INT POW OLD))
  (LOOP (IF (IL:UFEQP FARG OLD)
            (RETURN NIL))
        (SETQ INT (+ INT 2.0))
        (SETQ POW (* POW SQR))
        (SETQ OLD FARG)
        (SETQ FARG (+ FARG (IL:FQUOTIENT POW INT)))))
  (IF CONSTANT-FLAG
    (IF ADD-FLAG
      (SETQ FARG (+ CONSTANT FARG))
      (SETQ FARG (- CONSTANT FARG))))
  (IF NEGATE-FLAG
    (SETQ FARG (IL:UFMINUS FARG))))
;; Fix up
(IF X
  (COND
    ((IL:UFGREATERP FY 0.0)
     (IF (IL:UFLESSP FX 0.0)
         (SETQ FARG (- PI FARG))))
    ((IL:UFGREATERP FX 0.0)
     (SETQ FARG (IL:UFMINUS FARG)))
    (T (SETQ FARG (- FARG PI)))))
;; Box and return
FARG))

```

```

(DEFUN ATAN (Y &OPTIONAL X)
  (COND
    (X (%ATAN-FLOAT (FLOAT Y)
                    (FLOAT X)))
    ((COMPLEXP Y)
     (LET ((R (COMPLEX-REALPART Y))
           (I (COMPLEX-IMAGPART Y)))
       (IF (NOT (AND (ZEROP R)
                     (= (ABS I)
                        1)))
           (LET ((Z (COMPLEX (- I)
                              R)))
               (SETQ Z (* 0.5 (LOG (/ (+ 1 Z)
                                       (- 1 Z)))))
               (COMPLEX (COMPLEX-IMAGPART Z)
                        (- (COMPLEX-REALPART Z))))
           (ERROR "Argument not in domain for atan. ~S" Y))))
    (T (%ATAN-FLOAT Y))))

```

;; Cis (exp (i x))

```

(DEFUN CIS (RADIANS)
  (IF (TYPEP RADIANS ' (AND NUMBER (NOT COMPLEX)))
      (COMPLEX (%SIN-FLOAT RADIANS T)
               (%SIN-FLOAT RADIANS))

```

(%NOT-NONCOMPLEX-NUMBER-ERROR RADIANS))

:: Sinh, Cosh Tanh

```
(DEFUN SINH (NUMBER)
  ;; Computed directly from its
  (IF (COMPLEXP NUMBER)
    (LET ((Z (EXP NUMBER)))
      (/ (- Z (/ Z)
            2))
      (LET ((FZ (%EXP-FLOAT NUMBER)))
        (DECLARE (TYPE FLOAT FZ))
        (SETQ FZ (IL:FQUOTIENT (- FZ (IL:FQUOTIENT 1.0 FZ))
                               2.0))))))
```

```
(DEFUN COSH (NUMBER)
  (IF (COMPLEXP NUMBER)
    (LET ((Z (EXP NUMBER)))
      (/ (+ Z (/ Z)
            2))
      (LET ((FZ (%EXP-FLOAT NUMBER)))
        (DECLARE (TYPE FLOAT FZ))
        (SETQ FZ (IL:FQUOTIENT (+ FZ (IL:FQUOTIENT 1.0 FZ))
                               2.0))))))
```

```
(DEFUN TANH (NUMBER)
  (IF (COMPLEXP NUMBER)
    (/ (SINH NUMBER)
      (COSH NUMBER))
    (LET* ((FX (%EXP-FLOAT (* 2 NUMBER)))
           (FY (IL:FQUOTIENT 1.0 FX)))
      (DECLARE (TYPE FLOAT FX FY))
      (SETQ FX (- (IL:FQUOTIENT 1.0 (+ 1.0 FY))
                  (IL:FQUOTIENT 1.0 (+ 1.0 FX)))))))
```

:: Asinh Acosh Atanh

```
(DEFUN ASINH (NUMBER)
  (IF (COMPLEXP NUMBER)
    (LOG (+ NUMBER (SQRT (+ (* NUMBER NUMBER)
                              1))))
    (LET ((FX (FLOAT NUMBER))
          (BOX)
          (DECLARE (TYPE FLOAT FX BOX))
          (LOG (SETQ BOX (+ FX (SQRT (SETQ BOX (+ (* FX FX)
                                                    1.0))))))))))
```

```
(DEFUN ACOSH (NUMBER)
  (IF (OR (COMPLEXP NUMBER)
          (< NUMBER 1))
    (LOG (+ NUMBER (* (+ NUMBER 1)
                       (SQRT (/ (- NUMBER 1)
                                   (+ NUMBER 1))))))
    (LET ((FX (FLOAT NUMBER))
          (BOX)
          (DECLARE (TYPE FLOAT FX BOX))
          (LOG (SETQ BOX (+ FX (SQRT (SETQ BOX (- (* FX FX)
                                                    1.0))))))))))
```

```
(DEFUN ATANH (NUMBER)
  (IF (OR (COMPLEXP NUMBER)
          (> (ABS NUMBER)
             1))
    (IF (AND (ZEROP (IMAGPART NUMBER))
              (= (ABS (REALPART NUMBER))
                 1))
      (ERROR "Argument out of range. ~s" NUMBER)
      (* 0.5 (LOG (/ (+ 1 NUMBER)
                      (- 1 NUMBER)))))
    (IF (= NUMBER 1.0)
      (ERROR "Argument out of range. ~s" NUMBER)
      (LET ((FX (FLOAT NUMBER))
            (BOX)
            (DECLARE (TYPE FLOAT FX BOX))
            (SETQ BOX (* 0.5 (LOG (SETQ BOX (IL:FQUOTIENT (+ 1.0 FX)
                                                            (- 1.0 FX))))))))))
```

:: rational and rationalize

```

(DEFUN %RATIONAL-FLOAT (NUMBER)
  (IF (= NUMBER 0.0)
    0
    (LET (SIGN EXP HI LO MANT)
      (%FLOAT-UNBOX NUMBER SIGN EXP HI LO T)
      (SETQ MANT (+ (ASH HI 16)
                    LO))
      (IF (EQ SIGN 1)
          (SETQ MANT (- MANT)))
      (SETQ EXP (- EXP 23 IL:\EXONENT.BIAS))
      (IF (< EXP 0)
          (%BUILD-RATIO MANT (ASH 1 (- EXP)))
          (ASH MANT EXP))))))

```

```

(DEFUN %RATIONALIZE-FLOAT (X)
  ;; Produce a rational approximating X.
  ;; This routine presupposes familiarity with topics in number theory and IEEE FLOATP representation. The algorithm uses a standard mathematical
  ;; technique for approximating a real valued number, but in very sophisticated form more amenable to the computer and the nature of IEEE
  ;; FLOATPs and is not an algorithm you are likely to find published anywhere.

```

(IF (= X 0.0) ; In case X = 0, just return 0

```

0
(LET (SIGN EXPT HI LO XNUM XDEN R)
  ;; First of all, X is range reduced to the interval ((SQRT .5) (SQRT 2)) excluding (SQRT 2) This strategy has the property that FLOATPs
  ;; differing only by sign and a power of two rationalize into rationals differing only by sign and a power of two. The choice of interval
  ;; ((SQRT .5) (SQRT 2)) versus another interval such as (.5 1) is due to our wanting there to be roughly the same number of significant
  ;; bits in the numerator as in the denominator of the answer that is returned. Here, significant bits is taken to mean the number of bits in
  ;; the results returned by the continued fraction approximation and excludes the bits resulting from multiplying by the power of two.
  ;; Get SIGN XNUM XDEN and EXPT for X.

```

```

(LET (BIT-SIGN EXP HI LO)
  (%FLOAT-UNBOX X BIT-SIGN EXP HI LO T)
  (SETQ XNUM (+ (ASH HI 16)
                LO))
  (SETQ EXPT (- EXP (+ IL:\EXONENT.BIAS 23)))
  (SETQ SIGN (IF (EQ BIT-SIGN 0)
                 1
                 -1)) ; Compute r
  (LOOP (IF (NOT (EQ 0 (LOGAND HI IL:\HIDDENBIT))) ; Handle the denormalized case
          (RETURN NIL))
        (IL:.LLSH1. HI LO))
  (IL:.LLSH8. HI LO)
  (SETQ R (IL:\MAKEFLOAT 0 (1- IL:\EXONENT.BIAS)
                    HI LO)) ; 24 because FLOATPs have 24 bit mantissas.
  (SETQ XDEN (IL:CONSTANT (ASH 1 24)))
  (SETQ EXPT (+ EXPT 24))
  (COND
    ((< XNUM 11863283) ; 11863283 = (SQRT 0.5) mantissa.
     (SETQ XDEN (ASH XDEN -1))
     (SETQ EXPT (1- EXPT))
     (SETQ R (* 2 R))))

```

;; At this point, X = (\* (/ XNUM XDEN) (EXPT 2 EXPT)) and (/ XNUM XDEN) is in the interval ((SQRT 0.5) (SQRT 2))

```

(LET ((OLDNUM 1)
      (OLDDEN 0)
      (NUM 0)
      (DEN 1)) ; Continued fraction approximation loop.
  (LOOP (COND
        ((AND (NOT (EQ DEN 0))
              (= (IL:FQUOTIENT NUM DEN)
                 R))
         (COND
          ((> EXPT 0)
           (SETQ NUM (ASH NUM EXPT)))
          ((< EXPT 0)
           (SETQ DEN (ASH DEN (- EXPT))))
          (RETURN (/ (* SIGN NUM)
                    DEN))))
        (ROTATEF XNUM XDEN)
        (LET ((TRUNC (IL:IQUOTIENT XNUM XDEN)))
          (SETQ NUM (+ OLDNUM (* TRUNC (SETQ OLDNUM NUM))))
          (SETQ DEN (+ OLDDEN (* TRUNC (SETQ OLDDEN DEN))))
          (SETQ XNUM (- XNUM (* XDEN TRUNC))))))))

```

```

(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE
(IL:DECLARE\ : IL:DOEVAL@COMPILE IL:DONTCOPY
(IL:LOCALVARS . T)
)
)
(IL:PUTPROPS IL:CMLFLOAT IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "LISP"))
(IL:PUTPROPS IL:CMLFLOAT IL:FILETYPE COMPILE-FILE)

```



{MEDLEY}<sources>CMLFLOAT.;1

(IL:PUTPROPS **IL:CMLFLOAT IL:COPYRIGHT** ("Venue & Xerox Corporation" 1986 1987 1988 1990))

FUNCTION INDEX

%ASIN-FLOAT .....	12	%SQRT-COMPLEX .....	9	COS .....	10	LOG .....	8
%ATAN-FLOAT .....	13	%SQRT-FLOAT .....	9	COSH .....	15	SCALE-FLOAT .....	5
%EXP-FLOAT .....	6	%TAN-FLOAT .....	11	DECODE-FLOAT .....	5	SIN .....	10
%EXPT-FLOAT-INTEG	7	ACOS .....	13	EXP .....	6	SINH .....	15
%EXPT-INTEG	7	ACOSH .....	15	EXPT .....	7	SQRT .....	9
%FLOAT .....	2	ASIN .....	13	FLOAT-DIGITS .....	5	TAN .....	12
%LOG-FLOAT .....	8	ASINH .....	15	FLOAT-PRECISION .....	5	TANH .....	15
%RATIONAL-FLOAT .....	16	ATAN .....	14	FLOAT-RADIX .....	5		
%RATIONALIZE-FLOAT .....	16	ATANH .....	15	FLOAT-SIGN .....	5		
%SIN-FLOAT .....	10	CIS .....	14	INTEGER-DECODE-FLOAT .....	6		

CONSTANT INDEX

%-PI/2 .....	4	%SIN-EPSILON .....	9	MOST-NEGATIVE-DOUBLE-FLOAT .....	3
%-PI/4 .....	4	%SQRT2 .....	8	MOST-NEGATIVE-FIXNUM .....	2
%2-SQRT3 .....	13	%SQRT3 .....	13	MOST-NEGATIVE-LONG-FLOAT .....	3
%2/PI .....	4	%TAN-EPSILON .....	11	MOST-NEGATIVE-SHORT-FLOAT .....	3
%2PI .....	3	DOUBLE-FLOAT-EPSILON .....	3	MOST-NEGATIVE-SINGLE-FLOAT .....	2
%2PI/3 .....	3	DOUBLE-FLOAT-NEGATIVE-EPSILON .....	3	MOST-POSITIVE-DOUBLE-FLOAT .....	3
%ASIN-EPSILON .....	12	LEAST-NEGATIVE-DOUBLE-FLOAT .....	3	MOST-POSITIVE-FIXNUM .....	2
%E .....	3	LEAST-NEGATIVE-LONG-FLOAT .....	3	MOST-POSITIVE-LONG-FLOAT .....	3
%INV-2-SQRT3 .....	13	LEAST-NEGATIVE-SHORT-FLOAT .....	3	MOST-POSITIVE-SHORT-FLOAT .....	3
%LOG-BASE2-E .....	6	LEAST-NEGATIVE-SINGLE-FLOAT .....	2	MOST-POSITIVE-SINGLE-FLOAT .....	2
%LOG2 .....	8	LEAST-POSITIVE-DOUBLE-FLOAT .....	3	PI .....	3
%PI .....	3	LEAST-POSITIVE-LONG-FLOAT .....	3	SHORT-FLOAT-EPSILON .....	3
%PI/2 .....	4	LEAST-POSITIVE-SHORT-FLOAT .....	3	SHORT-FLOAT-NEGATIVE-EPSILON .....	3
%PI/3 .....	4	LEAST-POSITIVE-SINGLE-FLOAT .....	2	SINGLE-FLOAT-EPSILON .....	3
%PI/4 .....	4	LONG-FLOAT-EPSILON .....	3	SINGLE-FLOAT-NEGATIVE-EPSILON .....	3
%PI/6 .....	4	LONG-FLOAT-NEGATIVE-EPSILON .....	3		

VARIABLE INDEX

%ASIN-PPOLY .....	12	%EXP-POLY .....	6	%LOG-PPOLY .....	8	%SIN-PPOLY .....	9	%TAN-PPOLY .....	11
%ASIN-QPOLY .....	12	%EXP-TABLE .....	6	%LOG-QPOLY .....	8	%SIN-QPOLY .....	10	%TAN-QPOLY .....	11

MACRO INDEX

%FLOAT-UNBOX .....	4	%GET-TABLE-ENTRY ..	4	%POLYEVAL .....	4	%UFTRUNCATE .....	4	%UMAKE-FLOAT .....	4
--------------------	---	---------------------	---	-----------------	---	-------------------	---	--------------------	---

PROPERTY INDEX

IL:CMLFLOAT .....	16
-------------------	----