File created: 18-Oct-93 16:17:34 **{Pele:mv:envos}<LispCore>Sources>CLTL2>LLFLOAT.;2**

previous date: 3-Sep-91 18:05:23 {Pele:mv:envos}<LispCore>Sources>CLTL2>LLFLOAT.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

```
;;
```


```
(RPAQQ LLFLOATCOMS
       [(DECLARE%: DONTCOPY (MACROS \HAND.FLOATUNBOX)
              (EXPORT (MACROS POLYEVAL)))
        (COMS (FNS \PUTBASEFLOATP \GETBASEFLOATP)
              (MACROS \PUTBASEFLOATP \GETBASEFLOATP              ; the following deal with raw 32 bit numbers
                     \.PUTBASE32 \.GETBASE32))
        [COMS (FNS FTIMES FPLUS FQUOTIENT FDIFFERENCE FGREATERP FABS)
                                                                 ; UFNs
              (FNS \SLOWFDIFFERENCE \SLOWFPLUS2 \SLOWFTIMES2 \SLOWFQUOTIENT \SLOWFGREATERP)
              ;; Float and \float changed to coerce ratios.
              (FUNCTIONS FLOAT)
              (FNS \FZEROP FEQP \FLOAT \FIXP.FROM.FLOATP FIXR \BOXFPLUSDIF \BOXFQUOTIENT \BOXFTIMES2 \INFINITY
                   \MAKEFLOAT MAKEFLOATNUMBER PutFloat)
              (PROP DMACRO ZEROP)
              (FNS SQRT)
              (DECLARE%: EVAL@COMPILE DONTCOPY (EXPORT (RECORDS FLOATP)
                                                       (CONSTANTS (MAX.DIGITS.ACCURACY 9)))
                     (CONSTANTS (\8BITS 255)
                            (\MAX.HI.FRAC 127)
                            (\SIGNBIT 32768)
                            (\EXPONENT.BIAS 127)
                            (\HIDDENBIT 128)
                            (\MAX.EXPONENT 255))
                     (MACROS .FLOATUNBOX. .LLSH1. .LLSH8. .LRSH1. .LRSH8. .LRSHSTICKY. .ADDSMALL2. .ADDSMALL3.
                            .SUBSMALL. .POWEROF2.)
                     (LOCALVARS . T))
              (DECLARE%: DONTEVAL@LOAD DOCOPY (VARS (\UNDERFLOW)
                                                    (MAX.FLOAT (\INFINITY 0))
                                                    (MIN.FLOAT (\INFINITY 1)))
                     (P (MOVD? 'FGREATERP 'FGTP]
        [COMS ;; unboxed ufns
              (FNS \UNBOXFLOAT1 \UNBOXFLOAT2 \UNBOXFLOAT3)
              (FNS \MATMULT133 \MATMULT144 \MATMULT331 \MATMULT333 \MATMULT441 \MATMULT444)
                                                                 ; unboxed arg handling
              (DECLARE%: DONTCOPY (EXPORT (MACROS \CALLER.ARGS]
        (COMS (FNS FLOATP.TO.BCPL BCPL.TO.FLOATP)
              (DECLARE%: EVAL@COMPILE DONTCOPY (RECORDS BCPLNUM EFPN)))
        [COMS (VARIABLES INTPOWERS)
              (FUNCTIONS ENUM-STRING FNUM-STRING FLTSTR FLTINTLOG DIGITSBDP INTTOEXT EXTTOINT SPLIT8 TIMESPOW10
                     \EXTFTIMES \EXTFQUOTIENT \EXTNORMALIZE \CONVERT.FLOATING.NUMBER \FLOATINGSCALE)
              (FNS \INIT.POWERS.OF.TEN)
              (DECLARE%: DONTCOPY (RESOURCES \CFNSTRING)
                     (GLOBALVARS \POWERS.OF.TEN)
                     (MACROS \POWER.OF.TEN))
              (DECLARE%: DONTEVAL@LOAD DOCOPY (INITRESOURCES \CFNSTRING)
                     (P (\INIT.POWERS.OF.TEN]
        (PROP ARGNAMES \UNBOXFLOAT1 \UNBOXFLOAT2 \UNBOXFLOAT3)
        (PROP FILETYPE LLFLOAT)
        (DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS (ADDVARS (NLAMA)
                                                                              (NLAML)
                                                                              (LAMA FPLUS FTIMES])

(DECLARE%: DONTCOPY

(DECLARE%: EVAL@COMPILE

(PUTPROPS \HAND.FLOATUNBOX MACRO [LAMBDA (X)
                                        ;; this doesn't call \FLOATUNBOX because it's used by the UFN case of \FLOATUNBOX.  Takes a
                                        ;; FLOATP and returns the raw unboxed bits of the value.  Must be used with great caution as raw
                                        ;; unboxed bits are not allowed in many places.
                                        (\VAG2 (fetch (FLOATP HIWORD) of (SETQ X (FLOAT X)))
                                               (fetch (FLOATP LOWORD) of X])
)

;; FOLLOWING DEFINITIONS EXPORTED

(DECLARE%: EVAL@COMPILE

(PUTPROPS POLYEVAL DMACRO ((X COEFFS DEGREE)                     ; execute the POLYEVAL opcode on the value X, the array
                                                                 ; COEFFS with degree DEGREE
```

```
                              (\FLOATBOX ((OPCODES UBFLOAT3 0)
                                          (\FLOATUNBOX X)
                                          (fetch (ARRAYP BASE) of COEFFS)
                                          DEGREE))))
)
)

;; END EXPORTED DEFINITIONS

(DEFINEQ

(\PUTBASEFLOATP
  [LAMBDA (BASE OFFST VAL)                                    (* Pavel " 6-Oct-86 21:52")
    ;; put the floatp VAL at offset OFFST from BASE.  Used by REPLACEFIELD of floatp fields
    (\FLOATBOX (\.PUTBASE32 BASE OFFST (\FLOATUNBOX VAL])

(\GETBASEFLOATP
  [LAMBDA (BASE OFFST)                                        (* Pavel " 6-Oct-86 21:52")
    ;; get the floatp at OFFST from BASE
    (\FLOATBOX (\.GETBASE32 BASE OFFST])

)

(DECLARE%: EVAL@COMPILE

(PUTPROPS \PUTBASEFLOATP DMACRO [(BASE OFFST VAL)            ; put the floatp VAL at offset OFFST from BASE.  Used by
                                                             ; REPLACEFIELD of floatp fields
                                 (\FLOATBOX (\.PUTBASE32 BASE OFFST (\FLOATUNBOX VAL])

(PUTPROPS \GETBASEFLOATP DMACRO ((BASE OFFST)                ; get the floatp at OFFST from BASE
                                 (\FLOATBOX (\.GETBASE32 BASE OFFST))))

(PUTPROPS \.PUTBASE32 DMACRO (= . \PUTBASEPTR))

(PUTPROPS \.GETBASE32 DMACRO (APPLY* COMP.GETBASE NIL GETBASE.32))
)

(DEFINEQ

(FTIMES
  [LAMBDA N                                                   (* JonL "17-May-84 18:35")
    (PROG (R (J 1))
          [COND
             ((EQ 0 N)
              (RETURN 1.0))
             ((EQ N 1)
              (RETURN (FLOAT (ARG N 1]
          (SETQ R (ARG N 1))
      LP  (COND
             ((NEQ J N)
              (SETQ J (ADD1 J))                              ; assumes that FTIMES compiles into opcode that punts into
                                                             ; \FTIMES.UFN
              (SETQ R (FTIMES R (ARG N J)))
              (GO LP)))
          (RETURN R])

(FPLUS
  [LAMBDA N                                                   (* JonL "17-May-84 18:35")
    (PROG (R (J 1))
          [COND
             ((EQ 0 N)
              (RETURN 0.0))
             ((EQ N 1)
              (RETURN (FLOAT (ARG N 1]
          (SETQ R (ARG N 1))
      LP  (COND
             ((NEQ J N)
              (SETQ J (ADD1 J))
              (SETQ R (FPLUS R (ARG N J)))
              (GO LP)))
          (RETURN R])

(FQUOTIENT
  [LAMBDA (X Y)                                               (* lmm "11-FEB-82 14:02")
    ((OPCODES FQUOTIENT)
     X Y])

(FDIFFERENCE
  [LAMBDA (X Y)                                               (* lmm "14-MAR-84 22:20")
    ((OPCODES FDIFFERENCE)
     X Y])
```

```
(FGREATERP
  [LAMBDA (X Y)                                                    (* lmm "17-Oct-84 15:45")
    ;; to compare two floats, compare signbits, and if they are equal compare the remaining 31 bits of each number as unsigned integers
    ((OPCODES FGREATERP)
     X Y])

(FABS
  [LAMBDA (X)                                                      (* Pavel " 6-Oct-86 21:53")
    (\FLOATBOX ((OPCODES UBFLOAT1 2)
                (\FLOATUNBOX X])

)

;; UFNs

(DEFINEQ

(\SLOWFDIFFERENCE
  [LAMBDA (X Y)                                                    (* lmm "17-Oct-84 15:42")
    (\CALLME 'FDIFFERENCE)
    (\BOXFPLUSDIF X Y T])

(\SLOWFPLUS2
  [LAMBDA (X Y)                                                    (* lmm "17-Oct-84 15:42")
                                                                   ; UFN for FPLUS
    (\CALLME 'FPLUS)
    (\BOXFPLUSDIF X Y])

(\SLOWFTIMES2
  [LAMBDA (X Y)                                                    (* lmm "17-Oct-84 15:43")
    (\CALLME 'FTIMES)
    (\BOXFTIMES2 X Y])

(\SLOWFQUOTIENT
  [LAMBDA (X Y)                                                    (* lmm "17-Oct-84 15:43")
    (\CALLME 'FQUOTIENT)                                           ; UFN for FQUOTIENT
    (\BOXFQUOTIENT X Y NIL])

(\SLOWFGREATERP
  [LAMBDA (X Y)                                                    (* JonL "17-May-84 18:34")
    ;; to compare two floats, compare signbits, and if they are equal compare the remaining 31 bits of each number as unsigned integers
    (COND
       [(AND (FLOATP X)
             (FLOATP Y))
        ;; Can speed this up by not unpacking--check signs, then compare remaining 31d bits as unsigned numbers
        (PROG ((HX (fetch (FLOATP HIWORD) of X))
               (HY (fetch (FLOATP HIWORD) of Y))
               SIGNX)
              (RETURN (COND
                         ((NEQ (SETQ SIGNX (LOGAND HX \SIGNBIT))
                               (LOGAND HY \SIGNBIT))
                          (EQ 0 SIGNX))
                         [(EQ 0 SIGNX)                             ; numbers are positive
                          (OR (IGREATERP HX HY)
                              (AND (EQ HX HY)
                                   (IGREATERP (fetch LOWORD of X)
                                              (fetch LOWORD of Y]
                         (T                                        ; Numbers are negative, so compare in other direction
                            (OR (IGREATERP HY HX)
                                (AND (EQ HX HY)
                                     (IGREATERP (fetch LOWORD of Y)
                                                (fetch LOWORD of X]
       (T (PROG (HX LX SIGNX EXPX HY LY SIGNY EXPY)
                (.FLOATUNBOX. X SIGNX EXPX HX LX)
                (.FLOATUNBOX. Y SIGNY EXPY HY LY)
                (RETURN (COND
                           ((NEQ SIGNX SIGNY)
                            (EQ 0 SIGNX))
                           [(EQ 0 SIGNX)                         ; numbers are positive
                            (OR (IGREATERP EXPX EXPY)
                                (AND (EQ EXPX EXPY)
                                     (OR (IGREATERP HX HY)
                                         (AND (EQ HX HY)
                                              (IGREATERP LX LY]
                           (T                                    ; Numbers are negative, so compare in other direction
                              (OR (IGREATERP EXPY EXPX)
                                  (AND (EQ EXPY EXPX)
```

```
                                    (OR (IGREATERP HY HX)
                                        (AND (EQ HY HX)
                                             (IGREATERP LY LX])

)

;; Float and \float changed to coerce ratios.

(CL:DEFUN FLOAT (CL:NUMBER &OPTIONAL CL::OTHER)
   (\DTEST CL:NUMBER 'FLOATP))

(DEFINEQ
```

(\FZEROP

```
  [LAMBDA (X)                                                    (* JonL "27-Sep-84 22:20")
    ;; Support for generic ZEROP macro
    (AND [EQ 0 (fetch LOWORD of (\DTEST X 'FLOATP]
         (EQ 0 (fetch HIWORDNOSIGNBIT of X])
```

(FEQP

```
  [LAMBDA (X Y)                                                  (* JonL "17-May-84 20:26")
    (COND
       [(AND (FLOATP X)
             (FLOATP Y))
        ;; If they're both floatp already, can essentially compare contents, since floatps are generally normalized.
                                                                 ; Last OR clause is to check for comparing a negative zero to a
                                                                 ; positive zero.
        (AND (EQ (fetch LOWORD of X)
                 (fetch LOWORD of Y))
             (OR (EQ (fetch HIWORD of X)
                     (fetch HIWORD of Y))
                 (AND (EQ 0 (fetch HIWORDNOSIGNBIT of X))
                      (EQ 0 (fetch HIWORDNOSIGNBIT of Y]
       (T (PROG (SIGNX EXPX HX LX SIGNY EXPY HY LY)
                (.FLOATUNBOX. X SIGNX EXPX HX LX T)
                (.FLOATUNBOX. Y SIGNY EXPY HY LY T)
                (RETURN (AND (EQ HX HY)
                             (EQ LX LY)
                             (EQ EXPX EXPY)
                             (OR (EQ SIGNX SIGNY)
                                 (AND (EQ 0 EXPX)
                                      (EQ 0 HX)
                                      (EQ 0 LX])
```

(\FLOAT

```
  [LAMBDA (X)                                                    ; Edited 28-Feb-87 18:42 by jop
    (OR (FLOATP X)
        (SELECTC (NTYPX X)
            (\FIXP (LET ((HI (fetch (FIXP HINUM) of X))
                         (LO (fetch (FIXP LONUM) of X))
                         (SIGN 0))
                        (COND
                           ((IGREATERP HI MAX.POS.HINUM)
                            (.NEGATE. HI LO)
                            (SETQ SIGN 1)))
                        (\MAKEFLOAT SIGN (IPLUS \EXPONENT.BIAS 31)
                               HI LO T)))
            (\SMALLP (LET* [(HI 0)
                            (SIGN 0)
                            (LO (COND
                                   ((IGEQ X 0)
                                    X)
                                   (T (SETQ SIGN 1)             ; X is negative--negate it
                                      (COND
                                         ((EQ 0 (\LOLOC X))     ; Min small integer
                                          (SETQ HI 1)
                                          0)
                                         (T (ADD1 (IDIFFERENCE MAX.SMALL.INTEGER (\LOLOC X]
                           (\MAKEFLOAT SIGN (IPLUS \EXPONENT.BIAS 31)
                                  HI LO T)))
            (COND
               ((TYPENAMEP X 'BIGNUM)
                (\BIGNUM.TO.FLOAT X))
               ((TYPEP X 'RATIO)
                (FQUOTIENT (CL::RATIO-NUMERATOR X)
                       (CL::RATIO-DENOMINATOR X)))
               (T (CL:ERROR 'XCL:TYPE-MISMATCH :EXPECTED-TYPE '(AND NUMBER (NOT COMPLEX))
                         :NAME X :VALUE X :MESSAGE "a non-complex number"])
```

(\FIXP.FROM.FLOATP

```
  [LAMBDA (X)                                                    (* lmm "19-Sep-85 15:08")
```

```
    (PROG (SIGN EXP HI LO)
          (.FLOATUNBOX. X SIGN EXP HI LO (GO RETZERO))
          (SETQ EXP (IDIFFERENCE EXP (SUB1 \EXPONENT.BIAS)))      ; number of bits to left of binary point
          [COND
             ((ILESSP EXP 0)
              (RETURN 0))
             ([OR (IGREATERP EXP 32)
                  (AND (EQ EXP 32)
                       (OR (EQ 0 SIGN)
                           (NEQ HI \SIGNBIT)
                           (NEQ LO 0]
              (RETURN (LSH (LET ((VAL (PLUS (LSH HI 16)
                                            LO)))
                                 (if (EQ SIGN 1)
                                     then (MINUS VAL)
                                   else VAL))
                           (DIFFERENCE EXP 32]
          [COND
             ((IGEQ (SETQ EXP (IDIFFERENCE 32 EXP))
                    16)
              (SETQ LO (LRSH HI (IDIFFERENCE EXP 16)))
              (SETQ HI 0))
             (T                                                   ; large integer, have to manipulate both halves
                (FRPTQ EXP (.LRSH1. HI LO]
          (COND
             ((EQ SIGN 1)
              (.NEGATE. HI LO)))
          (RETURN (\MAKENUMBER HI LO))
      RETZERO
          (RETURN 0])
```

(**FIXR**

```
  [LAMBDA (X)                                                     (* lmm "22-JUL-84 20:48")
    (OR (FIXP X)
        (PROG (SIGN EXP HI LO ROUNDINGBITS)
              (.FLOATUNBOX. X SIGN EXP HI LO (GO RETZERO))
              (SETQ EXP (IDIFFERENCE EXP (SUB1 \EXPONENT.BIAS)))  ; number of bits to left of binary point
              [COND
                 ((ILESSP EXP 0)
                  (RETURN 0))
                 [(IGEQ EXP 32)                                   ; FIX handles this
                  (RETURN (FIX (FPLUS X 0.5]
                 ([OR (IGREATERP EXP 32)
                      (AND (EQ EXP 32)
                           (OR (EQ 0 SIGN)
                               (NEQ HI \SIGNBIT)
                               (NEQ LO 0]                         ; Overflow: number is larger than MAX.FIXP
                  (RETURN (SELECTQ \OVERFLOW
                                   (T (LISPERROR "OVERFLOW" X T))
                                   (COND
                                      ((EQ 0 SIGN)
                                       MAX.FIXP)
                                      (T MIN.FIXP]
              [COND
                 ((IGEQ EXP 24)                                   ; No decimal places to worry about, so no rounding, just shift into
                                                                  ; place
                  (FRPTQ (IDIFFERENCE 32 EXP)
                         (.LRSH1. HI LO)))
                 (T ;; Shift right until binary point is in the middle of LO, as per \MAKEFLOAT;  then decide how to round, and shift right once
                    ;; more
                    [COND
                       ((IGEQ (SETQ EXP (IDIFFERENCE 24 EXP))
                              16)                                 ; shifting all the way out of the high word
                        (SETQ LO (LRSH [LOGOR HI (COND
                                                    ((EQ 0 LO)
                                                     0)
                                                    (T            ; Sticky bits
                                                       (LRSH \8BITS 1]
                                               (IDIFFERENCE EXP 16)))
                        (SETQ HI 0))
                       (T                                         ; Shift both halves, keeping sticky bits in LO
                          (FRPTQ EXP (.LRSHSTICKY. HI LO]
                    (SETQ ROUNDINGBITS (LOGAND LO \8BITS))
                    (.LRSH8. HI LO)                               ; Shift the rest of the way
                    (COND
                       ((OR (IGREATERP ROUNDINGBITS 128)
                            (AND (EQ ROUNDINGBITS 128)
                                 (ODDP LO)))                      ; Round up if greater than .5, or exactly 0.5 and rounding up will
                                                                  ; make number even
                        (COND
                           ((EQ LO MAX.SMALL.INTEGER)
                            (SETQ LO 0)
                            (add HI 1))
                           (T (add LO 1]
              (COND
```

```
               ((EQ SIGN 1)
                (.NEGATE. HI LO)))
            (RETURN (\MAKENUMBER HI LO))
         RETZERO
            (RETURN 0])
```

(\BOXFPLUSDIF

```
  [LAMBDA (X Y SUBTRACT BOX)                                    (* JonL "17-May-84 18:56")
                                                                ; Does X-Y if SUBTRACT is true
    (PROG (SIGNX EXPX HX LX SIGNY EXPY HY LY EXPDIFF PLEASENORMALIZE CARRY)
          (.FLOATUNBOX. Y SIGNY EXPY HY LY)
          [COND
             (SUBTRACT (SETQ SIGNY (IDIFFERENCE 1 SIGNY]
          (.FLOATUNBOX. X SIGNX EXPX HX LX (GO RESULTISY))
          [COND
             ((AND (EQ 0 HY)
                   (EQ 0 LY))
              (GO DONE))
             ((EQ EXPX \MAX.EXPONENT)
              ;; X = infinity, so result is infinity.  This is not quite right if Y is infinity of opposite sign, though
              (RETURN (\INFINITY SIGNX BOX)))
             ((EQ EXPY \MAX.EXPONENT)
              (RETURN (\INFINITY SIGNY BOX]
          (SETQ EXPDIFF (IDIFFERENCE EXPX EXPY))                ; first align the binary points by right-shifting the smaller guy
          [COND
             [(IGREATERP EXPDIFF 0)
              (COND
                 ((IGREATERP EXPDIFF 31)                        ; Y would get shifted into oblivion
                  (GO DONE))
                 (T (FRPTQ EXPDIFF (.LRSHSTICKY. HY LY]
             ((NEQ EXPDIFF 0)
              (COND
                 ((ILESSP EXPDIFF -31)
                  (GO RESULTISY))
                 (T (FRPTQ (IMINUS EXPDIFF)
                           (.LRSHSTICKY. HX LX))
                    (SETQ EXPX EXPY]
          [COND
             [(EQ SIGNX SIGNY)                                  ; same sign, add magnitudes
              (SETQ CARRY (.ADDSMALL2. LX LY))
              (COND
                 ((EQ (.ADDSMALL3. HX HY CARRY)
                      1)                                        ; there was a carry out of HX, so shift everyone right and stick it
                                                                ; back in
                  (.LRSHSTICKY. HX LX)
                  (add HX \SIGNBIT)
                  (add EXPX 1]
             (T                                                 ; subtract magnitudes, smaller from larger
                (COND
                   ((OR (ILESSP HX HY)
                        (AND (EQ HX HY)
                             (ILESSP LX LY)))                   ; Y is bigger, so swap
                    (swap HX HY)
                    (swap LX LY)
                    (SETQ SIGNX SIGNY)))
                (SETQ PLEASENORMALIZE (NEQ (LOGAND HX \SIGNBIT)
                                           0))                  ; thus if neither operand is normalized, we won't waste time
                                                                ; normalizing and denormalizing the result
                (SETQ HX (IDIFFERENCE (IDIFFERENCE HX HY)
                                      (.SUBSMALL. LX LY]
      DONE
          (RETURN (\MAKEFLOAT SIGNX EXPX HX LX PLEASENORMALIZE BOX))
      RESULTISY
          (RETURN (\MAKEFLOAT SIGNY EXPY HY LY NIL BOX])
```

(\BOXFQUOTIENT

```
  [LAMBDA (X Y BOX)                                             (* lmm "18-DEC-80 13:40")
    (PROG (SIGNX EXPX HX LX (SIGNY 0)
                 (EXPY 0)
                 HY LY BORROW (HZ 0)
                 (LZ 0))
          (.FLOATUNBOX. X SIGNX EXPX HX LX (GO DONE))
          (.FLOATUNBOX. Y SIGNY EXPY HY LY (GO DIVZERO))
          (COND
             ((EQ EXPX \MAX.EXPONENT)                           ; X is infinity
              (RETURN (\INFINITY SIGNX BOX)))
             ((EQ EXPY \MAX.EXPONENT)                           ; Y = infinity, result is zero
              (GO DONE)))
      ;; Divide X -- double length, implicitly extended with zeros -- by Y.  At each step, Y is subtracted from X if possible, putting a one bit in the
      ;; quotient, and then X and the quotient are shifted left.  Result is a 32-bit quotient.
          (.LRSH1. HX LX)
          (.LRSH1. HY LY)                                       ; shift these right one so that we never have to worry about
                                                                ; carrying out of the high bit
```

```
          (FRPTQ 31 (PROGN (.LLSH1. HZ LZ)                          ; shift quotient left one as we accumulate it
                           (COND
                              ((OR (AND (EQ HX HY)
                                        (IGEQ LX LY))
                                   (IGREATERP HX HY))                ; X GE Y, so subtract Y
                               (SETQ HX (IDIFFERENCE (IDIFFERENCE HX HY)
                                                     (.SUBSMALL. LX LY)))
                               (SETQ LZ (ADD1 LZ))                   ; note that this never overflows, because of the left shift we did
                                                                     ; above

                               ))
                           ;; now shift dividend left one.  After the subtraction the high-order bit must be off, so this works okay
                           (.LLSH1. HX LX)))
          (.LLSH1. HZ LZ)                                           ; left shift result 1 to compensate for the earlier right shifts
          [COND
             ((OR (NEQ HX 0)
                  (NEQ LX 0))                                       ; set sticky bit
              (SETQ LZ (LOGOR LZ 1]
      DONE
          (RETURN (\MAKEFLOAT (LOGXOR SIGNX SIGNY)
                              (IPLUS (IDIFFERENCE EXPX EXPY)
                                     \EXPONENT.BIAS)
                              HZ LZ T BOX))
      DIVZERO
          (RETURN (COND
                     ((EQ \OVERFLOW T)
                      (ERROR "FLOATING DIVIDE BY ZERO" Y))
                     (T (\INFINITY SIGNX BOX])
```

(**BOXFTIMES2**

```
  [LAMBDA (X Y BOX)                                                 (* JonL "17-May-84 18:56")
    (PROG (SIGNX EXPX HX LX (SIGNY 0)
                 (EXPY 0)
                 HY LY (HHY 0)
                 (HHZ 0)
                 (HZ 0)
                 (LZ 0)
                 SAVEHY SAVELY CARRY)
          (.FLOATUNBOX. X SIGNX EXPX HX LX (GO DONE)
                 T)
          (.FLOATUNBOX. Y SIGNY EXPY HY LY (GO DONE)
                 T)
          [COND
             ((EQ EXPX \MAX.EXPONENT)                               ; X = infinity
              (RETURN (\INFINITY SIGNX BOX)))
             ((EQ EXPY \MAX.EXPONENT)
              (RETURN (\INFINITY SIGNY BOX]
```

;; Multiply the significands. We have two 24-bit integers, so have a 48-bit, 3-word product, stored as {HHZ,HZ,LZ}. Multiplication will be in two ;; steps: multiply LX by {HY,LY}, storing in result, and then multiply HX by {HY,LY}, storing in the top two words. The first multiplication can be ;; omitted in the not uncommon case of a zero low fraction, and the second multiplication is a little bit simpler, since result fits in two words.

```
          (COND
             ((EQ 0 LX)
              (GO LP2))
             ((EQ 0 LY)                                             ; swap operands to make life easier
              (swap HX HY)
              (swap LX LY)
              (GO LP2)))
          (SETQ SAVEHY HY)                                          ; we'll need these for second step
          (SETQ SAVELY LY)
      LP1                                                           ; multiply LX times HY,LY
          [COND
             ((NEQ (LOGAND LX 1)
                   0)
              (SETQ CARRY (.ADDSMALL2. LZ LY))
              (SETQ CARRY (.ADDSMALL3. HZ HY CARRY))
              (SETQ HHZ (IPLUS HHZ HHY CARRY]
          (COND
             ((EQ 0 (SETQ LX (LRSH LX 1)))                          ; done with this step
              (SETQ HY SAVEHY)
              (SETQ LY SAVELY)
              (GO LP2)))
          (SETQ HHY (LLSH HHY 1))                                   ; left shift Y by one
          (SETQ HY (LLSH (COND
                            ((IGREATERP HY MAX.POS.HINUM)
                             (add HHY 1)
                             (LOGAND HY MAX.POS.HINUM))
                            (T HY))
                         1))
          (SETQ LY (LLSH (COND
                            ((IGREATERP LY MAX.POS.HINUM)
                             (add HY 1)
                             (LOGAND LY MAX.POS.HINUM))
                            (T LY))
                         1))
```

```
      (GO LP1)
 LP2
```

;; multiply HX times HY,LY, adding into high two words of Z. No overflow here, since HX has at most (and usually exactly) 8 bits

```
      [COND
         ((NEQ (LOGAND HX 1)
               0)
          (SETQ CARRY (.ADDSMALL2. HZ LY))
          (SETQ HHZ (IPLUS HHZ HY CARRY]
      (COND
         ((NEQ (SETQ HX (LRSH HX 1))
               0)
          (.LLSH1. HY LY)
          (GO LP2)))
 DONE
```

;; We now have a 48-bit result in HHZ,HZ,LZ. \MAKEFLOAT can handle it from here. Note that the exponent we give is bumped by 1, because ;; the 'binary point' , which was between the first and second bits, was moved one to the right by multiplying

```
      (RETURN (\MAKEFLOAT (LOGXOR SIGNX SIGNY)
                     (IPLUS EXPX EXPY (IDIFFERENCE 1 \EXPONENT.BIAS))
                     HHZ HZ T BOX])
```

(\INFINITY

```
[LAMBDA (SIGN BOX)                                          (* lmm "17-DEC-80 20:32")
```

;; Returns 'infinity' of the appropriate SIGN (0 or 1), reusing floating BOX if given

;; For now, don't return true infinity, but rather the largest representable finite number, so that miscellaneous floating-point routines don't die

```
(OR (FLOATP BOX)
    (SETQ BOX (create FLOATP)))
(replace (FLOATP SIGNBIT) of BOX with SIGN)
(replace (FLOATP EXPONENT) of BOX with (SUB1 \MAX.EXPONENT))
(replace (FLOATP HIFRACTION) of BOX with \MAX.HI.FRAC)
(replace (FLOATP LOFRACTION) of BOX with 65535)
BOX])
```

(\MAKEFLOAT

```
[LAMBDA (SIGN EXP HI LO NORMALIZE BOX)                      (* JonL "17-May-84 18:56")
```

;;; packs up the pieces of a floating point result into a single number box, n the process checking for underflow, rounding, checking overflow. BOX is ;;; optional box to reuse. NORMALIZE is true if we should normalize the result first (make sign bit of HI 1); otherwise we assume result is already ;;; normalized

```
(PROG (ROUNDINGBITS)
      (OR (FLOATP BOX)
          (SETQ BOX (create FLOATP)))
  TOP (COND
         ((AND (EQ 0 HI)
               (EQ 0 LO))
          (replace HIWORD of BOX with (replace LOWORD of BOX with 0))
          (RETURN BOX)))
      [COND
         (NORMALIZE [COND
                       ((EQ 0 HI)
                        (SETQ HI LO)
                        (SETQ LO 0)
                        (SETQ EXP (IDIFFERENCE EXP 16]
                    (while (EQ 0 (LOGAND HI \SIGNBIT)) do (.LLSH1. HI LO)
                                                          (SETQ EXP (SUB1 EXP]
      [COND
         ((ILEQ EXP 0)                                      ; underflow. Scale by 2^Exponentbias in order to deliver a useful
                                                            ; value to the error handler
          (SELECTQ \UNDERFLOW
               (T (RETURN (LISPERROR "FLOATING UNDERFLOW" (\MAKEFLOAT SIGN (IPLUS EXP \EXPONENT.BIAS)
                                                                      HI LO NIL BOX)
                                  T)))
               NIL)
          ;; If we have to return a result, we must 'denormalize' this number. This gives us a little more time before vanishing to zero
          (COND
             ((ILESSP EXP -24)                              ; too small even as denormalized number
              (SETQ HI (SETQ LO 0))
              (GO TOP))
             (T ;; denormalize by shifting right until the exponent is logically 1; final result will have exponent zero, hidden bit zero
                (FRPTQ (IDIFFERENCE 1 EXP)
                       (.LRSHSTICKY. HI LO))
                (SETQ EXP 0]
      (SETQ ROUNDINGBITS (LOGAND LO \8BITS))                ; round result. low order 8 bits are used for rounding
      (.LRSH8. HI LO)
      [COND
         ([OR (IGREATERP ROUNDINGBITS 128)
              (AND (EQ ROUNDINGBITS 128)
                   (NOT (EQ 0 (LOGAND LO 1]
```

```
            ;; round up if the left over fraction was greater than 1/2;  if it was equal to a half, round to the even result
            (COND
               [(EQ LO MAX.SMALL.INTEGER)                          ; can't add 1 directly
                (SETQ LO 0)
                (SETQ HI (ADD1 HI))
                (COND
                   ((IGREATERP HI (LOGOR \HIDDENBIT \MAX.HI.FRAC))
                                                                   ; '1.11111--' became '10.000--'
                    (SETQ HI (LRSH HI 1))
                    (add EXP 1]
                  (T (SETQ LO (ADD1 LO]
          [COND
             ((AND (EQ HI 0)
                   (EQ LO 0))
            ;; result is zero.  This could have snuck in if we denormalized a number that didn't have enough digits to survive
             (GO TOP))
            ((IGEQ EXP \MAX.EXPONENT)
            ;; overflow.  If trap enabled, wrap the exponent around to middle of range (divide by 2^exponentbias) to provide a number of possible
            ;; use to error handler
             (SELECTQ \OVERFLOW
                  (T (RETURN (LISPERROR "FLOATING OVERFLOW" (\MAKEFLOAT SIGN (IDIFFERENCE EXP \EXPONENT.BIAS)
                                                                HI LO NIL BOX)
                                     T)))
                  NIL)
             (RETURN (\INFINITY SIGN BOX]
          (replace SIGNBIT of BOX with SIGN)
          (replace EXPONENT of BOX with EXP)
          (replace HIFRACTION of BOX with HI)
          (replace LOFRACTION of BOX with LO)
          (RETURN BOX])
```

(MAKEFLOATNUMBER

```
  [LAMBDA (N0 N1)                                                  (* lmm "12-Apr-85 18:19")
                                                                   ; CALLED FROM FETCHFIELD
    (LET [(VAL (NCREATE 'FLOATP]
         (replace (FLOATP HIWORD) of VAL with N0)
         (replace (FLOATP LOWORD) of VAL with N1)
        VAL])
```

(PutFloat

```
  [LAMBDA (PTR N)                                                  (* lmm "29-Dec-84 11:32")
                                                                   ; used by REPLACEFIELD
    (\PUTBASEFLOATP PTR 0 N)
    N])

)

(PUTPROPS ZEROP DMACRO [OPENLAMBDA (X)
                          (COND
                             ((EQ X 0))
                             ((FLOATP X)
                              (\FZEROP X])

(DEFINEQ
```

(SQRT

```
  [LAMBDA (N)                                                      (* lmm "24-Jan-85 19:13")
    (PROG ((X (FLOAT N))
           V)
          (DECLARE (TYPE FLOATP X V))
          (if (FLESSP X 0.0)
              then (ERROR "SQRT OF NEGATIVE VALUE" N)
            elseif (NOT (FGREATERP X 0.0))
              then                                                 ; Trichotomy ==> X = 0.0
                   (RETURN 0.0))
          (SETQ V (create FLOATP
                          EXPONENT _ (LOGAND (IPLUS \EXPONENT.BIAS (LRSH (LOGAND (IDIFFERENCE (fetch (FLOATP EXP)
                                                                                                 of X)
                                                                                               \EXPONENT.BIAS)
                                                                                 (MASK.1'S 0 BITSPERWORD))
                                                                           1))
                                             \MAX.EXPONENT)
                          HIFRACTION _ (fetch (FLOATP HIFRAC) of X)))
        ;; Exponent is stored as excess \EXPONENT.BIAS and although the LRSH doesn't really do division by 2 (e.g., when the arg is negative) at least
        ;; the low-order 8 bits will be right.  It doesn't even matter that it may be off-by-one, due to the infamous 'Arithmetic Shifting Considered Harmful'
        ;; since it is only an estimate.
          [FRPTQ 4 (SETQ V (FTIMES 0.5 (FPLUS V (FQUOTIENT X V]
          (RETURN V])

)

(DECLARE%: EVAL@COMPILE DONTCOPY
```

;; FOLLOWING DEFINITIONS EXPORTED

```
(DECLARE%: EVAL@COMPILE

[BLOCKRECORD FLOATP ((SIGNBIT BITS 1)
                     (EXPONENT BITS 8)
                     (HIFRACTION BITS 7)
                     (LOFRACTION BITS 16))
       (BLOCKRECORD FLOATP ((HIWORD WORD)
                            (LOWORD WORD)))
       (BLOCKRECORD FLOATP ((NIL BITS 9)
                            (LONGFRACTION BITS 23)))
       (BLOCKRECORD FLOATP ((FLOATCONTENTS BITS 32)))
       (BLOCKRECORD FLOATP ((NIL BITS 1)
                            (HIWORDNOSIGNBIT BITS 15)))
       (CREATE (\FLOATBOX (\VAG2 (LOGOR (LLSH SIGNBIT (PLUS 7 8))
                                        (LLSH EXPONENT 7)
                                        HIFRACTION)
                                 LOFRACTION)))
      LOFRACTION _ 0 HIFRACTION _ 0 EXPONENT _ 0 SIGNBIT _ 0 (ACCESSFNS
                                                               FLOATP
                                                               ((EXP (LOGAND (LRSH (\HILOC (\FLOATUNBOX DATUM))
                                                                                   7)
                                                                             255))
                                                                (HIFRAC (LOGAND (\HILOC (\FLOATUNBOX DATUM))
                                                                                127]
)

(DECLARE%: EVAL@COMPILE

(RPAQQ MAX.DIGITS.ACCURACY 9)

(CONSTANTS (MAX.DIGITS.ACCURACY 9))
)
```

;; END EXPORTED DEFINITIONS

```
(DECLARE%: EVAL@COMPILE

(RPAQQ \8BITS 255)

(RPAQQ \MAX.HI.FRAC 127)

(RPAQQ \SIGNBIT 32768)

(RPAQQ \EXPONENT.BIAS 127)

(RPAQQ \HIDDENBIT 128)

(RPAQQ \MAX.EXPONENT 255)

(CONSTANTS (\8BITS 255)
       (\MAX.HI.FRAC 127)
       (\SIGNBIT 32768)
       (\EXPONENT.BIAS 127)
       (\HIDDENBIT 128)
       (\MAX.EXPONENT 255))
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS .FLOATUNBOX. MACRO
          [(FLONUM SIGN EXP HI LO ZEROFORM DONTSHIFT RESTARTIFINTEGER)
           ;; Unpacks a floating point number FLONUM into its components.  ZEROFORM is evaluated if the number is true zero.  The fraction is
           ;; unpacked into HI and LO, with the binary point implicitly between bits 0 and 1 of HI.  If DONTSHIFT is true, the fraction is left in its
           ;; original state, with 8 bits in HI and 16 in LO.  If FLONUM is not floating, it is coerced.
           (PROG NIL
             RETRY
                 [COND
                    ((NOT (FLOATP FLONUM))                          ; Float and normalize the non-floatp
                     (COND
                        ('RESTARTIFINTEGER (SETQ FLONUM (LISPERROR "NON-NUMERIC ARG" FLONUM T))
                               (GO RESTARTIFINTEGER))
                        (T (SELECTC (NTYPX FLONUM)
                                (\FIXP (SETQ HI (fetch (FIXP HINUM) of FLONUM))
                                       (SETQ LO (fetch (FIXP LONUM) of FLONUM))
                                       (SETQ SIGN (COND
                                                      ((IGREATERP HI MAX.POS.HINUM)
                                                       (.NEGATE. HI LO)
                                                       1)
                                                      (T 0))))
                                (\SMALLP (SETQ HI 0)
                                         [SETQ LO (COND
                                                      ((SMALLPOSP FLONUM)
```

```
                                                (SETQ SIGN 0)
                                                FLONUM)
                                               (T (SETQ SIGN 1)
                                                                  ; FLONUM is negative--negate it
                                                  (COND
                                                     ((EQ 0 (LOLOC FLONUM))
                                                                  ; Min small integer
                                                      (SETQ HI 1)
                                                      0)
                                                     (T (ADD1 (IDIFFERENCE MAX.SMALL.INTEGER (LOLOC FLONUM
                                                                                                        ])
                                 (PROGN (SETQ FLONUM (FLOAT FLONUM))
                                        (GO RETRY)))
                             [COND
                                [(EQ 0 HI)
                                 (COND
                                    ((EQ 0 LO)
                                     (SETQ EXP 0)
                                     (PROGN ZEROFORM (RETURN)))
                                    (T (SETQ HI LO)
                                       (SETQ LO 0)
                                       (SETQ EXP (IPLUS \EXPONENT.BIAS 15]
                                ((IGREATERP HI 255)                ; Not exact, punt
                                 (SETQ FLONUM (FLOAT FLONUM))
                                 (GO UNPACK))
                                (T (SETQ EXP (IPLUS \EXPONENT.BIAS 31]
                             [COND
                                ((ILEQ HI 255)                     ; Do a big shift first.
                                 (.LLSH8. HI LO)
                                 (SETQ EXP (IDIFFERENCE EXP 8]
                             (while (EQ 0 (LOGAND HI \SIGNBIT)) do (.LLSH1. HI LO)
                                                                   (SETQ EXP (SUB1 EXP)))
                             (COND
                                (DONTSHIFT (.LRSH8. HI LO)))
                             (RETURN]
          UNPACK
              (SETQ SIGN (fetch (FLOATP SIGNBIT) of FLONUM))
              (SETQ LO (fetch (FLOATP LOFRACTION) of FLONUM))
              (SETQ HI (fetch (FLOATP HIFRACTION) of FLONUM))
              [COND
                 [(EQ 0 (SETQ EXP (fetch (FLOATP EXPONENT) of FLONUM)))
                                                                   ; zero or a de-normalized number from underflow
                  (COND
                     ((AND (EQ 0 HI)
                           (EQ 0 LO))                              ; A zero, regardless of the sign bit zero
                      ZEROFORM)
                     (T                                            ; need bias adjust to account for lack of hidden bit
                        (SETQ EXP 1]
                 ((NEQ EXP \MAX.EXPONENT)                          ; might want to check for NaN's here if EXP = \MAX.EXPONENT
                                                                   ; OR in the implicit high bit of fraction
                  (SETQ HI (IPLUS HI \HIDDENBIT]
              (COND
                 ((NOT DONTSHIFT)
                  (.LLSH8. HI LO])

(PUTPROPS .LLSH1. MACRO ((HI LO)                                   ; shift the pair left one, assuming no overflow
                         (SETQ HI (LLSH HI 1))
                         (SETQ LO (LLSH (COND
                                           ((IGREATERP LO MAX.POS.HINUM)
                                            (add HI 1)
                                            (LOGAND LO MAX.POS.HINUM))
                                           (T LO))
                                        1))))

(PUTPROPS .LLSH8. MACRO ((HI LO)                                   ; shift pair left 8, assuming no overflow
                         (SETQ HI (IPLUS (LLSH HI 8)
                                         (LRSH LO 8)))
                         (SETQ LO (LLSH (LOGAND LO \8BITS)
                                        8))))

(PUTPROPS .LRSH1. MACRO ((HI LO)
                         (SETQ LO (LRSH LO 1))
                         [COND
                            ((NEQ (LOGAND HI 1)
                                  0)
                             (SETQ LO (IPLUS LO \SIGNBIT]
                         (SETQ HI (LRSH HI 1))))

(PUTPROPS .LRSH8. MACRO ((HI LO)
                         (SETQ LO (IPLUS (LRSH LO 8)
                                         (LLSH (LOGAND HI \8BITS)
                                               8)))
                         (SETQ HI (LRSH HI 8))))

(PUTPROPS .LRSHSTICKY. MACRO ((HI LO)                              ; shifts pair right one, but low-order bit is sticky -- if it ever
                                                                   ; becomes 1, it stays 1
                              (SETQ LO (LOGOR (LRSH LO 1)
```

```
                                        (LOGAND LO 1)))
                                 [COND
                                    ((NEQ (LOGAND HI 1)
                                          0)
                                     (SETQ LO (IPLUS LO \SIGNBIT]
                                 (SETQ HI (LRSH HI 1))))

(PUTPROPS .ADDSMALL2. MACRO [(X Y)
                                 (PROGN                                 ; does X _ X+Y, returning the carry bit
                                        (COND
                                           ((IGREATERP X (IDIFFERENCE MAX.SMALL.INTEGER Y))
                                            [SETQ X (IDIFFERENCE X (IDIFFERENCE MAX.SMALL.INTEGER (SUB1 Y]
                                            1)
                                           (T (SETQ X (IPLUS X Y))
                                              0])

(PUTPROPS .ADDSMALL3. MACRO [(X Y CARRY)
                                 (PROGN                                 ; X _ X+Y+CARRY, returning the new carry bit
                                        (COND
                                           ((IGREATERP X (IDIFFERENCE (IDIFFERENCE MAX.SMALL.INTEGER Y)
                                                                      CARRY))
                                            (SETQ X (IDIFFERENCE X (IDIFFERENCE
                                                                    [IDIFFERENCE MAX.SMALL.INTEGER
                                                                           (SUB1 (COND
                                                                                    ((EQ Y 0)
                                                                                     (PROG1 CARRY (SETQ CARRY 0)))
                                                                                    (T Y]
                                                                    CARRY)))
                                            1)
                                           (T (SETQ X (IPLUS X Y CARRY))
                                              0])

(PUTPROPS .SUBSMALL. MACRO ((X Y)                                       ; Subtract Y from X, returning the borrow out of the next word
                                (COND
                                   ((ILEQ Y X)
                                    (SETQ X (IDIFFERENCE X Y))
                                    0)
                                   (T [SETQ X (ADD1 (IDIFFERENCE MAX.SMALL.INTEGER (IDIFFERENCE Y X]
                                      1))))

(PUTPROPS .POWEROF2. MACRO [OPENLAMBDA (X)
                                 (COND
                                    ((ILESSP X 16)
                                     (LSH 1 X))
                                    (T (LSH (LSH 1 (IDIFFERENCE X 16))
                                            16])
)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(LOCALVARS . T)
)
)

(DECLARE%: DONTEVAL@LOAD DOCOPY

(RPAQQ \UNDERFLOW NIL)

(RPAQ MAX.FLOAT (\INFINITY 0))

(RPAQ MIN.FLOAT (\INFINITY 1))

(MOVD? 'FGREATERP 'FGTP)
)

;; unboxed ufns

(DEFINEQ

(\UNBOXFLOAT1
  [LAMBDA (OP)                                                          (* lmm " 5-Mar-86 11:35")
    ;; UFN for the unboxed floating 1-arg cases
    (\SLOWRETURN)
    (SELECTQ OP
         (0                                                             ; BOX
            (\CALLER.ARGS (X)
                   (LET [(VAL (NCREATE 'FLOATP]
                        (replace (FLOATP HIWORD) of VAL with (\HILOC X))
                        (replace (FLOATP LOWORD) of VAL with (\LOLOC X))
                        VAL)))
         (1                                                             ; UNBOX
            (\CALLER.ARGS (X)
                   (\HAND.FLOATUNBOX X)))
         (2                                                             ; UFABS
            (\CALLER.ARGS ((X FLOATP))
                   (\FLOATUNBOX (ABS X))))
```

```
        (3                                                          ; UFNEGATE
           (\CALLER.ARGS ((X FLOATP))
                  (\FLOATUNBOX (FMINUS X))))
        (4                                                          ; UFIX
           (\CALLER.ARGS ((X FLOATP))
                  (FIX X)))
        (HELP "\UNBOXFLOAT1 called with illegal op " OP])
```

(**UNBOXFLOAT2**

```
  [LAMBDA (OP)                                                      (* lmm " 7-Mar-85 16:48")
```

;; UFN for the 2-arg floating cases

```
    (\CALLER.ARGS ((X FLOATP)
                   (Y FLOATP))
          (SELECTQ OP
               (0                                                   ; UFADD
                  (\HAND.FLOATUNBOX (FPLUS X Y)))
               (1                                                   ; UFSUB
                  (\HAND.FLOATUNBOX (FDIFFERENCE X Y)))
               (2                                                   ; UFISUB
                  (\HAND.FLOATUNBOX (FDIFFERENCE Y X)))
               (3                                                   ; UFMULT
                  (\HAND.FLOATUNBOX (FTIMES X Y)))
               (4                                                   ; UFDIV
                  (\HAND.FLOATUNBOX (FQUOTIENT X Y)))
               (5                                                   ; UFGREAT
                  (FGREATERP X Y))
               (6                                                   ; UFMAX
                  (\HAND.FLOATUNBOX (FMAX X Y)))
               (7                                                   ; UFMIN
                  (\HAND.FLOATUNBOX (FMIN X Y)))
               (8                                                   ; UFREM
                  (\HAND.FLOATUNBOX (FREMAINDER X Y)))
               (HELP "\UNBOXFLOAT2 called with illegal op " OP])
```

(**UNBOXFLOAT3**

```
  [LAMBDA (OP)                                                      (* jop%: "29-Aug-86 14:30")
    (if (EQ 0 OP)
        then [\CALLER.ARGS ((X FLOATP)
                            COEFFICIENTS DEGREE)                    ; Polynomial evaluation
                     (bind (RESULT _ (\GETBASEFLOATP COEFFICIENTS 0)) declare (TYPE FLOATP RESULT) for I
                        from 2 to (LLSH DEGREE 1) by 2 do (SETQ RESULT (FPLUS (FTIMES RESULT X)
                                                                              (\GETBASEFLOATP COEFFICIENTS I)))
                        finally (RETURN (\FLOATUNBOX RESULT]
      else (\CALLER.ARGS (MATRIX1 MATRIX2 RESULT)
                 (SELECTQ OP
                      (1                                            ; 3 x 3 matrix multiply
                         (\MATMULT333 MATRIX1 MATRIX2 RESULT))
                      (2                                            ; 4 x 4 matrix multiply
                         (\MATMULT444 MATRIX1 MATRIX2 RESULT))
                      (3                                            ; (1,3) * (3,3) => (1,3)
                         (\MATMULT133 MATRIX1 MATRIX2 RESULT))
                      (4                                            ; (3,3) * (3,1) => (3,1)
                         (\MATMULT331 MATRIX1 MATRIX2 RESULT))
                      (5                                            ; (1,4) * (4,4) => (1,4)
                         (\MATMULT144 MATRIX1 MATRIX2 RESULT))
                      (6                                            ; (4,4) * (4,1) => (4,1)
                         (\MATMULT441 MATRIX1 MATRIX2 RESULT))
                      (HELP "\UNBOXFLOAT3 called with illegal op " OP])

)

(DEFINEQ
```

(**MATMULT133**

```
  [LAMBDA (ABASE BBASE CBASE)                                       (* jop%: "29-Aug-86 12:20")
```

;;; Multiply a 3 vector times a 3 by 3 array

```
    (for K from 0 to 4 by 2
       do (bind (PRODUCT _ 0.0) declare (TYPE FLOATP PRODUCT) for I from 0 to 4 by 2 as J from K by 6
             do [SETQ PRODUCT (FPLUS PRODUCT (FTIMES (\GETBASEFLOATP ABASE I)
                                                     (\GETBASEFLOATP BBASE J]
             finally (\PUTBASEFLOATP CBASE K PRODUCT)))
    CBASE])
```

(**MATMULT144**

```
  [LAMBDA (ABASE BBASE CBASE)                                       (* jop%: "29-Aug-86 12:20")
```

;;; Multiply a 4 vector times a 4 by 4 array

```
    (for K from 0 to 6 by 2
       do (bind (PRODUCT _ 0.0) declare (TYPE FLOATP PRODUCT) for I from 0 to 6 by 2 as J from K by 8
             do [SETQ PRODUCT (FPLUS PRODUCT (FTIMES (\GETBASEFLOATP ABASE I)
```

```
                                                    (\GETBASEFLOATP BBASE J]
            finally (\PUTBASEFLOATP CBASE K PRODUCT)))
    CBASE])
```

(\MATMULT331

```
  [LAMBDA (ABASE BBASE CBASE)                                   (* jop%: "29-Aug-86 12:20")
```

;;; Multiply a 3 by 3 array by a 3 vector

```
    (for K from 0 to 4 by 2 do (bind (PRODUCT _ 0.0) declare (TYPE FLOATP PRODUCT) for I
                                  from (ITIMES K 3) by 2 as J from 0 to 4 by 2
                                  do [SETQ PRODUCT (FPLUS PRODUCT (FTIMES (\GETBASEFLOATP ABASE I)
                                                                          (\GETBASEFLOATP BBASE J]
                                  finally (\PUTBASEFLOATP CBASE K PRODUCT)))
    CBASE])
```

(\MATMULT333

```
  [LAMBDA (ABASE BBASE CBASE)                                   (* jop%: "29-Aug-86 12:31")
```

;;; Multiply two 3 by 3 arrays

```
    [bind (K _ 0) for ASTART from 0 to 12 by 6
       do (for BSTART from 0 to 4 by 2 do (bind (PRODUCT _ 0.0) declare (TYPE FLOATP PRODUCT) for I from ASTART
                                             to (IPLUS ASTART 4) by 2 as J from BSTART by 6
                                             do [SETQ PRODUCT (FPLUS PRODUCT (FTIMES (\GETBASEFLOATP ABASE I)
                                                                                     (\GETBASEFLOATP BBASE J]
                                             finally (\PUTBASEFLOATP CBASE K PRODUCT))
                                          (SETQ K (IPLUS K 2]
    CBASE])
```

(\MATMULT441

```
  [LAMBDA (ABASE BBASE CBASE)                                   (* jop%: "29-Aug-86 12:20")
```

;;; Multiply a 4 by 4 array by a 4 vector

```
    (for K from 0 to 6 by 2 do (bind (PRODUCT _ 0.0) declare (TYPE FLOATP PRODUCT) for I
                                  from (ITIMES K 4) by 2 as J from 0 to 6 by 2
                                  do [SETQ PRODUCT (FPLUS PRODUCT (FTIMES (\GETBASEFLOATP ABASE I)
                                                                          (\GETBASEFLOATP BBASE J]
                                  finally (\PUTBASEFLOATP CBASE K PRODUCT)))
    CBASE])
```

(\MATMULT444

```
  [LAMBDA (ABASE BBASE CBASE)                                   (* jop%: "29-Aug-86 12:31")
```

;;; Multiply two 4 by 4 arrays

```
    [bind (K _ 0) for ASTART from 0 to 24 by 8
       do (for BSTART from 0 to 6 by 2 do (bind (PRODUCT _ 0.0) declare (TYPE FLOATP PRODUCT) for I from ASTART
                                             to (IPLUS ASTART 6) by 2 as J from BSTART by 8
                                             do [SETQ PRODUCT (FPLUS PRODUCT (FTIMES (\GETBASEFLOATP ABASE I)
                                                                                     (\GETBASEFLOATP BBASE J]
                                             finally (\PUTBASEFLOATP CBASE K PRODUCT))
                                          (SETQ K (IPLUS K 2]
    CBASE])

)
```

;; unboxed arg handling

```
(DECLARE%: DONTCOPY
```

;; FOLLOWING DEFINITIONS EXPORTED

```
(DECLARE%: EVAL@COMPILE

(PUTPROPS \CALLER.ARGS MACRO
          [X
           (LET ((ARGS (CAR X))
                 (FORMS (CDR X)))
                `(PROGN (\SLOWRETURN)
                        (LET [(AL (\MYALINK))
                              NEXT
                              ,@(for VAR in ARGS collect (COND
                                                           ((LISTP VAR)
                                                            (LIST (CAR VAR)
                                                                  0))
                                                           (T VAR]
                             [DECLARE ,@(for VAR in ARGS when (LISTP VAR)
                                           collect `(TYPE ,(SELECTQ (CADR VAR)
                                                                    ((FLOATING FLOATP)
                                                                        (CADR VAR))
```

```
                                                    (HELP))
                                                ,(CAR VAR]
                                (SETQ NEXT (fetch (FX NEXTBLOCK) of AL))
                                ,@[for X in (REVERSE ARGS)
                                     collect (LET [(FORMS `(\.GETBASE32 \STACKSPACE (SETQ NEXT (IDIFFERENCE NEXT
                                                                                                 WORDSPERCELL]
                                                  (COND
                                                     [(LISTP X)
                                                      `(SETQ ,(CAR X)
                                                         (\FLOATBOX ,FORMS]
                                                     (T `(SETQ ,X ,FORMS]
                                (\MAKEFREEBLOCK NEXT (TIMES ,(LENGTH ARGS)
                                                            WORDSPERCELL))
                                (replace (FX NEXTBLOCK) of AL with NEXT)
                                (PROGN ,@FORMS])
)
)

;; END EXPORTED DEFINITIONS

(DEFINEQ

(FLOATP.TO.BCPL
  [LAMBDA (FLONUM)                                                    (* bvm%: "22-OCT-81 22:31")

;;; Converts a floating point number in IEEE format to an integer in BCPL floating-point format

    (OR (FLOATP FLONUM)
        (SETQ FLONUM (FLOAT FLONUM)))
    (PROG (RESULT FRAC (EXP (IPLUS (fetch EXPONENT of FLONUM)
                                   2)))
          (COND
             ((FEQP FLONUM 0.0)
              (RETURN 0)))
          [COND
             ((IGREATERP EXP 255)                                     ; Overflow, so just return BCPL infinity
              (SETQ EXP 255)
              (SETQ FRAC 4194303))
             (T (SETQ FRAC (LRSH (fetch LONGFRACTION of FLONUM)
                                 1]
          (SETQ RESULT (create BCPLNUM
                               BCPLEXPONENT _ EXP
                               SIGNIFICANTBIT _ 1
                               BCPLHIFRACTION _ (LRSH FRAC 16)
                               BCPLLOFRACTION _ (LOGAND FRAC MAX.SMALL.INTEGER)))
          (RETURN (COND
                     ((EQ (fetch SIGNBIT of FLONUM)
                          1)
                      (IMINUS RESULT))
                     (T RESULT])

(BCPL.TO.FLOATP
  [LAMBDA (BCPLNUM)                                                   (* bvm%: "22-OCT-81 22:34")
    ;; Converts BCPLNUM, an integer in BCPL floating-point format, to a FLOATP, which is IEEE standard
    (PROG (SIGN EXP FRAC)
          (COND
             ((ILESSP BCPLNUM 0)
              (SETQ BCPLNUM (IMINUS BCPLNUM))                          ; In a negative BCPL format, whole number is complemented
              (SETQ SIGN 1))
             ((IEQP BCPLNUM 0)                                         ; Canonical form for 0.0
              (RETURN (FPLUS 0.0)))
             (T (SETQ SIGN 0)))
          (COND
             ((OR (SMALLP BCPLNUM)
                  (NEQ (fetch SIGNIFICANTBIT of BCPLNUM)
                       1))
              (ERROR "Not a valid BCPL flonum" BCPLNUM)))
          [COND
             ((ILESSP (SETQ EXP (IDIFFERENCE (fetch BCPLEXPONENT of BCPLNUM)
                                             2))
                      0)
              ;; Underflow.  IEEE exponent is off by 2 because the bias is one smaller in IEEE format and we shift the mantissa left one
              (RETURN (FPLUS 0.0]
          (SETQ FRAC (LLSH (fetch RESTOFFRACTION of BCPLNUM)
                           1))
          (RETURN (create FLOATP
                          SIGNBIT _ SIGN
                          EXPONENT _ EXP
                          HIFRACTION _ (LRSH FRAC 16)
                          LOFRACTION _ (LOGAND FRAC MAX.SMALL.INTEGER])

)

(DECLARE%: EVAL@COMPILE DONTCOPY
```

```
(DECLARE%: EVAL@COMPILE

(BLOCKRECORD BCPLNUM ((BCPLSIGNBIT BITS 1)
                      (BCPLEXPONENT BITS 8)                    ; exponent, biased by 128
                      (SIGNIFICANTBIT BITS 1)                  ; Always 1 in a bcpl num;  binary point is to left
                      (RESTOFFRACTION BITS 22))
       (BLOCKRECORD BCPLNUM ((NIL BITS 10)
                             (BCPLHIFRACTION BITS 6)
                             (BCPLLOFRACTION BITS 16)))
       (CREATE (CREATECELL \FIXP)))

(RECORD EFPN (EXP HI LO))
)
)
```

```
(CL:DEFVAR INTPOWERS
   (LET ((AR (CL:MAKE-ARRAY 10)))
        ;; (MAKE-ARRAY 10 INITIAL-CONTENTS (QUOTE (1 10 100 1000 10000 100000 1000000 10000000 100000000 1000000000)))
        ;; This HORRIBLE hack is here because not enough of MAKE-ARRAY is in the loadup to let the above form work right.  (The ASETs are
        ;; written open to avoid the gensyms produced by the stupid SETF expansion.)  TBF by BOB BANE
        (ASET 1 AR 0)
        (ASET 10 AR 1)
        (ASET 100 AR 2)
        (ASET 1000 AR 3)
        (ASET 10000 AR 4)
        (ASET 100000 AR 5)
        (ASET 1000000 AR 6)
        (ASET 10000000 AR 7)
        (ASET 100000000 AR 8)
        (ASET 1000000000 AR 9)
        AR))
```

```
(CL:DEFUN ENUM-STRING (OUTSTR MANTSTR INTEXP DECPLACES EXPWIDTH)
   ;; Prints exponential notation observing rounding & exponent spacing
   (CL:MACROLET [(STRPUT (C)
                   `(CL:VECTOR-PUSH-EXTEND ,C OUTSTR]
         [LET ((DIGITS (CL:LENGTH MANTSTR))
               (POINTPLACE 1)
               (INDEX -1)
               EXPOFFSET)
              (CL:SETF (CL:FILL-POINTER OUTSTR)
                    0)
              (IF DECPLACES
                  THEN (SETQ POINTPLACE (- DIGITS DECPLACES)))
              (SETQ EXPOFFSET (- DIGITS POINTPLACE))
              ;; Print the mantissa
              [IF (MINUSP POINTPLACE)                          ; .0 before mantissa needed
                  THEN (STRPUT #\.)
                       (CL:DOTIMES (I (- POINTPLACE))
                           (STRPUT #\0))
                       (CL:DOTIMES (I DIGITS)
                           (STRPUT (CL:CHAR MANTSTR I)))
                ELSE (CL:DOTIMES (I POINTPLACE)
                           (STRPUT (CL:CHAR MANTSTR (CL:INCF INDEX))))
                     (STRPUT #\.)
                     (CL:DOTIMES (I EXPOFFSET)
                         (STRPUT (CL:CHAR MANTSTR (CL:INCF INDEX))))
                     (IF DECPLACES
                         THEN (CL:DOTIMES (I (- DECPLACES EXPOFFSET))
                                    (STRPUT #\0))
                           ELSE (IF (EQ 0 EXPOFFSET)
                                    THEN (STRPUT #\0)          ; Must print at least one decimal place in this case
                                    ]
              ;; mantissa done - now for the exponent
              (CL:INCF INTEXP EXPOFFSET)
              (SETQ MANTSTR (WITH-RESOURCES (\NUMSTR \NUMSTR1)
                                (\CONVERTNUMBER (ABS INTEXP)
                                                10 T NIL \NUMSTR \NUMSTR1)))
              (SETQ DIGITS (CL:LENGTH MANTSTR))
              (STRPUT #\E)
              (IF (MINUSP INTEXP)
                  THEN (STRPUT #\-)
                ELSE (STRPUT #\+))
              (IF EXPWIDTH
                  THEN (CL:DOTIMES (I (- EXPWIDTH DIGITS 2))
                             (STRPUT #\0)))
              (CL:DOTIMES (I DIGITS)
                  (STRPUT (CL:CHAR MANTSTR I)))]
         OUTSTR))
```

```
(CL:DEFUN FNUM-STRING (OUTSTR MANTSTR INTEXP DECPLACES)
    ;; Prints floating decimal output observing # of places required
    (CL:MACROLET [(STRPUT (C)
                    `(CL:VECTOR-PUSH ,C OUTSTR]
           [LET* ((DIGITS (CL:LENGTH MANTSTR))
                  (POINTPLACE (+ DIGITS INTEXP))
                  (INDEX -1)
                  PLACESOUT)
                 (CL:SETF (CL:FILL-POINTER OUTSTR)
                       0)
                 (COND
                    ((NOT (CL:PLUSP POINTPLACE))
                     (STRPUT #\0)
                     (STRPUT #\.)
                     (CL:DOTIMES (I (- POINTPLACE))
                         (STRPUT #\0))
                     (CL:DOTIMES (I DIGITS)
                         (STRPUT (CL:CHAR MANTSTR I)))
                     (SETQ PLACESOUT (- DIGITS POINTPLACE)))
                    ((MINUSP INTEXP)
                     (CL:DOTIMES (I POINTPLACE)
                         (STRPUT (CL:CHAR MANTSTR (CL:INCF INDEX))))
                     (STRPUT #\.)
                     (CL:DOTIMES (I (- INTEXP))
                         (STRPUT (CL:CHAR MANTSTR (CL:INCF INDEX))))
                     (SETQ PLACESOUT (- INTEXP)))
                    (T (CL:DOTIMES (I DIGITS)
                           (STRPUT (CL:CHAR MANTSTR I)))
                       (CL:DOTIMES (I INTEXP)
                           (STRPUT #\0))
                       (STRPUT #\.)
                       (STRPUT #\0)
                       (SETQ PLACESOUT 1)))
                 (IF DECPLACES
                     THEN (CL:DOTIMES (I (- DECPLACES PLACESOUT))
                              (STRPUT #\0))]
           OUTSTR))

(CL:DEFUN FLTSTR (F K)

;;; Returns a string MANT and a fixp EXP such that F = MANT * 10 ** EXP, to K digits. Algorithm copped from "An Implementation guide to a Proposed
;;; Standard for Floating-Point Arithmetic" by J T Coonen (IEEE Computer Jan. 1980), and modified somewhat. The hack here when printing to
;;; unspecified precision is to always generate 7 digits, check to see if that's enough, and then clip trailing zeros from whatever results. This tends to
;;; produce more digits than necessary for denormalized numbers, but it makes everything else print a LOT faster.

    [IF (= F 0.0)
        THEN ;; Foo! You have to do it this way because the people who call FLTSTR assume they can smash whatever it returns... It would also be
             ;; nice if it were documented somewhere that WITH-RESOURCES expands into a PROG1 and therefore can't return multiple-values...
             (LET (outstring)
                  (WITH-RESOURCES (\NUMSTR \NUMSTR1)
                         (RPLCHARCODE \NUMSTR 1 (CHARCODE 0))
                         (SETQ outstring (SUBSTRING \NUMSTR 1 1 \NUMSTR1)))
                  (CL:VALUES outstring 0))
      ELSE (PROG (SIGNF FEXP FHI FLO TEXP THI TLO MANT EXP ROUNDINGBITS (LOCALK (CL:IF K
                                                                                   (MIN 9 K)
                                                                                   7))
                         FLOG10 MANTSTRING)
                 (DECLARE (CL:SPECIAL TEXP THI TLO))            ; used by extended floating multiplier and inttoext
                 (.FLOATUNBOX. F SIGNF FEXP FHI FLO)
           ;; Re-normalize
                 (if (EQ 0 FHI)
                     then (SETQ FHI FLO)
                          (SETQ FLO 0)
                          (CL:DECF FEXP 16))
                 (while (EQ 0 (LOGAND FHI \SIGNBIT)) do (.LLSH1. FHI FLO)
                                                        (CL:DECF FEXP))
           ;; find # of digits before decimal point by looking up base 10 log in extpowers
                 (SETQ FLOG10 (FLTINTLOG FEXP FHI FLO))
            MOREDIGITS
                 (SETQ EXP (- FLOG10 LOCALK))
            AGAIN
                 (TIMESPOW10 (- EXP)
                        FEXP FHI FLO)                           ; results in texp thi tlo
                 (SETQ MANT (EXTTOINT TEXP THI TLO))
           ;; Now compare the result to 10**k to check if the exp guess was a good one
           ;; This code is in the original algorithm, but I'm not sure it's needed here. What the heck, it's pretty fast...
           ;; (cond ((>= mant (cl:1+ (cl:aref intpowers localk))) (cl:incf exp) (go again)) ((eql mant (cl:aref intpowers localk)) (cl:incf exp) (setq mant
           ;; (cl:aref intpowers (cl:1- localk)))) ((<= mant (cl:1- (cl:aref intpowers (cl:1- localk)))) (cl:decf exp) (go again)))
           ;; If K came in NIL, check to see if enough digits have been generated
```

```
                [if (NOT K)
                    then (INTTOEXT MANT)                          ; values in texp thi tlo
                         (TIMESPOW10 EXP TEXP THI TLO)
                         (while (NOT (> TEXP 0)) do (.LRSH1. THI TLO)
                                                    (CL:INCF TEXP))
                         (SETQ ROUNDINGBITS (LOGAND TLO 255))
                         (SETQ TLO (LOGAND TLO 65280))
                         ;; Round the  32-bit result to 24 bits to try and match F
                         (if [OR (IGREATERP ROUNDINGBITS 128)
                                 (AND (EQ ROUNDINGBITS 128)
                                      (NOT (EQ 0 (LOGAND TLO 256]
                             then (if (EQ TLO 65280)
                                      then (SETQ TLO 0)
                                           (if (EQ THI MAX.SMALL.INTEGER)
                                               then (SETQ THI \SIGNBIT)
                                                    (CL:INCF TEXP)
                                               else (CL:INCF THI))
                                  else (CL:INCF TLO 256)))
                         (if (OR (NOT (EQ FEXP TEXP))
                                 (NOT (EQ FHI THI))
                                 (NOT (EQ FLO TLO)))
                             then (CL:INCF LOCALK)
                                  (if (< LOCALK 10)
                                      then (GO MOREDIGITS]
            ;; Done! Convert integer mantissa to a string
                 (WITH-RESOURCES (\NUMSTR \NUMSTR1)
                        (\CONVERTNUMBER MANT 10 T NIL \NUMSTR \NUMSTR1)
                        (SETQ MANTSTRING \NUMSTR1))
            ;; If K came in NIL, clip trailing "0"s from mantissa string; if it came in bigger than 8, pad the string with 0s.
                 (if (NOT K)
                     then (LET [(ENDPOINTER (CL:1- (NCHARS MANTSTRING]
                               (while (EQ #\0 (CL:CHAR MANTSTRING ENDPOINTER)) do (GLC MANTSTRING)
                                                                                  (CL:INCF EXP)
                                                                                  (CL:DECF ENDPOINTER)))
                       else (freplace (ARRAY-HEADER FILL-POINTER-P) of MANTSTRING with T)
                            (freplace (ARRAY-HEADER TOTAL-SIZE) of MANTSTRING with 128)
                                                                  ; So VECTOR-PUSH will work...
                            (while (IGREATERP K LOCALK) do (CL:VECTOR-PUSH #\0 MANTSTRING)
                                                           (CL:DECF EXP)
                                                           (CL:INCF LOCALK)))
                 (RETURN (CL:VALUES MANTSTRING EXP])


(CL:DEFUN FLTINTLOG (FEXP FHI FLO)
   (DECLARE (GLOBALVARS EXTPOWERS))
   [LET ((RESULT (if (NOT (> FEXP 0))
                     then (LET (TEXP THI TLO)
                               (DECLARE (CL:SPECIAL TEXP THI TLO))
                               (TIMESPOW10 37 FEXP FHI FLO)
                               (CL:SETQ FEXP TEXP FHI THI FLO TLO)
                               -74)
                      else -37)))
        (FOR I FROM 76 TO 0 BIND TABENTRY
           DO (SETQ TABENTRY (CL:AREF EXTPOWERS I))
              (IF [OR (> FEXP (FETCH (EFPN EXP) OF TABENTRY))
                      (AND (EQ FEXP (FETCH (EFPN EXP) OF TABENTRY))
                           (OR (> FHI (FETCH (EFPN HI) OF TABENTRY))
                               (AND (EQ FHI (FETCH (EFPN HI) OF TABENTRY))
                                    (>= FLO (FETCH (EFPN LO) OF TABENTRY]
                THEN (RETURN (IPLUS RESULT I])


(CL:DEFUN DIGITSBDP (F)
   (LET (SIGNF FEXP FHI FLO)

;;; Returns the number of decimal places before the decimal point F has.

        (.FLOATUNBOX. F SIGNF FEXP FHI FLO)
        ;; Re-normalize
        (if (EQ 0 FHI)
            then (SETQ FHI FLO)
                 (SETQ FLO 0)
                 (CL:DECF FEXP 16))
        (while (EQ 0 (LOGAND FHI \SIGNBIT)) do (.LLSH1. FHI FLO)
                                               (CL:DECF FEXP))
        (FLTINTLOG FEXP FHI FLO)))


(CL:DEFUN INTTOEXT (N)

;;; Takes an integer N and returns a fixp exponent and a two-fixp mantissa (by setting non-locals (texp, thi, tlo)) . Ignores sign of N, range limitations of
;;; normal exponent (everything comes back "normalized").
```

```
   (LET ((EXP (IPLUS \EXPONENT.BIAS 31))
         HI LO)
        (if (EQ N 0)
            then (SETQ TEXP 0)
                 (SETQ THI 0)
                 (SETQ TLO 0)
          else (.UNBOX. N HI LO)
               [if (EQ 0 HI)
                   then (SETQ HI LO)
                        (SETQ LO 0)
                        (SETQ EXP (IDIFFERENCE EXP 16))
                        (while (EQ 0 (LOGAND HI \SIGNBIT)) do (SETQ HI (LLSH1 HI 1))
                                                              (SETQ EXP (SUB1 EXP)))
                 else (while (EQ 0 (LOGAND HI \SIGNBIT)) do (.LLSH1. HI LO)
                                                            (SETQ EXP (SUB1 EXP]
               (SETQ TEXP EXP)
               (SETQ THI HI)
               (SETQ TLO LO))))
```

```
(CL:DEFUN EXTTOINT (EXP HI LO)
```

;;; Takes a fixp exponent and a two-fixp mantissa and returns an integer which is the properly rounded integer. Ignores sign and out-of-range numbers.

```
   (SETQ EXP (IDIFFERENCE EXP \EXPONENT.BIAS))
   [if (NOT (EQ EXP 31))
       then (LET (ROUNDFLAG)
                 [if (< EXP 15)
                     then (SETQ LO HI)
                          (SETQ HI 0)
                          (CL:INCF EXP 16)
                          (SETQ LO (LRSH LO (- 30 EXP)))
                   else (while (NOT (EQ EXP 30)) do (.LRSH1. HI LO)
                                                    (SETQ EXP (ADD1 EXP]
                 (SETQ ROUNDFLAG (EQ 1 (LOGAND LO 1)))
                 (.LRSH1. HI LO)
                 (if ROUNDFLAG
                     then (if (EQ LO 65535)
                              then (SETQ LO 0)
                                   (CL:INCF HI)
                            else (CL:INCF LO]
   (\MAKENUMBER HI LO))
```

```
(DEFMACRO SPLIT8 (IN HI LO)
   `(CL:SETQ ,HI (LRSH ,IN 8)
             ,LO
             (LLSH (LOGAND ,IN 255)
                   8)))
```

```
(CL:DEFUN TIMESPOW10 (POWER EXP HI LO)
   (DECLARE (GLOBALVARS EXTPOWERS))
   [LET (TABENTRY CURPOWER)
        (CL:SETQ TEXP EXP THI HI TLO LO)
        (WHILE (NOT (EQ 0 POWER)) DO [COND
                                        ((> POWER 38)
                                         (SETQ CURPOWER 76))
                                        ((< POWER -38)
                                         (SETQ CURPOWER 0))
                                        (T (SETQ CURPOWER (+ POWER 38]
                                      (SETQ TABENTRY (CL:AREF EXTPOWERS (IABS CURPOWER)))
                                      ;; Results of this land in texp thi tlo (bound above, somewhere...)
                                      (\EXTFTIMES EXP HI LO (fetch (EFPN EXP)
                                                                   TABENTRY)
                                                  (fetch (EFPN HI)
                                                         TABENTRY)
                                                  (fetch (EFPN LO)
                                                         TABENTRY))
                                      (SETQ EXP TEXP)
                                      (SETQ HI THI)
                                      (SETQ LO TLO)
                                      (CL:DECF POWER (- CURPOWER 38])
```

```
(CL:DEFUN \EXTFTIMES (EXPX HX LX EXPY HY LY)
   (PROG ((SIGNX 0)
          (SIGNY 0)
          (HHZ 0)
          (HZ 0)
          (LZ 0)
          CARRY LOW8BITS FOO FOOHI FOOLO)
         (if (AND (EQ 0 EXPX)
                  (EQ 0 HX)
                  (EQ 0 LX))
             then (GO DONE))
```

```
          (if (AND (EQ 0 EXPY)
                   (EQ 0 HY)
                   (EQ 0 LY))
              then (GO DONE))
          (COND)
```

;;; Multiplying two 32-bit operands to get a 32-bit rounded product. Doing the multiplication 8 bits at a time and maintaining a 40-bit or so sum in (HHZ HZ ;;; LZ) that holds the lower order products as we go.

```
          (COND
             ((OR (EQ 0 HY)
                  (EQ 0 LY))                                    ; swap operands to make life easier
              (swap HX HY)
              (swap LX LY)))
          (CL:DOTIMES (I 4)
              (SETQ LOW8BITS (LOGAND LX 255))
              ;; Skip it if 8 bits are 0
              (if (NOT (EQ 0 LOW8BITS))
                  then (SETQ FOO (ITIMES LOW8BITS (LOGAND LY 255)))
                       (SETQ CARRY (.ADDSMALL2. LZ FOO))
                       (SETQ FOO (ITIMES LOW8BITS (LOGAND HY 255)))
                       (SETQ CARRY (.ADDSMALL3. HZ FOO CARRY))
                       (SETQ HHZ (IPLUS HHZ CARRY))
                       (SETQ FOO (ITIMES LOW8BITS (LRSH LY 8)))
                       (SPLIT8 FOO FOOHI FOOLO)
                       (SETQ CARRY (.ADDSMALL2. LZ FOOLO))
                       (SETQ FOOHI (IPLUS FOOHI CARRY))
                       (SETQ CARRY (.ADDSMALL2. HZ FOOHI))
                       (SETQ FOO (ITIMES LOW8BITS (LRSH HY 8)))
                       (SPLIT8 FOO FOOHI FOOLO)
                       (SETQ CARRY (IPLUS CARRY (.ADDSMALL2. HZ FOOLO)))
                       (SETQ HHZ (IPLUS HHZ FOOHI CARRY)))
              (.LRSH8. HX LX)                                   ; Shift over 8 bits
              ;; Don't shift sum the last time
              (if (NOT (EQ I 3))
                  then (.LRSH8. HZ LZ)
                       (SETQ HZ (LOGOR HZ (LLSH HHZ 8)))
                       (SETQ HHZ (LRSH HHZ 8))))
      ;; OK, shift (HHZ HZ LZ) over 8 so \EXTNORMALIZE can understand them
          (.LLSH8. HHZ HZ)
          (SETQ HZ (LOGOR HZ (LRSH LZ 8)))
          (SETQ LZ (LLSH (LOGAND LZ 255)
                         8))
      ;; We now have a 40-bit result in HHZ,HZ,LZ.  \EXTNORMALIZE can handle it from here.  Note that the exponent we give is bumped by 1, because
      ;; the 'binary point' , which was between the first and second bits, was moved one to the right by multiplying
       DONE
          (RETURN (\EXTNORMALIZE (IPLUS EXPX EXPY (IDIFFERENCE 1 \EXPONENT.BIAS))
                         HHZ HZ LZ))))


(CL:DEFUN \EXTFQUOTIENT (EXPX HX LX EXPY HY LY)
   (PROG ((SIGNX 0)
          (SIGNY 0)
          BORROW
          (HZ 0)
          (LZ 0))
          (if (AND (EQ EXPX 0)
                   (EQ HX 0)
                   (EQ LX 0))
              then (GO DONE))
          (if (AND (EQ EXPY 0)
                   (EQ HY 0)
                   (EQ LY 0))
              then (GO DIVZERO))
          (COND
             ((EQ EXPX \MAX.EXPONENT)                           ; X is infinity, result is too.
              (RETURN (CL:VALUES \MAX.EXPONENT 65535 65535)))
             ((EQ EXPY \MAX.EXPONENT)                           ; Y = infinity, result is zero
              (GO DONE)))
```

;;; Divide X -- double length, implicitly extended with zeros -- by Y. At each step, Y is subtracted from X if possible, putting a one bit in the quotient, and ;;; then X and the quotient are shifted left. Result is a 32-bit quotient.

```
          (.LRSH1. HX LX)
          (.LRSH1. HY LY)                                       ; shift these right one so that we never have to worry about
                                                                ; carrying out of the high bit
          (FRPTQ 31 (PROGN (.LLSH1. HZ LZ)                      ; shift quotient left one as we accumulate it
                           (COND
                              ((OR (AND (EQ HX HY)
                                        (IGEQ LX LY))
                                   (IGREATERP HX HY))           ; X GE Y, so subtract Y
                               (SETQ HX (IDIFFERENCE (IDIFFERENCE HX HY)
                                                     (.SUBSMALL. LX LY)))
```

```
                              (SETQ LZ (ADD1 LZ))                ; note that this never overflows, because of the left shift we did
                                                                 ; above

                              ))
                         ;; now shift dividend left one.  After the subtraction the high-order bit must be off, so this works okay
                         (.LLSH1. HX LX)))
          (.LLSH1. HZ LZ)                                        ; left shift result 1 to compensate for the earlier right shifts
          [COND
             ((OR (NEQ HX 0)
                  (NEQ LX 0))                                    ; set sticky bit
              (SETQ LZ (LOGOR LZ 1]
      DONE
          (RETURN (\EXTNORMALIZE (IPLUS (IDIFFERENCE EXPX EXPY)
                                        \EXPONENT.BIAS)
                                 HZ LZ))
      DIVZERO
          (SETQ TEXP \MAX.EXPONENT)
          (SETQ THI 65535)
          (SETQ TLO 65535)))


(CL:DEFUN \EXTNORMALIZE (EXP HI LO &OPTIONAL (ROUNDINGBITS 0))
    ;; Takes four fixps, one exponent and three mantissa, and returns a normalized, rounded exponent and two fixp mantissa. Does nothing about
    ;; exponent range or "denormalization"; just shifts mantissa, bumps exponent, and rounds
    (PROG NIL
          (COND
             ((AND (EQ 0 HI)
                   (EQ 0 LO))
              (SETQ EXP 0)
              (GO DONE)))
          [COND
             ((EQ 0 HI)
              (SETQ HI LO)
              (SETQ LO ROUNDINGBITS)
              (SETQ ROUNDINGBITS 0)
              (SETQ EXP (IDIFFERENCE EXP 16]
          (while (EQ 0 (LOGAND HI \SIGNBIT)) do (.LLSH1. HI LO)
                                                (if (NOT (EQ 0 (LOGAND ROUNDINGBITS \SIGNBIT)))
                                                    then (SETQ LO (ADD1 LO)))
                                                (SETQ ROUNDINGBITS (LLSH ROUNDINGBITS 1))
                                                (SETQ EXP (SUB1 EXP)))
                                                                 ; round result.
          [COND
             ([OR (IGREATERP ROUNDINGBITS \SIGNBIT)
                  (AND (EQ ROUNDINGBITS \SIGNBIT)
                       (NOT (EQ 0 (LOGAND LO 1]
              ;; round up if the left over fraction was greater than 1/2;  if it was equal to a half, round to the even result
              (COND
                 [(EQ LO MAX.SMALL.INTEGER)                      ; can't add 1 directly
                  (SETQ LO 0)
                  (COND
                     ((EQ HI MAX.SMALL.INTEGER)
                      (SETQ HI \SIGNBIT)
                      (add EXP 1]
                 (T (SETQ LO (ADD1 LO]
      DONE

      ;; Stuff results in texp thi tlo (bound somewhere above... hopefully...)
          (SETQ TEXP EXP)
          (SETQ THI HI)
          (SETQ TLO LO)))


(CL:DEFUN \CONVERT.FLOATING.NUMBER (F OUTSTR OUTSTRPTR FORMAT)
    (DECLARE (GLOBALVARS \FLOATFORMAT))
    [WITH-RESOURCE
     (\CFNSTRING)
     (CL:MACROLET
      [(STRPUT (C)
               `(CL:SETF (CL:CHAR OUTSTR (CL:INCF OSINDEX))
                         ,C]
      (OR FORMAT (SETQ FORMAT \FLOATFORMAT))
      (IF (NOT (CL:CONSP FORMAT))
          THEN (SETQ FORMAT NIL))
      (LET ((OSINDEX -1)
            [MINUSFLAG (AND (MINUSP F)
                            (SETQ F (- F]
            (PADSIZE 0)
            (PADCHAR #\Space)
            NUMSTR INTEXP NSWIDTH)
           [DESTRUCTURING-BIND
            (WIDTH DECPART EXPPART PAD0 ROUND)
            (CDR FORMAT)
```

```
        ;; Clip WIDTH to maximum size of buffer strings (128)
        (AND WIDTH (IGREATERP WIDTH 128)
             (SETQ WIDTH 128))
        (COND
           ((NULL EXPPART)
            (SETQ EXPPART 0)))
        (COND
           (PAD0 (SETQ PADCHAR #\0)))
        (IF [OR (NOT (EQ EXPPART 0))
                (AND (NOT (ZEROP F))
                     (OR (< F 0.001)
                         (>= F 1.0E+7]
            THEN ;; Exponential required - too big/small or explicitly requested
                 (IF (AND (NOT ROUND)
                          DECPART)
                     THEN (SETQ ROUND (CL:1+ DECPART)))
                 (CL:MULTIPLE-VALUE-SETQ (NUMSTR INTEXP)
                      (FLTSTR F ROUND))
                 (IF (AND ROUND (NOT DECPART)                    ; we might be able to remove some trailing zeros here
                          (NOT (ZEROP F)))
                     THEN (FOR STREND FROM (CL:1- (CL:LENGTH NUMSTR)) TO 0 BY -1
                               WHILE (EQ #\0 (CL:CHAR NUMSTR STREND)) DO (GLC NUMSTR)
                                                                         (CL:INCF INTEXP)))
                 (SETQ NUMSTR (ENUM-STRING \CFNSTRING NUMSTR INTEXP DECPART EXPPART))
          ELSE ;; Floating decimal printing
               [IF (AND (NOT ROUND)
                        DECPART)
                   ;; Foo! Must compute round from decpart to round to a specific decimal place
                   THEN (SETQ ROUND (IF (ZEROP F)
                                        THEN 1
                                      ELSE (MAX 0 (MIN 9 (+ (CL:TRUNCATE (CL:1+ (CL:LOG F 10)))
                                                            DECPART]
               (CL:MULTIPLE-VALUE-SETQ (NUMSTR INTEXP)
                    (FLTSTR F ROUND))
               (IF (AND ROUND (NOT DECPART)                      ; we might be able to remove some trailing zeros here
                        (NOT (ZEROP F)))
                   THEN (FOR STREND FROM (CL:1- (CL:LENGTH NUMSTR)) TO 0 BY -1
                             WHILE (EQ #\0 (CL:CHAR NUMSTR STREND)) DO (GLC NUMSTR)
                                                                       (CL:INCF INTEXP)))
               (SETQ NUMSTR (FNUM-STRING \CFNSTRING NUMSTR INTEXP DECPART)))
        (SETQ NSWIDTH (CL:LENGTH \CFNSTRING))                    ; Handle padding
        [IF (AND WIDTH (> WIDTH NSWIDTH))
            THEN (SETQ PADSIZE (- WIDTH NSWIDTH (CL:IF MINUSFLAG
                                                    1
                                                    0)]
        (AND PAD0 MINUSFLAG (STRPUT #\-))                        ; Minus sign before 0 pad
        (CL:DOTIMES (I PADSIZE)
            (STRPUT PADCHAR))
        (AND (NOT PAD0)
             MINUSFLAG
             (STRPUT #\-))                                       ; But after blank pad
        (CL:DOTIMES (I NSWIDTH)
            (STRPUT (CL:CHAR \CFNSTRING I)))]
       (SUBSTRING OUTSTR 1 (CL:1+ OSINDEX)
              OUTSTRPTR])
```

```
(CL:DEFUN \FLOATINGSCALE (INTMANT INTEXP &OPTIONAL BOX)
```

;;; Takes an integer mantissa and integer exponent and returns a floating-point number F such that F = intmant * 10**intexp. Smashes it into box if one
;;; is supplied

```
   (LET (TEXP THI TLO (SIGN 0))
        (DECLARE (CL:SPECIAL TEXP THI TLO))
        (IF (MINUSP INTMANT)
            THEN (SETQ INTMANT (- INTMANT))
                 (SETQ SIGN 1))
        (INTTOEXT INTMANT)                                       ; leaves result in texp thi tlo
        (TIMESPOW10 INTEXP TEXP THI TLO)                         ; ditto
        (\MAKEFLOAT SIGN TEXP THI TLO T BOX)))

(DEFINEQ
```

(\INIT.POWERS.OF.TEN

```
  [LAMBDA NIL                                                    ; Edited 14-Jan-87 11:21 by jrb:
```

;; Initialize array \POWERS.OF.TEN to values 10^-29 thru 10^+29. I suppose I could have the array cover the entire range of floats, but the range
;; is asymmetric and the numbers start losing significance at the ends, so it's not really worth it. Also initialize the array of 32-bit mantissa powers of
;; 10 used by \convert.floating.number and friends.

```
    (SETQ \POWERS.OF.TEN (ARRAY 59 'POINTER))
    (SETQ EXTPOWERS (CL:MAKE-ARRAY 77))
    (SETA \POWERS.OF.TEN 30 1.0)
    (CL:SETF (CL:AREF EXTPOWERS 38)
```

```
            (CREATE EFPN
                   EXP _ \EXPONENT.BIAS
                   HI _ 32768
                   LO _ 0))
    (for I from 1 to 29 bind (POWTEN _ 1.0) do (SETA \POWERS.OF.TEN (IPLUS I 30)
                                                     (SETQ POWTEN (FTIMES POWTEN 10.0)))
                                               (SETA \POWERS.OF.TEN (IDIFFERENCE 30 I)
                                                     (FQUOTIENT 1.0 POWTEN)))
    [LET (ENOW EEXP EHI ELO TEXP THI TLO)
         (DECLARE (CL:SPECIAL TEXP THI TLO))
         ;; First generate the powers of ten exactly representable in 32 bits
         (for I from 1 to 9 bind (POW10 _ 1)
            do (INTTOEXT (SETQ POW10 (CL:* POW10 10)))          ; Results in texp thi tlo
               (CL:SETF (CL:AREF EXTPOWERS (+ 38 I))
                      (CREATE EFPN
                             EXP _ TEXP
                             HI _ THI
                             LO _ TLO)))
         (SETQ ENOW (CL:AREF EXTPOWERS (+ 38 9)))
         (SETQ EEXP (FETCH (EFPN EXP)
                           ENOW))
         (SETQ EHI (FETCH (EFPN HI)
                          ENOW))
         (SETQ ELO (FETCH (EFPN LO)
                          ENOW))
         (for I from 1 to 4 do (TIMESPOW10 I EEXP EHI ELO)       ; this does out to 10^13
                               (CL:SETF (CL:AREF EXTPOWERS (+ I 9 38))
                                      (CREATE EFPN
                                             EXP _ TEXP
                                             HI _ THI
                                             LO _ TLO)))
         ;; Then use them to generate the others
         (SETQ ENOW (CL:AREF EXTPOWERS (+ 38 13)))
         (SETQ EEXP (FETCH (EFPN EXP)
                           ENOW))
         (SETQ EHI (FETCH (EFPN HI)
                          ENOW))
         (SETQ ELO (FETCH (EFPN LO)
                          ENOW))
         (for I from 1 to 13 do (TIMESPOW10 I EEXP EHI ELO)      ; this does out to 10^26
                                (CL:SETF (CL:AREF EXTPOWERS (+ I 13 38))
                                       (CREATE EFPN
                                              EXP _ TEXP
                                              HI _ THI
                                              LO _ TLO)))
         (SETQ ENOW (CL:AREF EXTPOWERS (+ 38 26)))
         (SETQ EEXP (FETCH (EFPN EXP)
                           ENOW))
         (SETQ EHI (FETCH (EFPN HI)
                          ENOW))
         (SETQ ELO (FETCH (EFPN LO)
                          ENOW))
         (for I from 1 to 12 do (TIMESPOW10 I EEXP EHI ELO)      ; this does out to 10^38
                                (CL:SETF (CL:AREF EXTPOWERS (+ I 26 38))
                                       (CREATE EFPN
                                              EXP _ TEXP
                                              HI _ THI
                                              LO _ TLO)))
         ;; Finally generate all the inverses: 10^-1 - 10^-38
         (for I from 1 to 38 do (SETQ ENOW (CL:AREF EXTPOWERS (+ I 38)))
                                (\EXTFQUOTIENT \EXPONENT.BIAS 32768 0 (FETCH (EFPN EXP)
                                                                             ENOW)
                                       (FETCH (EFPN HI)
                                              ENOW)
                                       (FETCH (EFPN LO)
                                              ENOW))
                                (CL:SETF (CL:AREF EXTPOWERS (- 38 I))
                                       (CREATE EFPN
                                              EXP _ TEXP
                                              HI _ THI
                                              LO _ TLO]
    \POWERS.OF.TEN])

)

(DECLARE%: DONTCOPY

(DECLARE%: EVAL@COMPILE

[PUTDEF '\CFNSTRING 'RESOURCES '(NEW (CL:MAKE-ARRAY 128 :ELEMENT-TYPE 'CL:STRING-CHAR :FILL-POINTER 0
                                          :ADJUSTABLE T]
)

(DECLARE%: DOEVAL@COMPILE DONTCOPY
```

```
(GLOBALVARS \POWERS.OF.TEN)
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS \POWER.OF.TEN MACRO ((N)
                               (ELT \POWERS.OF.TEN (IPLUS N 30))))
)
)

(DECLARE%: DONTEVAL@LOAD DOCOPY

(/SETTOPVAL '\\CFNSTRING.GLOBALRESOURCE NIL)

(\INIT.POWERS.OF.TEN)
)

(PUTPROPS \UNBOXFLOAT1 ARGNAMES (X OP))

(PUTPROPS \UNBOXFLOAT2 ARGNAMES (X Y OP))

(PUTPROPS \UNBOXFLOAT3 ARGNAMES (X Y Z OP))

(PUTPROPS LLFLOAT FILETYPE CL:COMPILE-FILE)

(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR NLAMA )

(ADDTOVAR NLAML )

(ADDTOVAR LAMA FPLUS FTIMES)
)

(PUTPROPS LLFLOAT COPYRIGHT ("Venue & Xerox Corporation" 1982 1984 1985 1986 1987 1988 1990 1991 1993))
```

FUNCTION INDEX

BCPL.TO.FLOATP 15
DIGITSBDP 18
ENUM-STRING 16
EXTTOINT 19
FABS 3
FDIFFERENCE 2
FEQP 4
FGREATERP 3
FIXR 5
FLOAT 4
FLOATP.TO.BCPL 15
FLTINTLOG 18
FLTSTR 17
FNUM-STRING 17
FPLUS 2
FQUOTIENT 2
FTIMES 2
INTTOEXT 18
MAKEFLOATNUMBER 9
PutFloat 9
SQRT 9
TIMESPOW10 19
\BOXFPLUSDIF 6
\BOXFQUOTIENT 6
\BOXFTIMES2 7
\CONVERT.FLOATING.NUMBER 21
\EXTFQUOTIENT 20
\EXTFTIMES 19
\EXTNORMALIZE 21
\FIXP.FROM.FLOATP 4
\FLOAT 4
\FLOATINGSCALE 22
\FZEROP 4
\GETBASEFLOATP 2
\INFINITY 8
\INIT.POWERS.OF.TEN 22
\MAKEFLOAT 8
\MATMULT133 13
\MATMULT144 13
\MATMULT331 14
\MATMULT333 14
\MATMULT441 14
\MATMULT444 14
\PUTBASEFLOATP 2
\SLOWFDIFFERENCE 3
\SLOWFGREATERP 3
\SLOWFPLUS2 3
\SLOWFQUOTIENT 3
\SLOWFTIMES2 3
\UNBOXFLOAT1 12
\UNBOXFLOAT2 13
\UNBOXFLOAT3 13

MACRO INDEX

.ADDSMALL2. 12
.ADDSMALL3. 12
.FLOATUNBOX. 10
.LLSH1. 11
.LLSH8. 11
.LRSH1. 11
.LRSH8. 11
.LRSHSTICKY. 11
.POWEROF2. 12
.SUBSMALL. 12
POLYEVAL 1
SPLIT8 19
ZEROP 9
\.GETBASE32 2
\.PUTBASE32 2
\CALLER.ARGS 14
\GETBASEFLOATP 2
\HAND.FLOATUNBOX ..1
\POWER.OF.TEN 24
\PUTBASEFLOATP 2

CONSTANT INDEX

MAX.DIGITS.ACCURACY 10
\8BITS 10
\EXPONENT.BIAS 10
\HIDDENBIT 10
\MAX.EXPONENT 10
\MAX.HI.FRAC 10
\SIGNBIT 10

PROPERTY INDEX

LLFLOAT 24
\UNBOXFLOAT1 24
\UNBOXFLOAT2 24
\UNBOXFLOAT3 24

VARIABLE INDEX

INTPOWERS 16
MAX.FLOAT 12
MIN.FLOAT 12
\UNDERFLOW 12

RECORD INDEX

BCPLNUM 16
EFPN 16
FLOATP 10