File created: 24-Sep-2023 15:37:27 **{WMEDLEY}<sources>CMLARITH.;3**

edit by: rmk

previous date: 23-Sep-2023 23:15:39 {WMEDLEY}<sources>CMLARITH.;2

Read Table: XCL

Package: LISP

Format: XCCS

```
(IL:RPAQQ IL:CMLARITHCOMS
          (
```

;;; Common Lisp Arithmetic

```
          (IL:COMS
                 ;; Error utilities
                 (IL:FUNCTIONS %NOT-NUMBER-ERROR %NOT-NONCOMPLEX-NUMBER-ERROR %NOT-INTEGER-ERROR
                        %NOT-RATIONAL-ERROR %NOT-FLOAT-ERROR))
          (IL:COMS
```

;;; Section 2.1.2 Ratios.

```
                 (IL:COMS (IL:STRUCTURES RATIO)
                        ;; The following makes NUMBERP true on ratios
                        (IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOCOPY (IL:P (IL:\\SETTYPEMASK (
                                                                              IL:\\TYPENUMBERFROMNAME
                                                                                'RATIO)
                                                                       (IL:LOGOR IL:\\TT.NUMBERP
                                                                                 IL:\\TT.ATOM)))))
                 (IL:FUNCTIONS DENOMINATOR NUMERATOR RATIONALP %RATIO-PRINT %BUILD-RATIO RATIONAL RATIONALIZE)
                 (IL:FUNCTIONS %RATIO-PLUS %RATIO-TIMES))
          (IL:COMS
```

;;; Section 2.1.4 Complex Numbers.

```
                 (IL:COMS (IL:STRUCTURES COMPLEX)
                        ;; So we don't inherit the deftype from defstruct
                        (IL:TYPES COMPLEX)
                        ;; Make Complex  NUMBERP
                        (IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOCOPY (IL:P (IL:\\SETTYPEMASK (
                                                                              IL:\\TYPENUMBERFROMNAME
                                                                                'COMPLEX)
                                                                       (IL:LOGOR IL:\\TT.NUMBERP
                                                                                 IL:\\TT.ATOM)))))
                 (IL:FUNCTIONS COMPLEX REALPART IMAGPART CONJUGATE PHASE %COMPLEX-PRINT %COMPLEX-+ %COMPLEX--
                        %COMPLEX-* %COMPLEX-/ %COMPLEX-ABS))
          (IL:COMS
```

;;; Datatype predicates

```
                 ;; cl:integerp is defined in cmlpred (has an optimizer)
                 ;; cl:floatp is defined in cmltypes (has an optimizer). il:floatp is defined on llbasic
                 ;; cl:complexp is a defstruct predicate (compiles in line)
                 ;; cl:numberp is defined in cmltypes (has an optimizer). il:numberp is defined on llbasic
                 )
          (IL:COMS
```

;;; Section 12.2 Predicates on Numbers (generic).

```
                 ;; cl:zerop  is not shared with il:zerop, although they are equivalent. There is no il;plusp
                 (IL:COMS (IL:FUNCTIONS ZEROP PLUSP)
                        (XCL:OPTIMIZERS ZEROP PLUSP))
                 ;; cl:minusp is shared with il:minusp, but must be redefined to work with ratios. Old version resides in llarith
                 (IL:COMS (IL:FUNCTIONS MINUSP)
                        (XCL:OPTIMIZERS MINUSP))
                 ;; Both cl:evenp and cl:oddp are shared with il:. The functions are extended by allowing a second optional modulus argument.
                 ;; Another version of il:oddp exists on llarith, but the definition of il:evenp has disappeared
                 (IL:COMS (IL:FUNCTIONS EVENP ODDP)
                        (XCL:OPTIMIZERS EVENP ODDP)))
          (IL:COMS
```

;;; Section 12.3 Comparisons on Numbers. (generic)

```
                 (IL:COMS (IL:FUNCTIONS %= %/= %> %< %>= %<=)
                        (IL:PROP IL:DOPVAL %= %> %<)
                                                                ; For the byte compiler
```

```
                (IL:PROP IL:DMACRO %> %< %>= %<=)
                (IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOCOPY (IL:P
                                                        ;; Backward compatibility

                                                        ;  il:%= is listed as the punt function for the = opcode
                                                                (IL:MOVD '%= 'IL:%=)

                                                        ; Greaterp is the UFN for the greaterp opcode. Effectively
                                                        ; redefines the opcode
                                                                (IL:MOVD '%> 'IL:GREATERP)

                                                        ; Interlisp Greaterp and Lessp are defined in llarith
                                                                (IL:MOVD '%< 'IL:LESSP))))
        ;; =, <, >, <=, and >= are shared with il:, but cl:/= is NOT shared (?!)
        (IL:COMS (IL:DECLARE\: IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FUNCTIONS %COMPARISON-MACRO))
                (IL:FNS = /= < > <= >=)
                (IL:FUNCTIONS %COMPARISON-OPTIMIZER)
                (XCL:OPTIMIZERS = /= < > <= >=))
        ;; Note: the related predicates EQL, EQUAL, and EQUALP should be consulted if any of the above change. EQL is on LLNEW
        ;; (?), EQUAL and EQUALP on CMLTYPES.
        ;; cl:min and cl:max are shared with il: (defined in llarith). They  are written in terms of GREATERP and hence work on ratios.
        ;; Note (min) returns #.max.integer , which is an extension on the CLtl spec. We only optimize the case of two args
        (XCL:OPTIMIZERS MIN MAX))
(IL:COMS
```

;;; Section 12.4 Arithmetic Operations (generic).

```
        (IL:COMS (IL:FUNCTIONS %+ %- %* %/)
                                                        ; NOTE: %/ cannot compile out to the existinq quotient opcode
                                                        ; because it produces ratios rather than truncating
                (IL:PROP IL:DOPVAL %+ %- %*)
                                                        ; For the byte compiler
                (IL:PROP IL:DMACRO %+ %- %*)
                (IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOCOPY (IL:P
                                                        ;; Backward compatibility
                                                                (IL:MOVD '%/ 'IL:%/)

                                                            ;; Redefine UFNs for generic plus, difference, and times.
                                                            ;; Old UFN defined in llarith.
                                                               (IL:MOVD '%+ 'IL:\\SLOWPLUS2)
                                                               (IL:MOVD '%- 'IL:\\SLOWDIFFERENCE)
                                                               (IL:MOVD '%* 'IL:\\SLOWTIMES2))))
        (IL:COMS (IL:FNS + - * /)
                (IL:FUNCTIONS 1+ 1- %RECIPROCOL)
                (XCL:OPTIMIZERS + - * / 1+ 1-)
                                                        ; For the byte compiler
                (IL:PROP IL:DMACRO + *)
                ;; Redefine Interlisp generic arithmetic to work with ratios
                (IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOCOPY (IL:P (IL:MOVD '+ 'IL:PLUS)

                                                        ;; Don't need to redefine difference since it is defined in terms of the
                                                        ;; difference opcode (redefined above)
                                                                (IL:MOVD '* 'IL:TIMES)
                                                                ;; So Interlisp quotient will do something reasonable
                                                                ;; with ratios
                                                                (IL:* (IL:MOVD 'IL:NEW-QUOTIENT
                                                                        'IL:QUOTIENT))

                                                            ;; because QUOTIENT is already defined in LLARITH to do
                                                            ;; something useful with ratios. AR 8062.
                                                                )))
        ;; INCF and DECF implemented by CMLSETF.
        (IL:FUNCTIONS %GCD %LCM)
        (IL:FNS GCD LCM))
(IL:COMS
        ;; Optimizers for Interlisp functions, so that they compile open with the PavCompiler.
        ;; optimizer of IL:minus
        (XCL:OPTIMIZERS IL:MINUS)
        (XCL:OPTIMIZERS IL:PLUS IL:IPLUS IL:FPLUS IL:TIMES IL:ITIMES IL:FTIMES IL:RSH)
        (IL:PROP IL:DOPVAL IL:PLUS2 IL:IPLUS2 IL:FPLUS2 IL:TIMES2 IL:ITIMES2 IL:FTIMES2))
(IL:COMS
```

;;; Section 12.5 Irrational and Transcendental functions. Most of these will be found on cmlfloat.

```
        (IL:FUNCTIONS ISQRT)
        ;; Abs is shared with il: abs ia also defined in llarith.
        (IL:FUNCTIONS ABS %ABS)
```

```
                (IL:FUNCTIONS SIGNUM %SIGNUM))
        (IL:COMS
```

;;; Section 12.6 Type Conversions and Component Extractions on Numbers.

```
                ;; Float implemented in cmlfloat
                (IL:DECLARE\: IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FILES IL:UNBOXEDOPS))
                                                        ; These should be exported from xcl
                (IL:COMS (IL:FUNCTIONS XCL::STRUNCATE XCL::SFLOOR XCL::SCEILING XCL::SROUND)
                        (XCL:OPTIMIZERS XCL::STRUNCATE XCL::SROUND))
                                                        ; Round is shared with il: (?!)
                (IL:COMS (IL:DECLARE\: IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FUNCTIONS %INTEGER-COERCE-MACRO))
                        (IL:FUNCTIONS TRUNCATE FLOOR CEILING ROUND)
                        (IL:FUNCTIONS %INTEGER-COERCE-OPTIMIZER)
                        (XCL:OPTIMIZERS TRUNCATE FLOOR CEILING ROUND))
                (IL:COMS (IL:FUNCTIONS FTRUNCATE FFLOOR FCEILING FROUND)
                        (XCL:OPTIMIZERS FTRUNCATE FFLOOR FCEILING FROUND))
                (IL:COMS (IL:FUNCTIONS MOD REM))
                ;; Should IL:remainder be equivalent to cl:rem?. Thereis no IL:mod in the IRM, although it has a macro which makes it
                ;; equivalent to imod.
                ;; See cmlfloat for ffloor and friends, decode-float and friends
                )
        (IL:COMS
```

;;; Section 12.7 Logical Operations on Numbers.

```
                ;; LOGXOR and LOGAND are shared with IL. (definitions in llarith)
                (IL:COMS (IL:FUNCTIONS %LOGICAL-OPTIMIZER)
                        (XCL:OPTIMIZERS LOGXOR LOGAND))
                (IL:COMS (IL:FUNCTIONS %LOGIOR %LOGEQV)
                        (IL:PROP IL:DOPVAL %LOGIOR)
                                                        ; for the byte compiler
                        (IL:PROP IL:DMACRO %LOGIOR))
                (IL:COMS (IL:FNS LOGIOR LOGEQV)
                        (XCL:OPTIMIZERS LOGIOR LOGEQV))
                (IL:COMS (IL:FUNCTIONS LOGNAND LOGNOR LOGANDC1 LOGANDC2 LOGORC1 LOGORC2)
                        (XCL:OPTIMIZERS LOGNAND LOGNOR LOGANDC1 LOGANDC2 LOGORC1 LOGORC2))
                (IL:COMS (IL:VARIABLES BOOLE-CLR BOOLE-SET BOOLE-1 BOOLE-2 BOOLE-C1 BOOLE-C2 BOOLE-AND
                                BOOLE-IOR BOOLE-XOR BOOLE-EQV BOOLE-NAND BOOLE-NOR BOOLE-ANDC1 BOOLE-ANDC2
                                BOOLE-ORC1 BOOLE-ORC2)
                        (IL:FUNCTIONS BOOLE))
                ;; Lognot is shared with IL.(in addarith)
                (IL:COMS (IL:FUNCTIONS LOGTEST LOGBITP)
                        (XCL:OPTIMIZERS LOGTEST))
                (IL:COMS (IL:FUNCTIONS ASH)
                        (IL:PROP IL:DOPVAL ASH)
                                                        ; For the byte compiler
                        (IL:PROP IL:DMACRO ASH))
                (IL:COMS (IL:FUNCTIONS LOGCOUNT %LOGCOUNT)
                        (IL:DECLARE\: IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FILES (IL:LOADCOMP)
                                                                        IL:LLBIGNUM))
                                                        ; Should be in llbignum
                        (IL:FUNCTIONS %BIGNUM-LOGCOUNT))
                (IL:FUNCTIONS INTEGER-LENGTH)
                ;; OPTIMIZERS FOR IL:LLSH AND IL:LRSH
                (IL:COMS (IL:FUNCTIONS %LLSH8 %LLSH1 %LRSH8 %LRSH1)
                        (IL:PROP IL:DOPVAL %LLSH8 %LLSH1 %LRSH8 %LRSH1)
                        (XCL:OPTIMIZERS IL:LLSH IL:LRSH)))
        (IL:COMS
```

;;; Section 12.8 Byte Manipulations Functions.

```
                (IL:COMS (IL:FUNCTIONS BYTE BYTE-SIZE BYTE-POSITION)
                        ;; Byte doesn't need an optimizer since the side-effects data-base will do constant folding, but the byte-compiler can
                        ;; profit from an optimizer
                        (IL:FUNCTIONS OPTIMIZE-BYTE)
                        (IL:PROP IL:DMACRO BYTE))
                (IL:COMS (IL:DECLARE\: IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FUNCTIONS %MAKE-BYTE-MASK-1
                                                                        %MAKE-BYTE-MASK-0))
                        (IL:FUNCTIONS LDB DPB MASK-FIELD DEPOSIT-FIELD)
                        (IL:FUNCTIONS %CONSTANT-BYTESPEC-P)
                        (XCL:OPTIMIZERS LDB DPB MASK-FIELD DEPOSIT-FIELD))
                (IL:COMS (IL:FUNCTIONS LDB-TEST)
                        (XCL:OPTIMIZERS LDB-TEST)))
        (IL:DECLARE\: IL:DONTCOPY IL:DOEVAL@COMPILE (IL:LOCALVARS . T))
        (IL:PROP (IL:MAKEFILE-ENVIRONMENT IL:FILETYPE)
                IL:CMLARITH)
        (IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY IL:COMPILERVARS
                (IL:ADDVARS (IL:NLAMA)
                        (IL:NLAML)
                        (IL:LAMA LOGEQV LOGIOR LCM GCD / * - + >= <= > < /= =)))))
```

```
;;; Common Lisp Arithmetic
;; Error utilities

(DEFUN %NOT-NUMBER-ERROR (OBJECT)
   (ERROR 'XCL:TYPE-MISMATCH :EXPECTED-TYPE 'NUMBER :NAME OBJECT :VALUE OBJECT))

(DEFUN %NOT-NONCOMPLEX-NUMBER-ERROR (OBJECT)
   (IF (NOT (NUMBERP OBJECT))
       (ERROR 'XCL:TYPE-MISMATCH :EXPECTED-TYPE 'NUMBER :NAME OBJECT :VALUE OBJECT)
       (ERROR "Arg a complex number~%~s" OBJECT)))

(DEFUN %NOT-INTEGER-ERROR (OBJECT)
   (ERROR 'XCL:TYPE-MISMATCH :EXPECTED-TYPE 'INTEGER :NAME OBJECT :VALUE OBJECT))

(DEFUN %NOT-RATIONAL-ERROR (OBJECT)
   (ERROR 'XCL:TYPE-MISMATCH :EXPECTED-TYPE 'RATIONAL :VALUE OBJECT :NAME OBJECT))

(DEFUN %NOT-FLOAT-ERROR (OBJECT)
   (ERROR 'XCL:TYPE-MISMATCH :EXPECTED-TYPE 'FLOAT :NAME OBJECT :VALUE OBJECT))

;;; Section 2.1.2 Ratios.

(DEFSTRUCT (RATIO (:CONSTRUCTOR %MAKE-RATIO (NUMERATOR DENOMINATOR))
                  (:PREDICATE %RATIO-P)
                  (:COPIER NIL)
                  (:PRINT-FUNCTION %RATIO-PRINT))
   (NUMERATOR :READ-ONLY)
   (DENOMINATOR :READ-ONLY))

;; The following makes NUMBERP true on ratios

(IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOCOPY

(IL:\\SETTYPEMASK (IL:\\TYPENUMBERFROMNAME 'RATIO)
       (IL:LOGOR IL:\\TT.NUMBERP IL:\\TT.ATOM))
)

(DEFUN DENOMINATOR (RATIONAL)
   ;; Returns the denominator of a rational.
   (TYPECASE RATIONAL
       (RATIO (RATIO-DENOMINATOR RATIONAL))
       (INTEGER 1)
       (T (%NOT-RATIONAL-ERROR RATIONAL))))

(DEFUN NUMERATOR (RATIONAL)
   ;; Returns the numerator of a rational.
   (TYPECASE RATIONAL
       (RATIO (RATIO-NUMERATOR RATIONAL))
       (INTEGER RATIONAL)
       (T (%NOT-RATIONAL-ERROR RATIONAL))))

(XCL:DEFINLINE RATIONALP (NUMBER)
   (OR (INTEGERP NUMBER)
       (%RATIO-P NUMBER)))

(DEFUN %RATIO-PRINT (NUMBER STREAM)
   (LET ((TOP (RATIO-NUMERATOR NUMBER))
         (BOTTOM (RATIO-DENOMINATOR NUMBER))
         PR)
        (COND
           ((NOT (IL:|fetch| (READTABLEP IL:COMMONNUMSYNTAX) IL:|of| *READTABLE*))
                                                    ; Can't print nice ratios to old read tables
            (IL:PRIN1 "|." STREAM)
            (IL:\\PRINDATUM (LIST '/ TOP BOTTOM)
                   STREAM))
           (T ;; If *PRINT-RADIX* is true, need to print radix prefix.  Of course, want it on whole ratio and not components, so we rebind to NIL
              ;; inside here.
              (IF *PRINT-RADIX*
                  (SETQ PR (CONCATENATE 'STRING (STRING (CODE-CHAR (IL:|fetch| (READTABLEP IL:HASHMACROCHAR)
                                                                      IL:|of| *READTABLE*)))
```

```
                                (CASE *PRINT-BASE*
                                    (2                              ; Binary
                                       "b")
                                    (8 "o")
                                    (16 "x")
                                    (T                              ; generalized radix prefix, even for decimal!
                                       (CONCATENATE 'STRING (LET* ((X *PRINT-BASE*)
                                                                   (*PRINT-BASE* 10)
                                                                   (*PRINT-RADIX* NIL))
                                                              (PRINC-TO-STRING X))
                                                            "r"))))))
            (IL:.SPACECHECK. STREAM (+ 1 (IL:NCHARS TOP)
                                       (IL:NCHARS BOTTOM)
                                       (IF PR
                                           (IL:NCHARS PR)
                                           0)))
            (LET ((IL:\\THISFILELINELENGTH NIL)
                  (*PRINT-RADIX* NIL))
                 (DECLARE (IL:SPECVARS IL:\\THISFILELINELENGTH))
                                                                    ; Turn off linelength check just in case the NCHARS count is off
                                                                    ; because of radices
                 (IF PR (IL:\\SOUT PR STREAM))
                 (IL:\\PRINDATUM TOP STREAM)
                 (IL:\\SOUT "/" STREAM)
                 (IL:\\PRINDATUM BOTTOM STREAM))))))
```

(DEFUN **%BUILD-RATIO** (X Y)

;; %BUILD-RATIO takes two integer arguments and builds the rational number which is their quotient.

```
(LET ((REM (IL:IREMAINDER X Y)))
     (IF (EQ 0 REM)
         (IL:IQUOTIENT X Y)
         (LET ((GCD (%GCD X Y)))
              (WHEN (NOT (EQ GCD 1))
                  (SETQ X (IL:IQUOTIENT X GCD))
                  (SETQ Y (IL:IQUOTIENT Y GCD)))
              (IF (MINUSP Y)
                  (%MAKE-RATIO (- X)
                               (- Y))
                  (%MAKE-RATIO X Y))))))
```

(DEFUN **RATIONAL** (NUMBER)

;; Rational produces a rational number for any numeric argument. Rational assumed that the floating point is completely accurate.

```
(TYPECASE NUMBER
    (RATIONAL NUMBER)
    (FLOAT (%RATIONAL-FLOAT NUMBER))
    (COMPLEX (%MAKE-COMPLEX (RATIONAL (COMPLEX-REALPART NUMBER))
                 (RATIONAL (COMPLEX-IMAGPART NUMBER))))
    (OTHERWISE (%NOT-NUMBER-ERROR NUMBER))))
```

(DEFUN **RATIONALIZE** (NUMBER)

;; Rationalize does a rational, but it assumes that floats are only accurate to their precision, and generates a good rational aproximation of them.

```
(TYPECASE NUMBER
    (RATIONAL NUMBER)
    (FLOAT (%RATIONALIZE-FLOAT NUMBER))
    (COMPLEX (%MAKE-COMPLEX (RATIONALIZE (COMPLEX-REALPART NUMBER))
                 (RATIONALIZE (COMPLEX-IMAGPART NUMBER))))
    (T (%NOT-NUMBER-ERROR NUMBER))))
```

(DEFUN **%RATIO-PLUS** (NUMERATOR-1 DENOMINATOR-1 NUMERATOR-2 DENOMINATOR-2)

```
  (LET ((GCD-D (%GCD DENOMINATOR-1 DENOMINATOR-2)))
       (IF (EQ GCD-D 1)
           (%MAKE-RATIO (+ (* NUMERATOR-1 DENOMINATOR-2)
                           (* NUMERATOR-2 DENOMINATOR-1))
                 (* DENOMINATOR-1 DENOMINATOR-2))
           (LET* ((D1/GCD-D (IL:IQUOTIENT DENOMINATOR-1 GCD-D))
                  (TOP (+ (* NUMERATOR-1 (IL:IQUOTIENT DENOMINATOR-2 GCD-D))
                          (* NUMERATOR-2 D1/GCD-D)))
                  (GCD-TOP (%GCD TOP GCD-D))
                  (D2/GCD-TOP DENOMINATOR-2))
                 (UNLESS (EQ GCD-TOP 1)
                     (SETQ D2/GCD-TOP (IL:IQUOTIENT DENOMINATOR-2 GCD-TOP))
                     (SETQ TOP (IL:IQUOTIENT TOP GCD-TOP)))
                 (IF (AND (EQ 1 D2/GCD-TOP)
                          (EQ 1 D1/GCD-D))
                     TOP
                     (%MAKE-RATIO TOP (* D1/GCD-D D2/GCD-TOP)))))))
```

(DEFUN **%RATIO-TIMES** (NUMERATOR-1 DENOMINATOR-1 NUMERATOR-2 DENOMINATOR-2)

```
  (LET ((GCD-1-2 (%GCD NUMERATOR-1 DENOMINATOR-2))
```

```
         (GCD-2-1 (%GCD NUMERATOR-2 DENOMINATOR-1)))
        (UNLESS (EQ GCD-1-2 1)
            (SETQ NUMERATOR-1 (IL:IQUOTIENT NUMERATOR-1 GCD-1-2))
            (SETQ DENOMINATOR-2 (IL:IQUOTIENT DENOMINATOR-2 GCD-1-2)))
        (UNLESS (EQ GCD-2-1 1)
            (SETQ NUMERATOR-2 (IL:IQUOTIENT NUMERATOR-2 GCD-2-1))
            (SETQ DENOMINATOR-1 (IL:IQUOTIENT DENOMINATOR-1 GCD-2-1))))
   (LET ((H (* NUMERATOR-1 NUMERATOR-2))
         (K (* DENOMINATOR-1 DENOMINATOR-2)))
        (IF (EQ K 1)
            H
            (IF (MINUSP K)
                (%MAKE-RATIO (- H)
                       (- K))
                (%MAKE-RATIO H K)))))
```

;;; Section 2.1.4 Complex Numbers.

```
(DEFSTRUCT (COMPLEX (:CONSTRUCTOR %MAKE-COMPLEX (REALPART IMAGPART))
                    (:PREDICATE COMPLEXP)
                    (:COPIER NIL)
                    (:PRINT-FUNCTION %COMPLEX-PRINT))
   (REALPART :READ-ONLY)
   (IMAGPART :READ-ONLY))
```

;; So we don't inherit the deftype from defstruct

```
(DEFTYPE COMPLEX (&OPTIONAL TYPE)
   (IF (EQ TYPE '*)
       '(:DATATYPE COMPLEX)
       `(AND COMPLEX (SATISFIES (IL:LAMBDA (IL:X)
                                  (AND (TYPEP (COMPLEX-REALPART IL:X)
                                              ',TYPE)
                                       (TYPEP (COMPLEX-IMAGPART IL:X)
                                              ',TYPE)))))))
```

;; Make Complex NUMBERP

```
(IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOCOPY

(IL:\\SETTYPEMASK (IL:\\TYPENUMBERFROMNAME 'COMPLEX)
       (IL:LOGOR IL:\\TT.NUMBERP IL:\\TT.ATOM))
)
```

```
(DEFUN COMPLEX (REALPART &OPTIONAL (IMAGPART 0))
    ;; Builds a complex number from the specified components.  Note: IMAGPART = 0.0 or floating REALPART implies that we must build a complex not
    ;; a real according to the manual while IMAGPART = 0 and rational REALPART implies that we build a real.
   (TYPECASE REALPART
       (RATIONAL (TYPECASE IMAGPART
                     (RATIONAL (IF (EQ 0 IMAGPART)
                                   REALPART
                                   (%MAKE-COMPLEX REALPART IMAGPART)))
                     (FLOAT (%MAKE-COMPLEX (FLOAT REALPART)
                                   IMAGPART))
                     (OTHERWISE (%NOT-NONCOMPLEX-NUMBER-ERROR IMAGPART))))
       (FLOAT (%MAKE-COMPLEX REALPART (FLOAT IMAGPART)))
       (OTHERWISE (%NOT-NONCOMPLEX-NUMBER-ERROR REALPART))))


(DEFUN REALPART (NUMBER)
   (TYPECASE NUMBER
       (COMPLEX (COMPLEX-REALPART NUMBER))
       (NUMBER NUMBER)
       (OTHERWISE (%NOT-NUMBER-ERROR NUMBER))))


(DEFUN IMAGPART (NUMBER)
   (TYPECASE NUMBER
       (COMPLEX (COMPLEX-IMAGPART NUMBER))
       (FLOAT 0.0)
       (NUMBER 0)
       (OTHERWISE (%NOT-NUMBER-ERROR NUMBER))))


(DEFUN CONJUGATE (NUMBER)
   (TYPECASE NUMBER
       (COMPLEX (%MAKE-COMPLEX (COMPLEX-REALPART NUMBER)
                     (- (COMPLEX-IMAGPART NUMBER))))
       (NUMBER NUMBER)
       (OTHERWISE (%NOT-NUMBER-ERROR NUMBER))))
```

```
(DEFUN PHASE (NUMBER)
   (COND
      ((= NUMBER 0)
       ;; The phase of zero is arbitrarily defined to be zero.
       0.0)
      ((COMPLEXP NUMBER)
       (ATAN (COMPLEX-IMAGPART NUMBER)
             (COMPLEX-REALPART NUMBER)))
      ((MINUSP NUMBER)
       PI)
      (T ;; Page 206 of the silver book: The phase of a positive non-complex number is zero. ... The result is a floating-point number.
         0.0)))


(DEFUN %COMPLEX-PRINT (NUMBER STREAM)
   (LET ((REALPART (COMPLEX-REALPART NUMBER))
         (IMAGPART (COMPLEX-IMAGPART NUMBER)))
        (IL:.SPACECHECK. STREAM (+ 5 (IL:NCHARS REALPART)
                                   (IL:NCHARS IMAGPART)))
        (IL:\\OUTCHAR STREAM (IL:FETCH (READTABLEP IL:HASHMACROCHAR) IL:OF *READTABLE*))
        (IL:\\SOUT "C" STREAM)
        (IL:\\SOUT "(" STREAM)
        (IL:\\PRINDATUM REALPART STREAM)
        (IL:\\SOUT " " STREAM)
        (IL:\\PRINDATUM IMAGPART STREAM)
        (IL:\\SOUT ")" STREAM)))


(DEFUN %COMPLEX-+ (REAL-1 IMAG-1 REAL-2 IMAG-2)
   (COND
      ((= IMAG-1 0)
       (COMPLEX (+ REAL-1 REAL-2)
              IMAG-2))
      ((= IMAG-2 0)
       (COMPLEX (+ REAL-1 REAL-2)
              IMAG-1))
      (T (COMPLEX (+ REAL-1 REAL-2)
                  (+ IMAG-1 IMAG-2)))))


(DEFUN %COMPLEX-- (REAL-1 IMAG-1 REAL-2 IMAG-2)
   (COND
      ((= IMAG-1 0)
       (COMPLEX (- REAL-1 REAL-2)
              (- IMAG-2)))
      ((= IMAG-2 0)
       (COMPLEX (- REAL-1 REAL-2)
              IMAG-1))
      (T (COMPLEX (- REAL-1 REAL-2)
                  (- IMAG-1 IMAG-2)))))


(DEFUN %COMPLEX-* (REAL-1 IMAG-1 REAL-2 IMAG-2)
   (COND
      ((= IMAG-1 0)
       (COMPLEX (* REAL-1 REAL-2)
              (* REAL-1 IMAG-2)))
      ((= IMAG-2 0)
       (COMPLEX (* REAL-1 REAL-2)
              (* IMAG-1 REAL-2)))
      (T (COMPLEX (- (* REAL-1 REAL-2)
                     (* IMAG-1 IMAG-2))
                  (+ (* IMAG-1 REAL-2)
                     (* REAL-1 IMAG-2))))))


(DEFUN %COMPLEX-/ (REAL-1 IMAG-1 REAL-2 IMAG-2)
   (COND
      ((= 0 IMAG-1)
       (LET ((MODULUS (+ (* REAL-2 REAL-2)
                         (* IMAG-2 IMAG-2))))
            (COMPLEX (/ (* REAL-1 REAL-2)
                        MODULUS)
                     (/ (- (* REAL-1 IMAG-2))
                        MODULUS))))
      ((= 0 IMAG-2)
       (COMPLEX (/ REAL-1 REAL-2)
              (/ IMAG-1 REAL-2)))
      (T (LET ((MODULUS (+ (* REAL-2 REAL-2)
                           (* IMAG-2 IMAG-2))))
              (COMPLEX (/ (+ (* REAL-1 REAL-2)
                             (* IMAG-1 IMAG-2))
                          MODULUS)
                       (/ (- (* IMAG-1 REAL-2)
```

```
                          (* REAL-1 IMAG-2))
                       MODULUS))))))

(DEFUN %COMPLEX-ABS (Z)
   (LET ((X (FLOAT (COMPLEX-REALPART Z)))
         (Y (FLOAT (COMPLEX-IMAGPART Z))))
        (DECLARE (TYPE FLOAT X Y))
        ;; Might want to use a BLUE algorithm here
        (SQRT (SETQ X (+ (* X X)
                         (* Y Y))))))
```

;;; Datatype predicates

;; cl:integerp is defined in cmlpred (has an optimizer)

;; cl:floatp is defined in cmltypes (has an optimizer). il:floatp is defined on llbasic

;; cl:complexp is a defstruct predicate (compiles in line)

;; cl:numberp is defined in cmltypes (has an optimizer). il:numberp is defined on llbasic

;;; Section 12.2 Predicates on Numbers (generic).

;; cl:zerop is not shared with il:zerop, although they are equivalent. There is no il;plusp

```
(DEFUN ZEROP (NUMBER)
   (= 0 NUMBER))

(DEFUN PLUSP (NUMBER)
   (> NUMBER 0))

(XCL:DEFOPTIMIZER ZEROP (NUMBER)
                        `(= 0 ,NUMBER))

(XCL:DEFOPTIMIZER PLUSP (NUMBER)
                        `(> ,NUMBER 0))
```

;; cl:minusp is shared with il:minusp, but must be redefined to work with ratios. Old version resides in llarith

```
(DEFUN MINUSP (NUMBER)
   (< NUMBER 0))

(XCL:DEFOPTIMIZER MINUSP (NUMBER)
                         `(< ,NUMBER 0))
```

;; Both cl:evenp and cl:oddp are shared with il:. The functions are extended by allowing a second optional modulus argument. Another version of il:oddp
;; exists on llarith, but the definition of il:evenp has disappeared

```
(DEFUN EVENP (INTEGER &OPTIONAL MODULUS)
   (IF (NULL MODULUS)
       (EQ (LOGAND INTEGER 1)
           0)
       (ZEROP (MOD INTEGER MODULUS))))

(DEFUN ODDP (INTEGER &OPTIONAL MODULUS)
   (IF (NULL MODULUS)
       (EQ (LOGAND INTEGER 1)
           1)
       (NOT (ZEROP (MOD INTEGER MODULUS)))))

(XCL:DEFOPTIMIZER EVENP (INTEGER &OPTIONAL (MODULUS NIL MODULUS-P))
                        (IF (NULL MODULUS-P)
                            `(EQ (LOGAND ,INTEGER 1)
                                 0)
                            'COMPILER:PASS))

(XCL:DEFOPTIMIZER ODDP (INTEGER &OPTIONAL (MODULUS NIL MODULUS-P))
                       (IF (NULL MODULUS-P)
                           `(EQ (LOGAND ,INTEGER 1)
                                1)
                           'COMPILER:PASS))
```

;;; Section 12.3 Comparisons on Numbers. (generic)

```
(DEFUN %= (X Y)
   ;; %= does coercion when checking numbers for equality.  Page 196 of silver book.
   ;; Punt function for opcode =(decimal 255) -- actually the UFN is IL:%=
   (IL:\\CALLME '=)
   (TYPECASE X
       (INTEGER (TYPECASE Y
                   (INTEGER (IL:IEQP X Y))
                   (FLOAT (IL:FEQP X Y))
                   (RATIO NIL)
                   (COMPLEX (AND (= X (COMPLEX-REALPART Y))
                                 (= 0 (COMPLEX-IMAGPART Y))))
                   (OTHERWISE (%NOT-NUMBER-ERROR Y))))
       (FLOAT (TYPECASE Y
                  ((OR INTEGER FLOAT RATIO) (IL:FEQP X Y))
                  (COMPLEX (AND (= X (COMPLEX-REALPART Y))
                                (= 0 (COMPLEX-IMAGPART Y))))
                  (OTHERWISE (%NOT-NUMBER-ERROR Y))))
       (RATIO (TYPECASE Y
                  (INTEGER NIL)
                  (RATIO (AND (EQL (RATIO-NUMERATOR X)
                                   (RATIO-NUMERATOR Y))
                              (EQL (RATIO-DENOMINATOR X)
                                   (RATIO-DENOMINATOR Y))))
                  (FLOAT (IL:FEQP X Y))
                  (COMPLEX (AND (= X (COMPLEX-REALPART Y))
                                (= 0 (COMPLEX-IMAGPART Y))))
                  (OTHERWISE (%NOT-NUMBER-ERROR Y))))
       (COMPLEX (TYPECASE Y
                    (COMPLEX (AND (= (COMPLEX-REALPART X)
                                     (COMPLEX-REALPART Y))
                                  (= (COMPLEX-IMAGPART X)
                                     (COMPLEX-IMAGPART Y))))
                    (NUMBER (AND (= Y (COMPLEX-REALPART X))
                                 (= 0 (COMPLEX-IMAGPART X))))
                    (OTHERWISE (%NOT-NUMBER-ERROR Y))))
       (OTHERWISE (%NOT-NUMBER-ERROR X))))


(DEFMACRO %/= (X Y)
   `(NOT (%= ,X ,Y)))


(DEFUN %> (X Y)
   ;; See page 196 of CLtl
   ;; Compiles out to greaterp opcode                                    ; So we appear as > in a frame backtrace
   (IL:\\CALLME '>)
   (TYPECASE X
       (INTEGER (TYPECASE Y
                   (INTEGER (IL:IGREATERP X Y))
                   (FLOAT (IL:FGREATERP X Y))
                   (RATIO (IL:IGREATERP (* (RATIO-DENOMINATOR Y)
                                           X)
                                        (RATIO-NUMERATOR Y)))
                   (OTHERWISE (%NOT-NONCOMPLEX-NUMBER-ERROR Y))))
       (FLOAT (TYPECASE Y
                  ((OR INTEGER FLOAT) (IL:FGREATERP X Y))
                  (RATIO (IL:FGREATERP (* (RATIO-DENOMINATOR Y)
                                          X)
                                       (RATIO-NUMERATOR Y)))
                  (OTHERWISE (%NOT-NONCOMPLEX-NUMBER-ERROR Y))))
       (RATIO (TYPECASE Y
                  (INTEGER (IL:IGREATERP (RATIO-NUMERATOR X)
                                         (* (RATIO-DENOMINATOR X)
                                            Y)))
                  (FLOAT (IL:FGREATERP (RATIO-NUMERATOR X)
                                       (* (RATIO-DENOMINATOR X)
                                          Y)))
                  (RATIO (IL:IGREATERP (* (RATIO-NUMERATOR X)
                                          (RATIO-DENOMINATOR Y))
                                       (* (RATIO-NUMERATOR Y)
                                          (RATIO-DENOMINATOR X))))
                  (OTHERWISE (%NOT-NONCOMPLEX-NUMBER-ERROR Y))))
       (OTHERWISE (%NOT-NONCOMPLEX-NUMBER-ERROR X))))


(DEFUN %< (X Y)
   (%> Y X))


(DEFMACRO %>= (X Y)
   `(NOT (%< ,X ,Y)))


(DEFMACRO %<= (X Y)
   `(NOT (%> ,X ,Y)))
```

```
(IL:PUTPROPS %= IL:DOPVAL (2 =))

(IL:PUTPROPS %> IL:DOPVAL (2 IL:GREATERP))

(IL:PUTPROPS %< IL:DOPVAL (2 IL:SWAP IL:GREATERP))
```

;; For the byte compiler

```
(IL:PUTPROPS %> IL:DMACRO (= . IL:GREATERP))

(IL:PUTPROPS %< IL:DMACRO (= . IL:LESSP))

(IL:PUTPROPS %>= IL:DMACRO (= . IL:GEQ))

(IL:PUTPROPS %<= IL:DMACRO (= . IL:LEQ))

(IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOCOPY
```

;; Backward compatibility
;; il:%= is listed as the punt function for the = opcode

```
(IL:MOVD '%= 'IL:%=)
```

;; Greaterp is the UFN for the greaterp opcode. Effectively redefines the opcode

```
(IL:MOVD '%> 'IL:GREATERP)
```

;; Interlisp Greaterp and Lessp are defined in llarith

```
(IL:MOVD '%< 'IL:LESSP)
)
```

;; =, <, >, <=, and >= are shared with il:, but cl:/= is NOT shared (?!)

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

(DEFMACRO %COMPARISON-MACRO (PREDICATE ARGS)
   `(PROGN (IF (%< ,ARGS 1)
               (ERROR ,(CONCATENATE 'STRING (SUBSEQ (STRING PREDICATE)
                                                    1)
                                    " requires at least one argument")))
           (LET ((LAST-ARG (IL:ARG ,ARGS 1))
                 (I 2)
                 CURRENT-ARG)
                (IF (OR (NOT (NUMBERP LAST-ARG))
                        (COMPLEXP LAST-ARG))
                    (%NOT-NUMBER-ERROR LAST-ARG))
                (LOOP (IF (%> I ,ARGS)
                          (RETURN T))
                      (SETQ CURRENT-ARG (IL:ARG ,ARGS I))
                      (IF (NOT (,PREDICATE LAST-ARG CURRENT-ARG))
                          (RETURN NIL))
                      (SETQ LAST-ARG CURRENT-ARG)
                      (SETQ I (1+ I))))))
)

(IL:DEFINEQ

(=
  (IL:LAMBDA ARGS                                                    ; Edited  8-Apr-87 14:40 by jop
    (IF (%< ARGS 1)
        (ERROR "= requires at least one argument"))
    (LET ((FIRST-ARG (IL:ARG ARGS 1))
          (I 2))
         (IF (NOT (NUMBERP FIRST-ARG))
             (%NOT-NUMBER-ERROR FIRST-ARG))
         (LOOP (IF (%> I ARGS)
                   (RETURN T))
               (IF (%/= FIRST-ARG (IL:ARG ARGS I))
                   (RETURN NIL))
               (SETQ I (1+ I))))))


(/=
  (IL:LAMBDA ARGS                                                    ; Edited  8-Feb-87 14:01 by jop
    (IF (%< ARGS 1)
        (ERROR "/= requires at least one argument"))
    (LET ((I 1)
          CURRENT-ARG J)
         (LOOP (IF (%> I ARGS)
                   (RETURN T))
               (SETQ CURRENT-ARG (IL:ARG ARGS I))
```

```
               (SETQ J (1+ I))
               (IF (NULL (LOOP (IF (%> J ARGS)
                                   (RETURN T))
                               (IF (%= CURRENT-ARG (IL:ARG ARGS J))
                                   (RETURN NIL))
                               (SETQ J (1+ J))))
                   (RETURN NIL))
               (SETQ I (1+ I))))))


(<
  (IL:LAMBDA ARGS                                                    ; Edited 8-Feb-87 14:17 by jop
    (%COMPARISON-MACRO %< ARGS)))


(>
  (IL:LAMBDA ARGS                                                    ; Edited 8-Feb-87 14:17 by jop
    (%COMPARISON-MACRO %> ARGS)))


(<=
  (IL:LAMBDA ARGS                                                    ; Edited 8-Feb-87 14:16 by jop
    (%COMPARISON-MACRO %<= ARGS)))


(>=
  (IL:LAMBDA ARGS                                                    ; Edited 8-Feb-87 14:18 by jop
    (%COMPARISON-MACRO %>= ARGS)))

)


(DEFUN %COMPARISON-OPTIMIZER (PREDICATE FIRST-NUMBER SECOND-NUMBER THIRD-NUMBER)
   (COND
      ((NULL SECOND-NUMBER)
       'COMPILER:PASS)
      ((NULL THIRD-NUMBER)
       `(,PREDICATE ,FIRST-NUMBER ,SECOND-NUMBER))
      (T `((IL:OPENLAMBDA (SI::%$$COMPARISON-FIRST-NUMBER SI::%$$COMPARISON-MIDDLE-NUMBER)
             (AND (,PREDICATE SI::%$$COMPARISON-FIRST-NUMBER SI::%$$COMPARISON-MIDDLE-NUMBER)
                  (,PREDICATE SI::%$$COMPARISON-MIDDLE-NUMBER ,THIRD-NUMBER)))
           ,FIRST-NUMBER
           ,SECOND-NUMBER))))


(XCL:DEFOPTIMIZER = (FIRST-NUMBER &OPTIONAL (SECOND-NUMBER NIL SECOND-NUMBER-P)
                          &REST MORE-NUMBERS)
                    (COND
                       ((NULL SECOND-NUMBER-P)
                        'COMPILER:PASS)
                       ((NULL MORE-NUMBERS)
                        `(%= ,FIRST-NUMBER ,SECOND-NUMBER))
                       (T (SETQ MORE-NUMBERS (CONS SECOND-NUMBER MORE-NUMBERS))
                          `((IL:OPENLAMBDA (SI::%$$=FIRST-NUMBER)
                              (AND ,@(LET ((RESULT NIL)
                                           (RESULT-TAIL NIL))
                                       (DOLIST (NUMBER MORE-NUMBERS RESULT)
                                           (%LIST-COLLECT RESULT RESULT-TAIL
                                                  (LIST `(%= SI::%$$=FIRST-NUMBER ,NUMBER)))))))
                            ,FIRST-NUMBER))))


(XCL:DEFOPTIMIZER /= (FIRST-NUMBER &OPTIONAL (SECOND-NUMBER NIL SECOND-NUMBER-P)
                           &REST MORE-NUMBERS)
                     (COND
                        ((NULL SECOND-NUMBER-P)
                         'COMPILER:PASS)
                        ((NULL MORE-NUMBERS)
                         `(%/= ,FIRST-NUMBER ,SECOND-NUMBER))
                        (T 'COMPILER:PASS)))


(XCL:DEFOPTIMIZER < (FIRST-NUMBER &OPTIONAL SECOND-NUMBER THIRD-NUMBER &REST MORE-NUMBERS)
                  (IF (NULL MORE-NUMBERS)
                      (%COMPARISON-OPTIMIZER '%< FIRST-NUMBER SECOND-NUMBER THIRD-NUMBER)
                      'COMPILER:PASS))


(XCL:DEFOPTIMIZER > (FIRST-NUMBER &OPTIONAL SECOND-NUMBER THIRD-NUMBER &REST MORE-NUMBERS)
                  (IF (NULL MORE-NUMBERS)
                      (%COMPARISON-OPTIMIZER '%> FIRST-NUMBER SECOND-NUMBER THIRD-NUMBER)
                      'COMPILER:PASS))


(XCL:DEFOPTIMIZER <= (FIRST-NUMBER &OPTIONAL SECOND-NUMBER THIRD-NUMBER &REST MORE-NUMBERS)
                   (IF (NULL MORE-NUMBERS)
                       (%COMPARISON-OPTIMIZER '%<= FIRST-NUMBER SECOND-NUMBER THIRD-NUMBER)
```

```
                        'COMPILER:PASS))


(XCL:DEFOPTIMIZER >= (FIRST-NUMBER &OPTIONAL SECOND-NUMBER THIRD-NUMBER &REST MORE-NUMBERS)
                     (IF (NULL MORE-NUMBERS)
                         (%COMPARISON-OPTIMIZER '%>= FIRST-NUMBER SECOND-NUMBER THIRD-NUMBER)
                         'COMPILER:PASS))
```

;; Note: the related predicates EQL, EQUAL, and EQUALP should be consulted if any of the above change. EQL is on LLNEW (?), EQUAL and ;; EQUALP on CMLTYPES.

;; cl:min and cl:max are shared with il: (defined in llarith). They are written in terms of GREATERP and hence work on ratios. Note (min) returns ;; #.max.integer , which is an extension on the CLtl spec. We only optimize the case of two args

```
(XCL:DEFOPTIMIZER MIN (&OPTIONAL (X NIL X-P)
                                 (Y NIL Y-P)
                                 &REST OTHER-NUMBERS)
                      (IF (AND (NULL OTHER-NUMBERS)
                               X-P Y-P)
                          `((IL:OPENLAMBDA (SI::%$$MIN-X SI::%$$MIN-Y)
                              (IF (< SI::%$$MIN-X SI::%$$MIN-Y)
                                  SI::%$$MIN-X
                                  SI::%$$MIN-Y))
                            ,X
                            ,Y)
                          'COMPILER:PASS))


(XCL:DEFOPTIMIZER MAX (&OPTIONAL (X NIL X-P)
                                 (Y NIL Y-P)
                                 &REST OTHER-NUMBERS)
                      (IF (AND (NULL OTHER-NUMBERS)
                               X-P Y-P)
                          `((IL:OPENLAMBDA (SI::%$$MAX-X SI::%$$MAX-Y)
                              (IF (> SI::%$$MAX-X SI::%$$MAX-Y)
                                  SI::%$$MAX-X
                                  SI::%$$MAX-Y))
                            ,X
                            ,Y)
                          'COMPILER:PASS))
```

;;; Section 12.4 Arithmetic Operations (generic).

```
(DEFUN %+ (X Y)
   (IL:\\CALLME '+)
   ;; Simple case for the sum of two numbers. Is the ufn for the plus2 opcode
   (TYPECASE X
       (INTEGER (TYPECASE Y
                    (INTEGER (IL:IPLUS X Y))
                    (FLOAT (IL:FPLUS X Y))
                    (RATIO (%RATIO-PLUS X 1 (RATIO-NUMERATOR Y)
                                            (RATIO-DENOMINATOR Y)))
                    (COMPLEX (%COMPLEX-+ X 0 (COMPLEX-REALPART Y)
                                             (COMPLEX-IMAGPART Y)))
                    (OTHERWISE (%NOT-NUMBER-ERROR Y))))
       (FLOAT (TYPECASE Y
                  ((OR INTEGER FLOAT) (IL:FPLUS X Y))
                  (RATIO (IL:FPLUS X (IL:FQUOTIENT (RATIO-NUMERATOR Y)
                                                   (RATIO-DENOMINATOR Y))))
                  (COMPLEX (%COMPLEX-+ X 0.0 (COMPLEX-REALPART Y)
                                             (COMPLEX-IMAGPART Y)))
                  (OTHERWISE (%NOT-NUMBER-ERROR Y))))
       (RATIO (TYPECASE Y
                  (INTEGER (%RATIO-PLUS (RATIO-NUMERATOR X)
                                        (RATIO-DENOMINATOR X)
                                        Y 1))
                  (FLOAT (IL:FPLUS (IL:FQUOTIENT (RATIO-NUMERATOR X)
                                                 (RATIO-DENOMINATOR X))
                                   Y))
                  (RATIO (%RATIO-PLUS (RATIO-NUMERATOR X)
                                      (RATIO-DENOMINATOR X)
                                      (RATIO-NUMERATOR Y)
                                      (RATIO-DENOMINATOR Y)))
                  (COMPLEX (%COMPLEX-+ X 0 (COMPLEX-REALPART Y)
                                           (COMPLEX-IMAGPART Y)))
                  (OTHERWISE (%NOT-NUMBER-ERROR Y))))
       (COMPLEX (TYPECASE Y
                    ((OR INTEGER RATIO) (%COMPLEX-+ (COMPLEX-REALPART X)
                                                    (COMPLEX-IMAGPART X)
                                                    Y 0))
                    (FLOAT (%COMPLEX-+ (COMPLEX-REALPART X)
                                       (COMPLEX-IMAGPART X)
                                       Y 0.0))
```

```
                    (COMPLEX (%COMPLEX-+ (COMPLEX-REALPART X)
                                         (COMPLEX-IMAGPART X)
                                         (COMPLEX-REALPART Y)
                                         (COMPLEX-IMAGPART Y)))
                    (OTHERWISE (%NOT-NUMBER-ERROR Y))))
        (OTHERWISE (%NOT-NUMBER-ERROR X))))


(DEFUN %- (X Y)
    ;; UFN for opcode difference.
    (IL:\\CALLME '-)
    (TYPECASE X
        (INTEGER (TYPECASE Y
                    (INTEGER (IL:IDIFFERENCE X Y))
                    (FLOAT (IL:FDIFFERENCE X Y))
                    (RATIO (%RATIO-PLUS X 1 (- (RATIO-NUMERATOR Y))
                                        (RATIO-DENOMINATOR Y)))
                    (COMPLEX (%COMPLEX-- X 0 (COMPLEX-REALPART Y)
                                             (COMPLEX-IMAGPART Y)))
                    (OTHERWISE (%NOT-NUMBER-ERROR Y))))
        (FLOAT (TYPECASE Y
                   ((OR INTEGER FLOAT) (IL:FDIFFERENCE X Y))
                   (RATIO (IL:FDIFFERENCE X (IL:FQUOTIENT (RATIO-NUMERATOR Y)
                                                          (RATIO-DENOMINATOR Y))))
                   (COMPLEX (%COMPLEX-- X 0.0 (COMPLEX-REALPART Y)
                                              (COMPLEX-IMAGPART Y)))
                   (OTHERWISE (%NOT-NUMBER-ERROR Y))))
        (RATIO (TYPECASE Y
                   (INTEGER (%RATIO-PLUS (RATIO-NUMERATOR X)
                                         (RATIO-DENOMINATOR X)
                                         (- Y)
                                         1))
                   (FLOAT (IL:FDIFFERENCE (IL:FQUOTIENT (RATIO-NUMERATOR X)
                                                        (RATIO-DENOMINATOR X))
                                          Y))
                   (RATIO (%RATIO-PLUS (RATIO-NUMERATOR X)
                                       (RATIO-DENOMINATOR X)
                                       (- (RATIO-NUMERATOR Y))
                                       (RATIO-DENOMINATOR Y)))
                   (COMPLEX (%COMPLEX-- X 0 (COMPLEX-REALPART Y)
                                            (COMPLEX-IMAGPART Y)))
                   (OTHERWISE (%NOT-NUMBER-ERROR Y))))
        (COMPLEX (TYPECASE Y
                     ((OR INTEGER RATIO) (%COMPLEX-- (COMPLEX-REALPART X)
                                                     (COMPLEX-IMAGPART X)
                                                     Y 0))
                     (FLOAT (%COMPLEX-- (COMPLEX-REALPART X)
                                        (COMPLEX-IMAGPART X)
                                        Y 0.0))
                     (COMPLEX (%COMPLEX-- (COMPLEX-REALPART X)
                                          (COMPLEX-IMAGPART X)
                                          (COMPLEX-REALPART Y)
                                          (COMPLEX-IMAGPART Y)))
                     (OTHERWISE (%NOT-NUMBER-ERROR Y))))
        (OTHERWISE (%NOT-NUMBER-ERROR X))))


(DEFUN %* (X Y)
    ;; UFN for opcode times2.
    (IL:\\CALLME '*)
    (TYPECASE X
        (INTEGER (TYPECASE Y
                    (INTEGER (IL:ITIMES X Y))
                    (FLOAT (IL:FTIMES X Y))
                    (RATIO (%RATIO-TIMES X 1 (RATIO-NUMERATOR Y)
                                             (RATIO-DENOMINATOR Y)))
                    (COMPLEX (%COMPLEX-* X 0 (COMPLEX-REALPART Y)
                                             (COMPLEX-IMAGPART Y)))
                    (OTHERWISE (%NOT-NUMBER-ERROR Y))))
        (FLOAT (TYPECASE Y
                   ((OR INTEGER FLOAT) (IL:FTIMES X Y))
                   (RATIO (IL:FTIMES X (IL:FQUOTIENT (RATIO-NUMERATOR Y)
                                                     (RATIO-DENOMINATOR Y))))
                   (COMPLEX (%COMPLEX-* X 0.0 (COMPLEX-REALPART Y)
                                              (COMPLEX-IMAGPART Y)))
                   (OTHERWISE (%NOT-NUMBER-ERROR Y))))
        (RATIO (TYPECASE Y
                   (INTEGER (%RATIO-TIMES (RATIO-NUMERATOR X)
                                          (RATIO-DENOMINATOR X)
                                          Y 1))
                   (FLOAT (IL:FTIMES (IL:FQUOTIENT (RATIO-NUMERATOR X)
                                                   (RATIO-DENOMINATOR X))
                                     Y))
                   (RATIO (%RATIO-TIMES (RATIO-NUMERATOR X)
                                        (RATIO-DENOMINATOR X)
```

```
                              (RATIO-NUMERATOR Y)
                              (RATIO-DENOMINATOR Y)))
                    (COMPLEX (%COMPLEX-* X 0 (COMPLEX-REALPART Y)
                                  (COMPLEX-IMAGPART Y)))
                    (OTHERWISE (%NOT-NUMBER-ERROR Y))))
          (COMPLEX (TYPECASE Y
                     ((OR INTEGER RATIO) (%COMPLEX-* (COMPLEX-REALPART X)
                                                     (COMPLEX-IMAGPART X)
                                                    Y 0))
                     (FLOAT (%COMPLEX-* (COMPLEX-REALPART X)
                                        (COMPLEX-IMAGPART X)
                                       Y 0.0))
                     (COMPLEX (%COMPLEX-* (COMPLEX-REALPART X)
                                          (COMPLEX-IMAGPART X)
                                          (COMPLEX-REALPART Y)
                                          (COMPLEX-IMAGPART Y)))
                     (OTHERWISE (%NOT-NUMBER-ERROR Y))))
          (OTHERWISE (%NOT-NUMBER-ERROR X))))


(DEFUN %/ (X Y)
   ;; The quotient of two numbers. Has no corresponding opcode.
   (IL:\\CALLME '/)
   (IF (AND (IL:SMALLP X)
            (IL:SMALLP Y)
            (EQ 0 (IL:IREMAINDER X Y)))
       ;; See if we can do the straight-forward thing
       (IL:IQUOTIENT X Y)
       ;; More exotic cases
       (TYPECASE X
           (INTEGER (TYPECASE Y
                      (INTEGER (IF (OR (EQ X IL:MIN.INTEGER)
                                       (EQ X IL:MAX.INTEGER)
                                       (EQ Y IL:MIN.INTEGER)
                                       (EQ Y IL:MAX.INTEGER))
                                   (IL:IQUOTIENT X Y)
                                   (%BUILD-RATIO X Y)))
                      (FLOAT (IL:FQUOTIENT X Y))
                      (RATIO (%RATIO-TIMES X 1 (RATIO-DENOMINATOR Y)
                                    (RATIO-NUMERATOR Y)))
                      (COMPLEX (%COMPLEX-/ X 0 (COMPLEX-REALPART Y)
                                       (COMPLEX-IMAGPART Y)))
                      (OTHERWISE (%NOT-NUMBER-ERROR Y))))
           (FLOAT (TYPECASE Y
                     ((OR INTEGER FLOAT) (IL:FQUOTIENT X Y))
                     (RATIO (IL:FQUOTIENT (* (RATIO-DENOMINATOR Y)
                                             X)
                                    (RATIO-NUMERATOR Y)))
                     (COMPLEX (%COMPLEX-/ X 0.0 (COMPLEX-REALPART Y)
                                      (COMPLEX-IMAGPART Y)))
                     (OTHERWISE (%NOT-NUMBER-ERROR Y))))
           (RATIO (TYPECASE Y
                     (INTEGER (%RATIO-TIMES (RATIO-NUMERATOR X)
                                     (RATIO-DENOMINATOR X)
                                    1 Y))
                     (FLOAT (IL:FQUOTIENT (RATIO-NUMERATOR X)
                                    (* (RATIO-DENOMINATOR X)
                                       Y)))
                     (RATIO (%RATIO-TIMES (RATIO-NUMERATOR X)
                                    (RATIO-DENOMINATOR X)
                                    (RATIO-DENOMINATOR Y)
                                    (RATIO-NUMERATOR Y)))
                     (COMPLEX (%COMPLEX-/ X 0 (COMPLEX-REALPART Y)
                                      (COMPLEX-IMAGPART Y)))
                     (OTHERWISE (%NOT-NUMBER-ERROR Y))))
           (COMPLEX (TYPECASE Y
                      ((OR INTEGER RATIO) (%COMPLEX-/ (COMPLEX-REALPART X)
                                                      (COMPLEX-IMAGPART X)
                                                     Y 0))
                      (FLOAT (%COMPLEX-/ (COMPLEX-REALPART X)
                                         (COMPLEX-IMAGPART X)
                                        Y 0.0))
                      (COMPLEX (%COMPLEX-/ (COMPLEX-REALPART X)
                                           (COMPLEX-IMAGPART X)
                                           (COMPLEX-REALPART Y)
                                           (COMPLEX-IMAGPART Y)))
                      (OTHERWISE (%NOT-NUMBER-ERROR Y))))
           (OTHERWISE (%NOT-NUMBER-ERROR X)))))

;; NOTE: %/ cannot compile out to the existinq quotient opcode because it produces ratios rather than truncating

(IL:PUTPROPS %+ IL:DOPVAL (2 IL:PLUS2))

(IL:PUTPROPS %- IL:DOPVAL (2 IL:DIFFERENCE))
```

```
(IL:PUTPROPS %* IL:DOPVAL (2 IL:TIMES2))
```

;; For the byte compiler

```
(IL:PUTPROPS %+ IL:DMACRO (= . IL:PLUS))

(IL:PUTPROPS %- IL:DMACRO (= . IL:DIFFERENCE))

(IL:PUTPROPS %* IL:DMACRO (= . IL:TIMES))

(IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOCOPY
```

;; Backward compatibility

```
(IL:MOVD '%/ 'IL:%/)
```

;; Redefine UFNs for generic plus, difference, and times. Old UFN defined in llarith.

```
(IL:MOVD '%+ 'IL:\\SLOWPLUS2)

(IL:MOVD '%- 'IL:\\SLOWDIFFERENCE)

(IL:MOVD '%* 'IL:\\SLOWTIMES2)
)

(IL:DEFINEQ

(+
  (IL:LAMBDA ARGS                                                    ; Edited  8-Apr-87 14:41 by jop
    (IF (EQ ARGS 0)
        0
        (LET ((ACCUMULATOR (IL:ARG ARGS 1))
              (I 2))
             (IF (NOT (NUMBERP ACCUMULATOR))
                 (%NOT-NUMBER-ERROR ACCUMULATOR))
             (LOOP (IF (%> I ARGS)
                       (RETURN ACCUMULATOR))
                   (SETQ ACCUMULATOR (%+ ACCUMULATOR (IL:ARG ARGS I)))
                   (SETQ I (1+ I)))))))


(-
  (IL:LAMBDA ARGS                                                    ; Edited  9-Feb-87 20:57 by jop
    (COND
       ((EQ ARGS 0)
        (ERROR "- requires at least one argument"))
       ((EQ ARGS 1)
        ;; Negate the argument
        (- 0 (IL:ARG ARGS 1)))
       (T (LET ((ACCUMULATOR (IL:ARG ARGS 1))
                (I 2))
               (LOOP (IF (%> I ARGS)
                         (RETURN ACCUMULATOR))
                     (SETQ ACCUMULATOR (%- ACCUMULATOR (IL:ARG ARGS I)))
                     (SETQ I (1+ I))))))))


(*
  (IL:LAMBDA ARGS                                                    ; Edited  8-Apr-87 14:41 by jop
    (IF (EQ ARGS 0)
        1
        (LET ((ACCUMULATOR (IL:ARG ARGS 1))
              (I 2))
             (IF (NOT (NUMBERP ACCUMULATOR))
                 (%NOT-NUMBER-ERROR ACCUMULATOR))
             (LOOP (IF (%> I ARGS)
                       (RETURN ACCUMULATOR))
                   (SETQ ACCUMULATOR (%* ACCUMULATOR (IL:ARG ARGS I)))
                   (SETQ I (1+ I)))))))


(/
  (IL:LAMBDA ARGS                                                    ; Edited  8-Feb-87 19:15 by jop
    (COND
       ((EQ ARGS 0)
        (ERROR "/ requires at least one argument"))
       ((EQ ARGS 1)
        (%RECIPROCOL (IL:ARG ARGS 1)))
       (T (LET ((ACCUMULATOR (IL:ARG ARGS 1))
                (I 2))
               (LOOP (IF (%> I ARGS)
                         (RETURN ACCUMULATOR))
                     (SETQ ACCUMULATOR (%/ ACCUMULATOR (IL:ARG ARGS I)))
                     (SETQ I (1+ I))))))))
```

```
)


(DEFUN 1+ (NUMBER)
   (+ NUMBER 1))


(DEFUN 1- (NUMBER)
   (- NUMBER 1))


(DEFUN %RECIPROCOL (NUMBER)
   (IF (FLOATP NUMBER)
       (IL:FQUOTIENT 1.0 NUMBER)
       (/ 1 NUMBER)))


(XCL:DEFOPTIMIZER + (&REST NUMBERS)
                    (IF (NULL NUMBERS)
                        0
                        (LET ((FORM (CAR NUMBERS)))
                             (DOLIST (NUM (CDR NUMBERS)
                                          FORM)
                                 (SETQ FORM `(%+ ,FORM ,NUM))))))


(XCL:DEFOPTIMIZER - (NUMBER &REST NUMBERS)
                    (IF (NULL NUMBERS)
                        `(%- 0 ,NUMBER)
                        (LET ((FORM NUMBER))
                             (DOLIST (NUM NUMBERS FORM)
                                 (SETQ FORM `(%- ,FORM ,NUM))))))


(XCL:DEFOPTIMIZER * (&REST NUMBERS)
                    (IF (NULL NUMBERS)
                        1
                        (LET ((FORM (CAR NUMBERS)))
                             (DOLIST (NUM (CDR NUMBERS)
                                          FORM)
                                 (SETQ FORM `(%* ,FORM ,NUM))))))


(XCL:DEFOPTIMIZER / (NUMBER &REST NUMBERS)
                    (IF (NULL NUMBERS)
                        `(%RECIPROCOL ,NUMBER)
                        (LET ((FORM NUMBER))
                             (DOLIST (NUM NUMBERS FORM)
                                 (SETQ FORM `(%/ ,FORM ,NUM))))))


(XCL:DEFOPTIMIZER 1+ (NUMBER)
                     `(+ ,NUMBER 1))


(XCL:DEFOPTIMIZER 1- (NUMBER)
                     `(- ,NUMBER 1))
```

;; For the byte compiler

```
(IL:PUTPROPS + IL:DMACRO (IL:ARGS (IL:|if| (IL:GREATERP (IL:LENGTH IL:ARGS)
                                                         1)
                                           IL:|then| `(IL:PLUS ,@IL:ARGS)
                                           IL:|else| 'IL:IGNOREMACRO)))

(IL:PUTPROPS * IL:DMACRO (IL:ARGS (IL:|if| (IL:GREATERP (IL:LENGTH IL:ARGS)
                                                         1)
                                           IL:|then| `(IL:TIMES ,@IL:ARGS)
                                           IL:|else| 'IL:IGNOREMACRO)))
```

;; Redefine Interlisp generic arithmetic to work with ratios

```
(IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOCOPY

(IL:MOVD '+ 'IL:PLUS)
```

;; Don't need to redefine difference since it is defined in terms of the difference opcode (redefined above)

```
(IL:MOVD '* 'IL:TIMES)
)
```

;; So Interlisp quotient will do something reasonable with ratios

(IL:* IL:* (IL:MOVD (QUOTE IL:NEW-QUOTIENT) (QUOTE IL:QUOTIENT)))

```
;; because QUOTIENT is already defined in LLARITH to do something useful with ratios. AR 8062.
;; INCF and DECF implemented by CMLSETF.

(DEFUN %GCD (X Y)
    ;; %GCD -- Gcd of two integers, no type checking.
    (SETQ X (%ABS X))
    (SETQ Y (%ABS Y))
    (COND
       ((EQ X 0)
        Y)
       ((EQ Y 0)
        X)
       ((OR (EQL 1 Y)
            (EQL 1 X))
        1)
       (T (LET ((K                                          ; Factor out powers of two
                   (DO ((K 0 (1+ K)))
                       ((OR (ODDP X)
                            (ODDP Y))
                        K)
                     (SETQ X (ASH X -1))
                     (SETQ Y (ASH Y -1)))))
                (DO ((J (IF (ODDP X)
                            (- Y)
                            (ASH X -1))
                        (- X Y)))
                    ((EQ J 0))
                  (LOOP (IF (ODDP J)
                            (RETURN NIL))
                        (SETQ J (ASH J -1)))
                  (IF (PLUSP J)
                      (SETQ X J)
                      (SETQ Y (- J))))
                (ASH X K)))))

(DEFUN %LCM (X Y)
   (COND
      ((EQ X 1)
       Y)
      ((EQ Y 1)
       X)
      ((OR (EQ X 0)
           (EQ Y 0))
       0)
      (T (SETQ X (%ABS X))
         (SETQ Y (%ABS Y))
         (LET ((GCD (%GCD X Y)))
              (IF (EQ GCD 1)
                  (* X Y)
                  (* (IL:IQUOTIENT X GCD)
                     Y))))))

(IL:DEFINEQ

(GCD
  (IL:LAMBDA ARGS                                           ; Edited 10-Feb-87 11:14 by jop
    ;; GCD -- gcd of an arbitrary number of integers.  Since the probability is >.6 that the GCD of two numbers is 1, it is worth to time to check for GCD
    ;; = 1 and quit if so.
    (COND
       ((EQ ARGS 0)
        0)
       ((EQ ARGS 1)
        (%ABS (IL:ARG ARGS 1)))
       (T (LET ((RESULT (%GCD (IL:ARG ARGS 1)
                              (IL:ARG ARGS 2)))
                (I 3))
            (LOOP (IF (OR (> I ARGS)
                          (EQ RESULT 1))
                      (RETURN RESULT))
                  (SETQ RESULT (%GCD RESULT (IL:ARG ARGS I)))
                  (SETQ I (1+ I))))))))

(LCM
  (IL:LAMBDA ARGS                                           ; Edited 25-Feb-87 12:20 by jop
    ;; LCM -- least common multiple.  At least one argument is required.
    (COND
       ((EQ ARGS 0)
        (ERROR "lcm requires at least one argument"))
       ((EQ ARGS 1)
        (%ABS (IL:ARG ARGS 1)))
```

```
       (T (LET ((RESULT (%LCM (IL:ARG ARGS 1)
                              (IL:ARG ARGS 2)))
                (I 3))
               (LOOP (IF (OR (> I ARGS)
                             (EQ RESULT 0))
                         (RETURN RESULT))
                     (SETQ RESULT (%LCM RESULT (IL:ARG ARGS I)))
                     (SETQ I (1+ I)))))))

)
```

;; Optimizers for Interlisp functions, so that they compile open with the PavCompiler.

;; optimizer of IL:minus

```
(XCL:DEFOPTIMIZER IL:MINUS (IL:X)
                           `(- 0 ,IL:X))

(XCL:DEFOPTIMIZER IL:PLUS (&REST NUMBERS)
                          (IF (NULL NUMBERS)
                              0
                              (LET ((FORM (CAR NUMBERS)))
                                   (DOLIST (NUM (CDR NUMBERS)
                                                FORM)
                                       (SETQ FORM `(IL:PLUS2 ,FORM ,NUM))))))

(XCL:DEFOPTIMIZER IL:IPLUS (&REST NUMBERS)
                           (IF (NULL NUMBERS)
                               0
                               (LET ((FORM (CAR NUMBERS)))
                                    (COND
                                       ((CDR NUMBERS)
                                        (DOLIST (NUM (CDR NUMBERS)
                                                     FORM)
                                            (SETQ FORM `(IL:IPLUS2 ,FORM ,NUM))))
                                       (T `(IL:IPLUS2 ,FORM 0)))))))

(XCL:DEFOPTIMIZER IL:FPLUS (&REST NUMBERS)
                           (IF (NULL NUMBERS)
                               0
                               (LET ((FORM (CAR NUMBERS)))
                                    (DOLIST (NUM (CDR NUMBERS)
                                                 FORM)
                                        (SETQ FORM `(IL:FPLUS2 ,FORM ,NUM))))))

(XCL:DEFOPTIMIZER IL:TIMES (&REST NUMBERS)
                           (IF (NULL NUMBERS)
                               0
                               (LET ((FORM (CAR NUMBERS)))
                                    (DOLIST (NUM (CDR NUMBERS)
                                                 FORM)
                                        (SETQ FORM `(IL:TIMES2 ,FORM ,NUM))))))

(XCL:DEFOPTIMIZER IL:ITIMES (&REST NUMBERS)
                            (IF (NULL NUMBERS)
                                0
                                (LET ((FORM (CAR NUMBERS)))
                                     (DOLIST (NUM (CDR NUMBERS)
                                                  FORM)
                                         (SETQ FORM `(IL:ITIMES2 ,FORM ,NUM))))))

(XCL:DEFOPTIMIZER IL:FTIMES (&REST NUMBERS)
                            (IF (NULL NUMBERS)
                                1.0
                                (LET ((FORM (CAR NUMBERS)))
                                     (DOLIST (NUM (CDR NUMBERS)
                                                  FORM)
                                         (SETQ FORM `(IL:FTIMES2 ,FORM ,NUM))))))

(XCL:DEFOPTIMIZER IL:RSH (IL:VALUE IL:SHIFT-AMOUNT)
                         `(IL:LSH ,IL:VALUE (IL:IMINUS ,IL:SHIFT-AMOUNT)))

(IL:PUTPROPS IL:PLUS2 IL:DOPVAL (2 IL:PLUS2))

(IL:PUTPROPS IL:IPLUS2 IL:DOPVAL (2 IL:IPLUS2))

(IL:PUTPROPS IL:FPLUS2 IL:DOPVAL (2 IL:FPLUS2))

(IL:PUTPROPS IL:TIMES2 IL:DOPVAL (2 IL:TIMES2))
```

```
(IL:PUTPROPS IL:ITIMES2 IL:DOPVAL (2 IL:ITIMES2))

(IL:PUTPROPS IL:FTIMES2 IL:DOPVAL (2 IL:FTIMES2))
```

;;; Section 12.5 Irrational and Transcendental functions. Most of these will be found on cmlfloat.

```
(DEFUN ISQRT (INTEGER)
    ;; ISQRT: Integer square root --- isqrt (n) **2 <= n.  Upper and lower bounds on the result are estimated using integer-length.  On each iteration, one
    ;; of the bounds is replaced by their mean.  The lower bound is returned when the bounds meet or differ by only 1. Initial bounds guarantee that lg
    ;; (sqrt (n)) = lg (n) /2 iterations suffice.
    (IF (NOT (AND (INTEGERP INTEGER)
                  (>= INTEGER 0)))
        (ERROR 'XCL:TYPE-MISMATCH :EXPECTED-TYPE '(INTEGER 0)
               :NAME INTEGER :VALUE INTEGER :MESSAGE "a nonnegative integer"))
    (LET* ((ILENGTH (INTEGER-LENGTH INTEGER))
           (LOW (ASH 1 (ASH (1- ILENGTH)
                            -1)))
           (HIGH (+ LOW (ASH LOW (IF (ODDP ILENGTH)
                                     -1
                                     0)))))
          (DO ((MID (ASH (+ LOW HIGH)
                         -1)
                    (ASH (+ LOW HIGH)
                         -1)))
              ((<= (1- HIGH)
                   LOW)
               LOW)
            (IF (<= (* MID MID)
                    INTEGER)
                (SETQ LOW MID)
                (SETQ HIGH MID)))))
```

;; Abs is shared with il: abs ia also defined in llarith.

```
(DEFUN ABS (NUMBER)
   (TYPECASE NUMBER
       (INTEGER (IF (< NUMBER 0)
                    (- 0 NUMBER)
                   NUMBER))
       (FLOAT (IF (< NUMBER 0.0)
                  (- 0.0 NUMBER)
                  NUMBER))
       (RATIO (IF (< (RATIO-NUMERATOR NUMBER)
                     0)
                  (%MAKE-RATIO (- 0 (RATIO-NUMERATOR NUMBER))
                          (RATIO-DENOMINATOR NUMBER))
                  NUMBER))
       (COMPLEX (%COMPLEX-ABS NUMBER))
       (T (%NOT-NUMBER-ERROR NUMBER))))

(DEFMACRO %ABS (INTEGER)
    ;; Integer version of abs
    `((IL:OPENLAMBDA (X)
        (IF (< X 0)
            (- 0 X)
           X))
      ,INTEGER))

(DEFUN SIGNUM (NUMBER)
    ;; If NUMBER is zero, return NUMBER, else return (/ NUMBER (ABS NUMBER)).
    (IF (ZEROP NUMBER)
        NUMBER
        (TYPECASE NUMBER
            (RATIONAL (IF (PLUSP NUMBER)
                          1
                          -1))
            (FLOAT (IF (PLUSP NUMBER)
                       1.0
                       -1.0))
            (COMPLEX (/ NUMBER (ABS NUMBER)))
            (OTHERWISE (%NOT-NUMBER-ERROR NUMBER)))))

(DEFMACRO %SIGNUM (INTEGER)
    ;; Integer version of signum
    `((IL:OPENLAMBDA (X)
        (COND
```

```
          ((EQ X 0)
           0)
          ((PLUSP X)
           1)
          (T -1)))
     ,INTEGER))
```

;;; Section 12.6 Type Conversions and Component Extractions on Numbers.

;; Float implemented in cmlfloat

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

(IL:FILESLOAD IL:UNBOXEDOPS)
)
```

;; These should be exported from xcl

```
(DEFUN XCL::STRUNCATE (NUMBER &OPTIONAL DIVISOR)
    ;; Returns number (or number/divisor) as an integer, rounded toward 0.0
    (IF (NULL DIVISOR)
        (TYPECASE NUMBER
            (FLOAT
                ;; Could be (IL:FIX NUMBER), but this is slightly faster
                (IL:\\FIXP.FROM.FLOATP NUMBER))
            (RATIO (IL:IQUOTIENT (RATIO-NUMERATOR NUMBER)
                                 (RATIO-DENOMINATOR NUMBER)))
            (INTEGER NUMBER)
            (OTHERWISE (%NOT-NONCOMPLEX-NUMBER-ERROR NUMBER)))
        (TYPECASE DIVISOR
            (INTEGER (IL:IQUOTIENT NUMBER DIVISOR))
            (FLOAT (LET ((FX (FLOAT NUMBER))
                         (FY (FLOAT DIVISOR)))
                        (DECLARE (TYPE FLOAT FX FY))
                        (IL:UFIX (IL:FQUOTIENT FX FY))))
            (RATIO (XCL::STRUNCATE (/ NUMBER DIVISOR)))
            (OTHERWISE (%NOT-NONCOMPLEX-NUMBER-ERROR DIVISOR)))))

(DEFUN XCL::SFLOOR (NUMBER &OPTIONAL DIVISOR)
    ;; Returns the greatest integer not greater than number, or number/divisor.
    (IF (NULL DIVISOR)
        (LET ((RESULT (XCL::STRUNCATE NUMBER)))
             (COND
                ((= RESULT NUMBER)
                 RESULT)
                ((< NUMBER 0)
                 (1- RESULT))
                (T RESULT)))
        (LET ((RESULT (XCL::STRUNCATE NUMBER DIVISOR)))
             (IF (= (REM NUMBER DIVISOR)
                    0)
                 RESULT
                 (IF (< NUMBER 0)
                     (IF (< DIVISOR 0)
                         RESULT
                         (1- RESULT))
                     (IF (< DIVISOR 0)
                         (1- RESULT)
                         RESULT)))))))

(DEFUN XCL::SCEILING (NUMBER &OPTIONAL DIVISOR)
    ;; Returns the least integer not less than number, or number/divisor.
    (IF (NULL DIVISOR)
        (LET ((RESULT (XCL::STRUNCATE NUMBER)))
             (COND
                ((= RESULT NUMBER)
                 RESULT)
                ((< NUMBER 0)
                 RESULT)
                (T (1+ RESULT))))
        (LET ((RESULT (XCL::STRUNCATE NUMBER DIVISOR)))
             (IF (= (REM NUMBER DIVISOR)
                    0)
                 RESULT
                 (IF (< NUMBER 0)
                     (IF (< DIVISOR 0)
                         (1+ RESULT)
                         RESULT)
                     (IF (< DIVISOR 0)
```

```
                        RESULT
                        (1+ RESULT)))))))
```

```
(DEFUN XCL::SROUND (NUMBER &OPTIONAL DIVISOR)
    ;; Returns number (or number/divisor) as an integer, rounded toward 0.0
    (IF (NULL DIVISOR)
        (IL:FIXR NUMBER)
        (IF (OR (FLOATP NUMBER)
                (FLOATP DIVISOR))
            (IL:FIXR (IL:FQUOTIENT NUMBER DIVISOR))
            (IL:FIXR (/ NUMBER DIVISOR)))))
```

```
(XCL:DEFOPTIMIZER XCL::STRUNCATE (NUMBER &OPTIONAL DIVISOR)
                                 (IF (INTEGERP DIVISOR)
                                     `(IL:IQUOTIENT ,NUMBER ,DIVISOR)
                                     'COMPILER:PASS))
```

```
(XCL:DEFOPTIMIZER XCL::SROUND (NUMBER &OPTIONAL (DIVISOR NIL DIVISOR-P))
                              (IF (NULL DIVISOR-P)
                                  `(IL:FIXR ,NUMBER)
                                  'COMPILER:PASS))
```

;; Round is shared with il: (?!)

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

```
(DEFMACRO %INTEGER-COERCE-MACRO (SINGLE-VALUE-FN NUMBER DIVISOR &OPTIONAL FLOAT-RESULT)
   `(LET* ((RESULT (IF (NULL ,DIVISOR)
                       (,SINGLE-VALUE-FN ,NUMBER)
                       (,SINGLE-VALUE-FN ,NUMBER ,DIVISOR)))
           (REMAINDER (IF (NULL ,DIVISOR)
                          (- ,NUMBER RESULT)
                          (- ,NUMBER (* ,DIVISOR RESULT)))))
          (VALUES ,(IF FLOAT-RESULT
                       '(FLOAT RESULT)
                       'RESULT)
                  REMAINDER)))
)
```

```
(DEFUN TRUNCATE (NUMBER &OPTIONAL DIVISOR)
    ;; Returns number (or number/divisor) as an integer, rounded toward 0.0 The second returned value is the remainder.
    (%INTEGER-COERCE-MACRO XCL::STRUNCATE NUMBER DIVISOR))
```

```
(DEFUN FLOOR (NUMBER &OPTIONAL DIVISOR)
    ;; Returns number (or number/divisor) as an integer, rounded toward - infinity. The second returned value is the remainder.
    (%INTEGER-COERCE-MACRO XCL::SFLOOR NUMBER DIVISOR))
```

```
(DEFUN CEILING (NUMBER &OPTIONAL DIVISOR)
    ;; Returns number (or number/divisor) as an integer, rounded toward + infinity. The second returned value is the remainder.
    (%INTEGER-COERCE-MACRO XCL::SCEILING NUMBER DIVISOR))
```

```
(DEFUN ROUND (NUMBER &OPTIONAL DIVISOR)
    ;; Returns number (or number/divisor) as an integer, rounded toward nearest integer. The second returned value is the remainder.
    (%INTEGER-COERCE-MACRO XCL::SROUND NUMBER DIVISOR))
```

```
(DEFUN %INTEGER-COERCE-OPTIMIZER (SINGLE-VALUE-FN NUMBER DIVISOR CONTEXT &OPTIONAL FLOAT-RESULT)
   (IF (EQ 1 (COMPILER:CONTEXT-VALUES-USED CONTEXT))
       (LET ((FORM `(,SINGLE-VALUE-FN ,NUMBER ,@(IF DIVISOR (LIST DIVISOR)))))
            (IF FLOAT-RESULT
                `(FLOAT ,FORM)
                FORM))
       'COMPILER:PASS))
```

```
(XCL:DEFOPTIMIZER TRUNCATE (NUMBER &OPTIONAL DIVISOR XCL:&CONTEXT CONTEXT)
                           (%INTEGER-COERCE-OPTIMIZER 'XCL::STRUNCATE NUMBER DIVISOR CONTEXT))
```

```
(XCL:DEFOPTIMIZER FLOOR (NUMBER &OPTIONAL DIVISOR XCL:&CONTEXT CONTEXT)
                        (%INTEGER-COERCE-OPTIMIZER 'XCL::SFLOOR NUMBER DIVISOR CONTEXT))
```

```
(XCL:DEFOPTIMIZER CEILING (NUMBER &OPTIONAL DIVISOR XCL:&CONTEXT CONTEXT)
```

```
                          (%INTEGER-COERCE-OPTIMIZER 'XCL::SCEILING NUMBER DIVISOR CONTEXT))

(XCL:DEFOPTIMIZER ROUND (NUMBER &OPTIONAL DIVISOR XCL:&CONTEXT CONTEXT)
                        (%INTEGER-COERCE-OPTIMIZER 'XCL::SROUND NUMBER DIVISOR CONTEXT))

(DEFUN FTRUNCATE (NUMBER &OPTIONAL DIVISOR)
   ;; Returns number (or number/divisor) as an integer, rounded toward 0.0 The second returned value is the remainder.
   (%INTEGER-COERCE-MACRO XCL::STRUNCATE NUMBER DIVISOR T))

(DEFUN FFLOOR (NUMBER &OPTIONAL DIVISOR)
   ;; Like floor, but returns the first result as a float
   (%INTEGER-COERCE-MACRO XCL::SFLOOR NUMBER DIVISOR T))

(DEFUN FCEILING (NUMBER &OPTIONAL DIVISOR)
   ;; Returns number (or number/divisor) as an integer, rounded toward + infinity. The second returned value is the remainder.
   (%INTEGER-COERCE-MACRO XCL::SCEILING NUMBER DIVISOR T))

(DEFUN FROUND (NUMBER &OPTIONAL DIVISOR)
   ;; Returns number (or number/divisor) as an integer, rounded toward nearest integer. The second returned value is the remainder.
   (%INTEGER-COERCE-MACRO XCL::SROUND NUMBER DIVISOR T))

(XCL:DEFOPTIMIZER FTRUNCATE (NUMBER &OPTIONAL DIVISOR XCL:&CONTEXT CONTEXT)
                            (%INTEGER-COERCE-OPTIMIZER 'XCL::STRUNCATE NUMBER DIVISOR CONTEXT T))

(XCL:DEFOPTIMIZER FFLOOR (NUMBER &OPTIONAL DIVISOR XCL:&CONTEXT CONTEXT)
                         (%INTEGER-COERCE-OPTIMIZER 'XCL::SFLOOR NUMBER DIVISOR CONTEXT T))

(XCL:DEFOPTIMIZER FCEILING (NUMBER &OPTIONAL DIVISOR XCL:&CONTEXT CONTEXT)
                           (%INTEGER-COERCE-OPTIMIZER 'XCL::SCEILING NUMBER DIVISOR CONTEXT T))

(XCL:DEFOPTIMIZER FROUND (NUMBER &OPTIONAL DIVISOR XCL:&CONTEXT CONTEXT)
                         (%INTEGER-COERCE-OPTIMIZER 'XCL::SROUND NUMBER DIVISOR CONTEXT T))

(DEFUN MOD (NUMBER DIVISOR)
   ;; Returns second result of FLOOR.
   (IF (OR (FLOATP NUMBER)
           (FLOATP DIVISOR))
       (LET ((FX (FLOAT NUMBER))
             (FY (FLOAT DIVISOR))
             REM)
            (DECLARE (TYPE FLOAT FX FY REM))
            (SETQ REM (- FX (* (FLOAT (IL:UFIX (IL:FQUOTIENT FX FY)))
                               FY)))
            (IF (IL:UFEQP REM 0.0)
                0.0
                (IF (IF (IL:UFGREATERP 0.0 FY)
                        (IL:UFGREATERP FX 0.0)
                        (IL:UFGREATERP 0.0 FX))
                    (SETQ REM (+ REM FY))))
            REM)
       (LET ((REM (REM NUMBER DIVISOR)))
            (IF (AND (NOT (ZEROP REM))
                     (IF (MINUSP DIVISOR)
                         (PLUSP NUMBER)
                         (MINUSP NUMBER)))
                (+ REM DIVISOR)
                REM))))

(DEFUN REM (NUMBER DIVISOR)
   ;; Returns the second value of truncate
   (COND
      ((AND (INTEGERP NUMBER)
            (INTEGERP DIVISOR))
       (IL:IREMAINDER NUMBER DIVISOR))
      ((OR (FLOATP NUMBER)
           (FLOATP DIVISOR))
       (LET ((FX (FLOAT NUMBER))
             (FY (FLOAT DIVISOR)))
            (DECLARE (TYPE FLOAT FX FY))
            (SETQ FX (- FX (* (FLOAT (IL:UFIX (IL:FQUOTIENT FX FY)))
```

```
                              FY)))))
      (T (- NUMBER (* DIVISOR (XCL::STRUNCATE NUMBER DIVISOR))))))
```

;; Should IL:remainder be equivalent to cl:rem?. Thereis no IL:mod in the IRM, although it has a macro which makes it equivalent to imod.
;; See cmlfloat for ffloor and friends, decode-float and friends

;;; Section 12.7 Logical Operations on Numbers.
;; LOGXOR and LOGAND are shared with IL. (definitions in llarith)

```
(DEFUN %LOGICAL-OPTIMIZER (BINARY-LOGICAL-FN IDENTITY FIRST-INTEGER SECOND-INTEGER MORE-INTEGERS)
   (COND
      ((NULL FIRST-INTEGER)
       IDENTITY)
      ((NULL SECOND-INTEGER)
       FIRST-INTEGER)
      ((NULL MORE-INTEGERS)
       `(,BINARY-LOGICAL-FN ,FIRST-INTEGER ,SECOND-INTEGER))
      (T (LET ((FORM `(,BINARY-LOGICAL-FN ,FIRST-INTEGER ,SECOND-INTEGER)))
           (DOLIST (INTEGER MORE-INTEGERS FORM)
               (SETQ FORM `(,BINARY-LOGICAL-FN ,FORM ,INTEGER)))))))


(XCL:DEFOPTIMIZER LOGXOR (FIRST-INTEGER SECOND-INTEGER &REST MORE-INTEGERS)
                         (IF (AND COMPILER::*NEW-COMPILER-IS-EXPANDING* MORE-INTEGERS)
                             (%LOGICAL-OPTIMIZER 'LOGXOR 0 FIRST-INTEGER SECOND-INTEGER MORE-INTEGERS)
                             'COMPILER:PASS))


(XCL:DEFOPTIMIZER LOGAND (FIRST-INTEGER SECOND-INTEGER &REST MORE-INTEGERS)
                         (IF (AND COMPILER::*NEW-COMPILER-IS-EXPANDING* MORE-INTEGERS)
                             (%LOGICAL-OPTIMIZER 'LOGAND -1 FIRST-INTEGER SECOND-INTEGER MORE-INTEGERS)
                             'COMPILER:PASS))


(DEFUN %LOGIOR (X Y)
   (IL:LOGOR X Y))


(DEFMACRO %LOGEQV (X Y)
   `(LOGNOT (LOGXOR ,X ,Y)))

(IL:PUTPROPS %LOGIOR IL:DOPVAL (2 IL:LOGOR2))
```

;; for the byte compiler

```
(IL:PUTPROPS %LOGIOR IL:DMACRO (= . IL:LOGOR))

(IL:DEFINEQ
```

(LOGIOR

```
  (IL:LAMBDA ARGS                                                    ; Edited 11-Feb-87 11:22 by jop
```

;; Cannot be called interpreted. This defn relies on fact that the compiler turns class to %LOGIOR calls into a sequence of opcodes

```
    (COND
       ((EQ ARGS 0)
        0)
       ((EQ ARGS 1)
        (IL:ARG ARGS 1))
       ((EQ ARGS 2)
        (%LOGIOR (IL:ARG ARGS 1)
                 (IL:ARG ARGS 2)))
       (T (LET ((RESULT (%LOGIOR (IL:ARG ARGS 1)
                                 (IL:ARG ARGS 2)))
                (I 3))
            (LOOP (IF (%> I ARGS)
                      (RETURN RESULT))
                  (SETQ RESULT (%LOGIOR RESULT (IL:ARG ARGS I)))
                  (SETQ I (1+ I)))))))))
```

(LOGEQV

```
  (IL:LAMBDA ARGS                                                    ; Edited 11-Feb-87 13:20 by jop
```

;; Cannot be called interpreted. This defn relies on fact that the compiler turns class to %LOGIOR calls into a sequence of opcodes

```
    (COND
       ((EQ ARGS 0)
        -1)
       ((EQ ARGS 1)
        (IL:ARG ARGS 1))
       ((EQ ARGS 2)
        (%LOGEQV (IL:ARG ARGS 1)
                 (IL:ARG ARGS 2)))
       (T (LET ((RESULT (%LOGEQV (IL:ARG ARGS 1)
                                 (IL:ARG ARGS 2)))
```

```
               (I 3))
              (LOOP (IF (%> I ARGS)
                        (RETURN RESULT))
                    (SETQ RESULT (%LOGEQV RESULT (IL:ARG ARGS I)))
                    (SETQ I (1+ I))))))))

)

(XCL:DEFOPTIMIZER LOGIOR (FIRST-INTEGER SECOND-INTEGER &REST MORE-INTEGERS)
                         (%LOGICAL-OPTIMIZER '%LOGIOR 0 FIRST-INTEGER SECOND-INTEGER MORE-INTEGERS))

(XCL:DEFOPTIMIZER LOGEQV (FIRST-INTEGER SECOND-INTEGER &REST MORE-INTEGERS)
                         (%LOGICAL-OPTIMIZER '%LOGEQV -1 FIRST-INTEGER SECOND-INTEGER MORE-INTEGERS))

(DEFUN LOGNAND (INTEGER1 INTEGER2)
   (LOGNOT (LOGAND INTEGER1 INTEGER2)))

(DEFUN LOGNOR (INTEGER1 INTEGER2)
   (LOGNOT (LOGIOR INTEGER1 INTEGER2)))

(DEFUN LOGANDC1 (INTEGER1 INTEGER2)
   (LOGAND (LOGNOT INTEGER1)
           INTEGER2))

(DEFUN LOGANDC2 (INTEGER1 INTEGER2)
   (LOGAND INTEGER1 (LOGNOT INTEGER2)))

(DEFUN LOGORC1 (INTEGER1 INTEGER2)
   (LOGIOR (LOGNOT INTEGER1)
           INTEGER2))

(DEFUN LOGORC2 (INTEGER1 INTEGER2)
   (LOGIOR INTEGER1 (LOGNOT INTEGER2)))

(XCL:DEFOPTIMIZER LOGNAND (INTEGER1 INTEGER2)
                          `(LOGNOT (LOGAND ,INTEGER1 ,INTEGER2)))

(XCL:DEFOPTIMIZER LOGNOR (INTEGER1 INTEGER2)
                         `(LOGNOT (LOGIOR ,INTEGER1 ,INTEGER2)))

(XCL:DEFOPTIMIZER LOGANDC1 (INTEGER1 INTEGER2)
                           `(LOGAND (LOGNOT ,INTEGER1)
                                    ,INTEGER2))

(XCL:DEFOPTIMIZER LOGANDC2 (INTEGER1 INTEGER2)
                           `(LOGAND ,INTEGER1 (LOGNOT ,INTEGER2)))

(XCL:DEFOPTIMIZER LOGORC1 (INTEGER1 INTEGER2)
                          `(LOGIOR (LOGNOT ,INTEGER1)
                                   ,INTEGER2))

(XCL:DEFOPTIMIZER LOGORC2 (INTEGER1 INTEGER2)
                          `(LOGIOR ,INTEGER1 (LOGNOT ,INTEGER2)))

(DEFCONSTANT BOOLE-CLR 0)

(DEFCONSTANT BOOLE-SET 1)

(DEFCONSTANT BOOLE-1 2)

(DEFCONSTANT BOOLE-2 3)

(DEFCONSTANT BOOLE-C1 4)

(DEFCONSTANT BOOLE-C2 5)
```

```
(DEFCONSTANT BOOLE-AND 6)

(DEFCONSTANT BOOLE-IOR 7)

(DEFCONSTANT BOOLE-XOR 8)

(DEFCONSTANT BOOLE-EQV 9)

(DEFCONSTANT BOOLE-NAND 10)

(DEFCONSTANT BOOLE-NOR 11)

(DEFCONSTANT BOOLE-ANDC1 12)

(DEFCONSTANT BOOLE-ANDC2 13)

(DEFCONSTANT BOOLE-ORC1 14)

(DEFCONSTANT BOOLE-ORC2 15)

(DEFUN BOOLE (OP INTEGER1 INTEGER2)
   (COND
      ((EQ OP BOOLE-CLR)
       0)
      ((EQ OP BOOLE-SET)
       -1)
      ((EQ OP BOOLE-1)
       INTEGER1)
      ((EQ OP BOOLE-2)
       INTEGER2)
      ((EQ OP BOOLE-C1)
       (LOGNOT INTEGER1))
      ((EQ OP BOOLE-C2)
       (LOGNOT INTEGER2))
      ((EQ OP BOOLE-AND)
       (LOGAND INTEGER1 INTEGER2))
      ((EQ OP BOOLE-IOR)
       (LOGIOR INTEGER1 INTEGER2))
      ((EQ OP BOOLE-XOR)
       (LOGXOR INTEGER1 INTEGER2))
      ((EQ OP BOOLE-EQV)
       (LOGEQV INTEGER1 INTEGER2))
      ((EQ OP BOOLE-NAND)
       (LOGNAND INTEGER1 INTEGER2))
      ((EQ OP BOOLE-NOR)
       (LOGNOR INTEGER1 INTEGER2))
      ((EQ OP BOOLE-ANDC1)
       (LOGANDC1 INTEGER1 INTEGER2))
      ((EQ OP BOOLE-ANDC2)
       (LOGANDC2 INTEGER1 INTEGER2))
      ((EQ OP BOOLE-ORC1)
       (LOGORC1 INTEGER1 INTEGER2))
      ((EQ OP BOOLE-ORC2)
       (LOGORC2 INTEGER1 INTEGER2))
      (T (ERROR "Not a valid op: ~s" OP))))
```

;; Lognot is shared with IL.(in addarith)

```
(DEFUN LOGTEST (INTEGER1 INTEGER2)
   (NOT (EQ 0 (LOGAND INTEGER1 INTEGER2))))

(XCL:DEFINLINE LOGBITP (INDEX INTEGER)
   (EQ 1 (LOGAND 1 (ASH INTEGER (- INDEX)))))

(XCL:DEFOPTIMIZER LOGTEST (INTEGER1 INTEGER2)
                          `(NOT (EQ 0 (LOGAND ,INTEGER1 ,INTEGER2))))

(DEFUN ASH (INTEGER COUNT)
   (IL:LSH INTEGER COUNT))

(IL:PUTPROPS ASH IL:DOPVAL (2 IL:LSH))
```

```
;; For the byte compiler

(IL:PUTPROPS ASH IL:DMACRO (= . IL:LSH))


(DEFUN LOGCOUNT (INTEGER)
    ;; Logcount returns the number of bits that are the complement of the sign in the integer argument x.
    ;; If INTEGER is negative, then the number of 0 bits is returned, otherwise number of 1 bits is returned.
    (IF (MINUSP INTEGER)
        (SETQ INTEGER (LOGNOT INTEGER)))
    (IF (NOT (IL:TYPENAMEP INTEGER 'BIGNUM))
        (%LOGCOUNT INTEGER)
        (%BIGNUM-LOGCOUNT INTEGER)))


(DEFUN %LOGCOUNT (POSITIVE-INTEGER)
    ;; Returns number of 1 bits in nonnegative integer N.
    (LET ((CNT 0))                                                          ; This loop uses a LOGAND trick to reduce the number of
                                                                            ; iterations.
         (LOOP (IF (EQ 0 POSITIVE-INTEGER)
                   (RETURN CNT))                                            ; Change rightmost 1 bit of N to a 0 bit.
               (SETQ CNT (1+ CNT))
               (SETQ POSITIVE-INTEGER (LOGAND POSITIVE-INTEGER (1- POSITIVE-INTEGER))))))

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

(IL:FILESLOAD (IL:LOADCOMP)
       IL:LLBIGNUM)
)

;; Should be in llbignum


(DEFUN %BIGNUM-LOGCOUNT (BIGNUM)
   (LET ((ELEMENTS (IL:|fetch| (BIGNUM IL:ELEMENTS) IL:|of| BIGNUM))
         (CNT 0))
        (DOLIST (ELEMENT ELEMENTS CNT)
            (SETQ CNT (+ CNT (%LOGCOUNT ELEMENT))))))


(DEFUN INTEGER-LENGTH (INTEGER)
   (COND
      ((< INTEGER 0)
       (SETQ INTEGER (- -1 INTEGER))))
    ;; This algorithm is basicly a binary search
   (MACROLET ((BITS-OR-LESS-P (INTEGER N)
                    `(< ,INTEGER ,(ASH 1 N))))
          (IF (BITS-OR-LESS-P INTEGER 16)
              (COND
                 ((BITS-OR-LESS-P INTEGER 8)
                  (COND
                     ((BITS-OR-LESS-P INTEGER 4)
                      (COND
                         ((BITS-OR-LESS-P INTEGER 2)
                          (IF (BITS-OR-LESS-P INTEGER 1)
                              INTEGER
                              2))
                         ((BITS-OR-LESS-P INTEGER 3)
                          3)
                         (T 4)))
                     ((BITS-OR-LESS-P INTEGER 6)
                      (IF (BITS-OR-LESS-P INTEGER 5)
                          5
                          6))
                     ((BITS-OR-LESS-P INTEGER 7)
                      7)
                     (T 8)))
                 ((BITS-OR-LESS-P INTEGER 12)
                  (COND
                     ((BITS-OR-LESS-P INTEGER 10)
                      (IF (BITS-OR-LESS-P INTEGER 9)
                          9
                          10))
                     ((BITS-OR-LESS-P INTEGER 11)
                      11)
                     (T 12)))
                 ((BITS-OR-LESS-P INTEGER 14)
                  (IF (BITS-OR-LESS-P INTEGER 13)
                      13
                      14))
                 ((BITS-OR-LESS-P INTEGER 15)
                  15)
                 (T 16))
```

```
              (+ 16 (INTEGER-LENGTH (ASH INTEGER -16))))))
```

;; OPTIMIZERS FOR IL:LLSH AND IL:LRSH

```
(DEFUN %LLSH8 (X)
   (IL:LLSH X 8))

(DEFUN %LLSH1 (X)
   (IL:LLSH X 1))

(DEFUN %LRSH8 (X)
   (IL:LRSH X 8))

(DEFUN %LRSH1 (X)
   (IL:LRSH X 1))

(IL:PUTPROPS %LLSH8 IL:DOPVAL (1 IL:LLSH8))

(IL:PUTPROPS %LLSH1 IL:DOPVAL (1 IL:LLSH1))

(IL:PUTPROPS %LRSH8 IL:DOPVAL (1 IL:LRSH8))

(IL:PUTPROPS %LRSH1 IL:DOPVAL (1 IL:LRSH1))

(XCL:DEFOPTIMIZER IL:LLSH (X N)
                          (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
                              (LET ((M (AND (CONSTANTP N)
                                            (EVAL N))))
                                   (IF (TYPEP M '(INTEGER 0))
                                       (LET ((FORM X))
                                            (LOOP (IF (< M 8)
                                                      (RETURN NIL))
                                                  (SETQ FORM `(%LLSH8 ,FORM))
                                                  (DECF M 8))
                                            (LOOP (IF (<= M 0)
                                                      (RETURN NIL))
                                                  (SETQ FORM `(%LLSH1 ,FORM))
                                                  (DECF M 1))
                                            FORM)
                                       'COMPILER:PASS))
                              'COMPILER:PASS))

(XCL:DEFOPTIMIZER IL:LRSH (X N)
                          (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
                              (LET ((M (AND (CONSTANTP N)
                                            (EVAL N))))
                                   (IF (TYPEP M '(INTEGER 0))
                                       (LET ((FORM X))
                                            (LOOP (IF (< M 8)
                                                      (RETURN NIL))
                                                  (SETQ FORM `(%LRSH8 ,FORM))
                                                  (DECF M 8))
                                            (LOOP (IF (<= M 0)
                                                      (RETURN NIL))
                                                  (SETQ FORM `(%LRSH1 ,FORM))
                                                  (DECF M 1))
                                            FORM)
                                       'COMPILER:PASS))
                              'COMPILER:PASS))
```

;;; Section 12.8 Byte Manipulations Functions.

```
(DEFUN BYTE (SIZE POSITION)
   (IF (OR (< SIZE 0)
           (< POSITION 0))
       (ERROR "Not a valid bytespec: ~s ~s" SIZE POSITION)
       (IF (AND (< SIZE 256)
                (< POSITION 256))
           (+ (ASH SIZE 8)
              POSITION)
           (CONS SIZE POSITION))))

(XCL:DEFINLINE BYTE-SIZE (BYTESPEC)
   (IF (TYPEP BYTESPEC 'FIXNUM)
       (ASH BYTESPEC -8)
       (CAR BYTESPEC)))
```

```
(XCL:DEFINLINE BYTE-POSITION (BYTESPEC)
   (IF (TYPEP BYTESPEC 'FIXNUM)
       (LOGAND BYTESPEC 255)
       (CDR BYTESPEC)))
```

;; Byte doesn't need an optimizer since the side-effects data-base will do constant folding, but the byte-compiler can profit from an optimizer

```
(DEFUN OPTIMIZE-BYTE (SIZE POSITION)
   (IF (AND (TYPEP SIZE '(INTEGER 0 255))
            (TYPEP POSITION '(INTEGER 0 255)))
       (+ (ASH SIZE 8)
          POSITION)
       'COMPILER:PASS))

(IL:PUTPROPS BYTE IL:DMACRO (IL:ARGS (OPTIMIZE-BYTE (CAR IL:ARGS)
                                                    (CADR IL:ARGS))))

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


(DEFMACRO %MAKE-BYTE-MASK-1 (SIZE POSITION)
   (IF (EQ POSITION 0)
       `(1- (ASH 1 ,SIZE))
       `(ASH (1- (ASH 1 ,SIZE))
             ,POSITION)))


(DEFMACRO %MAKE-BYTE-MASK-0 (SIZE POSITION)
   `(LOGNOT (%MAKE-BYTE-MASK-1 ,SIZE ,POSITION)))
)


(DEFUN LDB (BYTESPEC INTEGER)
   (LET ((SIZE (BYTE-SIZE BYTESPEC))
         (POSITION (BYTE-POSITION BYTESPEC)))
        (LOGAND (ASH INTEGER (- POSITION))
                (%MAKE-BYTE-MASK-1 SIZE 0))))


(DEFUN DPB (NEWBYTE BYTESPEC INTEGER)
   (LET ((SIZE (BYTE-SIZE BYTESPEC))
         (POSITION (BYTE-POSITION BYTESPEC)))
        (LOGIOR (ASH (LOGAND NEWBYTE (%MAKE-BYTE-MASK-1 SIZE 0))
                     POSITION)
                (LOGAND INTEGER (%MAKE-BYTE-MASK-0 SIZE POSITION)))))


(DEFUN MASK-FIELD (BYTESPEC INTEGER)
   (LET ((SIZE (BYTE-SIZE BYTESPEC))
         (POSITION (BYTE-POSITION BYTESPEC)))
        (LOGAND INTEGER (%MAKE-BYTE-MASK-1 SIZE POSITION))))


(DEFUN DEPOSIT-FIELD (NEWBYTE BYTESPEC INTEGER)
   (LET* ((SIZE (BYTE-SIZE BYTESPEC))
          (POSITION (BYTE-POSITION BYTESPEC))
          (MASK (%MAKE-BYTE-MASK-1 SIZE POSITION)))
         (LOGIOR (LOGAND NEWBYTE MASK)
                 (LOGAND INTEGER (LOGNOT MASK)))))


(DEFUN %CONSTANT-BYTESPEC-P (BYTESPEC)
   (COND
      ((TYPEP BYTESPEC 'FIXNUM)
       BYTESPEC)
      ((AND (CONSP BYTESPEC)
            (EQ (CAR BYTESPEC)
                'BYTE)
            (INTEGERP (CADR BYTESPEC))
            (INTEGERP (CADDR BYTESPEC)))
       (EVAL BYTESPEC))
      (T NIL)))


(XCL:DEFOPTIMIZER LDB (BYTESPEC INTEGER)
                      (LET ((CONSTANT-BYTE (%CONSTANT-BYTESPEC-P BYTESPEC)))
                           (IF CONSTANT-BYTE
                               (LET ((SIZE (BYTE-SIZE CONSTANT-BYTE))
                                     (POSITION (BYTE-POSITION CONSTANT-BYTE)))
                                    (IF (ZEROP POSITION)
                                        `(LOGAND ,INTEGER ,(%MAKE-BYTE-MASK-1 SIZE 0))
                                        `(LOGAND (ASH ,INTEGER ,(- POSITION))
                                                 ,(%MAKE-BYTE-MASK-1 SIZE 0))))
                               'COMPILER:PASS)))
```

```
(XCL:DEFOPTIMIZER DPB (NEWBYTE BYTESPEC INTEGER)
                      (LET ((CONSTANT-BYTE (%CONSTANT-BYTESPEC-P BYTESPEC)))
                           (IF CONSTANT-BYTE
                               (LET ((SIZE (BYTE-SIZE CONSTANT-BYTE))
                                     (POSITION (BYTE-POSITION CONSTANT-BYTE)))
                                    (IF (ZEROP POSITION)
                                        `(LOGIOR (LOGAND ,NEWBYTE ,(%MAKE-BYTE-MASK-1 SIZE 0))
                                                 (LOGAND ,INTEGER ,(%MAKE-BYTE-MASK-0 SIZE 0)))
                                        `(LOGIOR (ASH (LOGAND ,NEWBYTE ,(%MAKE-BYTE-MASK-1 SIZE 0))
                                                      ,POSITION)
                                                 (LOGAND ,INTEGER ,(%MAKE-BYTE-MASK-0 SIZE POSITION)))))
                               'COMPILER:PASS)))

(XCL:DEFOPTIMIZER MASK-FIELD (BYTESPEC INTEGER)
                             (LET ((CONSTANT-BYTE (%CONSTANT-BYTESPEC-P BYTESPEC)))
                                  (IF CONSTANT-BYTE
                                      (LET ((SIZE (BYTE-SIZE CONSTANT-BYTE))
                                            (POSITION (BYTE-POSITION CONSTANT-BYTE)))
                                           `(LOGAND ,INTEGER ,(%MAKE-BYTE-MASK-1 SIZE POSITION)))
                                      'COMPILER:PASS)))

(XCL:DEFOPTIMIZER DEPOSIT-FIELD (NEWBYTE BYTESPEC INTEGER)
                                (LET ((CONSTANT-BYTE (%CONSTANT-BYTESPEC-P BYTESPEC)))
                                     (IF CONSTANT-BYTE
                                         (LET* ((SIZE (BYTE-SIZE CONSTANT-BYTE))
                                                (POSITION (BYTE-POSITION CONSTANT-BYTE))
                                                (MASK (%MAKE-BYTE-MASK-1 SIZE POSITION)))
                                               `(LOGIOR (LOGAND ,NEWBYTE ,MASK)
                                                        (LOGAND ,INTEGER ,(LOGNOT MASK))))
                                         'COMPILER:PASS)))

(DEFUN LDB-TEST (BYTESPEC INTEGER)
   (NOT (EQ 0 (LDB BYTESPEC INTEGER))))

(XCL:DEFOPTIMIZER LDB-TEST (BYTESPEC INTEGER)
                           `(NOT (EQ 0 (LDB ,BYTESPEC ,INTEGER))))

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

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

(IL:LOCALVARS . T)
)
)

(IL:PUTPROPS IL:CMLARITH IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "LISP"))

(IL:PUTPROPS IL:CMLARITH IL:FILETYPE COMPILE-FILE)

(IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY IL:COMPILERVARS

(IL:ADDTOVAR IL:NLAMA )

(IL:ADDTOVAR IL:NLAML )

(IL:ADDTOVAR IL:LAMA LOGEQV LOGIOR LCM GCD / * - + >= <= > < /= =)
)
```

FUNCTION INDEX

OPTIMIZER INDEX

MACRO INDEX

PROPERTY INDEX

CONSTANT INDEX

{MEDLEY}<sources>CMLARITH.;1

STRUCTURE INDEX

TYPE INDEX
