

File created: 18-Oct-93 10:31:44 {Pele:mv:envos}<LispCore>Sources>CLTL2>CMLARRAY-SUPPORT.;2

previous date: 12-Oct-93 16:33:46 {Pele:mv:envos}<LispCore>Sources>CLTL2>CMLARRAY-SUPPORT.;1

Read Table: XCL

Package: INTERLISP

Format: XCCS

; Copyright (c) 1986, 1990, 1991, 1992, 1993 by Venue & Xerox Corporation. All rights reserved.

(RPAQQ CMLARRAY-SUPPORTCOMS

(;; Record def's

(RECORDS ARRAY-HEADER GENERAL-ARRAY ONED-ARRAY TWOD-ARRAY)

;; Cmlarray support macros and functions

; Fast predicates

(FUNCTIONS %ARRAYP %SIMPLE-ARRAY-P %SIMPLE-STRING-P %STRINGP %VECTORP)

(FUNCTIONS %CHECK-CIRCLE-PRINT %CHECK-INDICES %CHECK-NOT-WRITEABLE %EXPAND-BIT-OP

%GENERAL-ARRAY-ADJUST-BASE %GET-ARRAY-OFFSET %GET-BASE-ARRAY)

(FUNCTIONS %BIT-TYPE-P %CHAR-TYPE-P %CML-TYPE-TO-TYPENUMBER-EXPANDER %FAT-CHAR-TYPE-P %FAT-STRING-CHAR-P

%GET-TYPE-TABLE-ENTRY %LIT-SIZE-TO-SIZE %LIT-TYPE-TO-TYPE %LLARRAY-MAKE-ACCESSOR-EXPR

%LLARRAY-MAKE-SETTOR-EXPR %LLARRAY-TYPED-GET %LLARRAY-TYPED-PUT %LLARRAY-TYPEP

%MAKE-ARRAY-TYPE-TABLE %MAKE-CML-TYPE-TABLE %PACK-TYPENUMBER %SMALLFIXP-SMALLPOSP

%SMALLPOSP-SMALLFIXP %THIN-CHAR-TYPE-P %THIN-STRING-CHAR-P %TYPE-SIZE-TO-TYPENUMBER

%TYPENUMBER-TO-BITS-PER-ELEMENT %TYPENUMBER-TO-CML-TYPE %TYPENUMBER-TO-DEFAULT-VALUE

%TYPENUMBER-TO-GC-TYPE %TYPENUMBER-TO-SIZE %TYPENUMBER-TO-TYPE \\GETBASESMALL-FIXP

\\GETBASESTRING-CHAR \\GETBASETHINSTRING-CHAR \\PUTBASESMALL-FIXP \\PUTBASESTRING-CHAR

\\PUTBASETHINSTRING-CHAR)

;;; Describes each entry of \\ARRAY-TYPE-TABLE

(STRUCTURES ARRAY-TABLE-ENTRY)

;;; These vars contain all the necessary info for typed arrays

(VARIABLES %LIT-ARRAY-SIZES %LIT-ARRAY-TABLE %LIT-ARRAY-TYPES)

;;; Tables that drives various macros

(VARIABLES %ARRAY-TYPE-TABLE %CANONICAL-CML-TYPES)

;;; Constants for (SIGNED-BYTE 16)

(VARIABLES MAX.SMALLFIXP MIN.SMALLFIXP)

;;; Constants for STRING-CHARS

(VARIABLES %CHAR-TYPE %BIT-TYPE %THIN-CHAR-TYPENUMBER %FAT-CHAR-TYPENUMBER %MAXTHINCHAR)

;;; Array data-type numbers

(VARIABLES %GENERAL-ARRAY %ONED-ARRAY %TWOD-ARRAY)

;;; Compiler options

(DECLARE\ : DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY (LOCALVARS . T))

(PROP FILETYPE CMLARRAY-SUPPORT))

;; Record def's

(DECLARE\ : EVAL@COMPILE

(BLOCKRECORD ARRAY-HEADER (

;;; Describes common slots of all array headers

(NIL BITS 4)

(BASE POINTER)

(READ-ONLY-P FLAG)

(INDIRECT-P FLAG)

(BIT-P FLAG)

(STRING-P FLAG)

(ADJUSTABLE-P FLAG)

(DISPLACED-P FLAG)

(FILL-POINTER-P FLAG)

; First 8 bits are unused

; 24 bits of pointer. Points at raw storage or, in the indirect case,

; at another array header

; 8 bits of flags

; Used for headers pointing at symbols pnames

; Points at an array header rather than a raw storage block

; Is a bit array

; Is a string (implies is a vector)

; If any of the following flags are set, the array in non-simple

```

        (EXTENDABLE-P FLAG)
        (TYPE-NUMBER BITS 8) ; 8 bits of type + size
        (OFFSET WORD) ; For oned and general arrays
        (FILL-POINTER FIXP) ; For oned and general arrays
        (TOTAL-SIZE FIXP))
(BLOCKRECORD ARRAY-HEADER ((NIL POINTER)
        (FLAGS BITS 8)
        (TYPE BITS 4)
        (SIZE BITS 4)))
(ACCESSFNS (SIMPLE-P (EQ 0 (LOGAND (|fetch| (ARRAY-HEADER FLAGS) |of| DATUM)
        15))))
(SYSTEM))

(DATATYPE GENERAL-ARRAY ((NIL BITS 4) ; For alignment
        (STORAGE POINTER) ; 24 bits of pointer
        (READ-ONLY-P FLAG) ; 8 bits of flags
        (INDIRECT-P FLAG)
        (BIT-P FLAG)
        (STRING-P FLAG)
        (ADJUSTABLE-P FLAG)
        (DISPLACED-P FLAG)
        (FILL-POINTER-P FLAG)
        (EXTENDABLE-P FLAG)
        (TYPE-NUMBER BITS 8) ; 8 bits of typenumber
        (OFFSET WORD)
        (FILL-POINTER FIXP) ; As of 2.1, these 2 fields are fixp's.
        (TOTAL-SIZE FIXP)
        (DIMS POINTER)))

(DATATYPE ONED-ARRAY ((NIL BITS 4) ; Don't use high 8 bits
        (BASE POINTER) ; The raw storage base
        (READ-ONLY-P FLAG) ; 8 bits worth of flags
        (NIL BITS 1) ; Oned array's can't be indirect
        (BIT-P FLAG)
        (STRING-P FLAG)
        (NIL BITS 1) ; Oned-array's can't be adjustable
        (DISPLACED-P FLAG)
        (FILL-POINTER-P FLAG)
        (EXTENDABLE-P FLAG)
        (TYPE-NUMBER BITS 8) ; 4 bits of type and 4 bits of size
        (OFFSET WORD) ; For displaced arrays
        (FILL-POINTER FIXP) ; For filled arrays
        (TOTAL-SIZE FIXP) ; Total number of elements
        ))

(DATATYPE TWOD-ARRAY ((NIL BITS 4) ; For alignmnet
        (BASE POINTER) ; Raw storage pointer
        (READ-ONLY-P FLAG) ; 8 bits of flags
        (NIL BITS 1) ; Twod arrays can't be indirect
        (BIT-P FLAG)
        (NIL BITS 4) ; Twod arrays can't be strings, nor can they be adjustable,
        ; displaced, or have fill pointers
        (EXTENDABLE-P FLAG)
        (TYPE-NUMBER BITS 8)
        (BOUND0 FIXP) ; Zero dimension bound
        (BOUND1 FIXP) ; One dimension bound
        (TOTAL-SIZE FIXP)))
)

(/DECLAREDATATYPE 'GENERAL-ARRAY ' ((BITS 4)
        POINTER FLAG FLAG FLAG FLAG FLAG FLAG FLAG (BITS 8)
        WORD FIXP FIXP POINTER)
; ---field descriptor list elided by lister---
' 10)

(/DECLAREDATATYPE 'ONED-ARRAY ' ((BITS 4)
        POINTER FLAG (BITS 1)
        FLAG FLAG (BITS 1)
        FLAG FLAG FLAG (BITS 8)
        WORD FIXP FIXP)
; ---field descriptor list elided by lister---
' 8)

(/DECLAREDATATYPE 'TWOD-ARRAY ' ((BITS 4)
        POINTER FLAG (BITS 1)
        FLAG
        (BITS 4)
        FLAG
        (BITS 8)
        FIXP FIXP FIXP)
; ---field descriptor list elided by lister---
' 10)

```

:: Cmlarray support macros and functions

:: Fast predicates

```
(DEFMACRO %ARRAYP (ARRAY)
  (CL:IF (CL:SYMBOLP ARRAY)
    \ (OR (%ONED-ARRAY-P ,ARRAY)
        (%TWOD-ARRAY-P ,ARRAY)
        (%GENERAL-ARRAY-P ,ARRAY))
    (LET ((SYM (GENSYM)))
      \ (LET ((,SYM ,ARRAY))
          (OR (%ONED-ARRAY-P ,SYM)
              (%TWOD-ARRAY-P ,SYM)
              (%GENERAL-ARRAY-P ,SYM)))))))
```

```
(DEFMACRO %SIMPLE-ARRAY-P (ARRAY)
  (CL:IF (CL:SYMBOLP ARRAY)
    \ (AND (%ARRAYP ,ARRAY)
        (|fetch| (ARRAY-HEADER SIMPLE-P) |of| ,ARRAY))
    (LET ((SYM (GENSYM)))
      \ (LET ((,SYM ,ARRAY))
          (AND (%ARRAYP ,SYM)
              (|fetch| (ARRAY-HEADER SIMPLE-P) |of| ,SYM)))))))
```

```
(DEFMACRO %SIMPLE-STRING-P (STRING)
  (CL:IF (CL:SYMBOLP STRING)
    \ (AND (%ONED-ARRAY-P ,STRING)
        (|fetch| (ARRAY-HEADER SIMPLE-P) |of| ,STRING)
        (|fetch| (ARRAY-HEADER STRING-P) |of| ,STRING))
    (LET ((SYM (GENSYM)))
      \ (LET ((,SYM ,STRING))
          (AND (%ONED-ARRAY-P ,SYM)
              (|fetch| (ARRAY-HEADER SIMPLE-P) |of| ,SYM)
              (|fetch| (ARRAY-HEADER STRING-P) |of| ,SYM)))))))
```

```
(DEFMACRO %STRINGP (STRING)
  (CL:IF (CL:SYMBOLP STRING)
    \ (AND (OR (%ONED-ARRAY-P ,STRING)
              (%GENERAL-ARRAY-P ,STRING))
        (|fetch| (ARRAY-HEADER STRING-P) |of| ,STRING))
    (LET ((SYM (GENSYM)))
      \ (LET ((,SYM ,STRING))
          (AND (OR (%ONED-ARRAY-P ,SYM)
                  (%GENERAL-ARRAY-P ,SYM))
              (|fetch| (ARRAY-HEADER STRING-P) |of| ,SYM)))))))
```

```
(DEFMACRO %VECTORP (VECTOR)
  (CL:IF (CL:SYMBOLP VECTOR)
    \ (OR (%ONED-ARRAY-P ,VECTOR)
        (AND (%GENERAL-ARRAY-P ,VECTOR)
            (EQL 1 (LENGTH (|fetch| (GENERAL-ARRAY DIMS) |of| ,VECTOR)))))
    (LET ((SYM (GENSYM)))
      \ (LET ((,SYM ,VECTOR))
          (OR (%ONED-ARRAY-P ,SYM)
              (AND (%GENERAL-ARRAY-P ,SYM)
                  (EQL 1 (LENGTH (|fetch| (GENERAL-ARRAY DIMS) |of| ,SYM)))))))))
```

```
(DEFMACRO %CHECK-CIRCLE-PRINT (OBJECT STREAM &REST PRINT-FORMS)
  ;; If A has a circle label, print it. If it's not the first time or it has no label, print the contents
  \ (LET (CIRCLELABEL FIRSTTIME)
      (AND *PRINT-CIRCLE-HASHTABLE* (CL:MULTIPLE-VALUE-SETQ (CIRCLELABEL FIRSTTIME)
                                                             (PRINT-CIRCLE-LOOKUP ,OBJECT)))
      (CL:WHEN CIRCLELABEL
        (.SPACECHECK. ,STREAM (VECTOR-LENGTH CIRCLELABEL))
        (LET (*PRINT-CIRCLE-HASHTABLE*)
            (DECLARE (CL:SPECIAL *PRINT-CIRCLE-HASHTABLE*))
            (CL:WRITE-STRING CIRCLELABEL ,STREAM)
            ; No need to print-circle this string (dangerous if we do, in fact)
            (CL:WHEN FIRSTTIME
              (.SPACECHECK. ,STREAM 1)
              (CL:WRITE-CHAR #\Space ,STREAM)))
        (CL:WHEN (OR (NOT CIRCLELABEL)
                    FIRSTTIME)
          ,@PRINT-FORMS)))
```

```
(DEFMACRO %CHECK-INDICES (ARRAY START-ARG ARGS)
  \ (CL:DO ((I ,START-ARG (CL:1+ I))
           (DIM 0 (CL:1+ DIM))
           INDEX)
```

```
( (> I ,ARGS)
  T)
(SETQ INDEX (ARG ,ARGS I))
(CL:IF (OR (< INDEX 0)
           (>= INDEX (CL:ARRAY-DIMENSION ,ARRAY DIM))))
  (RETURN NIL)))
```

```
(DEFMACRO %CHECK-NOT-WRITEABLE (ARRAY TYPE-NUMBER NEWVALUE)
  ` (COND
    ((|fetch| (ARRAY-HEADER READ-ONLY-P) |of| ,ARRAY)
     (%MAKE-ARRAY-WRITEABLE ,ARRAY))
    ((AND (%THIN-CHAR-TYPE-P ,TYPE-NUMBER)
           (%FAT-STRING-CHAR-P ,NEWVALUE))
     (%MAKE-STRING-ARRAY-FAT ,ARRAY))))
```

```
(DEFMACRO %EXPAND-BIT-OP (OP BIT-ARRAY1 BIT-ARRAY2 RESULT-BIT-ARRAY)
  `(PROGN (CL:IF (NOT (BIT-ARRAY-P ,BIT-ARRAY1))
                 (CL:ERROR "BIT-ARRAY1 not a bit array: ~S" ,BIT-ARRAY1))
           (CL:IF (NOT (BIT-ARRAY-P ,BIT-ARRAY2))
                 (CL:ERROR "BIT-ARRAY2 not a bit array: ~S" ,BIT-ARRAY2))
           (CL:IF (NOT (EQUAL-DIMENSIONS-P ,BIT-ARRAY1 ,BIT-ARRAY2))
                 (CL:ERROR "Bit-arrays not of same dimensions")))
    (COND
      ((NULL ,RESULT-BIT-ARRAY)
       (SETQ ,RESULT-BIT-ARRAY (CL:MAKE-ARRAY (CL:ARRAY-DIMENSIONS ,BIT-ARRAY1)
                                                :ELEMENT-TYPE
                                                'BIT)))
      ((EQ ,RESULT-BIT-ARRAY T)
       (SETQ ,RESULT-BIT-ARRAY ,BIT-ARRAY1))
      ((NOT (AND (BIT-ARRAY-P ,RESULT-BIT-ARRAY)
                 (EQUAL-DIMENSIONS-P ,BIT-ARRAY1 ,RESULT-BIT-ARRAY)))
       (CL:ERROR "Illegal result array")))
    , (CL:ECASE OP
      ((AND IOR XOR ANDC2 ORC2) `(OR (EQ ,BIT-ARRAY1 ,RESULT-BIT-ARRAY)
                                     (%DO-LOGICAL-OP 'COPY ,BIT-ARRAY1 ,RESULT-BIT-ARRAY)))
      ((EQV NAND NOR ANDC1 ORC1) `(DO-LOGICAL-OP 'NOT ,BIT-ARRAY1 ,RESULT-BIT-ARRAY)))
    , (CL:ECASE OP
      (AND `(DO-LOGICAL-OP 'AND ,BIT-ARRAY2 ,RESULT-BIT-ARRAY))
      (IOR `(DO-LOGICAL-OP 'OR ,BIT-ARRAY2 ,RESULT-BIT-ARRAY))
      (XOR `(DO-LOGICAL-OP 'XOR ,BIT-ARRAY2 ,RESULT-BIT-ARRAY))
      (EQV `(DO-LOGICAL-OP 'XOR ,BIT-ARRAY2 ,RESULT-BIT-ARRAY))
      (NAND `(DO-LOGICAL-OP 'COR ,BIT-ARRAY2 ,RESULT-BIT-ARRAY))
      (NOR `(DO-LOGICAL-OP 'CAND ,BIT-ARRAY2 ,RESULT-BIT-ARRAY))
      (ANDC1 `(DO-LOGICAL-OP 'AND ,BIT-ARRAY2 ,RESULT-BIT-ARRAY))
      (ANDC2 `(DO-LOGICAL-OP 'CAND ,BIT-ARRAY2 ,RESULT-BIT-ARRAY))
      (ORC1 `(DO-LOGICAL-OP 'OR ,BIT-ARRAY2 ,RESULT-BIT-ARRAY))
      (ORC2 `(DO-LOGICAL-OP 'COR ,BIT-ARRAY2 ,RESULT-BIT-ARRAY)))
    ,RESULT-BIT-ARRAY))
```

```
(DEFMACRO %GENERAL-ARRAY-ADJUST-BASE (ARRAY ROW-MAJOR-INDEX)
  `(CL:IF (|fetch| (GENERAL-ARRAY INDIRECT-P) |of| ,ARRAY)
    (LET ((%OFFSET 0))
      (SETQ ,ARRAY (%GET-BASE-ARRAY ,ARRAY %OFFSET))
      (SETQ ,ROW-MAJOR-INDEX (+ ,ROW-MAJOR-INDEX %OFFSET))
      (CL:IF (NOT (< ,ROW-MAJOR-INDEX (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ,ARRAY)))
              (CL:ERROR "Row-major-index out of bounds (displaced to adjustable?)")))))
```

```
(DEFMACRO %GET-ARRAY-OFFSET (ARRAY)
  `(COND
    ((OR (%ONED-ARRAY-P ,ARRAY)
         (%GENERAL-ARRAY-P ,ARRAY))
     (|fetch| (ARRAY-HEADER OFFSET) |of| ,ARRAY))
    ((%TWO-D-ARRAY-P ,ARRAY)
     0)))
```

```
(DEFMACRO %GET-BASE-ARRAY (ARRAY OFFSET)
  `(CL:DO ((%BASE-ARRAY ,ARRAY (|fetch| (ARRAY-HEADER BASE) |of| %BASE-ARRAY)))
    ((NOT (|fetch| (ARRAY-HEADER INDIRECT-P) |of| %BASE-ARRAY))
     %BASE-ARRAY)
    (SETQ ,OFFSET (+ ,OFFSET (%GET-ARRAY-OFFSET %BASE-ARRAY))))
```

```
(DEFMACRO %BIT-TYPE-P (TYPE-NUMBER)
  `(EQ ,TYPE-NUMBER %BIT-TYPE))
```

```
(DEFMACRO %CHAR-TYPE-P (TYPE-NUMBER)
  `(EQ (%TYPENUMBER-TO-TYPE ,TYPE-NUMBER)
       %CHAR-TYPE))
```

```
(DEFMACRO %CML-TYPE-TO-TYPENUMBER-EXPANDER (CML-TYPE)
```

(\*\*)

```
(LET
  ((SIMPLE-TYPES (REMOVE T (CL:MAPCAN #'(CL:LAMBDA (ENTRY)
    (CL:IF (NOT (LISTP (CAR ENTRY)))
      (LIST (CAR ENTRY))))
    %CANONICAL-CML-TYPES)))
  (COMPOUND-TYPES (CL:REMOVE-DUPLICATES (CL:MAPCAN #'(CL:LAMBDA (ENTRY)
    (CL:IF (LISTP (CAR ENTRY))
      (LIST (CAAR ENTRY))))
    %CANONICAL-CML-TYPES))))
  `(CL:IF (EQ ,CML-TYPE T)
    ,(CADR (CL:ASSOC T %CANONICAL-CML-TYPES))
    (CL:IF (LISTP ,CML-TYPE)
      (CL:ECASE (CAR ,CML-TYPE)
        (\\,@
          (CL:MAPCAR
            #'(CL:LAMBDA (TYPE)
              `(.TYPE (CL:ECASE (CADR ,CML-TYPE)
                (\\,@ (CL:MAPCAN #'(CL:LAMBDA (ENTRY)
                  (CL:IF (AND (LISTP (CAR ENTRY))
                    (EQ (CAAR ENTRY)
                      TYPE))
                    (LIST (LIST (CADAR ENTRY)
                        (CADR ENTRY))))))
                %CANONICAL-CML-TYPES))))))
          COMPOUND-TYPES)))
      (CL:ECASE ,CML-TYPE
        (\\,@ (CL:MAPCAR #'(CL:LAMBDA (TYPE)
          (CL:ASSOC TYPE %CANONICAL-CML-TYPES))
          SIMPLE-TYPES))))))
```

```
(DEFMACRO %FAT-CHAR-TYPE-P (TYPE-NUMBER)
  `(EQ ,TYPE-NUMBER %FAT-CHAR-TYPENUMBER))
```

```
(DEFMACRO %FAT-STRING-CHAR-P (OBJECT)
  `(> (CL:CHAR-CODE ,OBJECT)
    %MAXTHINCHAR))
```

```
(CL:DEFUN %GET-TYPE-TABLE-ENTRY (TYPENUMBER)
  (CADR (CL:ASSOC TYPENUMBER %ARRAY-TYPE-TABLE)))
```

```
(CL:DEFUN %LIT-SIZE-TO-SIZE (LIT-SIZE)
  (CADR (CL:ASSOC LIT-SIZE %LIT-ARRAY-SIZES)))
```

```
(CL:DEFUN %LIT-TYPE-TO-TYPE (LIT-TYPE)
  (CADR (CL:ASSOC LIT-TYPE %LIT-ARRAY-TYPES)))
```

```
(CL:DEFUN %LLARRAY-MAKE-ACCESSOR-EXPR (TYPENUMBER BASE OFFSET)
  (LET* ((ENTRY (%GET-TYPE-TABLE-ENTRY TYPENUMBER))
    (ACCESSOR (ARRAY-TABLE-ENTRY-ACCESSOR ENTRY))
    (BITS-PER-ELEMENT (ARRAY-TABLE-ENTRY-BITS-PER-ELEMENT ENTRY))
    (NEEDS-SHIFT-P (ARRAY-TABLE-ENTRY-NEEDS-SHIFT-P ENTRY)))
    `(.ACCESSOR ,BASE ,(CL:IF NEEDS-SHIFT-P
      `(LLSH ,OFFSET ,NEEDS-SHIFT-P)
      OFFSET))))
```

```
(CL:DEFUN %LLARRAY-MAKE-SETTOR-EXPR (TYPENUMBER BASE OFFSET NEWVALUE)
  (LET* ((ENTRY (%GET-TYPE-TABLE-ENTRY TYPENUMBER))
    (SETTOR (ARRAY-TABLE-ENTRY-SETTOR ENTRY))
    (BITS-PER-ELEMENT (ARRAY-TABLE-ENTRY-BITS-PER-ELEMENT ENTRY))
    (NEEDS-SHIFT-P (ARRAY-TABLE-ENTRY-NEEDS-SHIFT-P ENTRY)))
    `(.SETTOR ,BASE ,(CL:IF NEEDS-SHIFT-P
      `(LLSH ,OFFSET ,NEEDS-SHIFT-P)
      OFFSET)
      ,NEWVALUE)))
```

```
(DEFMACRO %LLARRAY-TYPED-GET (BASE TYPENUMBER OFFSET)
  `(CL:ECASE ,TYPENUMBER
    (\\,@ (CL:MAPCAR #'(CL:LAMBDA (TYPEENTRY)
      `(. (CAR TYPEENTRY)
        ,(%LLARRAY-MAKE-ACCESSOR-EXPR (CAR TYPEENTRY)
          BASE OFFSET)))
    %ARRAY-TYPE-TABLE)))
```

```
(DEFMACRO %LLARRAY-TYPED-PUT (BASE TYPENUMBER OFFSET NEWVALUE)
  `(CL:ECASE ,TYPENUMBER
```

```
(\\, @ (CL:MAPCAR #'(CL:LAMBDA (TYPEENTRY)
  \ (, (CAR TYPEENTRY)
  , (%LLARRAY-MAKE-SETTOR-EXPR (CAR TYPEENTRY)
    BASE OFFSET NEWVALUE)))
  %ARRAY-TYPE-TABLE)))
```

```
(DEFMACRO %LLARRAY-TYPEP (TYPENUMBER VALUE)
  \ (CL:ECASE ,TYPENUMBER
    (\\, @ (CL:MAPCAR #'(CL:LAMBDA (TYPEENTRY)
      \ (, (CAR TYPEENTRY)
      , (ARRAY-TABLE-ENTRY-TYPE-TEST (CADR TYPEENTRY))
      , VALUE)))
    %ARRAY-TYPE-TABLE)))
```

```
(CL:DEFUN %MAKE-ARRAY-TYPE-TABLE (LIT-TABLE TYPES SIZES)
  (CL:MAPCAN #'(CL:LAMBDA (TYPE-ENTRY)
    (LET ((LIT-TYPE (CAR TYPE-ENTRY))
          (CL:MAPCAR #'(CL:LAMBDA (SIZE-ENTRY)
            (LIST (%TYPE-SIZE-TO-TYPENUMBER LIT-TYPE (CAR SIZE-ENTRY))
                  (CADR SIZE-ENTRY)))
            (CADR TYPE-ENTRY))))
      LIT-TABLE))
```

```
(CL:DEFUN %MAKE-CML-TYPE-TABLE (ARRAY-TABLE)
  (CL:MAPCAR #'(CL:LAMBDA (TYPE-ENTRY)
    (LET ((CMLTYPE (ARRAY-TABLE-ENTRY-CML-TYPE (CADR TYPE-ENTRY)))
          (LIST CMLTYPE (CAR TYPE-ENTRY))))
      ARRAY-TABLE))
```

```
(DEFMACRO %PACK-TYPENUMBER (ELTTYPE ELTSIZE)
  \ (\\ADDBASE (LSH ,ELTTYPE 4)
    ,ELTSIZE))
```

```
(DEFMACRO %SMALLFIXP-SMALLPOSP (NUM)
  \ (\\LOLOC ,NUM))
```

```
(DEFMACRO %SMALLPOSP-SMALLFIXP (NUM)
  (LET ((SYM (GENSYM)))
    \ (LET ((,SYM ,NUM)
      (CL:IF (> ,SYM MAX.SMALLFIXP)
        (\\VAG2 |\\SmallNegHi| ,SYM)
        ,SYM)))
```

```
(DEFMACRO %THIN-CHAR-TYPE-P (TYPE-NUMBER)
  \ (EQ ,TYPE-NUMBER %THIN-CHAR-TYPENUMBER))
```

```
(DEFMACRO %THIN-STRING-CHAR-P (OBJECT)
  \ (<= (CL:CHAR-CODE ,OBJECT)
    %MAXTHINCHAR))
```

```
(CL:DEFUN %TYPE-SIZE-TO-TYPENUMBER (LIT-TYPE LIT-SIZE)
  (LET ((TYPE (CADR (CL:ASSOC LIT-TYPE %LIT-ARRAY-TYPES)))
        (SIZE (CADR (CL:ASSOC LIT-SIZE %LIT-ARRAY-SIZES))))
    (%PACK-TYPENUMBER TYPE SIZE)))
```

```
(DEFMACRO %TYPENUMBER-TO-BITS-PER-ELEMENT (TYPE-NUMBER)
  \ (CL:ECASE ,TYPE-NUMBER
    (\\, @ (CL:MAPCAR #'(CL:LAMBDA (TYPEENTRY)
      \ (, (CAR TYPEENTRY)
      , (ARRAY-TABLE-ENTRY-BITS-PER-ELEMENT (CADR TYPEENTRY))))
    %ARRAY-TYPE-TABLE)))
```

```
(DEFMACRO %TYPENUMBER-TO-CML-TYPE (TYPE-NUMBER)
  \ (CL:ECASE ,TYPE-NUMBER
    (\\, @ (CL:MAPCAR #'(CL:LAMBDA (TYPEENTRY)
      \ (, (CAR TYPEENTRY)
      , (ARRAY-TABLE-ENTRY-CML-TYPE (CADR TYPEENTRY))))
    %ARRAY-TYPE-TABLE)))
```

```
(DEFMACRO %TYPENUMBER-TO-DEFAULT-VALUE (TYPE-NUMBER)
  \ (CL:ECASE ,TYPE-NUMBER
    (\\, @ (CL:MAPCAR #'(CL:LAMBDA (TYPEENTRY)
      \ (, (CAR TYPEENTRY)
      , (ARRAY-TABLE-ENTRY-DEFAULT-VALUE (CADR TYPEENTRY))))
    %ARRAY-TYPE-TABLE)))
```

```
(DEFMACRO %TYPENUMBER-TO-GC-TYPE (TYPE-NUMBER)
  `(CL:ECASE ,TYPE-NUMBER
    (\\, @ (CL:MAPCAR #'(CL:LAMBDA (TYPEENTRY)
      \',(CAR TYPEENTRY)
      , (ARRAY-TABLE-ENTRY-GC-TYPE (CADR TYPEENTRY))))
      %ARRAY-TYPE-TABLE)))
```

```
(DEFMACRO %TYPENUMBER-TO-SIZE (TYPE-NUMBER)
  `(LOGAND ,TYPE-NUMBER 15))
```

```
(DEFMACRO %TYPENUMBER-TO-TYPE (TYPE-NUMBER)
  `(LRSH ,TYPE-NUMBER 4))
```

```
(DEFMACRO \\GETBASESMALL-FIXP (BASE OFFSET)
  `(%SMALLPOSP-SMALLFIXP (\\GETBASE ,BASE ,OFFSET)))
```

```
(DEFMACRO \\GETBASESTRING-CHAR (PTR DISP)
  `(CL:CODE-CHAR (\\GETBASE ,PTR ,DISP)))
```

```
(DEFMACRO \\GETBASEHINSTRING-CHAR (PTR DISP)
  `(CL:CODE-CHAR (\\GETBASEBYTE ,PTR ,DISP)))
```

```
(DEFMACRO \\PUTBASESMALL-FIXP (BASE OFFSET VALUE)
  ` (\\PUTBASE ,BASE ,OFFSET (%SMALLFIXP-SMALLPOSP ,VALUE)))
```

```
(DEFMACRO \\PUTBASESTRING-CHAR (PTR DISP CHAR)
  ` (\\PUTBASE ,PTR ,DISP (CL:CHAR-CODE ,CHAR)))
```

```
(DEFMACRO \\PUTBASEHINSTRING-CHAR (PTR DISP CHAR)
  ` (\\PUTBASEBYTE ,PTR ,DISP (CL:CHAR-CODE ,CHAR)))
```

;;; Describes each entry of \\ARRAY-TYPE-TABLE

```
(CL:DEFSTRUCT (ARRAY-TABLE-ENTRY (:TYPE LIST)
  (:CONSTRUCTOR NIL)
  (:COPIER NIL)
  (:PREDICATE NIL))
  CML-TYPE
  ACCESSOR
  SETTOR
  BITS-PER-ELEMENT
  GC-TYPE
  DEFAULT-VALUE
  NEEDS-SHIFT-P
  TYPE-TEST)
```

;;; These vars contain all the necessary info for typed arrays

```
(CL:DEFPARAMETER %LIT-ARRAY-SIZES '( (1BIT 0)
  (8BIT 3)
  (16BIT 4)
  (32BIT 6))
  "Size codes")
```