File created: 6-Jan-93 12:21:21 **{DSK}<python>lde>lispcore>sources>CMLARRAY.;2**

previous date: 4-Jan-93 17:46:26 {DSK}<python>lde>lispcore>sources>CMLARRAY.;1

Read Table: XCL

Package: INTERLISP

Format: XCCS

```
(RPAQQ CMLARRAYCOMS
       (;; If you change the record declarations on CMLARRAY-SUPPORT, You need to re-make this file so the INITRECORDS get filled in right.

        ;; Contains table driven macros

        (DECLARE\: DONTCOPY EVAL@COMPILE (EXPORT (FILES (SYSLOAD FROM VALUEOF DIRECTORIES)
                                                         CMLARRAY-SUPPORT)))

        ;; User entry points

        (FUNCTIONS CL:ADJUST-ARRAY CL:ADJUSTABLE-ARRAY-P CL:ARRAY-DIMENSION CL:ARRAY-DIMENSIONS
               CL:ARRAY-ELEMENT-TYPE CL:ARRAY-HAS-FILL-POINTER-P ARRAY-NEEDS-INDIRECTION-P CL:ARRAY-RANK
               CL:ARRAY-TOTAL-SIZE BIT CL:BIT-AND CL:BIT-ANDC1 CL:BIT-ANDC2 BIT-ARRAY-P CL:BIT-EQV CL:BIT-IOR
               CL:BIT-NAND CL:BIT-NOR CL:BIT-NOT CL:BIT-ORC1 CL:BIT-ORC2 CL:BIT-VECTOR-P CL:BIT-XOR CL:CHAR
               CL:ARRAYP CL:STRINGP COPY-ARRAY COPY-VECTOR DISPLACED-ARRAY-P EQUAL-DIMENSIONS-P
               EXTENDABLE-ARRAY-P FILL-ARRAY CL:FILL-POINTER FILL-VECTOR CL:MAKE-ARRAY MAKE-VECTOR
               READ-ONLY-ARRAY-P CL:SBIT CL:SCHAR SET-FILL-POINTER SIMPLE-ARRAY-P CL:SIMPLE-BIT-VECTOR-P
               CL:SIMPLE-STRING-P CL:SIMPLE-VECTOR-P STRING-ARRAY-P CL:SVREF VECTOR-LENGTH CL:VECTOR-POP
               CL:VECTOR-PUSH CL:VECTOR-PUSH-EXTEND CL:VECTORP)
        (FNS CL:AREF CL:ARRAY-IN-BOUNDS-P CL:ARRAY-ROW-MAJOR-INDEX ASET CL:VECTOR)

        ;; New CLtL array functions

        (COMS (FNS XCL:ROW-MAJOR-AREF CL::ROW-MAJOR-ASET)
              (SETFS XCL:ROW-MAJOR-AREF))

        ;; Setfs

        (SETFS CL:AREF BIT CL:CHAR CL:FILL-POINTER CL:SBIT CL:SCHAR CL:SVREF)

        ;; Optimizers

        (FUNCTIONS %AREF-EXPANDER %ASET-EXPANDER)
        (OPTIMIZERS CL:AREF ASET BIT CL:CHAR CL:SBIT CL:SCHAR CL:SVREF)

        ;; Vars etc
                                                                    ; *PRINT-ARRAY* is defined in APRINT
        (VARIABLES CL:ARRAY-RANK-LIMIT CL:ARRAY-TOTAL-SIZE-LIMIT CL:ARRAY-DIMENSION-LIMIT
               *DEFAULT-PUSH-EXTENSION-SIZE*)

        ;; Run-time support

        (FNS %ALTER-AS-DISPLACED-ARRAY %ALTER-AS-DISPLACED-TO-BASE-ARRAY %AREF0 %AREF1 %AREF2 %ARRAY-BASE
             %ARRAY-CONTENT-INITIALIZE %ARRAY-ELEMENT-INITIALIZE %ARRAY-OFFSET %ARRAY-TYPE-NUMBER %ASET0 %ASET1
             %ASET2 %CHECK-SEQUENCE-DIMENSIONS %COPY-TO-NEW-ARRAY %DO-LOGICAL-OP %EXTEND-ARRAY %FAST-COPY-BASE
             %FAT-STRING-ARRAY-P %FILL-ARRAY-FROM-SEQUENCE %FLATTEN-ARRAY %MAKE-ARRAY-WRITEABLE
             %MAKE-DISPLACED-ARRAY %MAKE-GENERAL-ARRAY %MAKE-ONED-ARRAY %MAKE-STRING-ARRAY-FAT %MAKE-TWOD-ARRAY
             %TOTAL-SIZE SHRINK-VECTOR)
                                                                    ; For Interlisp string hack
        (FNS %SET-ARRAY-OFFSET %SET-ARRAY-TYPE-NUMBER)
                                                                    ; Low level predicates
        (FNS %ONED-ARRAY-P %TWOD-ARRAY-P %GENERAL-ARRAY-P %THIN-STRING-ARRAY-P)
        (OPTIMIZERS %ONED-ARRAY-P %TWOD-ARRAY-P %GENERAL-ARRAY-P)
                                                                    ; Real record def's on cmlarray-support
        (INITRECORDS GENERAL-ARRAY ONED-ARRAY TWOD-ARRAY)
        (SYSRECORDS GENERAL-ARRAY ONED-ARRAY TWOD-ARRAY)
        (PROP DOPVAL %AREF1 %AREF2 %ASET1 %ASET2)

        ;; I/O

        (FNS %DEFPRINT-ARRAY %DEFPRINT-BITVECTOR %DEFPRINT-GENERIC-ARRAY %DEFPRINT-VECTOR %DEFPRINT-STRING
             %PRINT-ARRAY-CONTENTS)
        (P (DEFPRINT 'ONED-ARRAY '%DEFPRINT-VECTOR)
           (DEFPRINT 'TWOD-ARRAY '%DEFPRINT-ARRAY)
           (DEFPRINT 'GENERAL-ARRAY '%DEFPRINT-ARRAY))

        ;; Needed at run time. low level functions for accessing, setting, and allocating raw storage. also includes cml type to typenumber converters

        (FNS %ARRAY-READ %ARRAY-WRITE %CML-TYPE-TO-TYPENUMBER %GET-CANONICAL-CML-TYPE %GET-ENCLOSING-SIGNED-BYTE
             %GET-ENCLOSING-UNSIGNED-BYTE %MAKE-ARRAY-STORAGE %REDUCE-INTEGER %REDUCE-MOD %SLOW-ARRAY-READ
             %SLOW-ARRAY-WRITE)
        (OPTIMIZERS %ARRAY-READ %ARRAY-WRITE)

        ;; Compiler options

        (DECLARE\: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY (LOCALVARS . T))
        (PROP FILETYPE CMLARRAY)
        (DECLARE\: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS (ADDVARS (NLAMA)
                                                                              (NLAML)
                                                                              (LAMA CL:VECTOR ASET
                                                                                    CL:ARRAY-ROW-MAJOR-INDEX
                                                                                    CL:ARRAY-IN-BOUNDS-P CL:AREF)
                                                                              ))))
```

```
;; If you change the record declarations on CMLARRAY-SUPPORT, You need to re-make this file so the INITRECORDS get filled in right.
;; Contains table driven macros

(DECLARE\: DONTCOPY EVAL@COMPILE

;; FOLLOWING DEFINITIONS EXPORTED

(FILESLOAD (SYSLOAD FROM VALUEOF DIRECTORIES)
       CMLARRAY-SUPPORT)
)

;; END EXPORTED DEFINITIONS
;; User entry points


(CL:DEFUN CL:ADJUST-ARRAY (ADJUSTABLE-ARRAY DIMENSIONS &KEY (ELEMENT-TYPE NIL ELEMENT-TYPE-P)
                                        (INITIAL-ELEMENT NIL INITIAL-ELEMENT-P)
                                        (INITIAL-CONTENTS NIL INITIAL-CONTENTS-P)
                                        (DISPLACED-TO NIL DISPLACED-TO-P)
                                        (DISPLACED-TO-BASE NIL DISPLACED-TO-BASE-P)
                                        (DISPLACED-INDEX-OFFSET 0 DISPLACED-INDEX-OFFSET-P)
                                        (FILL-POINTER NIL FILL-POINTER-P)
                                        FATP)

   ;; Do something wonderfull

   (CL:IF (NOT (EXTENDABLE-ARRAY-P ADJUSTABLE-ARRAY))
          (CL:ERROR "Not an adjustable or extendable array: ~S" ADJUSTABLE-ARRAY))
   (CL:IF (NOT (CL:LISTP DIMENSIONS))
       (SETQ DIMENSIONS (LIST DIMENSIONS)))
   (CL:IF (CL:DOLIST (DIM DIMENSIONS NIL)
              (CL:IF (OR (< DIM 0)
                         (>= DIM CL:ARRAY-DIMENSION-LIMIT))
                     (RETURN T)))
          (CL:ERROR "Dimensions out of bounds ~S" DIMENSIONS))
   (LET ((ADJUSTABLE-ARRAY-ELEMENT-TYPE (CL:ARRAY-ELEMENT-TYPE ADJUSTABLE-ARRAY))
         (NELTS (%TOTAL-SIZE DIMENSIONS))
         (RANK (LENGTH DIMENSIONS))
         (EXTENDABLE-P (NOT (CL:ADJUSTABLE-ARRAY-P ADJUSTABLE-ARRAY))))

        ;; Consistency checks

        (CL:IF (>= RANK CL:ARRAY-RANK-LIMIT)
               (CL:ERROR "Too many dimensions: ~A" RANK))
        (CL:IF (>= NELTS CL:ARRAY-TOTAL-SIZE-LIMIT)
               (CL:ERROR "Too many elements: ~A" NELTS))
        (CL:IF (NOT (EQ RANK (CL:ARRAY-RANK ADJUSTABLE-ARRAY)))
               (CL:ERROR "Rank mismatch: ~S" DIMENSIONS))
        (CL:IF ELEMENT-TYPE-P
            (CL:IF (NOT (EQUAL ELEMENT-TYPE ADJUSTABLE-ARRAY-ELEMENT-TYPE))
                   (CL:ERROR "ADJUSTABLE-ARRAY not of specified element-type: ~A" ELEMENT-TYPE))
            (SETQ ELEMENT-TYPE ADJUSTABLE-ARRAY-ELEMENT-TYPE))
        (CL:IF (AND FILL-POINTER-P (NULL FILL-POINTER)
                    (CL:ARRAY-HAS-FILL-POINTER-P ADJUSTABLE-ARRAY))
               (CL:ERROR "ADJUSTABLE-ARRAY has fill pointer"))
        (CL:IF (OR (AND DISPLACED-TO-P (OR INITIAL-ELEMENT-P INITIAL-CONTENTS-P DISPLACED-TO-BASE-P))
                   (AND DISPLACED-TO-BASE-P (OR INITIAL-ELEMENT-P INITIAL-CONTENTS-P DISPLACED-TO-P))
                   (AND FILL-POINTER-P FILL-POINTER (NOT (CL:ARRAY-HAS-FILL-POINTER-P ADJUSTABLE-ARRAY)))
                   (AND DISPLACED-INDEX-OFFSET-P (NOT (OR DISPLACED-TO-P DISPLACED-TO-BASE-P)))
                   (AND INITIAL-ELEMENT-P INITIAL-CONTENTS-P))
               (CL:ERROR "Inconsistent options to adjust-array"))
        (CL:IF DISPLACED-TO-P
            (COND
               ((NOT (%ARRAYP DISPLACED-TO))
                (CL:ERROR "Not displaced to an array: ~S" DISPLACED-TO))
               ((NOT (EQUAL ADJUSTABLE-ARRAY-ELEMENT-TYPE (CL:ARRAY-ELEMENT-TYPE DISPLACED-TO)))
                (CL:ERROR "Not displaced to an array of the same element-type:"))
               ((> (+ DISPLACED-INDEX-OFFSET NELTS)
                   (CL:ARRAY-TOTAL-SIZE DISPLACED-TO))
                (CL:ERROR "More elements than displaced-to array"))))
        (CL:IF FILL-POINTER
            (COND
               ((EQ FILL-POINTER T)
                (SETQ FILL-POINTER NELTS))
               ((NOT (<= 0 FILL-POINTER NELTS))
                (CL:ERROR "Fill pointer out of bounds: ~A" FILL-POINTER)))
            (CL:IF (CL:ARRAY-HAS-FILL-POINTER-P ADJUSTABLE-ARRAY)
                (SETQ FILL-POINTER (MIN (CL:FILL-POINTER ADJUSTABLE-ARRAY)
                                        NELTS))))
        (CL:IF EXTENDABLE-P
            (COND
               ((OR DISPLACED-TO-P DISPLACED-TO-BASE-P)
                (CL:ERROR "Cannot adjust an extendable array to be displaced"))
               ((< NELTS (CL:ARRAY-TOTAL-SIZE ADJUSTABLE-ARRAY))
                (CL:ERROR "Cannot extend an extendable array to have fewer elements"))))

        ;; Specs ready, do the surgury
        (COND
           (DISPLACED-TO-P (%ALTER-AS-DISPLACED-ARRAY ADJUSTABLE-ARRAY DIMENSIONS DISPLACED-TO
                                  DISPLACED-INDEX-OFFSET FILL-POINTER))
           (DISPLACED-TO-BASE-P (%ALTER-AS-DISPLACED-TO-BASE-ARRAY ADJUSTABLE-ARRAY DIMENSIONS ELEMENT-TYPE
                                       DISPLACED-TO-BASE DISPLACED-INDEX-OFFSET FILL-POINTER FATP))
           (T (CL:IF (EQUAL (CL:ARRAY-DIMENSIONS ADJUSTABLE-ARRAY)
                            DIMENSIONS)
                  (CL:IF FILL-POINTER (SET-FILL-POINTER ADJUSTABLE-ARRAY FILL-POINTER))
                  (LET ((NEW-ARRAY (CL:MAKE-ARRAY DIMENSIONS :ELEMENT-TYPE ELEMENT-TYPE :FATP (
                                                                                     %FAT-STRING-ARRAY-P
                                                                                         ADJUSTABLE-ARRAY
                                                                                         ))))
                       (COND
                          (INITIAL-CONTENTS-P (%ARRAY-CONTENT-INITIALIZE NEW-ARRAY INITIAL-CONTENTS))
                          (T (CL:IF INITIAL-ELEMENT-P (%ARRAY-ELEMENT-INITIALIZE NEW-ARRAY INITIAL-ELEMENT))
                             (%COPY-TO-NEW-ARRAY (CL:ARRAY-DIMENSIONS ADJUSTABLE-ARRAY)
                                    (%FLATTEN-ARRAY ADJUSTABLE-ARRAY)
                                    0 DIMENSIONS (%FLATTEN-ARRAY NEW-ARRAY)
                                    0)))
                       (%EXTEND-ARRAY ADJUSTABLE-ARRAY NEW-ARRAY DIMENSIONS FILL-POINTER)))))
        ;; Return the adjusted array
        ADJUSTABLE-ARRAY))


(CL:DEFUN CL:ADJUSTABLE-ARRAY-P (ARRAY)
   (CL:IF (%ARRAYP ARRAY)
       (|fetch| (ARRAY-HEADER ADJUSTABLE-P) |of| ARRAY)
       (CL:ERROR "Not an array: ~S" ARRAY)))


(CL:DEFUN CL:ARRAY-DIMENSION (ARRAY DIMENSION)
   (COND
      ((%ONED-ARRAY-P ARRAY)
       (CL:IF (EQ 0 DIMENSION)
           (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY)
           (CL:ERROR "Dimension out of bounds: ~A" DIMENSION)))
      ((%TWOD-ARRAY-P ARRAY)
       (CASE DIMENSION
           (0 (|ffetch| (TWOD-ARRAY BOUND0) |of| ARRAY))
           (1 (|ffetch| (TWOD-ARRAY BOUND1) |of| ARRAY))
           (T (CL:ERROR "Dimension out of bounds: ~A" DIMENSION))))
      ((%GENERAL-ARRAY-P ARRAY)
       (LET* ((DIMS (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY))
              (RANK (LENGTH DIMS)))
             (CL:IF (NOT (< -1 DIMENSION RANK))
                    (CL:ERROR "Dimension out of bounds: ~A" DIMENSION))
             (CL:IF (EQ RANK 1)
                 (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY)
                 (CL:NTH DIMENSION DIMS))))
      (T (CL:ERROR "Not an array: ~S" ARRAY))))


(CL:DEFUN CL:ARRAY-DIMENSIONS (ARRAY)
   (COND
      ((%ONED-ARRAY-P ARRAY)
       (LIST (|ffetch| (ONED-ARRAY TOTAL-SIZE) |of| ARRAY)))
      ((%TWOD-ARRAY-P ARRAY)
       (LIST (|ffetch| (TWOD-ARRAY BOUND0) |of| ARRAY)
             (|ffetch| (TWOD-ARRAY BOUND1) |of| ARRAY)))
      ((%GENERAL-ARRAY-P ARRAY)
       (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY))
      (T (CL:ERROR "Not an array: ~S" ARRAY))))


(CL:DEFUN CL:ARRAY-ELEMENT-TYPE (ARRAY)
   (CL:IF (%ARRAYP ARRAY)
       (%TYPENUMBER-TO-CML-TYPE (%ARRAY-TYPE-NUMBER ARRAY))
       (CL:ERROR "Not an array: ~S" ARRAY)))


(CL:DEFUN CL:ARRAY-HAS-FILL-POINTER-P (ARRAY)
   (CL:IF (%ARRAYP ARRAY)
       (|fetch| (ARRAY-HEADER FILL-POINTER-P) |of| ARRAY)
       (CL:ERROR "Not an array: ~S" ARRAY)))


(CL:DEFUN ARRAY-NEEDS-INDIRECTION-P (ARRAY)
   (COND
      ((OR (%ONED-ARRAY-P ARRAY)
           (%TWOD-ARRAY-P ARRAY))
       NIL)
      ((%GENERAL-ARRAY-P ARRAY)
       (|fetch| (ARRAY-HEADER INDIRECT-P) |of| ARRAY))
      (T (CL:ERROR "Not an array: ~S" ARRAY))))
```

```
(CL:DEFUN CL:ARRAY-RANK (ARRAY)
   (COND
      ((%ONED-ARRAY-P ARRAY)
       1)
      ((%TWOD-ARRAY-P ARRAY)
       2)
      ((%GENERAL-ARRAY-P ARRAY)
       (LENGTH (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY)))
      (T (CL:ERROR "Not an array: ~S" ARRAY))))


(CL:DEFUN CL:ARRAY-TOTAL-SIZE (ARRAY)
   (CL:IF (%ARRAYP ARRAY)
       (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY)
       (CL:ERROR "Not an array: ~S" ARRAY)))


(CL:DEFUN BIT (BIT-ARRAY &REST INDICES)
   (CL:ASSERT (TYPEP BIT-ARRAY '(CL:ARRAY BIT))
          (BIT-ARRAY)
          "Not a bit-array: ~S" BIT-ARRAY)
   (CL:APPLY #'CL:AREF BIT-ARRAY INDICES))


(CL:DEFUN CL:BIT-AND (BIT-ARRAY1 BIT-ARRAY2 &OPTIONAL BIT-RESULT)
   (%EXPAND-BIT-OP AND BIT-ARRAY1 BIT-ARRAY2 BIT-RESULT))


(CL:DEFUN CL:BIT-ANDC1 (BIT-ARRAY1 BIT-ARRAY2 &OPTIONAL BIT-RESULT)
   (%EXPAND-BIT-OP ANDC1 BIT-ARRAY1 BIT-ARRAY2 BIT-RESULT))


(CL:DEFUN CL:BIT-ANDC2 (BIT-ARRAY1 BIT-ARRAY2 &OPTIONAL BIT-RESULT)
   (%EXPAND-BIT-OP ANDC2 BIT-ARRAY1 BIT-ARRAY2 BIT-RESULT))


(CL:DEFUN BIT-ARRAY-P (ARRAY)
   (AND (%ARRAYP ARRAY)
        (|fetch| (ARRAY-HEADER BIT-P) |of| ARRAY)))


(CL:DEFUN CL:BIT-EQV (BIT-ARRAY1 BIT-ARRAY2 &OPTIONAL BIT-RESULT)
   (%EXPAND-BIT-OP EQV BIT-ARRAY1 BIT-ARRAY2 BIT-RESULT))


(CL:DEFUN CL:BIT-IOR (BIT-ARRAY1 BIT-ARRAY2 &OPTIONAL BIT-RESULT)
   (%EXPAND-BIT-OP IOR BIT-ARRAY1 BIT-ARRAY2 BIT-RESULT))


(CL:DEFUN CL:BIT-NAND (BIT-ARRAY1 BIT-ARRAY2 &OPTIONAL BIT-RESULT)
   (%EXPAND-BIT-OP NAND BIT-ARRAY1 BIT-ARRAY2 BIT-RESULT))


(CL:DEFUN CL:BIT-NOR (BIT-ARRAY1 BIT-ARRAY2 &OPTIONAL BIT-RESULT)
   (%EXPAND-BIT-OP NOR BIT-ARRAY1 BIT-ARRAY2 BIT-RESULT))


(CL:DEFUN CL:BIT-NOT (BIT-ARRAY &OPTIONAL RESULT-BIT-ARRAY)
   (CL:IF (NOT (BIT-ARRAY-P BIT-ARRAY))
          (CL:ERROR "BIT-ARRAY not a bit array"))
   (COND
      ((NULL RESULT-BIT-ARRAY)
       (SETQ RESULT-BIT-ARRAY (CL:MAKE-ARRAY (CL:ARRAY-DIMENSIONS BIT-ARRAY)
                                     :ELEMENT-TYPE
                                     'BIT)))
      ((EQ RESULT-BIT-ARRAY T)
       (SETQ RESULT-BIT-ARRAY BIT-ARRAY))
      ((NOT (AND (BIT-ARRAY-P RESULT-BIT-ARRAY)
                 (EQUAL-DIMENSIONS-P BIT-ARRAY RESULT-BIT-ARRAY)))
       (CL:ERROR "Illegal result array")))
   (%DO-LOGICAL-OP 'NOT BIT-ARRAY RESULT-BIT-ARRAY)
   RESULT-BIT-ARRAY)


(CL:DEFUN CL:BIT-ORC1 (BIT-ARRAY1 BIT-ARRAY2 &OPTIONAL BIT-RESULT)
   (%EXPAND-BIT-OP ORC1 BIT-ARRAY1 BIT-ARRAY2 BIT-RESULT))


(CL:DEFUN CL:BIT-ORC2 (BIT-ARRAY1 BIT-ARRAY2 &OPTIONAL BIT-RESULT)
   (%EXPAND-BIT-OP ORC2 BIT-ARRAY1 BIT-ARRAY2 BIT-RESULT))


(CL:DEFUN CL:BIT-VECTOR-P (VECTOR)
   (AND (%VECTORP VECTOR)
        (|fetch| (ARRAY-HEADER BIT-P) |of| VECTOR)))
```

```
(CL:DEFUN CL:BIT-XOR (BIT-ARRAY1 BIT-ARRAY2 &OPTIONAL BIT-RESULT)
   (%EXPAND-BIT-OP XOR BIT-ARRAY1 BIT-ARRAY2 BIT-RESULT))


(CL:DEFUN CL:CHAR (STRING INDEX)
   (CL:ASSERT (TYPEP STRING 'STRING)
          (STRING)
          "Not a string: ~S" STRING)
   (CL:AREF STRING INDEX))


(CL:DEFUN CL:ARRAYP (ARRAY)
   (%ARRAYP ARRAY))


(CL:DEFUN CL:STRINGP (STRING)
   (%STRINGP STRING))


(CL:DEFUN COPY-ARRAY (FROM-ARRAY &OPTIONAL TO-ARRAY)
   (CL:IF (NOT (%ARRAYP FROM-ARRAY))
          (CL:ERROR "Not an array: ~S" FROM-ARRAY))
   (COND
      ((NULL TO-ARRAY)
       (SETQ TO-ARRAY (CL:MAKE-ARRAY (CL:ARRAY-DIMENSIONS FROM-ARRAY)
                             :ELEMENT-TYPE
                             (CL:ARRAY-ELEMENT-TYPE FROM-ARRAY)
                             :FATP
                             (%FAT-STRING-ARRAY-P FROM-ARRAY))))
      ((NOT (EQUAL-DIMENSIONS-P FROM-ARRAY TO-ARRAY))
       (CL:ERROR "Dimensionality mismatch")))
   (CL:IF (|fetch| (ARRAY-HEADER READ-ONLY-P) |of| TO-ARRAY)
          (%MAKE-ARRAY-WRITEABLE TO-ARRAY))
   (LET ((FROM-TYPE-NUMBER (%ARRAY-TYPE-NUMBER FROM-ARRAY))
         (TO-TYPE-NUMBER (%ARRAY-TYPE-NUMBER TO-ARRAY)))
        (CL:WHEN (AND (%FAT-CHAR-TYPE-P FROM-TYPE-NUMBER)
                      (%THIN-CHAR-TYPE-P TO-TYPE-NUMBER))
            (%MAKE-STRING-ARRAY-FAT TO-ARRAY)
            (SETQ TO-TYPE-NUMBER (%ARRAY-TYPE-NUMBER TO-ARRAY)))
        (%FAST-COPY-BASE (%ARRAY-BASE FROM-ARRAY)
               (%ARRAY-OFFSET FROM-ARRAY)
               FROM-TYPE-NUMBER
               (%ARRAY-BASE TO-ARRAY)
               (%ARRAY-OFFSET TO-ARRAY)
               TO-TYPE-NUMBER
               (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| FROM-ARRAY))
        TO-ARRAY))


(CL:DEFUN COPY-VECTOR (FROM-VECTOR TO-VECTOR &KEY (START1 0)
                                        END1
                                        (START2 0)
                                        END2)
   (LET ((FROM-LENGTH (VECTOR-LENGTH FROM-VECTOR))
         (TO-LENGTH (VECTOR-LENGTH TO-VECTOR)))
        (CL:IF (NULL END1)
               (SETQ END1 FROM-LENGTH))
        (CL:IF (NULL END2)
               (SETQ END2 TO-LENGTH))
        (CL:IF (NOT (<= 0 START1 END1 FROM-LENGTH))
               (CL:ERROR "Bad subsequence for FROM-VECTOR"))
        (CL:IF (NOT (<= 0 START2 END2 TO-LENGTH))
               (CL:ERROR "Bad subsequence for TO-VECTOR"))
        (CL:IF (|fetch| (ARRAY-HEADER READ-ONLY-P) |of| TO-VECTOR)
               (%MAKE-ARRAY-WRITEABLE TO-VECTOR))
        (LET ((SUBLEN1 (- END1 START1))
              (SUBLEN2 (- END2 START2))
              (FROM-TYPE-NUMBER (%ARRAY-TYPE-NUMBER FROM-VECTOR))
              (TO-TYPE-NUMBER (%ARRAY-TYPE-NUMBER TO-VECTOR)))
             (CL:WHEN (AND (%FAT-CHAR-TYPE-P FROM-TYPE-NUMBER)
                           (%THIN-CHAR-TYPE-P TO-TYPE-NUMBER))
                 (%MAKE-STRING-ARRAY-FAT TO-VECTOR)
                 (SETQ TO-TYPE-NUMBER (%ARRAY-TYPE-NUMBER TO-VECTOR)))
             (%FAST-COPY-BASE (%ARRAY-BASE FROM-VECTOR)
                    (+ START1 (%ARRAY-OFFSET FROM-VECTOR))
                    FROM-TYPE-NUMBER
                    (%ARRAY-BASE TO-VECTOR)
                    (+ START2 (%ARRAY-OFFSET TO-VECTOR))
                    TO-TYPE-NUMBER
                    (MIN SUBLEN1 SUBLEN2))
             TO-VECTOR)))


(CL:DEFUN DISPLACED-ARRAY-P (ARRAY)
   (CL:IF (%ARRAYP ARRAY)
       (|fetch| (ARRAY-HEADER DISPLACED-P) |of| ARRAY)
       (CL:ERROR "Not an array: ~S" ARRAY)))
```

```
(CL:DEFUN EQUAL-DIMENSIONS-P (ARRAY-1 ARRAY-2)
   (COND
      ((%ONED-ARRAY-P ARRAY-1)
       (COND
          ((%ONED-ARRAY-P ARRAY-2)
           (EQ (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY-1)
               (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY-2)))
          ((%TWOD-ARRAY-P ARRAY-2)
           NIL)
          ((%GENERAL-ARRAY-P ARRAY-2)
           (AND (EQ 1 (LENGTH (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY-2)))
                (EQ (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY-1)
                    (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY-2))))
          (T NIL)))
      ((%TWOD-ARRAY-P ARRAY-1)
       (COND
          ((%ONED-ARRAY-P ARRAY-2)
           NIL)
          ((%TWOD-ARRAY-P ARRAY-2)
           (AND (EQ (|ffetch| (TWOD-ARRAY BOUND0) |of| ARRAY-1)
                    (|ffetch| (TWOD-ARRAY BOUND0) |of| ARRAY-2))
                (EQ (|ffetch| (TWOD-ARRAY BOUND1) |of| ARRAY-1)
                    (|ffetch| (TWOD-ARRAY BOUND1) |of| ARRAY-2))))
          ((%GENERAL-ARRAY-P ARRAY-2)
           (LET ((DIMS (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY-2)))
                (AND (EQ 2 (LENGTH DIMS))
                     (AND (EQ (|ffetch| (TWOD-ARRAY BOUND0) |of| ARRAY-1)
                              (CAR DIMS))
                          (EQ (|ffetch| (TWOD-ARRAY BOUND1) |of| ARRAY-1)
                              (CADR DIMS))))))
          (T NIL)))
      ((%GENERAL-ARRAY-P ARRAY-1)
       (LET ((DIMS (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY-1)))
            (COND
               ((%ONED-ARRAY-P ARRAY-2)
                (AND (EQ 1 (LENGTH DIMS))
                     (EQ (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY-1)
                         (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY-2))))
               ((%TWOD-ARRAY-P ARRAY-2)
                (AND (EQ 2 (LENGTH DIMS))
                     (AND (EQ (CAR DIMS)
                              (|ffetch| (TWOD-ARRAY BOUND0) |of| ARRAY-2))
                          (EQ (CADR DIMS)
                              (|ffetch| (TWOD-ARRAY BOUND1) |of| ARRAY-2)))))
               ((%GENERAL-ARRAY-P ARRAY-2)
                (EQUAL DIMS (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY-2)))
               (T NIL))))
      (T NIL)))


(CL:DEFUN EXTENDABLE-ARRAY-P (ARRAY)

          (* *)

   (COND
      ((%ARRAYP ARRAY)
       (|fetch| (ARRAY-HEADER EXTENDABLE-P) |of| ARRAY))
      ((STRINGP ARRAY)
       NIL)
      (T (CL:ERROR "Not an array ~S" ARRAY))))


(CL:DEFUN FILL-ARRAY (ARRAY VALUE)
   (CL:IF (NOT (%ARRAYP ARRAY))
          (CL:ERROR "Not an array: ~S" ARRAY))
   (LET ((TOTAL-SIZE (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY))
         (TYPE-NUMBER (%ARRAY-TYPE-NUMBER ARRAY)))
        (CL:IF (|fetch| (ARRAY-HEADER READ-ONLY-P) |of| ARRAY)
               (%MAKE-ARRAY-WRITEABLE ARRAY))
        (CL:WHEN (> TOTAL-SIZE 0)
            (CL:WHEN (AND (%THIN-CHAR-TYPE-P TYPE-NUMBER)
                          (%FAT-STRING-CHAR-P VALUE))
                (%MAKE-STRING-ARRAY-FAT ARRAY)
                (SETQ TYPE-NUMBER (%ARRAY-TYPE-NUMBER ARRAY)))
            (CL:IF (NOT (%LLARRAY-TYPEP TYPE-NUMBER VALUE))
                   (CL:ERROR "Value of incorrect type for this array: ~S" VALUE))
            (LET ((BASE (%ARRAY-BASE ARRAY))
                  (OFFSET (%ARRAY-OFFSET ARRAY)))                   ; Start things off
                 (%ARRAY-WRITE VALUE BASE TYPE-NUMBER OFFSET)       ; An overlapping blt
                 (%FAST-COPY-BASE BASE OFFSET TYPE-NUMBER BASE (CL:1+ OFFSET)
                        TYPE-NUMBER
                        (CL:1- TOTAL-SIZE))))
        ARRAY))
```

```
(CL:DEFUN CL:FILL-POINTER (VECTOR)
   (COND
      ((AND (OR (%ONED-ARRAY-P VECTOR)
                (%GENERAL-ARRAY-P VECTOR))
            (|fetch| (ARRAY-HEADER FILL-POINTER-P) |of| VECTOR))
       (|fetch| (ARRAY-HEADER FILL-POINTER) |of| VECTOR))
      ((%VECTORP VECTOR)
       (CL:ERROR "vector has no fill pointer"))
      (T (CL:ERROR "Not a vector: ~S" VECTOR))))


(CL:DEFUN FILL-VECTOR (VECTOR VALUE &KEY (START 0)
                              END)
   (CL:IF (NOT (%VECTORP VECTOR))
          (CL:ERROR "Not a vector: ~S" VECTOR))
   (LET ((TOTAL-SIZE (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| VECTOR)))
        (CL:IF (NULL END)
               (SETQ END TOTAL-SIZE))
        (CL:IF (NOT (<= START END TOTAL-SIZE))
               (CL:ERROR "Invalid subsequence" END))
        (LET ((CNT (- END START))
              (TYPE-NUMBER (%ARRAY-TYPE-NUMBER VECTOR)))
             (CL:IF (|fetch| (ARRAY-HEADER READ-ONLY-P) |of| VECTOR)
                    (%MAKE-ARRAY-WRITEABLE VECTOR))
             (CL:WHEN (> CNT 0)
                 (CL:WHEN (AND (%THIN-CHAR-TYPE-P TYPE-NUMBER)
                               (%FAT-STRING-CHAR-P VALUE))
                     (%MAKE-STRING-ARRAY-FAT VECTOR)
                     (SETQ TYPE-NUMBER (%ARRAY-TYPE-NUMBER VECTOR)))
                 (CL:IF (NOT (%LLARRAY-TYPEP TYPE-NUMBER VALUE))
                        (CL:ERROR "Value of incorrect type for this array: ~S" VALUE))
                 (LET ((BASE (%ARRAY-BASE VECTOR))
                       (OFFSET (+ START (%ARRAY-OFFSET VECTOR))))
                                                             ; Start things off
                      (%ARRAY-WRITE VALUE BASE TYPE-NUMBER OFFSET)
                                                             ; An overlapping blt
                      (%FAST-COPY-BASE BASE OFFSET TYPE-NUMBER BASE (CL:1+ OFFSET)
                             TYPE-NUMBER
                             (CL:1- CNT))))
             VECTOR)))


(CL:DEFUN CL:MAKE-ARRAY (DIMENSIONS &KEY (ELEMENT-TYPE T)
                                    (INITIAL-ELEMENT NIL INITIAL-ELEMENT-P)
                                    (INITIAL-CONTENTS NIL INITIAL-CONTENTS-P)
                                    (DISPLACED-TO NIL DISPLACED-TO-P)
                                    (DISPLACED-TO-BASE NIL DISPLACED-TO-BASE-P)
                                    (DISPLACED-INDEX-OFFSET 0 DISPLACED-INDEX-OFFSET-P)
                                    FILL-POINTER ADJUSTABLE EXTENDABLE FATP READ-ONLY-P)

   ;; String are by default thin unless FATP is T. DISPLACED-TO-BASE indicates displacement to a raw storage block. READ-ONLY-P indicates a
   ;; read only array

   (CL:IF (NOT (CL:LISTP DIMENSIONS))
       (SETQ DIMENSIONS (LIST DIMENSIONS)))
   (CL:IF (CL:DOLIST (DIM DIMENSIONS NIL)
              (CL:IF (OR (< DIM 0)
                         (>= DIM CL:ARRAY-DIMENSION-LIMIT))
                     (RETURN T)))
          (CL:ERROR "Dimensions out of bounds: ~S" DIMENSIONS))
   (LET ((RANK (LENGTH DIMENSIONS))
         (NELTS (%TOTAL-SIZE DIMENSIONS))
         ARRAY)

        ;; Consistency checks

        (CL:IF (>= RANK CL:ARRAY-RANK-LIMIT)
               (CL:ERROR "Too many dimensions: ~A" RANK))
        (CL:IF (>= NELTS CL:ARRAY-TOTAL-SIZE-LIMIT)
               (CL:ERROR "Too many elements: ~A" NELTS))
        (CL:IF (OR (AND DISPLACED-TO-P (OR INITIAL-ELEMENT-P INITIAL-CONTENTS-P DISPLACED-TO-BASE-P))
                   (AND DISPLACED-TO-BASE-P (OR INITIAL-ELEMENT-P INITIAL-CONTENTS-P DISPLACED-TO-P))
                   (AND FILL-POINTER (NOT (EQ RANK 1)))
                   (AND DISPLACED-INDEX-OFFSET-P (NOT (OR DISPLACED-TO-P DISPLACED-TO-BASE-P)))
                   (AND INITIAL-ELEMENT-P INITIAL-CONTENTS-P)
                   (AND ADJUSTABLE EXTENDABLE)
                   (AND READ-ONLY-P (OR EXTENDABLE ADJUSTABLE)))
               (CL:ERROR "Inconsistent options to make-array"))
        (CL:IF DISPLACED-TO-P
            (COND
               ((NOT (%ARRAYP DISPLACED-TO))
                (CL:ERROR "Not displaced to an array: ~s" DISPLACED-TO))
               ((NOT (EQUAL (%GET-CANONICAL-CML-TYPE ELEMENT-TYPE)
                            (CL:ARRAY-ELEMENT-TYPE DISPLACED-TO)))
                (CL:ERROR "Not displaced to an array of the same element-type"))
               ((> (+ DISPLACED-INDEX-OFFSET NELTS)
                   (CL:ARRAY-TOTAL-SIZE DISPLACED-TO))
                (CL:ERROR "Displaced array out of bounds"))))
        (CL:IF FILL-POINTER
            (COND
               ((EQ FILL-POINTER T)
                (SETQ FILL-POINTER NELTS))
               ((NOT (AND (>= FILL-POINTER 0)
                          (<= FILL-POINTER NELTS)))
                (CL:ERROR "Fill pointer out of bounds ~A" FILL-POINTER))))
```

;; Specs ready, make the array by case

```
        (SETQ ARRAY (COND
                       (DISPLACED-TO-P (%MAKE-DISPLACED-ARRAY NELTS DIMENSIONS ELEMENT-TYPE DISPLACED-TO
                                              DISPLACED-INDEX-OFFSET FILL-POINTER READ-ONLY-P ADJUSTABLE
                                              EXTENDABLE))
                       (DISPLACED-TO-BASE (CL:IF (OR (> RANK 1)
                                                     ADJUSTABLE)
                                              (%MAKE-GENERAL-ARRAY NELTS DIMENSIONS ELEMENT-TYPE FILL-POINTER
                                                     FATP READ-ONLY-P ADJUSTABLE EXTENDABLE DISPLACED-TO-BASE
                                                     DISPLACED-INDEX-OFFSET)
                                              (%MAKE-ONED-ARRAY NELTS ELEMENT-TYPE FILL-POINTER FATP
                                                     READ-ONLY-P EXTENDABLE DISPLACED-TO-BASE
                                                     DISPLACED-INDEX-OFFSET)))
                       ((AND (EQ RANK 1)
                             (NOT ADJUSTABLE))
                        (%MAKE-ONED-ARRAY NELTS ELEMENT-TYPE FILL-POINTER FATP READ-ONLY-P EXTENDABLE))
                       ((AND (EQ RANK 2)
                             (NOT ADJUSTABLE))
                        (%MAKE-TWOD-ARRAY NELTS DIMENSIONS ELEMENT-TYPE FATP READ-ONLY-P EXTENDABLE))
                       (T (%MAKE-GENERAL-ARRAY NELTS DIMENSIONS ELEMENT-TYPE FILL-POINTER FATP READ-ONLY-P
                                 ADJUSTABLE EXTENDABLE))))
```

;; Initialize the storage

```
        (COND
           (INITIAL-CONTENTS-P (%ARRAY-CONTENT-INITIALIZE ARRAY INITIAL-CONTENTS))
           (INITIAL-ELEMENT-P (%ARRAY-ELEMENT-INITIALIZE ARRAY INITIAL-ELEMENT)))
```

;; Return the array

```
        ARRAY))


(CL:DEFUN MAKE-VECTOR (SIZE &KEY (ELEMENT-TYPE T)
                                 (INITIAL-ELEMENT NIL INITIAL-ELEMENT-P)
                                 FATP)
   (CL:IF (OR (< SIZE 0)
              (>= SIZE CL:ARRAY-TOTAL-SIZE-LIMIT))
          (CL:ERROR "Size out of bounds: ~s" SIZE))
   (LET ((VECTOR (%MAKE-ONED-ARRAY SIZE ELEMENT-TYPE NIL FATP)))
        (CL:IF INITIAL-ELEMENT-P (FILL-ARRAY VECTOR INITIAL-ELEMENT))
        VECTOR))


(CL:DEFUN READ-ONLY-ARRAY-P (ARRAY)
   (CL:IF (%ARRAYP ARRAY)
       (|fetch| (ARRAY-HEADER READ-ONLY-P) |of| ARRAY)
       (CL:ERROR "Not an array: ~S" ARRAY)))


(CL:DEFUN CL:SBIT (SIMPLE-BIT-ARRAY &REST INDICES)
   (CL:ASSERT (TYPEP SIMPLE-BIT-ARRAY '(CL:SIMPLE-ARRAY BIT))
          (SIMPLE-BIT-ARRAY)
          "Not a bit-array: ~S" SIMPLE-BIT-ARRAY)
   (CL:APPLY #'CL:AREF SIMPLE-BIT-ARRAY INDICES))


(CL:DEFUN CL:SCHAR (SIMPLE-STRING INDEX)
   (CL:ASSERT (TYPEP SIMPLE-STRING 'CL:SIMPLE-STRING)
          (SIMPLE-STRING)
          "Not a simple-string: ~S" SIMPLE-STRING)
   (CL:AREF SIMPLE-STRING INDEX))


(CL:DEFUN SET-FILL-POINTER (VECTOR NEWVALUE)
   (COND
      ((AND (OR (%ONED-ARRAY-P VECTOR)
                (%GENERAL-ARRAY-P VECTOR))
            (|fetch| (ARRAY-HEADER FILL-POINTER-P) |of| VECTOR))
       (CL:IF (NOT (<= 0 NEWVALUE (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| VECTOR)))
              (CL:ERROR "Fill pointer out of bounds: ~S" NEWVALUE))
       (|replace| (ARRAY-HEADER FILL-POINTER) |of| VECTOR |with| NEWVALUE)
       NEWVALUE)
      ((%VECTORP VECTOR)
       (CL:ERROR "Vector has no fill pointer"))
      (T (CL:ERROR "Not a vector: ~S" VECTOR))))


(CL:DEFUN SIMPLE-ARRAY-P (ARRAY)
   (%SIMPLE-ARRAY-P ARRAY))
```

```
(CL:DEFUN CL:SIMPLE-BIT-VECTOR-P (VECTOR)
   (AND (%ONED-ARRAY-P VECTOR)
        (|fetch| (ARRAY-HEADER SIMPLE-P) |of| VECTOR)
        (|fetch| (ARRAY-HEADER BIT-P) |of| VECTOR)))


(CL:DEFUN CL:SIMPLE-STRING-P (STRING)
   (%SIMPLE-STRING-P STRING))


(CL:DEFUN CL:SIMPLE-VECTOR-P (VECTOR)
   (AND (%ONED-ARRAY-P VECTOR)
        (|fetch| (ARRAY-HEADER SIMPLE-P) |of| VECTOR)
        (EQ (CL:ARRAY-ELEMENT-TYPE VECTOR)
            T)))


(CL:DEFUN STRING-ARRAY-P (ARRAY)
   (%CHAR-TYPE-P (%ARRAY-TYPE-NUMBER ARRAY)))


(CL:DEFUN CL:SVREF (CL:SIMPLE-VECTOR INDEX)
   (CL:ASSERT (TYPEP CL:SIMPLE-VECTOR 'CL:SIMPLE-VECTOR)
          (CL:SIMPLE-VECTOR)
          "Not a simple-vector: ~S" CL:SIMPLE-VECTOR)
   (CL:AREF CL:SIMPLE-VECTOR INDEX))


(CL:DEFUN VECTOR-LENGTH (VECTOR)
   (CL:IF (%VECTORP VECTOR)
       (|fetch| (ARRAY-HEADER FILL-POINTER) |of| VECTOR)
       (CL:ERROR "Not a vector: ~s" VECTOR)))


(CL:DEFUN CL:VECTOR-POP (VECTOR)
   (COND
      ((AND (OR (%ONED-ARRAY-P VECTOR)
                (%GENERAL-ARRAY-P VECTOR))
            (|fetch| (ARRAY-HEADER FILL-POINTER-P) |of| VECTOR))
       (LET ((FILL-POINTER (|fetch| (ARRAY-HEADER FILL-POINTER) |of| VECTOR)))
            (CL:IF (<= FILL-POINTER 0)
                   (CL:ERROR "Can't pop from zero fill pointer"))
            (SETQ FILL-POINTER (CL:1- FILL-POINTER))
            (|replace| (ARRAY-HEADER FILL-POINTER) |of| VECTOR |with| FILL-POINTER)
            (CL:AREF VECTOR FILL-POINTER)))
      ((%VECTORP VECTOR)
       (CL:ERROR "Vector has no fill pointer"))
      (T (CL:ERROR "Not a vector: ~S" VECTOR))))


(CL:DEFUN CL:VECTOR-PUSH (NEW-ELEMENT VECTOR)
   (COND
      ((AND (OR (%ONED-ARRAY-P VECTOR)
                (%GENERAL-ARRAY-P VECTOR))
            (|fetch| (ARRAY-HEADER FILL-POINTER-P) |of| VECTOR))
       (LET ((FILL-POINTER (|fetch| (ARRAY-HEADER FILL-POINTER) |of| VECTOR)))
            (CL:WHEN (< FILL-POINTER (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| VECTOR))
                (ASET NEW-ELEMENT VECTOR FILL-POINTER)
                (|replace| (ARRAY-HEADER FILL-POINTER) |of| VECTOR |with| (CL:1+ FILL-POINTER))
                FILL-POINTER)))
      ((%VECTORP VECTOR)
       (CL:ERROR "Vector has no fill pointer"))
      (T (CL:ERROR "Not a vector: ~S" VECTOR))))


(CL:DEFUN CL:VECTOR-PUSH-EXTEND (NEW-ELEMENT VECTOR &OPTIONAL (EXTENSION-SIZE
                                                                 *DEFAULT-PUSH-EXTENSION-SIZE*))

   ;; Like VECTOR-PUSH except if VECTOR is adjustable -- in which case a push beyond (array-total-size VECTOR ) will call adjust-array

   (LET ((NEW-INDEX (CL:VECTOR-PUSH NEW-ELEMENT VECTOR)))
        (CL:IF (NULL NEW-INDEX)
            (COND
               ((> EXTENSION-SIZE 0)
                (CL:ADJUST-ARRAY VECTOR (+ (CL:ARRAY-TOTAL-SIZE VECTOR)
                                           EXTENSION-SIZE))
                (CL:VECTOR-PUSH NEW-ELEMENT VECTOR))
               (T (CL:ERROR "Extension-size not greater than zero")))
            NEW-INDEX)))


(CL:DEFUN CL:VECTORP (VECTOR)
   (%VECTORP VECTOR))

(DEFINEQ

(CL:AREF
  (LAMBDA ARGS                                                  ; Edited 11-Dec-87 15:32 by jop
    (CL:IF (< ARGS 1)
           (CL:ERROR "Aref takes at least one arg"))
    (LET ((ARRAY (ARG ARGS 1)))
         (CASE ARGS
             (1 (%AREF0 ARRAY))
             (2 (%AREF1 ARRAY (ARG ARGS 2)))
             (3 (%AREF2 ARRAY (ARG ARGS 2)
                       (ARG ARGS 3)))
             (T (COND
                   ((NOT (EQ (CL:ARRAY-RANK ARRAY)
                             (CL:1- ARGS)))
                    (CL:ERROR "Rank mismatch"))
                   (T ;; If we've gotten this far ARRAY must be a general array ; Check indices in bounds
                      (CL:DO ((I 2 (CL:1+ I))
                              (DIMLIST (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY)
                                     (CDR DIMLIST))
                              INDEX)
                             ((> I ARGS))
                          (SETQ INDEX (ARG ARGS I))
                          (CL:IF (NOT (< -1 INDEX (CAR DIMLIST)))
                                 (CL:ERROR "Index out of bounds: ~s" INDEX)))
                                                            ; Now proceed to extract the element
                      (LET ((ROW-MAJOR-INDEX (CL:DO ((I 2 (CL:1+ I))
                                                     (DIMLIST (CDR (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY))
                                                            (CDR DIMLIST))
                                                     (TOTAL 0))
                                                    ((EQ I ARGS)
                                                     (+ TOTAL (ARG ARGS ARGS)))
                                                 (SETQ TOTAL (CL:* (CAR DIMLIST)
                                                                   (+ TOTAL (ARG ARGS I))))))
                            (BASE-ARRAY ARRAY))
                           (%GENERAL-ARRAY-ADJUST-BASE BASE-ARRAY ROW-MAJOR-INDEX)
                           (%ARRAY-READ (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)
                                  (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY)
                                  (+ (%GET-ARRAY-OFFSET BASE-ARRAY)
                                     ROW-MAJOR-INDEX)))))))))
```

(CL:ARRAY-IN-BOUNDS-P

```
  (LAMBDA ARGS                                              ; Edited 11-Dec-87 15:32 by jop
    (CL:IF (< ARGS 1)
           (CL:ERROR "Array-in-bounds-p takes at least one arg"))
    (LET ((ARRAY (ARG ARGS 1)))
         (CL:IF (EQ (CL:ARRAY-RANK ARRAY)
                    (CL:1- ARGS))
             (%CHECK-INDICES ARRAY 2 ARGS)
             (CL:ERROR "Rank mismatch")))))
```

(CL:ARRAY-ROW-MAJOR-INDEX

```
  (LAMBDA ARGS                                              ; Edited 11-Dec-87 15:32 by jop
    (CL:IF (< ARGS 1)
           (CL:ERROR "Array-row-major-index takes at least one arg"))
    (LET ((ARRAY (ARG ARGS 1)))
         (COND
            ((NOT (EQ (CL:ARRAY-RANK ARRAY)
                      (CL:1- ARGS)))
             (CL:ERROR "Rank mismatch"))
            ((NOT (%CHECK-INDICES ARRAY 2 ARGS))
             (CL:ERROR "Index out of bounds"))
            (T (CL:DO ((I 2 (CL:1+ I))
                       (TOTAL 0))
                      ((EQ I ARGS)
                       (+ TOTAL (ARG ARGS ARGS)))
                   (SETQ TOTAL (CL:* (CL:ARRAY-DIMENSION ARRAY (CL:1- I))
                                     (+ TOTAL (ARG ARGS I)))))))))
```

(ASET

```
  (LAMBDA ARGS                                              ; Edited 11-Dec-87 15:33 by jop
    (CL:IF (< ARGS 2)
           (CL:ERROR "Aset takes at least two args"))
    (LET ((NEWVALUE (ARG ARGS 1))
          (ARRAY (ARG ARGS 2)))
         (CASE ARGS
             (2 (%ASET0 NEWVALUE ARRAY))
             (3 (%ASET1 NEWVALUE ARRAY (ARG ARGS 3)))
             (4 (%ASET2 NEWVALUE ARRAY (ARG ARGS 3)
                       (ARG ARGS 4)))
             (T (COND
                   ((NOT (EQ (CL:ARRAY-RANK ARRAY)
                             (- ARGS 2)))
                    (CL:ERROR "Rank mismatch"))
                   (T                                       ; If we've gotten this far array must be a general array

                      ;; Check indices

                      (CL:DO ((I 3 (CL:1+ I))
                              (DIMLIST (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY)
                                     (CDR DIMLIST))
                              INDEX)
                             ((> I ARGS))
                          (SETQ INDEX (ARG ARGS I))
                          (CL:IF (NOT (< -1 INDEX (CAR DIMLIST)))
                                 (CL:ERROR "Index out of bounds: ~s" INDEX)))
                    ;; Now proceed to extract the element
                    (LET ((ROW-MAJOR-INDEX (CL:DO ((I 3 (CL:1+ I))
                                                   (DIMLIST (CDR (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY))
                                                          (CDR DIMLIST))
                                                   (TOTAL 0))
                                                  ((EQ I ARGS)
                                                   (+ TOTAL (ARG ARGS ARGS)))
                                               (SETQ TOTAL (CL:* (CAR DIMLIST)
                                                                 (+ TOTAL (ARG ARGS I))))))
                          (BASE-ARRAY ARRAY))
                         (%GENERAL-ARRAY-ADJUST-BASE BASE-ARRAY ROW-MAJOR-INDEX)
                         (LET ((TYPE-NUMBER (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY)))
                              (CL:IF (%CHECK-NOT-WRITEABLE ARRAY TYPE-NUMBER NEWVALUE)
                                  (CL:APPLY 'ASET NEWVALUE ARRAY (CL:DO ((I ARGS (CL:1- I))
                                                                         LST)
                                                                        ((< I 1)
                                                                         LST)
                                                                     (SETQ LST (CONS (ARG ARGS I)
                                                                                     LST))))
                                  (%ARRAY-WRITE NEWVALUE (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)
                                         TYPE-NUMBER
                                         (+ (%GET-ARRAY-OFFSET BASE-ARRAY)
                                            ROW-MAJOR-INDEX))))))))))
```

(CL:VECTOR

```
  (LAMBDA ARGS                                                    ; Edited 18-Dec-86 18:09 by jop
    (LET ((VECTOR (%MAKE-ONED-ARRAY ARGS T)))
         (CL:DOTIMES (I ARGS)
             (ASET (ARG ARGS (CL:1+ I))
                   VECTOR I))
         VECTOR)))

)
```

;; New CLtL array functions

```
(DEFINEQ
```

(XCL:ROW-MAJOR-AREF

```
  (LAMBDA (ARRAY INDEX)                                           ; Edited 11-Dec-87 15:49 by jop
    ;; specialized aref for the one-d case. Also the punt function for the aref1 opcode.
    (CL:IF (NOT (AND (>= INDEX 0)
                     (< INDEX (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY))))
        (CL:ERROR "Index out of bounds: ~A" INDEX)
        (LET ((BASE-ARRAY ARRAY))
             ;; Now proceed to extract the element
             (%GENERAL-ARRAY-ADJUST-BASE BASE-ARRAY INDEX)
             (%ARRAY-READ (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)
                    (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY)
                    (+ (%GET-ARRAY-OFFSET BASE-ARRAY)
                       INDEX))))))
```

(CL::ROW-MAJOR-ASET

```
  (LAMBDA (ARRAY INDEX NEWVALUE)                                  ; Edited 11-Dec-87 15:54 by jop
    (CL:IF (NOT (AND (>= INDEX 0)
                     (< INDEX (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY))))
        (CL:ERROR "Index out of bounds: ~s" INDEX)
        (LET ((ROW-MAJOR-INDEX INDEX)
              (BASE-ARRAY ARRAY))
             ;; Now proceed to extract the element
             (%GENERAL-ARRAY-ADJUST-BASE BASE-ARRAY ROW-MAJOR-INDEX)
             (LET ((TYPE-NUMBER (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY)))
                  (CL:IF (%CHECK-NOT-WRITEABLE ARRAY TYPE-NUMBER NEWVALUE)
                      (CL::ROW-MAJOR-ASET ARRAY INDEX NEWVALUE)
                      (%ARRAY-WRITE NEWVALUE (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)
                             TYPE-NUMBER
                             (+ (%GET-ARRAY-OFFSET BASE-ARRAY)
                                ROW-MAJOR-INDEX))))))))

)

(CL:DEFSETF XCL:ROW-MAJOR-AREF CL::ROW-MAJOR-ASET)
```

;; Setfs

```
(CL:DEFSETF CL:AREF (ARRAY &REST INDICES) (NEWVALUE)
   `(ASET ,NEWVALUE ,ARRAY ,@INDICES))

(CL:DEFSETF BIT (ARRAY &REST INDICES) (NEWVALUE)
   `(ASET ,NEWVALUE ,ARRAY ,@INDICES))

(CL:DEFSETF CL:CHAR (ARRAY INDEX) (NEWVALUE)
   `(ASET ,NEWVALUE ,ARRAY ,INDEX))

(CL:DEFSETF CL:FILL-POINTER SET-FILL-POINTER)

(CL:DEFSETF CL:SBIT (ARRAY &REST INDICES) (NEWVALUE)
   `(ASET ,NEWVALUE ,ARRAY ,@INDICES))

(CL:DEFSETF CL:SCHAR (ARRAY INDEX) (NEWVALUE)
   `(ASET ,NEWVALUE ,ARRAY ,INDEX))

(CL:DEFSETF CL:SVREF (ARRAY INDEX) (NEWVALUE)
   `(ASET ,NEWVALUE ,ARRAY ,INDEX))
```

;; Optimizers

```
(CL:DEFUN %AREF-EXPANDER (ARRAY INDICES)
   (CASE (LENGTH INDICES)
       (1 `(%AREF1 ,ARRAY ,@INDICES))
       (2 `(%AREF2 ,ARRAY ,@INDICES))
       (T 'COMPILER:PASS)))

(CL:DEFUN %ASET-EXPANDER (NEWVALUE ARRAY INDICES)
   (CASE (LENGTH INDICES)
       (1 `(%ASET1 ,NEWVALUE ,ARRAY ,@INDICES))
       (2 `(%ASET2 ,NEWVALUE ,ARRAY ,@INDICES))
       (T 'COMPILER:PASS)))

(DEFOPTIMIZER CL:AREF (ARRAY &REST INDICES)
                      (%AREF-EXPANDER ARRAY INDICES))

(DEFOPTIMIZER ASET (NEWVALUE ARRAY &REST INDICES)
                   (%ASET-EXPANDER NEWVALUE ARRAY INDICES))

(DEFOPTIMIZER BIT (ARRAY &REST INDICES)
                  (%AREF-EXPANDER ARRAY INDICES))

(DEFOPTIMIZER CL:CHAR (STRING INDEX)
                      `(%AREF1 ,STRING ,INDEX))

(DEFOPTIMIZER CL:SBIT (ARRAY &REST INDICES)
                      (%AREF-EXPANDER ARRAY INDICES))

(DEFOPTIMIZER CL:SCHAR (STRING INDEX)
                       `(%AREF1 ,STRING ,INDEX))

(DEFOPTIMIZER CL:SVREF (CL:SIMPLE-VECTOR INDEX)
                       `(%AREF1 ,CL:SIMPLE-VECTOR ,INDEX))
```

;; Vars etc

;; *PRINT-ARRAY* is defined in APRINT

```
(CL:DEFCONSTANT CL:ARRAY-RANK-LIMIT (EXPT 2 7))

(CL:DEFCONSTANT CL:ARRAY-TOTAL-SIZE-LIMIT 65534)

(CL:DEFCONSTANT CL:ARRAY-DIMENSION-LIMIT CL:ARRAY-TOTAL-SIZE-LIMIT)
```

```
(CL:DEFPARAMETER *DEFAULT-PUSH-EXTENSION-SIZE* 20)

;; Run-time support

(DEFINEQ

(%ALTER-AS-DISPLACED-ARRAY
  (LAMBDA (ADJUSTABLE-ARRAY DIMENSIONS DISPLACED-TO DISPLACED-INDEX-OFFSET FILL-POINTER)
                                                                   ; Edited 18-Dec-86 17:11 by jop

    ;; Alter ADJUSTABLE-ARRAY to be displaced to displaced-to. ADJUSTABLE-ARRAY must be a general array

    (CL:IF (NULL DISPLACED-INDEX-OFFSET)
           (SETQ DISPLACED-INDEX-OFFSET 0))
    (LET ((DISPLACED-TO-READ-ONLY-P (|fetch| (ARRAY-HEADER READ-ONLY-P) |of| DISPLACED-TO))
          (TOTAL-SIZE (%TOTAL-SIZE DIMENSIONS))
          (OFFSET (OR DISPLACED-INDEX-OFFSET 0))
          BASE NEED-INDIRECTION-P)
         (COND
            ((OR (%THIN-CHAR-TYPE-P (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| DISPLACED-TO))
                 (|fetch| (ARRAY-HEADER EXTENDABLE-P) |of| DISPLACED-TO)
                 (|fetch| (ARRAY-HEADER ADJUSTABLE-P) |of| DISPLACED-TO)
                 (AND DISPLACED-TO-READ-ONLY-P (NOT (|fetch| (ARRAY-HEADER INDIRECT-P) |of| DISPLACED-TO))))
                                                                   ; Provide for indirection
             (SETQ BASE DISPLACED-TO)
             (SETQ NEED-INDIRECTION-P T))
            (T                                                     ; Fold double displacement to single displacement
               (SETQ BASE (|fetch| (ARRAY-HEADER BASE) |of| DISPLACED-TO))
               (SETQ OFFSET (+ OFFSET (%GET-ARRAY-OFFSET DISPLACED-TO)))
               (CL:IF (|fetch| (ARRAY-HEADER INDIRECT-P) |of| DISPLACED-TO)
                      (SETQ NEED-INDIRECTION-P T))))             ; Don't need to touch the type-number since it can't change
         (UNINTERRUPTABLY
             (|freplace| (GENERAL-ARRAY STORAGE) |of| ADJUSTABLE-ARRAY |with| BASE)
             (|freplace| (GENERAL-ARRAY READ-ONLY-P) |of| ADJUSTABLE-ARRAY |with| DISPLACED-TO-READ-ONLY-P)
             (|freplace| (GENERAL-ARRAY INDIRECT-P) |of| ADJUSTABLE-ARRAY |with| NEED-INDIRECTION-P)
             (|freplace| (GENERAL-ARRAY DISPLACED-P) |of| ADJUSTABLE-ARRAY |with| T)
             (|freplace| (GENERAL-ARRAY FILL-POINTER-P) |of| ADJUSTABLE-ARRAY |with| FILL-POINTER)
             (|freplace| (GENERAL-ARRAY OFFSET) |of| ADJUSTABLE-ARRAY |with| OFFSET)
             (|freplace| (GENERAL-ARRAY FILL-POINTER) |of| ADJUSTABLE-ARRAY |with| (OR FILL-POINTER TOTAL-SIZE))
             (|freplace| (GENERAL-ARRAY TOTAL-SIZE) |of| ADJUSTABLE-ARRAY |with| TOTAL-SIZE)
             (|freplace| (GENERAL-ARRAY DIMS) |of| ADJUSTABLE-ARRAY |with| DIMENSIONS))
         ADJUSTABLE-ARRAY)))

(%ALTER-AS-DISPLACED-TO-BASE-ARRAY
  (LAMBDA (ADJUSTABLE-ARRAY DIMENSIONS ELEMENT-TYPE DISPLACED-TO-BASE DISPLACED-INDEX-OFFSET FILL-POINTER FATP)
                                                                   ; Edited 18-Dec-86 17:12 by jop

    ;; Alter adjustable-array to be displaced to displaced-to-base

    (LET ((TOTAL-SIZE (%TOTAL-SIZE DIMENSIONS))
          (TYPE-NUMBER (%CML-TYPE-TO-TYPENUMBER ELEMENT-TYPE FATP)))
         (UNINTERRUPTABLY
             (|freplace| (GENERAL-ARRAY STORAGE) |of| ADJUSTABLE-ARRAY |with| DISPLACED-TO-BASE)
             (|freplace| (GENERAL-ARRAY INDIRECT-P) |of| ADJUSTABLE-ARRAY |with| NIL)
             (|freplace| (GENERAL-ARRAY DISPLACED-P) |of| ADJUSTABLE-ARRAY |with| T)
             (|freplace| (GENERAL-ARRAY FILL-POINTER-P) |of| ADJUSTABLE-ARRAY |with| FILL-POINTER)
             (|freplace| (GENERAL-ARRAY TYPE-NUMBER) |of| ADJUSTABLE-ARRAY |with| TYPE-NUMBER)
             (|freplace| (GENERAL-ARRAY OFFSET) |of| ADJUSTABLE-ARRAY |with| (OR DISPLACED-INDEX-OFFSET 0))
             (|freplace| (GENERAL-ARRAY FILL-POINTER) |of| ADJUSTABLE-ARRAY |with| (OR FILL-POINTER TOTAL-SIZE))
             (|freplace| (GENERAL-ARRAY TOTAL-SIZE) |of| ADJUSTABLE-ARRAY |with| TOTAL-SIZE)
             (|freplace| (GENERAL-ARRAY DIMS) |of| ADJUSTABLE-ARRAY |with| DIMENSIONS))
         ADJUSTABLE-ARRAY)))

(%AREF0
  (LAMBDA (ARRAY)                                                  ; Edited 11-Dec-87 15:33 by jop

    ;; Special aref for the zero dimensional case

    (CL:IF (EQ (CL:ARRAY-RANK ARRAY)
               0)
        (LET ((INDEX 0)
              (BASE-ARRAY ARRAY))

             ;; Must be a general array

             (%GENERAL-ARRAY-ADJUST-BASE BASE-ARRAY INDEX)
             (%ARRAY-READ (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)
                    (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY)
                    (+ (%GET-ARRAY-OFFSET BASE-ARRAY)
                       INDEX)))
        (CL:ERROR "Rank mismatch"))))

(%AREF1
  (LAMBDA (ARRAY INDEX)                                            ; Edited 11-Dec-87 15:50 by jop

    ;; specialized aref for the one-d case. Also the punt function for the aref1 opcode.

    (COND
        ((NOT (EQ (CL:ARRAY-RANK ARRAY)
                  1))
         (CL:ERROR "Rank mismatch"))
        ((NOT (AND (>= INDEX 0)
                   (< INDEX (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY))))
         (CL:ERROR "Index out of bounds: ~A" INDEX))
        (T ;; Now proceed to extract the element

           (LET ((BASE-ARRAY ARRAY))
                (%GENERAL-ARRAY-ADJUST-BASE BASE-ARRAY INDEX)
                (%ARRAY-READ (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)
                       (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY)
                       (+ (%GET-ARRAY-OFFSET BASE-ARRAY)
                          INDEX))))))
```

(%AREF2

```
  (LAMBDA (ARRAY I J)                                            ; Edited 11-Dec-87 15:33 by jop

    ;; Specialized aref for the two-d case. Also the punt function for the aref 2 opcode.

    (CL:IF (EQ (CL:ARRAY-RANK ARRAY)
               2)
        (LET (BOUND0 BOUND1 OFFSET)                              ;  ARRAY must be two-d or general

             ;; Get bounds and offset

             (COND
                ((%TWOD-ARRAY-P ARRAY)                           ; Twod array case
                 (SETQ BOUND0 (|ffetch| (TWOD-ARRAY BOUND0) |of| ARRAY))
                 (SETQ BOUND1 (|ffetch| (TWOD-ARRAY BOUND1) |of| ARRAY))
                 (SETQ OFFSET 0))
                (T                                               ; General array case
                   (SETQ BOUND0 (CAR (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY)))
                   (SETQ BOUND1 (CADR (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY)))
                   (SETQ OFFSET (|ffetch| (GENERAL-ARRAY OFFSET) |of| ARRAY))))
                                                                 ; Check indices
             (COND
                ((NOT (< -1 I BOUND0))
                 (CL:ERROR "Index out of bounds: ~A" I))
                ((NOT (< -1 J BOUND1))
                 (CL:ERROR "Index out of bounds: ~A" J)))        ; Extract the element
             (LET ((ROW-MAJOR-INDEX (+ J (CL:* BOUND1 I)))
                   (BASE-ARRAY ARRAY))
                  (%GENERAL-ARRAY-ADJUST-BASE BASE-ARRAY ROW-MAJOR-INDEX)
                  (%ARRAY-READ (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)
                         (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY)
                         (+ (%GET-ARRAY-OFFSET BASE-ARRAY)
                            ROW-MAJOR-INDEX))))
        (CL:ERROR "Rank mismatch"))))
```

(%ARRAY-BASE

```
  (LAMBDA (ARRAY)                                                ; Edited 18-Dec-86 17:20 by jop
    (COND
       ((OR (%ONED-ARRAY-P ARRAY)
            (%TWOD-ARRAY-P ARRAY))
        (|fetch| (ARRAY-HEADER BASE) |of| ARRAY))
       ((%GENERAL-ARRAY-P ARRAY)
        (|fetch| (ARRAY-HEADER BASE) |of| (CL:LOOP (CL:IF (NOT (|fetch| (ARRAY-HEADER INDIRECT-P) |of| ARRAY))
                                                           (RETURN ARRAY))
                                                  (SETQ ARRAY (|fetch| (ARRAY-HEADER BASE) |of| ARRAY)))))
       (T (CL:ERROR "Not an array: ~S" ARRAY)))))
```

(%ARRAY-CONTENT-INITIALIZE

```
  (LAMBDA (ARRAY INITIAL-CONTENTS)                               ; Edited 11-Dec-87 15:33 by jop
    (CL:IF (EQ 0 (CL:ARRAY-RANK ARRAY))
        (%ARRAY-ELEMENT-INITIALIZE ARRAY INITIAL-CONTENTS)
        (LET ((DIMS (CL:ARRAY-DIMENSIONS ARRAY)))
             (CL:IF (%CHECK-SEQUENCE-DIMENSIONS DIMS INITIAL-CONTENTS)
                 (%FILL-ARRAY-FROM-SEQUENCE DIMS INITIAL-CONTENTS (%FLATTEN-ARRAY ARRAY)
                        0)
                 (CL:ERROR "Dimensionality mismatch for Initial-contents"))))))
```

(%ARRAY-ELEMENT-INITIALIZE

```
  (LAMBDA (ARRAY INITIAL-ELEMENT)                                ; Edited 11-Dec-87 15:33 by jop

    ;; Initialize an array with a value

    (CL:UNLESS (EQ INITIAL-ELEMENT (%TYPENUMBER-TO-DEFAULT-VALUE (%ARRAY-TYPE-NUMBER ARRAY)))
           (FILL-ARRAY ARRAY INITIAL-ELEMENT))))
```

(%ARRAY-OFFSET

```
  (LAMBDA (ARRAY)                                                ; Edited 18-Dec-86 17:22 by jop

    ;; Get the true offset for ARRAY

    (COND
        ((%ONED-ARRAY-P ARRAY)
         (|fetch| (ARRAY-HEADER OFFSET) |of| ARRAY))
        ((%TWOD-ARRAY-P ARRAY)
         0)
        ((%GENERAL-ARRAY-P ARRAY)
         (CL:DO ((OFFSET (|fetch| (ARRAY-HEADER OFFSET) |of| ARRAY)
                        (+ OFFSET (%GET-ARRAY-OFFSET ARRAY))))
                ((NOT (|fetch| (ARRAY-HEADER INDIRECT-P) |of| ARRAY))
                 OFFSET)
             (SETQ ARRAY (|fetch| (ARRAY-HEADER BASE) |of| ARRAY))))
        (T (CL:ERROR "Not an array: ~S" ARRAY)))))
```

(%ARRAY-TYPE-NUMBER

```
  (LAMBDA (ARRAY)                                          ; Edited 18-Dec-86 17:23 by jop

    ;; Get the true array-typenumber for ARRAY

    (COND
       ((OR (%ONED-ARRAY-P ARRAY)
            (%TWOD-ARRAY-P ARRAY))
        (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| ARRAY))
       ((%GENERAL-ARRAY-P ARRAY)
        (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| (CL:LOOP (CL:IF (NOT (|fetch| (ARRAY-HEADER INDIRECT-P) |of| ARRAY))
                                                                  (RETURN ARRAY))
                                                           (SETQ ARRAY (|fetch| (ARRAY-HEADER BASE) |of| ARRAY)))))
       (T (CL:ERROR "Not an array: ~S" ARRAY)))))
```

(%ASET0

```
  (LAMBDA (NEWVALUE ARRAY)                                 ; Edited 11-Dec-87 15:33 by jop

    ;; Specialized aset for the zero-d case.

    (CL:IF (EQ (CL:ARRAY-RANK ARRAY)
               0)
        (LET ((INDEX 0)
              (BASE-ARRAY ARRAY))

             ;; Must be a general array

             (%GENERAL-ARRAY-ADJUST-BASE BASE-ARRAY INDEX)
             (LET ((TYPE-NUMBER (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY)))
                  (CL:IF (%CHECK-NOT-WRITEABLE ARRAY TYPE-NUMBER NEWVALUE)
                      (%ASET0 NEWVALUE ARRAY)
                      (%ARRAY-WRITE NEWVALUE (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)
                             TYPE-NUMBER
                             (+ (%GET-ARRAY-OFFSET BASE-ARRAY)
                                INDEX)))))
        (CL:ERROR "Rank mismatch"))))
```

(%ASET1

```
  (LAMBDA (NEWVALUE ARRAY INDEX)                           ; Edited 11-Dec-87 15:34 by jop

    ;; Specialized aset for the one-d case. Also the punt for the aset1 opcode.

    (COND
       ((NOT (EQ (CL:ARRAY-RANK ARRAY)
                 1))
        (CL:ERROR "Rank mismatch"))
       ((NOT (AND (>= INDEX 0)
                  (< INDEX (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| ARRAY))))
        (CL:ERROR "Index out of bounds: ~s" INDEX))
       (T ;; Now proceed to extract the element

          (LET ((ROW-MAJOR-INDEX INDEX)
                (BASE-ARRAY ARRAY))
               (%GENERAL-ARRAY-ADJUST-BASE BASE-ARRAY ROW-MAJOR-INDEX)
               (LET ((TYPE-NUMBER (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY)))
                    (CL:IF (%CHECK-NOT-WRITEABLE ARRAY TYPE-NUMBER NEWVALUE)
                        (%ASET1 NEWVALUE ARRAY INDEX)
                        (%ARRAY-WRITE NEWVALUE (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)
                               TYPE-NUMBER
                               (+ (%GET-ARRAY-OFFSET BASE-ARRAY)
                                  ROW-MAJOR-INDEX)))))))))
```

(%ASET2

```
  (LAMBDA (NEWVALUE ARRAY I J)                             ; Edited 11-Dec-87 15:34 by jop

    ;; Specialized aset for the two-d case. Also the punt function for the aset2 opcode.

    (CL:IF (EQ (CL:ARRAY-RANK ARRAY)
               2)
        (LET (BOUND0 BOUND1 OFFSET)

             ;; Get bounds and offset

             (COND
                ((%TWOD-ARRAY-P ARRAY)                     ; Twod case
                 (SETQ BOUND0 (|ffetch| (TWOD-ARRAY BOUND0) |of| ARRAY))
                 (SETQ BOUND1 (|ffetch| (TWOD-ARRAY BOUND1) |of| ARRAY))
                (SETQ OFFSET 0))
               (T                                                    ; General Case
                  (SETQ BOUND0 (CAR (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY)))
                  (SETQ BOUND1 (CADR (|ffetch| (GENERAL-ARRAY DIMS) |of| ARRAY)))
                  (SETQ OFFSET (|ffetch| (GENERAL-ARRAY OFFSET) |of| ARRAY))))
             ;; Check indices
             (COND
                ((NOT (< -1 I BOUND0))
                 (CL:ERROR "Index out of bounds ~s" I))
                ((NOT (< -1 J BOUND1))
                 (CL:ERROR "Index out of bounds ~s" J)))
             ;; Set element
             (LET ((ROW-MAJOR-INDEX (+ J (CL:* BOUND1 I)))
                   (BASE-ARRAY ARRAY))
                  (%GENERAL-ARRAY-ADJUST-BASE BASE-ARRAY ROW-MAJOR-INDEX)
                  (LET ((TYPE-NUMBER (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY)))
                       (CL:IF (%CHECK-NOT-WRITEABLE ARRAY TYPE-NUMBER NEWVALUE)
                           (%ASET2 NEWVALUE ARRAY I J)
                           (%ARRAY-WRITE NEWVALUE (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)
                                  TYPE-NUMBER
                                  (+ (%GET-ARRAY-OFFSET BASE-ARRAY)
                                     ROW-MAJOR-INDEX))))))
        (CL:ERROR "Rank mismatch"))))
```

(%CHECK-SEQUENCE-DIMENSIONS

```
  (LAMBDA (DIM-LST SEQUENCE)                                         ; Edited 11-Dec-87 15:34 by jop
    ;; Returns NIL if there is a mismatch
    (CL:IF (EQ (CAR DIM-LST)
               (CL:LENGTH SEQUENCE))
        (OR (NULL (CDR DIM-LST))
            (CL:DOTIMES (I (CAR DIM-LST)
                           T)
                (CL:IF (NOT (%CHECK-SEQUENCE-DIMENSIONS (CDR DIM-LST)
                                   (CL:ELT SEQUENCE I)))
                       (RETURN NIL)))))))
```

(%COPY-TO-NEW-ARRAY

```
  (LAMBDA (OLD-DIMS OLD-ARRAY OLD-OFFSET NEW-DIMS NEW-ARRAY NEW-OFFSET)
                                                                     ; Edited 13-Feb-87 15:52 by jop
    ;; It is assumed that OLD-ARRAY and NEW-ARRAY are of the same rank
    (LET ((SIZE (MIN (CAR OLD-DIMS)
                     (CAR NEW-DIMS))))
         (CL:IF (CDR OLD-DIMS)
             (CL:DOTIMES (I SIZE)
                 (%COPY-TO-NEW-ARRAY (CDR OLD-DIMS)
                        OLD-ARRAY
                        (CL:* (CADR OLD-DIMS)
                              (+ OLD-OFFSET I))
                        (CDR NEW-DIMS)
                        NEW-ARRAY
                        (CL:* (CADR NEW-DIMS)
                              (+ NEW-OFFSET I))))
             (%FAST-COPY-BASE (%ARRAY-BASE OLD-ARRAY)
                    (+ (%ARRAY-OFFSET OLD-ARRAY)
                       OLD-OFFSET)
                    (%ARRAY-TYPE-NUMBER OLD-ARRAY)
                    (%ARRAY-BASE NEW-ARRAY)
                    (+ (%ARRAY-OFFSET NEW-ARRAY)
                       NEW-OFFSET)
                    (%ARRAY-TYPE-NUMBER NEW-ARRAY)
                    SIZE)))))
```

(%DO-LOGICAL-OP

```
  (LAMBDA (OP SOURCE DEST)                                           ; Edited 18-Dec-86 17:43 by jop
    (LET ((SOURCE-BASE (%ARRAY-BASE SOURCE))
          (SOURCE-OFFSET (%ARRAY-OFFSET SOURCE))
          (SOURCE-SIZE (CL:ARRAY-TOTAL-SIZE SOURCE))
          (DEST-BASE (%ARRAY-BASE DEST))
          (DEST-OFFSET (%ARRAY-OFFSET DEST))
          (GBBT (DEFERREDCONSTANT (|create| PILOTBBT
                                          PBTHEIGHT _ 1
                                          PBTDISJOINT _ T)))
          SOURCE-OP LOG-OP)
         (UNINTERRUPTABLY
             (|replace| (PILOTBBT PBTSOURCE) |of| GBBT |with| SOURCE-BASE)
             (|replace| (PILOTBBT PBTSOURCEBIT) |of| GBBT |with| SOURCE-OFFSET)
             (|replace| (PILOTBBT PBTDEST) |of| GBBT |with| DEST-BASE)
             (|replace| (PILOTBBT PBTDESTBIT) |of| GBBT |with| DEST-OFFSET)
             (|replace| (PILOTBBT PBTDESTBPL) |of| GBBT |with| SOURCE-SIZE)
             (|replace| (PILOTBBT PBTSOURCEBPL) |of| GBBT |with| SOURCE-SIZE)
             (|replace| (PILOTBBT PBTWIDTH) |of| GBBT |with| SOURCE-SIZE)
             (CASE OP
                 (COPY
                    (SETQ SOURCE-OP 0)
                    (SETQ LOG-OP 0))
                 (NOT
                    (SETQ SOURCE-OP 1)
                    (SETQ LOG-OP 0))
                 (AND
                    (SETQ SOURCE-OP 0)
                    (SETQ LOG-OP 1))
                 (CAND
                    (SETQ SOURCE-OP 1)
                    (SETQ LOG-OP 1))
                 (OR
                    (SETQ SOURCE-OP 0)
                    (SETQ LOG-OP 2))
                 (COR
                    (SETQ SOURCE-OP 1)
                    (SETQ LOG-OP 2))
                 (XOR
                    (SETQ SOURCE-OP 0)
                    (SETQ LOG-OP 3))
                 (CXOR
                    (SETQ SOURCE-OP 1)
                    (SETQ LOG-OP 3)))
             (|replace| (PILOTBBT PBTSOURCETYPE) |of| GBBT |with| SOURCE-OP)
             (|replace| (PILOTBBT PBTOPERATION) |of| GBBT |with| LOG-OP)
                                                                          ; Execute the BLT
             (\\PILOTBITBLT GBBT 0)
             DEST))))
```

(**%EXTEND-ARRAY**

```
  (LAMBDA (EXTENDABLE-ARRAY NEW-ARRAY DIMENSIONS FILL-POINTER)    ; Edited 18-Dec-86 17:43 by jop

    ;; Extend ADJUSTABLE-ARRAY, using the base provided by NEW-ARRAY

    (LET ((TYPE-NUMBER (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| NEW-ARRAY))
          (TOTAL-SIZE (%TOTAL-SIZE DIMENSIONS))
          (BASE (|fetch| (ARRAY-HEADER BASE) |of| NEW-ARRAY)))
         (UNINTERRUPTABLY
             (|replace| (ARRAY-HEADER BASE) |of| EXTENDABLE-ARRAY |with| BASE)
             (|replace| (ARRAY-HEADER READ-ONLY-P) |of| EXTENDABLE-ARRAY |with| NIL)
             (|replace| (ARRAY-HEADER TYPE-NUMBER) |of| EXTENDABLE-ARRAY |with| TYPE-NUMBER)
             (|replace| (ARRAY-HEADER TOTAL-SIZE) |of| EXTENDABLE-ARRAY |with| TOTAL-SIZE)
             (COND
                ((%TWOD-ARRAY-P EXTENDABLE-ARRAY)
                 (|freplace| (TWOD-ARRAY BOUND0) |of| EXTENDABLE-ARRAY |with| (CAR DIMENSIONS))
                 (|freplace| (TWOD-ARRAY BOUND1) |of| EXTENDABLE-ARRAY |with| (CADR DIMENSIONS)))
                (T                                                ; must be oned or general
                   (|replace| (ARRAY-HEADER DISPLACED-P) |of| EXTENDABLE-ARRAY |with| NIL)
                   (|replace| (ARRAY-HEADER FILL-POINTER-P) |of| EXTENDABLE-ARRAY |with| FILL-POINTER)
                   (|replace| (ARRAY-HEADER OFFSET) |of| EXTENDABLE-ARRAY |with| 0)
                   (|replace| (ARRAY-HEADER FILL-POINTER) |of| EXTENDABLE-ARRAY |with| (OR FILL-POINTER TOTAL-SIZE))
                   (CL:WHEN (%GENERAL-ARRAY-P EXTENDABLE-ARRAY)
                       (|freplace| (GENERAL-ARRAY INDIRECT-P) |of| EXTENDABLE-ARRAY |with| NIL)
                       (|freplace| (GENERAL-ARRAY DIMS) |of| EXTENDABLE-ARRAY |with| DIMENSIONS)))))
         EXTENDABLE-ARRAY)))
```

(**%FAST-COPY-BASE**

```
  (LAMBDA (FROM-BASE FROM-OFFSET FROM-TYPENUMBER TO-BASE TO-OFFSET TO-TYPENUMBER CNT)
                                                                  ; Edited 11-Dec-87 15:34 by jop

    ;; Blts one array into another of the same element-type

    (CL:IF (OR (NOT (EQ FROM-TYPENUMBER TO-TYPENUMBER))
               (EQ (%TYPENUMBER-TO-GC-TYPE TO-TYPENUMBER)
                   PTRBLOCK.GCT))
        (CL:DO ((I FROM-OFFSET (CL:1+ I))
                (LIMIT (+ FROM-OFFSET CNT))
                (J TO-OFFSET (CL:1+ J)))
               ((EQ I LIMIT))
            (%ARRAY-WRITE (%ARRAY-READ FROM-BASE FROM-TYPENUMBER I)
                   TO-BASE TO-TYPENUMBER J))
        (LET ((BITS-PER-ELEMENT (%TYPENUMBER-TO-BITS-PER-ELEMENT TO-TYPENUMBER))
              (PBBT (DEFERREDCONSTANT (|create| PILOTBBT
                                             PBTDISJOINT _ T
                                             PBTSOURCETYPE _ 0
                                             PBTOPERATION _ 0))))

             ;; Uses \PILOTBITBLT instead of \BLT because offsets might not be word aligned, and BITS-PER-ELEMENT may be greater than
             ;; BITSPERWORD (16).

             (UNINTERRUPTABLY
                 (|freplace| (PILOTBBT PBTSOURCE) |of| PBBT |with| FROM-BASE)
                 (|freplace| (PILOTBBT PBTSOURCEBIT) |of| PBBT |with| (CL:* BITS-PER-ELEMENT FROM-OFFSET))
                 (|freplace| (PILOTBBT PBTDEST) |of| PBBT |with| TO-BASE)
                 (|freplace| (PILOTBBT PBTDESTBIT) |of| PBBT |with| (CL:* BITS-PER-ELEMENT TO-OFFSET))
                 (|freplace| (PILOTBBT PBTDESTBPL) |of| PBBT |with| BITS-PER-ELEMENT)
                 (|freplace| (PILOTBBT PBTSOURCEBPL) |of| PBBT |with| BITS-PER-ELEMENT)
                 (|freplace| (PILOTBBT PBTWIDTH) |of| PBBT |with| BITS-PER-ELEMENT)
                 (|freplace| (PILOTBBT PBTHEIGHT) |of| PBBT |with| CNT)
                 (\\PILOTBITBLT PBBT 0))
             NIL))))
```

(%FAT-STRING-ARRAY-P

```
  (LAMBDA (ARRAY)                                            ; Edited 18-Dec-86 17:44 by jop
    (%FAT-CHAR-TYPE-P (%ARRAY-TYPE-NUMBER ARRAY))))
```

(%FILL-ARRAY-FROM-SEQUENCE

```
  (LAMBDA (DIMS SEQUENCE FLATTENED-ARRAY OFFSET)             ; Edited 11-Dec-87 15:34 by jop
    (CL:IF (CDR DIMS)
        (CL:DOTIMES (I (CAR DIMS))
            (%FILL-ARRAY-FROM-SEQUENCE (CDR DIMS)
                   (CL:ELT SEQUENCE I)
                   FLATTENED-ARRAY
                   (CL:* (CADR DIMS)
                         (+ OFFSET I))))
        (CL:DO ((I 0 (CL:1+ I))
                (J OFFSET (CL:1+ J))
                (LIMIT (CAR DIMS)))
               ((EQ I LIMIT))
            (ASET (CL:ELT SEQUENCE I)
                  FLATTENED-ARRAY J)))))
```

(%FLATTEN-ARRAY

```
  (LAMBDA (ARRAY)                                            ; Edited 11-Dec-87 15:34 by jop

    ;; Make a oned-array that shares storage with array.  If array is already oned then return array

    (CL:IF (EQ 1 (CL:ARRAY-RANK ARRAY))
        ARRAY
        (CL:MAKE-ARRAY (CL:ARRAY-TOTAL-SIZE ARRAY)
               :ELEMENT-TYPE
               (CL:ARRAY-ELEMENT-TYPE ARRAY)
               :DISPLACED-TO ARRAY))))
```

(%MAKE-ARRAY-WRITEABLE

```
  (LAMBDA (ARRAY)                                            ; Edited 18-Dec-86 18:40 by jop
    (CL:IF (NOT (%ARRAYP ARRAY))
           (CL:ERROR "Not an array: ~S" ARRAY))
    (LET ((BASE-ARRAY ARRAY)
          NEW-BASE OFFSET TOTAL-SIZE TYPE-NUMBER)

         ;; Find the base array

         (CL:IF (|fetch| (ARRAY-HEADER INDIRECT-P) |of| ARRAY)
             (CL:LOOP (CL:IF (|fetch| (ARRAY-HEADER INDIRECT-P) |of| BASE-ARRAY)
                             (SETQ BASE-ARRAY (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY))
                             (RETURN NIL))))
         (CL:WHEN (|fetch| (ARRAY-HEADER READ-ONLY-P) |of| BASE-ARRAY)

             ;; Allocate the new storage                         ; Be careful about offsets
             (SETQ TOTAL-SIZE (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| BASE-ARRAY))
             (SETQ OFFSET (%GET-ARRAY-OFFSET BASE-ARRAY))
             (SETQ TYPE-NUMBER (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY))
             (SETQ NEW-BASE (%MAKE-ARRAY-STORAGE (+ TOTAL-SIZE OFFSET)
                                   TYPE-NUMBER))

             ;; Initialize it

             (%FAST-COPY-BASE (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)
                    OFFSET TYPE-NUMBER NEW-BASE OFFSET TYPE-NUMBER TOTAL-SIZE)

             ;; Smash the new base into the array-header

             (UNINTERRUPTABLY
                 (|replace| (ARRAY-HEADER BASE) |of| BASE-ARRAY |with| NEW-BASE)
                 (|replace| (ARRAY-HEADER READ-ONLY-P) |of| BASE-ARRAY |with| NIL)))

         ;; Declare the array (and all arrays on its access chain) readable

         (UNINTERRUPTABLY
             (CL:DO ((NEXT-ARRAY ARRAY (|fetch| (ARRAY-HEADER BASE) |of| NEXT-ARRAY)))
                    ((NOT (|fetch| (ARRAY-HEADER INDIRECT-P) |of| NEXT-ARRAY)))
                 (|replace| (ARRAY-HEADER READ-ONLY-P) |of| NEXT-ARRAY |with| NIL)))

         ;; return the original array

         ARRAY)))
```

(%MAKE-DISPLACED-ARRAY

```
  (LAMBDA (TOTALSIZE DIMENSIONS ELEMENT-TYPE DISPLACED-TO DISPLACED-INDEX-OFFSET FILL-POINTER READ-ONLY-P
                 ADJUSTABLE EXTENDABLE)                      ; Edited 18-Dec-86 17:48 by jop

    ;; Make a displaced array

    (LET ((DISPLACED-TO-TYPENUMBER (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| DISPLACED-TO))
          (DISPLACE-TO-READ-ONLY-P (|fetch| (ARRAY-HEADER READ-ONLY-P) |of| DISPLACED-TO))
          (OFFSET (OR DISPLACED-INDEX-OFFSET 0))
          BASE NEED-INDIRECTION-P)
         (COND
            ((OR (%THIN-CHAR-TYPE-P DISPLACED-TO-TYPENUMBER)
                 (|fetch| (ARRAY-HEADER EXTENDABLE-P) |of| DISPLACED-TO)
                 (|fetch| (ARRAY-HEADER ADJUSTABLE-P) |of| DISPLACED-TO)
                 (AND DISPLACE-TO-READ-ONLY-P (NOT (|fetch| (ARRAY-HEADER INDIRECT-P) |of| DISPLACED-TO))))
                                                                 ; Provide for indirection
             (SETQ BASE DISPLACED-TO)
             (SETQ NEED-INDIRECTION-P T))
            (T                                                   ; Fold double displacement to single displacement
               (SETQ BASE (|fetch| (ARRAY-HEADER BASE) |of| DISPLACED-TO))
               (SETQ OFFSET (+ OFFSET (%GET-ARRAY-OFFSET DISPLACED-TO)))
               (CL:IF (|fetch| (ARRAY-HEADER INDIRECT-P) |of| DISPLACED-TO)
                      (SETQ NEED-INDIRECTION-P T))))
         (COND
            ((OR NEED-INDIRECTION-P ADJUSTABLE (> (LENGTH DIMENSIONS)
                                                  1))            ; Indirect strings always have %FAT-CHAR-TYPENUMBER
             (%MAKE-GENERAL-ARRAY TOTALSIZE DIMENSIONS ELEMENT-TYPE FILL-POINTER (%CHAR-TYPE-P
                                                                                    DISPLACED-TO-TYPENUMBER
                                                                                     )
                    (OR READ-ONLY-P DISPLACE-TO-READ-ONLY-P)
                    ADJUSTABLE EXTENDABLE BASE OFFSET))
            (T (%MAKE-ONED-ARRAY TOTALSIZE ELEMENT-TYPE FILL-POINTER (%FAT-CHAR-TYPE-P DISPLACED-TO-TYPENUMBER
                                                                                          )
                      (OR READ-ONLY-P DISPLACE-TO-READ-ONLY-P)
                      EXTENDABLE BASE OFFSET))))))
```

(%MAKE-GENERAL-ARRAY

```
  (LAMBDA (TOTAL-SIZE DIMENSIONS ELEMENT-TYPE FILL-POINTER FATP READ-ONLY-P ADJUSTABLE-P EXTENDABLE-P
                 DISPLACED-TO DISPLACED-INDEX-OFFSET)              ; Edited 11-Dec-87 15:35 by jop

    ;; General arrays cover all make-array cases, including those requiring indirection.

    (LET ((TYPE-NUMBER (%CML-TYPE-TO-TYPENUMBER ELEMENT-TYPE FATP)))
         (|create| GENERAL-ARRAY
                STORAGE _ (OR DISPLACED-TO (%MAKE-ARRAY-STORAGE TOTAL-SIZE TYPE-NUMBER))
                READ-ONLY-P _ READ-ONLY-P
                INDIRECT-P _ (%ARRAYP DISPLACED-TO)
                BIT-P _ (%BIT-TYPE-P TYPE-NUMBER)
                STRING-P _ (AND (%CHAR-TYPE-P TYPE-NUMBER)
                                (EQ 1 (LENGTH DIMENSIONS)))
                ADJUSTABLE-P _ ADJUSTABLE-P
                DISPLACED-P _ DISPLACED-TO
                FILL-POINTER-P _ FILL-POINTER
                EXTENDABLE-P _ (OR EXTENDABLE-P ADJUSTABLE-P)
                TYPE-NUMBER _ TYPE-NUMBER
                OFFSET _ (OR DISPLACED-INDEX-OFFSET 0)
                FILL-POINTER _ (OR FILL-POINTER TOTAL-SIZE)
                TOTAL-SIZE _ TOTAL-SIZE
                DIMS _ DIMENSIONS))))
```

(%MAKE-ONED-ARRAY

```
  (LAMBDA (TOTAL-SIZE ELEMENT-TYPE FILL-POINTER FATP READ-ONLY-P EXTENDABLE-P DISPLACED-TO
                 DISPLACED-INDEX-OFFSET)                           ; Edited 18-Dec-86 17:48 by jop

    ;; Oned-arrays cover all one dimensional cases, except adjustable and displaced-to when indirection is necessary

    (LET ((TYPE-NUMBER (%CML-TYPE-TO-TYPENUMBER ELEMENT-TYPE FATP)))
         (|create| ONED-ARRAY
                BASE _ (OR DISPLACED-TO (%MAKE-ARRAY-STORAGE TOTAL-SIZE TYPE-NUMBER))
                READ-ONLY-P _ READ-ONLY-P
                BIT-P _ (%BIT-TYPE-P TYPE-NUMBER)
                STRING-P _ (%CHAR-TYPE-P TYPE-NUMBER)
                DISPLACED-P _ DISPLACED-TO
                FILL-POINTER-P _ FILL-POINTER
                EXTENDABLE-P _ EXTENDABLE-P
                TYPE-NUMBER _ TYPE-NUMBER
                OFFSET _ (OR DISPLACED-INDEX-OFFSET 0)
                FILL-POINTER _ (OR FILL-POINTER TOTAL-SIZE)
                TOTAL-SIZE _ TOTAL-SIZE))))
```

(%MAKE-STRING-ARRAY-FAT

```
  (LAMBDA (ARRAY)                                                  ; Edited 11-Dec-87 15:35 by jop

    ;; Like Adjust-array for the special case of Thin-string arrays

    (CL:IF (NOT (%ARRAYP ARRAY))
           (CL:ERROR "Not an array" ARRAY))
    (LET ((BASE-ARRAY ARRAY)
          NEW-BASE OFFSET LIMIT)

         ;; Find the base array

         (CL:IF (|fetch| (ARRAY-HEADER INDIRECT-P) |of| ARRAY)
             (CL:LOOP (CL:IF (|fetch| (ARRAY-HEADER INDIRECT-P) |of| BASE-ARRAY)
                             (SETQ BASE-ARRAY (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY))
```

```
                         (RETURN NIL))))
          ;; Consistency check
          (CL:IF (NOT (%THIN-CHAR-TYPE-P (|fetch| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY)))
                 (CL:ERROR "Not a thin string-char array: ~S" BASE-ARRAY))
          ;; Allocate the new storage                                   ; Be careful about offsets
          (SETQ OFFSET (%GET-ARRAY-OFFSET BASE-ARRAY))
          (SETQ LIMIT (+ (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| BASE-ARRAY)
                         OFFSET))
          (SETQ NEW-BASE (%MAKE-ARRAY-STORAGE LIMIT %FAT-CHAR-TYPENUMBER))
          ;; Initialize it                                              ; Can't use %fast-copy-base because of the differing type
                                                                        ; numbers
          (CL:DO ((I OFFSET (CL:1+ I))
                  (BASE-ARRAY-BASE (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)))
                 ((EQ I LIMIT))
              (%ARRAY-WRITE (%ARRAY-READ BASE-ARRAY-BASE %THIN-CHAR-TYPENUMBER I)
                     NEW-BASE %FAT-CHAR-TYPENUMBER I))
          ;; Smash the new base into the array-header
          (UNINTERRUPTABLY
              (|replace| (ARRAY-HEADER BASE) |of| BASE-ARRAY |with| NEW-BASE)
              (|replace| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY |with| %FAT-CHAR-TYPENUMBER))
          ;; return the original array
          ARRAY)))
```

(%MAKE-TWOD-ARRAY

```
  (LAMBDA (TOTAL-SIZE DIMENSIONS ELEMENT-TYPE FATP READ-ONLY-P EXTENDABLE-P)
                                                                        ; Edited 18-Dec-86 17:49 by jop
    ;; Two-d arrays are only simple or extendable twod-arrays
    (LET ((BOUND0 (CAR DIMENSIONS))
          (BOUND1 (CADR DIMENSIONS))
          (TYPE-NUMBER (%CML-TYPE-TO-TYPENUMBER ELEMENT-TYPE FATP)))
         (|create| TWOD-ARRAY
                BASE _ (%MAKE-ARRAY-STORAGE TOTAL-SIZE TYPE-NUMBER)
                READ-ONLY-P _ READ-ONLY-P
                BIT-P _ (%BIT-TYPE-P TYPE-NUMBER)
                EXTENDABLE-P _ EXTENDABLE-P
                TYPE-NUMBER _ TYPE-NUMBER
                BOUND0 _ BOUND0
                BOUND1 _ BOUND1
                TOTAL-SIZE _ TOTAL-SIZE))))
```

(%TOTAL-SIZE

```
  (LAMBDA (DIMS)                                                        ; Edited 18-Dec-86 17:53 by jop
    (CL:DO ((DIM DIMS (CDR DIM))
            (PROD 1))
           ((NULL DIM)
            PROD)
        (SETQ PROD (CL:* (CAR DIM)
                         PROD)))))
```

(SHRINK-VECTOR

```
  (LAMBDA (VECTOR NEW-SIZE)                                             ; Edited 18-Dec-86 18:08 by jop
    (COND
       ((%VECTORP VECTOR)
        (CL:IF (OR (< NEW-SIZE 0)
                   (> NEW-SIZE (|fetch| (ARRAY-HEADER TOTAL-SIZE) |of| VECTOR)))
               (CL:ERROR "Trying to shrink array ~s to bad size ~s" VECTOR NEW-SIZE))
        (|replace| (ARRAY-HEADER FILL-POINTER-P) |of| VECTOR |with| T)
        (|replace| (ARRAY-HEADER FILL-POINTER) |of| VECTOR |with| NEW-SIZE)
        VECTOR)
       (T (CL:ERROR "Not a vector: ~S" VECTOR)))))

)

;; For Interlisp string hack

(DEFINEQ
```

(%SET-ARRAY-OFFSET

```
  (LAMBDA (ARRAY NEWVALUE)                                              ; Edited 18-Dec-86 17:51 by jop
    ;; Set the true offset for ARRAY
    (COND
       ((%ONED-ARRAY-P ARRAY)
        (|replace| (ARRAY-HEADER OFFSET) |of| ARRAY |with| NEWVALUE))
       ((%TWOD-ARRAY-P ARRAY)
        (CL:ERROR "Twod-arrays have no offset"))
       ((%GENERAL-ARRAY-P ARRAY)
        (|replace| (ARRAY-HEADER OFFSET) |of| ARRAY |with| (- NEWVALUE (CL:DO* ((BASE-ARRAY ARRAY (|fetch| (ARRAY-HEADER
                                                                                                    BASE)
                                                                                                    |of| BASE-ARRAY))
                                                                                        (OFFSET 0 (+ OFFSET (
                                                                                                    %GET-ARRAY-OFFSET
                                                                                                    BASE-ARRAY))))
                                                                                      ((NOT (|fetch| (ARRAY-HEADER INDIRECT-P)
                                                                                                |of| BASE-ARRAY))
                                                                                       OFFSET)))))
       (T (CL:ERROR "Not an array: ~S" ARRAY)))
    NEWVALUE))
```

```
(%SET-ARRAY-TYPE-NUMBER
  (LAMBDA (ARRAY NEWVALUE)                                        ; Edited 18-Dec-86 17:52 by jop
    ;; Set the true type-number for array
    (COND
       ((OR (%ONED-ARRAY-P ARRAY)
            (%TWOD-ARRAY-P ARRAY))
        (|replace| (ARRAY-HEADER TYPE-NUMBER) |of| ARRAY |with| NEWVALUE))
       ((%GENERAL-ARRAY-P ARRAY)
        (CL:DO ((BASE-ARRAY ARRAY (|fetch| (ARRAY-HEADER BASE) |of| BASE-ARRAY)))
               ((NOT (|fetch| (ARRAY-HEADER INDIRECT-P) |of| BASE-ARRAY))
                (|replace| (ARRAY-HEADER TYPE-NUMBER) |of| BASE-ARRAY |with| NEWVALUE))))
       (T (CL:ERROR "Not an array ~S" ARRAY)))
    NEWVALUE))
)
```

;; Low level predicates

```
(DEFINEQ

(%ONED-ARRAY-P
  (LAMBDA (ARRAY)                                                 ; Edited 18-Dec-86 17:49 by jop
    (EQ (NTYPX ARRAY)
        %ONED-ARRAY)))

(%TWOD-ARRAY-P
  (LAMBDA (ARRAY)                                                 ; Edited 18-Dec-86 17:53 by jop
    (EQ (NTYPX ARRAY)
        %TWOD-ARRAY)))

(%GENERAL-ARRAY-P
  (LAMBDA (ARRAY)                                                 ; Edited 18-Dec-86 17:44 by jop
    (EQ (NTYPX ARRAY)
        %GENERAL-ARRAY)))

(%THIN-STRING-ARRAY-P
  (LAMBDA (ARRAY)                                                 ; Edited 18-Dec-86 17:53 by jop
    (%THIN-CHAR-TYPE-P (%ARRAY-TYPE-NUMBER ARRAY))))
)

(DEFOPTIMIZER %ONED-ARRAY-P (ARRAY)
                            `(AND ((OPCODES TYPEP 14)
                                   ,ARRAY)
                                  T))

(DEFOPTIMIZER %TWOD-ARRAY-P (ARRAY)
                            `(AND ((OPCODES TYPEP 15)
                                   ,ARRAY)
                                  T))

(DEFOPTIMIZER %GENERAL-ARRAY-P (ARRAY)
                               `(AND ((OPCODES TYPEP 16)
                                      ,ARRAY)
                                     T))
```

;; Real record def's on cmlarray-support

```
(/DECLAREDATATYPE 'GENERAL-ARRAY '((BITS 4)
                                   POINTER FLAG 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)

(ADDTOVAR SYSTEMRECLST
          (DATATYPE GENERAL-ARRAY ((NIL BITS 4)
                                   (STORAGE POINTER)
                                   (READ-ONLY-P FLAG)
                                   (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)
                                   (OFFSET WORD)
                                   (FILL-POINTER FIXP)
                                   (TOTAL-SIZE FIXP)
                                   (DIMS POINTER)))
          (DATATYPE ONED-ARRAY ((NIL BITS 4)
                                (BASE POINTER)
                                (READ-ONLY-P FLAG)
                                (NIL BITS 1)
                                (BIT-P FLAG)
                                (STRING-P FLAG)
                                (NIL BITS 1)
                                (DISPLACED-P FLAG)
                                (FILL-POINTER-P FLAG)
                                (EXTENDABLE-P FLAG)
                                (TYPE-NUMBER BITS 8)
                                (OFFSET WORD)
                                (FILL-POINTER FIXP)
                                (TOTAL-SIZE FIXP)))
          (DATATYPE TWOD-ARRAY ((NIL BITS 4)
                                (BASE POINTER)
                                (READ-ONLY-P FLAG)
                                (NIL BITS 1)
                                (BIT-P FLAG)
                                (NIL BITS 4)
                                (EXTENDABLE-P FLAG)
                                (TYPE-NUMBER BITS 8)
                                (BOUND0 FIXP)
                                (BOUND1 FIXP)
                                (TOTAL-SIZE FIXP))))

(PUTPROPS %AREF1 DOPVAL (2 AREF1))

(PUTPROPS %AREF2 DOPVAL (3 AREF2))

(PUTPROPS %ASET1 DOPVAL (3 ASET1))

(PUTPROPS %ASET2 DOPVAL (4 ASET2))

;; I/O

(DEFINEQ

(%DEFPRINT-ARRAY
  (LAMBDA (ARRAY STREAM)                                      ; Edited  5-Feb-88 10:10 by jop

    ;; This is the defprint for the array type

    (COND
       ((%VECTORP ARRAY)
        (%DEFPRINT-VECTOR ARRAY STREAM))
       ((NOT *PRINT-ARRAY*)
        (%DEFPRINT-GENERIC-ARRAY ARRAY STREAM))
       ((AND *PRINT-LEVEL* (<= *PRINT-LEVEL* 0))
        (\\ELIDE.PRINT.ELEMENT STREAM)
        T)
       (T (LET ((HASH (CL:CODE-CHAR (|fetch| (READTABLEP HASHMACROCHAR) |of| *READTABLE*)))
                (RANK (CL:ARRAY-RANK ARRAY))
                RANKSTR)
               (%CHECK-CIRCLE-PRINT ARRAY STREAM (SETQ RANKSTR (CL:PRINC-TO-STRING RANK))
                                                                     ; Make sure we have room for #na
                     (.SPACECHECK. STREAM (+ (VECTOR-LENGTH RANKSTR)
                                             2))
                     (CL:WRITE-CHAR HASH STREAM)
                     (CL:WRITE-STRING RANKSTR STREAM)
                     (CL:WRITE-CHAR (CONSTANT #\A)
                            STREAM)
                     (CL:IF (EQ RANK 0)
                         (\\PRINDATUM (CL:AREF ARRAY)
                                STREAM 0)
                         (%PRINT-ARRAY-CONTENTS (%FLATTEN-ARRAY ARRAY)
                                0
                                (CL:ARRAY-DIMENSIONS ARRAY)
                                STREAM)))
              T)))))
```

(%DEFPRINT-BITVECTOR

```
  (LAMBDA (CL:BIT-VECTOR STREAM)                                     ; Edited 11-Dec-87 15:35 by jop

    ;; *Print-level* is handled in %defprint-vector

    (LET ((HASH (CL:CODE-CHAR (|fetch| (READTABLEP HASHMACROCHAR) |of| *READTABLE*)))
          (SIZE (VECTOR-LENGTH CL:BIT-VECTOR))
          END.INDEX FINAL.INDEX ELIDED SIZESTR)
         (SETQ END.INDEX (CL:1- SIZE))
         (%CHECK-CIRCLE-PRINT CL:BIT-VECTOR STREAM (CL:UNLESS (EQ SIZE 0)
                                                       (CL:DO ((I (CL:1- END.INDEX)
                                                                  (CL:1- I))
                                                               (LAST.VALUE (CL:AREF CL:BIT-VECTOR END.INDEX)))
                                                              ((OR (< I 0)
                                                                   (NOT (EQL (CL:AREF CL:BIT-VECTOR I)
                                                                             LAST.VALUE))))
                                                           (SETQ END.INDEX I)))
                (SETQ FINAL.INDEX (COND
                                      ((AND *PRINT-LENGTH* (>= END.INDEX *PRINT-LENGTH*))
                                       (SETQ ELIDED T)
                                       (CL:1- *PRINT-LENGTH*))
                                      (T END.INDEX)))
                (CL:IF (NOT (EQ (CL:1- SIZE)
                                END.INDEX))
                    (SETQ SIZESTR (CL:PRINC-TO-STRING SIZE)))
                (.SPACECHECK. STREAM (+ (PROGN                       ; #* Plus 1 for final.index being 1 less than number bits printed
                                             3)
                                        (CL:IF SIZESTR
                                            (VECTOR-LENGTH SIZESTR)
                                            0)
                                        FINAL.INDEX
                                        (CL:IF ELIDED
                                            (PROGN                   ; Space for ...
                                                   3)
                                            0)))
                (CL:WRITE-CHAR HASH STREAM)
                (CL:IF SIZESTR (CL:WRITE-STRING SIZESTR STREAM))
                (CL:WRITE-CHAR (CONSTANT #\*)
                       STREAM)
                (CL:DO ((I 0 (CL:1+ I)))
                       ((> I FINAL.INDEX))
                    (\\OUTCHAR STREAM (+ (BIT CL:BIT-VECTOR I)
                                         (CONSTANT (CL:CHAR-CODE #\0)))))
                (CL:IF ELIDED (\\ELIDE.PRINT.TAIL STREAM)))
         T)))
```

(%DEFPRINT-GENERIC-ARRAY

```
  (LAMBDA (ARRAY STREAM)                                             ; Edited 18-Dec-86 17:40 by jop

    ;; Invoked when *PRINT-ARRAY* is NIL

    (LET ((HASH (CL:CODE-CHAR (|fetch| (READTABLEP HASHMACROCHAR) |of| *READTABLE*))))
         (%CHECK-CIRCLE-PRINT ARRAY STREAM                           ; Make sure we have room for #<
                (.SPACECHECK. STREAM 2)
                (CL:WRITE-CHAR HASH STREAM)
                (CL:WRITE-CHAR (CONSTANT #\<)
                       STREAM)
                (CL:WRITE-STRING (CL:PRINC-TO-STRING 'CL:ARRAY)
                       STREAM)
                (CL:WRITE-CHAR (CONSTANT #\Space)
                       STREAM)
                (CL:WRITE-STRING (CL:PRINC-TO-STRING (CL:ARRAY-ELEMENT-TYPE ARRAY))
                       STREAM)
                (CL:WRITE-CHAR (CONSTANT #\Space)
                       STREAM)
                (CL:WRITE-STRING (CL:PRINC-TO-STRING (CL:ARRAY-DIMENSIONS ARRAY))
                       STREAM)
                (CL:WRITE-CHAR (CONSTANT #\Space)
                       STREAM)
                (CL:WRITE-CHAR (CONSTANT #\@)
                       STREAM)
                (CL:WRITE-CHAR (CONSTANT #\Space)
                       STREAM)
                (\\PRINTADDR ARRAY STREAM)
                (CL:WRITE-CHAR (CONSTANT #\>)
                       STREAM))
         T)))
```

(**%DEFPRINT-VECTOR**

```
  (LAMBDA (VECTOR STREAM)                                              ; Edited 5-Feb-88 10:11 by jop

     ;; Defprint for the oned-array type

     (COND
        ((CL:STRINGP VECTOR)
         (%DEFPRINT-STRING VECTOR STREAM))
        ((NOT *PRINT-ARRAY*)
         (%DEFPRINT-GENERIC-ARRAY VECTOR STREAM))
        ((AND *PRINT-LEVEL* (<= *PRINT-LEVEL* 0))
         (\\ELIDE.PRINT.ELEMENT STREAM)
         T)
        ((CL:BIT-VECTOR-P VECTOR)
         (%DEFPRINT-BITVECTOR VECTOR STREAM))
        (T (LET ((HASH (CL:CODE-CHAR (|fetch| (READTABLEP HASHMACROCHAR) |of| *READTABLE*)))
                 (SIZE (VECTOR-LENGTH VECTOR))
                 END.INDEX FINAL.INDEX ELIDED SIZESTR)
                (SETQ END.INDEX (CL:1- SIZE))
                (%CHECK-CIRCLE-PRINT VECTOR STREAM (CL:UNLESS (EQ SIZE 0)
                                                       (CL:DO ((I (CL:1- END.INDEX)
                                                                  (CL:1- I))
                                                               (LAST.VALUE (CL:AREF VECTOR END.INDEX)))
                                                              ((OR (< I 0)
                                                                   (NOT (EQL (CL:AREF VECTOR I)
                                                                             LAST.VALUE))))
                                                           (SETQ END.INDEX I)))
                       (SETQ FINAL.INDEX (COND
                                            ((AND *PRINT-LENGTH* (>= END.INDEX *PRINT-LENGTH*))
                                             (SETQ ELIDED T)
                                             (CL:1- *PRINT-LENGTH*))
                                            (T END.INDEX)))
                       (CL:IF (NOT (EQ (CL:1- SIZE)
                                       END.INDEX))
                           (SETQ SIZESTR (CL:PRINC-TO-STRING SIZE)))
                       (.SPACECHECK. STREAM (+ (CL:IF SIZESTR
                                                   (VECTOR-LENGTH SIZESTR)
                                                   0)
                                               2))
                       (CL:WRITE-CHAR HASH STREAM)
                       (CL:IF SIZESTR (CL:WRITE-STRING SIZESTR STREAM))
                       (CL:WRITE-CHAR (CONSTANT #\()
                              STREAM)
                       (LET ((*PRINT-LEVEL* (AND *PRINT-LEVEL* (CL:1- *PRINT-LEVEL*))))
                            (CL:DO ((I 0 (CL:1+ I)))
                                   ((> I FINAL.INDEX))
                                (CL:IF (> I 0)
                                    (CL:WRITE-CHAR (CONSTANT #\Space)
                                           STREAM))
                                (\\PRINDATUM (CL:AREF VECTOR I)
                                       STREAM 0)))
                       (CL:IF ELIDED (\\ELIDE.PRINT.TAIL STREAM))
                       (CL:WRITE-CHAR (CONSTANT #\))
                              STREAM))
                T)))))
```

(**%DEFPRINT-STRING**

```
  (LAMBDA (STRING STREAM)                                              ; Edited 11-Dec-87 15:36 by jop

     ;; May never get called since (IL:typename (make-string 10)) returns IL:stringp

     (LET ((ESCAPECHAR (|fetch| (READTABLEP ESCAPECHAR) |of| *READTABLE*))
           (CLP (|fetch| (READTABLEP COMMONLISP) |of| *READTABLE*))
           (SIZE (VECTOR-LENGTH STRING)))
          (%CHECK-CIRCLE-PRINT STRING STREAM (.SPACECHECK. STREAM (CL:IF CLP
                                                                          2
                                                                          (+ 2 SIZE)))
                 (CL:WHEN *PRINT-ESCAPE*
                     (\\OUTCHAR STREAM (CONSTANT (CL:CHAR-CODE #\"))))
                 (CL:DO ((I 0 (CL:1+ I))
                         CH)
                        ((EQ I SIZE))
                     (SETQ CH (CL:CHAR-CODE (CL:CHAR STRING I)))
                     (CL:WHEN (AND *PRINT-ESCAPE* (OR (EQ CH (CONSTANT (CL:CHAR-CODE #\")))
                                                      (EQ CH ESCAPECHAR)))
                            (\\OUTCHAR STREAM ESCAPECHAR))
                     (\\OUTCHAR STREAM CH))
                 (CL:WHEN *PRINT-ESCAPE*
                     (\\OUTCHAR STREAM (CONSTANT (CL:CHAR-CODE #\")))))
          T)))
```

```
(%PRINT-ARRAY-CONTENTS
  (LAMBDA (FLAT-ARRAY OFFSET DIMENSIONS STREAM)                    ; Edited  5-Feb-88 10:11 by jop
    (LET ((NELTS (CAR DIMENSIONS))
          FINAL.INDEX ELIDED)
         (COND
            ((AND *PRINT-LENGTH* (> NELTS *PRINT-LENGTH*))
             (SETQ ELIDED T)
             (SETQ FINAL.INDEX (CL:1- *PRINT-LENGTH*)))
            (T (SETQ FINAL.INDEX (CL:1- NELTS))))
         (CL:WRITE-CHAR (CONSTANT #\()
                STREAM)
         (COND
            ((NULL (CDR DIMENSIONS))                               ; Down to bottom level, print the elements
             (CL:DO ((I OFFSET (CL:1+ I))
                     (END-INDEX (+ OFFSET FINAL.INDEX)))
                    ((> I END-INDEX))
                 (CL:IF (> I OFFSET)
                     (CL:WRITE-CHAR (CONSTANT #\Space)
                            STREAM))
                 (\\PRINDATUM (CL:AREF FLAT-ARRAY I)
                        STREAM 0)))
            ((EQ *PRINT-LEVEL* 1)                                  ; Elide at this level
             (CL:DO ((I 0 (CL:1+ I)))
                    ((> I FINAL.INDEX))
                 (CL:IF (> I OFFSET)
                     (CL:WRITE-CHAR (CONSTANT #\Space)
                            STREAM))
                 (\\ELIDE.PRINT.ELEMENT STREAM)))
            (T (LET ((*PRINT-LEVEL* (AND *PRINT-LEVEL* (CL:1- *PRINT-LEVEL*))))
                    (CL:DO ((I 0 (CL:1+ I)))
                           ((> I FINAL.INDEX))
                        (CL:IF (> I 0)
                            (CL:WRITE-CHAR (CONSTANT #\Space)
                                   STREAM))
                        (%PRINT-ARRAY-CONTENTS FLAT-ARRAY (CL:* (CADR DIMENSIONS)
                                                                (+ OFFSET I))
                               (CDR DIMENSIONS)
                               STREAM)))))
         (CL:IF ELIDED (\\ELIDE.PRINT.TAIL STREAM))
         (CL:WRITE-CHAR (CONSTANT #\))
                STREAM))))

)

(DEFPRINT 'ONED-ARRAY '%DEFPRINT-VECTOR)

(DEFPRINT 'TWOD-ARRAY '%DEFPRINT-ARRAY)

(DEFPRINT 'GENERAL-ARRAY '%DEFPRINT-ARRAY)
```

;; Needed at run time. low level functions for accessing, setting, and allocating raw storage. also includes cml type to typenumber converters

```
(DEFINEQ

(%ARRAY-READ
  (LAMBDA (BASE TYPE-NUMBER INDEX)
    (%SLOW-ARRAY-READ BASE TYPE-NUMBER INDEX)))

(%ARRAY-WRITE
  (LAMBDA (NEWVALUE BASE TYPE-NUMBER INDEX)                        ; Edited 18-Dec-86 17:23 by jop
    (%SLOW-ARRAY-WRITE NEWVALUE BASE TYPE-NUMBER INDEX)))

(%CML-TYPE-TO-TYPENUMBER
  (LAMBDA (ELEMENT-TYPE FATP)                                      ; Edited 18-Dec-86 17:30 by jop
    (LET ((CANONICAL-TYPE (%GET-CANONICAL-CML-TYPE ELEMENT-TYPE)))
         (CL:IF (AND FATP (EQ CANONICAL-TYPE 'CL:STRING-CHAR))
             %FAT-CHAR-TYPENUMBER
             (%CML-TYPE-TO-TYPENUMBER-EXPANDER CANONICAL-TYPE)))))

(%GET-CANONICAL-CML-TYPE
  (LAMBDA (ELEMENT-TYPE)                                           ; Edited 18-Dec-86 17:46 by jop

    ;; Returns the enclosing specialized array type

    (CL:IF (CL:CONSP ELEMENT-TYPE)
        (CASE (CAR ELEMENT-TYPE)
            (CL:UNSIGNED-BYTE (%GET-ENCLOSING-UNSIGNED-BYTE ELEMENT-TYPE))
            (CL:SIGNED-BYTE (%GET-ENCLOSING-SIGNED-BYTE ELEMENT-TYPE))
            (CL:MOD (%REDUCE-MOD ELEMENT-TYPE))
            (INTEGER (%REDUCE-INTEGER ELEMENT-TYPE))
            (T (LET ((EXPANDER (TYPE-EXPANDER (CAR ELEMENT-TYPE))))
                    (CL:IF EXPANDER
                        (%GET-CANONICAL-CML-TYPE (TYPE-EXPAND ELEMENT-TYPE EXPANDER))
                        T))))
        (CASE ELEMENT-TYPE
            ((T XPOINTER CL:SINGLE-FLOAT CL:STRING-CHAR) ELEMENT-TYPE)
            (POINTER T)
            (FLOAT 'CL:SINGLE-FLOAT)
            (CL:FIXNUM '(CL:SIGNED-BYTE 32))
            (CL:CHARACTER 'CL:STRING-CHAR)
            (BIT '(CL:UNSIGNED-BYTE 1))
            (T (LET ((EXPANDER (TYPE-EXPANDER ELEMENT-TYPE)))
                    (CL:IF EXPANDER
                        (%GET-CANONICAL-CML-TYPE (TYPE-EXPAND ELEMENT-TYPE EXPANDER))
                        T)))))))
```

(%GET-ENCLOSING-SIGNED-BYTE

```
  (LAMBDA (ELEMENT-TYPE)                                    ; Edited  8-May-88 15:21 by jop
    (LET ((NBITS (CADR ELEMENT-TYPE)))
         (CL:IF (CL:INTEGERP NBITS)
             (COND
                ((<= NBITS 16)
                 '(CL:SIGNED-BYTE 16))
                ((<= NBITS 32)
                 '(CL:SIGNED-BYTE 32))
                (T T))
             T))))
```

(%GET-ENCLOSING-UNSIGNED-BYTE

```
  (LAMBDA (ELEMENT-TYPE)                                    ; Edited  8-May-88 15:21 by jop
    (LET ((NBITS (CADR ELEMENT-TYPE)))
         (CL:IF (CL:INTEGERP NBITS)
             (COND
                ((<= NBITS 1)
                 '(CL:UNSIGNED-BYTE 1))
                ((<= NBITS 8)
                 '(CL:UNSIGNED-BYTE 8))
                ((<= NBITS 16)
                 '(CL:UNSIGNED-BYTE 16))
                (T T))
             T))))
```

(%MAKE-ARRAY-STORAGE

```
  (LAMBDA (NELTS TYPENUMBER INIT-ON-PAGE ALIGNMENT)         ; Edited 18-Dec-86 17:47 by jop

    ;; Allocates a raw storage block for an array of NELTS elements, of type TYPENUMBER

    (LET ((BITS-PER-ELEMENT (%TYPENUMBER-TO-BITS-PER-ELEMENT TYPENUMBER))
          (GC-TYPE (%TYPENUMBER-TO-GC-TYPE TYPENUMBER)))
         (\\ALLOCBLOCK (FOLDHI (CL:* NELTS BITS-PER-ELEMENT)
                              BITSPERCELL)
                GC-TYPE INIT-ON-PAGE ALIGNMENT))))
```

(%REDUCE-INTEGER

```
  (LAMBDA (ELEMENT-TYPE)                                    ; Edited  8-May-88 15:27 by jop
    (LET ((LOW (CADR ELEMENT-TYPE))
          (HIGH (CADDR ELEMENT-TYPE)))
         (CL:IF (CL:CONSP LOW)
             (SETQ LOW (CL:1+ (CAR LOW))))
         (CL:IF (CL:CONSP HIGH)
             (SETQ HIGH (CL:1- (CAR HIGH))))
         (CL:IF (AND (CL:INTEGERP LOW)
                     (CL:INTEGERP HIGH))
             (CL:IF (>= LOW 0)
                 (COND
                    ((< HIGH 2)
                     '(CL:UNSIGNED-BYTE 1))
                    ((< HIGH 256)
                     '(CL:UNSIGNED-BYTE 8))
                    ((< HIGH 65536)
                     '(CL:UNSIGNED-BYTE 16))
                    (T T))
                 (LET ((BOUND (MAX (- LOW)
                                   HIGH)))
                      (COND
                         ((< BOUND 32768)
                          '(CL:SIGNED-BYTE 16))
                         ((<= BOUND MAX.FIXP)
                          '(CL:SIGNED-BYTE 32))
                         (T T))))
             T))))
```

(%REDUCE-MOD

```
  (LAMBDA (ELEMENT-TYPE)                                    ; Edited  8-May-88 15:22 by jop
    (LET ((MODNUM (CADR ELEMENT-TYPE)))
         (CL:IF (CL:INTEGERP MODNUM)
              (COND
                 ((<= MODNUM 2)
                  '(CL:UNSIGNED-BYTE 1))
                 ((<= MODNUM 256)
                  '(CL:UNSIGNED-BYTE 8))
                 ((<= MODNUM 65536)
                  '(CL:UNSIGNED-BYTE 16))
                 (T T))
             T))))
```

(**%SLOW-ARRAY-READ**

```
  (LAMBDA (BASE TYPENUMBER ROW-MAJOR-INDEX)                     ; Edited 18-Dec-86 17:52 by jop
```

;; Punt function for opcode arrayread

```
    (%LLARRAY-TYPED-GET BASE TYPENUMBER ROW-MAJOR-INDEX)))
```

(**%SLOW-ARRAY-WRITE**

```
  (LAMBDA (NEWVALUE BASE TYPENUMBER ROW-MAJOR-INDEX)            ; Edited 18-Dec-86 17:53 by jop
```

;; Punt function for opcode arraywrite

```
    (CL:IF (NOT (%LLARRAY-TYPEP TYPENUMBER NEWVALUE))
        (CL:ERROR "Illegal value: ~S" NEWVALUE)
        (%LLARRAY-TYPED-PUT BASE TYPENUMBER ROW-MAJOR-INDEX NEWVALUE))
    NEWVALUE))

)


(DEFOPTIMIZER %ARRAY-READ (BASE TYPENUMBER INDEX)
                           `((OPCODES MISC3 9)
                             ,BASE
                             ,TYPENUMBER
                             ,INDEX))


(DEFOPTIMIZER %ARRAY-WRITE (NEWVALUE BASE TYPENUMBER INDEX)
                            `((OPCODES MISC4 7)
                              ,NEWVALUE
                              ,BASE
                              ,TYPENUMBER
                              ,INDEX))
```

;; Compiler options

```
(DECLARE\: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY

(DECLARE\: DOEVAL@COMPILE DONTCOPY

(LOCALVARS . T)
)
)

(PUTPROPS CMLARRAY FILETYPE CL:COMPILE-FILE)

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

(ADDTOVAR NLAMA )

(ADDTOVAR NLAML )

(ADDTOVAR LAMA CL:VECTOR ASET CL:ARRAY-ROW-MAJOR-INDEX CL:ARRAY-IN-BOUNDS-P CL:AREF)
)

(PUTPROPS CMLARRAY COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1988 1990 1992 1993))
```

FUNCTION INDEX

%ALTER-AS-DISPLACED-ARRAY 13
%ALTER-AS-DISPLACED-TO-BASE-ARRAY 13
%AREF-EXPANDER 12
%AREF0 13
%AREF1 13
%AREF2 14
%ARRAY-BASE 14
%ARRAY-CONTENT-INITIALIZE 14
%ARRAY-ELEMENT-INITIALIZE 14
%ARRAY-OFFSET 14
%ARRAY-READ 25
%ARRAY-TYPE-NUMBER 15
%ARRAY-WRITE 25
%ASET-EXPANDER 12
%ASET0 15
%ASET1 15
%ASET2 15
%CHECK-SEQUENCE-DIMENSIONS 16
%CML-TYPE-TO-TYPENUMBER 25
%COPY-TO-NEW-ARRAY 16
%DEFPRINT-ARRAY 22
%DEFPRINT-BITVECTOR 23
%DEFPRINT-GENERIC-ARRAY 23
%DEFPRINT-STRING 24
%DEFPRINT-VECTOR 24
%DO-LOGICAL-OP 16
%EXTEND-ARRAY 17
%FAST-COPY-BASE 17
%FAT-STRING-ARRAY-P 18
%FILL-ARRAY-FROM-SEQUENCE 18
%FLATTEN-ARRAY 18
%GENERAL-ARRAY-P 21
%GET-CANONICAL-CML-TYPE 25
%GET-ENCLOSING-SIGNED-BYTE 26
%GET-ENCLOSING-UNSIGNED-BYTE 26
%MAKE-ARRAY-STORAGE 26
%MAKE-ARRAY-WRITEABLE 18
%MAKE-DISPLACED-ARRAY 18
%MAKE-GENERAL-ARRAY 19
%MAKE-ONED-ARRAY 19
%MAKE-STRING-ARRAY-FAT 19
%MAKE-TWOD-ARRAY 20
%ONED-ARRAY-P 21
%PRINT-ARRAY-CONTENTS 25
%REDUCE-INTEGER 26
%REDUCE-MOD 26
%SET-ARRAY-OFFSET 20
%SET-ARRAY-TYPE-NUMBER 21
%SLOW-ARRAY-READ 27
%SLOW-ARRAY-WRITE 27
%THIN-STRING-ARRAY-P 21
%TOTAL-SIZE 20
%TWOD-ARRAY-P 21
CL:ADJUST-ARRAY 2
CL:ADJUSTABLE-ARRAY-P 3
CL:AREF 9
CL:ARRAY-DIMENSION 3
CL:ARRAY-DIMENSIONS 3
CL:ARRAY-ELEMENT-TYPE 3
CL:ARRAY-HAS-FILL-POINTER-P 3
CL:ARRAY-IN-BOUNDS-P 10
ARRAY-NEEDS-INDIRECTION-P 3
CL:ARRAY-RANK 4
CL:ARRAY-ROW-MAJOR-INDEX 10
CL:ARRAY-TOTAL-SIZE 4
CL:ARRAYP 5
ASET 10
BIT 4
CL:BIT-AND 4
CL:BIT-ANDC1 4
CL:BIT-ANDC2 4
BIT-ARRAY-P 4
CL:BIT-EQV 4
CL:BIT-IOR 4
CL:BIT-NAND 4
CL:BIT-NOR 4
CL:BIT-NOT 4
CL:BIT-ORC1 4
CL:BIT-ORC2 4
CL:BIT-VECTOR-P 4
CL:BIT-XOR 5
CL:CHAR 5
COPY-ARRAY 5
COPY-VECTOR 5
DISPLACED-ARRAY-P 5
EQUAL-DIMENSIONS-P 6
EXTENDABLE-ARRAY-P 6
FILL-ARRAY 6
CL:FILL-POINTER 7
FILL-VECTOR 7
CL:MAKE-ARRAY 7
MAKE-VECTOR 8
READ-ONLY-ARRAY-P 8
XCL:ROW-MAJOR-AREF 11
CL::ROW-MAJOR-ASET 11
CL:SBIT 8
CL:SCHAR 8
SET-FILL-POINTER 8
SHRINK-VECTOR 20
SIMPLE-ARRAY-P 8
CL:SIMPLE-BIT-VECTOR-P 9
CL:SIMPLE-STRING-P 9
CL:SIMPLE-VECTOR-P 9
STRING-ARRAY-P 9
CL:STRINGP 5
CL:SVREF 9
CL:VECTOR 11
VECTOR-LENGTH 9
CL:VECTOR-POP 9
CL:VECTOR-PUSH 9
CL:VECTOR-PUSH-EXTEND 9
CL:VECTORP 9

OPTIMIZER INDEX

%ARRAY-READ 27
%ARRAY-WRITE 27
%GENERAL-ARRAY-P 21
%ONED-ARRAY-P 21
%TWOD-ARRAY-P 21
CL:AREF 12
ASET 12
BIT 12
CL:CHAR 12
CL:SBIT 12
CL:SCHAR 12
CL:SVREF 12

SETF INDEX

CL:AREF 12
BIT 12
CL:CHAR 12
CL:FILL-POINTER 12
XCL:ROW-MAJOR-AREF 11
CL:SBIT 12
CL:SCHAR 12
CL:SVREF 12

PROPERTY INDEX

%AREF1 22
%AREF2 22
%ASET1 22
%ASET2 22
CMLARRAY 27

CONSTANT INDEX

CL:ARRAY-DIMENSION-LIMIT 12
CL:ARRAY-RANK-LIMIT 12
CL:ARRAY-TOTAL-SIZE-LIMIT 12

VARIABLE INDEX

DEFAULT-PUSH-EXTENSION-SIZE ...13
SYSTEMRECLST 22