File created: 10-Aug-92 13:17:53 **{PELE:MV:ENVOS}<LISPCORE>SOURCES>XCLC-TRANSFORMS.;3**

changes to: (IL:FUNCTIONS FIND-AND-PERFORM-RPLCONS-TRANSFORM)

previous date: 23-May-90 13:25:49 {PELE:MV:ENVOS}<LISPCORE>SOURCES>XCLC-TRANSFORMS.;2

Read Table: XCL

Package: COMPILER

Format: XCCS


```
(IL:RPAQQ IL:XCLC-TRANSFORMSCOMS
          (;; Function-specific transformations
           (IL:DEFINE-TYPES TRANSFORMS)
           (IL:FUNCTIONS DEFTRANSFORM)
           (IL:PROP IL:PROPTYPE TRANSFORM)
           (IL:FUNCTIONS IS-CALL-TO)
           ;; The various memory-reference primitives.
           (TRANSFORMS IL:\\ADDBASE)
           (TRANSFORMS IL:\\GETBASE IL:\\GETBASEPTR)
           (TRANSFORMS IL:\\PUTBASE IL:\\PUTBASEPTR IL:\\RPLPTR)
           (OPTIMIZERS IL:\\PUTBASE IL:\\PUTBASEPTR IL:\\RPLPTR)
           (TRANSFORMS IL:\\GETBITS IL:\\PUTBITS)
           (OPTIMIZERS IL:\\GETBITS IL:\\PUTBITS)
           (IL:FUNCTIONS ENSURE-EFFECT-CONTEXT TRANSFORM-GET/PUT-BASE)
           ;; List-structure functions
           (TRANSFORMS CAR CDR)
           (TRANSFORMS RPLACD)
           (IL:FUNCTIONS FIND-AND-PERFORM-RPLCONS-TRANSFORM)
           (OPTIMIZERS IL:FRPLACD)
           ;; Use the proper makefile-environment
           (IL:PROP IL:MAKEFILE-ENVIRONMENT IL:XCLC-TRANSFORMS)
           ;; Use the proper compiler.
           (IL:PROP IL:FILETYPE IL:XCLC-TRANSFORMS)))
```

;; Function-specific transformations

```
(DEF-DEFINE-TYPE TRANSFORMS "XCL Compiler transformations"
   :UNDEFINER (LAMBDA (NAME)
                      (WHEN (SYMBOLP NAME)
                          (REMPROP NAME 'TRANSFORM))))
```

```
(DEFDEFINER DEFTRANSFORM TRANSFORMS (FN-NAME ARG-LIST &BODY CODE)
```

;;; Transforms are called with two arguments: (1) the subtree of the program whose root is a CALL node with FN-NAME as the function, and (2) the
;;; evaluation context of the subtree, one of :ARGUMENT, :EFFECT, :RETURN, and :MV. The transform should return the new subtree and should, of
;;; course, be careful about releasing any nodes it does away with.

;;; Don't forget to set the META-P bit in any new nodes created and in the root node, if no more work is needed.

```
   (LET ((TRANSFORM-NAME (INTERN (CONCATENATE 'STRING "transform-" (STRING FN-NAME))
                                 (SYMBOL-PACKAGE FN-NAME))))
        `(PROGN (DEFUN ,TRANSFORM-NAME ,ARG-LIST
                   ,@CODE)
                (SETF (GET ',FN-NAME 'TRANSFORM)
                      ',TRANSFORM-NAME))))
```

```
(IL:PUTPROPS TRANSFORM IL:PROPTYPE IGNORE)
```

```
(DEFUN IS-CALL-TO (NAME NODE)
```

;;; Is NODE a function call to the global function named NAME?

```
   (AND (CALL-P NODE)
        (LET ((FN (CALL-FN NODE)))
             (AND (VAR-REF-P FN)
                  (LET ((VAR (VAR-REF-VARIABLE FN)))
                       (EQ :GLOBAL (VARIABLE-SCOPE VAR))
                       (EQ :FUNCTION (VARIABLE-KIND VAR))
                       (EQ NAME (VARIABLE-NAME VAR)))))))
```

;; The various memory-reference primitives.

```
(DEFTRANSFORM IL:\\ADDBASE (NODE CONTEXT)
```

;;; Get rid of nested \ADDBASE's. (\addbase (\addbase base n) m) => (\addbase base (+ n m)).

```
   (DESTRUCTURING-BIND (BASE OFFSET)
          (CALL-ARGS NODE)
          (WHEN (AND (IS-CALL-TO 'IL:\\ADDBASE BASE)
                     (LITERAL-P OFFSET)
                     (INTEGERP (LITERAL-VALUE OFFSET)))
              (DESTRUCTURING-BIND (INNER-BASE INNER-OFFSET)
                     (CALL-ARGS BASE)
                     (WHEN (AND (LITERAL-P INNER-OFFSET)
                                (INTEGERP (LITERAL-VALUE INNER-OFFSET)))
                         (LET ((NEW-OFFSET (MAKE-LITERAL :VALUE (+ (LITERAL-VALUE OFFSET)
                                                                   (LITERAL-VALUE INNER-OFFSET)))))
                              (RELEASE-TREE INNER-OFFSET)
                              (SETF (SECOND (CALL-ARGS BASE))
                                    NEW-OFFSET)
                              (POP (CALL-ARGS NODE))            ; Detach the inner \addbase call.
                              (RELEASE-TREE NODE)
                              (SETQ NODE BASE))))))
   (SETF (NODE-META-P NODE)
         CONTEXT)
   NODE)


(DEFTRANSFORM IL:\\GETBASE (NODE CONTEXT)
   (TRANSFORM-GET/PUT-BASE NODE CONTEXT :GET 'IL:GETBASE.N))


(DEFTRANSFORM IL:\\GETBASEPTR (NODE CONTEXT)
   (TRANSFORM-GET/PUT-BASE NODE CONTEXT :GET 'IL:GETBASEPTR.N))


(DEFTRANSFORM IL:\\PUTBASE (NODE CONTEXT)
   (COND
      ((EQ CONTEXT :EFFECT)
       (TRANSFORM-GET/PUT-BASE NODE :EFFECT :PUT 'IL:PUTBASE.N))
                                                                ; See transform for \RPLPTR.
      (T (SETF (NODE-META-P NODE)
               CONTEXT)
         NODE)))


(DEFTRANSFORM IL:\\PUTBASEPTR (NODE CONTEXT)
   (COND
      ((EQ CONTEXT :EFFECT)
       (TRANSFORM-GET/PUT-BASE NODE :EFFECT :PUT 'IL:PUTBASEPTR.N))
                                                                ; See transform for \RPLPTR.
      (T (SETF (NODE-META-P NODE)
               CONTEXT)
         NODE)))


(DEFTRANSFORM IL:\\RPLPTR (NODE CONTEXT)
   (COND
      ((EQ CONTEXT :EFFECT)
       (TRANSFORM-GET/PUT-BASE NODE :EFFECT :PUT 'IL:RPLPTR.N))
                                                                ; Sometimes we are meta-eval'ed in a less-precise context.  Let
                                                                ; it go; we'll be re-meta-eval'ed correctly in a moment.
      (T (SETF (NODE-META-P NODE)
               CONTEXT)
         NODE)))


(DEFOPTIMIZER IL:\\PUTBASE (&WHOLE FORM &CONTEXT CTXT)
                           (ENSURE-EFFECT-CONTEXT FORM CTXT 2))


(DEFOPTIMIZER IL:\\PUTBASEPTR (&WHOLE FORM &CONTEXT CTXT)
                              (ENSURE-EFFECT-CONTEXT FORM CTXT 2))


(DEFOPTIMIZER IL:\\RPLPTR (&WHOLE FORM &CONTEXT CTXT)
                          (ENSURE-EFFECT-CONTEXT FORM CTXT 2))


(DEFTRANSFORM IL:\\GETBITS (NODE CONTEXT)
   ;; Splice out the field-descriptor and pass it as a beta byte
   (LET ((FD-NODE (THIRD (CALL-ARGS NODE))))
        (ASSERT (AND (LITERAL-P FD-NODE)
                     (INTEGERP (LITERAL-VALUE FD-NODE)))
                NIL "BUG: Field-descriptor for \\getbits is not a literal integer.")
        (LET ((FD (LITERAL-VALUE FD-NODE)))
             (RELEASE-TREE FD-NODE)
             (SETF (CDDR (CALL-ARGS NODE))
                   NIL)
             (TRANSFORM-GET/PUT-BASE NODE CONTEXT :GET 'IL:GETBITS.N.FD FD))))
```

```
(DEFTRANSFORM IL:\\PUTBITS (NODE CONTEXT)
   (COND
      ((EQ CONTEXT :EFFECT)
       ;; Splice out the field-descriptor and pass it as a beta byte
       (LET ((FD-NODE (THIRD (CALL-ARGS NODE))))
            (ASSERT (AND (LITERAL-P FD-NODE)
                         (INTEGERP (LITERAL-VALUE FD-NODE)))
                    NIL "BUG: Field-descriptor for \\putbits is not a literal integer.")
            (LET ((FD (LITERAL-VALUE FD-NODE)))
                 (RELEASE-TREE FD-NODE)
                 (SETF (CDDR (CALL-ARGS NODE))
                       (CDDDR (CALL-ARGS NODE)))
                 (TRANSFORM-GET/PUT-BASE NODE :EFFECT :PUT 'IL:PUTBITS.N.FD FD))))
                                                                  ; See transform for \RPLPTR
      (T (SETF (NODE-META-P NODE)
               CONTEXT)
         NODE)))

(DEFOPTIMIZER IL:\\GETBITS (BASE OFFSET FIELD-DESCRIPTOR &WHOLE FORM)
                           (ASSERT (AND (INTEGERP OFFSET)
                                        (INTEGERP FIELD-DESCRIPTOR))
                                   NIL "BUG: The second and third arguments to \\GETBITS must be literal
                                   integers.")
                           (IF (= FIELD-DESCRIPTOR 15)          ; Silly case; wants whole word.
                               `(IL:\\GETBASE ,BASE ,OFFSET)
                               FORM))

(DEFOPTIMIZER IL:\\PUTBITS (BASE OFFSET FIELD-DESCRIPTOR NEW-VALUE &WHOLE FORM &CONTEXT CTXT)
                           (ASSERT (AND (INTEGERP OFFSET)
                                        (INTEGERP FIELD-DESCRIPTOR))
                                   NIL "BUG: The second and third arguments to \\PUTBITS must be literal
                                   integers.")
                           (IF (= FIELD-DESCRIPTOR 15)          ; Silly case; wants whole word.
                               `(IL:\\PUTBASE ,BASE ,OFFSET ,NEW-VALUE)
                               (ENSURE-EFFECT-CONTEXT FORM CTXT 3 '(1 2))))

(DEFUN ENSURE-EFFECT-CONTEXT (FORM CTXT RESULT-ARG-NUMBER &OPTIONAL SUBST-INDICES)

;;; If the form is not in effect context already, then wrap it in an OPENLAMBDA, returning the RESULT-ARG-NUMBER'th argument as the value.  This
;;; way, the form will always be in effect context.  SUBST-INDICES is a list of the indices of arguments whose values should be substituted in the actual
;;; call directly, not passing through the argument list.

   (IF (EQL 0 (CONTEXT-VALUES-USED CTXT))
       'PASS
       (LET (CALL-ARGS LAMBDA-PARAMS LAMBDA-ARGS RESULT-ARG)
            (IL:FOR A IL:IN (CDR FORM) IL:AS N IL:FROM 0 IL:DO (COND
                                                                  ((MEMBER N SUBST-INDICES)
                                                                   (PUSH A CALL-ARGS)
                                                                   (WHEN (= N RESULT-ARG-NUMBER)
                                                                         (SETQ RESULT-ARG A)))
                                                                  (T (LET ((NAME (IL:PACK* 'ARG- N)))
                                                                          (PUSH NAME CALL-ARGS)
                                                                          (PUSH NAME LAMBDA-PARAMS)
                                                                          (PUSH A LAMBDA-ARGS)
                                                                          (WHEN (= N RESULT-ARG-NUMBER)
                                                                                (SETQ RESULT-ARG NAME))))))
            `((IL:OPENLAMBDA ,(REVERSE LAMBDA-PARAMS)
                (,(CAR FORM)
                 ,@(REVERSE CALL-ARGS))
                ,RESULT-ARG)
              ,@(REVERSE LAMBDA-ARGS)))))

(DEFUN TRANSFORM-GET/PUT-BASE (NODE CONTEXT USE OPCODE &OPTIONAL BETA-BYTE)

;;; Transform a call on one of the memory-accessing functions \{GET,PUT}BASE{,PTR,FIXP} or \RPLPTR into the appropriate calls on \ADDBASE and
;;; the given OPCODE.  USE is one of :GET or :PUT.

;;; The following transformations take place here:

    ;; (fn (\ADDBASE base n) offset [new-value]) => (fn base (+ n offset) [new-value])

    ;; Check for a literal OFFSET between 0 and 255 and avoid an \ADDBASE call in that case.

    (LET* ((ARGS (CALL-ARGS NODE))
           (LITERAL-OFFSET-VALUE 0)
           (COMPUTED-OFFSET-TREE NIL)
           (ADDBASE-CALL NIL)
           (REAL-BASE (FIRST ARGS)))
          (WHEN (IS-CALL-TO 'IL:\\ADDBASE REAL-BASE)
              (SETQ ADDBASE-CALL REAL-BASE)
              (DESTRUCTURING-BIND (BASE OFFSET)
                     (CALL-ARGS ADDBASE-CALL)
                     (SETQ REAL-BASE BASE)
```

```
                    (COND
                       ((AND (LITERAL-P OFFSET)
                             (INTEGERP (LITERAL-VALUE OFFSET)))
                        (INCF LITERAL-OFFSET-VALUE (LITERAL-VALUE OFFSET))
                        (RELEASE-TREE OFFSET))
                       (T (SETQ COMPUTED-OFFSET-TREE OFFSET)))))
          (LET ((OFFSET (SECOND ARGS)))
               (COND
                  ((AND (LITERAL-P OFFSET)
                        (INTEGERP (LITERAL-VALUE OFFSET)))
                   (INCF LITERAL-OFFSET-VALUE (LITERAL-VALUE OFFSET))
                   (RELEASE-TREE OFFSET))
                  ((NULL COMPUTED-OFFSET-TREE)
                   (SETQ COMPUTED-OFFSET-TREE OFFSET))
                  (T (SETQ COMPUTED-OFFSET-TREE (MAKE-CALL :FN (MAKE-OPCODES :BYTES '(IL:PLUS2))
                                                           :ARGS
                                                           (LIST COMPUTED-OFFSET-TREE OFFSET)
                                                           :META-P T)))))
          (UNLESS (<= 0 LITERAL-OFFSET-VALUE 255)                ; The literal offset is not in range to be an alpha byte, so must
                                                                 ; use \ADDBASE.
              (SETQ COMPUTED-OFFSET-TREE (IF (NULL COMPUTED-OFFSET-TREE)
                                             (MAKE-LITERAL :VALUE LITERAL-OFFSET-VALUE :META-P T)
                                             (MAKE-CALL :FN (MAKE-OPCODES :BYTES '(IL:PLUS2))
                                                    :ARGS
                                                    (LIST COMPUTED-OFFSET-TREE (MAKE-LITERAL :VALUE
                                                                                             LITERAL-OFFSET-VALUE
                                                                                             :META-P T))
                                                    :META-P T)))
              (SETQ LITERAL-OFFSET-VALUE 0))
          (IF (NULL COMPUTED-OFFSET-TREE)
              ;; The \ADDBASE call is unnecessary.
              (WHEN (NOT (NULL ADDBASE-CALL))
                  (POP (CALL-ARGS ADDBASE-CALL))                 ; Detach the REAL-BASE from this useless node.
                  (RELEASE-TREE ADDBASE-CALL))
              ;; We need an \ADDBASE call.  Reuse the old one if there is one.
              (COND
                 ((NULL ADDBASE-CALL)
                  (SETQ REAL-BASE (MAKE-CALL :FN (MAKE-REFERENCE-TO-VARIABLE :NAME 'IL:\\ADDBASE :SCOPE :GLOBAL
                                                                             :KIND :FUNCTION)
                                             :ARGS
                                             (LIST REAL-BASE COMPUTED-OFFSET-TREE)
                                             :META-P T)))
                 (T (SETF (SECOND (CALL-ARGS ADDBASE-CALL))
                          COMPUTED-OFFSET-TREE)
                    (SETQ REAL-BASE ADDBASE-CALL))))
          ;; Finally put it all back together again.  REAL-BASE is the (possibly computed) base and LITERAL-OFFSET-VALUE is the alpha byte for
          ;; the opcode.  We can reuse the node for the original call.
          (RELEASE-TREE (CALL-FN NODE))
          (SETF (CALL-FN NODE)
                (MAKE-OPCODES :BYTES `(,OPCODE ,LITERAL-OFFSET-VALUE ,@(AND BETA-BYTE (LIST BETA-BYTE)))))
          (SETF (CALL-ARGS NODE)
                (IF (EQ USE :GET)
                    (LIST REAL-BASE)
                    (LIST REAL-BASE (THIRD ARGS))))
          (SETF (NODE-META-P NODE)
                CONTEXT)
          NODE))
```

;; List-structure functions

(DEFTRANSFORM **CAR** (NODE CONTEXT)

;;; Transforms (CAR (CONS X Y)) => (PROG1 X Y).

```
   (COND
      ((IS-CALL-TO 'CONS (FIRST (CALL-ARGS NODE)))
       (LET* ((CONS-ARGS (CALL-ARGS (FIRST (CALL-ARGS NODE)))))
             ;; First, release the CAR and CONS nodes by detaching the CONS's arguments from the tree (they'll be in the PROG1 created below)
             ;; and then releasing the NODE.
             (SETF (CALL-ARGS (FIRST (CALL-ARGS NODE)))
                   NIL)
             (RELEASE-TREE NODE)
             (CONSTRUCT-PROG1-TREE (FIRST CONS-ARGS)
                    (LIST (SECOND CONS-ARGS)))))
     (T (SETF (NODE-META-P NODE)
              CONTEXT)
        NODE)))
```

(DEFTRANSFORM **CDR** (NODE CONTEXT)

```
   (LET (TEMP)
        (COND
```

```
            ;; (CDR (CONS X Y)) => (PROGN X Y)
            ((IS-CALL-TO 'CONS (FIRST (CALL-ARGS NODE)))
             (PROG1 (MAKE-PROGN :STMTS (CALL-ARGS (FIRST (CALL-ARGS NODE)))
                                :META-P NIL)
                    (SETF (CALL-ARGS (FIRST (CALL-ARGS NODE)))
                          NIL)
                    (RELEASE-TREE NODE)))
            ;; (CDR (RPLACD X (SETQ Z1 ... (SETQ Zn (CONS Y NIL)) ... ) =>
            ;;      (SETQ Z1 ... (SETQ Zn (RPLCONS X Y)) ... )
            ((AND (IS-CALL-TO 'RPLACD (FIRST (CALL-ARGS NODE)))
                  (SETQ TEMP (FIND-AND-PERFORM-RPLCONS-TRANSFORM (FIRST (CALL-ARGS NODE))
                                                                 CONTEXT)))
             (SETF (CALL-ARGS NODE)
                   NIL)
             (RELEASE-TREE NODE)                                    ; Don't forget to free up the CDR node...
             TEMP)
            ;; Sometimes a CDR is just a CDR...
            (T (SETF (NODE-META-P NODE)
                     CONTEXT)
               NODE))))


(DEFTRANSFORM RPLACD (NODE CONTEXT)
   (COND
      ;; (RPLACD (CONS A B) C) => (CONS A (PROGN B C))
      ((IS-CALL-TO 'CONS (FIRST (CALL-ARGS NODE)))
       (LET ((CONS-NODE (FIRST (CALL-ARGS NODE))))
            (SETF (SECOND (CALL-ARGS CONS-NODE))
                  (MAKE-PROGN :STMTS (LIST (SECOND (CALL-ARGS CONS-NODE))
                                           (SECOND (CALL-ARGS NODE)))
                              :META-P NIL))
            (POP (CALL-ARGS NODE))                                  ; Detach the CONS call.
            (RELEASE-TREE NODE)
            (SETF (NODE-META-P CONS-NODE)
                  CONTEXT)
            CONS-NODE))
      ;; In :effect context,
      ;; (RPLACD X (SETQ Z1 ... (SETQ Zn (CONS Y NIL))) ... )) =>
      ;;      (SETQ Z1 ... (SETQ Zn (RPLCONS X Y)) ... )
      ((AND (EQ :EFFECT CONTEXT)
            (FIND-AND-PERFORM-RPLCONS-TRANSFORM NODE :EFFECT)))
      ;; No more transformations, so give up.
      (T (SETF (NODE-META-P NODE)
               CONTEXT)
         NODE)))

(DEFUN FIND-AND-PERFORM-RPLCONS-TRANSFORM (NODE CONTEXT)

;;; NODE is a CALL to RPLACD.

;;; Look for the pattern (RPLACD X (SETQ Z1 ... (SETQ Zn (CONS Y NIL)) ... ) and, if found, return the transformed version: (SETQ Z1 ... (SETQ Zn
;;; (RPLCONS X Y)) ... ).  If not found, return NIL.

;;; This transformation is valid in either :effect context or as the argument to CDR.

   (DO* ((INNER-NODE (SECOND (CALL-ARGS NODE))
                     (SETQ-VALUE INNER-NODE))
         (FIRST-SETQ INNER-NODE))
        ((NOT (SETQ-P INNER-NODE))
         ;; We've traced down the tree to the bottom of the SETQ's.  If the next thing is (CONS Y NIL), then change it into the appropriate RPLCONS
         ;; and return the top of the series of SETQ's.
         (COND
            ((OR (NOT (IS-CALL-TO 'CONS INNER-NODE))
                 (NOT (LITERAL-P (SECOND (CALL-ARGS INNER-NODE))))
                 (NOT (NULL (LITERAL-VALUE (SECOND (CALL-ARGS INNER-NODE))))))
             NIL                                                    ; Nope, it's not the pattern.

             )
            (T (SETF (CALL-ARGS INNER-NODE)
                     (LIST (FIRST (CALL-ARGS NODE))
                           (FIRST (CALL-ARGS INNER-NODE))))
               (RELEASE-TREE (CALL-FN INNER-NODE))
               (SETF (CALL-FN INNER-NODE)
                     (MAKE-OPCODES :BYTES '(IL:RPLCONS)))
               ;; The replacement node needs to have the same side-effects as the original RPLACD node did:
               (SETF (CALL-EFFECTS INNER-NODE)
                     (CALL-EFFECTS NODE))
```

```
                (SETF (CALL-AFFECTED INNER-NODE)
                      (CALL-EFFECTS NODE))
                ;; Now dispose of the original node
                (SETF (CALL-ARGS NODE)
                      NIL)
                (RELEASE-TREE NODE)
                (SETF (NODE-META-P FIRST-SETQ)
                      CONTEXT)
                FIRST-SETQ                                  ; Return the stack of SETQ's as the value of this clause of the
                                                            ; COND.

                )))))

(DEFOPTIMIZER IL:FRPLACD (&REST IL:ARGS)
                         `(RPLACD ,@IL:ARGS))

;; Use the proper makefile-environment

(IL:PUTPROPS IL:XCLC-TRANSFORMS IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE (DEFPACKAGE "COMPILER"
                                                                                     (:USE "LISP" "XCL")))
)

;; Use the proper compiler.

(IL:PUTPROPS IL:XCLC-TRANSFORMS IL:FILETYPE COMPILE-FILE)

(IL:PUTPROPS IL:XCLC-TRANSFORMS IL:COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1990 1992))
```

FUNCTION INDEX

TRANSFORM INDEX

OPTIMIZER INDEX

PROPERTY INDEX

DEFINE-TYPE INDEX

DEFINER INDEX