

File created: 13-Oct-93 17:36:04 {Pele:mv:envos}<LispCore>Sources>CLTL2>XCLC-PEEPHOLE.;1

changes to: (IL:FUNCTIONS PEEPHOLE-OPTIMIZE PEEPHOLE-OPTIMIZE-CODE)

previous date: 16-Aug-91 18:52:23 {DSK}<mo>usr>users>sybalsky>cltl2>sources>XCLC-PEEPHOLE.;1

Read Table: XCL

Package: COMPILER

Format: XCCS

; Copyright (c) 1986, 1987, 1988, 1990, 1991, 1993 by Xerox Corporation. All rights reserved.

(IL:RPAQQ **IL:XCLC-PEEPHOLECOMS**
(

::: Peephole Optimization

(IL:VARIABLES *REACHABLE-TAG-TABLE* *TAG-EQUIV-TABLE* *TAG-LOCATION-TABLE*)
(IL:FUNCTIONS PEEPHOLE-OPTIMIZE FIND-ALL-TAGS FIND-REACHABLE-TAGS FIND-TAG-DUPLICATION
PEEPHOLE-OPTIMIZE-CODE)

:: Arrange to use the proper compiler

(IL:PROP IL:FILETYPE IL:XCLC-PEEPHOLE)

:: Get the right reader environment

(IL:PROP IL:MAKEFILE-ENVIRONMENT IL:XCLC-PEEPHOLE))

::: Peephole Optimization

(DEFVAR ***REACHABLE-TAG-TABLE*** NIL

::: A hash-table of those tags that are reachable during execution. See FIND-REACHABLE-TAGS for details.

)

(DEFVAR ***TAG-EQUIV-TABLE*** NIL

::: Hash table mapping LAP tag identifiers into one of two things: 1) the keyword :REFERENCES paired with a list of the jump and push-tag instructions
::: referring to this tag, or 2) the keyword :EQUIV paired with the identifier of the representative of the tags equivalent to this one. Used in the current, ad
::: hoc peephole-optimizer.

)

(DEFVAR ***TAG-LOCATION-TABLE*** NIL

::: A hash-table mapping tag identifiers into the tails of code beginning with them.

)

(DEFUN **PEEPHOLE-OPTIMIZE** (LAP-FN)

:: Extremely ad-hoc peephole optimizer for LAP code. It currently has two obligations:

:: -- Eliminate jumps to the very next location. Those screw up the jump-resolution algorithm in the D-machine assembler.

:: -- Eliminate unreachable code. This is necessary to make keep stack-analysis from barfing during assembly.

::

:: First, optimize any local functions (e.g. FLET, LABELS, etc.):

(DESTRUCTURING-BIND ((REQUIRED &KEY OPTIONAL REST KEY ALLOW-OTHER-KEYS OTHERS NAME ARG-TYPE BLIP CLOSED-OVER
NON-LOCAL LOCAL-FUNCTIONS)
&REST BODY)

(CDR LAP-FN)

(IL:FOR LOCAL-FN IL:IN LOCAL-FUNCTIONS IL:DO (PEEPHOLE-OPTIMIZE (CADR LOCAL-FN))))

:: Now peephole-optimize the main-body code for this function:

(LET ((*TAG-EQUIV-TABLE* (MAKE-HASH-TABLE :TEST 'EQL))
(*REACHABLE-TAG-TABLE* (MAKE-HASH-TABLE :TEST 'EQL))
(TAGS-USED NIL))

DECLARE (SPECIAL TAGS-USED))

FIND-REACHABLE-TAGS (CDDR LAP-FN))

(MULTIPLE-VALUE-BIND (NEW-CODE CHANGED-P)

(**PEEPHOLE-OPTIMIZE-CODE** (CDDR LAP-FN))

(SETF (CDDR LAP-FN)

NEW-CODE)

(SETQ TAGS-USED NIL)

(IF CHANGED-P

(**PEEPHOLE-OPTIMIZE** LAP-FN)

LAP-FN))))

(DEFUN **FIND-ALL-TAGS** (CODE)

```
(DO* ((TAIL CODE (CDR TAIL))
      (INST (CAR TAIL)
            (CAR TAIL)))
      (ENDP TAIL))
;;
(CASE (CAR INST)
      ((:TAG) (SETF (GETHASH (SECOND INST)
                            *TAG-LOCATION-TABLE*
                            TAIL))
            ((:CLOSE :LAMBDA) (FIND-ALL-TAGS (CDDR INST)))
            ((:CALL) (LET ((FN-TO-CALL (SECOND INST))
                          (WHEN (AND (CONSP FN-TO-CALL)
                                      (EQ (FIRST FN-TO-CALL)
                                          :LAMBDA)))
                            (FIND-ALL-TAGS (CDDR FN-TO-CALL))))))))
```

```
(DEFUN FIND-REACHABLE-TAGS (CODE)
```

;; A tag is reachable if and only if

;; -- It lies in the direct path of execution, starting at the first instruction in CODE.

;; -- It is reachable from the beginning of the inner code of a reachable :CLOSE or :LAMBDA instruction.

;; -- It is the object of a reachable :PUSH-TAG instruction.

```
(LET ((*TAG-LOCATION-TABLE* (MAKE-HASH-TABLE :TEST 'EQL))
      (FIND-ALL-TAGS CODE)
      (DO ((ROOTS (LIST CODE))
          (NULL ROOTS))
          ;; For each root found, seek out tags reachable form that root.
```

```
(ASSERT (NOT (NULL (FIRST ROOTS)))
        NIL "A tag was referred to but not found.")
(DOLIST (INST (POP ROOTS))
        (CASE (FIRST INST)
              (:TAG)
```

; This is a reachable tag. If we already knew that, stop here.
; Else, mark it.

```
(IF (GETHASH (SECOND INST)
             *REACHABLE-TAG-TABLE*)
    (RETURN)
    (SETF (GETHASH (SECOND INST)
                  *REACHABLE-TAG-TABLE*
                  T)))
((:JUMP :TJUMP :FJUMP :NTJUMP :NFJUMP :PUSH-TAG)
```

; The object of the instruction is reachable, so add it to the list of
; roots and keep seeking from this root.

```
(LET ((NEW-ROOT (GETHASH (SECOND INST)
                        *TAG-LOCATION-TABLE*)))
    (ASSERT (NOT (NULL NEW-ROOT))
            NIL "The tag ~S was referred to but not found." (SECOND INST))
    (PUSH NEW-ROOT ROOTS))
(WHEN (EQ (FIRST INST)
          :JUMP)
      (RETURN)))
```

;; No more reachable tags from this root, so stop here.

```
((:RETURN) (RETURN))
((:CLOSE :LAMBDA) (PUSH (CDDR INST)
                        ROOTS))
;; Add the body of the instruction to the list of roots.
```

```
((:CALL) (LET ((FN-TO-CALL (SECOND INST))
              (WHEN (AND (CONSP FN-TO-CALL)
                          (EQ (FIRST FN-TO-CALL)
                              :LAMBDA)))
                (PUSH (CDDR FN-TO-CALL)
                      ROOTS))))))
;; Add the body of the directly called lambda to the list of roots.
```

```
(DEFUN FIND-TAG-DUPLICATION (CODE)
```

```
(LET ((NEW-CODE NIL)
      (FIND-P NIL)
      INST)
  (IL:FOR TAIL IL:ON CODE IL:EACHTIME (SETQ INST (CAR TAIL))
  IL:DO ;; Check for unreachable code.
```

```
(UNLESS (AND (IL:FMEMB (FIRST (FIRST NEW-CODE))
                      '(:JUMP :RETURN))
            (NOT (AND (EQ (FIRST INST)
                          :TAG)
                    (GETHASH (SECOND INST)
                              *REACHABLE-TAG-TABLE*))))))
```

```
(CASE (FIRST INST)
      (:TAG)
        (IF (EQ (FIRST (FIRST NEW-CODE))
                :TAG)
```

```
(PROGN (SETQ FIND-P T)
      (RETURN))
(PUSH INST NEW-CODE))
((:CLOSE :LAMBDA)
 (SETQ FIND-P (FIND-TAG-DUPLICATION (CDDR INST)))
 (PUSH INST NEW-CODE))
((:CALL) (LET ((FN-TO-CALL (SECOND INST)))
            (WHEN (AND (CONSP FN-TO-CALL)
                      (EQ (FIRST FN-TO-CALL)
                          :LAMBDA))
                  (SETQ FIND-P (FIND-TAG-DUPLICATION (CDDR INST)))
                  (PUSH INST NEW-CODE))))
 (OTHERWISE (PUSH INST NEW-CODE))))
FIND-P))
```

```
(DEFUN PEEPHOLE-OPTIMIZE-CODE (CODE)
```

;;; Run through the given code collapsing adjacent TAGs into a single one and eliminating jumps to immediately following TAGs. Also eliminate code
 ;;; that cannot be reached. Return the new version of the code.

```
(LET
 ( (NEW-CODE NIL)
   (CHANGED-P NIL)
   (TAG-DUPLICATED-P (FIND-TAG-DUPLICATION CODE))
   INST)
 (IL:FOR TAIL IL:ON CODE IL:EACHTIME (SETQ INST (CAR TAIL))
  IL:DO ;; Check for unreachable code.
        ;; Code is unreachable if the last instruction was a JUMP or RETURN, and the next thing coming isn't a TAG that is reachable from
        ;; somewhere else.
        ;; (If dead code is removed here, that's worth a CHANGED-P indication)
        (UNLESS (AND (IL:FMEMB (FIRST (FIRST NEW-CODE))
                              ' (:JUMP :RETURN))
                    (NOT (AND (EQ (FIRST INST)
                                  :TAG)
                              (GETHASH (SECOND INST)
                                        *REACHABLE-TAG-TABLE*))))
                (SETQ CHANGED-P T))
          (CASE (FIRST INST)
            ((:JUMP :TJUMP :FJUMP :NTJUMP :NFJUMP :PUSH-TAG)
             (LET ((LOOKUP (GETHASH (SECOND INST)
                                    *TAG-EQUIV-TABLE*)))
               (PUSH INST NEW-CODE)
               (ECASE (CAR LOOKUP)
                ((NIL) ; This tag is not yet in the table. Put it in there mapping to a list of
                    ; references including only this one.
                 (PUSHNEW (SECOND INST)
                          TAGS-USED)
                 (SETF (GETHASH (SECOND INST)
                                *TAG-EQUIV-TABLE*)
                       (CONS :REFERENCES (LIST INST))))
                ((:REFERENCES) ; We haven't seen the TAG for this reference yet. Add it to the
                    ; list of references to that tag.
                 (PUSHNEW (SECOND INST)
                          TAGS-USED)
                 (PUSH INST (CDR LOOKUP)))
                ((:EQUIV) ; We know what the right tag for this reference is now.
                 (PUSHNEW (CDR LOOKUP)
                          TAGS-USED)
                 (SETF (SECOND INST)
                       (CDR LOOKUP))))))
            ((:TAG)
             (LET ((LOOKUP (GETHASH (SECOND INST)
                                    *TAG-EQUIV-TABLE*)))
               (IF (EQ (FIRST (FIRST NEW-CODE))
                      :TAG)
                   (PROGN ;; Mark this tag in the table as being equivalent either to the directly previous tag, if any, or to itself.
                         (SETF (GETHASH (SECOND INST)
                                        *TAG-EQUIV-TABLE*)
                              (CONS :EQUIV (SECOND (FIRST NEW-CODE))))
                         (PUSHNEW (SECOND (FIRST NEW-CODE))
                                  TAGS-USED)
                         ;; If there were forward references to this tag, update all of them to refer to the EQUIV-TAG.
                         (IF (EQ (CAR LOOKUP)
                                :REFERENCES)
                             (IL:FOR REFERENCE IL:IN (CDR LOOKUP)
                              IL:DO (SETF (SECOND REFERENCE)
                                           (SECOND (FIRST NEW-CODE))))
                             (ASSERT (NULL LOOKUP)
                                     NIL "This tag has been seen before!"))
                         (SETQ CHANGED-P T))
                   (COND
                    ((AND (NOT TAG-DUPLICATED-P)
```

```

(EQ (FIRST (SECOND TAIL))
:JUMP))
(SETF (GETHASH (SECOND INST)
*TAG-EQUIV-TABLE*)
(CONS :EQUIV (SECOND (SECOND TAIL))))
(IF (EQ (CAR LOOKUP)
:REFERENCES)
(IL:FOR REFERENCE IL:IN (CDR LOOKUP)
IL:DO (WHEN (NOT (EQL (SECOND REFERENCE)
(SECOND (SECOND TAIL))))
(SETF (SECOND REFERENCE)
(SECOND (SECOND TAIL)))
(SETQ CHANGED-P T))))
(PUSH INST NEW-CODE))
(T (SETF (GETHASH (SECOND INST)
*TAG-EQUIV-TABLE*)
(CONS :EQUIV (SECOND INST)))
(IF (EQ (CAR LOOKUP)
:REFERENCES)
(IL:FOR REFERENCE IL:IN (CDR LOOKUP) IL:DO (SETF (SECOND REFERENCE)
(SECOND INST)))
(ASSERT (NULL LOOKUP)
NIL "This tag has been seen before!"))
(PUSH INST NEW-CODE))))
;; If the next instruction is not a :TAG, then it's time to check for useless jumps and to eliminate them.
(WHEN (OR (NULL (CDR TAIL))
(NOT (EQ (FIRST (CDR TAIL))
:TAG)))
(LOOP ;; Repeatedly examine the top 2 or 3 instructions, looking for sequences
;; JUMP x - TAG x or
;; JUMP x - SET-STACK - TAG x
;; cJUMP x - JUMP y - TAG x
;; and reducing them to just the TAG, with a POP if need be.
(LET ((TAG-INST (FIRST NEW-CODE))
(JUMP-INST (SECOND NEW-CODE))
(SET-STACK-INST (THIRD NEW-CODE)))
(IF (EQL (SECOND TAG-INST)
(SECOND JUMP-INST))
;; Looks like something to eliminate.
(CASE (FIRST JUMP-INST)
(:JUMP)
(SETF (CDR NEW-CODE)
(CDDR NEW-CODE))
(SETQ CHANGED-P T))
(:FJUMP :TJUMP)
(SETF (SECOND NEW-CODE)
'(:POP))
(SETQ CHANGED-P T)
(RETURN))
(:NJUMP :NFJUMP) (ERROR "BUG: Non-popping jump to very next
location."))
(OTHERWISE
;; The instruction before the :TAG was not a jump, so do nothing.
(RETURN)))
(IF (EQL (SECOND TAG-INST)
(SECOND SET-STACK-INST))
;; Looks like it might be JUMP-SET-TAG or cJUMP - JUMP - TAG
(COND
((EQ (FIRST JUMP-INST)
:DSET-STACK)
;; YES, it's JUMP - SET - TAG
(ROTATEF JUMP-INST SET-STACK-INST)
(CASE (FIRST JUMP-INST)
(:JUMP)
(SETF (CDR NEW-CODE)
(CDDDR NEW-CODE))
(SETQ CHANGED-P T))
(:FJUMP :TJUMP)
(SETF (SECOND NEW-CODE)
'(:POP))
(SETF (CDDR NEW-CODE)
(CDDDR NEW-CODE))
(SETQ CHANGED-P T)
(RETURN))
(:NJUMP :NFJUMP) (ERROR "BUG: Non-popping jump to very
next location."))
(OTHERWISE
;; The instruction before the :SET was not a jump, so do nothing.

```

```

                (RETURN)))
            ((EQ (FIRST JUMP-INST)
                :JUMP)
             ;; YES, it's cJUMP - JUMP - TAG
             (CASE (FIRST SET-STACK-INST)
                  ((:TJUMP)
                   (RPLACA JUMP-INST :FJUMP)
                   (SETF (CDDR NEW-CODE)
                         (CDDDR NEW-CODE)))
                   (SETQ CHANGED-P T))
                  ((:FJUMP)
                   (RPLACA JUMP-INST :TJUMP)
                   (SETF (CDDR NEW-CODE)
                         (CDDDR NEW-CODE)))
                   (SETQ CHANGED-P T))
                  (OTHERWISE
                   ;; The instruction before the JUMP was not a cJUMP, so do nothing
                   (RETURN)))
            (T ;; The instruction before the :TAG was not a SET, so do nothing.
             (RETURN)))
            ;; Nothing (more) to get rid of, so stop.
            (RETURN))))))
    ( (:VAR)
      ;; Eliminate any unnecessary POPs, e.g.:
      ;; VAR_ x ; POP ; VAR x
      (LET ((SET-INST (SECOND NEW-CODE))
            (POP-INST (FIRST NEW-CODE)))
          (COND
           ((AND (EQ (FIRST POP-INST)
                     :POP)
                 (EQ (FIRST SET-INST)
                     :VAR_))
            (EQL (SECOND SET-INST)
                 (SECOND INST)))
            (SETF NEW-CODE (CDR NEW-CODE))
            (SETQ CHANGED-P T))
           (T (PUSH INST NEW-CODE))))))
    ( (:CLOSE :LAMBDA)
      (MULTIPLE-VALUE-BIND (CODE-SET CHANGED?)
        (PEEPHOLE-OPTIMIZE-CODE (CDDR INST))
        (SETF (CDDR INST)
              CODE-SET)
        (SETQ CHANGED-P (OR CHANGED-P CHANGED?)))
        (PUSH INST NEW-CODE)))
    ( (:CALL)
      (LET ((FN-TO-CALL (SECOND INST))
            (WHEN (AND (CONSP FN-TO-CALL)
                       (EQ (FIRST FN-TO-CALL)
                           :LAMBDA))
              (MULTIPLE-VALUE-BIND (CODE-SET CHANGED?)
                (PEEPHOLE-OPTIMIZE-CODE (CDDR FN-TO-CALL))
                (SETF (CDDR FN-TO-CALL)
                      CODE-SET)
                (SETQ CHANGED-P (OR CHANGED-P CHANGED?))))))
          (PUSH INST NEW-CODE))
        (OTHERWISE (PUSH INST NEW-CODE))))))
    ;; Now remove unused tags, and put things back into first-to-last order.
    (VALUES (NREVERSE (IL:FOR INST IL:IN NEW-CODE IL:WHEN (OR (IL:NEQ (FIRST INST)
                                                                       :TAG)
                                                                (IL:FMEMB (SECOND INST)
                                                                TAGS-USED)
                                                                (NOT (SETQ CHANGED-P T))))
              IL:COLLECT INST))
            CHANGED-P)))

;; Arrange to use the proper compiler
(IL:PUTPROPS IL:XCLC-PEEPHOLE IL:FILETYPE COMPILE-FILE)

;; Get the right reader environment
(IL:PUTPROPS IL:XCLC-PEEPHOLE IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE (DEFPACKAGE "COMPILER"
                                                (:USE "LISP" "XCL"))))

(IL:PUTPROPS IL:XCLC-PEEPHOLE IL:COPYRIGHT ("Xerox Corporation" 1986 1987 1988 1990 1991 1993))

```

FUNCTION INDEX

FIND-ALL-TAGS	1	FIND-TAG-DUPLICATION	2	PEEPHOLE-OPTIMIZE-CODE	3
FIND-REACHABLE-TAGS	2	PEEPHOLE-OPTIMIZE	1		

VARIABLE INDEX

REACHABLE-TAG-TABLE	1	*TAG-EQUIV-TABLE*	1	*TAG-LOCATION-TABLE*	1
-----------------------------	---	-------------------------	---	----------------------------	---

PROPERTY INDEX

IL:XCLC-PEEPHOLE	5
------------------------	---
