CLL Compiler Work

How many memory references per iteration in `copy-@s-@r-@d-st`? Problem grew out of Nathan Wetzler's PhD effort.

```
(defstobj st

  (m :type (array (signed-byte 60)  ; array of 60-bit integers
                  (*init-m-size*))  ; with this initial length
     :initially 0
     :resizable t)

  :inline t                         ; for performance
  :non-memoizable t                 ; also for performance
  :renaming                         ; for brevity
  ((update-mi  !mi) (m-length  ml)))

(defun copy-@s-@r-@d-st (@s @r @d st)
  (declare (xargs :guard (and (natp-lst @s @r @d)
                              (<= @s @r)
                              (ml-limit @r *2^59*)
                              (ml-limit (+ @d (- @r @s)) *2^59*))
                  :stobjs (st)
                  :measure (nfix (- @r @s))))
  (b* ((@s (u59 @s))    ; NFIX above (in measure) is necessary!
       (@r (u59 @r))    ; NFIX below is not necessary!
       (@d (u59 @d)))
      (if (mbe :logic (zp (- @r @s)) :exec (>= @s @r))
          st
        (b* ((v    (s60 (mi @s st)))
             (st        (!mi @d v st))
             (@s+1 (u59 (1+ @s)))
             (@d+1 (u59 (1+ @d))))
            (copy-@s-@r-@d-st @s+1 @r @d+1 st)))))
```

Theorems about copy-@s-@r-@d-st

We can learn some facts about our `copy` procedure.

```
(defthm stp-copy-@s-@r-@d-st
  (implies (and (stp st)
                (natp-lst @s @d) ; @r
                (<= @r (ml st))
                (<= (+ @d (- @r @s)) (ml st)))
           (stp (copy-@s-@r-@d-st @s @r @d st))))
```

```
(defthm ml-copy-@s-@r-@d-st
  (implies (and (stp st)
                (natp-lst @s @d) ; @r
                (<= @r (ml st))
                (<= (+ @d (- @r @s)) (ml st)))
           (equal (ml (copy-@s-@r-@d-st @s @r @d st))
                  (ml st))))
```

Some More Theorems about copy-@s-@r-@d-st

```
(defthm mi-below-copy-@s-@r-@d-st
  (implies (and (stp st)
                (natp-lst @s @d below) ; @r
                (<= @r (ml st))
                (<= (+ @d (- @r @s)) (ml st))
                (< below @d))
           (equal (mi below (copy-@s-@r-@d-st @s @r @d st))
                  (mi below st))))


(defthm mi-above-copy-@s-@r-@d-st
  (implies (and (stp st)
                (natp-lst @s @d above) ; @r
                (<= @r (ml st))
                (<= (+ @d (- @r @s)) (ml st))
                (<= (+ @d (- @r @s)) above))
           (equal (mi above (copy-@s-@r-@d-st @s @r @d st))
                  (mi above st))))


(defthm mi-copy-@s-@r-@d-st
  (implies (and (stp st)
                (natp-lst @s @r @d dest)
                (< @r (ml st))
                (<= 0 (- @r @s))
                (<= (+ @d (- @r @s)) (ml st))
                (or (<= (+ @d (- @r @s)) @s)
                    (<= @r @d))
                (<= @d dest)
                (< dest (+ @d (- @r @s))))
           (equal (mi dest
                      (copy-@s-@r-@d-st @s @r @d st))
                  (mi (+ @s (- dest @d))
                      st))))
```

The ccl Compiler Output for copy-@s-@r-@d-st

```
;;; (defun copy-@s-@r-@d-st (@s @r @d st) ...
L0
    (leaq (@ (:^ L0) (% rip)) (% fn))       ;     [0]
    (movq (% rbp) (@ 16 (% rsp)))           ;     [7]
    (leaq (@ 16 (% rsp)) (% rbp))           ;    [12]
    (popq (@ 8 (% rbp)))                    ;    [17]
    (pushq (% arg_x))                       ;    [20]
    (pushq (% save0))                       ;    [22]
    (pushq (% save1))                       ;    [24]
    (pushq (% save2))                       ;    [26]
    (movq (% arg_z) (% save0))              ;    [28]
    (movq (% arg_y) (% save2))              ;    [31]
    (movq (@ -8 (% rbp)) (% save1))         ;    [34]

;;; (>= @s @r)
L38
    (cmpq (% arg_x) (% save1))              ;    [38]
    (jl L54)                                ;    [41]

;;; (if (>= @s @r) st (let* ((v (s60 (mi @s st))) ...
    (movq (% save0) (% arg_z))              ;    [43]
    (popq (% save2))                        ;    [46]
    (popq (% save1))                        ;    [48]
    (popq (% save0))                        ;    [50]
    (leaveq)                                ;    [52]
    (retq)                                  ;    [53]
```

```
;;; (mi @s st)
L54
    (movq (@ -5 (% save0)) (% arg_y))       ;    [54]
    (movq (@ -5 (% arg_y) (% save1)) (% imm0)) ;    [58]
    (imulq ($ 8) (% imm0) (% arg_z))        ;    [63]

;;; (let* ((v (s60 (mi @s st))) (st (!mi @d v st)) ...
    (pushq (% arg_z))                       ;    [67]  <===***

;;; (!mi @d v st)
    (movq (@ -5 (% save0)) (% arg_x))       ;    [68]
    (movq (% arg_z) (% imm0))               ;    [72]
    (sarq ($ 3) (% imm0))                   ;    [75]
    (movq (% imm0) (@ -5 (% arg_x) (% save2))) ;    [79]
    (movq (% save0) (% arg_y))              ;    [84]

;;; (let* ((v (s60 (mi @s st))) (st (!mi @d v st)) ...
    (pushq (% arg_y))                       ;    [87]  <===***

;;; (1+ @s)
    (leaq (@ 8 (% save1)) (% arg_x))        ;    [88]

;;; (let* ((v (s60 (mi @s st))) (st (!mi @d v st)) ...
    (pushq (% arg_x))                       ;    [92]  <===***

;;; (1+ @d)
    (leaq (@ 8 (% save2)) (% temp1))        ;    [94]

;;; (let* ((v (s60 (mi @s st))) (st (!mi @d v st)) ...
    (pushq (% temp1))                       ;    [99]  <===***

;;; (copy-@s-@r-@d-st @s+1 @r @d+1 st)
    (movq (% arg_x) (% save1))              ;   [101]
    (movq (@ -16 (% rbp)) (% arg_x))        ;   [104]
    (movq (% temp1) (% save2))              ;   [108]
    (movq (% arg_y) (% save0))              ;   [111]
    (addq ($ 32) (% rsp))                   ;   [114]  <===***
    (jmp L38)                               ;   [118]
```

The `ccl` Compiler Output for `copy-@s-@r-@d-st`

```
? (disassemble 'COPY-@S-@R-@D-ST)
;;; (ILISP:ilisp-eval "(defun copy-@s-@r-@d-st (@s @r @d st) ...
(type (unsigned-byte 59) @s @r @d)
     (recover-fn-from-rip)                      ;     [7]
     (popq (@ 16 (% rsp)))                      ;    [14]
     (popq (% arg_w))                           ;    [18]
     (addq ($ 8) (% rsp))                       ;    [20]
     (pushq (% rbp))                            ;    [24]
     (movq (% rsp) (% rbp))                     ;    [25]
L21
     (movq (% arg_w) (% temp3))                 ;    [28]
     (movq (% arg_x) (% temp4))                 ;    [31]
     (movq (% arg_y) (% arg_w))                 ;    [34]
     (movq (% arg_z) (% temp2))                 ;    [37]
     (movq (@ -8 (% rbp)) (% temp0))            ;    [40]
     (movq (% temp3) (% temp1))                 ;    [44]
     (movq (% temp4) (% arg_x))                 ;    [47]
     (cmpq (% arg_x) (% temp1))                 ;    [50]
     (jl L53)                                   ;    [53]
     (movq (% temp2) (% arg_z))                 ;    [55]
     (leaveq)                                   ;    [58]
     (retq)                                     ;    [59]
L53
     (movq (% temp3) (% temp1))                 ;    [60]
     (movq (% temp2) (% arg_x))                 ;    [63]
     (movq (% arg_x) (% arg_y))                 ;    [66]
     (movq (@ -5 (% arg_y)) (% arg_x))          ;    [69]
     (movq (% temp1) (% arg_y))                 ;    [73]
     (movq (@ -5 (% arg_x) (% arg_y)) (% imm1)) ;    [76]
     (imulq ($ 8) (% imm1) (% temp1))           ;    [81]
     (movq (% arg_w) (% arg_x))                 ;    [85]
     (movq (% temp1) (% arg_y))                 ;    [88]
```

```
(movq (% temp2) (% temp1))                  ;    [91]
(movq (% temp1) (% temp2))                  ;    [94]
(movq (@ -5 (% temp2)) (% arg_z))           ;    [97]
(movq (% arg_x) (% temp2))                  ;   [101]
(movq (% arg_y) (% arg_x))                  ;   [104]
(movq (% arg_x) (% imm1))                   ;   [107]
(sarq ($ 3) (% imm1))                       ;   [110]
(movq (% imm1) (@ -5 (% arg_z) (% temp2))) ;   [114]
(movq (% temp1) (% temp2))                  ;   [119]
(movq (% temp3) (% temp1))                  ;   [122]
(addq ($ 8) (% temp1))                      ;   [125]
(movq (% arg_w) (% temp3))                  ;   [129]
(addq ($ 8) (% temp3))                      ;   [132]
(movq (% temp1) (% arg_w))                  ;   [136]
(movq (% temp4) (% temp1))                  ;   [139]
(movq (% temp3) (% temp4))                  ;   [142]
(movq (% temp2) (% temp3))                  ;   [145]
(movq (% temp3) (% arg_z))                  ;   [148]
(movq (% temp4) (% arg_y))                  ;   [151]
(movq (% temp1) (% arg_x))                  ;   [154]
(jmpq L21)                                  ;   [157]
```

This seems to be a lot better. I don't know what is causing the stack reference at [40], and nothing references the register loaded there. The hysteria in the function prologue has to do with the fact that we want to act as if the function got 4 arguments in registers (arg_w -arg_z). but the calling conventions didn't change. The tail call passes all arguments in register as expected in this case.

I suspect that whatever is causing [40] will be relatively easy to find, and there are no other uses of the frame pointer so the instruction that are saving and restoring it will be eliminated when [40] goes away.

Some Comments

Bob Boyer works daily with Gary Byers (the author of `ccl`) on trying the compiler on ACL2.

This work has involved years of effort.

This kind of effort requires real money.

Additional support would be welcomed!