

Table of contents

2-	1	CSHIO	-- Check if data is in cache
3-	1	CSHRD	-- Read request
4-	1	READIO	-- Start read I/O operation
5-	1	REDCPL	-- Read I/O completion routine
6-	1	CSHWRT	-- Write request
7-	1	WRTCPL	-- Write operation completion routine
8-	1	CSHUP	-- Update cache with fresh data
9-	1	RDMOVE	-- Move data from cache to user's buffer
10-	1	WTMOVE	-- Move data from user's buffer to cache
11-	1	CSHBLD	-- Build empty cache descriptors for new entries
12-	1	CSHCLN	-- Remove all entries for a specified device
13-	1	CSHLOC	-- Determine if entry exists in cache table
14-	1	CSHUSE	-- Make cache entry be most-recently-used
15-	1	CSHADD	-- Add cache table entry to appropriate lists
16-	1	CSHREM	-- Remove cache block entry from all lists
17-	1	CSHGET	-- Try to get a free cache block entry
18-	1	CSHFRE	-- Return a cache block entry to the free list
19-	1	CSHTST	-- Determine if device is mounted
20-	1	CCBGET	-- Get a free cache control block
21-	1	CCBFRE	-- Free a cache control block
22-	1	CSHFIN	-- Completion of a mapped I/O operation
23-	1	CSHGTQ	-- Get a free I/O queue element
24-	1	CSHCNT	-- Keep track of cache usage statistics
25-	1	CSHZRD	-- Reset all statistic counters to zero
26-	1	CSHSCH	-- See if system job scheduler should be called
27-	1	CSHINI	-- Data caching system initialization

```

1          .TITLE  TSCASH -- TSX-Plus Data Caching Facility
2          .ENABL  LC
3          .DSABL  QBL
4 000000   .CSECT  TSCASH
5          ;
6          ; TSCASH is the TSX-Plus module that contains the routines
7          ; used to perform data caching.
8          ;
9          ; Copyright (c) 1984,1985.
10         ; S&H Computer Systems, Inc.
11         ; Nashville, Tennessee USA
12         ; All rights reserved.
13         ;
14         ; Macro calls
15         ;
16         .MCALL .ADDR
17         ;
18         ; Global definitions
19         ;
20         .GLOBL TSCASH
21         ;
22         ; Global references
23         ;
24         .GLOBL CC$CBP, Q. FUNC, Q. COMP, Q. CSW, CC$WCT, DVRHC
25         .GLOBL CC$WFL, CD$DVU, CC$DVU, CC$UBD, CC$UBP, CC$BLK
26         .GLOBL Q. UNIT, CA$DVU, C. CSW, CA$WCT, CA$UFL, CA$UBL
27         .GLOBL CS$EOF, CA$HFL, Q. WCNT, ENSYS, CSHDVN, KPAR5
28         .GLOBL KPAR6, Q. DEVX, CA$BLK, CA$HBL, CSHMRU, CA$HSH
29         .GLOBL VPAR6, VPAR5, CC$DQE, SYQID, Q. JOB, CD$$SZ
30         .GLOBL CSHLRU, CSHDEV, CSHBFP, CS$ERR, CCBHD, S$QCCB
31         .GLOBL QNSPNX, CC$LNK, Q. BLKN, Q. BUFF, Q. PAR
32         .GLOBL UREGD, IOCMPL, IOQSIZ, Q. CHAN, FP$MOV, USWPCH
33         .GLOBL INTPRI, PSW, CSHFHD, VCSHNB, JOBCCB, NPCCB
34         .GLOBL Q. FLAG, QF$CIO, QF$SCR, GETQ, DOSCHD, SCHED
35         .GLOBL CASTBR, CASCBR, CASTBW, CASCUP, CORUSR
36         .GLOBL CASTRO, CASTWO, DVFLAG, DX$NCA, $NOABT, LSW9
37         ;
38         ; Macro to map to a data caching table
39         ;
40         .MACRO MAPTO TABLE
41         MOV     @#TABLE, @#KPAR5
42         .ENDM  MAPTO
43         ;
44         ; Macro to disable device interrupts
45         ;
46         .MACRO DISABL
47         BIS     #340, @#PSW
48         .ENDM  DISABL
49         ;
50         ; Macro to enable interrupts
51         ;
52         .MACRO ENABL
53         BIC     @#INTPRI, @#PSW
54         .ENDM  ENABL
55         ;
56         ; Macro to call a system overlay
57         ;

```

```

58          .MACRO  OCALL  ENTADD
59          .IF      B,ENTADD
60          .ERROR   ;OCALL without entry address
61          .ENDC
62          CALL     @#OVRHC
63          .WORD    ENTADD
64          .ENDM    OCALL
65          ;
66          ;-----
67          ; Module header.  Must be first data in segment.
68          ;
69          ; Overlay segment ID word
70          ;
71 000000    012700  TSCASH: .RAD50  /CSH/
72          ;
73          ; Table of offsets to entry points in this module
74          ;
75 000002    003622          .WORD    CSHINI-TSCASH
76 000004    000020          .WORD    CSHIO-TSCASH
77 000006    001530          .WORD    CSHCLN-TSCASH
78 000010    003346          .WORD    CSHFIN-TSCASH
79          ;
80          ; Data areas
81          ;
82 000012    000000  CURCCB: .WORD    0          ;Current cache control block
83 000014    000000  IOQIN:  .WORD    0          ;Current original I/O queue element
84 000016    000000  IOQOUT: .WORD    0          ;Current secondary I/O queue element

```

CSHIO -- Check if data is in cache

```

1          .SBTTL  CSHIO  -- Check if data is in cache
2
3          ;-----
4          ; CSHIO is called before each I/O operation is initiated to determine
5          ; if the data being read is in the cache or, in the case of a write,
6          ; whether the data cache needs to be updated.
7
8          ; Inputs:
9          ;   R1 = Address of I/O queue element.
10
11         ; Outputs:
12         ;   C-flag cleared ==> Queue request normally.
13         ;   C-flag set    ==> Don't queue I/O request. We will do completion.
14 000020 010346 CSHIO:  MOV     R3,-(SP)
15 000022 010546         MOV     R5,-(SP)
16 000024 013746 000000G        MOV     @#KPAR5,-(SP) ; Save kernel PAR 5 mapping
17
18         ; Ignore this I/O operation if it is being performed by the system
19         ; or if this is a secondary I/O operation started by the caching system.
20
21 000030 132761 000000G 000000G        BITB    #QF%CI0,Q.FLAG(R1) ; Is this a secondary op started by us?
22 000036 001053         BNE     9$                ; Br if yes
23 000040 105761 000000G        TSTB    Q.JOB(R1)         ; Is this a system I/O operation?
24 000044 001450         BEQ     9$                ; Don't do caching for system I/O operations
25 000046 126127 000000G 000000G        CMPB    Q.CHAN(R1),#USWPCH ; Is this user channel for swap file?
26 000054 001444         BEQ     9$                ; Br if yes -- Don't cache I/O to swap file
27
28         ; Determine if the device being accessed is mounted by anyone
29
30 000056 004767 002636         CALL    CSHTST           ; Is this device mounted?
31 000062 103441         BCS     9$                ; Br if not
32
33         ; Determine if caching has been disabled
34
35 000064 005737 000000G        TST     @#VCSHNB        ; Are there any active cache buffers?
36 000070 001436         BEQ     9$                ; Br if not
37
38         ; This device is mounted and hence should be cached.
39         ; Determine if this is a .SPFUN operation.
40
41 000072 105761 000000G        TSTB    Q.FUNC(R1)       ; Is this a .SPFUN operation?
42 000076 001413         BEQ     3$                ; Br if not
43
44         ; .SPFUN being done to cached device.
45         ; Clean out all cache entries for this device.
46
47 000100 116100 000000G        MOVB    Q.UNIT(R1),R0    ; Get device unit number
48 000104 042700 177770         BIC     #^C7,R0         ; Clear all but unit number
49 000110 000300         SWAB   R0              ; Put in high-order byte
50 000112 116103 000000G        MOVB    Q.DEVX(R1),R3    ; Get device index number
51 000116 050003         BIS     R0,R3          ; Combine unit # with device #
52 000120 004767 001404         CALL    CSHCLN         ; Clean out the cache
53 000124 000420         BR     9$              ; Ignore this I/O operation
54
55         ; This is not a .SPFUN.
56         ; Ignore the operation if the word count is zero.
57

```

```

58 000126 005761 0000000 3$:   TST      Q.WCNT(R1)      ;Is word count zero?
59 000132 001415          BEQ      9$              ;Br if yes -- Don't do caching
60                                     ;
61                                     ;   Get a cache control block.
62                                     ;
63 000134 004767 002656   CALL     CCBGET          ;Get a cache control block
64                                     ;
65                                     ;   Determine if it is a .READ or .WRITE operation
66                                     ;
67 000140 005761 0000000   TST      Q.WCNT(R1)      ;Read/Write
68 000144 002404          BLT      1$              ;Br if write operation
69                                     ;
70                                     ;   This is a read request
71                                     ;
72 000146 004767 000030   CALL     CSHRD          ;See if we can find data in cache
73 000152 000261          SEC                      ;Signal that caching is being done
74 000154 000405          BR       10$
75                                     ;
76                                     ;   This is a write request
77                                     ;
78 000156 004767 000306   1$:     CALL     CSHWRT          ;Start the write operation
79 000162 000261          SEC                      ;Signal that caching is being done
80 000164 000401          BR       10$
81                                     ;
82                                     ;   We do not want to do caching for this operation
83                                     ;
84 000166 000241          9$:     CLC                      ;Signal normal request queueing
85                                     ;
86                                     ;   Finished
87                                     ;
88 000170 012637 0000000   10$:    MOV      (SP)+, @#KPAR5 ;Restore kernel PAR 5 mapping
89 000174 012605          MOV      (SP)+, R5
90 000176 012603          MOV      (SP)+, R3
91 000200 000207          RETURN

```

```

1
2
3
4
5
6
7
8
9 000202 010146
10 000204 010346
11
12
13
14 000206 012700 000001
15 000212 004167 003262
16 000216 0000000
17 000220 016500 0000000
18 000224 062700 000377
19 000230 000300
20 000232 042700 177400
21 000236 004167 003236
22 000242 0000000
23
24
25
26 000244 004767 001354
27 000250 103442
28
29
30
31
32 000252 016503 0000000
33 000256 020327 000400
34 000262 101402
35 000264 012703 000400
36 000270
37 000276 020311
38 000300 101026
39
40
41
42 000302 004767 000442
43
44
45
46
47 000306 004767 001460
48
49
50
51 000312 012700 000001
52 000316 004167 003156
53 000322 0000000
54
55
56
57

```

```

.SBTTL CSHRD -- Read request
-----
; A read request is being performed.
; See if we can get all or some of the requested data from the cache.
;
; Inputs:
; R5 = Pointer to cache control block for this operation.
;
CSHRD: MOV R1, -(SP)
      MOV R3, -(SP)
;
; Count total number of blocks read from mounted devices
;
      MOV #1, R0 ; Count 1 more read operation
      JSR R1, CSHCNT ; Update statistics counter
      .WORD CASTRO ; Total number of read operations
      MOV CC#WCT(R5), R0 ; Get total word count
      ADD #255., R0 ; Round up to block size
      SWAB R0 ; Convert to # blocks
      BIC #^C377, R0 ; Clear all but low byte
      JSR R1, CSHCNT ; Update statistics counter
      .WORD CASTBR ; Total number of blocks read
;
; Begin loop to determine which blocks are in the cache.
;
1$: CALL CSHLOC ; Determine if next block is in the cache
   BCS 2$ ; Br if not
;
; There is an entry for this block in the cache.
; See if we have at least the requested number of words.
;
      MOV CC#WCT(R5), R3 ; Get the remaining word count
      CMP R3, #256. ; Is there more than a block left to read?
      BLOS 3$ ; Br if not
      MOV #256., R3 ; Say we are reading a block this time
3$: MAPTO CA#WCT ; Access cache table word count entry
   CMP R3, (R1) ; Are the required number of words in cache?
   BHI 2$ ; Br if not
;
; Transfer data from cache buffer to user's buffer
;
      CALL RDMOVE ; Move data from cache to user's buffer
;
; Move this cache block to the head of the LRU chain
; (That is, make it the most recently used)
;
      CALL CSHUSE ; Make this block be the most recently used
;
; Count another block that was read from the cache
;
      MOV #1, R0 ; Add 1 to counter
      JSR R1, CSHCNT ; Update statistics counter
      .WORD CASCBR ; Number of blocks read from cache
;
; See if the job scheduler needs to be called to give someone else a
; chance to run
;

```

```
58 000324 004767 003256          CALL    CSHSCH          ; See if job scheduler needs to be called
59                               ;
60                               ; Advance block number, buffer address, and reduce remaining word count.
61                               ;
62 000330 005265 000000G          INC     CC$BLK(R5)      ; Advance block number
63 000334 062765 000010 000000G   ADD     #8, CC$UBP(R5) ; Advance buffer address
64 000342 160365 000000G          SUB     R3, CC$WCT(R5) ; Reduce remaining word count
65 000346 001336                   BNE     1$             ; Loop if more blocks need to be read
66                               ;
67                               ; We were able to satisfy entire read request from the cache.
68                               ;
69 000350 004767 002772          CALL    CSHFIN          ; Finished I/O operation
70 000354 000402          BR     9$
71                               ;
72                               ; Not all of the data is in the cache.
73                               ; Read remainder in from I/O device.
74                               ;
75 000356 004767 000006   2$:    CALL    READIO          ; Start read from I/O device
76                               ;
77                               ; Finished
78                               ;
79 000362 012603   9$:    MOV     (SP)+, R3
80 000364 012601          MOV     (SP)+, R1
81 000366 000207          RETURN
```

```
1          .SBTTL  READIO -- Start read I/O operation
2          ;-----
3          ; READIO is called to start an I/O operation to read data that is not
4          ; available in the cache.
5          ;
6          ; Inputs:
7          ;   R5 = Pointer to cache control block.
8          ;
9          ; Results:
10         ;   Data is read into user's buffer and then is moved into cache.
11         ;
12 000370 010146 READIO: MOV      R1,-(SP)
13         ;
14         ; Build cache block descriptors for all the blocks that are going
15         ; to be read. The descriptors indicate that the cache buffers are empty.
16         ;
17 000372 004767 000756          CALL    CSHBLD          ;Build empty cache block descriptors
18         ;
19         ; Get and initialize a free I/O queue element
20         ;
21 000376 004767 002766          CALL    CSHGTQ          ;Get a free I/O queue element
22         ;
23         ; Set address of completion routine
24         ;
25 000402          .ADDR  #REDCPL,R0          ;Get address of read completion routine (PIC)
26 000410 010061 0000009        MOV     R0,Q.COMP(R1) ;Set address of read completion routine
27         ;
28         ; Initiate the read operation
29         ;
30 000414 004737 0000000        CALL    @#SYQID          ;Queue the I/O operation
31         ;
32         ; Finished
33         ; When the read finishes the REDCPL routine will be called.
34         ;
35 000420 012601          MOV     (SP)+,R1
36 000422 000207          RETURN
```



```

1          .SBTTL  REDCPL -- Read I/O completion routine
2          ;-----
3          ; REDCPL is called as a completion routine when a read operation completes.
4          ;
5          ; Inputs:
6          ; R1 = Address of cache control block.
7          ;
8 000424 010546 REDCPL: MOV      R5, -(SP)
9 000426 010105          MOV      R1, R5          ; Carry control block address in R5
10         ;
11         ; If end of file or hardware error was reported by handler,
12         ; terminate the operation.
13         ;
14 000430 016500 000000G          MOV      CC#QOE(R5), R0  ; Get address of original I/O queue element
15 000434 016000 000000G          MOV      Q.CSW(R0), R0  ; Get address of channel block
16 000440 032760 000000C 000000G BIT      #<CS#EOF!CS#ERR>, C.CSW(R0) ; End of file or error?
17 000446 001002          BNE      B$          ; Br if yes
18         ;
19         ; Move each block of data that was read from user's buffer to cache buffer
20         ;
21 000450 004767 000164          1$:    CALL      CSHUP          ; Move data from user's buffer to cache
22         ;
23         ; Finished with operation
24         ;
25 000454 004767 002666          B$:    CALL      CSHFIN         ; Finished with entire operation
26         ;
27         ; Finished
28         ;
29 000460 005067 177332          CLR      IOQOUT          ; Say we have finished with secondary queue
30 000464 012605          MOV      (SP)+, R5
31 000466 000207          RETURN

```

```

1          .SBTTL  CSHWRT -- Write request
2          ;-----
3          ; Process a write request to a cached device.
4          ; This involves moving the data being written from the user's buffer
5          ; into the cache buffers.
6          ;
7          ; Inputs:
8          ; R5 = Pointer to cache control block.
9          ;
10         000470 010146 CSHWRT: MOV      R1,-(SP)
11         ;
12         ; Get and setup a free I/O queue element
13         ;
14         000472 004767 002672          CALL    CSHGTQ          ;Get a free I/O queue element
15         000476 005461 000000G        NEG     Q.WCNT(R1)      ;Negative word count ==> write operation
16         000502          .ADDR #WRTCPL,R0      ;Get address of I/O completion routine
17         000510 010061 000000G        MOV     R0,Q.COMP(R1)  ;Set address of I/O completion routine
18         ;
19         ; Start the I/O operation
20         ;
21         000514 112765 000200 000000G  MOVVB  #200,CC#WFL(R5) ;Set flag saying I/O in progress
22         000522 004737 000000G        CALL    @#SYQIO        ;Initiate the I/O operation
23         ;
24         ; Count the total number of blocks that are being written
25         ;
26         000526 012700 000001          MOV     #1,R0          ;Count 1 more write operation
27         000532 004167 002742          JSR    R1,CSHCNT      ;Update statistics counter
28         000536 000000G        .WORD  CASTWO        ;Count 1 more write operation
29         000540 016500 000000G        MOV     CC#WCT(R5),R0 ;Get total word count
30         000544 062700 000377          ADD    #255,R0        ;Round up to block size
31         000550 000300          SWAB   R0             ;Convert to # blocks
32         000552 042700 177400          BIC    #^C377,R0     ;Clear all but low byte
33         000556 004167 002716          JSR    R1,CSHCNT      ;Update statistics counter
34         000562 000000G        .WORD  CASTBW        ;Total number of blocks written
35         ;
36         ; Now, while the device I/O transfer is taking place, move the data
37         ; being written from the user's buffer into the cache buffers.
38         ;
39         ; Build empty cache table entries for all blocks being written.
40         ;
41         000564 004767 000564          CALL    CSHBLD        ;Build empty cache table descriptors
42         ;
43         ; Move the data from the user's buffer into the cache
44         ;
45         000570 004767 000044          CALL    CSHUP         ;Move data into cache
46         ;
47         ; See if I/O write operation finished before our data move
48         ;
49         000574 106365 000000G        ASLB   CC#WFL(R5)     ;Has the I/O operation already finished?
50         000600 103402          BCS    9$            ;Br if not -- Wait for it to finish
51         ;
52         ; I/O operation has already finished. That means the overall operation
53         ; is now finished.
54         ;
55         000602 004767 002540          CALL    CSHFIN        ;Say operation is complete
56         ;
57         ; Finished
    
```

```
58  
59 000606 012601  
60 000610 000207  
          ;  
          9#:  MOV    (SP)+,R1  
          RETURN
```

WRTCPL -- Write operation completion routine

```

1          .SBTTL  WRTCPL -- Write operation completion routine
2          ;-----
3          ; Completion routine called when the write I/O operation is finished.
4          ;
5          ; Inputs:
6          ; R1 = Pointer to cache control block.
7          ;
8 000612 010546 WRTCPL: MOV     R5, -(SP)
9 000614 010105          MOV     R1, R5          ; Carry control block pointer in R5
10         ;
11         ; See if data transfer to cache has already finished
12         ;
13 000616 106365 0000000 ASLB   CC$WFL(R5)      ; Has data transfer to cache already finished?
14 000622 103402          BCS    9$          ; Br if not
15         ;
16         ; Data transfer to cache beat us.
17         ; The overall operation is now complete.
18         ;
19 000624 004767 002516          CALL  CSHFIN          ; Finished operation
20         ;
21         ; Finished
22         ;
23 000630 005067 177162 9$:   CLR    IOQOUT          ; Say finished with secondary I/O queue element
24 000634 012605          MOV    (SP)+, R5
25 000636 000207          RETURN

```

CSHUP -- Update cache with fresh data

```

1          .SBTTL  CSHUP  -- Update cache with fresh data
2          ;-----
3          ; CSHUP is called to move data from the user's buffer into the cache.
4          ;
5          ; Inputs:
6          ; R5 = Pointer to cache control block.
7          ;
8 000640 010146 CSHUP:  MOV    R1, -(SP)
9 000642 010246        MOV    R2, -(SP)
10         ;
11         ; Determine how many words are being moved this time
12         ;
13 000644 016502 0000000 5$:  MOV    CC$WCT(R5), R2 ;Get remaining word count
14 000650 020227 000400    CMP    R2, #256. ;More than a block left?
15 000654 101402          BLOS   3$ ;Br if not
16 000656 012702 000400    MOV    #256., R2 ;Move a block this time
17         ;
18         ; See if there is a cache table entry for the next block
19         ;
20 000662 004767 000736 3$:  CALL   CSHLOC ;Is there a cache entry for the block?
21 000666 103415          BCS   1$ ;Br if not -- Don't move the data into cache
22         ;
23         ; Move a block of data from user's buffer to cache buffer
24         ;
25 000670 004767 000256    CALL   WTMOVE ;Move data from user's buffer to cache buffer
26         ;
27         ; Set word count for block in cache control table
28         ;
29 000674          MAPTD  CA$WCT ;Map to word count table
30 000702 010211    MOV    R2, (R1) ;Set word count for this block
31         ;
32         ; Count number of blocks of data that are moved into cache
33         ;
34 000704 012700 000001    MOV    #1, R0 ;One more block moved into cache
35 000710 004167 002564    JSR   R1, CSHCNT ;Update statistics counter
36 000714 0000000 .WORD  CASCUP ;Number of blocks moved into cache
37         ;
38         ; See if the job scheduler needs to be called to give someone else a
39         ; chance to run
40         ;
41 000716 004767 002664    CALL   CSHSCH ;See if job scheduler needs to be called
42         ;
43         ; Advance block number, buffer address, and decrement remaining word count.
44         ;
45 000722 005265 0000000 1$:  INC    CC$BLK(R5) ;Increment block number
46 000726 062765 000010 0000000  ADD    #8., CC$UBP(R5) ;Advance buffer address
47 000734 160265 0000000  SUB    R2, CC$WCT(R5) ;Reduce remaining word count
48 000740 001341          BNE   5$ ;Loop if more left to move
49         ;
50         ; Finished
51         ;
52 000742 012602          MOV    (SP)+, R2
53 000744 012601          MOV    (SP)+, R1
54 000746 000207          RETURN

```

RDMOVE -- Move data from cache to user's buffer

```

1          .SBTTL  RDMOVE -- Move data from cache to user's buffer
2          ;-----
3          ; RDMOVE is called to move up to a block of data from a cache buffer
4          ; to the user's buffer.
5          ;
6          ; Inputs:
7          ; R5 = Pointer to cache control block.
8          ;
9 000750 010246 RDMOVE: MOV     R2,-(SP)
10 000752 010446      MOV     R4,-(SP)
11 000754 010546      MOV     R5,-(SP)
12 000756 010502      MOV     R5,R2          ; Carry control block pointer in R2
13          ;
14          ; Enter system state so we can use PAR5 and PAR6 to map to the cache
15          ; and user buffers.
16          ;
17 000760          . ADDR   #9$,R0          ; Go to 9$ on exit from system state
18 000766 004737 0000000 CALL   @#ENSYS          ; Enter system state
19 000772 0000000      . WORD  FP$MOV          ; Set fork priority for ENSYS
20          ;
21          ; We are now running in system state on the interrupt stack.
22          ; Kernel PAR5 and PAR6 are free.
23          ; Setup buffer pointers.
24          ;
25 000774 016237 0000000 0000000 MOV    CC$UBP(R2),@#KPAR5 ; Set PAR base for user's buffer
26 001002 016237 0000000 0000000 MOV    CC$CBP(R2),@#KPAR6 ; Set PAR base for cache buffer
27 001010 012704 0000000      MOV    #V$PAR5,R4          ; Get virtual address base for user's buffer
28 001014 066204 0000000      ADD    CC$UBO(R2),R4      ; Add offset within 64-byte block region
29 001020 012705 0000000      MOV    #V$PAR6,R5          ; Get virtual address base for cache buffer
30          ;
31          ; Get number of words to move
32          ;
33 001024 016200 0000000      MOV    CC$WCT(R2),R0      ; Get total remaining word count
34 001030 020027 000400      CMP    R0,#256          ; More than a block wanted?
35 001034 101402          BLOS   5$              ; Br if not
36 001036 012700 000400      MOV    #256.,R0         ; Move 1 block this time
37          ;
38          ; Move as many blocks of 16 words as possible
39          ;
40 001042 010002          5$:  MOV    R0,R2          ; Get total number of words to move
41 001044 072227 177774      ASH   #-4,R2           ; Compute # blocks of 16 words to move
42 001050 001421          BEQ   6$              ; Br if less than 16 words need to be moved
43 001052 012524          7$:  MOV    (R5)+,(R4)+      ; Move 1
44 001054 012524      MOV    (R5)+,(R4)+      ; Move 2
45 001056 012524      MOV    (R5)+,(R4)+      ; Move 3
46 001060 012524      MOV    (R5)+,(R4)+      ; Move 4
47 001062 012524      MOV    (R5)+,(R4)+      ; Move 5
48 001064 012524      MOV    (R5)+,(R4)+      ; Move 6
49 001066 012524      MOV    (R5)+,(R4)+      ; Move 7
50 001070 012524      MOV    (R5)+,(R4)+      ; Move 8
51 001072 012524      MOV    (R5)+,(R4)+      ; Move 9
52 001074 012524      MOV    (R5)+,(R4)+      ; Move 10
53 001076 012524      MOV    (R5)+,(R4)+      ; Move 11
54 001100 012524      MOV    (R5)+,(R4)+      ; Move 12
55 001102 012524      MOV    (R5)+,(R4)+      ; Move 13
56 001104 012524      MOV    (R5)+,(R4)+      ; Move 14
57 001106 012524      MOV    (R5)+,(R4)+      ; Move 15

```

```
58 001110 012524          MOV      (R5)+,(R4)+      ;Move 16
59 001112 077221          SDB      R2,7#           ;Loop if more blocks of 16 left to move
60                          ;
61                          ; Now move any residual words (less than 16)
62                          ;
63 001114 042700 177760   6#:      BIC      #^C17,R0      ;Any words left to move?
64 001120 001407          BEQ      4#              ;Br if not
65 001122 006200          ASR      R0              ;Get number of doublewords to move
66 001124 001403          BEQ      3#              ;Br if less than 2 words to move
67 001126 012524          2#:      MOV      (R5)+,(R4)+      ;Move a word
68 001130 012524          MOV      (R5)+,(R4)+      ;Move second word of pair
69 001132 077003          SDB      R0,2#           ;Loop if more doublewords left to move
70 001134 103001          3#:      BCC      4#              ;Br if don't have odd word left to move
71 001136 011514          MOV      (R5),(R4)       ;Move last word
72                          ;
73                          ; Finished moving data.
74                          ; Exit from system state (go to 9#)
75                          ;
76 001140 000207          4#:      RETURN          ;Exit from system state -- go to 9#
77                          ;
78                          ; Finished
79                          ;
80 001142 012605          9#:      MOV      (SP)+,R5
81 001144 012604          MOV      (SP)+,R4
82 001146 012602          MOV      (SP)+,R2
83 001150 000207          RETURN
```

```

1          .SBTTL  WTMOVE -- Move data from user's buffer to cache
2          ;-----
3          ; WTMOVE is called to move a block of data from the user's buffer into
4          ; a cache buffer.
5          ;
6          ; Inputs:
7          ; R5 = Pointer to cache control block.
8          ;
9 001152 010246 WTMOVE: MOV     R2, -(SP)
10 001154 010446      MOV     R4, -(SP)
11 001156 010546      MOV     R5, -(SP)
12 001160 010502      MOV     R5, R2          ; Carry control block pointer in R2
13          ;
14          ; Enter system state so we can use PAR5 and PAR6 to map to the cache
15          ; and user buffers.
16          ;
17 001162          .ADDR  #9$, R0          ; Go to 9$ on exit from system state
18 001170 004737 000000G CALL   @#ENSYS          ; Enter system state
19 001174 000000G      .WORD  FP$MOV          ; Set fork priority level
20          ;
21          ; We are now running in system state on the interrupt stack.
22          ; Kernel PAR5 and PAR6 are free.
23          ; Setup buffer pointers.
24          ;
25 001176 016237 000000G 000000G      MOV     CC$UBP(R2), @#KPAR5 ; Set PAR base for user's buffer
26 001204 016237 000000G 000000G      MOV     CC$CBP(R2), @#KPAR6 ; Set PAR base for cache buffer
27 001212 012704 000000G      MOV     #VPAR5, R4          ; Get virtual address base for user's buffer
28 001216 066204 000000G      ADD     CC$UBO(R2), R4      ; Add offset within 64-byte block region
29 001222 012705 000000G      MOV     #VPAR6, R5          ; Get virtual address base for cache buffer
30          ;
31          ; Get number of words to move
32          ;
33 001226 016200 000000G      MOV     CC$WCT(R2), R0      ; Get total remaining word count
34 001232 020027 000400      CMP     R0, #256.          ; More than a block wanted?
35 001236 101402          BLOS   5$                ; Br if not
36 001240 012700 000400      MOV     #256., R0          ; Move 1 block this time
37          ;
38          ; Move as many blocks of 16 words as possible
39          ;
40 001244 010002          5$:  MOV     R0, R2          ; Get total number of words to move
41 001246 072227 177774      ASH   #-4, R2            ; Get # blocks of 16 words to move
42 001252 001421          BEQ   6$                ; Br if less than 16 words need to be moved
43 001254 012425          7$:  MOV     (R4)+, (R5)+      ; Move 1
44 001256 012425      MOV     (R4)+, (R5)+      ; Move 2
45 001260 012425      MOV     (R4)+, (R5)+      ; Move 3
46 001262 012425      MOV     (R4)+, (R5)+      ; Move 4
47 001264 012425      MOV     (R4)+, (R5)+      ; Move 5
48 001266 012425      MOV     (R4)+, (R5)+      ; Move 6
49 001270 012425      MOV     (R4)+, (R5)+      ; Move 7
50 001272 012425      MOV     (R4)+, (R5)+      ; Move 8
51 001274 012425      MOV     (R4)+, (R5)+      ; Move 9
52 001276 012425      MOV     (R4)+, (R5)+      ; Move 10
53 001300 012425      MOV     (R4)+, (R5)+      ; Move 11
54 001302 012425      MOV     (R4)+, (R5)+      ; Move 12
55 001304 012425      MOV     (R4)+, (R5)+      ; Move 13
56 001306 012425      MOV     (R4)+, (R5)+      ; Move 14
57 001310 012425      MOV     (R4)+, (R5)+      ; Move 15

```



```
58 001312 012425          MOV      (R4)+,(R5)+      ; Move 16
59 001314 077221          SOB      R2,7$             ; Loop if more blocks of 16 to move
60                          ;
61                          ; Now move any remaining words (less than 16)
62                          ;
63 001316 042700 177760   6$:      BIC      #^C17,R0      ; Any words left to move?
64 001322 001407          BEQ      4$              ; Br if not
65 001324 006200          ASR      R0              ; Get number of doublewords to move
66 001326 001403          BEQ      3$              ; Br if less than 2 words to move
67 001330 012425          2$:      MOV      (R4)+,(R5)+      ; Move a word
68 001332 012425          MOV      (R4)+,(R5)+      ; Move second word of pair
69 001334 077003          SOB      R0,2$          ; Loop if more doublewords left to move
70 001336 103001          3$:      BCC      4$              ; Br if don't have odd word left to move
71 001340 011415          MOV      (R4),(R5)       ; Move last word
72                          ;
73                          ; Finished moving data.
74                          ; Exit from system state (go to 9$)
75                          ;
76 001342 000207          4$:      RETURN          ; Exit from system state -- go to 9$
77                          ;
78                          ; Finished
79                          ;
80 001344 012605          9$:      MOV      (SP)+,R5
81 001346 012604          MOV      (SP)+,R4
82 001350 012602          MOV      (SP)+,R2
83 001352 000207          RETURN
```

```

1          .SBTTL  CSHBLD -- Build empty cache descriptors for new entries
2          ;-----
3          ; CSHBLD is called when we need to move new data into the cache.
4          ; It builds cache block descriptors for all blocks that will be added
5          ; to the cache and marks the descriptors as empty (CC$WCT=0).
6          ;
7          ; Inputs:
8          ; R5 = Pointer to cache control block.
9          ;
10         CSHBLD: MOV     R1,-(SP)
11                MOV     R4,-(SP)
12         001354 010146      MOV     CC$BLK(R5),-(SP); Save starting block number
13         001356 010446      MOV     CC$WCT(R5),R4  ; Get total word count
14         001360 016546 0000000
15         001364 016504 0000000
16          ;
17          ; Begin loop to build an entry for each block.
18          ; See if there is already an entry for the next block.
19         1$: CALL     CSHLOC      ; Is there an entry for the next block?
20                BCS     3$        ; Br if not
21          ;
22          ; An entry already exists for the next block.
23          ; Mark the entry as empty.
24                MAPTO   CA$WCT      ; Map to word count table
25                CLR     (R1)        ; Say entry is empty
26          ;
27          ; Move this cache block to the head of the LRU chain
28          ; (That is, make it the most recently used)
29          ;
30         CALL     CSHUSE      ; Make this block be the most recently used
31         BR      2$
32          ;
33          ; There is not an existing entry for this block.
34          ; See if we can get a free cache block entry.
35          ;
36         3$: CALL     CSHGET      ; Try to get a free cache block entry
37                BCC     5$        ; Br if we got a free entry
38          ;
39          ; There are no free cache block entries.
40          ; Get the least-recently-used entry and use it.
41          ;
42         MOV     @#CSHLRU,R1    ; Get pointer to least-recently-used entry
43         001422 013701 0000000  CALL     CSHREM      ; Remove that entry from all lists
44         001426 004767 001006
45          ;
46          ; Set up descriptive information about this entry
47          ;
48         5$: MAPTO   CA$BLK      ; Point to block number table
49         001432 001432 0000000  MOV     CC$BLK(R5),(R1) ; Set block number
50         001440 016511 0000000  MAPTO   CA$DVU      ; Map to device & unit number table
51         001444 001444 0000000  MOV     CC$DVU(R5),(R1) ; Set device and unit number
52         001452 016511 0000000  MAPTO   CA$WCT      ; Map to word count table
53         001456 001456 0000000  CLR     (R1)        ; Say entry is empty
54         001464 005011
55          ;
56          ; Add new entry to lists
57         CALL     CSHADD      ; Add new cache entry to appropriate lists

```

```
58          ; Advance block number and see if there are more entries to build
59          ;
60 001472 005265 0000000 2#:      INC      CC#BLK(R5)      ;Increment block number
61 001476 010400          ;      MOV      R4,R0          ;Get remaining word count
62 001500 020027 000400          ;      CMP      R0,#256.      ;More than a block?
63 001504 101402          ;      BLOS     4$           ;Br if not
64 001506 012700 000400          ;      MOV      #256.,R0      ;# words moved this time
65 001512 160004          4#:      SUB      R0,R4          ;Need to build more entries?
66 001514 001325          ;      BNE     1$           ;Br if yes
67          ;
68          ; Finished
69          ;
70 001516 012665 0000000          ;      MOV      (SP)+,CC#BLK(R5);Restore original block number
71 001522 012604          ;      MOV      (SP)+,R4
72 001524 012601          ;      MOV      (SP)+,R1
73 001526 000207          ;      RETURN
```

CSHCLN -- Remove all entries for a specified device

```

1          . SBTTL  CSHCLN -- Remove all entries for a specified device
2          ;-----
3          ; CSHCLN is called to remove from the cache all entries for a specified
4          ; device.
5          ;
6          ; Inputs:
7          ; R3 = Unit # (high byte), device # (low byte)
8          ;
9 001530 010146 CSHCLN: MOV      R1,-(SP)
10 001532 010246      MOV      R2,-(SP)
11 001534 013746 0000000      MOV      @#KPAR5,-(SP) ; Save system overlay mapping status
12          ;
13          ; Determine if caching has been disabled
14          ;
15 001540 005737 0000000      TST      @#VCSHNB      ; Are there any active cache buffers?
16 001544 001422      BEQ      9$          ; Br if not
17          ;
18          ; Begin search through list of all cache entries.
19          ;
20 001546 013701 0000000      MOV      @#CSHLRU,R1      ; Point to least-recently-used entry
21 001552 001417      BEQ      9$          ; Br if no entries on use list
22          ;
23          ; See if this entry is for a block on specified device
24          ;
25 001554      1$:      MAPTO    CA$UBL      ; Map to backward link
26 001562 011102      MOV      (R1),R2      ; Get pointer to next entry to check
27 001564      MAPTO    CA$DVU      ; Map to device & unit # table
28 001572 020311      CMP      R3,(R1)      ; Is this entry for specified device?
29 001574 001004      BNE      2$          ; Br if not
30          ;
31          ; We found an entry that needs to be deallocated.
32          ; Remove it from all lists.
33          ;
34 001576 004767 000636      CALL     CSHREM      ; Remove entry from all lists
35          ;
36          ; Put entry on the free list.
37          ;
38 001602 004767 001070      CALL     CSHFRE      ; Put entry on the free list
39          ;
40          ; Check next entry
41          ;
42 001606 010201      2$:      MOV      R2,R1      ; Get pointer to next entry to check
43 001610 001361      BNE      1$          ; Br if there is another to check
44          ;
45          ; Finished
46          ;
47 001612 012637 0000000      9$:      MOV      (SP)+,@#KPAR5 ; Restore system overlay mapping status
48 001616 012602      MOV      (SP)+,R2
49 001620 012601      MOV      (SP)+,R1
50 001622 000207      RETURN

```

```

1          .SBTTL  CSHLOC -- Determine if entry exists in cache table
2          ;-----
3          ; Determine if an entry exists in the cache for a specified block
4          ; on a specified device.
5          ;
6          ; Inputs:
7          ; R5 = Pointer to cache control block.
8          ; CC$BLK(R5) = Block number.
9          ; CC$DVU(R5) = Device and unit numbers.
10         ;
11         ; Outputs:
12         ; C-flag cleared ==> Found entry.
13         ; C-flag set    ==> No entry for block in cache.
14         ; R1 = Cache table index.
15         ; CC$CBP(R5) = 64-byte par offset to start of cache buffer.
16         ;
17 001624 010246 CSHLOC: MOV     R2,-(SP)
18 001626 010346      MOV     R3,-(SP)
19         ;
20         ; Compute hash head index for this block
21         ;
22 001630 016503 0000000  MOV     CC$BLK(R5),R3 ;Get block number
23 001634 005002      CLR     R2             ;Clear for divide
24 001636 071237 0000000  DIV     @#VCSHNB,R2   ;Get block number mod number of hash heads
25 001642 006303      ASL     R3             ;Convert remainder to word table index
26 001644 062703 0000000  ADD     #VPAR5,R3     ;Add virtual address base
27         ;
28         ; Search hash list for the specified block
29         ;
30 001650 016502 0000000  MOV     CC$BLK(R5),R2 ;Get requested block number
31 001654 016500 0000000  MOV     CC$DVU(R5),R0 ;Get device & unit number
32 001660      MAPTO  CA$HSH ;Map to hash list heads
33 001666 011301      MOV     (R3),R1       ;Get pointer to 1st entry on hash list
34 001670 001434      BEQ     B$           ;Br if no entries on this hash list
35 001672      1$: MAPTO  CA$BLK ;Map to block number entries
36 001700 020211      CMP     R2,(R1)       ;Is entry for desired block?
37 001702 101006      BHI     2$           ;No, but there may be more entries
38 001704 103426      BLO     B$           ;No, entry is not on the list
39 001706      MAPTO  CA$DVU ;Map to device and unit numbers
40 001714 020011      CMP     R0,(R1)       ;Is this entry for correct device?
41 001716 001406      BEQ     3$           ;Br if yes -- We found the entry
42 001720      2$: MAPTO  CA$HFL ;Map to hash forward links
43 001726 011101      MOV     (R1),R1       ;Get pointer to following entry
44 001730 001360      BNE     1$           ;Loop if there is another entry
45 001732 000413      BR      B$           ;Entry does not exist
46         ;
47         ; We found the entry.
48         ; Set up information about this entry
49         ;
50 001734 010103      3$: MOV     R1,R3       ;Get address of entry
51 001736 162703 0000000  SUB     #VPAR5,R3     ;Subtract address bias
52 001742 072327 0000002  ASH     #2,R3         ;Compute corresponding buffer 64-byte #
53 001746 063703 0000000  ADD     @#CSHBFP,R3   ;Add base 64-byte block # of buffer area
54 001752 010365 0000000  MOV     R3,CC$CBP(R5) ;Save 64-byte pointer to cache buffer base
55 001756 000241      CLC     ;Signal success on return
56 001760 000401      BR      9$
57         ;

```

```
58          ; No entry exists for this block
59          ;
60 001762 000261      B#:      SEC          ;Signal failure on return
61          ;
62          ; Finished
63          ;
64 001764 012603      9#:      MOV      (SP)+,R3
65 001766 012602      MOV      (SP)+,R2
66 001770 000207      RETURN
```

CSHUSE -- Make cache entry be most-recently-used

```

1          .SBTTL  CSHUSE -- Make cache entry be most-recently-used
2          ;-----
3          ; CSHUSE is called to make a specified cache entry be the
4          ; most recently used entry.
5          ;
6          ; Inputs:
7          ; R1 = Cache entry to be made most-recently-used.
8          ;
9 001772 010246 CSHUSE: MOV     R2, -(SP)
10 001774 010346      MOV     R3, -(SP)
11          ;
12          ; See if entry already is most recently used
13          ;
14 001776 020137 0000000  CMP     R1, @#CSHMRU    ; Is entry already most recently used?
15 002002 001446      BEQ     9$                ; Br if yes
16          ;
17          ; Remove entry from LRU list
18          ;
19 002004      MAPTO   CA$UFL        ; Map to LRU forward link
20 002012 011103      MOV     (R1), R3        ; Get pointer to entry that follows us
21 002014      MAPTO   CA$UBL        ; Map to LRU backward link
22 002022 011102      MOV     (R1), R2        ; Get pointer to entry that precedes us
23 002024 001003      BNE     1$                ; Br if we are not at the head of the list
24 002026 010337 0000000  MOV     R3, @#CSHMRU    ; Say following entry is now head of list
25 002032 000404      BR      2$
26 002034      1$:     MAPTO   CA$UFL        ; Map to LRU forward link
27 002042 010312      MOV     R3, (R2)        ; Make previous entry point to following entry
28 002044 005703      2$:     TST     R3                ; Is there a following entry?
29 002046 001003      BNE     3$                ; Br if yes
30 002050 010237 0000000  MOV     R2, @#CSHLRU    ; Say previous entry is now last entry in list
31 002054 000404      BR      4$
32 002056      3$:     MAPTO   CA$UBL        ; Map to LRU backward link
33 002064 010213      MOV     R2, (R3)        ; Make following entry point to previous entry
34          ;
35          ; Add entry to end of LRU list
36          ;
37 002066 013702 0000000  4$:     MOV     @#CSHMRU, R2    ; Get index to current MRU entry
38 002072      MAPTO   CA$UBL        ; Map to backward link table
39 002100 010112      MOV     R1, (R2)        ; Make previous MRU entry point back to us
40 002102 005011      CLR     (R1)                ; Make our backward link 0
41 002104      MAPTO   CA$UFL        ; Map to forward link table
42 002112 010211      MOV     R2, (R1)        ; Make us point to previous MRU entry
43 002114 010137 0000000  MOV     R1, @#CSHMRU    ; Say we are MRU entry
44          ;
45          ; Finished
46          ;
47 002120 012603      9$:     MOV     (SP)+, R3
48 002122 012602      MOV     (SP)+, R2
49 002124 000207      RETURN

```

CSHADD --- Add cache table entry to appropriate lists

```

1          .SBTTL  CSHADD --- Add cache table entry to appropriate lists
2          ;-----
3          ; CSHADD is called to add a new cache table entry to the LRU and Hash lists.
4          ; The entry is placed at the head of the LRU list and is placed on the
5          ; appropriate hash list based on its block number.
6          ;
7 002126  010246  CSHADD: MOV      R2,-(SP)
8 002130  010346      MOV      R3,-(SP)
9 002132  010446      MOV      R4,-(SP)
10         ;
11         ; Add entry to end of LRU list
12         ; (That is, make it the most recently used entry)
13         ;
14 002134  013702  0000000  MOV      @#CSHMRU,R2      ;Get index to current MRU entry
15 002140  001414      BEQ      4$              ;Br if list is empty
16 002142      MAPTD   CA$UBL      ;Map to backward link table
17 002150  010112  MOV      R1,(R2)        ;Make previous MRU entry point back to us
18 002152  005011  CLR      (R1)          ;Make our backward link 0
19 002154      MAPTD   CA$UFL      ;Map to forward link table
20 002162  010211  MOV      R2,(R1)        ;Make us point to previous MRU entry
21 002164  010137  0000000  MOV      R1,@#CSHMRU    ;Say we are MRU entry
22 002170  000414      BR       5$
23         ;
24         ; We are the only entry on the LRU list
25         ;
26 002172  010137  0000000  4$:  MOV      R1,@#CSHMRU    ;Say we are the most recently used entry
27 002176  010137  0000000  MOV      R1,@#CSHLRU    ;And the least recently used entry
28 002202      MAPTD   CA$UFL      ;Map to forward link list
29 002210  005011  CLR      (R1)          ;No forward link for us
30 002212      MAPTD   CA$UBL      ;Map to backward link list
31 002220  005011  CLR      (R1)          ;No backward link for us
32         ;
33         ; Now add entry to the appropriate hash list based on the block number.
34         ;
35 002222  5$:  MAPTD   CA$BLK      ;Map to block # table
36 002230  MOV      (R1),R3    ;Get block # of this entry
37 002232  005002  CLR      R2            ;Clear for divide
38 002234  071237  0000000  DIV      @#VCSHNB,R2    ;Make block # mod number of list heads
39 002240  006303  ASL      R3            ;Convert remainder to word table index
40 002242  062703  0000000  ADD      #VPAR5,R3      ;All addresses are relative to par5
41 002246      MAPTD   CA$HSH      ;Map to hash list heads
42 002254  011302  MOV      (R3),R2      ;Get current entry at front of list
43 002256  001012  BNE     1$            ;Br if list has some entry
44         ;
45         ; The hash list is currently empty.
46         ; Make our item be only entry on list.
47         ;
48 002260  010113  MOV      R1,(R3)      ;Make hash list head point to us
49 002262      MAPTD   CA$HFL      ;Map to hash forward link table
50 002270  005011  CLR      (R1)          ;Say we are last entry on list
51 002272      MAPTD   CA$HBL      ;Map to hash backward link table
52 002300  005011  CLR      (R1)          ;Say we are first entry on list
53 002302  000452  BR       9$
54         ;
55         ; There are some entries on the hash list.
56         ; Insert our entry in front of 1st entry with same or higher block number.
57         ;

```



```

58 002304          1$:      MAPTO  CA$BLK      ;Map to block # table
59 002312  011104      MOV    (R1),R4      ;Get our block number
60 002314          2$:      MAPTO  CA$BLK      ;Map to block number table
61 002322  020412      CMP    R4,(R2)      ;Search for 1st entry with same or higher blk
62 002324  101415      BLOS   3$          ;Br if found insert point
63 002326  010200      MOV    R2,R0          ;Save address of last entry checked
64 002330          MAPTO  CA$HFL      ;Map to hash forward link
65 002336  011202      MOV    (R2),R2      ;Get pointer to next entry on this list
66 002340  001365      BNE   2$          ;Loop if there is another one
67
68                ; Add our entry to the tail of the hash list
69                ;
70 002342  010110      MOV    R1,(R0)      ;Make last forward link point to us
71 002344  005011      CLR   (R1)          ;Say nothing follows us
72 002346          MAPTO  CA$HBL      ;Map to hash backward link table
73 002354  010011      MOV    R0,(R1)      ;Set pointer to previous entry
74 002356  000424      BR    9$
75
76                ; Insert our entry into list in front of entry pointed to by R2.
77                ;
78 002360          3$:      MAPTO  CA$HFL      ;Map to hash forward links
79 002366  010211      MOV    R2,(R1)      ;Remember which entry follows us
80 002370          MAPTO  CA$HBL      ;Map to hash backward links
81 002376  011200      MOV    (R2),R0      ;Find out which entry precedes us
82 002400  010112      MOV    R1,(R2)      ;Make following entry point back to us
83 002402  010011      MOV    R0,(R1)      ;Set our backward pointer
84 002404  001005      BNE   6$          ;Br if we are not at head of list
85 002406          MAPTO  CA$HSH      ;Map to hash list head
86 002414  010113      MOV    R1,(R3)      ;Make list head point to us
87 002416  000404      BR    9$
88 002420          6$:      MAPTO  CA$HFL      ;Map to forward links
89 002426  010110      MOV    R1,(R0)      ;Make previous entry point to us
90
91                ; Finished
92                ;
93 002430  012604          9$:      MOV    (SP)+,R4
94 002432  012603      MOV    (SP)+,R3
95 002434  012602      MOV    (SP)+,R2
96 002436  000207      RETURN
    
```

```

1          .SBTTL  CSHREM -- Remove cache block entry from all lists
2          ;-----
3          ; CSHREM is called to remove a cache block entry from the LRU list
4          ; and from the hash list.
5          ;
6          ; Inputs:
7          ; R1 = Index to cache block entry to be removed.
8          ;
9 002440 010246 CSHREM: MOV     R2, -(SP)
10 002442 010346      MOV     R3, -(SP)
11 002444 010446      MOV     R4, -(SP)
12          ;
13          ; Remove entry from LRU list
14          ;
15 002446          MAPTO  CA$UFL      ;Map to LRU forward link
16 002454 011103      MOV     (R1),R3      ;Get pointer to entry that follows us
17 002456          MAPTO  CA$UBL      ;Map to LRU backward link
18 002464 011102      MOV     (R1),R2      ;Get pointer to entry that precedes us
19 002466 001003      BNE     1$          ;Br if we are not at the head of the list
20 002470 010337 0000000      MOV     R3, @#CSHMRU    ;Say following entry is now head of list
21 002474 000404      BR      2$
22 002476          1$:  MAPTO  CA$UFL      ;Map to LRU forward link
23 002504 010312      MOV     R3, (R2)      ;Make previous entry point to following entry
24 002506 005703      2$:  TST     R3          ;Is there a following entry?
25 002510 001003      BNE     3$          ;Br if yes
26 002512 010237 0000000      MOV     R2, @#CSHLRU    ;Say previous entry is now last entry in list
27 002516 000404      BR      4$
28 002520          3$:  MAPTO  CA$UBL      ;Map to LRU backward link
29 002526 010213      MOV     R2, (R3)      ;Make following entry point to previous entry
30          ;
31          ; Remove entry from hash list.
32          ; Compute the hash table index.
33          ;
34 002530          4$:  MAPTO  CA$BLK      ;Map to block number table
35 002536 011103      MOV     (R1),R3      ;Get block number
36 002540 005002      CLR     R2          ;Clear for divide
37 002542 071237 0000000      DIV     @#VCSHNB,R2    ;Get index modulus number of hash list heads
38 002546 006303      ASL     R3          ;Convert remainder to word table index
39 002550 062703 0000000      ADD     #VPAR5,R3      ;Bias by PAR5 base
40 002554 010304      MOV     R3,R4          ;Save hash head index
41          ;
42          ; Get our hash forward and backward links
43          ;
44 002556          MAPTO  CA$HFL      ;Map to hash forward link
45 002564 011103      MOV     (R1),R3      ;Get pointer to following entry
46 002566          MAPTO  CA$HBL      ;Map to hash backward link
47 002574 011102      MOV     (R1),R2      ;Get pointer to previous entry
48 002576 001405      BEQ     5$          ;Br if we are at the head of the list
49          ;
50          ; We are not the 1st entry in the list.
51          ;
52 002600          MAPTO  CA$HFL      ;Map to hash forward link
53 002606 010312      MOV     R3, (R2)      ;Make previous entry point to following entry
54 002610 000404      BR      6$
55          ;
56          ; We are the 1st entry in the list
57          ;

```

```
58 002612          5$:      MAPTD  CA$HSH      ;Map to hash list heads
59 002620  010314      MOV    R3,(R4)    ;Make hash list head point to following entry
60
61                ; Update entry that follows us
62
63 002622  005703      6$:      TST    R3          ;Is there a following entry?
64 002624  001404      BEQ    9$          ;Br if not
65 002626          MAPTD  CA$HBL      ;Map to hash backward link
66 002634  010213      MOV    R2,(R3)    ;Make following entry point to preceding entry
67
68                ; Finished
69
70 002636  012604      9$:      MOV    (SP)+,R4
71 002640  012603      MOV    (SP)+,R3
72 002642  012602      MOV    (SP)+,R2
73 002644  000207      RETURN
```

```
1          .SBTTL  CSHGET -- Try to get a free cache block entry
2          ;-----
3          ; CSHGET is called to try to get a free cache block entry.
4          ;
5          ; Outputs:
6          ; C-flag cleared ==> A free cache block entry was gotten.
7          ; C-flag set      ==> There are no free cache blocks available.
8          ; R1 = Address of free cache block.
9          ;
10         CSHGET:
11         ;
12         ; Get entry off of front of free list
13         ;
14         002646 013701 0000000  MOV     @#CSHFHD,R1    ;Is there a free entry?
15         002652 001407          BEQ     B#                ;Br if not
16         ;
17         ; We got a free entry, remove it from the list.
18         ;
19         002654          MAPTO  CA#UFL          ;Map to forward link list
20         002662 011137 0000000  MOV     (R1),@#CSHFHD ;Remove entry from list
21         002666 000241          CLC                    ;Signal success on return
22         002670 000401          BR     9#
23         ;
24         ; There are no free entries
25         ;
26         002672 000261 8#:     SEC                    ;Signal failure on return
27         ;
28         ; Finished
29         ;
30         002674 000207 9#:     RETURN
```

CSHFRE -- Return a cache block entry to the free list

```
1 .SBTTTL CSHFRE -- Return a cache block entry to the free list
2 ;-----
3 ; CSHFRE is called to return a cache block entry to the free list.
4 ;
5 ; Inputs:
6 ; R1 = Index of cache block entry to be freed.
7 ;
8 002676 013700 0000000 CSHFRE: MOV @CSHFHD,R0 ;Get pointer to 1st entry on free list now
9 002702 010137 0000000 MOV R1,@CSHFHD ;Say we are 1st entry on free list
10 002706 MAPTO CA#UFL ;Map to forward link list
11 002714 010011 MOV RO,(R1) ;Make us point to next entry on free list
12 002716 000207 RETURN
```

CSHTST -- Determine if device is mounted

```

1          .SBTTL  CSHTST -- Determine if device is mounted
2          ;-----
3          ; CSHTST is called to determine if the device associated with the current
4          ; I/O operation is mounted and should be cached.
5          ;
6          ; Inputs:
7          ; R1 = Pointer to I/O queue element.
8          ;
9          ; Outputs:
10         ; C-flag cleared ==> Device is mounted and should be cached.
11         ;
12 002720 010246 CSHTST: MOV     R2,-(SP)
13 002722 010546         MOV     R5,-(SP)
14         ;
15         ; Get device index number and unit number
16         ;
17 002724 116100 000000G         MOVB   Q.UNIT(R1),R0 ;Get unit number
18 002730 042700 177770         BIC    #^C7,R0      ;Clear all but unit number
19 002734 000300         SWAB   R0          ;Put in high-order byte
20 002736 116102 000000G         MOVB   Q.DEVX(R1),R2 ;Get device index number
21 002742 001417         BEQ    3$          ;Br if I/O operation has been cancelled
22         ;
23         ; See if this device is eligible for caching
24         ;
25 002744 032762 000000G 000000G         BIT    #DX#NCA,DVFLAG(R2);Is the no-cache flag set for this device?
26 002752 001013         BNE    3$          ;Br if caching not allowed
27         ;
28         ; Search table of mounted devices
29         ;
30 002754 050002         BIS    R0,R2          ;Combine unit number and device number
31 002756 013705 000000G         MOV    @#CSHDEV,R5      ;Point to start of table
32 002762 020265 000000G 1$:  CMP    R2,CD$DVU(R5)    ;Is this entry for this device?
33 002766 001407         BEQ    2$          ;Br if yes -- Device is mounted
34 002770 062705 000000G         ADD    #CD#$SZ,R5      ;Point to next entry
35 002774 020537 000000G         CMP    R5,@#CSHDVN     ;Checked all?
36 003000 103770         BLO    1$          ;Loop if not
37         ;
38         ; Device is not cached
39         ;
40 003002 000261 3$:  SEC          ;Signal that device is not mounted
41 003004 000401         BR     9$
42         ;
43         ; Device is mounted
44         ;
45 003006 000241 2$:  CLC          ;Signal that device is mounted
46         ;
47         ; Finished
48         ;
49 003010 012605 9$:  MOV    (SP)+,R5
50 003012 012602         MOV    (SP)+,R2
51 003014 000207         RETURN

```

CCBGET -- Get a free cache control block

```

1          .SBTTL  CCBGET -- Get a free cache control block
2          ;-----
3          ; Get a free cache control block and initialize it with information
4          ; from the I/O queue element.
5          ;
6          ; Inputs:
7          ; R1 = Pointer to I/O queue element.
8          ;
9          ; Outputs:
10         ; R5 = Pointer to cache control block.
11         ; CC#DVU(R5) = Unit number (high byte), device number (low byte).
12         ; CC#BLK(R5) = Starting block number for transfer.
13         ; CC#OQE(R5) = Address of original I/O queue element.
14         ; CC#UBP(R5) = 64-byte block number of base of user's buffer.
15         ; CC#UBD(R5) = Offset within 64-byte block of user's buffer base.
16         ; CC#WCT(R5) = Total word count.
17         ;
18 003016 010246 CCBGET: MOV      R2,-(SP)
19 003020 010346      MOV      R3,-(SP)
20         ;
21         ; See if there is a free cache control block available
22         ;
23 003022      1$:  DISABL          ;; Disable interrupts
24 003030 013705 000000G  MOV      @#CCBHD,R5      ;; Is there a free control block?
25 003034 001015          BNE      2$          ;; Br if yes
26         ;
27         ; There are no free cache control blocks.
28         ; Suspend job until one becomes available.
29         ;
30 003036 113705 000000G      MOVB     @#CORUSR,R5      ;; Get job index number
31 003042 052765 000000G 000000G  BIS      ##NOABT,LSW9(R5) ;; Disallow job aborts
32 003050 012700 000000G  MOV      #S$QCCB,R0      ;; Waiting for cache control block
33 003054 004737 000000G  CALL     @#QNSPNX        ;; Suspend job till control block freed
34 003060 042765 000000G 000000G  BIC      ##NOABT,LSW9(R5) ;; Reenable job aborts
35 003066 000755          BR       1$          ; Go try again
36         ;
37         ; There is a free cache control block.
38         ; Claim it for our job.
39         ;
40 003070 016537 000000G 000000G  2$:  MOV      CC$LNK(R5),@#CCBHD ;; Remove control block from free list
41         ;
42         ; Put control block on list for job so it will be freed if job aborts
43         ;
44 003076 013765 000000G 000000G  MOV      @#JOBCCB,CC$LNK(R5) ;; Add to list for job
45 003104 010537 000000G  MOV      R5,@#JOBCCB      ;;
46 003110 005237 000000G  INC      @#NPCCB          ;; One more pending cache control block
47 003114          ENABL          ; Enable interrupts
48         ;
49         ; Set up information in control block
50         ;
51 003122 010567 174664      MOV      R5,CURCCB        ; Save address of current control block
52 003126 010167 174662      MOV      R1,IOQIN         ; Save address of original queue element
53 003132 010165 000000G  MOV      R1,CC#OQE(R5)   ; Save address of original queue element
54 003136 016165 000000G 000000G  MOV      Q,BLKN(R1),CC#BLK(R5) ; Set block number
55 003144 016100 000000G  MOV      Q,WCNT(R1),R0   ; Get word count
56 003150 002001          BGE      3$          ; Br if positive
57 003152 005400          NEG      R0          ; Get positive word count

```

CCBGET -- Get a free cache control block

```

58 003154 010065 0000000 3#:   MOV    R0,CC#WCT(R5) ;Set word count
59 003160 116100 0000000   MOVB   Q.UNIT(R1),R0 ;Get unit number
60 003164 042700 177770    BIC    #^C7,R0 ;Clear all but unit number
61 003170 110065 0000010   MOVB   R0,CC#DVU+1(R5) ;Put unit number in high-order byte
62 003174 116165 0000000 0000000   MOVB   Q.DEVX(R1),CC#DVU(R5) ;Put device number in low-order byte
63 003202 016102 0000000   MOV    Q.BUFF(R1),R2 ;Get virtual address of user's buffer
64 003206 162702 0000000   SUB    #VPAR6,R2 ;Remove virtual address bias
65 003212 005003   CLR    R3 ;Clear for shift
66 003214 073227 177772   ASHC   #-6.,R2 ;Convert address to 64-byte block number
67 003220 066102 0000000   ADD    Q.PAR(R1),R2 ;Add base 64-byte block number
68 003224 010265 0000000   MOV    R2,CC#UBP(R5) ;This is the 64-byte block # of buffer base
69 003230 000303   SWAB   R3 ;Get byte within block to low-order of R3
70 003232 072327 177776   ASH    #-2,R3
71 003236 010365 0000000   MOV    R3,CC#UBO(R5) ;Save offset within 64-byte block
72                                     ;
73                                     ; Finished
74                                     ;
75 003242 012603   MOV    (SP)+,R3
76 003244 012602   MOV    (SP)+,R2
77 003246 000207   RETURN

```



```

1          .SBTTL  CCBFRE -- Free a cache control block
2          ;-----
3          ; Free a cache control block.
4          ;
5          ; Inputs:
6          ; R5 = Address of cache control block being freed.
7          ;
8 003250   CCBFRE:
9          ;
10         ; Remove cache control block from active list for this job
11         ;
12 003250   012700   000000C   MOV      #JOBCCB-CC$LNK,R0 ;Get dummy pointer to list head
13 003254                   DISABL                   ;** Disable interrupts **
14 003262   026005   000000G   1$:  CMP      CC$LNK(R0),R5   ;Is our block next one in list?
15 003266   001403                   BEQ      2$                ;Br if yes
16 003270   016000   000000G   MOV      CC$LNK(R0),R0   ;Link forward to next block
17 003274   000772                   BR       1$                ;
18 003276   016560   000000G 000000G 2$:  MOV      CC$LNK(R5),CC$LNK(R0);;Remove block from list
19 003304   005337   000000G   DEC      @#NPCCB        ;One less pending cache control block
20         ;
21         ; Put control block on free list
22         ;
23 003310   013765   000000G 000000G   MOV      @#CCBHD,CC$LNK(R5);;Put control block on free list
24 003316   010537   000000G   MOV      R5,@#CCBHD    ;
25 003322                   ENABL                   ;Enable interrupts
26 003330   005067   174456   CLR      CURCCB        ;Say no longer using this control block
27         ;
28         ; Restart any jobs that are waiting for a control block
29         ;
30 003334   012700   000000G   MOV      #$QCCB,R0    ;Get wait state code
31 003340   004737   000000G   CALL    @#UREGO      ;Restart any waiting jobs
32         ;
33         ; Finished
34         ;
35 003344   000207                   RETURN

```

```
1          .SBTTL  CSHFIN -- Completion of a mapped I/O operation
2          ;-----
3          ; CSHFIN is called when a cached data transfer is completed.
4          ;
5          ; Inputs:
6          ; R5 = Address of cache control block.
7          ;
8 003346 010446 CSHFIN: MOV      R4,-(SP)
9          ;
10         ; Get the address of the original I/O queue element
11         ;
12 003350 016504 0000000 MOV      CC#QGE(R5),R4 ;Get address of original queue element
13         ;
14         ; Free the cache control block
15         ;
16 003354 004767 177670 CALL     CCBFRE          ;Free the cache control block
17         ;
18         ; Do I/O completion for the original queue element
19         ;
20 003360 004737 0000000 CALL     @#IOCMPL        ;Do I/O completion
21         ;
22         ; Finished
23         ;
24 003364 012604 MOV      (SP)+,R4
25 003366 000207 RETURN
```

```

1          .SBTTL  CSHGTQ -- Get a free I/O queue element
2          ;-----
3          ; Get a free I/O queue element and initialize it with information from
4          ; the cache control block and the original I/O queue element.
5          ;
6          ; Inputs:
7          ; R5 = Pointer to cache control block.
8          ;
9          ; Outputs:
10         ; R1 = Pointer to new queue element.
11         ;
12 003370 010346 CSHGTQ: MOV     R3,-(SP)
13 003372 010446         MOV     R4,-(SP)
14         ;
15         ; Get a free I/O queue element
16         ;
17 003374 016504 000000G MOV     CC#QOE(R5),R4 ;Get address of original queue element
18 003400 004737 000000G CALL    @#GETQ        ;Get a free I/O queue element
19 003404 010167 174406 MOV     R1,IQQOUT     ;Save address of secondary I/O queue element
20         ;
21         ; Copy information from original queue element to new queue element
22         ;
23 003410 010103         MOV     R1,R3          ;Get address of new queue element
24 003412 012700 000000G MOV     #IQQSIZ/2,R0  ;Get number of words to move
25 003416 012423 1#:    MOV     (R4)+,(R3)+ ;Copy contents of original queue element
26 003420 077002         SOB     R0,1#
27         ;
28         ; Set flags in new I/O queue element saying: (1) this is a secondary
29         ; I/O operation for data caching; and, (2) the completion routine is
30         ; a system routine that is to be called in kernel mode.
31         ;
32 003422 152761 000000G 000000G BISB   #QF#CID!QF#SCR,Q.FLAG(R1) ;Caching I/O & system compl rtn
33         ;
34         ; Store the address of the cache control block into the cell of the
35         ; I/O queue element normally used for the channel number.
36         ; This will cause the address of the cache control block to be passed
37         ; to the completion routine in R1.
38         ;
39 003430 010561 000000G MOV     R5,Q.CHAN(R1) ;Set address of control block as "channel #"
40         ;
41         ; Now set up information from the cache control block
42         ;
43 003434 016561 000000G 000000G MOV     CC#BLK(R5),Q.BLKN(R1) ;Set block number
44 003442 016561 000000G 000000G MOV     CC#WCT(R5),Q.WCNT(R1) ;Set word count
45 003450 016561 000000G 000000G MOV     CC#UBP(R5),Q.PAR(R1) ;Set 64-byte block number of buffer base
46 003456 012700 000000G MOV     #VPAR6,R0      ;Get virtual address bias of buffer
47 003462 066500 000000G ADD     CC#UBD(R5),R0   ;Add buffer offset within 64-byte block
48 003466 010061 000000G MOV     R0,Q.BUFF(R1)  ;Set buffer virtual address
49         ;
50         ; Finished
51         ;
52 003472 012604         MOV     (SP)+,R4
53 003474 012603         MOV     (SP)+,R3
54 003476 000207         RETURN
    
```

CSHCNT -- Keep track of cache usage statistics

```

1          .SBTTL  CSHCNT -- Keep track of cache usage statistics
2          ;-----
3          ; CSHCNT is called to add a value to one of the cache statistic counters.
4          ; The form of the call is:
5          ;
6          ;     MOV     value,R0       ;Get value to be added to counter
7          ;     JSR     R1,CSHCNT     ;Update counter
8          ;     .WORD  counter       ;Address of counter to be augmented
9          ;
10         ; The counters are all 32-bits long with the high-order value stored
11         ; in the first word.
12         ; If a counter overflows, all counters are zeroed.
13         ;
14 003500 010246 CSHCNT: MOV     R2,-(SP)
15 003502 012102      MOV     (R1)+,R2      ;Get pointer to statistitcs cells
16         ;
17         ; Add value to low-order word and propogate carry to high-order word
18         ;
19 003504 060062 000002      ADD     R0,2(R2)      ;Add value to low-order word
20 003510 005512      ADC     (R2)        ;Propogate carry to high order word
21 003512 102002      BVC     9$          ;Br if counter did not overflow
22         ;
23         ; A counter overflowed.
24         ; Clear all statistic counters to zero.
25         ;
26 003514 004767 000004      CALL    CSHZRO       ;Zero all statistic counters
27         ;
28         ; Finished
29         ;
30 003520 012602 9$:      MOV     (SP)+,R2
31 003522 000201      RTS     R1

```

```
1 .SBTTL CSHZRD -- Reset all statistic counters to zero
2 ;-----
3 ; Zero all data cache statistic counters.
4 ;
5 003524 CSHZRD:
6 ;
7 ; Zero all statistic counters
8 ;
9 003524 005037 0000000 CLR @#CASTRO ;Total number of read operations
10 003530 005037 0000020 CLR @#CASTRO+2
11 003534 005037 0000000 CLR @#CASTBR ;Total number of blocks read
12 003540 005037 0000020 CLR @#CASTBR+2
13 003544 005037 0000000 CLR @#CASCBR ;Blocks read from cache
14 003550 005037 0000020 CLR @#CASCBR+2
15 003554 005037 0000000 CLR @#CASTWO ;Total number of write operations
16 003560 005037 0000020 CLR @#CASTWO+2
17 003564 005037 0000000 CLR @#CASTBW ;Total number of blocks written
18 003570 005037 0000020 CLR @#CASTBW+2
19 003574 005037 0000000 CLR @#CASCUP ;Total number of blocks moved into cache
20 003600 005037 0000020 CLR @#CASCUP+2
21 ;
22 ; Finished
23 ;
24 003604 000207 RETURN
25
```

CSHSCH -- See if system job scheduler should be called

```
1          .SBTTL  CSHSCH -- See if system job scheduler should be called
2          ;-----
3          ; CSHSCH is called periodically during lengthy operations such as moving
4          ; data from cache buffers to/from user buffers to see if the job
5          ; scheduler should be called to give other jobs a chance to run.
6          ; This is done so that the data caching operations don't lock out
7          ; other jobs for an extended period of time.
8          ;
9 003606 105737 0000000 CSHSCH: TSTB   @#DOSCHD   ;Does the job scheduler need to be called?
10 003612 001402        BEQ     9#           ;Br if not
11          ;
12          ; Call job scheduler and allow higher priority jobs to run
13          ;
14 003614 004737 0000000 CALL    @#SCHED   ;Call job scheduler
15          ;
16          ; Finished
17          ;
18 003620 000207 9#:    RETURN
```

CSHINI -- Data caching system initialization

```

1          .SBTTL  CSHINI -- Data caching system initialization
2          ;-----
3          ; CSHINI is called during system initialization to initialize the
4          ; data caching system.
5          ;
6 003622 010146 CSHINI: MOV     R1,-(SP)
7 003624 010246      MOV     R2,-(SP)
8 003626 013746 0000000      MOV     @#KPAR5,-(SP) ; Save kernel PAR5 mapping
9          ;
10         ; See if there are any active cache buffers
11         ;
12 003632 005737 0000000      TST     @#VCSHNB      ; Any active cache buffers?
13 003636 001437      BEQ     9$          ; Br if not
14         ;
15         ; Put all data cache entries on the free list
16         ;
17 003640 012701 0000000      MOV     #VPAR5,R1      ; Get virtual address of 1st entry
18 003644 010137 0000000      MOV     R1,@#CSHFHD   ; Set pointer to 1st free entry
19 003650      MAPTD   CA#UFL      ; Map to forward link chain
20 003656 013700 0000000      MOV     @#VCSHNB,R0   ; Get total number of data cache entries - 1
21 003662 005300      DEC     R0
22 003664 003406      BLE     3$          ; Br if only 1 entry
23 003666 010102 1$:      MOV     R1,R2      ; Get address of current entry
24 003670 062702 0000002      ADD     #2,R2      ; Get address of next entry
25 003674 010211      MOV     R2,(R1)     ; Make current entry point to next entry
26 003676 010201      MOV     R2,R1      ; Get pointer to next entry
27 003700 077006      SOB     R0,1$      ; Loop if more to link together
28 003702 005011 3$:      CLR     (R1)        ; Zero the last forward link
29         ;
30         ; Say LRU chain is empty
31         ;
32 003704 005037 0000000      CLR     @#CSHMRU     ; No most-recently-used entry
33 003710 005037 0000000      CLR     @#CSHLRU     ; No least-recently-used entry
34         ;
35         ; Zero all of the hash list heads
36         ;
37 003714 012701 0000000      MOV     #VPAR5,R1      ; Get virtual address of 1st entry
38 003720 013700 0000000      MOV     @#VCSHNB,R0   ; Get total number of entries
39 003724      MAPTD   CA#HSH      ; Map to hash list heads
40 003732 005021 2$:      CLR     (R1)+       ; Zero each one
41 003734 077002      SOB     R0,2$
42         ;
43         ; Zero all cache statistic counters
44         ;
45 003736 004767 177562 9$:      CALL    CSHZRO      ; Zero all statistic counters
46         ;
47         ; Finished
48         ;
49 003742 012637 0000000      MOV     (SP)+,@#KPAR5 ; Restore kernel PAR5 mapping
50 003746 012602      MOV     (SP)+,R2
51 003750 012601      MOV     (SP)+,R1
52 003752 000207      RETURN
53         .END

```

Errors detected: 0

*** Assembler statistics

Work file reads: 0
Work file writes: 0
Size of work file: 8618 Words (34 Pages)
Size of core pool: 17920 Words (70 Pages)
Operating system: RT-11

Elapsed time: 00:00:25.45
DK: TSCASH, LP: TSCASH=DK: TSCASH. MAC/C/N: SYM

#NOABT	1-36	20-31	20-34										
...V2	4-25	4-25	4-25	4-25	4-25	4-25	4-25	4-25	4-25	4-25	4-25	4-25	4-25
	4-25	4-25	4-25	4-25	4-25#	4-25#	4-25#	6-16	6-16	6-16	6-16	6-16	6-16
	6-16	6-16	6-16	6-16	6-16	6-16	6-16	6-16	6-16	6-16	6-16	6-16	6-16#
	6-16#	6-16#	9-17	9-17	9-17	9-17	9-17	9-17	9-17	9-17	9-17	9-17	9-17
	9-17	9-17	9-17	9-17	9-17	9-17	9-17#	9-17#	9-17#	10-17	10-17	10-17	10-17
	10-17	10-17	10-17	10-17	10-17	10-17	10-17	10-17	10-17	10-17	10-17	10-17	10-17
	10-17	10-17#	10-17#	10-17#									
C. CSW	1-26	5-16											
CA#BLK	1-28	11-47	13-35	15-35	15-58	15-60	16-34						
CA#DVU	1-26	11-49	12-27	13-39									
CA#HBL	1-28	15-51	15-72	15-80	16-46	16-65							
CA#HFL	1-27	13-42	15-49	15-64	15-78	15-88	16-44	16-52					
CA#HSH	1-28	13-32	15-41	15-85	16-58	27-39							
CA#UBL	1-26	12-25	14-21	14-32	14-38	15-16	15-30	16-17	16-28				
CA#UFL	1-26	14-19	14-26	14-41	15-19	15-28	16-15	16-22	17-19	18-10	27-19		
CA#WCT	1-26	3-36	8-29	11-24	11-51								
CASCBR	1-35	3-53	25-13*	25-14*									
CASCUP	1-35	8-36	25-19*	25-20*									
CASTBR	1-35	3-22	25-11*	25-12*									
CASTBW	1-35	6-34	25-17*	25-18*									
CASTRO	1-36	3-16	25-9*	25-10*									
CASTWO	1-36	6-28	25-15*	25-16*									
CC#BLK	1-25	3-62*	8-45*	11-12	11-48	11-60*	11-70*	13-22	13-30	20-54*	23-43		
CC#CBP	1-24	9-26	10-26	13-54*									
CC#DVU	1-25	11-50	13-31	20-61*	20-62*								
CC#LNK	1-31	20-40	20-44*	21-12	21-14	21-16	21-18	21-18*	21-23*				
CC#OQE	1-29	5-14	20-53*	22-12	23-17								
CC#UBO	1-25	9-28	10-28	20-71*	23-47								
CC#UBP	1-25	3-63*	8-46*	9-25	10-25	20-68*	23-45						
CC#WCT	1-24	3-17	3-32	3-64*	6-29	8-13	8-47*	9-33	10-33	11-13	20-58*	23-44	
CC#WFL	1-25	6-21*	6-49*	7-13*									
CCBFRE	21-8#	22-16											
CCBGET	2-63	20-18#											
CCBHD	1-30	20-24	20-40*	21-23	21-24*								
CD#\$SZ	1-29	19-34											
CD#DVU	1-25	19-32											
CORUSR	1-35	20-30											
CS#EOF	1-27	5-16											
CS#ERR	1-30	5-16											
CSHADD	11-56	15-7#											
CSHBFP	1-30	13-53											
CSHBLD	4-17	6-41	11-10#										
CSHCLN	1-77	2-52	12-9#										
CSHCNT	3-15	3-21	3-52	6-27	6-33	8-35	24-14#						
CSHDEV	1-30	19-31											
CSHDVN	1-27	19-35											
CSHFHD	1-33	17-14	17-20*	18-8	18-9*	27-18*							
CSHFIN	1-78	3-69	5-25	6-55	7-19	22-8#							
CSHFRE	12-38	18-8#											
CSHGET	11-36	17-10#											
CSHGTO	4-21	6-14	23-12#										
CSHINI	1-75	27-6#											
CSHIO	1-76	2-14#											
CSHLOC	3-26	8-20	11-18	13-17#									
CSHLRU	1-30	11-42	12-20	14-30*	15-27*	16-26*	27-33*						

VCSHNB	1-33	2-35	12-15	13-24	15-38	16-37	27-12	27-20	27-38
VPAR5	1-29	9-27	10-27	13-26	13-51	15-40	16-39	27-17	27-37
VPAR6	1-29	9-29	10-29	20-64	23-46				
WRTCPL	6-16	6-16	7-8#						
WTMOVE	8-25	10-9#							

