

6.31

Table of contents

2-	1	WRITTT -- .WRITE emt with terminal as output device
3-	1	TTREAD -- .READ emt with input from terminal
5-	1	.PRINT
6-	1	.TTYOUT
7-	1	.TTYIN
8-	1	High-efficiency TTY EMTs
9-	1	ASKLIN -- Accept line from terminal
10-	1	OUTSTR -- Print a system message on the user's console
10-	26	CSIMSG -- Print a CSI asciz error message
11-	1	UOTSTR -- print an asciz user's message on the user's console
12-	1	
12-	2	** Program Level Output Character Processing **
12-	3	PUTCHR -- Send character to terminal
13-	1	PUTCH1 -- Queue character for terminal
14-	1	PUTCH2 -- Queue character for a terminal
17-	1	QUECHR -- Queue character for transmission
18-	1	BUFCHR -- Insert char or suspend if full
19-	1	HIPUT -- High efficiency PUTCHR
20-	1	ESCHK -- Check for echo suppression restart
21-	1	LIFUN -- Process lead-in function sequences
33-	1	
33-	2	** Program Level Input Character Processing **
33-	3	GETCHR -- Get next input char
39-	1	GTCFCH -- Try to get char from command file
40-	1	CFCHAR -- Do command file I/O
41-	1	CFTEST -- Determine if TT input is from file
42-	1	CFSTOP -- Suspend command file input
42-	22	CFSTRT -- Restart command file input
43-	1	CFPOP -- Pop up to next command file
44-	1	LOGCHR -- Write character to log file
46-	1	ILWAIT -- Wait for activation char from terminal
47-	1	DFRREL -- Release deferred echo mode
48-	1	
48-	2	** Fork Level Input Character Processing **
48-	3	TTINCP -- Process received input characters
51-	1	REGCHR -- Process normal characters
52-	1	DOCTRL -- Process control characters
53-	1	ICPCR -- Carriage-return processing
54-	1	ICPLF -- Line-feed processing
55-	1	ICPCTC -- Control-C processing
56-	1	ICPCTD -- Control-D processing
57-	1	ICPCTG -- Control-G processing
58-	1	ICPCTO -- Control-O processing
59-	1	ICPCTR -- Control-R processing
60-	1	ICPCTU -- Control-U processing
61-	1	ICPCTX -- Control-X processing
62-	1	ICPCTZ -- Control-Z processing
63-	1	ICPESC -- Escape processing
64-	1	ICPRUB -- Rubout processing
65-	1	CKVTAC -- Check for VTxx escape-letter activation
66-	1	CHKODT -- Check for ODT activation characters
67-	1	INFIN -- TT input wait completed
68-	1	KILCHR -- Delete a character from input buffer
69-	1	INCHR -- Store and echo a character
70-	1	STRCHR -- Store a character into TT buffer
71-	1	STRACT -- Store activation character
72-	1	STRSNG -- Store char with single-character input

Table of contents

73-	1	FETCHR	-- Fetch next char from TT input ring buffer
74-	1	INSCHR	-- Insert character into TT input ring buffer
75-	1	DELCHR	-- Delete character from TT input ring buffer
76-	1	ECHO	-- Echo character to terminal
76-	22	ECOCTL	-- Echo a control character
76-	45	RBEND	-- Terminate rubout sequence
77-	1	SCACHK	-- Check for single-character activation
78-	1	SLCHK	-- Check for single line editor mode
79-	1	SCACHR	-- Handle single-character activation characters
80-	1	CVTLC	-- Convert lower-case chars to upper-case
81-	1	SIGWAT	-- Signal virtual line wait condition
81-	42	SIGBRK	-- Signal program that Break character was received

```

1          .TITLE  TSTTY -- TSX Terminal I/O routines
2          .ENABL  LC
3          .ENABL  AMA
4          .DSABL  GBL
5 000000   .CSECT  TSTTY
6 000000   100071 TSTTY: .RAD50 /TTY/           ;Overlay region id
7          ;
8          ; TSTTY is the TSX module that contains routines related
9          ; to doing I/O to the user's terminal.
10         ;
11         ; Copyright (c) 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987.
12         ; S&H Computer Systems, Inc. Nashville, Tn.
13         ;
14         ;
15         ; Macro calls
16         ;
17         .MCALL .READW, .PURGE, .REOPEN, .WRITW
18         ;
19         ; Global definitions
20         ;
21         .GLOBL PUTCHR, BUFCHR
22         .GLOBL DELCHR, CVTLC
23         .GLOBL TTINCP, TSTTY, LOGCHR, LOGCR
24         .GLOBL STRACT, PCSPND
25         .GLOBL WRITTT, TTREAD, CSIMSG, ASKLIN
26         .GLOBL PRINT, TTYOUT, TTYIN, SIGWAT, QUECHR, BUFCHR
27         .GLOBL XHISSET, XHIOUT, XHIIN, XTERCK, XRDTIM
28         .GLOBL SETRBF, CMDB, CMDC, GTSPAC, CMDE, CMDF, CMDG, CMDH, CMDI
29         .GLOBL CMDJ, CMDK, CMDL, CMDM, CMDN, CMDO, RSSPAC, SFWAC, CMDR
30         .GLOBL CMDS, CMDT, CMDU, SFWL, CMDW, CMDX, CMDY, CMDZ, MAXCC
31         ;
32         ; Global references
33         ;
34         .GLOBL $AUTO, $RBRK, $RFRSH, LSW4, R$CFST, CFACFL, $SCCA, AF$CCA, AFCF
35         .GLOBL SILFET, $SUSPN, WINPRT, $VBELL, LTTCR, TTCPL, AF$NPW, SUCF2
36         .GLOBL $RTCS, AUTSPD, DOSWIT, VVPWCH, PRIVFO, PRIVSO, PRIVCO, PVNPW
37         .GLOBL MXSPAC, LOTSIZ, LSTACT, SETERR, LSW11, $PWKEY, $NOWIN
38         .GLOBL TRNSTR, GTSLCH, $SLON, $SLTTY, $V52EM
39         .GLOBL $NOWTT, NOWAIT, FRKPRI, LSW7
40         .GLOBL CFARG, DISSLE, $DBKMN, $CTRLD, CXTRMN
41         .GLOBL LSW, LSW2, LSW3, LSW4, LSW5, LJSW, $NTGCC
42         .GLOBL $DILUP, $DQOFF, $DISCN, IN$ACT
43         .GLOBL $DETCH, $CTRLC, $1ESC, $NDICP
44         .GLOBL $SCOPE, $ECHO, CFHOLD, ILSW2, $CHACT
45         .GLOBL $TAB, $FORM, $NOINT, LABTIM
46         .GLOBL $LC, $NOVLN, $DEFER, $NOOUT, NEDCDI
47         .GLOBL $NOIN, $TRNSP, LINSPC, QCOMPL
48         .GLOBL $CTRLD, $RBOU, $1STCH, $CTRLW
49         .GLOBL $CTRLS, $DODFR, $GCECO, $GCESC
50         .GLOBL $QUIET, $INKMN, $UCTLC, $SETCC, VVLSCH
51         .GLOBL $ODTMD, $CFOPN, $CFALL, $GTLIN
52         .GLOBL LSW10, $BBIT, LWINDO, WINCHR
53         .GLOBL LESCHR, LESRTN, LSNDCH
54         .GLOBL $TTERR, $CFSOT, $HITTY, $FLAGC
55         .GLOBL $1CTL, $VTESC, HAZEL
56         .GLOBL S$$RUN, $DEBUG, $DBGBK
57         .GLOBL $DEAD

```

```

58      . GLOBL BKSPAC, CR, $UKMON
59      . GLOBL $LOFCF, $SUCF, LSW9
60      . GLOBL LNMAP, S$OTFN, ENQTL, STOP
61      . GLOBL CORUSR, CFPNT, LACTIV, LINPNT
62      . GLOBL LINEND, LINBUF, MXCPRM, PRMPNT, LSCCA
63      . GLOBL CURPRM, CFEND, CFCHAN, CFBUF, $SLINI
64      . GLOBL LNSPAC, LSPACT, CTRLZ, CTRLC, CTRLX
65      . GLOBL LINNXT
66      . GLOBL TAB, BKSPAC, RUBOUT, FORCEEX, LINSIZ, LINCNT
67      . GLOBL BELL, LNPRIM, LF
68      . GLOBL TRNSFL, ESC, LTSCMD, VTSLCH, LCBIT
69      . GLOBL CFSPND, PR7, ACFLAG, LAFSIZ, INTPRI
70      . GLOBL INTPRI, PSW, DOSCHD, ESCFLG
71      . GLOBL INITFL, JSWLOC, STPFLG
72      . GLOBL LCOL, LOTSPC, LOTNXT, LOTEND, LOTBUF
73      . GLOBL S$OTWT, CHKABT
74      . GLOBL QNSPNX, QHDSPN
75      . GLOBL LOTPNT, FF
76      . GLOBL LSTATE, SPCTTY, $CCLRN, CFBLK, LINCUR
77      . GLOBL S$INWT
78      . GLOBL MAXSEC, LRBFIL, SPACE
79      . GLOBL LCBIT, INTPRI
80      . GLOBL LRDTIM, LRTCHR, LBRKCH
81      . GLOBL S$TTFN, VQUAN1, $NOLF, S$TTSC
82      . GLOBL LBRKCG, $DBQMD
83      . GLOBL LFWLIM
84      . GLOBL VT52, VT100, LTRMTP, VT2007, VT2008
85      . GLOBL UHIMEM, TTCSCH
86      . GLOBL $TAPE, $XSTOP, LSW6, KPAR6, LTTPAR, $CFABT
87      . GLOBL BRKPT, $CFDCC, $CFCCCL, $CFKIL
88      . GLOBL GETUCH, PUTUCH, VALADB, EMTBLK, FAKCMP
89      . GLOBL CS$EOF, CFLAG, CTRLZ, $FORMO, FF, OVRHC
90      . GLOBL CHNADR, EMTPS, URO, $TLTTY, LJSW, SETC, EMTXIT
91      . GLOBL CFIND, R$INST, CFNEST, CFSP, LSTPRM
92      . GLOBL PBFEND, PRMBUF, CFLFL4, INITLN
93      . GLOBL VINTIO, LHIPCT
94      . GLOBL LOGBUF, LOGEND, LOGCHN, LOGBLK, ABORT, EMTADR
95      . GLOBL LOGPTR, LOGFLG, LF$WRT, LITIME, LF$IN, LF$OUT
96      ;
97      ; Macro definitions:
98      ;
99      . MACRO DISABL          ; DISABLE INTERRUPTS
100     BIS      #PR7, @#PSW
101     . ENDM  DISABL
102
103     . MACRO ENABL           ; ENABLE INTERRUPTS
104     BIC      INTPRI, @#PSW
105     . ENDM  ENABL
106
107     . MACRO OCALL  ENTADD
108     . IF    B, ENTADD
109           . ERROR ; OCALL without entry address
110     . ENDC
111     CALL    OVRHC
112     . WORD  ENTADD
113     . ENDM  OCALL
114

```

```

115 ; The TTMAP and TTMAPX macros are used to map kernel-mode par6 to the
116 ; terminal character buffer area. The previous contents of par6 map
117 ; register are pushed on the stack and may be restored by using the
118 ; UNMAP or UNMAPX macros.
119 ; R1 must contain the line index number of the line whose buffers
120 ; are being accessed.
121 ; The difference between the TTMAP-UNMAP macros and the TTMAPX-UNMAPX
122 ; macros is that the X-versions are more efficient but may only be
123 ; used from within interrupt service routines where we are guaranteed
124 ; to be running on the system stack.
125 ; The TTMAP and UNMAP versions of the macros must only be
126 ; used in sections of code where the interrupts are disabled.
127 ;
128 ; .MACRO TTMAPX
129 ; MOV LTTPAR(R1),@#KPAR6
130 ; .ENDM TTMAPX
131 ;
132 ;
133 ; .MACRO UNMAPX
134 ; .ENDM UNMAPX
135 ;
136 ;
137 ; .MACRO TTMAP
138 ; MOV @#KPAR6,MAPHLD
139 ; MOV LTTPAR(R1),@#KPAR6
140 ; .ENDM TTMAP
141 ;
142 ; .MACRO UNMAP
143 ; MOV MAPHLD,@#KPAR6
144 ; .ENDM UNMAP
145 ;
146 ; Data areas
147 ;
148 000002 000000 MAPHLD: .WORD 0 ;TEMP CELL USED BY TTMAP MACRO
149 ;
150 ; CSI table of error messages.
151 ;
152 000004 000022' CSIERR: .WORD CSEMIL
153 000006 000051' .WORD CSEMID
154 000010 000077' .WORD CSEPRO
155 000012 000163' .WORD CSEMFO
156 000014 000226' .WORD CSEMNF
157 000016 000256' .WORD CSEMIS
158 000020 000304' .WORD CSEMIV
159 ;
160 ; CSI text messages.
161 ;
162 ; .NLIST BEX
163 000022 077 103 123 CSEMIL: .ASCIZ /?CSI-F-Illegal command/
164 000051 077 103 123 CSEMID: .ASCIZ /?CSI-F-Illegal device/
165 000077 077 103 123 CSEPRO: .ASCIZ /?CSI-F-Protected file with same name already exists/
166 000163 077 103 123 CSEMFO: .ASCIZ /?CSI-F-Insufficient space for file/
167 000226 077 103 123 CSEMNF: .ASCIZ /?CSI-F-Cannot find file/
168 000256 077 103 123 CSEMIS: .ASCIZ /?CSI-F-Invalid switch/
169 000304 077 103 123 CSEMIV: .ASCIZ /?CSI-F-Invalid switch value/
170 ; .LIST BEX
171 ;

```

```
172 ; Bit mask table used to test, set, and clear the activation-character
173 ; flags for characters
174 ;
175 000340 001 002 004 BITMSK: .BYTE 1,2,4,10,20,40,100,200
    000343 010 020 040
    000346 100 200
176 .EVEN
```

```

1          .SBTTL  WRITTT  -- .WRITE emt with terminal as output device
2          ;-----
3          ; WRITTT is executed when it is determined that the .WRITE emt is
4          ; directed to the terminal device.
5          ;
6 000350 013703 000004G WRITTT: MOV      EMTBLK+4,R3      ;GET BUFFER ADDRESS
7 000354 010300          MOV      R3,R0          ;VALIDATE BUFFER ADDRESS
8 000356 004737 000000G          CALL     VALADB
9 000362 013704 000006G          MOV      EMTBLK+6,R4      ;GET # WORDS TO WRITE
10 000366 001454          BEQ      2$          ;BR IF NO WORDS TO BE WRITTEN
11 000370 006304          ASL      R4          ;CONVERT TO # BYTES
12 000372 060400          ADD      R4,R0          ;GET ADDRESS OF END OF BUFFER
13 000374 005300          DEC      R0
14 000376 004737 000000G          CALL     VALADB          ;VALIDATE IT
15 000402 042777 000000G 000000G BIC      #CS$EOF,@CHNADR ;RESET CHANNEL END OF FILE FLAG
16 000410 005737 000002G          TST      EMTBLK+2      ;WRITE TO BLOCK 0?
17 000414 001010          BNE      16$          ;BR IF NOT
18 000416 032761 000000G 000000G BIT      $$FORM0,LSW4(R1); DOES HE WANT FF ON WRITE OF BLK 0?
19 000424 001404          BEQ      16$          ;BR IF NOT
20 000426 112700 000000G          MOVB     #FF,R0          ;OUTPUT FF TO GET TO TOP OF FORM
21 000432 004737 002662'          CALL     PUTCHR
22          ;
23          ; Determine if buffer is on even byte boundary
24          ;
25 000436 032703 000001          16$:    BIT      #1,R3          ;IS BUFFER STARTING ON EVEN BYTE BOUNDARY?
26 000442 001430          BEQ      9$          ;BR IF YES
27          ;
28          ; Use slow routine if buffer is on odd byte boundary
29          ;
30 000444 012702 000000G          17$:    MOV      #TTCSCH,R2      ;RESET CHARACTER COUNT
31          ; See if we need to interrupt tt output processing to do a job
32          ; scheduler cycle.
33 000450 105737 000000G          TSTB     DOSCHD          ;IS JOB SCHEDULER CYCLE NEEDED?
34 000454 001410          BEQ      4$          ;BR IF NOT
35 000456 004737 000000G          CALL     CHKABT          ;SEE IF WE HAVE BEEN ABORTED
36 000462 016100 000000G          MOV      LSTATE(R1),R0      ;GET JOB'S CURRENT EXECUTION STATE
37 000466 004737 000000G          CALL     QNSPNX          ;REQUEUE JOB AND CALL JOB SCHEDULER
38 000472 004737 000000G          CALL     CHKABT          ;SEE IF WE WERE ABORTED WHILE ASLEEP
39 000476 005302          4$:    DEC      R2          ;TIME TO CHECK SCHEDULER?
40 000500 003761          BLE      17$          ;BR IF YES
41 000502 004737 000000G          CALL     GETUCH          ;GET CHAR FROM USER'S BUFFER
42 000506 005700          TST      R0          ;IS CHAR A NULL?
43 000510 001402          BEQ      1$          ;SKIP NULLS
44 000512 004737 002662'          CALL     PUTCHR          ;PLACE IN USER'S BUFFER
45 000516 077411          1$:    SOB      R4,4$          ;BR IF MORE CHARS TO DO
46 000520 000137 001004'          2$:    JMP      TTFIN
47          ;
48          ; Use faster routine if buffer is on even byte boundary
49          ;
50 000524 013704 000006G          9$:    MOV      EMTBLK+6,R4      ;GET NUMBER OF WORDS TO WRITE
51 000530 012702 000000G          7$:    MOV      #TTCSCH,R2      ;RESET CHAR COUNT FOR CONTINUATION
52          ; See if we need to interrupt tt output processing to do a job
53          ; scheduler cycle.
54 000534 105737 000000G          TSTB     DOSCHD          ;IS JOB SCHEDULER CYCLE NEEDED?
55 000540 001410          BEQ      8$          ;BR IF NOT
56 000542 004737 000000G          CALL     CHKABT          ;SEE IF WE HAVE BEEN ABORTED
57 000546 016100 000000G          MOV      LSTATE(R1),R0      ;GET JOB'S CURRENT EXECUTION STATE

```

58	000552	004737	0000000	CALL	QNSPNX	; REQUEUE JOB AND CALL JOB SCHEDULER
59	000554	004737	0000000	CALL	CHKART	; SEE IF WE WERE ABORTED WHILE ASLEEP
60	000562	005302		8\$: DEC	R2	; TIME TO CHECK FOR SCHEDULER CYCLE?
61	000564	003761		BLE	7\$	; BR IF YES
62	000566	106523		MFPD	(R3)+	; GET DATA WORD FROM USER'S BUFFER
63	000570	111600		MOVB	(SP), R0	; GET LOW-ORDER BYTE OF WORD
64	000572	001402		BEQ	5\$	; IGNORE IT IF IT IS NULL
65	000574	004737	002662'	CALL	PUTCHR	; SEND CHAR TO TERMINAL
66	000600	012600		5\$: MOV	(SP)+, R0	; GET DATA WORD
67	000602	105000		CLRB	R0	; CLEAR LOW-ORDER BYTE
68	000604	000300		SWAB	R0	; GET HIGH-ORDER BYTE TO LOW-ORDER
69	000606	001402		BEQ	6\$	; BR IF LOW-ORDER BYTE IS NULL
70	000610	004737	002662'	CALL	PUTCHR	; SEND CHAR TO TERMINAL
71	000614	077416		6\$: SOB	R4, 8\$	; LOOP IF MORE WORDS TO WRITE
72	000616	000137	001004'	JMP	TTFIN	; FINISHED



TTREAD -- .READ emt with input from terminal

```

1          .SBTTL  TTREAD -- .READ emt with input from terminal
2          ;-----
3          ; TTREAD is executed when a .READ was issued directing input
4          ; from the terminal.
5          ;
6 000622 013703 000004G TTREAD: MOV EMTBLK+4,R3 ;GET BUFFER ADDRESS
7 000626 010300          MOV R3,R0 ;VALIDATE ADDRESS OF BUFFER
8 000630 004737          CALL VALADB
9 000634 013704 000006G MOV EMTBLK+6,R4 ;GET # WORDS TO READ
10 000640 006304        ASL R4 ;CONVERT TO # BYTES
11 000642 060400        ADD R4,R0 ;GET ADDRESS OF END OF BUFFER
12 000644 005300        DEC R0
13 000646 004737 000000G CALL VALADB ;VALIDATE IT
14 000652 032777 000000G 000000G BIT #CS$EOF,@CHNADR ;DID LAST READ GET END OF FILE?
15 000660 001407        BEQ 2$ ;BRANCH IF NOT
16 000662 042777 000000G 000000G BIC #CS$EOF,@CHNADR ;RESET END OF FILE FLAG
17 000670 052737 000000G 000000G BIS #CFLAG,EMTPS ;SET C-BIT IN USER'S PS
18 000676 000425        BR TTZERO
19 000700 005737 000002G 2$: TST EMTBLK+2 ;READ BLOCK 0?
20 000704 001004        BNE 1$ ;BRANCH IF NOT BLOCK 0
21 000706 112700 000136        MOVB #'^,R0 ;PRINT PROMPT CHARACTER
22 000712 004737 002662'        CALL PUTCHR
23 000716 004737 005756' 1$: CALL GETCHR ;GET NEXT INPUT CHAR
24 000722 120027 000000G        CMPB R0,#CTRLZ ;CTRL-Z ==> END OF FILE
25 000726 001406        BEQ TTEOF ;BRANCH IF END OF FILE
26 000730 105700 4$: TSTB R0 ;IGNORE NULLS
27 000732 001771        BEQ 1$
28 000734 004737 000000G        CALL PUTUCH ;MOVE CHAR TO USER'S BUFFER
29 000740 077412        SOB R4,1$ ;BR IF MORE ROOM LEFT
30          ; NORMAL END OF READ
31 000742 000420        BR TTFIN
32          ; END OF FILE ON INPUT FROM TT:
33 000744 052777 000000G 000000G TTEOF: BIS #CS$EOF,@CHNADR ;REMEMBER END OF FILE HIT
34          ;
35          ; Hit end of file -- Fill remainder of buffer with nulls
36          ;
37 000752 005704 TTZERO: TST R4 ;ROOM LEFT IN BUFFER?
38 000754 001413        BEQ TTFIN ;BR IF NOT
39 000756 010300 1$: MOV R3,R0 ;GET NEXT CHAR POSITION
40 000760 163700 000004G        SUB EMTBLK+4,R0 ;GET # CHARACTERS TRANSFERRED SO FAR
41 000764 001403        BEQ 2$ ;BR IF HAVEN'T GOTTEN ANY CHARS AT ALL
42 000766 032700 000777        BIT #777,R0 ;FILLED TO BLOCK BOUNDARY?
43 000772 001404        BEQ TTFIN ;BR IF YES
44 000774 005000 2$: CLR R0 ;STORE NULL TO FILL OUR BLOCK
45 000776 004737 000000G        CALL PUTUCH ;MOVE NULL TO USER'S BUFFER
46 001002 077413        SOB R4,1$ ;LOOP TO FILL REST OF USER'S BUFFER
47          ; Fall into TTFIN

```

```

1      ;
2      ;   FINISHED WITH TT READ/WRITE
3      ;
4 001004 010304      TTFIN:  MOV     R3,R4      ;GET CURRENT BUFFER POINTER
5 001006 163704 0000040      SUB     EMTBLK+4,R4    ;GET # BYTES TRANSFERED
6 001012 006204      ASR     R4        ;CONVERT TO # WORDS
7 001014 103001      BCC    2$        ;BR IF NO ODD BYTE
8 001016 005204      INC     R4
9 001020 010437 000000G      2$:   MOV     R4,URO    ;RETURN # WORDS IN USER'S RO
10     ;
11     ;   See if we need to call user's completion routine.
12     ;
13 001024 023727 000010G 000001      CMP     EMTBLK+10,#1  ;COMPLETION ROUTINE SPECIFIED?
14 001032 101415      BLOS   1$        ;BR IF NOT
15     ;   Enter a request to call user's completion routine.
16 001034 004737 000000G      CALL   FAKCMP       ;QUEUE A COMPLETION REQUEST
17 001040 105737 000000G      TSTB  DOSCHD       ;IS JOB SCHEDULER CYCLE NEEDED?
18 001044 001410      BEQ   1$        ;BR IF NOT
19 001046 004737 000000G      CALL   CHKABT      ;SEE IF WE HAVE BEEN ABORTED
20 001052 016100 000000G      MOV   LSTATE(R1),R0 ;GET JOB'S CURRENT EXECUTION STATE
21 001056 004737 000000G      CALL   QNSPNX      ;REQUEUE JOB AND CALL JOB SCHEDULER
22 001062 004737 000000G      CALL   CHKABT      ;SEE IF WE WERE ABORTED WHILE ASLEEP
23     ;
24     ;   Finished
25     ;
26 001066 000137 000000G      1$:   JMP     EMTXIT

```

.PRINT

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11

.SBTTL .PRINT

```
-----  
; PROCESS THE .PRINT EMT  
;  
PRINT:  MOV    URO,R2      ;GET ADDRESS OF STRING TO PRINT  
        CMP    R2,UHMEM    ;IS BUFFER ADDRESS IN NORMAL JOB REGION?  
        BLOS   1$          ;BR IF YES  
        MOV    R2,R0       ;VALIDATE ADDRESS OF BUFFER  
        CALL   VALADB  
1$:     CALL   UOTSTR      ;PRINT THE STRING FROM USER'S BUFFER  
        JMP    EMTXIT
```

.TTYOUT

```

1          .SBTTL .TTYOUT
2          ;-----
3          ; Process the .TTYOUT EMT
4          ;
5 001122   TTYOUT:
6          ;
7          ; See if output ring buffer is full
8          ;
9 001122   026127 000000G 000010       CMP     LOTSPC(R1),#8.  ;Is there plenty of free space in output buf?
10 001130   101022                BHI     3$                ;Br if yes
11          ;
12          ; Output buffer is full.
13          ; See if user wants to suspend job or have carry flag set on return
14          ;
15 001132   032761 000000G 000000G       BIT     #NOWAIT,LJSW(R1);Did he request nowait TT I/O?
16 001140   001416                BEQ     3$                ;Br if not
17 001142   032761 000000G 000000G       BIT     #NOWTT,LSW5(R1);Did he enable no-wait TT I/O?
18 001150   001412                BEQ     3$                ;Br if not
19          ;
20          ; Output buffer is full and user has enable no-wait mode.
21          ; See if instruction following EMT 341 is a BCS .-2
22          ;
23 001152   013700 000000G                MOV     EMTADR,R0        ;Get address of EMT 341 instruction
24 001156   006560 000002                MFPI   2(R0)           ;Fetch following instruction
25 001162   022627 103776                CMP     (SP)+,#103776   ;BCS .-2 instruction?
26 001166   001403                BEQ     3$                ;Br if yes -- Don't return if he will spin
27          ;
28          ; Return with carry flag set, and error code 0 to signal that output
29          ; buffer is full.
30          ;
31 001170   005000                CLR     R0              ;Return error code 0
32 001172   000137 000000G                JMP     SETERR
33          ;
34          ; Transmit the character (wait if output buffer is full)
35          ;
36 001176   013700 000000G 3$:          MOV     URO,R0          ;GET THE CHAR TO SEND
37 001202   032761 000000G 000000G       BIT     ##HITTY,LSW4(R1);ARE WE IN HIGH EFFICIENCY MODE?
38 001210   001003                BNE     1$              ;BR IF YES
39 001212   004737 002662'                CALL   PUTCHR          ;SEND THE CHAR
40 001216   000402                BR     2$
41 001220   004737 003672' 1$:          CALL   HIPUT           ;SEND CHAR IN HIGH EFFICIENCY MODE
42 001224   000137 000000G 2$:          JMP     EMTXIT

```

.TTYIN

```

1
2
3
4
5 001230 032761 000000G 000000G TTYIN: BIT #NOWAIT,LJSW(R1);DID HE REQUEST NOWAIT .TTYIN?
6 001236 001445 BEQ 1$ ;BR IF NOT
7
8
9
10
11
12
13 001240 005761 000000G TST LACTIV(R1) ;ANY ACTIVATION CHARS PENDING?
14 001244 001042 BNE 1$ ;BR IF YES
15 001246 004737 007760' CALL CFTEST ;IS INPUT COMING FROM A COMMAND FILE?
16 001252 103037 BCC 1$ ;BRANCH IF YES
17 001254 032761 000000G 000000G BIT ##LOFCF,LSW9(R1);Are we processing a logoff command file?
18 001262 001405 BEQ 4$ ;Br if not
19 001264 052761 000000G 000000G BIS ##DOOFF,LSW(R1) ;Force job logoff
20 001272 004737 000000G CALL STOP ;Stop job execution
21 001276 042761 000000G 000000G 4$: BIC ##NOIN,LSW3(R1) ;ALLOW TERMINAL INPUT TO OCCUR
22 001304 032761 000000G 000000G BIT ##DBGMD,LSW6(R1); IS DEBUGGER DOING TERMINAL INPUT?
23 001312 001004 BNE 3$ ;BR IF YES -- WAIT FOR ACTIVATION CHAR
24 001314 032761 000000G 000000G BIT ##NOWTT,LSW5(R1);DID HE ENABLE NO-WAIT TT INPUT?
25 001322 001007 BNE 2$ ;IF YES THEN SIGNAL NO CHARS AVAILABLE
26 001324 032761 000000G 000000G 3$: BIT ##FLAGC,LSW4(R1);HAVE WE ALREADY TOLD HIM ONCE?
27 001332 001007 BNE 1$ ;BR IF YES (WAIT FOR INPUT NOW)
28 001334 052761 000000G 000000G BIS ##FLAGC,LSW4(R1);REMEMBER THAT WE HAVE TOLD HIM
29 001342 004737 011346' 2$: CALL DFRREL ;RELEASE DEFERRED ECHO MODE
30 001346 000137 000000G JMP SETC ;RETURN WITH C-FLAG SET
31
32
33
34 001352 042761 000000G 000000G 1$: BIC ##FLAGC,LSW4(R1);HAVEN'T TOLD HIM NO CHARS AVAILABLE
35 001360 004737 005756' CALL GETCHR ;GO GET A CHARACTER
36 001364 010037 000000G MOV RO,URO ;MOVE TO USER'S RO
37 001370 000137 000000G JMP EMTXIT

```

```

1          .SBTTL High-efficiency TTY EMTs
2          ;-----
3          ; TURN HIGH-EFFICIENCY TTY MODE ON OR OFF.
4          ;
5 001374 105737 000000G XHISSET: TSTB EMTBLK ;TURN MODE ON OR OFF?
6 001400 001403 BEQ 1$ ;BR TO TURN IF OFF
7 001402 004737 005374' CALL HION ;TURN ON HIGH-EFFICIENCY MODE
8 001406 000403 BR HIRTN
9 001410 042761 000000G 000000G 1$: BIC ##HITTY,LSW4(R1);TURN HIGH-EFFICIENCY MODE OFF
10 001416 000137 000000G HIRTN: JMP EMTXIT
11          ;
12          ;-----
13          ; HIGH-EFFICIENCY OUTPUT.
14          ;
15 001422 013703 000002G XHIOUT: MOV EMTBLK+2,R3 ;BUFFER ADDRESS
16 001426 010300 MOV R3,R0 ;VALIDATE THE ADDRESS
17 001430 004737 000000G CALL VALADB
18 001434 013704 000004G MOV EMTBLK+4,R4 ;# BYTES TO SEND
19 001440 001766 BEQ HIRTN ;BR IF NO CHARACTERS TO SEND
20 001442 032761 000000G 000000G BIT ##HITTY,LSW4(R1);ARE WE SENDING IN HIGH EFFICIENCY MODE?
21 001450 001406 BEQ 1$ ;BR IF NOT
22 001452 004737 000000G 2$: CALL GETUCH ;GET CHAR FROM USER'S BUFFER
23 001456 004737 003672' CALL HIPUT ;SEND CHAR IN HIGH EFFICIENCY MODE
24 001462 077405 SOB R4,2$
25 001464 000754 BR HIRTN
26 001466 004737 000000G 1$: CALL GETUCH ;GET CHAR FROM USER'S BUFFER
27 001472 004737 002662' CALL PUTCHR ;SEND THE CHARACTER
28 001476 077405 SOB R4,1$
29 001500 000746 BR HIRTN
30          ;
31          ;-----
32          ; HIGH-EFFICIENCY TTY INPUT
33          ;
34 001502 013703 000002G XHIIN: MOV EMTBLK+2,R3 ;ADDRESS OF USER'S BUFFER
35 001506 010300 MOV R3,R0 ;VALIDATE THE ADDRESS
36 001510 004737 000000G CALL VALADB
37 001514 013704 000004G MOV EMTBLK+4,R4 ;SIZE OF BUFFER
38 001520 060400 ADD R4,R0 ;GET ADDRESS OF END OF BUFFER
39 001522 005300 DEC R0
40 001524 004737 000000G CALL VALADB ;VALIDATE THE ADDRESS
41 001530 005037 000000G CLR URO ;RETURN # CHARS GOTTEN IN R0
42 001534 004737 005756' 4$: CALL GETCHR ;GET NEXT CHARACTER
43 001540 004737 000000G CALL PUTUCH ;MOVE CHAR TO USER'S BUFFER
44 001544 005237 000000G INC URO ;COUNT CHARACTERS IN USER'S R0
45 001550 005761 000000G TST LACTIV(R1) ;ARE THERE ANY PENDING ACTIVATION CHARS?
46 001554 001720 BEQ HIRTN ;WE ARE DONE IF NOT
47          ; NOT ACTIVATION CHARACTER. SEE IF BUFFER IS FULL.
48 001556 077412 1$: SOB R4,4$ ;LOOP IF ROOM LEFT IN BUFFER
49          ; OVERFLOWED USER'S BUFFER. SET C-FLAG AND RETURN.
50 001560 000417 BR XTCC ;RETURN WITH C-FLAG SET
51          ;
52          ;-----
53          ; CHECK FOR TT INPUT ERRORS.
54          ;
55 001562 105737 000000G XTERCK: TSTB EMTBLK ;CHECK FOR ERRORS?
56 001566 001405 BEQ 1$ ;BR IF YES
57 001570 016137 000000G 000000G MOV LINCNT(R1),URO ;PUT # OF INPUT CHARS PENDING IN USER'S R0

```

```
58 001576 000137 000000G          JMP      EMTXIT
59 001602 032761 000000G 000000G 1$:  BIT      #$TTERR,LSW4(R1);DID ANY TT INPUT ERRORS OCCUR?
60 001610 001702                   BEQ      HIRTN          ;BR IF NOT
61 001612 042761 000000G 000000G    BIC      #$TTERR,LSW4(R1);RESET ERROR FLAG
62 001620 000137 000000G          XTCC:  JMP      SETC          ;RETURN WITH C-FLAG SET
63
64                                     ;-----
65                                     ;  START TIMER FOR TT READ REQUEST
66                                     ;
67 001624 013761 000002G 000000G XRDTIM: MOV      EMTBLK+2,LRDTIM(R1);SET TT READ TIMEOUT VALUE (0.5 SEC UNITS)
68 001632 013761 000004G 000000G      MOV      EMTBLK+4,LRTCHR(R1);SET TT TIMEOUT ACTIVATION CHARACTER
69 001640 052761 100000 000000G    BIS      #100000,LRTCHR(R1) ;SET FLAG SAYING WE HAVE A TIME-OUT CHAR
70 001646 000137 000000G          JMP      EMTXIT
```

ASKLIN -- Accept line from terminal

```

1          .SBTTL  ASKLIN -- Accept line from terminal
2          ;-----
3          ; ASKLIN is an internal subroutine called from .csispc, .csigen & .gtlin
4          ; to print a prompt and accept a line of input from the tty or a
5          ; command file.
6          ;
7          ; Inputs:
8          ; R2 = Address of prompt string (in user's area).
9          ; R3 = Address of buffer where accepted string is to be stored.
10         ; (Must be in kernel space)
11         ;
12         ASKLIN: MOV     R1,-(SP)
13         MOV     R2,-(SP)
14         MOV     R3,-(SP)
15         MOV     R4,-(SP)
16         MOV     R5,-(SP)
17         ;
18         ; Set up buffer pointer and buffer length info
19         ;
20         MOV     R3,R4          ;REMEMBER ADDRESS OF START OF BUFFER
21         MOV     R3,R5          ;GET ADDRESS OF END OF RECEIVING BUFFER
22         ADD     #80,R5         ;(GETLIN RESTRICTS BUFFER TO 81 CHARS)
23         MOVB   CORUSR,R1      ;GET CURRENT USER INDEX #
24         BIS     #GTLIN,LSW4(R1);REMEMBER THAT .GTLIN IS BEING DONE
25         ;
26         ; Determine if we need to process a deferred control-C character
27         ; that was previously acquired from an expanded CCL command by a
28         ; non-terminating .GTLIN
29         ;
30         BIT     #CFDCC,LSW4(R1);DO WE HAVE A DEFERRED ^C?
31         BEQ     10$           ;BR IF NOT
32         BIT     #GTLTTY,LJSW(R1);IS THIS A NON-TERMINATING .GTLIN?
33         BEQ     11$           ;BR IF NOT -- REPORT CONTROL-C NOW
34         ;
35         ; If this is a terminating .GTLIN and we pushed a control-C
36         ; previously due to a non-terminating .GTLIN, halt the execution
37         ; of the program without printing prompt.
38         ;
39         BIT     #NTGCC,LSW9(R1);Did we push a control-C?
40         BEQ     20$           ;Br if not
41         BIT     #GTLTTY,LJSW(R1);Is this a non-terminating .GTLIN?
42         BNE     15$           ;Br if non-terminating .GTLIN
43         BIC     #NTGCC,LSW9(R1);Say ctrl-C not pushed
44         CALL    STOP          ;Stop program execution
45         ;
46         ; Determine if the input is coming from a command file
47         ;
48         CLR     CFSPND        ;No suspended command file yet
49         CALL    GTCFCH        ;TRY TO GET A CHAR FROM COMMAND FILE
50         BCC     B$           ;BR IF GOT A CHAR FROM COMMAND FILE
51         ;
52         ; Input is not coming from a command file.
53         ; See if we need to reenter Kmon to get next command from IND or
54         ; from user command processor.
55         ;
56         BIT     #INKMN,LSW4(R1);ARE WE IN KMON NOW?
57         BEQ     3$           ;BR IF NOT

```



```

58 002002 032761 000000G 000000G BIT    #UKMON,LSW7(R1); IS USER COMMAND PROCESSOR ACTIVE?
59 002010 001006 BNE    11$ ; BR IF YES -- GET COMMAND FROM IT
60 002012 013700 000000G MOV    CXTRMN,RO ; GET ADDR OF SIMULATED RMON DATA FOR JOB
61 002016 132760 000000G 000000G BITB   #IN$ACT,R$INST(RO); IS INPUT BEING PROVIDED BY IND?
62 002024 001457 BEQ    3$ ; BR IF NOT
63 002026 004737 000000G 11$: CALL  STOP ; REENTER KMON AND GET NEXT COMMAND FROM IND
64 ;
65 ; We got a character from a control file.
66 ; See if character is a control-C or control-Z.
67 ; Note: we treat ctrl-Z the same as ctrl-C if KMON is reading file.
68 ;
69 002032 120027 000000G B$: CMPB  RO,#CTRLZ ; IS CHAR CTRL-Z?
70 002036 001006 BNE    16$ ; BR IF NOT
71 002040 032761 000000G 000000G BIT    #IN$KMN,LSW4(R1); IS KMON READING FILE?
72 002046 001402 BEQ    16$ ; BR IF NOT
73 002050 112700 000000G MOVB   #CTRLC,RO ; TRANSLATE CTRL-Z TO CTRL-C
74 002054 120027 000000G 16$: CMPB  RO,#CTRLC ; IS CHAR CTRL-C?
75 002060 001033 BNE    7$ ; BR IF NOT ^C
76 ;
77 ; Character is control-C.
78 ;
79 ; There are 3 cases to deal with:
80 ; 1. This is not a non-terminating .GTLIN. In this case we simply
81 ; accept the ctrl-C which terminates the execution of the program.
82 ; 2. This is a non-terminating .GTLIN and we are getting characters
83 ; from an expanded CCL command. In this case, we defer the ctrl-C
84 ; processing until the next non-terminating .GTLIN is done.
85 ; 3. This is a non-terminating .GTLIN and we are not getting characters
86 ; from an expanded CCL command. In this case we defer the ctrl-C
87 ; and get characters from the terminal.
88 ;
89 002062 032761 000000G 000000G BIT    #GTLTTY,LJSW(R1); IS THIS A NON-TERMINATING .GTLIN?
90 002070 001427 BEQ    7$ ; BR IF NOT -- GO TERMINATE PROGRAM
91 002072 004737 007262' 14$: CALL  GTCFCH ; SKIP TO END OF LINE THAT HAS ^C
92 002076 103403 BCS    15$
93 002100 120027 000000G CMPB   RO,#LF ; REACHED END OF LINE?
94 002104 001372 BNE    14$ ; LOOP IF NOT
95 002106 032761 000000G 000000G 15$: BIT    #CFCCCL,LSW4(R1); IS THIS THE END OF AN EXPANDED CCL COMMAND?
96 002114 001404 BEQ    12$ ; BR IF NOT
97 002116 052761 000000G 000000G BIS    #CFDCC,LSW4(R1); REMEMBER WE HAVE A DEFERRED CONTROL-C
98 002124 000717 BR     13$ ; GO BACK AND GET NEXT CHAR FOLLOWING CONTROL-C
99 002126 052761 000000G 000000G 12$: BIS    #NTGCC,LSW9(R1); Push a control-C for next .GTLIN
100 002134 013737 000000G 000000G MOV    CFPNT,CFSPND ; SAVE COMMAND FILE POINTER -- Suspend file
101 002142 004737 010016' CALL   CFSTOP ; Suspend command file input
102 002146 000406 BR     3$
103 002150 110037 000000G 7$: MOVB   RO,CFHOLD ; PUSH COMMAND FILE CHAR
104 002154 032761 000000G 000000G BIT    #QUIET,LSW4(R1); ARE WE LISTING THE COMMAND FILE?
105 002162 001012 BNE    1$ ; BR IF NOT -- DON'T LIST PROMPT THEN
106 ;
107 ; List prompt string
108 ;
109 002164 005702 3$: TST    R2 ; Is there a prompt string to print?
110 002166 001410 BEQ    1$ ; Br if not
111 002170 005737 000000G TST    CFPNT ; Is input coming from a command file?
112 002174 001003 BNE    21$ ; Br if yes
113 002176 042761 000000G 000000G BIC    #CTRLD,LSW3(R1); Reset control-D
114 002204 004737 002460' 21$: CALL  UOTSTR ; Print the prompt

```

ASKLIN -- Accept line from terminal

```

115 ;
116 ; Get the input line
117 ;
118 002210 005737 000000G 1$: TST CFPNT ; Input coming from a command file?
119 002214 001003 BNE 19$ ; Br if yes
120 002216 052761 000000G 000000G BIS ##CFABT,LSW6(R1); If ctrl-C received, abort command files
121 002224 004737 005756' 19$: CALL GETCHR ; GET AN INPUT CHAR
122 002230 042761 000000G 000000G BIC ##CFABT,LSW6(R1); Clear command file abort flag
123 002236 120027 000000G CMPB RO,#LF ; REACHED END OF LINE?
124 002242 001424 BEQ 2$ ; BR IF YES
125 002244 120027 000041 CMPB RO,#'! ; START OF COMMENT?
126 002250 001005 BNE 17$ ; BR IF NOT
127 002252 004737 005756' 18$: CALL GETCHR ; SKIP OVER REST OF COMMENT
128 002256 120027 000000G CMPB RO,#CR
129 002262 001373 BNE 18$ ; LOOP TILL WE REACH END OF COMMENT
130 002264 020305 17$: CMP R3,R5 ; ARE PAST THE END OF THE BUFFER?
131 002266 101350 BHI 1$ ; BR IF YES (DISCARD THE CHARACTER)
132 002270 110023 MOVB RO,(R3)+ ; MOVE CHAR TO BUFFER
133 002272 120027 000000G CMPB RO,#CTRLZ ; IS CHAR CTRL-Z?
134 002276 001344 BNE 1$ ; BR IF NOT
135 002300 032761 000000G 000000G BIT ##INKMN,LSW4(R1); IS KMON DOING .GTLINE?
136 002306 001740 BEQ 1$ ; BR IF NOT KMON
137 002310 004737 000000G CALL STOP ; TERMINATE PROGRAM IF CTRL-Z HIT
138 002314 020304 2$: CMP R3,R4 ; IGNORE THE LF IF IT IS AT START OF LINE
139 002316 001734 BEQ 1$
140 002320 105043 CLRB -(R3) ; REPLACE CR WITH NULL
141 ;
142 ; Finished getting input line
143 ;
144 002322 013700 000000G MOV CFSPND,R0 ; GET @FILE POINTER
145 002326 001404 BEQ 5$ ; BR IF NO NEED TO REPLACE
146 002330 004737 010042' CALL CFSTRT ; Restart command file input
147 002334 005037 000000G CLR CFSPND ; SAY THERE IS NO SUSPENDED COMMAND FILE
148 002340 042761 000000G 000000G 5$: BIC ##GTLIN,LSW4(R1); SAY .GTLINE EMT IS FINISHED
149 002346 012605 MOV (SP)+,R5
150 002350 012604 MOV (SP)+,R4
151 002352 012603 MOV (SP)+,R3
152 002354 012602 MOV (SP)+,R2
153 002356 012601 MOV (SP)+,R1
154 002360 000207 RETURN

```

OUTSTR -- Print a system message on the user's console

```

1          .SBTTL  OUTSTR -- Print a system message on the user's console
2          ;-----
3          ;  OUTSTR is called to print a system message on the user's terminal.
4          ;  Inputs:
5          ;    R2 = Address of ASCIZ string to be printed (in kernel space).
6          ;
7 002362 010046  OUTSTR:  MOV     R0,-(SP)
8 002364 010146          MOV     R1,-(SP)
9 002366 010246          MOV     R2,-(SP)
10 002370 113701 000000G  MOVB  CORUSR,R1      ;GET USER'S INDEX #
11 002374 112200 1$:  MOVB  (R2)+,R0      ;GET NEXT CHAR OF MESSAGE
12 002376 001406          BEQ   2$              ;BRANCH WHEN END HIT
13 002400 120027 000200  CMPB  R0,#200        ;STOP WITHOUT CR-LF?
14 002404 001413          BEQ   9$              ;BRANCH IF YES
15 002406 004737 002662'  CALL  PUTCHR        ;OUTPUT THE CHARACTER
16 002412 000770          BR   1$
17 002414 112700 000015 2$:  MOVB  #15,R0      ;PUT OUT CR-LF
18 002420 004737 002662'  CALL  PUTCHR
19 002424 112700 000012  MOVB  #12,R0
20 002430 004737 002662'  CALL  PUTCHR
21 002434 012602 9$:  MOV   (SP)+,R2
22 002436 012601          MOV   (SP)+,R1
23 002440 012600          MOV   (SP)+,R0
24 002442 000207          RETURN

```

```

25
26          .SBTTL  CSIMSG -- Print a CSI asciz error message
27          ;-----
28          ;  CSIMSG is called to print a CSI asciz error message on the user's
29          ;  terminal.
30          ;
31          ;  Inputs:
32          ;    R2 = CSI error message code.
33          ;
34 002444          CSIMSG:
35 002444 006302          ASL   R2              ;MULTIPLY BY TWO
36 002446 016202 000004'  MOV   CSIERR(R2),R2  ;INDEX TO THE CORRECT ERROR STRING
37 002452 004737 002362'  CALL  OUTSTR        ;PRINT THE ERROR MESSAGE
38 002456 000207          RETURN        ;RETURN TO CONTINUE PROCESSING

```

```

1          .SBTTL  UOTSTR -- print an asciz user's message on the user's console
2          ;-----
3          ; UOTSTR is called to print an asciz message that is stored in the
4          ; user's area.
5          ;
6          ; Inputs:
7          ; R2 = Address of asciz string to be printed (in user's space).
8          ;
9 002460 010046 UOTSTR: MOV      R0,-(SP)
10 002462 010146      MOV      R1,-(SP)
11 002464 010246      MOV      R2,-(SP)
12 002466 010346      MOV      R3,-(SP)
13 002470 010446      MOV      R4,-(SP)
14 002472 012704 000000G  MOV      #TTCSCH,R4      ;SET CHARACTER COUNTER FOR SCHEDULING CALL
15 002476 113701 000000G  MOVB     CORUSR,R1      ;GET USER'S INDEX #
16 002502 012703 000200  MOV      #200,R3      ;GET CHAR USED TO STOP STRING WITHOUT CR-LF
17 002506 032702 000001  BIT      #1,R2      ;IS BUFFER ON EVEN OR ODD BYTE BOUNDARY?
18 002512 001420      BEQ      3$      ;BR IF ON EVEN BYTE BOUNDARY
19 002514 005302      DEC      R2      ;POINT TO 1ST BYTE OF WORD
20 002516 106522      MFPD    (R2)+      ;GET WORD CONTAINING BYTE
21 002520 000426      BR      4$
22          ;
23          ; See if we need to interrupt output processing for a job scheduling cycle.
24          ;
25 002522 012704 000000G  1$:      MOV      #TTCSCH,R4      ;RESET THE CHARACTER COUNT FOR SCHEDULING CALL
26 002526 105737 000000G  TSTB    DOSCHD      ;IS JOB SCHEDULER CYCLE NEEDED?
27 002532 001410      BEQ      3$      ;BR IF NOT
28 002534 004737 000000G  CALL    CHKABT      ;SEE IF WE HAVE BEEN ABORTED
29 002540 016100 000000G  MOV     LSTATE(R1),R0 ;GET JOB'S CURRENT EXECUTION STATE
30 002544 004737 000000G  CALL    QNSPNX      ;REQUEUE JOB AND CALL JOB SCHEDULER
31 002550 004737 000000G  CALL    CHKABT      ;SEE IF WE WERE ABORTED WHILE ASLEEP
32 002554 005304      3$:      DEC      R4      ;TIME TO CHECK FOR SCHEDULER CYCLE?
33 002556 003761      BLE     1$      ;BR IF YES
34 002560 106522      MFPD    (R2)+      ;GET WORD WITH NEXT 2 BYTES OF STRING
35 002562 111600      MOVB   (SP),R0      ;GET NEXT BYTE OF STRING
36 002564 001417      BEQ     5$      ;BR IF NULL AT END OF STRING HIT
37 002566 120003      CMPB   R0,R3      ;IS CHAR #200?
38 002570 001413      BEQ     10$     ;IF YES THEN END STRING WITHOUT CRLF
39 002572 004737 002662'  CALL    PUTCHR      ;SEND CHAR TO TERMINAL
40 002576 012600      4$:      MOV     (SP)+,R0     ;GET WORD WITH CHAR IN HIGH-ORDER BYTE
41 002600 105000      CLRB   R0      ;CLEAR LOW-ORDER BYTE
42 002602 000300      SWAB   R0      ;MOVE HIGH-ORDER BYTE TO LOW-ORDER
43 002604 001410      BEQ     2$      ;BR IF NULL AT END OF STRING HIT
44 002606 120003      CMPB   R0,R3      ;IS THIS 200 AT END OF STRING?
45 002610 001416      BEQ     9$      ;BR IF YES
46 002612 004737 002662'  CALL    PUTCHR      ;SEND CHAR TO TERMINAL
47 002616 000756      BR     3$      ;GO SEND MORE CHARS
48 002620 005726      10$:     TST    (SP)+      ;CLEAN OFF STACK
49 002622 000411      BR     9$
50 002624 005726      5$:      TST    (SP)+      ;CLEAN OFF STACK
51 002626 112700 000015  2$:      MOVB   #15,R0      ;PUT OUT CR-LF
52 002632 004737 002662'  CALL    PUTCHR
53 002636 112700 000012  MOVB   #12,R0
54 002642 004737 002662'  CALL    PUTCHR
55 002646 012604      9$:      MOV    (SP)+,R4
56 002650 012603      MOV    (SP)+,R3
57 002652 012602      MOV    (SP)+,R2

```

58 002654 012601  
59 002656 012600  
60 002660 000207

MOV (SP)+,R1  
MOV (SP)+,R0  
RETURN

```

1          .SBTTL
2          .SBTTI  ** Program Level Output Character Processing **
3          .SBTTL  PUTCHR -- Send character to terminal
4          ;-----
5          ; PUTCHR is called to queue a character to be sent to a terminal.
6          ;
7          ; Inputs:
8          ;   RO = Character to be sent.
9          ;   R1 = Virtual line number.
10         ;
11 002662  PUTCHR:
12         ;
13         ; If line is in "high efficiency" mode, bypass a lot of normal
14         ; processing and use the high efficiency version of PUTCHR.
15         ;
16 002662  032761  000000G 000000G          BIT    ##HITTY,LSW4(R1); Is this line in high efficiency mode?
17 002670  001402                      BEQ    1$          ;Br if not
18 002672  000137  003672'                JMP    HIPUT        ;Use high efficiency version of PUTCHR
19         ;
20         ; Return immediately if the line has disconnected
21         ;
22 002676  032761  000000C 000000G 1$:    BIT    #<$DISCN!$CTRLC>,LSW(R1) ;Has job disconnected?
23 002704  001406                      BEQ    2$          ;Br if not
24 002706  032761  000000G 000000G          BIT    ##NOIN,LSW3(R1) ;Are we doing logoff processing now?
25 002714  001057                      BNE    9$          ;Br if yes -- return immediately from PUTCHR
26 002716  004737  000000G                CALL   STOP        ;Terminate execution of job and enter KMON
27         ;
28         ; Mask character to 7 or 8 bits depending on setting of EIGHTBIT option.
29         ;
30 002722  042700  177400                2$:    BIC    #^C<377>,RO    ;Mask character to 8 bits
31 002726  032761  000000G 000000G          BIT    ##8BIT,LSW2(R1) ;Is eightbit support option selected?
32 002734  001002                      BNE    8$          ;Br if yes
33 002736  042700  177600                BIC    #^C<177>,RO    ;Mask character to 7 bits
34         ;
35         ; See if we should suppress output from programs such as
36         ; EDIT, TECO, and DIBOL which try to echo characters themselves.
37         ;
38 002742  032761  000000G 000000G 8$:    BIT    ##NOOUT,LSW3(R1); Is output echo suppression in effect?
39 002750  001403                      BEQ    3$          ;Br if not
40 002752  004737  004052'                CALL   ESCHK        ;See if we should discard this character
41 002756  103436                      BCS    9$          ;Br if we should discard this character
42         ;
43         ; Remember the last character output to this line
44         ;
45 002760  110061  000000G                3$:    MOVB   RO,LSNDCH(R1) ;Remember last character sent to line
46         ;
47         ; If character is being sent in transparent mode, set transparent
48         ; flag for this character
49         ;
50 002764  032761  000000G 000000G          BIT    ##TRNSP,LSW3(R1); Is line is transparency mode?
51 002772  001403                      BEQ    4$          ;Br if not
52 002774  052700  000000G                BIS    #TRNSFL,RO    ;Set transparency flag for this character
53 003000  000415                      BR     5$
54         ;
55         ; Determine if this character is a leadin character or part of
56         ; a lead-in function sequence.
57         ;

```



PUTCH1 -- Queue character for terminal

```

1          .SBTTL  PUTCH1 -- Queue character for terminal
2          ;-----
3          ; PUTCH1 is the second level of character queueing routines.
4          ; Calling this routine rather than PUTCHR bypasses the following
5          ; character processing operations:
6          ;
7          ; 1. Stop the job if it is disconnected.
8          ; 2. Echo suppression.
9          ; 3. Transparent character flagging.
10         ; 4. TSX lead-in function processing.
11         ; 5. Terminal logging.
12         ;
13         ; Inputs:
14         ; RO = Character to be queued for the terminal.
15         ; R1 = Virtual line number of the terminal.
16         ;
17 003056  PUTCH1:
18         ;
19         ; If this is a detached job, discard its output
20         ;
21 003056 032761 000000G 000000G      BIT    %%DETCH,LSW(R1) ;Is this a detached job?
22 003064 001015                BNE    9$          ;Br if yes -- discard its output
23         ;
24         ; If we are in a command file and program output is being suppressed,
25         ; discard this character.
26         ;
27 003066 032761 000000G 000000G      BIT    %%CFSOT,LSW4(R1);Should we suppress command file output?
28 003074 001403                BEQ    1$          ;Br if not
29 003076 005737 000000G          TST    CFPNT      ;Is input coming from a command file?
30 003102 001006                BNE    9$          ;Br if yes -- discard this character
31         ;
32         ; If Control-O has been typed, discard this character
33         ;
34 003104 032761 000000G 000000G 1$:  BIT    %%CTRL0,LSW3(R1);Has ctrl-O been typed?
35 003112 001002                BNE    9$          ;Br if yes -- discard this character
36         ;
37         ; We want to queue this character for the terminal.
38         ; Call PUTCH2 to do the next level of processing
39         ;
40 003114 004737 003122'          CALL    PUTCH2          ;Queue character for the terminal
41         ;
42         ; Finished
43         ;
44 003120 000207          9$:  RETURN

```



PUTCH2 -- Queue character for a terminal

```

1          .SBTTL  PUTCH2 -- Queue character for a terminal
2          ;-----
3          ; PUTCH2 is the third level of character processing associated with
4          ; PUTCHR. Calling PUTCH2 rather than PUTCH1 bypasses the following
5          ; character processing operations:
6          ;
7          ; 1. Discarding output for detached jobs.
8          ; 2. Suppressing output while inside command file.
9          ; 3. Suppressing output if Control-Q typed.
10         ;
11         ; Inputs:
12         ; R0 = Character to be queued for the terminal.
13         ; R1 = Virtual line index number.
14         ;
15 003122 010246  PUTCH2: MOV      R2,-(SP)
16         ;
17         ; If character is being sent in transparency mode, by pass
18         ; keeping track of its position.
19         ;
20 003124 032700 000000G      BIT      #TRNSFL,R0      ;Is character in transparency mode?
21 003130 001015          BNE      2$              ;Br if yes
22         ;
23         ; Determine if this is a regular or control character.
24         ;
25 003132 020027 000037 3$:   CMP      R0,#37          ;Is this a control character?
26 003136 101005          BHI      1$              ;Br if not
27         ;
28         ; This is a control character.
29         ; Call appropriate routine to process it.
30         ;
31 003140 010002          MOV      R0,R2          ;Get character
32 003142 006302          ASL      R2              ;Convert character to word table index
33 003144 004772 003174'   CALL   @PC2VEC(R2)    ;Call processing routine
34 003150 000407          BR       9$              ;Finished with character
35         ;
36         ; This is a regular character.
37         ; Keep track of cursor position.
38         ;
39 003152 120027 000000G 1$:   CMPB   R0,#RUBOUT    ;Is this a rubout character?
40 003156 001402          BEQ      2$              ;Br if yes
41 003160 105261 000000G   INCB   LCOL(R1)      ;Advance cursor column position
42         ;
43         ; Insert this character into the terminal buffer
44         ;
45 003164 004737 003462' 2$:   CALL   QUECHR        ;Insert character into terminal buffer
46         ;
47         ; Finished
48         ;
49 003170 012602          9$:   MOV      (SP)+,R2
50 003172 000207          RETURN

```

PUTCH2 -- Queue character for a terminal

```

1
2 ; -----
3 ; Vector of addresses of character processing routines for
4 ; control characters encountered by PUTCH2.
5 ;
6 ; On entry to the processing routine, the following registers will
7 ; be set up:
8 ;   R0 = Character
9 ;   R1 = Virtual line number

```

```

10 003174 003274' PC2VEC: .WORD PCCINS ; 00 - NUL
11 003176 003274' .WORD PCCINS ; 01 - SOH
12 003200 003274' .WORD PCCINS ; 02 - STX
13 003202 003274' .WORD PCCINS ; 03 - ETX
14 003204 003274' .WORD PCCINS ; 04 - EOT
15 003206 003274' .WORD PCCINS ; 05 - ENQ
16 003210 003274' .WORD PCCINS ; 06 - ACK
17 003212 003274' .WORD PCCINS ; 07 - BEL
18 003214 003302' .WORD PCCBS ; 10 - BS
19 003216 003334' .WORD PCCHT ; 11 - HT
20 003220 003274' .WORD PCCINS ; 12 - LF
21 003222 003274' .WORD PCCINS ; 13 - VT
22 003224 003420' .WORD PCCFF ; 14 - FF
23 003226 003322' .WORD PCCCR ; 15 - CR
24 003230 003274' .WORD PCCINS ; 16 - SO
25 003232 003274' .WORD PCCINS ; 17 - SI
26 003234 003274' .WORD PCCINS ; 20 - DLE
27 003236 003274' .WORD PCCINS ; 21 - DC1
28 003240 003274' .WORD PCCINS ; 22 - DC2
29 003242 003274' .WORD PCCINS ; 23 - DC3
30 003244 003274' .WORD PCCINS ; 24 - DC4
31 003246 003274' .WORD PCCINS ; 25 - NAK
32 003250 003274' .WORD PCCINS ; 26 - SYN
33 003252 003274' .WORD PCCINS ; 27 - ETB
34 003254 003274' .WORD PCCINS ; 30 - CAN
35 003256 003274' .WORD PCCINS ; 31 - EM
36 003260 003274' .WORD PCCINS ; 32 - SUB
37 003262 003274' .WORD PCCINS ; 33 - ESC
38 003264 003274' .WORD PCCINS ; 34 - FS
39 003266 003274' .WORD PCCINS ; 35 - GS
40 003270 003274' .WORD PCCINS ; 36 - RS
41 003272 003274' .WORD PCCINS ; 37 - US

```

```

1 ; -----
2 ; Processing routines for control characters sent to terminal through
3 ; PUTCH2.
4 ;
5 ; Normal control character.
6 ; Insert into terminal buffer but do not affect cursor position.
7 ;
8 003274 004737 003462' PCCINS: CALL QUECHR ;Put char into terminal buffer
9 003300 000207 RETURN
10 ;
11 ; Backspace -- Backup cursor position
12 ;
13 003302 105361 000000G PCCBS: DECB LCOL(R1) ;Backup cursor position
14 003306 002002 BGE 1$ ;Br if didn't go past front of line
15 003310 105061 000000G CLRB LCOL(R1) ;Don't allow to go to left of line start
16 003314 004737 003462' 1$: CALL QUECHR ;Queue the character
17 003320 000207 RETURN
18 ;
19 ; Carriage return
20 ;
21 003322 105061 000000G PCCCR: CLRB LCOL(R1) ;Say cursor is at left margin
22 003326 004737 003462' CALL QUECHR ;Queue the character
23 003332 000207 RETURN
24 ;
25 ; Tab -- Translate to spaces if tab simulation wanted
26 ;
27 003334 010246 PCCHT: MOV R2, -(SP)
28 003336 116102 000000G MOVB LCOL(R1), R2 ;Get current column position
29 003342 032761 000000G 000000G BIT ##TAB, LSW2(R1) ;Should we simulate tabs?
30 003350 001011 BNE 1$ ;Br if not
31 003352 112700 000040 MOVB #' , R0 ;Get space character
32 003356 004737 003462' 2$: CALL QUECHR ;Send a space
33 003362 005202 INC R2 ;Advance cursor position
34 003364 032702 000007 BIT #7, R2 ;Are we up to next tab stop?
35 003370 001372 BNE 2$ ;Loop if not
36 003372 000406 BR 9$
37 003374 004737 003462' 1$: CALL QUECHR ;Send the tab character to the terminal
38 003400 062702 000010 ADD #8, R2 ;Bound up to next tab stop
39 003404 042702 000007 BIC #7, R2
40 003410 110261 000000G 9$: MOVB R2, LCOL(R1) ;Save new cursor position
41 003414 012602 MOV (SP)+, R2
42 003416 000207 RETURN
43 ;
44 ; Form feed -- See if we should translate to spaces
45 ;
46 003420 032761 000000G 000000G PCCFF: BIT ##FORM, LSW2(R1) ;Should we simulate form-feeds?
47 003426 001012 BNE 2$ ;Br if not
48 003430 010246 MOV R2, -(SP)
49 003432 012702 000010 MOV #8, R2 ;Put out 8 LF characters
50 003436 112700 000000G MOVB #LF, R0 ;Get line feed character
51 003442 004737 003462' 1$: CALL QUECHR ;Send a line feed
52 003446 077203 SOB R2, 1$ ;Loop till all line feeds sent
53 003450 012602 MOV (SP)+, R2
54 003452 000402 BR 9$
55 003454 004737 003462' 2$: CALL QUECHR ;Send FF character to terminal
56 003460 000207 9$: RETURN

```

QUECHR -- Queue character for transmission

```

1          .SBTTL  QUECHR -- Queue character for transmission
2          ;-----
3          ; QUECHR is called to queue a character for transmission.
4          ; If the job has display windowing turned on, the character is passed
5          ; to the window manager before being queued for transmission.
6          ;
7          ; Inputs:
8          ;   R0 = Character to be queued.
9          ;   R1 = Virtual terminal number.
10         ;
11 003462  QUECHR:
12         ;
13         ; See if display windowing is turned on for this line
14         ;
15 003462 005761 000000G          TST      LWINDO(R1)      ; Is job doing display windowing?
16 003466 001410                BEQ      1$              ; Br if not
17 003470 032761 000000G 000000G  BIT      #$NOWIN,LSW11(R1); Are we suppressing windowing now?
18 003476 001004                BNE      1$              ; Br if yes
19 003500                OCALL   WINCHR          ; Pass character to window manager
20 003506 103402                BCS      9$              ; Br if we should not display this char
21         ;
22         ; Queue the character for transmission
23         ;
24 003510 004737 003516'        1$:     CALL   BUFCHR          ; Queue character for transmission
25         ;
26         ; Finished
27         ;
28 003514 000207                9$:     RETURN

```

BUFCHR -- Insert char or suspend if full

```

1          .SBTTL  BUFCHR -- Insert char or suspend if full
2          ;-----
3          ; BUFCHR is called to queue a character for a terminal.
4          ; If there is adequate space in the terminal output buffer the character
5          ; is inserted.  If there is not adequate space the job is suspended
6          ; until space becomes available in the output terminal buffer.
7          ;
8          ; Inputs:
9          ;   RO = Character to be queued.
10         ;   R1 = Virtual terminal number.
11         ;
12 003516 010246  BUFCHR: MOV      R2,-(SP)
13         ;
14         ; Disable interrupts
15         ;
16 003520 1$:      DISABL          ;** Disable **
17         ;
18         ; See if there is room in the output buffer for the character
19         ;
20 003526 026127 000000G 000010  CMP      LOTSPC(R1),#8.  ;Is there plenty of free space in buffer?
21 003534 101016      BHI      2$          ;Br if yes
22 003536 105737 000000G      TSTB     FRKPRI      ;Are we running at interrupt level? (echoing)
23 003542 001404      BEQ      3$          ;Br if not at interrupt level
24 003544 005761 000000G      TST      LOTSPC(R1)    ;Allow char echoing to use all of buffer
25 003550 003010      BGT      2$          ;Br if buffer not completely full
26 003552 000442      BR       9$          ;Discard char if no space for echo
27         ;
28         ; Output buffer is full.
29         ; Suspend job's execution and wait for buffer space to becom available.
30         ;
31 003554 004737 005674' 3$:      CALL     PCSPND      ;Suspend execution of job ** Enable **
32 003560 032761 000000G 000000G  BIT      %#CTRL0,LSW3(R1);Was Ctrl-O typed while we were asleep?
33 003566 001754      BEQ      1$          ;Br if not
34 003570 000433      BR       9$          ;Discard character if ctrl-O typed
35         ;
36         ; There is room for the character in the output buffer
37         ; Insert character into the buffer
38         ;
39 003572 005361 000000G 2$:      DEC      LOTSPC(R1)    ;Decrease free space count for buffer
40 003576 016102 000000G      MOV      LOTNXT(R1),R2  ;Get pointer to next free cell in buffer
41 003602      TTMAP      ;Map kernel PAR 6 to TT buffer
42 003616 110022      MOVB     RO,(R2)+      ;Move character into TT buffer
43 003620      UNMAP     ;Remap kernel PAR 6 to job context block
44 003626 020261 000000G      CMP      R2,LOTEND(R1)    ;Did we move beyond end of buffer?
45 003632 103402      BLO      4$          ;Br if not
46 003634 016102 000000G      MOV      LOTBUF(R1),R2  ;Wrap around to front of buffer
47 003640 010261 000000G 4$:      MOV      R2,LOTNXT(R1)  ;Save next-character pointer
48         ;
49         ; Start transmitter for this line
50         ;
51 003644      ENABL     ;** Enable **
52 003652 004777 000000G      CALL     @TRNSTR      ;Start transmitter for this line
53 003656 000403      BR       10$
54         ;
55         ; Finished
56         ;
57 003660 9$:      ENABL     ;** Enable **

```

58 003666 012602  
59 003670 000207

10\$: MOV (SP)+, R2  
RETURN

HIPUT -- High efficiency PUTCHR

```

1          .SBTTL  HIPUT  -- High efficiency PUTCHR
2          ;-----
3          ; HIPUT -- Routine to move a character to the terminal output buffer
4          ; and print it in high efficiency mode.
5          ;
6          ; Inputs:
7          ; R0 = Character to be sent.
8          ; R1 = Virtual line number of job.
9          ;
10         003672  HIPUT;;
11         003672  010246      MOV      R2,-(SP)
12         ;
13         ; If this is a detached job, discard its output
14         ;
15         003674  032761  000000G 000000G      BIT      ##DETCH,LSW(R1) ;Is this a detached job?
16         003702  001061          BNE      9$          ;Br if yes -- discard its output
17         003704  032761  000000C 000000G      BIT      #<#$DISCN+#$CTRLC+#$DETCH>,LSW(R1) ;IS LINE DISCONNECTED?
18         003712  001402          BEQ      1$          ;BR IF NOT
19         003714  004737  000000G          CALL     STOP          ;STOP JOB
20         ;
21         ; See if display windowing is turned on for this line
22         ;
23         003720  005761  000000G      1$:      TST      LWINDO(R1)      ;Is job doing display windowing?
24         003724  001404          BEQ      4$          ;Br if not
25         003726          DCALL     WINCHR      ;Pass character to window manager
26         003734  103444          BCS      9$          ;Br if we should not display this char
27         ;
28         ; See if there is room in the output buffer for this character
29         ;
30         003736      4$:      DISABL          ;** DISABLE **
31         003744  026127  000000G 000010      CMP      LOTSPC(R1),#8. ;ENOUGH SPACE IN OUTPUT BUFFER?
32         003752  101003          BHI      2$          ;BR IF PLENTY OF SPACE
33         003754  004737  005674'          CALL     PCSPND      ;SUSPEND PROGRAM TILL SPACE IS AVAILABLE
34         003760  000766          BR       4$          ;NOW GO TRY AGAIN
35         ;
36         ; Insert character into output buffer
37         ;
38         003762  005361  000000G      2$:      DEC      LOTSPC(R1)      ;SAY LESS SPACE AVAILABLE IN BUFFER
39         003766  016102  000000G      MOV      LOTNXT(R1),R2 ;GET POINTER TO FREE CELL IN BUFFER
40         003772          TTMAP          ;MAP TO TT BUFFER AREA
41         004006  110022          MOVVB   R0,(R2)+      ;MOVE CHAR INTO TT BUFFER
42         004010          UNMAP          ;
43         004016  020261  000000G      CMP      R2,LOTEND(R1) ;HAVE WE HIT THE END OF THE TT BUFFER?
44         004022  103402          BLO      3$          ;BR IF NOT
45         004024  016102  000000G      MOV      LOTBUF(R1),R2 ;WRAPAROUND TO FRONT OF BUFFER
46         004030  010261  000000G      3$:      MOV      R2,LOTNXT(R1) ;SAVE NEW BUFFER POINTER
47         004034          ENABL          ;** ENABLE **
48         ;
49         ; Try to start transmission to line
50         ;
51         004042  004777  000000G          CALL     @TRNSTR      ;TRY TO START TRANSMISSION TO LINE
52         ;
53         ; Finished
54         ;
55         004046  012602      9$:      MOV      (SP)+,R2
56         004050  000207          RETURN

```

```

1          .SBTTL  ESCHK  -- Check for echo suppression restart
2          ;-----
3          ; ESCHK is called from PUTCHR to determine if echo suppression
4          ; that is currently in effect should be terminated.
5          ;
6          ; Inputs:
7          ;   RO = Current character being output.
8          ;   RI = Virtual line number.
9          ;
10         ; Outputs:
11         ;   C-flag set ==> Discard the character.
12         ;   RO = Character to output to terminal.
13         ;
14 004052  ESCHK:
15         ;
16         ; Jump to appropriate restart routine based on echo suppression class
17         ;
18 004052  000171  000000G      JMP      @LESRTN(RI)      ; Jump to echo suppression restart routine
19         ;
20         ; Restart after a normal character
21         ;
22 004056  010046      ESUAC:  MOV      RO, -(SP)      ; PUSH CHARACTER BEING SENT
23 004060  020027  000141      CMP      RO, #141      ; IS IT A LOWER CASE LETTER?
24 004064  103405      BLO     1$          ; BR IF NOT
25 004066  020027  000172      CMP      RO, #172
26 004072  101002      BHI     1$
27 004074  162716  000040      SUB     #40, (SP)      ; CONVERT LOWER-CASE TO UPPER-CASE
28 004100  122661  000000G  1$:  CMPB   (SP)+, LESCHR(RI) ; IS CHAR BEING SENT SAME AS ECHO-SUPPRESSION CHAR?
29 004104  001445      BEQ     ESR2         ; IF YES THEN RESTART WITH NEXT CHAR
30 004106  000453      BR     ESR1         ; IF NOT RESTART IMMEDIATELY
31         ;
32         ; Restart output after control type activation chars (ctrl-x etc.)
33         ;
34 004110  120061  000000G  ESCTL:  CMPB   RO, LESCHR(RI) ; CONTROL CHAR ECHOED BACK?
35 004114  001441      BEQ     ESR2         ; IF YES RESTART WITH NEXT CHAR
36 004116  120027  000136      CMPB   RO, #'^      ; ECHO ^-CHAR?
37 004122  001441      BEQ     ESRTN        ; IF YES WAIT FOR NEXT CHAR
38 004124  126127  000000G  000136  CMPB   LSNDCH(RI), #'^ ; WAS PREVIOUS CHAR ^?
39 004132  001432      BEQ     ESR2         ; IF YES, RESTART WITH NEXT CHAR
40 004134  000440      BR     ESR1         ; RESTART WITH THIS CHAR
41         ;
42         ; Restart output after cr or lf.
43         ;
44 004136  120027  000000G  ESCRLF:  CMPB   RO, #CR      ; IS THIS CHAR CR?
45 004142  001431      BEQ     ESRTN        ; YES -- KEEP SKIPPING
46 004144  120027  000000G      CMPB   RO, #LF      ; IS IT LF?
47 004150  001423      BEQ     ESR2         ; YES -- RESTART WITH NEXT CHAR.
48 004152  000431      BR     ESR1         ; NOT CR OF LF. RESTART IMMEDIATELY
49         ;
50         ; Restart after form feed
51         ;
52 004154  120027  000000G  ESFF:  CMPB   RO, #FF      ; ECHO FORM FEED CHAR?
53 004160  001417      BEQ     ESR2         ; IF YES, RESTART WITH NEXT CHAR
54 004162  120027  000000G      CMPB   RO, #LF      ; IGNORE A STRING OF LF'S
55 004166  001417      BEQ     ESRTN
56 004170  000422      BR     ESR1
57         ;

```



ESCHK -- Check for echo suppression restart

```

58          ; Restart after escape.
59          ;
60 004172 120027 000044      ESESC:  CMPB    RO,#'$      ; OK TO ECHO $ TOO.
61 004176 001410              BEQ     ESR52
62 004200 000416              BR      ESR51      ; RESTART IMMEDIATELY.
63          ;
64          ; Restart after tab.
65          ;
66 004202 120027 000000G     ESHT:   CMPB    RO,#TAB      ; ECHO TAB CHAR?
67 004206 001404              BEQ     ESR52      ; IF YES, RESTART WITH NEXT CHAR
68 004210 120027 000040      CMPB    RO,#'      ; IGNORE A STRING OF BLANKS
69 004214 001404              BEQ     ESRTN
70 004216 000407              BR      ESR51
71          ;
72          ; Set output restart with next character.
73          ;
74 004220 042761 000000G 000000G ESRS2:  BIC     ##NOOUT,LSW3(R1); RESTART OUTPUT WITH NEXT CHAR
75 004226 110061 000000G     ESRTN:  MOVB   RO,LSNDCH(R1) ; REMEMBER LAST CHAR SENT
76 004232 000261              SEC     ; SAY TO DISCARD THIS CHARACTER
77 004234 000404              BR      ESXIT
78          ;
79          ; Restart output immediately with this character.
80          ;
81 004236 042761 000000G 000000G ESRS1:  BIC     ##NOOUT,LSW3(R1); SAY ECHO SUPPRESSION IS TURNED OFF
82 004244 000241              CLC     ; Say to restart immediately
83 004246 000207      ESXIT:  RETURN

```

LIFUN -- Process lead-in function sequences

```

1          .SBTTL LIFUN -- Process lead-in function sequences
2          ;-----
3          ; LIFUN is called from PUTCHR to process TSX lead-in character sequences
4          ;
5          ; Inputs:
6          ;   RO = Current character
7          ;   R1 = Virtual line number
8 004250 010246 LIFUN:  MOV     R2,-(SP)
9 004252 010346          MOV     R3,-(SP)
10 004254 010446          MOV     R4,-(SP)
11 004256 010546          MOV     R5,-(SP)
12          ;
13          ; Enter correct processing routine
14          ;
15 004260 016102 000000G  MOV     LTSCMD(R1),R2 ;Get address of correct processing routine
16 004264 005061 000000G  CLR     LTSCMD(R1)   ;Say no processing routine pending now
17 004270 004712          CALL    (R2)         ;Enter processing routine
18          ;
19          ; Finished
20          ;
21 004272 012605          MOV     (SP)+,R5
22 004274 012604          MOV     (SP)+,R4
23 004276 012603          MOV     (SP)+,R3
24 004300 012602          MOV     (SP)+,R2
25 004302 000207          RETURN
26          ;
27          ;-----
28          ; Get letter after leadin character and identify command.
29          ;
30 004304 162700 000101 GTCM1:  SUB     #'A,R0   ; CONVERT LETTER TO TABLE INDEX
31 004310 100406          BMI     9$         ; BR IF NOT LEGAL COMMAND
32 004312 020027 000032  CMP     R0,#MAXCC  ; SEE IF IN LEGAL RANGE
33 004316 103003          BHI     9$         ; BR IF TOO BIG
34 004320 006300          ASL    R0         ; MAKE INTO WORD TABLE INDEX
35 004322 004770 004330'  CALL    @CCVECT(R0) ; ENTER PROCESSING ROUTINE
36          ;
37          ; Finished
38          ;
39 004326 000207 9$:    RETURN

```

```

1          ; -----
2          ;   Define TSX command control characters
3          ;
4 004330 004414' CCVECT: .WORD  CMDA          ; SET RUBOUT FILLER CHARACTER
5 004332 004432'          .WORD  CMDB          ; ENABLE VT50 ESC-LETTER ACTIVATION
6 004334 004442'          .WORD  CMDC          ; DISABLE VT50 ESC-LETTER ACTIVATION
7 004336 004452'          .WORD  CMDD          ; DEFINE ACTIVATION CHARACTER
8 004340 004702'          .WORD  CMDE          ; TURN ON ECHO
9 004342 004712'          .WORD  CMDF          ; TURN OFF ECHO
10 004344 004722'         .WORD  CMDG          ; NOP
11 004346 004724'         .WORD  CMDH          ; DISABLE VIRTUAL LINES
12 004350 004734'         .WORD  CMDI          ; ENABLE LOWER CASE INPUT
13 004352 004764'         .WORD  CMDJ          ; DISABLE LOWER CASE INPUT
14 004354 004774'         .WORD  CMDK          ; SET DEFERRED CHAR ECHO MODE
15 004356 005004'         .WORD  CMDL          ; SET IMMEDIATE CHAR ECHO MODE
16 004360 005022'         .WORD  CMDM          ; SET TRANSPARENCY MODE ON
17 004362 005032'         .WORD  CMDN          ; SUSPEND COMMAND FILE INPUT
18 004364 005052'         .WORD  CMDO          ; RESTART COMMAND FILE INPUT
19 004366 005072'         .WORD  CMDP          ; RESET USER ACTIVATION CHAR
20 004370 005202'         .WORD  CMDQ          ; SET ACTIVATION ON FIELD WIDTH
21 004372 005356'         .WORD  CMDR          ; TURN ON HIGH-EFFICIENCY TTY MODE
22 004374 005414'         .WORD  CMDS          ; TURN ON SINGLE CHARACTER ACTIVATION MODE
23 004376 005424'         .WORD  CMDT          ; TURN OFF SINGLE CHARACTER ACTIVATION MODE
24 004400 005434'         .WORD  CMDU          ; ENABLE NON-WAIT TT INPUT
25 004402 005444'         .WORD  CMDV          ; SET FIELD WIDTH LIMIT
26 004404 005634'         .WORD  CMDW          ; TURN TAPE MODE ON
27 004406 005644'         .WORD  CMDX          ; TURN TAPE MODE OFF
28 004410 005654'         .WORD  CMDY          ; DISABLE AUTO LF ECHO
29 004412 005664'         .WORD  CMDZ          ; ENABLE AUTO LF ECHO
30          000032          MAXCC =          <. -CCVECT>/2 ; # COMMANDS

```

```
1 ;-----  
2 ; Command - A  
3 ; The next char will be the rubout filler character.  
4 ;  
5 004414 012761 004424' 000000G CMDA:  MOV    #SETRBF,LTSCMD(R1);SET NEXT PROCESSING ROUTINE  
6 004422 000207                RETURN  
7 ;  
8 ; THIS CHAR IS THE RUBOUT FILLER CHARACTER.  
9 ;  
10 004424 110061 000000G   SETRBF:  MOVB   RO,LRBFIL(R1)  ;SET NEW RUBOUT FILLER CHARACTER  
11 004430 000207                RETURN  
12 ;  
13 ;-----  
14 ; Command - B  
15 ; Enable activation on vt50 escape-letter sequence.  
16 ;  
17 004432 052761 000000G 000000G CMDB:  BIS    #$VTEC,LSW5(R1);ENABLE THAT ACTIVATION SEQUENCE  
18 004440 000207                RETURN  
19 ;  
20 ;-----  
21 ; Command - C  
22 ; Disable activation on vt50 escape-letter sequence.  
23 ;  
24 004442 042761 000000G 000000G CMDC:  BIC    #$VTEC,LSW5(R1);DISABLE ACTIVATION CLASS  
25 004450 000207                RETURN
```

```

1          ; -----
2          ; Command - D
3          ; Define additional activation character (which follows).
4          ;
5 004452 012761 004462' 000000G CMDD:  MOV    #GTSPAC,LTSCMD(R1); SET PROCESSING ROUTINE
6 004460 000207          RETURN
7          ;
8          ; This character is a new activation character.
9          ;
10 004462 010346 GTSPAC: MOV    R3,-(SP)
11 004464 010446     MOV    R4,-(SP)
12 004466 010546     MOV    R5,-(SP)
13 004470 032761 000000G 000000G     BIT    #$DETCH,LSW(R1) ; Is this a detached job?
14 004476 001075     BNE    9$          ; Br if yes -- Ignore function
15 004500 004737 005120'     CALL  DELUAC          ; DEL CHAR IF ALREADY IN TABLE
16 004504 016102 000000G     MOV    LNSPAC(R1),R2 ; GET # OF CHARS DEFINED SO FAR
17 004510 020227 000000G     CMP    R2,#MXSPAC    ; DEFINED ALL ALLOWED?
18 004514 103066     BHIS   9$          ; CAN'T HAVE ANY MORE
19 004516 066102 000000G     ADD    LSPACT(R1),R2 ; POINT TO FREE SPOT IN TABLE
20 004522 110012     MOVB   R0,(R2)       ; STORE AWAY CHARACTER
21 004524 005261 000000G     INC    LNSPAC(R1)    ; SAY 1 MORE CHAR IN TABLE
22 004530 120027 000000G     CMPB   R0,#CTRLC    ; IS CTRL-C A SPECIAL ACTIV CHAR?
23 004534 001003     BNE    1$          ; BRANCH IF NOT CTRL-C
24 004536 052761 000000G 000000G     BIS    #$UCTLC,LSW4(R1); REMEMBER SPECIAL CTRL-C
25          ;
26          ; If user typed ahead any of the characters that are being declared
27          ; to be an activation char, mark them as activation chars.
28          ;
29 004544 005003 1$:    CLR    R3          ; Count chars in R3
30 004546 010005     MOV    R0,R5          ; Carry new activation char in R5
31 004550 016102 000000G     MOV    LINCNT(R1),R2 ; Get pointer to next char in TT buffer
32 004554 020361 000000G 2$:    CMP    R3,LINCNT(R1) ; Any more chars to check?
33 004560 103044     BHIS   9$          ; Br if not
34 004562 010204     MOV    R2,R4          ; Save current character pointer
35 004564 004737 016320'     CALL  FETCHR          ; Fetch a char from TT input buffer
36 004570 005203     INC    R3            ; Count characters that we check
37 004572 120005     CMPB   R0,R5          ; Is this the activation char?
38 004574 001367     BNE    2$          ; Loop if not
39          ;
40          ; We found the activation char in the type-ahead characters.
41          ; Mark it as an activation character.
42          ;
43 004576 032700 000000G 4$:    BIT    #ACFLAG,R0    ; Is char already marked for activation?
44 004602 001013     BNE    3$          ; Br if yes
45 004604 052700 000000G     BIS    #ACFLAG,R0    ; Set activation-character flag
46 004610 010402     MOV    R4,R2          ; Get back pointer to character
47 004612 004737 016462'     CALL  INSCHR          ; Insert character in buffer
48 004616 005261 000000G     INC    LACTIV(R1)    ; Say another pending activation char
49 004622 005061 000000G     CLR    LAFSIZ(R1)    ; Clear field width activation
50 004626 005061 000000G     CLR    LFWLIM(R1)    ; Clear field width limit
51          ;
52          ; If the activation character is a carriage-return, check the following
53          ; character and delete it if it is a line-feed.
54          ;
55 004632 120027 000000G 3$:    CMPB   R0,#CR          ; Is activation character a carriage-return?
56 004636 001346     BNE    2$          ; Br if not
57 004640 020361 000000G     CMP    R3,LINCNT(R1) ; Any chars left in buffer?

```

```
58 004644 103012          BHIS  9#           ;Br if not
59 004646 010204          MOV   R2,R4        ;Save position of possible line feed
60 004650 004737 016320'  CALL  FETCHR       ;Fetch character that follows CR
61 004654 120027 000000G  CMPB  R0,#LF       ;Is that character a line feed?
62 004660 001335          BNE   2#           ;Br if not
63 004662 010402          MOV   R4,R2        ;Get back pointer to line feed
64 004664 004737 016612'  CALL  DELCHR       ;Delete the line feed
65 004670 000731          BR    2#
66                          ;
67                          ; Finished
68                          ;
69 004672 012605          9#:  MOV   (SP)+,R5
70 004674 012604          MOV   (SP)+,R4
71 004676 012603          MOV   (SP)+,R3
72 004700 000207          RETURN
```

LIFUN -- Process lead-in function sequences

```

1          ; -----
2          ; Command - E
3          ; Turn on character echoing.
4          ;
5 004702 052761 000000G 000000G CMDE:  BIS    ##ECHO,LSW2(R1) ;ENABLE ECHOING
6 004710 000207          RETURN
7          ;
8          ; -----
9          ; Command - F
10         ; Turn off character echoing.
11        ;
12 004712 042761 000000G 000000G CMDF:  BIC    ##ECHO,LSW2(R1) ;DISABLE ECHOING
13 004720 000207          RETURN
14        ;
15        ; -----
16        ; Command - G
17        ;
18 004722 000207          CMDG:  RETURN
19        ;
20        ; -----
21        ; Command - H
22        ; Disable virtual lines.
23        ;
24 004724 052761 000000G 000000G CMDH:  BIS    ##NOVLN,LSW2(R1);DISABLE VIRTUAL LINES
25 004732 000207          RETURN
26        ;
27        ; -----
28        ; Command - I
29        ; Enable lower case input.
30        ;
31 004734 052761 000000G 000000G CMDI:  BIS    ##LC,LSW2(R1) ;DO "SET TTY LC"
32 004742 106537 000000G          MFPD   @#JSWLOC ;GET USER'S JSW VALUE
33 004746 052716 000000G          BIS    #LCBIT,(SP) ;SET LOWER-CASE ENABLE FLAG
34 004752 011661 000000G          MOV    (SP),LJSW(R1) ;SAVE JSW IN INTERNAL TABLE
35 004756 106637 000000G          MTPD   @#JSWLOC ;STORE BACK INTO USER'S AREA
36 004762 000207          RETURN
37        ;
38        ; -----
39        ; Command - J
40        ; Disable lower case input.
41        ;
42 004764 042761 000000G 000000G CMDJ:  BIC    ##LC,LSW2(R1) ;DO "SET TTY NQLC"
43 004772 000207          RETURN
44        ;
45        ; -----
46        ; Command - K
47        ; Set deferred character echo mode.
48        ;
49 004774 052761 000000G 000000G CMDK:  BIS    ##DEFER,LSW2(R1);SET DEFERRED ECHO FLAG
50 005002 000207          RETURN
51        ;
52        ; -----
53        ; Command - L
54        ; Set immediate mode char echo.
55        ;
56 005004 042761 000000G 000000G CMDL:  BIC    ##DEFER,LSW2(R1) ;Echo immediately from now on
57 005012 042761 000000C 000000G      BIC    #<#DODFR!#GCECO>,LSW3(R1) ;Say we are not now deferring

```

```
58 005020 000207 RETURN
59
60 ; -----
61 ; Command - M
62 ; Set output transparency mode
63 ;
64 005022 052761 000000G 000000G CMDM: BIS    $#TRNSP,LSW3(R1);TURN ON TRANSPARENCY MODE
65 005030 000207 RETURN
66
67 ; -----
68 ; Command - N
69 ; Suspend input from a control file
70 ;
71 005032 013702 000000G CMDN:  MOV    CFPNT,R2      ; IS A COMMAND FILE ACTIVE NOW?
72 005036 001404      BEQ    1$          ; BR IF NOT
73 005040 010237 000000G      MOV    R2,CFSPND      ; SAVE POINTER
74 005044 004737 010016'      CALL   CFSTOP        ; Suspend command file input
75 005050 000207 1$:      RETURN
76
77 ; -----
78 ; Command - O
79 ; Restart input from suspended command file
80 ;
81 005052 013700 000000G CMDO:  MOV    CFSPND,R0    ; IS COMMAND FILE SUSPENDED?
82 005056 001404      BEQ    1$          ; BR IF NOT
83 005060 004737 010042'      CALL   CFSTRT        ; Restart command file input
84 005064 005037 000000G      CLR    CFSPND
85 005070 000207 1$:      RETURN
```



LIFUN -- Process lead-in function sequences

```

1          ; -----
2          ; Command - P
3          ; Reset user specified activation char.
4          ;
5 005072 012761 005102' 000000G CMDP:  MOV    #RSSPAC,LTSCMD(R1);SET PROCESSING ROUTINE
6 005100 000207          RETURN
7          ;
8          ; THIS CHAR IS THE CHAR TO RESET.
9          ;
10 005102 032761 000000G 000000G RSSPAC: BIT    #$DETCH,LSW(R1) ;Is this a detached job?
11 005110 001002          BNE    9$          ;Br if yes -- Ignore function
12 005112 004737 005120'          CALL   DELUAC          ;CALL SUBROUTINE TO DEL CHAR FROM TABLE
13 005116 000207          9$:   RETURN
14          ;
15          ; SUBROUTINE TO DELETE A USER-DEFINED ACTIVATION FROM THE TABLE
16          ; OF ACTIVATION CHARACTERS.
17          ; WHEN CALLED THE CHARACTER TO BE DELETED MUST BE IN R0.
18          ;
19 005120 010246          DELUAC: MOV    R2,-(SP)
20 005122 010346          MOV    R3,-(SP)
21 005124 016102 000000G          MOV    LN$PAC(R1),R2 ;GET # OF USER ACTIVATION CHARS
22 005130 001421          BEQ    3$          ;BR IF NONE TO RESET
23 005132 016103 000000G          MOV    L$PACT(R1),R3 ;POINT TO START OF TABLE
24 005136 120023          2$:   CMPB   R0,(R3)+ ;SEARCH FOR CHAR
25 005140 001402          BEQ    1$          ;BR WHEN FOUND
26 005142 077203          SOB   R2,2$          ;LOOP IF MORE TO CHECK
27          ; COULDN'T FIND CHAR
28 005144 000413          BR    3$
29          ; FOUND CHAR
30 005146 112363 177776          1$:   MOVB   (R3)+,-2(R3) ;MOVE BACK REST OF CHARS
31 005152 077203          SOB   R2,1$
32 005154 005361 000000G          DEC    LN$PAC(R1) ;SAY ONE LESS CHAR TO ACTIVATE ON
33 005160 120027 000000G          CMPB   R0,#CTRLC ;DE-ACTIVATING CTRL-C?
34 005164 001003          BNE    3$          ;BR IF NOT
35 005166 042761 000000G 000000G          BIC    #$UCTLC,LSW4(R1);REMEMBER NO MORE CTRL-C
36 005174 012603          3$:   MOV    (SP)+,R3
37 005176 012602          MOV    (SP)+,R2
38 005200 000207          RETURN

```

LIFUN -- Process lead-in function sequences

```

1          ; -----
2          ; Command - Q
3          ; Set field width as an activation condition
4          ;
5 005202 012761 005212' 000000G CMDQ:  MOV    #SFWAC,LTSCMD(R1);SET PROCESSING ROUTINE
6 005210 000207          RETURN
7          ;
8          ; Interpret this char as value of field width
9          ;
10 005212 010246 SFWAC:  MOV    R2,-(SP)
11 005214 010346          MOV    R3,-(SP)
12 005216 010446          MOV    R4,-(SP)
13 005220 032761 000000G 000000G BIT    ##DETCH,LSW(R1) ; Is this a detached job?
14 005226 001047          BNE    9$          ; Br if yes -- Ignore function
15 005230 010003          MOV    R0,R3          ; CARRY FIELD WIDTH IN R3
16 005232 004737 007760' CALL   CFTEST         ; INPUT COMING FROM COMMAND FILE?
17 005236 103043          BCC   9$          ; BR IF YES
18 005240 004737 017262' CALL   SLCHK          ; IS SINGLE LINE EDITOR IN CONTROL?
19 005244 103436          BCS   1$          ; BR IF YES -- SET VALUE AND LET SL HANDLE IT
20 005246 005761 000000G TST    LACTIV(R1)     ; IS ACTIVATION CHAR PENDING NOW?
21 005252 001035          BNE    9$          ; IF YES THEN IGNORE FIELD WIDTH
22          ;
23          ; No activation chars are pending and input is not
24          ; coming from a command file.
25          ; See if he has typed ahead enough to cause activation
26          ;
27 005254 005703          TST    R3          ; IS HE RESETTING FIELD WIDTH ACTIVATION?
28 005256 001431          BEQ   1$          ; BR IF YES
29 005260 020361 000000G CMP    R3,LINCNT(R1) ; REACHED ACTIVATION ALREADY?
30 005264 101026          BHI   1$          ; BR IF NOT
31          ;
32          ; User has typed ahead enough to fill this field.
33          ; Set activation flag in last char in field.
34          ;
35 005266 016102 000000G          MOV    LIMPNT(R1),R2 ; POINT TO 1ST CHAR PENDING
36 005272 010204 3$:    MOV    R2,R4          ; SAVE POINTER TO NEXT CHAR
37 005274 004737 016320' CALL   FETCHR         ; FETCH THE NEXT INPUT CHARACTER
38 005300 103422          BCS   9$          ; STRANGE -- NO MORE CHARS TO GET
39 005302 032700 000000G BIT    #ACFLAG,R0     ; IS THIS CHAR ALREADY FLAGGED FOR ACTIVATION?
40 005306 001017          BNE    9$          ; BR IF YES
41 005310 077310          SOB   R3,3$         ; LOOP TILL WE GET LAST CHAR IN FIELD
42 005312 052700 000000G BIS    #ACFLAG,R0     ; SET ACTIVATION-CHARACTER FLAG FOR CHAR
43 005316 010402          MOV    R4,R2          ; GET BACK POINTER TO CHAR POS IN BUFFER
44 005320 004737 016462' CALL   INSCHR         ; REINSERT CHARACTER INTO BUFFER
45 005324 005261 000000G INC    LACTIV(R1)     ; SAY ONE MORE PENDING ACTIVATION CHAR
46 005330 005061 000000G CLR    LAFSIZ(R1)    ; SAY FIELD WIDTH ACTIVATION NOT IN EFFECT
47 005334 005061 000000G CLR    LFWLIM(R1)    ; SAY NO FIELD WIDTH LIMIT
48 005340 000402          BR    9$
49          ;
50          ; User has not typed ahead entire field.
51          ; Remember field size as activation condition.
52          ;
53 005342 010061 000000G 1$:    MOV    R0,LAFSIZ(R1) ; REMEMBER FIELD SIZE
54          ;
55          ; Finished
56          ;
57 005346 012604 9$:    MOV    (SP)+,R4

```

58 005350 012603  
59 005352 012602  
60 005354 000207

MOV (SP)+,R3  
MOV (SP)+,R2  
RETURN

LIFUN -- Process lead-in function sequences

```

1          ; -----
2          ; Command - R
3          ; Turn on high-efficiency tty mode
4          ;
5 005356 032761 000000G 000000G CMDR:  BIT    ##DETCH,LSW(R1) ; Is this a detached job?
6 005364 001002          BNE    9$          ; Br if yes -- Ignore function
7 005366 004737 005374'    CALL   HION          ; TURN ON HIGH-EFFICIENCY MODE
8 005372 000207          9$:    RETURN
9          ;
10         ; HION IS CALLED TO ENABLE HIGH-EFFICIENCY TTY MODE.
11         ; IT IS CALLED FROM TSEMT AS WELL AS FROM TSEXEC.
12         ; WHEN CALLED, THE LINE INDEX # MUST BE IN R1.
13         ;
14 005374 032761 000000G 000000G HION:  BIT    ##DETCH,LSW(R1) ; IS THIS A DETACHED JOB?
15 005402 001003          BNE    1$          ; BR IF YES (CAN'T USE THIS MODE)
16 005404 052761 000000G 000000G    BIS    ##HITTY,LSW4(R1); ENABLE HIGH-EFFICIENCY MODE
17 005412 000207          1$:    RETURN
18         ;
19         ; -----
20         ; Command - S
21         ; Turn on single character activation mode
22         ;
23 005414 052761 000000G 000000G CMDS:  BIS    ##CHACT,LSW5(R1); TURN ON SINGLE CHARACTER ACTIVATION
24 005422 000207          RETURN
25         ;
26         ; -----
27         ; Command - T
28         ; Turn off single character activation mode
29         ;
30 005424 042761 000000G 000000G CMDT:  BIC    ##CHACT,LSW5(R1); TURN OFF SINGLE CHARACTER ACTIVATION
31 005432 000207          RETURN
32         ;
33         ; -----
34         ; Command - U
35         ; Enable non-wait TT input (.TTINR)
36         ;
37 005434 052761 000000G 000000G CMDU:  BIS    ##NOWTT,LSW5(R1); ENABLE NON-WAIT TT INPUT
38 005442 000207          RETURN

```

```

1          ;-----
2          ; Command - V
3          ; Set field width limit.
4          ;
5 005444 012761 005454' 000000G CMDV:  MOV    #SFWL,LTSCMD(R1);NEXT CHAR WILL BE FIELD WIDTH SIZE
6 005452 000207          RETURN
7          ;
8          ; This character specifies the field size
9          ;
10 005454 010346 SFWL:  MOV    R3,-(SP)
11 005456 010446     MOV    R4,-(SP)
12 005460 010546     MOV    R5,-(SP)
13 005462 032761 000000G 000000G BIT    ##DETCH,LSW(R1) ; Is this a detached job?
14 005470 001055     BNE    9$          ; Br if yes -- Ignore function
15 005472 005061 000000G     CLR    LFWLIM(R1) ; Initially clear field width
16 005476 010004     MOV    R0,R4          ; IS SPECIFIED FIELD WIDTH ZERO?
17 005500 001451     BEQ    9$          ; Br if yes
18 005502 004737 007760' 1$:  CALL   CFTEST      ; IS INPUT COMING FROM A COMMAND FILE?
19 005506 103046     BCC    9$          ; BR IF YES
20 005510 004737 017262'     CALL   SLCHK       ; IS SINGLE LINE EDITOR RUNNING?
21 005514 103441     BCS    6$          ; BR IF YES
22         ;
23         ; See if user typed ahead.
24         ; Check size of field and discard any chars beyond end of specified size.
25         ;
26 005516 005003 14$:  CLR    R3          ; COUNT TOTAL CHARACTERS IN R3
27 005520 016102 000000G     MOV    LINPNT(R1),R2 ; GET POINTER TO NEXT CHAR IN BUFFER
28 005524 020361 000000G 5$:  CMP    R3,LINCNT(R1) ; HAVE WE CHECKED ALL CHARS IN TT BUFFER?
29 005530 103033     BHIS   6$          ; BR IF YES -- NO OVERFLOW HAS OCCURRED YET
30 005532 010205     MOV    R2,R5          ; SAVE POINTER TO NEXT CHAR
31 005534 004737 016320'     CALL   FETCHR      ; GET NEXT CHAR FROM TT INPUT BUFFER
32 005540 120027 000000G     CMPB   R0,#CR       ; IS THIS CHAR CARRIAGE-RETURN?
33 005544 001427     BEQ    9$          ; BR IF YES -- THIS ENDS THE FIELD
34 005546 032700 000000G     BIT    #ACFLAG,R0  ; IS THIS CHAR AN ACTIVATION CHAR?
35 005552 001024     BNE    9$          ; BR IF YES -- FIELD TERMINATED WITHOUT OVRFLOW
36 005554 120027 000000G     CMPB   R0,#ESCFLG  ; IS THIS CHAR PART OF VT100 ESCAPE SEQ?
37 005560 001004     BNE    2$          ; BR IF NOT
38 005562 032761 000000G 000000G BIT    #VTEESC,LSW5(R1); ARE WE ACTIVATING ON VT100 ESC SEQUENCES?
39 005570 001015     BNE    9$          ; BR IF YES -- FIELD WIDTH OK
40 005572 005203 2$:  INC    R3          ; COUNT TOTAL # CHARS IN FIELD
41 005574 020304     CMP    R3,R4        ; DOES THIS CHAR OVERFLOW THE FIELD?
42 005576 101752     BLOS   5$          ; BR IF NOT
43         ;
44         ; Field overflow. Discard all chars typed beyond end of field.
45         ;
46 005600 010502     MOV    R5,R2          ; POINT TO CHAR THAT OVERFLOWED FIELD
47 005602 004737 016612' 10$: CALL   DELCHR      ; DELETE ALL CHARS THAT ARE BEYOND FIELD END
48 005606 103375     BCC    10$         ; LOOP IF MORE CHARS TO DELETE
49         ; Ring bell to signal user that field overflowed.
50 005610 012700 000000G     MOV    #BELL,R0     ; GET BELL CHARACTER
51 005614 004737 003462'     CALL   QUECHR      ; END TO TERMINAL
52         ;
53         ; Field did not overflow
54         ;
55 005620 010461 000000G 6$:  MOV    R4,LFWLIM(R1) ; SET FIELD WIDTH LIMIT FOR THIS FIELD
56         ;
57         ; Finished

```

```
58  
59 005624 012605 ;  
60 005626 012604 9#: MOV (SP)+, R5  
61 005630 012603 MOV (SP)+, R4  
62 005632 000207 MOV (SP)+, R3  
RETURN
```

LIFUN -- Process lead-in function sequences

```

1          ; -----
2          ; Command - W
3          ; Turn "paper-tape" mode on.
4          ;
5 005634 052761 000000G 000000G CMDW:  BIS    #$TAPE,LSW2(R1) ;TURN TAPE MODE ON
6 005642 000207          RETURN
7          ;
8          ; -----
9          ; Command - X
10         ; Turn "paper-tape" mode off.
11         ;
12 005644 042761 000000G 000000G CMDX:  BIC    #$TAPE,LSW2(R1) ;TURN TAPE MODE OFF
13 005652 000207          RETURN
14         ;
15         ; -----
16         ; Command - Y
17         ; Disable echoing of line-feed following carriage-return
18         ;
19 005654 052761 000000G 000000G CMDY:  BIS    #$NOLF,LSW6(R1) ;DISABLE AUTO LINE-FEED ECHO
20 005662 000207          RETURN
21         ;
22         ; -----
23         ; Command - Z
24         ; Enable echoing of line-feed following carriage-return
25         ;
26 005664 042761 000000G 000000G CMDZ:  BIC    #$NOLF,LSW6(R1) ;REENABLE AUTO LINE-FEED ECHO
27 005672 000207          RETURN

```

LIFUN -- Process lead-in function sequences

```

1 ;-----
2 ; PCSPND is called to suspend execution of the current job because
3 ; the terminal output buffer is full.
4 ;
5 ; Inputs:
6 ; R1 = Job index number.
7 ;
8 ; Outputs:
9 ; All registers are preserved.
10 ;
11 005674 010046 PCSPND: MOV RO, -(SP)
12 005676 004737 017474' CALL SIGWAT ; SIGNAL JOB WAIT
13 005702 012700 000000G MOV #S$OTWT, RO ; CHANGE JOB STATE TO WAITING-FOR-OUTPUT-SPACE
14 005706 004737 000000G CALL QHDSPN ; CHANGE STATE AND SUSPEND THE JOB
15 005712 004737 000000G CALL CHKABT ; SEE IF WE WERE ABORTED WHILE ASLEEP
16 005716 012600 MOV (SP)+, RO
17 005720 000207 RETURN
18 ;-----
19 ; TRYCHR is similar to PUTCCHR except if the user's
20 ; output buffer is full TRYCHR simply returns
21 ; and does not halt execution.
22 ; TRYCHR also does not give special treatment to
23 ; such chars as form feed, backspace, etc.
24 ; when called r0 must contain the character to be
25 ; sent and r1 must contain the virtual line index #.
26 ; all registers are preserved.
27 ;
28 005722 TRYCHR: DISABL ; ** DISABLE **
29 005730 005761 000000G TST LOTSPC(R1) ; IS THERE ROOM FOR THE CHAR?
30 005734 001402 BEQ 9$ ; BR IF NOT
31 005736 004737 003462' CALL QUECHR ; PUT CHARACTER INTO BUFFER
32 005742 000207 9$: RETURN

```



```
1 ;-----  
2 ; OTREGO IS CALLED TO RESTART A USER WHO IS SUSPENDED  
3 ; WAITING FOR SPACE IN THE OUTPUT BUFFER.  
4 ; WHEN CALLED THE USER NUMBER MUST BE IN R1.  
5 ;  
6 005744 012700 000000G OTREGO: MOV #S$OTFN,R0 ;PUT USER IN OUTPUT-NEEDED QUEUE  
7 005750 004737 000000G CALL ENQTL  
8 005754 000207 RETURN
```

```

1          .SBTTL
2          SBTTL  ** Program Level Input Character Processing **
3          .SBTTL  GETCHR -- Get next input char
4          ;-----
5          ; GETCHR -- ROUTINE TO MOVE THE NEXT CHARACTER FROM THE
6          ; INPUT BUFFER TO RO.
7          ; IF NO CHARACTERS ARE AVAILABLE THE USER IS SUSPENDED
8          ; UNTIL A RECORD IS RECEIVED.
9          ;
10         GETCHR: MOV      R1,-(SP)
11         MOV      R2,-(SP)
12         MOVB    CORUSR,R1          ;GET USER'S INDEX #
13         BIC     ##GCESC,LSW3(R1);WE'RE NOT PROCESSING VT50 ESC SEQUENCE
14         GCH1:  BIT     #<#DISCN+#CTRLC>,LSW(R1);DISCONNECT OR CTRL-C?
15         BEQ     B$                ;BRANCH IF NEITHER
16         CALL    STOP              ;STOP PROGRAM IF EITHER
17         ;
18         ; See if input is coming from a command file or from the terminal.
19         ;
20         B$:    CALL    GTCFCH      ;TRY TO GET A CHARACTER FROM A COMMAND FILE
21         BCS     15$              ;BR IF DID NOT GET ONE
22         JMP     GCCKES          ;PROCESS CHAR FROM COMMAND FILE
23         ;
24         ; We did not get a character from a command file.
25         ; If we are processing a logoff command file, finish the logoff now.
26         ;
27         15$:  BIT     ##LOFCF,LSW9(R1);Are we processing a logoff command file?
28         BEQ     13$              ;Br if not
29         BIS     ##DOOFF,LSW(R1);Force job logoff
30         CALL    STOP
31         ;
32         ; If this is a detached job, halt its execution if command file finished
33         ;
34         13$:  BIT     ##DETCH,LSW(R1);Is this a detached job?
35         BEQ     16$              ;Br if not
36         BIS     ##DISCN,LSW(R1);Force job logoff
37         CALL    STOP              ;Halt detached jobs that want terminal input
38         ;
39         ; See if we should get a character from the single line editor.
40         ;
41         16$:  BIT     #<#SPCTTY!DISSLE>,LJSW(R1);Special TTY mode or SL disabled?
42         BNE     11$              ;Br if yes -- don't use SL
43         BIT     ##SLON,LSW7(R1);Is SL enabled for this line?
44         BEQ     11$              ;Br if not
45         BIT     #<#ODTMD!$HITTY>,LSW4(R1);Are we in ODT or high efficiency?
46         BNE     11$              ;Br if yes
47         BIT     ##VTESC,LSW5(R1);VTxxx activation enabled?
48         BNE     11$              ;Br if yes
49         BIT     ##GTLIN,LSW4(R1);Is a .GTLIN being done?
50         BNE     12$              ;Br if yes
51         BIT     ##SLTTY,LSW7(R1);Is SL enabled for TTY input?
52         BEQ     11$              ;Br if not
53         12$:  DCALL   GTSLCH      ;Get character from single line editor
54         JMP     GCEXIT          ;Exit from GETCHR
55         ;
56         ; Get character from the terminal.
57         ;

```

GETCHR -- Get next input char

```

58 006160 032761 000000G 000000G 11$: BIT    ##NOINT,LSW7(R1); Does user want non-interactive execution?
59 006166 001006          BNE    14$          ;Br if yes
60 006170 013761 000000G 000000G      MOV    VINTID,LHIPCT(R1);Reset interactive I/O limit for job
61 006176 013761 000000G 000000G      MOV    VQUAN1,LITIME(R1);Reset interactive CPU time limit
62 006204 005761 000000G          14$: TST    LACTIV(R1)      ;Any activation chars pending?
63 006210 001003          BNE    9$          ;Branch if yes
64 006212 004737 011156'          CALL   ILWAIT      ;Wait for activation
65 006216 000666          BR     GCH1        ;Go try again
66 006220 016102 000000G          9$:  MOV    LINPNT(R1),R2  ;Get pointer to next char to get
67 006224 004737 016612'          CALL   DELCHR      ;Get char and delete from input buffer
68 006230 103661          BCS   GCH1        ;Br if no more characters are available
69 006232 005061 000000G          CLR    LRTCHR(R1)  ;Say no read time-out pending
70
71          ; See if we are in single-character-activation mode.
72
73 006236 032761 000000G 000000G      BIT    #SPCTTY,LJSW(R1); DOES PROGRAM WANT SINGLE-CHAR INPUT?
74 006244 001415          BEQ    6$          ;BR IF NOT
75 006246 032761 000000G 000000G      BIT    ##CHACT,LSW5(R1); IS SINGLE-CHARACTER INPUT ENABLED?
76 006254 001411          BEQ    6$          ;BR IF NOT
77 006256 004737 017422'          CALL   CVTLC      ;SET IF WE NEED TO CONVERT TO UPPER-CASE
78 006262 032700 000000G          BIT    #ACFLAG,R0  ; IS THIS AN ACTIVATION CHARACTER?
79 006266 001402          BEQ    10$         ;BR IF NOT
80 006270 000137 006526'          JMP    GCCKDS
81 006274 000137 006572'          10$: JMP    GCCKCC
82
83          ; See if we are in high-efficiency mode.
84
85 006300 032761 000000G 000000G 6$:  BIT    ##HITTY,LSW4(R1); ARE WE IN HIGH-EFFICIENCY MODE?
86 006306 001402          BEQ    3$          ;BR IF NOT
87
88          ; We are in high-efficiency mode
89
90 006310 000137 006732'          5$:  JMP    GCEND      ; DO EXPRESS CHARACTER PROCESSING
91
92          ; SEE IF THIS IS FLAG CHAR MEANING NEXT CHAR IS PART OF
93          ; VT50 ESCAPE SEQUENCE.
94
95 006314 032761 000000G 000000G 3$:  BIT    ##GCESC,LSW3(R1); IS THIS CHAR PART OF ESC SEQUENCE?
96 006322 001402          BEQ    1$          ;BR IF NOT
97 006324 000137 006512'          JMP    GCCKAC      ; TAKE CHAR WITHOUT FURTHER CHECKING
98 006330 120027 000000G          1$:  CMPB   R0,#ESCFLG  ; IS THIS FLAG FOR NEXT CHAR IN ESC SEQ?
99 006334 001010          BNE    GCCKES      ;BR IF NOT
100 006336 032761 000000G 000000G      BIT    ##VTEESC,LSW5(R1); ARE WE ACTIVATING ON ESCAPE SEQUENCES?
101 006344 001404          BEQ    GCCKES      ;BR IF NOT
102 006346 052761 000000G 000000G      BIS    ##GCESC,LSW3(R1); REMEMBER NEXT CHAR PART OF ESC SEQ
103 006354 000721          BR     9$          ; GO GET NEXT CHAR

```

GETCHR -- Get next input char

```

1
2 ; WE HAVE THE NEXT CHARACTER IN R0.
3 ; SEE IF WE NEED TO TRANSLATE LOWER CASE CHARS TO UPPER CASE.
4 ;
5 006356 004737 017422' GCCKES: CALL CVTLC ; CONVERT LC TO UC IF NEEDED
6 ;
7 ; SEE IF WE NEED TO DO ECHO SUPPRESSION.
8 ;
9 006362 032761 000000G 000000G BIT #SPCTTY,LJSW(R1); IS ECHO SUPPRESSION NEEDED?
10 006370 001450 BEQ GCCKAC ; BRANCH IF NOT
11 006372 032761 000000G 000000G BIT #CHACT,LSW5(R1); ARE WE IN SINGLE-CHAR-ACTIVATION MODE?
12 006400 001074 BNE GCCKCC ; BR IF YES
13 ;
14 ; BEGIN ECHO SUPPRESSION.
15 ;
16 006402 010346 MOV R3,-(SP)
17 006404 010002 MOV R0,R2 ; GET THE CHAR
18 006406 042702 177400 BIC #177400,R2 ; CLEAR ACTIVATION CHAR FLAG
19 ; SEE IF THIS IS A REGULAR OR CONTROL CHARACTER
20 006412 120227 000037 CMPB R2,#37 ; REGULAR OR CONTROL?
21 006416 101013 BHI 4$ ; BRANCH IF REGULAR CHAR
22 ; THIS IS A CONTROL CHAR -- CHECK FOR SPECIAL CASES.
23 006420 012703 000004 MOV #NESCTL,R3 ; # OF SPECIAL CONTROL CHARS
24 006424 120263 007006' 5$: CMPB R2,SESCTL(R3) ; IS THIS ONE?
25 006430 001402 BEQ 8$ ; BRANCH IF YES
26 006432 005303 DEC R3 ; TRY REST
27 006434 100373 BPL 5$
28 ; FALL THROUGH WITH R3=-1 FOR REGULAR CONTROL CHAR.
29 006436 006303 8$: ASL R3 ; GET WORD TABLE INDEX
30 006440 016303 007016' MOV SESRTN(R3),R3 ; GET ROUTINE ADDRESS
31 006444 000402 BR 6$
32 ; THIS IS NOT A CONTROL CHAR.
33 006446 012703 004056' 4$: MOV #ESUAC,R3 ; SET NORMAL HANDLER ROUTINE
34 006452 010361 000000G 6$: MOV R3,LESRTN(R1) ; SET HANDLER ROUTINE FOR PUTCHR
35 006456 120227 000141 CMPB R2,#141 ; IS CHAR A LOWER-CASE LETTER?
36 006462 103405 BLO 1$ ; BR IF NOT
37 006464 120227 000172 CMPB R2,#172
38 006470 101002 BHI 1$
39 006472 162702 000040 SUB #40,R2 ; CONVERT LOWER-CASE TO UPPER-CASE
40 006476 110261 000000G 1$: MOVB R2,LESCHR(R1) ; REMEMBER CHAR
41 006502 052761 000000G 000000G BIS #NDOUT,LSW3(R1); TURN ON ECHO SUPPRESSION
42 006510 012603 MOV (SP)+,R3

```

GETCHR -- Get next input char

```

1          ;
2          ; See if this is an activation character.
3          ;
4 006512 004737 007760' GCCKAC: CALL CFTEST ; INPUT FROM CONTROL FILE?
5 006516 103017          BCC GCCKCE ; BR IF YES
6 006520 032700 000000G BIT #ACFLAG,RO ; IS THIS AN ACTIVATION CHAR?
7 006524 001474          BEQ GCCKDE ; BRANCH IF NOT.
8          ;
9          ; This is an activation character.
10         ; See if we need to start doing deferred echoing.
11         ;
12 006526 032761 000000G 000000G GCCKDS: BIT #DODFR,LSW3(R1); IS ECHOING BEING DEFERRED?
13 006534 001416          BEQ GCCKCC ; BRANCH IF NOT
14 006536 032761 000000G 000000G BIT #GCECD,LSW3(R1); HAVE WE ALREADY STARTED?
15 006544 001010          BNE GCEAC ; BRANCH IF YES
16 006546 052761 000000G 000000G BIS #GCECD,LSW3(R1); BEGIN DEFERRED ECHOING WITH NEXT CHAR
17 006554 000406          BR GCCKCC
18         ;
19         ; See if want to list the contents of a control file.
20         ;
21 006556 032761 000000G 000000G GCCKCE: BIT #QUIET,LSW4(R1); LIST COMMAND FILE?
22 006564 001002          BNE GCCKCC ; BR IF NO LIST
23         ;
24         ; ECHO THE ACTIVATION CHAR NOW.
25         ;
26 006566 004737 007030' GCEAC: CALL GCECHO ; ECHO THE CHARACTER
27         ;
28         ; See if character we got is control-c.
29         ;
30 006572 120027 000000G GCCKCC: CMPB RO,#CTRLC ; IS CHAR CTRL-C?
31 006576 001055          BNE GCEND ; BRANCH IF NOT
32 006600 032761 000000G 000000G BIT #DBGMD,LSW6(R1); IS A DEBUGGING PROGRAM IN CONTROL?
33 006606 001016          BNE 7$ ; BR IF YES
34 006610 032761 000000G 000000G BIT #UCTLC,LSW4(R1); IS CTRL-C A USER DEF ACTIV CHAR?
35 006616 001045          BNE GCEND ; BRANCH IF IT IS
36 006620 005761 000000G TST LSCCA(R1) ; DID USER DO .SCCA?
37 006624 001042          BNE GCEND ; BR IF YES
38 006626 032761 000000G 000000G BIT #CCLRN,LSW5(R1); IS CCL TRANSLATOR RUNNING?
39 006634 001403          BEQ 7$ ; BR IF NOT
40 006636 004737 007760' CALL CFTEST ; IS INPUT COMING FROM A COMMAND FILE?
41 006642 103033          BCC GCEND ; BR IF YES
42 006644 032737 000000G 000000G 7$: BIT #LF$IN,LOGFLG ; Are we logging input characters?
43 006652 001417          BEQ 8$ ; BR IF NOT
44 006654 004737 007760' CALL CFTEST ; IS INPUT COMING FROM A CONTROL FILE?
45 006660 103404          BCS 6$ ; BR IF NOT
46 006662 032761 000000G 000000G BIT #QUIET,LSW4(R1); SHOULD WE DISPLAY CONTROL FILES?
47 006670 001010          BNE 8$ ; BR IF NOT
48 006672 032761 000000G 000000G 6$: BIT #ECHO,LSW2(R1) ; ARE WE ECHOING CHARACTERS?
49 006700 001404          BEQ 8$ ; DON'T LOG IF NOT ECHOING
50 006702 004737 010714' CALL LOGCHR ; LOG THE CONTROL-C
51 006706 004737 010764' CALL LOGCR ; LOG CR-LF
52 006712 004737 000000G B$: CALL STOP ; STOP PROGRAM EXECUTION

```

GETCHR -- Get next input char

```

1          ;
2          ;   THIS CHAR IS NOT AN ACTIVATION CHAR
3          ;
4          ;   SEE IF WE NEED TO DO DEFERRED ECHOING.
5          ;
6 006716 032761 000000G 000000G GCCKDE: BIT    #GCECO,LSW3(R1); HAVE WE STARTED DEFERRED ECHOING?
7 006724 001402          BEQ    GCEND          ; BRANCH IF NOT
8 006726 004737 007030'          CALL   GCECHO          ; ECHO THE CHARACTER
9          ;
10         ;   See if we should write character to log file
11        ;
12 006732 032737 000000G 000000G GCEND:  BIT    #LF$IN,LOGFLG  ; Are we logging input characters?
13 006740 001415          BEQ    GCEXIT          ; Br if not
14 006742 004737 007760'          CALL   CFTEST          ; Is input coming from a control file?
15 006746 103404          BCS    1$          ; Br if not
16 006750 032761 000000G 000000G          BIT    #GQUIET,LSW4(R1); Should we log control file characters?
17 006756 001006          BNE    GCEXIT          ; Br if not
18 006760 032761 000000G 000000G 1$:  BIT    #ECHO,LSW2(R1) ; Is echo suppression in effect?
19 006766 001402          BEQ    GCEXIT          ; Do not log if not echoing
20 006770 004737 010714'          CALL   LOGCHR          ; Log the character
21 006774 042700 177400          GCEXIT: BIC   #^C377,R0          ; Mask character to 8 bits
22 007000 012602          MOV    (SP)+,R2
23 007002 012601          MOV    (SP)+,R1
24 007004 000207          RETURN

```

GETCHR -- Get next input char

```

1          ;
2          ; TABLE OF CONTROL CHARACTERS WHICH REQUIRE SPECIAL
3          ; PROCESSING WITH REGARD TO ECHO SUPPRESSION.
4          ;
5          ; TABLE OF CONTROL CHARACTERS.
6          ;
7 007006   000G   SESCTL: . BYTE   FF           ; FORM FEED
8 007007   000G   . BYTE   ESC          ; ESCAPE
9 007010   000G   . BYTE   LF           ; LINE FEED
10 007011   000G   . BYTE   CR          ; CARRIAGE RETURN
11 007012   000G   . BYTE   TAB         ; TAB
12         000004 NESCTL = . -SESCTL-1   ; # OF SPECIAL CHARS.
13         . EVEN
14         ;
15         ; PARALLEL TABLE OF ADDRESSES OF ROUTINES TO RESTART
16         ; OUTPUT WHEN ECHO SUPPRESSION IS IN EFFECT FOR
17         ; SPECIAL CONTROL CHAR.
18         ; SESRTN TABLE MUST BE PARALLEL TO SESCTL TABLE.
19         ; NOTE: (-1) TABLE ENTRY IS USED FOR REGULAR CONTROL CHARS.
20         ;
21 007014   004110' . WORD   ESCTL          ; (-1) REGULAR CONTROL CHAR
22 007016   004154' SESRTN: . WORD   ESFF          ; FORM FEED
23 007020   004172' . WORD   EESCC          ; ESCAPE
24 007022   004136' . WORD   ESCRLF         ; LINE FEED
25 007024   004136' . WORD   ESCRLF         ; CARRIAGE RETURN
26 007026   004202' . WORD   ESHT          ; TAB

```

GETCHR -- Get next input char

```

1
2 ; -----
3 ; GCECHO IS CALLED TO ECHO A CHARACTER AS IT IS PASSED
4 ; TO THE USER IF WE ARE IN DEFERRED ECHO MODE.
5 ; WHEN CALLED, THE CHARACTER TO BE ECHOED MUST BE IN
6 ; R0 AND THE USER INDEX NUMBER MUST BE IN R1.
7 ; ALL REGISTERS ARE PRESERVED.
8 007030 004737 017222' GCECHO: CALL SCACHK ; ARE WE IN SINGLE CHARACTER INPUT MODE?
9 007034 103511 BCS 99$ ; BR IF YES -- DON'T ECHO CHARACTER
10 007036 032761 000000G 000000G BIT ##ECHO,LSW2(R1) ; IS CHAR ECHOING WANTED?
11 007044 001505 BEQ 99$ ; RETURN IF NOT
12 007046 032761 000000G 000000G BIT ##GCEESC,LSW3(R1); IS THIS CHAR PART OF ESC SEQUENCE?
13 007054 001101 BNE 99$ ; BR IF YES
14 007056 010046 MOV R0,-(SP)
15 007060 010246 MOV R2,-(SP)
16 007062 010346 MOV R3,-(SP)
17 007064 042700 000000G BIC #ACFLAG,R0 ; STRIP OFF ACTIVATION-CHAR FLAG
18
19 ; See if we should echo line-feed characters
20
21 007070 120027 000000G CMPB R0,#LF ; IS THIS CHAR LINE-FEED?
22 007074 001011 BNE 7$ ; BR IF NOT
23 007076 032761 000000G 000000G BIT ##NOLF,LSW6(R1) ; IS LINE-FEED ECHO SUPPRESSION IN EFFECT?
24 007104 001405 BEQ 7$ ; BR IF NOT
25 007106 032761 000000G 000000G BIT ##DBGMD,LSW6(R1); IS A DEBUGGER RUNNING NOW?
26 007114 001456 BEQ 2$ ; BR IF NOT (DON'T ECHO LF)
27 007116 000453 BR 5$ ; ECHO LF
28
29 ; SEE IF CHAR IS A USER DEFINED ACTIVATION CHAR.
30
31 007120 016102 000000G 7$: MOV LNSPAC(R1),R2 ; GET # OF USER DEF ACTIV CHARS
32 007124 001405 BEQ 1$ ; BRANCH IF NONE
33 007126 016103 000000G MOV LSPACT(R1),R3 ; POINT TO TABLE FOR USER
34 007132 120023 3$: CMPB R0,(R3)+ ; SEE IF THIS IS ONE
35 007134 001446 BEQ 2$ ; BRANCH IF IT IS
36 007136 077203 SOB R2,3$ ; LOOP IF MORE TO CHECK
37
38 007140 120027 000037 1$: CMPB R0,#37 ; REGULAR OR CONTROL CHAR?
39 007144 101040 BHI 5$ ; BRANCH IF REGULAR CHARACTER
40 007146 120027 000000G CMPB R0,#ESC ; IS CHAR ESCAPE?
41 007152 001003 BNE 4$ ; BRANCH IF NOT
42 007154 112700 000044 MOVB #'$,R0 ; OTHERWISE, ECHO $ FOR ESCAPE
43 007160 000432 BR 5$
44
45 007162 120027 000000G 4$: CMPB R0,#CTRLZ ; CTRL-Z?
46 007166 001406 BEQ 6$ ; BRANCH IF YES
47 007170 120027 000000G CMPB R0,#CTRLC ; CTRL-C?
48 007174 001403 BEQ 6$
49 007176 120027 000000G CMPB R0,#CTRLX ; CTRL-X?
50 007202 001021 BNE 5$ ; BRANCH IF NOT SPECIAL CONTROL CHAR
51
52 007204 110003 6$: ECHO ^-CHAR FOR SPECIAL CONTROL CHARS
53 007206 112700 000136 MOVB R0,R3 ; SAVE CONTROL CHAR
54 007212 004737 003056' MOVB #136,R0 ; ECHO ^
55 007216 110300 CALL PUTCH1
56 007220 052700 000100 MOVB R3,R0 ; GET CONTROL CHAR
57 007224 004737 003056' BIS #100,R0 ; CONVERT TO PRINTING CHAR
CALL PUTCH1 ; PRINT CHAR

```



GETCHR -- Get next input char

```

58 007230 112700 000000G          MOVB    #CR,R0          ; ECHO CR
59 007234 004737 003056'          CALL    PUTCH1
60 007240 112700 000000G          MOVB    #LF,R0
61 007244 000400                    BR      5#
62                                ;
63                                ; THIS IS A REGULAR CHARACTER.
64                                ;
65                                ; ECHO CHAR IN R0.
66 007246 004737 003056'          5#:    CALL    PUTCH1          ; ECHO THE CHARACTER
67                                ; RETURN
68 007252 012603                    2#:    MOV     (SP)+,R3
69 007254 012602                    MOV     (SP)+,R2
70 007256 012600                    MOV     (SP)+,R0
71 007260 000207                    99#:   RETURN

```

```

1          .SBTTL  GTCFCH -- Try to get char from command file
2          ;-----
3          ; GTCFCH is called to try to obtain a character from a command file.
4          ;
5          ; Inputs:
6          ;   R1 = Job index number.
7          ;
8          ; Outputs:
9          ;   C-flag cleared if a character is obtained from command file.
10         ;   RO = Character obtained.
11         ;
12         ; See if we are getting characters from a command file.
13         ;
14 007262 004737 007760' GTCFCH: CALL    CFTEST          ;SEE IF INPUT IS COMING FROM A COMMAND FILE
15 007266 103543          BCS      13$           ;BR IF NOT
16         ;
17         ; See if we are holding a character
18         ;
19 007270 113700 000000G          MOVB    CFHOLD,RO      ;ARE WE HOLDING A CHARACTER?
20 007274 001403          BEQ      10$           ;BR IF NOT
21 007276 105037 000000G          CLRB    CFHOLD          ;SAY CHARACTER IS GONE
22 007302 000532          BR       12$           ;TAKE SUCCESSFUL RETURN
23         ;
24         ; Input is coming from a command file.
25         ;
26 007304 004737 007600' 10$:   CALL    CFCHAR          ;GET CHAR FROM CONTROL FILE
27 007310 103531          BCS      14$           ;BR IF END OF COMMAND FILE HIT
28 007312 120027 000136          CMPB    RO,#'^        ;IS THIS A CONTROL CHAR?
29 007316 001124          BNE      12$           ;BRANCH IF NOT
30         ;
31         ; We found "^" character in command file.
32         ; This means we may have to treat the next character as a control char.
33         ;
34 007320 004737 007600'          CALL    CFCHAR          ;GET NEXT CHAR
35 007324 103523          BCS      14$           ;BR IF END OF COMMAND FILE HIT
36         ;
37         ; ^$ is interpreted as escape
38         ;
39 007326 120027 000044          CMPB    RO,#'$        ;TREAT ^$ AS ESCAPE CHAR
40 007332 001003          BNE      3$           ;
41 007334 012700 000000G          MOV     #ESC,RO      ;
42 007340 000513          BR       12$           ;
43         ;
44         ; ^^ is interpreted as a single ^
45         ;
46 007342 120027 000136 3$:   CMPB    RO,#'^        ;ANOTHER "^"?
47 007346 001510          BEQ      12$           ;IF YES THEN RETURN "^" AS CHARACTER
48         ;
49         ; ^ ( means stop listing control file
50         ;
51 007350 120027 000050          CMPB    RO,#'('      ;STOP LISTING COMMAND?
52 007354 001007          BNE      4$           ;BR IF NOT
53 007356 052761 000000G 000000G  BIS     #$QUIET,LSW4(R1);SET NO LISTING FLAG
54 007364 042761 000000G 000000G  BIC     #$CFSOT,LSW4(R1);ALLOW PROGRAM OUTPUT TO PRINT
55 007372 000744          BR       10$           ;GO GET NEXT CHAR FROM FILE
56         ;
57         ; ^ ) means start listing control file

```

```

58 ;
59 007374 120027 000051 4$: CMPB RO,#' ) ; START-LISTING COMMAND?
60 007400 001004 BNE 7$ ; BR IF NOT
61 007402 042761 000000C 000000G BIC #< $QUIET! $CFSOT>, LSW4(R1); START LISTING FILE
62 007410 000735 BR 10$ ; GO GET NEXT CHAR
63 ;
64 ; ^! means suppress all output -- command file and program
65 ;
66 007412 120027 000041 7$: CMPB RO,#' ! ; SUPPRESS OUTPUT?
67 007416 001004 BNE 5$ ; BR IF NOT
68 007420 052761 000000C 000000G BIS #< $CFSOT! $QUIET>, LSW4(R1); BEGIN SUPPRESSING OUTPUT
69 007426 000726 BR 10$
70 ;
71 ; ^> means accept all chars from @file
72 ;
73 007430 120027 000076 5$: CMPB RO,#' > ; ACCEPT ALL CHARS?
74 007434 001004 BNE 6$ ; BR IF NOT
75 007436 052761 000000G 000000G BIS # $CFALL, LSW4(R1); SET FLAG
76 007444 000717 BR 10$
77 ;
78 ; ^< means accept only .gtlin chars from @file
79 ;
80 007446 120027 000074 6$: CMPB RO,#' < ; ACCEPT ONLY .GTLIN CHARS?
81 007452 001004 BNE 2$ ; BR IF NOT
82 007454 042761 000000G 000000G BIC # $CFALL, LSW4(R1); RESET FLAG
83 007462 000677 BR GTCFCH
84 ;
85 ; ^digit is used to indicate a parameter substitution
86 ;
87 007464 120027 000061 2$: CMPB RO,#' 1 ; ^DIGIT MEANS SUBSTITUTE PARAM
88 007470 103414 BLD 1$ ; BR IF NOT PARAM
89 007472 120027 000060G CMPB RO,#< 60+MXCPRM> ; IN VALID RANGE?
90 007476 101011 BHI 1$ ; BR IF NOT
91 ;
92 ; Switch input to a parameter string
93 ;
94 007500 042700 177760 BIC #177760,RO ; LEAVE VALUE ONLY
95 007504 006300 ASL RO ; CONVERT TO WORD TABLE INDEX
96 007506 016037 177776G 000000G MOV <PRMPNT-2>(RO),CURPRM ; SET CHAR STRING POINTER
97 007514 004737 000000G CALL CHKABT ; ALLOW ABORTS WHILE GETTING PARAMETERS
98 007520 000671 BR 10$ ; GO GET 1ST CHAR FROM STRING
99 ;
100 ; ^letter causes the letter to be converted to a control character
101 ;
102 007522 120027 000101 1$: CMPB RO,#' A ; IS THIS A LETTER?
103 007526 103414 BLD 15$ ; BR IF NOT
104 007530 120027 000132 CMPB RO,#' Z
105 007534 101406 BLOS 16$ ; BR IF UPPER-CASE LETTER
106 007536 120027 000141 CMPB RO,#141 ; SEE IF THIS IS A LOWER-CASE LETTER
107 007542 103406 BLD 15$ ; BR IF NOT
108 007544 120027 000172 CMPB RO,#172
109 007550 101003 BHI 15$ ; BR IF NOT LETTER
110 007552 042700 177740 16$: BIC #177740,RO ; CONVERT ALPHA TO CONTROL CHAR
111 007556 000404 BR 12$ ; RETURN CONTROL CHARACTER AS CHARACTER GOTTEN
112 ;
113 ; ^ was followed by something we don't recognize.
114 ; Just return the ^ as the character gotten.

```

```
115 ;  
116 007560 110037 000000G 15#: MOV B RO,CFHOLD ; "PUSH" THE NEXT CHARACTER  
117 007564 112700 000136 MOV B #'^,RO ; RETURN "^" AS THIS CHARACTER  
118 ;  
119 ; Finished  
120 ;  
121 007570 000241 12#: CLC ; INDICATE SUCCESSFUL RETURN  
122 007572 000401 BR 13#  
123 007574 000261 14#: SEC ; INDICATE UNSUCCESSFUL RETURN  
124 007576 000207 13#: RETURN
```

```

1          .SBTTL  CFCHAR -- Do command file I/O
2          ;-----
3          ; CFCHAR is the lowest level routine called to get a character from
4          ; a command file.  It does the actual I/O to read the command file
5          ; and returns the next character from the buffer or from a parameter
6          ; string if string substitution is being done.
7          ;
8          ; Inputs:
9          ; R1 = Current job index number.
10         ;
11         ; Outputs:
12         ; R0 = Next character from command file.
13         ; C-flag set on return if end of file hit.
14         ;
15 007600 010246 CFCHAR: MOV      R2,-(SP)
16 007602 013702 000000G      MOV      CURPRM,R2      ;INPUT FROM PARAM STRING?
17 007606 001407      BEQ      5$          ;BR IF NOT
18         ;
19         ; INPUT IS COMING FROM A PARAMETER STRING
20         ;
21 007610 112200      MOVVB   (R2)+,R0      ;GET NEXT CHAR FROM STRING
22 007612 001403      BEQ      6$          ;BR IF HIT END OF STRING
23 007614 010237 000000G      MOV      R2,CURPRM      ;UPDATE CHAR POINTER
24 007620 000452      BR       9$          ;RETURN
25         ; HIT END OF PARAMETER STRING.
26         ; SWITCH INPUT BACK TO CONTROL FILE.
27 007622 005037 000000G 6$:   CLR      CURPRM      ;CLEAR PARAM STRING POINTER
28         ;
29         ; GET CHARACTER FROM CONTROL FILE
30         ;
31 007626 013702 000000G 5$:   MOV      CFPNT,R2      ;GET POINTER INTO BUFFER
32 007632 020227 000000G 4$:   CMP      R2,#CFEND      ;HIT END OF BUFFER?
33 007636 103437      BLD      1$          ;BRANCH IF NOT
34         ; REACHED END OF BUFFER -- READ IN NEXT BLOCK.
35 007640 113746 000052      MOVVB   @#52,-(SP)      ;SAVE I/O ERROR CODE
36 007644 005237 000000G      INC      CFBLK      ;INC FILE BLOCK NUMBER
37 007650      .READW  #CFARG,#CFCHAN,#CFBUF,#256.,CFBLK
38 007710 112637 000052      MOVVB   (SP)+,@#52      ;REPLACE ERROR CELL
39 007714 103006      BCC      3$          ;BR IF NOT AT END OF FILE
40         ;
41         ; End of file has been hit.
42         ; Try to pop up to higher level file.
43         ;
44 007716 004737 010070'      CALL     CFPOP      ;TRY TO POP UP TO HIGHER LEVEL FILE
45 007722 005737 000000G      TST      CFPNT      ;WAS THERE A HIGHER LEVEL FILE?
46 007726 001337      BNE      5$          ;BR IF YES
47 007730 000410      BR       11$
48         ; GET NEXT CHAR FROM BUFFER
49 007732 012702 000000G 3$:   MOV      #CFBUF,R2      ;RESET BUFFER POINTER
50 007736 112200 1$:   MOVVB   (R2)+,R0      ;GET NEXT CHAR
51 007740 001734      BEQ      4$          ;IGNORE NULLS
52 007742 010237 000000G 8$:   MOV      R2,CFPNT      ;SAVE NEW CHAR POINTER
53         ; Successful return.
54 007746 000241 9$:   CLC
55 007750 000401      BR       12$
56         ; Unsuccessful return.
57 007752 000261 11$:  SEC          ;SIGNAL UNSUCCESSFUL RETURN

```

58 007754 012602  
59 007756 000207

12#: MOV (SP)+, R2  
RETURN

```

1          .SBTTL  CFTEST -- Determine if TT input is from file
2          ;-----
3          ; CFTEST IS CALLED TO DETERMINE IF TT INPUT IS COMING FROM
4          ; A COMMAND FILE.
5          ; IF YES, THE C-FLAG IS RESET ON RETURN.
6          ; IF NO, THE C-FLAG IS SET ON RETURN.
7          ; WHEN CALLED, R1 MUST CONTAIN THE USER INDEX NUMBER.
8          ; ALL REGISTERS ARE PRESERVED.
9          ;
10         007760 005737 000000G CFTEST: TST      CFPNT      ; INPUT FROM COMMAND FILE?
11         007764 001412          BEQ      CFTNO      ; BR IF NOT
12         007766 032761 000000G 000000G CFTST1: BIT    ##GTLIN,LSW4(R1); IS THIS A .GTLIN INPUT EMT?
13         007774 001004          BNE      2$        ; BR IF YES
14         007776 032761 000000G 000000G BIT    ##CFALL,LSW4(R1); GET .TTYIN INPUT FROM @FILE?
15         010004 001402          BEQ      CFTNO      ; BR IF NOT
16         010006 000241          2$:    CLC          ; SAY INPUT COMING FROM FILE
17         010010 000207          RETURN
18         010012 000261          CFTNO: SEC          ; SAY INPUT NOT COMING FROM FILE
19         010014 000207          RETURN
  
```

CFSTOP -- Suspend command file input

```

1          .SBTTL  CFSTOP -- Suspend command file input
2          ;-----
3          ; Suspend command file input by setting CFPNT=0.
4          ; All registers are preserved.
5          ;
6 010016 010046 CFSTOP: MOV      RO,-(SP)
7          ;
8          ; Say input not coming from a command file
9          ;
10 010020 005037 0000000 CLR      CFPNT          ;Suspend command file input
11          ;
12          ; Clear command-file-active status flag in RMON cell
13          ;
14 010024 013700 0000000 MOV      CXTRMN,RO      ;Get virtual address of RMON in cxt blk
15 010030 042760 0000000 0000000 BIC      #CFACFL,R#CFST(RO) ;Say command file not active
16          ;
17          ; Finished
18          ;
19 010036 012600 MOV      (SP)+,RO
20 010040 000207 RETURN
21
22          .SBTTL  CFSTRT -- Restart command file input
23          ;-----
24          ; Restart command file input by storing a non-zero value into CFPNT.
25          ;
26          ; Inputs:
27          ; RO = Value to store into CFPNT
28          ;
29 010042 010046 CFSTRT: MOV      RO,-(SP)
30          ;
31          ; Restart command file input
32          ;
33 010044 010037 0000000 MOV      RO,CFPNT      ;Set command file buffer pointer
34 010050 001405 BEQ      9$          ;Br if not starting command file
35          ;
36          ; Set command-file-active flag in RMON status cell
37          ;
38 010052 013700 0000000 MOV      CXTRMN,RO      ;Get virtual address of RMON in cxt blk
39 010056 052760 0000000 0000000 BIS      #CFACFL,R#CFST(RO) ;Set command-file-active flags
40          ;
41          ; Finished
42          ;
43 010064 012600 9$: MOV      (SP)+,RO
44 010066 000207 RETURN

```



CFPOP -- Pop up to next command file

```

1          .SBTTL  CFPOP  -- Pop up to next command file
2          ;-----
3          ; CFPOP is called to close the current command file and pop up to
4          ; the next higher command file.
5          ;
6 010070 010146 CFPOP:  MOV     R1,-(SP)
7 010072 010346      MOV     R3,-(SP)
8 010074 010446      MOV     R4,-(SP)
9 010076 010546      MOV     R5,-(SP)
10 010100 113746 000052  MOVB   @#52,-(SP)      ;SAVE I/O ERROR CODE CELL
11 010104 113701 000000G  MOVB   CORUSR,R1      ;GET JOB INDEX NUMBER
12 010110 042761 000000G 000000G  BIC    #CFCCCL,LSW4(R1);SAY WE ARE NOT GETTING CHARS FROM CCL COMMAND
13 010116 005737 000000G  TST    CFPNT          ;IS A COMMAND FILE IN USE NOW?
14 010122 001002      BNE    11$           ;Br if yes
15 010124 000137 010676'  JMP    9$
16 010130 013705 000000G 11$:  MOV    CXTRMN,R5      ;GET ADDRESS OF SIMULATED RMON DATA
17 010134 113765 000000G 000000G  MOVB   CFIND,R$INST(R5);RESTORE IND STATUS BYTE
18 010142 105037 000000G  CLRB   CFHOLD        ;CLEAR ANY COMMAND FILE HOLDING CHAR
19          ;
20          ; Close currently open file
21          ;
22 010146 032761 000000G 000000G  BIT    #CFOPN,LSW4(R1);IS THE COMMAND FILE CHANNEL OPEN?
23 010154 001406      BEQ    1$           ;BR IF NOT
24 010156      .PURGE  #CFCHAN      ;CLOSE CURRENT FILE
25 010164 042761 000000G 000000G  BIC    #CFOPN,LSW4(R1);SAY COMMAND FILE CHANNEL IS NOW CLOSED
26          ;
27          ; See if there is a higher level command file to restore
28          ;
29 010172 105737 000000G 1$:  TSTB   CFNEST        ;ANY HIGHER LEVEL COMMAND FILES?
30 010176 001060      BNE    2$           ;BR IF YES
31          ;
32          ; There are no higher level command files
33          ;
34 010200 004737 010016'  CALL   CFSTOP        ;Say no data coming from command file
35 010204 042761 000000C 000000G  BIC    #<#CFALL!#CFSOT>,LSW4(R1);CLEAR COMMAND FILE CONTROL FLAGS
36 010212 032761 000000G 000000G  BIT    #LDFCF,LSW9(R1);Are we leaving a log off command file?
37 010220 001006      BNE    15$          ;If so, do not restore terminal input
38 010222 042761 000000G 000000G  BIC    #NDIN,LSW3(R1);ALLOW INPUT TO BE ACCEPTED FOR LINE
39 010230 042761 000000G 000000G  BIC    #SUCF,LSW9(R1);Say we are no longer executing startup file
40 010236 005000 15$:  CLR    RO           ;Init privilege vector index
41 010240 016060 000000G 000000G 10$:  MOV    PRIVSO(RO),PRIVFO(RO);Reset command file priv to set priv
42 010246 032761 000000G 000000G  BIT    #INKMN,LSW4(R1);Are we in TSKMON now?
43 010254 001403      BEQ    12$          ;Br if not
44 010256 016060 000000G 000000G  MOV    PRIVSO(RO),PRIVCO(RO);Reset current privileges
45 010264 062700 000002 12$:  ADD    #2,RO         ;Increment word index
46 010270 020027 000000C  CMP    RO,#PVNPW*2    ;Done all?
47 010274 103761      BLD    10$          ;Loop if more
48 010276 005037 000000G  CLR    AFCF         ;Clear command file attribute flags
49 010302 032761 000000G 000000G  BIT    #INKMN,LSW4(R1);Are we in TSKMON now?
50 010310 001572      BEQ    9$           ;Br if not
51 010312 042761 000000G 000000G  BIC    #SCCA,LSW5(R1);Clear abort-suppression flag
52 010320 042761 000000G 000000G  BIC    #NDWIN,LSW11(R1);Clear window suppression
53 010326 105737 000000G  TSTB   SUCF2        ;Is there a pending secondary file?
54 010332 001561      BEQ    9$           ;Br if not
55 010334 004737 000000G  CALL   STOP         ;Reenter KMON to start secondary file
56          ;
57          ; Reopen next higher level file

```

```

58 ; Restore parameter pointers
59 ;
60 010340 013705 000000G 2$: MOV CFSP,R5 ;GET COMMAND FILE STACK POINTER
61 010344 012703 000000G MOV #PRMPNT,R3 ;POINT TO PARAM POINTER CELLS
62 010350 012504 4$: MOV (R5)+,R4 ;GET A PARAMETER POINTER
63 010352 001402 BEQ 3$ ;BR IF END OF LIST HIT
64 010354 010423 MOV R4,(R3)+ ;RESTORE POINTER
65 010356 000774 BR 4$ ;GO BACK AND DO NEXT ONE
66 010360 020327 000000G 3$: CMP R3,#LSTPRM ;ZERO ALL OTHER PARAMETER POINTERS
67 010364 103002 BHIS 5$
68 010366 005023 CLR (R3)+
69 010370 000773 BR 3$
70 ;
71 ; Restore parameter strings
72 ;
73 010372 012504 5$: MOV (R5)+,R4 ;GET ADDRESS OF END OF STRING
74 010374 010437 000000G MOV R4,PBFEND
75 010400 020427 000000G 7$: CMP R4,#PRMBUF ;RESTORED ALL OF STRING?
76 010404 101402 BLOS 6$ ;BR IF YES
77 010406 012544 MOV (R5)+,-(R4) ;POP STRING OFF OF STACK
78 010410 000773 BR 7$
79 010412 012537 000000G 6$: MOV (R5)+,CURPRM ;POP POINTER INTO STRING
80 ;
81 ; Restore IND status flags
82 ;
83 010416 012537 000000G MOV (R5)+,CFIND ;RESTORE IND STATUS FLAGS
84 ;
85 ; Reset command file control flags
86 ;
87 010422 012704 000000G MOV #CFLFL4,R4 ;GET CONTROL FLAG MASK
88 010426 040461 000000G BIC R4,LSW4(R1) ;CLEAR THOSE FLAGS
89 010432 005104 COM R4 ;MASK ALL BUT THOSE FLAGS
90 010434 040415 BIC R4,(R5)
91 010436 052561 000000G BIS (R5)+,LSW4(R1) ;SET DESIRED FLAGS
92 ;
93 ; Restore command file attribute flags
94 ;
95 010442 012537 000000G MOV (R5)+,AFCF ;Restore command file attribute flags
96 010446 052761 000000G 000000G BIS ##SCCA,LSW5(R1) ;Assume ctrl-C abort suppression wanted
97 010454 032737 000000G 000000G BIT #AF$CCA,AFCF ;Is abort suppression wanted?
98 010462 001003 BNE 13$ ;Br if yes
99 010464 042761 000000G 000000G BIC ##SCCA,LSW5(R1) ;Clear abort-suppression flag
100 010472 032737 000000G 000000G 13$: BIT #AF$NPW,AFCF ;Suppressing process windowing?
101 010500 001003 BNE 14$ ;Br if yes
102 010502 042761 000000G 000000G BIC ##NOWIN,LSW11(R1);Clear window suspend flag
103 ;
104 ; Restore command file privileges
105 ;
106 010510 005000 14$: CLR R0 ;Init vector index
107 010512 011560 000000G 8$: MOV (R5),PRIVFO(R0) ;Restore each privilege word
108 010516 012560 000000G MOV (R5)+,PRIVCO(R0);Reset current privilege too
109 010522 062700 000002 ADD #2,R0 ;Increment index
110 010526 020027 000000C CMP R0,#PVNPW*2 ;Done all?
111 010532 103767 BLD 8$ ;Loop if more
112 ;
113 ; Restore buffer pointer information
114 ;

```

CFPOP -- Pop up to next command file

```

115 010534 012500          MOV      (R5)+,R0          ; POINTER INTO BUFFER
116 010536 004737 010042'  CALL     CFSTRT           ; Reset command file pointer
117 010542 012537 000000G   MOV      (R5)+,CFBLK      ; CURRENT BLOCK NUMBER
118                               ;
119                               ; Reopen the command file
120                               ;
121 010546                . REOPEN #CFARG,#CFCHAN,R5 ; REOPEN COMMAND FILE
122 010564 062705 000012    ADD      #10.,R5          ; POP SAVE STATUS INFO
123 010570 052761 000000G 000000G  BIS      #CFOPN,LSW4(R1); SAY COMMAND FILE CHANNEL IS OPEN
124                               ;
125                               ; Reread current buffer from file
126                               ;
127 010576                . READW #CFARG,#CFCHAN,#CFBUF,#256.,CFBLK
128                               ;
129                               ; Finished restoring file
130                               ;
131 010636 010537 000000G   MOV      R5,CFSP         ; SAVE UPDATED STACK POINTER
132 010642 105337 000000G   DECB    CFNEST           ; SAY ONE LESS FILE ON STACK
133                               ;
134                               ; If we just reached the end of a command file created to hold
135                               ; an expanded IND command and KMON is trying to read another
136                               ; command, call STOP to force KMON to go back to IND to get the
137                               ; next command before continuing on with an outer level
138                               ; command file.
139                               ;
140 010646 032761 000000G 000000G  BIT      #IN$KMN,LSW4(R1); IS KMON RUNNING?
141 010654 001410          BEQ      9$              ; BR IF NOT
142 010656 013705 000000G   MOV      CXTRMN,R5       ; GET ADDRESS OF SIMULATED RMON DATA
143 010662 132765 000000G 000000G  BITB    #IN$ACT,R$INST(R5); IS IND ACTIVE?
144 010670 001402          BEQ      9$              ; BR IF NOT
145 010672 004737 000000G   CALL     STOP            ; REENTER KMON TO FORCE REENTRY OF IND
146 010676 112637 000052    9$:  MOVB   (SP)+,@#52      ; RESTORE I/O ERROR CELL
147 010702 012605          MOV      (SP)+,R5
148 010704 012604          MOV      (SP)+,R4
149 010706 012603          MOV      (SP)+,R3
150 010710 012601          MOV      (SP)+,R1
151 010712 000207          RETURN

```

LOGCHR -- Write character to log file

```

1          .SBTTL LOGCHR -- Write character to log file
2          ;-----
3          ; LOGCHR is called to write a character to the log file.
4          ; Control characters are converted into ^character sequences.
5          ;
6          ; Inputs:
7          ;   RO = Character to be written to log.
8          ;
9 010714   010046 LOGCHR: MOV     RO,-(SP)
10         ;
11         ; See if this is a control character
12         ;
13 010716   042700 177400         BIC     #^C377,RO      ;Strip character down to 8 bits
14 010722   120027 000003         CMPB   RO,#3          ;Is character control-C?
15 010726   001403                 BEQ    1$             ;Br if yes
16 010730   120027 000032         CMPB   RO,#32         ;Is character control-Z?
17 010734   001007                 BNE    2$             ;Br if not
18 010736   112700 000136 1$:    MOVB   #'^,RO        ;Log "^"
19 010742   004737 011012'        CALL   LOGCH1
20 010746   011600                 MOV    (SP),RO       ;Get back original character
21 010750   062700 000100                 ADD    #100,RO       ;Convert to printing character
22 010754   004737 011012' 2$:    CALL   LOGCH1        ;Log the character
23         ;
24         ; Finished
25         ;
26 010760   012600                 MOV    (SP)+,RO
27 010762   000207                 RETURN
28         ;-----
29         ; LOGCR is called to send a carriage-return and line-feed sequence
30         ; to the log file.
31         ;
32         ;
33 010764   010046 LOGCR:  MOV     RO,-(SP)
34 010766   112700 000000G        MOVB   #CR,RO        ;Log carriage-return
35 010772   004737 011012'        CALL   LOGCH1
36 010776   112700 000000G        MOVB   #LF,RO        ;Log line-feed
37 011002   004737 011012'        CALL   LOGCH1
38 011006   012600                 MOV    (SP)+,RO
39 011010   000207                 RETURN

```

LOGCHR -- Write character to log file

```

1 ;-----
2 ; LOGCHR is called to move a character to the log buffer.
3 ; No tests or conversions are performed on the character.
4 ; No logging is done if echo suppression is in effect.
5 ;
6 ; Inputs:
7 ; RO = Character to be written.
8 ;
9 011012 010246 LOGCHR: MOV R2,-(SP)
10 ;
11 ; Check to see if log file output has been suspended (NOWRITE option)
12 ;
13 011014 032737 0000000 0000000 BIT #LF#WRT,LOGFLG ;Has log file been suspended?
14 011022 001453 BEQ 9$ ;Br if yes
15 ;
16 ; Get current buffer pointer and make sure buffer is not full
17 ;
18 011024 013702 0000000 MOV LOGPTR,R2 ;Get current log file buffer pointer
19 011030 001450 BEQ 9$ ;Br if not doing logging
20 011032 020227 0000000 CMP R2,#LOGEND ;Is buffer full?
21 011036 103442 BLO 1$ ;Br if not
22 ;
23 ; Log file buffer is full. Write it to the log file.
24 ;
25 011040 012702 0000000 MOV #LOGBUF,R2 ;Point to front of buffer
26 011044 010046 MOV RO,-(SP)
27 011046 113746 000052 MOVB @#52,-(SP) ;Save job's error cell
28 011052 . WRITW #CFARG,#LOGCHN,R2,#256.,LOGBLK ;Write block to log file
29 011110 103010 BCC 2$ ;Br if no write error
30 011112 012705 177753 MOV #-25,R5 ;Abort job if error writing to log file
31 011116 013704 0000000 MOV EMTADR,R4 ;Get address of EMT
32 011122 005037 0000000 CLR LOGPTR ;Say we have stopped using log file
33 011126 000137 0000000 JMP ABORT ;Abort the job
34 011132 005237 0000000 2$: INC LOGBLK ;Advance log file block number
35 011136 112637 000052 MOVB (SP)+,@#52 ;Restore error cell
36 011142 012600 MOV (SP)+,RO
37 ;
38 ; Move character to log buffer
39 ;
40 011144 110022 1$: MOVB RO,(R2)+ ;Move char to log file buffer
41 011146 010237 0000000 MOV R2,LOGPTR ;Save new buffer pointer
42 ;
43 ; Finished
44 ;
45 011152 012602 9$: MOV (SP)+,R2
46 011154 000207 RETURN

```

```

1          .SBTTL  ILWAIT -- Wait for activation char from terminal
2          ;-----
3          ; ILWAIT WAITS UNTIL AN ACTIVATION CHARACTER IS RECEIVED
4          ; FOR CURRENT USER.  ALL REGISTERS ARE PRESERVED.
5          ; WHEN CALLED R1 MUST CONTAIN THE USER INDEX #.
6          ;
7 011156  032761  000000G 000000G ILWAIT: BIT    #$DETCH,LSW(R1) ; IS THIS A DETACHED JOB?
8 011164  001405                    BEQ     15$      ; BR IF NOT
9 011166  052761  000000G 000000G        BIS    #$DISCN,LSW(R1) ; FORCE JOB LOGOFF
10 011174  004737  000000G          CALL   STOP      ; HALT DETACHED JOBS THAT WANT TERMINAL INPUT
11          ;
12          ; If we previously stopped input from silo buffer, Restart it now
13          ; if we are about to run out of characters.
14          ;
15 011200  032761  000000G 000000G 15$:  BIT    #$XSTOP,LSW6(R1); DID WE SUSPEND TRANSMISSION TO US?
16 011206  001410                    BEQ     11$      ; BR IF NOT
17 011210  042761  000000G 000000G        BIC    #$XSTOP,LSW6(R1); RESTART INPUT
18 011216  052761  000000G 000000G        BIS    #$NDICP,LSW10(R1); Say line needs input character servicing
19 011224  005237  000000G          INC    NEDCDI      ; Say input processing needed
20          ;
21          ; If we are in deferred echo mode, echo any pending
22          ; Characters on last line.
23          ;
24 011230  004737  011346'          11$:  CALL   DFRREL      ; Release deferred echo mode
25          ;
26          ; Suspend user till activation character received.
27          ;
28 011234  004737  017474'          CALL   SIGWAT     ; SIGNAL VIRTUAL LINE WAIT CONDITION
29 011240  042761  000000G 000000G        BIC    #$NDIN,LSW3(R1) ; ALLOW INPUT TO BE ACCEPTED FOR LINE
30 011246  042761  000000G 000000G        BIC    #$SUCF,LSW9(R1) ; Say we are no longer executing startup file
31 011254  004737  000000G          13$:  CALL   CHKABT     ; SEE IF JOB HAS BEEN ABORTED
32 011260          DISABL          ; ** DISABLE **
33 011266  005761  000000G          TST    LACTIV(R1)   ; GOT ANY ACTIVATION CHARS YET?
34 011272  001021          BNE    1$        ; BR IF YES
35 011274  032761  000000G 000000G        BIT    #$NDINT,LSW7(R1); Does user want non-interactive execution?
36 011302  001006          BNE    5$        ; Br if yes
37 011304  013761  000000G 000000G        MOV    VINTIO,LHIPCT(R1); RESET INTERACTIVE I/O LIMIT FOR JOB
38 011312  013761  000000G 000000G        MOV    VQUAN1,LITIME(R1); RESET INTERACTIVE CPU TIME LIMIT
39 011320  010046          5$:  MOV    RO,-(SP)
40 011322  012700          MOV    #$INWT,RO   ; GET WAITING-FOR-INPUT STATE
41 011326  004737  000000G          CALL   QHDSPN     ; SUSPEND JOB AND WAIT FOR ACTIVATION *ENABLE*
42 011332  012600          MOV    (SP)+,RO
43 011334  000747          BR     13$      ; NOW CHECK AGAIN
44 011336          1$:  ENABL          ; ** ENABLE **
45 011344  000207          RETURN

```

DFRREL -- Release deferred echo mode

```

1          .SBTTL  DFRREL -- Release deferred echo mode
2          ;-----
3          ; DFRREL is called to release deferred echo mode and to echo any
4          ; pending deferred characters.
5          ;
6          ; Inputs:
7          ; R1 = Job index number
8          ;
9 011346 010246 DFRREL: MOV     R2,-(SP)
10 011350 010346      MOV     R3,-(SP)
11 011352 010446      MOV     R4,-(SP)
12          ;
13          ; See if we are currently doing deferred echoing
14          ;
15 011354          DISABL          ;** DISABLE INTERRUPTS **
16 011362 032761 000000G 000000G BIT     ##DODFR,LSW3(R1); ARE WE IN DEFERRED ECHO MODE?
17 011370 001460      BEQ     6$          ; BRANCH IF NOT
18 011372 005761 000000G      TST     LACTIV(R1)      ; ARE THERE ANY PENDING ACTIVATION CHARS?
19 011376 001055      BNE     6$          ; BR IF YES -- DON'T ECHO IF PENDING ACTIV
20 011400          ENABL          ;** ENABLE INTERRUPTS **
21          ;
22          ; We are in deferred echo mode
23          ;
24 011406 042761 000000G 000000G BIC     ##GCECO,LSW3(R1); RESET GETCHR ECHOING
25 011414 052761 000000G 000000G BIS     ##1STCH,LSW3(R1); SAY WE'VE GOT 1ST CHAR
26 011422 116161 000000G 000000G MOVB   LCOL(R1),LINCUR(R1); SAVE CURSOR POSITION
27 011430 005004      CLR     R4          ; R4 IS FLAG FOR ESC SEQ CHARS
28 011432 005003      CLR     R3
29 011434 016102 000000G      MOV     LIMPNT(R1),R2 ; POINT TO START OF PENDING CHARS
30          ;
31          ; Begin loop to echo all pending characters
32          ;
33 011440 020361 000000G 5$:  CMP     R3,LINCNT(R1) ; ECHOED ALL PENDING?
34 011444 103027      BHIS   4$          ; BRANCH IF FINISHED
35 011446 004737 016320'  CALL   FETCHR      ; GET NEXT CHAR FROM TT INPUT BUFFER
36 011452 005704      TST     R4          ; IS THIS CHAR PART OF ESC SEQ?
37 011454 001015      BNE     8$          ; BR IF YES
38 011456 120027 000000G      CMPB   RO,#ESCFLG   ; FLAG THAT NEXT CHAR PART OF ESC SEQUENCE?
39 011462 001006      BNE     9$          ; BR IF NOT
40 011464 032761 000000G 000000G BIT     ##VTEESC,LSW5(R1); Are we activating on escape sequences?
41 011472 001402      BEQ     9$          ; Br if not -- Treat ESCFLG like normal char
42 011474 005204      INC     R4          ; SET ESC SEQ FLAG
43 011476 000410      BR     10$
44 011500 004737 017422'  9$:  CALL   CVTLC      ; CONVERT LOWER CASE TO UPPER CASE
45 011504 004737 007030'  CALL   GCECHO     ; ECHO IT
46 011510 005004      8$:  CLR     R4          ; CLEAR ESC SEQ FLAG
47 011512 032700 000000G      BIT     #ACFLAG,R0 ; IS THIS AN ACTIVATION CHAR?
48 011516 001005      BNE     6$          ; BRANCH IF YES
49 011520 005203 10$:  INC     R3          ; COUNT CHARS ECHOED
50 011522 000746      BR     5$
51          ;
52          ; Release deferred echo mode
53          ;
54 011524 042761 000000G 000000G 4$:  BIC     ##DODFR,LSW3(R1); BEGIN IMMEDIATE CHAR ECHOING
55          ;
56          ; Finished
57          ;

```

```
58 011532
59 011540 012604
60 011542 012603
61 011544 012602
62 011546 000207
```

6#: ENABL ;\*\* ENABLE INTERRUPTS \*\*  
MOV (SP)+,R4  
MOV (SP)+,R3  
MOV (SP)+,R2  
RETURN



```

1          .SBTTL
2          .SBTTL  ** Fork Level Input Character Processing **
3          .SBTTL  TTINCP -- Process received input characters
4          ;-----
5          ; TTINCP is called at fork level after each received character
6          ; has been stored in the high speed input ring buffer.
7          ; The function of TTINCP is to remove characters from the
8          ; high speed input ring buffer and perform the TSX-Plus
9          ; character processing which will eventually cause the character
10         ; to be stored in the input ring buffer for the line.
11         ;
12         ; Inputs:
13         ;   R4 = Line index number
14         ;
15 011550 010146 TTINCP: MOV     R1,-(SP)
16 011552 010546         MOV     R5,-(SP)
17 011554 010401         MOV     R4,R1           ;Carry line index number in R1 too
18         ;
19         ; See if TT input ring buffer is full
20         ;
21 011556 032764 000000G 000000G 5$: BIT     ##XSTOP,LSW6(R4);Has input been suspended due to buffer full
22 011564 001007         BNE     9$           ;Br if yes
23         ;
24         ; Get next character from input silo
25         ;
26 011566 004777 000000G         CALL    @SILFET           ;Get next character from input silo
27 011572 103404         BCS     9$           ;Br if no characters in silo
28 011574 010005         MOV     R0,R5           ;Put character in R5 for GOTCHR
29         ;
30         ; Now call main routine to process a character for this line
31         ;
32 011576 004737 011612'         CALL    GOTCHR           ;Process a character for this line
33         ;
34         ; Finished processing a character.
35         ; Go back and see if there are any more characters to process.
36         ;
37 011602 000765         BR      5$           ;Check for more characters to process
38         ;
39         ; Finished processing all pending characters for this line.
40         ;
41         ;
42 011604 012605 9$: MOV     (SP)+,R5
43 011606 012601         MOV     (SP)+,R1
44 011610 000207         RETURN

```

TTINCP -- Process received input characters

```

1 ;-----
2 ; A character has been received from a line.
3 ; Check to see if the line is active or needs to be started.
4 ;
5 ; Inputs:
6 ; R4 = Physical line index number.
7 ; R5 = Received character.
8 ;
9 011612 010146 GOTCHR: MOV R1, -(SP)
10 ;
11 ; Ignore the character if system initialization is not yet complete
12 ;
13 011614 105737 000000G TSTB INITFLG ;Is system initialization finished yet?
14 011620 001052 BNE 9$ ;Br if not
15 ;
16 ; See if this line has been initialized yet
17 ;
18 011622 116401 000000G MOVB LNMAP(R4),R1 ;Get virtual line index number
19 011626 032761 000000G 000000G BIT #$DILUP,LSW(R1) ;Has line been started yet?
20 011634 001033 BNE 1$ ;Br if yes
21 ;
22 ; Line has not been initialized yet.
23 ; See if we should start it now.
24 ;
25 011636 105737 000000G TSTB STPFLG ;Is a system shutdown in progress?
26 011642 001041 BNE 9$ ;Br if yes -- Don't start any lines
27 011644 032764 000000G 000000G BIT #$DEAD,LSW3(R4) ;Is this line marked as dead?
28 011652 001035 BNE 9$ ;Br if yes
29 ;
30 ; See if we should do autobaud speed selection for this line
31 ;
32 011654 032764 000000G 000000G BIT #$AUTO,ILSW2(R4) ;Is autobaud speed selection wanted?
33 011662 001405 BEQ 3$ ;Br if not
34 011664 DCALL AUTSPD ;Do autobaud speed selection
35 011672 103007 BCC 2$ ;Br if we should start the line now
36 011674 000424 BR 9$ ;Do not start the line yet
37 011676 120527 000000G 3$: CMPB R5,#CR ;Is character Carriage-return?
38 011702 001403 BEQ 2$ ;Br if yes
39 011704 120527 000000G CMPB R5,#CTRLC ;Is character ctrl-C?
40 011710 001016 BNE 9$ ;Br if not
41 ;
42 ; Start up a previously inactive line
43 ;
44 011712 005000 2$: CLR R0 ;No secondary start-up command file
45 011714 DCALL INITLN ;Initialize the line
46 011722 000403 BR 5$
47 ;
48 ; See if we are ignoring trash characters that may be received after
49 ; the autobaud start-up character.
50 ;
51 011724 005764 000000G 1$: TST LABTIM(R4) ;Is autobaud masking trash characters?
52 011730 001404 BEQ 4$ ;Br if not
53 011732 042761 000000G 000000G 5$: BIC #$RBRK,LSW10(R1) ;Clear break-received flag
54 011740 000402 BR 9$ ;Wait for autobaud mask time to end
55 ;
56 ; Process an input character for an active line
57 ;

```

```
58 011742 004737 011752'      4$:    CALL    PRCHAR      ;Process the character
59                               ;
60                               ; Finished
61                               ;
62 011746 012601      9$:    MOV     (SP)+, R1
63 011750 000207      RETURN
```

TTINCP -- Process received input characters

```

1          ; -----
2          ; Process a character received by an active line.
3          ;
4          ; Inputs:
5          ; R1 = Job's virtual line index number.
6          ; R5 = Received character.
7          ;
8 011752 010246 PRCHAR: MOV      R2, -(SP)
9 011754 010346      MOV      R3, -(SP)
10 011756 010446      MOV      R4, -(SP)
11         ;
12         ; See if we received a real break (long space).
13         ;
14 011760 016104 000000G      MOV      LNPRIM(R1), R4 ; Get primary line #
15 011764 032764 000000G 000000G      BIT      ##RBRK, LSW10(R4); Did we receive a break?
16 011772 001406      BEQ      1$ ; Br if not
17 011774 004737 017552'      CALL     SIGBRK ; Signal that we received a break
18 012000 042764 000000G 000000G      BIC      ##RBRK, LSW10(R4); Clear break-received flag
19 012006 000554      BR       PRCEND ; Finished with break character
20         ;
21         ; Mask character to 8 bits and ignore nulls
22         ;
23 012010 042705 177400      1$:     BIC      #^C377, R5 ; Mask character to 8 bits
24 012014 001551      BEQ      PRCEND ; Ignore null characters
25         ;
26         ; If debugger is in control, bypass some special character processing
27         ;
28 012016 032761 000000G 000000G      BIT      ##DBGMD, LSW6(R1); Is debugging program in control?
29 012024 001054      BNE      CKCW ; If yes then bypass some checking
30         ;
31         ; See if user defined an asynchronous break character
32         ;
33 012026 016102 000000G      MOV      LBRKCH(R1), R2 ; Did user define an asynch break char?
34 012032 001405      BEQ      CKSPAC ; Br if not
35 012034 020502      CMP      R5, R2 ; Is this the break character?
36 012036 001003      BNE      CKSPAC ; Br if not
37 012040 004737 017552'      CALL     SIGBRK ; Tell user that break char was received
38 012044 000535      BR       PRCEND
39         ;
40         ; See if this is a user-defined activation character
41         ;
42 012046 016102 000000G      CKSPAC: MOV     LNXPAC(R1), R2 ; Get number of user-defined activation chars
43 012052 001424      BEQ      CKVTES ; Br if there are none
44 012054 010500      MOV      R5, R0 ; Get current character
45 012056 004737 017422'      CALL     CVTLC ; Convert to upper-case if needed
46 012062 016103 000000G      MOV      LSPACT(R1), R3 ; Get pointer to table of activation chars
47 012066 120023      1$:     CMPB   R0, (R3)+ ; Is this an activation character?
48 012070 001402      BEQ      2$ ; Br if yes
49 012072 077203      SOB     R2, 1$ ; Loop if more to check
50 012074 000404      BR      CKHIIN ; This is not a user-defined activation char
51 012076 010005      2$:     MOV     R0, R5 ; Get converted character to R5
52 012100 004737 016200'      CALL     STRACT ; Store the activation character
53 012104 000515      BR      PRCEND
54         ;
55         ; See if we are in high-efficiency mode
56         ;
57 012106 032761 000000G 000000G      CKHIIN: BIT     ##HITTY, LSW4(R1); Are we in high-efficiency mode?

```

TTINCP -- Process received input characters

```

58 012114 001403          BEQ      CKVTES          ;Br if not
59 012116 004737 015702'  CALL      STRCHR          ;Store the character
60 012122 000506          BR        PRCEND
61
62          ; See if this is a VT52 escape-letter sequence
63
64 012124 032761 000000G 000000G CKVTES: BIT      ##VTESE,LSW5(R1); Activate on esc-letter sequence?
65 012132 001411          BEQ      CKCW            ;Br if not
66 012134 004737 017222'  CALL      SCACHK          ;Are we in single character activ mode?
67 012140 103406          BCS      CKCW            ;Br if yes
68 012142 005761 000000G  TST      LTTTCR(R1)      ;Is terminal input cml rtn scheduled?
69 012146 001003          BNE      CKCW            ;Br if yes
70 012150 004737 015210'  CALL      CKVTAC          ;Check for escape-letter sequence
71 012154 103071          BCC      PRCEND          ;Br if char was part of escape sequence
72
73          ; Check for request to switch to a virtual line
74
75 012156 120537 000000G  CKCW:    CMPB      R5,VVLSCH ;Is this char a request to switch to vir line?
76 012162 001024          BNE      1$             ;Br if not
77 012164 032761 000000G 000000G  BIT      ##CTRLW,LSW3(R1); Was last character also control-W?
78 012172 001414          BEQ      2$             ;Br if not
79 012174 042761 000000G 000000G  BIC      ##1ESC,LSW(R1)  ;Say last char was not escape
80 012202 042761 000000G 000000G  BIC      ##CTRLW,LSW3(R1); Say last char not control-W
81 012210 004737 017404'  CALL      SCACHR          ;See if we are in single-char activation mode
82 012214 103451          BCS      PRCEND          ;Br if in single-char activation mode
83 012216 004737 015664'  CALL      INCHR          ;Pass control-W to program as normal char
84 012222 000446          BR        PRCEND
85 012224 052761 000000G 000000G 2$:    BIS      ##CTRLW,LSW3(R1); Remember last char was control-W
86 012232 000442          BR        PRCEND
87 012234 032761 000000G 000000G 1$:    BIT      ##CTRLW,LSW3(R1); Was control-W the last character?
88 012242 001420          BEQ      CKICTL          ;Br if not
89 012244 020527 000037  CMP      R5,#37          ;Is current character a control character?
90 012250 101415          BLOS     CKICTL          ;Br if yes
91 012252 042761 000000G 000000G  BIC      ##CTRLW,LSW3(R1); Say control-W is not the last char
92 012260 162705 000060  SUB      #'0,R5          ;Convert line # digit to binary value
93 012264 002425          BLT      PRCEND          ;Br if too small
94 012266 020527 000000G  CMP      R5,#MAXSEC      ;Don't exceed max line # allowed
95 012272 003022          BGT      PRCEND          ;Br if too large
96 012274          DCALL     DOSWIT          ;Switch to a virtual line
97 012302 000416          BR        PRCEND
98
99          ; Determine if this is a normal or control character
100
101 012304 020527 000037  CKICTL:  CMP      R5,#37          ;Is this a normal or control char?
102 012310 101003          BHI      1$             ;Br if not control character
103 012312 004737 012566'  CALL      DOCTRL          ;Process a control character
104 012316 000410          BR        PRCEND
105 012320 120527 000000G 1$:    CMPB      R5,#RUBOUT      ;Is this a rubout character?
106 012324 001003          BNE      2$             ;Br if not
107 012326 004737 014652'  CALL      ICPRUB          ;Process rubout character
108 012332 000402          BR        PRCEND
109 012334 004737 012350' 2$:    CALL      REGCHR          ;Process a normal character
110
111          ; Finished
112
113 012340 012604  PRCEND:  MOV      (SP)+,R4
114 012342 012603  MOV      (SP)+,R3

```

115 012344 012602  
116 012346 000207

MOV (SP)+, R2  
RETURN

REGCHR -- Process normal characters

```

1          .SBTTL  REGCHR -- Process normal characters
2          ;-----
3          ; Process normal (non-control) characters.
4          ;
5          ; Inputs:
6          ; R1 = Virtual line index number.
7          ; R5 = Current input character.
8          ;
9 012350 010246 REGCHR: MOV     R2,-(SP)
10 012352 010546      MOV     R5,-(SP)
11 012354 010500      MOV     R5,R0          ;Copy the character
12          ;
13          ; Ignore all input while processing logoff command file
14          ;
15 012356 032761 000000G 000000G      BIT     #$LOFCF,LSW9(R1);Are we processing a logoff command file?
16 012364 001075      BNE     9$          ;Br if yes -- ignore the character
17          ;
18          ; Say last character was not control-C
19          ;
20 012366 042761 000000G 000000G      BIC     #$1ESC,LSW(R1) ;Say last char was not escape
21 012374 042761 000000G 000000G      BIC     #$1CTL,LSW5(R1);Say last character was not control-C
22          ;
23          ; See if we are in single character activation mode
24          ;
25 012402 004737 017262'      CALL    SLCHK          ;Are we in single-char activation mode or SL?
26 012406 103003      BCC     6$          ;Br if not
27 012410 004737 016260'      CALL    STRSNG        ;Store character and activate
28 012414 000461      BR     9$
29          ;
30          ; See if ODT is in control
31          ;
32 012416 032761 000000G 000000G 6$:  BIT     #$ODTMD,LSW4(R1);Is ODT activation in effect?
33 012424 001413      BEQ     3$          ;Br if not
34 012426 004737 015422'      CALL    CHKODT        ;Is this an ODT activation char?
35 012432 103010      BCC     3$          ;Br if not
36 012434 120027 000040      CMPB   R0,#40         ;Is this a control character?
37 012440 103402      BLD     4$          ;Br if yes -- Don't echo it
38 012442 004737 017040'      CALL    ECHO          ;Echo the character
39 012446 004737 016260'      4$:  CALL    STRSNG        ;Store character and activate
40 012452 000442      BR     9$
41          ;
42          ; Check for activation on input of a certain number of characters
43          ;
44 012454 016102 000000G      3$:  MOV     LAFSIZ(R1),R2 ;Was field width activation specified?
45 012460 001416      BEQ     1$          ;Br if not
46 012462 005302      DEC     R2          ;Is field full yet?
47 012464 020261 000000G      CMP     R2,LINCNT(R1)
48 012470 101012      BHI     1$          ;Br if not
49 012472 032761 000000G 000000G      BIT     #$DBGMD,LSW6(R1);Is a debugger running?
50 012500 001006      BNE     1$          ;Br if yes
51 012502 004737 017040'      CALL    ECHO          ;Echo character
52 012506 010005      MOV     R0,R5        ;Save the converted character
53 012510 004737 016200'      CALL    STRACT        ;Store the activation character
54 012514 000421      BR     9$
55          ;
56          ; See if we need to limit number of characters that can be
57          ; typed into a field

```

```
58  
59 012516 016102 000000G      1$:   MOV     LFWLIM(R1),R2   ; If field width limit specified?  
60 012522 001414              BEQ     2$             ; Br if not  
61 012524 026102 000000G      CMP     LINCNT(R1),R2   ; Would this character overflow the field?  
62 012530 103411              BLO     2$             ; Br if not  
63 012532 032761 000000G 000000G  BIT     ##DBGMD,LSW6(R1); Is debugger in control?  
64 012540 001005              BNE     2$             ; Br if yes  
65 012542 012700 000000G      MOV     #BELL,R0       ; Echo bell  
66 012546 004737 017100'      CALL   ECHO2  
67 012552 000402              BR      9$             ; Discard the character  
68  
69                          ; Normal character being input  
70  
71 012554 004737 015664'      2$:   CALL   INCHR       ; Store and echo the character  
72  
73                          ; Finished  
74  
75 012560 012605              9$:   MOV     (SP)+,R5  
76 012562 012602              MOV     (SP)+,R2  
77 012564 000207              RETURN
```



DOCTRL -- Process control characters

```

1          .SBTTL DOCTRL -- Process control characters
2          ;-----
3          ; DOCTRL is called from the input interrupt character processing when
4          ; we determine that the character being processed is a control character.
5          ;
6          ; Inputs:
7          ;   R1 = Virtual line number.
8          ;   R5 = Character to process.
9          ;
10         DOCTRL:
11         ;
12         ; See if this is a request to print the current window
13         ;
14         012566 120537 000000G          CMPB    R5,VVPWCH      ;Request to print screen?
15         012572 001016                  BNE     1$           ;Br if not
16         012574 032761 000000G 000000G  BIT     ##PWKEY,LSW11(R1); Is print window control char enabled?
17         012602 001412                  BEQ     1$           ;Br if not
18         012604 016100 000000G          MOV     LWINDO(R1),R0 ; Is windowing enabled for this process?
19         012610 001407                  BEQ     1$           ;Br if not -- Treat char like ordinary char
20         ;
21         ; This is a request to print the current window contents
22         ;
23         012612 010246                  MOV     R2,-(SP)
24         012614 010002                  MOV     R0,R2        ;Get address of current window control blk
25         012616                  DCALL  WINPRT        ;Print the window
26         012624 012602 2$: MOV     (SP)+,R2
27         012626 000404                  BR     9$
28         ;
29         ; This is an ordinary control character
30         ;
31         012630 010500 1$: MOV     R5,R0        ;Get the control character
32         012632 006300                  ASL     R0           ;Convert to word table index
33         012634 004770 012642'          CALL   @CTLRTN(R0)  ;Call appropriate processing routine
34         ;
35         ; Finished
36         ;
37         012640 000207 9$: RETURN
38         ;
39         ;-----
40         ; Branch table for control character processing routines.
41         ;
42         CTLRTN: .WORD REGCHR          ; 00 - NUL
43         .WORD REGCHR          ; 01 - SOH (control-A)
44         .WORD REGCHR          ; 02 - STX (control-B)
45         .WORD ICPCTC          ; 03 - ETX (control-C)
46         .WORD ICPCTD          ; 04 - EDT (control-D)
47         .WORD REGCHR          ; 05 - ENQ (control-E)
48         .WORD REGCHR          ; 06 - ACK (control-F)
49         .WORD ICPCTG          ; 07 - BEL (control-G)
50         .WORD REGCHR          ; 10 - Backspace
51         .WORD REGCHR          ; 11 - TAB (control-I)
52         .WORD ICPLF          ; 12 - Line feed
53         .WORD REGCHR          ; 13 - VT (control-K)
54         .WORD REGCHR          ; 14 - FF (control-L)
55         .WORD ICPCR          ; 15 - Carriage return
56         .WORD REGCHR          ; 16 - SO (control-N)
57         .WORD ICPCTO          ; 17 - SI (control-O)

```

DOCTRL -- Process control characters

58 012702 012350'	. WORD	REGCHR	; 20 - DLE (control-P)
59 012704 012350'	. WORD	RFGCHR	; 21 - DC1 (control-Q)
60 012706 013756'	. WORD	ICPCTR	; 22 - DC2 (control-R)
61 012710 012350'	. WORD	REGCHR	; 23 - DC3 (control-S)
62 012712 012350'	. WORD	REGCHR	; 24 - DC4 (control-T)
63 012714 014022'	. WORD	ICPCTU	; 25 - NAK (control-U)
64 012716 012350'	. WORD	REGCHR	; 26 - SYN (control-V)
65 012720 012350'	. WORD	REGCHR	; 27 - ETB (control-W)
66 012722 014402'	. WORD	ICPCTX	; 30 - CAN (control-X)
67 012724 012350'	. WORD	REGCHR	; 31 - EM (control-Y)
68 012726 014466'	. WORD	ICPCTZ	; 32 - SUB (control-Z)
69 012730 014522'	. WORD	ICPESC	; 33 - ESC
70 012732 012350'	. WORD	REGCHR	; 34 - FS
71 012734 012350'	. WORD	REGCHR	; 35 - GS
72 012736 012350'	. WORD	REGCHR	; 36 - RS
73 012740 012350'	. WORD	REGCHR	; 37 - US

ICPCR -- Carriage-return processing

```

1          .SBTTL  ICPCR  -- Carriage-return processing
2          ;-----
3          ; Process Carriage-return character.
4          ;
5          ; Inputs:
6          ; R1 = Virtual line index number.
7          ; R5 = Current input character.
8          ;
9 012742  010546  ICPCR:  MOV      R5,-(SP)
10         ;
11         ; See if we are in single-character activation mode
12         ;
13 012744  004737  017262'      CALL    SLCHK          ;Are we in single-char activation mode?
14 012750  103005          BCC     1$              ;Br if not
15 012752  004737  016200'      CALL    STRACT         ;Pass carriage return to program
16 012756  112705  000000G     MOVB   #LF,R5         ;and follow with line feed
17 012762  000433          BR     3$
18         ;
19         ; We are not in single-character activation mode.
20         ; See if ODT is running.
21         ;
22 012764  032761  000000G 000000G 1$:  BIT    #ODTMD,LSW4(R1); Is ODT in control?
23 012772  001410          BEQ    4$              ;Br if not
24 012774  010500          MOV    R5,R0          ;Get the CR character
25 012776  004737  017040'      CALL    ECHO          ;Echo CR
26 013002  012700  000000G     MOV    #LF,R0         ;Get LF character
27 013006  004737  017044'      CALL    ECHO1         ;Echo LF
28 013012  000417          BR     3$              ;Store CR as activation character
29         ;
30         ; We are not in ODT mode
31         ;
32 013014  004737  015664'      4$:  CALL    INCHR         ;Store and echo CR
33 013020  112705  000000G     MOVB   #LF,R5         ;We will follow CR with LF
34 013024  032761  000000G 000000G  BIT    #DBGMD,LSW6(R1); Is a debug program running?
35 013032  001004          BNE   2$              ;Br if yes
36 013034  032761  000000G 000000G  BIT    #NOLF,LSW6(R1); Are we suppressing LF echoing after CR?
37 013042  001003          BNE   3$              ;Br if yes
38 013044  010500          2$:  MOV    R5,R0          ;Get character to echo
39 013046  004737  017044'      CALL    ECHO1         ;Echo line-feed
40 013052  004737  016200'      3$:  CALL    STRACT         ;Store activation character
41         ;
42         ; Finished
43         ;
44 013056  042761  000000G 000000G 9$:  BIC    #1CTL,LSW5(R1); Say last character was not control-C
45 013064  042761  000000G 000000G  BIC    #1ESC,LSW(R1) ; Say last char was not escape
46 013072  012605          MOV    (SP)+,R5
47 013074  000207          RETURN

```

```

1          .SBTTL  ICPLF  -- Line-feed processing
2          ;-----
3          ; Process Line-feed input character.
4          ;
5          ; Inputs:
6          ;   R1 = Virtual line index number.
7          ;   R5 = Current input character.
8          ;
9 013076 010546 ICPLF:  MOV    R5,-(SP)
10         ;
11         ; If we are in "paper-tape" mode, ignore line-feed characters.
12         ;
13 013100 032761 000000G 000000G      BIT    ##TAPE,LSW2(R1) ;Are we in paper-tape mode?
14 013106 001025          BNE    9$          ;Br if yes
15         ;
16         ; See if we are in single-character activation mode
17         ;
18 013110 004737 017404'          CALL   SCACHR          ;See if we are in single-char activation mode
19 013114 103422          BCS    9$          ;Br if yes
20         ;
21         ; We are not in single-character activation mode.
22         ; See if we are in ODT mode.
23         ;
24 013116 032761 000000G 000000G      BIT    ##ODTMD,LSW4(R1); Is ODT in control?
25 013124 001412          BEQ    2$          ;Br if not
26 013126 112700 000000G          MOVB   #CR,R0          ;Echo carriage return
27 013132 004737 017040'          CALL   ECHO
28 013136 110500          MOVB   R5,R0          ;Get line feed character
29 013140 004737 017044'          CALL   ECHO1         ;Echo line-feed
30 013144 004737 016200'          CALL   STRACT        ;Store line feed and activate
31 013150 000404          BR     9$
32         ;
33         ; Treat line-feed exactly like carriage-return.
34         ;
35 013152 112705 000000G 2$:  MOVB   #CR,R5          ;Get CR character
36 013156 004737 012742'          CALL   ICPCR         ;Process the carriage-return
37         ;
38         ; Finished
39         ;
40 013162 042761 000000G 000000G 9$: BIC    ##1CTL,LSW5(R1); Say last character was not control-C
41 013170 042761 000000G 000000G      BIC    ##1ESC,LSW(R1) ; Say last char was not escape
42 013176 012605          MOV    (SP)+,R5
43 013200 000207          RETURN

```

```

1          .SBTTL  ICPCTC -- Control-C processing
2          ;-----
3          ; Process a control-C input character.
4          ;
5          ; Inputs:
6          ;   R1 = Virtual line index number.
7          ;   R5 = Current input character.
8          ;
9 013202   010246   ICPCTC: MOV      R2,-(SP)
10 013204   010446   MOV      R4,-(SP)
11          ;
12          ; Determine if we have received 2 consecutive control-C characters.
13          ;
14 013206   032761   000000G 000000G   BIT      ##1CTL,LSW5(R1);Was last character control-C?
15 013214   001021   BNE      2$          ;Br if yes
16          ;
17          ; This is 1st control-C
18          ;
19 013216   052761   000000G 000000G   BIS      ##1CTL,LSW5(R1);Remember last char was control-C
20 013224   042761   000000G 000000G 7$: BIC      ##1ESC,LSW(R1) ;Say last char was not escape
21 013232   042761   000000G 000000G   BIC      ##RTCS,LSW9(R1) ;Say we are not receiving control sequence
22 013240   004737   017404'   CALL    SCACHR          ;See if we are in single-char activation mode
23 013244   103527   BCS      9$          ;Br if yes
24 013246   004737   017116'   CALL    ECOCTL          ;Echo "^C"
25 013252   004737   016200'   CALL    STRACT          ;Store control-C as activation char
26 013256   000522   BR      9$
27          ;
28          ; We received two consecutive control-C characters
29          ;
30 013260   032761   000000C 000000G 2$: BIT      #<$SUCF!$LODFCF>,LSW9(R1);Are we doing logon/logoff processing?
31 013266   001356   BNE      7$          ;Br if yes -- don't allow ctrl-C abort here
32 013270   032761   000000G 000000G   BIT      ##SCCA,LSW5(R1) ;Suppressing control-C aborts for program?
33 013276   001352   BNE      7$          ;Br if yes
34 013300   032761   000000G 000000G   BIT      ##NOIN,LSW3(R1) ;Special no-input flag set?
35 013306   001346   BNE      7$          ;Br if yes
36 013310   032761   000000G 000000G   BIT      ##RFRSH,LSW4(R1);Is a window refresh being done now?
37 013316   001342   BNE      7$          ;Don't allow abort during refresh
38 013320   016102   000000G   MOV     LSCCA(R1),R2   ;Did user do .SCCA EMT?
39 013324   001412   BEQ     3$          ;Br if not
40 013326   032761   000000G 000000G   BIT      ##DBGMD,LSW6(R1);Is debugger in control?
41 013334   001006   BNE     3$          ;Br if yes -- ignore .SCCA
42          ;
43          ; User did a .SCCA -- Set flag to remember to tell him about ctrl-C's
44          ;
45 013336   052761   000000G 000000G   BIS     ##SETCC,LSW4(R1);Remember to tell him later
46 013344   105237   000000G   INCB   DOSCHD          ;Request a job scheduler cycle
47 013350   000725   BR      7$
48          ;
49          ; User did not do a .SCCA so abort him
50          ;
51 013352   042761   000000C 000000G 3$: BIC     #<LCBIT!SPCTTY>,LJSW(R1) ;Clean out JSW
52 013360   016161   000000G 000000G   MOV     LINNXT(R1),LNPNT(R1) ;Kill all input
53 013366   016161   000000G 000000G   MOV     LINNXT(R1),LSTACT(R1)
54 013374   005061   000000G   CLR     LINCNT(R1)
55 013400   016161   000000G 000000G   MOV     LINSIZ(R1),LINSPC(R1)
56 013406   016161   000000G 000000G   MOV     LOTSIZ(R1),LOTSPC(R1) ;Kill all output
57 013414   016161   000000G 000000G   MOV     LOTNXT(R1),LOTPNT(R1)

```

```
58 013422 005061 000000G CLR LFWLIM(R1) ;No field width limit
59 013426 005061 000000G CLR LAFST7(R1) ;No field width activation
60 013432 042761 000000G 000000G BIC ##SLINI,LSW7(R1);Say SL must reinitialize for next line
61 013440 042761 000000C 000000G BIC #<#DODFR!#GCECO>,LSW3(R1) ;Reset deferred echoing
62 013446 016104 000000G MOV LNPRIM(R1),R4 ;Get primary job number
63 013452 042764 000000G 000000G BIC ##CTRLS,LSW3(R4);Reset control-S output suspension
64 013460 004737 017116' CALL ECOCTL ;Echo "^C"
65 013464 052761 000000G 000000G BIS ##CTRLC,LSW(R1) ;Set job abort flag
66 013472 052761 000000G 000000G BIS ##CFKIL,LSW6(R1);Set flag to abort all open command files
67 013500 042761 000000G 000000G BIC ##SUSPN,LSW(R1) ;Clear job-suspended flag
68 013506 004737 000000G CALL FORCEX ;Force execution of the job
69 013512 005061 000000G CLR LACTIV(R1) ;Say no chars received yet
70 013516 042761 000000G 000000G BIC ##1STCH,LSW3(R1)
71 ;
72 ; Finished
73 ;
74 013524 012604 9#: MOV (SP)+,R4
75 013526 012602 MOV (SP)+,R2
76 013530 000207 RETURN
```

```

1          .SBTTL  ICPCTD -- Control-D processing
2          ;-----
3          ; Process a Control-D character.
4          ; This forces a debugger breakpoint if we are running under the debugger.
5          ;
6          ; Inputs:
7          ; R1 = Virtual line index number.
8          ; R5 = Current input character.
9          ;
10         013532 032761 000000G 000000G ICPCTD: BIT    ##INKMN,LSW4(R1);Are we running in TSKMON now?
11         013540 001405                BEQ    2$      ;Br if not
12         013542 032761 000000G 000000G        BIT    ##DBKMN,LSW9(R1);Should we debug TSKMON?
13         013550 001411                BEQ    1$      ;Br if not
14         013552 000404                BR     3$      ;Yes, go force breakpoint
15         013554 032761 000000C 000000G 2$:    BIT    ##DEBUG!$CTRLD,LSW9(R1);Is job running with the debugger?
16         013562 001404                BEQ    1$      ;Br if not
17         013564 052761 000000G 000000G 3$:    BIS    ##DBGBK,LSW9(R1);Set flag to force debug breakpoint
18         013572 000402                BR     9$
19         ;
20         ; We are not running with debugger, treat Ctrl-D like normal character
21         ;
22         013574 004737 012350' 1$:    CALL  REGCHR      ;Store as regular character
23         ;
24         ; Finished
25         ;
26         013600 000207 9$:    RETURN
  
```

```

1          .SBTTL  ICPCTG -- Control-G processing
2          ;-----
3          ; Process a Control-G (Bell) character.
4          ;
5          ; Inputs:
6          ; R1 = Virtual line index number.
7          ; R5 = Current input character.
8          ;
9 013602 032761 000000G 000000G ICPCTG: BIT    #SPCTTY,LSW(R1); Is special character processing wanted?
10 013610 001415                BEQ     2$      ; Br if not
11 013612 032761 000000G 000000G        BIT    #$CHACT,LSW5(R1); Is single character activation wanted?
12 013620 001403                BEQ     1$      ; Br if not
13 013622 004737 016260'              CALL   STRSNG          ; Store the character
14 013626 000410                BR      9$
15          ;
16          ; Special-TTY mode is set but not single character activation
17          ;
18 013630 010500                1$:    MOV     R5,R0          ; Get char to echo
19 013632 004737 017040'              CALL   ECHO           ; Echo a bell
20 013636 004737 016200'              CALL   STRACT        ; Store and activate
21 013642 000402                BR      9$
22          ;
23          ; Normal character processing
24          ;
25 013644 004737 012350'              2$:    CALL   REGCHR          ; Treat as regular character
26          ;
27          ; Finished
28          ;
29 013650 042761 000000G 000000G 9$:    BIC     #$1CTL,LSW5(R1); Say last character was not control-C
30 013656 042761 000000G 000000G        BIC     #$1ESC,LSW(R1) ; Say last char was not escape
31 013664 000207                RETURN

```



```
1          .SBTTL  ICPCTO -- Control-O processing
2          ;-----
3          ; Process a Control-O input character.
4          ;
5          ; Inputs:
6          ;   R1 = Virtual line index number.
7          ;   R5 = Current input character.
8          ;
9 013666 032761 000000G 000000G ICPCTO: BIT    ##CTRLD,LSW3(R1); Is output currently suppressed?
10 013674 001022                BNE      1$      ;Br if yes (reenable it)
11          ;
12          ; Begin suppressing output
13          ;
14 013676 016161 000000G 000000G          MOV     LOTSIZ(R1),LOTSPC(R1) ;Empty output buffer
15 013704 016161 000000G 000000G          MOV     LOTNXT(R1),LOTPNT(R1)
16 013712 004737 017116'                CALL    ECOCTL          ;Echo "^O"
17 013716 052761 000000G 000000G          BIS     ##CTRLD,LSW3(R1);Remember output is being suppressed
18 013724 026127 000000G 000000G          CMP     LSTATE(R1),#S#OTWT ;Is job waiting for output space?
19 013732 001010                BNE     2$      ;Br if not
20 013734 004737 005744'                CALL    OTREGO         ;Reactivate the job
21 013740 000405                BR      2$
22          ;
23          ; Reenable output
24          ;
25 013742 042761 000000G 000000G 1$:    BIC     ##CTRLD,LSW3(R1);Remove output suppression
26 013750 004737 017116'                CALL    ECOCTL          ;Echo "^O"
27          ;
28          ; Finished
29          ;
30 013754 000207                2$:    RETURN
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9 013756 010246  
10 013760 004737 000000G  
11 013764 004737 017404'  
12 013770 103412  
13  
14  
15  
16 013772 004737 017116'  
17 013776 016102 000000G  
18 014002 004737 016320'  
19 014006 103403  
20 014010 004737 017044'  
21 014014 000772  
22  
23  
24  
25 014016 012602  
26 014020 000207
```

```
          .SBTTL  ICPCTR -- Control-R processing  
-----  
; Process control-R input character.  
;  
; Inputs:  
; R1 = Virtual line index number.  
; R5 = Current input character.  
;  
ICPCTR:  MOV     R2, -(SP)  
          CALL   BRKPT      ; Call TSEXEC for debugging breakpoint  
          CALL   SCACHR     ; Is job in single-char activation mode?  
          BCS   9$         ; Br if yes  
;  
; Redisplay the current input line  
;  
4$:      CALL   ECOCTL     ; Echo "^R"  
          MOV   LSTACT(R1), R2 ; Point past last activation char  
1$:      CALL   FETCHR    ; Get next char from TT input buffer  
          BCS   9$         ; Br if no more characters  
          CALL   ECHO1     ; Echo the character  
          BR    1$         ; Continue printing line  
;  
; Finished  
;  
9$:      MOV   (SP)+, R2  
          RETURN
```

```

1                                     .SBTTL ICPCTU -- Control-U processing
2                                     ;-----
3                                     ; Process a Control-U character.
4                                     ;
5                                     ; Inputs:
6                                     ; R1 = Virtual line index number.
7                                     ; R5 = Current input character.
8                                     ;
9 014022 010246 ICPCTU: MOV      R2,-(SP)
10 014024 010346      MOV      R3,-(SP)
11                                     ;
12                                     ; Remember that last character was not escape or control-C
13                                     ;
14 014026 042761 000000G 000000G      BIC      ##1CTLC,LSW5(R1);Last character was not control-C
15 014034 042761 000000G 000000G      BIC      ##1ESC,LSW(R1) ;Say last char was not escape
16                                     ;
17                                     ; Determine if we are in single-character activation mode
18                                     ;
19 014042 004737 017404'      CALL     SCACHR      ;See if we are in single-char activation mode
20 014046 103545      BCS      27$          ;Br if yes
21                                     ;
22                                     ; We are not in single-charactr activation mode
23                                     ;
24 014050 042761 000000G 000000G      BIC      ##RBOU,LSW3(R1);Reset rubout mode
25 014056 032761 000000G 000000G      BIT      ##DODFR,LSW3(R1);Doing deferred echoing?
26 014064 001133      BNE      5$          ;Br if yes
27                                     ;
28                                     ; Determine if this is a scope type terminal
29                                     ;
30 014066 032761 000000G 000000G      BIT      ##SCOPE,LSW2(R1);Is this a scope terminal?
31 014074 001003      BNE      15$         ;Br if yes
32                                     ;
33                                     ; Echo "^U" for non-scope terminals
34                                     ;
35 014076 004737 017116'      CALL     ECOCTL      ;Echo "^U"
36 014102 000524      BR       5$          ;Go delete the characters
37                                     ;
38                                     ; Do line erase for scope terminals
39                                     ;
40 014104 032761 000000G 000000G 15$: BIT      ##1STCH,LSW3(R1);Any characters received yet?
41 014112 001523      BEQ      27$         ;Br if not
42 014114 116103 000000G      MOVB     LCOL(R1),R3      ;Get current column position
43 014120 042703 177400      BIC      #^C377,R3      ;Kill sign extension
44 014124 116102 000000G      MOVB     LINCUR(R1),R2    ;Get start of line position
45 014130 042702 177400      BIC      #^C377,R2      ;Kill sign extension
46 014134 160203      SUB      R2,R3          ;Get # of columns in field being erased
47 014136 003506      BLE      5$          ;Br if nothing to erase
48 014140 126127 000000G 000000G      CMPB     LRFIL(R1),#SPACE; Is rubout filler char = space?
49 014146 001056      BNE      8$          ;Br if not
50 014150 032761 000000C 000000G      BIT      #VT52!VT100!VT2007!VT2008!HAZEL,LTRMTP(R1) ;Can we line erase?
51 014156 001452      BEQ      8$          ;Br if not
52 014160 005702      TST      R2          ;Are we erasing entire line?
53 014162 001005      BNE      10$         ;Br if not
54                                     ;
55                                     ; Going to start of line. Use carriage-return to get there.
56                                     ;
57 014164 112700 000000G      MOVB     #CR,R0          ;Send carriage return to terminal

```

ICPCTU -- Control-U processing

```

58 014170 004737 003122'          CALL    PUTCH2
59 014174 000405                   BR      11$
60                               ;
61                               ; Not going to start of line.  Use backspaces to get to start of field.
62                               ;
63 014176 112700 000000G          10$:    MOVB    #BKSPAC,R0      ;Get backspace character
64 014202 004737 017044'          12$:    CALL    ECHO1        ;Send backspace to terminal
65 014206 077303                   SOB     R3,12$        ;Send enough to get to start of field
66                               ;
67                               ; Use scope control sequence to erase the line
68                               ;
69 014210 012703 014370'          11$:    MOV     #ERV52,R3      ;Assume this is a VT52 terminal
70 014214 116100 000000G          MOVB    LNPRIM(R1),R0    ;Get primary job index number
71 014220 032760 000000G 000000G BIT     #$V52EM,LSW11(R0);Are we emulating a VT52?
72 014226 001014                   BNE     13$          ;Br if yes
73 014230 016100 000000G          MOV     LTRMTP(R1),R0   ;Get terminal type code
74 014234 032700 000000G          BIT     #VT52,R0      ;Is this a VT52 terminal?
75 014240 001007                   BNE     13$          ;Br if yes
76 014242 012703 014377'          MOV     #ERHAZL,R3    ;Assume hazeltine terminal
77 014246 032700 000000G          BIT     #HAZEL,R0     ;Is this a Hazeltine terminal?
78 014252 001002                   BNE     13$          ;Br if yes
79 014254 012703 014373'          MOV     #ERV100,R3   ;Assume this is a VT100 or VT200
80 014260 112300                   13$:    MOVB    (R3)+,R0    ;Get next char from control sequence
81 014262 001434                   BEQ     5$           ;Br if end reached
82 014264 120027 000037          CMPB   RO,#37        ;Is this a control or printing character
83 014270 101402                   BLOS   14$          ;Br if control character
84 014272 105361 000000G          DECB   LCOL(R1)     ;Correct column counter if sending printing ch
85 014276 004737 003122'          14$:    CALL    PUTCH2        ;Send character to terminal
86 014302 000766                   BR      13$        ;Continue sending control sequence
87                               ;
88                               ; Kill all input characters by sending
89                               ; backspace-space-backspace...
90                               ;
91 014304 116102 000000G          8$:    MOVB    LRBFIL(R1),R2 ;Get rubout-filler character
92 014310 032761 000000G 000000G BIT     #$DBGMD,LSW6(R1);Is debugger doing I/O now?
93 014316 001402                   BEQ     9$           ;Br if not
94 014320 112702 000040          MOVB   #' ,R2       ;Use blank for debugger rubout
95 014324 112700 000000G          9$:    MOVB    #BKSPAC,R0
96 014330 004737 017044'          CALL    ECHO1        ;Send backspace
97 014334 110200                   MOVB   R2,R0        ;Get rubout-filler character
98 014336 004737 017044'          CALL    ECHO1        ;Send filler character
99 014342 112700 000000G          MOVB   #BKSPAC,R0   ;Send backspace
100 014346 004737 017044'         CALL    ECHO1
101 014352 077314                   SOB    R3,9$        ;Loop on number of characters to kill
102                               ;
103                               ; Now delete characters from the input buffer
104                               ;
105 014354 004737 015572'         5$:    CALL    KILCHR        ;Kill last character in the buffer
106 014360 103375                   BCC    5$           ;Loop if more left to kill
107                               ;
108                               ; Finished
109                               ;
110 014362 012603          27$:    MOV     (SP)+,R3
111 014364 012602          MOV     (SP)+,R2
112 014366 000207          RETURN
113                               ;
114                               ; Terminal control sequences to erase a line.

```

```
115 ;  
116 014370 000G 113 000 ERV52: .BYTE ESC,113,0 ;VT52  
117 014373 000G 133 113 ERV100: .BYTE ESC,133,113,0 ;VT100 and VT200  
    014376 000  
118 014377 176 017 000 ERHAZL: .BYTE 176,17,0 ;Hazeltine  
119 .EVEN
```

```

1                                     .SBTTL  ICPCTX -- Control-X processing
2                                     ;-----
3                                     ; Process a Control-X character.
4                                     ;
5                                     ; Inputs:
6                                     ; R1 = Virtual line index number.
7                                     ; R5 = Current input character.
8                                     ;
9 014402 032761 000000G 000000G ICPCTX: BIT    #SPCTTY,LJSW(R1); Is special character processing wanted?
10 014410 001415                BEQ     2$      ; Br if not
11 014412 032761 000000G 000000G        BIT    #$CHACT,LSW5(R1); Is single character activation wanted?
12 014420 001403                BEQ     1$      ; Br if not
13 014422 004737 016260'              CALL   STRSNG          ; Store the character
14 014426 000410                BR      9$
15                                     ;
16                                     ; Special-TTY mode is set but not single character activation
17                                     ;
18 014430 010500                1$:    MOV     R5,R0          ; Get char to echo
19 014432 004737 017116'              CALL   ECOCTL          ; Echo "^X"
20 014436 004737 016200'              CALL   STRACT          ; Store the character and activate
21 014442 000402                BR      9$
22                                     ;
23                                     ; Normal character processing
24                                     ;
25 014444 004737 012350'              2$:    CALL   REGCHR          ; Treat as regular character
26                                     ;
27                                     ; Finished
28                                     ;
29 014450 042761 000000G 000000G 9$:    BIC     #$1CTL,LSW5(R1); Say last character was not control-C
30 014456 042761 000000G 000000G        BIC     #$1ESC,LSW(R1) ; Say last char was not escape
31 014464 000207                RETURN

```

```
1 .SBTTL ICPCTZ -- Control-Z processing
2 ;-----
3 ; Process Control-Z character.
4 ;
5 ; Inputs:
6 ; R1 = Virtual line index number.
7 ; R5 = Current input character.
8 ;
9 014466 004737 017404' ICPCTZ: CALL SCACHR ;Are we in single character activation mode?
10 014472 103404 BCS 9$ ;Br if yes
11 014474 004737 017116' CALL ECOCTL ;Echo "^Z"
12 014500 004737 016200' CALL STRACT ;Store character and activate
13 ;
14 ; Say last character was not escape or control-C
15 ;
16 014504 042761 000000G 000000G 9$: BIC ##1CTL,LSW5(R1);Say last character was not control-C
17 014512 042761 000000G 000000G BIC ##1ESC,LSW(R1) ;Say last char was not escape
18 014520 000207 RETURN
```

ICPESC -- Escape processing

```

1                                     .SBTTL  ICPESC -- Escape processing
2                                     ;-----
3                                     ; Process an escape character.
4                                     ;
5                                     ; Inputs:
6                                     ; R1 = Virtual line index number.
7                                     ; R5 = Current input character.
8                                     ;
9 014522 004737 017262' ICPESC: CALL    SLCHK          ; See if SL should get escape
10 014526 103410          BCS      4$          ; Br if SL wants characters
11 014530 032761 000000G 000000G    BIT      #SPCTTY,LJSW(R1); Is job in special TTY mode?
12 014536 001407          BEQ      1$          ; Br if not
13 014540 032761 000000G 000000G    BIT      ##CHACT,LSW5(R1); Is single char activation enabled?
14 014546 001412          BEQ      2$          ; Br if not
15 014550 004737 016260' 4$:      CALL    STRSNG          ; Store and activate
16 014554 000432          BR       9$
17                                     ;
18                                     ; Escape and not single character activation
19                                     ;
20 014556 112700 000044 1$:      MOVB    #'$,R0          ; Echo "$"
21 014562 004737 017040'          CALL    ECHO
22 014566 004737 015702'          CALL    STRCHR          ; Store escape as non-activation char
23 014572 000423          BR       9$
24                                     ;
25                                     ; Program is in special-TTY mode but single character activation
26                                     ; is not enabled.
27                                     ;
28 014574 112700 000044 2$:      MOVB    #'$,R0          ; Echo "$"
29 014600 004737 017040'          CALL    ECHO
30 014604 032761 000000G 000000G    BIT      ##1ESC,LSW(R1) ; Was last character ESC?
31 014612 001006          BNE      3$          ; Br if yes
32 014614 004737 015702'          CALL    STRCHR          ; Store the escape
33 014620 052761 000000G 000000G    BIS      ##1ESC,LSW(R1) ; Remember that last char is escape
34 014626 000405          BR       9$
35 014630 004737 016200' 3$:      CALL    STRACT          ; Store escape and activate
36 014634 042761 000000G 000000G    BIC      ##1ESC,LSW(R1) ; Say last char was not escape
37                                     ;
38                                     ; Finished
39                                     ;
40 014642 042761 000000G 000000G 9$:  BIC      ##1CTLC,LSW5(R1); Say last character was not control-C
41 014650 000207          RETURN

```



ICPRUB -- Rubout processing

```

1          .SBTTL  ICPRUB -- Rubout processing
2          ;-----
3          ; Process a rubout character.
4          ;
5          ; Inputs:
6          ; R1 = Virtual line number.
7          ; R5 = Rubout character.
8          ;
9 014652 010246 ICPRUB: MOV     R2,-(SP)
10 014654 010346      MOV     R3,-(SP)
11          ;
12          ; Remember that last character was not control-C
13          ;
14 014656 042761 000000G 000000G      BIC     ##1CTLC,LSW5(R1);Last character was not control-C
15 014664 042761 000000G 000000G      BIC     ##1ESC,LSW(R1) ;Say last char was not escape
16          ;
17          ; See if we are in single-character activation mode
18          ;
19 014672 004737 017404'      CALL    SCACHR          ;Are we in single-char activation mode
20 014676 103541              BCS     2$              ;Br if yes
21          ;
22          ; We are not in single character activation mode.
23          ;
24 014700 032761 000000G 000000G      BIT     ##ECHO,LSW2(R1) ;Is echoing turned on?
25 014706 001004              BNE     12$              ;Br if yes
26 014710 032761 000000G 000000G      BIT     ##DBGMD,LSW6(R1);Is debugger in control?
27 014716 001417              BEQ     30$              ;Br if not
28 014720 032761 000000G 000000G 12$: BIT     ##SCOPE,LSW2(R1);Scope type terminal?
29 014726 001021              BNE     27$              ;Br if yes
30          ;
31          ; Rubout on non-scope terminal.
32          ; Echo \xxx\ type rubout sequence.
33          ;
34 014730 032761 000000G 000000G      BIT     ##RBDOUT,LSW3(R1);Are we already doing rubout sequence?
35 014736 001007              BNE     30$              ;Br if yes
36 014740 112700 000134      MOVB   #' \,RO          ;Get backslash to begin rubout sequence
37 014744 004737 017044'      CALL    ECHO1          ;Echo "\"
38 014750 052761 000000G 000000G      BIS     ##RBDOUT,LSW3(R1);Say we have started rubout sequence
39 014756 004737 015572'      30$: CALL    KILCHR          ;Delete one character from buffer
40 014762 103507              BCS     2$              ;Br if no char to delete
41 014764 004737 017044'      CALL    ECHO1          ;Echo deleted char
42 014770 000504              BR     2$
43          ;
44          ; Do backspace editing for scope type terminals
45          ;
46 014772 004737 015572'      27$: CALL    KILCHR          ;Delete one character from buffer
47 014776 103501              BCS     2$              ;Br if no character was left to delete
48 015000 120027 000000G      CMPB   RO,#TAB          ;Did we delete a tab character?
49 015004 001044              BNE     10$
50          ;
51          ; We just deleted a tab character.
52          ; Do correct backspacing.
53          ;
54 015006 116103 000000G      MOVB   LINCUR(R1),R3    ;Position of start of line
55 015012 042703 177400      BIC     #^C377,R3      ;Kill sign extension
56 015016 016102 000000G      MOV     LSTACT(R1),R2  ;Start of input string
57 015022 004737 016320'      3$: CALL    FETCHR          ;Get next char from TT input buffer

```

ICPRUB -- Rubout processing

```

58 015026 103417          BCS      6$          ;Br if no more characters
59 015030 120027 000000G  CMPB    RO,#TAR      ;Is character a tab?
60 015034 001005          BNE     7$          ;Br if not
61 015036 062703 000010  ADD     #8.,R3       ;Calculate spaces over it
62 015042 042703 000007  BIC     #7,R3
63 015046 000765          BR      3$
64 015050 120027 000000G  7$:    CMPB    RO,#BKSPAC ;Was character backspace?
65 015054 001002          BNE     5$          ;Br if not
66 015056 005303          DEC     R3          ;Backup cursor position
67 015060 000760          BR      3$
68 015062 005203          5$:    INC     R3          ;Advance cursor for normal character
69 015064 000756          BR      3$
70 015066 010302          6$:    MOV     R3,R2       ;Save position in front of last tab
71 015070 062702 000010  ADD     #8.,R2       ;Calc position after tab
72 015074 042702 000007  BIC     #7,R2
73 015100 160302          SUB     R3,R2       ;Calc number of backspace needed
74 015102 012700 000000G  MOV     #BKSPAC,R0   ;Now backspace cursor
75 015106 004737 017044'  9$:    CALL    ECHO1      ;Backspace over tab columns
76 015112 077203          SOB    R2,9$
77 015114 000432          BR      2$
78
79          ;
80          ; Check for rubout of backspace character
81
81 015116 120027 000000G  10$:   CMPB    RO,#BKSPAC   ;Is deleted character a backspace?
82 015122 001005          BNE     11$         ;Br if not
83 015124 112700 000000G  MOVB   #SPACE,R0    ;Output a space to kill the backspace
84 015130 004737 017044'  CALL    ECHO1
85 015134 000422          BR      2$
86
87          ;
88          ; Rubout of regular character.
89          ; Echo backspace-space-backspace
90
90 015136 112700 000000G  11$:   MOVB   #BKSPAC,R0   ;Echo backspace
91 015142 004737 017044'  CALL    ECHO1
92 015146 116100 000000G  MOVB   LRFIL(R1),R0  ;Get rubout-filler character
93 015152 032761 000000G 000000G  BIT     ##DBGMD,LSW6(R1);Is a debugger running?
94 015160 001402          BEQ     13$         ;Br if not
95 015162 112700 000040  MOVB   #' ,R0       ;Use space for debugger rubout
96 015166 004737 017044'  13$:   CALL    ECHO1      ;Echo rubout-filler character
97 015172 112700 000000G  MOVB   #BKSPAC,R0   ;Now echo backspace
98 015176 004737 017044'  CALL    ECHO1
99
100          ;
101          ; Finished
102
102 015202 012603          2$:    MOV     (SP)+,R3
103 015204 012602          MOV     (SP)+,R2
104 015206 000207          RETURN

```

CKVTAC -- Check for VTxx escape-letter activation

```

1          .SBTTL  CKVTAC -- Check for VTxx escape-letter activation
2          ;-----
3          ; We are activating on VTxx escape-letter sequences.
4          ; See if this is one and we should activate.
5          ;
6          ; Inputs:
7          ; R1 = Virtual line index number.
8          ; R5 = Current input character.
9          ;
10         ; Outputs:
11         ; C-flag cleared ==> Character was totally processed here.
12         ; C-flag set      ==> Character was not processed by us.
13         ;
14 015210  CKVTAC:
15         ;
16         ; See if this character is ESC, CSI, or SS3 which starts a terminal
17         ; control sequence.
18         ;
19 015210  120527  000033          CMPB   R5,#33          ; Is character ESC?
20 015214  001007          BNE    1$              ; Br if not
21 015216  052761  000000G 000000G  BIS    ##1ESC,LSW(R1) ; Remember last char was an escape
22 015224  052761  000000G 000000G  BIS    ##RTCS,LSW9(R1) ; Say we are receiving a control sequence
23 015232  000450          BR     15$              ; Char is part of sequence
24 015234  120527  000233  1$:    CMPB   R5,#233         ; Is character CSI?
25 015240  001403          BEQ    14$              ; Br if yes
26 015242  120527  000217          CMPB   R5,#217         ; Is character SS3?
27 015246  001004          BNE    2$              ; Br if not
28         ;
29         ; This character begins a new control sequence
30         ;
31 015250  052761  000000G 000000G 14$:   BIS    ##RTCS,LSW9(R1) ; Say we are receiving a control sequence
32 015256  000433          BR     3$              ; Go store the character
33         ;
34         ; See if this character is part of a terminal control sequence
35         ;
36 015260  032761  000000G 000000G 2$:   BIT    ##RTCS,LSW9(R1) ; Are we currently receiving a control seq?
37 015266  001453          BEQ    8$              ; Br if not
38         ;
39         ; We are currently receiving a terminal control sequence.
40         ; See if this character terminates the sequence.
41         ;
42 015270  120527  000101          CMPB   R5,#'A         ; Is this a upper-case letter?
43 015274  103424          BLO    3$              ; Br if not
44 015276  120527  000132          CMPB   R5,#'Z         ;
45 015302  101010          BHI    7$              ; Br if not upper-case letter
46 015304  120527  000117          CMPB   R5,#'O         ; Letter O?
47 015310  001013          BNE    4$              ; Br if not
48 015312  032761  000000G 000000G  BIT    ##1ESC,LSW(R1) ; Was last character ESC?
49 015320  001407          BEQ    4$              ; Br if not
50 015322  000411          BR     3$              ; ESC O is equivalent to SS3
51 015324  120527  000141  7$:    CMPB   R5,#141        ; Is this a lower-case letter?
52 015330  103406          BLO    3$              ; Br if not
53 015332  120527  000176          CMPB   R5,#176        ; Lower case letter or ~
54 015336  101003          BHI    3$              ; Br if not
55         ;
56         ; This character terminates the control sequence
57         ;

```

```
58 015340 042761 000000G 000000G 4$:      BIC      $#RTCS,LSW9(R1) ;Say no longer receiving control sequence
59                                     ;
60                                     ; This character is part of control sequence -- Pass to program
61                                     ;
62 015346 042761 000000G 000000G 3$:      BIC      $#1ESC,LSW(R1) ;Say last char was not escape
63 015354 010546                                     15$:     MOV      R5,-(SP) ;Save the character
64 015356 012705 000000G                                     MOV      #ESCFLG,R5 ;Get char that says escape sequence follows
65 015362 004737 015702'                                     CALL     STRCHR ;Store flag character
66 015366 012605                                     MOV      (SP)+,R5 ;Recover real character
67 015370 032761 000000G 000000G                                     BIT      $#RTCS,LSW9(R1) ;Have we terminated the sequence?
68 015376 001403                                     BEQ      5$ ;Br if yes
69 015400 004737 015702'                                     CALL     STRCHR ;Store char that is part of sequence
70 015404 000402                                     BR       6$
71 015406 004737 016200'                                     5$:     CALL     STRACT ;Store char as activation character
72 015412 000241                                     6$:     CLC ;Say we finished processing the character
73 015414 000401                                     BR       9$
74                                     ;
75                                     ; This character is not part of a control sequence
76                                     ;
77 015416 000261                                     8$:     SEC ;Signal char is not part of sequence
78                                     ;
79                                     ; Finished
80                                     ;
81 015420 000207                                     9$:     RETURN
```

CHKODT -- Check for ODT activation characters

```

1          .SBTTL  CHKODT -- Check for ODT activation characters
2          ;-----
3          ;  CHKODT is called to determine whether ODT character
4          ;  activation is desired and whether the character in RO
5          ;  is an ODT activation character.
6          ;  When called RO must contain the character to be tested
7          ;  and R1 must contain the user index number.
8          ;  On return the c-flag is set if the character is an
9          ;  ODT activation character.
10         ;  All registers are preserved.
11         ;
12 015422 032761 000000G 000000G  CHKODT: BIT    #ODTMD,LSW4(R1);ODT ACTIVATION DESIRED?
13 015430 001417          BEQ    1$          ;BRANCH IF NOT
14         ;
15         ;  User does want odt activation.
16         ;  Is this an ODT activation char?
17         ;
18 015432 120027 000054          CMPB   RO,#' ,          ;DON'T ACTIVATE ON ', '
19 015436 001414          BEQ    1$
20 015440 120027 000044          CMPB   RO,#'$          ;OR '$ '
21 015444 001411          BEQ    1$
22 015446 120027 000073          CMPB   RO,#';          ;OR '; '
23 015452 001406          BEQ    1$
24 015454 120027 000060          CMPB   RO,#'0          ;IS THIS A DIGIT?
25 015460 103405          BLO    2$          ;BRANCH IF NOT DIGIT
26 015462 120027 000071          CMPB   RO,#'9          ;BRANCH IF NOT DIGIT
27 015466 101002          BHI    2$
28         ;
29         ;  Not activation character.
30         ;
31 015470 000241          1$:    CLC          ;CLEAR C-FLAG
32 015472 000207          RETURN
33         ;
34         ;  Got activation char.
35         ;
36 015474 000261          2$:    SEC          ;SET C-FLAG
37 015476 000207          RETURN

```

```

1          .SBTTL  INFIN  -- TT input wait completed
2          ;-----
3          ; INFIN -- Routine to add the user whose line index # is
4          ; in R1 to the end of the TTFN queue if the user is
5          ; currently waiting for input.
6          ;
7          ; Inputs:
8          ; R1 = Virtual line number of job being activated.
9          ;
10         015500  010246  INFIN:  MOV      R2,-(SP)
11         ;
12         ; Determine which state to put job in
13         ;
14         015502  004737  017262'      CALL     SLCHK      ;Is program doing single character activation?
15         015506  103003                BCC     2$          ;Br if not
16         015510  012700  000000G      MOV     #S$TTSC,R0  ;If yes, put job in special high-prio state
17         015514  000402                BR      3$
18         015516  012700  000000G  2$:   MOV     #S$TTFN,R0  ;Put user in normal input-done state
19         ;
20         ; If job is already in at least this high a priority state, leave it alone
21         ;
22         015522  016102  000000G  3$:   MOV     LSTATE(R1),R2 ;Get job's current execution state
23         015526  020200                CMP     R2,R0       ;Is job already at that high a priority?
24         015530  101416                BLOS   9$          ;Br if yes -- Don't change its priority
25         ;
26         ; Job's current execution priority is lower than that associated
27         ; with receiving an activation character.
28         ; If job is currently waiting for an activation character, boost its
29         ; priority to get it running again.
30         ;
31         015532  020227  000000G      CMP     R2,#S$INWT  ;Is job waiting for TT input?
32         015536  001003                BNE    4$          ;Br if not
33         015540  005061  000000G      CLR    LRDTIM(R1)  ;Clear TT read time-out counter
34         015544  000406                BR     1$          ;Go boost priority of job to get it running
35         ;
36         ; Job is not waiting for TT input.
37         ; If job is in any other wait state, leave it alone.
38         ;
39         015546  020227  000000G  4$:   CMP     R2,#S$#RUN  ;Is job in a wait state?
40         015552  103005                BHIS   9$          ;Br if yes -- Leave it alone
41         ;
42         ; Job is not in a wait state.
43         ; If job is still classified as "interactive", give it a prio boost.
44         ;
45         015554  005761  000000G      TST    LITIME(R1)  ;Is job still interactive?
46         015560  001402                BEQ    9$          ;Br if not
47         ;
48         ; Boost priority of job
49         ;
50         015562  004737  000000G  1$:   CALL    ENQTL      ;Requeue job at tail of high-prio queue
51         ;
52         ; Finished
53         ;
54         015566  012602  9$:   MOV     (SP)+,R2
55         015570  000207                RETURN

```

KILCHR -- Delete a character from input buffer

```

1          .SBTTL  KILCHR -- Delete a character from input buffer
2          ;-----
3          ; KILCHR is called to delete a character
4          ; from the input buffer.
5          ;
6          ; Inputs:
7          ; R1 = user index #.
8          ;
9          ; Outputs:
10         ; The C-flag is set on attempt to delete activation char.
11         ; R0 = The character that was deleted.
12         ;
13 015572 010246 KILCHR: MOV     R2,-(SP)
14 015574 010346      MOV     R3,-(SP)
15         ;
16         ; See if there are any characters in the TT input ring buffer
17         ;
18 015576 005761 000000G      TST     LINCNT(R1)      ; ANY CHARS IN BUFFER NOW?
19 015602 001424      BEQ     3#                ; IF NOT NOTHING TO DELETE
20         ;
21         ; Locate the last character in the TT input buffer
22         ;
23 015604 016102 000000G      MOV     LINNXT(R1),R2    ; Get pointer past last char in buffer
24 015610 005302      DEC     R2                ; Point to last char in buffer
25 015612 020261 000000G      CMP     R2,LINBUF(R1)    ; Did we go past front of buffer?
26 015616 103003      BHIS   1#                ; Br if not
27 015620 016102 000000G      MOV     LINEND(R1),R2   ; Get pointer past right end of buffer
28 015624 005302      DEC     R2                ; Get pointer to last char in buffer
29 015626 010203      1#:    MOV     R2,R3                ; Save pointer to character being deleted
30 015630 004737 016320'      CALL   FETCHR           ; Get the last character in the buffer
31         ;
32         ; Delete the last character unless it is an activation character
33         ;
34 015634 032700 000000G      BIT     #ACFLAG,R0     ; Is this character an activation char?
35 015640 001005      BNE   3#                ; Br if yes
36 015642 010302      MOV     R3,R2                ; Get pointer to character to delete
37 015644 004737 016612'      CALL   DELCHR           ; Delete the character
38         ;
39         ; We successfully deleted a character
40         ;
41 015650 000241      2#:    CLC                ; SAY CHAR WAS DELETED
42 015652 000401      BR     9#                ; Finished
43         ;
44         ; There are no characters to delete
45         ;
46 015654 000261      3#:    SEC                ; Signal that there are no chars to delete
47         ;
48         ; Finished
49         ;
50 015656 012603      9#:    MOV     (SP)+,R3
51 015660 012602      MOV     (SP)+,R2
52 015662 000207      RETURN

```

INCHR -- Store and echo a character

```
1  
2  
3  
4  
5  
6  
7 015664 004737 015702'  
8 015670 103403  
9 015672 010500  
10 015674 004737 017040'  
11 015700 000207
```

```
      .SBTTL  INCHR  -- Store and echo a character  
      ;-----  
      ; INCHR is called to store and echo the  
      ; input character in R5.  
      ; the user's index # must be in R1.  
      ;  
INCHR: CALL  STRCHR      ; STORE THE CHARACTER  
        BCS  1$          ; BR IF INPUT BUFFER OVERFLOW  
        MOV  R5,R0      ; GET CHARACTER INTO R0 FOR ECHO  
        CALL ECHO       ; ECHO THE CHARACTER  
1$:    RETURN
```



STRCHR -- Store a character into TT buffer

```

1          .SBTTL  STRCHR -- Store a character into TT buffer
2          ;-----
3          ; STRCHR is called to store the character
4          ; in R5 into the input buffer.
5          ; The C-flag is set if the buffer overflows.
6          ; R1 = User index number.
7          ; All registers are preserved.
8          ;
9 015702   010046   STRCHR:  MOV     R0,-(SP)
10 015704   010246       MOV     R2,-(SP)
11 015706   010346       MOV     R3,-(SP)
12          ;
13          ; Keep track of cursor position of 1st character on the line
14          ;
15 015710   032761   000000C 000000G   BIT     <#1STCH!#DODFR>,LSW3(R1);1ST CHAR AFTER ACTIVATION?
16 015716   001006       BNE     4$           ;BRANCH IF NOT FIRST
17 015720   052761   000000G 000000G   BIS     #1STCH,LSW3(R1);SAY WE'VE GOT A CHAR
18 015726   116161   000000G 000000G   MOVB   LCOL(R1),LINCUR(R1);SAVE CURSOR POSITION
19          ;
20          ; See if buffer is about to overflow
21          ;
22 015734   016103   000000G   4$:     MOV     LINSPC(R1),R3 ;Get # free bytes in TT input ring buffer
23 015740   020327   000012       CMP     R3,#10.      ;Got at least 10 free bytes?
24 015744   103022       BHIS   7$           ;Br if yes
25 015746   005761   000000G   TST     LACTIV(R1)   ;Gotten an activation char yet?
26 015752   001403       BEQ   8$           ;Br if not
27 015754   052761   000000G 000000G   BIS     ##XSTOP,LSW6(R1);Don't move any more chars out of silo
28 015762   020327   000006   8$:     CMP     R3,#6.      ;Got at least 6 bytes left?
29 015766   103011       BHIS   7$           ;Br if yes -- accept the character
30 015770   020327   000002       CMP     R3,#2.      ;Is the buffer as full as we will allow?
31 015774   101470       BLOS   5$           ;Br if yes -- Discard the character
32 015776   120527   000000G   CMPB   R5,#CR       ;Is this a carriage return?
33 016002   001403       BEQ   7$           ;Br if yes -- accept it
34 016004   032705   000000G   BIT     #ACFLAG,R5  ;Is this an activation character?
35 016010   001462       BEQ   5$           ;Br if not -- Reserve last 4 bytes for act chr
36          ;
37          ; Determine position where char is to be stored in input ring buffer
38          ; and update pointer to next free position.
39          ;
40 016012   7$:     DISABL           ;;;Disable interrupts
41 016020   016102   000000G   MOV     LINNXT(R1),R2 ;;;Get pointer to next position in TT buffer
42 016024   010203       MOV     R2,R3       ;;;
43 016026   005203       INC     R3           ;;;Compute address of next char position
44 016030   020361   000000G   CMP     R3,LINEND(R1) ;;;Did we go past end of buffer?
45 016034   103402       BLO   1$           ;;;Br if not
46 016036   016103   000000G   MOV     LINBUF(R1),R3 ;;;Wrap around to front of buffer
47 016042   010361   000000G   1$:     MOV     R3,LINNXT(R1) ;;;Save pointer to where next char goes
48 016046       ENABL           ;Enable interrupts
49          ;
50          ; Store character into buffer
51          ;
52 016054   010500       MOV     R5,R0       ;Get character to store
53 016056   004737   016462'   CALL   INCHR        ;Store the character
54          ;
55          ; Update character counts.
56          ;
57 016062   005361   000000G   DEC     LINSPC(R1)  ;Reduce free space count for buffer

```

STRCHR -- Store a character into TT buffer

```

58 016066 005261 000000G          INC      LINCNT(R1)          ;Count another char in input buffer
59                                     ;
60                                     ; Check for storing activation character.
61                                     ;
62 016072 032705 000000G          BIT      #ACFLAG,R5          ; IS THIS AN ACTIVATION CHAR?
63 016076 001414                   BEQ      3$                  ; BRANCH IF NOT
64 016100 005261 000000G          INC      LACTIV(R1)          ; Count another pending activ char
65 016104 016161 000000G 000000G  MOV      LINNXT(R1),LSTACT(R1);REMEMBER POS OF CHAR
66 016112 032761 000000G 000000G  BIT      #$DODFR,LSW3(R1); DOING DEFERED ECHOING?
67 016120 001003                   BNE      3$                  ; BRANCH IF YES
68 016122 042761 000000G 000000G  BIC      #$1STCH,LSW3(R1); SAY NO CHARS AFTER ACTIVATION
69                                     ;
70                                     ; See if we need to trigger a completion routine for TT input
71                                     ;
72 016130 005761 000000G 3$:     TST      LTTCR(R1)          ; Does job want TT input compl routine?
73 016134 001403                   BEQ      10$                 ; Br if not
74 016136                   DCALL   TTCPL              ; Trigger completion routine
75                                     ;
76                                     ; Finished
77                                     ;
78 016144 000241 10$:     CLC                          ; SIGNAL NO BUFFER OVERFLOW
79 016146 012603 6$:     MOV      (SP)+,R3
80 016150 012602                   MOV      (SP)+,R2
81 016152 012600                   MOV      (SP)+,R0
82 016154 000207                   RETURN
83                                     ;
84                                     ; Signal buffer overflow
85                                     ;
86 016156 052761 000000G 000000G 5$:     BIS      #$TTERR,LSW4(R1);REMEMBER THAT AN ERROR OCCURED
87 016164 012700 000000G          MOV      #BELL,R0          ; Get a bell character
88 016170 004737 017100'          CALL    ECHO2              ; Ring the bell to signal buffer nearly full
89 016174 000261                   SEC                          ; SIGNAL OVERFLOW
90 016176 000763                   BR      6$

```

STRACT -- Store activation character

```

1
2
3
4
5
6
7
8
9
10
11
12 016200 052705 000000G
13 016204 004737 015702'
14 016210 042705 000000G
15 016214 032761 000000G 000000G
16 016222 001004
17 016224 005061 000000G
18 016230 005061 000000G
19
20
21
22 016234 032761 000000G 000000G 3$:
23 016242 001403
24 016244 052761 000000G 000000G
25
26
27
28 016252 004737 015500' 1$:
29 016256 000207

```

.SBTTL STRACT -- Store activation character

---

```

; STRACT is called to store an activation character
; which is contained in R5. The high order bit of
; the word (ACFLAG) is set on to indicate to GETCHR
; that the character is an activation character.
; After the character is stored the user is activated.
; When called, R5 must contain the character to be stored.
; R1 must contain the user index #.
; All registers are preserved.
;
STRACT: BIS #ACFLAG,R5 ;SET ACTIVATION FLAG WITH CHAR
CALL STRCHR ;STORE THE CHARACTER
BIC #ACFLAG,R5 ;RESET THE FLAG
BIT ##DBGMD,LSW6(R1); IS DEBUGGER DOING I/O NOW?
BNE 3$ ;BR IF YES - DON'T RESET ACTIVATION INFO
CLR LAFSIZ(R1) ;RESET FIELD-WIDTH ACTIVATION
CLR LFWLIM(R1) ;CLEAR FIELD WIDTH LIMIT
;
; See if we should begin deferred char echoing.
;
3$: BIT ##DEFER,LSW2(R1); DEFERRED MODE WANTED?
BEQ 1$ ;BRANCH IF NOT
BIS ##DODFR,LSW3(R1); BEGIN DEFERRED ECHOING
;
; Activate the job
;
1$: CALL INFIN ;ACTIVATE THE USER
RETURN

```

STRSNG -- Store char with single-character input

```

1          .SBTTL  STRSNG -- Store char with single-character input
2          ;-----
3          ; STRSNG is called to store a character received while in
4          ; single character activation mode.
5          ;
6          ; Inputs:
7          ;   R1 = Job index number.
8          ;   R5 = Character received.
9          ;
10         016260 010546 STRSNG: MOV      R5,-(SP)      ;Save input character
11         ;
12         ; Store the activation character
13         ;
14         016262 052705 000000G  BIS      #ACFLAG,R5      ;Set activation flag with char
15         016266 004737 015702'  CALL    STRCHR      ;Store the character
16         ;
17         ; See if we need to begin deferred echo mode
18         ;
19         016272 032761 000000G 000000G  BIT      #$DEFER,LSW2(R1);Deferred mode wanted?
20         016300 001403  BEQ      4$              ;Branch if not
21         016302 052761 000000G 000000G  BIS      #$DODFR,LSW3(R1);Begin deferred echoing
22         ;
23         ; Activate the job
24         ;
25         016310 004737 015500'  4$:    CALL    INFIN      ;Activate the job
26         ;
27         ; Finished
28         ;
29         016314 012605  9$:    MOV      (SP)+,R5      ;Recover the original character
30         016316 000207  RETURN

```

FETCHR -- Fetch next char from TT input ring buffer

```

1          .SBTTL  FETCHR -- Fetch next char from TT input ring buffer
2          ;-----
3          ;  FETCHR is called to fetch the next character from the TT input
4          ;  ring buffer.
5          ;
6          ;  Inputs:
7          ;  R1 = Line index number.
8          ;  R2 = Pointer to character in TT input ring buffer.
9          ;
10         ;  Outputs:
11         ;  C-flag cleared ==> A char was gotten.
12         ;  C-flag set   ==> No more chars in TT input ring buffer.
13         ;  R0 = Character gotten (if C-flag cleared).
14         ;  The ACFLAG flag is set in R0 if the char is an activation char.
15         ;  R2 = Updated to point to next character.
16         ;
17 016320 010446  FETCHR: MOV     R4,-(SP)
18 016322 010546          MOV     R5,-(SP)
19         ;
20         ;  See if there is another character in the ring buffer
21         ;
22 016324 020261 000000G      CMP     R2,LINNXT(R1)  ;Are there any more chars in the buffer?
23 016330 001450          BEQ     B$                ;Br if not
24         ;
25         ;  Compute index into bit vector that indicates if character is an
26         ;  activation character.
27         ;
28 016332 010204          MOV     R2,R4                ;Get pointer to character
29 016334 166104 000000G      SUB     LINBUF(R1),R4  ;Get character index into buffer
30 016340 073427 177775      ASHC   #-3,R4          ;Get byte index into vector in R2
31 016344 072527 177763      ASH    #-13,R5         ;Right justify bit-within-byte index
32 016350 042705 177770      BIC    #^C7,R5        ;Clear possible sign extension from shift
33 016354 066104 000000G      ADD     LINEND(R1),R4 ;Get address of byte with flag bit
34         ;
35         ;  Get next character from TT ring buffer.
36         ;
37 016360          DISABL          ;;;** Disable interrupts **
38 016366          TTMAP          ;;;Map to TT buffer area
39 016402 005000          CLR     R0                ;;;Initially clear R0
40 016404 152200          BISB   (R2)+,R0        ;;;Get character without sign extension
41 016406 136514 000340'    BITB   BITMSK(R5),(R4) ;;;Is this an activation character?
42 016412 001402          BEQ     1$                ;;;Br if not
43 016414 052700 000000G    BIS    #ACFLAG,R0     ;;;Remember this is an activation character
44 016420          1$: UNMAP          ;;;Restore mapping
45 016426          ENABL          ;;;** Enable interrupts **
46         ;
47         ;  See if we need to wrap around to front of ring buffer
48         ;
49 016434 020261 000000G    3$:  CMP     R2,LINEND(R1) ;Did we just go past end of ring buffer?
50 016440 103402          BLO     4$                ;Br if not
51 016442 016102 000000G    MOV     LINBUF(R1),R2 ;Wrap around to front of ring buffer
52         ;
53         ;  We got a character
54         ;
55 016446 000241          4$:  CLC          ;Signal that we got a character
56 016450 000401          BR     9$
57         ;

```

```
58 ; There are no more characters in the buffer
59 ;
60 016452 000261 B#: SEC ;Signal that there are no more characters
61 ;
62 ; Finished
63 ;
64 016454 012605 9#: MOV (SP)+, R5
65 016456 012604 MOV (SP)+, R4
66 016460 000207 RETURN
```

INSCHR -- Insert character into TT input ring buffer

```

1          .SBTTL  INSCHR -- Insert character into TT input ring buffer
2          ;-----
3          ; This routine is called to insert a character into a specified position
4          ; in the TT input-character buffer for a line.  If there is an existing
5          ; character in the position, it is overwritten.
6          ;
7          ; Inputs:
8          ;   R0 = Character to be stored (optionally with ACFLAG flag).
9          ;   R1 = Line index number.
10         ;   R2 = Pointer to position in buffer where char is to be stored
11         ;
12         ; Outputs:
13         ;   R2 = Pointer to next character position in buffer.
14         ;
15 016462 010446  INSCHR: MOV     R4, -(SP)
16 016464 010546          MOV     R5, -(SP)
17         ;
18         ; Compute pointer into parallel bit vector with activation-character flags
19         ;
20 016466 010204          MOV     R2, R4          ;Get character buffer pointer
21 016470 166104 000000G  SUB     LINBUF(R1), R4      ;Compute byte index into buffer
22 016474 073427 177775  ASHC    #-3, R4           ;Get byte index in R4
23 016500 072527 177763  ASH     #-13, R5          ;Right justify bit-within-byte index
24 016504 042705 177770  BIC     #^C7, R5         ;Clear possible sign extension
25 016510 116505 000340'  MOVVB  BITMSK(R5), R5     ;Get bit mask
26 016514 066104 000000G  ADD     LINEND(R1), R4    ;Get address of byte with activation flag
27         ;
28         ; Store character into buffer and set or clear the activation-character flag
29         ;
30 016520          DISABL          ;** Disable interrupts **
31 016526          TTMAP          ;Map to TT buffer area
32 016542 110022  MOVVB    R0, (R2)+      ;Store character into buffer
33 016544 140514  BICB    R5, (R4)          ;Clear the activation-character flag
34 016546 032700 000000G  BIT     #ACFLAG, R0     ;Is this an activation character?
35 016552 001401          BEQ     2$,          ;Br if not
36 016554 150514  BISB    R5, (R4)          ;Set the activation-character flag
37 016556          2$: UNMAP          ;Restore mapping
38 016564          ENABL          ;** Enable interrupts **
39         ;
40         ; See if we need to wrap around buffer pointer
41         ;
42 016572 020261 000000G  CMP     R2, LINEND(R1)   ;Do we need to wrap around to buffer front?
43 016576 103402          BLO     9$,          ;Br if not
44 016600 016102 000000G  MOV     LINBUF(R1), R2   ;Wrap around to front of buffer
45         ;
46         ; Finished
47         ;
48 016604 012605 9$:  MOV     (SP)+, R5
49 016606 012604          MOV     (SP)+, R4
50 016610 000207          RETURN

```

```

1          .SBTTL  DELCHR -- Delete character from TT input ring buffer
2          ;-----
3          ; DELCHR is called to remove a character from the TT input ring buffer.
4          ; If there are other characters in the ring buffer in front of the one
5          ; being deleted, the ring buffer is compressed.
6          ;
7          ; Inputs:
8          ;   R1 = Line index number.
9          ;   R2 = Pointer to character to delete.
10         ;
11         ; Outputs:
12         ;   C-flag set ==> No more characters in buffer.
13         ;   R0 = Deleted character (Same format as FETCHR).
14         ;   R2 = Pointer to character that follows deleted character.
15         ;
16 016612 010346 DELCHR: MOV     R3,-(SP)
17 016614 010546         MOV     R5,-(SP)
18         ;
19         ; Get the character being deleted
20         ;
21 016616 010203         MOV     R2,R3           ; Save original character pointer
22 016620 004737 016320' CALL    FETCHR         ; Get character being deleted
23 016624 103502         BCS     20$           ; Br if no character to delete
24         ;
25         ; See if we are deleting the 1st character in the buffer
26         ;
27 016626 020361 000000G CMP     R3,LINPNT(R1)   ; Is this the 1st char in the buffer?
28 016632 001003         BNE     3$           ; Br if not
29 016634 010261 000000G MOV     R2,LINPNT(R1)   ; Set new pointer to 1st char in buffer
30 016640 000434         BR      8$           ; Go update buffer counts
31         ;
32         ; We are not deleting the 1st character in the buffer.
33         ; If we are deleting a character from the middle of the buffer, we move
34         ; over any following characters to compress the free space.
35         ; If we are deleting the last character in the buffer, no compression
36         ; is necessary.
37         ;
38 016642 010046 3$:    MOV     R0,-(SP)         ; Save character that is being deleted
39 016644 010346         MOV     R3,-(SP)         ; Save pointer to deleted char position
40 016646         4$:    DISABL                ; ** Disable interrupts **
41 016654 020261 000000G CMP     R2,LINNXT(R1)   ; Are we at the end of the chars now?
42 016660 001415         BEQ     5$           ; Br if yes
43 016662         ENABL                ; ** Enable interrupts **
44         ;
45         ; Slide characters over in buffer to fill in gap left by deleted character
46         ;
47 016670 010205         MOV     R2,R5           ; Save pointer to following character
48 016672 004737 016320' CALL    FETCHR         ; Get next character in buffer
49 016676 010246         MOV     R2,-(SP)         ; Save updated pointer
50 016700 010302         MOV     R3,R2           ; Get pointer to pos to store character
51 016702 004737 016462' CALL    INSCHR         ; Reinsert the character
52 016706 010503         MOV     R5,R3           ; Advance insert pointer
53 016710 012602         MOV     (SP)+,R2        ; Get pointer to next char to move over
54 016712 000755         BR      4$           ; Loop until all characters have been moved
55         ;
56         ; We have moved over all characters that followed the deleted one.
57         ; Save new pointer to the end of the characters in the buffer.

```



```

58
59 016714 010361 000000G 5$:      MOV      R3,LINNXT(R1)    ;;;Set new pointer to end of chars in buffer
60 016720                                ENABL                    ;** Enable interrupts **
61 016726 012602                                MOV      (SP)+,R2        ;Get pointer to char following deleted one
62 016730 012600                                MOV      (SP)+,R0        ;Get character being deleted
63
64                                ; Update character counters
65
66 016732 005261 000000G 8$:      INC      LINSPC(R1)      ;Another free char space in buffer
67 016736 005361 000000G                                DEC      LINCNT(R1)      ;One less char in buffer
68 016742 032700 000000G                                BIT      #ACFLAG,R0      ;Is deleted char an activation char?
69 016746 001402                                BEQ      19$             ;Br if not
70 016750 005361 000000G                                DEC      LACTIV(R1)      ;One fewer pending activation chars
71
72                                ; If we sent an XOFF to the terminal,
73                                ; send an XON if the input buffer is nearly empty.
74
75 016754 032761 000000G 000000G 19$:    BIT      ##XSTOP,LSW6(R1);Have we stopped input from silo buffer?
76 016762 001422                                BEQ      16$             ;Br if not
77 016764 026127 000000G 000017                                CMP      LINCNT(R1),#15. ;Is input buffer almost empty?
78 016772 101406                                BLOS    17$             ;Br if yes
79 016774 032700 000000G                                BIT      #ACFLAG,R0      ;Is this an activation character?
80 017000 001413                                BEQ      16$             ;Br if not
81 017002 005761 000000G                                TST     LACTIV(R1)      ;Is this last activation char in buffer?
82 017006 001010                                BNE     16$             ;Br if not
83 017010 042761 000000G 000000G 17$:    BIC     ##XSTOP,LSW6(R1);Reenable input from silo
84 017016 052761 000000G 000000G                                BIS     ##NDICP,LSW10(R1);Say line needs input character servicing
85 017024 005237 000000G                                INC     NEDCDI          ;Say input character processing needed
86
87                                ; We deleted a character
88
89 017030 000241                                16$:    CLC                    ;Signal that we deleted a character
90
91                                ; Finished
92
93 017032 012605                                20$:    MOV      (SP)+,R5
94 017034 012603                                MOV      (SP)+,R3
95 017036 000207                                RETURN

```

```

1
2
3
4
5
6
7
8
9 017040 004737 017166'
10 017044 004737 017422'
11 017050 032761 000000G 000000G
12 017056 001016
13 017060 032761 000000G 000000G
14 017066 001004
15 017070 032761 000000G 000000G
16 017076 001406
17 017100 026127 000000G 000017
18 017106 002402
19 017110 004737 003122'
20 017114 000207
21
22
23
24
25
26
27
28
29
30
31
32 017116 010046
33 017120 112700 000136
34 017124 004737 017040'
35 017130 010500
36 017132 052700 000100
37 017136 004737 017044'
38 017142 112700 000000G
39 017146 004737 017044'
40 017152 112700 000000G
41 017156 004737 017044'
42 017162 012600
43 017164 000207
44
45
46
47
48
49
50
51
52
53 017166 032761 000000G 000000G
54 017174 001411
55 017176 042761 000000G 000000G
56 017204 010046
57 017206 112700 000134

```

```

      .SBTTL  ECHO  -- Echo character to terminal
      -----
      ; Subroutine ECHO is called to echo the character in R0.
      ; User index must be in R1.
      ; ECHO1 does not check for rubout character echoing.
      ; ECHO2 does not check for deferred character echoing.
      ; All registers are preserved.
      ;
      ECHO:  CALL    RBEND                ; TERMINATE ANY RUBOUT FIELD
      ECHO1: CALL    CVTLC                ; CONVERT LOWER CASE CHARS TO UPPER CASE
            BIT     #$DODFR,LSW3(R1)    ; ARE WE DEFERRING ECHO NOW?
            BNE    ECHOR                ; BRANCH IF WE ARE
            BIT     #$ECHO,LSW2(R1)    ; IS CHARACTER ECHOING WANTED?
            BNE    ECHO2                ; BR IF YES
            BIT     #$DBGMD,LSW6(R1)    ; IS A DEBUGGER USING TERMINAL NOW?
            BEQ    ECHOR                ; BR IF NOT
      ECHO2: CMP     LOTSPC(R1),#15     ; ROOM TO ECHO CHAR?
            BLT    ECHOR                ; BRANCH IF NOT
            CALL   PUTCH2                ; PUT CHAR IN OUTPUT BUFFER
      ECHOR: RETURN

```

```

      .SBTTL  ECHOCTL -- Echo a control character
      -----
      ; ECHOCTL is called to echo certain control characters
      ; such as ctrl-C and ctrl-U.  When called R5 must
      ; contain the control character.  The control character
      ; is converted to a printing ascii char and printed
      ; following an up arrow and before a cr-lf.
      ; When called R1 must contain the line index.
      ; all registers are preserved.
      ;
      ECHOCTL: MOV    R0,-(SP)
            MOVB   #136,R0              ; ECHO '^'
            CALL   ECHO
            MOV    R5,R0                ; GET CONTROL CHARACTER
            BIS   #100,R0               ; CONVERT TO PRINTING CHAR
            CALL   ECHO1                ; PRINT CONTROL CHAR
            MOVB   #CR,R0               ; PRINT CR-LF
            CALL   ECHO1
            MOVB   #LF,R0
            CALL   ECHO1
            MOV    (SP)+,R0
            RETURN

```

```

      .SBTTL  RBEND  -- Terminate rubout sequence
      -----
      ; RBEND is called to terminate any current rubout field.
      ; If a rubout field is in progress RBEND puts out
      ; a back slash and terminates the rubout state.
      ; When called R1 must contain the user index #.
      ; All registers are preserved.
      ;
      RBEND: BIT     #$RBOU,LSW3(R1)    ; ARE WE IN A RUBOUT FIELD?
            BEQ    1$                  ; BRANCH IF NOT
            BIC   #$RBOU,LSW3(R1)    ; CLEAR RUBOUT STATE
            MOV    R0,-(SP)
            MOVB   #'\,R0              ; OUTPUT BACK SLASH

```

58	017212	004737	017044'	CALL	ECHO1
59	017216	012600		MOV	(SP)+,R0
60	017220	000207	1#:	RETURN	

```

1          .SBTTL  SCACHK -- Check for single-character activation
2          ;-----
3          ; SCACHK is called to determine if this job is in single character
4          ; activation mode.
5          ;
6          ; Inputs:
7          ; R1 = Job index number.
8          ;
9          ; Outputs:
10         ; C-flag set if in single character activation mode.
11         ;
12 017222 032761 000000G 000000G SCACHK: BIT    #SPCTTY,LJSW(R1); Does program want single char input?
13 017230 001412          BEQ    1$          ; Br if not
14 017232 032761 000000G 000000G          BIT    ##CHACT,LSW5(R1); Is single character activation enabled?
15 017240 001406          BEQ    1$          ; Br if not
16 017242 032761 000000G 000000G          BIT    ##DBGMD,LSW6(R1); Is a debugger using terminal now?
17 017250 001002          BNE    1$          ; Br if yes
18         ;
19         ; Job is in single character activation mode
20         ;
21 017252 000261          SEC          ; Signal that single char activation wanted
22 017254 000401          BR     9$
23         ;
24         ; Job does not want single char activation
25         ;
26 017256 000241 1$:    CLC          ; Signal no single char activation
27         ;
28         ; Finished
29         ;
30 017260 000207 9$:    RETURN

```

```

1          .SBTTL  SLCHK  -- Check for single line editor mode
2          ;-----
3          ; SLCHK is called to determine if this job is in either single character
4          ; activation mode or if the input is going to the single line editor.
5          ;
6          ; Inputs:
7          ;   R1 = Job index number
8          ;
9          ; Outputs:
10         ;   C-flag set if in single-character-activation or SL mode.
11         ;   All registers are preserved.
12         ;
13 017262 032761 000000G 000000G SLCHK:  BIT    #SPCTTY,LJSW(R1); Does program want single char input?
14 017270 001411                BEQ    1$      ;Br if not
15 017272 032761 000000G 000000G        BIT    #CHACT,LSW5(R1); Is single character activation enabled?
16 017300 001437                BEQ    2$      ;Br if not
17 017302 032761 000000G 000000G        BIT    #DBGMD,LSW6(R1); Is a debugger using terminal now?
18 017310 001033                BNE    2$      ;Br if yes
19 017312 000430                BR     3$      ;We are in single character mode
20         ;
21         ; We are not in single character activation mode.
22         ; See if we are in Single Line Editor mode.
23         ;
24 017314 032761 000000G 000000G 1$:  BIT    #SLON,LSW7(R1) ; Is SL enabled for this line?
25 017322 001426                BEQ    2$      ;Br if not
26 017324 032761 000000G 000000G        BIT    #DISSLE,LJSW(R1); Did program disable SL?
27 017332 001022                BNE    2$      ;Br if yes
28 017334 032761 000000C 000000G        BIT    #<#ODTMD!#HITTY>,LSW4(R1); Are we in ODT or high efficiency?
29 017342 001016                BNE    2$      ;Br if yes
30 017344 032761 000000G 000000G        BIT    #VTESC,LSW5(R1); VTxxx activation enabled?
31 017352 001012                BNE    2$      ;Br if yes
32 017354 032761 000000G 000000G        BIT    #GTLIN,LSW4(R1); Is a .GTLIN being done?
33 017362 001004                BNE    3$      ;Br if yes
34 017364 032761 000000G 000000G        BIT    #SLTTY,LSW7(R1); Is SL enabled for TTY input?
35 017372 001402                BEQ    2$      ;Br if not
36         ;
37         ; We are in single line activation or single line editor mode.
38         ;
39 017374 000261                3$:  SEC                    ;Signal single character mode
40 017376 000401                BR     9$
41         ;
42         ; We are not in single character mode.
43         ;
44 017400 000241                2$:  CLC                    ;Signal not single character mode
45         ;
46         ; Finished
47         ;
48 017402 000207                9$:  RETURN

```

SCACHR -- Handle single-character activation characters

```

1          .SBTTL  SCACHR -- Handle single-character activation characters
2          ;-----
3          ; SCACHR is called to check to see if the program is in single-character
4          ; activation mode.  If yes then the current character is passed to the
5          ; program as an activation character.
6          ; If not then the carry-flag is cleared on return.
7          ;
8          ; Inputs:
9          ;   R1 = Virtual line index number.
10         ;   R5 = Current input character.
11         ;
12         ; Outputs:
13         ;   C-flag set      ==> In single-char activation mode. Char processed.
14         ;   C-flag cleared ==> Not in single character activation mode.
15         ;
16 017404 004737 017262' SCACHR: CALL    SLCHK          ;Are we in single-char activation mode?
17 017410 103003          BCC     9$           ;Br if not
18         ;
19         ; We are in single-character activation mode.
20         ; Pass the character to the program as an activation char.
21         ;
22 017412 004737 016260'          CALL    STRSNG        ;Store character and activate
23 017416 000261          SEC     9$           ;Say character was processed by us
24         ;
25         ; Finished
26         ;
27 017420 000207          9$:   RETURN

```

CVTLC -- Convert lower-case chars to upper-case

```

1          .SBTTL  CVTLC  -- Convert lower-case chars to upper-case
2          ;-----
3          ; CVTLC is called to see if TT input characters entered
4          ; in lower case should be translated to upper case.
5          ; When called, R0 must contain the character to be tested.
6          ; R1 must contain the user line index #.
7          ; CVTLC checks to see if set-lc has been done and if the
8          ; bit is set in the JSW allowing lower case characters.
9          ; On return, the resulting character is returned in R0.
10         ; All other registers are preserved.
11         ;
12 017422 032761 000000G 000000G CVTLC:  BIT    #LC,LSW2(R1)  ; WAS "SET TT LC" DONE?
13 017430 001404                BEQ    1$              ; BR IF NOT
14 017432 032761 000000G 000000G        BIT    #LCBIT,LJSW(R1) ; IS LC-BIT SET IN JSW?
15 017440 001014                BNE    2$              ; BR IF YES (LC OK)
16         ;
17         ; Translate lower case to upper case
18         ;
19 017442 010046                1$:   MOV    R0,-(SP)      ; SAVE ORIGINAL CHARACTER VALUE
20 017444 042700 000000G        BIC    #ACFLAG,R0    ; CLEAR ACTIVATION FLAG
21 017450 020027 000141        CMP    R0,#141      ; LC('A')
22 017454 002405                BLT    3$          ; BR IF NOT LOWER-CASE LETTER
23 017456 020027 000172        CMP    R0,#172     ; LC('Z')
24 017462 101002                BHI    3$          ; BR IF NOT LOWER-CASE LETTER
25 017464 042716 000040        BIC    #40,(SP)   ; CONVERT LOWER-CASE LETTER TO UPPER-CASE
26 017470 012600                3$:   MOV    (SP)+,R0 ; GET POSSIBLY CONVERTED LETTER BACK TO R0
27 017472 000207                2$:   RETURN

```

SIGWAT -- Signal virtual line wait condition

```

1          .SBTTL SIGWAT -- Signal virtual line wait condition
2          ;-----
3          ; SIGWAT IS CALLED TO SIGNAL THE USER THAT ONE OF HIS
4          ; VIRTUAL LINES IS ENTERING A WAIT CONDITION.
5          ; IF THE LINE ENTERING THE WAIT STATE IS NOT THE ONE
6          ; WHICH IS CURRENTLY ASSOCIATED WITH THE TERMINAL A BELL
7          ; IS SENT TO THE USER'S TERMINAL.
8          ;
9          ; Inputs:
10         ; R1 = Job index of job entering wait condition.
11         ; All registers are preserved.
12         ;
13 017474 010046 SIGWAT: MOV      RO,-(SP)
14 017476 010146         MOV      R1,-(SP)
15         ;
16         ; Only signal if the job that is entering the wait state is not
17         ; currently connected to the terminal.
18         ;
19 017500 016100 000000G MOV      LNPRIM(R1),RO ;GET PRIMARY LINE #
20 017504 026001 000000G CMP      LNMAP(RO),R1  ;IS THE LINE CONNECTED TO TERM?
21 017510 001415         BEQ      9$ ;IF YES THEN NO NEED TO SIGNAL
22         ;
23         ; If we have already signaled that job is in a wait state, don't
24         ; signal again until user reconnects to this job.
25         ;
26 017512 032761 000000G 000000G BIT      ##VBELL,LSW9(R1);Have we already signaled wait state?
27 017520 001011         BNE      9$ ;Br if yes
28 017522 052761 000000G 000000G BIS      ##VBELL,LSW9(R1);Set flag saying we have signaled
29         ;
30         ; Send bell to signal wait condition
31         ;
32 017530 016001 000000G MOV      LNMAP(RO),R1  ;GET CURRENTLY CONNECTED LINE #
33 017534 112700 000000G MOVB    #BELL,RO ;SEND BELL AS SIGNAL CHARACTER
34 017540 004737 005722' CALL    TRYCHR
35         ;
36         ; Finished
37         ;
38 017544 012601 9$: MOV      (SP)+,R1
39 017546 012600         MOV      (SP)+,RO
40 017550 000207         RETURN
41         ;
42         .SBTTL SIGBRK -- Signal program that Break character was received
43         ;-----
44         ; SIGBRK is called to signal a program that a Break character was received.
45         ; If the program has requested notification of Break character reception,
46         ; an asynchronous completion routine request is queued for the program.
47         ;
48         ; Inputs:
49         ; R1 = Virtual line number.
50         ;
51 017552 010446 SIGBRK: MOV      R4,-(SP)
52 017554 016104 000000G MOV      LBRKCQ(R1),R4 ;DOES USER WANT NOTIFICATION OF BREAK?
53 017560 001404         BEQ      1$ ;BR IF NOT
54 017562 005061 000000G CLR      LBRKCQ(R1) ;SAY THAT BREAK QUEUE ELEMENT HAS BEEN USED UP
55 017566 004737 000000G CALL    QCOMPL ;QUEUE COMPLETION ROUTINE FOR THE JOB
56 017572 012604 1$: MOV      (SP)+,R4
57 017574 000207         RETURN

```



TSTTY -- TSX Terminal I/O routi MACRO V05.04 Monday 25-Jan-88 08:31 Page 81-1  
SIGBRK -- Signal program that Break character was received

58 000001 .END  
Errors detected: 0

\*\*\* Assembler statistics

Work file reads: 0  
Work file writes: 0  
Size of work file: 10440 Words ( 41 Pages)  
Size of core pool: 17920 Words ( 70 Pages)  
Operating system: RT-11

Elapsed time: 00:01:09.44  
DK: TSTTY, LP: TSTTY=DK: TSTTY. MAC/C/N: SYM

\$1CTLC	1-55	51-21	53-44	54-40	55-14	55-19	57-29	60-14	61-29	62-16	63-40	64-14
\$1ESC	1-43	50-79	51-20	53-45	54-41	55-20	57-30	60-15	61-30	62-17	63-30	63-33
	63-36	64-15	65-21	65-48	65-62							
\$1STCH	1-48	47-25	55-70	60-40	70-15	70-17	70-68					
\$8BIT	1-52	12-31										
\$AUTO	1-34	49-32										
\$CCLRN	1-76	35-38										
\$CFABT	1-86	9-120	9-122									
\$CFALL	1-51	39-75	39-82	41-14	43-35							
\$CFCCCL	1-87	9-95	43-12									
\$CFDCC	1-87	9-30	9-97									
\$CFKIL	1-87	55-66										
\$CFOPN	1-51	43-22	43-25	43-123								
\$CFSOT	1-54	13-27	39-54	39-61	39-68	43-35						
\$CHACT	1-44	28-23	28-30	33-75	34-11	57-11	61-11	63-13	77-14	78-15		
\$CTRLC	1-43	12-22	19-17	33-14	55-65							
\$CTRLD	1-40	56-15										
\$CTRLQ	1-48	9-113	13-34	18-32	58-9	58-17	58-25					
\$CTRLS	1-49	55-63										
\$CTRLW	1-48	50-77	50-80	50-85	50-87	50-91						
\$DBGBK	1-56	56-17										
\$DBGMD	1-82	7-22	35-32	38-25	50-28	51-49	51-63	53-34	55-40	60-92	64-26	64-93
	71-15	76-15	77-16	78-17								
\$DBKMN	1-40	56-12										
\$DEAD	1-57	49-27										
\$DEBUG	1-56	56-15										
\$DEFER	1-46	25-49	25-56	71-22	72-19							
\$DETC	1-43	13-21	19-15	19-17	24-13	26-10	27-13	28-5	28-14	29-13	33-34	46-7
\$DILUP	1-42	49-19										
\$DISCN	1-42	12-22	19-17	33-14	33-36	46-9						
\$DODFR	1-49	25-57	35-12	47-16	47-54	55-61	60-25	70-15	70-66	71-24	72-21	76-11
\$DOOFF	1-42	7-19	33-29									
\$ECHO	1-44	25-5	25-12	35-48	36-18	38-10	64-24	76-13				
\$FLAGC	1-54	7-26	7-28	7-34								
\$FORM	1-45	16-46										
\$FORMO	1-89	2-18										
\$GCECO	1-49	25-57	35-14	35-16	36-6	47-24	55-61					
\$GCESC	1-49	33-13	33-95	33-102	38-12							
\$GTLIN	1-51	9-24	9-148	33-49	41-12	78-32						
\$HITTY	1-54	6-37	8-9	8-20	12-16	28-16	33-45	33-85	50-57	78-28		
\$INKMN	1-50	9-56	9-71	9-135	43-42	43-49	43-140	56-10				
\$LC	1-46	25-31	25-42	80-12								
\$LOFCF	1-59	7-17	33-27	43-36	51-15	55-30						
\$NDICP	1-43	46-18	75-84									
\$NOIN	1-47	7-21	12-24	43-38	46-29	55-34						
\$NOINT	1-45	33-58	46-35									
\$NOLF	1-81	30-19	30-26	38-23	53-36							
\$NOOUT	1-46	12-38	20-74	20-81	34-41							
\$NOVLN	1-46	25-24										
\$NOWIN	1-37	17-17	43-52	43-102								
\$NOWTT	1-39	6-17	7-24	28-37								
\$NTGCC	1-41	9-39	9-43	9-99								
\$ODTMD	1-51	33-45	51-32	53-22	54-24	66-12	78-28					
\$PWKEY	1-37	52-16										
\$QUIET	1-50	9-104	35-21	35-46	36-16	39-53	39-61	39-68				
\$RBOU	1-48	60-24	64-34	64-38	76-53	76-55						



CFSTRT	9-146	25-83	42-29#	43-114								
CFTEST	7-15	27-16	29-18	35-4	35-40	35-44	36-14	39-14	41-10#			
CFTND	41-11	41-15	41-18#									
CFTST1	41-12#											
CHKABT	1-73	2-35	2-38	2-56	2-59	4-19	4-22	11-28	11-31	31-15	39-97	46-31
CHKODT	51-34	66-12#										
CHNADR	1-90	2-15*	3-14	3-16*	3-33*							
CKCW	50-29	50-65	50-67	50-69	50-75#							
CKHIIN	50-50	50-57#										
CKICTL	50-88	50-90	50-101#									
CKSPAC	50-34	50-36	50-42#									
CKVTAC	50-70	65-14#										
CKVTES	50-43	50-58	50-64#									
CMDA	22-4	23-5#										
CMDB	1-28	22-5	23-17#									
CMDC	1-28	22-6	23-24#									
CMDD	22-7	24-5#										
CMDE	1-28	22-8	25-5#									
CMDF	1-28	22-9	25-12#									
CMDG	1-28	22-10	25-18#									
CMDH	1-28	22-11	25-24#									
CMDI	1-28	22-12	25-31#									
CMDJ	1-29	22-13	25-42#									
CMDK	1-29	22-14	25-49#									
CMDL	1-29	22-15	25-56#									
CMDM	1-29	22-16	25-64#									
CMDN	1-29	22-17	25-71#									
CMDO	1-29	22-18	25-81#									
CMDP	22-19	26-5#										
CMDQ	22-20	27-5#										
CMDR	1-29	22-21	28-5#									
CMDS	1-30	22-22	28-23#									
CMDT	1-30	22-23	28-30#									
CMDU	1-30	22-24	28-37#									
CMDV	22-25	29-5#										
CMDW	1-30	22-26	30-5#									
CMDX	1-30	22-27	30-12#									
CMDY	1-30	22-28	30-19#									
CMDZ	1-30	22-29	30-26#									
CORUSR	1-61	9-23	10-10	11-15	33-12	43-11						
CR	1-58	9-128	20-44	24-55	29-32	37-10	38-58	44-34	49-37	54-26	54-35	60-57
	70-32	76-38										
CS*EOF	1-89	2-15	3-14	3-16	3-33							
CSEMFO	1-155	1-166#										
CSEMID	1-153	1-164#										
CSEMIL	1-152	1-163#										
CSEMIS	1-157	1-168#										
CSEMIV	1-158	1-169#										
CSEMNF	1-156	1-167#										
CSEPRO	1-154	1-165#										
CSIERR	1-152#	10-36										
CSIMSG	1-25	10-34#										
CTLRTN	52-33	52-42#										
CTRLC	1-64	9-73	9-74	24-22	26-33	35-30	38-47	49-39				
CTRLX	1-64	38-49										
CTRLZ	1-64	1-89	3-24	9-69	9-133	38-45						



GOTCHR	48-32	49-9#												
OTCFCH	9-49	9-91	33-20	39-14#	39-83									
OTCM1	12-64	21-30#												
OTLTTY	1-90	9-32	9-41	9-89										
OTSLCH	1-38	33-53												
OTSPAC	1-28	24-5	24-10#											
HAZEL	1-55	60-50	60-77											
HION	8-7	28-7	28-14#											
HIPUT	6-41	8-23	12-18	19-10#										
HIRTN	8-8	8-10#	8-19	8-25	8-29	8-46	8-60							
ICPCR	52-55	53-9#	54-36											
ICPCTC	52-45	55-9#												
ICPCTD	52-46	56-10#												
ICPCTG	52-49	57-9#												
ICPCTO	52-57	58-9#												
ICPCTR	52-60	59-9#												
ICPCTU	52-63	60-9#												
ICPCTX	52-66	61-9#												
ICPCTZ	52-68	62-9#												
ICPESC	52-69	63-9#												
ICPLF	52-52	54-9#												
ICPRUB	50-107	64-9#												
ILSW2	1-44	49-32												
ILWAIT	33-64	46-7#												
IN#ACT	1-42	9-61	43-143											
INCHR	50-83	51-71	53-32	69-7#										
INFIN	67-10#	71-28	72-25											
INITFL	1-71	49-13												
INITLN	1-92	49-45												
INSCHR	24-47	27-44	70-53	74-15#	75-51									
INTPRI	1-69	1-70	1-79	18-51	18-57	19-47	46-44	47-20	47-58	70-48	73-45	74-38		
	75-43	75-60												
JSWLOC	1-71	25-32	25-35*											
KILCHR	60-105	64-39	64-46	68-13#										
KPAR6	1-86	18-41	18-41*	18-43*	19-40	19-40*	19-42*	73-38	73-38*	73-44*	74-31	74-31*		
	74-37*													
LABTIM	1-45	49-51												
LACTIV	1-61	7-13	8-45	24-48*	27-20	27-45*	33-62	46-33	47-18	55-69*	70-25	70-64*		
	75-70*	75-81												
LAFSIZ	1-69	24-49*	27-46*	27-53*	51-44	55-59*	71-17*							
LBRKCH	1-80	50-33												
LBRKCQ	1-82	81-52	81-54*											
LCBIT	1-68	1-79	25-33	55-51	80-14									
LCOL	1-72	14-41*	16-13*	16-15*	16-21*	16-28	16-40*	47-26	60-42	60-84*	70-18			
LESCHR	1-53	20-28	20-34	34-40*										
LESRTN	1-53	20-18	34-34*											
LF	1-67	9-93	9-123	16-50	20-46	20-54	24-61	37-9	38-21	38-60	44-36	53-16		
	53-26	53-33	76-40											
LF#IN	1-95	35-42	36-12											
LF#OUT	1-95	12-69												
LF#WRT	1-95	45-13												
LFWLIM	1-83	24-50*	27-47*	29-15*	29-55*	51-59	55-58*	71-18*						
LHIPCT	1-93	33-60*	46-37*											
LIFUN	12-60	21-8#												
LINBUF	1-62	68-25	70-46	73-29	73-51	74-21	74-44							
LINCNT	1-66	8-57	24-32	24-57	27-29	29-28	47-33	51-47	51-61	55-54*	68-18	70-58*		







QNSPNX	1-74	2-37	2-58	4-21	11-30							
QUECHR	1-26	14-45	16-8	16-16	16-22	16-32	16-37	16-51	16-55	17-11#	29-51	31-31
R#CFST	1-34	42-15*	42-39*									
R#INST	1-91	9-61	43-17*	43-143								
RBEND	76-9	76-53#										
REGCHR	50-109	51-9#	52-42	52-43	52-44	52-47	52-48	52-50	52-51	52-53	52-54	52-56
	52-58	52-59	52-61	52-62	52-64	52-65	52-67	52-70	52-71	52-72	52-73	56-22
	57-25	61-25										
RSSPAC	1-29	26-5	26-10#									
RUBOUT	1-66	14-39	50-105									
S#*RUN	1-56	67-39										
S#INWT	1-77	46-40	67-31									
S#OTFN	1-60	32-6										
S#OTWT	1-73	31-13	58-18									
S#TTFN	1-81	67-18										
S#TTSC	1-81	67-16										
SCACHK	38-8	50-66	77-12#									
SCACHR	50-81	54-18	55-22	59-11	60-19	62-9	64-19	79-16#				
SECTL	34-24	37-7#	37-12									
SESRTN	34-30	37-22#										
SETC	1-90	7-30	8-62									
SETERR	1-37	6-32										
SETRBF	1-28	23-5	23-10#									
SFWAC	1-29	27-5	27-10#									
SFWL	1-30	29-5	29-10#									
SIGBRK	50-17	50-37	81-51#									
SIGWAT	1-26	31-12	46-28	81-13#								
SILFET	1-35	48-26										
SLCHK	27-18	29-20	51-25	53-13	63-9	67-14	78-13#	79-16				
SPACE	1-78	60-48	64-83									
SPCTTY	1-76	33-41	33-73	34-9	55-51	57-9	61-9	63-11	77-12	78-13		
STOP	1-60	7-20	9-44	9-63	9-137	12-26	19-19	33-16	33-30	33-37	35-52	43-55
	43-145	46-10										
STPFLG	1-71	49-25										
STRACT	1-24	50-52	51-53	53-15	53-40	54-30	55-25	57-20	61-20	62-12	63-35	65-71
	71-12#											
STRCHR	50-59	63-22	63-32	65-65	65-69	69-7	70-9#	71-13	72-15			
STRSNG	51-27	51-39	57-13	61-13	63-15	72-10#	79-22					
SUCF2	1-35	43-53										
TAB	1-66	20-66	37-11	64-48	64-59							
TRNSFL	1-68	12-52	14-20									
TRNSTR	1-38	18-52	19-51									
TRYCHR	31-28#	81-34										
TSTTY	1-6#	1-23										
TTCPL	1-35	70-74										
TTCSCH	1-85	2-30	2-51	11-14	11-25							
TTEOF	3-25	3-33#										
TTFIN	2-46	2-72	3-31	3-38	3-43	4-4#						
TTINCP	1-23	48-15#										
TTREAD	1-25	3-6#										
TTYIN	1-26	7-5#										
TTYOUT	1-26	6-5#										
TTZERO	3-18	3-37#										
UHIMEM	1-85	5-6										
UOTSTR	5-10	9-114	11-9#									
URO	1-90	4-9*	5-5	6-36	7-36*	8-41*	8-44*	8-57*				

VALADB	1-88	2-8	2-14	3-8	3-13	5-9	8-17	8-34	8-40
VINTIO	1-93	33-60	46-37						
VQUAN1	1-81	33-61	46-38						
VT100	1-84	60-50							
VT2007	1-84	60-50							
VT2008	1-84	60-50							
VT52	1-84	60-50	60-74						
VTSLCH	1-68	12-62							
VVLSCH	1-50	50-75							
VVPWCH	1-36	52-14							
WINCHR	1-52	17-19	19-25						
WINPRT	1-35	52-25							
WRITTT	1-25	2-6#							
XHIIN	1-27	8-34#							
XHIOUT	1-27	8-15#							
XHISSET	1-27	8-5#							
XRDTIM	1-27	8-67#							
XTCC	8-50	8-62#							
XTERCK	1-27	8-55#							

