

Table of contents

4-	1	ININT	-- Terminal input interrupt entry point
5-	1	TTINPT	-- Process received character
6-	1	IRINGP	-- Store character into silo buffer
7-	1	SILFET	-- Fetch a character from the input silo
8-	1	TTRSAV	-- Save registers for TT input interrupt
9-	1	CIPFRK	-- Fork level input character processing
10-	1	GOTXOF	-- Process received XOFF character
11-	1	GOTXON	-- Process received XON character
12-	1	TRNSTR	-- Start line transmitter
13-	1	NEDCHR	-- Get next char for device driver
15-	1	CDORTN	-- Clock driven output character processing
16-	1	CDIRTN	-- Clock driven input character processing

```
1          .TITLE  TSTIOX -- Relocatable terminal service code
2 000000   .PSECT  TSTIOX
3         .ENABL  LC
4         .DSABL  GBL
5
6         ; TSTIOX contains the portion of the terminal service code that
7         ; is linked as an overlay and relocated over TSINIT when the
8         ; system is started.
9         ;
10        ; Copyright (c) 1984, 1985.
11        ; S&H Computer Systems, Inc.
12        ; Nashville, Tennessee USA
13        ;
14        ; Global definitions
15        ;
16        .GLOBL  TSTIOX
17        ;
18        ; Global references
19        ;
20        .GLOBL  OVRHC, CL$LIX, LCXTBL
21        .GLOBL  NEDCDO, NEDCDI, LCLUNT, LINIR, LCDTYP, CTRLS, LOTSIZ, LOTBUF
22        .GLOBL  CTRLQ, CDX$VH, LMSQBF, LOTPNT, CDSXOF, CDSXON, CLTIMR, LXCL
23        .GLOBL  LNMAP, OTRASZ, CDOFLG, LHIRBS, LHIRBE, LHIRBB, INTPRI, LHIRBP
24        .GLOBL  INITFL, LHIRBC, LHIRBG, LHIRBA, $DETCH, CO$BBT, $HARD, $CTRLS
25        .GLOBL  CDX$DH, FRMERR, $INIT, $SXON, LSW, LSW7, S$OTWT
26        .GLOBL  $IITIM, $SXOFF, LSW5, $BBIT, LITIME, S$$RUN, $DHCDO, LSW3
27        .GLOBL  LOTEND, SB$LNK, $SOTFN, NMUMB, $PAGE, DHTIMR, S$OTLO, LSTHL
28        .GLOBL  S$OTFN, $NDICP, INTEN, FP$CDI, MBFFLG, FORK, $RBRK, CDIFLG
29        .GLOBL  LSW2, SNMSHD, CL$OPT, NEDSOT, $ICPFK, $DEAD, NEDCLO, SB$PNT
30        .GLOBL  LNPRIM, LSW10, CDSTRT, $HISTP, LSTATE, FASTIN, CDSTOP
31        .GLOBL  CO$BNO, CO$BNI, LOTSPC, $DNICP, PSW, KPAR6, LTTPAR
32        .GLOBL  LINNUM, RDINT, RSR, RCVDON, RBR, MUXNUM, MXRBUF, MXLNT
```

```

1      ;-----
2      ; Macro definitions:
3      ;
4      ; Disable interrupts
5      ;
6      ;     .MACRO  DISABL          ;Disable interrupts
7      ;     BIS    #340, @PSW
8      ;     .ENDM  DISABL
9      ;
10     ; Enable interrupts
11     ;
12     ;     .MACRO  ENABL          ;Enable interrupts
13     ;     BIC    @INTPRI, @PSW
14     ;     .ENDM  ENABL
15     ;
16     ; Call a routine in a system overlay
17     ;
18     ;     .MACRO  OCALL  ENTADD
19     ;     .IF    B, ENTADD
20     ;         .ERROR ;OCALL without entry address
21     ;     .ENDC
22     ;     CALL   @OVRHC
23     ;     .WORD  ENTADD
24     ;     .ENDM  OCALL
25     ;
26     ; The TTMAP and TTMAPX macros are used to map kernel-mode par6 to the
27     ; terminal character buffer area.  The previous contents of par6 map
28     ; register are pushed on the stack and may be restored by using the
29     ; UNMAP or UNMAPX macros.
30     ; R1 must contain the line index number of the line whose buffers
31     ; are being accessed.
32     ; The difference between the TTMAP-UNMAP macros and the TTMAPX-UNMAPX
33     ; macros is that the X-versions are more efficient but may only be
34     ; used from within interrupt service routines where we are guaranteed
35     ; to be running on the system stack.
36     ; The TTMAP and UNMAP versions of the macros must only be
37     ; used in sections of code where the interrupts are disabled.
38     ;
39     ;     .MACRO  TTMAPX
40     ;     MOV    LTPAR(R1), @KPAR6
41     ;     .ENDM  TTMAPX
42     ;
43     ;
44     ;
45     ;     .MACRO  UNMAPX
46     ;     .ENDM  UNMAPX
47     ;
48     ;     .MACRO  TTMAP
49     ;     MOV    @KPAR6, MAPHLD
50     ;     MOV    LTPAR(R1), @KPAR6
51     ;     .ENDM  TTMAP
52     ;
53     ;     .MACRO  UNMAP
54     ;     MOV    MAPHLD, @KPAR6
55     ;     .ENDM  UNMAP

```

```

1          ; -----
2          ; Module header. This data must be the first data in this segment
3          ;
4          ; Overlay segment ID word
5          ;
6 000000 077167 TSTIOX: .RAD50 /TIO/
7          ;
8          ; Table of offsets to entry points in this module
9          ;
10         .WORD ININT-TSTIOX
11         .WORD TTINPT-TSTIOX
12         .WORD SILFET-TSTIOX
13         .WORD TTRSAV-TSTIOX
14         .WORD TRNSTR-TSTIOX
15         .WORD NEDCHR-TSTIOX
16         .WORD CDORTN-TSTIOX
17         .WORD CDIRTN-TSTIOX
18         .WORD SNDFRE-TSTIOX
19         ;
20         ; Data areas
21         ;
22 000024 000000 MAPHLD: .WORD 0 ;Used by TTMAP macro
23         ;
24 000026 000 TTIFRK: .BYTE 0 ;Non-zero ==> Request fork level input process
25         .EVEN

```

ININT -- Terminal input interrupt entry point

```

1          .SBTTL  ININT  -- Terminal input interrupt entry point
2          ;-----
3          ;
4          ; Receiver interrupt routine - enter with interrupts disabled
5          ;
6 000030   ININT:
7          ;
8          ; Save registers R0, R1, R4, and R5.
9          ;
10 000030 004567 000716      JSR      R5,TTRSAV      ;Save registers
11          ;
12          ; R0, R1, R4 and R5 are now available.
13          ; Determine which line interrupted.
14          ;
15 000034 113704 000000G    MOVB     @#LINNUM,R4      ;Get interrupting line number
16 000040 001426          BEQ      1$              ;Branch if mux line interrupted
17          ;
18          ; Interrupt came from a DL11 line
19          ;
20 000042 105037 000000G    CLRB     @#LINNUM      ;Reset line number cell
21 000046 006304          ASL      R4              ;Get TSX line number index
22 000050 042774 000000G 000000G    BIC     #RDINT,@RSR(R4) ;Drop DL11 interrupt enable
23 000056 032774 000000G 000000G    BIT     #RCVDON,@RSR(R4);Did we receive a character?
24 000064 001410          BEQ      3$              ;Branch if not
25 000066 017405 000000G    MOV     @RBR(R4),R5    ;Get receiver buffer register info
26 000072 052774 000000G 000000G    BIS     #RDINT,@RSR(R4) ;Reenable DL11 interrupt
27 000100 004767 000066      CALL    TTINPT         ;Process the character
28 000104 000431          BR      9$              ;Finished
29 000106 052774 000000G 000000G 3$:   BIS     #RDINT,@RSR(R4) ;Reenable DL11 interrupts
30 000114 000425          BR      9$
31          ;
32          ; Interrupt came from a DZ11 or DH11 line
33          ;
34 000116 010346          1$:   MOV     R3,-(SP)      ;Get another work register for mux
35 000120 113703 000000G    MOVB     @#MUXNUM,R3    ;Get MUX number
36 000124 105037 000000G    CLRB     @#MUXNUM      ;Reset cell
37 000130 006303          ASL      R3              ;Convert to word table index
38 000132 017305 000000G    2$:   MOV     @MXRBUF(R3),R5 ;Get MUX receiver buffer register
39 000136 002013          BGE     4$              ;Br if mux silo is empty
40 000140 010504          MOV     R5,R4          ;Get receiver buffer info
41 000142 042704 170377      BIC     #^C<7400>,R4    ;Clear all but line #
42 000146 000304          SWAB    R4              ;Move to low order byte
43 000150 066304 000000G    ADD     MXLNT(R3),R4    ;Get pointer into map table
44 000154 111404          MOVB     (R4),R4        ;Get TSX line #
45 000156 001765          BEQ     2$              ;Br if line not genned into TSX
46 000160 004767 000006      CALL    TTINPT         ;Process the received character
47 000164 000762          BR      2$              ;Go see if more chars in mux silo
48 000166 012603          4$:   MOV     (SP)+,R3
49          ;
50          ; Finished
51          ;
52 000170 000207          9$:   RETURN          ;Return from interrupt

```

TTINPT -- Process received character

```

1          .SBTTL  TTINPT -- Process received character
2          ;-----
3          ; TTINPT is called each time a character is received from a serial line.
4          ; This routine stores the character into the initial ring buffer for the line
5          ; and then requests a .FORK to trigger the actual input character processing.
6          ;
7          ; Inputs:
8          ; R4 = Index number of interrupting line.
9          ; R5 = Received character and status flags in DL11 format.
10         ;
11 000172 010046 TTINPT: MOV     R0,-(SP)
12 000174 010346         MOV     R3,-(SP)
13         ;
14         ; Reset input interrupt timer for the line
15         ;
16 000176 042764 000000G 000000G         BIC     #IITIM,LSW5(R4);Reset input interrupt timer for line
17         ;
18         ; See if a framing error was detected
19         ;
20 000204 032705 000000G         BIT     #FRMERR,R5         ;Was a framing error detected?
21 000210 001404         BEQ     7$         ;Br if not
22 000212 052764 000000G 000000G         BIS     #RBRK,LSW10(R4);Set flag saying break was detected
23 000220 000456         BR      2$         ;Skip over special character tests
24         ;
25         ; See if we should mask character to 7 bits
26         ;
27 000222 042705 177400         7$:    BIC     #^C377,R5         ;Mask character to 8 bits
28 000226 016403 000000G         MOV     LCLUNT(R4),R3 ;Is this line being used by CL?
29 000232 002411         BLT     6$         ;Br if not
30 000234 032763 000000G 000000G         BIT     #CO$BNI,CL$OPT(R3);Binary input mode?
31 000242 001045         BNE     2$         ;Br if yes -- Accept all characters
32 000244 032763 000000G 000000G         BIT     #CO$BBT,CL$OPT(R3);Is CL unit set for 8 bit support?
33 000252 001007         BNE     3$         ;Br if 8 bit mode
34 000254 000404         BR      4$
35 000256 032764 000000G 000000G 6$:    BIT     #8BIT,LSW2(R4) ;Is line set to receive 8-bit chars?
36 000264 001002         BNE     3$         ;Br if yes
37 000266 042705 177600         4$:    BIC     #^C177,R5         ;Mask out all but 7 bits of character
38         ;
39         ; Ignore nulls unless this is a CL line in binary input mode
40         ;
41 000272 105705         3$:    TSTB    R5         ;Is the character null?
42 000274 001466         BEQ     9$         ;Ignore nulls unless in binary input mode
43         ;
44         ; See if character is ctrl-S (XOFF)
45         ;
46 000276 120527 000000G         CMPB    R5,#CTRLS         ;Is character ctrl-S?
47 000302 001011         BNE     1$         ;Br if not
48 000304 005703         TST     R3         ;Is line being used by CL?
49 000306 002004         BGE     10$         ;Br if yes
50 000310 032764 000000G 000000G         BIT     #PAGE,LSW2(R4) ;Is ctrl-S processing wanted for line?
51 000316 001417         BEQ     2$         ;Br if not -- treat ctrl-S like normal char
52 000320 004767 000670         10$:   CALL    GOTXOF         ;Process the ctrl-S character
53 000324 000452         BR      9$         ;Finished with character
54         ;
55         ; See if character is ctrl-Q (XON)
56         ;
57 000326 120527 000000G         1$:    CMPB    R5,#CTRLQ         ;Is character ctrl-Q?

```

TTINPT -- Process received character

```

58 000332 001011          BNE      2$          ;Br if not
59 000334 005703          TST      R3          ;Is line being used by CL?
60 000336 002004          BGE      11$         ;Br if yes
61 000340 032764 000000G 000000G  BIT      ##PAGE,LSW2(R4) ;Is ctrl-Q processing wanted for line?
62 000346 001403          BEQ      2$          ;Br if not -- treat ctrl-Q like normal char
63 000350 004767 000666          11$:    CALL    GOTXON      ;Process the ctrl-Q character
64 000354 000436          BR       9$
65                          ;
66                          ; Move character into silo buffer for this line
67                          ;
68 000356 004767 000076          2$:    CALL    IRINGP      ;Store character into silo buffer
69                          ;
70                          ; Set flag requesting fork-level processing for this line
71                          ;
72 000362 032764 000000G 000000G  BIT      ##NDICP,LSW10(R4);Have we already requested input processing?
73 000370 001030          BNE      9$          ;Br if yes
74 000372 052764 000000G 000000G  BIS      ##NDICP,LSW10(R4);Set flag saying input char processing needed
75 000400 005237 000000G          INC      @#NEDCDI      ;Say input character processing needed
76 000404 005727 000000G          TST      #FASTIN      ;Are we to allow non-clock driven forks?
77 000410 001020          BNE      9$          ;Br if not
78 000412 032764 000000G 000000G  BIT      ##DNICP,LSW10(R4);Have we already done a fork this clock cycle
79 000420 001014          BNE      9$          ;Br if yes -- Only do fork once per clock cycl
80 000422 032764 000000G 000000G  BIT      ##ICPFK,LSW10(R4);Is fork processing already pending for line?
81 000430 001010          BNE      9$          ;Br if yes
82 000432 105737 000000G          TSTB    @#INITFL      ;Is system initialization complete?
83 000436 001005          BNE      9$          ;Br if not -- Don't fork till init complete
84 000440 052764 000000G 000000G  BIS      ##DNICP,LSW10(R4);Set flag saying we forked during cycle
85 000446 105267 177354          INCB    TTIFRK        ;Request fork processing
86                          ;
87                          ; Finished
88                          ;
89 000452 012603          9$:    MOV      (SP)+,R3
90 000454 012600          MOV      (SP)+,R0
91 000456 000207          RETURN

```

IRINGP -- Store character into silo buffer

```

1          .SBTTL  IRINGP -- Store character into silo buffer
2          ;-----
3          ; IRINGP is called to store a received character into the input
4          ; silo buffer for a line.
5          ;
6          ; Inputs:
7          ; R4 = Line index number.
8          ; R5 = Character to be stored.
9          ;
10         000460 010146 IRINGP: MOV     R1,-(SP)
11         000462 010346         MOV     R3,-(SP)
12         ;
13         ; Store character into silo buffer
14         ;
15         000464 005764 00000000         TST     LHIRBS(R4) ;Any space remaining in ring buffer?
16         000470 001425         BEQ     2$ ;Br if not
17         000472 016403 00000000         MOV     LHIRBP(R4),R3 ;Get pointer to next pos in ring buffer
18         000476 110523         MOVB   R5,(R3)+ ;Store char into ring buffer
19         000500 020364 00000000         CMP     R3,LHIRBE(R4) ;Have we gone past the end of the buffer?
20         000504 103402         BLD    1$ ;Br if not
21         000506 016403 00000000         MOV     LHIRBB(R4),R3 ;Wrap around to the front of the buffer
22         000512 010364 00000000 1$: MOV     R3,LHIRBP(R4) ;Save pointer for next character
23         000516 005364 00000000         DEC     LHIRBS(R4) ;One less free space available
24         ;
25         ; If buffer is nearly full, send an XOFF to try to stop transmission
26         ;
27         000522 126464 00000000 00000000 CMPB   LHIRBS(R4),LHIRBC(R4);Is buffer nearly full?
28         000530 101021         BHI    3$ ;Br if not
29         000532 001404         BEQ     2$ ;Always send ctrl-S at set point
30         000534 032764 00000001 00000000 BIT     #1,LHIRBS(R4) ;Send ctrl-S on every other char if < set pt
31         000542 001014         BNE    3$ ;Br if not time to send ctrl-S
32         000544 016403 00000000 2$: MOV     LCLUNT(R4),R3 ;Is this line a CL unit?
33         000550 002404         BLT    4$ ;Br if not
34         000552 032763 00000000 00000000 BIT     #CD$BNO,CL$OPT(R3);Are we in binary output mode?
35         000560 001005         BNE    3$ ;Br if yes -- Cannot send XOFF
36         000562 010401         4$: MOV     R4,R1 ;Get line index number to R1
37         000564 016103 00000000         MOV     LCDTYP(R1),R3 ;Get hardware device type index
38         000570 004773 00000000         CALL   @CDSXOF(R3) ;Call routine to stuff an XOFF character
39         ;
40         ; Finished
41         ;
42         000574 012603 3$: MOV     (SP)+,R3
43         000576 012601         MOV     (SP)+,R1
44         000600 000207         RETURN

```

SILFET -- Fetch a character from the input silo

```

1          .SBTTL  SILFET -- Fetch a character from the input silo
2          ;-----
3          ; SILFET is called to fetch the next character from the input silo.
4          ;
5          ; Inputs:
6          ; R1 = Line index number.
7          ;
8          ; Outputs:
9          ; C-flag cleared ==> A character is available.
10         ; C-flag set    ==> No more characters in silo.
11         ; RO = Character gotten from silo.
12         ;
13 000602 010346 SILFET: MOV     R3,-(SP)
14 000604 005046      CLR     -(SP)          ;Will restore this into RO
15         ;
16         ; See if there are any characters in the silo
17         ;
18 000606 026161 000000G 000000G      CMP     LHIRBS(R1),LHIRBA(R1);Are there any chars in the silo?
19 000614 001452      BEQ     B$          ;Br if silo is empty
20         ;
21         ; There are characters in the silo.
22         ; Get the next character out of the silo.
23         ;
24 000616 016103 000000G      MOV     LHIRBG(R1),R3    ;Get pointer to next character
25 000622 112316      MOVB    (R3)+,(SP)    ;Fetch char from silo
26 000624 020361 000000G      CMP     R3,LHIRBE(R1)  ;Did we go beyond end of silo?
27 000630 103402      BLO     1$          ;Br if not
28 000632 016103 000000G      MOV     LHIRBB(R1),R3  ;Wrap around to front of silo
29 000636 010361 000000G 1$:  MOV     R3,LHIRBG(R1)  ;Save new pointer into silo
30 000642 005261 000000G      INC     LHIRBS(R1)    ;Say there is another free space in silo
31         ;
32         ; If we have transmitted an XOFF because the silo was nearly full,
33         ; transmit an XON now if we have nearly emptied the silo.
34         ;
35 000646 032761 000000G 000000G      BIT     ##HISTP,LSW10(R1);Have we transmitted an XOFF?
36 000654 001416      BEQ     2$          ;Br if not
37 000656 016103 000000G      MOV     LHIRBA(R1),R3  ;Get total silo size
38 000662 166103 000000G      SUB     LHIRBS(R1),R3  ;Compute # chars in silo now
39 000666 120361 000001G      CMPB   R3,LHIRBC+1(R1);Is it time to send XON?
40 000672 101007      BHI     2$          ;Br if not
41 000674 042761 000000G 000000G      BIC     ##HISTP,LSW10(R1);Say XOFF has been cancelled
42 000702 016100 000000G      MOV     LCDTYP(R1),RO  ;Get device type code
43 000706 004770 000000G      CALL   @CDSXON(RO)    ;Call routine to transmit an XON
44         ;
45         ; See if we need to translate this character
46         ;
47 000712 016103 000000G 2$:  MOV     LCXTBL(R1),R3  ;Get pointer to translation table
48 000716 001407      BEQ     3$          ;Br if there is no translation table
49 000720 012300 4$:  MOV     (R3)+,RO      ;Get next entry from translation table
50 000722 001405      BEQ     3$          ;Br if hit end of table
51 000724 000300      SWAB   RO           ;Get external char to low order byte
52 000726 121600      CMPB   (SP),RO       ;Does this match received character?
53 000730 001373      BNE     4$          ;Loop if not
54 000732 000300      SWAB   RO           ;Get internal char to low order byte
55 000734 110016      MOVB  RO,(SP)       ;Replace external char with internal char
56         ;
57         ; We got a character

```

```
58 ;  
59 000736 000241 3$: CLC ;Signal that we got a character  
60 000740 000401 BR 9$  
61 ;  
62 ; There are no characters available  
63 ;  
64 000742 000261 8$: SEC ;Signal that no characters are available  
65 ;  
66 ; Finished  
67 ;  
68 000744 012600 9$: MOV (SP)+,R0  
69 000746 012603 MOV (SP)+,R3  
70 000750 000207 RETURN
```

TTRSAV -- Save registers for TT input interrupt

```

1          .SBTTL  TTRSAV -- Save registers for TT input interrupt
2          ;-----
3          ; TTRSAV should be called from all terminal input interrupt reception
4          ; routines. It saves registers R0, R1, and R4 and sets up the stack
5          ; so that the RTI instruction that would normally return from the
6          ; interrupt will return to this routine to restore the registers before
7          ; returning from the interrupt.
8          ; This is done to allow us to defer doing a .INTEN until later in the
9          ; character processing.
10         ;
11         ; Form of call:
12         ;
13         ;     JSR      R5,TTRSAV
14         ;
15         ; Inputs:
16         ;   Interrupted PC and PSW must be on the top of the stack at the
17         ;   time that the JSR is executed.
18         ;
19 000752   TTRSAV:
20         ;
21         ; On entry, the following items are on the stack: R5, PC, and PSW.
22         ; Push R0, R1, and R4.
23         ;
24 000752   010046   MOV      R0,-(SP)
25 000754   010146   MOV      R1,-(SP)
26 000756   010446   MOV      R4,-(SP)
27 000760   013746   00000006  MOV      @#INTPRI,-(SP) ;Save current running interrupt priority
28         ;
29         ; Say our running interrupt priority is 7
30         ;
31 000764   005037   00000006  CLR      @#INTPRI      ;Say running priority = 7
32         ;
33         ; Now call calling routine as a coroutine
34         ;
35 000770   004715   CALL     (R5)          ;Call calling routine as a coroutine
36         ;
37         ; Finished character input interrupt processing.
38         ; Restore values we pushed.
39         ;
40 000772   012637   00000006  MOV      (SP)+,@#INTPRI ;Restore running interrupt priority
41 000776   012604   MOV      (SP)+,R4
42 001000   012601   MOV      (SP)+,R1
43 001002   012600   MOV      (SP)+,R0
44 001004   012605   MOV      (SP)+,R5
45         ;
46         ; At this point, only the interrupt PC and PSW are left on the stack.
47         ; See if we need to queue a fork request for input character processing.
48         ;
49 001006   105767   177014   TSTB    TTIFRK        ;Do we need to queue a fork request?
50 001012   001420   BEQ     9$            ;Br if not
51 001014   105067   177006   CLR     TTIFRK        ;Clear fork request flag
52 001020   105737   00000006  TSTB    @#CDIFLG      ;Is character processing already active?
53 001024   001013   BNE     9$            ;Br if yes
54 001026   105237   00000006  INCB    @#CDIFLG      ;Say we are doing input processing now
55         ;
56         ; Do a .INTEN as if we were accepting the interrupt initially
57         ; (The stack is set up as if an interrupt just occurred)

```

TTRSAV -- Save registers for TT input interrupt

```
58 ;
59 001032 004537 0000000 JSR R5,@#INTEN ;Do Inten
60 001036 000100 .WORD 100 ;Priority = 5
61 ;
62 ; Now fork and call TT input processing routine
63 ;
64 001040 004537 0000000 JSR R5,@#FORK ;Do .FORK
65 001044 0000000 .WORD FP$CDI ;Fork priority
66 001046 004767 0000004 CALL CIPFRK ;Call TT input processing routine
67 001052 000207 RETURN ;Return to inten routine which exits from int
68 ;
69 ; Return from interrupt
70 ;
71 001054 0000002 9#: RTI ;Return from interrupt
```

```

1          .SBTTL  CIPFRK -- Fork level input character processing
2          ;-----
3          ; CIPFRK is called at fork level to do input character processing.
4          ;
5 001056  010146  CIPFRK: MOV     R1,-(SP)
6 001060  010446          MOV     R4,-(SP)
7          ;
8          ; Begin loop to service each line
9          ;
10 001062  012701  000002          MOV     #2,R1          ;Start with line 1
11          ;
12          ; See if this routine needs input character processing
13          ;
14 001066  032761  000000G 000000G 1$: BIT     ##HARD,LSW3(R1) ;Is this a real line?
15 001074  001435          BEQ     3$             ;Br if not
16 001076          DISABL          ;** Disable interrupts **
17 001104  032761  000000G 000000G BIT     ##NDICP,LSW10(R1);;Does this line need input char processing?
18 001112  001423          BEQ     2$             ;;Br if not
19 001114  032761  000000G 000000G BIT     ##ICPFK,LSW10(R1);;Are we doing processing now?
20 001122  001017          BNE     2$             ;;Br if yes
21          ;
22          ; This line does need input character processing
23          ;
24 001124  052761  000000G 000000G BIS     ##ICPFK,LSW10(R1);;Set flag saying processing taking place
25 001132          ENABL          ;** Enable interrupts **
26 001140  042761  000000G 000000G BIC     ##NDICP,LSW10(R1);Say input character processing has been done
27 001146  010104          MOV     R1,R4          ;Get line index number in R4 also
28 001150  004771  000000G          CALL    @LINIR(R1)     ;Call input processing routine for line
29 001154  042761  000000G 000000G BIC     ##ICPFK,LSW10(R1);Say no longer doing input processing
30          ;
31          ; See if there are more lines to check
32          ;
33 001162          2$: ENABL          ;** Enable interrupts **
34 001170  062701  000002          3$: ADD     #2,R1          ;Get next line index number
35 001174  020127  000000G          CMP     R1,#LSTHL     ;Have we done all lines?
36 001200  101732          BLOS    1$             ;Loop if more to service
37          ;
38          ; Say fork level input processing is finished
39          ;
40 001202  105037  000000G          CLR    @#CDIFLG       ;Say input processing finished
41          ;
42          ; Finished
43          ;
44 001206  012604          MOV     (SP)+,R4
45 001210  012601          MOV     (SP)+,R1
46 001212  000207          RETURN

```

```
1 .SBTTL GOTXOF -- Process received XOFF character
2 ;-----
3 ; GOTXOF is called when we receive an XOFF (ctrl-S) character.
4 ; It suspends transmission.
5 ;
6 ; Inputs:
7 ; R4 = Line index number.
8 ;
9 001214 010146 GOTXOF: MOV R1, -(SP)
10 001216 010401 MOV R4, R1 ; Carry line index number in R1
11 ;
12 ; Set flag saying transmission is to be suspended
13 ;
14 001220 052761 0000000 0000000 BIS #CTRLS, LSW3(R1); Set flag saying ctrl-S received
15 ;
16 ; Call device-dependent routine to stop the transmitter
17 ;
18 001226 016100 0000000 MOV LCDTYP(R1), R0 ; Get comm device type code
19 001232 004770 0000000 CALL @CDSTOP(R0) ; Stop the transmitter
20 ;
21 ; Finished
22 ;
23 001236 012601 MOV (SP)+, R1
24 001240 000207 RETURN
```

GOTXON -- Process received XON character

```

1          .SBTTL  GOTXON -- Process received XON character
2          ;-----
3          ; GOTXON is called when we receive an XON (ctrl-Q) character.
4          ;
5          ; Inputs:
6          ; R4 = Line index number.
7          ;
8 001242 010146 GOTXON: MOV     R1,-(SP)
9 001244 010401          MOV     R4,R1          ;Get line index number to R1
10         ;
11         ; Clear flag that says output is suspended
12         ;
13 001246 042761 0000000 0000000          BIC     #$CTRLS,LSW3(R1);Cancel control-S output suspension
14         ;
15         ; Call routine to restart output to the line
16         ;
17 001254 016100 0000000          MOV     LCDTYP(R1),R0  ;Get device type code
18 001260 004770 0000000          CALL    @CDSTRT(R0)   ;Call routine to start transmitter
19         ;
20         ; Finished
21         ;
22 001264 012601          MOV     (SP)+,R1
23 001266 000207          RETURN

```

TRNSTR -- Start line transmitter

```

1          .SBTTL  TRNSTR -- Start line transmitter
2          ;-----
3          ; TRNSTR is called each time a character is added to a terminal's
4          ; output buffer to try to start the transmitter for the terminal.
5          ; All registers are preserved.
6          ;
7          ; Inputs:
8          ; R1 = Virtual line index number
9          ;
10         TRNSTR: MOV     RO,-(SP)
11         MOV     R1,-(SP)
12         MOV     LNPRIM(R1),R1 ;Get primary line index number
13         ;
14         ; Never try to start output for a detached job
15         ;
16         BIT     #$DEATCH,LSW(R1) ;Is this a detached job?
17         BNE     9$ ;Br if yes -- No output to detached jobs
18         ;
19         ; Don't start output if output has been suspended.
20         ;
21         BIT     #<$CTRLS!$DEAD>,LSW3(R1);Line suspended or dead?
22         BNE     9$ ;Br if yes
23         BIT     #$HARD,LSW3(R1) ;Is hardware connected to this line?
24         BEQ     9$ ;Br if not
25         ;
26         ; Line is able to receive a character.
27         ; Call device-dependent routine to start output.
28         ;
29         MOV     LCDTYP(R1),RO ;Get comm device type index
30         CALL    @CDSTRT(RO) ;Call routine to start transmitter for line
31         ;
32         ; Finished
33         ;
34         9$: MOV     (SP)+,R1
35         MOV     (SP)+,RO
36         RETURN

```

NEDCHR -- Get next char for device driver

```

1          .SBTTL  NEDCHR -- Get next char for device driver
2          ;-----
3          ; NEDCHR is called from a terminal device driver to get the
4          ; next char that is to be sent to the terminal.
5          ;
6          ; Inputs:
7          ; R4 = Physical line number for which a character is being gotten.
8          ;
9          ; Outputs:
10         ; C-flag cleared ==> Another character is available.
11         ; C-flag set      ==> No more characters available.
12         ; RO = Character to send if C-flag cleared.
13         ;
14 001346 010146 NEDCHR: MOV     R1,-(SP)
15 001350 010246      MOV     R2,-(SP)
16 001352 010446      MOV     R4,-(SP)
17         ;
18         ; See if we need to send an XOFF (ctrl-S)
19         ;
20 001354 032764 000000G 000000G      BIT     $$SXOFF,LSW10(R4);Do we need to send an XOFF?
21 001362 001406      BEQ     12$          ;Br if not
22 001364 042764 000000G 000000G      BIC     $$SXOFF,LSW10(R4);Clear flag saying XOFF needed
23 001372 112700 000000G      MOVB    #CTRLS,RO      ;Get ctrl-S character
24 001376 000556      BR      15$          ;Go send it
25         ;
26         ; See if we need to send an XON (ctrl-Q)
27         ;
28 001400 032764 000000G 000000G 12$:  BIT     $$SXON,LSW10(R4);Do we need to send an XON?
29 001406 001406      BEQ     13$          ;Br if not
30 001410 042764 000000G 000000G      BIC     $$SXON,LSW10(R4);Clear flag saying XON needed
31 001416 112700 000000G      MOVB    #CTRLQ,RO      ;Get ctrl-Q character
32 001422 000544      BR      15$          ;Go send it
33         ;
34         ; Get virtual line number to R1
35         ;
36 001424 116401 000000G      13$:  MOVB    LNMAP(R4),R1      ;Get virtual line number for this phys line
37         ;
38         ; Determine if output to line is suspended due to Control-S character
39         ;
40 001430 032761 000000G 000000G      BIT     $$INIT,LSW(R1) ;Has line been initialized yet?
41 001436 001540      BEQ     8$          ;Br if line is not initialized
42 001440 032764 000000G 000000G      BIT     $$CTRLS,LSW3(R4);Is output to line suspended?
43 001446 001134      BNE     8$          ;Br if yes
44         ;
45         ; See if the next character is coming from a message buffer
46         ;
47 001450 016402 000000G      2$:  MOV     LMSGBF(R4),R2      ;Is output coming from a message buffer?
48 001454 001414      BEQ     3$          ;Br if not
49 001456 016200 000000G      MOV     SB$PNT(R2),RO      ;Get address of next character in buffer
50 001462 005262 000000G      INC     SB$PNT(R2)      ;Advance the character pointer
51 001466 112000      MOVB    (RO)+,RO      ;Get the character from the buffer
52 001470 001110      BNE     5$          ;Br if we got a character to send
53 001472 016264 000000G 000000G      MOV     SB$LNK(R2),LMSGBF(R4);Remove this element from list for job
54 001500 004767 000246      CALL    SNDFRE      ;Free the message buffer
55 001504 000761      BR      2$          ;See if there is another to send
56         ;
57         ; See if line is cross connected to a CL line

```

```

58 ;
59 001506 016402 0000000 3#: MOV LXCL(R4),R2 ;Is line cross connected to a CL line?
60 001512 002406 BLT 14$ ;Br if not
61 001514 016201 0000000 MOV CL$LIX(R2),R1 ;Get # of line CL unit is using
62 001520 004767 177056 CALL SILFET ;Try to get char from input silo
63 001524 103072 BCC 5$ ;Br if got a character
64 001526 000504 BR 8$ ;No character to transmit
65 ;
66 ; Determine how many characters are in terminal output buffer for line
67 ;
68 001530 016100 0000000 14#: MOV LOTSIZ(R1),R0 ;Get size of output character buffer
69 001534 166100 0000000 SUB LOTSPC(R1),R0 ;Determine number of characters
70 001540 001477 BEQ 8$ ;Br if there are no characters in output buf
71 ;
72 ; If we are about to run out of characters, see if we need
73 ; to give the job an execution priority boost
74 ;
75 001542 026127 0000000 0000000 CMP LSTATE(R1),#S$OTFN ;Is job already in a high-priority state?
76 001550 101433 BLOS 10$ ;Br if yes -- no need to boost priority
77 001552 005761 0000000 TST LITIME(R1) ;Is this an interactive job?
78 001556 001004 BNE 11$ ;Br if yes
79 001560 026127 0000000 0000000 CMP LSTATE(R1),#S$OTLO ;Non-interactive high priority state?
80 001566 101424 BLOS 10$ ;Br if already in high prio state
81 001570 020027 0000005 11#: CMP R0,#5. ;Are we really about to run out of chars?
82 001574 103410 BLO 6$ ;Br if yes -- Definitely give boost
83 001576 026127 0000000 0000000 CMP LSTATE(R1),#S$OTWT ;Is job waiting for output buffer space?
84 001604 001015 BNE 10$ ;Br if not
85 001606 020027 0000000 CMP R0,#OTRASZ ;Are we down to reactivation point?
86 001612 003012 BGT 10$ ;Br if not
87 001614 000404 BR 7$ ;Br if yes -- reactivate user
88 001616 026127 0000000 0000000 6#: CMP LSTATE(R1),#S$RUN ;Is job in a run or wait state?
89 001624 101005 BHI 10$ ;Br if in wait state -- Don't boost
90 001626 052761 0000000 0000000 7#: BIS ##SOTFN,LSW7(R1);Set flag to cause clock int to boost priority
91 001634 005237 0000000 INC @#NEDSOT ;Say we need job scheduler processing
92 ;
93 ; Send next character from normal terminal output buffer
94 ;
95 001640 016102 0000000 10#: MOV LOTPNT(R1),R2 ;Get pointer to next character in buffer
96 001644 TTMAP ;Map kernel PAR 6 to TT buffer for job
97 001660 112200 MOV (R2)+,R0 ;Get next character from terminal buffer
98 001662 UNMAP ;Remap kernel PAR 6
99 001670 005261 0000000 INC LOTSPC(R1) ;Say there is one more char space in buffer
100 001674 020261 0000000 CMP R2,LOTEND(R1) ;Did we just go beyond end of buffer?
101 001700 103402 BLO 4$ ;Br if not
102 001702 016102 0000000 MOV LOTBUF(R1),R2 ;Wrap around to front of buffer
103 001706 010261 0000000 4#: MOV R2,LOTPNT(R1) ;Save updated buffer pointer
104 ;
105 ; There is a character to send.
106 ; See if we need to do any character translation.
107 ;
108 001712 016402 0000000 5#: MOV LCXTBL(R4),R2 ;Is there a character translation table?
109 001716 001406 BEQ 15$ ;Br if not
110 001720 012201 16#: MOV (R2)+,R1 ;Get entry from translation table
111 001722 001404 BEQ 15$ ;Br if hit end of table
112 001724 120001 CMPB R0,R1 ;Compare character with that in table
113 001726 001374 BNE 16$ ;Br if no match
114 001730 000301 SWAB R1 ;Get replacement char to low order

```

```
115 001732 110100          MOVB    R1,R0          ;Get replacement character
116                          ;
117                          ; We have a character to send in R0
118                          ;
119 001734 000241        15$:   CLC              ;Signal success on return
120 001736 000401          BR      9$
121                          ;
122                          ; No characters left to send
123                          ;
124 001740 000261        8$:   SEC              ;Signal failure on return
125                          ;
126                          ; Finished
127                          ;
128 001742 012604        9$:   MOV     (SP)+,R4
129 001744 012602          MOV     (SP)+,R2
130 001746 012601          MOV     (SP)+,R1
131 001750 000207          RETURN
```

NEDCHR -- Get next char for device driver

```

1      ;
2      ;
3      ;-----
4      ; SNDFRE is called to return a system message buffer to the free list.
5      ;
6      ; Inputs:
7      ; R2 = Address of system message buffer to be freed.
8      ; Interrupts must be disabled when this routine is called.
9      ;
10     .GLOBL SNMSHD, SB*LNK, NMUMB, MBFFLG
11
12 001752 SNDFRE:
13      ;
14      ; Return buffer to free list.
15      ;
16 001752 013762 0000000 0000000      MOV     @SNMSHD,SB*LNK(R2);Add buffer to free list
17 001760 010237 0000000      MOV     R2,@SNMSHD
18 001764 005237 0000000      INC     @NMUMB          ;Increment # free message buffers
19      ;
20      ; Set flag saying a message buffer has been freed so clock routine
21      ; will restart any waiting jobs.
22      ;
23 001770 005237 0000000      INC     @MBFFLG          ;Set flag for clock routine
24      ;
25      ; Finished
26      ;
27 001774 000207      RETURN

```

```

1          .SBTTL  CDORTN -- Clock driven output character processing
2          ;-----
3          ; CDORTN is called on every clock tick (50/60 Hz) to do clock
4          ; driven output character processing.
5          ;
6 001776  010144  CDORTN: MOV     R1,-(SP)
7 002000  005037  0000000  CLR     @#NEDCDO      ; Say don't need output processing
8          ;
9          ; Do processing for CL lines
10         ;
11 002004  005737  0000000  TST     @#NEDCLO      ; Need CL output processing?
12 002010  001404  BEQ     1$            ; Br if not
13 002012  005037  0000000  CLR     @#NEDCLO      ; Reset output-processing-needed flag
14 002016  004737  0000000  CALL    @#CLTIMR      ; Call routine for CL lines
15         ;
16         ; Do output character processing for DH11 and DHV11 lines
17         ;
18 002022  012701  0000000  1$:    MOV     #LSTHL,R1      ; Get index to last line
19 002026  032761  0000000  0000000  2$:    BIT     ##DHCDO,LSW10(R1); Is clock-driven processing needed for line?
20 002034  001415  BEQ     4$            ; Br if not
21 002036  026127  0000000  0000000  CMP     LCDTYP(R1),#CDX$VH; Is this a DHV11 line?
22 002044  001404  BEQ     3$            ; Br if yes
23 002046  026127  0000000  0000000  CMP     LCDTYP(R1),#CDX$DH; Is this a DH11 line?
24 002054  001005  BNE     4$            ; Br if not
25 002056  042761  0000000  0000000  3$:    BIC     ##DHCDO,LSW10(R1); Clear processing-needed flag for line
26 002064  004737  0000000  CALL    @#DHTIMR      ; Call clock driven routine for DH
27 002070  162701  0000002  4$:    SUB     #2,R1          ; Get next line index number
28 002074  003354  BGT     2$            ; Loop if more lines to check
29         ;
30         ; Say processing is finished
31         ;
32 002076  105037  0000000  CLRB    @#CDOFLG      ; Say processing is finished
33         ;
34         ; Finished
35         ;
36 002102  012601  MOV     (SP)+,R1
37 002104  000207  RETURN

```

```

1          .SBTTL  CDIRTN -- Clock driven input character processing
2          ;-----
3          ; CDIRTN is called on every clock tick (50/60 Hz) to do clock
4          ; driven input character processing.
5          ;
6 002106 010146 CDIRTN: MOV     R1,-(SP)
7 002110 010446      MOV     R4,-(SP)
8 002112 005037 0000000 CLR     @#NEDCDI      ; Say don't need input processing
9          ;
10         ; Begin loop to service each line
11         ;
12 002116 012701 0000002      MOV     #2,R1      ; Start with line 1
13         ;
14         ; See if this routine needs input character processing
15         ;
16 002122 032761 0000000 0000000 1$: BIT     ##HARD,LSW3(R1) ; Is this a real line?
17 002130 001440      BEQ     3$          ; Br if not
18 002132      DISABL      ;;; ** Disable interrupts **
19 002140 032761 0000000 0000000 BIT     ##NDICP,LSW10(R1);; Does this line need input char processing?
20 002146 001423      BEQ     2$          ;;; Br if not
21 002150 032761 0000000 0000000 BIT     ##ICPFK,LSW10(R1);; Are we doing processing now?
22 002156 001017      BNE     2$          ;;; Br if yes
23         ;
24         ; This line does need input character processing
25         ;
26 002160 052761 0000000 0000000 BIS     ##ICPFK,LSW10(R1);; Set flag saying processing taking place
27 002166      ENABL      ;;; ** Enable interrupts **
28 002174 042761 0000000 0000000 BIC     ##NDICP,LSW10(R1); Say input character processing has been done
29 002202 010104      MOV     R1,R4      ; Get line index number in R4 also
30 002204 004771 0000000      CALL    @LINIR(R1)    ; Call input processing routine for line
31 002210 042761 0000000 0000000 BIC     ##ICPFK,LSW10(R1); Say no longer doing input processing
32 002216 042761 0000000 0000000 2$: BIC     ##DNICP,LSW10(R1); Say input processing done for clock cycle
33         ;
34         ; See if there are more lines to check
35         ;
36 002224      ENABL      ;;; ** Enable interrupts **
37 002232 062701 0000002 3$: ADD     #2,R1      ; Get next line index number
38 002236 020127 0000000      CMP     R1,#LSTHL    ; Have we done all lines?
39 002242 101727      BLOS    1$          ; Loop if more to service
40         ;
41         ; Say clock driven input processing is finished
42         ;
43 002244 105037 0000000      CLRB    @#CDIFLG    ; Say clock driven input processing finished
44         ;
45         ; Finished
46         ;
47 002250 012604      MOV     (SP)+,R4
48 002252 012601      MOV     (SP)+,R1
49 002254 000207      RETURN
50 000001 000001      .END

```

Errors detected: 0

*** Assembler statistics

Work file reads: 0
 Work file writes: 0

TSTIOX -- Relocatable terminal MACRO V05.04 Friday 18-Dec-87 11:58 Page 16-1
CDIRTN -- Clock driven input character processing

Size of work file: 248 Words (1 Pages)
Size of core pool: 17920 Words (70 Pages)
Operating system: RT-11

Elapsed time: 00:00:42.43
DK:TSTIOX,LP:TSTIOX=DK:TSTIOX.MAC/C/N:SYM

\$BBIT	1-26	5-35							
\$CTRLS	1-24	10-14	11-13	12-21	13-42				
\$DEAD	1-29	12-21							
\$DETCH	1-24	12-16							
\$DHCDO	1-26	15-19	15-25						
\$DNICP	1-31	5-78	5-84	16-32					
\$HARD	1-24	9-14	12-23	16-16					
\$HISTP	1-30	7-35	7-41						
\$ICPFK	1-29	5-80	9-19	9-24	9-29	16-21	16-26	16-31	
\$IITIM	1-26	5-16							
\$INIT	1-25	13-40							
\$NDICP	1-28	5-72	5-74	9-17	9-26	16-19	16-28		
\$PAGE	1-27	5-50	5-61						
\$RBRK	1-28	5-22							
\$SOTFN	1-27	13-90							
\$SXOFF	1-26	13-20	13-22						
\$SXON	1-25	13-28	13-30						
CDIFLG	1-28	8-52	8-54*	9-40*	16-43*				
CDIRTN	3-17	16-6#							
CDOFLG	1-23	15-32*							
CDORTN	3-16	15-6#							
CDSTOP	1-30	10-19							
CDSTRT	1-30	11-18	12-30						
CDSXOF	1-22	6-38							
CDSXON	1-22	7-43							
CDX\$DH	1-25	15-23							
CDX\$VH	1-22	15-21							
CIPFRK	8-66	9-5#							
CL\$LIX	1-20	13-61							
CL\$OPT	1-29	5-30	5-32	6-34					
CLTIMR	1-22	15-14							
CO\$BBT	1-24	5-32							
CO\$BNI	1-31	5-30							
CO\$BNO	1-31	6-34							
CTRLQ	1-22	5-57	13-31						
CTRLS	1-21	5-46	13-23						
DHTIMR	1-27	15-26							
FASTIN	1-30	5-76							
FORK	1-28	8-64							
FP\$CDI	1-28	8-65							
FRMERR	1-25	5-20							
GOTXOF	5-52	10-9#							
GOTXON	5-63	11-8#							
ININT	3-10	4-6#							
INITFL	1-24	5-32							
INTEN	1-28	8-59							
INTPRI	1-23	8-27	8-31*	8-40*	9-25	9-33	16-27	16-36	
IRINGP	5-68	6-10#							
KPAR6	1-31	13-96	13-96*	13-98*					
LCDTYP	1-21	6-37	7-42	10-18	11-17	12-29	15-21	15-23	
LCLUNT	1-21	5-28	6-32						
LCXTBL	1-20	7-47	13-108						
LHIRBA	1-24	7-18	7-37						
LHIRBB	1-23	6-21	7-28						
LHIRBC	1-24	6-27	7-39						
LHIRBE	1-23	6-19	7-26						

TTINPT	3-11	4-27	4-46	5-11#
TTRSAV	3-13	4-10	8-19#	

DISABL	2-6#	9-16	16-18		
ENABL	2-12#	9-25	9-33	16-27	16-36
OCALL	2-18#				
TTMAP	2-48#	13-96			
TTMAPX	2-39#				
UNMAP	2-53#	13-98			
UNMAPX	2-45#				