

Table of contents

3-	1	WINCHR	-- Process next output character
4-	1	REGCHR	-- Process regular characters
5-	1	DOCTRL	-- Process control characters
6-	1	WCSCCHK	-- Check for start of control sequence
7-	1	WCPESC	-- Escape processing
8-	1	EPCH1	-- Accrue a terminal control sequence
10-	1	ACRCS	-- Store a character into control seq buffer
11-	1	CSFIN	-- Process received control sequence
13-	1	CSPARM	-- Extract parameter value from control string
14-	1	TCBS	-- Process backspace character
15-	1	TCHT	-- Process horizontal tab character
16-	1	TCCR	-- Process carriage return
17-	1	TCUP	-- Move cursor up specified number of lines
18-	1	TCDOWN	-- Move cursor down specified number of lines
19-	1	TCLF	-- Process Line Feed character
20-	1	TCESCE	-- Move to 1st column of next line
21-	1	TCIXUP	-- Index up one line
22-	1	TCRIT	-- Move cursor right specified number of chars
23-	1	TCLEFT	-- Move cursor left specified number of chars
24-	1	TCCADR	-- Do direct cursor addressing
25-	1	TCCPSV	-- Save current cursor position
25-	23	TCCPRS	-- Restore cursor position
26-	1	TCINSL	-- Insert lines
27-	1	TCDELL	-- Delete specified number of lines
28-	1	TCGOAS	-- Designate G0 as ascii character set
28-	11	TCGODS	-- Designate G0 as DEC supplemental character set
28-	21	TCGOUK	-- Designate G0 as UK national character set
28-	31	TCGOGR	-- Designate G0 as graphics character set
29-	1	TCG1AS	-- Designate G1 as ascii character set
29-	11	TCG1DS	-- Designate G1 as DEC supplemental character set
29-	21	TCG1UK	-- Designate G1 as UK national character set
29-	31	TCG1GR	-- Designate G1 as graphics character set
30-	1	TCG2AS	-- Designate G2 as ascii character set
30-	11	TCG2DS	-- Designate G2 as DEC supplemental character set
30-	21	TCG2GR	-- Designate G2 as graphics character set
31-	1	TCG3AS	-- Designate G3 as ascii character set
31-	11	TCG3DS	-- Designate G3 as DEC supplemental character set
31-	21	TCG3GR	-- Designate G3 as graphics character set
32-	1	TCSO	-- Shift-out character - Lock shift G1 to GL
32-	11	TCSI	-- Shift-in character - Lock shift G0 to GL
32-	21	TCLS2	-- Lock shift G2 to GL
32-	31	TCLS3	-- Lock shift G3 to GL
32-	41	TCLS1R	-- Lock shift G1 into GR
32-	51	TCLS2R	-- Lock shift G2 into GR
32-	61	TCLS3R	-- Lock shift G3 into GR
32-	71	TCSS2	-- Single shift G2 to GL
32-	82	TCSS3	-- Single shift G3 to GL
33-	1	TCSAA	-- Set ANSI attribute
33-	15	TCRAA	-- Reset ANSI attribute
33-	29	TCSDA	-- Set DEC attribute
33-	43	TCRDA	-- Reset DEC attribute
34-	1	TCSTA	-- Set terminal attribute
35-	1	TCRTA	-- Reset terminal attribute
36-	1	CVTTAV	-- Convert terminal attribute value to flag
37-	1	CKTAC	-- Check for complex mode changes
38-	1	TCAKM	-- Select application keypad mode
38-	11	TCNKM	-- Select numeric keypad mode

Table of contents

39-	1	TCN3	-- Set line attributes
40-	1	TCSCA	-- Set character attributes
42-	1	TC100	-- Select VT100/VT200 mode
43-	1	TCREST	-- Terminal reset
44-	1	TCPRT	-- Printer control operation
45-	1	TCLERS	-- Erase within a line
46-	1	TCPERS	-- Erase within a page
47-	1	TCSSR	-- Set scrolling region
48-	1	SETHOM	-- Set cursor to the home position
49-	1	SCRLUP	-- Scroll screen up one line
50-	1	SCRLDN	-- Scroll screen down one line
51-	1	SCRCHK	-- Check for scrolling limits
52-	1	WINSPN	-- Suspend job until it reattaches to terminal
53-	1	ERSLIN	-- Erase the current line
54-	1	ERSCTL	-- Erase from cursor to the end of the line
55-	1	ERSLTC	-- Erase from beginning of line to cursor
56-	1	ERSCTP	-- Erase from cursor to the end of the page
57-	1	ERSPTC	-- Erase from beginning of page to cursor
58-	1	ERSPAG	-- Clear entire page to spaces
59-	1	CLRLIN	-- Clear a line to spaces
60-	1	CPYLIN	-- Copy characters from one line to another
61-	1	SETLAB	-- Set attributes for current line
62-	1	GETLAB	-- Get attributes for current line
63-	1	SETLIN	-- Select a line as current line
64-	1	WINSF	-- Switch from a job with a display window
65-	1	WINST	-- Switch to job with a window
66-	1	WINDSP	-- Redisplay current window for job
67-	1	REFRSH	-- Refresh screen from window contents
68-	1	SNDLIN	-- Send line of characters to terminal
69-	1	GENCLR	-- Gen control sequence to clear screen
70-	1	GENCAF	-- Generate sequence to set character attributes
71-	1	GENCSC	-- Generate terminal sequence to select char set
72-	1	GENMAP	-- Generate sequence to set up char set mapping
73-	1	GENSSS	-- Generate sequence for split screen scrolling
74-	1	GENTEM	-- Turn VT52 emulation mode on or off
75-	1	GENWAF	-- Generate terminal sequence to set window attrib
76-	1	GENAAS	-- Generate control sequence for ANSI attributes
77-	1	GENDAS	-- Generate control sequence for DEC attributes
78-	1	GENLAF	-- Generate terminal sequence to set line attrib
79-	1	GENSPC	-- Generate sequence to move cursor over
80-	1	GENCSR	-- Generate cursor addressing sequence
81-	1	GENESC	-- Generate escape character
81-	8	GENCSI	-- Generate CSI character sequence
81-	30	GENCHR	-- Send a character to the terminal
82-	1	GENVAL	-- Convert value to digits and send
83-	1	EMTWIN	-- Dispatch window control EMT's
84-	1	WFNEW	-- EMT to create a new window
85-	1	WFMAP	-- EMT to select a window as the current window
86-	1	WFDEL	-- EMT to delete a window
87-	1	WFSPND	-- Suspend window processing
87-	15	WFRSUM	-- Resume window processing
88-	1	WFPRNT	-- Cause contents of a window to be printed
89-	1	WFREAD	-- Copy window information to program buffer
90-	1	WFSTT	-- EMT to set terminal type for windowing
91-	1	WINSTT	-- Set terminal type for a window
92-	1	WINREL	-- Release all display windows for a job
93-	1	WINDEL	-- Delete a window

Table of contents

94-	1	WINSRC	-- Locate control block for a window
95-	1	MAKWSB	-- Create named region for window screen buffer
96-	1	MAKRDB	-- Make a region definition block
97-	1	FREWSB	-- Free memory used by window screen buffer
98-	1	WININI	-- TSWIN initialization
99-	1	WINPRT	-- Window print-screen function

```

1          .TITLE  TSWIN -- TSX-Plus Display Windows
2          .ENABL  LC
3          .ENABL  AMA
4          .DSABL  GBL
5 000000   .CSECT  TSWIN
6 000000   110466 TSWIN: .RAD50 /WIN/           ;Overlay region id
7          ;
8          ; TSWIN is the TSX-Plus system overlay that provides display window support.
9          ;
10         ; Copyright (c) 1985.
11         ; S&H Computer Systems, Inc.
12         ; Nashville, Tennessee USA
13         ; All rights reserved.
14         ;
15         ; Global definitions
16         ;
17         .GLOBL  WINCHR, EMTWIN, WININI, WINREL, WINDSP
18         .GLOBL  WINST, WINSF, WINPRT
19         ;
20         ; Global references
21         ;
22         .GLOBL  LSW11, %V52EM, LNPRIM
23         .GLOBL  LSTSL, LPRG1, LPRG2, LBRKCG, NMFREQ, GETRTQ
24         .GLOBL  IQSIZ, CQ$R1, QCOMPL, VALADW, VALADB
25         .GLOBL  P2$CXT, PRIVC2, $8BIT, LSW2, AW$PRT
26         .GLOBL  RC$EXC, RC$FLG, LOTSIZ, LOTSPC, LOTNXT, LOTPNT
27         .GLOBL  $CTRL0, LSW3, $RFRSH, LSW4, AW$SPN
28         .GLOBL  $WDISP, LSW6, $OTWT, LSTATE, RC$BLK, RC$CNT
29         .GLOBL  $VNOTT, LSW, PCSPND, FRKPRI, FORCEX
30         .GLOBL  DW$CSB, DW$CSR, DW$CSP, TCSBSZ, LWINDO
31         .GLOBL  ESC, CSICHR, SS3CHR, AW$52, DW$AW, BUFCHR, AW$52
32         .GLOBL  KPAR6, VPAR6, OVRHC, INTPRI, PSW, PR7
33         .GLOBL  DW$CCA, DW$COL, DW$CPL, DW$LIN, DW$SRB
34         .GLOBL  DW$SRT, DW$LPP, DW$TLN, DW$LPT, DW$MAP
35         .GLOBL  DW$RID, DW$JOB, DW$ID, RC$BAS, R. @ID
36         .GLOBL  R. @SIZ, R. @STS, R. NAME, RS. CGR, RS. PVT
37         .GLOBL  RS. @BL, EMTBLK, SETERR, EMTXIT, RS. EGR
38         .GLOBL  DW$$$SZ, CR, LF, AC$BLD, AC$BLK, AC$REV
39         .GLOBL  CORUSR, AC$ULN, AL$DHT, AL$DHB, AL$DWD, VT100
40         .GLOBL  BUFCHR, VMXWIN, VT52, VT2007, VT2008, LTRMTP
41         .GLOBL  AW$200, AW$132, AW$INS, AW$ACK, AW$REV, AW$ORS
42         .GLOBL  AW$AKM, AW$VCR, LSW7, $SLON, $SLKED, LCOL
43         .GLOBL  AW$PRM, $DETCH, LSW, LSW6, $WDISP, AW$RPT
44         .GLOBL  AW$SS, DW$GLS, DW$GLM, DW$GRM, AC$SET
45         .GLOBL  DW$MSL, DW$NSL, AW$DDC
46         .GLOBL  DW$GOM, DW$G1M, DW$G2M, DW$G3M
47         .GLOBL  DW$SLN, DW$SCL, DW$SCA, WINTOP

```

```

1      ; -----
2      ; Macro definitions
3      ;
4      ; Macros to enable and disable interrupts
5      ;
6      .MACRO  DISABL          ;Disable interrupts
7      BIS    #PR7,@#PSW
8      .ENDM  DISABL
9
10     .MACRO  ENABL          ;Enable interrupts
11     BIC    INTPRI,@#PSW
12     .ENDM  ENABL
13
14     ; Macro to call a system overlay
15     ;
16     .MACRO  OCALL  ENTADD
17     .IF    B,ENTADD
18     .ERROR ;OCALL without entry address
19     .ENDC
20     CALL   OVRHC
21     .WORD ENTADD
22     .ENDM  OCALL
23
24     ; Macros to map to screen buffer and restore mapping
25     ;
26     .MACRO  BUFMAP
27     DISABL
28     MOV    @#KPAR6,MAPHLD
29     MOV    DW$MAP(R2),@#KPAR6
30     .ENDM  BUFMAP
31
32     .MACRO  UNMAP
33     MOV    MAPHLD,@#KPAR6
34     ENABL
35     .ENDM  UNMAP
36
37     ; Macro to send a character to the terminal
38     ;
39     .MACRO  SEND  CHAR
40     MOVB   CHAR,R0
41     CALL   GENCHR
42     .ENDM  SEND
43
44     ; -----
45     ; Data areas and parameters
46     ;
47     SEMI   =      73          ;Semicolon
48     ;
49     DWBAS: .WORD  WINTOP      ;Pointer to 1st window control block
50     DWEND: .WORD   0          ;Pointer past last window control block
51     MAPHLD: .WORD  0          ;Temp cell used by BUFMAP/UNMAP macros
52     R5OWIN: .RAD50 /WIN/
53     R5OPRT: .RAD50 /PRT/
54     CSBUF:  .BLKB  16.       ;Cursor string work buffer
55     .EVEN

```

WINCHR -- Process next output character

```

1          .SBTTL  WINCHR -- Process next output character
2          ;-----
3          ; This routine is called to process each character sent to the terminal
4          ; by a running program.  It determines if the character is part of an
5          ; escape sequence or is a regular character.
6          ;
7          ; Inputs:
8          ;   RO = Character being sent.
9          ;   R1 = Job index number.
10         ;
11         ; Outputs:
12         ;   C-flag set ==> Do not send this char to terminal now.
13         ;
14 000034 010046 WINCHR: MOV     RO,-(SP)
15 000036 010246      MOV     R2,-(SP)
16 000040 010546      MOV     R5,-(SP)
17 000042 010005      MOV     RO,R5          ;Carry character in R5
18         ;
19         ; Get pointer to current window control block for job
20         ;
21 000044 016102 000000G      MOV     LWINDO(R1),R2  ;Get pointer to window control block
22         ;
23         ; See if processing is suspended for this window
24         ;
25 000050 032762 000000G 000000G      BIT     #AW$SPN,DW$AW(R2); Is window processing suspended?
26 000056 001045      BNE     8$          ;Br if yes -- Just pass through the char
27         ;
28         ; Mask character to 7 bits unless 8 bit support has been enabled
29         ;
30 000060 032761 000000G 000000G      BIT     #$8BIT,LSW2(R1) ; Is 8 bit char support wanted?
31 000066 001002      BNE     1$          ;Br if yes
32 000070 042705 177600      BIC     #^C<177>,R5    ;Mask character to 7 bits
33         ;
34         ; Ignore nulls
35         ;
36 000074 105705 1$:      TSTB   R5          ; Is character null?
37 000076 001427      BEQ     7$          ; Br if null
38         ;
39         ; Determine if this character begins a terminal control sequence
40         ;
41 000100 004737 000510'      CALL   WCSCHK         ; Check for control sequence start
42 000104 103024      BCC     7$          ; Br if start of control sequence
43         ;
44         ; See if we are currently accepting a terminal control sequence
45         ;
46 000106 016200 000000G      MOV     DW$CSR(R2),RO  ; Are we processing a control sequence?
47 000112 001402      BEQ     3$          ; Br if not
48 000114 004710      CALL   (RO)         ; Call routine to process this character
49 000116 000417      BR     7$          ; Finished with this character
50         ;
51         ; If output is being directed to the terminal printer port, then
52         ; don't update window image with this character.
53         ;
54 000120 032762 000000G 000000G 3$:  BIT     #AW$PRT,DW$AW(R2); Is output going to printer?
55 000126 001013      BNE     7$          ; Br if yes -- Bypass window update
56         ;
57         ; Determine if this is a normal or control character

```

WINCHR -- Process next output character

```

58 ;
59 000130 010500          MOV     R5,R0          ;Get the character
60 000132 042700 177600   BIC     #^C<177>,R0    ;Mask to 7 bits
61 000136 020027 000037   CMP     R0,#37        ;Is this a control character?
62 000142 101003          BHI     4$            ;Br if not
63 000144 004737 000370'   CALL    DOCTRL        ;Process a control character
64 000150 000402          BR      7$            ;
65 ;
66 ; Process regular character
67 ;
68 000152 004737 000204'  4$:    CALL    REGCHR    ;Process a regular character
69 ;
70 ; See if we should display this character now
71 ;
72 000156 032762 000000G 000000G 7$:    BIT     #AW$DDC,DW$AW(R2);Should we suppress char display?
73 000164 001402          BEQ     8$            ;Br if not
74 ;
75 ; We don't want to display this character
76 ;
77 000166 000261          SEC                      ;Signal to not display the character
78 000170 000401          BR      9$            ;
79 ;
80 ; We want to display this character
81 ;
82 000172 000241          8$:    CLC                      ;Signal to display the character
83 ;
84 ; Finished
85 ;
86 000174 012605          9$:    MOV     (SP)+,R5
87 000176 012602          MOV     (SP)+,R2
88 000200 012600          MOV     (SP)+,R0
89 000202 000207          RETURN

```

```

1          .SBTTL  REGCHR -- Process regular characters
2          ;-----
3          ; Process regular characters which are stored into the appropriate
4          ; cell of the character matrix.
5          ;
6          ; Inputs:
7          ;   R2 = Pointer to window control block
8          ;   R5 = Character
9          ;
10         REGCHR: MOV     R3, -(SP)
11         MOV     R5, -(SP)
12         ;
13         ; Get pointer to 1st character on current line
14         ;
15         MOV     DW$LPT(R2), R3 ; Get pointer to 1st char on line
16         ;
17         ; Get pointer to current character on the line
18         ;
19         MOV     DW$COL(R2), R0 ; Get current column number
20         DEC     R0 ; 1st column is # 1
21         ASL     R0 ; Two bytes used per column
22         ADD     R0, R3 ; Get pointer to char byte for current col
23         ;
24         ; Get number of character set for this character
25         ;
26         MOVB    DW$GRM(R2), R0 ; Assume character is in GR
27         BIT     #200, R5 ; Is character in GR (8 bit character)
28         BNE     1$, ; Br if yes
29         MOVB    DW$GLM(R2), R0 ; Determine which set GL is mapped to
30         BIT     #AW$SS, DW$AW(R2); Are we single-shifting this character?
31         BEQ     1$, ; Br if not
32         MOVB    DW$GLS(R2), R0 ; Get single-shift mapping for char
33         BIC     #AW$SS, DW$AW(R2); Reset single-shift flag
34         1$: ADD     R2, R0 ; Add base address of window control block
35         MOVB    DW$GOM(R0), R0 ; Get char set for this character
36         BIC     #^C<177>, R5 ; Mask character to 7 bits
37         ;
38         ; Get character attribute flags (bold, blinking, etc.)
39         ;
40         BISB    DW$CCA(R2), R0 ; Add character attribute flags
41         ;
42         ; Store character and attribute into that cell
43         ;
44         BUFBMAP ;;; Map to screen buffer
45         MOVB    R5, (R3)+ ;;; Store character
46         MOVB    R0, (R3)+ ;;; Store character attribute flags
47         UNMAP ; Restore mapping
48         ;
49         ; Advance column number unless we are at the end of the line
50         ;
51         CMP     DW$COL(R2), DW$CPL(R2); Are we at the end of the line now?
52         BHIS    9$, ; Br if at end of the line now
53         INC     DW$COL(R2) ; Advance to next column
54         ;
55         ; Finished
56         ;
57         9$: MOV     (SP)+, R5

```


58 000364 012603
59 000366 000207

MOV (SP)+, R3
RETURN

```

1          .SBTTL  DOCTRL -- Process control characters
2          ;-----
3          ; This routine is called when the character being output is a
4          ; control character (000 - 037).
5          ;
6          ; Inputs:
7          ;   R2 = Pointer to window control block
8          ;   R5 = Character
9          ;
10         DOCTRL:
11         ;
12         ; Call processing routine for this character
13         ;
14         MOV     R5,R0          ;Get the character
15         BIC     #^C<177>,R0   ;Mask to 7 bits
16         ASL     R0            ;Convert to word table index
17         CALL    @CCRTN(R0)    ;Call appropriate processing routine
18         ;
19         ; Finished
20         ;
21         RETURN
22         ;-----
23         ; Branch table for control character processing routines
24         ;
25         CCRTN: .WORD  CCEXIT   ; 000 - NUL
26         .WORD  CCEXIT   ; 001 - SOH
27         .WORD  CCEXIT   ; 002 - STX
28         .WORD  CCEXIT   ; 003 - ETX
29         .WORD  CCEXIT   ; 004 - EOT
30         .WORD  CCEXIT   ; 005 - ENQ
31         .WORD  CCEXIT   ; 006 - ACK
32         .WORD  CCEXIT   ; 007 - BEL
33         .WORD  TCBS     ; 010 - BS
34         .WORD  TCHT     ; 011 - HT
35         .WORD  TCLF     ; 012 - LF
36         .WORD  TCLF     ; 013 - VT
37         .WORD  TCLF     ; 014 - FF
38         .WORD  TCCR     ; 015 - CR
39         .WORD  TCSD     ; 016 - SO
40         .WORD  TCSI     ; 017 - SI
41         .WORD  CCEXIT   ; 020 - DLE
42         .WORD  CCEXIT   ; 021 - DC1
43         .WORD  CCEXIT   ; 022 - DC2
44         .WORD  CCEXIT   ; 023 - DC3
45         .WORD  CCEXIT   ; 024 - DC4
46         .WORD  CCEXIT   ; 025 - NAK
47         .WORD  CCEXIT   ; 026 - SYN
48         .WORD  CCEXIT   ; 027 - ETB
49         .WORD  CCEXIT   ; 030 - CAN
50         .WORD  CCEXIT   ; 031 - EM
51         .WORD  REGCHR   ; 032 - SUB (ctrl-Z)
52         .WORD  CCEXIT   ; 033 - ESC
53         .WORD  CCEXIT   ; 034 - FS
54         .WORD  CCEXIT   ; 035 - GS
55         .WORD  CCEXIT   ; 036 - RS
56         .WORD  CCEXIT   ; 037 - US
57         .WORD  CCEXIT

```

```
58 ;  
59 ; Immediate return for most control characters  
60 ;  
61 000506 000207 CCEXIT: RETURN
```

```

1          .SBTTL  WCSCHK -- Check for start of control sequence
2          ;-----
3          ; This routine is called to determine if the current character begins
4          ; a terminal control sequence.
5          ;
6          ; Inputs:
7          ; R2 = Window control block.
8          ; R5 = Current character
9          ;
10         ; Outputs:
11         ; C-flag cleared ==> Start of control sequence
12         ; C-flag set    ==> Not start of control sequence
13         ;
14 000510  WCSCHK:
15         ;
16         ; See if this is the start of an escape sequence
17         ;
18 000510 120527 000000G      CMPB   R5,#ESC      ;Is this character escape?
19 000514 001003             BNE    1$              ;Br if not
20 000516 004737 000546'     CALL   WCPESC      ;Begin escape sequence
21 000522 000405             BR     8$
22         ;
23         ; See if this character is CSI, beginning of VT200 control sequence
24         ;
25 000524 120527 000000G  1$:  CMPB   R5,#CSICHR    ;CSI character?
26 000530 001004             BNE    7$              ;Br if not
27 000532 004737 000600'     CALL   WCPCSI      ;Start VT200 control sequence
28         ;
29         ; We began a control sequence
30         ;
31 000536 000241             B$:   CLC                ;Signal control sequence began
32 000540 000401             BR     9$
33         ;
34         ; We did not begin a control sequence
35         ;
36 000542 000261             7$:   SEC                ;Signal that not control sequence
37         ;
38         ; Finished
39         ;
40 000544 000207             9$:   RETURN

```

```

1          .SBTTL  WCPESC -- Escape processing
2          ;-----
3          ; Process an Escape character
4          ;
5          ; Inputs:
6          ;   R2 = Window control block
7          ;   R5 = Escape character
8          ;
9 000546   WCPESC:
10         ;
11         ; Initialize control sequence buffer pointer
12         ;
13 000546   005062   000000G      CLR      DW$CSP(R2)      ; Say no chars accrued yet
14         ;
15         ; Set address of routine to be called to process 1st character after
16         ; escape.
17         ;
18 000552   012700   000624'      MOV      #EPCH1,R0      ; Routine for 1st character after escape
19 000556   032762   000000C 000000G  BIT      #<AW$52!AW$S52>,DW$AW(R2); Is terminal in VT52 emulation mode?
20 000564   001402                                     BEQ      1$             ; Br if not
21 000566   012700   000710'      MOV      #EP52,R0      ; Set routine for VT52 mode
22 000572   010062   000000G      1$:     MOV      R0,DW$CSR(R2) ; Set address of processing routine
23 000576   000207                                     RETURN
24
25         ;-----
26         ; Received a CSI control character for a VT200 terminal.
27         ;
28         ; Inputs:
29         ;   R2 = Window control block address
30         ;
31 000600   012762   000624' 000000G  WCPCSI: MOV      #EPCH1,DW$CSR(R2) ; Set address of routine for next char
32 000606   005062   000000G      CLR      DW$CSP(R2)      ; Say no chars accrued yet
33 000612   112700   000133      MOVB    #'[,R0         ; Treat CSI char like ESC [
34 000616   004737   001136'      CALL   ACRC5          ; Store "[" as 1st char of sequence
35 000622   000207                                     RETURN

```

EPCH1 -- Accrue a terminal control sequence

```

1          .SBTTL  EPCH1  -- Accrue a terminal control sequence
2          ;-----
3          ; EPCH1 is called to accrue a VT100 or VT200 control sequence.
4          ;
5          ; There are two types of "control" sequences:
6          ;
7          ; Escape sequence format:  ESC I...I F
8          ;     I = Intermediate character in the range 040 to 057
9          ;     F = Final character in the range 060 to 176
10         ;
11         ; Control sequence format:  ESC P...P I...I F
12         ;     CSI = CSI character or ESC [
13         ;     P   = Parameter values in the range 060 to 077
14         ;     I   = Intermediate characters in the range 040 to 057
15         ;     F   = Final character in the range 100 to 176
16         ;
17         ; Inputs:
18         ;     R2 = Address of window control block.
19         ;     R5 = Received character
20         ;
21 000624 010546 EPCH1:  MOV    R5, -(SP)
22 000626 042705 177600      BIC    #^C<177>, R5      ;Mask character to 7 bits
23         ;
24         ; Add received character to end of control sequence already received
25         ;
26 000632 010500      MOV    R5, R0      ;Get received character
27 000634 004737 001136'    CALL   ACRCB      ;Store into control sequence buffer
28         ;
29         ; Determine if this character terminates the control sequence
30         ;
31 000640 120527 000060      CMPB   R5, #60      ;Is this the terminating character?
32 000644 103417      BLO    9%          ;Br if not
33 000646 120527 000176      CMPB   R5, #176     ;Is this the terminating character?
34 000652 101014      BHI    9%          ;Br if not
35 000654 120527 000133      CMPB   R5, #'[     ;Is this part of string header?
36 000660 001411      BEQ    9%          ;Br if string header
37 000662 120527 000100      CMPB   R5, #100    ;100 to 176 always terminates
38 000666 103004      BHIS  1%          ;
39 000670 126227 000000G 000133  CMPB   DW$CSB(R2), #'[ ;Is this an escape or control sequence?
40 000676 001402      BEQ    9%          ;Br if control sequence. Only 100-176 term
41         ;
42         ; This is the final character of the control sequence
43         ;
44 000700 004737 001174'    1$:   CALL   CSFIN      ;End of control sequence
45         ;
46         ; Finished
47         ;
48 000704 012605      9$:   MOV    (SP)+, R5
49 000706 000207      RETURN

```

EPCH1 -- Accrue a terminal control sequence

```

1          ; -----
2          ;   Accrue VT52 control sequences.
3          ;   All VT52 control sequences consist of ESC followed by a single character
4          ;   except for the cursor addressing sequence ESC Y line col.
5          ;   The VT52 sequences are converted to VT52 compatible sequences.
6          ;
7          ;   Inputs:
8          ;   R2 = Address of window control block
9          ;   R5 = Received character
10         ;
11 000710 010546 EP52:  MOV    R5, -(SP)
12 000712 042705 177600    BIC    #^C<177>, R5      ;Mask character to 7 bits
13         ;
14         ;   If this character is a letter, insert "L" in front of it.
15         ;
16 000716 120527 000101    CMPB   R5, #'A          ;Letter
17 000722 103416          BLD    2$              ;Br if not
18 000724 120527 000132    CMPB   R5, #'Z          ;Letter
19 000730 101013          BHI    2$              ;Br if not
20 000732 012700 000133    MOV    #'L, R0         ;Store "L"
21 000736 004737 001136'   CALL   ACRCS
22         ;
23         ;   Determine if this is the cursor addressing function
24         ;
25 000742 120527 000131    CMPB   R5, #'Y          ;Cursor addressing function?
26 000746 001004          BNE    2$              ;Br if not
27 000750 012762 000776' 000000G  MOV    #EP52L, DW$CSR(R2);Set routine to process next char
28 000756 000405          BR     9$
29         ;
30         ;   This character terminates the control sequence.
31         ;
32 000760 010500 2$:      MOV    R5, R0          ;Get received character
33 000762 004737 001136'   CALL   ACRCS          ;Add to string we are accruing
34 000766 004737 001174'   CALL   CSFIN          ;This terminates the control sequence
35         ;
36         ;   Finished
37         ;
38 000772 012605 9$:      MOV    (SP)+, R5
39 000774 000207          RETURN
40         ;
41         ; -----
42         ;   Accept the character which represents the line number in a VT52
43         ;   cursor addressing sequence.
44         ;
45         ;   Inputs:
46         ;   R2 = Address of window control block
47         ;   R5 = Received character (whose binary value is line number + 037)
48         ;
49 000776 042705 177600 EP52L: BIC    #^C<177>, R5      ;Mask character to 7 bits
50 001002 120527 000070    CMPB   R5, #70         ;Octal 70 ==> Remain on current line
51 001006 103404          BLD    1$              ;Br if new line specified
52 001010 016205 000000G  MOV    DW$LIN(R2), R5  ;Get current line number
53 001014 062705 000037    ADD    #37, R5         ;Convert to ascii character
54 001020 004737 001066'  1$:   CALL   CV52          ;Convert char to digit string
55 001024 112700 000073    MOVB   #SEMI, R0      ;Get parameter separator character
56 001030 004737 001136'   CALL   ACRCS          ;Store after line number value
57 001034 012762 001044' 000000G  MOV    #EP52C, DW$CSR(R2);Set routine to process column number char

```

```

58 001042 000207                RETURN
59
60 ;-----
61 ; Accept the character which represents the column number in a VT52
62 ; cursor addressing sequence.
63 ;
64 ; Inputs:
65 ; R2 = Address of window control block
66 ; R5 = Received character (whose binary value is column number + 037)
67 ;
68 001044 004737 001066' EP52C: CALL CV52 ;Convert char to digit string
69 001050 112700 000110      MOVB #'H,R0 ;Store terminating character
70 001054 004737 001136'      CALL ACRC5
71 001060 004737 001174'      CALL CSFIN ;This terminates the control sequence
72 001064 000207                RETURN
73
74 ;-----
75 ; Convert a received character whose binary value represents a line
76 ; number or a column number in a VT52 cursor addressing sequence
77 ; into an ascii digit string and store into the accrued control
78 ; sequence.
79 ;
80 ; Inputs:
81 ; R2 = Address of window control block
82 ; R5 = Received character whose binary value - 37 is the stored result
83 ;
84 001066 010446 CV52: MOV R4,-(SP)
85 001070 010546      MOV R5,-(SP)
86 ;
87 ; Subtract bias of 37
88 ;
89 001072 162705 000037      SUB #37,R5 ;Remove bias from character value
90 ;
91 ; Convert to digit string
92 ;
93 001076 005004      CLR R4 ;Clear high-order for divide
94 001100 071427 000012      DIV #10,R4 ;Split into two decimal digits
95 001104 062704 000060      ADD #'0,R4 ;Form high-order ascii digit
96 001110 010400      MOV R4,R0 ;Get the character
97 001112 004737 001136'      CALL ACRC5 ;Store into control string
98 001116 062705 000060      ADD #'0,R5 ;Form low-order ascii digit
99 001122 010500      MOV R5,R0 ;Get the character
100 001124 004737 001136'      CALL ACRC5 ;Store into control string
101 ;
102 ; Finished
103 ;
104 001130 012605      MOV (SP)+,R5
105 001132 012604      MOV (SP)+,R4
106 001134 000207                RETURN
    
```


ACRCS -- Store a character into control seq buffer

```

1          .SBTTL  ACRCS  -- Store a character into control seq buffer
2          ;-----
3          ; This routine is called to add another character to the end of the
4          ; accrued control sequence in the DW$CSB buffer area.
5          ;
6          ; Inputs:
7          ; R0 = Character to be added to end of string
8          ; R2 = Address of window descriptor block
9          ;
10         ACRCS:  MOV      R3, -(SP)
11         001136 010346      BIC      #^C<177>, R0      ;Mask character to 7 bits
12         001140 042700 177600
13         ;
14         ; Make sure we don't overflow the control sequence buffer
15         001144 016203 000000G      MOV      DW$CSP(R2), R3      ;Get # chars in buffer now
16         001150 020327 177777G      CMP      R3, #TCSBSZ-1      ;Is buffer full already?
17         001154 103005      BHIS     9$,              ;Br if yes -- Ignore this character
18         ;
19         ; Store character into buffer
20         ;
21         001156 060203      ADD      R2, R3              ;Point to next char pos in buffer
22         001160 110063 000000G      MOVB   R0, DW$CSB(R3)      ;Store char into buffer
23         001164 005262 000000G      INC     DW$CSP(R2)        ;Increment character count
24         ;
25         ; Finished
26         ;
27         001170 012603      9$:     MOV      (SP)+, R3
28         001172 000207      RETURN

```

CSFIN -- Process received control sequence

```

1          .SBTTL  CSFIN  -- Process received control sequence
2          ;-----
3          ; We have finished accruing a control sequence, process it.
4          ;
5          ; Inputs:
6          ; R2 = Address of window control block.
7          ;
8 001174 010346 CSFIN:  MOV    R3, -(SP)
9 001176 010446      MOV    R4, -(SP)
10 001200 010546      MOV    R5, -(SP)
11          ;
12          ; Store null at end of control sequence string
13          ;
14 001202 010203      MOV    R2, R3          ;Get address of window control block
15 001204 066203 000000G  ADD    DW$CSP(R2), R3 ;Point to next char pos in buffer
16 001210 105063 000000G  CLRB  DW$CSB(R3)     ;Store null at end of string
17          ;
18          ; Move control sequence to work buffer and remove any parameter characters
19          ;
20 001214 012705 000014'  MOV    #CSBUF, R5     ;Point to work buffer
21 001220 010203      MOV    R2, R3          ;Get address of buffer with full string
22 001222 062703 000000G  ADD    #DW$CSB, R3
23 001226 121327 000133  CMPB  (R3), #'['      ;Do we need to exclude parameter values?
24 001232 001014      BNE    3$             ;Br if not
25          ;
26          ; This is a control sequence. Ignore parameter value characters in the
27          ; range 060 to 076.
28          ;
29 001234 112300 1$:    MOVB  (R3)+, R0     ;Get next char from input string
30 001236 001410      BEQ    4$             ;Br if hit end of string
31 001240 120027 000060  CMPB  R0, #060        ;Is this a parameter character?
32 001244 103403      BLD    2$             ;Br if not
33 001246 120027 000076  CMPB  R0, #076        ;Parameter character?
34 001252 101770      BLOS  1$             ;Br if parameter character
35 001254 110025 2$:    MOVB  R0, (R5)+   ;Store char into work buffer
36 001256 000766      BR     1$             ;Keep moving
37 001260 105025 4$:    CLRB  (R5)+   ;Store null at end of string
38 001262 000402      BR     5$
39          ;
40          ; This is an escape sequence. We don't have to remove parameter values.
41          ;
42 001264 112325 3$:    MOVB  (R3)+, (R5)+ ;Move char to work buffer
43 001266 001376      BNE    3$             ;Loop till null moved
44          ;
45          ; The control sequence is now in the work buffer (CSBUF) with any parameter
46          ; value strings removed.
47          ; Begin loop to try to identify the control sequence.
48          ;
49 001270 012704 001432' 5$:    MOV    #CSTBLL, R4   ;Point to 1st set of control sequences
50 001274 126527 177776 000140  CMPB  -2(R5), #140    ;Is final char of sequence less than 140?
51 001302 103002      BHIS  10$            ;Br if not
52 001304 012704 001560'  MOV    #CSTBLU, R4   ;Point to control seq table for final < 140
53          ;
54          ; Compare accrued control sequence with sequence stored in table
55          ;
56 001310 012703 000014' 10$:   MOV    #CSBUF, R3     ;Point to accrued control sequence
57 001314 122324 12$:   CMPB  (R3)+, (R4)+ ;Compare the strings

```

CSFIN -- Process received control sequence

```

58 001316 001004          BNE      13$          ;Br if mismatch
59 001320 105764 177777   TSTB    -1(R4)         ;Was that the end of both strings?
60 001324 001373          BNE      12$          ;Br if not
61 001326 000412          BR       14$          ;We have a match
62                          ;
63                          ; Strings do not match -- Move on to next table entry
64                          ;
65 001330 005304          13$:    DEC      R4          ;Point to last char compared in table
66 001332 105724          15$:    TSTB    (R4)+        ;Search for null at end of asciz string
67 001334 001376          BNE      15$          ;Loop till null found
68 001336 062704 000003   ADD     #3,R4         ;Bound up and skip over following word
69 001342 042704 000001   BIC     #1,R4         ;Get to word boundary
70 001346 005714          TST     (R4)         ;Reached end of table?
71 001350 001357          BNE      10$          ;Loop if not
72 001352 000417          BR       20$          ;Control sequence is not in our table
73                          ;
74                          ; We found the control sequence in our table.
75                          ; Call processing routine.
76                          ; Set R5 to point to start of any possible parameter string in control seq.
77                          ;
78 001354 010205          14$:    MOV     R2,R5          ;Point to window control block
79 001356 062705 000001G  ADD     #DW$CSB+1,R5 ;Point to start of possible parameter string
80 001362 005204          INC     R4           ;Bound up to next word
81 001364 042704 000001   BIC     #1,R4         ;Get to word boundary
82 001370 011404          MOV     (R4),R4       ;Get address of routine to call
83 001372 032762 000000G 000000G BIT     #AW$PRT,DW$AW(R2);Are we directing output to printer port?
84 001400 001403          BEQ     16$          ;Br if not
85 001402 020427 004674'  CMP     R4,#TCPRT     ;Is this printer control operation?
86 001406 001001          BNE      20$          ;Br if not -- Ignore till printer turned off
87 001410 004714          16$:    CALL    (R4)         ;Call processing routine
88                          ;
89                          ; Finished with control sequence
90                          ;
91 001412 005062 000000G  20$:    CLR     DW$CSP(R2)    ;No accrued chars in control sequence
92 001416 005062 000000G  CLR     DW$CSR(R2)    ;Not accruing a control sequence now
93 001422 012605          MOV     (SP)+,R5
94 001424 012604          MOV     (SP)+,R4
95 001426 012603          MOV     (SP)+,R3
96 001430 000207          RETURN

```

CSFIN -- Process received control sequence

```

1          ; -----
2          ;   Generate table of terminal control sequences and associated processing
3          ;   routines.
4          ;
5          .MACRO  ESCSEQ  STRING,RTN
6          .ASCIZ  /STRING/
7          .EVEN
8          .WORD   RTN
9          .ENDM   ESCSEQ
10         ;
11         ;   Table for control sequences whose final character is in the
12         ;   range 140 to 177
13         ;
14         CSTBLL:
15         ESCSEQ  <[f>,TCCADR      ;Cursor addressing
16         ESCSEQ  <[!p>,TC100     ;Select VT100/VT200 mode
17         ESCSEQ  <[h>,TCSAA      ;Set ANSI terminal attribute
18         ESCSEQ  <[l>,TCRAA      ;Reset ANSI terminal attribute
19         ESCSEQ  <[?h>,TCSDA     ;Set DEC terminal attribute
20         ESCSEQ  <[?l>,TCRDA     ;Reset DEC terminal attribute
21         ESCSEQ  <[i>,TCPRT      ;Printer control code
22         ESCSEQ  <[m>,TCSCA      ;Set character attribute
23         ESCSEQ  <[r>,TCSSR      ;Set scrolling region
24         ESCSEQ  <[!p>,TCREST    ;Soft reset
25         ESCSEQ  <c>,TCREST      ;Hard reset
26         ESCSEQ  <~>,TCLS1R     ;Lock shift Q1 as QR character set
27         ESCSEQ  <n>,TCLS2      ;Lock shift Q2 as GL character set
28         ESCSEQ  <o>,TCLS3      ;Lock shift Q3 as GL character set
29         ESCSEQ  <}>,TCLS2R     ;Lock shift Q2 as QR character set
30         ESCSEQ  <!>,TCLS3R     ;Lock shift Q3 as QR character set
31         .WORD   0              ;End of table
32         ;
33         ;   Table for control sequences whose final character is in the
34         ;   range 60 to 137
35         ;
36         CSTBLU:
37         ESCSEQ  <[A>,TCUP       ;Move cursor up
38         ESCSEQ  <[B>,TCDOWN     ;Move cursor down
39         ESCSEQ  <[C>,TCRIT      ;Move cursor right
40         ESCSEQ  <[D>,TCLEFT     ;Move cursor left
41         ESCSEQ  <[F>,TCGODS     ;Select graphic char set for VT52
42         ESCSEQ  <[G>,TCGOAS     ;Select normal char set for VT52
43         ESCSEQ  <[H>,TCCADR     ;Cursor addressing
44         ESCSEQ  <[I>,TCIXUP     ;Reverse index (VT52)
45         ESCSEQ  <[J>,TCPERS     ;Erase within a page
46         ESCSEQ  <[K>,TCLERS     ;Erase within a line
47         ESCSEQ  <[L>,TCINSL     ;Insert lines
48         ESCSEQ  <[M>,TCDELL     ;Delete lines
49         ESCSEQ  <D>,TCLF        ;Index (Move cursor down one line)
50         ESCSEQ  <E>,TCESCE     ;Move to 1st column of next line
51         ESCSEQ  <M>,TCIXUP     ;Reverse index (move up 1 line)
52         ESCSEQ  <N>,TCSS2      ;Single shift next char to Q2
53         ESCSEQ  <O>,TCSS3      ;Single shift next char to Q3
54         ESCSEQ  <(A>,TCGOUK     ;Designate GO as UK
55         ESCSEQ  <(B>,TCGOAS     ;Designate GO as ascii
56         ESCSEQ  <(O>,TCGOGR     ;Designate GO as graphics
57         ESCSEQ  <A>,TCG1AS     ;Designate G1 as UK

```

CSFIN -- Process received control sequence

58	001744				ESCSEQ	, TCG1AS	; Designate G1 as ascii
59	001752				ESCSEQ	<O>, TCG1GR	; Designate G1 as graphics
60	001760				ESCSEQ	<*B>, TCG2AS	; Designate G2 as ascii
61	001766				ESCSEQ	<*O>, TCG2GR	; Designate G2 as graphics
62	001774				ESCSEQ	<+B>, TCG3AS	; Designate G3 as ascii
63	002002				ESCSEQ	<+O>, TCG3GR	; Designate G3 as graphics
64	002010				ESCSEQ	<#3>, TCN3	; Line is top half of double-high line
65	002016				ESCSEQ	<#4>, TCN4	; Line is bottom half of double high
66	002024				ESCSEQ	<#5>, TCN5	; Line is single width
67	002032				ESCSEQ	<#6>, TCN6	; Line is double width
68	002040				ESCSEQ	<7>, TCCPSV	; Save cursor position
69	002044				ESCSEQ	<8>, TCCPRS	; Restore cursor position
70	002050				ESCSEQ	<=>, TCAKM	; Select application keypad mode
71	002054	050	074	000	.ASCIZ	/(<74>	; "("
72					.EVEN		
73	002060	003354'			.WORD	TCG0DS	; Designate G0 as DEC supplemental graphics
74	002062	051	074	000	.ASCIZ	/)<74>	; ")"
75					.EVEN		
76	002066	003412'			.WORD	TCG1DS	; Designate G1 as DEC supplemental graphics
77	002070	052	074	000	.ASCIZ	/*<74>	; "*"
78					.EVEN		
79	002074	003450'			.WORD	TCG2DS	; Designate G2 as DEC supplemental graphics
80	002076	053	074	000	.ASCIZ	/+<74>	; "+"
81					.EVEN		
82	002102	003476'			.WORD	TCG3DS	; Designate G3 as DEC supplemental graphics
83	002104	074	000		.ASCIZ	<74>	; Select ANSI mode with "<" character
84	002106	004474'			.WORD	TC100	; Select ANSI mode
85	002110	076	000		.ASCIZ	<76>	; Generate entry with ">" character
86	002112	004240'			.WORD	TCNKM	; Select numeric keypad mode
87	002114	000000			.WORD	0	; End of table

CSPARM -- Extract parameter value from control string

```

1          .SBTTL  CSPARM -- Extract parameter value from control string
2          ;-----
3          ; CSPARM obtains the next parameter value from the control string.
4          ; If there is no parameter value, 0 is returned as the value.
5          ; Non-parameter characters are skipped over.
6          ;
7          ; Inputs:
8          ;   R5 = Pointer into control string where parameter may start.
9          ;
10         ; Outputs:
11         ;   C-flag set ==> No more parameter values.
12         ;   R0 = Accrued parameter value.
13         ;   R5 = Points past end of parameter value.
14         ;
15 002116 010146 CSPARM: MOV     R1, -(SP)
16 002120 005001      CLR     R1             ;Accrue the value in R1
17         ;
18         ; Scan up to the start of a parameter value
19         ;
20 002122 112500 1$:      MOVB   (R5)+, R0      ;Get next char from control seq
21 002124 001430      BEQ     5$             ;Br if hit end of the string
22 002126 120027 000073  CMPB   R0, #SEMI      ;Semicolon is value separator
23 002132 001432      BEQ     7$             ;Br if no value specified for this param
24 002134 120027 000071  CMPB   R0, #'9       ;Is this a parameter character?
25 002140 101370      BHI     1$             ;Br if not
26 002142 120027 000060  CMPB   R0, #'0       ;Br if not digit
27 002146 103765      BLO     1$
28         ;
29         ; We found the start of a parameter value.  Accrue it.
30         ;
31 002150 000412      BR      4$             ;Enter loop in middle
32 002152 112500 3$:      MOVB   (R5)+, R0      ;Get next char of parameter
33 002154 001420      BEQ     2$             ;Br if hit null at end of string
34 002156 120027 000071  CMPB   R0, #'9       ;Is this char part of parameter?
35 002162 101016      BHI     7$             ;Br if not
36 002164 120027 000060  CMPB   R0, #'0       ;Br if not
37 002170 103413      BLO     7$
38 002172 070127 000012  MUL    #10., R1      ;Multiply previous value by 10.
39 002176 162700 000060 4$:      SUB    #'0, R0      ;Convert ascii digit to binary value
40 002202 060001      ADD    R0, R1        ;Add into previous value
41 002204 000762      BR      3$             ;Keep accruing
42         ;
43         ; There are no more parameter values
44         ;
45 002206 005305 5$:      DEC    R5             ;Point to string terminator
46 002210 005000      CLR    R0             ;Return 0 as "value"
47 002212 000261      SEC             ;Signal that there are no more parameters
48 002214 000403      BR      9$
49         ;
50         ; Finished getting the parameter value
51         ;
52 002216 005305 2$:      DEC    R5             ;Point to parameter terminator
53 002220 010100 7$:      MOV    R1, R0        ;Get accrued value
54 002222 000241      CLC             ;Signal success on return
55         ;
56         ; Finished
57         ;

```

58 002224 012601
59 002226 000207

9#: MOV (SP)+,R1
RETURN

TCBS -- Process backspace character

```

1          .SBTTL  TCBS  -- Process backspace character
2          ;-----
3          ; Process a backspace character.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;
8 002230   TCBS:
9          ;
10         ; Set column number back one unless we are already at column one
11         ;
12 002230   026227 000000G 000001      CMP    DW$COL(R2),#1  ;Are we already at column 1?
13 002236   001402          BEQ    9$      ;Br if yes
14 002240   005362 000000G          DEC    DW$COL(R2)      ;Move column number back one
15         ;
16         ; Finished
17         ;
18 002244   000207      9$:  RETURN

```


TCHT -- Process horizontal tab character

```

1          .SBTTL  TCHT  -- Process horizontal tab character
2          ;-----
3          ; Process horizontal tab character.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block
7          ;
8 002246   TCHT:
9          ;
10         ; Get current column number and advance to next multiple of 8
11         ;
12 002246   016200 000000G      MOV     DW$COL(R2),R0    ;Get current column number
13 002252   005300          DEC     R0                    ;Make 1st column be # 0
14 002254   062700 000010      ADD     #8.,R0           ;Advance to next tab stop
15 002260   042700 000007      BIC     #7,R0           ;Bound to next multiple of 8
16 002264   005200          INC     R0                    ;Make 1st column be # 1
17         ;
18         ; Make sure we don't go beyond end of line
19         ;
20 002266   020062 000000G      CMP     R0,DW$CPL(R2)   ;Compare with # columns per line
21 002272   101402          BLOS   1$              ;Br if within line
22 002274   016200 000000G      MOV     DW$CPL(R2),R0   ;Force to end of line
23 002300   010062 000000G      1$:    MOV     R0,DW$COL(R2) ;Set new column number
24         ;
25         ; Finished
26         ;
27 002304   000207          RETURN

```

TCCR -- Process carriage return

```
1          .SBTTL  TCCR  -- Process carriage return
2          ;-----
3          ; Process carriage return, move to 1st char of current line.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block.
7          ;
8 002306 012762 000001 000000G TCCR:  MOV    #1,DW$COL(R2)  ;Say current column = 1
9          ;
10         ; Finished
11         ;
12 002314 000207          RETURN
```

TCUP -- Move cursor up specified number of lines

```

1          .SBTTL  TCUP  -- Move cursor up specified number of lines
2          ;-----
3          ; Move the cursor up the page the specified number of lines.
4          ; Do not scroll if we reach the bottom of the page.
5          ;
6          ; Inputs:
7          ;   R2 = Pointer to window control block
8          ;   R5 = Pointer to parameter value in control sequence
9          ;
10         002316 TCUP:
11         ;
12         ; Get count of number of lines to move up
13         ;
14         002316 004737 002116'          CALL  CSPARM          ;Accrue parameter value
15         002322 005700                  TST    RO            ;Was specified value 0?
16         002324 001001                  BNE   1$            ;Br if not
17         002326 005200                  INC   RO            ;Treat 0 like 1
18         ;
19         ; Move up specified number of lines
20         ;
21         002330 005400          1$:     NEG   RO            ;Get negative number of lines to move
22         002332 066200 000000G        ADD   DW$LIN(R2),RO  ;Compute new line
23         002336 003002                  BGT   2$            ;Br if didn't go past top of page
24         002340 012700 000001          MOV   #1,RO         ;Constrain to top of page
25         002344 026262 000000G 000000G 2$:  CMP   DW$LIN(R2),DW$SRT(R2);Were we already above top of scroll rgn?
26         002352 103405                  BLD   3$            ;Br if yes -- Don't constrain
27         002354 020062 000000G        CMP   RO,DW$SRT(R2) ;Would this put us above top of scroll regn?
28         002360 103002                  BHIS  3$            ;Br if not
29         002362 016200 000000G        MOV   DW$SRT(R2),RO ;Constrain to top of scroll region
30         002366 004737 006710'        3$:   CALL  SETLIN          ;Set new line for cursor
31         ;
32         ; Finished
33         ;
34         002372 000207          RETURN

```

TCDOWN -- Move cursor down specified number of lines

```

1          .SBTTL  TCDOWN -- Move cursor down specified number of lines
2          ;-----
3          ; Move the cursor down the page the specified number of lines.
4          ; Do not scroll if we reach the bottom of the page.
5          ;
6          ; Inputs:
7          ;   R2 = Pointer to window control block
8          ;   R5 = Pointer to parameter value in control sequence
9          ;
10         TCDOWN:
11         ;
12         ; Get count of number of lines to move down
13         ;
14         002374 004737 002116'          CALL    CSPARM          ;Accrue parameter value
15         002400 005700                   TST     RO              ;Was specified value 0?
16         002402 001001                   BNE    1$              ;Br if not
17         002404 005200                   INC     RO              ;Treat 0 like 1
18         ;
19         ; Move down specified number of lines
20         ;
21         002406 066200 000000G          1$:    ADD     DW$LIN(R2),RO ;Compute new line
22         002412 020062 000000G          CMP     RO,DW$LPP(R2)   ;Would this go past end of page?
23         002416 101402                   BLOS   3$              ;Br if not
24         002420 016200 000000G          MOV     DW$LPP(R2),RO  ;Constrain to bottom of page
25         ;
26         ; If we are currently with the scroll region, don't move out
27         ;
28         002424 026262 000000G 000000G 3$:    CMP     DW$LIN(R2),DW$SRB(R2);Were we already past end of scroll?
29         002432 101005                   BHI    2$              ;Br if past end -- Don't constrain
30         002434 020062 000000G          CMP     RO,DW$SRB(R2)  ;Did we go past end of scroll region?
31         002440 101402                   BLOS   2$              ;Br if not
32         002442 016200 000000G          MOV     DW$SRB(R2),RO  ;Stop at bottom of scroll region
33         002446 004737 006710'          2$:    CALL   SETLIN      ;Set new line for cursor
34         ;
35         ; Finished
36         ;
37         002452 000207                   RETURN

```

TCLF -- Process Line Feed character

```

1          .SBTTL  TCLF  -- Process Line Feed character
2          ;-----
3          ; Process a line feed character.  Move down to the next line.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;
8 002454   TCLF:
9          ;
10         ; If we are not at the bottom line of a scrolling region, simply
11         ; advance the line number.
12         ;
13 002454   MOV     DW$LIN(R2),R0    ;Get the current line number
14 002460   CMP     RO,DW$SRB(R2)    ;Are we at bottom of scrolling region?
15 002464   BEQ     1$              ;Br if yes
16 002466   INC     RO              ;Increment the line number
17 002470   CMP     RO,DW$LPP(R2)    ;Are we past the end of the page?
18 002474   BLOS   2$              ;Br if not
19 002476   MOV     DW$LPP(R2),RO    ;Constrain to bottom of page
20 002502   2$:   CALL  SETLIN      ;Set this as current line
21 002506   BR     9$
22         ;
23         ; We are at the bottom line of a scroll region.
24         ; We must scroll up one line.
25         ;
26 002510   1$:   CALL  SCRLUP      ;Scroll up one line
27         ;
28         ; Finished
29         ;
30 002514   9$:   RETURN

```

TCESCE -- Move to 1st column of next line

```
1 .SBTTL TCESCE -- Move to 1st column of next line
2 ;-----
3 ; Move to first column of next line. Equivalent to carriage-return,
4 ; line-feed.
5 ;
6 ; Inputs:
7 ; R2 = Pointer to window control block.
8 ;
9 002516 004737 002306' TCESCE: CALL TCCR ;Simulate carriage return
10 002522 004737 002454' CALL TCLF ;Simulate line feed
11 002526 000207 RETURN
```

TCIXUP -- Index up one line

```

1
2
3
4
5
6
7
8
9 002530
10
11
12
13
14 002530 016200 000000G
15 002534 020062 000000G
16 002540 001406
17 002542 005300
18 002544 003001
19 002546 005200
20 002550 004737 006710'
21 002554 000402
22
23
24
25
26 002556 004737 005274'
27
28
29
30 002562 000207

```

```

.SBTTL TCIXUP -- Index up one line
-----
; Move the cursor up one line. Scroll the screen if we are on the top line.
;
; Inputs:
; R2 = Pointer to window control block
; R5 = Pointer to parameter value start in control sequence
;
TCIXUP:
;
; If we are not at the top line of a scrolling region,
; simply decrement the line number.
;
; MOV DW$LIN(R2),R0 ;Get current line number
; CMP RO,DW$SRT(R2) ;Are we at top of scrolling region?
; BEQ 1$ ;Br if yes
; DEC RO ;Decrement line number
; BGT 2$ ;Br if didn't go past line 1
; INC RO ;Constrain to line 1
2$: CALL SETLIN ;Set this as current line
; BR 9$
;
; We are at the top line of a scroll region.
; We must scroll down one line.
;
1$: CALL SCRLDN ;Scroll down one line
;
; Finished
;
9$: RETURN

```

TCRIT -- Move cursor right specified number of chars

```

1          .SBTTL  TCRIT  -- Move cursor right specified number of chars
2          ;-----
3          ; Move cursor right a specified number of characters.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;   R5 = Pointer to parameter value in control sequence
8          ;
9 002564   TCRIT:
10          ;
11          ; Get parameter value of number of chars to move
12          ;
13 002564   004737   002116'   CALL    CSPARM          ;Get parameter value
14          ;
15          ; Translate parameter value of 0 to 1
16          ;
17 002570   005700   TST     R0          ;Parameter value zero?
18 002572   001001   BNE    1$          ;Br if not
19 002574   005200   INC     R0          ;Translate to 1
20          ;
21          ; Move the cursor the specified number of columns but don't go past
22          ; right margin.
23          ;
24 002576   066200   000000G   1$:    ADD     DW$COL(R2),R0   ;Get updated cursor column
25 002602   020062   000000G   CMP     R0,DW$CPL(R2)      ;Did we go past right column?
26 002606   101402   BLOS   2$          ;Br if not
27 002610   016200   000000G   MOV     DW$CPL(R2),R0     ;Constrain to right column
28 002614   010062   000000G   2$:    MOV     R0,DW$COL(R2) ;Set new column number
29          ;
30          ; Finished
31          ;
32 002620   000207   RETURN

```


TCLEFT -- Move cursor left specified number of chars

```

1          .SBTTL  TCLEFT -- Move cursor left specified number of chars
2          ;-----
3          ; Move cursor left a specified number of characters.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;   R5 = Pointer to parameter value in control sequence
8          ;
9 002622   TCLEFT:
10         ;
11         ; Get parameter value of number of chars to move
12         ;
13 002622 004737 002116'   CALL    CSPARM      ;Get parameter value
14         ;
15         ; Translate parameter value of 0 to 1
16         ;
17 002626 005700           TST     R0          ;Parameter value zero?
18 002630 001001           BNE     1$          ;Br if not
19 002632 005200           INC     R0          ;Translate to 1
20         ;
21         ; Move the cursor the specified number of columns but don't go past
22         ; left margin.
23         ;
24 002634 005400           1$:   NEG     R0          ;Get negative movement count
25 002636 066200 000000G   ADD     DW$COL(R2),R0   ;Get updated cursor column
26 002642 003002           BGT     2$          ;Br if didn't go past column 1
27 002644 012700 000001   MOV     #1,R0          ;Constrain to column 1
28 002650 010062 000000G   2$:   MOV     R0,DW$COL(R2) ;Set new column number
29         ;
30         ; Finished
31         ;
32 002654 000207           RETURN

```

TCCADR -- Do direct cursor addressing

```

1          .SBTTL  TCCADR -- Do direct cursor addressing
2          ;-----
3          ; Move cursor to a specified line and column.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block.
7          ;   R5 = Pointer to parameter string within control sequence
8          ;
9 002656   TCCADR:
10         ;
11         ; Get line number parameter
12         ;
13 002656 004737 002116'   CALL    CSPARM      ;Accrue line number parameter
14 002662 005700           TST      RO          ;Translate 0 to 1
15 002664 001001           BNE     1$
16 002666 005200           INC     RO          ;Force to 1
17 002670 032762 000000G 000000G 1$:  BIT    #AW#ORS,DW$AW(R2); Is origin relative to scrolling region?
18 002676 001410           BEQ     5$        ;Br if not
19 002700 066200 000000G   ADD     DW$SRT(R2),RO  ;Bias line number by top of scrolling region
20 002704 005300           DEC     RO          ;Top line number is 1
21 002706 020062 000000G   CMP     RO,DW$SRB(R2) ;Don't allow to go past end of scroll region
22 002712 101402           BLOS   5$        ;Br if ok
23 002714 016200 000000G   MOV     DW$SRB(R2),RO ;Constrain to bottom of scroll region
24 002720 020062 000000G   5$:   CMP     RO,DW$LPP(R2) ;Don't go past bottom of page
25 002724 101402           BLOS   2$        ;Br if ok
26 002726 016200 000000G   MOV     DW$LPP(R2),RO ;Constrain to bottom of page
27 002732 004737 006710'   2$:   CALL    SETLIN     ;Set new line as current line
28         ;
29         ; Get column number parameter
30         ;
31 002736 004737 002116'   CALL    CSPARM      ;Accrue column number parameter
32 002742 005700           TST     RO          ;Don't allow column 0
33 002744 001001           BNE     3$        ;Br if non zero
34 002746 005200           INC     RO          ;Translate 0 to 1
35 002750 020062 000000G   3$:   CMP     RO,DW$CPL(R2) ;Don't allow to go past right margin
36 002754 101402           BLOS   4$        ;Br if ok
37 002756 016200 000000G   MOV     DW$CPL(R2),RO ;Constrain to right margin
38 002762 010062 000000G   4$:   MOV     RO,DW$COL(R2) ;Set new column number
39         ;
40         ; Finished
41         ;
42 002766 000207           RETURN

```

TCCPSV -- Save current cursor position

```

1          .SBTTL  TCCPSV -- Save current cursor position
2          ;-----
3          ; Save the current cursor position and character attributes.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block.
7          ;
8 002770   TCCPSV:
9          ;
10         ; Save cursor position
11         ;
12 002770   116262  000000G 000000G      MOVB    DW$LIN(R2),DW$SLN(R2);Save current line number
13 002776   116262  000000G 000000G      MOVB    DW$COL(R2),DW$SCL(R2);Save current column number
14         ;
15         ; Save character attributes
16         ;
17 003004   116262  000000G 000000G      MOVB    DW$CCA(R2),DW$SCA(R2);Save character attribute flags
18         ;
19         ; Finished
20         ;
21 003012   000207      RETURN
22
23         .SBTTL  TCCPRS -- Restore cursor position
24         ;-----
25         ; Restore cursor position and character attributes from saved data.
26         ;
27         ; Inputs:
28         ;   R2 = Pointer to window control block.
29         ;
30 003014   TCCPRS:
31         ;
32         ; Restore the column number
33         ;
34 003014   116262  000000G 000000G      MOVB    DW$SCL(R2),DW$COL(R2);Restore column number
35         ;
36         ; Restore the line number
37         ;
38 003022   116200  000000G      MOVB    DW$SLN(R2),R0    ;Get saved line number
39 003026   004737  006710'      CALL    SETLIN        ;Set as current line number
40         ;
41         ; Restore character attributes
42         ;
43 003032   116262  000000G 000000G      MOVB    DW$SCA(R2),DW$CCA(R2);Restore character attribute flags
44         ;
45         ; Finished
46         ;
47 003040   000207      RETURN

```

TCINSL -- Insert lines

```

1          .SBTTL  TCINSL -- Insert lines
2          ;-----
3          ;  Insert a specified number of lines in front of the line in which
4          ;  the cursor is currently resting.  The cursor is left positioned
5          ;  at column 1 of the first inserted line.
6          ;
7          ;  Inputs:
8          ;  R2 = Pointer to window control block
9          ;  R5 = Pointer to command string argument
10         ;
11 003042 010146 TCINSL: MOV     R1,-(SP)
12 003044 010346          MOV     R3,-(SP)
13 003046 010446          MOV     R4,-(SP)
14 003050 010546          MOV     R5,-(SP)
15         ;
16         ;  Ignore this command if we are not within the scroll region
17         ;
18 003052 016203 000000G          MOV     DW$LIN(R2),R3  ;Get current line number
19 003056 020362 000000G          CMP     R3,DW$SRT(R2)  ;Are we above top to scroll region?
20 003062 103444          BLD     9$              ;Br if yes
21 003064 020362 000000G          CMP     R3,DW$SRB(R2)  ;Are we below bottom of scroll region?
22 003070 101041          BHI     9$              ;Br if yes
23         ;
24         ;  Determine how many lines will be inserted
25         ;
26 003072 004737 002116'          CALL    CSPARM          ;Get command parameter
27 003076 005700          TST     R0              ;Was 0 specified?
28 003100 003002          BGT     1$              ;Br if not
29 003102 012700 000001          MOV     #1,R0          ;Set it to 1
30 003106 016201 000000G 1$:  MOV     DW$SRB(R2),R1  ;Get bottom line # of scroll region
31 003112 160301          SUB     R3,R1          ;Get # lines from current to end of region
32 003114 005201          INC     R1              ;This is max # inserted lines
33 003116 020100          CMP     R1,R0          ;Can we scroll as many as requested?
34 003120 101413          BLOS   2$              ;Br if all lines are scrolled off
35         ;
36         ;  Scroll down lines to make room for inserted lines
37         ;
38 003122 010103          MOV     R1,R3          ;Get number of lines to scroll
39 003124 160003          SUB     R0,R3
40 003126 010001          MOV     R0,R1          ;Get # lines being inserted
41 003130 016205 000000G          MOV     DW$SRB(R2),R5 ;Get # of line at bottom of scroll reg
42 003134 010504 3$:  MOV     R5,R4
43 003136 160104          SUB     R1,R4          ;Get # of line to copy to bottom
44 003140 004737 006354'          CALL    CPYLIN        ;Copy a line down
45 003144 005305          DEC     R5              ;Get next destination line number
46 003146 077306          SOB    R3,3$          ;Loop if more lines to scroll down
47         ;
48         ;  Blank fill inserted lines
49         ;
50 003150 016203 000000G 2$:  MOV     DW$LIN(R2),R3  ;Get top line to blank
51 003154 010300 4$:  MOV     R3,R0          ;Get # of line to clear
52 003156 004737 006240'          CALL    CLRLIN        ;Clear that line
53 003162 005203          INC     R3              ;Advance line number
54 003164 077105          SOB    R1,4$          ;Loop if more lines to clear
55         ;
56         ;  Move cursor to column 1
57         ;

```

TCINSL -- Insert lines

```
58 003166 012762 000001 000000G      MOV      #1,DW$COL(R2) ; Say cursor is at column 1
59                                     ;
60                                     ; Finished
61                                     ;
62 003174 012605      9$:      MOV      (SP)+,R5
63 003176 012604      MOV      (SP)+,R4
64 003200 012603      MOV      (SP)+,R3
65 003202 012601      MOV      (SP)+,R1
66 003204 000207      RETURN
```

TCDELL -- Delete specified number of lines

```

1          .SBTTL  TCDELL -- Delete specified number of lines
2          ;-----
3          ; Delete the specified number of lines starting with the one in which
4          ; the cursor is positioned.
5          ;
6          ; Inputs:
7          ; R2 = Pointer to window control block
8          ; R5 = Pointer to command string argument
9          ;
10         003206 010146 TCDELL: MOV      R1,-(SP)
11         003210 010346          MOV      R3,-(SP)
12         003212 010446          MOV      R4,-(SP)
13         003214 010546          MOV      R5,-(SP)
14         ;
15         ; Ignore this command if we are not within the scroll region
16         ;
17         003216 016203 000000G          MOV      DW$LIN(R2),R3 ;Get current line number
18         003222 020362 000000G          CMP      R3,DW$SRT(R2) ;Are we above top to scroll region?
19         003226 103442          BLO     9$ ;Br if yes
20         003230 020362 000000G          CMP      R3,DW$SRB(R2) ;Are we below bottom of scroll region?
21         003234 101037          BHI     9$ ;Br if yes
22         ;
23         ; Determine how many lines will be deleted
24         ;
25         003236 004737 002116'          CALL     CSPARM ;Get command parameter
26         003242 005700          TST     R0 ;Was 0 specified?
27         003244 003002          BGT     1$ ;Br if not
28         003246 012700 000001          MOV     #1,R0 ;Set it to 1
29         003252 016201 000000G 1$: MOV     DW$SRB(R2),R1 ;Get bottom line # of scroll region
30         003256 160301          SUB     R3,R1 ;Get # lines from current to end of region
31         003260 005201          INC     R1 ;This is max # deleted lines
32         003262 020100          CMP     R1,R0 ;Can we delete as many as requested?
33         003264 101412          BLOS   2$ ;Br if all lines are deleted
34         ;
35         ; Scroll up the remaining lines.
36         ;
37         003266 010305          MOV     R3,R5 ;Get current line number
38         003270 010103          MOV     R1,R3 ;Get number of lines to scroll
39         003272 160003          SUB     R0,R3
40         003274 010001          MOV     R0,R1 ;Get # lines being inserted
41         003276 010504 3$: MOV     R5,R4
42         003300 060104          ADD     R1,R4 ;Get line to copy from
43         003302 004737 006354'          CALL   CPYLIN ;Copy a line up
44         003306 005205          INC     R5 ;Advance line number
45         003310 077306          SOB    R3,3$ ;Br if more lines to copy
46         ;
47         ; Blank fill lines at end of scrolling region
48         ;
49         003312 016203 000000G 2$: MOV     DW$SRB(R2),R3 ;Get # of line at bottom of region
50         003316 010300 4$: MOV     R3,R0 ;Get # of line to blank fill
51         003320 004737 006240'          CALL   CLRLIN ;Blank the line
52         003324 077104          SOB    R1,4$ ;Loop if more lines to blank
53         ;
54         ; Say cursor is at column 1
55         ;
56         003326 012762 000001 000000G          MOV     #1,DW$COL(R2) ;Say cursor is at column 1
57         ;

```

```
58 ; Finished
59 ;
60 003334 012605 9$: MOV (SP)+, R5
61 003336 012604 MOV (SP)+, R4
62 003340 012603 MOV (SP)+, R3
63 003342 012601 MOV (SP)+, R1
64 003344 000207 RETURN
```

TCGOAS -- Designate GO as ascii character set

```

1          .SBTTL  TCGOAS -- Designate GO as ascii character set
2          ;-----
3          ; Designate the GO character set as ascii characters.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block
7          ;
8 003346 105062 000000G TCGOAS: CLRB   DW$GOM(R2)   ;Set character set number
9 003352 000207          RETURN

```

```

10         .SBTTL  TCGODS -- Designate GO as DEC supplemental character set
11         ;-----
12         ; Designate the GO character set as DEC supplemental character set.
13         ;
14         ; Inputs:
15         ; R2 = Pointer to window control block.
16         ;
17         ;
18 003354 112762 000001 000000G TCGODS: MOVB   #1,DW$GOM(R2) ;Set character set number
19 003362 000207          RETURN

```

```

20         .SBTTL  TCGOUK -- Designate GO as UK national character set
21         ;-----
22         ; Designate the GO character set as UK national character set.
23         ;
24         ; Inputs:
25         ; R2 = Pointer to window control block.
26         ;
27         ;
28 003364 112762 000002 000000G TCGOUK: MOVB   #2,DW$GOM(R2) ;Set character set number
29 003372 000207          RETURN

```

```

30         .SBTTL  TCGOGR -- Designate GO as graphics character set
31         ;-----
32         ; Designate the GO character set as graphics characters.
33         ;
34         ; Inputs:
35         ; R2 = Pointer to window control block.
36         ;
37         ;
38 003374 112762 000003 000000G TCGOGR: MOVB   #3,DW$GOM(R2) ;Set character set number
39 003402 000207          RETURN

```


TCG1AS -- Designate G1 as ascii character set

```

1          .SBTTL  TCG1AS -- Designate G1 as ascii character set
2          ;-----
3          ; Designate the G1 character set as ascii characters.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block
7          ;
8 003404 105062 000000G TCG1AS: CLRB   DW$G1M(R2)   ;Set character set number
9 003410 000207          RETURN

```

```

10         .SBTTL  TCG1DS -- Designate G1 as DEC supplemental character set
11        ;-----
12        ; Designate the G1 character set as DEC supplemental character set.
13        ;
14        ; Inputs:
15        ; R2 = Pointer to window control block.
16        ;
17        ;
18 003412 112762 000001 000000G TCG1DS: MOVB  #1,DW$G1M(R2) ;Set character set number
19 003420 000207          RETURN

```

```

20        .SBTTL  TCG1UK -- Designate G1 as UK national character set
21        ;-----
22        ; Designate the G1 character set as UK national character set.
23        ;
24        ; Inputs:
25        ; R2 = Pointer to window control block.
26        ;
27        ;
28 003422 112762 000002 000000G TCG1UK: MOVB  #2,DW$G1M(R2) ;Set character set number
29 003430 000207          RETURN

```

```

30        .SBTTL  TCG1GR -- Designate G1 as graphics character set
31        ;-----
32        ; Designate the G1 character set as graphics characters.
33        ;
34        ; Inputs:
35        ; R2 = Pointer to window control block.
36        ;
37        ;
38 003432 112762 000003 000000G TCG1GR: MOVB  #3,DW$G1M(R2) ;Set character set number
39 003440 000207          RETURN

```

TCG2AS -- Designate G2 as ascii character set

```

1          .SBTTL  TCG2AS -- Designate G2 as ascii character set
2          ;-----
3          ; Designate the G2 character set as ascii characters.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block
7          ;
8 003442 105062 000000G TCG2AS: CLRB   DW$G2M(R2)   ;Set character set number
9 003446 000207          RETURN

```

```

10         .SBTTL  TCG2DS -- Designate G2 as DEC supplemental character set
11        ;-----
12        ; Designate the G2 character set as DEC supplemental character set.
13        ;
14        ; Inputs:
15        ; R2 = Pointer to window control block.
16        ;
17        ;
18 003450 112762 000001 000000G TCG2DS: MOVB   #1,DW$G2M(R2) ;Set character set number
19 003456 000207          RETURN

```

```

20        .SBTTL  TCG2GR -- Designate G2 as graphics character set
21        ;-----
22        ; Designate the G2 character set as graphics characters.
23        ;
24        ; Inputs:
25        ; R2 = Pointer to window control block.
26        ;
27        ;
28 003460 112762 000003 000000G TCG2GR: MOVB   #3,DW$G2M(R2) ;Set character set number
29 003466 000207          RETURN

```

TCG3AS -- Designate G3 as ascii character set

```

1          .SBTTL  TCG3AS -- Designate G3 as ascii character set
2          ;-----
3          ; Designate the G3 character set as ascii characters.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block
7          ;
8 003470 105062 000000G TCG3AS: CLRB   DW$G3M(R2)   ;Set character set number
9 003474 000207          RETURN

```

```

10         .SBTTL  TCG3DS -- Designate G3 as DEC supplemental character set
11         ;-----
12         ; Designate the G3 character set as DEC supplemental character set.
13         ;
14         ; Inputs:
15         ; R2 = Pointer to window control block.
16         ;
17         ;
18 003476 112762 000001 000000G TCG3DS: MOVB  #1,DW$G3M(R2) ;Set character set number
19 003504 000207          RETURN

```

```

20         .SBTTL  TCG3GR -- Designate G3 as graphics character set
21         ;-----
22         ; Designate the G3 character set as graphics characters.
23         ;
24         ; Inputs:
25         ; R2 = Pointer to window control block.
26         ;
27         ;
28 003506 112762 000003 000000G TCG3GR: MOVB  #3,DW$G3M(R2) ;Set character set number
29 003514 000207          RETURN

```

TCSO -- Shift-out character - Lock shift G1 to GL

```

1          .SBTTL TCSO  -- Shift-out character - Lock shift G1 to GL
2          ;-----
3          ; The shift-out character is used to lock character set G1 to GL.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block
7          ;
8 003516 112762 000001 000000G TCSO:  MOVB   #1,DW$GLM(R2) ;Map GL to G1
9 003524 000207                RETURN
10
11         .SBTTL TCSI  -- Shift-in character - Lock shift G0 to GL
12         ;-----
13         ; The shift-in character is used to lock character set G0 to GL.
14         ;
15         ; Inputs:
16         ; R2 = Pointer to window control block.
17         ;
18 003526 105062 000000G TCSI:  CLR   DW$GLM(R2) ;Map GL to G0
19 003532 000207                RETURN
20
21         .SBTTL TCLS2 -- Lock shift G2 to GL
22         ;-----
23         ; Set G2 as current GL character set.
24         ;
25         ; Inputs:
26         ; R2 = Pointer to window control block.
27         ;
28 003534 112762 000002 000000G TCLS2: MOV   #2,DW$GLM(R2) ;Map GL to G2
29 003542 000207                RETURN
30
31         .SBTTL TCLS3 -- Lock shift G3 to GL
32         ;-----
33         ; Set G3 as current GL character set.
34         ;
35         ; Inputs:
36         ; R2 = Pointer to window control block.
37         ;
38 003544 112762 000003 000000G TCLS3: MOV   #3,DW$GLM(R2) ;Map GL to G3
39 003552 000207                RETURN
40
41         .SBTTL TCLS1R -- Lock shift G1 into GR
42         ;-----
43         ; Set G1 as current GR character set.
44         ;
45         ; Inputs:
46         ; R2 = Pointer to window control block.
47         ;
48 003554 112762 000001 000000G TCLS1R: MOV   #1,DW$GRM(R2) ;Map GR to G1
49 003562 000207                RETURN
50
51         .SBTTL TCLS2R -- Lock shift G2 into GR
52         ;-----
53         ; Set G2 as the GR character set.
54         ;
55         ; Inputs:
56         ; R2 = Pointer to window control block.
57         ;

```

TCLS2R -- Lock shift G2 into GR

```
58 003564 112762 000002 000000G TCLS2R: MOVB #2,DW$GRM(R2) ;Map GR to G2
59 003572 000207 RETURN
```

60

61

```
.SBTTL TCLS3R -- Lock shift G3 into GR
```

62

```
-----
```

63

```
; Set G3 as the GR character set.
```

64

```
;
```

65

```
; Inputs:
```

66

```
; R2 = Pointer to window control block.
```

67

```
;
```

```
68 003574 112762 000003 000000G TCLS3R: MOVB #3,DW$GRM(R2) ;Map GR to G3
```

```
69 003602 000207 RETURN
```

70

71

```
.SBTTL TCSS2 -- Single shift G2 to GL
```

72

```
-----
```

73

```
; Set G2 as GL for next character only.
```

74

```
;
```

75

```
; Inputs:
```

76

```
; R2 = Pointer to window control block.
```

77

```
;
```

```
78 003604 112762 000002 000000G TCSS2: MOVB #2,DW$GLS(R2) ;Map GL to G2 for next character only
```

```
79 003612 052762 000000G 000000G BIS #AW$SS,DW$AW(R2);Remember single shift is pending
```

```
80 003620 000207 RETURN
```

81

82

```
.SBTTL TCSS3 -- Single shift G3 to GL
```

83

```
-----
```

84

```
; Set G3 as GL for next character only.
```

85

```
;
```

86

```
; Inputs:
```

87

```
; R2 = Pointer to window control block.
```

88

```
;
```

```
89 003622 112762 000003 000000G TCSS3: MOVB #3,DW$GLS(R2) ;Map GL to G3 for next character only
```

```
90 003630 052762 000000G 000000G BIS #AW$SS,DW$AW(R2);Remember single shift is pending
```

```
91 003636 000207 RETURN
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9 003640 010346  
10 003642 012703 004120'  
11 003646 004737 003730'  
12 003652 012603  
13 003654 000207  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23 003656 010346  
24 003660 012703 004120'  
25 003664 004737 003766'  
26 003670 012603  
27 003672 000207  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37 003674 010346  
38 003676 012703 004126'  
39 003702 004737 003730'  
40 003706 012603  
41 003710 000207  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51 003712 010346  
52 003714 012703 004126'  
53 003720 004737 003766'  
54 003724 012603  
55 003726 000207
```

```
.SBTTL TCSAA -- Set ANSI attribute  
-----  
; Set an ANSI terminal attribute.  
;  
; Inputs:  
; R2 = Pointer to window control block  
; R5 = Pointer to parameter values  
;  
TCSAA: MOV R3, -(SP)  
MOV #ASATAT, R3 ;Point to ANSI conversion table  
CALL TCSTA ;Go set the attribute  
MOV (SP)+, R3  
RETURN
```

```
.SBTTL TCRAA -- Reset ANSI attribute  
-----  
; Reset an ANSI terminal attribute.  
;  
; Inputs:  
; R2 = Pointer to window control block  
; R5 = Pointer to parameter values  
;  
TCRAA: MOV R3, -(SP)  
MOV #ASATAT, R3 ;Point to ANSI conversion table  
CALL TCRTA ;Reset the attribute  
MOV (SP)+, R3  
RETURN
```

```
.SBTTL TCSDA -- Set DEC attribute  
-----  
; Set a DEC terminal attribute  
;  
; Inputs:  
; R2 = Pointer to window control block  
; R5 = Pointer to parameter values  
;  
TCSDA: MOV R3, -(SP)  
MOV #DECTAT, R3 ;Point to DEC conversion table  
CALL TCSTA ;Set the attribute  
MOV (SP)+, R3  
RETURN
```

```
.SBTTL TCRDA -- Reset DEC attribute  
-----  
; Reset a DEC terminal attribute  
;  
; Inputs:  
; R2 = Pointer to window control block  
; R5 = Pointer to parameter values  
;  
TCRDA: MOV R3, -(SP)  
MOV #DECTAT, R3 ;Point to DEC conversion table  
CALL TCRTA ;Reset the attribute  
MOV (SP)+, R3  
RETURN
```

TCSTA -- Set terminal attribute

```

1          .SBTTL  TCSTA  -- Set terminal attribute
2          ;-----
3          ; Set a terminal attribute
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;   R3 = Pointer to table to convert attribute value to flag bit
8          ;   R5 = Pointer to parameter values
9          ;
10         003730  010446  TCSTA:  MOV      R4, -(SP)
11         ;
12         ; Begin loop to accrue attribute parameters until end of string hit
13         ;
14         003732  004737  002116'  1$:   CALL    CSPARM      ;Accrue next parameter value
15         003736  103411          BCS     9$           ;Br if no more parameter values
16         ;
17         ; Convert parameter value into flag mask
18         ;
19         003740  010004          MOV     RO, R4       ;Save parameter value
20         003742  004737  004066'  CALL    CVTTAV      ;Convert parameter value into flag mask
21         003746  103771          BCS     1$           ;Br if parameter not recognized
22         ;
23         ; Set appropriate flag bit
24         ;
25         003750  050062  0000000  BIS     RO, DW$AW(R2) ;Set appropriate attribute flag
26         ;
27         ; See if this change requires doing more than setting a flag bit
28         ;
29         003754  004737  004160'  CALL    CKTAC       ;Check for additional changes
30         ;
31         ; See if there are more attributes to set
32         ;
33         003760  000764          BR      1$
34         ;
35         ; Finished
36         ;
37         003762  012604  9$:   MOV     (SP)+, R4
38         003764  000207          RETURN

```

TCRTA -- Reset terminal attribute

```

1
2
3
4
5
6
7
8
9
10 003766 010446
11
12
13
14 003770 004737 002116'
15 003774 103432
16
17
18
19 003776 020327 004120'
20 004002 001416
21 004004 020027 000002
22 004010 001013
23 004012 052762 000000G 000000G
24 004020 116204 000000G
25 004024 116404 000000G
26 004030 052764 000000G 000000G
27 004036 000754
28
29
30
31 004040 010004
32 004042 004737 004066'
33 004046 103750
34
35
36
37 004050 040062 000000G
38
39
40
41 004054 004737 004160'
42
43
44
45 004060 000743
46
47
48
49 004062 012604
50 004064 000207

```

```

.SBTTL TCRTA -- Reset terminal attribute
-----
; Reset a terminal attribute
;
; Inputs:
; R2 = Pointer to window control block
; R3 = Pointer to table to convert attribute value to flag bit
; R5 = Pointer to parameter values
;
TCRTA: MOV R4, -(SP)
;
; Begin loop to accrue attribute parameters until end of string hit
;
1$: CALL CSPARM ;Accrue next parameter value
BCS 9$ ;Br if no more parameters
;
; Check for special parameter value 2 which means select VT52 mode
;
CMP R3, #ASATAT ;ANSI or DEC reset mode
BEQ 2$ ;BR if ANSID
CMP R0, #2 ;Select VT52 mode?
BNE 2$ ;Br if not
BIS #AW$S52, DW$AW(R2); Enable VT52 simulation mode for window
MOVB DW$JOB(R2), R4 ;Get out job index number
MOVB LNPRIM(R4), R4 ;Get our primary job #
BIS #$V52EM, LSW11(R4); Say we are emulating a VT52
BR 1$
;
; Convert parameter value into flag mask
;
2$: MOV R0, R4 ;Save parameter value
CALL CVTTAV ;Convert parameter value into flag mask
BCS 1$ ;Br if parameter value not recognized
;
; Reset appropriate flag bit
;
BIC R0, DW$AW(R2) ;Reset appropriate attribute flag
;
; See if this change requires doing more than clearing a bit
;
CALL CKTAC ;Check for special mode changes
;
; See if there are more attributes to reset
;
BR 1$
;
; Finished
;
9$: MOV (SP)+, R4
RETURN

```


CVTTAV -- Convert terminal attribute value to flag

```

1          .SBTTL  CVTTAV -- Convert terminal attribute value to flag
2          ;-----
3          ; Convert a terminal attribute parameter value into a flag mask.
4          ;
5          ; Inputs:
6          ;   R0 = Attribute value
7          ;   R3 = Pointer to table to convert attribute value to flag bit
8          ;
9          ; Outputs:
10         ;   C-flag set ==> Attribute value not recognized
11         ;   R0 = Flag mask if attribute value recognized
12         ;
13 004066 010346 CVTTAV: MOV     R3,-(SP)
14         ;
15         ; See if we can find attribute value in table
16         ;
17 004070 020023 1$:      CMP     R0,(R3)+      ;Is next entry for our value?
18 004072 001406      BEQ     2$              ;Br if yes
19 004074 005723      TST     (R3)+          ;Skip over flag mask
20 004076 005713      TST     (R3)          ;Is there another entry in table?
21 004100 001373      BNE     1$              ;Br if yes
22         ;
23         ; Attribute value is not in table
24         ;
25 004102 005000      CLR     R0              ;Return no flag mask
26 004104 000261      SEC              ;Signal failure on return
27 004106 000402      BR     9$
28         ;
29         ; We found the attribute in the table
30         ;
31 004110 011300 2$:      MOV     (R3),R0      ;Get the flag mask
32 004112 000241      CLC              ;Signal success on return
33         ;
34         ; Finished
35         ;
36 004114 012603 9$:      MOV     (SP)+,R3
37 004116 000207      RETURN
38         ;-----
39         ;
40         ; Table to relate ANSI attribute values with flag masks
41         ;
42 004120 000004 000000G ASATAT: .WORD 4.,AW$INS      ;Insert mode
43 004124 000000      .WORD 0
44         ;
45         ; Table to relate DEC attribute values with flag masks
46         ;
47 004126 000001 000000G DECTAT: .WORD 1.,AW$ACK      ;Application mode for cursor keys
48 004132 000003 000000G      .WORD 3.,AW$132      ;132 column mode
49 004136 000005 000000G      .WORD 5.,AW$REV      ;Reverse video
50 004142 000006 000000G      .WORD 6.,AW$ORS      ;Origin relative to scroll region
51 004146 000010 000000G      .WORD 8.,AW$RPT      ;Automatic keypad repeat
52 004152 000031 000000G      .WORD 25.,AW$VCR      ;Make cursor visible
53 004156 000000      .WORD 0              ;End of table

```

CKTAC -- Check for complex mode changes

```

1          .SBTTL  CKTAC  -- Check for complex mode changes
2          ;-----
3          ; This routine is called each time a terminal attribute changes to see
4          ; if we must do things other than setting or clearing an attribute flag.
5          ;
6          ; Inputs:
7          ;   R2 = Pointer to window control block
8          ;   R4 = Parameter value specified with attribute change command.
9          ;
10         CKTAC:
11         ;
12         ; If we just changed the origin mode, home the cursor
13         ;
14         004160 020427 000006          CMP     R4,#6          ;Change origin mode?
15         004164 001003          BNE     1$          ;Br if not
16         004166 004737 005116'      CALL   SETHOM        ;Say cursor has moved to home
17         004172 000415          BR     9$
18         ;
19         ; If we just changed line width (80/132 mode), clear the screen,
20         ; set scrolling region to full screen, and home the cursor.
21         ;
22         004174 020427 000003      1$:   CMP     R4,#3          ;Change line width?
23         004200 001012          BNE     9$          ;Br if not
24         004202 012762 000001 000000G  MOV     #1,DW$SRT(R2) ;Set top scroll line to 1
25         004210 016262 000000G 000000G  MOV     DW$LPP(R2),DW$SRB(R2) ;Set bottom scroll line to end of page
26         004216 004737 005116'      CALL   SETHOM        ;Set cursor to home position
27         004222 004737 006216'      CALL   ERSPAG        ;Erase the entire page
28         ;
29         ; Finished
30         ;
31         004226 000207      9$:   RETURN

```

TCAKM -- Select application keypad mode

```
1 .SBTTL TCAKM -- Select application keypad mode
2 ;-----
3 ; Select application keypad mode
4 ;
5 ; Inputs:
6 ; R2 = Pointer to window control block
7 ;
8 004230 052762 000000G 000000G TCAKM: BIS #AW$AKM,DW$AW(R2);Say we are in application keypad mode
9 004236 000207 RETURN
10
11 .SBTTL TCNKM -- Select numeric keypad mode
12 ;-----
13 ; Select numeric keypad mode
14 ;
15 ; Inputs:
16 ; R2 = Pointer to window control block
17 ;
18 004240 042762 000000G 000000G TCNKM: BIC #AW$AKM,DW$AW(R2);Reset application keypad mode
19 004246 000207 RETURN
```

TCN3 -- Set line attributes

```

1
2
3
4
5
6
7
8
9
10 004250 004737 006620'
11 004254 042700 000000G
12 004260 052700 000000G
13 004264 004737 006530'
14 004270 000207
15
16
17
18 004272 004737 006620'
19 004276 042700 000000G
20 004302 052700 000000G
21 004306 004737 006530'
22 004312 000207
23
24
25
26 004314 004737 006620'
27 004320 042700 000000G
28 004324 004737 006530'
29 004330 000207
30
31
32
33 004332 004737 006620'
34 004336 052700 000000G
35 004342 004737 006530'
36 004346 000207

```

```

.SBTTL TCN3 -- Set line attributes
-----
; Set attributes for the current line.
;
; Inputs:
; R2 = Pointer to window control block
;
; Line is top half of double-high line
TCN3: CALL GETLAB ;Get current line attribute flags
      BIC #AL$DHB,RO ;Clear double-high-bottom flag
      BIS #AL$DHT,RO ;Set double-high-top flag
      CALL SETLAB ;Store new attribute flags for line
      RETURN
;
; Line is bottom half of double-high line
TCN4: CALL GETLAB ;Get current line attribute flags
      BIC #AL$DHT,RO ;Clear double-high-top flag
      BIS #AL$DHB,RO ;Set double-high-bottom flag
      CALL SETLAB ;Store new attribute flags for line
      RETURN
;
; Line is single width
TCN5: CALL GETLAB ;Get current line attribute flags
      BIC #AL$DWD,RO ;Clear double-wide attribute
      CALL SETLAB ;Store new attribute flags for line
      RETURN
;
; Line is double width
TCN6: CALL GETLAB ;Get current line attribute flags
      BIS #AL$DWD,RO ;Set double-wide flag
      CALL SETLAB ;Store new attribute flags for line
      RETURN

```

TCSCA -- Set character attributes

```

1          .SBTTL  TCSCA  -- Set character attributes
2          ;-----
3          ; Specify attributes for characters that follow
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block
7          ; R5 = Pointer to start of parameter values
8          ;
9 004350 010346 TCSCA:  MOV      R3, -(SP)
10         ;
11         ; If no parameters were specified, reset all character attributes
12         ;
13 004352 121527 000155          CMPB   (R5), #'m          ; Any attributes specified?
14 004356 001004          BNE    1$              ; Br if yes
15 004360 142762 000000C 000000G BICB   #<AC$BLD!AC$BLK!AC$REV!AC$ULN>, DW$CCA(R2); Clear all attributes
16 004366 000422          BR     9$              ; Finished
17         ;
18         ; Begin loop to accrue and process each parameter
19         ;
20 004370 004737 002116' 1$:    CALL   CSPARM          ; Accrue next parameter value
21 004374 103417          BCS    9$              ; Br if no more parameter values
22         ;
23         ; Look up parameter value to determine which attribute flags
24         ; are to be set and reset.
25         ;
26 004376 012703 004440'          MOV    #CATABL, R3        ; Point to attribute table
27 004402 120023 2$:    CMPB   R0, (R3)+        ; Search for parameter value in table
28 004404 001406          BEQ    3$              ; Br if found it
29 004406 062703 000002          ADD   #2, R3           ; Skip over flag bytes
30 004412 020327 004473'          CMP   R3, #CATEND      ; Reached end of table?
31 004416 103771          BLD   2$              ; Keep looking if not
32 004420 000763          BR    1$              ; Cannot find this parameter value
33         ;
34         ; Found parameter value in table.
35         ; Set and reset some flags
36         ;
37 004422 152362 000000G 3$:    BISB   (R3)+, DW$CCA(R2); Set some attributes
38 004426 142362 000000G          BICB   (R3)+, DW$CCA(R2); Reset some attributes
39 004432 000756          BR    1$              ; Process any other attributes
40         ;
41         ; Finished
42         ;
43 004434 012603 9$:    MOV    (SP)+, R3
44 004436 000207          RETURN

```

TCSCA -- Set character attributes

```

1          ; -----
2          ; Table to convert attribute parameter values into flags to be
3          ; set and reset.
4          ; Each entry has three values: (1) numeric parameter value that
5          ; occurs in control sequence; (2) flags to be set; (3) flags to
6          ; be reset.
7          ;
8 004440    000    000    000C CATABL: .BYTE 0.,0,AC$BLD!AC$BLK!AC$REV!AC$ULN
9 004443    001    000G 000      .BYTE 1.,AC$BLD,0
10 004446   004    000G 000      .BYTE 4.,AC$ULN,0
11 004451   005    000G 000      .BYTE 5.,AC$BLK,0
12 004454   007    000G 000      .BYTE 7.,AC$REV,0
13 004457   026    000    000G    .BYTE 22.,0,AC$BLD
14 004462   030    000    000G    .BYTE 24.,0,AC$ULN
15 004465   031    000    000G    .BYTE 25.,0,AC$BLK
16 004470   033    000    000G    .BYTE 27.,0,AC$REV
17          CATEND:
18          .EVEN

```

TC100 -- Select VT100/VT200 mode

```

1          .SBTTL  TC100  -- Select VT100/VT200 mode
2          ;-----
3          ; Set terminal to VT100 or VT200 mode of operation
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;   R5 = Pointer to parameter values
8          ;
9 004474   TC100:
10         ;
11         ; Say terminal is not in VT52 mode and not in VT200 mode
12         ; (i.e., set to VT100 mode)
13         ;
14 004474   042762   000000C 000000G      BIC      #<AW$52!AW$S52!AW$200>,DW$AW(R2) ;Terminal is in VT100 mode
15         ;
16         ; Reset VT52 emulation-mode flag in entry for primary line so that
17         ; other windows connected to this terminal will know.
18         ;
19 004502   116200   000000G      MOVB     DW$JOB(R2),RO      ;Get our line index number
20 004506   116000   000000G      MOVB     LNPRIM(R0),RO     ;Get number of primary line
21 004512   042760   000000G 000000G      BIC      #$V52EM,LSW11(R0) ;Say terminal not emulating a VT52
22         ;
23         ; If job's terminal type is VT200, set that mode
24         ;
25 004520   116200   000000G      MOVB     DW$JOB(R2),RO     ;Get job index number
26 004524   032760   000000C 000000G      BIT      #<VT2007!VT2008>,LTRMTP(R0); Is terminal a VT200?
27 004532   001005          BNE      1$              ;Br if yes
28         ;
29         ; Get 1st parameter value and see if we should go to VT200 mode
30         ;
31 004534   004737   002116'      CALL     CSPARM           ;Accrue 1st parameter value
32 004540   020027   000076      CMP      RO,#62.         ;VT200 mode?
33 004544   001003          BNE      9$              ;Br if not
34         ;
35         ; Set terminal type to VT200
36         ;
37 004546   052762   000000G 000000G 1$:   BIS      #AW$200,DW$AW(R2); Set VT200 mode
38         ;
39         ; Finished
40         ;
41 004554   000207          9$:   RETURN

```

```

1          .SBTTL  TCREST -- Terminal reset
2          ;-----
3          ; Perform a terminal reset operation
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;
8 004556 010146 TCREST: MOV      R1,-(SP)
9          ;
10         ; Turn off VT52 emulation mode
11         ;
12 004560 042762 000000G 000000G      BIC      #AW$S52,DW$AW(R2);Turn off attribute for window
13 004566 116201 000000G      MOVVB   DW$JOB(R2),R1 ;Get our job index number
14 004572 116100 000000G      MOVVB   LNPRIM(R1),RO ;Get primary line number
15 004576 042760 000000G 000000G      BIC      #$V52EM,LSW11(RO) ;Say terminal is not in VT52 mode
16         ;
17         ; Reset scrolling region
18         ;
19 004604 012762 000001 000000G      MOV      #1,DW$SRT(R2) ;Set top scroll line to 1
20 004612 016262 000000G 000000G      MOV      DW$LPP(R2),DW$SRB(R2) ;Set bottom scroll line to end of page
21         ;
22         ; Reset misc window status flags
23         ;
24 004620 042762 000000C 000000G      BIC      #<AW$132!AW$INS!AW$ACK!AW$ORS!AW$AKM!AW$SS!AW$PRT>,DW$AW(R2)
25         ;
26         ; Say cursor is at home
27         ;
28 004626 004737 005116'          CALL     SETHOM          ;Set cursor to home position
29         ;
30         ; Erase the entire page
31         ;
32 004632 004737 006216'          CALL     ERSPAG          ;Erase the entire page
33         ;
34         ; Reset character set mapping for a VT200
35         ;
36 004636 032761 000000C 000000G      BIT      #<VT2007!VT2008>,LTRMTP(R1) ;Is this a VT200?
37 004644 001411          BEQ      9$          ;Br if not
38 004646 112762 000001 000000G      MOVVB   #1,DW$G2M(R2) ;Map G2 to DEC supplemental chars
39 004654 112762 000001 000000G      MOVVB   #1,DW$G3M(R2) ;Map G3 to DEC supplemental chars
40 004662 112762 000002 000000G      MOVVB   #2,DW$GRM(R2) ;Map GR to G2
41         ;
42         ; Finished
43         ;
44 004670 012601          9$: MOV      (SP)+,R1
45 004672 000207          RETURN

```


TCPRT -- Printer control operation

```

1          .SBTTL  TCPRT  -- Printer control operation
2          ;-----
3          ; Direct output to printer or back to terminal.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;   R5 = Pointer to start of parameter value
8          ;
9 004674   TCPRT:
10         ;
11         ; Get first parameter value and see if we should direct output to printer
12         ; or to terminal.
13         ;
14 004674  004737  002116'      CALL    CSPARM      ;Get first parameter value
15 004700  020027  000005      CMP     RO,#5      ;Direct output to printer?
16 004704  001004              BNE     1$          ;Br if not
17 004706  052762  000000G 000000G  BIS    #AW$PRT,DW$AW(R2);Direct output to printer
18 004714  000406              BR     9$          ;
19 004716  020027  000004      1$:    CMP     RO,#4      ;Direct output to terminal?
20 004722  001003              BNE     9$          ;Br if not
21 004724  042762  000000G 000000G  BIC    #AW$PRT,DW$AW(R2);Direct output to terminal
22         ;
23         ; Finished
24         ;
25 004732  000207      9$:    RETURN

```

TCLERS -- Erase within a line

```

1          .SBTTL  TCLERS -- Erase within a line
2          ;-----
3          ; Erase within a line.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block
7          ; R5 = Pointer to start of parameter value
8          ;
9 004734   TCLERS:
10         ;
11         ; Accrue parameter value to determine what type of erase this is
12         ;
13 004734 004737 002116'      CALL    CSPARM      ; Accrue parameter value
14 004740 005700              TST      R0          ; Parameter = 0?
15 004742 001003              BNE     1$          ; Br if not
16 004744 004737 005640'      CALL    ERSCTL      ; Erase from cursor to end of line
17 004750 000413              BR      9$
18 004752 020027 000001      1$:    CMP     R0,#1      ; Parameter = 1?
19 004756 001003              BNE     2$          ; Br if not
20 004760 004737 005752'      CALL    ERSLTC      ; Erase from beginning of line to cursor
21 004764 000405              BR      9$
22 004766 020027 000002      2$:    CMP     R0,#2      ; Parameter = 2?
23 004772 001002              BNE     9$          ; Br if not
24 004774 004737 005554'      CALL    ERSLIN      ; Erase entire line
25         ;
26         ; Finished
27         ;
28 005000 000207      9$:    RETURN

```

```

1          .SBTTL  TCPERS -- Erase within a page
2          ;-----
3          ; Erase characters within the current page.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;   R5 = Pointer to start of parameter value
8          ;
9 005002   TCPERS:
10         ;
11         ; Accrue parameter value and determine how much to erase
12         ;
13 005002   004737 002116'   CALL    CSPARM      ;Accrue parameter
14 005006   005700          TST     R0              ;Parameter = 0?
15 005010   001003          BNE     1$             ;Br if not
16 005012   004737 006036'   CALL    ERSCTP      ;Erase from cursor to end of page
17 005016   000413          BR      9$
18 005020   020027 000001   1$:    CMP     R0,#1        ;Parameter = 1?
19 005024   001003          BNE     2$             ;Br if not
20 005026   004737 006120'   CALL    ERSPTC      ;Erase from beginning of page to cursor
21 005032   000405          BR      9$
22 005034   020027 000002   2$:    CMP     R0,#2        ;Parameter = 2?
23 005040   001002          BNE     9$             ;Br if not
24 005042   004737 006216'   CALL    ERSPAG      ;Erase entire page
25         ;
26         ; Finished
27         ;
28 005046   000207   9$:    RETURN

```

TCSSR -- Set scrolling region

```

1          .SBTTL  TCSSR  -- Set scrolling region
2          ;-----
3          ; Set the scrolling region.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;   R5 = Pointer to parameter values
8          ;
9 005050   TCSSR:
10         ;
11         ; Accrue scrolling region top value
12         ;
13 005050   004737   002116'   CALL    CSPARM      ;Accrue scrolling region top line
14 005054   005700           TST     RO              ;Was top line number specified?
15 005056   001002           BNE     1$              ;Br if yes
16 005060   012700   000001   MOV     #1,RO          ;Set 1 as top line
17 005064   010062   000000G  1$:    MOV     RO,DW$SRT(R2) ;Set top line of scrolling region
18         ;
19         ; Accrue scrolling region bottom value
20         ;
21 005070   004737   002116'   CALL    CSPARM      ;Accrue scrolling region bottom line
22 005074   005700           TST     RO              ;Was bottom line specified?
23 005076   001002           BNE     2$              ;Br if yes
24 005100   016200   000000G  MOV     DW$LPP(R2),RO ;Set last line as bottom of region
25 005104   010062   000000G  2$:    MOV     RO,DW$SRB(R2) ;Set bottom line of scrolling region
26         ;
27         ; Set the cursor to the home position
28         ;
29 005110   004737   005116'   CALL    SETHOM      ;Set cursor to home position
30         ;
31         ; Finished
32         ;
33 005114   000207           RETURN

```

SETHOM -- Set cursor to the home position

```

1          .SBTTL  SETHOM -- Set cursor to the home position
2          ;-----
3          ; This routine is called to set the current cursor position to
4          ; home.  The home line depends on whether absolute or relative
5          ; origin has been selected.
6          ;
7          ; Inputs:
8          ; R2 = Pointer to window control block.
9          ;
10 005116  SETHOM:
11          ;
12          ; Set the cursor line number.
13          ; See if absolute or relative origin has been selected.
14          ;
15 005116  032762  000000G 000000G          BIT    #AW$ORS,DW$AW(R2); Is origin relative to scroll region?
16 005124  001403          BEQ    1$          ;Br if not
17 005126  016200  000000G          MOV    DW$SRT(R2),R0 ;Say cursor is at top of scroll reg
18 005132  000402          BR     2$
19 005134  012700  000001          1$:   MOV    #1,R0          ;Say cursor is at line 1
20 005140  004737  006710'          2$:   CALL   SETLIN          ;Set the line number
21          ;
22          ; Set column to 1
23          ;
24 005144  012762  000001  000000G          MOV    #1,DW$COL(R2) ;Say cursor is at column 1
25          ;
26          ; Finished
27          ;
28 005152  000207          RETURN

```

SCRLUP -- Scroll screen up one line

```

1          .SBTTL  SCRLUP -- Scroll screen up one line
2          ;-----
3          ; Scroll the screen up one line.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block.
7          ;
8 005154 010346 SCRLUP: MOV     R3,-(SP)
9 005156 010446      MOV     R4,-(SP)
10 005160 010546      MOV     R5,-(SP)
11          ;
12          ; See if we are allowed to scroll any more lines
13          ;
14 005162 012700 000001      MOV     #1,R0      ;We are scrolling up one line
15 005166 004737 005422'    CALL    SCRCHK      ;See if scrolling allowed
16          ;
17          ; If the scrolling region is the entire screen, we can do the scrolling
18          ; by simply changing the number of the top line.
19          ;
20 005172 026227 000000G 000001      CMP     DW$SRT(R2),#1 ;Top line of scrolling region = 1?
21 005200 001013      BNE     2$          ;Br if not
22 005202 026262 000000G 000000G    CMP     DW$SRB(R2),DW$LPP(R2);Scrolling region go to bottom of page?
23 005210 001007      BNE     2$          ;Br if not
24          ;
25          ; Scrolling region covers the entire page
26          ;
27 005212 005362 000000G      DEC     DW$TLN(R2)    ;Change top line number
28 005216 003016      BGT     1$          ;Br if did not just scroll line 1
29 005220 016262 000000G 000000G    MOV     DW$LPP(R2),DW$TLN(R2);Cycle to last line on page
30 005226 000412      BR      1$
31          ;
32          ; The scrolling region does not cover the entire page.
33          ; Move characters from line to line to do the scrolling.
34          ;
35 005230 016203 000000G    2$:    MOV     DW$SRT(R2),R3 ;Get top line # of scrolling region
36 005234 010305    3$:    MOV     R3,R5      ;Set number of line to copy to
37 005236 005203      INC     R3
38 005240 010304      MOV     R3,R4      ;Set number of line to copy from
39 005242 004737 006354'    CALL    CPYLIN      ;Copy characters from one line to another
40 005246 020362 000000G    CMP     R3,DW$SRB(R2) ;Have we copied all that is needed?
41 005252 103770      BLO     3$          ;Loop if not
42          ;
43          ; Clear the line at the bottom of the scrolling region
44          ;
45 005254 016200 000000G    1$:    MOV     DW$SRB(R2),R0 ;Get # of line at bottom of scrolling regn
46 005260 004737 006240'    CALL    CLRLIN      ;Clear that line
47          ;
48          ; Finished
49          ;
50 005264 012605      MOV     (SP)+,R5
51 005266 012604      MOV     (SP)+,R4
52 005270 012603      MOV     (SP)+,R3
53 005272 000207      RETURN

```

SCRLDN -- Scroll screen down one line

```

1          .SBTTL  SCRLDN -- Scroll screen down one line
2          ;-----
3          ; Scroll the screen down one line.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block.
7          ;
8 005274 010346 SCRLDN: MOV     R3, -(SP)
9 005276 010446      MOV     R4, -(SP)
10 005300 010546      MOV     R5, -(SP)
11          ;
12          ; See if we are allowed to scroll any more lines
13          ;
14 005302 012700 177777      MOV     #-1, R0          ; Say we are scrolling down one line
15 005306 004737 005422'    CALL    SCRCHK          ; See if we are allowed to scroll more
16          ;
17          ; If the scrolling region is the entire screen, we can do the scrolling
18          ; by simply changing the number of the top line.
19          ;
20 005312 026227 000000G 000001      CMP     DW$SRT(R2), #1    ; Top line of scrolling region = 1?
21 005320 001016      BNE     2$          ; Br if not
22 005322 026262 000000G 000000G    CMP     DW$SRB(R2), DW$LPP(R2); Scrolling region go to bottom of page?
23 005330 001012      BNE     2$          ; Br if not
24          ;
25          ; Scrolling region covers the entire page
26          ;
27 005332 005262 000000G      INC     DW$TLN(R2)       ; Change top line number
28 005336 026262 000000G 000000G    CMP     DW$TLN(R2), DW$LPP(R2); Did we just scroll last line?
29 005344 101416      BLOS    1$          ; Br if not
30 005346 012762 000001 000000G    MOV     #1, DW$TLN(R2)   ; Cycle back to line 1
31 005354 000412      BR      1$
32          ;
33          ; The scrolling region does not cover the entire page.
34          ; Move characters from line to line to do the scrolling.
35          ;
36 005356 016203 000000G    2$:    MOV     DW$SRB(R2), R3 ; Get bottom line # of scrolling region
37 005362 010305    3$:    MOV     R3, R5      ; Set number of line to copy to
38 005364 005303      DEC     R3
39 005366 010304      MOV     R3, R4          ; Set number of line to copy from
40 005370 004737 006354'    CALL    CPYLIN          ; Copy characters from one line to another
41 005374 020362 000000G    CMP     R3, DW$SRT(R2)  ; Have we copied all that is needed?
42 005400 101370      BHI     3$          ; Loop if not
43          ;
44          ; Clear the line at the top of the scrolling region
45          ;
46 005402 016200 000000G    1$:    MOV     DW$SRT(R2), R0 ; Get # of line at top of scrolling regn
47 005406 004737 006240'    CALL    CLRLIN          ; Clear that line
48          ;
49          ; Finished
50          ;
51 005412 012605      MOV     (SP)+, R5
52 005414 012604      MOV     (SP)+, R4
53 005416 012603      MOV     (SP)+, R3
54 005420 000207      RETURN

```

SCRCHK -- Check for scrolling limits

```

1          .SBTTL  SCRCHK -- Check for scrolling limits
2          ;-----
3          ; Determine if we are allowed to scroll the screen any further while the
4          ; job is detached from the terminal.  If we have reached the limit of
5          ; our scrolling, the job is suspended until the terminal is reconnected.
6          ;
7          ; Inputs:
8          ;   RO = +1 if scrolling up, -1 if scrolling down.
9          ;   R2 = Pointer to window control block.
10         ;
11 005422 010146  SCRCHK: MOV      R1,-(SP)
12         ;
13         ; See if job is currently attached to terminal
14         ;
15 005424 032762 000000G 000000G  BIT      #AW$DDC,DW$AW(R2);Are we suppressing char display?
16 005432 001426          BEQ      9$          ;Br if not
17 005434 116201 000000G          MOVVB   DW$JOB(R2),R1  ;Get number of job
18 005440 032761 000000G 000000G  BIT      #V$NOTT,LSW(R1) ;Is job currently connected to terminal?
19 005446 001420          BEQ      9$          ;Br if yes
20         ;
21         ; Job is not currently attached to the terminal.
22         ; See if we are allowed to scroll an unlimited number of lines.
23         ;
24 005450 116201 000000G          MOVVB   DW$MSL(R2),R1  ;Are we allowed to scroll unlimited # lines?
25 005454 002415          BLT      9$          ;Br if yes
26 005456 001412          BEQ      2$          ;Br if no scrolling is allowed
27         ;
28         ; See if this scroll would exceed limit.
29         ;
30 005460 116201 000000G          MOVVB   DW$NSL(R2),R1  ;Get # lines scrolled so far
31 005464 060001          ADD      RO,R1          ;Add number scrolled this time
32 005466 110162 000000G          MOVVB   R1,DW$NSL(R2) ;Save new total
33 005472 002001          BGE      1$          ;Br if total is positive
34 005474 005401          NEG      R1          ;Get positive line count
35 005476 120162 000000G 1$:  CMPB   R1,DW$MSL(R2) ;Would this exceed max allowed scroll?
36 005502 101402          BLOS   9$          ;Br if not
37         ;
38         ; Suspend job until job reattaches to the terminal
39         ;
40 005504 004737 005514' 2$:  CALL   WINSPN          ;Suspend job
41         ;
42         ; Finished
43         ;
44 005510 012601 9$:  MOV      (SP)+,R1
45 005512 000207          RETURN

```


WINSPN -- Suspend job until it reattaches to terminal

```

1          .SBTTL  WINSPN -- Suspend job until it reattaches to terminal
2          ;-----
3          ; Suspend the execution of a job until it reattaches to a terminal.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;
8 005514  010146  WINSPN: MOV      R1,-(SP)
9          ;
10         ; Never suspend if we are running at fork level (doing character echoing)
11         ;
12 005516  105737  000000G      TSTB   FRKPRI      ;Are we running at fork level?
13 005522  001012          BNE    9$          ;Br if yes
14         ;
15         ; See if job has reattached to the terminal
16         ;
17 005524  116201  000000G  1$:   MOVB   DW$JOB(R2),R1  ;Get job index number
18 005530  032761  000000G  000000G  BIT    #$VNOTT,LSW(R1) ;Is job attached to terminal?
19 005536  001404          BEQ    9$          ;Br if yes
20         ;
21         ; Suspend the job
22         ;
23 005540          OCALL  PCSPND      ;Suspend execution of the job
24 005546  000766          BR     1$          ;Wait till job reconnects to terminal
25         ;
26         ; Finished
27         ;
28 005550  012601  9$:   MOV    (SP)+,R1
29 005552  000207          RETURN

```

ERSLIN -- Erase the current line

```

1          .SBTTL  ERSLIN -- Erase the current line
2          ;-----
3          ; Erase the current line.  Do not change the line attributes and leave
4          ; the cursor where it is.
5          ;
6          ; Inputs:
7          ;   R2 = Pointer to window control block
8          ;
9 005554 010346  ERSLIN: MOV      R3, -(SP)
10         ;
11         ; Store blanks with all attributes cleared into all line columns
12         ;
13 005556 016203 000000G      MOV      DW$LPT(R2), R3    ;Get pointer to 1st char in line
14 005562 016200 000000G      MOV      DW$CPL(R2), R0    ;Get number of columns per line
15 005566          1$:  BUFMAP          ;;; Map to display buffer
16 005610 112723 000040      MOV      #40, (R3)+      ;;; Store blank as the character
17 005614 105023          CLR      (R3)+          ;;; Clear all attribute flags
18 005616          UNMAP          ;Restore mapping
19 005632 077023          SOB      R0, 1$      ;Clear entire line
20         ;
21         ; Finished
22         ;
23 005634 012603          MOV      (SP)+, R3
24 005636 000207          RETURN

```

ERSCTL -- Erase from cursor to the end of the line

```

1          .SBTTL  ERSCTL -- Erase from cursor to the end of the line
2          ;-----
3          ; Erase all characters from the cursor position to the end of the line.
4          ; The cursor does not move. Character attributes are reset for all
5          ; erased characters.
6          ;
7          ; Inputs:
8          ; R2 = Pointer to window control block
9          ;
10         ERSCTL: MOV      R3, -(SP)
11         MOV      R4, -(SP)
12         ;
13         ; Get pointer to current cursor position and to end of line
14         ;
15         MOV      DW$COL(R2), R3 ;Get current column number
16         DEC      R3             ;Convert to column offset
17         ASL      R3             ;Two bytes are used by each column
18         ADD      DW$LPT(R2), R3 ;Point to byte for current column
19         MOV      DW$CPL(R2), R4 ;Get number of columns per line
20         DEC      R4             ;Convert to offset
21         ASL      R4             ;Two bytes per column
22         ADD      DW$LPT(R2), R4 ;Get pointer to byte for last column
23         ;
24         ; Clear the line to blanks
25         ;
26         1$:  BUFMAP                ;;; Map to display buffer
27         MOV      #40, (R3)+        ;;; Store blank as character
28         CLRB     (R3)+            ;;; Clear attribute flags
29         UNMAP                ;Restore mapping
30         CMP      R3, R4           ;Finished line?
31         BLOS     1$              ;Loop if not
32         ;
33         ; Finished
34         ;
35         MOV      (SP)+, R4
36         MOV      (SP)+, R3
37         RETURN

```

```

1          .SBTTL  ERSLTC -- Erase from beginning of line to cursor
2          ;-----
3          ; Erase from the beginning of the line up to and including the character
4          ; pointed to by the cursor.  The cursor does not move.
5          ;
6          ; Inputs:
7          ; R2 = Pointer to window control block.
8          ;
9 005752 010346  ERSLTC: MOV      R3, -(SP)
10         ;
11         ; Set up pointer to start of line and to cursor position
12         ;
13 005754 016203 000000G      MOV      DW$LPT(R2), R3 ;Get pointer to 1st byte for the line
14 005760 016200 000000G      MOV      DW$COL(R2), R0 ;Get current cursor column
15         ;
16         ; Blank the characters
17         ;
18 005764          1$:      BUFMAP          ;;; Map to display buffer
19 006006 112723 000040      MOV      #40, (R3)+      ;;; Store blank as the character
20 006012 105023          CLRB      (R3)+      ;;; Clear attribute flags
21 006014          UNMAP          ;Restore mapping
22 006030 077023          SOB      R0, 1$      ;Loop if more characters to clear
23         ;
24         ; Finished
25         ;
26 006032 012603          MOV      (SP)+, R3
27 006034 000207          RETURN

```

ERSCTP -- Erase from cursor to the end of the page

```

1          .SBTTL  ERSCTP -- Erase from cursor to the end of the page
2          ;-----
3          ; Erase from the current cursor position to the end of the page.
4          ; The cursor does not move. Line attributes for completely erased
5          ; lines are reset.
6          ;
7          ; Inputs:
8          ; R2 = Pointer to window control block
9          ;
10         006036 010546 ERSCTP: MOV R5,-(SP)
11         006040 016246 000000G MOV DW$LIN(R2),-(SP); Save the current line number
12         ;
13         ; Initialize the line number to the current line
14         ;
15         006044 016205 000000G MOV DW$LIN(R2),R5 ; Get current line number
16         ;
17         ; Determine if we are clearing all of the current line
18         ;
19         006050 026227 000000G 000001 CMP DW$COL(R2),#1 ; Is cursor at column 1 of current line?
20         006056 001403 BEQ 1$ ; Br if yes -- Clear all of current line
21         ;
22         ; Clear from cursor to end of current line
23         ;
24         006060 004737 005640' CALL ERSCTL ; Erase to end of current line
25         006064 005205 INC R5 ; Advance line number
26         ;
27         ; Now clear all lines below current line
28         ;
29         006066 020562 000000G 1$: CMP R5,DW$LPP(R2) ; Have we erased all lines?
30         006072 101005 BHI 9$ ; Br if yes
31         006074 010500 MOV R5,R0 ; Get number of line to be cleared
32         006076 004737 006240' CALL CLRLIN ; Clear the line
33         006102 005205 INC R5 ; Increment line number
34         006104 000770 BR 1$ ; See if there are more to clear
35         ;
36         ; Finished
37         ;
38         006106 012600 9$: MOV (SP)+,R0 ; Recover original line number
39         006110 004737 006710' CALL SETLIN ; Restore line number
40         006114 012605 MOV (SP)+,R5
41         006116 000207 RETURN

```

ERSPTC -- Erase from beginning of page to cursor

```

1          .SBTTL  ERSPTC -- Erase from beginning of page to cursor
2          ;-----
3          ; Erase all of the characters from the beginning of the page up to and
4          ; including the character under the cursor.  Line attributes are cleared
5          ; for lines that are completely erased.
6          ;
7          ; Inputs:
8          ;   R2 = Pointer to window control block
9          ;
10         ERSPTC: MOV     R4, -(SP)
11         MOV     R5, -(SP)
12         ;
13         ; Initialize line numbers
14         ;
15         MOV     DW$LIN(R2), R5 ;Get number of current line
16         MOV     #1, R4        ;Get number of first line
17         ;
18         ; Erase all lines up to, but not including, current line
19         ;
20         1$:  CMP     R4, R5        ;Have we reached current line?
21         BHS    2$                ;Br if yes
22         MOV     R4, R0          ;Get number of line to erase
23         CALL   CLRLIN          ;Erase the line
24         INC     R4              ;Advance the line number
25         BR     1$              ;See if more lines to erase
26         ;
27         ; Erase the line the cursor is on
28         ;
29         2$:  MOV     R4, R0          ;Get number of current line
30         CMP     DW$COL(R2), DW$CPL(R2); Is cursor at the end of the line?
31         BNE    3$                ;Br if not
32         CALL   CLRLIN          ;Clear it
33         BR     9$
34         3$:  CALL   SETLIN        ;Select current line
35         CALL   ERSLTC          ;Erase up to cursor
36         ;
37         ; Finished
38         ;
39         9$:  MOV     R5, R0          ;Get back current line number
40         CALL   SETLIN        ;Select as current line
41         MOV     (SP)+, R5
42         MOV     (SP)+, R4
43         RETURN
44
45

```

```
1          .SBTTL  ERSPAG -- Clear entire page to spaces
2          ;-----
3          ; Clear the entire page to spaces
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block
7          ;
8 006216 010546  ERSPAG: MOV      R5, -(SP)
9          ;
10         ; Begin loop to clear all lines
11         ;
12 006220 016205 000000G      MOV      DW$LPP(R2), R5 ;Get last line number
13 006224 010500      1$:  MOV      R5, R0 ;Get number of line to clear
14 006226 004737 006240'      CALL   CLRLIN ;Clear line to spaces
15 006232 077504      SOB      R5, 1$ ;Loop if more lines to clear
16         ;
17         ; Finished
18         ;
19 006234 012605      MOV      (SP)+, R5
20 006236 000207      RETURN
```

CLRLIN -- Clear a line to spaces

```

1          .SBTTL  CLRLIN -- Clear a line to spaces
2          ;-----
3          ; Clear a line to spaces and set the line attributes to single high,
4          ; single wide.
5          ;
6          ; Inputs:
7          ; R0 = Number of line to be cleared
8          ; R2 = Pointer to window control block
9          ;
10         CLRLIN: MOV     R3, -(SP)
11         MOV     R4, -(SP)
12         MOV     DW$LIN(R2), -(SP); Save current line number
13         ;
14         ; Select the line to be cleared
15         ;
16         CALL    SETLIN          ; Select line to be cleared
17         ;
18         ; Blank the line
19         ;
20         MOV     DW$LPT(R2), R3  ; Get pointer to start of line
21         MOV     DW$CPL(R2), R4  ; Get # columns in the line
22         1$: BUFMAP                ; Map to screen buffer
23         MOVB    #40, (R3)+       ; Store blank character
24         CLRB    (R3)+           ; Clear character attributes
25         UNMAP                ; Restore mapping
26         SOB    R4, 1$          ; Loop if need to store more blanks
27         ;
28         ; Reset line attributes
29         ;
30         CLR     R0              ; Set attribute to 0
31         CALL    SETLAB          ; Reset all line attributes
32         ;
33         ; Finished
34         ;
35         MOV     (SP)+, R0        ; Get back original line #
36         CALL    SETLIN          ; Reselect original line
37         MOV     (SP)+, R4
38         MOV     (SP)+, R3
39         RETURN

```


CPYLIN -- Copy characters from one line to another

```

1                                     .SBTTL  CPYLIN -- Copy characters from one line to another
2                                     -----
3                                     ; Copy the characters and attributes from one line to another.
4                                     ; The line attribute is also copied.
5                                     ;
6                                     ; Inputs:
7                                     ; R2 = Pointer to window control block
8                                     ; R4 = Number of line to copy from
9                                     ; R5 = Number of line to copy to
10                                    ;
11 006354 010346  CPYLIN: MOV      R3, -(SP)
12 006356 010446      MOV      R4, -(SP)
13 006360 010546      MOV      R5, -(SP)
14 006362 016246 000000G      MOV      DW$LIN(R2), -(SP); Save current line number
15                                    ;
16                                    ; Copy the line attribute byte
17                                    ;
18 006366 010400      MOV      R4, R0          ; Get source line #
19 006370 004737 006710'      CALL     SETLIN        ; Set up info about that line
20 006374 004737 006620'      CALL     GETLAB        ; Get line attribute information (to R0)
21 006400 010003      MOV      R0, R3          ; Save line attribute
22 006402 010500      MOV      R5, R0          ; Get dest line #
23 006404 004737 006710'      CALL     SETLIN        ; Set up info about that line
24 006410 010300      MOV      R3, R0          ; Get line attribute byte
25 006412 004737 006530'      CALL     SETLAB        ; Set line attribute byte
26                                    ;
27                                    ; Get pointers to characters in from and to lines
28                                    ;
29 006416 010400      MOV      R4, R0          ; Get # of line we are copying from
30 006420 004737 006710'      CALL     SETLIN        ; Set up information about the line
31 006424 016204 000000G      MOV      DW$LPT(R2), R4 ; Get pointer to 1st char in from line
32 006430 010500      MOV      R5, R0          ; Get # of line we are copying to
33 006432 004737 006710'      CALL     SETLIN        ; Set up information about the line
34 006436 016205 000000G      MOV      DW$LPT(R2), R5 ; Get pointer to 1st char in to line
35                                    ;
36                                    ; Begin loop to copy the characters
37                                    ;
38 006442 016200 000000G      MOV      DW$CPL(R2), R0 ; Get # columns to copy
39 006446      1$:  BUFMAP      ;;; Map to screen buffer
40 006470 112425      MOV      (R4)+, (R5)+    ;;; Copy character
41 006472 112425      MOV      (R4)+, (R5)+    ;;; Copy attribute
42 006474      UNMAP      ; Restore mapping
43 006510 077022      SOB      R0, 1$        ; Loop to copy all chars in the line
44                                    ;
45                                    ; Finished
46                                    ;
47 006512 012600      MOV      (SP)+, R0        ; Get back original line #
48 006514 004737 006710'      CALL     SETLIN        ; Reselect that line
49 006520 012605      MOV      (SP)+, R5
50 006522 012604      MOV      (SP)+, R4
51 006524 012603      MOV      (SP)+, R3
52 006526 000207      RETURN

```

SETLAB -- Set attributes for current line

```

1          .SBTTL  SETLAB -- Set attributes for current line
2          ;-----
3          ; SETLAB is called to set the line attribute byte for the current line.
4          ;
5          ; Inputs:
6          ;   R0 = Line attribute byte value
7          ;   R2 = Pointer to window control block
8          ;
9 006530 010346 SETLAB: MOV      R3, -(SP)
10         ;
11         ; Compute virtual address of byte with attribute for current line
12         ;
13 006532 016203 000000G      MOV      DW$LIN(R2), R3      ;Get current line number
14 006536 166203 000000G      SUB      DW$TLN(R2), R3      ;Get offset from line at top of buffer
15 006542 002002              BGE      1$              ;Br if line greater than 1st line
16 006544 066203 000000G      ADD      DW$LPP(R2), R3      ;Get offset to line we want
17 006550 062703 000000G 1$:  ADD      #VPAR6, R3      ;Add virtual base address
18         ;
19         ; Store the new attribute byte
20         ;
21 006554              BUFMAP              ;;;Map to screen buffer
22 006576 110013      MOV      RO, (R3)      ;;;Store new attribute for line
23 006600              UNMAP              ;Restore mapping
24         ;
25         ; Finished
26         ;
27 006614 012603      MOV      (SP)+, R3
28 006616 000207      RETURN

```

GETLAB -- Get attributes for current line

```

1          .SBTTL  GETLAB -- Get attributes for current line
2          ;-----
3          ; GETLAB is called to get the line attribute byte for the current line.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;
8          ; Outputs:
9          ;   R0 = Line attribute byte for current line.
10         ;
11 006620 010346 GETLAB: MOV      R3, -(SP)
12         ;
13         ; Compute virtual address of byte with attribute for current line
14         ;
15 006622 016203 0000000 MOV     DW$LIN(R2), R3 ;Get current line number
16 006626 166203 0000000 SUB     DW$TLN(R2), R3 ;Get offset from line at top of buffer
17 006632 002002          BGE     1$ ;Br if line greater than 1st line
18 006634 066203 0000000 ADD     DW$LPP(R2), R3 ;Get offset to line we want
19 006640 062703 0000000 1$: ADD     #VPAR6, R3 ;Add virtual base address
20         ;
21         ; Get the attribute byte
22         ;
23 006644          BUFMAP          ;;; Map to screen buffer
24 006666 111300          MOV     (R3), R0 ;;; Get the line attribute byte
25 006670          UNMAP          ;Restore mapping
26         ;
27         ; Finished
28         ;
29 006704 012603          MOV     (SP)+, R3
30 006706 000207          RETURN

```

SETLIN -- Select a line as current line

```

1          .SBTTL  SETLIN -- Select a line as current line
2          ;-----
3          ; Select a specified line as the current line and set up pointer information
4          ; for it.
5          ;
6          ; Inputs:
7          ;   R0 = Line number being selected (1-24)
8          ;   R2 = Pointer to window control block
9          ;
10         ; Outputs:
11         ;   DW$LIN(R2) = Set to current line number
12         ;   DW$LPT(R2) = Virtual address of 1st character on current line
13         ;
14 006710  010146  SETLIN: MOV      R1, -(SP)
15         ;
16         ; Set the current line number in WCB
17         ;
18 006712  010062  0000000  MOV      R0, DW$LIN(R2) ; Set current line number
19         ;
20         ; Compute virtual address of 1st character in this line
21         ;
22 006716  166200  0000000  SUB      DW$TLN(R2), R0 ; Get # lines from top to line wanted
23 006722  002002                BGE     1$ ; Br if top line is LE one wanted
24 006724  066200  0000000  ADD     DW$LPP(R2), R0 ; Add total number of lines per page
25 006730  010001                1$: MOV    R0, R1 ; Get # lines offset to line we want
26 006732  070162  0000000  MUL    DW$CPL(R2), R1 ; Multiply by number of columns per line
27 006736  006301                ASL    R1 ; Times two because of attribute bytes
28 006740  066201  0000000  ADD    DW$LPP(R2), R1 ; Skip over bytes with line attributes
29 006744  062701  0000000  ADD    #VPAR6, R1 ; Add base virtual address
30 006750  010162  0000000  MOV    R1, DW$LPT(R2) ; This is virtual addr of start of line
31         ;
32         ; Finished
33         ;
34 006754  012601                MOV    (SP)+, R1
35 006756  000207                RETURN

```

WINSF -- Switch from a job with a display window

```

1          .SBTTL  WINSF  -- Switch from a job with a display window
2          ;-----
3          ; WINSF is called from TSTTY when we are switching from a job which
4          ; has an active display window.
5          ;
6          ; Inputs:
7          ; R2 = Pointer to display window control block for job being switched from
8          ;
9 006760   WINSF:
10         ;
11         ; Set flag which prevents characters from being passed through to the
12         ; screen.
13         ;
14 006760  052762  000000G 000000G          BIS      #AW$DDC,DW$AW(R2);Don't pass through any more chars
15         ;
16         ; Reset number of lines scrolled since we disconnected from window
17         ;
18 006766  105062  000000G          CLRB    DW$NSL(R2)      ;No lines scrolled yet
19         ;
20         ; Finished
21         ;
22 006772  000207          RETURN

```

WINST -- Switch to job with a window

```

1          .SBTTL  WINST  -- Switch to job with a window
2          ;-----
3          ; WINST is called from TSTTY when we are switching to a job that has
4          ; a display window active.
5          ;
6          ; Inputs:
7          ; R2 = Pointer to current window control block for job being switched to
8          ;
9 006774 010146 WINST:  MOV     R1,-(SP)
10         ;
11         ; Get job index number
12         ;
13 006776 116201 000000G      MOVB   DW$JOB(R2),R1  ;Get job index number
14         ;
15         ; Set flag which will cause the window to be redisplayed when the
16         ; job resumes execution.
17         ;
18 007002 052761 000000G 000000G  BIS    #$WDISP,LSW6(R1);Set flag saying to redisplay window
19         ;
20         ; If job is in terminal-output-wait state, restart its execution
21         ;
22 007010 026127 000000G 000000G  CMP    LSTATE(R1),#S$OTWT;TT-output-wait state?
23 007016 001002          BNE    9$          ;Br if not
24 007020 004737 000000G      CALL   FORCEX          ;Restart execution of job
25         ;
26         ; Finished
27         ;
28 007024 012601          9$:   MOV    (SP)+,R1
29 007026 000207          RETURN

```

WINDSP -- Redisplay current window for job

```

1          .SBTTL  WINDSP -- Redisplay current window for job
2          ;-----
3          ; WINDSP is called from the scheduler when it is about ready to
4          ; reenter a job whose window needs to be redisplayed.
5          ;
6          ; Inputs:
7          ; R1 = Current job index number
8          ;
9 007030 010246 WINDSP: MOV      R2,-(SP)
10         ;
11         ; Reset flag that says window redisplay needed
12         ; and set flag that says a window refresh is being done.
13         ;
14 007032 042761 000000G 000000G      BIC      #$WDISP,LSW6(R1);Window redisplay no longer needed
15 007040 052761 000000G 000000G      BIS      #$RFRSH,LSW4(R1);Window refresh currently being done
16         ;
17         ; Redisplay current window for job
18         ;
19 007046 016102 000000G          MOV      LWINDO(R1),R2 ;Get addr of current window control block
20 007052 001402          BEQ      9$ ;Br if job is not doing windowing
21 007054 004737 007072'        CALL     REFRSH ;Redisplay window
22         ;
23         ; Finished
24         ;
25 007060 042761 000000G 000000G 9$: BIC      #$RFRSH,LSW4(R1);Say window refresh is finished
26 007066 012602          MOV      (SP)+,R2
27 007070 000207          RETURN

```

REFRSH -- Refresh screen from window contents

```

1          .SBTTL  REFRSH -- Refresh screen from window contents
2          ;-----
3          ; This routine is called to clear the screen and refresh its contents
4          ; from the screen buffer.
5          ;
6          ; Inputs:
7          ; R2 = Pointer to window control block
8          ;
9 007072 010146 REFRSH: MOV      R1,-(SP)
10 007074 010546      MOV      R5,-(SP)
11          ;
12          ; Stop characters from being passed from program to screen while we are
13          ; doing the refresh
14          ;
15 007076 052762 000000G 000000G      BIS      #AW$DDC,DW$AW(R2); Suppress character pass-through
16          ;
17          ; Clear contents of line's output buffer and reset control-D flag
18          ;
19 007104 116201 000000G      MOVVB   DW$JOB(R2),R1 ;Get job index number
20 007110      DISABL      ;;;Disable interrupts
21 007116 016161 000000G 000000G      MOV      LOTSIZ(R1),LOTSPC(R1);;Clear contents of line's output buffer
22 007124 016161 000000G 000000G      MOV      LOTNXT(R1),LOTPNT(R1);;
23 007132      ENABL      ;Enable interrupts
24 007140 042761 000000G 000000G      BIC      #$CTRLD,LSW3(R1);Reset control-D flag
25          ;
26          ; See if we need to turn VT52 emulation mode on or off
27          ;
28 007146 004737 011116'      CALL     GENTEM      ;Check VT52 emulation mode status
29          ;
30          ; Clear the screen
31          ;
32 007152 004737 007620'      CALL     GENCLR      ;Clear the screen
33          ;
34          ; Set overall window attributes
35          ;
36 007156 004737 011424'      CALL     GENWAF      ;Set overall window attributes
37          ;
38          ; Initialize character attribute flags
39          ;
40 007162 005005      CLR      R5      ;No character attribute flags
41          ;
42          ; Begin loop to send each line to the screen
43          ;
44 007164 016246 000000G      MOV      DW$LIN(R2),-(SP); Save current line number
45 007170 012701 000001      MOV      #1,R1      ;Get 1st line number
46 007174 010100 1#:      MOV      R1,R0      ;Get line number
47 007176 004737 006710'      CALL     SETLIN      ;Select as current line
48 007202 004737 007310'      CALL     SNDLIN      ;Send line to screen
49 007206 005201      INC      R1      ;Increment line number
50 007210 020162 000000G      CMP      R1,DW$LPP(R2) ;Have we done all lines?
51 007214 101767      BLOS     1#      ;Loop if not
52          ;
53          ; Reset line number
54          ;
55 007216 012600      MOV      (SP)+,R0      ;Get back line where we should be
56 007220 004737 006710'      CALL     SETLIN      ;Set this as the current line
57          ;

```


REFRSH -- Refresh screen from window contents

```

58          ; Set up scrolling region status
59          ;
60 007224 004737 010754'          CALL    GENSSS          ;Set up split screen scrolling
61          ;
62          ; Place cursor where it should be
63          ;
64 007230 004737 012270'          CALL    GENCSR          ;Gen cursor positioning commands
65 007234 016200 000000G          MOV     DW$COL(R2),R0    ;Get current column number
66 007240 005300                      DEC     R0                ;Make 1st column be 0
67 007242 116201 000000G          MOVB   DW$JOB(R2),R1    ;Get our job number
68 007246 110061 000000G          MOVB   R0,LCOL(R1)     ;Tell TSTTY where cursor is
69          ;
70          ; Make sure character attribute is set correctly
71          ;
72 007252 120562 000000G          CMPB   R5,DW$CCA(R2)    ;Is character attribute set correctly?
73 007256 001404                      BEQ    2$                ;Br if yes
74 007260 116200 000000G          MOVB   DW$CCA(R2),R0    ;Get needed character attribute
75 007264 004737 010130'          CALL    GENCAF          ;Set character attribute
76          ;
77          ; Set up correct character set mapping
78          ;
79 007270 004737 010542'          2$:   CALL    GENMAP          ;Set up correct character set mapping
80          ;
81          ; Now allow characters to be passed to the screen
82          ;
83 007274 042762 000000G 000000G          BIC    #AW$DDC,DW$AW(R2);Enable character pass-through
84          ;
85          ; Finished
86          ;
87 007302 012605          9$:   MOV     (SP)+,R5
88 007304 012601          MOV     (SP)+,R1
89 007306 000207          RETURN

```

SNDLIN -- Send line of characters to terminal

```

1          .SBTTL  SNDLIN -- Send line of characters to terminal
2          ;-----
3          ; Send a line of characters from the screen buffer to the terminal.
4          ; The currently selected line is the one sent.
5          ;
6          ; Inputs:
7          ; R2 = Pointer to window control block.
8          ; R5 = Current character attribute flags
9          ;
10         ; Outputs:
11         ; R5 = Character attribute flags for last character on line
12         ;
13 007310 010146  SNDLIN: MOV     R1,-(SP)
14 007312 010346          MOV     R3,-(SP)
15 007314 010446          MOV     R4,-(SP)
16 007316 016246 000000G  MOV     DW$COL(R2),-(SP); Save current column number
17         ;
18         ; Set line attribute flag and determine number of columns to display
19         ;
20 007322 004737 006620'  CALL    GETLAB          ;Get line attribute flags for cur line to R0
21 007326 012701 000120          MOV     #80.,R1        ;Assume 80 columns to display
22 007332 032762 000000G 000000G  BIT     #AW$132,DW$AW(R2); Are we in 132 column mode?
23 007340 001402          BEQ     1$              ;Br if not in 132 col mode
24 007342 012701 000204          MOV     #132.,R1       ;Say we need to display 132 columns
25 007346 132700 000000C  1$: BITB  #<AL$DHT!AL$DHB!AL$DWD>,R0 ;Double wide line?
26 007352 001401          BEQ     3$              ;Br if not double wide
27 007354 006201          ASR     R1              ;Only need to display half as many chars
28 007356 004737 012006'  3$: CALL    GENLAF        ;Generate line attribute flag info
29         ;
30         ; Store number of columns to display on top of stack
31         ;
32 007362 020162 000000G          CMP     R1,DW$CPL(R2)   ;Are we storing as many cols as needed?
33 007366 101402          BLOS   5$              ;Br if yes
34 007370 016201 000000G          MOV     DW$CPL(R2),R1  ;Get # columns we are storing
35 007374 010146          5$: MOV     R1,-(SP)      ;Store # columns to display on top of stack
36         ;
37         ; Begin loop to output the characters on the line
38         ;
39 007376 012701 000001          MOV     #1,R1          ;Init column number
40 007402 010162 000000G          MOV     R1,DW$COL(R2)  ;Init col where next character goes
41 007406 016203 000000G          MOV     DW$LPT(R2),R3  ;Get pointer to 1st char on this line
42         ;
43         ; Scan over spaces that have no attributes set
44         ;
45 007412          4$: BUFMAP          ;;; Map to screen buffer
46 007434 112304          MOVVB  (R3)+,R4        ;;; Get character
47 007436 112300          MOVVB  (R3)+,R0        ;;; Get attribute flags for character
48 007440          UNMAP          ;Restore mapping
49 007454 120427 000040          CMPB   R4,#40         ;Is character a space?
50 007460 001005          BNE    2$              ;Br if not
51 007462 120005          CMPB   R0,R5          ;Are attributes changing?
52 007464 001003          BNE    2$              ;Br if yes
53 007466 032700 000000C          BIT     #<AC$BLD!AC$BLK!AC$REV!AC$ULN>,R0 ;Bold, blink, reverse, under
54 007472 001420          BEQ    B$              ;Br if not
55         ;
56         ; We found a character that we need to send.
57         ; (R4 = character, R0 = attribute flags)

```

SNDLIN -- Send line of characters to terminal

```

58          ; Position to correct column for the character.
59          ;
60 007474 020162 000000G 2$:      CMP      R1,DW$COL(R2) ;Are we at correct column now?
61 007500 001404          BEQ      6$          ;Br if yes
62 007502 010046          MOV      R0,-(SP)   ;Save character attribute flags
63 007504 004737 012122' CALL      GENSPC   ;Generate spaces to move to correct col
64 007510 012600          MOV      (SP)+,R0   ;Restore character attribute flags
65          ;
66          ; See if we need to change attributes for this character
67          ;
68 007512 120005 6$:      CMPB     R0,R5      ;Do we need to change attributes?
69 007514 001402          BEQ      7$          ;Br if not
70 007516 004737 010130' CALL      GENCAF   ;Set character attribute flags
71          ;
72          ; Send the character to the terminal
73          ;
74 007522 010400 7$:      MOV      R4,R0      ;Get character to send
75 007524 004737 012500' CALL      GENCHR   ;Send the character to the terminal
76 007530 005262 000000G INC      DW$COL(R2) ;Increment current column number
77          ;
78          ; Advance to next character
79          ;
80 007534 005201 8$:      INC      R1          ;Increment column number
81 007536 020116          CMP      R1,(SP)   ;Done all columns?
82 007540 101724          BLOS     4$          ;Loop if not
83          ;
84          ; Output carriage-return line-feed at the end of the line
85          ;
86 007542 026262 000000G 000000G CMP      DW$LIN(R2),DW$LPP(R2);Is this the last line on the page?
87 007550 001414          BEQ      20$         ;Br if yes -- Don't send CR-LF
88 007552 026227 000000G 000001 CMP      DW$COL(R2),#1 ;Are we still at col 1 (entire line blank)?
89 007560 001404          BEQ      10$         ;Br if yes
90 007562          SEND     #CR      ;Send carriage return
91 007572          10$:     SEND     #LF      ;Send line feed
92          ;
93          ; Finished
94          ;
95 007602 005726 20$:     TST      (SP)+      ;Pop number of columns per line
96 007604 012662 000000G MOV      (SP)+,DW$COL(R2);Restore cursor column number
97 007610 012604          MOV      (SP)+,R4
98 007612 012603          MOV      (SP)+,R3
99 007614 012601          MOV      (SP)+,R1
100 007616 000207          RETURN

```

GENCLR -- Gen control sequence to clear screen

```

1          .SBTTL  GENCLR -- Gen control sequence to clear screen
2          ;-----
3          ; Generate the appropriate terminal control sequence to clear the screen.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block
7          ;
8 007620   GENCLR:
9          ;
10         ; See if we need to generate a VT100 or VT52 sequence
11         ;
12 007620   032762   000000C 000000G   BIT    #<AW$52!AW$S52>,DW$AW(R2);VT52?
13 007626   001115   BNE    1$          ;Br if yes
14         ;
15         ; Generate VT100 sequence
16         ; Reset scrolling region
17         ;
18 007630   004737   012456'   CALL    GENCSI          ;Generate CSI header
19 007634   SEND    #'r          ;Reset scrolling region
20         ;
21         ; Set absolute origin
22         ;
23 007644   004737   012456'   CALL    GENCSI          ;Generate CSI header
24 007650   SEND    #'?          ;Send ?
25 007660   SEND    #'6          ;Send 6
26 007670   SEND    #'1          ;Send 1
27         ;
28         ; Home cursor and clear to end of screen
29         ;
30 007700   004737   012456'   CALL    GENCSI          ;Generate CSI header
31 007704   SEND    #'H          ;Send H -- Home cursor
32 007714   004737   012456'   CALL    GENCSI          ;Generate CSI header
33 007720   SEND    #'J          ;Send J -- Clear to end of screen
34         ;
35         ; Reset character attributes
36         ;
37 007730   004737   012456'   CALL    GENCSI          ;Send CSI O m to reset char attributes
38 007734   SEND    #'O
39 007744   SEND    #'m
40         ;
41         ; Map GL to G0 and designate G0 as ascii
42         ;
43 007754   SEND    #17          ;Send shift-in -- Lock shift G0 to GL
44 007764   004737   012444'   CALL    GENESC          ;Send escape
45 007770   SEND    #'<          ;Select G0
46 010000   SEND    #'B          ;Map G0 to ascii
47         ;
48         ; Map GR to G2 and designate G2 as ascii
49         ;
50 010010   032762   000000G 000000G   BIT    #AW$200,DW$AW(R2);Is this a VT200 terminal?
51 010016   001443   BEQ    9$          ;Br if not -- Don't have to mess with GR
52 010020   004737   012444'   CALL    GENESC          ;Send ESC
53 010024   SEND    #'>          ;Lock shift GR to G2
54 010034   004737   012444'   CALL    GENESC          ;Send escape
55 010040   SEND    #'*          ;Select G2
56 010050   SEND    #'B          ;Designate ascii
57 010060   000422   BR     9$

```

GENCLR -- Gen control sequence to clear screen

```
58 ;  
59 ; Generate VT52 sequence  
60 ;  
61 010062 004737 012444' 1#: CALL GENESC ;Generate ESC character  
62 010066 SEND #'H ;Send H -- Home cursor  
63 010076 004737 012444' CALL GENESC ;Generate ESC character  
64 010102 SEND #'J ;Send J -- Clear to end of screen  
65 010112 004737 012444' CALL GENESC ;Generate ESC character  
66 010116 SEND #'G ;Select ascii character set  
67 ;  
68 ; Finished  
69 ;  
70 010126 000207 9#: RETURN
```

```

1          .SBTTL  GENCAF -- Generate sequence to set character attributes
2          ;-----
3          ; Generate the terminal sequence to set character attributes.
4          ;
5          ; Inputs:
6          ;   R0 = Character attribute wanted
7          ;   R2 = Pointer to window control block
8          ;   R5 = Current character attribute
9          ;
10         ; Outputs:
11         ;   R5 = New character attributes
12         ;
13 010130 010146  GENCAF: MOV     R1,-(SP)
14 010132 010346          MOV     R3,-(SP)
15 010134 010446          MOV     R4,-(SP)
16 010136 010001          MOV     R0,R1          ;Carry new attributes in R1
17         ;
18         ; See if we need to change character sets
19         ;
20 010140 004737 010362'  CALL    GENCSK          ;See if we need to do character set change
21         ;
22         ; Don't have to worry about attributes if this is a VT52
23         ;
24 010144 032762 000000C 000000G  BIT     #<AW$52!AW$52>,DW$AW(R2); Is this a VT52?
25 010152 001076          BNE     9$              ;Br if yes
26         ;
27         ; See if character attributes have changed
28         ;
29 010154 010103          MOV     R1,R3          ;Get new attributes
30 010156 010504          MOV     R5,R4          ;Get current attributes
31 010160 042703 000000C  BIC     #^C<AC$BLD!AC$BLK!AC$REV!AC$ULN>,R3
32 010164 042704 000000C  BIC     #^C<AC$BLD!AC$BLK!AC$REV!AC$ULN>,R4
33 010170 020304          CMP     R3,R4          ;Have attributes changed?
34 010172 001466          BEQ     9$              ;Br if not
35         ;
36         ; Initialize control sequence and turn off all attributes
37         ;
38 010174 004737 012456'  CALL    GENCSI          ;Send CSI
39 010200          SEND    #'0          ;Send 0
40         ;
41         ; See if bold is wanted
42         ;
43 010210 032701 000000G  BIT     #AC$BLD,R1      ;Is bold wanted?
44 010214 001410          BEQ     1$              ;Br if not
45 010216          SEND    #SEMI          ;Send ';1'
46 010226          SEND    #'1
47         ;
48         ; See if underscore is wanted
49         ;
50 010236 032701 000000G  1$: BIT     #AC$ULN,R1    ;Underline wanted?
51 010242 001410          BEQ     2$              ;Br if not
52 010244          SEND    #SEMI          ;Send ";4"
53 010254          SEND    #'4
54         ;
55         ; See if blinking is wanted
56         ;
57 010264 032701 000000G  2$: BIT     #AC$BLK,R1    ;Blinking wanted?

```

GENCAF -- Generate sequence to set character attributes

```

58 010270 001410          BEQ      3$          ;Br if not
59 010272                SEND      #SEMI        ;Send ";5"
60 010302                SEND      #'5
61                      ;
62                      ; See if reverse video is wanted
63                      ;
64 010312 032701 000000G 3$:      BIT      #AC$REV,R1      ;Reverse video wanted?
65 010316 001410          BEQ      7$          ;Br if not
66 010320                SEND      #SEMI        ;Send ";7"
67 010330                SEND      #'7
68                      ;
69                      ; Terminate the string by sending "m"
70                      ;
71 010340                7$:      SEND      #'m          ;Send "m"
72                      ;
73                      ; Finished
74                      ;
75 010350 010105          9$:      MOV      R1,R5          ;Change current attributes to new attributes
76 010352 012604          MOV      (SP)+,R4
77 010354 012603          MOV      (SP)+,R3
78 010356 012601          MOV      (SP)+,R1
79 010360 000207          RETURN

```

GENCSC -- Generate terminal sequence to select char set

```

1          .SBTTL  GENCSC -- Generate terminal sequence to select char set
2          ;-----
3          ; Generate terminal control sequence to select the correct character
4          ; set for the next character.
5          ;
6          ; Inputs:
7          ; R0 = Character attribute wanted
8          ; R2 = Pointer to window control block
9          ; R5 = Current character attribute
10         ;
11 010362 010046 GENCSC: MOV     R0,-(SP)
12 010364 010146      MOV     R1,-(SP)
13 010366 010346      MOV     R3,-(SP)
14 010370 010446      MOV     R4,-(SP)
15 010372 010001      MOV     R0,R1          ;Carry new attributes in R1
16         ;
17         ; Determine if character set has changed
18         ;
19 010374 010103      MOV     R1,R3          ;Get new attributes
20 010376 042703 000000C BIC     #^C<AC$SET>,R3 ;Select character set number only
21 010402 010504      MOV     R5,R4          ;Get current attributes
22 010404 042704 000000C BIC     #^C<AC$SET>,R4 ;Select character set number only
23 010410 020304      CMP     R3,R4          ;Has character set changed?
24 010412 001443      BEQ     9$          ;Br if char set has not changed
25         ;
26         ; Character set has changed
27         ;
28 010414 032762 000000C 000000G BIT     #<AW$52!AW$552>,DW$AW(R2);Is this a VT52 terminal?
29 010422 001031      BNE     1$          ;Br if VT52
30         ;
31         ; VT100/VT200 terminal.
32         ; Generate sequence to map G0 and G2 to correct character set
33         ;
34 010424 004737 012444' CALL    GENESC          ;Send ESC character
35 010430      SEND    #'(          ;Select G0
36 010440      SEND    SET200(R3) ;Send char to map G0 to correct char set
37 010450 032762 000000G 000000G BIT     #AW$200,DW$AW(R2);Is this a VT200 terminal?
38 010456 001421      BEQ     9$          ;Br if not -- Don't need to change G2
39 010460 004737 012444' CALL    GENESC          ;Send ESC character
40 010464      SEND    #'*          ;Select G2
41 010474      SEND    SET200(R3) ;Send char to map G2 to correct char set
42 010504 000406      BR     9$
43         ;
44         ; VT52 terminal.
45         ; Select ascii or graphics character set.
46         ;
47 010506 004737 012444' 1$: CALL    GENESC          ;Send ESC character
48 010512      SEND    SET52(R3) ;Select ascii or graphics characters
49         ;
50         ; Finished
51         ;
52 010522 012604 9$: MOV     (SP)+,R4
53 010524 012603      MOV     (SP)+,R3
54 010526 012601      MOV     (SP)+,R1
55 010530 012600      MOV     (SP)+,R0
56 010532 000207      RETURN
57         ;

```


GENCSC -- Generate terminal sequence to select char set

```
58          ; Characters to be sent to change GO character set designation
59          ;
60 010534    102    074    101 SET200: .ASCII /B<A0/
    010537    060
61          ;
62          ; Characters to select ascii or graphics mode on a VT52
63          ;
64 010540    107    106    SET52:  .ASCII /GF/
65          .EVEN
```

GENMAP -- Generate sequence to set up char set mapping

```

1          .SBTTL  GENMAP -- Generate sequence to set up char set mapping
2          ;-----
3          ; Generate control sequence to set up the correct character set
4          ; mapping for GL, GR, GO, G1, G2, and G3.
5          ;
6          ; Inputs:
7          ; R2 = Pointer to window control block
8          ;
9 010542 010346 GENMAP: MOV      R3,-(SP)
10 010544 032762 000000C 000000G BIT      #<AW$52!AW$52>,DW$AW(R2);Is this a VT52 terminal?
11 010552 001060          BNE      5$          ;Br if VT52
12          ;
13          ; This is a VT100/VT200 terminal.
14          ; Set up mapping for GL.
15          ;
16 010554 116203 000000G          MOVVB   DW$GLM(R2),R3  ;Get mapping for GL
17 010560 120327 000001          CMPB   R3,#1          ;Mapping to GO or G1?
18 010564 101402          BLOS   1$          ;Br if yes
19 010566 004737 012444'          CALL   GENESC          ;Send ESC
20 010572          1$: SEND   LSLCHR(R3)  ;Send correct lock shift character
21          ;
22          ; Set up mapping for GR
23          ;
24 010602 032762 000000G 000000G BIT      #AW$200,DW$AW(R2);Is this a VT200?
25 010610 001410          BEQ     2$          ;Br if not -- Don't need to mess with GR
26 010612 116203 000000G          MOVVB   DW$GRM(R2),R3  ;Get mapping for GR
27 010616 004737 012444'          CALL   GENESC          ;Send ESC
28 010622          SEND   LSRCHR(R3)  ;Send correct lock shift character
29          ;
30          ; Set up mapping for GO, G1, G2, and G3
31          ;
32 010632 012703 000001          2$: MOV    #1,R3          ;Assume we only need to map GO and G1
33 010636 032762 000000G 000000G BIT      #AW$200,DW$AW(R2);Is this a VT200?
34 010644 001402          BEQ     3$          ;Br if not
35 010646 012703 000003          MOV    #3,R3          ;Map G2 and G3 too
36 010652 004737 012444'          3$: CALL   GENESC          ;Send ESC
37 010656          SEND   GNMCHR(R3)  ;Select G#
38 010666 010300          MOV    R3,R0          ;Get G number
39 010670 060200          ADD    R2,R0          ;Point into window control block
40 010672 116000 000000G          MOVVB   DW$GOM(R0),R0  ;Get mapping wanted for G#
41 010676          SEND   SET200(R0)  ;Set mapping for the G#
42 010706 005303          DEC    R3            ;Have more G# to do?
43 010710 002360          BGE    3$            ;Loop if yes
44 010712 000410          BR     9$
45          ;
46          ; Set up mapping for a VT52 terminal
47          ;
48 010714 004737 012444'          5$: CALL   GENESC          ;Generate ESC character
49 010720 116203 000000G          MOVVB   DW$GOM(R2),R3  ;Get ascii/graphics mapping code
50 010724          SEND   SET52(R3)  ;Select correct mapping
51          ;
52          ; Finished
53          ;
54 010734 012603          9$: MOV    (SP)+,R3
55 010736 000207          RETURN
56          ;
57          ; Characters used to lock shift a mapping into GL

```

```
58 ;  
59 010740 017 016 156 LSLCHR: .ASCII <17><16>/no/  
010743 157  
60 ;  
61 ; Characters used to lock shift a mapping into GR  
62 ;  
63 010744 077 176 175 LSRCHR: .ASCII /?~}!/  
010747 174  
64 ;  
65 ; Characters used to select G0, G1, G2, or G3 for mapping  
66 ;  
67 010750 050 051 052 GNMCHR: .ASCII /()*+/  
010753 053  
68 .EVEN
```

GENSSS -- Generate sequence for split screen scrolling

```

1          .SBTTL  GENSSS -- Generate sequence for split screen scrolling
2          ;-----
3          ; Generate the terminal control sequence needed to set the terminal to
4          ; split screen scrolling mode if that is needed.
5          ;
6          ; Inputs:
7          ; R2 = Pointer to window control block
8          ;
9 010754   GENSSS:
10         ;
11         ; We never set split screen scrolling for VT52 terminals
12         ;
13 010754   032762   000000C 000000G          BIT    #<AW$52!AW$52>,DW$AW(R2);Is this a VT52?
14 010762   001054          BNE    9$          ;Br if yes
15         ;
16         ; See if see need to set split screen scrolling
17         ;
18 010764   026227   000000G 000001          CMP    DW$SRT(R2),#1  ;Is top line of scrolling region 1?
19 010772   001004          BNE    2$          ;Br if not
20 010774   026262   000000G 000000G          CMP    DW$SRB(R2),DW$LPP(R2);Is bottom scroll line bottom of page?
21 011002   001444          BEQ    9$          ;Br if yes
22         ;
23         ; Declare scrolling region
24         ;
25 011004   004737   012456' 2$:          CALL   GENCSI          ;Generate CSI character
26 011010   016200   000000G          MOV    DW$SRT(R2),R0  ;Get top line # of scrolling region
27 011014   004737   012520'          CALL   GENVAL          ;Cvt to ascii digits and send them
28 011020          SEND   #SEMI          ;Send semicolon
29 011030   016200   000000G          MOV    DW$SRB(R2),R0  ;Get bottom line of scrolling region
30 011034   004737   012520'          CALL   GENVAL          ;Cvt to ascii digits and send them
31 011040          SEND   #'r          ;Terminate control sequence
32         ;
33         ; See if we need to set relative origin mode
34         ;
35 011050   032762   000000G 000000G          BIT    #AW$ORS,DW$AW(R2);Is origin to be relative to scroll region?
36 011056   001416          BEQ    9$          ;Br if not
37 011060   004737   012456'          CALL   GENCSI          ;Generate CSI character
38 011064          SEND   #'?          ;Send ?
39 011074          SEND   #'6          ;Send 6
40 011104          SEND   #'h          ;Send h
41         ;
42         ; Finished
43         ;
44 011114   000207 9$:          RETURN

```

GENTEM -- Turn VT52 emulation mode on or off

```

1          .SBTTL  GENTEM -- Turn VT52 emulation mode on or off
2          ;-----
3          ; Generate the terminal control sequence to turn VT52 emulation mode
4          ; on or off as needed by this window.
5          ;
6          ; Inputs:
7          ;   R2 = Pointer to window control block
8          ;
9          ;
10         ; See if we need to go into VT52 emulation mode
11         ;
12 011116 116200 000000G GENTEM: MOVB   DW$JOB(R2),R0   ;Get our job index number
13 011122 116000 000000G         MOVB   LNPRIM(R0),R0  ;Get our primary line index number
14 011126 032762 000000G 000000G        BIT    #AW$S52,DW$AW(R2) ;Does window want VT52 emulation mode?
15 011134 001426                BEQ    2$           ;Br if not
16 011136 032760 000000G 000000G        BIT    ##V52EM,LSW11(R0);Is VT52 emulation mode on now?
17 011144 001126                BNE    9$           ;Br if yes
18 011146 052760 000000G 000000G        BIS    ##V52EM,LSW11(R0);Say we are turning on VT52 emulation
19 011154 004737 012456'          CALL   GENCSI           ;Send CSI character
20 011160                SEND   #'?           ;Send "?21" to reset ANSI mode
21 011170                SEND   #'2
22 011200                SEND   #'1
23 011210 000504                BR     9$
24         ;
25         ; See if we need to turn off VT52 emulation mode
26         ;
27 011212 032760 000000G 000000G 2$:    BIT    ##V52EM,LSW11(R0);Is VT52 emulation mode on now?
28 011220 001500                BEQ    9$           ;Br if not
29 011222 042760 000000G 000000G        BIC    ##V52EM,LSW11(R0);Say we are turning off VT52 emulation
30 011230 010046                MOV    R0,-(SP)       ;Save R0
31 011232 004737 012444'          CALL   GENESC           ;Send ESC
32 011236                SEND   #'<           ;Tell VT52 to enter ANSI mode
33 011246 012600                MOV    (SP)+,R0       ;Restore R0
34 011250 026027 000000G 000000G        CMP    LTRMTP(R0),#VT100 ;Is this a VT100?
35 011256 001461                BEQ    9$           ;BR if so
36 011260 026027 000000G 000000G        CMP    LTRMTP(R0),#VT2008 ;Is term set VT2208
37 011266 001023                BNE    8$           ;BR if not
38 011270 004737 012456'          CALL   GENCSI           ;Send CSI
39 011274                SEND   #'6           ;Send rest of sequence to set
40 011304                SEND   #'2           ;VT220 8 bit contrils
41 011314                SEND   #'"
42 011324                SEND   #'p
43 011334 000432                BR     9$
44 011336 004737 012456'          8$:    CALL   GENCSI           ;Send CSI
45 011342                SEND   #'6           ;Sent reset to sequence to set
46 011352                SEND   #'2           ;VT220 7 bit contrils
47 011362                SEND   #SEMI
48 011372                SEND   #'1
49 011402                SEND   #'"
50 011412                SEND   #'p
51         ;
52         ; Finished
53         ;
54 011422 000207          9$:    RETURN

```

GENWAF -- Generate terminal sequence to set window attrib

```

1          .SBTTL  GENWAF -- Generate terminal sequence to set window attrib
2          ;-----
3          ; Generate the terminal control sequence needed to set attributes for
4          ; the entire window.
5          ;
6          ; Inputs:
7          ; R2 = Pointer to window control block
8          ;
9 011424 010346 GENWAF: MOV     R3, -(SP)
10 011426 010446      MOV     R4, -(SP)
11          ;
12          ; Get control flags for the window attributes
13          ;
14 011430 016203 000000G      MOV     DW$AW(R2), R3      ;Get window attribute flags
15          ;
16          ; Generate control sequence to reset some flags
17          ;
18 011434 005103      COM     R3          ;Set flags that should be reset
19 011436 012704 000154      MOV     #'l, R4          ;Terminate string with lower-case L
20 011442 004737 011534'      CALL    GENAAS          ;Generate for ANSI attributes
21 011446 004737 011576'      CALL    GENDAS          ;Generate for DEC attributes
22          ;
23          ; Generate control sequence to set some flags
24          ;
25 011452 005103      COM     R3          ;Set flags that should be set
26 011454 012704 000150      MOV     #'h, R4          ;Terminate string with lower-case H
27 011460 004737 011534'      CALL    GENAAS          ;Generate for ANSI attributes
28 011464 004737 011576'      CALL    GENDAS          ;Generate for DEC attributes
29          ;
30          ; See if we should set or reset application keypad mode
31          ;
32 011470 004737 012444'      CALL    GENESC          ;Send escape
33 011474 032762 000000G 000000G  BIT    #AW$AKM, DW$AW(R2); Is application keypad mode wanted?
34 011502 001405      BEQ     1$,          ;Br if not
35 011504          SEND    #'=          ;Send equal sign
36 011514 000404      BR     9$,          ;
37 011516          1$: SEND    #'>          ;Send greater-than sign
38          ;
39          ; Finished
40          ;
41 011526 012604      9$:  MOV     (SP)+, R4
42 011530 012603      MOV     (SP)+, R3
43 011532 000207      RETURN

```

GENAAS -- Generate control sequence for ANSI attributes

```

1          .SBTTL  GENAAS -- Generate control sequence for ANSI attributes
2          ;-----
3          ; Generate the control sequence to set or reset ANSI attribute flags.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;   R3 = Word with flags
8          ;   R4 = Character to terminate string with
9          ;
10         011534 GENAAS:
11         ;
12         ; Don't send anything if this is a VT52
13         ;
14         011534 032762 000000C 000000G          BIT    #<AW$52!AW$S52>,DW$AW(R2);Is this a VT52?
15         011542 001014          BNE    9$          ;Br if yes
16         ;
17         ; See if insert mode is wanted
18         ;
19         011544 032703 000000G          BIT    #AW$INS,R3          ;Insert mode wanted?
20         011550 001411          BEQ    9$          ;Br if not
21         011552 004737 012456'          CALL   GENCSI          ;Send CSI character
22         011556          SEND   #'4          ;Send 4
23         011566          SEND   R4          ;Terminate string
24         ;
25         ; Finished
26         ;
27         011574 000207          9$:    RETURN

```

GENDAS -- Generate control sequence for DEC attributes

```

1          .SBTTL  GENDAS -- Generate control sequence for DEC attributes
2          ;-----
3          ; Generate a terminal control sequence to set or reset DEC attributes.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to window control block
7          ;   R3 = AW$xxx attribute flags
8          ;   R4 = Character to terminate string with
9          ;
10         011576 010146 GENDAS: MOV     R1, -(SP)
11         011600 010346         MOV     R3, -(SP)
12         011602 010546         MOV     R5, -(SP)
13         ;
14         ; We don't have to set any DEC private attributes for VT52 terminals.
15         ;
16         011604 032762 000000C 000000G         BIT     #<AW$52!AW$S52>, DW$AW(R2); Is this a VT52?
17         011612 001056         BNE     9$          ; Br if VT52
18         ;
19         ; VT100/VT200
20         ; If this is a VT100, clear VT200-only attribute flags.
21         ;
22         011614 032762 000000G 000000G         BIT     #AW$200, DW$AW(R2); Is this a VT200?
23         011622 001002         BNE     5$          ; Br if VT200
24         011624 042703 000000G         BIC     #AW$VCR, R3          ; Clear VT200-only flags
25         ;
26         ; Init flag that says string hasn't been initiated yet
27         ;
28         011630 005005 5$:      CLR     R5          ; Nothing sent yet
29         ;
30         ; Begin loop to decide what values need to be sent
31         ;
32         011632 012701 011760'         MOV     #DECAT, R1          ; Point to attribute table
33         ;
34         ; See if next attribute needs to be selected
35         ;
36         011636 036103 0000002 1$:      BIT     2(R1), R3          ; Is this attribute needed?
37         011642 001431         BEQ     4$          ; Br if not
38         ;
39         ; We need to select this attribute, see if we need to send
40         ; CSI to start the string or semicolon to separate parameter values
41         ;
42         011644 005705         TST     R5          ; Have we initiated the string?
43         011646 001010         BNE     2$          ; Br if yes
44         011650 004737 012456'         CALL   GENCSI          ; Send CSI character
45         011654         SEND   #'?          ; Send question mark
46         011664 005205         INC     R5          ; Remember string started
47         011666 000404         BR     3$          ;
48         011670 2$:      SEND   #SEMI          ; Send semicolon
49         ;
50         ; Now send the parameter value to selete the attribute
51         ;
52         011700 011100 3$:      MOV     (R1), R0          ; Get parameter value for attribute
53         011702 004737 012520'         CALL   GENVAL          ; Convert value to digit string and send
54         ;
55         ; If this is a VT100, terminate the string for each attribute.
56         ;
57         011706 032762 000000G 000000G         BIT     #AW$200, DW$AW(R2); VT200?

```


GENDAS -- Generate control sequence for DEC attributes

```

58 011714 001004          BNE      4$          ;Br if yes
59 011716                SEND     R4          ;Send terminating character
60 011724 005005          CLR      R5          ;Say string is not in progress
61                      ;
62                      ; See if more attributes need to be selected
63                      ;
64 011726 062701 000004  4$:   ADD     #4,R1          ;Point to next attribute entry
65 011732 005711          TST     (R1)         ;Reached end of table?
66 011734 001340          BNE     1$          ;Loop if not
67                      ;
68                      ; If we started the string, terminate it
69                      ;
70 011736 005705          TST     R5          ;Did we start the string?
71 011740 001403          BEQ     9$          ;Br if not
72 011742                SEND     R4          ;Send terminating character
73                      ;
74                      ; Finished
75                      ;
76 011750 012605  9$:   MOV     (SP)+,R5
77 011752 012603          MOV     (SP)+,R3
78 011754 012601          MOV     (SP)+,R1
79 011756 000207          RETURN
80                      ;
81                      ;-----
82                      ; Table of attribute values and DEC attribute flags
83                      ;
84 011760 000001 000000G  DECAT: .WORD 1.,AW$ACK      ;Application mode for cursor keys
85 011764 000003 000000G      .WORD 3.,AW$132      ;132 column mode
86 011770 000005 000000G      .WORD 5.,AW$REV      ;Reverse video mode
87 011774 000010 000000G      .WORD 8.,AW$RPT      ;Automatic keypad repeat
88 012000 000031 000000G      .WORD 25.,AW$VCR     ;Make cursor visible
89 012004 000000          .WORD 0          ;End of table

```

GENLAF -- Generate terminal sequence to set line attrib

```

1          .SBTTL  GENLAF -- Generate terminal sequence to set line attrib
2          ;-----
3          ; Generate the terminal control sequence needed to set the line attribute.
4          ;
5          ; Inputs:
6          ;   RO = Line attribute flags (AL$xxx)
7          ;
8 012006 010546 GENLAF: MOV      R5, -(SP)
9 012010 010005      MOV      RO, R5          ;Get attribute flags
10         ;
11         ; See if double height wanted
12         ;
13 012012 032705 000000C      BIT      #<AL$DHT!AL$DHB>, R5; Double high wanted?
14 012016 001422      BEQ      1$,          ;Br if not
15 012020 004737 012444'      CALL     GENESC          ;Send ESC
16 012024      SEND     #'#          ;Send "#"
17 012034 032705 000000G      BIT      #AL$DHT, R5      ;Top or bottom half of char?
18 012040 001405      BEQ      2$,          ;Br if bottom
19 012042      SEND     #'3          ;Send 3
20 012052 000404      BR       1$,
21 012054      2$: SEND     #'4          ;Send 4
22         ;
23         ; See if double width is wanted
24         ;
25 012064 032705 000000G      1$: BIT      #AL$DWD, R5      ;Is double width wanted?
26 012070 001412      BEQ      9$,          ;Br if not
27 012072 004737 012444'      CALL     GENESC          ;Send ESC
28 012076      SEND     #'#          ;Send "#"
29 012106      SEND     #'6          ;Send 6
30         ;
31         ; Finished
32         ;
33 012116 012605      9$: MOV      (SP)+, R5
34 012120 000207      RETURN

```

GENSPC -- Generate sequence to move cursor over

```

1          .SBTTL  GENSPC -- Generate sequence to move cursor over
2          ;-----
3          ; Generate spaces or a control sequence to move the cursor right
4          ; from the current cursor position (specified by DW$COL(R2)) to
5          ; a specified column.
6          ;
7          ; Inputs:
8          ;   R2 = Pointer to window control block
9          ;   R1 = Column number we are to move to
10         ;
11 012122  010346  GENSPC: MOV      R3, -(SP)
12         ;
13         ; Determine how many columns we want to move over
14         ;
15 012124  010103          MOV      R1, R3          ; Get destination column
16 012126  166203 000000G  SUB      DW$COL(R2), R3      ; Subtract current column number
17 012132  003452          BLE      9$              ; Br if don't need to move at all
18         ;
19         ; If we need to move no more than 5 columns, output spaces
20         ;
21 012134  020327 000005G  CMP      R3, #5          ; Need to move more than 5 columns?
22 012140  003006          BGT      2$              ; Br if yes
23 012142          1$: SEND      #40          ; Send a space
24 012152  077305          SOB      R3, 1$         ; Loop to send more
25 012154  000441          BR       9$              ; Finished
26         ;
27         ; We need to move more than 5 columns; generate control sequence.
28         ; Determine terminal type.
29         ;
30 012156  032762 000000C 000000G 2$: BIT      #<AW$52!AW$S52>, DW$AW(R2); VT52 terminal?
31 012164  001012          BNE      5$              ; Br if VT52
32         ;
33         ; Generate control sequence for VT100
34         ;
35 012166  004737 012456'  CALL     GENCSI          ; Generate CSI header
36 012172  010300          MOV      R3, R0          ; Get # columns we need to move
37 012174  004737 012520'  CALL     GENVAL          ; Convert and output that value
38 012200          SEND      #'C          ; Terminate control sequence
39 012210  000423          BR       9$
40         ;
41         ; Generate control sequence for VT52
42         ;
43 012212  004737 012444'  5$: CALL     GENESC          ; Generate escape
44 012216          SEND      #'Y          ; Start of cursor addressing sequence
45 012226  016200 000000G  MOV      DW$LIN(R2), R0    ; Get current line number
46 012232  062700 000037  ADD      #37, R0          ; Form ascii char for line number
47 012236          SEND      R0          ; Send it
48 012244  010100          MOV      R1, R0          ; Get wanted column number
49 012246  062700 000037  ADD      #37, R0          ; Form ascii char for column number
50 012252          SEND      R0          ; Send it
51         ;
52         ; Change current column number
53         ;
54 012260  010162 000000G  9$: MOV      R1, DW$COL(R2) ; Say we have moved up to this column
55         ;
56         ; Finished
57         ;

```

58 012264 012603
59 012266 000207

MOV (SP)+,R3
RETURN

GENCSR -- Generate cursor addressing sequence

```

1
2
3
4
5
6
7
8
9
10
11 012270
12
13
14
15 012270 032762 000000C 000000G
16 012276 001035
17
18
19
20 012300 004737 012456'
21 012304 016200 000000G
22 012310 032762 000000G 000000G
23 012316 001406
24 012320 166200 000000G
25 012324 005200
26 012326 003002
27 012330 012700 000001
28 012334 004737 012520'
29 012340
30 012350 016200 000000G
31 012354 004737 012520'
32 012360
33 012370 000424
34
35
36
37 012372 004737 012444'
38 012376
39 012406 016200 000000G
40 012412 062700 000037
41 012416
42 012424 016200 000000G
43 012430 062700 000037
44 012434
45
46
47
48 012442 000207

```

```

.SBTTL GENCSR -- Generate cursor addressing sequence
-----
; Generate the appropriate terminal control sequence to move the
; cursor to the current line and column.
;
; Inputs:
; R2 = Pointer to window control block
; DW$LIN(R2) = Current line number
; DW$COL(R2) = Current column number
;
GENCSR:
; Determine what type of terminal this is
;
; BIT #<AW$52!AW$55>,DW$AW(R2); Is this a VT52?
; BNE 5$ ; Br if VT52
;
; Generate sequence for a VT100
;
; CALL GENCSI ; Generate CSI character
; MOV DW$LIN(R2),R0 ; Get line number
; BIT #AW$ORS,DW$AW(R2); Is origin relative to scroll region?
; BEQ 1$ ; Br if not
; SUB DW$SRT(R2),R0 ; Subtract top line of scroll region
; INC R0 ; 1st line is 1
; BGT 1$ ; Br if we are within scroll region
; MOV #1,R0 ; Force to 1st line of scroll region
1$: ;
; CALL GENVAL ; Convert to ascii digit string
; SEND #SEMI ; Send semicolon
; MOV DW$COL(R2),R0 ; Get column number
; CALL GENVAL ; Convert to ascii digit string
; SEND #'H ; Terminate sequence
; BR 9$
;
; Generate control sequence for VT52
;
5$: ;
; CALL GENESC ; Generate escape
; SEND #'Y ; Start of cursor addressing sequence
; MOV DW$LIN(R2),R0 ; Get current line number
; ADD #37,R0 ; Form ascii char for line number
; SEND R0 ; Send it
; MOV DW$COL(R2),R0 ; Get current column number
; ADD #37,R0 ; Form ascii char for column number
; SEND R0 ; Send it
;
; Finished
;
9$: RETURN

```

GENESC -- Generate escape character

```

1          .SBTTL  GENESC -- Generate escape character
2          ;-----
3          ; Send an escape character to the terminal
4          ;
5 012444   GENESC: SEND  #33          ;Send escape to the terminal
6 012454   000207   RETURN
7
8          .SBTTL  GENCSI -- Generate CSI character sequence
9          ;-----
10         ; Send a CSI (ESC [) sequence to the terminal.
11         ;
12 012456   GENCSI:
13         ;
14         ; Send 8-bit CSI character to VT200 terminals
15         ;
16         ; BIT #AW$200,DW$AW(R2); Is this a VT200 terminal?
17         ; BEQ 1$ ;Br if not
18         ; SEND #CSICHR ;Send 8-bit CSI character to VT200
19         ; BR 9$
20         ;
21         ; Send ESC [ to VT100 terminals
22         ;
23 012456   1$: SEND #33          ;Send escape
24 012466   SEND #'[          ;Send "["
25         ;
26         ; Finished
27         ;
28 012476   000207   9$: RETURN
29
30         .SBTTL  GENCHR -- Send a character to the terminal
31         ;-----
32         ; Send a single character to the terminal
33         ;
34         ; Inputs:
35         ; R0 = Character to be sent
36         ; R2 = Pointer to window control block
37         ;
38 012500   010146   GENCHR: MOV R1, -(SP)
39 012502   116201   000000G   MOVB DW$JOB(R2),R1 ;Get # of job we are sending to
40 012506   OCALL  BUFCHR ;Send the character
41 012514   012601   MOV (SP)+,R1
42 012516   000207   RETURN

```

GENVAL -- Convert value to digits and send

```

1          .SBTTL  GENVAL -- Convert value to digits and send
2          ;-----
3          ; Convert a binary value to an ascii digit string and send to the
4          ; terminal.
5          ;
6          ; Inputs:
7          ; R0 = Value to be converted and sent
8          ;
9 012520 010446 GENVAL: MOV     R4, -(SP)
10 012522 010546      MOV     R5, -(SP)
11 012524 010005      MOV     R0, R5          ;Get value to be converted
12          ;
13          ; Convert to digits and stack the digits
14          ;
15 012526 005046      CLR     -(SP)          ;Store null on stack to indicate end
16 012530 005004 1$:  CLR     R4              ;Clear high-order for divide
17 012532 071427 000012 DIV     #10, R4          ;Divide R4-R5 by 10.
18 012536 062705 000060 ADD     #'0, R5          ;Convert remainder to ascii digit
19 012542 010546      MOV     R5, -(SP)      ;Stack the character
20 012544 010405      MOV     R4, R5        ;Get quotient
21 012546 001370      BNE     1$            ;Loop if more digits to convert
22          ;
23          ; Now pop the digits and send them
24          ;
25 012550 012600 2$:  MOV     (SP)+, R0      ;Get next char to send
26 012552 001404      BEQ     9$            ;Br if finished
27 012554      SEND     R0                ;Send a digit
28 012562 000772      BR     2$            ;Send rest of number
29          ;
30          ; Finished
31          ;
32 012564 012605 9$:  MOV     (SP)+, R5
33 012566 012604      MOV     (SP)+, R4
34 012570 000207      RETURN

```

EMTWIN -- Dispatch window control EMT's

```

1          .SBTTL  EMTWIN -- Dispatch window control EMT's
2          ;-----
3          ; EMTWIN is entered from EMT processing when a window control EMT
4          ; is executed.  It dispatches control to the correct processing
5          ; routine based on the subfunction code.
6          ;
7          ; Inputs:
8          ;   R1 = Job index number
9          ;   EMTBLK = EMT argument block
10         ;
11 012572  EMTWIN:
12         ;
13         ; Ignore window control EMT's if this is a detached job
14         ;
15 012572  113700  000000G      MOVB   EMTBLK,RO      ;Get EMT subfunction code
16 012576  020027  000006      CMP    RO,#6        ;Read-window-data function?
17 012602  001407              BEQ    2$            ;Br if yes -- OK for detached jobs
18 012604  032761  000000G 000000G  BIT    ##DETC,LSW(R1) ;Is this a detached job?
19 012612  001403              BEQ    2$            ;Br if not
20 012614  005000              CLR    RO          ;Return error 0 for detached job
21 012616  000137  000000G      JMP    SETERR
22         ;
23         ; Get EMT subfunction code and make sure it is valid
24         ;
25 012622  006300              2$:  ASL    RO          ;Convert subfun code to word table index
26 012624  020027  000020      CMP    RO,#MAXSFC    ;Is the subfunction code valid?
27 012630  103403              BLD    1$          ;Br if yes
28 012632  005000              CLR    RO          ;Return error code 0 if not
29 012634  000137  000000G      JMP    SETERR
30         ;
31         ; Enter processing routine
32         ;
33 012640  000170  012644'     1$:  JMP    @WINVEC(RO) ;Enter processing routine
34         ;
35         ; Branch vector for subfunction codes
36         ;
37 012644  012664'  WINVEC: .WORD  WFNEW      ;0 - Create a new window
38 012646  013354'      .WORD  WFMAP      ;1 - Select a window
39 012650  013422'      .WORD  WFDEL      ;2 - Delete a window
40 012652  013534'      .WORD  WFSPND     ;3 - Suspend window processing
41 012654  013554'      .WORD  WFRSUM     ;4 - Resume window processing
42 012656  013574'      .WORD  WFPRNT     ;5 - Print the specified window
43 012660  013642'      .WORD  WFPREAD    ;6 - Read window data into program buffer
44 012662  014124'      .WORD  WFSTT      ;7 - Set terminal type
45         000020      MAXSFC = .-WINVEC ;Highest branch vector index
46

```


WFNEW -- EMT to create a new window

```

1          .SBTTL  WFNEW  -- EMT to create a new window
2          ;-----
3          ; WINNEW is the EMT used to define a new window.
4          ; The form of the EMT argument block is:
5          ;
6          ;     .BYTE    0,161
7          ;     .BYTE    window_number,perm_flag
8          ;     .BYTE    window_width,max_scroll
9          ;     .WORD    copy_window,copy_job
10         ;     .WORD    0
11         ;
12         ; Inputs:
13         ; R1 = Job index number
14         ; EMTBLK = EMT argument block
15         ;
16 012664  WFNEW:
17         ;
18         ; If a window with this number already exists for this job, delete it.
19         ;
20 012664  113702  000002G      MOVB    EMTBLK+2,R2      ;Get specified window number
21 012670  004737  014324'      CALL    WINSRC        ;Try to locate window control block
22 012674  103402                      BCS    1$             ;Br if that window not currently defined
23 012676  004737  014270'      CALL    WINDEL        ;Delete the window
24         ;
25         ; Try to find a free window control block
26         ;
27 012702  013702  000002'      1$:    MOV     DWBAS,R2      ;Point to 1st window control block
28 012706  105762  000000G      2$:    TSTB   DW$JOB(R2)    ;Is this block free?
29 012712  001411                      BEQ    5$             ;Br if yes
30 012714  062702  000000G      ADD     #DW$$SZ,R2      ;Point to next window block
31 012720  020237  000004'      CMP     R2,DWEND       ;Checked all blocks?
32 012724  103770                      BLO   2$             ;Br if yes
33 012726  012700  000001      MOV     #1,R0          ;Return error 0 if no free blocks
34 012732  000137  000000G      JMP    SETERR
35         ;
36         ; Initialize the window control block
37         ;
38 012736  012700  000000C      5$:    MOV     #DW$$SZ/2,R0 ;Get # words in window block
39 012742  010203                      MOV     R2,R3          ;Get pointer to control block
40 012744  005023      3$:    CLR     (R3)+        ;Zero the entire block
41 012746  077002                      SOB    R0,3$
42 012750  110162  000000G      MOVB   R1,DW$JOB(R2)   ;Set our job number
43 012754  113762  000002G 000000G  MOVB   EMTBLK+2,DW$ID(R2);Set window ID number
44 012762  012762  000000C 000000G  MOV    #<AW$VCR!AW$RPT>,DW$AW(R2);Initialize window attribute flags
45 012770  113700  000004G      MOVB   EMTBLK+4,R0     ;Get # columns per line
46 012774  042700  177400      BIC    #^C<377>,R0     ;Kill sign extension
47 013000  001002                      BNE   11$            ;Br if width not zero
48 013002  012700  000120      MOV    #80.,R0        ;Set width to 80
49 013006  020027  000204      11$:   CMP    R0,#132.   ;Compare with max width allowed
50 013012  101402                      BLOS  12$            ;Br if ok
51 013014  012700  000204      MOV    #132.,R0       ;Constrain to 132
52 013020  010062  000000G      12$:   MOV    R0,DW$CPL(R2) ;Set number of columns per line
53 013024  012762  000030 000000G  MOV    #24.,DW$LPP(R2) ;Set number of lines per page
54 013032  113762  000005G 000000G  MOVB   EMTBLK+5,DW$MSL(R2);Set max-scrolled-lines parameter
55 013040  012762  000001 000000G  MOV    #1,DW$TLN(R2)  ;Say line 1 is at top of buffer
56 013046  012762  000001 000000G  MOV    #1,DW$SRT(R2)  ;Say line 1 is top of scrolling region
57 013054  016262  000000G 000000G  MOV    DW$LPP(R2),DW$SRB(R2);scrolling region goes to end of page

```

WFNEW -- EMT to create a new window

```

58 ;
59 ; Initialize terminal type
60 ;
61 013062 004737 014160' CALL WINSTT ;Set correct terminal type
62 ;
63 ; Initialize character set mapping for a VT200
64 ;
65 013066 032761 000000C 000000G BIT #<VT2007!VT2008>,LTRMTP(R1); Is terminal a VT200?
66 013074 001411 BEQ 7$ ;Br if not
67 013076 112762 000001 000000G MOVB #1,DW$G2M(R2) ;Map G2 to DEC supplemental chars
68 013104 112762 000001 000000G MOVB #1,DW$G3M(R2) ;Map G3 to DEC supplemental chars
69 013112 112762 000002 000000G MOVB #2,DW$GRM(R2) ;Map GR to G2
70 ;
71 ; Set application keypad mode if Single line editor is in KED mode
72 ;
73 013120 032761 000000G 000000G 7$: BIT ##SLON,LSW7(R1) ;Is single-line editor turned on?
74 013126 001407 BEQ 10$ ;Br if not
75 013130 032761 000000G 000000G BIT ##SLKED,LSW7(R1); Is SL in KED mode?
76 013136 001403 BEQ 10$ ;Br if not
77 013140 052762 000000G 000000G BIS #AW$AKM,DW$AW(R2); Set application keypad mode
78 ;
79 ; See if this window should be permanent
80 ;
81 013146 105737 000003G 10$: TSTB EMTBLK+3 ;Is permanent-window flag set?
82 013152 001403 BEQ 8$ ;Br if not
83 013154 052762 000000G 000000G BIS #AW$PRM,DW$AW(R2); Remember window is permanent
84 ;
85 ; See if we should copy any window attributes from an existing window
86 ;
87 013162 005737 000006G 8$: TST EMTBLK+6 ;Should we copy attributes from another windo?
88 013166 001443 BEQ 13$ ;Br if not
89 013170 010203 MOV R2,R3 ;Save addr of new window control block
90 013172 013702 000006G MOV EMTBLK+6,R2 ;Get window_id and job number
91 013176 010201 MOV R2,R1
92 013200 042702 177400 BIC #^C<377>,R2 ;Clear all but window ID
93 013204 000301 SWAB R1 ;Get job # to low-order byte
94 013206 042701 177400 BIC #^C<377>,R1 ;Clear all but job number
95 013212 006301 ASL R1 ;Get job index number
96 013214 001002 BNE 14$ ;Br if job number specified
97 013216 113701 000000G MOVB CORUSR,R1 ;Assume current job
98 013222 004737 014324' 14$: CALL WINSRC ;Try to find existing window control block
99 013226 103422 BCS 15$ ;Br if could not find one
100 013230 016263 000000G 000000G MOV DW$LPP(R2),DW$LPP(R3) ;Copy # lines per page
101 013236 016263 000000G 000000G MOV DW$CPL(R2),DW$CPL(R3) ;Copy # columns per line
102 013244 116263 000000G 000000G MOVB DW$MSL(R2),DW$MSL(R3) ;Copy max # scroll lines
103 013252 016200 000000G MOV DW$AW(R2),R0 ;Get window attribute flags
104 013256 042700 000000C BIC #^C<AW$132!AW$REV>,R0 ;Leave only ones to be copied
105 013262 042763 000000C 000000G BIC #<AW$132!AW$REV>,DW$AW(R3) ;Clear in new control block
106 013270 050063 000000G BIS R0,DW$AW(R3) ;Transfer the flags
107 013274 010302 15$: MOV R3,R2 ;Get back address of new window control block
108 ;
109 ; Create a named local region for the screen buffer for the window
110 ;
111 013276 004737 014370' 13$: CALL MAKWSB ;Make a window screen buffer
112 013302 103006 BCC 4$ ;Br if we made the region
113 013304 004737 014270' CALL WINDEL ;Delete the window
114 013310 012700 000002 MOV #2,R0 ;Return error code 2

```

WFNEW -- EMT to create a new window

```

115 013314 000137 000000G          JMP      SETERR
116                                ;
117                                ; Clear entire screen contents to blanks
118                                ;
119 013320 004737 006216'        4#:      CALL      ERSPAG          ;Clear all positions to spaces
120                                ;
121                                ; Set line 1, column 1 as cursor position
122                                ;
123 013324 012700 000001          MOV      #1,R0          ;Get value 1
124 013330 110062 000000G        MOVB     RO,DW$SLN(R2) ;Init saved line # to 1
125 013334 110062 000000G        MOVB     RO,DW$SCL(R2) ;Init saved column # to 1
126 013340 010062 000000G        MOV      RO,DW$COL(R2) ;Set column = 1
127 013344 004737 006710'        CALL     SETLIN        ;Set line =1
128                                ;
129                                ; Finished
130                                ;
131 013350 000137 000000G          JMP      EMTXIT

```

WFMAP -- EMT to select a window as the current window

```

1          .SBTTTL  WFMAP  -- EMT to select a window as the current window
2          ;-----
3          ; Select a window as the current window for the job and refresh
4          ; the display screen from the window contents.
5          ;
6          ; Inputs:
7          ;   R1 = Job index number
8          ;   EMTBLK = EMT argument block
9          ;
10         013354 WFMAP:
11         ;
12         ;   If current window number is zero, switch back to no window
13         ;
14         013354 013702 000002G          MOV     EMTBLK+2,R2      ;Get specified window number
15         013360 001003                   BNE     2$              ;Br if some window specified
16         013362 005061 000000G          CLR     LWINDO(R1)     ;Say we are not doing windowing
17         013366 000413                   BR      9$
18         ;
19         ;   Locate the window control block for the specified window
20         ;
21         013370 004737 014324' 2$:    CALL    WINSRC        ;Find window control block
22         013374 103004                   BCC     1$              ;Br if found it
23         013376 012700 000003           MOV     #3,R0           ;Error 3 if not
24         013402 000137 000000G          JMP     SETERR
25         ;
26         ;   Set this window as the current window for the job
27         ;
28         013406 010261 000000G 1$:    MOV     R2,LWINDO(R1) ;Say this is current window for job
29         ;
30         ;   Refresh the display from selected window
31         ;
32         013412 004737 007072'          CALL    REFRSH         ;Refresh display from window contents
33         ;
34         ;   Finished
35         ;
36         013416 000137 000000G 9$:    JMP     EMTXIT

```

WFDEL -- EMT to delete a window

```

1          .SBTTL  WFDEL  -- EMT to delete a window
2          ;-----
3          ; EMT to delete a window.
4          ;
5          ; Inputs:
6          ; R1 = Job index number
7          ;
8 013422   WFDEL:
9          ;
10         ; Try to locate control block for the window
11         ;
12 013422   113702   000002G          MOVB    EMTBLK+2,R2      ;Get specified window number
13 013426   001406          BEQ      3$              ;Br if window number is zero
14 013430   120227   177777          CMPB    R2,#-1        ;Is window number -1?
15 013434   001024          BNE      2$              ;Br if not
16         ;
17         ; Delete all windows for job (temp and permanent)
18         ;
19 013436   004737   014236'          CALL    WINREL        ;Release all windows for the job
20 013442   000432          BR      9$
21         ;
22         ; Window number zero means delete all temporary windows
23         ;
24 013444   013702   000002'          3$:    MOV     DWBAS,R2      ;Point to 1st window control block
25 013450   120162   000000G          5$:    CMPB   R1,DW$JOB(R2) ;Does this window belong to our job?
26 013454   001006          BNE     4$              ;Br if not
27 013456   032762   000000G 000000G  BIT     #AW$PRM,DW$AW(R2);Is this a perm or temp window?
28 013464   001002          BNE     4$              ;Br if permanent window
29 013466   004737   014270'          CALL    WINDEL        ;Delete this window
30 013472   062702   000000G          4$:    ADD     #DW$$SZ,R2   ;Point to next window control block
31 013476   020237   000004'          CMP     R2,DWEND      ;Checked all?
32 013502   103762          BLO     5$              ;Br if not
33 013504   000411          BR      9$
34         ;
35         ; We are deleting a specific window
36         ;
37 013506   004737   014324'          2$:    CALL    WINSRC      ;Try to locate window control block
38 013512   103004          BCC     1$              ;Br if located the window control block
39 013514   012700   000003          MOV     #3,RO         ;Error code 3 if cannot locate window
40 013520   000137   000000G          JMP     SETERR
41         ;
42         ; Delete the window
43         ;
44 013524   004737   014270'          1$:    CALL    WINDEL        ;Delete the window
45         ;
46         ; Finished
47         ;
48 013530   000137   000000G          9$:    JMP     EMTXIT

```

WFSPND -- Suspend window processing

```

1          .SBTTL  WFSPND -- Suspend window processing
2          ;-----
3          ; Suspend window processing.  After executing this emt, window
4          ; processing is suspended.  Characters are passed through to the
5          ; screen without being processed by the window routine.
6          ;
7          ; Inputs:
8          ;   R1 = Job index number
9          ;
10         013534 016102 000000G WFSPND: MOV     LWINDO(R1),R2  ;Get pointer to current window control block
11         013540 001403          BEQ     9$              ;Br if job has no window active now
12         013542 052762 000000G 000000G      BIS     #AW$SPN,DW$AW(R2);Set flag suspending window processing
13         013550 000137 000000G      9$:  JMP     EMTXIT          ;Finished
14
15         .SBTTL  WFRSUM -- Resume window processing
16         ;-----
17         ; Resume window processing.
18         ;
19         ; Inputs:
20         ;   R1 = Job index number
21         ;
22         013554 016102 000000G WFRSUM: MOV     LWINDO(R1),R2  ;Get pointer to current window control block
23         013560 001403          BEQ     9$              ;Br if job has no window active now
24         013562 042762 000000G 000000G      BIC     #AW$SPN,DW$AW(R2);Say window no longer suspended
25         013570 000137 000000G      9$:  JMP     EMTXIT          ;Finished

```

WFPRT -- Cause contents of a window to be printed

```

1          .SBTTL  WFPRT -- Cause contents of a window to be printed
2          ;-----
3          ; Cause the contents of a window for the current job to be printed.
4          ;
5          ; Inputs:
6          ;   R1 = Job index number
7          ;
8          ; EMT argument block:
9          ;
10         ;   .BYTE  5,161
11         ;   .BYTE  window_id,0
12         ;
13 013574  WFPRT:
14         ;
15         ;   Locate the window control block for the specified window
16         ;
17 013574  113702  000002G      MOVB    EMTBLK+2,R2      ;Get specified window ID
18 013600  001403              BEQ     3$              ;Zero is invalid
19 013602  004737  014324'    CALL    WINSRC        ;Find window control block
20 013606  103004              BCC    1$              ;Br if found it
21 013610  012700  000003    3$:   MOV     #3,R0          ;Error 3 if invalid window ID
22 013614  000137  000000G    JMP     SETERR
23         ;
24         ;   Print this window
25         ;
26 013620  004737  015074'    1$:   CALL    WINPRT      ;Try to print the window
27 013624  103004              BCC    9$              ;Br if successful
28 013626  012700  000004    MOV     #4,R0          ;Error 4 if can't print
29 013632  000137  000000G    JMP     SETERR
30         ;
31         ;   Finished
32         ;
33 013636  000137  000000G    9$:   JMP     EMTXIT

```

WFREAD -- Copy window information to program buffer

```

1          .SBTTL  WFREAD -- Copy window information to program buffer
2          ;-----
3          ; Copy window data into buffer in program area.
4          ;
5          ; EMT argument block:
6          ;
7          ; .BYTE  6,161
8          ; .BYTE  window_id,job_number
9          ; .WORD  buffer_address
10         ; .WORD  buffer_size
11         ;
12 013642  WFREAD:
13         ;
14         ; Make sure specified buffer address and size is ok
15         ;
16 013642 013700 000004G      MOV      EMTBLK+4,R0      ;Get base address of buffer
17 013646 004737 000000G      CALL     VALADW      ;Make sure it's ok
18 013652 063700 000006G      ADD      EMTBLK+6,R0      ;Add specified size
19 013656 004737 000000G      CALL     VALADB      ;Make sure end is ok too
20         ;
21         ; Get the number of the job whose window we want to read
22         ;
23 013662 113701 000003G      MOVVB   EMTBLK+3,R1      ;Get specified job number
24 013666 006301              ASL      R1              ;Convert to job index number
25 013670 001002              BNE     1$              ;Br if job number specified
26 013672 113701 000000G      MOVVB   CORUSR,R1      ;Get current job index number
27 013676 120137 000000G      1$:    CMPB   R1,CORUSR      ;Are we reading window for our job?
28 013702 001410              BEQ     5$              ;Br if yes
29 013704 032737 000000G 000000G  BIT     #P2#CXT,PRIVC2 ;Do we have GETCXT privilege?
30 013712 001004              BNE     5$              ;Br if yes
31 013714 012700 000001      MOV     #1,R0          ;Error 1 if not
32 013720 000137 000000G      JMP     SETERR
33         ;
34         ; Find the specified window control block
35         ;
36 013724 113702 000002G      5$:    MOVVB   EMTBLK+2,R2      ;Get window ID
37 013730 004737 014324'      CALL     WINSRC      ;Search for specified window
38 013734 103004              BCC     2$              ;Br if found the window control blk
39 013736 012700 000003      MOV     #3,R0          ;Invalid window ID
40 013742 000137 000000G      JMP     SETERR
41         ;
42         ; Determine how much data must be copied
43         ;
44 013746 016203 000000G      2$:    MOV     DW$LPP(R2),R3 ;Get # lines on page
45 013752 070362 000000G      MUL     DW#CPL(R2),R3 ;Times # columns per line
46 013756 006303              ASL     R3              ;Need two bytes for each char cell
47 013760 066203 000000G      ADD     DW$LPP(R2),R3 ;Add space used by line attrib vector
48 013764 005203              INC     R3              ;Round up to next word
49 013766 042703 000001      BIC     #1,R3
50 013772 010300              MOV     R3,R0          ;Get # bytes of window buffer data
51 013774 062700 000006      ADD     #3*2,R0        ;Add # bytes for parameter words
52 014000 020037 000006G      CMP     R0,EMTBLK+6   ;Is buffer large enough?
53 014004 101404              BLOS   3$              ;Br if yes
54 014006 012700 000004      MOV     #4,R0          ;Error 4 if buffer too small
55 014012 000137 000000G      JMP     SETERR
56         ;
57         ; Copy some window parameter information before actual window buffer

```


WFREAD -- Copy window information to program buffer

```

58
59 014016 013705 0000040      3$:   MOV     EMTBLK+4,R5      ;Get address of user's buffer
60 014022 016246 0000000      MOV     DW$TLN(R2),-(SP);Store top line number
61 014026 106625                MTPD    (R5)+
62 014030 016246 0000000      MOV     DW$LPP(R2),-(SP);Store # lines per page
63 014034 106625                MTPD    (R5)+
64 014036 016246 0000000      MOV     DW$CPL(R2),-(SP);Store # columns per line
65 014042 106625                MTPD    (R5)+
66
67      ; Copy data from window region to user's buffer
68      ;
69 014044 006203                ASR     R3                ;Get # words to copy
70 014046 012704 0000000      MOV     #VPAR6,R4        ;Get virt address of window region
71 014052                4$:   BUFMAP                ;;Map to window region
72 014074 012400                MOV     (R4)+,R0          ;;Get a word from screen region
73 014076                UNMAP                ;Restore mapping
74 014112 010046                MOV     R0,-(SP)          ;Push word we are transferring
75 014114 106625                MTPD    (R5)+            ;Store data into user's buffer
76 014116 077323                SOB     R3,4$            ;Loop if more data to copy
77      ;
78      ; Finished
79      ;
80 014120 000137 0000000      JMP     EMTXIT

```

WFSTT -- EMT to set terminal type for windowing

```

1          .SBTTL  WFSTT  -- EMT to set terminal type for windowing
2          ;-----
3          ; Reset the terminal type for all windows being used by this job.
4          ;
5          ; Inputs:
6          ;   R1 = Job index number.
7          ;
8 014124   WFSTT:
9          ;
10         ; Begin loop to process all windows belonging to this job
11         ;
12 014124 013702 000002'      MOV     DWBAS,R2      ;Point to 1st window control block
13 014130 120162 000000G     1$:    CMPB   R1,DW$JOB(R2)  ;Does this one belong to this job?
14 014134 001002             BNE     2$              ;Br if not
15 014136 004737 014160'     CALL   WINSTT      ;Reset terminal type for this window
16 014142 062702 000000G     2$:    ADD    #DW$$SZ,R2    ;Point to next window control block
17 014146 020237 000004'     CMP    R2,DWEND     ;Checked all?
18 014152 103766             BLD    1$            ;Loop if not
19         ;
20         ; Finished
21         ;
22 014154 000137 000000G     JMP    EMTXIT

```

WINSTT -- Set terminal type for a window

```

1          .SBTTL  WINSTT -- Set terminal type for a window
2          ;-----
3          ; WINSTT sets up the terminal type in the window control block
4          ; attribute word based on the terminal type specified for the job.
5          ;
6          ; Inputs:
7          ;   R2 = Pointer to window control block
8          ;
9 014160  010146  WINSTT: MOV      R1, -(SP)
10 014162  116201  000000G  MOVB     DW$JOB(R2), R1 ;Get job index number
11          ;
12          ; Determine terminal type based on line type
13          ;
14 014166  042762  000000C 000000G  BIC      #<AW$52!AW$200>, DW$AW(R2) ;Say not VT52 and not VT200
15          ;
16          ; See if terminal is a VT52
17          ;
18 014174  032761  000000G 000000G  BIT      #VT52, LTRMTP(R1); Is this a VT52?
19 014202  001404  BEQ      1$ ;Br if not
20 014204  052762  000000G 000000G  BIS      #AW$52, DW$AW(R2); Set VT52 as terminal type
21 014212  000407  BR       9$
22          ;
23          ; See if terminal is a VT200
24          ;
25 014214  032761  000000C 000000G 1$: BIT      #<VT2007!VT2008>, LTRMTP(R1) ;VT200 terminal?
26 014222  001403  BEQ      9$ ;Br if not
27 014224  052762  000000G 000000G  BIS      #AW$200, DW$AW(R2); Set VT200 as terminal type
28          ;
29          ; Finished
30          ;
31 014232  012601  9$: MOV      (SP)+, R1
32 014234  000207  RETURN

```

WINREL -- Release all display windows for a job

```

1          .SBTTL  WINREL -- Release all display windows for a job
2          ;-----
3          ; Release all display windows for the current job.
4          ;
5          ; Inputs:
6          ;   R1 = Job index number for job whose windows are to be released.
7          ;
8 014236   WINREL:
9          ;
10         ; Begin loop to search for window control blocks belonging to this
11         ; job.
12         ;
13 014236  013702  000002'   MOV      DWBAS,R2      ;Point to first window control block
14 014242  120162  000000G   1$:     CMPB     R1,DW$JOB(R2)  ;Is this one belong to our job?
15 014246  001002          BNE      2$          ;Br if not
16 014250  004737  014270'   CALL    WINDEL      ;Delete the windoe
17 014254  062702  000000G   2$:     ADD     #DW$$SZ,R2  ;Point to next window control block
18 014260  020237  000004'   CMP     R2,DWEND    ;Checked all?
19 014264  103766          BLO     1$          ;Loop if not
20         ;
21         ; Finished
22         ;
23 014266  000207          RETURN

```

WINDEL -- Delete a window

```

1          .SBTTL  WINDEL -- Delete a window
2          ;-----
3          ; Delete a window and release its screen buffer region.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to window control block
7          ;
8 014270  WINDEL:
9          ;
10         ; If this is the current window, reset window info for job
11         ;
12 014270 116200 000000G      MOVB   DW$JOB(R2),R0    ;Get # of job that owns this window
13 014274 020260 000000G      CMP    R2,LWINDO(R0)   ;Is this current window for the job?
14 014300 001002              BNE    1$                ;Br if not
15 014302 005060 000000G      CLR    LWINDO(R0)      ;Say job is not doing windowing
16         ;
17         ; Free the region used for the screen buffer
18         ;
19 014306 004737 014700'     1$:  CALL   FREWSB          ;Free the window screen buffer
20         ;
21         ; Free the window control block
22         ;
23 014312 105062 000000G      CLRB   DW$ID(R2)       ;Say block is free
24 014316 105062 000000G      CLRB   DW$JOB(R2)      ;Say no job using block
25         ;
26         ; Finished
27         ;
28 014322 000207              RETURN

```

WINSRC -- Locate control block for a window

```

1          .SBTTL  WINSRC -- Locate control block for a window
2          ;-----
3          ; Locate a window control block for a specified window for the current
4          ; job.
5          ;
6          ; Inputs:
7          ;   R1 = Index number of job that owns the window
8          ;   R2 = Window number
9          ;
10         ; Outputs:
11         ;   C-flag cleared ==> Found window control block
12         ;   C-flag set    ==> Cannot find window control block
13         ;   R2 = Pointer to window control block if found
14         ;
15 014324  WINSRC:
16         ;
17         ; Search for specified window control block
18         ;
19 014324 010200      MOV     R2,R0          ;Get window ID
20 014326 013702 000002'  MOV     DWBAS,R2        ;Point to first window control block
21 014332 120062 000000G 1$:  CMPB   R0,DW$ID(R2)    ;Does this block have correct ID?
22 014336 001003      BNE     2$          ;Br if not
23 014340 120162 000000G  CMPB   R1,DW$JOB(R2)   ;Is this control block for right job?
24 014344 001407      BEQ     3$          ;Br if yes
25 014346 062702 000000G 2$:  ADD     #DW$$SZ,R2    ;Point to next control block
26 014352 020237 000004'  CMP    R2,DWEND       ;Checked all?
27 014356 103765      BLD     1$          ;Loop if not
28         ;
29         ; Cannot find specified window control block
30         ;
31 014360 000261      SEC          ;Signal error on return
32 014362 000401      BR     9$
33         ;
34         ; Found the window control block
35         ;
36 014364 000241 3$:  CLC          ;Signal success on return
37         ;
38         ; Finished
39         ;
40 014366 000207 9$:  RETURN

```

MAKWSB -- Create named region for window screen buffer

```

1          .SBTTL  MAKWSB -- Create named region for window screen buffer
2          ;-----
3          ; Create a named local region large enough to hold the window
4          ; screen buffer.
5          ;
6          ; Inputs:
7          ; R2 = Pointer to window control block
8          ; EMTBLK+2 = Arg block for EMT to create the window
9          ;
10         ; Outputs:
11         ; C-flag cleared ==> Successfully created region
12         ; C-flag set    ==> Unable to create region
13         ;
14 014370 010146 MAKWSB: MOV     R1,-(SP)
15 014372 010346      MOV     R3,-(SP)
16 014374 010446      MOV     R4,-(SP)
17 014376 010546      MOV     R5,-(SP)
18 014400 010605      MOV     SP,R5      ;Save original stack pointer in R5
19         ;
20         ; Calculate number of 64-byte blocks needed for region
21         ;
22 014402 016203 000000G  MOV     DW$LPP(R2),R3  ;Get # lines on page
23 014406 070362 000000G  MUL     DW$CPL(R2),R3  ;Times number of columns
24 014412 006303          ASL     R3                ;Need two bytes per character
25 014414 066203 000000G  ADD     DW$LPP(R2),R3  ;Need byte vector for line attributes
26 014420 062703 000077  ADD     #63.,R3        ;Bound up to 64-byte boundary
27 014424 072327 177772  ASH     #-6.,R3        ;Get # 64-byte blocks needed
28         ;
29         ; Build a region definition block on the stack
30         ;
31 014430 162706 000012  SUB     #10.,SP        ;Allocate space for region def block
32 014434 010604          MOV     SP,R4          ;Get pointer to start of RDB
33 014436 004737 014570'  CALL   MAKRDB         ;Make a region definition block on the stack
34 014442 010364 000000G  MOV     R3,R.GSIZ(R4) ;Set # 64-byte units needed
35 014446 012764 000000C 000000G  MOV     #<CRS.GBL!RS.CGR>,R.GSTS(R4) ;Set status flags
36         ;
37         ; Build EMT argument block on stack
38         ;
39 014454 162706 000004  SUB     #4.,SP        ;Allocate space for EMT arg block
40 014460 010601          MOV     SP,R1          ;Save pointer to EMT arg block
41 014462 010600          MOV     SP,R0          ;Get pointer to EMT arg block area
42 014464 012710 017000  MOV     #<36*400>,(R0) ;Set EMT function code
43 014470 010460 000002  MOV     R4,2(R0)      ;Set address of region def block
44         ;
45         ; Try to create the region
46         ;
47 014474 104375          EMT     375            ;Try to create the region
48 014476 103426          BCS     9%            ;Br if error on region creation
49         ;
50         ; We successfully created the region.
51         ; Set up information in the window control block.
52         ;
53 014500 016403 000000G  MOV     R.GID(R4),R3  ;Get region control block address
54 014504 016362 000000G 000000G  MOV     RC$BLK(R3),DW$RID(R2);Save addr of global RCB
55 014512 016362 000000G 000000G  MOV     RC$BAS(R3),DW$MAP(R2);Save mapping base for region
56         ;
57         ; Now detach from the global region

```

```
58 ;  
59 014520 005064 000000G CLR R.GSTS(R4) ;Clear all status flags  
60 014524 010100 MOV R1,R0 ;Get pointer to EMT arg block  
61 014526 012710 017001 MOV #<<36*400>+1>, (R0); Set .ELRG function code  
62 014532 104375 EMT 375 ;Detach from the region  
63 ;  
64 ; Now go back and set the exclusive-access flag in the global region  
65 ; control block and increment its attachment count by one.  
66 ;  
67 014534 016203 000000G MOV DW$RID(R2),R3 ;Get pointer to global RCB  
68 014540 052763 000000G 000000G BIS #RC$EXC,RC$FLG(R3);Set exclusive-use flag  
69 014546 105263 000000G INCB RC$CNT(R3) ;Increment attachment count  
70 ;  
71 ; Successful completion  
72 ;  
73 014552 000241 CLC ;Signal success on return  
74 ;  
75 ; Finished -- Carry flag indicates results  
76 ;  
77 014554 010506 9#: MOV R5,SP ;Reset stack pointer  
78 014556 012605 MOV (SP)+,R5  
79 014560 012604 MOV (SP)+,R4  
80 014562 012603 MOV (SP)+,R3  
81 014564 012601 MOV (SP)+,R1  
82 014566 000207 RETURN
```


MAKRDB -- Make a region definition block

```

1          .SBTTL  MAKRDB -- Make a region definition block
2          ;-----
3          ; Initialize a region definition block to access a global region
4          ; used as a window buffer.
5          ;
6          ; Inputs:
7          ;   R2 = Pointer to window control block
8          ;   R4 = Pointer to area where region definition block is to be built
9          ;
10         014570  010146  MAKRDB: MOV     R1,-(SP)
11         014572  010346         MOV     R3,-(SP)
12         ;
13         ; Initially zero some cells in the RDB
14         ;
15         014574  005064  000000G      CLR     R.GID(R4)      ;Zero addr of region control block
16         014600  005064  000000G      CLR     R.GSTS(R4)     ;No status flags
17         014604  005064  000000G      CLR     R.GSIZ(R4)     ;No size specified yet
18         ;
19         ; Construct name for global region.
20         ; The first three characters are "WIN".
21         ; The next two characters are the job number.
22         ; The last character indicates the window ID.
23         ;
24         014610  013764  000010' 000000G  MOV     R5OWIN,R.NAME(R4);Set 1st 3 chars of name to "WIN"
25         014616  116201  000000G      MOVVB  DW#JOB(R2),R1    ;Get job index number
26         014622  006201          ASR     R1              ;Convert to job number
27         014624  005000          CLR     R0              ;Clear high-order for divide
28         014626  071027  000012      DIV     #10.,R0        ;Split job number into two digits
29         014632  062701  000036      ADD     #36,R1         ;Convert binary value to rad50 digit
30         014636  070127  000050      MUL     #50,R1         ;Shift remainder left 1 rad50 digit position
31         014642  010103          MOV     R1,R3          ;Save shifted low-order digit of job #
32         014644  010001          MOV     R0,R1          ;Get high-order digit of job #
33         014646  062701  000036      ADD     #36,R1         ;Convert binary value to rad50 digit
34         014652  070127  003100      MUL     #3100,R1       ;Shift over 2 rad50 digits
35         014656  060103          ADD     R1,R3          ;Job # is two high-order digits
36         014660  116201  000000G      MOVVB  DW#ID(R2),R1    ;Get window number
37         014664  060103          ADD     R1,R3          ;Put window number as 3rd digit of word
38         014666  010364  000002G      MOV     R3,R.NAME+2(R4);Set 2nd word of name
39         ;
40         ; Finished
41         ;
42         014672  012603          MOV     (SP)+,R3
43         014674  012601          MOV     (SP)+,R1
44         014676  000207          RETURN

```

```

1                                     .SBTTL  FREWSB -- Free memory used by window screen buffer
2                                     ;-----
3                                     ; Free the memory region used for a window screen buffer.
4                                     ;
5                                     ; Inputs:
6                                     ; R2 = Pointer to window control block
7                                     ;
8 014700 010146  FREWSB: MOV      R1,-(SP)
9 014702 010446      MOV      R4,-(SP)
10 014704 010546     MOV      R5,-(SP)
11 014706 010605     MOV      SP,R5          ;Save initial stack pointer
12                                     ;
13                                     ; See if this window has an associated region
14                                     ;
15 014710 016204 000000G  MOV      DW$RID(R2),R4  ;Does window have an assoc region?
16 014714 001436      BEQ      9$          ;Br if not
17                                     ;
18                                     ; Decrement attachment-count in global region RCB and
19                                     ; reset exclusive-use flag.
20                                     ;
21 014716 105364 000000G  DECB     RC$CNT(R4)      ;Say one less job using region
22 014722 042764 000000G 000000G  BIC      #RC$EXC,RC$FLG(R4);Clear exclusive-use flag
23                                     ;
24                                     ; Build region definition block on the stack
25                                     ;
26 014730 162706 000012  SUB      #10.,SP          ;Reserve room for region def block
27 014734 010604      MOV      SP,R4          ;Get addr of base of region def block
28 014736 004737 014570'  CALL     MAKRDB          ;Build the region definition block
29 014742 012764 000000G 000000G  MOV      #RS.GBL,R.GSTS(R4);Set global status flag in RDB
30                                     ;
31                                     ; Build EMT argument block on stack to attach to region
32                                     ;
33 014750 162706 000004  SUB      #4.,SP          ;Reserve room for EMT arg block
34 014754 010601      MOV      SP,R1          ;Save pointer to EMT arg block
35 014756 010600      MOV      SP,R0          ;Get pointer to arg block area
36 014760 012710 017000  MOV      #<<36*400>+0>,(R0);Set function code
37 014764 010460 000002  MOV      R4,2(R0)        ;Set address of region def block
38 014770 104375      EMT      375          ;Try to attach to the region
39 014772 103407      BCS     9$          ;Br if cannot attach to region
40                                     ;
41                                     ; Eliminate the region
42                                     ;
43 014774 012764 000000C 000000G  MOV      #<RS.GBL!RS.EGR>,R.GSTS(R4);Set status flags
44 015002 010100      MOV      R1,R0          ;Point to EMT argument block
45 015004 012710 017001  MOV      #<<36*400>+1>,(R0);Set EMT function code
46 015010 104375      EMT      375          ;Eliminate the region
47                                     ;
48                                     ; Finished
49                                     ;
50 015012 010506 9$:  MOV      R5,SP          ;Restore stack pointer
51 015014 012605      MOV      (SP)+,R5
52 015016 012604      MOV      (SP)+,R4
53 015020 012601      MOV      (SP)+,R1
54 015022 000207      RETURN

```

WININI -- TSWIN initialization

```

1                                     .SBTTL  WININI -- TSWIN initialization
2                                     -----
3                                     ; WININI is called during system initialization to initialize the
4                                     ; display management system.
5                                     ;
6 015024 010546 WININI: MOV      R5, -(SP)
7                                     ;
8                                     ; Allocate space for the window control blocks
9                                     ;
10 015026 013705 000000G          MOV      VMXWIN, R5          ;Get # control blocks wanted
11 015032 070527 000000G          MUL      #DW$$SZ, R5          ;Times size of each block
12 015036 062705 015234'          ADD      #WINTOP, R5          ;Add base of area
13 015042 020527 000000C          CMP      R5, #140000-DW$$SZ ;Would this overflow available area?
14 015046 101402                  BLOS    1$              ;Br if not
15 015050 012705 000000C          MOV      #140000-DW$$SZ, R5 ;Constrain to available area
16 015054 010537 000004' 1$: MOV      R5, DWEND          ;Set pointer to end of control blocks
17                                     ;
18                                     ; Zero the entire display control block area
19                                     ;
20 015060 005045                  2$: CLR      -(R5)          ;Zero entire area
21 015062 020537 000002'          CMP      R5, DWBAS          ;Reached start of area?
22 015066 101374                  BHI     2$              ;Loop if not
23                                     ;
24                                     ; Finished
25                                     ;
26 015070 012605                  MOV      (SP)+, R5
27 015072 000207                  RETURN

```

WINPRT -- Window print-screen function

```

1          .SBTTL  WINPRT -- Window print-screen function
2          ;-----
3          ; WINPRT is called when a control character is received that means
4          ; that we should perform a print-screen function for the current
5          ; window for the job.
6          ;
7          ; Inputs:
8          ;   R2 = Pointer to current window control block for job
9          ;
10         ; Outputs:
11         ;   C-flag set on return if unable to print the window.
12         ;
13 015074 010146 WINPRT: MOV     R1,-(SP)
14 015076 010446         MOV     R4,-(SP)
15 015100 010546         MOV     R5,-(SP)
16         ;
17         ; See if "WINPRT" detached job is running now
18         ;
19 015102 012701 000000G         MOV     #LSTSL,R1      ;Get index to last job
20 015106 026137 000000G 000010' 1$:  CMP     LPRG1(R1),R5OWIN; Is this the WINPRT program?
21 015114 001004         BNE     2$          ;Br if not
22 015116 026137 000000G 000012'     CMP     LPRG2(R1),R5OPRT
23 015124 001404         BEQ     3$          ;Br if yes
24 015126 162701 000002     2$:  SUB     #2,R1      ;More jobs to check?
25 015132 003365         BGT     1$          ;Loop if yes
26 015134 000432         BR      8$          ;WINPRT program is not running
27         ;
28         ; The WINPRT program is running.
29         ; Schedule a completion routine to tell it to print a window.
30         ;
31 015136 016104 000000G 3$:  MOV     LBRKCQ(R1),R4  ;Get address of its completion Q element
32 015142 001427         BEQ     8$          ;Br if no completion routine pending
33 015144 005737 000000G         TST     NMFREQ      ;Can we get a free queue element?
34 015150 003424         BLE     8$          ;Br if not
35 015152 004737 000000G         CALL    GETRTQ     ;Get a free completion Q element (R1=addr)
36 015156 012700 000000C         MOV     #IQGSIZ/2,R0  ;Get # words in queue element
37 015162 010105         MOV     R1,R5      ;Get pointer to new queue element
38 015164 012425 4$:  MOV     (R4)+,(R5)+  ;Copy original completion Q element
39 015166 077002         SOB     R0,4$
40 015170 116200 000000G         MOVB   DW$JOB(R2),R0  ;Get # of job whose window to print
41 015174 006200         ASR     R0          ;Convert to job number
42 015176 110061 000001G         MOVB   R0,CQ$R1+1(R1) ;Set job # in high byte of R1
43 015202 116261 000000G 000000G  MOVB   DW$ID(R2),CQ$R1(R1); Set window ID in low byte of R1
44 015210 010104         MOV     R1,R4      ;Get addr of compl Q element for QCOMPL
45 015212 004737 000000G         CALL    QCOMPL     ;Queue the completion request for WINPRT
46 015216 000241         CLC          ;Signal success on return
47 015220 000401         BR      9$
48         ;
49         ; Unable to print the window
50         ;
51 015222 000261 8$:  SEC          ;Signal failure on return
52         ;
53         ; Finished
54         ;
55 015224 012605 9$:  MOV     (SP)+,R5
56 015226 012604         MOV     (SP)+,R4
57 015230 012601         MOV     (SP)+,R1

```

58 015232 000207
59
60
61
62
63
64 015234
65
66 000001

RETURN

;

; Top of TSWIN code (Window control blocks are allocated from this
; point to the end of the overlay).

;

WINTOP:

;

.END

Errors detected: 0

*** Assembler statistics

Work file reads: 0

Work file writes: 0

Size of work file: 8284 Words (33 Pages)

Size of core pool: 18176 Words (71 Pages)

Operating system: RT-11

Elapsed time: 00:01:56.02

,LP: TSWIN=DK: TSWIN/C/N: SYM

LSLCHR	72-20	72-59#											
LSRCHR	72-28	72-63#											
LSTATE	1-28	65-22											
LSTSL	1-23	99-19											
LSW	1-29	1-43	51-18	52-18	83-18								
LSW11	1-22	35-26*	42-21*	43-15*	74-16	74-18*	74-27	74-29*					
LSW2	1-25	3-30											
LSW3	1-27	67-24*											
LSW4	1-27	66-15*	66-25*										
LSW6	1-28	1-43	65-18*	66-14*									
LSW7	1-42	84-73	84-75										
LTRMTP	1-40	42-26	43-36	74-34	74-36	84-65	91-18	91-25					
LWINDO	1-30	3-21	66-19	85-16*	85-28*	87-10	87-22	93-13	93-15*				
MAKRDB	95-33	96-10#	97-28										
MAKWSB	84-111	95-14#											
MAPHLD	2-51#	4-44*	4-47	53-15*	53-18	54-26*	54-29	55-18*	55-21	59-22*	59-25	60-39*	
	60-42	61-21*	61-23	62-23*	62-25	68-45*	68-48	89-71*	89-73				
MAXSFC	83-26	83-45#											
NMFREQ	1-23	99-33											
OVRHC	1-32	52-23	81-40										
P2#CXT	1-25	89-29											
PCSPND	1-29	52-23											
PR7	1-32	4-44	53-15	54-26	55-18	59-22	60-39	61-21	62-23	67-20	68-45	89-71	
PRIVC2	1-25	89-29											
PSW	1-32	4-44*	4-47*	53-15*	53-18*	54-26*	54-29*	55-18*	55-21*	59-22*	59-25*	60-39*	
	60-42*	61-21*	61-23*	62-23*	62-25*	67-20*	67-23*	68-45*	68-48*	89-71*	89-73*		
QCOMPL	1-24	99-45											
R. GID	1-35	95-53	96-15*										
R. GSIZ	1-36	95-34*	96-17*										
R. GSTS	1-36	95-35*	95-59*	96-16*	97-29*	97-43*							
R. NAME	1-36	96-24*	96-38*										
R50PRT	2-53#	99-22											
R50WIN	2-52#	96-24	99-20										
RC#BAS	1-35	95-55											
RC#BLK	1-28	95-54											
RC#CNT	1-28	95-69*	97-21*										
RC#EXC	1-26	95-68	97-22										
RC#FLG	1-26	95-68*	97-22*										
REFRSH	66-21	67-9#	85-32										
REGCHR	3-68	4-10#	5-52										
RS. CGR	1-36	95-35											
RS. EGR	1-37	97-43											
RS. GBL	1-37	95-35	97-29	97-43									
RS. PVT	1-36												
S#OTWT	1-28	65-22											
SCRCHK	49-15	50-15	51-11#										
SCRLDN	21-26	50-8#											
SCRLUP	19-26	49-8#											
SEMI	2-47#	9-55	13-22	70-45	70-52	70-59	70-66	73-28	74-47	77-48	80-29		
SET200	71-36	71-41	71-60#	72-41									
SET52	71-48	71-64#	72-50										
SETERR	1-37	83-21	83-29	84-34	84-115	85-24	86-40	88-22	88-29	89-32	89-40	89-55	
SETHOM	37-16	37-26	43-28	47-29	48-10#								
SETLAB	39-13	39-21	39-28	39-35	59-31	60-25	61-9#						
SETLIN	17-30	18-33	19-20	21-20	24-27	25-39	48-20	56-39	57-34	57-40	59-16	59-36	
	60-19	60-23	60-30	60-33	60-48	63-14#	67-47	67-56	84-127				

SNDLIN	67-48	68-13#				
SS3CHR	1-31					
TC100	12-16	12-84	42-9#			
TCAKM	12-70	38-8#				
TCBS	5-34	14-8#				
TCCADR	12-15	12-43	24-9#			
TCCPRS	12-69	25-30#				
TCCPSV	12-68	25-8#				
TCCR	5-39	16-8#	20-9			
TCDELL	12-48	27-10#				
TCDOWN	12-38	18-10#				
TCESCE	12-50	20-9#				
TCG0AS	12-42	12-55	28-8#			
TCG0DS	12-41	12-73	28-18#			
TCG0GR	12-56	28-38#				
TCG0UK	12-54	28-28#				
TCG1AS	12-57	12-58	29-8#			
TCG1DS	12-76	29-18#				
TCG1GR	12-59	29-38#				
TCG1UK	29-28#					
TCG2AS	12-60	30-8#				
TCG2DS	12-79	30-18#				
TCG2GR	12-61	30-28#				
TCG3AS	12-62	31-8#				
TCG3DS	12-82	31-18#				
TCG3GR	12-63	31-28#				
TCHT	5-35	15-8#				
TCINSL	12-47	26-11#				
TCIXUP	12-44	12-51	21-9#			
TCLEFT	12-40	23-9#				
TCLERS	12-46	45-9#				
TCLF	5-36	5-37	5-38	12-49	19-8#	20-10
TCLS1R	12-26	32-48#				
TCLS2	12-27	32-28#				
TCLS2R	12-29	32-58#				
TCLS3	12-28	32-38#				
TCLS3R	12-30	32-68#				
TCN3	12-64	39-10#				
TCN4	12-65	39-18#				
TCN5	12-66	39-26#				
TCN6	12-67	39-33#				
TCNKM	12-86	38-18#				
TCPERS	12-45	46-9#				
TCPRT	11-85	12-21	44-9#			
TCRAA	12-18	33-23#				
TCRDA	12-20	33-51#				
TCREST	12-24	12-25	43-8#			
TCRIT	12-39	22-9#				
TCRTA	33-25	33-53	35-10#			
TCSAA	12-17	33-9#				
TCSBSZ	1-30	10-16				
TCSCA	12-22	40-9#				
TCSDA	12-19	33-37#				
TCSI	5-41	32-18#				
TCSD	5-40	32-8#				
TCSS2	12-52	32-78#				

TCSS3	12-53	32-89#					
TCSSR	12-23	47-9#					
TCSTA	33-11	33-39	34-10#				
TCUP	12-37	17-10#					
TSWIN	1-6#						
VALADB	1-24	89-19					
VALADW	1-24	89-17					
VMXWIN	1-40	98-10					
VPAR6	1-32	61-17	62-19	63-29	89-70		
VT100	1-39	74-34					
VT2007	1-40	42-26	43-36	84-65	91-25		
VT2008	1-40	42-26	43-36	74-36	84-65	91-25	
VT52	1-40	91-18					
WCPCSI	6-27	7-31#					
WCPESC	6-20	7-9#					
WCSCHK	3-41	6-14#					
WFDEL	83-39	86-8#					
WFMAP	83-38	85-10#					
WFNEW	83-37	84-16#					
WFPRNT	83-42	88-13#					
WFREAD	83-43	89-12#					
WFRSUM	83-41	87-22#					
WFSPND	83-40	87-10#					
WFSTT	83-44	90-8#					
WINCHR	1-17	3-14#					
WINDEL	84-23	84-113	86-29	86-44	92-16	93-8#	
WINDSP	1-17	66-9#					
WININI	1-17	98-6#					
WINPRT	1-18	88-26	99-13#				
WINREL	1-17	86-19	92-8#				
WINSF	1-18	64-9#					
WINSPN	51-40	52-8#					
WINSRC	84-21	84-98	85-21	86-37	88-19	89-37	94-15#
WINST	1-18	65-9#					
WINSTT	84-61	90-15	91-9#				
WINTOP	1-47	2-49	98-12	99-64#			
WINVEC	83-33	83-37#	83-45				

BUFMAP	2-26#	4-44	53-15	54-26	55-18	59-22	60-39	61-21	62-23	68-45	89-71	
DISABL	2-6#	4-44	53-15	54-26	55-18	59-22	60-39	61-21	62-23	67-20	68-45	89-71
ENABL	2-10#	4-47	53-18	54-29	55-21	59-25	60-42	61-23	62-25	67-23	68-48	89-73
ESCSEQ	12-5#	12-15	12-16	12-17	12-18	12-19	12-20	12-21	12-22	12-23	12-24	12-25
	12-26	12-27	12-28	12-29	12-30	12-37	12-38	12-39	12-40	12-41	12-42	12-43
	12-44	12-45	12-46	12-47	12-48	12-49	12-50	12-51	12-52	12-53	12-54	12-55
	12-56	12-57	12-58	12-59	12-60	12-61	12-62	12-63	12-64	12-65	12-66	12-67
	12-68	12-69	12-70									
OCALL	2-16#	52-23	81-40									
SEND	2-39#	68-90	68-91	69-19	69-24	69-25	69-26	69-31	69-33	69-38	69-39	69-43
	69-45	69-46	69-53	69-55	69-56	69-62	69-64	69-66	70-39	70-45	70-46	70-52
	70-53	70-59	70-60	70-66	70-67	70-71	71-35	71-36	71-40	71-41	71-48	72-20
	72-28	72-37	72-41	72-50	73-28	73-31	73-38	73-39	73-40	74-20	74-21	74-22
	74-32	74-39	74-40	74-41	74-42	74-45	74-46	74-47	74-48	74-49	74-50	75-35
	75-37	76-22	76-23	77-45	77-48	77-59	77-72	78-16	78-19	78-21	78-28	78-29
	79-23	79-38	79-44	79-47	79-50	80-29	80-32	80-38	80-41	80-44	81-5	81-23
	81-24	82-27										
UNMAP	2-32#	4-47	53-18	54-29	55-21	59-25	60-42	61-23	62-25	68-48	89-73	