```
.MCALL .MODULE
.MODULE SPINT,VERSION=11,COMMENT=<SPOOL Data Flow Module>

; Copyright (c) 1998 by Mentec, Inc., Nashua, NH.
; All rights reserved
;
; This software is furnished under a license for use only on a
; single computer system and  may  be  copied  only  with  the
; inclusion of the above copyright notice.  This software,  or
; any other copies thereof, may not be provided  or  otherwise
; made available to any other person except for  use  on  such
; system and to one who agrees to these license  terms.  Title
; to  and  ownership of the software shall at all times remain
; in Mentec, Inc.
;
; The information in this document is subject to change without
; notice and should not be construed as a commitment by Digital
; Equipment Corporation, or Mentec, Inc.
;
; Digital and Mentec  assume  no  responsibility for the use or
; reliability of its software on equipment which is not supplied
; by Digital  or Mentec,  and listed  in the  Software  Product
; Description.
```

```
        .ENABL LC
        .NLIST BEX

; -------------------------------------------------------------
; This task is entered every n ticks to check IF the SP handler
; has its interrupt request bit (INTEN) set.  If so,  a fake in-
; terrupt is generated and a buffers worth of data is retrieved.
; The buffer is written to the spooler work file, to be copied
; to the output device by the SDWTDN task at a later time.
; -------------------------------------------------------------

        .SBTTL  Edit History

;  Author: GA
;
;  05-MAY-80    GA      SET SP SHOW didn't report correct free blocks.
;  25-AUG-83    GA      Updated for RT-11 V5.1
;  04-OCT-83    GA      Added support for sending DATE and TIME to SP.
;  07-NOV-83    GA      Modified to allow virtual overlaying.
;  26-JAN-87    RHH     Check for interrupt forwarding before TIME call
;  03-SEP-87    RHH     Modify for virtualness
;  20-Apr-88    RHH     Add true Multi-stream support
;  27-Jun-88    RHH     Final cleanup for V5.5
;  12-Dec-88    RHH     STA.KL - handle KILL while stream still open
;  07-Jan-89    RHH     Make streams into device queues
;  20-Apr-89    RHH     Fix "new output causes MMU fault when
;                         SET SP KILL in progress"
;
;  01-Nov-1990  JFW     bracket error messages with ;+/;ERROR/.../;-

        .SBTTL  Conditionals

;+
;COND
;       V$JOB   (0)             REL job
;                1              SAV (virtual) job
;
;       SP$MLS  (=V$JOB)        stream(s)
;               0               1 stream
;               1               multiple streams
;
;       SP$PPS  (=V$JOB)        PRO print screen function
;               0               no PRO print screen
;               1               PRO print screen
;
;       SP$DBG  (0)             no special debug code
;               1               special debug code
;-
        .IIF NDF V$JOB  V$JOB  =: 0     ; By default, build the REL version

        SP$PPS  =: V$JOB                ; Include Print-Screen support
                                        ;  only with Virtual Job version

        SP$MLS  =: V$JOB                ; Include Multi-Stream support
                                        ;  only with Virtual Job version

; The following conditionals are temporary, and serve as development aids
; only:

        .IIF NDF SP$DBG SP$DBG =: 0     ; Set for special debugging code
.IF NE SP$MLS
```

```
        .SBTTL  Description of Multistream Spooling
;+
; The Multi-stream version of SPOOL incorporates a number of data
; structures to maintain and track output from multiple channels
; and/or jobs to various output devices.  Principal structures are
; an Output Device Table, a pool of stream descriptors, a workfile
; that holds blocks destined for output, and a workfile map that
; contains information about each workfile block.  Each device
; table entry contains a status word, the physical device name,
; the output channel, and a pointer to a stream queue.  If a device
; is idle, its stream queue pointer is NULL.  New streams (print
; jobs) can be added to the end of a device's queue, while they
; are printed from the first stream in line.  When a stream is
; completely printed and marked as .CLOSEd, then the next stream
; in line becomes the current stream for that device.
;
;  Output Device Table         Stream Descriptors
; +---------------------+    +--------+   +--------+
; | SO0 | stream pointer|--->| stream |-->| stream |-->0
; +---------------------+    +--------+   +--------+
; | SO1 | stream pointer|--  +--------+
; +---------------------+  ->| stream |-->0
; | SO2 | stream pointer|-+  +--------+
; +---------------------+ |
;            .            +->0
;            .
;            .
;
; The workfile contains blocks yet to be output.  Each block in the
; workfile is represented by one byte in the workfile map, which is
; memory resident.  The low seven bits of the byte identify the stream
; descriptor.  The high bit is set when the corresponding block is
; a "block zero" block; it marks the start of a new file, and is used
; to find the next file in sequence when SET SP NEXT is issued.
; A workfile map byte whose value is zero represents an unused workfile
; block.
;
; A stream descriptor looks like this:
;
;       +---------------+---------------+
;       |               |     Status    |       ST.STA
;       +---------------+---------------+
;       |   Job No.     | Dev Tbl Entry |       ST.JOB, ST.DTE
;       +---------------+---------------+
;       |    Stream Identifier (CSWA)   |       ST.ID
;       +---------------+---------------+
;       |      First Workfile Block     |       ST.WFB
;       +---------------+---------------+
;       |  Last Workfile Block Written  |       ST.LWB
;       +---------------+---------------+
;       | Next Workfile Block for Output|       ST.NXO
;       +---------------+---------------+
;       |      Link to Next Stream      |       ST.NST
;       +---------------+---------------+
```

```
; The stream descriptors are allocated in a contiguous area, and are
; identified in the workfile map by number; for example, 1st, 2nd,
; 3rd, etc.  This "ID" is different than the CSWA (ST.ID) value,
; but allows an ID to be stored in just a few bits.  The ST.ID value
; is the address of the Channel Status Word associated with the
; originating I/O.  This is used to identify I/O requests that may
; be coming from the same or different jobs as belonging to a particular
; print stream.  Once a LOOKUP or ENTER has been performed to SP on
; a given channel, SPOOL records the CSWA in ST.ID.  Later, subsequent
; WRITE requests are inspected for a known CSWA, and the appropriate
; ordinal stream number is placed in the workfile map byte that
; corresponds to the workfile block where the data is stored.
;
; An Output Device Table entry looks like this:
;
;       +---------------+---------------+
;       |    Channel    |     Status    |       OT.CHA, OT.STA
;       +---------------+---------------+
;       |   Pointer to Current Stream   |       OT.CST
;       +---------------+---------------+
;       |     Output Device (RAD50)     |       OT.DEV
;       +---------------+---------------+
;
; The output processor is driven by printer interrupts.  When a
; particular printer device interrupts, it schedules a pass through
; SDWTDN.  The block just output is deallocated by clearing the work-
; file map byte.  A scan is made through the workfile map for the
; next occurance of the stream number associated with the device.
; If another byte with that value is found, the corresponding block
; is written to the printer.  If there are no more blocks, and the
; stream has not been .CLOSEd, that output channel is hung, because
; there is presumably more output to come from SP.  If the stream
; is marked CLOSED, then that stream descriptor is erased (deallocated),
; and the device's Current Stream Pointer is changed to the next stream
; in line, if one exists.
;
; SET SP KILL is processed the next time the printer interrupts.
; All streams associated with the device are deallocated, as are
; the workfile blocks.  SET SP NEXT is similar, except that the
; deallocation process ends when a block-zero marker is found in
; the workfile map.
;
.ENDC; NE SP$MLS
```

```
        .SBTTL  Macros and other definitions

        .MCALL  .ASSUME, .ABTIO, .BR, .PRINT, .EXIT, .GVAL
        .MCALL  .READW, .SPFUN, .PVAL, .PEEK, .POKE
        .MCALL  .WRITC, .WRITW

        .NLIST CND
.IF EQ V$JOB
        .MCALL  SOB
.ENDC; EQ V$JOB

.IF NE SP$MLS
        .ASSUME V$JOB NE 0              ; ensure multi-stream is V-job version
.ENDC; NE SP$MLS

        .MACRO  ......
        .ENDM
```

```
        .SBTTL  Definitions

T$VEC   =: 020                          ; IOT vector address.
PR7     =: 340                          ; Priority 7 mask.

CLO$SE  =: 1                            ; Special directory operations
DELE$T  =: 2
LOOK$U  =: 3
ENT$ER  =: 4
PURGE$  =: 6

DATIME  =: 20000                        ; $SPSTA TIME request bit

LF      =: 12                           ; Linefeed character
FF      =: 14                           ; Formfeed character
CR      =: 15                           ; Carriage return character

SP.WCR  =: -6                           ; SP comm offsets
SP.DEV  =: -7
SP.JAF  =: -10
SP.CSA  =: -44                          ; Channel Status Address (ID)
SP.ABO  =: -46                          ; ABORT flag/job word

.IF NE SP$PPS                           ; PRO PRINT SCREEN support
```

```
; Debugging macro for displaying a string and a value

        .macro  .type1  thing,string,?a,?b
.IF NE SP$DBG
        .save
        .psect  .text.,d
a:      .ascii  string
        .ascii  / /
b:      .ascii  /     /
        .byte   0
        .even
        .restore
        mov     r5,-(SP)
        mov     r1,-(SP)
        mov     r0,-(SP)
        mov     thing,r5
        mov     #b,r1
        call    COMPUTE
        .print  #a
        mov     (SP)+,r0
        mov     (SP)+,r1
        mov     (SP)+,r5
.ENDC; NE SP$DBG
        .endm

        .macro  .dprint thing
.IF NE SP$DBG
        mov     R0,-(SP)
        .print  thing
        mov     (SP)+,R0
.ENDC; NE SP$DBG
        .endm
```

```
        .SBTTL  PRO PRINT_SCREEN Equates

COL80   =:      1                       ; Screen width attribute bit.
DBLTOP  =:      1                       ; Double high top half bit.
DBLBOT  =:      2                       ; Double high bottom half bit.
DBLWID  =:      4                       ; Double wide line.

BOLD    =:  40000                       ; Bold bit in attribute word.
UNDERS  =:  20000                       ; Underscore bit in attr word.
LINE    =:   1000                       ; Line drawing set to be used bit.

G1      =:     16                       ; Select G1.
G0      =:     17                       ; Select G0.
ESC     =:     33                       ; escape character.

PIID    =:    376                       ; special stream ID for Print_screen
ALID    =:    377                       ; special stream ID for WF allocation

        .NLIST
        .INCLUDE "SRC:PISET.MAC"
        .LIST

        PISFUN                          ; Define PI functions and arguments

        SETTYP  ,OFFSET,YES             ; Set expansion mode for following
        SETDAT                          ; Define keyboard/video tables

        HANUSE  =: 1                    ; Read from HANdler to USEr
        .TITLE  SPINT - SPOOL Data Flow Module
.ENDC; NE SP$PPS
```

```
        .SBTTL  Impure Data
.IF NE V$JOB
        .PSECT  .IMPD.,D
.IFF; NE V$JOB
        .PSECT  .IMPD.,D,GBL,REL,CON
.ENDC; NE V$JOB

PRMBLK: .WORD   1,0                     ; TIME call parameter block.
NUMDVW::.WORD   NUMDEV                  ; Number of SP devices

.IF NE SP$MLS
 .IF NE SP$DBG
fulcnt: .word   0
 .ENDC; NE SP$DBG
NXTWFB: .WORD   0                       ; Next available WORKFILE block
STRNUM: .WORD   0                       ; Current stream number
ODVNDX: .WORD   0                       ; Index to device table entry
STRID:  .WORD   0                       ; Stream ID
FUNCP:  .WORD   0                       ; function pointer
RBLOCK: .WORD   0                       ; workfile read block number
WBLOCK: .WORD   1                       ; printer block number
 .IF NE SP$PPS
ENTFAK: .WORD   ENT$ER                  ; Pro PRINT_SCREEN pseudo-stream
IOFAK:  .WORD   0                       ;  variables
CLOFAK: .WORD   CLO$SE
 .ENDC; NE SP$PPS

LAST1:  .WORD   0                       ; variables for SHOW output
NOBLKS: .WORD   0
EMPTYS: .WORD   0
R50NAM: .BLKW   2
.IFF; NE SP$MLS
FFDONE: .BYTE   0
        .EVEN
.ENDC; NE SP$MLS

.IF NE SP$PPS
PCOUNT: .WORD   -1,-1                   ; Print screen activity count.
..GENP::.WORD   0                       ; Generic Printer Mode
ATFLAG: .WORD   -1                      ; Character attributes (force change).
.ENDC; NE SP$PPS

DEVNUM: .WORD   0                       ; Working storage for device number.
```

```
        .SBTTL  SPINT  - Code
        .PSECT  .CODE.,I

.IF EQ SP$MLS
; ----- FB.BUF - Flag (not flagpage) Buffer.  Convenient ImPure Data
;                situated in Pure CODE psect.

FB.BUF  == .                            ; Start of flag buffer.
FB.BLK:.WORD    0                       ; Flagged block number.
FB.WCT:.WORD    0                       ; Flagged word count.
FB.SIZ  = .-FB.BUF
.ENDC; EQ SP$MLS

; ----- Code begins: Check for an interrupt request ------------------

        .ENABL  LSB

CK.INT::                                ; Enter here every n ticks.
.IF NE SP$PPS
        INC     PCOUNT                  ; Count number of ticks since last
        BNE     10$                     ;  print screen request.
        INC     PCOUNT+2
.ENDC; NE SP$PPS

; ----- Determine if there is any action to perform ------------------

10$:    CALL    GETCSR

        MOV     @R3,R0                  ; get the $SPSTA word
        BIC     #^C<SP$MSK>,R0          ; isolate the special op mask bits
        CMP     R0,#EX$IT               ; is this SET SP EXIT combination?
        BNE     15$                     ; branch if not.
        INCB    CCAFLG                  ; set CTRL/C flag, let SPTASK abort.
15$:
.IF NE SP$MLS
; ----- Is this an ABORT function?

        CMP     R0,#<AB$ORT>            ; is this the ABORT combination?
        BNE     20$                     ; branch if not.
        JMP     ABSPOL                  ; if so, .CLOSE all job's streams
        ......

20$:
.ENDC; NE SP$MLS
        BIT     #NEXT!ON!OFF!SHOW!KILL,@R3      ; Is a SET command requested?
        BEQ     30$                     ; No, then carry on.
        JMP     SETSP                   ; Yes, then go process the command.
        ......

30$:    BIT     #PRTSCR,@R3             ; Is PRINT SCREEN requested?
        BEQ     SPLIO                   ; No, then carry on.
        JMP     PSCREEN                 ; Yes, then go print the screen.
        ......

        .DSABL  LSB
```

```
        .SBTTL  Text Strings
        .PSECT  .TEXT.,D

;+
;ERROR
.IF NE SP$MLS
BADSTR: .ASCIZ  /?SPOOL-W-Output to unrecognized channel/
;BADSTR:        .ASCII  /?SPOOL-W-Invalid stream, op=/<200>
;BADST0:        .ASCII  /       id=/<200>
;BADST1:        .ASCIZ  /      /

DVWERR: .ASCIZ  /?SPOOL-F-Output error /<200>
.ENDC; NE SP$MLS
;-

        .EVEN
CRFF:   .ASCII  <CR><FF>
SHOHED: .ASCIZ  / Unit  Device  Status/
;                 0    LS0     ACTIVE - nnnnn blocks spooled
;                SP1   LR0     IDLE
SHO.LN: .ASCII  / SP/
SHO.UN: .ASCII  / :  /
SHO.DN: .ASCII  /ddn   /
SHO.NB: .ASCII  / nnnnn blocks spooled /<0>
LEFT:   .ASCIZ  / 00000 Free blocks in workfile/

SHO.ID: .ASCII  /IDLE/
CRLF:   .BYTE   0
SHO.HO: .ASCII  /HOLD,  /<200>
SHO.AC: .ASCII  /ACTIVE,/<200>

;+
;ERROR
.IF NE SP$PPS
BADSPF: .ASCIZ  "?SPOOL-F-Input error on PI during PRINT SCREEN"
.ENDC; NE SP$PPS
IOERR:  .ASCIZ  "?SPOOL-F-I/O error on work file"
BADDEV: .ASCIZ  "?SPOOL-W-Invalid SP unit"
ABOMSG: .ASCIZ  /?SPOOL-I-Terminated/
;-
        .EVEN

; ----- DEBUGGING messages ---------------------------------------

.IF NE SP$DBG
        .SBTTL  Debugging Messages
abrtng: .asciz  /ABORT code.../
changg: .asciz  /  Changing to PENDING STREAM!/
devdon: .asciz  /  Device done/
clsing: .asciz  /CLOSEing stream/
erase:  .asciz  /  Erasing stream entry/
equal:  .asciz  /  ST.NXO = ST.LWB/
full:   .asciz  /WF full.../
msg71:  .asciz  /(Ignoring MLSENT call)/
SDWENT: .asciz  /  SDWENT entered/
penerr: .asciz  /  OT.PEN went negative/
nostre: .asciz  /  NO PENDING STREAM?/
wrongd: .asciz  /WRONG DEVICE/
blk0:   .asciz  /Storing BLOCK-0 bit/
        .even
.ENDC; NE SP$DBG
```

```
; ----- Continue here if NOT PRO PRINT_SCREEN and NOT SET/SHOW command

        .ENABL  LSB

SPLIO:
.IF NE V$JOB
        CALL    GETCSR                  ; Get current $SPSTA value
.ENDC; NE V$JOB
        BIT     #INTEN,@R3              ; Is it an interrupt request?
        BEQ     RETIM1                  ; No, then go start another mark time.

; ----- I/O Request

.IF EQ V$JOB
        MOV     @#SP$VEC,R3             ; R3 -> SPINT:
        MOVB    SP.DEV(R3),R1           ; R1 = spooled device number.
.IFF; EQ V$JOB
                              .ASSUME SP.JAF EQ SP.DEV-1
        MOV     #SP$HTM+2,R5            ; Point to static comm area in SPX
 .IF NE SP$MLS
        MOV     SP.CSA(R5),STRID        ; Get stream ID (SP$CSA) from SPX
        MOV     #<SP$HTM+2+SP.JAF>,FUNCP        ; Point to SP$JAF in SPX
 .ENDC; NE SP$MLS
        MOV     SP.JAF(R5),R1           ; get the JOB/UNIT/FUNCTION word
        SWAB    R1                      ; high byte holds UNIT number
.ENDC; EQ V$JOB
.IF NE SP$MLS

; Translate unit number to Output Device Table Entry Address.  That
; address is returned (from CHKDEV) in R4.

PIDO1:                                  ; (Join in here for PRO Print_screen)
.ENDC; NE SP$MLS
        CALL    CHKDEV                  ; Check/save device unit number
        BCC     10$                     ; proceed if good.

BAD:
.IF EQ V$JOB
        BIS     #ERROR,@SPCSR           ; Set error bit in the CSR.
.IFF; EQ V$JOB
        MOV     #ERROR,R0               ; Set error bit in the CSR.
        CALL    CSRBIS
.ENDC; EQ V$JOB
        CALL    INTGEN                  ; Generate a pseudo interrupt.
        BR      RETIM1                  ; Go start another mark time.
        ......

.IF NE SP$MLS
;+
;ERROR
BADST:  .PRINT  #BADSTR                 ; Invalid stream
;-
;       MOVB    @R5,R5                  ; get the operation code
;       MOV     #<BADST0>,R1            ; R1 => target string.
;       CALL    COMPUTE                 ; Compute the value in decimal.
;       .PRINT  #BADST0                 ; Print
;       MOV     STRID,R5                ; get the stream id
;       MOV     #<BADST1>,R1            ; R1 => target string.
;       CALL    COMPUTE                 ; Compute the value in decimal.
;       .PRINT  #BADST1                 ; Print
        BR      BAD
        ......
.ENDC; NE SP$MLS
```

```
; ----- Verify that there is adequate space in the work file

10$:
.IF NE SP$MLS
        MOV     R4,ODVNDX               ; save dev table index
        CLR     R1                      ; beginning of WFMAP
        CLR     R2                      ; want stream_ID = 0
        CALL    GNXWFB                  ; Get next avail WF block
 .IF NE SP$DBG
        BCS     full                    ; return empty-handed.
 .IFF; NE SP$DBG
        BCS     RETIM1                  ; return empty-handed.
 .ENDC; NE SP$DBG
        MOV     R0,R5                   ; next available WF block number
        ADD     WFMAPA,R5               ; point to WFMAP entry
        MOVB    #ALID,@R5               ; allocate it with pseudo-stream no.
        MOV     R0,R1                   ; try again
        MOV     R0,-(SP)                ; save the entry
        CALL    GNXWFB                  ; ensure at least two blks avail.
        MOV     (SP)+,R0                ;*C* restore the entry
        BICB    @R5,@R5                 ;*C* deallocate the block
        BCC     20$                     ; was there a 2nd WF blk available?
 .IF NE SP$DBG
full:   inc     fulcnt
        .type1  fulcnt,<'WF full #'>
 .ENDC; NE SP$DBG
.IFF; NE SP$MLS
        JSR     R5,SIZEIT               ; Make sure there is enough room.
        .WORD   2                       ; 2 blocks required.
        BCC     20$                     ; continue if there is room
.ENDC; NE SP$MLS
RETIM1: JMP     RETIME                  ; No, come back when not so full.
        ......
```

```
20$:
.IF NE SP$MLS
        MOV     R0,NXTWFB               ; Save the next avail WFB

; Check function for .ENTER and .LOOKUP...
; If so, initialize a new I/O stream.  If not, treat as .WRITE

                                .ASSUME V$JOB NE 0
        MOV     FUNCP,R5                ; Point to function code
.IF NE SP$DBG
        MOVB    @R5,R0
        .type1  R0,<'*** USR Function '>
.ENDC; NE SP$DBG
        MOV     STRID,R0                ; Get stream ID
        CMPB    @R5,#ENT$ER             ; is it .ENTER?
        BEQ     30$                     ; (R4 -> Device table entry)
        CMPB    @R5,#LOOK$U             ; or .LOOKUP?
        BNE     50$                     ; branch if not.
30$:    CALL    MLSENT                  ; Initialize stream
        BCC     40$                     ; Branch on success
;+
;ERROR
        .PRINT  #BADDEV                 ; ?SPOOL-E-Invalid SP device
;-
        BR      BAD
        ......

; New or appended stream is established.  If this is the first call
; to SPINT from this stream, initialize ST.WFB and ST.NXO with the
; Next_WF_block_available, and then go check to see whether SP needs
; the formatted TIME string.

40$:    CALL    GSTNUM                  ; Get the stream number
        .type1  r2,<'NEW stream #'>
;                               .ASSUME ST.STA EQ 0
        BITB    #<STA.AP!STA.E2>,@R1    ; Stream already established?
        BNE     100$                    ; Branch if .ENTER stuff already done
        MOV     R2,STRNUM               ; save the new stream number
        MOV     NXTWFB,R0               ; next avail WF block
        MOV     R0,ST.WFB(R1)           ; save starting workfile block
        MOV     R0,ST.NXO(R1)           ;  as next to output,
        BR      OKIO                    ; Proceed if new stream
        ......

; Request is NOT .ENTER or .LOOKUP.  Find the stream and its associated
; device.  If the stream does not already exist, then something's wrong.

50$:    MOV     OT.CST(R4),R1           ; Point to potential stream
60$:    BEQ     BADST                   ; no such stream?
        CMP     STRID,ST.ID(R1)         ; is this the one?
        BNE     70$                     ; branch if not.
;                               .ASSUME ST.STA EQ 0
        BITB    #STA.CL,@R1             ; ID is the same.  stream closed?
        BEQ     80$                     ; If it's open, this is it.
70$:    MOV     ST.NST(R1),R1           ; traverse to next element
        BR      60$

80$:    CALL    GSTNUM                  ; get stream number
```

```
; We have a stream, an operation, and a device.
;
;       R1 -> stream entry
;       R2 =  stream number
;       R4 -> device table entry

;                               .ASSUME ST.STA EQ 0
90$:    BICB    #STA.EP,@R1             ; Clear .ENTER_IN_PROGRESS bit
100$:   MOV     R2,STRNUM               ; save the stream number

; Search the workfile map for TWO available workfile blocks outside the
; range ST.NXO -> ST.LWB

        MOV     WFMAPA,R0               ; point to WF Map
        MOV     ST.NXO(R1),R3           ; get Next Block to Output
        ADD     R0,R3                   ; point to its entry for CMP below.
        ADD     ST.LWB(R1),R0           ; point to Last Block Written entry
        INC     R0                      ; start just past it
        CLR     R2                      ; initialize count

110$:   CMP     R0,WFMAPL               ; time to wrap?
        BLO     120$
        MOV     WFMAPA,R0               ; ok, wrap.
120$:   CMP     R0,R3                   ; did we reach the fence?
        BEQ     RETIM1                  ; if so, we didn't find enough blocks
        TSTB    (R0)+                   ; is the WF block free? (advance ptr)
        BNE     110$                    ; loop if not

        TST     R2                      ; If so, is this the first?
        BNE     130$                    ; branch if not the first
        MOV     R0,-(SP)
        SUB     WFMAPA,R0               ; derive block number equivalent
        DEC     R0                      ; undo (R0)+ above
        MOV     R0,NXTWFB               ; save the pointer,
        MOV     (SP)+,R0
130$:   INC     R2                      ; count an unused block
        CMP     R2,#2                   ; enough free blocks found yet?
        BLO     110$

; There are indeed at least two blocks in the safe zone.  Check to see
; if the request is to .CLOSE or .PURGE the SP unit.

        CMPB    @R5,#CLO$SE             ; is it a .CLOSE?
        BEQ     140$                    ; branch if so.
        CMPB    @R5,#PURGE$             ; or .PURGE?
        BNE     OKIO
;                               .ASSUME ST.STA EQ 0
140$:   .dprint #clsing
        BISB    #STA.CL,@R1             ; set CLOSED bit in stream entry
        BICB    #STA.KL,@R1             ; clear KILLED bit too.
        BITB    #STA.E2,@R1             ; was it ever written to?
        BNE     OKIO                    ; proceed if so.
        CALLR   STRDON                  ; otherwise, erase stream
        ......
OKIO:
.ENDC; NE SP$MLS
        .DSABL  LSB
```

```
        .ENABL  LSB

; ----- Check to see if SP needs the formatted TIME string ----------

.IF EQ V$JOB
        BIT     #DATIME,@SPCSR          ; Is the date and time requested?
        BEQ     10$                     ; No, then go process the interrupt.
        BIC     #DATIME,@SPCSR          ; Clear the date/time request.
        CALL    DT.SEND                 ; Go send the date and time.
.IFF; EQ V$JOB
        CALL    GETCSR                  ; Get current copy of $SPSTA
        BIT     #DATIME,@SPCSR          ; Is the TIME requested?
        BEQ     10$

        CALL    DT.SEND                 ; Go send formatted time to SP
        MOV     #DATIME,R0
        CALL    CSRBIC                  ; clear the TIME request bit

        CALL    GETCSR                  ; Get current copy of $SPSTA
.ENDC; EQ V$JOB
        BIT     #INTEN,@SPCSR           ; Are interrupts still enabled?
        BEQ     RETIME                  ; No, then go retime us.

10$:
.IF NE <SP$MLS & SP$PPS>
        CMP     STRID,#PIID             ; is this a PRINT_SCREEN call?
        BEQ     20$                     ; if so, skip interrupt request
.ENDC; NE <SP$MLS & SP$PPS>

; ----- All appears OK, so get a buffer's worth of data from SP -----

.IF EQ V$JOB
        MOV     @#SP$VEC,R3             ; R3 -> SPINT:
        MOV     #SPFBUF,SP.BPR(R3)      ; Yes, tell SP where to stuff data.
.ENDC; EQ V$JOB
        CALL    INTGEN                  ; Go call SP for data

; ----- Process the I/O request ---------------------------------

20$:
.IF EQ SP$MLS
 .IF EQ V$JOB
        TST     SP.BNR(R3)              ; Is this block 0 (ie. start of file)?
 .IFF; EQ V$JOB
        MOV     #SP$HTM+2,R3            ; Point to static COM pseudo SPINT
        MOV     SP.WCR(R3),R0           ; Get word count in R0,
        MOV     SP.BNR(R3),R1           ; TST SP.BNR(R3) - block zero?
 .ENDC; EQ V$JOB
        BEQ     30$                     ; Yes, then go write a flag block.

        BIT     #$BUSY,D.STAT(R4)       ; Is the device busy?
        BEQ     30$                     ; No, then go write a flag block.

 .IF EQ V$JOB
        CMP     SP.WCR(R3),#256.        ; Is this block completely filled?
        BEQ     40$                     ; If so, skip flag block this time.

30$:    MOV     SP.BNR(R3),FB.BLK       ; Set the flag block, block number.
        MOV     SP.WCR(R3),FB.WCT       ; Set the flag block, word count.
 .IFF; EQ V$JOB
        CMP     R0,#256.                ; Is this block completely filled?
        BEQ     40$                     ; If so, skip flag block this time.

30$:    MOV     R1,FB.BLK               ; Set the flag block, block number.
        MOV     R0,FB.WCT               ; Set the flag block, word count.
```

```
 .ENDC; EQ V$JOB

        MOV     #FB.BUF,R5              ; Indicate this write is a flag block.
        CALL    SFWRIT                  ; Write flag block to work file.
.IFF; EQ SP$MLS
        MOV     #SP$HTM+2,R3            ; Point to static COM pseudo SPINT
        MOV     SP.WCR(R3),R0           ; Get word count in R0,
        BEQ     50$                     ; if none, skip write.
        BITB    #STA.KL,@R1             ; stream has been killed?
        BNE     50$                     ; (this can happen if output is
                                        ;  coming from FG or SYS job)
.ENDC; EQ SP$MLS

; Write data block to workfile

40$:    MOV     SPFBFA,R5               ; Indicate this write is a data block.
        CALL    SFWRIT                  ; Write it to the work file.
.IF EQ SP$MLS
        BIT     #$BUSY,D.STAT(R4)       ; Is the device already busy?
        BNE     60$                     ; Yes, then lucky us.
        MOV     DEVNUM,R1               ; Set up a fake device done completion
        CALL    CR.DVD                  ; routine entry to start the device.
        BIS     #$BUSY,D.STAT(R4)       ; Flag it as busy now.
.IFF; EQ SP$MLS
        MOV     ODVNDX,R4               ; point to Output Device Table entry
;                               .ASSUME OT.STA EQ 0
        BITB    #OTPRM,@R4              ; Is the device already primed?
        BNE     50$                     ; Yes, then just let things flow...

; Initiate WRITE operations on the printer device by calling its
; completion routine.

        CALL    CMPADR                  ; Determine device's cmplt address
        CALL    @R2                     ; call it to schedule 1st write
;                               .ASSUME OT.STA EQ 0
        BISB    #OTPRM,@R4              ; Flag it as primed now.
50$:
 .IF NE SP$PPS
        CMP     STRID,#PIID             ; is this a PRINT_SCREEN?
        BNE     60$                     ; branch if not.
        RETURN
        ......
 .ENDC; NE SP$PPS
.ENDC; EQ SP$MLS

60$:    CALLR   CR.TIM                  ; Reschedule ourself and return.
        ......

; ----- Restart another mark-time for INTEN bit checking -----------

RETIME: CALLR   TIMERON                 ; Call the timer start routine.
        ......

        .DSABL  LSB
.IF NE SP$MLS
```

```
        .SBTTL  INTGEN - Call SP Handler

; -----------------------------------------------------------------
; Interrupt generator subprogram - Calls SP handler for data
;
; This routine puts a copy of the SP interrupt vector contents in the
; IOT vector.  It then executes an IOT instruction, which takes us
; instantly to SP's AST point to do some function there.
;
; R0 is spoiled.  R1-R5 are preserved.
; -----------------------------------------------------------------

        .ENABL  LSB

INTGEN:
.IF EQ V$JOB
        MOV     @#SP$VEC,@#T$VEC        ; Initialize IOT vector.
        MOV     #PR7,@#T$VEC+2          ; Set for priority 7.
.IFF; EQ V$JOB
20$:
        .PEEK   #AREA,#SP$VEC           ; Get SP's AST address

        MOV     R1,-(SP)
        MOV     R0,R1
        .POKE   #AREA,#T$VEC,R1         ; Stuff SP's AST adrs in IOT vector
        MOV     (SP)+,R1

        .POKE   #AREA,#T$VEC+2,#PR7

        .GVAL   #AREA,#$SPSTA           ; Get SPCSR
        MOV     R0,SPCSRV               ; put it somewhere local

.ENDC; EQ V$JOB
        BIT     #INTEN,@SPCSR           ; Is INTEN bit still set?
        BNE     30$                     ; Yes, go IOT.
        CMP     (SP)+,R0                ; No, then remove caller's RTN adrs,
.IF EQ SP$MLS
        BR      RETIME                  ; go reschedule ourself.
        ......
.IFF; EQ SP$MLS
        BR      RETIM2                  ; go reschedule ourself.
        ......
.ENDC; EQ SP$MLS

30$:    IOT                             ; Call SP handler routine
        RETURN                          ; then return to caller.
        ......

        .DSABL  LSB
```

```
        .SBTTL  ABSPOL - Abort streams for a job
;+
; The SP handler's abort entry was called because a job outputing
; to it was aborted.  For all open streams with that job number,
; simulate .CLOSE. (Don't stop their output - this isn't the same
; as a KILL.  It's per-job; not per-unit.)
;
; On entry, R3 points to a copy of the $SPSTA word.
;; On entry, R0 points to the high byte of SP$ABO, the ABORT flag
;; word.
;; The low three bits of that word contains the number of the job
;; being aborted.
;-
        .ENABL  LSB

ABSPOL:
        MOV     @R3,-(SP)               ; Get $SPSTA
 .IF EQ V$JOB
        BIC     #<AB$ORT+7>,@R3         ; clear the request bits in the CSR.
 .IFF; EQ V$JOB
        MOV     #<AB$ORT+7>,R0          ; clear the request bits in the CSR.
        CALL    CSRBIC
 .ENDC; EQ V$JOB
        MOV     (SP)+,R0                ; get saved copy of $SPSTA
        BIC     #177770,R0              ; isolate job number/2
        ASL     R0                      ; make real job number
        MOV     #STRTAB,R1              ; point to stream table
        MOV     #NUMSTR,R2              ; this many streams
        .dprint #abrtng
;                               .ASSUME ST.STA EQ 0
10$:    TSTB    @R1                     ; this stream in use?
        BEQ     20$                     ; skip if not.
        CMPB    ST.JOB(R1),R0           ; this stream from aborting job?
        BNE     20$                     ; skip if not.
.If NE SP$DBG
;                               .ASSUME ST.STA EQ 0
        bitb    #sta.cl,@r1             ; iS THE STREAM open?
        bne     20$                     ; BRANCH IF NOT.
        mov     r2,-(sp)
        neg     r2
        add     #8.,r2
        .type1  R2,<"ABORT CLOSING stream ">
        mov     (sp)+,r2
.EndC; NE SP$DBG
;                               .ASSUME ST.STA EQ 0
        BISB    #STA.CL,@R1             ; Force the stream CLOSED
20$:    ADD     #ST.ESZ,R1              ; point to next stream
        SOB     R2,10$                  ; loop through all streams

RETIM2: JMP     RETIME
        ......

        .DSABL  LSB
.ENDC; NE SP$MLS
```

```
        .SBTTL  SFWRIT - Write to work file

; ----- Subroutine to write to work file -------------------------
;
; On entry,
.IF NE SP$MLS
;               R1 points to stream table entry
.ENDC; NE SP$MLS
;               R5 points to workfile buffer

        .ENABL  LSB

SFWRIT:
.IF NE SP$MLS
        MOV     R2,-(SP)
        MOV     NXTWFB,R2               ; workfile block number
        .WRITW  #AREA,#SPCHAN,R5,#256.,R2
.IFF; NE SP$MLS
        .WRITW  #AREA,#SPCHAN,R5,#256.,B.TAIL(R4)
.ENDC; NE SP$MLS
        BCC     10$                     ; Write ok?
;+
;ERROR
        .PRINT  #IOERR                  ; No.  We're dead, folks.
;-
        .EXIT
        ......

10$:
.IF NE SP$MLS
; Update the appropriate workfile map entry to reflect the block
; just written.

        MOV     STRNUM,R0               ; what stream was it?
        INC     R0                      ; offset by 1 for WFMAP
;                               .ASSUME ST.STA EQ 0
        BITB    #STA.EP,@R1             ; doing .ENTER blocks?
        BEQ     20$                     ; branch if not.
;                               .ASSUME ST.STA EQ 0
        BITB    #STA.E2,@R1             ; block-0 code already done?
        BNE     20$
        BISB    #WM.BK0,R0              ; mark WFMAP entry with BLOCK-ZERO bit
        .dprint #blk0
;                               .ASSUME ST.STA EQ 0
        BISB    #STA.E2,@R1             ; acknowledge .ENTER initiated

20$:    MOV     R2,ST.LWB(R1)           ; save WF block just written
        ADD     WFMAPA,R2               ; point to WF map entry
        MOVB    R0,@R2                  ; put stream number in WFMAP
        MOV     (SP)+,R2                ; restore R2
.IFF; NE SP$MLS
        INC     B.USED(R4)              ; Count the block as being added.
        INC     B.TAIL(R4)              ; Update the tail pointer.
        CMP     B.TAIL(R4),B.END(R4)    ; Are we at the end?
        BLOS    30$
        MOV     B.BEG(R4),B.TAIL(R4)    ; Yes, then wrap around.
30$:
.ENDC; NE SP$MLS
        RETURN                          ; Return to the caller.
        ......

        .DSABL  LSB
```

```
        .SBTTL  DT.SEN - Send formatted TIME to SP

; ----- Send a formatted time string to SP ---------------------

        .ENABL  LSB

DT.SEN: MOV     R5,-(SP)                ; Save
        MOV     R4,-(SP)                ;   important
        MOV     R3,-(SP)                ;     registers.
        MOV     R2,-(SP)
        MOV     R1,-(SP)

.IF EQ V$JOB
        MOV     @#SP$VEC,R3             ; R3 -> SPINT:
        ADD     #SP.TIM,R3              ; R3 -> SP$TIM.
        MOV     R3,PRMBLK+2             ; PRMBLK -> SP$VEC.
.IFF; EQ V$JOB
        MOV     #SP$HTM,PRMBLK+2        ; PRMBLK -> High mem time
        CALL    GETCSR                  ; Update the local $SPSTA
.ENDC; EQ V$JOB
        BIT     #INTEN,@SPCSR           ; Interrupt still enabled?
        BEQ     10$                     ; No, then just exit.
        MOV     #PRMBLK,R5              ; R5 -> parameter block.
        CALL    TIME                    ; Call the SYSLIB time routine.

10$:    MOV     (SP)+,R1                ;    restore
        MOV     (SP)+,R2                ;      registers.
        MOV     (SP)+,R3
        MOV     (SP)+,R4
        MOV     (SP)+,R5
        RETURN                          ; Return to the caller.
        ......



        .SBTTL  CINDEX - Create an index into device table

; ----- Subroutine to create an index into DEVICE TABLE ----------------
;       based on UNIT passed in R1 (destroyed)
;       Index is returned in R4

CINDEX: CLR     R4                      ; R4 = device number
40$:    DEC     R1                      ; Does R4 contain proper offset?
        BLT     50$                     ; Yes, go add base address.
.IF EQ SP$MLS
        ADD     #SDCESZ,R4              ; No, increment offset by entry size.
        BR      40$

50$:    ADD     #SDCTAB,R4              ; Add the base address.
.IFF; EQ SP$MLS
        ADD     #OT.ESZ,R4
        BR      40$

50$:    ADD     #ODVTAB,R4
.ENDC; EQ SP$MLS
        RETURN                          ; Return to the caller.
        ......

        .DSABL  LSB
```

```
        .SBTTL  CHKDEV - Check device unit

; Check/save SP unit number in R1.  If invalid, print message.
; On success, R4 contains the device table entry address.

        .ENABL  LSB

CHKDEV: BIC     #177770,R1              ; Strip all but unit number bits.
        MOV     R1,DEVNUM               ; Save the device number.
        CMP     R1,#NUMDEV              ; Is the unit number legal?
        BHIS    10$                     ; branch if not.

        CALL    CINDEX                  ; Get index
.IF NE SP$MLS
        TST     OT.DEV(R4)              ; Is a device assigned to that unit?
        BEQ     10$                     ; branch if not.
.IFF; NE SP$MLS
        CLC
.ENDC; NE SP$MLS
        RETURN                          ; with C=0 (TST)
        ......

;+
;ERROR
10$:    .PRINT  #BADDEV                 ; No, report a bad device.
;-
        SEC                             ; Indicate error
        RETURN
        ......

.IF NE SP$MLS
        .SBTTL  GSTNUM - Get Stream Number

;+
;  Translate stream entry address in R1, to a stream number in R2.
;  R0 is destroyed.
;-

GSTNUM: MOV     R1,R0                   ; address of stream entry
        CLR     R2                      ; count streams here
20$:    SUB     #ST.ESZ,R0
        CMP     R0,#STRTAB
        BLO     30$
        INC     R2                      ; count the streams
        BR      20$
        ......

30$:    RETURN
        ......
.ENDC; NE SP$MLS

        .DSABL  LSB
```

```
        .SBTTL  CSRBIC, CSRBIS, GETCSR

        .ENABL  LSB
.IF NE V$JOB

; Do a BIC (or BIS) of what's in R0 in $SPSTA
; (Here is a wide open window - need a better way to do bic in kernel space)

CSRBIC::CALL    GETCSR                  ; Get current $SPSTA in (R3)
        BIC     R0,@R3                  ; Clear the bit(s)
        BR      CSRCOM                  ; join common code below

CSRBIS::CALL    GETCSR                  ; Get current $SPSTA in (R3)
        BIS     R0,@R3                  ; Set the bit(s)

CSRCOM::.PVAL   #AREA,#$SPSTA,@R3       ; store new value in $SPSTA
        RETURN
        ......

.ENDC; NE V$JOB

; ----- Get address of $SPSTA in R3 ---------------------------------

GETCSR::
.IF NE V$JOB
        MOV     R0,-(SP)
        .GVAL   #AREA,#$SPSTA           ; Get a copy of $SPSTA
        MOV     R0,SPCSRV               ; put it in local place
        MOV     #SPCSRV,SPCSR           ; put address of local place in SPCSR
        MOV     (SP)+,R0
.ENDC; NE V$JOB
        MOV     SPCSR,R3                ; Put address of $SPSTA in R3
        RETURN
        ......

        .DSABL  LSB
```

```
        .SBTTL  SETSP  - Process SET and SHOW commands

; ----- Processing for SP handler SET commands ---------------------

        .ENABL  LSB

SETSP:  MOV     #PRTSCR,R0              ; clear the bit,
        MOV     @R3,R1                  ; Get unit number for the SET.
.IF NE SP$MLS
        CLR     ALLUNI                  ; Clear ALL_UNITS flag
        BIT     R0,@R3                  ; is PRINT_SCREEN bit set?
        BEQ     10$                     ; branch if not
 .IF EQ V$JOB
        BIC     R0,@R3                  ; clear the bit,
 .IFF; EQ V$JOB
        CALL    CSRBIC                  ; clear the bit,
 .ENDC; EQ V$JOB
        INC     (PC)+                   ; Set ALL_UNITS flag
ALLUNI: .WORD   0
10$:
.ENDC; NE SP$MLS
.IF EQ V$JOB
        BIC     #7,@R3                  ; Clear the unit number in the CSR.
.IFF; EQ V$JOB
        MOV     #7,R0                   ; Clear the unit number in the CSR.
        CALL    CSRBIC
.ENDC; EQ V$JOB
        BIT     #SHOW,@R3               ; is it SHOW QUEUE?
        BEQ     20$                     ; Br if not.
        JMP     DOSHOW                  ; If so, go do that.
        ......

; Check unit number, get index to device table entry

20$:    CALL    CHKDEV                  ; Check/save unit number
        BCC     30$                     ; It's ok.  Proceed.
.IF EQ V$JOB
        BIC     #ACTION,@R3             ; Clear everybody but INTEN & ERROR.
.IFF; EQ V$JOB
        MOV     #ACTION,R0              ; Clear everybody but INTEN & ERROR.
        CALL    CSRBIC
.ENDC; EQ V$JOB
        JMP     RETIME                  ; Reschedule ourself.
        ......

; SET SP[n] NEXT?

30$:    MOV     #NEXT,R0
        BIT     R0,@R3                  ; Find out which SET is requested.
        BEQ     40$                     ; Not this one!
.IF EQ V$JOB
        BIC     R0,@R3                  ; clear the NEXT request bit
.IFF; EQ V$JOB
        CALL    CSRBIC                  ; clear the NEXT request bit
.ENDC; EQ V$JOB
.IF EQ SP$MLS
        BIT     #$BUSY,D.STAT(R4)       ; Is the device busy?
        BEQ     RETIM3                  ; No.  Don't request next.
        BIS     #$NEXT,D.STAT(R4)       ; Yes, set device specific next flag.
.IFF; EQ SP$MLS
;                               .ASSUME OT.STA EQ 0
        BITB    #OTBSY,@R4              ; Is the device busy?
        BEQ     RETIM3                  ; No.  Don't request next.
;                               .ASSUME OT.STA EQ 0
        BISB    #OTNXT,@R4              ; Yes, set device specific next flag.
```

```
.ENDC; EQ SP$MLS
        BR      RETIM3                  ; Go set for a timed reschedule of us.
        ......
```

```
        .DSABL  LSB
.IF NE SP$MLS
```

```
; SET SP[n] HOLD?

40$:    MOV     #OFF,R0                 ; look at OFF bit
.IF NE SP$MLS
        MOV     #1,R1                   ; set HOLD BIT
.ENDC; NE SP$MLS
        BIT     R0,@R3                  ; was HOLD specified?
.IF NE SP$MLS
        BNE     70$                     ; go process HOLD
.IFF; NE SP$MLS
        BEQ     50$                     ; branch if not.
 .IF EQ V$JOB
        BIC     R0,@R3                  ; reset the HOLD bit
 .IFF; EQ V$JOB
        CALL    CSRBIC                  ; reset the HOLD bit
 .ENDC; EQ V$JOB
        BIS     #$HOLD,D.STAT(R4)       ; set the device's HOLD bit
        BR      RETIM3
        ......
.ENDC; NE SP$MLS

; SET SP[n] NOHOLD?

50$:    MOV     #ON,R0                  ; look at ON bit
.IF NE SP$MLS
        MOV     #2,R1                   ; indicate CLEAR HOLD
.ENDC; NE SP$MLS
        BIT     R0,@R3                  ; was NOHOLD specified?
.IF NE SP$MLS
        BNE     70$                     ; go process HOLD
.IFF; NE SP$MLS
        BEQ     60$                     ; branch if not.
 .IF EQ V$JOB
        BIC     R0,@R3                  ; reset the NOHOLD bit
 .IFF; EQ V$JOB
        CALL    CSRBIC                  ; reset the NOHOLD bit
 .ENDC; EQ V$JOB
        BIC     #$HOLD,D.STAT(R4)       ; CLEAR the device's HOLD bit
        BR      RETIM3
        ......
.ENDC; NE SP$MLS

; SET SP[n] KILL?

60$:    MOV     #KILL,R0
.IF NE SP$MLS
        MOV     #3,R1                   ; indicate KILL
.ENDC; NE SP$MLS
        BIT     R0,@R3                  ; SET SPn KILL?
        BEQ     DOSHOW                  ; branch if not
.IF EQ V$JOB
70$:    BIC     R0,@R3                  ; clear the bit
.IFF; EQ V$JOB
70$:    CALL    CSRBIC                  ; clear the bit
.ENDC; EQ V$JOB

.IF EQ SP$MLS
        BIT     #$BUSY,D.STAT(R4)
        BEQ     RETIM3
        BIS     #<$KILL!$NEXT>,D.STAT(R4)
.IFF; EQ SP$MLS
        CALL    MODOTV                  ; go set KILL bit(s)
.ENDC; EQ SP$MLS
RETIM3: JMP     RETIME
        ......
```

```
        .SBTTL  MODOTV - SET support routines
;+
; These routines are used by SET routines above to apply
; conditions and attributes to multiple device units where
; applicable.  For example, SET SP KILL is distinguished
; from SET SP0 KILL here.
;
; MODOTV is MODIFY OUTPUT DEVICE TABLE.
; It is called with the following register contents:
;
;       R1 = 1 to SET HOLD bit in device table entry(s)
;       R1 = 2 to CLEAR HOLD bit in device table entry(s)
;       R1 = 3 to SET KILL bit in device table entry(s)
;
;       R4 points to the output device table entry for the device
;
;       R0, R1, and R4 are destroyed
;-
        .ENABL  LSB

MODOTV: TST     ALLUNI                  ; ALL units?
        BNE     10$                     ; branch down if so.
;                               .ASSUME OT.STA EQ 0
;       ADD     #OT.STA,R4              ; One unit only. Point to status
        CALLR   DOBIT                   ; set/clear the bit.
        ......

10$:    MOV     #ODVTAB+OT.STA,R4       ; point to 1st device's status byte
        MOV     #NUMDEV,R0              ; for each device,
20$:    CALL    DOBIT                   ; set/clear the bit
        ADD     #OT.ESZ,R4              ; point to next device's entry
        SOB     R0,20$                  ; count down until done.
        RETURN
        ......

DOBIT:  DEC     R1                      ; set OFF bit?
        BNE     30$
        BISB    #OTOFF,@R4              ; set each unit's HOLD bit
        RETURN
        ......

30$:    DEC     R1                      ; set ON bit?
        BNE     40$
        BICB    #OTOFF,@R4              ; CLEAR unit's HOLD bit
        RETURN
        ......
```

```
;+
; SET SPn KILL
;
; For the specified device "n", find the current streams and set their
; KILLED bit.  This prevents writing into the workfile of any blocks output
; to SP.
;
; Also, set the KILL and NEXT bits in the device table entry.  Upon the next
; printer write completion, code at NEXOUT will deallocate appropriate
; pieces of the workfile and its map.
;-

40$:    MOV     OT.CST(R4),R5           ; Get current stream pointer
;                               .ASSUME ST.STA EQ 0
60$:    BISB    #STA.KL,@R5             ; Set KILLED bit in stream table
        MOV     ST.NST(R5),R5           ; point to next stream in queue
        BNE     60$                     ; kill that too.

        BITB    #OTBSY,@R4              ; Device BUSY?
        BEQ     70$                     ; branch if not.
        BISB    #<OTKIL!OTNXT>,@R4      ; Kill all
70$:    RETURN
        ......

        .DSABL  LSB
.ENDC; NE SP$MLS
```

```
; Check IDLE/ACTIVE/HOLD status

.IF NE SP$MLS
;                                       .ASSUME OT.STA EQ 0
        BITB    #OTBSY,@R4              ; Is the device busy?
.IFF; NE SP$MLS
        BIT     #$BUSY,D.STAT(R4)       ; Is the device busy?
.ENDC; NE SP$MLS
        BNE     40$
        .PRINT  #SHO.ID                 ; print "IDLE"<CR>
        BR      120$
        ......

40$:
.IF NE SP$MLS
;                                       .ASSUME OT.STA EQ 0
        BITB    #OTOFF,@R4              ; Is output on hold?
.IFF; NE SP$MLS
        BIT     #$HOLD,D.STAT(R4)       ; Is output on hold?
.ENDC; NE SP$MLS
        BEQ     50$
        .PRINT  #SHO.HO                 ; print "HOLD"
        BR      60$
        ......

50$:    .PRINT  #SHO.AC                 ; print "ACTIVE"

60$:
.IF NE SP$MLS
; Count the WF map for entries whose stream numbers are associated
; with the device indicated by R3.

        CLR     R5                      ; init block count
        MOV     SP,LAST1                ; set optimizer value to ridiculous
        MOV     WFMAPA,R0               ; point to WFMAP

70$:    CMP     R0,WFMAPL               ; beyond end of WFMAP?
        BHIS    110$
        MOVB    (R0)+,R1                ; get stream number from map
        BIC     #^C<STRMSK>,R1          ; clear control bit(s)
        BEQ     70$                     ; don't look at unused blocks
        DEC     R1                      ; account for WFMAP's STREAM+1 format
        CMP     R1,LAST1                ; same stream as last time?
        BEQ     100$                    ; branch if optimizer worked...
        MOV     R1,LAST1                ; Shucks, have to find stream entry
        MOV     #STRTAB,R2              ; point to proper stream entry
90$:    DEC     R1
        BLT     100$                    ; branch out when we have it.
        ADD     #ST.ESZ,R2              ; point to next stream entry,
        BR      90$                     ;  and see if that's it.

100$:   CMPB    R3,ST.DTE(R2)           ; is it for this device?
        BNE     70$                     ; if not, go to next block
        INC     R5                      ; if so, chalk it up.
        BR      70$
        ......
.ENDC; NE SP$MLS

.IF EQ SP$MLS
        MOV     B.USED(R4),R5           ; R5 =  blocks left to output.
.ENDC; EQ SP$MLS
110$:   MOV     #<SHO.NB+1>,R1          ; R1 => target string.
        CALL    COMPUTE                 ; Compute the value in decimal.
        .PRINT  #SHO.NB                 ; Print how many blocks are used.
```

```
        .SBTTL  DOSHOW - Process SHOW QUEUE

        .ENABL  LSB

DOSHOW:
.IF EQ V$JOB
        BIC     #SHOW,@R3               ; Clear it so we don't repeat.
.IFF; EQ V$JOB
        MOV     #SHOW,R0                ; Clear it so we don't repeat.
        CALL    CSRBIC
.ENDC; EQ V$JOB

        .PRINT  #SHOHED                 ; display SHOW header

.IF NE SP$MLS
; Count the unused blocks

        CLR     EMPTYS                  ; init unused block count
        MOV     WFMAPA,R0               ; point to WFMAP
25$:    CMP     R0,WFMAPL               ; beyond end of WFMAP?
        BHIS    26$
        TSTB    (R0)+                   ; get stream number
        BNE     25$                     ; branch if NOT empty block
        INC     EMPTYS                  ; count the empties.
        BR      25$

; For each printer device, display one line of status.

26$:    MOV     #ODVTAB,R4              ; point to output device table
.IFF; NE SP$MLS
        MOV     #SDCTAB,R4              ; point to device table
.ENDC; NE SP$MLS
        CLR     R3                      ; start with SP0

30$:
.IF NE SP$MLS
; Loop here for each device...
; Start by making a device name.

        CMP     R3,#NUMDEV              ; beyond the last one?
        BHIS    130$                    ; branch if so.
.ENDC; NE SP$MLS
        MOV     R3,R1                   ; get unit number
        ADD     #'0,R1                  ; make it a digit
        MOVB    R1,SHO.UN               ; store it in string

        MOV     #SHO.DN,R1              ; point to device name area
.IF NE SP$MLS
        CALL    CVTDEV                  ; Convert device name
        BCS     120$                    ; if none, skip down.
.IFF; NE SP$MLS
        MOV     #PHYNAM,R0              ; point to physical name
        MOVB    (R0)+,(R1)+             ; move it into message area
        MOVB    (R0)+,(R1)+
        MOVB    (R0)+,(R1)+
        MOVB    #':,(R1)+
.ENDC; NE SP$MLS
        MOV     #' ,R0                  ; store some blanks
        .REPT 4
        MOVB    R0,(R1)+
        .ENDR
        MOVB    #200,@R1                ; terminate and print
        .PRINT  #SHO.LN
```

```
120$:
.IF NE SP$MLS
        ADD     #OT.ESZ,R4              ; point to next entry,
        INC     R3                      ; next device number
        BR      30$                     ; and print another one.
        ......
.ENDC; NE SP$MLS
```

```
; Finish the listing

130$:   .PRINT  #CRLF                   ; blank line
.IF NE SP$MLS
        MOV     EMPTYS,R5               ; Get the empty value
.IFF; NE SP$MLS
        MOV     B.END(R4),R5            ; Compute how many free blocks
        SUB     B.BEG(R4),R5            ; are available in the work file.
        INC     R5
        SUB     B.USED(R4),R5
.ENDC; NE SP$MLS
        MOV     #<LEFT+1>,R1            ; format the unused count
        CALL    COMPUTE
        .PRINT  #LEFT                   ; Print no. of blocks empty

.IF NE SP$DBG
 .IF NE SP$MLS
        .print  #crlf
        call    dumpo                   ; print dump of streams
 .ENDC; NE SP$MLS
.ENDC; NE SP$DBG

150$:   JMP     RETIME
        ......

.IF NE SP$MLS
; Convert device name in output table entry to ASCII, and put the
; result where R1 points.

CVTDEV: MOV     #R50NAM,R0
        MOV     OT.DEV(R4),@R0          ; is there a device assigned?
        BEQ     160$                    ; branch if not
        CLR     2(R0)
        CALL    $FNASC                  ; convert RAD50 to ASCII
        TST     (PC)+
160$:   SEC
        RETURN
        ......

.ENDC; NE SP$MLS
        .DSABL  LSB
```

```
        .SBTTL  COMPUT - Convert Binary to Decimal
;+
; Convert Binary to Decimal ASCII
;
;       R1 points to destination string.
;       R5 contains the value to be converted.
;-

        .ENABL  LSB

COMPUT: MOV     R2,-(SP)
        MOV     #DECTBL,R2              ; Point to power table

        MOV     R3,-(SP)
        MOV     R1,-(SP)                ; Save pointer to output string
        MOV     R1,R3
        ADD     #4,R3                   ; point to final digit in result area
COMPU1:

; Loop for each digit...

10$:    MOVB    #'0,@R1                 ; Start digit with zero

20$:    SUB     @R2,R5                  ; take away power of ten
        BCS     30$                     ; branch if too much
        INCB    @R1                     ; otherwise bump the digit.
        BR      20$                     ; and try again.

; Get ready to do next digit

30$:    CMP     R1,R3                   ; are we at the end?
        BHIS    40$                     ; if no more, quit.
        INC     R1                      ; point to next digit position
        ADD     (R2)+,R5                ; compensate for over-withdrawl
        BR      10$                     ;  and do next.

; Zero-suppress the string

40$:    MOV     (SP)+,R1                ; restore pointer to result
50$:    CMP     R1,R3                   ; at the end yet?
        BEQ     70$                     ; don't blank the final zero.
60$:    CMPB    (R1)+,#'0               ; is the digit zero?
        BNE     70$                     ; if not, exit.
        MOVB    #40,-(R1)               ; blank it out.
        INC     R1                      ; point to next.
        BR      50$                     ; look at the next digit

70$:    MOV     (SP)+,R3
        MOV     (SP)+,R2
        RETURN
        ......

DECTBL: .WORD   10000., 1000., 100., 10., 1.

        .DSABL  LSB
.IF NE SP$DBG
```

```
        .SBTTL  Binary to Octal conversion routine
;+
; This routine converts a binary number stored on R5 to an octal
; ASCII representation and prints it.
;
;       R5 = BINARY NUMBER TO BE PRINTED
;
;       CALL    OCTIMA
;
;-
        .ENABL  LSB

FMBUF:  .BLKB   8.

OCTIMA::.print                  ; print intro message

OCTIMF::MOV     R1,-(SP)                ;save R1
        MOV     #FMBUF,R1               ;Use FMBUF as temp buffer
        CALL    OCTIMB                  ;Do conversion,
        CLRB    (R1)                    ;plant terminator
        .print  #FMBUF                  ;Output the number
        MOV     (SP)+,R1
        RETURN
        ......

; Binary-to-OCTAL Ascii, R5=num, R1->buf

OCTIMB:
OCTIM1: MOV     R5,-(SP)                ;Store the value
        CLR     R5                      ;Serve as counter
        MOV     #8.,R0                  ; BASE
10$:    INC     R5                      ;Count one
        SUB     R0,(SP)                 ;Divide by repetive subtract
        BHIS    10$                     ;Branch if not minus
        ADD     #60,(SP)                ;Restore the remaining number
        ADD     R0,(SP)
        DEC     R5                      ;Decrement counter
        BEQ     20$                     ;If = 0 finish
        JSR     PC,OCTIM1               ;Repeat if not

20$:    MOV     (SP)+,R0
        MOVB    R0,(R1)+
        RETURN                          ;Keep printing till return address
        ......

        .DSABL  LSB
.ENDC; NE SP$DBG
```

```
        .SBTTL  SDWTDN - Printer Output Processor

; ----- Processing for Spooled Device WriTes DoNe  -----------------

        .ENABL  LSB

SDWTDN::
.IF EQ SP$MLS
        CLR     DEVNUM                  ; Device zero
        MOV     #SDCTAB,R4              ; point to device table
.IFF; EQ SP$MLS
        MOV     OORING,R4               ; point to output output ring ptr
        MOVB    (R4)+,R1                ; get requesting device number
        CMP     R4,#RINGMX              ; beyond end?
        BLOS    10$                     ; branch if not
        MOV     #ORING,R4               ; reset pointer to beginning of table
10$:    MOV     R4,OORING               ; and put it back
;       .type1  R4,<"  OORING = ">
        MOV     R1,DEVNUM               ; Store the device number safely away.
        CALL    CINDEX                  ; get device table entry address
.ENDC; EQ SP$MLS

20$:
        .type1  DEVNUM,<"  Completion on device ">
.IF EQ SP$MLS
        BIC     #$CRPND,D.STAT(R4)      ; Clear the postcompretproflagbit.
        BIT     #$HOLD,D.STAT(R4)       ; Is the output on hold?
.IFF; EQ SP$MLS
;                               .ASSUME OT.STA EQ 0
        BITB    #OTPND,@R4              ; post-completion bit set?
        BNE     30$
        .type1  R4,<"  DOESNT NEED COMPLETION?">
30$:
;                               .ASSUME OT.STA EQ 0
        BICB    #OTPND,@R4              ; Clear post-completion bit
;                               .ASSUME OT.STA EQ 0
        BITB    #OTOFF,@R4              ; Is output on hold for this device?
.ENDC; EQ SP$MLS
        BEQ     WTDN2                   ; No, carry on.
WTDNFK: MOV     DEVNUM,R1               ; get device number for fake
        CALLR   FAKEIO                  ; Yes, then do a timed fake outdone.
        ......

        .DSABL  LSB
```

```
        .ENABL  LSB

; Printer device write completion (continued)

WTDN2:
.IF EQ SP$MLS
; If a POST_NEXT or POST_KILL PAGE_EJECT hasn't been issued yet, do it now.

        BIT     #$NEXT,D.STAT(R4)       ; in NEXT mode?
        BEQ     10$                     ; branch if not.
        TSTB    FFDONE                  ; Post-kill FF needed?
        BNE     10$                     ; branch around it if not.
        .WRITW  #AREA,DEVNUM,#CRFF,#1,#1        ; do <CR><FF> on printer
        INCB    FFDONE                  ; flag it as done.

10$:    MOV     DEVBUF,R1               ; point to block buffer
        TST     B.USED(R4)              ; Is there anything more in storage?
        BNE     50$                     ; Yes, continue.
        BIC     #$BUSY!$KILL!$NEXT,D.STAT(R4) ; No, then set free of control
        RETURN                          ;  and postcompretpro is done.
        ......
.IFF; EQ SP$MLS

; Get output device's current stream table entry and stream number.

        MOV     OT.CST(R4),R5           ; Get current stream pointer
        MOV     R5,R1
        CALL    GSTNUM                  ; Get the stream number
        MOV     R2,STRNUM

; If stream is just now being started, use it's "First Workfile Block"

20$:    BITB    #STA.BS,@R5             ; stream in-progress?
        BEQ     80$                     ; if not, start up the stream.

; Check to see if the WF block just written was the last one available.

        MOV     ST.NXO(R5),R1           ; block just written
        CMP     ST.LWB(R5),R1           ; was it the last one available?
        BNE     WFBLK                   ; Go try to print some more.
        .dprint #equal
        .BR     STRDON
```

```
; Associate our device with the new stream.  Get new stream's 1st
; workfile block number.

70$:    .dprint #changg
        CALL    GSTNUM                  ; get stream number
        MOV     R2,STRNUM               ; store it away
        MOV     R1,R5                   ; Get that stream address

80$:    MOV     ST.WFB(R5),R0           ; start at this block
        BR      NEXOUT
        ......

; Search WFMAP for next WF block associated with the open stream.

WFBLK:  MOV     STRNUM,R2               ; R1 contains last "Next" block
        INC     R2                      ; Bias for WFMAP entries
        CALL    GNXWFB                  ; determine next "Next" workfile block
        BCS     STRDON                  ; ? none ?
        .BR     NEXOUT                  ; drop through...

        .DSABL  LSB
```

```
;+
; Output driver has caught up with input.  If the stream is not CLOSED,
; then we're hung - other streams can't be started.  If it IS CLOSED,
; then search for another pending stream on this device, and start it
; printing.
;-

;                               .ASSUME ST.STA EQ 0
STRDON: BITB    #STA.CL,@R5             ; test CLOSED bit
        BEQ     WTDNFK                  ; not closed?  Pretend we're HOLDing

; We're CLOSED and DONE PRINTING on this stream.  Wipe it out.

        MOV     ST.NST(R5),R1           ; Save next stream in queued...
        MOV     R1,OT.CST(R4)           ; make new (or no) stream current

        .dprint #erase
        MOV     #<ST.ESZ/2>,R3
30$:    CLR     (R5)+                   ; Erase stream entry.
        SOB     R3,30$

; Check to see whether there are any pending streams for output on
; this device.  If so, continue with the next one.

        TST     R1                      ; Any pending stream(s)?
        BNE     60$                     ; Branch if so.

; There are no more streams pending output on this device.  If we
; just processed a KILL, output a final page eject and then declare the
; device NOT BUSY.

        .dprint #devdon
;                               .ASSUME OT.STA EQ 0
        BITB    #<OTNXT!OTKIL>,@R4      ; Just did NEXT or KILL?
        BEQ     40$
        CALL    NEWPAG                  ; do a page-eject.
;                               .ASSUME OT.STA EQ 0
40$:    BICB    #<OTBSY!OTNXT!OTKIL>,@R4        ; Done with this device.
;                               .ASSUME OT.STA EQ 0
50$:    BICB    #OTPRM,@R4              ; force kick-start next time
        RETURN
        ......

; There is another stream waiting for output.  If we've been KILLing
; output, and this one is NOT marked KILL, then output a page eject,
; reset the KILLING bits in the device table entry, and begin processing
; it.

;                               .ASSUME ST.STA EQ 0
60$:    BITB    #STA.KL,@R1             ; Has this stream been KILLed?
        BNE     70$                     ; If so, continue processing it.
;                               .ASSUME OT.STA EQ 0
        BITB    #<OTNXT!OTKIL>,@R4      ; Just did NEXT or KILL?
        BEQ     70$
        CALL    NEWPAG                  ; do a page-eject.
;                               .ASSUME OT.STA EQ 0
        BICB    #<OTNXT!OTKIL>,@R4      ; From now on, no more killing.
```

```
        .SBTTL  NEXOUT - Next WF block to printer
;+
; Check to see if SET SPn NEXT or SET SPn KILL was typed.
;
; R0 contains the next workfile block on this stream for output.
; R4 -> Device table entry
; R5 -> Stream table entry
;-
        .ENABL  LSB

;                               .ASSUME OT.STA EQ 0
NEXOUT: BITB    #<OTNXT!OTKIL>,@R4      ; NEXT?
        BEQ     40$                     ; branch if not.
;                               .ASSUME ST.STA EQ 0
        BITB    #STA.BS,@R5             ; has stream started yet?
        BEQ     10$                     ; skip compare if not
        CMP     R0,ST.NXO(R5)           ; once around WFmap already?
        BEQ     STRDON                  ; if so, delete the stream.

; KILL or NEXT is happening on this device!  Deallocate this block,
; go back to WFBLK and get more until all are deallocated until
; a "block-0" block is found (NEXT), or until one complete cycle
; has been made through the workfile map.

;                               .ASSUME ST.STA EQ 0
10$:    BISB    #STA.BS,@R5             ; declare stream active
        MOV     R0,R1                   ; get next block with our stream no.
        ADD     WFMAPA,R1               ; point to WF map entry
;                               .ASSUME WM.BK0 EQ 200
        TSTB    @R1                     ; Block-zero bit set?
        BPL     20$                     ; branch if not.
;                               .ASSUME OT.STA EQ 0
        BITB    #OTKIL,@R4              ; Yes.  Is this a KILL?
        BEQ     30$                     ; branch if not - it's a NEXT.
20$:    CLRB    @R1                     ; otherwise, free it up.
        MOV     R0,R1                   ; starting at this block,
        BR      WFBLK                   ;  look for more.
        ......

; Resume with another stream after SET SPn NEXT

30$:    CALL    NEWPAG                  ; do a page-eject after SET SPn NEXT

; Get ready to read a block from the workfile, and write it to the printer.

;                               .ASSUME OT.STA EQ 0
40$:    BICB    #<OTKIL!OTNXT>,@R4      ; clear the NEXT and KILL bits
        MOV     R0,ST.NXO(R5)           ; this is the new block to do
        MOV     R0,RBLOCK               ; save it for write routine
;                               .ASSUME ST.STA EQ 0
        BISB    #STA.BS,@R5             ; declare stream in progress
        MOVB    ST.DTE(R5),R1           ; get device number
        SWAB    R1                      ; Calculate where the buffer is.
        ASL     R1                      ; BUFADR=(DEVNUM*512.)+DEVBUF
        ADD     DEVBUF,R1
.ENDC; EQ SP$MLS
50$:    CALL    SFREAD                  ; Go read a workfile block,
.IF EQ SP$MLS
        MOV     #256.,FB.WCT            ; Presume the write will be = 1 block.
        MOV     R1,R3                   ; R3 => DEVBUF for this device.
        ADD     #FB.SIZ,R3              ; Move past data values.
        MOV     #FB.BUF+4,R0            ; R0 => pattern matching flag buffer.
        MOV     #64.,R5                 ; R5 =  number of words to be matched.
60$:    CMP     (R3)+,(R0)+             ; Do the two locations match?
        BNE     DVWRIT                  ; No, then this isn't a flag block.
```

```
        DEC     R5                      ; All locations compared?
        BGT     60$                     ; No, then check some more.

; A "Flag block" has been found

        MOV     2(R1),FB.WCT            ; Yes, so store word count and block
        MOV     (R1),D.BLK(R4)          ; no. (they are possibly different).
        BIT     #$NEXT,D.STAT(R4)       ; Searching for the start of the
        BNE     70$                     ; Yes. Check out this one.
        CLRB    FFDONE                  ; No.  Reset the Post-Kill-FF flag.
        BEQ     80$                     ;  and carry on.

; we're in NEXT mode, searching for a BLOCK_0

70$:    TST     D.BLK(R4)               ; Yes, then is this block 0?
        BNE     80$                     ; No, then carry on.
        BIT     #$KILL,D.STAT(R4)       ; Is this a full kill?
        BNE     80$                     ; Yes, then don't unnext it.
        BIC     #$NEXT,D.STAT(R4)       ; No, then we're there so unflag.

80$:    CALL    SFREAD                  ; Now read in the real data block.
.IFF; EQ SP$MLS
        .BR     DVWRIT
.ENDC; EQ SP$MLS

        .DSABL  LSB
```

```
        .SBTTL  SFREAD - Read block from workfile
;+
; Read one block from the spool workfile
;
; R1 points to buffer,
; R4 points to device table entry
;-
        .ENABL  LSB

SFREAD:
.IF EQ SP$MLS
        .READW  #AREA,#SPCHAN,R1,#256.,B.HEAD(R4) ; Get the head block.
        BCC     10$                       ; Any read errors?
;+
;ERROR
        .PRINT  #IOERR                    ; Yes, report and exit.
;-
        .EXIT
        ......

10$:    DEC     B.USED(R4)                ; Count the block as being read.
        INC     B.HEAD(R4)                ; Update the head block pointer.
        CMP     B.HEAD(R4),B.END(R4)      ; Is our head at the top?
        BLOS    20$
        MOV     B.BEG(R4),B.HEAD(R4)      ; Yes, then reset it to the start.
20$:    RETURN                            ; No, then just return.
        ......
.IFF; EQ SP$MLS
        .READW  #AREA,#SPCHAN,R1,#256.,RBLOCK
        BCC     30$                       ; Any read errors?
;+
;ERROR
        .PRINT  #IOERR                    ; Yes, report and exit.
;-
        .EXIT
        ......

30$:    MOV     WFMAPA,R0                 ; point to workfile map
        ADD     RBLOCK,R0                 ; point to entry of block just read
        CLRB    @R0                       ; declare it UNUSED.
        RETURN
        ......
.ENDC; EQ SP$MLS

        .DSABL  LSB
.IF EQ SP$MLS
```

```
        .SBTTL  DVWRIT - Write to a PRINTER device
;+
; Write the contents of one block to a printer device.
;
; R1 points to buffer
; R4 points to device table entry
;-
        .ENABL  LSB

DVWRIT:
.IF EQ SP$MLS
        BIT     #$NEXT,D.STAT(R4)       ; Are we seeking next start of data?
        BEQ     10$                     ; No, go write the output.
        MOV     DEVNUM,R1               ; Yes, then fake it as if the output
        CALL    CR.DVD                  ; has been done.
        BR      20$                     ; Don't do the write.

10$:    .WRITC  #AREA,DEVNUM,R1,FB.WCT,#CR.DVD,D.BLK(R4) ; Write to device.
20$:    INC     D.BLK(R4)               ; Increment the device output block.
.IFF; EQ SP$MLS

        MOV     R2,-(SP)                ; save R2,
        CALL    CMPADR                  ; get completion routine address in R2
        .WRITC  #AREA,OT.CHA(R4),R1,#256.,R2,WBLOCK ; Write to printer
        BCS     OUTERR                  ; Abort on write errors.
        MOV     (SP)+,R2                ; Otherwise, restore R2,
        INC     WBLOCK                  ; (as if it really mattered)
.ENDC; EQ SP$MLS
        RETURN
        ......

.IF NE SP$MLS
; Compute completion routine address, based on the output unit number

CMPADR: MOV     DEVNUM,R2               ; Get device number,
        ASL     R2                      ; make word offset
        MOV     R2,-(SP)
        ASL     R2                      ; make 2-word offset
        ADD     (SP)+,R2                ; make 3-word offset
        ADD     #CR.DVD,R2              ; point to appropriate cmplt routine
60$:    RETURN
        ......

; Output a new page, typically after SET SP NEXT or SET SP KILL

NEWPAG: MOV     R0,-(SP)                ; save the block number
        .WRITW  #AREA,OT.CHA(R4),#CRFF,#1,#1    ; write <CR><FF> to printer
        MOV     (SP)+,R0
        BCC     60$                     ; return if no error.
;+
;ERROR
OUTERR: .PRINT  #DVWERR                 ; -F-Output error_
;-
        MOV     #SHO.DN,R1
        CALL    CVTDEV                  ; get ASCII printer name
        CLRB    @R1                     ; put NULL at end,
        .PRINT  #SHO.DN
        .EXIT
        ......
.ENDC; NE SP$MLS

        .DSABL  LSB
```

```
        .SBTTL  SIZEIT - Determine available space in workfile

        .ENABL  LSB
SIZEIT:
        MOV     B.END(R4),R1              ; Determine no. of unused blocks in
        SUB     B.BEG(R4),R1              ;  work file avail for this dev.
        SUB     B.USED(R4),R1
        CMP     R1,(R5)+                  ; There must be as many as he wants.
        BLE     50$                       ; No, then set the carry and return
        TST     (PC)+                     ; Clear the carry.
50$:    SEC                               ; Set the carry.
        RTS     R5                        ; Return
        ......

        .DSABL  LSB
.ENDC; EQ SP$MLS
.IF NE SP$MLS
```

```
        .SBTTL  MLSENT - Multi-Stream .ENTER processing
;+
; .ENTER action: Search stream table for
;
;       1. an unused stream (create)
;       2. specified channel/job ID with status = CLOSED (append)
;
; Traverse the queue of streams for this device.  If there is an existing
; closed stream with the same CSWA ID at the end of the queue, we will
; append to that stream.  Otherwise, add an entry to the end of the device's
; queue.
;
; On entry,
;       R0 =  stream CSWA ID
;       R4 -> head of output device's stream queue
;       R5 -> Q-element's JOB/FUNCTION word (not necessarily in a
;             Q-element; it may be a copy)
;
; On exit,
;       R1 = stream table entry pointer (on SUCCESS)
;       R4 -> output device table entry
;       R0, R2, R3, R5 are destroyed
;       C-bit indicates SUCCESS or FAILURE
;-
```

```
        .ENABL  LSB

MLSENT:
        .type1  r0,<'MLSENT entered; ID ='>

; Traverse the queue for this device, until the end is found.

        MOV     R4,R3                   ; Point to
        ADD     #OT.CST,R3              ;  head of device queue
        TST     @R3                     ; any streams hanging on?
        BEQ     40$                     ; branch if device is idle.
10$:    MOV     @R3,R1                  ; point to its next print stream.
        CMP     ST.ID(R1),R0            ; requested but already exists?
        BNE     20$                     ; branch if not.
;                               .ASSUME ST.STA EQ 0
        BITB    #STA.CL,@R1             ; Yes, but is it closed?
        BEQ     60$                     ; branch if it's additional .ENTER IO

20$:    MOV     R1,R3                   ; otherwise, let R3
        ADD     #ST.NST,R3              ;  point to its link word
        TST     @R3                     ; last entry in queue?
        BNE     10$                     ; and try next stream.

; If the stream at the end is CLOSED, and is not in the process of being
; KILLED, then use it ("append").  Otherwise, search for an unused stream,
; start it, and link it onto the device queue.

;                               .ASSUME ST.STA EQ 0
        BITB    #STA.CL,@R1             ; If stream is closed we can append...
        BEQ     40$                     ; Setup new stream if not.
;                               .ASSUME ST.STA EQ 0
        BITB    #STA.KL,@R1             ; If stream is being killed,
        BNE     40$                     ;  don't try to append

;                               .ASSUME ST.STA EQ 0
30$:    BISB    #STA.AP,@R1             ; Declare APPENDed stream
        BICB    #<STA.E2+STA.CL>,@R1    ;  Block-0 not marked, not closed
        BR      50$                     ; and set up APPEND.
        ......

; Begin a new stream here.  Look for an UNUSED stream.  Clear its
; status word.

40$:    MOV     R0,-(SP)                ; save the ID,
        CALL    SRCHUU                  ; search for unused entry
        MOV     (SP)+,R0                ;*C* get back the ID
        BCS     70$                     ; return if no free streams
```

```
;                               .ASSUME ST.STA EQ 0
        CLR     @R1                     ; reset status word
        MOV     R1,@R3                  ; add stream to this device's queue

; For either a NEW stream or an APPENDED stream...

50$:    MOV     R0,ST.ID(R1)            ; store stream ID
        MOV     @R5,R0                  ; get JOB/FUNCTION word
        SWAB    R0
        ASR     R0
        ASR     R0                      ; position on JOB part
        ASR     R0
        BIC     #177760,R0              ; mask out everything else
        MOVB    R0,ST.JOB(R1)           ; store job in stream entry
        MOVB    DEVNUM,ST.DTE(R1)       ; store device number
;                               .ASSUME ST.STA EQ 0
        BISB    #STA.EP,@R1             ; Declare .ENTER_IN_PROGRESS
;                               .ASSUME OT.STA EQ 0
        BISB    #OTBSY,@R4              ; Declare DEVICE BUSY
.IF NE SP$DBG
        CALL DUMPO
.ENDC; NE SP$DBG
60$:    CLC                             ; and SUCCESS
70$:    RETURN
        ......

100$:   TST     (SP)+                   ; Eat our return address,
        JMP     RETIME                  ; and exit.  Wait for KILL
        ......                          ;  to finish.

        .DSABL  LSB
```

```
        .SBTTL  SRCHST - Search for a stream table entry
;+
; SRCHST - Search for stream with ID in R0
; SRCHUU - Search for UNUSED stream
;
; Return with C-CLEAR if found, or C-SET if not found.
; If C=CLEAR (SUCCESS), R1 points to the stream table entry, and
; R2 contains the stream number
;
; Uses R1 and R2
;
; The entry SRCONT can be called to continue a search, assuming
; that R0, R1, and R2 have not been modified.
;-

        .ENABL  LSB

; Search for an UNUSED stream table entry

SRCHUU: CLR     R0                      ; let search ID = 0

; Search for stream with ID in R0

SRCHST: MOV     #STRTAB,R1              ; point to stream table
        CLR     R2                      ; entry counter
10$:    CMP     ST.ID(R1),R0            ; is this the one?
        BEQ     20$                     ; quit looping if found

SRCONT: ADD     #ST.ESZ,R1              ; point to next entry
        INC     R2                      ; count the entry
        CMP     #<NUMSTR+1>,R2          ; was that the last one?
        BHIS    10$                     ;*C* (C is set on fall-through)
20$:    RETURN
        ......

        .DSABL  LSB
```

```
        .SBTTL  GNXWFB - Search for next block

; Search the Workfile Map for the next block associated with stream R2.
; The search is performed in two passes; 1st is from the middle to
; the end, and 2nd is from the beginning to the middle.
;
; On entry, R1 should contain the block number to begin searching on,
; and R2 should contain a desired code (stream number+1 or zero for
; testing UNUSED).
;
; Returns with C-bit CLEAR if entry was found, R0 contains block no.
; or C-bit SET if no block found for specified stream.

        .ENABL  LSB

GNXWFB: MOV     R3,-(SP)                ; save registers
        MOV     R4,-(SP)
        MOV     WFMAPA,R0               ; point to workfile map
        ADD     R1,R0                   ; point to n'th entry
        MOV     WFMAPL,R3               ; 1st pass limit

10$:    CMP     R0,R3                   ; beyond limit?
        BHIS    30$                     ; branch if so.
        MOVB    (R0)+,R4                ; get the entry
        BIC     #^C<STRMSK>,R4          ; isolate the stream no.
        CMP     R4,R2                   ; is it the right one?
        BNE     10$                     ; go use it, if so.
        DEC     R0                      ; back up to the block entry
        SUB     WFMAPA,R0               ; determine block number (C-clear)
20$:    MOV     (SP)+,R4
        MOV     (SP)+,R3
        RETURN
        ......

30$:    CMP     R0,WFMAPL               ; beyond end?
        BLO     20$                     ; if not, it was 2nd pass - nothing?
                                        ;  (BLO = BCS; returns with C-set)
        MOV     WFMAPA,R0               ; 2nd pass - first map entry
        MOV     R0,R3
        ADD     R1,R3                   ; 2nd pass limit
        BR      10$                     ; do 2nd pass
        ......

        .DSABL  LSB
 .IF NE SP$DBG
```

```
        MOVB    #' ,(R1)+               ; store a blank
        MOVB    #' ,(R1)+               ; store a blank
        RETURN
        ......

FMT10:  MOV     R1,-(SP)                ; do a base-10 number
        CALL    COMPUTE                 ; format it.
        MOV     (SP)+,R1
        ADD     #5.,R1
        MOVB    #'.,(R1)+               ; store a dot
        MOVB    #' ,(R1)+               ; store a blank
        RETURN
        ......

DUMHD1: .ASCII  / Addres     ID    Status  Dev#   Job#  StrtBk/
        .ASCIZ  / LastBk NextBk  NextSt/
STLINE: .BLKB   80.
        .EVEN
OCTTBL: .WORD   100000, 10000, 1000, 100, 10, 1

COMPU8: MOV     R2,-(SP)
        MOV     #OCTTBL,R2
        MOV     R3,-(SP)
        MOV     R1,-(SP)                ; Save pointer to output string
        MOV     R1,R3
        ADD     #5,R3                   ; point to final digit in result area
        JMP     COMPU1
        ......

        .DSABL  LSB
 .endc; ne sp$dbg
.ENDC; NE SP$MLS
```

```
        .SBTTL  DUMPO  - Dump stream table

        .ENABL  LSB

; Do dump of stream table

DUMPO:  MOV     R0,-(SP)
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        MOV     R4,-(SP)
        MOV     R5,-(SP)

        .PRINT  #DUMHD1

        MOV     #NUMSTR,R2              ; for each stream,
        MOV     #STRTAB,R4
10$:    MOV     #STLINE,R1              ; point to output line

        MOV     R4,R5                   ; get address of entry
        CALL    FMT8

        MOV     ST.ID(R4),R5            ; get ID
        CALL    FMT8

        MOVB    ST.STA(R4),R5           ; get STATUS BYTE
        CALL    FMT8                    ; format it.

        MOVB    ST.DTE(R4),R5           ; get DEVICE TABLE ENTRY
        CALL    FMT10                   ; format it.

        MOVB    ST.JOB(R4),R5           ; get JOB NUMBER
        CALL    FMT10                   ; format it.

        MOV     ST.WFB(R4),R5           ; get FIRST WORKFILE BLOCK
        CALL    FMT10                   ; format it.

        MOV     ST.LWB(R4),R5           ; get LAST WORKFILE BLOCK WRITTEN
        CALL    FMT10                   ; format it.

        MOV     ST.NXO(R4),R5           ; get NEXT WORKFILE BLOCK TO WRITE
        CALL    FMT10                   ; format it.

        MOV     ST.NST(R4),R5           ; get NEXT STREAM
        CALL    FMT8                    ; format it.

        CLRB    @R1                     ; terminate string

        .PRINT  #STLINE                 ; print status line
        ADD     #ST.ESZ,R4              ; point to next stream
        SOB     R2,10$                  ; and do it, if there is one.

        MOV     (SP)+,R5
        MOV     (SP)+,R4
        MOV     (SP)+,R3
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        RETURN
        ......

FMT8:   MOV     R1,-(SP)                ; do a base-8 number
        CALL    COMPU8                  ; format it.
        MOV     (SP)+,R1
        ADD     #6.,R1
```

```
        .SBTTL  STOP   - Stop SPOOL immediately
;+
;  This sequence is executed when SPOOL is aborted, either by double
;  control-C, the KMON ABORT command, or by SET SP EXIT.
;-
        .ENABL  LSB

STOP::
.If EQ SP$MLS
        .ABTIO  DEVNUM                  ; Abort I/O on printer
.IfF; EQ SP$MLS
10$:    MOV     #ODVTAB+OT.CHA,R4       ; point to 1st device's channel
        MOV     #NUMDEV,R1              ; for each device,
20$:    .ABTIO  @R4                     ; Abort I/O on printer
        ADD     #OT.ESZ,R4              ; point to next device's entry
        SOB     R1,20$                  ; count down until done.
.EndC; EQ SP$MLS
        .PVAL   #AREA,#$SPSTA,#0        ; clear the $SPSTA word
;+
;ERROR
        .PRINT  #ABOMSG                 ; print an epitaph
;-
        .EXIT

        .DSABL  LSB
```

```
; ----- Print Screen for Professional Interface Handler --------------
;
; (Note: This code is only included in the XM (V$JOB) version)
;
; In this code, registers are:
;
;       R0 -  Scratch (for EMTs etc.)
;       R1 -  Scratch
;       R3 -> Spooler CSR word (RMON fixed offset)
;       R4 -> Spooled device control table entry for unit 0 (after CINDEX)
;       R5 -> Output buffer for SFWRITE, otherwise scratch.

; ----- Make sure it is a Professional 300 ---------------------------

        .ENABL  LSB

PSCREEN:
.IF NE SP$PPS
        TST     PRO                     ; Is this a PC300?
        BNE     10$                     ; Yes, carry on.
.ENDC; NE SP$PPS
.IF EQ V$JOB
        BIC     #PRTSCR,@R3             ; No, clear the print screen flag.
.IFF; EQ V$JOB
        MOV     #PRTSCR,R0
        CALL    CSRBIC                  ; No, clear the print screen flag.
.ENDC; EQ V$JOB
        JMP     RETIME                  ; And try again later.
        ......
.IF NE SP$PPS
10$:                                    ; Fall through.

 .IF EQ SP$MLS
; ----- Create unit index -------------------------------------------

PCINDX: CLR     R1                      ; Print screen to unit 0 always.
        CALL    CINDEX                  ; Create the index.
        JSR     R5,SIZEIT               ; Make sure there is enough room.
        .WORD   10.                     ; 10 blocks approx are needed.
        BCC     PCBLK0                  ; Branch if so.
        JMP     SPLIO                   ; No, try again later.
        ......
```

```
; ----- Write a flag block, pretend first block is block 0 -----------

PCBLK0: MOV     #1,FB.BLK               ; Assume we do not want a form feed.
        TST     PCOUNT+2                ; A while since the last PS?
        BNE     20$                     ; Yes, then select block 0.
        CMP     PCOUNT,#300.            ; At least 30 seconds since last?
        BLOS    30$                     ; No, then don't do block 0.
20$:    CLR     FB.BLK                  ; Yes, then start with block 0.
30$:    CLR     PCOUNT                  ; Notify that we've just printed.
        CLR     PCOUNT+2
        MOV     #256.,FB.WCT            ; There will be at least 1 block.
        MOV     #FB.BUF,R5              ; R5 -> flag block buffer.
        CALL    SFWRITE                 ; Write the flag block.
 .ENDC; EQ SP$MLS

 .IF NE SP$MLS
        MOV     #PIID,STRID             ; Special stream ID
        MOV     #ENTFAK,FUNCP           ; Point to fake .ENTER code
 .ENDC; NE SP$MLS

        .SPFUN  #AREA,#PICHAN,#FN$UPD,PSBUF,#HANUSE,#CURR$S ;Get tables
        BCC     50$                     ;No errors...
;+
;ERROR
40$:    .PRINT  #BADSPF                 ;Error with spfun...
;-
        .EXIT
        ......

50$:    MOV     PSBUF,R1                ;R1 -> Table area
        MOVB    S$VDF(R1),G2SUFX        ;Save the G2 suffix character for
                                        ; tracking NRC's on print screen

; ----- Loop Through Printing Each Line ------------------------------

        MOV     #1,R1                   ; R1 = Line number to read.
        MOV     SPFBFA,OUTBUF           ; OUTBUF -> spooler forwarding buf.

.IF NE SP$MLS
; ----- If timer has expired, send initial formfeed ------------------

        TST     PCOUNT+2                ; A while since the last PS?
        BNE     60$                     ; Yes, then output an FF.
        CMP     PCOUNT,#300.            ; At least 30 seconds since last?
        BLOS    70$                     ; No, then DON'T output an FF.
60$:    MOVB    #FF,R0
        CALL    CHROUT                  ; Output initial formfeed
70$:    CLR     PCOUNT
        CLR     PCOUNT+2
.ENDC; NE SP$MLS

; ----- Get the line data from PI ------------------------------------

GETLIN: MOV     PSBUF,R5                ; R5 -> Print screen input buffer.
        .SPFUN  #AREA,#PICHAN,#375,PSBUF,#1,R1  ; Read a line.
        BCS     40$                     ; Exit on errors from PI

80$:    MOV     R1,-(SP)                ; Save line number for later.
        BIT     #DBLTOP,2(R5)           ; The top half of a double wide line?
        BNE     NXTLIN                  ; Yes, then don't output anything.
```

```
; ----- Select the character width -----------------------------------

SELWID: MOV     #61,R0                  ; Assume 80 column, single width.
        MOV     #80.,R1                 ; R1 = characters on the line.
        BIT     #COL80,@R5              ; Is it?
        BEQ     90$                     ; If 0 then it is 80 column.
        MOV     #64,R0                  ; It is 132 column mode.
        MOV     #132.,R1                ; 132. characters to print.

90$:    BIT     #DBLBOT!DBLWID,2(R5)    ; Is this line to be double wide?
        BEQ     100$                    ; No, then the selected is correct.
        ADD     #4,R0                   ; Select double wide.
        ASR     R1                      ; Half as many characters.

100$:   TST     ..GENP                  ; Generic printer?
        BNE     120$
        MOVB    R0,110$                 ; Save the escape sequence value.
        JSR     R5,CHATER               ; Set the line attributes.
110$:    .BYTE  0,'w                    ; Escape sequence termination.

120$:   ADD     #4,R5                   ; Move to the character field.

; Truncate blanks at end of line

        MOV     R5,-(SP)
        ADD     R1,R5                   ; Point to end of line
        ADD     R1,R5                   ; (account for attribute bytes)
130$:   DEC     R5                      ; back over attribute byte
        CMPB    -(R5),#40               ; blank?
        BNE     140$                    ; if not, start printing.
        DEC     R1                      ; decrement char count
        CMP     R1,#1                   ; reached the beginning?
        BGT     130$
140$:   MOV     (SP)+,R5                ; restore pointer

; ----- Determine if the character attributes have changed ----------

CHRATR: TST     ..GENP                  ; Generic printer?
        BNE     WRDATA
        CMPB    1(R5),ATFLAG            ; Have the attributes changed?
        BEQ     WRDATA                  ; No, then go output the data.
        MOVB    1(R5),ATFLAG            ; Save the current atributes.
        .BR     SELSET

        .DSABL  LSB
```

```
; ----- Select the appropriate character set -------------------------

        .ENABL  LSB

SELSET: MOV     @R5,R0                  ;Get character attributes
        SWAB    R0                      ;Move to low byte
        ASR     R0                      ;Isolate font number
        BIC     #^C7,R0
        BNE     10$                     ;If not font 0 (G0)...
        JSR     R5,STROUT
         .BYTE  ESC,'(,'B,G0,0,0
        BR      SELREN
        ......

10$:    CMPB    R0,#1                   ;Line drawing set (g1)?
        BNE     20$                     ;Nope...
        JSR     R5,STROUT               ;Yes, select it...
         .BYTE  ESC,'),'0,G1,0,0
        BR      SELREN
        ......

20$:    JSR     R5,STROUT               ;G2 character set, move to g0
         .BYTE  ESC,'(
G2SUFX:  .BYTE  'B,G0,0,0               ; and select g0

; ----- Select the appropriate rendition -----------------------------

SELREN: JSR     R5,CHATER               ; Turn off the current rendition.
         .BYTE  '0,'m

        BIT     #BOLD,@R5               ; Has bold been selected?
        BEQ     40$
        JSR     R5,CHATER
         .BYTE  '1,'m

40$:    BIT     #UNDERS,@R5             ; Has underscore been selected?
        BEQ     WRDATA
        JSR     R5,CHATER
         .BYTE  '4,'m

; ----- Write the character itself -----------------------------------

WRDATA: MOVB    (R5)+,R0                ; Get the associated character.
        INC     R5                      ; Move past the attribute byte.
        CALL    CHROUT                  ; Output the character.
        SOB     R1,CHRATR               ; Repeat until whole line is output.
```

```
; ----- Move to the next line -------------------------------------------

NXTLIN: MOV     #15,R0                  ; Output carriage return/line feed.
        CALL    CHROUT
        MOV     #12,R0
        CALL    CHROUT
        MOV     (SP)+,R1                ; Restore old R1.
        INC     R1                      ; Select the next line?
        CMP     R1,#24.                 ; All lines printed?
        BHI     CLRUP
        JMP     GETLIN                  ; No, loop back till done.
        ......

; ----- Reset printer to normal rendition and G0 ------------------------

CLRUP:  TST     ..GENP                  ; Generic printer?
        BNE     60$                     ; skip if so
        JSR     R5,STROUT               ; Restore G0
         .BYTE  ESC,'(,'B,G0,0,0
        JSR     R5,CHATER               ; And select normal rendition.
         .BYTE  '0,'w
60$:    CALL    FLUSH                   ; Flush the rest of the buffer.

; ----- Start up the output if device not busy ----------------------

PCGO:
.IF EQ V$JOB
        BIC     #PRTSCR,@R3             ; Clear the print screen flag.
.IFF; EQ V$JOB
        MOV     #PRTSCR,R0              ; Clear the print screen flag.
        CALL    CSRBIC
.ENDC; EQ V$JOB
.IF NE SP$MLS
        MOV     #CLOFAK,FUNCP           ; point to close function
        CLR     R1                      ; output unit=0
.IF NE V$JOB
        CLR     SP$HTM+2+SP.WCR         ; pass word count=0
.ENDC; NE V$JOB
        CALL    PIDO1                   ; execute as an SP .CLOSE
        CALLR   TIMERO                  ; Reschedule ourself and return.
        ......
.IFF; NE SP$MLS
        BIT     #$BUSY,D.STAT(R4)       ; Are we running?
        BNE     70$                     ; Yes, then don't worry about startup.
        CLR     R1                      ; R1 = unit 0.
        CALL    CR.DVD                  ; Pretend the output device is done.
        BIS     #$BUSY,D.STAT(R4)       ; Set the unit busy.
70$:    CALLR   CR.TIM                  ; Reschedule ourself and return.
        ......
.ENDC; EQ SP$MLS

; ----- Subroutine to change standard attributes ----------------------

CHATER: MOV     #ESC,R0                 ; <esc>
        CALL    CHROUT                  ; Output the character.
        MOV     #'[,R0                  ; [
        CALL    CHROUT                  ; Output it.
        MOVB    (R5)+,R0                ; Selection character.
        CALL    CHROUT                  ; Output it.
        MOVB    (R5)+,R0                ; Termination character.
        CALL    CHROUT                  ; Output it.
        RTS     R5                      ; Return
        ......

        .DSABL  LSB

        .SBTTL  STROUT - Output an ASCIZ string

        .ENABL  LSB

; ----- Subroutine to output a control sequence pointed to by R5 -------

STROUT: MOVB    (R5)+,R0                ; Get a character.
        BEQ     10$                     ; Is it a null?
        CALL    CHROUT                  ; No, then output the character.
        BR      STROUT                  ; Back for more.

10$:    INC     R5                      ; Round R5 to the next even address.
        BIC     #1,R5                   ; Make it even.
        RTS     R5
        ......

        .SBTTL  CHROUT - Output Character

; ----- Subroutine to buffer a single character for output ------------

CHROUT: MOVB    R0,@OUTBUF              ; Store a character for output.
        BEQ     20$                     ; Bypass if it is a null.
CHROU2: INC     OUTBUF                  ; Count it.
        CMP     OUTBUF,OUTBEN           ; Buffer full?
        BLO     20$                     ; No, continue.
        MOV     R5,-(SP)                ; Save R5 for a moment.
.IF NE SP$MLS
        MOV     R4,-(SP)
        MOV     R3,-(SP)
        MOV     R2,-(SP)
        MOV     R1,-(SP)
        MOV     #256.,SP$HTM+2+SP.WCR   ; pass word count
        CLR     R1                      ; output unit=0
        CALL    PIDO1                   ; Write block to work file
        MOV     (SP)+,R1
        MOV     (SP)+,R2
        MOV     (SP)+,R3
        MOV     (SP)+,R4
        MOV     #IOFAK,FUNCP            ; Now point to fake .WRITE code
.IFF; NE SP$MLS
        MOV     SPFBFA,R5               ; R5 -> Spooler forwarding buffer.
        CALL    SFWRIT                  ; Write the block.
.ENDC; NE SP$MLS
        MOV     SPFBFA,OUTBUF           ; Reset buffer pointer.
        MOV     (SP)+,R5                ; Restore R5.
20$:    RETURN                          ; Return
        ......

; ----- Subroutine to flush the output buffer ---------------------------

FLUSH:  CMP     OUTBUF,SPFBFA           ; At start of buffer?
        BEQ     20$                     ; Yes, all done so return.
        CLRB    @OUTBUF                 ; No, store a null character.
        CALL    CHROU2                  ; And print it.
        BR      FLUSH
        ......

        .DSABL  LSB
.ENDC; NE SP$PPS

        .END
```