following the directions in A.8.5, be sure to use the Version 03 Linker:

```
          LINK PICTUR,VTLIB
```

VTLIB (Handler Modules):

Module	CSECT	Contains	Globals
VTCAL1	$GT1	.CLEAR .START .STOP .INSRT .REMOV	$VINIT $VSTRT $VSTOP $VNSRT $VRMOV
VTCAL2	$GT2	.BLANK .RESTR	$VBLNK $VRSTR
VTCAL3	$GT3	.LPEN .NAME .STAT .SYNC .NOSYN .TRACK	$VLPEN $NAME $VSTPM $SYNC $NOSYN $VTRAK
VTCAL4	$GT4	.LNKRT .UNLNK .SCROL	$VRTLK $VUNLK $VSCRL
VTBASE	$GTB	Interrupt handlers and internal display file.	$DFILE

The five modules in VTHDLR can be used in three different ways. When space is not critical, the most straightforward way is to link VTHDLR directly with a display program. The following command is an example.

```
     LINK PICTUR,VTHDLR
```

It is often necessary to conserve space, however, and selective loading of modules is possible by first creating an indexed object module library from VTHDLR and then by making global calls within the display program. The following command creates an indexed object module library.

```
     LIBRARY/CREATE VTLIB VTHDLR
```

To further conserve space with overlays, it is also possible to extract individual object modules from a library and create separate object module files. For example, to link a display program using overlays, the following statements are a typical sequence of creating, extracting and linking commands. (NOTE: the modules VTCAL1 and VTCAL2 must be in the same overlay if any global in either one is used.)

```
           .
           .
           .
.LIBRARY/CREATE VTLIB VTHDLR
           .
           .
           .
.LIBRARY/EXTRACT VTLIB VTCAL1
GLOBAL? $VSTRT !moves entire module with $VSTRT to VTCAL1
GLOBAL? !Terminates prompting sequence
.LIBRARY/EXTRACT VTLIB VTCAL2
GLOBAL? $VBLNK !Moves the entire module to VTCAL2
GLOBAL?
.LIBRARY/EXTRACT VTLIB VTCAL3
GLOBAL? $VLPEN !Moves the entire module
GLOBAL?
.LIBRARY/EXTRACT VTLIB VTCAL4
GLOBAL? $VRTLK !Moves the entire module
GLOBAL?
.LIBRARY/EXTRACT VTLIB VTBASE
GLOBAL? $DFILE !Moves the entire module
GLOBAL?
           .
           .
           .
.LINK/PROMPT PICTUR,VTBASE
*VTCAL1,VTCAL2,VTCAL3/O:1
*VTCAL4/O:1
*//
           .
           .
           .
```

A.5 DISPLAY FILE STRUCTURE

The Display File Handler supports a variety of display file structures, takes over the job of display processor management for the programmer, and may be used for both assembly language graphics programming and for systems program development. For example, the Handler supports the tagged subpicture file structure used by the BASIC-11 graphics software, as well as simple file structures. These are discussed in this section.

A.5.1 Subroutine Calls

A subroutine call instruction, with the mnemonic DJSR, is implemented using the display stop (DSTOP) instruction with an interrupt. The display stop interrupt routine in the Display File Handler simulates the DJSR instruction, and this allows great flexibility in choosing the characteristics of the DJSR instruction.

The DJSR instruction stops the display processor and requests an interrupt. The DJSR instruction may be followed by two or more words, and in this implementation the exact number may be varied by the programmer at any time. The basic subroutine call has this form:

```
DJSR
Return Address
Subroutine Address
```

In practice, simple calls to subroutines could look like:

```
DJSR
.WORD     .+4
.WORD     SUB
```

where SUB is the address of the subroutine. Control will return to the display instruction following the last word of the subroutine call. This structure permits a call to the subroutine to be easily by-passed without stopping the display processor, by replacing the DJSR with a display jump (DJMP) instruction:

```
DJMP
.WORD     .+4
.WORD     SUB
```

A more complex display file structure is possible if the return address is generalized:

```
.DJSR
.WORD     NEXT
.WORD     SUB
```

where NEXT is the generalized return address. This is equivalent to the sequence:

```
DJSR
.WORD     .+4
.WORD     SUB
DJMP
.WORD     NEXT
```

It is also possible to store non-graphic data such as tags and pointers in the subroutine call sequence, such as is done in the tagged subpicture display file structure of the BASIC-11 graphics software. This technique looks like:

```
          DJSR
          .WORD     NEXT
          .WORD     SUB
          DATA
   NEXT:    .
            .
            .
```

For simple applications where the flexibility of the DJSR instruction described above is not needed and the resultant overhead is not desired, the Display File Handler (VTBASE.MAC and VTCALL.MAC) can be conditionally re-assembled to produce a simple DJSR call. If NOTAG is defined during the assembly, the Handler will be configured to support this simple DJSR call:

```
DJSR
.WORD     SUB
```

where SUB is the address of the subroutine. Defining NOTAG will eliminate the subpicture tag capability, and with it the tracking object, which uses the tag feature to identify itself to the light pen interrupt handler.

Whatever the DJSR format used, all subroutines and the user main file must be terminated with a subroutine return instruction. This is implemented as a display stop instruction (given the mnemonic DRET) with an argument of zero. A subroutine then has the form:

```
SUB: Display Code
     DRET
     .WORD 0
```

A.5.2 **Main File/Subroutine Structure**

A common method of structuring display files is to have a main file which calls a series of display subroutines. Each subroutine will produce a picture element and may be called many times to build up a picture, producing economy of code. If the following macros are defined:

```
.MACRO    CALL <ARG>
DJSR
.WORD     .+4
.WORD     ARG
.ENDM
.MACRO    RETURN
DRET
.WORD     0
.ENDM
```

then a main file/subroutine file structure would look like:

```
;MAIN DISPLAY FILE
;
MAIN:     Display Code
          CALL SUB1        ;CALL SUBROUTINE 1
          Display Code
          CALL SUB2        ;CALL SUBROUTINE 2
           .               ;ETC
           .
           .
          RETURN
```

```
;
;DISPLAY  SUBROUTINES
;
SUB1:     Display Code    ;SUBROUTINE 1
          RETURN
;
SUB2:     Display Code    ;SUBROUTINE 2
          RETURN
            .             ;ETC.
            .
            .
```

A.5.3 BASIC-11 Graphic Software Subroutine Structure

An example of another method of structuring display files is the tagged subpicture structure used by BASIC-11 graphic software. The display file is divided into distinguishable elements called subpictures, each of which has its own unique tag.

The subpicture is constructed as a subroutine call followed by the subroutine. It is essentially a merger of the main file/subroutine structure into an in-line sequence of calls and subroutines. As such, it facilitates the construction of display files in real time, one of the important advantages of BASIC-11 graphic software.

The following is an example of the subpicture structure. Each subpicture has a call to a subroutine with the return address set to be the address of the next subpicture. The subroutine called may either immediately follow the call, or may be a subroutine defined as part of a subpicture created earlier in the display file. This permits a subroutine to be used by several subpictures without duplication of code. Each subpicture has a tag to identify it, and it is this tag which is returned by the light pen interrupt routine. To facilitate finding subpictures in the display file, they are made into a linked list by inserting a forward pointer to the next tag.

```
SUB1:     DJSR                  ;START OF SUBPICTURE 1
          .WORD     SUB2        ;NEXT SUBPICTURE
          .WORD     SUB1+12     ;JUMP TO THIS SUBPICTURE
          .WORD     1           ;TAG = 1
          .WORD     SUB2+6      ;POINTER TO NEXT TAG

;BODY OF SUBPICTURE 1

          DRET                  ;RETURN FROM
          0                     ;SUBPICTURE 1

SUB2:     DJSR                  ;START SUBPICTURE 2
          .WORD     SUB3        ;NEXT SUBPICTURE
          .WORD     SUB2+12     ;JUMP TO THIS SUBPICTURE
          .WORD     2           ;TAG 2
          .WORD     SUB3+6      ;PTR TO NEXT TAG
```

```
;BODY OF SUBPICTURE 2

          DRET                   ;RETURN FROM
          .WORD     0            ;SUBPICTURE 2

SUB3:     DJSR                   ;START SUBPICTURE 3
          .WORD     SUB4         ;NEXT SUBPICTURE
          .WORD     SUB1+12      ;COPY SUBPICTURE 1
                                 ;FOR THIS SUBPICTURE
          .WORD     3            ;BUT TAG IT 3.
          .WORD     SUB4+6       ;PTR TO NEXT TAG

SUB4:     DJSR                   ;START SUBPICTURE 4
          .                      ;ETC.
          .
          .
```

A.6 SUMMARY OF GRAPHICS MACRO CALLS

Mnemonic	Function	MACRO Call (see Note 1)	Assembly Language Expansion (see Note 2)
.BLANK	Temporarily blanks a user display file.	.BLANK faddr	.GLOBL $VBLNK .IF NB, faddr MOV faddr, ^100 .ENDC JSR ^07, $VBLNK
.CLEAR	Initializes handler.	.CLEAR	.GLOBL $VINIT JSR ^07, $VINIT
.INSRT	Inserts a call to user display file in handler's master display file.	.INSRT faddr	.GLOBL $VNSRT .IF NB, faddr MOV faddr, ^00 .ENDC JSR ^07, $VNSRT
.LNKRT	Sets up vectors and links display file handler to RT-11 scroller.	.LNKRT	.GLOBL $VRTLK JSR ^07, $VRTLK
.LPEN	Sets up light pen status buffer.	.LPEN baddr	.GLOBL $VLPEN .IF NB, baddr MOV baddr, ^00 .ENDC JSR ^07, $VLPEN
.NAME	Sets up buffer to receive name register stack contents.	.NAME \baddr	.GLOBL $NAME .IF NB, baddr MOV .BEDDR, ^00 .endc JSR ^07, $NAME
.NOSYN	Disables power line synchronization.	.NOSYN	.GLOBL $NOSYN JSR ^07, $NOSYN

Mnemonic	Function	MACRO Call (see Note 1)	Assembly Language Expansion (see Note 2)
.REMOV	Removes the call to a user display file.	.REMOV faddr	.GLOBL $VRMOV .IF NB, faddr MOV faddr, ^O0 .ENDC JSR ^O7, $VRMOV
.RESTR	Unblanks the user display file.	.RESTR faddr	.GLOBL $VRSTR IF NB, faddr MOV faddr, ^O0 .ENDC JSR ^O7, $VRSTR
.SCROL	Adjusts monitor scroller parameters.	.SCROL baddr	.GLOBL $VSCRL .IF NB, baddr MOV baddr, ^O0 .ENDC JSR ^O7, $VSCRL
.START	Starts the display.	.START	.GLOBL $VSTRT JSR ^O7, $VSTRT
.STAT	Sets up status buffer.	.STAT baddr	.GLOBL $VSTPM .IF NB, baddr MOV baddr, ^O0 .ENDC JSR ^O7, $VSTPM
.STOP	Stops the display.	.STOP	.GLOBL $VSTOP JSR ^O7, $VSTOP
.SYNC	Enables power line synchronization.	.SYNC	.GLOBL $SYNC JSR ^O7, $SYNC
.TRACK	Enables the track object.	.TRACK baddr, croutine	.GLOBL $VTRAK .IF NB, baddr MOV baddr, ^O0 .ENDC .IF NB, croutine MOV croutine,- (^O6) .IFF CLR-(^O6) .ENDC .NARG T .IF EQ, T CLR ^O0 .ENDC JSR ^O7, $VTRAK

Mnemonic	Function	MACRO Call (see Note 1)	Assembly Language Expansion (see Note 2)
.UNLNK	Unlinks display handler from RT-11 if linked (otherwise leaves display stopped).	.UNLNK	.GLOBL $VUNLK JSR ^07, $VUNLK

NOTE 1

baddr — Address of data buffer.

faddr — Address of start of user display file.

croutine — Address of .TRACK completion routine.

NOTE 2

The lines preceded by a dot will not be assembled. The code they enclose may or may not be assembled depending on the conditionals.

A.7 DISPLAY PROCESSOR MNEMONICS

Mnemonic		Value	Function
CHAR	=	100000	Character Mode
SHORTV	=	104000	Short Vector Mode
LONGV	=	110000	Long Vector Mode
POINT	=	114000	Point Mode
GRAPHX	=	120000	Graphplot X Mode
GRAPHY	=	124000	Graphplot Y Mode
RELATV	=	130000	Relative Point Mode
INT0	=	2000	Intensity 0 (Dim)
INT1	=	2200	Intensity 1
INT2	=	2400	Intensity 2
INT3	=	2600	Intensity 3
INT4	=	3000	Intensity 4
INT5	=	3200	Intensity 5
INT6	=	3400	Intensity 6
INT7	=	3600	Intensity 7 (Bright)
LPOFF	=	100	Light Pen Off
LPON	=	140	Light Pen On
BLKOFF	=	20	Blink Off
BLKON	=	30	Blink On
LINE0	=	4	Solid Line

LINE1	=	5	Long Dash
LINE2	=	6	Short Dash
LINE3	=	7	Dot Dash
DJMP	=	160000	Display Jump
DNOP	=	164000	Display No Operation
STATSA	=	170000	Load Status A Instruction
LPLITE	=	200	Light Pen Hit On
LPDARK	=	300	Light Pen Hit Off
ITAL0	=	40	Italics Off
ITAL1	=	60	Italics On
SYNC	=	4	Halt and Resume Synchronized
STATSB	=	174000	Load Status B Instruction
INCR	=	100	Graphplot Increment
(Vector/Point Mode)			
INTX	=	40000	Intensity Vector or Point
MAXX	=	1777	Maximum X Component
MAXY	=	1377	Maximum Y Component
MINUSX	=	20000	Negative X Component
MINUSY	=	20000	Negative Y Component
(Short Vector Mode)			
SHIFTX	=	200	
MAXSX	=	17600	Maximum X Component
MAXSY	=	77	Maximum Y Component
MISVX	=	20000	Negative X Component
MISVY	=	100	Negative Y Component

A.8 ASSEMBLY INSTRUCTIONS

A.8.1 General Instructions

All programs can be assembled in 16K, using RT-11 MACRO. All assemblies and all links should be error free. The following conventions are assumed:

1. Default file types are not explicitly typed. These are .MAC for source files, .OBJ for assembler output, and .SAV for Linker output.

2. The default device (DK) is used for all files in the example command strings.

3. Listings and link maps are not generated in the example command strings.

A.8.2 VTBASE

To assemble VTBASE with RT-11 link-up capability:

```
MACRO VTBASE
```

A.8.3 VTCAL1 - VTCAL4

To assemble the modules VTCAL1 through VTCAL4:

```
MACRO VTCAL1,VTCAL2,VTCAL3,VTCAL4
```

A.8.4 VTHDLR

To create the concatenated handler module:

```
COPY/BINARY  VTCAL1.OBJ,VTCAL2.OBJ,VTCAL3.OBJ,-
VTCAL4.OBJ,VTBASE.OBJ  VTHDLR.OBJ
```

A.8.5 Building VTLIB.OBJ

To build the VTLIB library:

```
LIBRARY/CREATE VTLIB VTHDLR
```

A.9 VTMAC

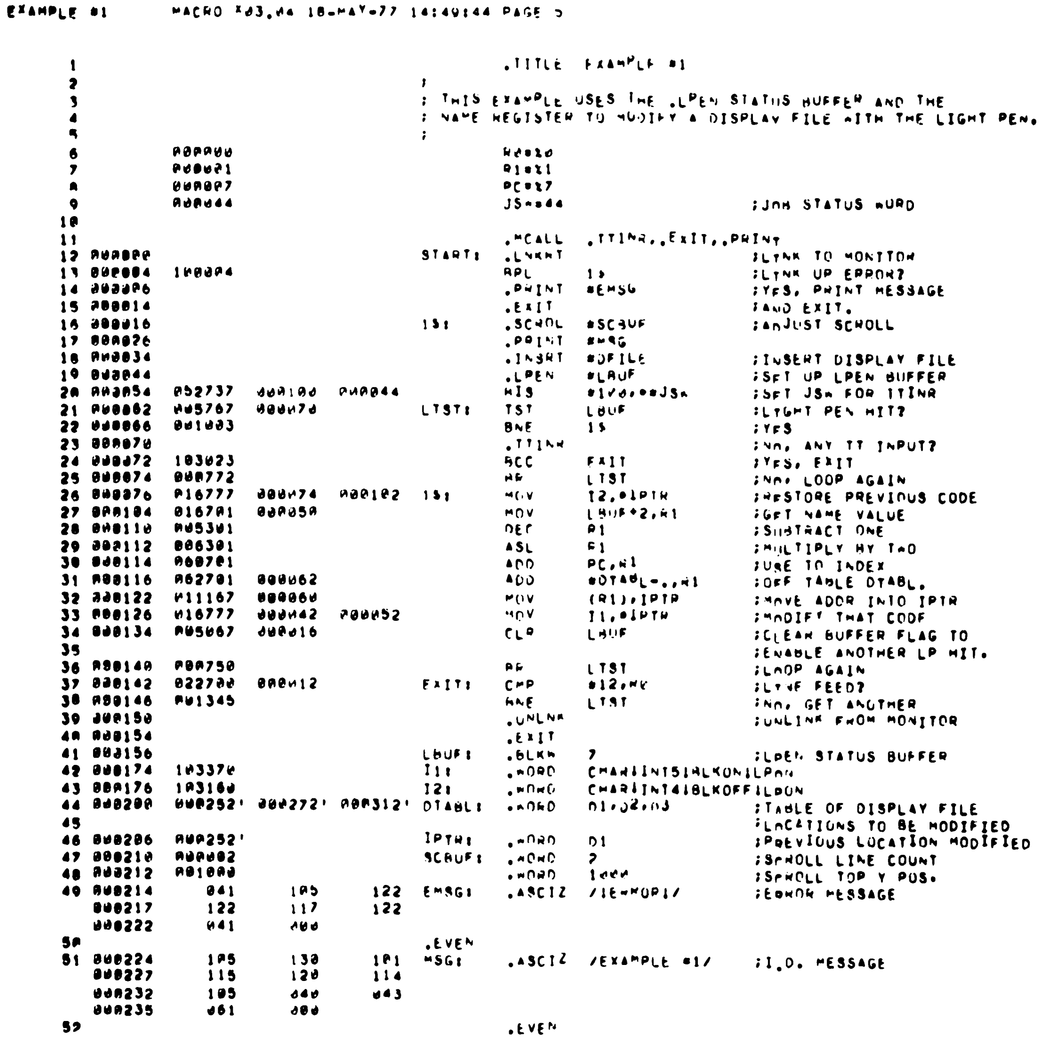
```
        .TITLE  VTMAC
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY ONLY BE USED
; OR COPIED IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE.
;
; COPYRIGHT (C) 1978, DIGITAL EQUIPMENT CORPORATION.
;
; VTMAC IS A LIBRARY OF MACRO CALLS AND MNEMONIC DEFINITIONS WHICH
; PROVIDE SUPPORT OF THE VT11 DISPLAY PROCESSOR. THE MACROS PRODUCE
; CALLS TO THE VT11 DEVICE SUPPORT PACKAGE, USING GLOBAL REFERENCES.

; MACRO TO GENERATE A MACRO WITH ZERO ARGUMENTS.
```

```
.MACRO MAC0 NAME,CALL
        .MACRO NAME
        .GLOBL  CALL
        JSR     PC,CALL
        .ENDM
.ENDM

; MACRO TO GENERATE A MACRO WITH ONE ARGUMENT

.MACRO MAC1 NAME,CALL
        .MACRO NAME ARG
        .IF NB,ARG
        MOV     ARG,%^00
        .ENDC
        .GLOBL  CALL
        JSR     PC,CALL
        .ENDM
.ENDM

; MACRO TO GENERATE A MACRO WITH TWO OPTIONAL ARGUMENTS

.MACRO MAC2 NAME,CALL
        .MACRO NAME ARG1,ARG2
        .GLOBL  CALL
        .IF NB,ARG1
        MOV     ARG1,%^00
        .ENDC
        .IF NB,ARG2
        MOV     ARG2,-(SP)
        .IFF
        CLR     -(SP)
        .NARG   T
        .IF EQ,T
        CLR     %^00
        .ENDC
        .ENDC
        JSR     PC,CALL
        .ENDM
.ENDM

; MACRO LIBRARY FOR VT11:

MAC0    <.CLEAR>,<$VINIT>
MAC0    <.STOP>,<$VSTOP>
MAC0    <.START>,<$VSTRT>
MAC1    <.INSRT>,<$VNSRT>
MAC1    <.REMOV>,<$VRMOV>
MAC1    <.BLANK>,<$VBLNK>
MAC1    <.RESTR>,<$VRSTR>
MAC1    <.STAT>,<$VSTPM>
MAC1    <.LPEN>,<$VLPEN>
MAC1    <.SCROL>,<$VSCRL>
MAC2    <.TRACK>,<$VTRAK>
MAC0    <.LNKRT>,<$VRTLK>
MAC0    <.UNLNK>,<$VUNLK>
```

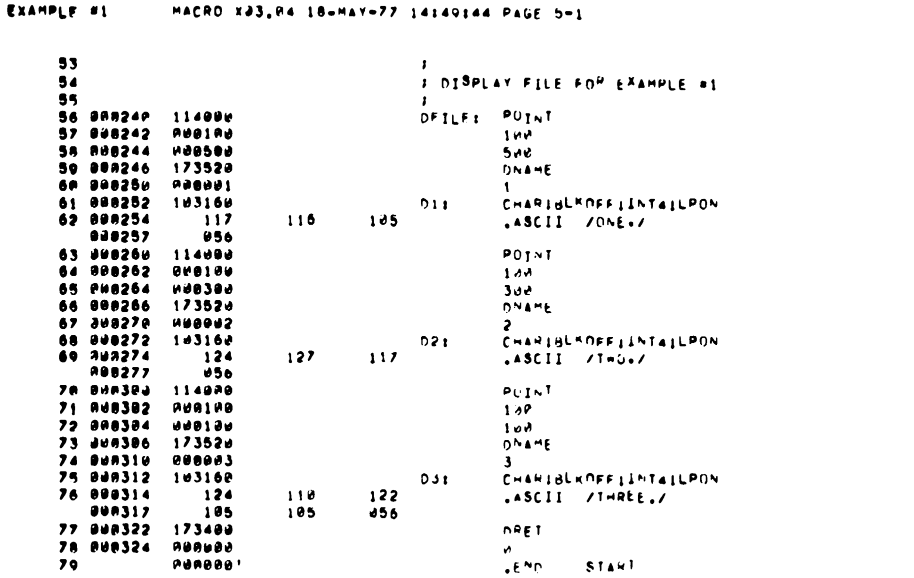
DISPLAY FILE HANDLER

```
; MNEMONIC DEFINITIONS FOR THE VT11 DISPLAY PROCESSOR

DJMP=160000        ;DISPLAY JUMP
DNOP=164000        ;DISPLAY NOP
DJSR=173400        ;DISPLAY SUBROUTINE CALL
DRET=173400        ;DISPLAY SUBROUTINE RETURN
DNAME=173520       ;SET NAME REGISTER
DSTAT=173420       ;RETURN STATUS DATA
DHALT=173500       ;STOP DISPLAY AND RETURN STATUS DATA

CHAR=100000        ;CHARACTER MODE
SHORTV=104000      ;SHORT VECTOR MODE
LONGV=110000       ;LONG VECTOR MODE
POINT=114000       ;POINT MODE
GRAPHX=120000      ;GRAPH X MODE
GRAPHY=124000      ;GRAPH Y MODE
RELATV=130000      ;RELATIVE VECTOR MODE

INT0=2000          ;INTENSITY 0
INT1=2200
INT2=2400
INT3=2600
INT4=3000
INT5=3200
INT6=3400
INT7=3600

LPOFF=100          ;LIGHT PEN OFF
LPON=140           ;LIGHT PEN ON
BLKOFF=20          ;BLINK OFF
BLKON=30           ;BLINK ON
LINE0=4            ;SOLID LINE
LINE1=5            ;LONG DASH
LINE2=6            ;SHORT DASH
LINE3=7            ;DOT DASH

STATSA=170000      ;LOAD STATUS REG A
LPLITE=200         ;INTENSIFY ON LPEN HIT
LPDARK=300         ;DON'T INTENSIFY
ITAL0=40           ;ITALICS OFF
ITAL1=60           ;ITALICS ON
SYNC=4             ;POWER LINE SYNC

STATSB=174000      ;LOAD STATUS REG B
INCR=100           ;GRAPH PLOT INCREMENT
INTX=40000         ;INTENSIFY VECTOR OR POINT
MAXX=1777          ;MAXIMUM X INCR. - LONGV
MAXY=1377          ;MAXIMUM Y INCR. - LONGV
MINUSX=20000       ;NEGATIVE X INCREMENT
MINUSY=20000       ;NEGATIVE Y INCREMENT
MAXSX=17600        ;MAXIMUM X INCR. - SHORTV
MAXSY=77           ;MAXIMUM Y INCR. - SHORTV
MISVX=20000        ;NEGATIVE X INCR. - SHORTV
MISVY=100          ;NEGATIVE Y INCR. - SHORTV
```

A.10 EXAMPLES USING GTON



```
EXAMPLE #1      MACRO X03.04 18-MAY-77 14:40:44 PAGE 5


                                                .TITLE  EXAMPLE #1
                                        ;
                                        ; THIS EXAMPLE USES THE .LPEN STATUS BUFFER AND THE
                                        ; NAME REGISTER TO MODIFY A DISPLAY FILE WITH THE LIGHT PEN.
                                        ;
           000000                               R0=%0
           000001                               R1=%1
           000007                               PC=%7
           000044                               JSW=44                          ;JOB STATUS WORD

                                                .MCALL  .TTINR,.EXIT,.PRINT
    000000                              START:  .LNKRT                          ;LINK TO MONITOR
    000004 100004                               BPL     1$                      ;LINK UP ERROR?
    000006                                      .PRINT  #EMSG                   ;YES, PRINT MESSAGE
    000014                                      .EXIT                           ;AND EXIT.
    000016                              1$:     .SCROL  #SCBUF                  ;ADJUST SCROLL
    000026                                      .PRINT  #MSG
    000034                                      .INSRT  #DFILE                  ;INSERT DISPLAY FILE
    000044                                      .LPEN   #LBUF                   ;SET UP LPEN BUFFER
    000054 052737  000100  000044               BIS     #100,@#JSW              ;SET JSW FOR TTINR
    000062 005767  000070               LTST:   TST     LBUF                    ;LIGHT PEN HIT?
    000066 001003                               BNE     1$                      ;YES
    000070                                      .TTINR                          ;NO, ANY TT INPUT?
    000072 103023                               BCC     EXIT                    ;YES, EXIT
    000074 000772                               BR      LTST                    ;NO, LOOP AGAIN
    000076 016777  000074  000102       1$:     MOV     I2,@IPTR                ;RESTORE PREVIOUS CODE
    000104 016701  000050                       MOV     LBUF+2,R1               ;GET NAME VALUE
    000110 005301                               DEC     R1                      ;SUBTRACT ONE
    000112 006301                               ASL     R1                      ;MULTIPLY BY TWO
    000114 060701                               ADD     PC,R1                   ;USE TO INDEX
    000116 062701  000062                       ADD     #DTABL-.,R1             ;OFF TABLE DTABL.
    000122 011167  000060                       MOV     (R1),IPTR               ;MOVE ADDR INTO IPTR
    000126 016777  000042  000052               MOV     I1,@IPTR                ;MODIFY THAT CODE
    000134 005067  000016                       CLR     LBUF                    ;CLEAR BUFFER FLAG TO
                                                                                ;ENABLE ANOTHER LP HIT.
    000140 000750                               BR      LTST                    ;LOOP AGAIN
    000142 022700  000012               EXIT:   CMP     #12,R0                  ;LINE FEED?
    000146 001345                               BNE     LTST                    ;NO, GET ANOTHER
    000150                                      .UNLNK                          ;UNLINK FROM MONITOR
    000154                                      .EXIT
    000156                              LBUF:   .BLKW   7                       ;LPEN STATUS BUFFER
    000174 103370                       I1:     .WORD   CHAR!INT5!BLKON!LPON
    000176 103160                       I2:     .WORD   CHAR!INT4!BLKOFF!LPON
    000200 000252' 000272' 000312'      DTABL:  .WORD   D1,D2,D3                ;TABLE OF DISPLAY FILE
                                                                                ;LOCATIONS TO BE MODIFIED
    000206 000252'                      IPTR:   .WORD   D1                      ;PREVIOUS LOCATION MODIFIED
    000210 000002                       SCBUF:  .WORD   2                       ;SCROLL LINE COUNT
    000212 001000                               .WORD   1000                    ;SCROLL TOP Y POS.
    000214    041     105     122       EMSG:   .ASCIZ  /!ERROR!/               ;ERROR MESSAGE
    000217    122     117     122
    000222    041     000
                                        .EVEN
    000224    105     130     101       MSG:    .ASCIZ  /EXAMPLE #1/            ;I.D. MESSAGE
    000227    115     120     114
    000232    105     040     043
    000235    061     000
                                                .EVEN
```



```
EXAMPLE #1      MACRO X03.04 18-MAY-77 14:40:44 PAGE 5-1


                                        ;
                                        ; DISPLAY FILE FOR EXAMPLE #1
                                        ;
    000240 114000                       DFILE:  POINT
    000242 000100                               100
    000244 000500                               500
    000246 173520                               DNAME
    000250 000001                               1
    000252 103160                       D1:     CHAR!BLKOFF!INT4!LPON
    000254    117     116     105               .ASCII  /ONE./
    000257    056
    000260 114000                               POINT
    000262 000100                               100
    000264 000300                               300
    000266 173520                               DNAME
    000270 000002                               2
    000272 103160                       D2:     CHAR!BLKOFF!INT4!LPON
    000274    124     127     117               .ASCII  /TWO./
    000277    056
    000300 114000                               POINT
    000302 000100                               100
    000304 000100                               100
    000306 173520                               DNAME
    000310 000003                               3
    000312 103160                       D3:     CHAR!BLKOFF!INT4!LPON
    000314    124     110     122               .ASCII  /THREE./
    000317    105     105     056
    000322 173400                               DRET
    000324 000000                               0
           000000'                              .END    START
```

```
EXAMPLE #1      MACRO X03.04 18-MAY-77 14:49:44 PAGE 5-2
SYMBOL TABLE

D2        000272R     INTX   = 040000     INT4   = 003000     ITAL1  = 000060     DFILE    000240R
EXIT      000142R     INT2   = 002400     LPON   = 000140     ITAT     000062R     INT7   = 003600
INT0    = 002000      LINE1  = 000005     MINUSX= 020000      INT5   = 003200     MAXY   = 001377
MAXSX   = 017600      DJMP   = 160000     MINUSY= 020000      I2       000170R    SHORTV= 104000
D3        000312R     LONGV  = 110000     POINT  = 114000     IPTR     000220R    D1       000252R
MAXSY   = 000077      LPDARK= 000300      EMSG     000214R    DSTAT  = 173420     STATSA= 170000
MISVY   = 000100      LINE2  = 000006     ITAL0  = 000040     SCBUF    000210R    STATSB= 174000
MSG       000224R     INT3   = 002600     T1       000174R    SYNC   = 000004     $VSCPL= ****** G
INT1    = 002200      RELATV= 130000      $VLPEN= ****** G    INT6   = 003400     GRAPHX= 120000
BLKON   = 000030      DRET   = 173400     BLKOFF= 000020      JSW    = 000044     GRAPHY= 124000
DHALT   = 173500      LINE3  = 000007     DJSR   = 173400     MAXX   = 001777     $VNSRT= ****** G
LINE0   = 000004      LBUF     000156R    $VRTLK= ****** G    START    000000R    LPOFF  = 000100
CHAR    = 100000      INCR   = 000100     DNAME  = 173520     $VUNLK= ****** G    MISVX  = 020000
DTABL     000200R     LPLITE= 000200      DNOP   = 164000

. ABS.    000000      000
          000326      001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  3564 WORDS  ( 14 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  64 PAGES
,LP:=VTMAC,MANEX1
```

```
EXAMPLE #2      MACRO X03.04 18-MAY-77 14:49:57 PAGE 5

      1                                        .TITLE  EXAMPLE #2
      2                               ;
      3                               ; THIS EXAMPLE USES THE TRACKING OBJECT AND THE TRACK
      4                               ; COMPLETION ROUTINE TO CAUSE A VECTOR TO FOLLOW
      5                               ; THE LIGHT PEN FROM A SET POINT AT (500,500).
      6                               ;
      7            000000                      R0=%0
      8            000001                      R1=%1
      9            000006                      SP=%6
     10            000007                      PC=%7
     11                                        .MCALL  .EXIT,.TTYIN,.PRINT
     12 000000                        START:   .LNKRT                          ;LINK TO MONITOR
     13 000004     100004                      BPL     1$                      ;LINK UP ERROR?
     14 000006                                 .PRINT  #EMSG                   ;YES, INFORM USER
     15 000014                                 .EXIT                           ;AND EXIT
     16 000016                        1$:      .INSRT  #DFILE                  ;INSERT DISPLAY FILE
     17 000026                                 .TRACK  #TBUF,#TCOM             ;DISPLAY TRACK OBJECT
     18 000042     004767   000006             JSR     PC,WAIT                 ;WAIT FOR <CR>
     19 000046                                 .UNLNK                          ;UNLINK FROM MONITOR
     20 000052                                 .EXIT
     21 000054                        WAIT:    .TTYIN                          ;GET CHAR. FROM TTY
     22 000060     022700   000012             CMP     #12,R0                  ;LINE FEED?
     23 000064     001373                      BNE     WAIT                    ;NO, GET ANOTHER
     24 000066     000207                      RTS     PC
     25 000070     000500   000500    TBUF:    .WORD   500,500                 ;TRACK BUFFER INITED TO
     26                                                                        ;START TRACK AT (500,500)
     27                               ;
     28                               ; TRACK COMPLETION ROUTINE ENTERED AT INTERRUPT LEVEL
     29                               ; FROM DISPLAY FILE HANDLER WITH DISPLAY STOPPED.
     30                               ; USED TO UPDATE DISPLAY FILE WITH DATA FROM TBUF.
     31                               ;
     32 000074     010146             TCOM:    MOV     R1,-(SP)                ;SAVE R1
     33 000076     016701   177766             MOV     TBUF,R1                 ;NEW X
     34 000102     166701   000052             SUB     OX,R1                   ;NEW X - OLD X
     35 000106     100003                      BPL     1$                      ;POSITIVE DIFFERENCE?
     36 000110     005401                      NEG     R1                      ;NO, SO MAKE POSITIVE
     37 000112     052701   020000             BIS     #MINUSX,R1              ;BUT SET MINUS BIT
     38 000116     052701   040000    1$:      BIS     #INTX,R1                ;ALSO SET INTENSIFY BIT
     39 000122     010167   000040             MOV     R1,DX                   ;THEN STORE IN DFILE.
     40 000126     016701   177740             MOV     TBUF+2,R1               ;NEW Y
     41 000132     166701   000024             SUB     OY,R1                   ;NEW Y - OLD Y
     42 000136     100003                      BPL     2$                      ;POSITIVE DIFFERENCE?
     43 000140     005401                      NEG     R1                      ;NO,SO MAKE POSITIVE
     44 000142     052701   020000             BIS     #MINUSX,R1              ;AND SET MINUS BIT
     45 000146     010167   000016    2$:      MOV     R1,DY                   ;THEN STORE IN DFILE
     46 000152     012601                      MOV     (SP)+,R1                ;RESTORE R1
     47 000154     000207                      RTS     PC                      ;EXIT FROM COMPLETION ROUTINE
     48                              -;
     49                               ; DISPLAY FILE FOR EXAMPLE #2
     50                               ;
     51 000156     114000             DFILE:   POINT                           ;SET POINT AT
     52 000160     000500             OX:      500
     53 000162     000500             OY:      500                             ;(500,500)
     54 000164     113000                      LONGV!INT4                      ;DRAW A VECTOR
     55 000166     000000             DX:      .WORD   0                       ;INITIALLY NOWHERE
     56 000170     000000             DY:      .WORD   0
     57 000172     173400                      DRET                            ;DISPLAY FILE END
```

```
EXAMPLE #2      MACRO X03.04 18-MAY-77 14:49:57 PAGE 5-1


    58 000174  000000                                0
    59 000176     123       117       122  EMSG:    .ASCIZ  /SORRY, THERE SEEMS TO BE A PROBLEM/
       000201     122       131       054
       000204     040       124       110
       000207     105       122       105
       000212     040       123       105
       000215     105       115       123
       000220     040       124       117
       000223     040       102       105
       000226     040       101       040
       000231     120       122       117
       000234     102       114       105
       000237     115       000
    60                                     .EVEN
    61                000000'                       .END    START



EXAMPLE #2      MACRO X03.04 18-MAY-77 14:49:57 PAGE 5-2
SYMBOL TABLE

INT0  = 002000         LONGV = 110000          LPLITE= 000200          $VDTLK= ****** G        $VUNLK= ****** G
MAXSX = 017600         LPDARK= 000300          WAIT    000054R         DNAME = 173520          DFILE   000150R
MAXSY = 000077         LINE2 = 000006          $VTRAK= ****** G        DNOP  = 164000          INT7  = 003600
MISVY = 000100         DX      000166R         INT4  = 003000          ITAL1 = 000060          MAXY  = 001377
INT1  = 002200         INT3  = 002600          LPON  = 000140          INT5  = 003200          SHORTV= 104000
BLKON = 000030         RELATV= 130000          MINUSX= 020000          DSTAT = 173420          STATSA= 170000

DHALT = 173500         TCOM    000074R         MINUSY= 020000          SYNC  = 000004          STATSB= 174000
LINE0 = 000004         DRET  = 173400          POINT = 114000          INT6  = 003400          GRAPHX= 120000
CHAR  = 100000         LINE3 = 000007          EMSG    000176R         MAXX  = 001777          GRAPHY= 124000
INTX  = 040000         DY      000170R         ITAL0 = 000040          OX      000160R         $VNSRT= ****** G
INT2  = 002400         TBUF    000070R         BLKOFF= 000020          START   000000R         LPOFF = 000100
LINE1 = 000005         INCR  = 000100          DJSR  = 173400          OY      000162R         MISVX = 020000
DJMP  = 160000

. ABS.  000000      000
        000242      001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  3717 WORDS  ( 15 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  64 PAGES
,LP:=VTMAC,MANEX2
```

APPENDIX B

SYSTEM MACRO LIBRARY

The following is a listing of the system macro library (SYSMAC.SML) for the RT-11 V03B release. This library is stored on the system device and is used by MACRO when it expands the programmed requests discussed in Chapter 2.

```
; SYSMAC.MAC--SYSTEM MACRO LIBRARY
;
; RT-11 VERSION 3B
;
; COPYRIGHT (C) 1977, 1978
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS. 01754
;
; 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 DEC.
;
; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
; SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.

.MACRO ..V1..
.MCALL  ...CM0,...CM1,...CM2,...CM3,...CM4,...CM5,...CM6
...V1=1
.ENDM

.MACRO ..V2..
.MCALL  ...CM0,...CM1,...CM2,...CM3,...CM4,...CM5,...CM6
...V1=2.
.ENDM

.MACRO .MACS
.MCALL  ...CM0,...CM1,...CM2,...CM3,...CM4,...CM5,...CM6
...V1=3.
.ENDM
```

```
.MACRO ...CM0 STARG
.IF B <STARG>
        CLR     -(6.)
.IFF
.IF IDN <STARG>,#0
        CLR     -(6.)
.IFF
        MOV     STARG,-(6.)
.ENDC
.ENDC
.ENDM

.MACRO ...CM1   AREA,IC,CHAN,FLAG
...CM5  <AREA>
...V2=0
.IF B <FLAG>
.IIF B <AREA>, ...V2=1
.IFF
.IIF DIF <FLAG>,SET, ...V2=1
.ENDC
.IF NE ...V2
.IF IDN <CHAN>,<#0>
        CLRB    (0)
.IFF
.IF NB <CHAN>
        MOVB    CHAN,(0)
.ENDC
.ENDC
.IFF
.IF B <CHAN>
        MOVB    #IC,1(0)
.IFF
.NTYPE ...V2,CHAN
.IF EQ ...V2-^027
        MOV     CHAN+<IC*^0400>,(0)
.IFF
        MOV     #IC*^0400,(0)
        MOVB    CHAN,(0)
.ENDC
.ENDC
.ENDC
.ENDM

.MACRO ...CM2   ARG,OFFSE,INS,CSET,BB
.IF B <ARG>
.IF NB <CSET>
.IF NE ...V1-3.
        CLR'BB  OFFSE(0)
.ENDC
.ENDC
.IFF
.IF IDN <ARG>,#0
        CLR'BB  OFFSE(0)
.IFF
        MOV'BB  ARG,OFFSE(0)
.ENDC
.ENDC
.IF NB <INS>
        EMT     ^0375
.ENDC
.ENDM

.MACRO ...CM3   CHAN,IC
.IF B <CHAN>
        MOV     #IC*^0400,%0
```

```
.IFF
.NTYPE ...V2,CHAN
.IF EQ ...V2-^027
        MOV     CHAN+<IC*^0400>,%0
.IFF
        MOV     #IC*^0400,%0
        BISB    CHAN,%0
.ENDC
.ENDC
        EMT     ^0374
.ENDM

.MACRO ...CM4   AREA,CHAN,BUF,WCNT,BLK,CRTN,IC,CODE
...CM1 <AREA>,<IC>,<CHAN>,<CODE>
...CM2 <BLK>,2.
...CM2 <BUF>,4.
...CM2 <WCNT>,6.
...CM2 <CRTN>,8.,X
.ENDM

.MACRO ...CM5   SRC,BB
.IF NB <SRC>
.IF DIF <SRC>,R0
        MOV'BB  SRC,%0
.ENDC
.ENDC
.ENDM

.MACRO ...CM6   AREA,IC,CHAN,FLAG
...CM5  <AREA>
.IF B <FLAG>
.IF NB <AREA>
        MOV     #IC*^0400+CHAN,(0)
.ENDC
.IFF
.IF IDN <FLAG>,SET
        MOV     #IC*^0400+CHAN,(0)
.ENDC
.ENDC
.ENDM

.MACRO .CDFN    AREA,ADDR,NUM,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,13.,0,<CODE>
...CM2  <ADDR>,2.
...CM2  <NUM>,4.,X
.ENDM

.MACRO .CHAIN
        MOV     #8.*^0400,%0
        EMT     ^0374
.ENDM

.MACRO .CHCOP   AREA,CHAN,OCHAN,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM1  <AREA>,11.,<CHAN>,<CODE>
...CM2  <OCHAN>,2.,X
.ENDM
```

```
.MACRO .CLOSE   CHAN
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
        EMT     ^O<160+CHAN>
.IFF
...CM3 <CHAN>,6.
.ENDC
.ENDM

.MACRO .CNTXS   AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,27.,0,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .CMKT    AREA,ID,TIME,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,19.,0,<CODE>
...CM2  <ID>,2.
...CM2  <TIME>,4.,X,X
.ENDM

.MACRO .CRAW    AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,30.,2.,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .CRRG    AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,30.,0,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .CSIGE   DEVSPC,DEFEXT,CSTRNG,LINBUF
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF NB <LINBUF>
...CM0  <LINBUF>
.NTYPE  ...V2,DEVSPC
.IF EQ ...V2-^O27
...CM0  <DEVSPC'+1>
.IFF
...CM0  <DEVSPC>
        INC     (6.)
.ENDC
.IFF
```

```
...CM0  <DEVSPC>
.ENDC
...CM0  <DEFEXT>
...CM0  <CSTRNG>
        EMT     ^O344
.ENDM

.MACRO .CSISP   OUTSPC,DEFEXT,CSTRNG,LINBUF
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF NB <LINBUF>
...CM0  <LINBUF>
.NTYPE  ...V2,OUTSPC
.IF EQ ...V2-^O27
...CM0  <OUTSPC'+1>
.IFF
...CM0  <OUTSPC>
        INC     (6.)
.ENDC
.IFF
...CM0  <OUTSPC>
.ENDC
...CM0  <DEFEXT>
...CM0  <CSTRNG>
        EMT     ^O345
.ENDM

.MACRO .CSTAT   AREA,CHAN,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM1  <AREA>,23.,<CHAN>,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .CTIMI   TBK
        JSR     %5,@$TIMIT
        .WORD   TBK-.
        .WORD   1
.ENDM

.MACRO .DATE
        MOV     #10.*^O400,%0
        EMT     ^O374
.ENDM

.MACRO .DELET   AREA,CHAN,DBLK,SEQNUM,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
...CM5  <CHAN>
        EMT     ^O<AREA>
.IFF
...CM5  <AREA>
.IF IDN <CHAN>,#0
        CLR     (0)
.IFF
...V2=0
.IF B <CODE>
```

```
.IIF B <AREA>, ...V2=1
.IFF
.IIF DIF <CODE>,SET, ...V2=1
.ENDC
.IF NE ...V2
.IF NB <CHAN>
        MOVB    CHAN,(0)
.ENDC
.IFF
.IF B <CHAN>
        CLRB    1(0)
.IFF
.NTYPE ...V2,CHAN
.IF EQ ...V2-^O27
        MOV     CHAN,(0)
.IFF
        CLR     (0)
        MOVB    CHAN,(0)
.ENDC
.ENDC
.ENDC
.ENDC
...CM2  <DBLK>,2.
...CM2  <SEQNUM>,4.,X,X
.ENDC
.ENDM

.MACRO .DEVIC   AREA,ADDR,LINK,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF B LINK
...CM6  <AREA>,12.,0,<CODE>
.IFF
...CM6  <AREA>,12.,1,<CODE>
.ENDC
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .DRAST   NAME,PRI,ABT
.GLOBL  $INPTR
.IIF B <ABT>    RTS     %7
.IIF NB <ABT>   BR      ABT
NAME'INT:: JSR  %5,@$INPTR
        .WORD   ^C<PRI*^O40>&^O340
.ENDM

.MACRO .DRBEG   NAME,VEC,DSIZ,DSTS,VTBL
.IF NDF $SYSDV
.ASECT
. = 52
.GLOBL  NAME'END
        .WORD   <NAME'END - NAME'STRT>
        .WORD   DSIZ
        .WORD   DSTS
.PSECT
.IFF
$SYDSZ == DSIZ
.PSECT  SYSHND
.ENDC
NAME'STRT::
.IF B VTBL
```

```
.GLOBL  NAME'INT
        .WORD   VEC
        .WORD   NAME'INT - .
.IFF
.GLOBL  VTBL,NAME'INT
        .WORD   <VTBL-.>/2. -1 + ^0100000
        .WORD   NAME'INT - .
.ENDC
        .WORD   ^0340
NAME'SYS::
NAME'LQE::      .WORD   0
NAME'CQE::      .WORD   0
.ENDM

.MACRO .DREND   NAME
...V2=0
.IF NE MMG$T
...V2=...V2+2.
.IF DF $SYSDV
.GLOBL $RELOC,$MPPHY,$GETBYT,$PUTBYT,$PUTWRD
$RLPTR:: .WORD  $RELOC
$MPPTR:: .WORD  $MPPHY
$GTBYT:: .WORD  $GETBYT
$PTBYT:: .WORD  $PUTBYT
$PTWRD:: .WORD  $PUTWRD
.IFF
$RLPTR:: .WORD  0
$MPPTR:: .WORD  0
$GTBYT:: .WORD  0
$PTBYT:: .WORD  0
$PTWRD:: .WORD  0
.ENDC
.ENDC
.IF NE ERL$G
...V2=...V2+1
.IF DF $SYSDV
.GLOBL  $ERLOG
$ELPTR:: .WORD  $ERLOG
.IFF
$ELPTR:: .WORD  0
.ENDC
.ENDC
.IF NE TIM$IT
...V2=...V2+4.
.IF DF $SYSDV
.GLOBL  $TIMIO
$TIMIT:: .WORD  $TIMIO
.IFF
$TIMIT:: .WORD  0
.ENDC
.ENDC
.IF DF $SYSDV
.GLOBL $FORK,$INTEN
$INPTR:: .WORD  $INTEN
$FKPTR:: .WORD  $FORK
.IFF
$INPTR:: .WORD  0
$FKPTR:: .WORD  0
.IFTF
.GLOBL  NAME'STRT
NAME'END == .
.IFT
$SYHSZ == NAME'END - NAME'STRT
.IFF
```

```
.ASECT
.=60
          .WORD    ...V2
.PSECT
.ENDC
.ENDM

.MACRO .DRFIN    NAME
.GLOBL   NAME'CQE
         MOV      %7,%4
         ADD      #NAME'CQE-.,%4
         MOV      @#^054,%5
         JMP      @^0270(5)
.ENDM

.MACRO .DSTAT    RETSPC,DNAM
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM5   <DNAM>
...CM0   <RETSPC>
         EMT      ^0342
.ENDM

.MACRO .ELAW     AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6   <AREA>,30.,3.,<CODE>
...CM2   <ADDR>,2.,X
.ENDM

.MACRO .ELRG     AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6   <AREA>,30.,1,<CODE>
...CM2   <ADDR>,2.,X
.ENDM

.MACRO .ENTER    AREA,CHAN,DBLK,LEN,SEQNUM,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
...CM5   <CHAN>
...CM0   <DBLK>
         EMT      ^O<40+AREA>
.IFF
...CM1   <AREA>,2.,<CHAN>,<CODE>
...CM2   <DBLK>,2.
...CM2   <LEN>,4.,,X
...CM2   <SEQNUM>,6.,X,X
.ENDC
.ENDM

.MACRO .EXIT
         EMT      ^0350
.ENDM
```

```
.MACRO .FETCH   ADDR,DNAM
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM5  <DNAM>
...CM0  <ADDR>
        EMT     ^0343
.ENDM

.MACRO .FORK    FKBLK
        JSR     %5,@$FKPTR
        .WORD   FKBLK - .
.ENDM

.MACRO .GMCX    AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,30.,6.,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .GTIM    AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,17.,0,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .GTJB    AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,16.,0,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .GTLIN   LINBUF,PROMPT
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM0  <LINBUF>
...CM0  #1
...CM0  <PROMPT>
        CLR     -(6.)
        EMT     ^0345
.ENDM

.MACRO .GVAL    AREA,OFFSE,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,28.,0,<CODE>
...CM2  <OFFSE>,2.,X
.ENDM
```

```
.MACRO .HERR
        MOV     #5.*^0400,%0
        EMT     ^0374
.ENDM

.MACRO .HRESE
        EMT     ^0357
.ENDM

.MACRO .INTEN   PRIO,PIC
.IF B PIC
        JSR     5.,@^054
.IFF
        MOV     @#^054,-(6.)
        JSR     5.,@(6.)+
.ENDC
        .WORD   ^C<PRIO*32.>&224.
.ENDM

.MACRO .LOCK
        EMT     ^0346
.ENDM

.MACRO .LOOKU   AREA,CHAN,DBLK,SEQNUM,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
...CM5  <CHAN>
        EMT     ^O<20+AREA>
.IFF
...CM1  <AREA>,1,<CHAN>,<CODE>
...CM2  <DBLK>,2.
...CM2  <SEQNUM>,4.,X,X
.ENDC
.ENDM

.MACRO .MAP     AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,30.,4.,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .MTATC   AREA,ADDR,UNIT,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,31.,5.,<CODE>
...CM2  <ADDR>,2.
...CM2  <UNIT>,4.,X,,B
.ENDM

.MACRO .MTDTC   AREA,UNIT,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,31.,6.,<CODE>
...CM2  <UNIT>,4.,X
.ENDM
```

```
.MACRO .MTPRN   AREA,ADDR,UNIT,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  AREA,31.,7.,<CODE>
...CM2  ADDR,2.
...CM2  <UNIT>,4.,X,,B
.ENDM

.MACRO .MFPS    ADDR
        MOV     @#^054,-(6.)
        ADD     #^0362,(6.)
        JSR     7.,@(6.)+
.IIF NB <ADDR>  MOVB    (6.)+,ADDR
.ENDM

.MACRO .MTRCT   AREA,UNIT,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,31.,4.,<CODE>
...CM2  <UNIT>,4.,X
.ENDM

.MACRO .MRKT    AREA,TIME,CRTN,ID,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,18.,0,<CODE>
...CM2  <TIME>,2.
...CM2  <CRTN>,4.
...CM2  <ID>,6.,X
.ENDM

.MACRO .MTGET   AREA,ADDR,UNIT,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  AREA,31.,1,<CODE>
...CM2  ADDR,2.
...CM2  <UNIT>,4.,X,,B
.ENDM

.MACRO .MTPS    ADDR
.IIF NB <ADDR>  CLR     -(6.)
.IIF NB <ADDR>  MOVB    ADDR,(6.)
        MOV     @#^054,-(6.)
        ADD     #^0360,(6.)
        JSR     7.,@(6.)+
.ENDM

.MACRO .MTSET   AREA,ADDR,UNIT,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  AREA,31.,0,<CODE>
...CM2  ADDR,2.
...CM2  <UNIT>,4.,X,,B
.ENDM
```

```
.MACRO .MTIN    AREA,ADDR,UNIT,CHRCNT,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  AREA,31.,2.,<CODE>
...CM2  ADDR,2.
...CM2  <UNIT>,4.,,,B
...CM2  <CHRCNT>,5.,X,,B
.ENDM

.MACRO .MTOUT   AREA,ADDR,UNIT,CHRCNT,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  AREA,31.,3.,<CODE>
...CM2  ADDR,2.
...CM2  <UNIT>,4.,,,B
...CM2  <CHRCNT>,5.,X,,B
.ENDM

.MACRO .MWAIT
        MOV     #9.*^0400,%0
        EMT     ^0374
.ENDM

.MACRO .PRINT   ADDR
.IF NB <ADDR>
.IF DIF <ADDR>,R0
        MOV     ADDR,%0
.ENDC
.ENDC
        EMT     ^0351
.ENDM

.MACRO .PROTE   AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,25.,0,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .PURGE   CHAN
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM3  <CHAN>,3.
.ENDM

.MACRO .QELDF
Q.LINK=0
Q.CSW=2.
Q.BLKN=4.
Q.FUNC=6.
Q.JNUM=7.
Q.UNIT=7.
Q.BUFF=^O10
Q.WCNT=^O12
Q.COMP=^O14
.IF EO MMG$T
```

```
Q.ELGH=^O16
.IFF
Q.PAR=^O16
Q.ELGH=^O24
.ENDC
.ENDM

.MACRO .QSET    ADDR,LEN
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM5  <LEN>,B
...CM0  <ADDR>
        EMT     ^O353
.ENDM

.MACRO .RCTRL
        EMT     ^O355
.ENDM

.MACRO .RCVD    AREA,BUF,WCNT,CRTN=#1,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IIF IDN <CODE>,NOSET, ...CM4 <AREA>,,<BUF>,<WCNT>,,<CRTN>,22.,<CODE>
.IIF DIF <CODE>,NOSET, ...CM4 <AREA>,#0,<BUF>,<WCNT>,,<CRTN>,22.,<CODE>
.ENDM

.MACRO .RCVDC   AREA,BUF,WCNT,CRTN,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IIF IDN <CODE>,NOSET, ...CM4 <AREA>,,<BUF>,<WCNT>,,<CRTN>,22.,<CODE>
.IIF DIF <CODE>,NOSET, ...CM4 <AREA>,#0,<BUF>,<WCNT>,,<CRTN>,22.,<CODE>
.ENDM

.MACRO .RCVDW   AREA,BUF,WCNT,CRTN=#0,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IIF IDN <CODE>,NOSET, ...CM4 <AREA>,,<BUF>,<WCNT>,,<CRTN>,22.,<CODE>
.IIF DIF <CODE>,NOSET, ...CM4 <AREA>,#0,<BUF>,<WCNT>,,<CRTN>,22.,<CODE>
.ENDM

.MACRO .RDBBK   RGSIZ
.MCALL  .RDBDF
.RDBDF
        .WORD
        .WORD   RGSIZ
        .WORD
.ENDM

.MACRO .RDBDF
R.GID   =0
R.GSIZ  =2.
R.GSTS  =4.
R.GLGH  =6.
RS.CRR  =^O100000
RS.UNM  =^O40000
RS.NAL  =^O20000
.ENDM
```

```
.MACRO .READ    AREA,CHAN,BUF,WCNT,BLK,CRTN=#1,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
...CM5  <WCNT>
...CM0  #1
...CM0  <BUF>
...CM0  <CHAN>
        EMT     ^O<200+AREA>
.IFF
...CM4 <AREA>,<CHAN>,<BUF>,<WCNT>,<BLK>,<CRTN>,8.,<CODE>
.ENDC
.ENDM

.MACRO .READC   AREA,CHAN,BUF,WCNT,CRTN,BLK,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
...CM5  <CRTN>
...CM0  <WCNT>
...CM0  <BUF>
...CM0  <CHAN>
        EMT     ^O<200+AREA>
.IFF
...CM4 <AREA>,<CHAN>,<BUF>,<WCNT>,<BLK>,<CRTN>,8.,<CODE>
.ENDC
.ENDM

.MACRO .READW   AREA,CHAN,BUF,WCNT,BLK,CRTN=#0,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
...CM5  <WCNT>
...CM0
...CM0  <BUF>
...CM0  <CHAN>
        EMT     ^O<200+AREA>
.IFF
...CM4 <AREA>,<CHAN>,<BUF>,<WCNT>,<BLK>,<CRTN>,8.,<CODE>
.ENDC
.ENDM

.MACRO .REGDEF
.ENDM

.MACRO .RELEA   DNAM
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM5  <DNAM>
...CM0
        EMT     ^O343
.ENDM

.MACRO .RENAM   AREA,CHAN,DBLK,CODE
.IF NDF ...V1
.MCALL .MACS
```

```
.MACS
.ENDC
.IF EQ ...V1-1
...CM5  <CHAN>
        EMT     ^O<100+AREA>
.IFF
...CM1  <AREA>,4.,<CHAN>,<CODE>
...CM2  <DBLK>,2.,X
.ENDC
.ENDM

.MACRO .REOPE   AREA,CHAN,CBLK,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
...CM5  <CHAN>
        EMT     ^O<140+AREA>
.IFF
...CM1  <AREA>,6.,<CHAN>,<CODE>
...CM2  <CBLK>,2.,X
.ENDC
.ENDM

.MACRO .SAVES   AREA,CHAN,CBLK,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
...CM5  <CHAN>
        EMT     ^O<120+AREA>
.IFF
...CM1  <AREA>,5.,<CHAN>,<CODE>
...CM2  <CBLK>,2.,X
.ENDC
.ENDM

.MACRO .RSUM
        MOV     #2.*^O400,%0
        EMT     ^O374
.ENDM

.MACRO .SDAT    AREA,BUF,WCNT,CRTN=#1,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IIF IDN <CODE>,NOSET, ...CM4 <AREA>,,<BUF>,<WCNT>,,<CRTN>,21.,<CODE>
.IIF DIF <CODE>,NOSET, ...CM4 <AREA>,#0,<BUF>,<WCNT>,,<CRTN>,21.,<CODE>
.ENDM

.MACRO .SDATC   AREA,BUF,WCNT,CRTN,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IIF IDN <CODE>,NOSET, ...CM4 <AREA>,,<BUF>,<WCNT>,,<CRTN>,21.,<CODE>
.IIF DIF <CODE>,NOSET, ...CM4 <AREA>,#0,<BUF>,<WCNT>,,<CRTN>,21.,<CODE>
.ENDM

.MACRO .SDATW   AREA,BUF,WCNT,CRTN=#0,CODE
.IF NDF ...V1
.MCALL .MACS
```

```
.MACS
.ENDC
.IIF IDN <CODE>,NOSET, ...CM4 <AREA>,,<BUF>,<WCNT>,,<CRTN>,21.,<CODE>
.IIF DIF <CODE>,NOSET, ...CM4 <AREA>,#0,<BUF>,<WCNT>,,<CRTN>,21.,<CODE>
.ENDM

.MACRO .SERR
        MOV     #4.*^0400,%0
        EMT     ^0374
.ENDM

.MACRO .SETTO   ADDR
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM5  <ADDR>
        EMT     ^0354
.ENDM

.MACRO .SCCA    AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,29.,0,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .SFPA    AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,24.,0,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .SPFUN   AREA,CHAN,FUNC,BUF,WCNT,BLK,CRTN,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM1  <AREA>,26.,<CHAN>,<CODE>
...CM2  <BLK>,2.
...CM2  <BUF>,4.
...CM2  <WCNT>,6.
.IF NB FUNC
.NTYPE ...V2,FUNC
.IF NE ...V2-^027
.IIF DIF <CODE>,NOSET,...CM2    #^0377,8.,,,B
...CM2  <FUNC>,9.,,,B
.IFF
...CM2  <FUNC'*^0400+^0377>,8.
.ENDC
.ENDC
...CM2  <CRTN>,10.,X,X
.ENDM

.MACRO .SRESE
        EMT     ^0352
.ENDM
```

```
.MACRO .SPND
        MOV     #1*^0400,%0
        EMT     ^0374
.ENDM

.MACRO .SYNCH   AREA,PIC
.IF B PIC
.IIF NB <AREA>  MOV     AREA,%4
.IFF
.IF NB AREA
        MOV     %7,%4
        ADD     #AREA-.,%4
.ENDC
.ENDC
        MOV     @#^054,%5
        JSR     5.,@^0324(5.)
.ENDM

.MACRO .TIMIO   TBK,HI,LO
        JSR     %5,@$TIMIT
        .WORD   TBK-.
        .WORD   0
        .WORD   HI
        .WORD   LO
.ENDM

.MACRO .TLOCK
        MOV     #7.*^0400,%0
        EMT     ^0374
.ENDM

.MACRO .TRPSE   AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,3.,0,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .TTINR
        EMT     ^0340
.ENDM

.MACRO .TTYIN   CHAR
        EMT     ^0340
        BCS     .-2.
.IF NB <CHAR>
.IF DIF <CHAR>,R0
        MOVB    %0,CHAR
.ENDC
.ENDC
.ENDM

.MACRO .TTOUT
        EMT     ^0341
.ENDM

.MACRO .TTYOU   CHAR
.IF NB <CHAR>
.IF DIF <CHAR>,R0
        MOVB    CHAR,%0
.ENDC
```

```
.ENDC
        EMT     ^O341
        BCS     .-2.
.ENDM

.MACRO .TWAIT   AREA,TIME,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,20.,0,<CODE>
...CM2  <TIME>,2.,X
.ENDM

.MACRO .UNLOC
        EMT     ^O347
.ENDM

.MACRO .UNMAP   AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,30.,5.,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .UNPRO   AREA,ADDR,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
...CM6  <AREA>,25.,1,<CODE>
...CM2  <ADDR>,2.,X
.ENDM

.MACRO .WAIT    CHAN
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
        EMT     ^O<240+CHAN>
.IFF
.IF B <CHAN>
        CLR     %0
.IFF
.NTYPE ...V2,CHAN
.IF EQ ...V2-^O27
.IF IDN <CHAN>,#0
        CLR     %0
.IFF
        MOV     CHAN,%0
.ENDC
.IFF
        CLR     %0
        BISB    CHAN,%0
.ENDC
.ENDC
        EMT     ^O374
.ENDC
.ENDM
```

```
.MACRO .WDBBK   WNAPR,WNSIZ,WNRID,WNOFF,WNLEN,WNSTS
.MCALL  .WDBDF
.WDBDF
        .BYTE
        .BYTE   WNAPR
        .WORD
        .WORD   WNSIZ
        .WORD   WNRID
        .WORD   WNOFF
        .WORD   WNLEN
        .WORD   WNSTS
.ENDM

.MACRO .WDBDF
W.NID   =0
W.NAPR  =1
W.NBAS  =2.
W.NSIZ  =4.
W.NRID  =6.
W.NOFF  =^O10
W.NLEN  =^O12
W.NSTS  =^O14
W.NLGH  =^O16
WS.CRW  =^O100000
WS.UNM  =^O40000
WS.ELW  =^O20000
WS.MAP  =^O400
.ENDM

.MACRO .WRITC   AREA,CHAN,BUF,WCNT,CRTN,BLK,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
...CM5  <CRTN>
...CM0  <WCNT>
...CM0  <BUF>
...CM0  <CHAN>
        EMT     ^O<220+AREA>
.IFF
...CM4 <AREA>,<CHAN>,<BUF>,<WCNT>,<BLK>,<CRTN>,9.,<CODE>
.ENDC
.ENDM

.MACRO .WRITE   AREA,CHAN,BUF,WCNT,BLK,CRTN=#1,CODE
.IF NDF ...V1
.MCALL .MACS
.MACS
.ENDC
.IF EQ ...V1-1
...CM5  <WCNT>
...CM0  #1
...CM0  <BUF>
...CM0  <CHAN>
        EMT     ^O<220+AREA>
.IFF
...CM4 <AREA>,<CHAN>,<BUF>,<WCNT>,<BLK>,<CRTN>,9.,<CODE>
.ENDC
.ENDM

.MACRO .WRITW   AREA,CHAN,BUF,WCNT,BLK,CRTN=#0,CODE
.IF NDF ...V1
.MCALL .MACS
```

```
.MACS
.ENDC
.IF EQ ...V1-1
...CM5 <WCNT>
...CM0
...CM0 <BUF>
...CM0 <CHAN>
        EMT     ^O<220+AREA>
.IFF
...CM4 <AREA>,<CHAN>,<BUF>,<WCNT>,<BLK>,<CRTN>,9.,<CODE>
.ENDC
.ENDM
```

APPENDIX C

ADDITIONAL I/O INFORMATION

This appendix provides some additional information on I/O processing that is useful especially to users who need to write their own device handlers. It contains the I/O data structure formats, a flowchart of the sequence of events involved in queued I/O processing, and source listings of two RT-11 device handlers with liberal comments. In addition, this appendix provides information on device directory formats and file structures.

Before writing a device handler, programmers should be familiar with the material in Chapter 1 of this manual. RT-11 provides macros to make handler writing easier; Chapter 1 describes these macros. Appendix B contains a listing of the RT-11 system macro library. It can be helpful to consult the library listing in order to understand how the macros expand and, therefore, how use them correctly.

Programmers should have a thorough knowledge of the hardware device for which they are writing the handler. The PDP-11 Peripherals Handbook contains information on DIGITAL peripherals. The hardware manuals and engineering prints are the most complete source of information for DIGITAL devices and those from other manufacturers.

C.1 I/O Data Structures

RT-11 I/O data structures are described in this section. These data structures provide conventions for communication among an application program, the monitor, and a device handler.

C.1.1 Monitor Device Tables

Tables in the Resident Monitor keep track of the devices on the RT-11 system. These tables are contained in the module SYSTBL, which is created by system generation and which is assembled separately from the module RMON. SYSTBL is linked with RMON and other modules to form the resident monitor. The symbol $SLOT, which is defined at system generation time, defines the maximum number of devices the system can have.

C.1.1.1 **$PNAME Table** - The permanent name table is called $PNAME. It is the central table around which all the others are constructed. The total number of entries is fixed at assembly time. Extra slots can be allocated at assembly time. Entries are made in $PNAME at monitor assembly time for each device that is built into the system. Free slots can be created by deleting or renaming one or more of the device handler files from the system device and rebooting the system, or by issuing the REMOVE keyboard monitor command. The INSTALL keyboard monitor command can be used to install a different device handler into the table after the system has been booted. INSTALL does not make a device entry permanent. The DEV macro in SYSTBL must be used to permanently add a device to the system. The DEV macro is described in Section C.1.1.7.

Each table entry consists of a single word that contains the Radix-50 code for the 2-character physical device name. For example, the entry for DECtape is .RAD50 /DT/. The TT device must be first in the table. After that, the position of a device in this table is not critical. Once the entries are made into this table, their relative position (that is, their order in the table) determines the general device index used in various places in the monitor. Thus, the other tables are organized in the same order as $PNAME. The offset of a device name entry in $PNAME serves as the index into the other tables for a given device.

The bootstrap checks the system generation parameters of a handler with those of the current monitor, and zeroes the $PNAME entry for that device if the parameters do not match. INSTALL cannot install a handler whose conditional parameters do not match those of the monitor.

C.1.1.2 **$STAT Table** - The device status table is called $STAT. Entries to this table are made at assembly time for those devices that are built into the RT-11 system. When the system is bootstrapped, the entries for those devices that are built into the system are updated with information in the handler files that are present on the system device. The system device handler does not have to be present on the system device as a separate .SYS file because it is already a part of the monitor. Entries are made for devices that are not built into the system at assembly time when they are installed with the INSTALL monitor command. Each device in the system must have a status entry in its corresponding slot in $STAT. The device status word identifies each physical device and provides information about it, such as whether it is random or sequential access. Figure C-1 shows the meaning of the bits in the status word. For a user-written handler, the programmer sets up the device status word according to the layout in Figure C-1 so it can be stored in block 0 of the handler file. Figures C-10 and C-12, below, show examples of the device status word as it is set up in device handlers. The device status word is part of the information returned to a running program by the .DSTATUS programmed request.

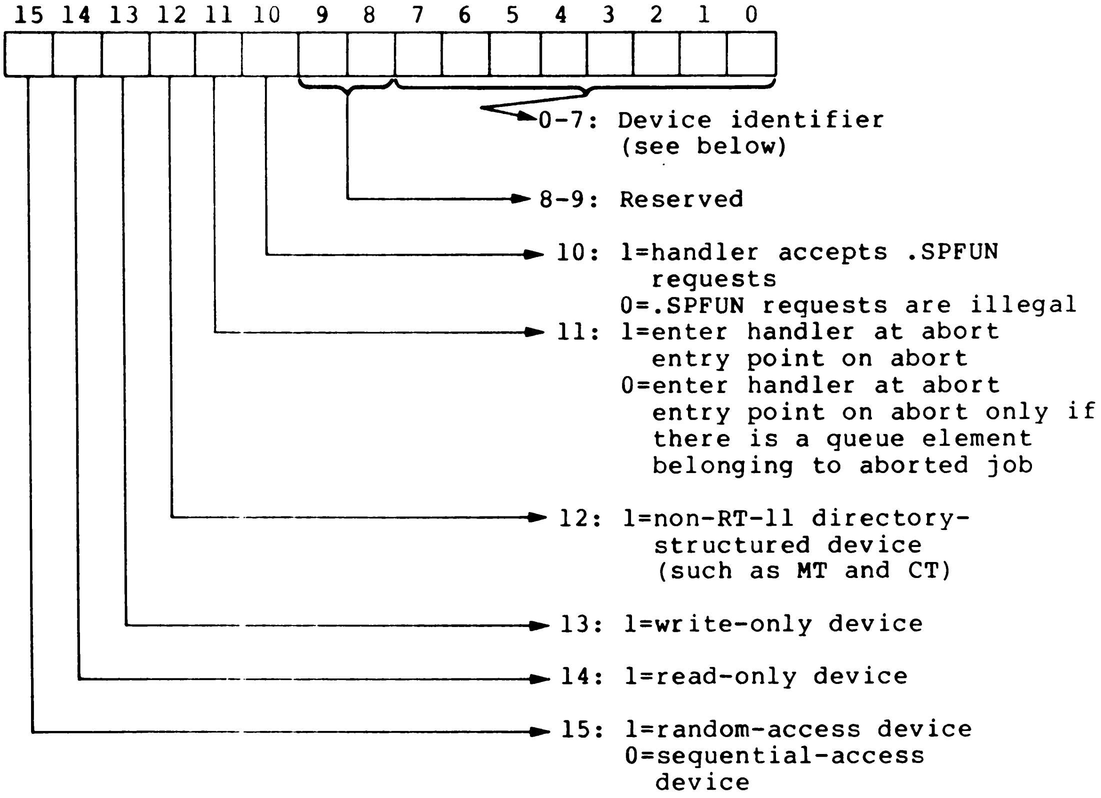

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0-7: Device identifier (see below)
8-9: Reserved
10: 1=handler accepts .SPFUN requests
0=.SPFUN requests are illegal
11: 1=enter handler at abort entry point on abort
0=enter handler at abort entry point on abort only if there is a queue element belonging to aborted job
12: 1=non-RT-11 directory-structured device (such as MT and CT)
13: 1=write-only device
14: 1=read-only device
15: 1=random-access device
0=sequential-access device

Figure C-1 Device Status Word

Note that bit 11 in the status word should be set only for device handlers that remove the queue element on entry and queue internally.

All device handlers that have bit 15 set are assumed to be RT-11 file-structured devices by most system utility programs.

In RT-11, symbolic names are defined for certain bit patterns. This provides a meaningful way to refer to the bits in the device status word. The SYSTBL source file defines the following bit patterns:

FILST$	=	100000
RONLY$	=	40000
WONLY$	=	20000
SPECL$	=	10000
HNDLR$	=	4000
SPFUN$	=	2000

A programmer can first use direct assignment statements to set up the symbolic names for the bit patterns, as shown above. Then the device status word can easily be constructed by adding the device identifier (described below) to the appropriate bit patterns, according to the following outline:

```
      .WORD     device identifier + symbol
```

An example of this is the way the RT-11 code in the file SYSTBL.MAC sets up the device status word for device DX:

```
      .WORD     22 + FILST$ + SPFUN$
```

See Section C.1.1.7 for more information on the DEV macro in SYSTBL.

The device-identifier byte uniquely identifies each device in the system. The values are currently defined in octal as follows:

0 =	RK05 disk
1 =	TC11 DECtape
2 =	reserved
3 =	line printer
4 =	console terminal or batch handler
5 =	RL01 disk
6 =	RX02 diskette
7 =	PC11 high-speed paper tape reader and punch
10 =	reserved
11 =	magtape
12 =	RF11 disk
13 =	TA11 cassette
14 =	card reader (CR11,CM11)
15 =	reserved
16 =	RJS03/4 fixed-head disks
17 =	reserved
20 =	TJU16 magtape
21 =	RP02/RP03 disk
22 =	RX01 diskette
23 =	RK06/RK07 disk
24 =	error log handler
25 =	null handler
26-30 =	reserved (for Networks)
31-33 =	reserved (for DIBOL LQ, LR, LS)
34 =	TU58 data cartridge

To create device identifier codes for devices that are not already supported by RT-11, programmers should start by using code 377 (octal) for the first new device, 376 for the second, and so on. This procedure should avoid conflict with codes that RT-11 will use in the future for new hardware devices.

C.1.1.3 **$DVREC Table** - The device handler block number table is called $DVREC. Entries to this table are made at bootstrap time for devices that are built into the system, and at INSTALL time for additional devices. The entries are the absolute block numbers where each of the device handlers resides on the system device. Since handlers are treated as files, their positions on the system device are not necessarily fixed. Thus, each time the system is bootstrapped, the handlers are located and $DVREC is updated with their locations on the system device. The pointer in $DVREC points to block 1 of the file. (Because handlers are linked at 1000, the actual handler code starts in the second block of the file.) A zero entry in the $DVREC table indicates that no handler for the device in that slot was found on the system device. (Note that if block 0 of the handler file resides on a bad block on the system device, RT-11 cannot install or fetch the handler.) Note that 0 is a valid $DVREC entry for permanently resident devices.

C.1.1.4 **$ENTRY Table** - The handler entry point table is called $ENTRY. Entries in this table are made whenever a handler is loaded into memory by either the .FETCH programmed request or by the LOAD keyboard monitor command. The entry for each device is a pointer to the fourth word of the device handler in memory. The entry is zeroed when the handler is removed by the .RELEASE programmed request or by the UNLOAD keyboard monitor command.

Some device handlers are permanently resident. These include the system device handler and, for FB and XM systems, the TT: handler. The $ENTRY values for such devices are fixed at boot time.

C.1.1.5 **$UNAM1 and $UNAM2 Tables** - The tables that keep track of logical device names and the physical names that are assigned to them are called $UNAM1 and $UNAM2. Entries are made in these tables when the ASSIGN monitor command is issued. The physical device name is stored in $UNAM1 and the logical name associated with it is stored in the corresponding slot in $UNAM2. When the system is first bootstrapped, there are two assignments already in effect. These assignments associate the logical names DK: and SY: with the device from which the system was booted. The value of $SLOT limits the total number of logical name assignments (excluding SY and DK).

The $UNAM1 and $UNAM2 tables are not indexed by the $PNAME table offset. The fact that the tables are the same size is interesting, but not significant.

C.1.1.6 **$OWNER Table** - The device ownership table is called $OWNER. It is used in the FB and XM environments to arbitrate device ownership. The table is ($SLOT*2) words in length and is divided into 2-word entries for each device. Entries are made into this table when the LOAD keyboard monitor command is issued. Each 2-word entry is in turn divided into eight 4-bit fields capable of holding a job number. The low four bits of the first byte correspond to unit 0, and the high four bits correspond to unit 1. The low four bits of the next byte correspond to unit 2, and so on. Thus, each device is presumed to have up to eight units, each assigned independently of the others. However, if the device is nonfile-structured, units are not assigned independently: the monitor ASSIGN code ensures that ownership of all units is assigned to one job.

When either a background or a foreground job attempts to access a particular unit of a device, the monitor checks to be sure the unit being accessed is either public or belongs to the requesting job. If the other job owns the unit, a fatal error is generated.

The device is assumed to be public if the 4-bit field is 0. If the device is not public, the field contains a code equal to the job number plus 1. Since job numbers are always even, the ownership code is odd. Bit 0 of the field being set indicates that the unit ownership is assigned to a job (1 for the background job and 3 for the foreground job).

C.1.1.7 **Adding a Device to the Tables** - The DEV macro in SYSTBL.MAC is used to define devices in the system. The format of the DEV macro is as follows:

```
     DEV name,s,type
```

The arguments in the macro shown above have the following meaning:

name — represents the two-character physical device name, such as RK or DX.

s — represents the device status word. This word consists of a device identification code plus a set of device characteristics bits from the following set:

```
FILST$ = 100000
RONLY$ =  40000
WONLY$ =  20000
SPECL$ =  10000
HNDLR$ =   4000
SPFUN$ =   2000
```

type — must be SYS if the device can be a system device. A device can be a system device if it is random-access and file-structured.

Examples of the DEV macro as used in SYSTBL are as follows:

```
DEV RK,0+FILST$,SYS

DEV LP,3+WONLY$

DEV MT,11+SPECL$+SPFUN$
```

C.1.2 The Low Memory Protection Bitmap

RT-11 maintains a bitmap that reflects the protection status of low memory, locations 0 through 477. This map is required in order to avoid conflicts in the use of the vectors. In FB and XM, the .PROTECT programmed request allows a program to gain exclusive control of a vector or a set of vectors. When a vector is protected, the bitmap is updated to indicate which words are protected. If a word in low memory is not protected, it is loaded from block 0 of the executable file. If a word in low memory is protected, it is not loaded from block 0 of the file. In addition, if the word is protected by a foreground job, it is not destroyed when a new background program is run.

The bitmap is a 20 (decimal) byte table that starts 326 (octal) bytes from the beginning of the Resident Monitor. Table C-1 lists the offset from RMON and the corresponding locations represented by that byte.

Table C-1
Low Memory Bitmap

Offset	Locations (octal)	Offset	Locations (octal)
326	0-17	340	240-257
327	20-37	341	260-277
330	40-57	342	300-317
331	60-77	343	320-337
332	100-117	344	340-357
333	120-137	345	360-377
334	140-157	346	400-417
335	160-177	347	420-437
336	200-217	350	440-457
337	220-237	351	460-477

Each byte in the table reflects the status of 8 words of memory. The first byte in the table controls locations 0 through 17, the second byte controls locations 20 through 37, and so on. The bytes are read from left to right. Thus, if locations 0 through 3 are protected, the first byte of the table contains:

```
11000000
```

NOTE

> Only individual words are protected, not bytes. Thus, protecting word 0 means that both locations 0 and 1 are protected.

If locations 24 and 26 are protected, the second byte of the table contains:

```
00110000
```

The leftmost bit represents location 20 and the rightmost bit represents location 36. To protect locations 300 through 306, the leftmost four bits of the byte at offset 342 must be set to result in a value of 360 for that byte:

```
11110000
```

The SJ monitor does not support the .PROTECT programmed request. If users need to protect vectors, they should use one of the two following methods:

1. Use PATCH to manually modify the bitmap

2. Dynamically modify the bitmap from within a running program

For example, to protect locations 300 through 306 dynamically, the following instructions can be used:

```
MOV  @#54,R0
BISB #^B11110000,342(R0)
```

Protecting locations with PATCH means that the vector is permanently protected, even if the system is rebootstrapped. The dynamic method provides a temporary measure and does not remain effective across bootstraps. Users are cautioned that the dynamic method involves storing data directly into the monitor. For this reason, it is recommended that SJ users use PATCH to protect vectors.

C.1.3 Queue Elements

The RT-11 system uses queues to organize requests in a first-in/first-out order. Requests for I/O transfers, completion routines, and timer routines are queued for later service. Each request uses one queue element. The elements are arranged in linked lists so that they are processed in order. Each element contains all the information necessary to initiate and process a single request. Foreground requests are added to an I/O queue in front of background requests. However, a foreground request cannot replace an active background request (the current queue element).

C.1.3.1 **I/O Queue Element** - Once a device handler is in memory, any .READ/.WRITE programmed request for the corresponding device is interpreted by the monitor and translated into a call to the I/O device handler. To facilitate the overlapping of I/O and computation, all I/O requests in RT-11 are processed through an I/O queue.

The RT-11 I/O queue is made up of one linked list of queue elements for each resident device handler. I/O queue elements are seven words long for SJ and FB systems, and ten words long for XM systems. RT-11 provides one queue element in the Resident Monitor for the SJ environment. For the FB and XM environments, each job has one queue element in its impure area. This is sufficient for any program that uses wait mode I/O (.READW/.WRITW). However, for maximum throughput, the .QSET programmed request should be used at the beginning of a program to create one additional queue element for each asynchronous I/O request that can be outstanding. Then, asynchronous I/O should be used.

If an I/O transfer is requested and a queue element is not available, RT-11 must wait until an element is free before it can queue the request. This obviously slows program execution. If the program requires asynchronous I/O, it must allocate extra queue elements. It is always sufficient to allocate N new queue elements, where N is the maximum number of pending requests that can be outstanding at any time in a particular program. This produces a total of N+1 available elements, since the element in the job's impure area is added to the list of available elements.

Figure C-2 shows the format of an I/O queue element and the meaning of each entry. The .QELDF macro defines symbolic names for the offsets from the beginning of the I/O queue element and a symbolic name for the size of the queue element. Figure C-2 also shows the offsets and the symbolic name that is associated with each offset.

Note that .QELDF defines offsets from the beginning of the queue element. From within a device handler, the pointer to the current queue element points to the third word of the element. Therefore, the offsets from .QELDF cannot be used directly to access words in the queue element. The following example from the PC handler illustrates a construction that is typically used in handlers to account for this discrepancy:

```
    BUFF = Q.BUFF - Q.BLKN
```

Name	Offset	Contents			
Q.LINK	0	Link to next queue element; 0 if none			
Q.CSW	2	Pointer to channel status word in I/O channel (see Figure C-7)			
Q.BLKN	4	Physical block number			
Q.FUNC Q.UNIT Q.JNUM	6 7 7	reserved (1 bit)	Job Number (4 bits) 0 = BG 2 = FG	Device Unit (3 bits)	Special Function Code (8 bits)
Q.BUFF	10	User buffer address (mapped through PAR1 with Q.PAR value, if XM)			

Figure C-2 I/O Queue Element Format

Name	Offset	Contents

Q.WCNT	12	Word count if <0, operation is WRITE
		if =0, operation is SEEK
		if >0, operation is READ
		The true word count is the absolute value of this word.

Q.COMP	14	Completion routine code if 0, this is wait mode I/O if 1, just queue the request and return if even, completion routine address
Q.PAR	16	PAR1 Relocation Bias (XM only)
		reserved (XM only)
		reserved (DECnet)

Figure C-2 I/O Queue Element Format (Cont.)

Q.LINK, the link to the next queue element, points to the third word of the next queue element, not to its first word.

Q.LINK and Q.CSW are 16-bit physical addresses. They are always used in kernel mode, and therefore must always be in the lower 28K words of memory.

In XM systems, Q.BUFF is always an address between 20000 and 37777. To access the byte in the user's physical memory, the monitor loads PAR1 (Page Address Register 1 of the KT11 memory management hardware) with the Q.PAR values and then uses Q.BUFF as a pointer to the correct byte.

C.1.3.2 **Timer Queue Element** - Another queue maintained by the monitor is the timer queue. This queue is used to implement the .MRKT time and .TIMIO requests, which schedule completion routines to be entered after a specified period of time.

Figure C-3 shows the format of a timer queue element. It includes the symbolic names and offsets as well as the contents of each word in the data structure. Note that time is stored as a 2-word number, a modified expression of the number of ticks until the timed wait expires. (There are sixty ticks per second when 60 Hz power is used, and 50 ticks per second when 50 Hz power is used.) The timer queue elements are stored in the queue in order of their expiration times. An optional sequence number can be added to the request to distinguish it from others issued by the same job.

The monitor uses the timer queue internally to implement the .TWAIT programmed request. The .TWAIT request causes the issuing job to be suspended. A timer request is placed in the queue with the .RSUM programmed request logic as the completion routine. This causes execution to wait until the desired time has elapsed. Then execution resumes when the monitor itself issues the .RSUM programmed request.

A range of owner's sequence number IDs is reserved for use by DIGITAL software. All values in the range from 177400 through 177777 are reserved for DIGITAL. These values should not be used by customer programs.

There are several uses for system timer elements. If C.SYS is -1, the element is being used for either multi-terminal time-out support, or for device handler time-out support. If C.SYS is -3, the element is being used to implement a .TWAIT request in the XM monitor.

In XM, completion routines that have -1 in C.SYS are run in kernel mode and the queue element is discarded. That is, the queue element is not linked into the list of available elements. If C.SYS is -3, the completion routine is still run in kernel mode. However, the queue element is linked into the user's available queue when the completion routine is run. (The timer queue element is used as the completion queue element.) In all other cases, the queue element is linked into the available queue and completion routines run in user mode.

Name	Offset	Contents
C.HOT	0	High order time
C.LOT	2	Low order time
C.LINK	4	Link to next queue element; 0 if none
C.JNUM	6	Owner's job number
C.SEQ	10	Owner's sequence number ID
C.SYS	12	**-1 if system timer element** **-3 if .TWAIT element in XM**
C.COMP	14	Address of completion routine

Figure C-3 Timer Queue Element Format

C.1.3.3 **Completion Queue Element** - The FB and XM monitors maintain one queue of I/O completion requests for each job. When an I/O transfer completes, the I/O queue element indicates whether or not a completion routine was specified in the request. If the seventh word of the I/O queue element is even and nonzero, a completion routine was requested. The queue completion logic in the monitor transfers the I/O request queue element to the completion queue. It places the channel status word and channel offset in the element. This has the effect of serializing completion routines, rather than nesting them. Elements are also added to this queue when a timer request expires and when a .SYNCH request is issued. The completion queue is a first-in/first-out queue. The completion routines are entered at priority level 0 rather than at interrupt level. In SJ, completion routines can interrupt each other. In FB and XM, no other code except interrupts can execute when a completion routine is running.

Note that completion routines are not serialized in the SJ environment, because there is no completion queue in SJ. Completion routines in SJ do not run in a first-in/first-out order. They are executed as soon as the I/O or timer request is complete.

Figure C-4 shows the format of a completion queue element. It includes the symbolic names and offsets as well as the contents of each word in the data structure.

Name	Offset	Contents
Q.LINK	0	Link to next queue element; 0 if none
	2	Undefined
	4	Undefined
	6	Undefined
Q.BUFF	10	Channel status word
Q.WCNT	12	Channel offset
Q.COMP	14	Completion routine address

Figure C-4 Completion Queue Element Format

C.1.3.4 **Synch Queue Element** - In the FB and XM monitors the .SYNCH request makes use of the completion queue. When the .SYNCH programmed request is entered, the 7-word area supplied with the request is linked into the head of the completion queue, where it appears to be a request for a completion routine. The .SYNCH request then does an interrupt exit. The completion queue manager next calls the code following the .SYNCH request at priority level 0 (after a possible context switch to bring in the correct job). To prevent the .SYNCH block from the user's program from being linked in the queue of available queue elements after the routine exits, the sixth word is set to -1. The completion queue manager checks the sixth word before linking a queue element back into the list of available elements, and skips elements with -1 there.

In the SJ monitor, the .SYNCH request sets up the registers, drops priority to 0, and calls the code following the request as a co-routine. When the co-routine returns, the .SYNCH logic does an interrupt exit.

Figure C-5 shows the format of a synch queue element. It includes the symbolic names and offsets as well as the contents of each word in the data structure.

Name	Offset	Contents
Q.LINK	0	Link to next queue element; 0 if none
Q.CSW	2	Job number
Q.BLKN	4	Undefined
Q.FUNC	6	Undefined
Q.BUFF	10	Synch ID
Q.WCNT	12	-1
Q.COMP	14	Synch routine address

Figure C-5 Synch Queue Element Format

C.1.3.5 **Fork Queue Element** - The RT-11 system maintains one fork queue. Its root is in the Resident Monitor. The device handler must provide a 4-word fork block that will be used as the fork queue element. Section 1.4.4.1 in this manual describes the .FORK macro.

Figure C-6 shows the format of a fork queue element. It includes the symbolic names and offsets as well as the contents of each word in the data structure.

Name	Offset	Contents
F.BLNK	0	Link to next queue element; 0 if none
F.BADR	2	Fork routine address
F.BR5	4	R5 save area
F.BR4	6	R4 save area

Figure C-6 Fork Queue Element Format

C.1.4 I/O Channel Format

Figure C-7 shows the format of an I/O channel. Since each channel uses five words, the size of the monitor's channel area is five times the number of channels. RT-11 allocates 16 channels for each job. The channel area is 80 (decimal) words long. For SJ, a single channel area is located in RMON. For FB and XM, one channel area for each job is located in the job's impure area. The .CDFN programmed request can provide more channels.

<table>
<tr><th>Name</th><th>Offset</th><th colspan="2">Contents</th></tr>
<tr><td></td><td>0</td><td colspan="2">Channel status word</td></tr>
<tr><td>C.SBLK</td><td>2</td><td colspan="2">Starting block number of this file
(0 if nonfile structured)</td></tr>
<tr><td>C.LENG</td><td>4</td><td colspan="2">Length of file (if opened by .LOOKUP);
Size of empty area (if opened by .ENTER)</td></tr>
<tr><td>C.USED</td><td>6</td><td colspan="2">Actual data length (if .LOOKUP);
Highest block written (if .ENTER)</td></tr>
<tr><td>C.DEVQ</td><td>10</td><td>Device unit number</td><td>Number of requests pending on this channel</td></tr>
</table>

Figure C-7 I/O Channel Description

Figure C-8 shows the significant bits in the channel status word.

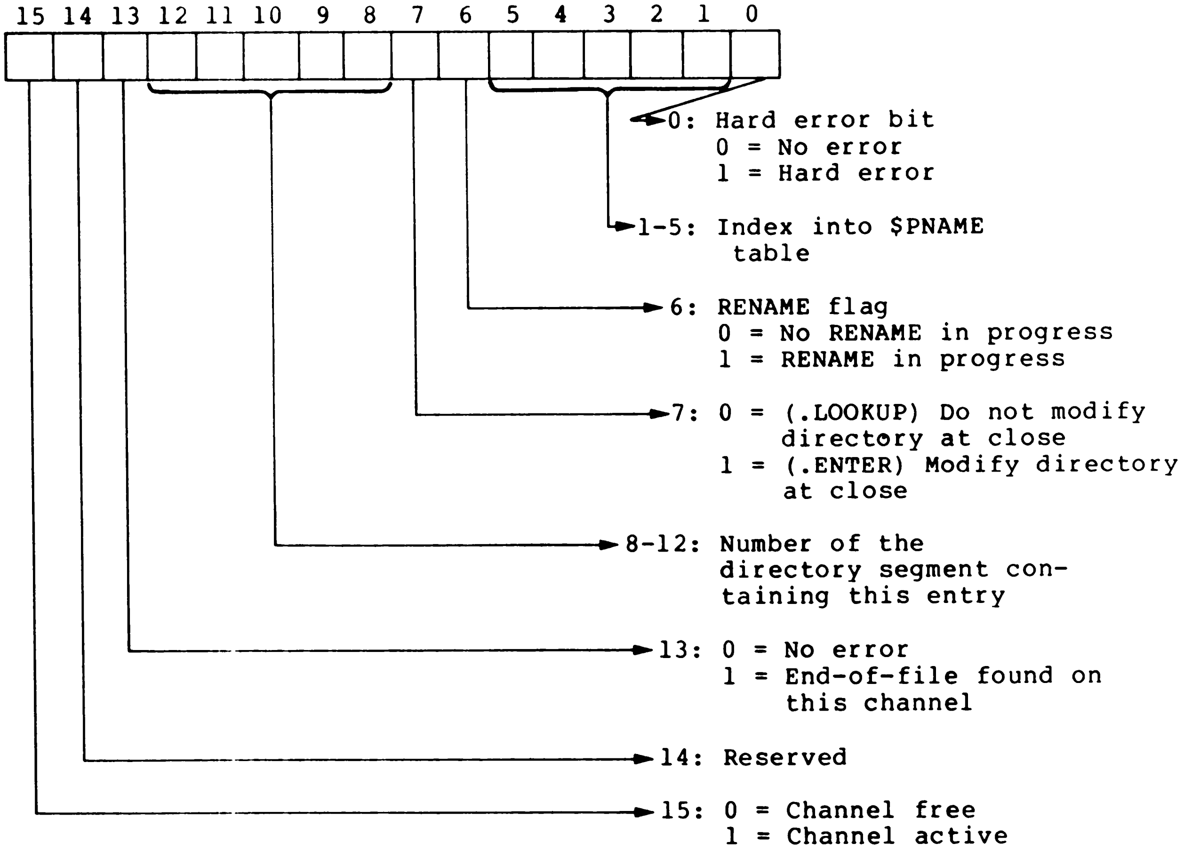
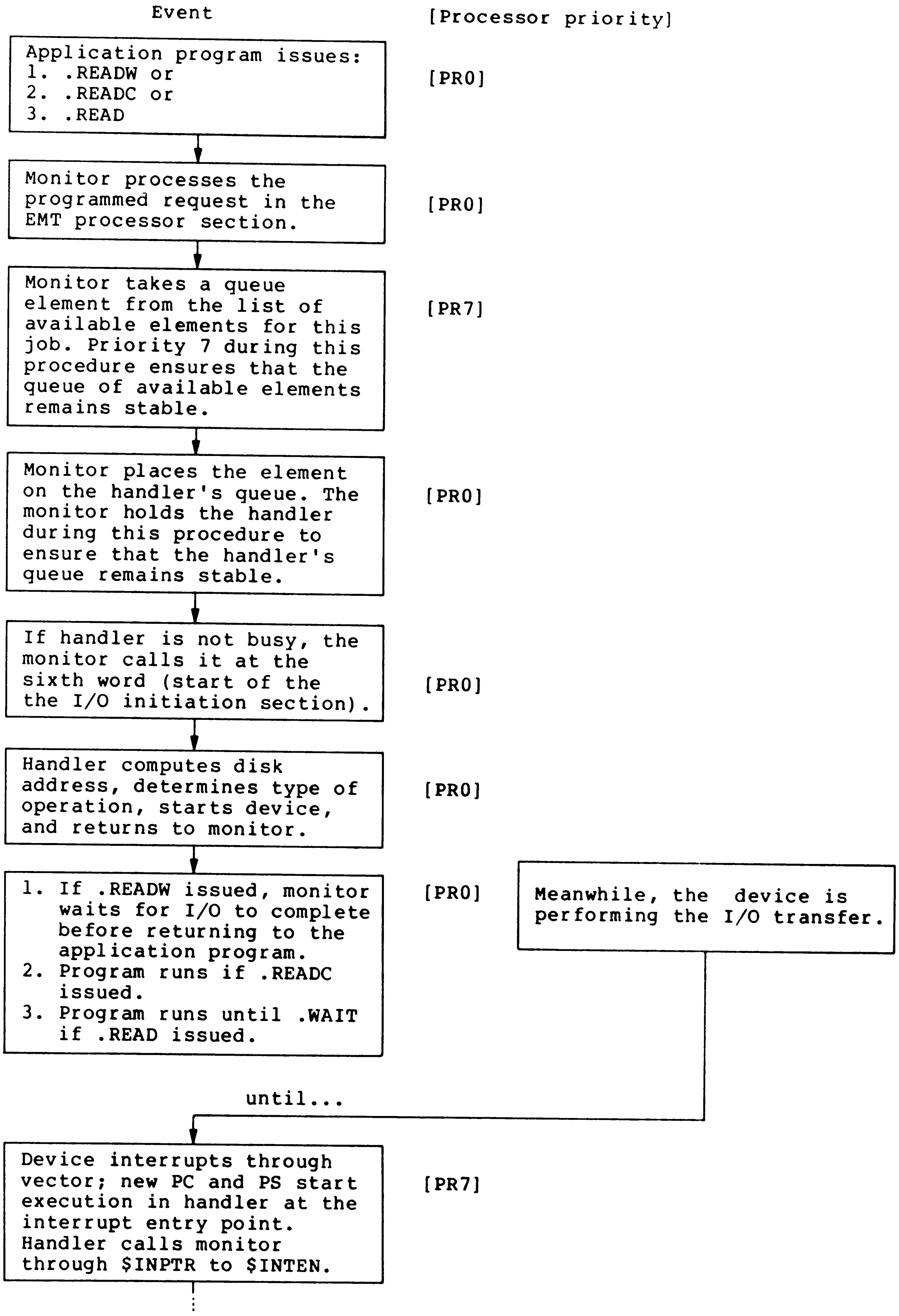

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0: Hard error bit
0 = No error
1 = Hard error
1-5: Index into $PNAME table
6: RENAME flag
0 = No RENAME in progress
1 = RENAME in progress
7: 0 = (.LOOKUP) Do not modify directory at close
1 = (.ENTER) Modify directory at close
8-12: Number of the directory segment containing this entry
13: 0 = No error
1 = End-of-file found on this channel
14: Reserved
15: 0 = Channel free
1 = Channel active

Figure C-8 Channel Status Word

C.2 Flow of Events in I/O Processing

Figure C-9 shows a simplified diagram of the flow of events involved in an I/O transfer. The following example, a read request to the RK disk handler, shows the relationship between the application program and the queue elements, and between the queue elements and the device handler. The flow of events for a non-DMA device is slightly different. (Figure C-12 shows a device handler for a non-DMA device, the paper tape reader and punch.)

This simplified diagram assumes that no other interrupts occur during this processing, and that the FB monitor is being used.



Event

[Processor priority]

Application program issues:
1. .READW or
2. .READC or
3. .READ

[PR0]

Monitor processes the programmed request in the EMT processor section.

[PR0]

Monitor takes a queue element from the list of available elements for this job. Priority 7 during this procedure ensures that the queue of available elements remains stable.

[PR7]

Monitor places the element on the handler's queue. The monitor holds the handler during this procedure to ensure that the handler's queue remains stable.

[PR0]

If handler is not busy, the monitor calls it at the sixth word (start of the the I/O initiation section).

[PR0]

Handler computes disk address, determines type of operation, starts device, and returns to monitor.

[PR0]

1. If .READW issued, monitor waits for I/O to complete before returning to the application program.
2. Program runs if .READC issued.
3. Program runs until .WAIT if .READ issued.

[PR0]

Meanwhile, the device is performing the I/O transfer.

until...

Device interrupts through vector; new PC and PS start execution in handler at the interrupt entry point. Handler calls monitor through $INPTR to $INTEN.

[PR7]

Figure C-9 Flow of Events in I/O Processing

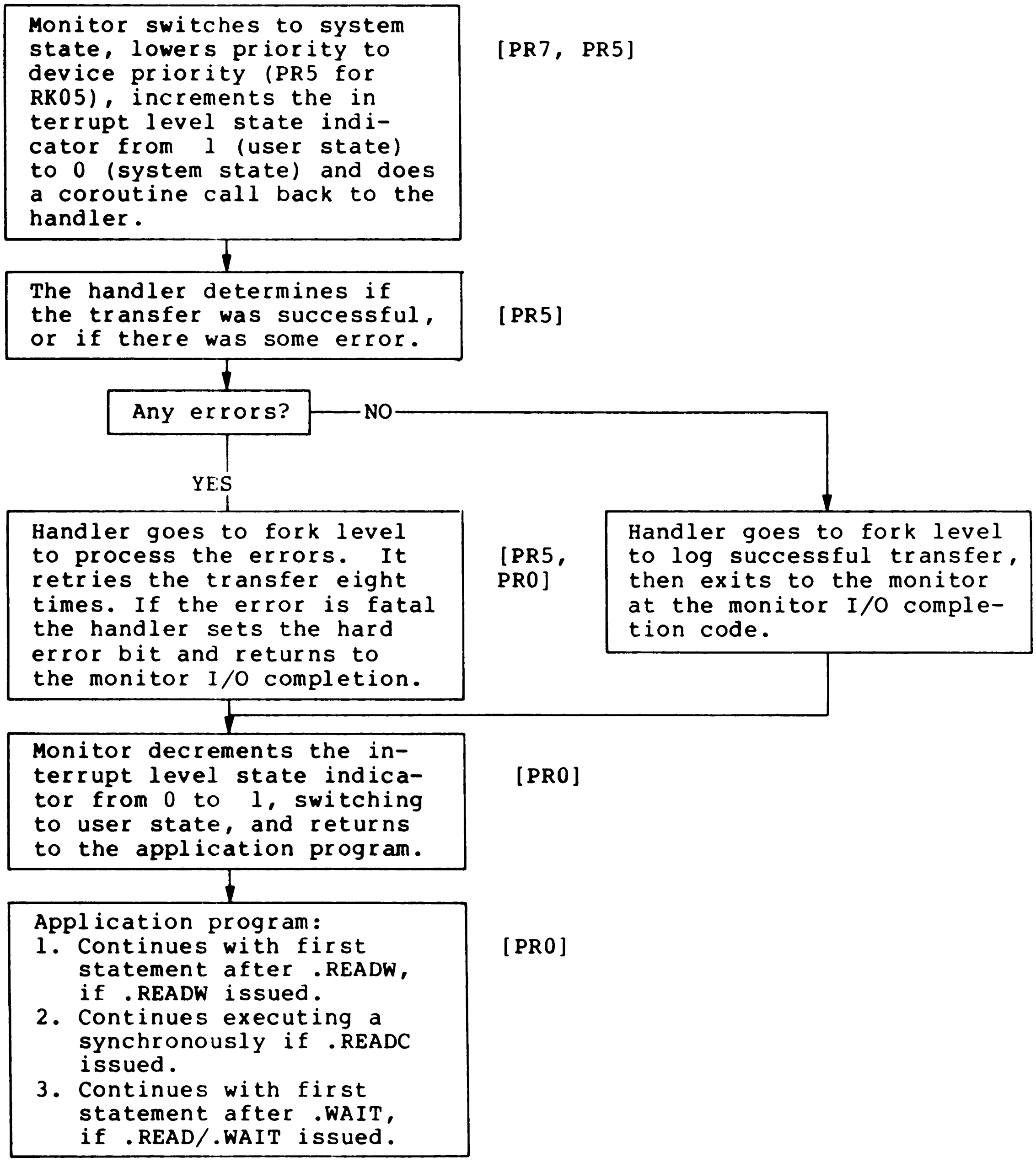

Monitor switches to system state, lowers priority to device priority (PR5 for RK05), increments the in terrupt level state indicator from 1 (user state) to 0 (system state) and does a coroutine call back to the handler.
[PR7, PR5]
The handler determines if the transfer was successful, or if there was some error.
[PR5]
Any errors?
NO
YES
Handler goes to fork level to process the errors. It retries the transfer eight times. If the error is fatal the handler sets the hard error bit and returns to the monitor I/O completion.
[PR5, PR0]
Handler goes to fork level to log successful transfer, then exits to the monitor at the monitor I/O completion code.
Monitor decrements the interrupt level state indicator from 0 to 1, switching to user state, and returns to the application program.
[PR0]
Application program:
1. Continues with first statement after .READW, if .READW issued.
2. Continues executing a synchronously if .READC issued.
3. Continues with first statement after .WAIT, if .READ/.WAIT issued.
[PR0]

Figure C-9 Flow of Events in I/O Processing (Cont.)

C.3 Study of the RK05 Handler

Figure C-10 provides a listing of the assembled RK05 handler file. The comments give a detailed explanation of the handler. The RK05 handler was chosen as a representative handler for a random access disk that can be a system device. For this example, the RK handler was assembled as a data device only. See Section C.4 for information on system device handlers.

In Figure C-10, black ink is used for text and comments. Red ink is used for the actual computer output of the RK05 handler assembly listing.

Device handlers are written in position independent code, called PIC. The PDP-11 processors offer addressing modes that make it possible to write instructions that are not dependent on the virtual addresses to which they are linked. A body of such code is termed position independent, and can be loaded and executed at any virtual address. (See Appendix G, "Writing Position Independent Code", in the PDP-11 MACRO-11 Language Reference Manual, order number AA-5075A-TC.) Throughout the RK05 handler listing, coding constructions that were used specifically to make the handler position independent are marked as [PIC].

This listing was produced by assembling the conditional file RKCND.MAC together with the RK handler source file, RK.MAC. The command strings to produce this assembly and the listing file RK.LST are as follows:

Keyboard monitor command:

```
.MACRO/LIST:RK.LST/NOOBJECT/SHOW:ME:MEB:TTM RKCND.MAC+RK.MAC
```

MACRO program commands:

```
.R MACRO
*,RK.LST/L:ME:MEB:TTM=RKCND.MAC,RK.MAC
```

The first file listed below, RKCND.MAC, was created especially for this example. It was assembled together with the handler source file, RK.MAC, to produce code for the three system generation conditions: memory management, error logging, and device time-out. Normally, a device handler is assembled with the system conditional file, SYCND.MAC, to ensure that the handler has the same system generation parameters as does the current monitor.

This listing was produced by assembling the conditional file RKCND.MAC together with the RK handler source file, RK.MAC. The command strings to produce this assembly and the listing file RK.LST are as follows:

Keyboard monitor command:

```
.MACRO/LIST:RK.LST/NOOBJECT/SHOW:ME:MEB:TTM RKCND.MAC+RK.MAC
```

MACRO program commands:

```
.R MACRO
*,RK.LST/L:ME:MEB:TTM=RKCND.MAC,RK.MAC
```

The first file listed below, RKCND.MAC, was created especially for this example. It was assembled together with the handler source file, RK.MAC, to produce code for the three system generation conditions: memory management, error logging, and device time-out. Normally, a device handler is assembled with the system conditional file, SYCND.MAC, to ensure that the handler has the same system generation parameters as does the current monitor.

```
RK05  V03.01 MACRO V03.02B6-SEP-78 11:55:53 PAGE 1

                      ;CONDITIONAL FILE FOR RK HANDLER EXAMPLE
                      ;
                      ;RKCND.MAC
                      ;
                      ;9/1/78 JAD
                      ;
                      ;ASSEMBLE WITH RK.MAC TO TURN ON 18-BIT I/O,
                      ;TIME-OUT SUPPORT, AND ERROR LOGGING FOR
                      ;RK HANDLER
                      ;
          000001  MMG$T   = 1            ;TURN ON 18-BIT I/O
          000001  ERL$G   = 1            ;TURN ON ERROR LOGGING
          000001  TIM$IT  = 1            ;TURN ON TIME-OUT SUPPORT
```

The listing of the RK handler source file that follows is current for RT-11 V03B; it includes one source patch. Comments that are part of the source file itself are all upper-case characters and begin with a semicolon (;). Comments that were added as documentation in this appendix are upper- and lower-case characters.

Figure C-10 RK05 Handler Listing

```
RK05  V03.01 MACRO V03.02B6-SEP-78 11:55:53 PAGE 2

                ;RK EDIT LEVEL 0
                .TITLE  RK05  V03.01
                .IDENT  /V03.01/
                ; RT-11 DISK (RK11) HANDLER
                ;
                ; COPYRIGHT (C) 1978
                ;
                ; DIGITAL EQUIPMENT CORPORATION
                ; MAYNARD, MASSACHUSETTS  01754
                ;
                ; 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 DEC.
                ;
                ; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO
                ; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
                ; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
                ;
                ; DEC ASSUMES NO RESPONSIBILITY FOR THE USE
                ; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
                ; WHICH IS NOT SUPPLIED BY DEC.
```

```
RK05  V03.01 MACRO V03.02B6-SEP-78 11:55:53 PAGE 3

                .ENABL LC
```

****

The device handler Preamble Section starts here.

****

```
                .MCALL  .DRBEG,.DREND,.FORK,.DRAST,.DRFIN,.QELDF
```

Each macro that is used in the handler requires the .MCALL statement, as shown above. The .QELDF, .DRBEG, .DRAST, .DRFIN, and .DREND macros are provided in the system macro library in order to simplify writing a device handler.

Figure C-10 RK05 Handler Listing (Cont.)

```
     4                  ; SYSTEM GENERATION OPTIONS:
```

The code in this handler contains many conditional assembly directives. They test for the presence or absence of time-out support, extended memory support, and error logging. Code is generated differently depending on which of those system generation options are present in the system. When a system is generated, the handler files are assembled together with SYCND.MAC, the system conditional file, so that the correct conditionals are defined in the handler files. If a handler is to be added to an existing system, it should be assembled with the same conditional file that was used for the rest of the system. If there is no conditional file assembled with the handler file, the conditionals are turned off by the following three lines of code (for the purpose of this example, the three following conditionals were set to 1 by the preceding file, RKCND.MAC):

```
     5                  .IIF NDF TIM$IT,TIM$IT=0 [No device time-out support]
     6                  .IIF NDF MMG$T, MMG$T=0  [No memory management]
     7                  .IIF NDF ERL$G, ERL$G=0  [No error logging]
     8
     9                  .NLIST CND
```

For the purpose of this listing, printing of conditional source lines is suppressed within the expansion of system macros. This is accomplished by the .NLIST CND and .LIST CND pair of directives.

```
    10 000000           .QELDF
```

The .QELDF macro defines symbolic offsets into the I/O queue elements. See Figure C-2 above for a diagram of the I/O queue element.

```
            000000  Q.LINK=0      [Link to next queue element]
            000002  Q.CSW=2.      [Pointer to channel status word]
            000004  Q.BLKN=4.     [Physical block number]
            000006  Q.FUNC=6.     [Special function code]
            000007  Q.JNUM=7.     [Job number]
            000007  Q.UNIT=7.     [Device unit number]
            000010  Q.BUFF=^O10   [User virtual memory buffer address]
            000012  Q.WCNT=^O12   [Word count]
            000014  Q.COMP=^O14   [Completion routine code]
            000016  Q.PAR=^O16    [PAR1 relocation bias]
            000024  Q.ELGH=^O24   [End of queue element, used to find length]
```

Figure C-10 RK05 Handler Listing (Cont.)

```
                .LIST CND

```

The following direct assignment statements are required only if the handler can be a system device. For this example the RK handler was assembled as a mass storage device only, and not as a system device. Therefore, the symbol $RKSYS in SYCND.MAC was not set to 1. It does not cause a problem to leave the assignment statements in place if the handler is being assembled only as a storage device and not as a system device. The globals being defined here are the entry points for all the other system devices in the RT-11 system.

```
        000000  DTSYS   == 0                            ;THIS IS RK HANDLER
        000000  DLSYS   == 0
        000000  DSSYS   == 0
        000000  DXSYS   == 0
        000000  DPSYS   == 0
        000000  RFSYS   == 0
        000000  DMSYS   == 0
        000000  DYSYS   == 0

                ; RK CONTROL DEFINITIONS:
```

The next two statements define the vector and CSR addresses for the RK device, if they have not already been defined in the system conditional file, SYCND.MAC. The default vector is 220; the default CSR address is 177400.

```
                .IIF NDF RK$VEC, RK$VEC == 220
                .IIF NDF RK$CSR, RK$CSR == 177400
```

The following group of direct assignment statements set up the device control registers. The device control register names, locations, and operation codes can be found in the PDP-11 Peripherals Handbook and in the hardware manual for the device.

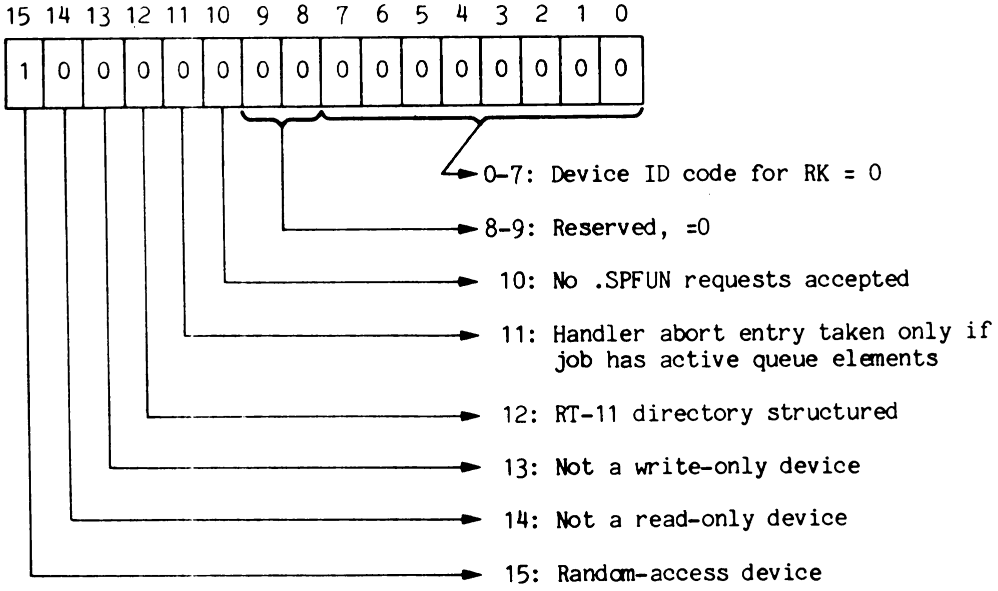
```
        177400  RKDS    = RK$CSR    [Drive Status Register]
        177402  RKER    = RKDS+2    [Error Register]
        177404  RKCS    = RKDS+4    [Control Status Register]
        177406  RKWC    = RKDS+6    [Word Count Register]
        177410  RKBA    = RKDS+10   [Current Bus Address Register]
        177412  RKDA    = RKDS+12   [Disk Address Register]
                [RKDB, the Data Buffer Register, is not used]

```

Figure C-10 RK05 Handler Listing (Cont.)

The symbol RKCNT represents the number of times to retry an I/O transfer should an error occur.

```
        000010  RKCNT   = 10                    ;# ERROR RETRYS

```

The device status word RKSTS and the device size word RKDSIZ are set up here. The information they contain is used by the .DSTATUS programmed request, which returns the information to a running program. See Figure C-1 for the format of the device status word. The diagram below shows how the code 100000 was selected for the RK device status word.


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0-7: Device ID code for RK = 0
8-9: Reserved, =0
10: No .SPFUN requests accepted
11: Handler abort entry taken only if job has active queue elements
12: RT-11 directory structured
13: Not a write-only device
14: Not a read-only device
15: Random-access device

```
        100000  RKSTS   = 100000                ;DEVICE SYSTEM STATUS
                                                ;WORD ($STAT)
        011300  RKDSIZ  = 11300                 ;DEVICE BLOCK SIZE ($DVSIZ)
```

The next four direct assignment statements are for error logging.

```
        000000  RKIDEN  = 0                     ;RK11 ID = 0 IN HIGH BYTE
                                                ;FOR ERROR LOG
        000377  RKIDS   = 377                   ;RK11 DEVICE ID = 0 IN HIGH
                                                ;BYTE
                                                ;-1 IN LOW BYTE FOR I/O
                                                ;SUCCESS TO ERROR LOG
```

Figure C-10 RK05 Handler Listing (Cont.)

```
          004000  RKRCNT  = 4000                    ;I/O RETRY COUNT IN HIGH BYTE
          000007  RKNREG  = 7                       ;# OF REGISTERS TO READ
                                                    ;FOR ERROR LOG
```

****

The device handler Header Section begins here.

****

```
                        ; START OF DRIVER
                        .NLIST CND
   000000                       .DRBEG  RK,RK$VEC,RKDSIZ,RKSTS
```

The .DRBEG macro generates the following block of code (up to the next .LIST CND directive):

```
    000000              .ASECT   [Stores information in block 0 of handler]
            000052      . = 52
                        .GLOBL  RKEND
    000052  000550              .WORD   <RKEND - RKSTRT>
    000054  011300              .WORD   RKDSIZ
    000056  100000              .WORD   RKSTS
```

The three words shown above are extracted by the bootstrap.

Normally, determining the size of the device for the xxDSIZ word, above, is a simple matter. However, some device handlers can control devices that permit two different size volumes to be used. An example of this is the DM handler, which can access either RK06 or RK07 disks through a single controller. Such handlers should place the size of the smaller volume in the xxDSIZ word, above. If necessary, the handler can permit a running program to issue an .SPFUN programmed request to determine the size of the volume that is currently mounted. Bit 10 (SPFUN$) of the device status word must be set by the handler at assembly time to indicate that .SPFUN requests are allowed.

The DM handler, for example, handles I/O to the RK06 and RK07 disks as follows. First, it selects a unit (0 through 7) of the device by placing opcode 01 in RKCS1 (the RK06/07 Control and Status Register 1). Then it gets the value of bit 8 from RKDS (Drive Status Register). A value of 0 means that the selected unit is an RK06. A value of 1 indicates RK07. Next, the handler puts this value, the 0 or 1, into bit 10 of RKCS1. Finally, it is ready to calculate the correct disk address and do a data transfer.

Figure C-10 RK05 Handler Listing (Cont.)

```
000000              .CSECT  [Returns to the unnamed .PSECT]
000000              RKSTRT::
                    .GLOBL  RKINT
```

The first word of the handler, RK$VEC, contains the vector address for the device:

```
000000  000220           .WORD   RK$VEC
```

The second word of the handler, shown below, is the self-relative byte offset to the interrupt entry point RKINT:. It is also used by the monitor abort I/O request code to find the abort entry point of the handler. The abort entry point is the word preceding the RKINT label.

```
000002  000172           .WORD   RKINT -
```

The third word of the handler, shown below, contains the PS to be inserted into the device vector. The high byte must be 0. The low byte should be 340, for priority 7. However, if the low byte is lower than 340, the .FETCH code forces it to the actual new PS in the vector in order to specify priority 7. The condition bits can be used to distinguish up to 16 different interrupts or controllers. They are copied into the PS word of the vector and set in the PS when the device interrupts using that vector.

The monitor also uses the third word of the handler as a flag area in order to hold the handler. When the monitor needs to manipulate the I/O queue of a handler while I/O is active, or if it must abort the handler, it prevents the handler from completing a transfer and releasing a queue element by setting bit 15 of this word. It actually does this by rotating the C bit into bit 15. If the handler does a .DRFIN operation while it is held, the monitor shifts word 3 right again, effectively setting bit 14, and returns without affecting the queue. When the handler is freed later, the monitor checks to see if bit 14 was set, indicating that the handler tried to return a queue element while it was held. If that is so, monitor routine COMPLT is called for the handler to return the queue element and start an I/O operation on the next queue element.

```
000004  000340           .WORD   ^O340

000006              RKSYS::[Required if the device can be a system device]
```

Figure C-10 RK05 Handler Listing (Cont.)

The address of the fourth word of the handler, RKLQE, is placed in the monitor $ENTRY table. RKLQE points to the last queue element in the queue for this handler, thus making it easier for RMON to add elements to the end of the queue. If there are no more elements in the queue, this word is 0.

```
        000006  000000  RKLQE:: .WORD   0
```

The fifth word of the handler, RKCQE, points to the third word, Q.BLKN, of the current queue element. If there is no current queue element, RKCQE is 0.

```
        000010  000000  RKCQE:: .WORD   0
                        .LIST CND
```

```
***********************************************
```

The handler I/O Initiation Section begins here.

```
***********************************************
```

```
                        .IF EQ MMG$T
```

Most of the code in the handler is assembled based on the value of certain conditionals, such as MMG$T. The IF statement above controls the assembly of the code in this handler. If the handler is assembled with MMG$T = 1 (that is, with extended memory support enabled), code following the .IFF statements is assembled. If the handler does not have extended memory support enabled (that is, if MMG$T = 0), code following the .IFT statements is assembled. Code following the .IFTF statements is always assembled, regardless of the value of MMG$T.

```
                        .IFTF
```

The next statement is the first executable statement of the handler code. This point is reached after a .READ or .WRITE programmed request is issued in a program. The monitor queue manager calls the handler with a JSR PC at the sixth word whenever a new queue element becomes the first element in the handler's queue. This situation occurs when an element is added to an empty queue, or when an element becomes first in the queue because a prior element was released. This section initiates the I/O transfer.

Figure C-10 RK05 Handler Listing (Cont.)