PART I
TUTORIAL

CONTENTS

PREFACE

Part I of the Guide to Writing OS/32 Driver's Manual is a tutorial for writing basic drivers, with the last chapter devoted to more advanced concepts. Chapter 1 is a review of basic terminology and concepts. Chapter 2 covers the essentials for writing driver code and Chapter 3 gives complete information for including a driver in the operating system. Two sample drivers are provided in Chapter 4, a simple digital input/output (DIO) interface and a complex TELEX tridensity magnetic tape driver. Chapter 5 introduces advanced concepts in drivers. These include the translation table, nonphysical device drivers and supervisor call 6 (SVC6) and trap generating device drivers.

CHAPTER 1

INTRODUCTION

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 BASIC CONCEPTS

When a new user installs Perkin-Elmer's 0S/32, it is often necessary to connect the operating system to the user's equipment and peripherals via a customized driver. Figure 1-1 demonstrates this situation.

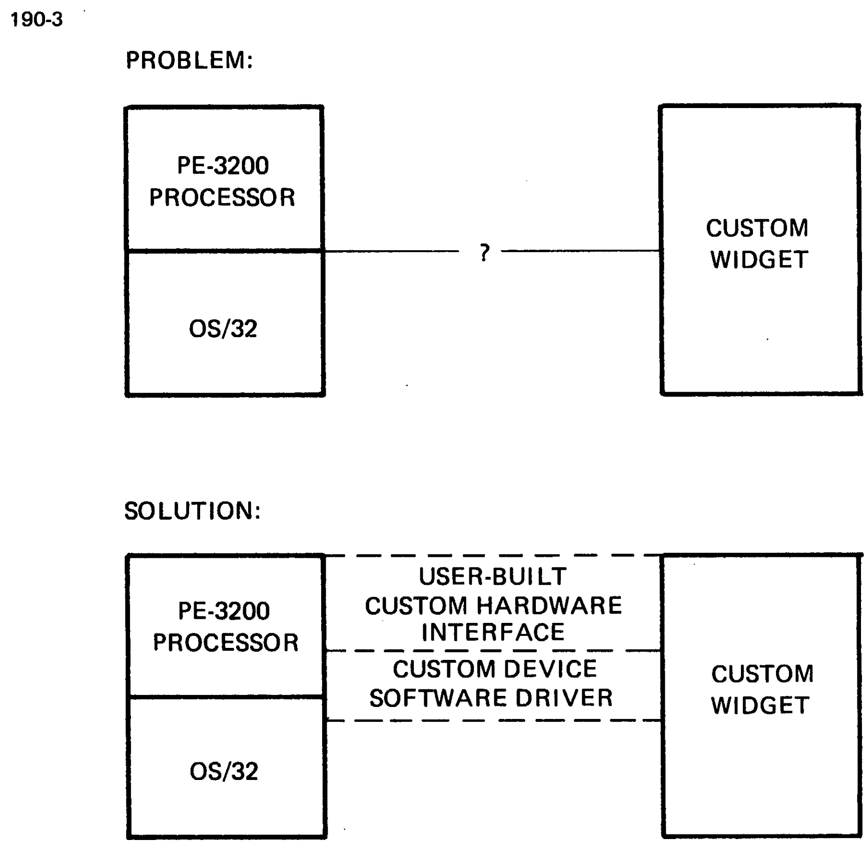

Figure 1-1. The Customized Driver as Interface

The Perkin-Elmer input/output (I/O) driver performs I/O to the device at the hardware level, such that user programs merely issue reads and writes, thus making the command sequences for each type of device transparent to the user. FORTRAN and other high-level languages use READ and WRITE statements which are translated into supervisor call 1 (SVC1) instructions by their respective compilers. The operating system then executes the SVC1 which calls the device driver to do the I/O to the requested device.

How does the system get into the I/O driver?

EXAMPLE: A FORTRAN program

```
    READ(1,900) IX
```

is translated by the compiler into:

```
    SVC 1,PBLK (assembly code)
```

which is then assembled into object code:

```
    E110 xxxx
```

The E1 is the op code for an SVC which causes the OS to initiate the I/O.

For those of you who are interested, the operating system processes the SVC instruction in module EXIN (set up in low memory to handle microcoded vectors from the SVC instruction). EXIN passes the I/O request (SVC1) to module EXIO which handles pre and postprocessing of the I/O. EXIO validates the request, dispatches the driver and processes I/O termination as in the following:

```
   EXIN--->EXIO--->driver--->EXIO
```

1.2 TERMINOLOGY

For the user who is unfamiliar with Perkin-Elmer terminology, those terms required for use of this manual are given here. The order in which these terms are presented correspond to the structural level within the software at which they are encountered.

- User task (u-task): a complete applications job, typically written in a high-level language such as FORTRAN, COBOL and Pascal. A u-task can be written in Common Assembly Language/32 (CAL/32), although the existence of powerful optimizing compilers greatly reduces the requirements for u-tasks written in CAL/32. A u-task performs a useful applications function.

- SVC1: an SVC code 1 is a single machine-level instruction that can be issued by a u-task to initiate an I/O operation. All Series 3200 Processors provide this instruction. Typically, the SVC1 instruction is issued by a user-callable subroutine or by code compiled as the result of high-level language statements being processed into machine code by a compiler. Typically, the user is not explicitly aware of the SVC1 instruction. The user simply issues reads and writes.

- SVC1 parameter block: the SVC1 instruction references a six-fullword (24 byte) block which defines the I/O operation and is interpreted by the OS/32 I/O subsystem. The definition of the SVC1 parameter block does not necessarily concern the writer of a device driver. If the driver performs device-dependent functions, then the SVC1 parameter block can be partially redefined by the device driver.

- Device control block (DCB): This is the principle structure with which the writer of a device driver must be concerned. The DCB provides all of the parameters required to define the device to both the device driver and to the OS/32 I/O subsystem software. The DCB consists of a device-independent and a device-dependent segment. The device-independent segment is defined in the same manner for all devices. A device driver can reference data in this area, but in general can alter only a few specific items, since it is reserved for the OS/32 I/O subsystem. The device-dependent segment can be defined differently for each type of device and can be of arbitrary size. There is exactly one, and only one, DCB for each device configured in a system. For example, a system with four CRTs and a disk drive contains five DCBs.

- Channel control block (CCB): This is a small control block defined by the interrupt service features implemented in the microcode of the Series 3200 Processors. An additional area of the CCB has been defined by OS/32 for use by device drivers. The CCB is the mechanism available to the device driver for passing control and device information to the interrupt service routines in the device driver.

- Device driver: A software module consisting of both a definition of the DCB for a specific type of device, and various sections of executable code required to actually control the device, communicate with the OS/32 I/O subsystem, and service device interrupts in a manner consistent with the Series 3200 Processor interrupt service scheme. In addition to the DCB, a device driver written to control any interface consists of a CCB, a driver initialization routine (DIR), one or more interrupt service routines (ISRs) and one or more event service routines (ESRs). Several other routines, specified by the I/O handler list (IOH) are options.

- Interrupt service pointer table (ISPT): An ordered list containing one entry for each possible device address in the system. This table starts at memory location X'D0' with each entry occupying one halfword. It is the responsibility of the software controlling the I/O (the device driver) to set up this table with the address of the immediate ISR or the address of the CCB+1 for auto driver channel operation.

- Auto driver channel: A microcode routine which is entered whenever the halfword entry in the ISPT is odd. This routine interprets the address in the ISPT as the address of a CCB plus one. The CCB consists of a description of the operation to be performed, and a list of parameters associated with the operation.

1.3 COMPONENTS OF A DEVICE DRIVER

A device driver is divided into major components that carry out all necessary functions. The most important functions are described in the following list.

- Driver initialization routine (DIR):

  The DIR is the first section of code executed in a device driver. Generally, it performs certain bookkeeping functions and the initial communication to the device hardware interface to set up the I/O operation. The I/O subsystem uses register set 5 for driver initialization service. The user does not have to worry about saving and restoring registers upon entry and exit from driver initialization routines. The writer of an I/O driver can use 15 of the 16 registers of set 5. Register D, which contains the DCB address, must not be destroyed. Routines in EXIO require that this register contain the DCB address after exit from the driver.

- Interrupt service routine (ISR):

  ISRs service the hardware interrupts from the device interface. Generally, the ISR performs the minimum amount of control required to maintain the progress of the I/O operation and to determine when the operation is complete. When coding an ISR, certain register conventions must be upheld to insure the integrity of the system. The operating system expects registers 8-F of set 0 to be untouched by the driver. Therefore, the writer of an ISR should use only registers 0-7 of set 0. The register set selected (0,1,2, or 3) depends upon the hardware configuration. Normally, this would be set 0. If other than 0, in addition to backplane wiring, Sysgen32 or cupmt must also reflect the interrupt level in the device specifications.

  NOTE

  If set 1, 2, or 3 is used, of course, all registers would be available for ISR use; however, for compatibility, it is strongly suggested that only 0-7 be used.

  When an ISR is entered, the microcode sets the program status word (PSW) to X'2800', which enables all higher level interrupts and machine malfunction interrupts. If it is necessary to start two or more devices at the same time, it is usually necessary to disable all interrupts if your interrupt level is 1, 2 or 3. This prevents higher level interrupts from disrupting the appropriate sequencing of operations. The PSW to disable all interrupts, leaving machine malfunction enabled is X'2000'.

- Event service routine (ESR):

  ESRs provide termination functions, such as shutting down the device, issuing retries where appropriate and doing various bookkeeping functions for the OS/32 I/O subsystem.

- The I/O handler (IOH):

  An IOH is an ordered parameter list which defines the appropriate processing routines for various I/O functions. The list is defined by the $IOH macro. Special routines for initialization, end of task, as well as device-dependent functions can be defined via an IOH.

These IOH routines are called by various modules within the operating system. EXIO checks for read, write and command function IOH entries. EXIN checks for the presence of an initialization IOH during the SYSINIT routine. If an INIT IOH has been specified, EXIN does a branch and link to the routine. EXSV, during end of task processing, checks for the presence of an EOT IOH. If this routine exists, EXSV does a branch and link to the routine.

The following is a sample of the way the IOH macro is used within a device driver. Since this is a macro, column 72 must contain a continuation character, and each item except the last is followed by a comma.

```
IOH    NAME=MMDIOH,                        1
       READ=BARERD,                        1
       WRITE=BAREWR,                       1
       AIT=SVC1WAIT,                       1
       TEST=SVC1TEST,                      1
       SET=BARERD,                         1
       HALT=SVC1NOOP,                      1
       REW=SVC1NOOP,                       1
       BSR=SVC1NOOP,                       1
       FSR=SVC1NOOP,                       1
       FFM=SVC1NOOP,                       1
       BFM=SVC1NOOP,                       1
       INIT=INIT.MMD,                      1
       WFM=SVC1NOOP
```

Some of the entries in the IOH list are as follows:

INIT:

If a device requires some special setup at system initialization time, the INIT parameter can be used to specify the name of the routine to be executed. The sysinit code in EXIN branches and links to this routine. An example of such processing would be the initialization of a driver's internal table.

```
Register Conventions:

Entry:                                 Exit:

R8.........return address              R8.........return address
R11........DCB address                 R11........DCB address
                                       All other registers can be
                                       destroyed.
```

EOT:

If a driver must perform special cleanup operations when a task terminates, then the EOT parameter must be specified. At EOT time, the SVC3 process or (module EXSV) checks for an EOT handler for each logical unit (lu). The drawback to this cleanup method is that if the logical units are already closed, then no cleanup is done, since there is no close IOH.

```
Register Conventions:

Entry:                                  Exit:

R8.........return address               R8.........return address
R9.........TCB of task going            R9.........TCB of task going
to end of task                          to end of task
R10........DCB address                  R10........DCB address
                                        All other register can be
                                        destroyed.
```

- DDF:

A device-dependent function is a routine that is to be executed for a particular device. The function code for all device-dependent functions is X'81'; however, each device driver defines this code in a manner specific to each device. See Chapter 4 for more information on mag tape drivers.

1.4 OS/32 DRIVER INTERFACE ROUTINES

The writer of an I/O driver needs only to know when and how to call OS/32 routines. There is no need to be intimately familiar with the details of the operating system interface routines.

DIRDONE - a routine called at the end of a DIR. This routine sets DCB.ESR with a default termination handler. (Register D must contain the address of the DCB.)

```
B     DIRDONE
```

IODONE - a routine called at the end of ESR. This routine places the status in the SVC1 parameter block (if any), sets a requested task trap, removes the task from an I/O wait and disconnects the leaf from the tree. (Register D must contain the address of the DCB.)

```
B      IODONE
```

EVRTE - a routine which is called after a driver has modified the address of the termination routine. This routine is called in place of DIRDONE. (Register D must contain the address of the DCB.)

```
B      EVRTE
```

EVMOD - a routine in module EXIO to set another entry as the ESR address (modifies DCB.ESR). For current revisions of the operating system, some system overhead can be eliminated by performing this function directly:

```
LA     UE,TERMXXX1
OI      UE,Y'80000000'        set indicator like EVMOD does
ST      UE,DCB.ESR(UD)
```

In most cases, it is recommended that this subroutine convention be used, so that the driver is compatible with future revisions of OS/32. (Register D must contain the address of the DCB.)

```
LA     UE,TERMXXX1    set new term address
BAL     U8,EVMOD
```

EVREL - a routine to release resources such as controllers and selector channels (SELCHs) acquired by a driver.

```
LIS        UE,3             release at SELCH level
L          UF,DCB.LEAF(UD)
BAL        U8,EVREL
```

TOCHON - a timer routine that is called, usually before a SINT instruction, to enable the system to return a time-out condition if a device does not respond within a given period of time. (Register D must contain the address of the DCB.)

```
BAL        U8,TOCHON
```

TOCHOFF - a timer routine that is called, after the driver has finished its work, to remove a given device from the time-out chain. (Register D must contain the address of the DCB.)

```
BAL        U8,TOCHOFF
```

III - an entry point in module EXTI. This routine is the null interrupt routine which performs an LPSWR. The address of III is stored in the ISPT when the system is to perform no action upon a device interrupt.

```
LA         E7,III
STH        E7,ISPTAB(E2,E2)
```

SQ - the address of the system queue. This structure is used by a driver ISR to schedule driver termination.

```
L      E6,DCB.LEAF(E5)
ATL     E6,SQ
```

The above routines are the major interfaces between a device driver and OS/32.

Associated subroutine file: SUBS.MAC - contains commonly used subroutines for driver use. For efficiency, since many of these routines are only one or two lines of code, it is recommended that the code be placed in-line (especially for interrupt service coding).

1.5 DRIVER COMMUNICATION WITH THE TOTAL COMPUTER SYSTEM

The DCB is the principle means by which a device driver communicates with the rest if the system. Most of the DCB fields are used by various other components of the system to define the environment or a specific operation to the driver; a few DCB fields are used by the driver to report the results of an operation to the remainder of the system. In certain situations, specific registers are used to communicate between the driver and the operating system or processor firmware.

The following system components communicate with a device driver:

1. The user, who issues an SVC1 instruction.
2. The operating system, which provides several services.
3. The user, who defines the system through system generation (sysgen) parameters.
4. The sysgen process, which allocates data structure areas.
5. The processor firmware, which responds to interrupts.
6. The writer of the driver, who defines values in the DCB.

The writer of a driver must be aware of how each one of these system components communicates with the device driver.

The user of the system issues an SVC1 instruction to initiate an I/O operation. The associated SVC1 parameter block defines the operation to be done and receives some information on the results of the operation. The communication between the user and the driver is carried out by the operating system. The situation is equivalent to two people carrying on a phone conversation: each person accesses his own telephone set; the telephone company's equipment moves the signals between the two telephone sets.

```
The USER accesses                         The DRIVER accesses
an SVC1 parameter                           the DCB fields:
block:

 SVC1.FC ---------- I/O function code --------> DCB.FC
 SVC1.LU -------------- lu number ------------> DCB.LU
 SVC1.STA <----- operation result status ------ DCB.STAT
 SVC1.SAD ------- I/O buffer start address ---> DCB.SADR
 SVC1.EAD ------- I/O buffer end address -----> DCB.EADR
 SVC1.RAD ---------- "random address" --------> DCB.RAND
 SVC1.LXF <------- length of transfer --------- DCB.LLXF
 SVC1.XOP --------- (user-defined) -----------> DCB.SV1X
```

In addition to moving operation parameters between the DCB and the user's SVC1 parameter block, the operating system is responsible for starting the execution of various sections of the driver, being sure that the driver does not take too much time to complete the operation and informing the driver as to which device the operation is to be directed to.

```
   The Operating                                   The Driver:
    System:

    Register 13 ------- address of DCB -------> Register 13

                         address of the
    DCB.ESR <-------- next event service ------ DCB.ESR
                       routine to execute

    DCB.TOUT <-------- time-out value --------> DCB.TOUT

    SQ <----- event tree leaf address --------- DCB.LEAF
              (schedules ESR execution)

                         address of the
    DCB.UPBK ------- user's SVC1 block -------> DCB.UPBK
                         as seen by the
                             user

                         address of the
    DCB.PBLK ------- user's SVC1 block -------> DCB.PBLK
                         as seen by the
                            driver

                          address of the
    DCB.TCB -------- task control block ------> DCB.TCB
                         for the user
```

Note that since register 13 always points to the DCB, it is important that the driver never modify it. The DCB.TOUT field is used by all drivers. The DCB.ESR field is occasionally used. The fields DCB.UPBK, DCB.PBLK and DCB.TCB are almost never needed by a device driver. These fields are listed here only because a few special circumstances require them.

The user (perhaps unknowingly) defines some parameters through the specifications written in a sysgen file. For example, the user writes the statement:

```
    D300:,FE,54,SELCH=F0,CONTR=FB
```

This is mapped to the DCB as follows:

```
"FE" -------- device address ----------> DCB.DN
"54" -------- device type code --------> DCB.DCOD
"SELCH=FO" --- SELCH address ----------> DCB.SDN
"CONTR=FB" --- controller address -----> DCB.CDN
```

For another example, the user writes:

```
MD5E:,5E,39,XDC=X0810,RECL=132
```

This is mapped to the DCB as follows:

```
"5E" -------- device address ----------> DCB.DN
"39" -------- device type code --------> DCB.DCOD
"XDC=X0810" --- user-defined ----------> DCB.XDCD
"RECL=132" --- device record length ---> DCB.RECL
```

When the sysgen procedure is run, various data structure areas (such as the DCB) are assigned physical locations within the operating system. It is necessary that these data structures contain pointers to other data structures within the system, so that the various operating system routines can find their way around. For example, a device driver must be able to locate the DCB if it knows only the location of the CCB. The operating system must be able to locate the start of the device driver if it knows only the location of the DCB. Since these pointers are set up at sysgen time, they are static and must never be modified by the driver. Some of these pointers are:

```
Sysgen procedure                          Driver

DCB.CCB ------- address of CCB --------> DCB.CCB
CCB.DCB ------- address of DCB --------> CCB.DCB
DCB.LEAF ---- event tree leaf address -> DCB.LEAF
```

Some entries in the DCB are used primarily by the operating system to locate various routines within the driver. These fields are set up at sysgen time, and are not modified by the driver. These fields include:

```
Sysgen procedure                             operating system

     DCB.INIT ---- address of driver's -----> DCB.INIT
                  initialization routine
     DCB.TERM ---- address of driver's -----> DCB.TERM
                  termination routine
                   address of driver's
     DCB.FUNC ---- command function --------> DCB.FUNC
                   processing routine
```

A device driver must also communicate directly with the processor firmware routines that process interrupts. The driver must tell the processor what ISR is to be executed to process the next interrupt from the device. The processor must in turn tell the ISR what device interrupted and where the associated CCB is located. Also, the processor provides the ISR with the proper PSW and location counter with which to exit from interrupt service. The communication between the processor and the driver is as follows:

```
Processor hardware                                 Device
and firmware                                       driver:

ISPTAB <------ address of CCB -------------- DCB.CCB
CCB.SUBA <------ address of ISR to execute -- "LA" instruction
hardware -- address of interrupting device -> register 2
hardware -- status of interrupting device --> register 3
microcode --- address of CCB ---------------> register 4
microcode --- PSW at time of interrupt -----> register 0
microcode --- LOC at time of interrupt -----> register 1
```

1.6 A SPECIAL NOTE ON DEVICE TIME-OUTS

OS/32 uses the line frequency clock (LFC) (which interrupts 120 times per second) to maintain the device time-out mechanism. Every second, the system decrements the time-out constant for each device by one. If the time-out constant is X'7FFF', the device cannot be timed out. When a driver's interrupt service has gone to completion, the time-out constant should be set to -1 (X'FFFF') to indicate that a time-out did not occur. If the time-out constant decrements to zero, the device did not respond in the specified length of time and the driver should set an X'8282' status, to indicate time-out.

The timer management routine which controls the device time-out mechanism is the OS/32 module EXTI in routine TIMCH3. If this routine finds X'7FFF', 0, or a negative value in DCB.TOUT, it processes the next device on the time-out chain. The value in DCB.TOUT is decremented by one and if the value is now zero, the device has timed out. If this time-out condition occurs, EXTI adds the device leaf to the system queue for termination processing.

Drivers should always check for time-out before adding a leaf to the system queue. Crash code 153 may result if a leaf is added to the system queue twice. A safety check mechanism in EXIO prevents the item from being added twice (if the system is generated with safety checks enabled).

CHAPTER 2

WRITING THE DRIVER

CHAPTER 2

WRITING THE DRIVER

2.1 INTRODUCTION

The previous chapter outlined the basic components required to write a device driver. These basic components included the device control block (DCB) relationships, the input/output (I/O) subsystems relationships and the specific routines required within the device driver. With this conceptual background, the user can now address the essential question, "How do I write the code?"

2.2 WRITING DRIVER CODE

The best way to begin a discussion of writing driver code is to begin with an anyalysis of some simple examples. The following examples show very basic drivers.

Step 1: The Simplest Case

The following is the simplest possible routine that satisfies the basic operating system interface requirement for a driver. This driver performs only the function of branching back to the operating system. The termination routine is never executed, since the initialization routine branches to the I/O operation complete routine in the operating system.

The last instruction executed by a driver must be a branch to IODONE, which is an entry point to the operating system. This is the only way that a driver can indicate to the operating system that the I/O operation is complete. The operating system enters the driver at the label INITDVR1. This label is specified in the DCB. Standard Perkin-Elmer convention requires that the first four characters be INIT.

```
*
INITDVR1 B    IODONE      branches back to the operating system
*
TERMDRV1 EQU   *          this will never be executed
*                         but must be here because it must
*                         be specified in the DCB.
END
```

Step 2: Scheduling Event Service Routine (ESR)

To schedule an ESR routine, add the contents of the DCB.LEAF field to the system queue. This can be done anywhere in the driver, but normally is done only in an interrupt service routine (ISR), as is illustrated later. Since this driver only has one ESR routine, (the termination routine), this is the ESR scheduled, by default.

This driver also illustrates another basic operating system interface, which is the branch to DIRDONE. A driver which has initiated an I/O operation, but has not completed that operation, should branch to DIRDONE (another entry in the operating system) to allow the processor to do other unrelated tasks while the I/O operation is in progress. Branching to DIRDONE normally forces the next ESR routine that is executed to be the termination routine, labeled TERM..., as specified in the DCB definition.

This driver also illustrates the fact that register 13 (UD) always points to the DCB of the device to which the I/O is directed. The operating system sets up this register before branching to the driver. The contents of register 13 should never be modified by the driver, for several reasons that will be shown more completely later.

```
*
* SCHEDULE ESR EXECUTION THROUGH SYSTEM QUEUE SERVICE
*
INITDRV2 L    UO,DCB.LEAF(UD)       Driver is always entered
         ATL  UO,SQ                 with reg 13 = addr of DCB
         B    DIRDONE               Register set 5 is always
*                                   used, except on some
*                                   8/32 and all 7/32 CPUs.
TERMDRV2 B    IODONE
*
         END
*
```

Step 3: Setting Up Entry To ISRs

The following example illustrates the basic execution sequence of a typical driver: from initialization to interrupt service to termination. This driver utilizes the minimum possible code to set up a path for entry to an ISR. It also illustrates the important step of entering an ISR to initiate an I/O operation. This is done to avoid responding to an unexpected interrupt while the device is being started. Also, as will be shown later, it is sometimes necessary to insure that certain sequences of operations are executed without any interruption, due to timing requirements. This can only be accomplished in an ISR.

In this driver, the initialization routine consists only of the code required to set up the interrupt service routine. The termination routine consists only of the exit to the operating system. The one ISR is entered by execution of the SINT instruction, since it is the first ISR to be executed. Since it is also the last ISR to be executed, it dismantles the path to itself, to avoid responding to any additional interrupts from the device.

```
INITDRV3 EQU   *                    entry point specified by DCB
         LH    U2,DCB.DN(UD)        get physical address of device
         LHL   UC,DCB.CCB(UD)       get address of CCB for device
         LIS   U0,0                 set up a nonexecute command
         STH   U0,CCB.CCW(UC)       into the channel control word
         LA    U0,ISR0              put the address of the
         STH   U0,CCB.SUBA(UC)      ISR into the CCB
         AIS   UC,1                 make CCB address odd for ISPT
         STH   UC,X'D0'(U2,U2)      put CCB addr into ISPT
         LH    U1,DCB.ILVL(UD)      get the interrupt level
         SINT  U1,0(U2)             force entry into the ISR
*
*   (the code at "ISR0" is now executed; after the
*   ISR exits, the following instruction is executed)
*
         B     DIRDONE              after exit from first ISR,
*                                   go back to the OS.
*
*   The following ESR is executed sometime after
*   the branch to DIRDONE is executed. It is executed
*   because it was scheduled by the ISR routine.
*
TERMDRV4 EQU   *                    termination routine entry
         B     IODONE               exits to operating system
*
*   The following ISR is entered
*   by execution of the SINT instruction. The ISR is in
*   "PURE" code to insure that it is located
*   in the lower part of memory (it must be in the first
*   64KB of memory). The ISR schedules the ESR for
*   execution, and disables further interrupts.
*
* Note that on entry to the ISR, the following registers
* have been set up by the processor microcode:
*
*        E0,E1 = return program status word (PSW)/location counter (LOC)
*        E2    = address of interrupting device
*        E3    = status of interrupting device
*        E4    = address of CCB for interrupting device
*
```

```
ISRO      EQU    *
          L      E5,CCB.DCB(E4)        get address of associated
                                       DCB out of CCB.
          LA     E7,III                put address of null
          STH    E7,X'DO'(E2,E2)       interrupt return into ISPT
          L      E6,DCB.LEAF(E5)       now schedule the ESR for
          ATL    E6,SQ                 execution.
          LPSWR  EO                    exit interrupt service
```

NOTE

The examples in Steps 1, 2, and 3 are all complete drivers with respect to the operating system. The reader can include any one of them into a system; an supervisor call 1 (SVC1) directed at a "device" handled by one of these drivers always returns immediately to the caller. Any one of these three drivers can be used to time basic SVC1 service processor overhead.

Step 4: Executing a Sequence of ISR

In the example in Step 3, there was only one ISR, which is generally not very useful. To control a real device, we must have a minimum of two ISR routines: one to start the device and one to stop the device. The initialization routine sets up the interrupt path to the first ISR; the first ISR sets up the interrupt path to the second ISR and starts the device. The second ISR receives the hardware interrupt, stops the device, dismantles the interrupt path and schedules the ESR for execution; the ESR exits to the operating system.

For ease of understanding the sequence of execution of code, the following driver is written in a way such that the lines of code are in the same order as they are executed by the processor. This is an acceptable way to write a driver, as long as the placement of the IMPUR and PURE statements is correctly done.

However, the reader will probably find that a listing of the driver with the code organized as shown in the example for Step 3 is easier to use when debugging on a system, as all of the IMPUR and PURE code listings are separated into two separate blocks, just as it is in memory.

```
         IMPUR
INITDRV4 EQU    *                 entry point from operating system
         LH     U2,DCB.DN(UD)     get physical address of device
         LHL    UC,DCB.CCB(UD)    get address of CCB for device
         LIS    U0,0              set up a nonexecute command
         STH    U0,CCB.CCW(UC)    into the channel control word
         LA     U0,ISR0           put the address of the
         STH    U0,CCB.SUBA(UC)   ISR into the CCB
         AIS    UC,1              make CCB address odd for ISPT
         STH    UC,X'D0'(U2,U2)   put CCB addr into ISPT
         LH     U1,DCB.ILVL(UD)   get the interrupt level
         SINT   U1,0(U2)          force entry into the ISR
*
         PURE
ISR0     EQU    *                 entered from SINT instruction
         LA     E7,ISR1           get address of next ISR
         STH    E7,CCB.SUBA(E4)   & put into CCB to set up path
*
*    (commands to start the device go here)
*
         LPSWR E0                 exit from this ISR
*                                 (returns to instruction after
*                                 SINT, which is as follows:
*
         IMPUR
*
*  Now exit to the operating system to let the CPU do some unrelated
*  activity while we wait for the hardware interrupt:
*
         B      DIRDONE           this is logically part of
*                                 the initialization routine
*
ISR1     EQU    *                 The interrupt from the
*                                 hardware will force this to
*                                 be executed.
         LA     E7,III            get address of null interrupt
         STH    E7,ISPTAB(E2,E2)  routine & put into ISPT to
*                                 disable receipt of more int's.
*
*     (commands to stop the device go here)
*
```

```
          L      E5,CCB.DCB(E4)     get address of DCB for device
          L      E7,DCB.LEAF(E5)    get value to put on SQ to
          ATL    E7,SQ              schedule ESR execution
          LPSWR  E0                 exit this ISR.
*
          IMPUR
TERMDRV4  EQU    *                  termination routine
          B      IODONE             exits to operating system.
```

Step 5: Setting Up Device Time-outs and User Status

The operating system is used to force an ESR to be executed if the I/O operation takes too much time. This happens if the device somehow does not work right. The code sequence shown below is used to tell the operating system to force a "time-out" (which means forcing an ESR to be executed) if the operation does not complete within a the number of seconds specified by the driver.

A time-out should always be set up before starting an I/O operation. The reason for doing this is that if the device should ever fail to interrupt, the I/O operation would not complete. This in turn would cause the user task (u-task) that issued the I/O operation to be "stuck": it could not be removed from the system (cancelled).

This sample driver illustrates the proper interface to the operating system for "getting on and off the timer chain": in other words, specifying the time-out interval, and how to tell the operating system to start and stop timing the I/O operation. Notice that the value DCB.TOUT must be set and reset in an ISR to avoid a possible conflict with the operating system routines that are necessary for the timing of the I/O operation.

```
          IMPUR
INITDRV5  EQU    *                  entry point from operating system
*
          BAL    U8,TOCHON          this call tells the operating system
*                                   to be ready to time an I/O operation
*                                   This call MUST be made in either
*                                   the initialization routine or
*                                   an ESR routine, NEVER in an ISR!
*                                   This call leaves DCB.TOUT='7FFF'
*
          LH     U2,DCB.DN(UD)       get physical address of device
          LHL    UC,DCB.CCB(UD)      get address of CCB for device
          LIS    U0,0                set up a nonexecute command
          STH    U0,CCB.CCW(UC)      into the channel control word
          LA     U0,ISR0             put the address of the
          STH    U0,CCB.SUBA(UC)     ISR into the CCB
          AIS    UC,1                make CCB address odd for ISPT
```

```
         STH   UC,X'DO'(U2,U2)    put CCB addr into ISPT
         LH    U1,DCB.ILVL(UD)    get the interrupt level
         SINT  U1,0(U2)           force entry into the ISR
*
         PURE
ISR0     EQU   *                  entered from SINT instruction
         LA    E7,ISR1            get address of next ISR
         STH   E7,CCB.SUBA(E4)    & put into CCB to set up path
         L     E5,CCB.DCB(E4)     get address of DCB
*
*  now "turn on" the timing of the operation by the operating system:
*
         LIS   E7,5               specify a five second time-out
         STH   E7,DCB.TOUT(E5)    by putting 5 into DCB.TOUT
*                                 (notice: it is a HALFWORD!)
*
*   (commands to start the device go here)
*
         LPSWR E0                 exit from this ISR
*
         IMPUR
         B     DIRDONE            this is logically part of
*                                 the initialization routine
*
*    Since a value of 5 was set into DCB.TOUT, no more than
*    five seconds can elapse before the hardware interrupts.
*    If the interrupt does not occur, the driver is
*    reentered by the operating system at 'TERMDRV5', and
*    the code at 'ISR1' is not executed.
*
ISR1     EQU   *                  The interrupt from the
*                                 hardware forces this to
*                                 be executed.
         LA    E7,III             get address of null interrupt
         STH   E7,ISPTAB(E2,E2)   routine & put into ISPT to
*                                 disable receipt of more int's.
*
*    (commands to stop the device go here)
*
         L     E5,CCB.DCB(E4)     get address of DCB for device
*
```

```
*  Now check that the operating system did not time-out the
*  operation just prior to the hardware interrupt occurring,
*  but before the ESR could dismantle the interrupt path.
*  This is a unlikely event, but must be checked for the
*  possibility. If the operating system times out the operation,
*  DCB.TOUT = 0, and the execution of the ESR is already
*  scheduled. Therefore, if the ISR finds that DCB.TOUT = 0,
*  then the ISR should not try to schedule the ESR. Doing so
*  can cause a system crash under some software configurations.
*
         LH    E7,DCB.TOUT(E5)    get current DCB value
         BZ    ISR1EXIT           if already zero, exit now.
*
*  Since some time elapses between now and the time that
*  the ESR is executed, it is necessary to indicate to both
*  the operating system and the ESR that the operation
*  completed (i.e., the expected interrupt was received within
*  the specified length of time). The operating system expects
*  that the value 'FFFF' is used for this purpose.
*
         LCS   E7,1               get value of 'FFFF'
         STH   E7,DCB.TOUT(E5)    to set in DCB.TOUT
         L     E7,DCB.LEAF(E5)    get value to put on SQ to
         ATL   E7,SQ              schedule ESR execution
*
ISR1EXIT LPSWR E0                 exit this ISR.
*
* The termination routine customarily sets a status for the
* user of '8282' if a time-out has occurred; otherwise some
* other status is set, such as '0000' if the operation completes
* normally. Thus, the termination routine checks the time-out
* value to determine what has happened. The value put into
* into DCB.STAT is copied by the operating system into the
* user's SVC1 parameter block, so that the user can know
* of the success or failure of the requested operation.
*
         IMPUR
TERMDRV5 EQU   *                  termination routine
         LH    U1,DCB.TOUT(UD)    get the time-out value
         BZ    TIMEOUT5           if zero, we timed out
         LIS   U2,0               and set zero user status
         STH   U2,DCB.STAT(UD)    *
*
```

```
TERM5X   EQU   *
         BAL   U8,TOCHOFF           now tell operating system
*                                   that we do not need any more
*                                   timing services
*
         B     IODONE               exit to operating system.
*
TIMEOUT5 LHI   U2,X'8282'           on time-out, tell user what
         STH   U2,DCB.STAT(UD)      happened.
         B     TERM5X               go to common processing
```

Step 6: Selector Channel (SELCH) I/O, Data Buffer Addresses, and Length of Transfer

NOTE

If your custom device operates under direct processor control on the processor multiplexor bus, you may wish to skip directly to Step 7.

Many custom devices operate under a SELCH. To control the I/O operation, both the custom interface and the SELCH must be controlled. The basic points to be considered when a SELCH is used to control a device are:

1. **The SELCH must have the starting and ending addresses of the data buffer in memory written to it before the transfer starts.**
2. **Once the SELCH is commanded to start the transfer, the device under the SELCH cannot be addressed. The SELCH must be stopped before the device can be accessed.**
3. **The SELCH terminates the transfer, and interrupts the processor, when either of two conditions are met:**
   a. **The SELCH determines that the data transfer has proceeded to the end of the data buffer; in other words, either all of the data in the user's buffer has been sent to the device, or the user's buffer has been filled with data from the device.**
   b. **The device controller indicates to the SELCH that no further data transfer is possible, by activating the appropriate signals on the SELCH private bus. (How this happens electrically is of no importance to the driver software).**

The following driver, which is complete except for the specific commands required to control the device, shows all of the steps required to control an I/O operation through a SELCH. To aid comprehension, all of the SELCH commands are explicitly written. In many Perkin-Elmer supplied device drivers, subroutines are called to interface to the SELCH. Those routines perform the same functions as the code written here. It is a matter of individual programming preference as to whether those subroutines are used, or the commands are written explicitly as shown here. The routines used by some Perkin-Elmer drivers can be found in the module SUBS.MAC; if you plan to

use these subroutines, you should refer to the source code in this module to determine the proper calling sequences.

The fields DCB.FC, DCB.SADR, DCB.EADR, and DCB.LLXF are used in this driver to control the SELCH. The driver references, but does not modify, DCB.FC, DCB.SADR and DCB.EADR. The driver does set DCB.LLXF, which is then copied by the operating system back into the user's SVC1 parameter block to allow the user to know how much data was actually transferred. The field DCB.FC, which is copied directly from the user's SVC1 parameter block, is used to determine whether to set the SELCH and the device in a device read or device write mode.

```
         IMPUR
INITDRV6 EQU   *                  entry from operating system to start I/O
         BAL   U8,TOCHON          go get on timer chain
         LH    U3,DCB.SDN(UD)     get address of the SELCH
         LHL   UC,DCB.CCB(UD)     get address of CCB for device
         LIS   U0,0               set up a nonexecute command
         STH   U0,CCB.CCW(UC)     * in the channel control word
         LA    U0,ISR0            get the address of the first ISR,
         STH   U0,CCB.SUBA(UC)    & set up for start-up SINT instruction
         AIS   UC,1               make CCB address odd, for
         STH   UC,ISPTAB(U3,U3)   * set up of ISP table
*                                 (note SELCH address is used)
         LH    U1,DCB.ILVL(UD)    get the interrupt level
         SINT  U1,0(U2)           force entry to first ISR
*
         PURE
ISR0     EQU   *                  entry from SINT to start I/O
         L     E5,CCB.DCB(E4)     get address of DCB for device
         LA    E7,ISR1            get address of next ISR
         STH   E7,CCB.SUBA(E4)    & set up CCB for hardware interrupt
         LH    E3,DCB.SDN(E5)     get SELCH address
         OC    E3,SLCHSTOP        be sure SELCH is reset
*
* The values DCB.SADR and DCB.EADR were set up by the operating system
* by copying the values SVC1.SAD and SVC1.EAD from the user's SVC 1 block,
* and converting those values to physical memory addresses.
*
         WD    E3,DCB.SADR+1(E5) write 24-bit starting addr
         WH    E3,DCB.SADR+2(E5) * to the SELCH
         WD    E3,DCB.EADR+1(E5) write 24-bit ending address
         WH    E3,DCB.EADR+2(E5) * to the SELCH
         LIS   E6,5               set up the device time-out
         STH   E6,DCB.TOUT(E5)    value for the operating system
```

```
* Here are some things to consider at this point relative
* to programming a SELCH-controlled I/O operation:
*       1. Once the SELCH is commanded to a "go" state, the device is
*          unaccessable from the processor. Thus, we must issue the
*          appropriate commands to the device first, then start SELCH
*       2. Some devices (e.g., magnetic tapes and disks), require that
*          the SELCH be started at essentially the same instant as the
*          device, to insure that a data overflow/underflow does not occur.
*          Thus, ALL interrupts must be shut off while the device and
*          SELCH are started!
*       3. The SELCH (and typically the device) have different command
*          sequences that must be used to start read and write modes.
*          This driver uses two separate sets of code for starting read
*          and write, to simplify understanding of the logic used.
*       4. Most simple devices that operate under a SELCH are most efficiently
*          programmed in such a way that only the interrrupt from the SELCH
*          is needed toindicate the end of the transfer. Thus, the interrupts
*          from the device may be left shut off or ignored, as is done in
*          this driver.
*
         EPSR  E7,E7              get current PSW value
         NI    E7,Y'FF20F0'       reset bit 20 for
*                                 noninterruptable state
*
         LB    E6,DCB.FC(E5)      get the function code that was
         THI   E6,X'40'           specified by the user, then
         BNZ   ISROR              test for read/write, and go
*                                 to the appropriate routine
*
ISROW    EQU   *                  come here to start write I/O
         EPSR  E6,E7              become noninterruptable
         OC    E2,DEVCWRIT        start the device in write mode
         OC    E3,SLCHWRIT        start the SELCH in write mode
         LPSWR E0                 exit the ISR routine
*
ISROR    EQU   *                  come here to start read I/O
         EPSR  E6,E7              become noninterruptable
         OC    E2,DEVCREAD        start the device in read mode
         OC    E3,SLCHREAD        start the SELCH in read mode
         LPSWR E0                 exit the ISR routine
SLCHSTOP DB    X'48'              command to "stop" (reset) SELCH
SLCHWRIT DB    X'50'              SELCH "go & write" command
```

```
SLCHREAD DB    X'70'                SELCH "go & read" command
DEVCWRIT DB    .....                appropriate device write command
DEVCREAD DB    .....                appropriate device read command
*
         IMPUR
         B     DIRDONE              driver initialization routine
*                                   exit back to operating system.
*
         PURE
         ALIGN 2                    be sure we are not on a
*                                   byte boundary due to DB entries
*                                   in previous ISR routine
*
ISR1     EQU   *                    SELCH interrupt forces this
*                                   routine to be executed.
*
*  note that E2 contains the address of the SELCH
*
         OC    E2,SLCHSTOP          give the SELCH a stop command
         LA    E7,III               dismantle the interrupt path
         STH   E7,ISPTAB(E2,E2)     * to avoid any more interrupts,
         L     E5,CCB.DCB(E4)       get address of DCB of device
         LH    E6,DCB.TOUT(E5)      chech the time-out constant
         BZ    ISR1X                if already timed-out, do nothing,
         LCS   E6,1                 otherwise, indicate no time-out
         STH   E6,DCB.TOUT(E5)      by setting flag to 'FFFF'
         L     E7,DCB.LEAF(E5)      now schedule the ESR to
         ATL   SQ,E7                * be executed.
ISR1X    LPSWR E0                   exit from this ISR.
*
         IMPUR
TERMDRV6 EQU   *                    operating system enters here
*                                   for I/O operation termination
*
         LH    U3,DCB.SDN(UD)       get SELCH address
         OC    U3,SLCHSTOP          be sure SELCH is stopped
*                                   (if we timed-out, it may not be)
         RDR   U3,U0                now get SELCH final address
         RHR   U3,U1                (all 24 bits)
         SLL   U0,16                make up a 24-bit value...
         OR    U0,U1                now U0 contains SELCH final addr
         C     U0,DCB.EADR(UD)      is final addr = user's end addr?
         BNE   TERM6X               if not, branch to special logic
```

```
TERM6A   S     U0,DCB.SADR(UD)   now calculate the length of xfer
         AIS   U0,1              adjust for inclusive addressing
         ST    U0,DCB.LLXF(UD)   and save length of xfer for user
         LIS   U1,0              no error if equal, so set status
TERM6B   STH   U1,DCB.STAT(UD)   for user.
*
         BAL   U8,TOCHOFF        get off the timer chain
         B     IODONE            and exit to operating system I/O termination.
*
* The following code is executed only if there is not a final address match:
* SELCH end address equals user's specified end address.  Exactly what
* should be done under these circumstances depends upon the device, and
* how the system designer wants it to behave.  For the purpose of this
* driver, make the following choices:
*     1. If the final address is not in the range of the
*        the user's start and end addresses, then set the
*        status = X'8490', the length of transfer to zero,
*        and exit. (Some Perkin-Elmer supplied drivers crash the system
*        deliberately if the final address is outside the
*        user's start/end address, under some conditions.)
*     2. If the operation timed-out, set the status = X'8282',
*        set the length of transfer to zero, and exit
*     3. If the operation did not time-out, but the end address
*        is less than the user's end address, calculate the
*        actual length of transfer, and set the status = 0.
*
TERM6X   EQU   *                 come here if SELCH end addr
*                                is not equal user's end address
         C     U0,DCB.SADR(UD)   check end address for being
         BM    TERM6E            out of range
         C     U0,DCB.EADR(UD)   *
         BP    TERM6E            *
         LH    U4,DCB.TOUT(UD)   did we time-out?
         BNZ   TERM6A            if not, no error
*
         LHI   U1,X'8282'        if time-out, set proper status
         LA    U8,III            and reset the ISP table
         STH   U8,ISPTAB(U3,U3)  *
         B     TERM6B            *
*
TERM6E   LHI   U1,X'8490'        if end address out of range,
         B     TERM6B            set appropriate status
*
```

Step 7: Data Transfer Using Autodriver Channel Programming

Many low-speed devices, primarily interactive terminals and some communications equipment, have controllers that are designed to present an interrupt to the processor for each character transferred. The following driver provides an explanation of the specific coding sequences that are peculiar to autodriver channel programming.

Note that this driver is designed to operate on a byte transfer device. This means that:

1. The device controller, when addressed, leaves the HW0 (halfword) signal on the multiplexor (MUX) bus in the inactive (high) state.
2. The device controller, in response to an active DR0 signal on the MUX bus (i.e., data request), activates only eight data lines (D080 through D150).
3. The device controller, in response to an active DA0 signal on the MUX bus (i.e., data available), responds to only eight data lines (D080 through D150).

It is important to note that the distinction between byte- and halfword-oriented devices is made based on the behavior of the device controller on the MUX bus (or private SELCH bus). The characteristics of the data lines that the device controller present to the device have no bearing on the byte vs. halfword distinction for the purposes of autodriver channel programming. The hardware design manual, or circuit logic diagrams, for the device controller must be consulted to determine if the device controller is a byte or halfword device.

```
INITDRV7 PROG        Sample driver #7.
         NLIST
         $REGS$
         $DCB$
         $CCB
         LIST
         EXTRN DIRDONE,III,IODONE,ISPTAB,SQ,TOCHOFF,TOCHON
         ENTRY INITDRV7,TERMDRV7
         IMPUR
INITDRV7 EQU   *                entered here by operating system to do I/O op
         LH    U2,DCB.DN(UD)    get the device address
         LHL   UC,DCB.CCB(UD)   get the address of the CCB
*
* For the purposes of showing how the two-buffer concept can be
* used in autodriver channel programming, this driver outputs
* a prefix and a suffix for each message the user writes to
* the device. Thus, both buffer 0 and buffer 1 are used.
*
* We initially set up buffer 0 to output the prefix and buffer 1 to output
* the user's message. Reuse buffer 0 to output the suffix of the message.
* Since, by design choice, we wish for the complete line
* of output to fit on the CRT screen, we will limit the
* user to a message of 40 characters or less. If the user
* specifies a message greater than 40 characters, the
* initialization routine outputs a status of X'8490'
* and a length of transfer of zero, and exit. This
* illustrates the common practice of doing some initial
* parameter and device status checks in the initialization
* routine, before the first ISR is entered.
*
         L      U0,DCB.EADR(UD)  get the user's ending address
         ST     U0,CCB.EB1(UC)   & save as buffer 1 address.
         S      U0,DCB.SADR(UD)  calculate requested message
         AIS    U0,1             * length, then verify it is
         CHI    U0,40            not greater than 40.
         BP     INIT7ERR         If more than 40, error exit.
         LIS    U1,1             now calculate value to put
         SR     U1,U0            * into CCB as buffer count
         STH    U1,CCB.LB1(UC)   & save for later use
*
         LA     U1,PREFIX.S      now set up the CCB for the
         LA     U0,PREFIX.E      prefix to be output
         ST     U0,CCB.EB0(UC)   (use buffer 0)
         SR     U1,U0            calculate value for counter
         STH    U1,CCB.LB0(UC)   and save for later use.
*
* At this point, both sets of buffer pointers are set up.
* Set up the remainder of the CCB in the usual way to point
* to the first ISR. Notice that we wait
* until after the initial parameter checks to initialize the
* I/O operation timing services of the operating system. We
* do that now, and go to the first ISR.
*
         BAL    U8,TOCHON        go tell operating system we need timing done
         LIS    U0,0             reset the channel command word
         STH    U0,CCB.CCW(UC)   *
         LA     U0,ISR0          get address of first ISR
         STH    U0,CCB.SUBA(UC)  & save in CCB.
*
* Use the CCB.MISC field to allow the ISR routines
* to tell the ESR that a transfer failed, if this should
* happen. Thus, we initialize this field to zero.
```

```
*
         LHI   U0,0              get a zero
         STH   U0,CCB.MISC(UC)   *
         AIS   UC,1              make CCB addr odd for ISPT
         STH   UC,ISPTAB(U2,U2)  (set up ISPT)
         LH    U1,DCB.ILVL(UD)   get the interrupt level
         SINT  U1,0(U2)          go start the transfer
         B     DIRDONE           after first ISR, exit to do
*                                something else while I/O runs
*
INIT7ERR EQU   *                 initialization error exit
         LHI   U5,X'8490'        get a status code
         STH   U5,DCB.STAT(UD)   & save for user.
         LIS   U0,0              reset the length of transfer
         ST    U0,DCB.LLXF(UD)   for user
         B     IODONE            terminate the operation.
*
         PURE
ISR0     EQU   *                 come here to start transfer
         LA    E7,ISR1           get address of next ISR to use
         STH   E7,CCB.SUBA(E4)   and set up CCB for next ISR.
         L     E5,CCB.DCB(E4)    get DCB address
         LIS   E6,8              set up an 8 second time-out
         STH   E6,DCB.TOUT(E5)   (this is an arbitrary choice)
         OC    E2,CMD2           give the COMM-MUX command 2,
         OC    E2,CMD1           & command 1
*
* The autodriver channel operates in response to interrupts.
* Since we are now in an ISR, we can not issue another SINT
* to start the I/O operation. Instead, we utilize the SCP
* instruction to maintain the buffer pointer/counter in the
* CCB, and do a WD to the device. This causes a hardware interrupt,
* which is then handled by the autodriver channel firmware.
*
         LH    E6,CCW0           set the channel command word
         STH   E6,CCB.CCW(E4)    * for buffer 0
         SCP   E7,CCB.CCW(E4)    get a character from buffer 0
         WDR   E2,E7             write it to the device
         LPSWR E0                and exit back to init routine
         ALIGN 2
CCW0     DC    X'A084'          execute, buffer 0
CCW1     DC    X'A08C'          execute, buffer 1
CMD2     DB    X'F8'            COMM-MUX command 2
```

```
CMD1     DB     X'63'            COMM-MUX command 1
DISARM   DB     X'C0'            kill interrupts on COMM-MUX
         ALIGN 2
*
ISR1     EQU    *                 come here when buffer 0 done:
         LH     E7,CCB.LB0(E4)    did all of buffer 0 go out?
         BNP    ISRERR            if count not positive, error
         LA     E6,ISR2           set up the CCB for third ISR
         STH    E6,CCB.SUBA(E4)   * to be used.
         LH     E7,CCW1           set command word for buffer 1
         STH    E7,CCB.CCW(E4)    *
         SCP    E7,CCB.CCW(E4)    get 1st char from user's buffer
         WDR    E7,E2             and send to device
         LPSWR E0                 exit to wait for I/O
*
ISR2     EQU    *                 come here when buffer 1 done:
         LH     E7,CCB.LB1(E4)    did all of buffer 1 go out?
         BNP    ISRERR            if count not positive, error
         LA     E6,ISR3           set up CCB to point to last
         STH    E6,CCB.SUBA(E4)   * ISR routine to be used
         LA     E7,SUFFIX.E       set up CCB for different
         ST     E7,CCB.EB0(E4)    * buffer 0 pointers
         LA     E6,SUFFIX.S       now get the value for the
         SR     E6,E7             * count field, and then set
         STH    E6,CCB.LB0(E4)    * it up.
         SCP    E7,CCB.CCW(E4)    get 1st char from suffix buffer
         WDR    E7,E2             and send it to device
         LPSWR E0                 exit to wait for I/O
*
ISR3     EQU    *                 come here when all I/O done
         LH     E6,CCB.LB0(E4)    check that all of the last
         BNP    ISRERR            transfer completed.
ISRDONE  L      E5,CCB.DCB(E4)    get address of DCB of device
         LA     E7,III            clear the interrupt path
         STH    E7,ISPTAB(E2,E2)  *
         OC     E2,DISARM         kill device interrupts
         LCS    E6,1              indicate no time-out occurred
         STH    E6,DCB.TOUT(E5)   *
         L      E7,DCB.LEAF(E5)   schedule the termination
         ATL    E7,SQ             * routine for execution
         LPSWR E0                 and exit interrupt service.
ISRERR   EQU    *                 come here if I/O fails
         LHI    E6,X'8484'        set a status code for ESR
         STH   E6,CCB.MISC(E4)   *
         B     ISRDONE           & go exit ISR sequence
         IMPUR
TERMDRV7 EQU   *                 operating system enters here when I/O done
*
* The error checking sequence is somewhat arbitrary. Here, a time-out is
* the most important error; and failure to complete the transfer will be
* less important.
*
         LH    U0,DCB.TOUT(UD)   get current time-out value
         BZ    TIMEOUT7          error exit if timed-out
         LH    UC,DCB.CCB(UD)    get CCB address.
         LH    U0,CCB.MISC(UC)   check error indicator
         BNZ   IOFAIL7           if not zero, I/O failed
         LIS   U1,0              otherwise, indicate no error.
TERM7B   STH   U1,DCB.STAT(UD)   set error status
         BAL   U8,TOCHOFF        indicate no more timing.
*
* By design choice, we will report the length of transfer as only that part
* of the user's buffer that was actually transferred.  We use the count
* value for buffer 1 for this calculation:
*
         L     U0,DCB.EADR(UD)   get user's end address
         S     U0,DCB.SADR(UD)   less user's start address
         AH    U0,CCB.LB1(UC)    plus the residual buffer 1 count
         ST    U0,DCB.LLXF(UD)   equals count of bytes moved.
         B     IODONE            go to operating system I/O termination
TIMEOUT7 EQU   *                 set time-out status
         LHI   U1,X'8282'        *
         B     TERM7B
IOFAIL7  EQU   *                 set I/O failed status
         LHI   U1,X'8484'        *
         B     TERM7B
         ALIGN 4
*
* Define the suffix and prefix output by this driver
*
PREFIX.S DC    C'<PREFIX>'
PREFIX.E EQU   *-1
SUFFIX.S DC    C'<SUFFIX>'
         DB    X'00',X'0D',X'00',X'0A',X'00',X'00'
SUFFIX.E EQU   *-1
         END
```

Step 8: Generating User-Level Task Traps

In special circumstances, it is desirable to have the device driver add an item to the user's task queue, thereby generating a task trap. This feature is most likely to be used in applications that require buffer chaining, so that the u-task can be notified each time a data buffer has been processed by the driver.

The following small driver has been written to illustrate the code sequence required to add items to the user's task queue, and thus generate user-level task traps. This particular driver causes an item to be added to the user's task queue each time an I/O request is made to the driver. Note that this sample driver does not interface to a physical device; this is only because a physical device interface is not necessary in order to illustrate the add to task queue mechanism.

```
INITDRV8 EQU    *                 entry from operating system
         L      U9,DCB.TCB(UD)    get user's TCB address
         LI     UA,QUEITEM        get value to put on task queue
         BAL    U8,TMATQ1         branch to operating system
*                                 routine to add item to
*                                 the task queue.
         B      IODONE            exit from driver.
TERMDRV8 EQU    *                 not executed: used to
*                                 satisfy reference from
*                                 DCB.TERM entry in DCB.
*
QUEITEM  EQU    ...      (item definition is arbitrary)
         END
```

It should be noted that the routine "TMATQ1" is subject to being renamed in future revisions of the operating system. In revision 6.2 and lower, the label was "SV9.ATQ1". For this reason, the writer of a driver may want to replace the statement:

```
BAL    U8,TMATQ1
```

with the macro call:

```
ADDTTSKQ REASON=(UA),TASKSW=NO
```

This coding practice assures compatability with future releases of the operating system.

Step 9: Making Changes for IOP Execution

The following two drivers operate identically. However, the first driver does not execute correctly on an IOP. The second driver contains calls to the new macros needed to generate code that operate correctly under an IOP.

```
INITDUMB PROG DUMMY DRIVER FOR OS/32 REVISION 7.
*
* THIS DRIVER WILL NOT OPERATE CORRECTLY ON AN IOP
*
         MLIBS 8,9,10
         ENTRY INITDUMB                DRIVER INITIALIZATION
         ENTRY TERMDUMB                DRIVER TERMINATION
         EXTRN DIRDONE,IODONE,SQ,III,ISPTAB
INITDUMB EQU   *
         LH    U2,DCB.DN(UD)       GET THE DEVICE ADDRESS.
         LA    UF,III              SET NULL INTERRUPT VECTOR
         STH   UF,ISPTAB(U2,U2)    FOR THE DEVICE.
         L     UF,DCB.LEAF(UD)     GET LEAF TO SCHEDULE TERMINATION
         ATL   UF,SQ               BY PUTTING IT ON SYSTEM QUEUE.
         B     DIRDONE             EXIT TO OS ROUTINE.
*
TERMDUMB EQU   *
         B     IODONE              DRIVER IS DONE - QUIT NOW.
```

```
INITDUMB PROG DUMMY DRIVER FOR OS/32 REVISION 8.
*
* THIS CODE WILL EXECUTE CORRECTLY ON AN IOP
*
         MLIBS 8,9,10
         ENTRY INITDUMB                DRIVER INITIALIZATION
         ENTRY TERMDUMB                DRIVER TERMINATION
         EXTRN DIRDONE,IODONE,SQ,III,ISPTAB
INITDUMB EQU   *
         LH    U2,DCB.DN(UD)       GET THE DEVICE ADDRESS.
         LA    U6,III              GET THE NULL INTERRUPT VECTOR.
         ISPMOD ITEM=(U6),DCB=(UD),DN=(U2),WORK=(UF)
         L     UF,DCB.LEAF(UD)     GET LEAF TO SCHEDULE TERMINATION
         ADDSQ ITEM=(UF),DCB=(UD),WORK=(UE)
         B     DIRDONE             EXIT TO OS ROUTINE.
*
TERMDUMB EQU   *
         B     IODONE              DRIVER IS DONE - QUIT NOW.
```

2.3 WRITING DEVICE CONTROL BLOCK/CHANNEL CONTROL BLOCK (DCB/CCB) SPECIFICATIONS

Every device driver operates primarily on two associated data structures, the DCB and the CCB. It is the responsibility of the writer of the I/O driver to provide definitions of these two data structures, along with the actual I/O driver. The remainder of this chapter provides some procedures for specifying the DCB and CCB structures.

It is very important that the DCB and CCB specification be done exactly right the first time! An error in the DCB or CCB specification can result in any of the following symptoms:

1. Errors in the Sysgen32 phase of the sysgen procedure.
2. Errors in the macro expansion phase of the sysgen procedure.
3. Errors in the CAL assembly phase of the sysgen procedure.
4. Errors in the link phase of the sysgen procedure.
5. System crashes of all kinds when attempting to use the driver.
6. Other malfunctions of the driver.

The DCB definition contains all of the keyword definitions which can be used in the sysgen statement, such as the device name, device address, device code, controller address, SELCH address, and extended device codes. The DCB definition also provides a definition of all of the device-dependent data items that can be referenced by the driver, such as (for disks) the number of sectors per track, tracks per cylinder and cylinders per disk. Other device-dependent data can include special command tables for setting baud rates on asynchronous lines, and tables of tape motion speeds for magnetic tape drives. For some devices, data buffers or scratchpads can be included in the DCB definition.

Fortunately, most custom devices can utilize a DCB definition that is very nearly the same as a DCB definition already provided for a standard Perkin-Elmer device. Due to the large amount of detail that must be provided in the DCB definition, the best way to build a new DCB definition is to start with an existing DCB definition. The following steps provide a guide for building a new DCB definition:

1. Copy a DCB specification of a similar type of device from SYSGEN32.MLB into a new file, DCBxxx.MAC (xxx is the new device code, which must be in the range of 240 to 255). This can be done using MLU32, and listing the appropriate macro definition directly to DCBxxx.MAC. For example, a device that operates under a SELCH and supports both read and write, but does not support random access, is similar to a magnetic tape drive, so the macro definition DCB65 would probably be a good starting point. For example, the following session copies the definition of a magnetic tape driver into a file named DCB248.MAC:

```
*lo mlu32;st
PERKIN-ELMER OS/32 MACRO LIBRARY UTILITY 03-340 R00-01
MLU>get sysgen32.mlb/s
MLU>list dcb248.mac,DCB64
1 MACRO LISTED TO NEW FILE SYS:DCB248.MAC
MLU>end
RW      -END OF TASK CODE=    0     PROCESSOR=0.907  TSK-ELAPSED=38
```

2. In DCBxxx.MAC, change the DCB number (e.g., 65), to the new device code (e.g., 240) everywhere that it appears. Be careful not to miss any appearances of the old DCB number, but also be careful (if you are using EDIT32) not to modify any other character strings that just happen to match the new DCB number (e.g., 240).

3. After the line:

   ```
   CONVNUM
   ```

   Add the statement:

   ```
   USERINIT
   ```

4. Modify the parameters for the DCBI macro statement. This includes at least the INIT= and TERM= parameters, and probably the ATRB= parameter. This macro call specifies the values assumed by various data items within the DCB. Look over the DCBI macro statement carefully to be sure that you have adjusted all the terms that apply to your specific custom device. The device-dependent part of the DCB is specified in the DCBI macro statement with a COPY= parameter.

5. It is very unlikely that you will need to modify the CCBI macro statement, other than what was done in Step 2 above. The CCB is usually set up at run time by the driver. The possible exception would be a translation table name.

6. Add the statement

   ```
   DCB%DCOD%IDVAL    PROG    USER DCB
   ```

   after the CCBI macro statement, as shown in the example below. This provides a label that prints when the macro is assembled at sysgen time. This is a handy way to know if your DCB definition successfully passed that particular stage of the sysgen procedure.

7. Add the definition of any specific values in the device-dependent portion of the driver. This is done by the following constructs, which should be placed immediately preceding the $ST%OFFS statement near the end of the DCBxxx.MAC file:

   ```
   ORG    DCB%OFFS+DCB.yyyy
   DC     ......
   ```

   The label DCB.yyyy is a label defined in the device-dependent part of the DCB. The existence of the label, and its assigned value, are determined strictly by the requirements of your specific custom device.

8. After the statement

   ```
   ORG    $ST%OFFS
   ```

   Add the statements:

   ```
   ASIS
   END
   ```

If these eight steps are carefully done, the DCB definition will be complete.

To use the new DCB definition, which has been created in the file DCBxxx.MAC, it must be put into a macro library known as USERDLIB.MLB. This is done by using the MLU32 utility program.

Note that the device-dependent part of the DCB is determined by the specific requirements of your driver. In general, write your driver code first to determine what, if any, specific device-dependent fields you need, then include them in your DCB definition.

For programmers who are familiar with the now unsupported Configuration Utility Program (CUP), it is important to know that the use of Sysgen/32 requires that the DCB and the actual driver code be constructed in separate files, and placed in separate libraries. This is considerably different from the procedures used with CUP. The Sysgen/32 procedures require that the DCBxxx macro definition be placed in a macro library, USERDLIB.MLB; the driver code is expanded, assembled, and included in object format in an object library, USERDLIB.LIB. After this is done, the supplied SYSGEN.CSS can be used to perform the sysgen.

The major item of concern in writing the DCBXXX macro is the specification of the DCB macro. This macro defines the initialization, termination, function, and IOH routines used by the driver. Also specified here is the device code, attributes, and size of the DCB.

The DCB specification macro:

```
DCB DCOD = ,INIT = ,TERM =, ATRB = ,RECL = ,SIZE =, IOH =
```

DCOD - DCB number (gives each device type a unique code)

INIT - entry point to driver initialization routine.

TERM - name of first ESR to execute.

ATRB - attributes of the device

RECL - record length supported by the device

SIZE - size of the driver + dependent structures

IOH - name of I/O handler for this device.

A sample DCBXXX macro to be included in USERDLIB.MLB:

```
          MACRO
          DCB243 %DCOD=,%DN=,%CLAS=,%ILVL=,%NAME=,%SHCCB=,            1
                 %SIZE=,%RECLN=,%XDCD=,%IOP=0
          GBLC  %IDVAL
          BGBLA %ID243
          LCLA  %CCBFL
          LCLA  %CLASN
          LCLC  %RXLT,%RQU
          LCLC  %CORDNM,%PTRPAS
          LCLC  %OFFS
          LCLA  %RDN
          LCLC  %MDN,%MCNT,%MSLCH
          LCLA  %TRCNT,%UPTR
          LCLB  %FOUND,%DA
          BGBLA %FIRST
%RQU      SETC  'COMQ'               DEFAULT DEVICE QHANDLER
%MDN      SETC  '%DN'                DEVICE ADDRESS
%CCBFL    SETA  0
          AIF   (T'%CLAS EQ 'U')&CLSNTD
%CLASN    SETA  %CLAS*12            IOCLASS*12
```

```
&CLSNTD  ANOP
         CONVNUM VAL=%ID243    CONVERT CURRENT ID TO HEX.
         USERINIT
         $DCB$
         DCBI   DCOD=243,SIZE=DCB.DVDP,INIT=INITCORD,IOC=2,            x
                TERM=TERMCORD,FLGS=DFLG.LNM,RECL=132,ID=%IDVAL,        x
                ATRB=2B80,IOP=%IOP
         CCBI   DCOD=243,ID=%IDVAL
CCB%NAME EQU    CCB%DCOD%IDVAL
%ID243   SETA   %ID243+1
&DCBOPT  ANOP
DCB%DCOD%IDVAL  PROG   USER DCB
%OFFS    SETC   '%DCOD':'%IDVAL'      ESTABLISH PROPER OFFSET
DCB_%NAME EQU   DCB%OFFS
          ENTRY DCB_%NAME
          ORG   DCB%OFFS+DCB.DN         DEVICE ADDRESS
          DC    H'%DN'
          ORG   DCB%OFFS+DCB.LEAF       LEAF POINTER
          AIF   (T'%SHCCB' EQ 'U')&NSLEAF   B IF NOT SHARED
          DAC   LF%SHCCB              USE SHARED DEVICE LEAF
          EXTRN LF%SHCCB
          AGO   &NRMLFX
 &NSLEAF  ANOP
          DAC   LF%OFFS               GENERATE STANDARD LEAF NAME
          EXTRN LF%OFFS
 &NRMLFX  ANOP
 &NOLEAF  ANOP
          AIF   (T'%CLAS EQ 'U')&NOCLAS
          ORG   DCB%OFFS+DCB.CLAS       IO CLASS
          DC    H'%CLASN'             IOCLASS*12
 &NOCLAS  ANOP
          AIF   (T'%ILVL EQ 'U')&NOILVL
          ORG   DCB%OFFS+DCB.ILVL       ILEVEL
          DC    H'%ILVL'
 &NOILVL  ANOP
      AIF   (T'%XDCD EQ 'U')&NOXDCD   IF NOT ENTERED
      ORG   DCB%OFFS+DCB.XOPT             ELSE MOVE XDCD
      DC    %XDCD                   EXTENDED DCOD
```

```
&NOXDCD  ANOP
      ORG    DCB%OFFS+DCB.DMT
      DC     DMT_%NAME        A(DMT ENTRY)
      EXTRN  DMT_%NAME
      AIF    ('%RQU' EQ '')&NOQU
      ORG    DCB%OFFS+DCB.Q
      DAC    %RQU
      EXTRN  %RQU
&NOQU    ANOP
      AIF    (T'%RECLN EQ 'U')&NORECLN
      ORG    DCB%OFFS+DCB.RECL       RECORD SIZE
      DC     H'%RECLN'
&NORECLN ANOP
      ORG    $ST%OFFS             ORG TO END OF DCB
      ASIS
      END
%RDN     SETA  %DN+1
      MEND
```

Chapter 3

INCLUDING THE DRIVER IN YOUR OPERATING SYSTEM

CHAPTER 3

INCLUDING THE DRIVER IN YOUR OPERATING SYSTEM

3.1 INTRODUCTION

Special procedures are necessary at system generation (sysgen) time to include the driver into your operating system. These include expanding and assembling the driver before incorporating the driver into the sysgen configuration. Please note that although the Configuration Utility Program (CUP) is no longer supported, instructions for sysgening with CUPMT have been included for those users who work with older versions of OS/32.

3.2 EXPANDING AND ASSEMBLING DRIVERS

The conventional name for operating system and driver source modules is XXXX.MAC (where XXXX is a symbolic name for the device which the driver controls). The command substitution system (CSS) procedures which follow are typical procedures which can be used; however, these can be modified for individual installations.

```
*EXPAND.CSS <name of module - (XXXX)>

XAL @1.CAL,IN,80/8
L .BG,MACRO32,50
T .BG
AS 1,@1.MAC,SRO
AS 2,@1.CAL
AS 3,NULL:           (this is the list file)
AS 7,SYSGEN.MAC/S,SRO
AS 8,SYSSTRUC.MLB/S,SRO
AS 9,SYSMACRO.MLB/S,SRO
AS 10,DVRM.MLB/S,SRO
AS 11,ITMS.MLB/S,SRO
AS 12,SYSMAC32.MLB/S,SRO
ST,MLIBS=(8,9,10,11,12)
$EXIT
```

```
*ASSEMBLE.CSS <name of module - (XXXX)>

XAL @1.OBJ,IN,126
XAL @1.LST,IN,132/10
L .BG,CAL32,80
T .BG
AS 1,@1.CAL,SRO
AS 2,@1.OBJ
AS 3,@1.LST
ST,CROSS,SQUEZ,NLSTM,NUREX,NLSTU
$EXIT
```

The above code expands and assembles both CUP- and Sysgen/32-generated drivers. However, if it is Sysgen/32, the device control block (DCB) requires special code which is explained below.

After the macro definition for the device is prepared, it must be included in the user's macro library. If this library, USERDLIB.MLB, does not currently exist, it must be created by the macro library utility.

```
L  MLU32
ST
> ESTABLISH USERDLIB.MLB
> INCLUDE DCB240.MAC    (include user DCB/CCB definition)
> S*
> END
```

If the macro library already exists, use the GET command rather than the ESTABLISH command. If you are replacing DCB240 with a newer version, use the DELETE command before you do the include.

```
L MLU32
ST
> GET USERDLIB.MLB
> DIR     (this lists all macros in the library
DCB240
> DELETE DCB240
> INCLUDE DCB240.MAC  (include the new DCB/CCB definition)
> S*
> END
```

For CUPMT-generated systems, it is a requirement that the NUREX option to Common Assembly Language/32 (CAL/32) be used. If this option is not used, FRMT errors from CUPMT may be generated.

3.3 SYSTEM GENERATION (SYSGEN) WITH CUPMT

For those users who have software release R06 or lower, this section is necessary. If you have software release R07 or higher, you may continue on to Section 3.4.

The following is a discussion of the relationship of the statements in the CUP file to elements in the DCB.

```
*Sample device configuration statement in CUP file.

1 DEV:64,247
```

where:

1	is a control number for CUP
DEV	is the device mnemonic name by which the user references the device through the "assign" function. This does NOT appear in the DCB. However, the address where this name can be found appears at DCB.DMT.
64	is the physical device address is determined by the hardware configuration. This entry in the configuration statement must agree with hardware address settings. This value appears in the DCB as the 16-bit (halfword) value DCB.DN.
247	is the device type code that is defined in the DCB.DCOD field. This is the code used by CUP to select a particular device driver. The DCB definition statement DCB DCOD=247 corresponds to this value.

To build a new version of OS/32 incorporating a custom device interface, the following files are required. All of these files, with the exception of USERDLIB.LIB (the file you will build to contain your driver) are supplied by Perkin-Elmer with OS/32:

```
SYSGEN1.CSS   SYS.LIB              CUPMT.TSK
SYSGEN2.CSS   ITBSYS.LIB           LIBLDR.TSK
SYSGEN3.CSS   DRIVER.LIB           LINK.TSK
              ITBDLIB.LIB
              UBOT.OBJ
              USERDLIB.LIB
              configuration file
```

The "SYSGEN1" procedure provided with OS/32 software packages assumes that all customer-provided drivers are contained in a file named USERDLIB.LIB, in whatever account is used for doing the sysgen procedure. Therefore, the incorporation of customer drivers requires only that the customer driver source file be macro-expanded, assembled and the resultant object file be placed in a file known as USERDLIB.LIB. These procedures can be customized to your installation (i.e. list files can be assigned instead of the printers, etc.) The following procedure can be used to build USERDLIB.LIB.

```
$BUILD USERDLIB.CMD
FI 2 DCBXXX
CO 2,1
RW 2
CO 2,1
END
$ENDB
L .BG,LIBDLR
T .BG
AS 1,USERDLIB.LIB
AS 2,DRVR.OBJ,SRO
AS 5,USERDLIB.CMD
ST
```

The following is a sample of a configuration file:

```
ACCOUNTING 4
VERSION 11-18-83                 * 3210
CPU 3210                         * 3210 WITH 8 REGISTER SETS
MEMORY 4096                      * 4 MEGABYTES OF MEMORY
CLOCK 60,6C,6D                   * STANDARD CLOCK ADDRESSES
BACKGROUND 128,50                * .BG MAXPRI=128,MAXSYS1=50KB
CMDLEN 80                        * COMMAND LENGTH = 80 BYTES
CSS 8                            * CSS NESTING LEVEL = 8
DATE MMDDYY                      * U.S. DATE FORMAT
DEVADS 0                         * MAX DEVICE ADDR = X'FF'
DIRECTORY                        * CORE DIRECTORY SUPPORT
DISCBLOCK 255                    * MAX BLOCK SIZE = 255 SECTORS
DSYS 200                         * SYS SPACE ALLOCATION
ERRORREC SYS:ERROR.LOG,200,5     * SUPPORT MEMORY ERROR LOGGING
FLOAT H,H                        * SOFTWARE FLOATING POINT
ITAM
JOURNAL 0                        * NO JOURNAL SUPPORT
LOGLEN 80                        * LOG MESSAGE BUFFER LENGTH
MAXTASK 40                       * 40 CONCURRENT TASKS
MODULE
INTC.F02
ENDM
ROLL SYS                         * ROLL VOLUME = SYS
SPOOL SYS                        * SPOOL VOLUME = SYS
TEMP SYS                         * TEMP VOLUME = SYS
VOLUME SYS                       * SYSTEM VOLUME = SYS
DEVICES                          * BEGIN DEVICE STATEMENTS
```

```
1 CON:10,39,C,XD                   * CONSOLE-550 ON COM-MUX
1 CRT1:12,39,,XD                   * CRT AT ADDRESS X'12'
1 CRT2:20,39,,XD                   * CRT AT ADDRESS X'20'
1 CRT3:22,39,,XD
1 CRT4:24,39,,XD
1 CRT5:26,39,,XD
1 CRT6:28,39,,XD
1 CRT7:2A,39,,XD
1 CRT8:2C,39,,XD
1 CRT9:2E,39,,XD
1 LP:62,113                        * LINE PRINTER
1 PR:0,0,S                         * SPOOL DEVICE
1:F1,0                             * SELCH F1--FOR DISK DEVICES
2:0,0                              * MAG TAPE CONTROLLER
3 MAG1:C5,70                       * 6250 BPI TELEX TAPE AT X'c5'
2:EB,0
3 D80:EC,53,D
3 D300:ED,54,D
1 XXXX:64,247                      * user developed device
ENDD
ENDC
```

First Step of Sysgen Procedure:

```
*
**SYSGEN1.CSS     [CUP FD],[SEGSIZE]

$IFNULL @1
  $WR ***@0: CONFIGURATION STATEMENT FD OMITTED
  $CLEAR
$ENDC
EXIST @1
EXIST DRIVER.LIB/S
EXIST ITBDLIB.LIB/S
XDE CUPOUT.OBJ
AL  CUPOUT.OBJ,IN,126/2
$IFNULL @2
  LO .BG,CUPMT
$ELSE
  LO .BG,CUPMT,@2
$ENDC
TA .BG
```

```
AS 1,@1,SRO
AS 2,CUPOUT.OBJ
AS 3,PR:
$IFX USERDLIB.LIB
  AS 4,USERDLIB.LIB,SRO
$ELSE
  AS 4,NULL:
$ENDC
AS 5,DRIVER.LIB/S,SRO
$IFX ITED2780.LIB/S
  AS 6,ITED2780.LIB/S,SRO
$ELSE
  AS 6,NULL:
$ENDC
$IFX ITEDZDLC.LIB/S
  AS 7,ITEDZDLC.LIB/S,SRO
$ELSE
  AS 7,NULL:
$ENDC
$IFX ITED327S.LIB/S
  AS 8,ITED327S.LIB/S,SRO
$ELSE
  AS 8,NULL:
$ENDC
$IFX ITED327E.LIB/S
  AS 9,ITED327E.LIB/S,SRO
$ELSE
  AS 9,NULL:
$ENDC
$IFX ITBDLIB.LIB/S
  AS 10,ITBDLIB.LIB/S,SRO
$ELSE
  AS 10,NULL:
$ENDC
$IFX DRIVER.LIB/S
  AS 11,DRIVER.LIB/S,SRO
$ELSE
  AS 11,NULL:
$ENDC
START
$IFNE 0
   $WR ***@0: ERRORS DETECTED BY CUP
   DEL CUPOUT.OBJ
```

```
 $CLEAR
$ENDC
$WR *** ENTER SYSGEN2  IMPURE,PURE [,SEGSIZE]
$EXIT
```

Second Step of Sysgen Procedure:

```
**SYSGEN2.CSS     [IMPURE],[PURE],[SEGSIZE]

$IFNULL @1
  $WR ***@0: MISSING PARAMETER (IMPURE BIAS)
  $CLEAR
$ENDC
$IFNULL @2
  $WR ***@0: MISSING PARAMETER (PURE BIAS)
  $CLEAR
$ENDC
EXIST SYS.LIB/S
EXIST ITBSYS.LIB/S
EXIST UBOT.OBJ/S
EXIST CUPOUT.OBJ
XAL LIBLDOUT.OBJ,IN,126
XDEL   LIBLDR.CMD
$BUILD LIBLDR.CMD
TO FFFFF
OU 2
BI @1
PB @2
LO 1
ED 1
ED 4
ED 6
ED 7
ED 8
ED 9
ED 10
ED 11
ED 12
ED 13
XO
MA 3
AM 3
EN
$ENDB
```

```
$IFNULL @3
  LO .BG,LIBLDR
$ELSE
  LO .BG,LIBLDR,@3
$ENDC
TA .BG
AS 1,CUPOUT.OBJ
AS 2,LIBLDOUT.OBJ
AS 3,PR:
AS 5,LIBLDR.CMD,SRO
AS 4,SYS.LIB/S,SRO
$IFX USERSYS.LIB
  AS 6,USERSYS.LIB,SRO
$ELSE
  AS 6,NULL:
$ENDC
$IFX ITES2780.LIB/S
  AS 7,ITES2780.LIB/S,SRO
$ELSE
  AS 7,NULL:
$ENDC
$IFX ITESZDLC.LIB/S
  AS 8,ITESZDLC.LIB/S,SRO
$ELSE
  AS 8,NULL:
$ENDC
$IFX ITES327S.LIB/S
  AS 9,ITES327S.LIB/S,SRO
$ELSE
  AS 9,NULL:
$ENDC
$IFX ITES327E.LIB/S
$ELSE
  AS 10,NULL:
$ENDC
$IFX MCONFIG.OBJ
  AS 11,MCONFIG.OBJ
$ELSE
  AS 11,NULL:
$ENDC
AS 12,ITBSYS.LIB/S,SRO
AS 13,UBOT.OBJ/S,SRO
START
```

```
$IFNE O
  $WR ***@O: ERRORS DETECTED BY LIBLDR
  XDEL LIBLDOUT.OBJ,LIBLDR.CMD
$CLEAR
$ENDC
XDEL CUPOUT.OBJ,LIBLDR.CMD
$WR *** ENTER SYSGEN3  ,OSFD [,SEGSIZE]
$EXIT
```

Third Step of Sysgen Procedure.

```
**SYSGEN3.CSS     [],[OS FD],[SEGSIZE]
**
$IFNULL @2
   $WR ***@O: MISSING PARAMETER (OUTPUT FD)
   $CLEAR
$ENDC
EXIST LIBLDOUT.OBJ
XDEL @2
$BUILD LINKOS.LNK
ESTAB OS
MAP PR:
INCL LIBLDOUT.OBJ
BUILD @2
END
$ENDB
$IFNULL @3
  LO .BG,LINK
$ELSE
LO .BG,LINK,@3
$ENDC
TA .BG
ST ,C=LINKOS.LNK,L=PR:
$IFNE O
   $WR ***@O: ERRORS DETECTED BY LINK
   XDE LINKOS.LNK,@2
   $CLEAR
$ENDC
XDE LINKOS.LNK,LIBLDOUT.OBJ
$WR *** OS @2 LINKED ***
$EXIT
```

3.4 SYSTEM GENERATION (SYSGEN) WITH SYSGEN/32

The following is a discussion of Sysgen/32 device specification statements.

The device has the following characteristics:

```
Device Name = NEWD
Address = X'64'
Device Code = 247
```

This device operates on the multiplexor (MUX) bus:

```
DEVICES
NEWD:,64,247
ENDD

       DEVICES
       NEWD:,64,247,SELCH=FO,CONTR=0
       ENDD

COPY
        MCALL DCBI,CCBI,CONVNUM,EVNGEN,MMDGEN
ENDCOPY
ACCOUNTING = 4,NOFILEACCOUNTING        *4 accounting classes
BACKGROUND = 128,50                    *.bg maxpri=128,sys=50KB
CLOCK = 60,6C,6D                       *standard clock addresses
CMDLEN = 80                            *80 byte command buffer
CPU = 3210                             *processor model 3210
CSS = 8                                *8 levels of CSS nesting
DATE = MMDDYY                          *U.S.A. date format
DEVADS = 0                             *less than 256 devices
DIRECTORY                              *secondary directory support
DISCBLOCK = 255                        *permit max blocking
DSYS = 1024                            *1 MB of system space
ERRORREC = SYS:ERROR.LOG,1024,2        *error log readout .. 2 min.
FLOAT = H,H                            *hardware floating point
ILEVEL = 0                             *all interrupts on level 0
INTERCEPT                              *permit SVC interception
IREADER                                *permits use of SVC 2,14
ITAM
JOURNAL = 0                            *don't waste time on journal
LOGLEN = 80                            *message buffer size
MAXTASK = 32                           *default for max# of tasks
MCONFIG  BLOCK=0,START=0,RANGE=4,INTERL=0  *mem config for err log
MEMCHECK                               *verify mem exists at IPL
MEMORY = 4096                          *4 MB system
```

```
SPL32                                           *spool32 support
SSTABLE = 32                                    *shared segment tbl entries
TEMP = SYS                                      *temp volume = 'SYS'
VERSION = OS321212                              *6.2 os build on Dec 12.
VOLUME = SYS                                    *system volume = 'SYS'
DEVICES                                         * BEGIN DEVICE STATEMENTS
* System console.
CON:,10,39,CONSOLE,CLOCK=XD                     *system console-fast clock
* Crt type devices.
*  a. Non-bioc CRT driver is selected by USER=(CRT=1).
*  b. Fast clock is selected by XD.
NEC:,12,40,REC=132,SIZ=66,XD=X6966
IPC:,12,39,REC=132,CLOCK=XB
CRT2:,20,156,XDC=X0030,RECL=132,PAD=3
WP02:,20,39,CLOCK=XD
CRT3:,22,156,XDC=X0030,RECL=132,PAD=3
WP03:,22,39,CLOCK=XD
CRT4:,24,156,XDC=X0030,RECL=132,PAD=3
WP04:,24,39,CLOCK=XD
CRT5:,26,156,XDC=X0030,RECL=132,PAD=3
WP05:,26,39,CLOCK=XD
CRT6:,28,156,XDC=X0030,RECL=132,PAD=3
WP06:,28,39,CLOCK=XD
CRT7:,2A,156,XDC=X0030,RECL=132,PAD=3
WP07:,2A,39,CLOCK=XD
*CRT8:,2C,156,XDC=X0030,RECL=132,PAD=3
MODA:,2C,156,XD=X0810,RECL=132
WP08:,2C,39,CLOCK=XD
CRT9:,2E,156,XDC=X0030,RECL=132,PAD=3
WP09:,2E,39,CLOCK=XD
*printers
LP:,62,113
D80A:,EC,53,SELCH=F1,CONTR=EB
D300:,ED,54,SELCH=F1,CONTR=EB
MAG1:,C5,68,SELCH=F1,CONTR=0
XXXX:,64,247                                     *user developed device
ENDD
ENDC
```

Once the specifications have been placed in the configuration file (named xxxxxxxx.sys), sysgen is performed by typing SYSGEN xxxxxxxxx, from account 0. The procedures can be modified to look for the macro libraries and other files required on any account which you choose. The following procedures are used with Sysgen/32 for the sysgen procedure:

```
**SYSGEN.CSS
**
**  THIS CSS IS THE FIRST ONE USED TO RUN THE SYSGEN32 PROCEDURE
**
**  @1 = CONFIGURATION STATEMENT INPUT FILE                (REQUIRED)
**  @2 = CORE INCREMENT FOR SYSGEN32 TASK                  (OPTIONAL)
**  @3 = VOLUME FOR LIBRARIES  (.MLB & .LIB)               (OPTIONAL)
**  @4 = MACRO AND CAL LISTING FLAG.  @1.PRT FILENAME      (OPTIONAL)
**       USED DEFAULT IS ERROR MESSAGES TO @1.PRT ONLY.
**  @5 = FILE SAVE FLAG.  CSS WILL SAVE ALL INTERMEDIATE   (OPTIONAL)
**       FILES CREATED IN THE SYSGEN PROCESS.  DEFAULT IS
**       TO DELETE ALL INTERMEDIATE FILES.
**  @6 = MACRO & CAL LIST FILE- DEFAULT IS @1.PRT          (OPTIONAL)
**
$WR *   SYSGEN.CSS
$WR *
$WR *       STATEMENT FORMAT:  SYSGEN CS,CI,V,LF,SF,MCL
$WR *
$WR *   CS = CONFIGURATION STATEMENT INPUT FILE            (REQUIRED)
$WR *   CI = CORE INCREMENT FOR SYSGEN32 TASK              (OPTIONAL)
$WR *   V  = VOLUME FOR LIBRARIES  (.MLB & .LIB)           (OPTIONAL)
$WR *   LF = MACRO AND CAL LISTING FLAG.                   (OPTIONAL)
$WR *        DEFAULT IS ONLY ERROR MESSAGES TO @1.PRT.
$WR *   SF = FILE SAVE FLAG.  CSS WILL SAVE ALL            (OPTIONAL)
$WR *        FILES CREATED IN THE SYSGEN PROCESS.
$WR *        DEFAULT IS TO DELETE ALL INTERMEDIATE FILES.
$WR *   MCL= MACRO & CAL LIST FILE- DEFAULT IS 'CS'.PRT    (OPTIONAL)
$WR *
$WR *    EXAMPLE:    SYSGEN OS32SYS,30,M300:,,,CON:
$CL;$ENDC

SYSGEN32 @1,@2

**SYSGEN32.CSS
**
**  THIS CSS IS USED TO RUN THE FIRST PART OF SYSGEN32 PROCEDURE
**
**  @1 = CONFIGURATION STATEMENT INPUT FILE                (REQUIRED)
**  @2 = SIZE INCREMENT FOR TASK                           (OPTIONAL)
**
$IFNULL @1; $WR *** INPUT FILENAME MISSING ***; $CLEAR; $ENDC
$IFNX @1.SYS
$WR *** INPUT FILE @1.SYS DOES NOT EXIT, SYSGEN32 ABORTED ***; $CL
```

```
$ENDC
XDE @1.MAC
XAL @1.LST,IN,132/5
$IFNULL @2
LO .BG,SYSGEN32;    TA .BG
$ELSE
LO .BG,SYSGEN32,@2; TA .BG
$ENDC
ST ,IN=@1.SYS,OUT=@1.MAC,LIST=@1.LST
$IFNE 0; $WR *** SYSGEN32 ERROR ***; $CLEAR; $ENDC
$EXIT
SYSMACRO @1,@3,@4,@5,@6
SYSLINK  @1,@3,@5
$EXIT
**SYSMACRO.CSS
**
** THIS CSS IS THE SECOND ONE USED IN THE SYSGEN32 PROCEDURE
** IT WILL EXPAND AND ASSEMBLE THE MACROS GENERATED BY SYSGEN32
**
**  @1 = CONFIGURATION INPUT FILE NAME                    (REQUIRED)
**  @2 = VOLUME WITH MACRO LIBRARIES                      (OPTIONAL)
**  @3 = LISTING PROVIDED IF NOT NULL                     (OPTIONAL)
**  @4 = DO NOT DELETE FILES                              (OPTIONAL)
**  @5 = LIST FILE- DEFAULT IS @1.PRT                     (OPTIONAL)
**
$IFNULL @1; $WR *** INPUT FILENAME MISSING ***; $CLEAR; $ENDC
$IFNX @1.MAC
$WR *** MACRO FILE @1.MAC DOES NOT EXIST, SYSGEN ABORTED ***; $CL
$ENDC
XAL @1.CAL,IN,80/20
LO .BG,MACRO32/S,50; TA .BG
AS 1,@1.MAC,SRO
AS 2,@1.CAL,EWO
$IFNULL @5
XAL @1.PRT,IN,132
AS 3,@1.PRT,SWO
$ELSE
AS 3,@5,SWO;$ENDC
MLBCK @2SYSGEN32,8
MLBCK @2DVRM,9
MLBCK @2ITMS,10
MLBCK @2SYSSTRUC,11
MLBCK @2SYSMACRO,12
```

```
*MLBCK @2ITED327S,13
*MLBCK @2ITED327E,14
$IFX    USERDLIB.MLB
AS 7,USERDLIB.MLB,SRO
$IFNULL @3
ST ,MLIB=(7,8,9,10,11,12),BATCH,MLIST=(ND,NG)
*   ST ,MLIB=(7,8,9,10,11,12,13,14),BATCH,MLIST=(ND,NG)
$ELSE
ST ,MLIB=(7,8,9,10,11,12),BATCH
*   ST ,MLIB=(7,8,9,10,11,12,13,14),BATCH
$ENDC
$ELSE
$IFNULL @3
ST ,MLIB=(8,9,10,11,12),BATCH,MLIST=(ND,NG)
*  ST ,MLIB=(8,9,10,11,12,13,14),BATCH,MLIST=(ND,NG)
$ELSE
ST ,MLIB=(8,9,10,11,12),BATCH
*  ST ,MLIB=(8,9,10,11,12,13,14),BATCH
$ENDC
$ENDC
$IFNE 0
$WR *** ERRORS IN MACRO EXPANSION, SYSGEN ABORTED ***; $CL
$ENDC
***
*** ASSEMBLE THE EXPANDED MACRO CODE ****
***
$IFNULL @4; XDE @1.MAC; $ENDC
XAL @1.OBJ,IN,126/5
LO .BG,CAL32/S,50; TA .BG
AS 1,@1.CAL,SRO
AS 2,@1.OBJ,EWO
$IFNULL @5
AS 3,@1.PRT,SWO
$ELSE
AS 3,@5,SWO;$ENDC
TE 5,IN,256/5/10
$IFNULL @3; ST ,BATCH,NOSQZ,NLIST
$ELSE;      ST ,BATCH,NOSQZ,NLSTU,NFREZ; $ENDC
$IFNE 0
$WR *** ERRORS IN ASSEMBLY, SYSGEN ABORTED ***; $CL
$ENDC
$IFNULL @4; XDE @1.CAL; XDE @1.PRT; $ENDC
$EXIT
```

```
**SYSLINK.CSS
**
** THIS CSS IS THE LAST USED IN THE SYSGEN PROCEDURE
** IT WILL LINK THE OBJECT CREATED BY CAL WITH THE APPROPRIATE
** LIBRARIES TO PRODUCE THE OS.  THE MAP FOR THE OS WILL BE
** APPENDED TO THE LIST FILE.
**
**  @1 = CONFIGURATION INPUT FILE NAME                     (REQUIRED)
**  @2 = VOLUME WHERE SYSTEM AND DRIVER LIBRARIES RESIDE   (OPTIONAL)
**  @3 = SAVE LINK CMD FILE - DEFAULT IS DELETE            (OPTIONAL)
**
$IFNULL @1
$WR *** INPUT FILENAME MISSING ***
$CLEAR
$ENDC
$IFNX @1.OBJ
$WR *** OBJECT OF OS @1.OBJ MISSING ***
$CLEAR
$ENDC
$BUILD @1.LNK
NLOG
ESTAB OS
INCL @1.OBJ
$ENDB
SYSCHECK @1,USERDLIB.LIB,1
SYSCHECK @1,USERSYS.LIB,1
SYSCHECK @1,@2DRIVER.LIB
SYSCHECK @1,@2SYS.LIB
SYSCHECK @1,@2ITBDLIB.LIB,1
SYSCHECK @1,@2ITBSYS.LIB
SYSCHECK @1,@2ITES2780.LIB,1
SYSCHECK @1,@2ITED2780.LIB,1
SYSCHECK @1,@2ITEDZDLC.LIB,1
SYSCHECK @1,@2ITESZDLC.LIB,1
*SYSCHECK @1,ITED327S.LIB,1
*SYSCHECK @1,ITED327E.LIB,1
SYSCHECK @1,@2UBOT.OBJ
$BUILD @1.LNK,APPEND
MAP @1.LST,ADDR,ALPHA
BUILD @1.OS
END
$ENDB
XDE @1.OS
```

```
LO .BG,LINK.TSK/S,100; TA .BG
ST ,COMM=@1.LNK,LOG=NULL:
$IFNE 0; $WR *** LINK ERROR ***; $CLEAR; $ENDC
$IFNULL @3
XDE @1.LNK;$ENDC
$WR      OS MAP ==>  @1.LST
$WR      OS OBJ ==>  @1.OBJ
$WR      OS TSK ==>  @1.OS
$EXIT
*
```

3.5 CUPMT VS. SYSGEN/32

It is appropriate to introduce here a discussion of the major differences between CUPMT and SYSGEN/32. The differences lie in the specification of a shared busy condition and specification of controllers without specific device addresses.

A shared busy condition (a common leaf) must exist if a device shares the same address as another device. (This is not necessarily true for pseudo-devices.) When generating a system with CUPMT, this condition was specified explicitly by an asterisk as the control number, in column 1 of the device(s) sharing the busy condition. With Sysgen/32, the shared busy condition is set automatically by the program if the device address specifications are the same.

Under CUPMT, a shared busy condition must be explicitly declared:

```
3 D80F:FC,62,D
* D16R:FC,59,D
```

Under CUPMT, controllers are on separate SELCHs:

```
1:F0,0
2:0,0       specify the controller as zero
3 MAG1:85,65
1:F1,0
2:0,0       specify the controller as zero
3 MAG2:C5,65
```

Under Sysgen/32, a shared busy condition is generated automatically when the same address is specified:

```
D80F:,FC,62,SELCH=F0,CONTR=FB
D16R:,FC,59,SELCH=F0,CONTR=FB
```

In addition, under SYSGEN32 controllers are on separate SELCHs:

```
MAG1:,85,65,SELCH=F0,CONTR=0
MAG2:,C5,65,SELCH=F1,CONTR=1
```

CHAPTER 4

SAMPLE DRIVERS

CHAPTER 4

SAMPLE DRIVERS

4.1 INTRODUCTION

This chapter provides two sample drivers as models. The first, a digital input/output (DIO) driver, is a very simple and typical driver. The other sample driver, a TELEX tridensity tape driver, is a complex driver containing many of the possible features that can be found in a driver.

4.2 SAMPLE DRIVER FOR DIGITAL INPUT/OUTPUT (DIO) INTERFACE

The driver for DIO is a typical driver needed by users seeking an interface with the operating system. It is a very simple driver written under a selector channel (SELCH), and it provides an interface in a situation where the configuration is unlike the one supported by the standard Perkin-Elmer driver. In this driver, starting in column 73, a description of the code, i.e., the driver initialization routine (DIR), interrupt service routine (ISR), event service routine (ESR) and I/O handlers (IOH), is provided. Where more than one of these routines occur, a sequence code (1-9, or a-z) follows the basic description.

```
* BASIC DRIVER LOGIC - CASE 7
*
* THE FOLLOWING  DRIVER  IS  PROVIDED  TO  ILLUSTRATE  PROGRAMMING OF A
* RELATIVELY STRAIGHT FORWARD DIO DEVICE
*
         BATCH
**SDIO
         MLIBS 8,9,10,11
SDIO     $DVPROG TAL I/O,05, 07-083          $REGS$
         $DCB$                                                           structs
         $CCB                                                            structs
         $TCB                                                            structs
         ENTRY INITSDIO,TERMSDIO                                         entrys
         EXTRN IODONE,SQ,ISPTAB,DIRDONE,III                              extrns
         EXTRN TOCHON,TOCHOFF                                            extrns
         TITLE INITIATION PHASE FOR DIGITAL I/O
*  INITSDIO   EQU *
         LHI   U1,X'8600'          POSSIBLE ERROR CODE: SET UP           dir
         L     U6,DCB.SADR(UD)     GET START ADDRESS                     dir
         THI   U6,1                IS IT ON HW BOUNDARY?                 dir
         BNZ   SDIOSTAT            NO, ERROR                             dir
         LHI   U1,X'C000'          POSSIBLE ERROR CODE: SET UP           dir
         LH    U9,DCB.DN(UD)       GET DEVICE ADDRESS                    dir
         LH    UA,DCB.FC(UD)       GET FUNCTION CODE                     dir
         THI    UA,X'4000'          IS IT A READ?                       dir
         BNZ    SDIOREAD            YES, BRANCH                         dir
         THI    U9,1                IS DEV ADDR ODD?                    dir
         BNZ    SDIOSTAT            YES, ERROR                          dir
         B      SDIOCOM                                                 dir
SDIOREAD EQU    *                                                       dir
         THI    U9,1                IS DEV ADDR EVEN?                   dir
         BZ     SDIOSTAT            YES, ERROR                          dir
SDIOCOM  EQU    *                                                       dir
         LH     UB,DCB.SDN(UD)      GET SELCH ADDRESS                   dir
         LHI    U1,X'A000'          POSSIBLE ERROR CODE: SET UP         dir
         OC     UB,STOPSEL          RESET THE SELCH                     dir
         BTC    4,SDIOSTAT          IF FALSE SYNC, ISSUE D.U. STAT      dir
         LHI    U2,X'7FFF'          SET TIME OUT TO 'DON'T CARE'        dir
         STH    U2,DCB.TOUT(UD)     * UNTIL ISR IS STARTED              dir
         LIS    U2,0                CLEAR DEV STAT - WE WILL  USE THIS  dir
         STH    U2,DCB.STAT(UD)     * AS A FLAG BETWEEN ISR AND ESR.    dir
         BAL    U8,TOCHON           GO GET ON TIMER CHAIN.              dir
         LHL    UC,DCB.CCB(UD)      GET CCB                             dir
         LA     U2,ISRSDIO          ISR RETURN ADDRESS                  dir
         STH    U2,CCB.SUBA(UC)     PUT IN CCB                          dir
         LA     U2,1(UC)            GET CCB ADDR + 1                    dir
         LH     UB,DCB.SDN(UD)      GET SELCH ADDRESS                   dir
         STH    U2,ISPTAB(UB,UB)    PUT IN ISPTAB                       dir
         LH     U8,DCB.ILVL(UD)     GET INTERRUPT LEVEL                 dir
         SINT   U8,0(UB)            FIRE UP SELCH                       dir
         B      DIRDONE                                                 dir
         ALIGN  2
STOPSEL  DC     X'4800'
         ALIGN  4
SDIOSTAT EQU    *                                                       err
         LIS    U2,0                                                    err
         ST     U2,DCB.LLXF(UD)     ZERO LLXF                           err
         STH    U1,DCB.STAT(UD)     STORE IN STATUS                     err
         B      IODONE              QUIT                                err
*
* TRANSFER INITIATION INTERRUPT
*
         PURE                                                           isr1
ISRSDIO  EQU    *                                                       isr1
         L      E5,CCB.DCB(E4)      GET DCB ADDRESSS                    isr1
         THI    E3,X'34'            CHECK STATUS ON SELCH               isr1
         BNZ    FALSYNC             DEVICE UNAVAILABLE                  isr1
```

```
          LH    E6,DCB.DN(E5)        GET DEVICE NUMBER                   isr1
          OC    E6,X80               DISABLE INTERRUPTS FROM DIO         isr1
          BTC   4,FALSYNC            IF DIO GIVES FALSE SYNC, EXIT NOW   isr1
          OC    E2,STOPSLCH          WE ARE NOW COMMITTED TO DOING THE   isr1
          WD    E2,DCB.SADR+1(E5)    * TRANSFER, SO GET THE START AND    isr1
          WH    E2,DCB.SADR+2(E5)    * END ADDRESSES WRITTEN TO THE SELCH isr1
          WD    E2,DCB.EADR+1(E5)    *                                   isr1
          WH    E2,DCB.EADR+2(E5)    *                                   isr1
          LA    E3,ISR1SDIO                                              isr1
          STH   E3,CCB.SUBA(E4)      ISR RETURN ADDRESS                  isr1
          LIS   E3,5                 SET 5 SECOND TIMEOUT.               isr1
          STH   E3,DCB.TOUT(E5)                                          isr1
*                                                                        isr1
          LH    E7,DCB.FC(E5)        GET FUNCTION CODE                   isr1
          THI   E7,X'4000'           IS IT A READ?                       isr1
          BNZ   ISRSELRD             YES, BRANCH                         isr1
          WH    E6,X00               PRIME OUTPUT DEVICE                 isr1
          OC    E2,SELCHWRT          START SELCH                         isr1
          LPSWR E0                                                       isr1
ISRSELRD EQU    *                                                        isr1
          RHR   E6,E7                PRIME INPUT DEVICE                  isr1
          OC    E2,SELCHRD           START SELCH                         isr1
          LPSWR E0                                                       isr1
*
FALSYNC  EQU    *                                                        isr2
          LHI   E6,X'A000'           DEVICE UNAVAILABLE                  isr2
          STH   E6,DCB.STAT(E5)      SAVE IN STATUS                      isr2
          B     ISRDSARM                                                 isr2
X80      DC     X'8000'
X00      DC     X'0000'
SELCHWRT DB     X'14'
SELCHRD  DB     X'34'
STOPSLCH DB     X'48'
          ALIGN 4
*
* OPERATION COMPLETE INTERRUPT.                                          isr2
*                                                                        isr2
ISR1SDIO EQU    *                                                        isr2
          L     E5,CCB.DCB(E4)       GET DCB ADDRESS                     isr2
          THI   E3,X'34'             BAD STATUS ON SELCH?                isr2
          BZ    ISRDSARM             IF NO PROBLEM, GO ON                isr2
          LHI   E7,X'8400'           UNRECOVERABLE ERROR                 isr2
          STH   E7,DCB.STAT(E5)                                          isr2
```

```
ISRDSARM EQU    *                                                       isr2
         LA     E3,III              RESET THE INTERRUPT VECTOR          isr2
         STH    E3,ISPTAB(E2,E2)    * SO THAT THERE WILL BE NO STRAY    isr2
         LHI    E3,Y'7FFF'          THIS WILL RESET TIMEOUT VALUE       isr2
         LH     E7,DCB.TOUT(E5)     DID WE TIME OUT?                    isr2
         STH     E3,DCB.TOUT(E5)    CLEAR TIMEOUT COUNTER               isr2
         BZ     ISRRTN              YES, DON'T SCHEDULE TERM            isr2
         L      E7,DCB.LEAF(E5)                                         isr2
         ATL    E7,SQ               PUT LEAF ADDRESS ON SYSTEM QUEUE    isr2
         LPSWR EO                   AND EXIT ISR                        isr2
ISRRTN   EQU    *                                                       timeout
         LHI    E3,X'8298'          SET TIMEOUT ERROR CODE              timeout
         STH    E3,DCB.STAT(E5)     *                                   timeout
         LPSWR EO                                                       timeout
*
* EVENT SERVICE ROUTINE.                                                esr
*                                                                       esr
         IMPUR                                                          esr
TERMSDIO EQU    *                                                       esr
         LH     U9,DCB.STAT(UD)     DID WE GET AN ERROR?                esr
         BNZ    XFERERR             IF SO, DO SPECIAL STUFF.            esr
         LH     U9,DCB.SDN(UD)      GET SELCH ADDRESS                   esr
         OC     U9,STOPSEL          STOP SELCH, EXTENDED ADDRESS READ   esr
         RDR     U9,UA              READ SELCH FINAL ADDRESS            esr
         RHR     U9,UB              ( ALL 3 BYTES )                     esr
         SLL     UA,16              MAKE A 24 BIT WORD OUT OF THIS      esr
         OR      UA,UB              GOT FINAL SELCH ADDRESS NOW.        esr
         C       UA,DCB.EADR(UD)    DID IT EXCEED SPECIFIED FINAL?      esr
         BP      SLCHFAIL           IF SO, EXIT.                        esr
         S       UA,DCB.SADR(UD)    CALCULATE LENGTH OF TRANSFER        esr
         BM      SLCHFAIL           IF WE WENT BACKWARDS - FAILED.      esr
         AIS     UA,1               ARITHMETIC ADJUSTMENT               esr
         ST      UA,DCB.LLXF(UD)    GIVE USER LENGTH OF TRANSFER.       esr
         LH     U8,DCB.TOUT(UD)     TIMED OUT?                          esr
         BNZ    TDIOCHOF            NO, BRANCH                          esr
         LHI    U6,X'8298'          UNRECOVERABLE ERROR                 esr
         STH    U6,DCB.STAT(UD)                                         esr
TDIOCHOF EQU    *                                                       esr
         BAL    U8,TOCHOFF          REMOVE FROM TIMER CHAIN             esr
         B      IODONE              WE'RE THROUGH                       esr
*
XFERERR  EQU    *                                                       err
         LIS    UO,O                CLEAR LENGTH OF TRANSFER            err
```

```
          ST     U0,DCB.LLXF(UD)      *                                   err
          LHI    U0,X'7FFF'           RESET TIME-OUT VALUE TO DONT CARE.  err
          STH    U0,DCB.TOUT(UD)      *                                   err
          BAL    U8,TOCHOFF           GET OFF TIMER CHAIN                 err
          B      IODONE                                                   err
SLCHFAIL EQU    *                                                         err
          LHI    U0,X'9000'           INDICATE EOM ON SELCH               err
          STH    U0,DCB.STAT(UD)                                          err
          B      XFERERR              *                                   err
          END
```

The following device control block (DCB) shows the code required to write a DCB when using CUPMT to generate the system. The DCB and channel control block (CCB) macros may be found in SYSSTRUC.MLB. These macros are also listed in Appendix C of Part II of this manual.

```
**DCB143
          MLIBS 8,9,10
          NLIST
          DPROG DCOD=143
          LIST
      EXTRN  Z(CDN1)
*
*   BUILD DCB FOR DIGITAL I/O DRIVER
*
          DCB    DCOD=143,INIT=INITSDIO,TERM=TERMSDIO,                 1
                 ATRB=7F00,RECL=0,SIZE=DCB.DVDP+8,                     1
                 FLGS=DFLG.LNM
DCB.IVAL EQU    DCB.RTRY
          ORG    DCB143+DCB.IVAL
          DC     H'0'                    INITIAL VALUE
*  DEVICE DEPENDENT PART
          ORG    DCB143+DCB.CCB
          DC     Z(CCB143)               CCB
          DC     X'0'
      ORG    DCB143+DCB.SDN
      DC     Z(CDN1)
*
*   BUILD CCB FOR DIGITAL I/O DRIVER
*
          CCB    DCOD=143
          NLIST
          END
          BEND
```

The following macro shows the DCB specification for the DIO driver when using Sysgen/32 for sysgen. Note that the DCB specification is a separate macro file and is NOT a part of the driver (as is the case in a CUPMT system). The DCBI and CCBI macros can be found in SYSGEN32.MLB. These macros are also listed in Appendix C of Part II of this manual.

```
          MACRO
          DCB242 %DCOD=,%DN=,%CLAS=,%ILVL=,%NAME=,%SHCCB=,              1
                   %SLCH=,%CNTR=
          GBLB  %DCB$,%PDCB,%DDCB,%EVN,%CCB,%DFLG,%SDCB
          GBLB  %IDCB,%ODCB,%S125DCB,%ICCB,%BDCB
          GBLB  %ADCB,%TCB,%IOB,%IOB$,%CRTDCB,%LPDCB
          GBLB  %MMDDX,%DDEX,%VFDCB,%MTP,%CRPDCB,%MGDCBX,%HFWDST
          GBLB  %PSDCBX,%CRDP,%AOBDCB,%BIOCDCB,%LPTDCB
          GBLB  %CORD242
          GBLC  %IDVAL
          BGBLA %ID242
          LCLA  %CCBFL
          LCLA  %CLASN
          LCLC  %RXLT,%RQU
          LCLC  %CORDNM,%PTRPAS
          LCLC  %OFFS
          LCLA  %RDN
          LCLC  %MDN,%MCNT,%MSLCH
          LCLA  %TRCNT,%UPTR
          LCLB  %FOUND,%DA
          BGBLA %FIRST
%RQU      SETC  'COMQ'                 DEFAULT DEVICE QHANDLER
%MDN      SETC  '%DN'                  DEVICE ADDRESS
%CCBFL    SETA  0
%CORD242  SETB  0
          AIF   (T'%CLAS EQ 'U')&CLSNTD
%CLASN    SETA  %CLAS*12               IOCLASS*12
&CLSNTD   ANOP
          CONVNUM VAL=%ID242           CONVERT CURRENT ID TO HEX.
          USERINIT
          $DCB$
          DCBI  DCOD=242,SIZE=DCB.DVDP+8,INIT=INITSDIO,                 1
                TERM=TERMSDIO,FLGS=DFLG.LNM,                            2
                ID=%IDVAL,ATRB=7F00
          CCBI  DCOD=242,ID=%IDVAL,SUBA=III
CCB%NAME  EQU   CCB%DCOD%IDVAL
%ID242    SETA  %ID242+1
```

```
&DCBOPT  ANOP
DCB%DCOD%IDVAL  PROG   USER DCB
%OFFS    SETC  '%DCOD':'%IDVAL'      ESTABLISH PROPER OFFSET
DCB.%NAME EQU  DCB%OFFS
         ENTRY DCB.%NAME
         ORG   DCB%OFFS+DCB.DMT
         DC    DMT.%NAME
         EXTRN DMT.%NAME
         ORG   DCB%OFFS+DCB.DN       DEVICE ADDRESS
         DC    H'%DN'
DCB.IVAL EQU  DCB.RTRY
         ORG   DCB%OFFS+DCB.IVAL
         DC    H'0'
         ORG   DCB%OFFS+DCB.LEAF     LEAF POINTER
         AIF   (T'%SHCCB' EQ 'U')&NSLEAF   B IF NOT SHARED
         DAC   LF%SHCCB              USE SHARED DEVICE LEAF
         EXTRN LF%SHCCB
         AGO   &NRMLFX
&NSLEAF  ANOP
         DAC   LF%OFFS               GENERATE STANDARD LEAF NAME
         EXTRN LF%OFFS
&NRMLFX  ANOP
&NOLEAF  ANOP
         AIF   (T'%CLAS EQ 'U')&NOCLAS
         ORG   DCB%OFFS+DCB.CLAS     IO CLASS
         DC    H'%CLASN'             IOCLASS*12
&NOCLAS  ANOP
         AIF   (T'%ILVL EQ 'U')&NOILVL
         ORG   DCB%OFFS+DCB.ILVL     ILEVEL
         DC    H'%ILVL'
&NOILVL  ANOP
         AIF   ('%RQU' EQ '')&NOQU
         ORG   DCB%OFFS+DCB.Q
         DAC   %RQU
         EXTRN %RQU
&NOQU    ANOP
         AIF   ('%SLCH' EQ 'U')&NSLCH
         ORG   DCB%OFFS+DCB.SDN
         DC    X'%SLCH'
&NSLCH   ANOP
         AIF   ('%CNTR' EQ 'U')&NCNTR
         ORG   DCB%OFFS+DCB.CDN
         DC    X'%CNTR'
```

```
&NCNTR   ANOP
         ORG   $ST%OFFS            ORG TO END OF DCB
         ASIS
         END
%RDN     SETA  %DN+1
         MEND
```

4.3 TELEX TRIDENSITY MAGNETIC TAPE DRIVER

This sample driver, a TELEX magnetic tape driver, was written to accommodate a modified Perkin-Elmer 35-820 tape interface. This includes special device-dependent functions, such as erase gap, rewind and unload, and read status. (Tridensity refers to the fact that it operates on three different recording densities.) This driver, as opposed to the previous one, is a very complex driver and was chosen as an example because it contains so many of the features that a driver is capable of having. For example, a TELEX driver contains retry logic, multiple ESRs, and supervisor call 1 (SVC1) command function processing. It also contains resource releases, a mechanism by which you can release a particular resource (e.g., SELCH or controller) into the system.

Note that in this type of driver, the ISR (PURE) code is put immediately after the DIR or ESR that enters the ISR via a SINT. Similarly, the ESR code immediately follows the ISR that schedules it. This is done to allow the reader to more easily follow the logic flow through the driver. Also note that generally, in the case of conditional branches, the normal logic path is to NOT take the branch. This is important because the driver is then easier to read and the execution speed on the Model 3240/3250 processors is improved.

Of further interest is the fact that this driver happens to contain some sense status loops. This is not normal practice in writing drivers and should be avoided whenever possible. The loops in this driver were included after experimentally determining that the no-motion status bit was set after a minimal amount of time, i.e., after a minimal number of times through the loop. Otherwise, if the no-motion status bit is not set right away, the status loop is monopolizing too much of the processor's time, and it is preferable to set up an interrupt, which is the normal programming practice in writing drivers. In this particular case, the status loop was necessary.

```
         BATCH
         LCNT  50
         MLIBS 8,9,10
SGN.EOV  EQU   1         USED BY $MTP MACRO
INITTELX PROG      TELEX MAG TAPE DRIVER
         NLIST
         $UREGS                                                    strucs
         $EREGS                                                    strucs
         $PSW                                                      strucs
         $SVC1                                                     strucs
         $SVC7                                                     strucs
         $EVN                                                      strucs
         $IOH                                                      strucs
         $DCB$                                                     strucs
         $CCB                                                      strucs
         $TCB                                                      strucs
         $TOPT                                                     strucs
         $MTP                                                      strucs
         LIST
         TITLE ENTRY'S, EXTRN'S & EQUATES
         ENTRY INITTELX,CMDTELX,TERMTELX
         EXTRN ADCHKNS,DMT,EVREL,EVRTE,III,IODONE,ISPTAB
         EXTRN MEMFAULT,RELIOB,SQ,SV9.ATQ1
         EXTRN SVC1BFM,SVC1BSR,SVC1DDF,SVC1FFM,SVC1FSR,SVC1HALT
         EXTRN SVC1NOOP,SVC1READ,SVC1REW,SVC1TEST,SVC1WAIT
         EXTRN SVC1WFM,SVC1WRIT,TOCHOFF,TOCHON,UBOT
         SPACE
* STATUS BYTE DEFINITION.
*
* THE FOLLOWING STATUS INFORMATION MAY ALSO BE FOUND IN
* THE PERKIN-ELMER HIGH PERFORMANCE TAPE DRIVE (HPTD)
* CONTROLLER INSTALLATION AND MAINTENANCE MANUAL.
* PUBLICATION NUMBER (47-028R00).
*
* THE CONTROLLER STATUS BYTE IS OBTAINED BY USE OF THE
* SENSE STATUS OR SENSE STATUS REGISTER INSTRUCTIONS.
*
ERR      EQU   X'80'               ERROR STATUS
TERR     EQU   X'40'               TRANSFER ERROR STATUS
EOM      EQU   X'20'               END OF TAPE STATUS
NMTN     EQU   X'10'               NO MOTION STATUS
BSY      EQU   X'08'               BUSY STATUS
EX       EQU   X'04'               EXAMINE STATUS
```

```
EOF       EQU   X'02'                 TAPE MARK STATUS (END-OF-FILE)
DU        EQU   X'01'                 DEVICE UNAVAILABLE STATUS
TAPELTH   EQU   28800                 MAXIMUM TAPE LENGTH
*
* THERE ARE TWO METHODS OTHER THAN A SENSE STATUS ON THE
* CONTROLLER, TO OBTAIN REQUIRED INFORMATION ABOUT A TELEX
* TAPE UNIT.
*
* NO-OP COMMAND.
*
* THE NO-OP COMMAND RETURNS DEVICE STATUS HALFWORD INFORMATION.
* THERE ARE FOUR POSSIBLE HALFWORDS THAT MAY BE READ.  ONLY
* THE READ STATUS COMMAND RETURNS ALL FOUR TO THE USER.  THE
* DRIVER USES THE NO-OP COMMAND TO DETERMINE THE DENSITY OF
* THE DRIVES (REQUIRED FOR APPROPRIATE TIMEOUT CALCULATIONS).
*
* UPPER HALFWORD IS THE SAME FOR ALL 4 DEVICE STATUS HALFWORDS.
*
* BIT 8 - ZERO.
* BITS 9 & 10 - DENSITY
*          BIT 9     BIT 10
*            1         0            NRZI
*            0         0            PE
*            0         1            GCR
* BIT 11 - BLOCK (SET WHEN BLOCK STATUS DETECTED FROM FCU)
* BIT 12 - ODDBYTE (SET WHEN TRANSFER ENDS ON ODD-BYTE BOUNDARY)
* BIT 13 - WRITE UNDERFLOW
* BIT 14 - BUS PARITY
* BIT 15 - READ OVERRUN
*
*
* SENSE COMMAND (X'30')
*
* SENSE BYTE 0
*  BIT 0 - LOAD POINT...SET WHEN THE TAPE UNIT IS AT LOAD POINT.
*  BIT 1 - FILE PROTECT.SET WHEN TAPE UNIT IS WRITE PROTECTED.
*  BIT 2 - BACKWARD STATUS.SET WHEN TAPE UNIT IS PERFORMING OR
*          HAS PERFORMED A BACKWARD OPERATION.
*  BIT 3 - WRITE STATUS...TAPE UNIT IS NOT IN READ MODE.
*  BIT 4 - EOT..SET WHEN LEADING EDGE OF EOT MARKER IS SENSED
*          DURING A FORWARD OPERATION AND RESET WHEN THE
*          TRAILING EDGE OF EOT MARKER IS SENSED DURING
*          A BACKWARD OPERATION.
*  BIT 5 - LO DENSITY...SET WHEN A DUAL DENSITY TAPE UNIT IS
*          OPERATING IN THE LOWER DENSITY MODE.
*  BIT 6 - READY...SET WHEN A TAPE IS LOADED AND THE TAPE UNIT
*          IS ON-LINE.
*  BIT 7 - COMMAND REJECT..SET IS THE COMMAND BYTE HAS EVEN
*          PARITY OF IF A WRITE COMMAND IS RECEIVED IN
*          CONJUCTION WITH A BACKWARD MOTION COMMAND.
*
*
        TITLE SPECIAL STATUS DEFINITIONS
*
* FOR EASE OF DEBUGGING BOTH SOFTWARE AND HARDWARE RELATED
* PROBLEMS, THIS DRIVER REPORTS AS MUCH UNIQUE STATUS INFORMATION
* AS IS POSSIBLE WITHOUT CONFLICTING WITH THE NORMAL (EXPECTED)
* STATUS SUCH AS EOF, AND EOT.
*
*
* X'8241' - TIME-OUT ON READ DEVICE HALFWORD ISSUED AT THE
*           START OF THE DRIVER TO OBTAIN DENSITY.  NO-MOTION
*           DID NOT SET AFTER A LOOP COUNT OF 63.
* X'8282' - TIME-OUT ON A WRITE OPERATION.
* X'8263' - TIME-OUT ON A READ OPERATION.
* X'8484' - SELCH FINAL ADDRESS WAS LESS THAN START ADDR(READ/WRITE)
* X'8271' - TIME-OUT ON FORWARD/BACKSPACE RECORD.
* X'8275' - TIME-OUT ON WRITE FILEMARK.
* X'8272' - TIME-OUT ON FORWARD/BACKFILE OPERATIONS.
* X'8273' - TIME-OUT ON ERASE GAP FUNCTION.
* X'82DD' - TIME-OUT ON READ STATUS FUNCTION.
* X'82DE' - TIME-OUT ON READ DEVICE HALFWORD ISSUED TO
*           CHECK FOR WRITE PROTECT CONDITION.
* X'82DF' - TIME-OUT ON READ DEVICE HALFWORD ISSUED TO
*           CHECK FOR EOT/BOT CONDITIONS.
* X'8283' - WRITE PROTECT INDICATOR.
* X'8277' - TIME-OUT ON REWIND FUNCTION.
* X'A000' - DEVICE UNAVAILABLE.
* X'9000' - END OF TAPE.
* X'8400' - UNRECOVERABLE ERROR (ERR OR TERR BUT NOT NO-MOTION).
* X'82FA' - PARITY ERROR (TRUE PARITY) OR ATTEMPT TO
*           READ LESS THAN A FULL RECORD ON NON-EXTENDED
*           OPTION I/O.
* X'82AA' - FILE MARK ERROR - OCCURS WHEN READING LARGE
*           AREAS OF ERASED TAPE.  AS LONG AS THIS STATUS
*           OCCURS THE USER SHOULD FORWARD SPACE RECORD TO
*           GET BY THE ERASE AREA.  (THIS STATUS OCCURS
*           AFTER 10 FEET OF ERASED TAPE, WHICH IS AS MUCH
*           AS THE FORMATTER WILL MOVE THE TAPE WITHOUT
*           EXPECTING A FILE MARK IN THE ERASED AREA).
*
* CRASH CODES GENERATED WITHIN THIS MODULE.
* CRASH 502 - OCCURS WHEN THE SELCH FINAL ADDRESS IS LESS
*             THAN THE SELCH START ADDRESS AND THE FINAL
*             ADDRESS IS LESS THAN UBOT (THIS MEANS O.S
*             CODE WAS OVERWRITTEN).
*
*
*
* REGISTER CONVENTIONS
*
* (REGISTER SET 5 )
* U0 - SCRATCH
* U1 - SCRATCH
* U2 - SCRATCH
* U3 - SCRATCH
* U4 - SCRATCH
* U5 - SCRATCH
* U6 - DEVICE ADDRESS
* U7 - SCRATCH
* U8 - SCRATCH
* U9 - SCRATCH
* UA - SCRATCH
* UB - SCRATCH
* UC - CCB ADDRESS
* UD - DCB ADDRESS
* UE - SCRATCH
* UF - LEAF ADDRESS BEFORE CALL TO EVREL
         TITLE DRIVER INITIALIZATION
INITTELX EQU    *                                               dir
CMDTELX  EQU    *                                               dir
TLX0010  EQU    *                  ALSO EQUAL TO INITMAG        dir
         LHL    UC,DCB.CCB(UD)     SET UP CCB POINTER           dir
         LHL    U6,DCB.DN(UD)      GET DEVICE ADDRESS           dir
         LH     U7,DCB.SDN(UD)     GET SELCH ADDRESS            dir
         OC     U7,CLEARSEL        CLEAR SELCH                  dir
         LHL    U8,DCB.FLG1(UD)    GET THE FLAGS                dir
         SSR    U6,U7              GET THE DRIVE STATUS         dir
         BTC    5,TLX2250          IF EX OR DU, DO EXTRA CHECKS dir
```

```
TLX0015  EQU   *                   RETURN HERE FROM MORE CHECKING          dir
         THI   U7,ERR!TERR!NMTN    IF NONE OF THESE ARE SET,               dir
         BZ    TLX2260             WE ASSUME IT IS REWINDING:BRANCH        dir
         THI   U7,NMTN             IF ERR OR TERR AND NOT NMTN,            dir
         BZ    TLX2350             EXIT WITH X'8400' STATUS.               dir
TLX0018  EQU   *                   START HERE AFTER REWIND FINISHES        dir
         OC    U6,DCB.CCLC(UD)     CLEAR CONTROLLER                        dir
         LA    U9,TLX0020          SET EOT/BOT CHECK RETURN                dir
         THI   U8,X'0800'          WAS EOT PREVIOUSLY ENCOUNTERED          dir
         BNZ   TLX2050             IF SO, MUST GO DO FULL CHECK            dir
         THI   U7,EOM              ARE WE NOW AT END OF TAPE?              dir
         BNZ   TLX2050             YES, GO DO FULL CHECK                   dir
         NHI   U8,X'33FF'          RESET BITS 0,1,4,5                      dir
         STH   U8,DCB.FLG1(UD)     SAVE THE FLAGS                          dir
TLX0020  LIS   U0,0                                                        dir
         STH   U0,DCB.STAT(UD)     CLEAR STATUS BEFORE WE START            dir
*                                                                          dir
* GET THE DENSITY SELECTION OF THE TRANSPORT UNIT.                         dir
* FROM THIS INFORMATION, COMPUTE THE DATA TRANSFER RATE,                   dir
* ASSUMING THAT THE DRIVE OPERATES AT 125 IPS.                             dir
*                                                                          dir
         OC    U6,DCB.DSB0(UD)     GIVE FORMATTER A NO-OP CMD              dir
         LHI   U1,63               SET LOOP COUNT - PREVENT HANGS!         dir
TLX0030  SIS   U1,1                DECR THE LOOP COUNT                     dir
         BNP   TLX0055             COUNTED OUT - TIME OUT THIS OP.         dir
         SSR   U6,U2               GET STATUS                              dir
         THI   U2,X'10'            DID NO-MOTION SET?                      dir
         BZ    TLX0030             IF NOT, LOOP UNTIL IT IS SET            dir
TLX0040  RHR   U6,U2               GET THE DEVICE STATUS H/W               dir
         NHI   U2,X'60'            MASK DENSITY BITS                       dir
         SRLS  U2,3                MAKE FULLWORD INDEX                     dir
         L     U1,DCB.XRT(UD,U2)   GET ACTUAL TRANSFER RATE                dir
         ST    U1,DCB.RATE(UD)     AND SAVE FOR LATER USE                  dir
* NOTE - U2 MUST BE PRESERVED IF THE REQUEST IS AN                         dir
* 'ERASE BUFFER' DEVICE-DEPENDENT COMMAND FUNCTION.                        dir
*                                                                          dir
*  COMPUTE THE REQUESTED LENGTH OF TRANSFER.                               dir
         L    U9,DCB.EADR(UD)      GET END ADDRESS                         dir
         S    U9,DCB.SADR(UD)      LESS THE START ADDRESS                  dir
         AIS  U9,1                 ADJUST FOR INCLUSIVE ADDRS              dir
         ST   U9,CCB.XLT(UC)       SAVE FOR LATER REFERENCE                dir
*                                                                          dir
* U9 WILL BE USED A LITTLE LATER - DO NOT DESTROY!                         dir
*                                                                dir
*                                                                dir
* IF THE SVC 1 FUNCTION CODE INDICATES A COMMAND FUNCTION,       dir
* BRANCH TO DETERMINE WHAT FUNCTION TO PERFORM.                  dir
*                                                                dir
         LB     U3,DCB.FC(UD)       GET SVC 1 CODE               dir
         THI    U3,X'80'            CHECK FOR COMMAND BIT ON     dir
         BP     TLX1280                                          dir
*                                                                dir
* IF THE TASK IS NOT LINKED WITH EXTENDED SVC1 OPTION, FORCE     dir
* THE EXTENDED SVC 1 FUNCTION FIELD TO ZERO FOR LATER USE.       dir
* THEN DETERMINE WHICH ROUTINE TO GO TO, BASED ON THE            dir
* EXTENDED FUNCTION CODE AND BITS 1 AND 2 IN THE STANDARD        dir
* SVC 1 FUNCTION FIELD.                                          dir
*                                                                dir
         LI     U0,TOPT.X1B         FURTHER CHECK FOR TASK OPTION    dir
         L      U1,DCB.TCB(UD)      GET TCB ADDRESS              dir
         TBT    U0,TCB.OPT(U1)      GET TASK OPTION FIELD        dir
         BZ     TLX0070             NOT LINKED WITH EXTENDED OPTRWNS dir
         L      U1,DCB.SV1X(UD)     FETCH USER EXTEND OPTION     dir
         NI     U1,X'1F'            EXTRACT THE EXTEND CODE      dir
         SLLS   U1,2                                             dir
         L      U1,TLX0050(U1)      INDEX INTO TABLE FOR EXEC ROUTINE dir
         BR     U1                                               dir
*                                                                dir
         SPACE 1                                                 dir
         ALIGN 4                                                 dir
TLX0050  EQU    *                                                dir
         DAC    A(TLX0080)        0                              dir
         DAC    A(TLX0080)        1                              dir
         DAC    A(TLX0060)        2                              dir
         DAC    A(TLX0060)        3                              dir
         DAC    A(TLX0080)        4                              dir
         DAC    A(TLX0080)        5                              dir
         DAC    A(TLX0080)        6                              dir
         DAC    A(TLX0080)        7                              dir
         DAC    A(TLX0080)        8                              dir
         DAC    A(TLX0080)        9                              dir
         DO     22                                               dir
         DAC    A(TLX2360)                                       dir
         SPACE 1                                                 dir
*                                                                dir
* COME HERE ONLY ON TIME-OUT OF READ DEVICE STATUS HALFWORD      dir
*                                                                      dir
TLX0055  EQU   *                                                       dir
         LHI   U8,X'8241'          SET A TIME-OUT STATUS               dir
         STH   U8,DCB.STAT(UD)     STORE STATUS                        dir
         B     TLX2150             CHECK FOR REWIND IN PROGRESS & EXIT. dir
TLX0060  EQU   *                                                       dir
         BAL   UB,TLX1940          IF FILE PROTECT NO RETURN           dir
         B     TLX0920                                                 dir
TLX0070  EQU   *                                                       dir
         LIS   U1,0                RESET EXTENDED OPTIONS FIELD        dir
         ST    U1,DCB.SV1X(UD)     *                                   dir
TLX0080  EQU   *                                                       dir
*                                                                      dir
*   U3 = SVC 1 FUNCTION CODE FIELD.                                    dir
*                                                                      dir
*     IF THE REQUESTED LENGTH IS LESS THAN 4, EXIT                     dir
*                                                                      dir
         CHI  U9,4                 TOO SHORT (LESS THAN 4 BYTES)       dir
         BL   TLX2360              IF SO, EXIT WITH ILLEGAL FNCTN      dir
*                                                                      dir
* CALCULATE THE APPROPRIATE TIME-OUT VALUE, BASED ON                   dir
* THE DENSITY SETTING READ FROM THE FORMATTER.                         dir
*                                                                      dir
         LIS   U1,1                SET UP TO CALCULATE TIME-OUT        dir
         L     U0,CCB.XLT(UC)      GET COMPUTED XFER LENGTH            dir
TLX0085  AIS   U1,1                ADD 1 SECOND FOR EACH RATE SIZE     dir
         S     U0,DCB.RATE(UD)     COMPARE SIZE TO RATE                dir
         BP    TLX0085             LOOP UNTIL SIZE FIELD IS COUNTED    dir
         STH   U1,CCB.LBO(UC)      SAVE TIME-OUT TEMPORARILY           dir
*                                                                      dir
         THI   U3,SV1.READ         READ?                               dir
         BNZ   TLX0320             YES, GO READ ONE RECORD             dir
         THI   U3,SV1.WRIT         WRITE?                              dir
         BZ    TLX2360             NO,ILLEGAL FUNCTION                 dir
         BAL   UB,TLX1940          IF FILE PROTECT NO RETURN           dir
         TITLE WRITE OPERATIONS                                        dir
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **        dir
*              WRITE ONE RECORD                                        dir
*                                                                      dir
* SET UP RETRY COUNTERS                                                dir
*                                                                      dir
         LIS   U0,0                INITIALIZE RETRY                    dir
         STH   U0,DCB.RTRY(UD)     COUNTER                             dir
         LH    U1,DCB.WRY1(UD)     GET DEFAULT WRITE RETRY COUNT        dir
         L     U0,DCB.SV1X(UD)     GET EXTENDED FUNCTION CODE           dir
         BZ    TLX0090             IF EXT. FNCTN = 0, USE DEFAULT       dir
         LB    U1,DCB.SV1X(UD)     IF NOT, USE USER'S COUNT             dir
TLX0090  STH   U1,DCB.RMAX(UD)     SET RETRY COUNT.                     dir
         L     U2,CCB.XLT(UC)      GET REQUESTED LENGTH OF XFER         dir
         THI   U2,1                ODD BYTE XFER ?                      dir
         BZ    TLX0120             NO, BRANCH                           dir
         LI    U1,MAGHLFB          YES, SET FLAG FOR ODD BYTE XFER      dir
         SBT   U1,DCB.FLG1(UD)                                          dir
* SET UP FOR ISR AND ESR SCHEDULING AND EXECUTION                       dir
*                                                                       dir
TLX0120  LA    U1,TLX0130          GET INTERRUPT SERVICE ADDRESS        dir
         STH   U1,CCB.SUBA(UC)     SETUP ISR POINTER                    dir
         LA    U1,TLX0240          GET EVENT SERVICE ADDRESS            dir
         ST    U1,DCB.ESR(UD)      SET ESR ADDRESS FOR SQS              dir
         LH    U1,DCB.SDN(UD)      GET SELCH ADDRESS                    dir
         LA    U0,1(UC)            MAKE ODD CCB ADDRESS                 dir
         STH   U0,ISPTAB(U1,U1)    SET SELCH ISPT ENTRY                 dir
         STH   U0,ISPTAB(U6,U6)    SET CONTROLLER ISPT ENTRY            dir
         BAL   U8,TOCHON           PUT ON TIMEOUT CHAIN                 dir
         LH    U8,DCB.ILVL(UD)     FETCH INTERRUPT LEVEL                dir
         SINT  U8,0(U6)            ENTER INTO THE ISR                   dir
         B     EVRTE               EXIT ESR STATE                       dir
         SPACE
         PURE                                                           isr1
TLX0130  EQU   *                                                        isr1
         L     E5,CCB.DCB(E4)      GET DCB ADDRESS                      isr1
         LH    E7,CCB.LBO(E4)      GET CALCULATED TIME-OUT VALUE        isr1
         STH   E7,DCB.TOUT(E5)     STORE IT                             isr1
         LH    E3,DCB.SDN(E5)      GET SELCH ADDRESS                    isr1
         OC    E3,CLEARSEL         CLEAR SELCH                          isr1
         WD    E3,DCB.SADR+1(E5)   WRITE START ADDRESS TO SELCH         isr1
         WH    E3,DCB.SADR+2(E5)   *                                    isr1
         WD    E3,DCB.EADR+1(E5)   WRITE END ADDRESS TO SELCH           isr1
         WH    E3,DCB.EADR+2(E5)   *                                    isr1
         EPSR  E6,E6               GET CURRENT PSW                      isr1
         NHI   E6,X'20F0'          TURN OFF ALL INTERRUPTS              isr1
         EPSR  E7,E6               (GO NONINTERRUPTABLE)                isr1
         LI    E6,MAGHLFB          CHECK THE ODD BYTE XFER FLAG         isr1
         RBT   E6,DCB.FLG1(E5)     RESET THE FLAG                       isr1
         BZ    TLX0140             0 , SKIP                             isr1
         OC    E2,ODDWRITE         GIVE CONTROLLER ODDBYTE CMD.         isr1
```

```
TLX0140  EQU    *                                          isr1
         OC     E2,DCB.CENB(E5)    ENABLE INTERRUPTS       isr1
         OC     E2,DCB.CWRT(E5)    START THE DRIVE         isr1
         OC     E3,GOWRITE         START THE SELCH         isr1
         SPACE                                             isr1
LX0150   EQU    *                                          isr1
         LA     E7,TLX0190         SET NEXT ISR ADDRESS    isr1
         STH    E7,CCB.SUBA(E4)    *                       isr1
         LPSWR  E0                 EXIT INTERRUPT SERVICE  isr1
GOWRITE  DC     X'1400'            SELCH WRITE COMMAND
CLEARSEL DC     X'4800'            CLEAR SELCH COMMAND
ODDWRITE DC     X'4B00'            ODD BYTE WRITE COMMAND
         SPACE
TLX0190  EQU    *                                          isr2
         L      E5,CCB.DCB(E4)     GET DCB ADDRESS         isr2
         LH     E6,DCB.TOUT(E5)    GET TIME-OUT CONSTANT   isr2
         BZ     TLX0195            IF TIMED-OUT, EXIT NOW  isr2
         CLH    E2,DCB.SDN(E5)     FROM THE SELCH?         isr2
         BNE    TLX0150            NO, WAIT SOME MORE      isr2
         LA     E7,III             RESET SELCH ISPT NOW    isr2
         STH    E7,ISPTAB(E2,E2)   *                       isr2
         LH     E2,DCB.DN(E5)      FETCH DEVICE ADDRESS    isr2
         SSR    E2,E3              SENSE DEVICE STATUS     isr2
         THI    E3,DU!NMTN         NO MOTION OR DU??       isr2
         BNZ    TLX0230            YES, BRANCH             isr2
         LA     E7,TLX0210         SET NEXT ISR ADDRESS    isr2
         STH    E7,CCB.SUBA(E4)    *                       isr2
TLX0195  EQU    *                                          isr2
         LPSWR  E0                 EXIT INTERRUPT SERVICE  isr2
         SPACE
TLX0210  EQU    *                                          isr3
         L      E5,CCB.DCB(E4)     GET DCB ADDRESS         isr3
         LH     E6,DCB.TOUT(E5)    GET TIME-OUT VALUE      isr3
         BZ     TLX0220            IF TIMED OUT, EXIT NOW  isr3
         CLH    E2,DCB.DN(E5)      FROM DEVICE?            isr3
         BNE    TLX0220            NO, WAIT SOME MORE      isr3
         THI    E3,DU!NMTN         DU OR NO MOTION?        isr3
         BNZ    TLX0230            YES, GO PROCESS         isr3
TLX0220  EQU    *                                          isr3
         LPSWR  E0                 WAIT FOR IT TO COME AGAIN isr3
         SPACE                                             isr3
TLX0230  EQU    *                                          isr3
         STB    E3,DCB.DDPS(E5)    SAVE DRIVE STATUS       isr3
         OC    E2,DCB.CDAR(E5)    DISARM INTERRUPTS                      isr3
         LH    E6,DCB.TOUT(E5)    GET TIME-OUT COUNTER                   isr3
         BTFS  2,2                RETURN NOW IF TIMED-OUT                isr3
         LPSWR E0                 DON'T RE-ADD TO S.Q.                   isr3
         LCS   E6,1               OTHERWISE SET MINUS TO                 isr3
         STH   E6,DCB.TOUT(E5)    SHOW WE HAVE BEEN HERE.                isr3
         L     E6,DCB.LEAF(E5)    GET LEAF ADDRESS                       isr3
         ATL   E6,SQ              SCHEDULE DRIVER TERMINATION            isr3
         LPSWR E0                 EXIT AND WAIT FOR TERM.                isr3
         SPACE
         IMPUR
TLX0240  EQU   *                                                         esr1
         LHL   UC,DCB.CCB(UD)     A(CCB)                                 esr1
         LHL   U6,DCB.DN(UD)      DEVICE NUMBER                          esr1
         LH    U7,DCB.STAT(UD)    GET DEVICE DEPENDENT STATUS            esr1
         THI   U7,DU              DID IT GO UNAVAILABLE                  esr1
         BP    TLX2340            ERROR EXIT, IF SO.                     esr1
         LH    U2,DCB.SDN(UD)     GET THE SELCH DEVICE ADDRESS           esr1
         OC    U2,CLEARSEL        STOP THE SELCH                         esr1
         RDR   U2,U0              AND OBTAIN THE SELCH FINAL             esr1
         EXHR  U0,U0              * ADDRESS...PLACE IT IN                esr1
         RHR   U2,U1              THE APPROPRIATE POSITION.              esr1
         OR    U0,U1              *                                      esr1
         C     U0,DCB.SADR(UD)    IF THE FINAL ADDRESS LESS THAN         esr1
         BM    TLX0610            THE START ADDR-SELCH ERROR.            esr1
         C     U0,DCB.EADR(UD)    IS THE FINAL ADDRESS EQUAL             esr1
         BNE   TLX0265            TO REQUESTED END--CHECK FURTHER.       esr1
         L     U0,CCB.XLT(UC)     GET THE LENGTH OF TRANSFER             esr1
         ST    U0,DCB.LLXF(UD)    AND SAVE IT IN THE DCB.                esr1
         BAL   U8,TOCHOFF         REMOVE FROM TIMEOUT CHAIN              esr1
         LH    U7,DCB.TOUT(UD)    CHECK FOR TIME OUT                     esr1
         BZ    TLX0310            IF SO, GET OUT                         esr1
         LB    U3,DCB.DDPS(UD)    GET STATUS IN ISR                      esr1
         THI   U3,ERR!TERR        TAPERR ?                               esr1
         BNZ   TLX0270            YES                                    esr1
TLX0250  LIS   U0,0               CLEAR STATUS                           esr1
         THI   U3,X'20'           EOT/BOT SET?                           esr1
         BZ    TLX0260            NO - GO ON                             esr1
         OHI   U0,X'9000'         YES - SET USUAL STATUS                 esr1
         LI    U1,MAGWAT2B        AND SET FLAG FIELD                     esr1
         SBT   U1,DCB.FLG1(UD)    *                                      esr1
TLX0260  EQU   *                                                         esr1
         STH   U0,DCB.STAT(UD)    STORE STATUS                           esr1
```

```
        B     TLX215 0            GO CHECK REWIND CONDITIONS            esr1
*                                                                       esr1
* CODE BELOW HERE IS EXECUTED ONLY ON ERROR CONDITIONS                  esr1
*                                                                       esr1
TLX0265 EQU   *                   CHECK FOR EXTENDED FUNCTION 4         esr1
        S     U0,DCB.SADR(UD)     COMPUTE ACTUAL LENGTH OF TRANSFER     esr1
        ST    U0,DCB.LLXF(UD)     AND SAVE IT IN THE DCB.               esr1
TLX0270 EQU   *                                                         esr1
        LB    U0,DCB.SV1X+3(UD)   GET USER'S OPTION CODE                esr1
        CHI   U0,4                IGNORE PARITY?                        esr1
        BE    TLX0250             IF SO, JUST CHECK FOR EOT             esr1
*                                                                       esr1
* COME HERE IF SELCH WROTE LESS THAN REQUESTED, OR IF                   esr1
*   THERE WAS A TAPE ERROR AND USER DID NOT REQUEST                     esr1
*     EXTENDED FUNCTION CODE 4                                          esr1
*                                                                       esr1
        LH    U0,DCB.RTRY(UD)   GET RETRY COUNT                         esr1
        CH    U0,DCB.RMAX(UD)   AND COMPARE AGAINST MAX COUNT           esr1
        BM    TLX0300           OK SO FAR, KEEP RETRYING                esr1
* RETRIES EXHAUSTED-- CHECK EXTENDED OPTION CODE.                       esr1
        LB    UB,DCB.SV1X+3(UD) GET EXTENDED CODE                       esr1
        CLH   UB,X'8'           IS IT USER SPECIFIED NUMBER OF          esr1
*                               RETRIES--DO NOT BACKSPACE               esr1
        BE    TLX2390           REPORT RECOVERABLE ERROR.               esr1
TLX0300 EQU   *                                                         esr1
        OC    U6,DCB.CCLC(UD)    CLEAR CONTROLLER BEFORE RETRYING       esr1
        LB    U2,DCB.CBSR(UD)    GET COMMAND BYTE                       esr1
        BAL   UB,TLX1400         GO TO COMMON LOGIC FOR BACKSPACE       esr1
        LH    U0,DCB.RTRY(UD)    GET RETRY COUNT                        esr1
        CH    U0,DCB.RMAX(UD)    COMPARE TO MAX RETRIES                 esr1
        BNM   TLX2390            FINISHED RETRIES, GET OUT              esr1
        AIS   U0,1               UP RETRY COUNT                         esr1
        STH   U0,DCB.RTRY(UD)    STORE                                  esr1
        BAL   UB,TLX1760         ERASE ONE GAP'S WORTH                  esr1
        B     TLX0120            GO DO WRITE AGAIN                      esr1
        SPACE
*
* COME HERE ONLY IF A TIME-OUT ERROR HAS OCCURRED
*
TLX0310 EQU   *
        LHI   U8,X'8282'          TIMEOUT STATUS
        STH   U8,DCB.STAT(UD)     STORE STATUS
        B     TLX2150             CHECK FOR REWIND IN PROGRESS & EXIT.
```

```
         SPACE
         TITLE READ OPERATIONS                                              dir
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *       dir
*              READ OPERATIONS                                              dir
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *       dir
TLX0320  EQU   *                                                            dir
*                                                                           dir
* ON A READ, IF WE ARE AT LOAD POINT, WE MUST ASSUME THAT                   dir
* WE ARE GOING TO READ AT 800 BPI. UNTIL THE TAPE HAS BEEN                  dir
* READ, ITS DENSITY IS UNKNOWN. SO, IF WE ARE AT LOAD POINT,                dir
* WE WILL RE-DO THE TIME-OUT CALCULATION.                                   dir
*                                                                           dir
         LI     U0,MAGBOTB                                                  dir
         TBT    U0,DCB.FLG1(UD)      TEST BOT FLAG                          dir
         BP     TLX0658              IF SO, BRANCH TO RE-CALCULATE          dir
TLX0325  EQU    *                                                           dir
         LIS    U0,0                INITIALIZE RETRY                        dir
         STH    U0,DCB.RTRY(UD)     COUNTER                                 dir
         LH     U0,DCB.RRTY(UD)     GET DEFAULT READ RETRY COUNT            dir
         L      U1,DCB.SV1X(UD)     GET THE EXTENDED OPTIONS                dir
         NHI    U1,X'1F'            SAVE ONLY OPTION FIELD                  dir
         CHI    U1,1                CHECK FOR USER RETRY COUNT              dir
         BNP    TLX0330             OPTION 0 OR 1, USE DEFAULT              dir
         LB     U0,DCB.SV1X(UD)     GET USER RETRY COUNT                    dir
TLX0330  STH    U0,DCB.RMAX(UD)     SET RETRY COUNT                         dir
TLX0340  EQU    *                   BEGIN READ OPERATION                    dir
         LI     UB,0                                                        dir
         ST     UB,DCB.ODDA(UD)     CLEAR ADDRESS SAVE AREA                 dir
         THI    U2,X'1'             ODD BYTE XFER?(U2  =XFER SIZE)          dir
         BZ     TLX0350             NO, GO PAST ODD BYTE WORK               dir
         L      U1,DCB.EADR(UD)     GET BUFFER ENDING ADDRESS               dir
         ST     U1,DCB.ODDA(UD)     SAVE THE END ADDRESS                    dir
         LB     U1,1(UB)            GET THE CONTENT OF NEXT BYTE            dir
         STB    U1,DCB.ODDC(UD)     SAVE IT                                 dir
TLX0350  LA     U1,TLX0360          SET UP INTERRUPT SERVICE ADDRESS        dir
         STH    U1,CCB.SUBA(UC)     * INTO THE CCB.                         dir
         LA     U1,TLX0470          SET THE EVENT SERVICE ROUTINE           dir
         ST     U1,DCB.ESR(UD)      INTO THE DCB FOR SQS TO USE.            dir
         LH     U1,DCB.SDN(UD)      GET SELCH ADDRESS                       dir
         LA     U0,1(UC)            MAKE ODD CCB ADDRESS                    dir
         STH    U0,ISPTAB(U1,U1)    SET SELCH ISPT ENTRY                    dir
         STH    U0,ISPTAB(U6,U6)    SET CONTROLLER ISPT ENTRY               dir
         BAL    U8,TOCHON           PUT ON TIMEOUT CHAIN                    dir
         LH     U8,DCB.ILVL(UD)      FETCH INTERRUPT LEVEL                dir
         SINT   U8,0(U6)             ENTER INTO THE ISR                   dir
         B      EVRTE                RETURN FROM DRIVER                   dir
         SPACE
         PURE
TLX0360  EQU    *                                                         isr4
         L      E5,CCB.DCB(E4)       A(DCB)                               isr4
         LH     E7,CCB.LBO(E4)       GET CALCULATED TIME-OUT              isr4
         AIS    E7,3                 ADD 3 SEC FOR 25 FT. ERASED TAPE     isr4
         STH    E7,DCB.TOUT(E5)      STORE IT                             isr4
         LH     E3,DCB.SDN(E5)       GET SELCH ADDRESS                    isr4
         OC     E3,CLEARSEL          CLEAR SELCH                          isr4
         WD     E3,DCB.SADR+1(E5)    WRITE START ADDRESS TO SELCH         isr4
         WH     E3,DCB.SADR+2(E5)    *                                    isr4
* ADJUST END ADDRESS AND WRITE IT TO SELCH                                isr4
         LH     E7,DCB.EADR+2(E5)                                         isr4
         OHI    E7,1                 MAKE IT ODD                          isr4
         WD     E3,DCB.EADR+1(E5)    WRITE END ADDRESS TO SELCH           isr4
         WHR    E3,E7                *                                    isr4
         LB     E6,DCB.CRDF(E5)      GET FOREWARD READ COMMAND            isr4
         L      E7,DCB.SV1X(E5)      GET EXTENDED OPTIONS                 isr4
         THI    E7,1                 IS IT REALLY BACKWARD?               isr4
         BZ     TLX0370              NO - IS FOREWARD                     isr4
         LB     E6,DCB.CRDB(E5)      GET BACKWARD READ COMMAND            isr4
TLX0370  EQU    *                                                         isr4
         STB    E6,DCB.CMD(E5)       SAVE THIS COMMAND                    isr4
         EPSR   E7,E7                GET CURRENT PSW                      isr4
         NHI    E7,X'B7FF'           TURN OFF ALL INTERRUPTS              isr4
         EPSR   E6,E7                AND GO NON-INTERRUPTABLE             isr4
         OC     E2,DCB.CENB(E5)      ENABLE INTERRUPTS FROM DRIVE         isr4
         OC     E2,DCB.CMD(E5)       START THE DRIVE                      isr4
         OC     E3,GOREAD            START THE SELCH                      isr4
         SPACE                                                            isr4
TLX0380  EQU    *                                                         isr4
         LA     E7,TLX0400           SET NEXT ISR ADDRESS                 isr4
         STH    E7,CCB.SUBA(E4)      *                                    isr4
         LPSWR  E0                   EXIT INTERRUPT SERVICE               isr4
         SPACE
GOREAD   DC     X'3400'              SELCH READ COMMAND
         SPACE
TLX0400  EQU    *                                                         isr5
         L      E5,CCB.DCB(E4)       GET DCB ADDRESS                      isr5
         LH     E6,DCB.TOUT(E5)      GET TIME-OUT VALUE                   isr5
         BZ     TLX0410             EXIT NOW IF TIMED OUT              isr5
         CLH    E2,DCB.SDN(E5)      SELCH INTERRUPT?                   isr5
         BNE    TLX0380             NOT,THE SELCH, -WAIT FOR SELCH     isr5
         LA     E7,III              SET TO IGNORE INTERRUPT            isr5
         STH    E7,ISPTAB(E2,E2)    IF THE SELCH BOTHERS US AGAIN.     isr5
         SPACE                                                         isr5
         OC     E2,CLEARSEL         STOP SELCH                         isr5
         LH     E2,DCB.DN(E5)       FETCH DEVICE ADDRESS               isr5
         SSR    E2,E3               SENSE DEVICE STATUS                isr5
         THI    E3,DU!NMTN          NO MOTION OR DU?                   isr5
         BNZ    TLX0440             IF SO, PRETEND DEVICE INTERRUPTED  isr5
         LA     E7,TLX0420          SET NEXT ISR ADDRESS               isr5
         STH    E7,CCB.SUBA(E4)     *                                  isr5
TLX0410  EQU    *                                                      isr5
         LPSWR  E0                  EXIT INTERRUPT SERVICE             isr5
         SPACE
TLX0420  EQU    *                                                      isr6
         L      E5,CCB.DCB(E4)      GET DCB ADDRESS                    isr6
         LH     E6,DCB.CCB(E5)      GET TIME-OUT VALUE                 isr6
         BZ     TLX0430             IF TIMED OUT, EXIT NOW             isr6
         CLH    E2,DCB.DN(E5)       INT. FROM DEVICE                   isr6
         BNE    TLX0430             NO, WAIT SOME MORE                 isr6
         THI    E3,NMTN!EOF!DU      TERR SETS NO MOTION WITH H/W CHNG  isr6
         BNZ    TLX0440             YES, PROCESS IT                    isr6
TLX0430  EQU    *                                                      isr6
         LPSWR  E0                  WAIT FOR INTERRUPT                 isr6
         SPACE                                                         isr6
TLX0440  EQU    *                                                      isr6
         STB    E3,DCB.DDPS(E5)     SAVE DRIVE STATUS                  isr6
         OC     E2,DCB.DSB0(E5)     SEND NOP CMD FOR STATUS            isr6
         LA     E7,TLX0450          SET NEXT ISR ADDRESS               isr6
         STH    E7,CCB.SUBA(E4)     *                                  isr6
         LPSWR  E0                  EXIT INTERRUPT SERVICE             isr6
*
TLX0450  EQU    *                                                      isr7
         L      E5,CCB.DCB(E4)      GET CCB ADDRESS                    isr7
         RHR    E2,E7               GET DEVICE STATUS HALFWORD         isr7
         STH    E7,CCB.MISC(E4)     SAVE FOR ESR USE                   isr7
         OC     E2,DCB.CDAR(E5)     NOW TURN OFF INTERRUPTS            isr7
         LH     E6,DCB.TOUT(E5)     CHECK TIMEOUT CONSTANT             isr7
         BNP    TLX0460             DON'T ADD TO QUEUE TWICE.          isr7
         LCS    E7,1                RESET THE TIME-OUT VALUE           isr7
         STH    E7,DCB.TOUT(E5)     *                                  isr7
         L      E7,DCB.LEAF(E5)       GET LEAF ADDRESS                    isr7
         ATL    E7,SQ                 PUT ESR ON QUEUE                    isr7
TLX0460  LPSWR  E0                    EXIT FROM ISR.                      isr7
         SPAC
         IMPUR
TERMTELX EQU    *                                                         esr2
TLX0470  EQU    *                                                         esr2
         LHL    UC,DCB.CCB(UD)        A(CCB)                              esr2
         LHL    U6,DCB.DN(UD)         DEVICE NUMBER                       esr2
         LH     U7,DCB.STAT(UD)       GET DEVICE DEPENDENT STATUS         esr2
         THI    U7,DU                 DID IT GO OFF-LINE?                 esr2
         BP     TLX2340               IF SO, EXIT HERE                    esr2
         L      U8,DCB.ODDA(UD)       GET THE CONTENT IN ADDRESS SAVE     esr2
         BZ     TLX0480               NOTHING, NORMAL I/O                 esr2
         LB     U0,DCB.ODDC(UD)       GET WHATEVER IN SAVE AREA           esr2
         STB    U0,1(U8)              PUT IT BACK                         esr2
         LI     U0,0                  REINITIALIZE SAVE AREA              esr2
         STB    U0,DCB.ODDC(UD)                                           esr2
         ST     U0,DCB.ODDA(UD)                                           esr2
TLX0480  LHL    U8,CCB.MISC(UC)       GET DEVICE STATUS HALFWORD          esr2
* NOW READ THE SELCH FINAL ADDRESS TO SEE IF WE GOT WHAT WE WANTED.       esr2
         LH      U2,DCB.SDN(UD)       GET THE SELCH ADDRESS               esr2
         OC      U2,CLEARSEL          AND ISSUE STOP & EXTENDED ADDR REA  esr2
         RDR     U2,U7                GET THE SELCH FINAL ADDRESS         esr2
         EXHR    U7,U7                *                                   esr2
         RHR     U2,U1                *                                   esr2
         OR      U7,U1                * INTO U7.                          esr2
         C       U7,DCB.SADR(UD)      IS FINAL ADDRESS LESS THAN START    esr2
         BM      TLX0610              IF SO WE MAY HAVE TO CRASH SYSTEM.  esr2
         C       U7,DCB.EADR(UD)      DID WE EXACTLY FILL OUR BUFFER?     esr2
         BNE     TLX0522              NO--MAY BE SHORT READ.(CHECK MORE)  esr2
         L       U7,CCB.XLT(UC)       GET THE LENGTH OF TRANSFER          esr2
TLX0490  EQU    *                                                         esr2
         ST     U7,DCB.LLXF(UD)       AND SAVE IT IN THE DCB.             esr2
         LH     U0,DCB.TOUT(UD)       CHECK FOR LENGTH OF TRANSFER        esr2
         BZ     TLX0655                                                   esr2
         BAL    U8,TOCHOFF            NOW REMOVE FROM TIME-OUT CHAIN      esr2
         LB     U0,DCB.DDPS(UD)       GET DEV.DEP.STAT.                   esr2
         THI    U0,TERR!ERR           TAPE ERROR?                         esr2
         BNZ    TLX0590               ERROR--GO SEE WHICH ERROR           esr2
* COME HERE IF NO ERROR STATUS. FOR NORMAL I/O, THE LENGTH OF             esr2
* TRANSFER MUST BE EXACTLY AS REQUESTED, OR IT IS DEFINED TO BE           esr2
* A PARITY ERROR.                                                         esr2
*                                                                   esr2
        LB    U0,DCB.SV1X+3(UD)  GET EXTENDED OPTION CODE           esr2
        CHI   U0,1               CHECK FOR NORMAL I/O               esr2
        BP    TLX0530            SPECIAL CASE - BRANCH.             esr2
        THI   U8,X'0001'         DID OVERRUN SET?                   esr2
        BZ    TLX0530            ALL O.K.-NO ERROR- GET OUT.        esr2
*                                                                   esr2
TLX0510 LIS   U0,0               SET UP FOR EOT/EOF CHECK           esr2
        ST    U0,DCB.LLXF(UD)    AND ZERO LENGTH OF TRANSFER        esr2
*                                (BY DEFINITION)                    esr2
TLX0520 EQU   *                                                     esr2
        LH    U8,DCB.STAT(UD)    GET THE STATUS                     esr2
        THI   U8,X'22'           END OF TAPE/END OF FILE?           esr2
        BNZ   TLX0530            GO SET STATUS & EXIT               esr2
        LB    U8,DCB.SV1X+3(UD)  IF EXTEND CODE 4, NO PARITY        esr2
        CHI   U8,4               ERROR IS PERMITTED                 esr2
        BE    TLX0570            SO DO NORMAL EXIT, BUT             esr2
*                                KEEP EOF/EOT.                      esr2
        CHI   U8,5               BACKWARD IGNORE PARITY CODE?       esr2
        BE    TLX0570            YES, BRANCH.                       esr2
        LI    U0,Y'82FA'         DEFINE THIS AS PARITY ERROR        esr2
        STH   U0,DCB.STAT(UD)    *                                  esr2
        B     TLX0630            GO TRY AGAIN.                      esr2
        SPACE                                                       esr2
** COME HERE IF WE NEED TO ADJUST LENGTH OF TRANSFER AND SELCH      esr2
* FINAL ADDRESS                                                     esr2
*                                                                   esr2
TLX0522 EQU  *                                                      esr2
        S    U7,DCB.SADR(UD)     CALCULATE THE ACTUAL LENGTH OF XFER esr2
        B    TLX0490             AND RETURN TO MAINLINE ROUTINE.    esr2
*                                                                   esr2
        SPACE                                                       esr2
TLX0530 EQU   *                                                     esr2
        LIS   U0,0               CLEAR STATUS                       esr2
*                                                                   esr2
* BUILD A 4-BIT INDEX INTO THE STATUS TABLE                         esr2
*                                                                   esr2
        LB    U1,DCB.SV1X+3(UD)  GET FOREWARD/BACKWARD FLAG         esr2
        NHI   U1,1               *                                  esr2
        LR    U0,U0              GET PARITY ERROR FLAG              esr2
        BZ    TLX0540                                               esr2
        OHI   U1,4                                                  esr2
TLX0540 LH    U0,DCB.STAT(UD)    GET EOT/BOT AND F.M. FLAGS         esr2
```

```
        THI    U0,X'02'             FILE MARK?                          esr2
        BZ     TLX0550                                                  esr2
        OHI    U1,2                                                     esr2
TLX0550 THI    U0,X'20'             EOT/BOT?                            esr2
        BZ     TLX0560                                                  esr2
        OHI    U1,8                                                     esr2
TLX0560 AR     U1,U1                MAKE HALF-WORD INDEX                esr2
        LH     U0,TLX0580(U1)       GET THE STATUS                      esr2
TLX0570 EQU    *                                                        esr2
        STH    U0,DCB.STAT(UD)      SET STATUS HERE.                    esr2
* IF WE COME HERE, THE READ OPERATION WAS A SUCCESS. EXIT ESR.          esr2
        B      TLX2150              CHECK FOR REWIND OP. AND EXIT.      esr2
        SPACE                                                           esr2
TLX0580 EQU    *     EOT/BOT PARITY  FILE MK  BACKWARD                  esr2
        DC     X'0000'  0       0        0        0                     esr2
        DC     X'0000'  0       0        0        1                     esr2
        DC     X'8800'  0       0        1        0                     esr2
        DC     X'8800'  0       0        1        1                     esr2
        DC     X'82FA'  0       1        0        0                     esr2
        DC     X'82FA'  0       1        0        1                     esr2
        DC     X'8AFA'  0       1        1        0    IMPOSSIBLE       esr2
        DC     X'8AFA'  0       1        1        1    IMPOSSIBLE       esr2
        DC     X'9000'  1       0        0        0                     esr2
        DC     X'92FE'  1       0        0        1                     esr2
        DC     X'9800'  1       0        1        0                     esr2
        DC     X'9AFE'  1       0        1        1                     esr2
        DC     X'92FA'  1       1        0        0                     esr2
        DC     X'92FE'  1       1        0        1                     esr2
        DC     X'9A00'  1       1        1        0    IMPOSSIBLE       esr2
        DC     X'9AFE'  1       1        1        1    IMPOSSIBLE       esr2
*                                                                       esr2
*                                                                       esr2
* WE MUST HANDLE THE SITUATION WHEREBY A TAPE CANNOT BE READ            esr2
* BECAUSE OF LARGE AREAS OF 'ERASED' TAPE (CREATED BY AN                esr2
* ERASE GAP FUNCTION). ATTEMPTS TO READ A FILE WHICH HAS AREAS          esr2
* OF ERASED TAPE RESULTS IN A DEVICE DEPENDENT STATUS OF                esr2
* X'9C' (ERR,NO-MOTION,BUSY,EXAMINE).  THE DEVICE STATUS                esr2
* HALFWORD WILL BE A X'1000', INDICATING THAT A FILE MARK WAS           esr2
* NOT DETECTED AS EXPECTED. (EMBO - READ BY NO-OP COMMAND-X'00')        esr2
* SET A SPECIAL STATUS SO THAT THE USER MAY ISSUE A FORWARD             esr2
* RECORD COMMAND TO BYPASS THE 'ERASED' AREA OF TAPE AND PROCEED        esr2
* WITH OPERATIONS.                                                      esr2
*                                                                       esr2
* MAKE SURE OTHER BITS IN EMBO ARE NOT SET--IF THEY ARE, DO NOT          esr2
* ASSUME THAT THE TAPE CANNOT BE READ DUE TO AREAS OF ERASED TAPE.       esr2
*                                                                        esr2
TLX0590  EQU    *                                                        esr2
         LR     U0,U8              DON'T WIPE OUT U8 YET, WE NEED IT     esr2
         NHI    U0,X'FF00'         STRIP OFF LOW BITS OF HALFWORD        esr2
         CHI    U0,X'1000'         FILE MARK ERROR?                      esr2
         BE     TLX1470            YES-GO DO SPECIAL PROCESSING          esr2
*                                                                        esr2
* WE HAVE AN ERROR. WE MUST DISTINGUISH BETWEEN 'REAL' PARITY            esr2
* ERRORS, AND SHORT READS THAT HAVE NO ERROR. WE MUST ALSO               esr2
* TREAT NORMAL I/O AND EXTENDED OPTION I/O DIFFERENTLY.                  esr2
*                                                                        esr2
* REMEMBER THAT U7 CONTAINS REQUESTED LENGTH OF TRANSFER. DON'T          esr2
* DESTROY IT, AS WE WILL USE IT LATER!                                   esr2
*                                                                        esr2
* IF WE HAVE AN ERROR, AND WE ARE DOING NORMAL I/O, THIS IS              esr2
* DEFINED AS A PARITY ERROR. SET THE APPROPRIATE CODE IN                 esr2
* THE DCB STATUS AND RETRY.                                              esr2
*                                                                        esr2
TLX0600  EQU    *                                                        esr2
         LB     U0,DCB.SV1X+3(UD)    GET EXTENDED OPTION CODE            esr2
         CHI    U0,1                 LOOK FOR NORMAL I/O                 esr2
         BNP    TLX0510              WE HAVE NORMAL I/O--BRANCH.         esr2
         LIS    U0,0                 SET UP U0 FOR EOT/EOF TEST          esr2
         C      U7,DCB.LLXF(UD)      CHECK LENGTH OF XFER                esr2
         BP     TLX0520              IF SO, IT IS AN ERROR               esr2
         B      TLX0530              CHECK EOF/EOT & QUIT.               esr2
TLX0610  EQU    *                                                        esr2
         OC     U6,DCB.CCLC(UD)      CLEAR CONTROLLER                    esr2
* IF WE COME HERE, THE SELCH START ADDRESS  WAS GREATER THAN             esr2
* THE SELCH FINAL ADDRESS. IF WE HAVE OVERWRITTEN O.S. CODE              esr2
* GIVE A CRASH CODE 500. OTHERWISE, GIVE THE USER A '8484'               esr2
* STATUS.                                                                esr2
         CI     U0,UBOT              HAVE WE OVERWRITTEN O.S.?           esr2
         BP     TLX0620              NO--GIVE THE USER AN ERROR.         esr2
         CRSH   502                  YES--CRASH THE SYSTEM (SORRY!)      esr2
TLX0620  EQU    *                                                        esr2
         LHI    U8,X'8484'           SELCH ERROR ON READ OR WRITE        esr2
         STH    U8,DCB.STAT(UD)      STORE STATUS                        esr2
         B      TLX2150              CHECK FOR RWND IN PROGRESS & EXIT   esr2
         SPACE                                                           esr2
TLX0630  EQU    *                                                        esr2
```

```
         OC     U6,DCB.CCLC(UD)       CLEAR DEVICE                     esr2
         LH     U0,DCB.RTRY(UD)       GET RETRY COUNT                  esr2
         CH     U0,DCB.RMAX(UD)       AND COMPARE AGAINST MAX COUNT    esr2
         BM     TLX0640               OK SO FAR, KEEP RETRYING         esr2
* RETRIES EXHAUSTED-- CHECK EXTENDED OPTION CODE.                      esr2
         LB     UB,DCB.SV1X+3(UD)     GET EXTENDED CODE                esr2
         CLHI   UB,X'8'               IS IT USER SPECIFIED NUMBER OF   esr2
*                                     RETRIES--DO NOT BACKSPACE        esr2
         BE     TLX2390               REPORT RECOVERABLE ERROR.        esr2
         CLHI   UB,X'9'               BACKWARD OPERATION--DO NOT       esr2
         BE     TLX2390               FORWARD SPACE AFTER ERROR.       esr2
TLX0640  EQU    *                                                      esr2
         LB     U2,DCB.CBSR(UD)       GET COMMAND BYTE                 esr2
         LB     UB,DCB.SV1X+3(UD)     GET EXTENDED OPTION CODE         esr2
         THI    UB,1                  IS IT ODD (IF SO BACKWARD OP)    esr2
         BZ     TLX0650               FORWARD OPERATION--DO BSR.       esr2
         LB     U2,DCB.CFSR(UD)       BACKWARD OPERATION--DO FSR.      esr2
TLX0650  EQU    *                                                      esr2
         BAL    UB,TLX1400            GO TO COMMON LOGIC FOR BACKSPACE esr2
         LH     U0,DCB.RTRY(UD)       CHECK RETRY COUNTER              esr2
         CH     U0,DCB.RMAX(UD)       COMPARE TO MAX RETRIES           esr2
         BNM    TLX2390               NO MORE RETRIES--GIVE UP.        esr2
         AIS    U0,1                  ADD 1 TO THE COUNTER             esr2
         STH    U0,DCB.RTRY(UD)       INCREMENT RETRY COUNT            esr2
         B      TLX0340                                                esr2
*                                                                      esr2
* COME HERE ONLY IF WE TIME OUT WAITING FOR INTERRUPTS                 esr2
* DURING A READ OPERATION                                              esr2
*                                                                      esr2
TLX0655  EQU    *                                                      esr2
         LHI    U0,X'2050'            GO NON-INTERRUPTABLE             esr2
         EPSR   U1,U0                 *                                esr2
         SSR    U6,U2                 GET DEVICE STATUS                esr2
         THI    U2,X'91'              DID ANYTHING HAPPEN?             esr2
         BNZ    TLX0657               YES - IT DIED OR SOMETHING       esr2
         LH     U0,DCB.MAXT(UD)       NO - SET TIME-OUT VALUE AGAIN    esr2
         STH    U0,DCB.TOUT(UD)       AND WAIT FOR LONG FILE (MAYBE)   esr2
         EPSR   U0,U1                 NOW GET OUT                      esr2
         B      EVRTE                 *                                esr2
TLX0657  EQU    *                                                      esr2
         EPSR   U0,U1                 TIMED-OUT, REALLY - FINISH UP    esr2
         BAL    U8,TOCHOFF            GET OFF THE TIMEOUT CHAIN        esr2
         LHI    U8,X'8263'            SET UP STATUS FIELD              esr2
         STH    U8,DCB.STAT(UD)        STORE STATUS                        esr2
         B     TLX2150                CHECK FOR RWND IN PROGRESS & EXIT   esr2
*
* CALCULATE THE APPROPRIATE TIME-OUT VALUE, BASED ON
* THE ASSUMPTION THAT THE DENSITY IS 800 BPI.
*
TLX0658  EQU    *
         LIS    U1,1                   SET UP TO CALCULATE TIME-OUT
         LR     U0,U2                  GET COMPUTED XFER LENGTH
TLX0659  AIS    U1,1                   ADD 1 SECOND FOR EACH RATE SIZE
         S      U0,DCB.XRT(UD)         COMPARE SIZE TO RATE AT 800 BPI
         BP     TLX0659                LOOP UNTIL SIZE FIELD IS COUNTED
         STH    U1,CCB.MISC(UC)        SAVE TIME-OUT TEMPORARILY
         B      TLX0325                GO BACK TO NORMAL PROCESSING.
*
         SPACE
         TITLE COMMAND FUNCTION DECODING
         IMPUR
TLX1280  EQU    *                                                          dir
* NOTE - U2 MUST BE PRESERVED IF THE REQUEST IF FOR                        dir
* AN 'ERASE BUFFER' FUNCTION.                                              dir
         CHI    U3,X'C0'               REWIND ?                            dir
         BE     TLX1350                                                    dir
         CHI    U3,X'A0'               BACK SPACE RECORD?                  dir
         BE     TLX1370                                                    dir
         CHI    U3,X'90'               FOREWARD SPACE RECORD?              dir
         BE     TLX1360                                                    dir
         CHI    U3,X'88'               WRITE FILE MARK                     dir
         BE     TLX1490                                                    dir
         CHI    U3,X'84'               FOREWARD FILE MARD                  dir
         BE     TLX1600                                                    dir
         CHI    U3,X'82'               BACKWARD FILE MARK                  dir
         BE     TLX1590                                                    dir
         CHI    U3,X'81'               DEVICE DEPENDENT FUNCTION?          dir
         BE     TLX1320                                                    dir
         B      TLX2360                ERROR, ILLEGAL FUNCTION CODE        dir
         SPACE                                                             dir
TLX1320  EQU    *                                                          dir
         LA     UB,TLX2150             SET LINK REGISTER TO CHECK RWD.     dir
         LIS    U0,TOPT.X1B            CHECK THE EXTEND OPT IN TASK        dir
         L      U1,DCB.TCB(UD)         GET TCB ADDRESS                     dir
         TBT    U0,TCB.OPT(U1)         CHECK THE EXTEND OPTION BIT         dir
         BZ     TLX2360                EXTEND OPT IS NOT SET, BRANCH       dir
         LB     U1,DCB.SV1X+3(UD)      GET THE EXTENDED OPTION FIELD       dir
         CHI    U1,7                   ERASE GAP?                          dir
         BE     TLX1760                                                    dir
         CHI    U1,10                  ERASE BUFFER?                       dir
         BE     TLX1710                                                    dir
         CHI    U1,0                   UNLOAD/                             dir
         BE     TLX1700                                                    dir
         CHI    U1,8                   READ STATUS?                        dir
         BE     TLX1860                                                    dir
         B      TLX2360                NONE OF THE ABOVE - ILLEGAL         dir
         TITLE REWIND                                                      dir
TLX1350  EQU    *                                                          dir
         LI     U3,MAGBOTB             IS THE TAPE AT BOT?                 dir
         TBT    U3,DCB.FLG1(UD)        *                                   dir
         BNZ    TLX2150                CHECK FOR OTHER REWINDS OCCURRING   dir
         LI     U3,MAGEOTB             IS THE TAPE AT EOT?                 dir
         RBT    U3,DCB.FLG1(UD)        RESET BIT (SOON BE AT LOAD PT.)     dir
         LB     U2,DCB.CREW(UD)        FETCH COMMAND                       dir
         STB    U2,DCB.CMD(UD)         SAVE COMMAND                        dir
         OCR    U6,U2                  ISSUE COMMAND                       dir
         LHI    U0,200                 STALL FOR TIME                      dir
         SIS    U0,1                   (THIS IS STUPID)                    dir
         BTBS   2,1                    * * *                               dir
         LI     U0,MAGREWB                                                 dir
         SBT    U0,DCB.FLG1(UD)        SET REWIND FLAG                     dir
         B      TLX2150                CHECK FOR OTHER REWINDS             dir
         SPACE                                                             dir
         TITLE BACKSPACE AND FOREWARD SPACE RECORD                         dir
*                                                                          dir
* FOREWARD SPACE RECORD ENTRY POINT                                        dir
*                                                                          dir
TLX1360  EQU    *                                                          dir
         LI     U0,MAGEOTB             CHECK THE EOT CONDITION             dir
         RBT    U0,DCB.FLG1(UD)        RESET EOT FLAG                      dir
         LB     U2,DCB.CFSR(UD)        GET COMMAND BYTE                    dir
         B      TLX1390                GO TO COMMON PROCESSING             dir
*                                                                          dir
* BACKSPACE RECORD ENTRY POINT                                             dir
*                                                                          dir
TLX1370  EQU    *                                                          dir
         LI     U0,MAGBOTB                                                 dir
         TBT    U0,DCB.FLG1(UD)        CHECK BOT FLAG                      dir
         BZ     TLX1380                NOT SET ! BRANCH                    dir
         B      TLX2150             CHECK FOR REWIND CONDITIONS       dir
TLX1380  EQU    *                                                     dir
         LB     U2,DCB.CBSR(UD)     GET COMMAND BYTE.                 dir
*                                                                     dir
* COMMON BACKSPACE RECORD AND FOREWARD SPACE RECORD LOGIC             dir
*                                                                     dir
TLX1390  EQU    *                                                     dir
         BAL    UB,TLX1400                                            dir
         LIS    U0,0                CLEAR REGISTER                    dir
         OH     U0,DCB.STAT(UD)     OR STATUS                         dir
         STH    U0,DCB.STAT(UD)     SAVE STATUS                       dir
         B      TLX2150             CHECK FOR REWIND CONDITIONS.      dir
TLX1400  EQU     *                                                    dir
         ST     UB,CCB.EB0(UC)      SAVE RETURN REGISTER              dir
         STB    U2,DCB.CMD(UD)      SAVE COMMAND                      dir
         LA     U1,TLX1410          GET INTERRUPT SERVICE ADDRESS     dir
         STH    U1,CCB.SUBA(UC)     SETUP ISR POINTER                 dir
         LA     U1,TLX1460          SET THE EVENT SERVICE ADDRESS     dir
         ST     U1,DCB.ESR(UD)      * INTO THE DCB FOR SQS.           dir
         LA     U0,1(UC)            MAKE ODD CCB ADDRESS              dir
         STH    U0,ISPTAB(U6,U6)    SET CONTROLLER ISPT ENTRY         dir
         BAL    U8,TOCHON           GET ON TIMER CHAIN                dir
         LH     U8,DCB.ILVL(UD)     FETCH INTERRUPT LEVEL             dir
         SINT   U8,0(U6)            ENTER ISR                         dir
         LI     UE,3                LEVEL                             dir
         L      UF,DCB.LEAF(UD)     LEAF ADDRESS                      dir
         BAL    U8,EVREL            RELEASE SELCH                     dir
         B      EVRTE               RETURN FROM ESR                   dir
         SPACE
         PURE
TLX1410  EQU    *                                                     isr8
         L      E5,CCB.DCB(E4)      A(DCB)                            isr8
         OC     E2,DCB.CENB(E5)     ENABLE INTERRUPTS                 isr8
         LH     E6,DCB.MAXT(E5)     MAXIMUM TIMEOUT VALUE             isr8
         STH    E6,DCB.TOUT(E5)     START TIMER                       isr8
         OC     E2,DCB.CMD(E5)      ISSUE COMMAND                     isr8
         LA     E7,TLX1420          SET NEXT ISR ADDRESS              isr8
         STH    E7,CCB.SUBA(E4)     *                                 isr8
         LPSWR  E0                  EXIT INTERRUPT SERVICE            isr8
TLX1420  EQU    *                                                     isr8
         L      E5,CCB.DCB(E4)      GET DCB ADDRESS                   isr8
         LH     E6,DCB.CCB(E5)      GET TIME-OUT VALUE                isr8
         BZ     TLX1425             IF TIMED OUT, EXIT NOW            isr8
         THI    E3,X'11'            CHECK NMTN & DU                 isr8
         BNZ    TLX1430             YES, GO TOWARDS ESR             isr8
TLX1425  EQU    *                                                   isr8
         LPSWR  E0                  GO WAIT FOR ANOTHER INTERRUPT   isr8
         SPACE
TLX1430  EQU    *                                                   isr9
         STB    E3,DCB.DDPS(E5)     SAVE STATUS                     isr9
         OC     E2,DCB.DSB0(E5)     SEND NOP CMD FOR STATUS         isr9
         LA     E7,TLX1440          SET NEXT ISR ADDRESS            isr9
         STH    E7,CCB.SUBA(E4)     *                               isr9
         LPSWR  E0                  EXIT INTERRUPT SERVICE          isr9
*
TLX1440  EQU    *                                                   isra
         L      E5,CCB.DCB(E4)      GET CCB ADDRESS                 isra
         RHR    E2,E7               GET DEVICE STATUS HALFWORD      isra
         STH    E7,CCB.MISC(E4)     SAVE FOR ESR USE                isra
         OC     E2,DCB.CDAR(E5)     NOW TURN OFF INTERRUPTS         isra
         LH     E6,DCB.TOUT(E5)     CHECK TIMEOUT CONSTANT          isra
         BNP    TLX1450             DON'T ADD TO QUEUE TWICE.       isra
         LCS    E7,1                RESET THE TIME-OUT VALUE        isra
         STH    E7,DCB.TOUT(E5)     *                               isra
         L      E7,DCB.LEAF(E5)     GET LEAF ADDRESS                isra
         ATL    E7,SQ               PUT ESR ON QUEUE                isra
TLX1450  LPSWR  E0                  EXIT FROM ISR.                  isra
         SPACE
         IMPUR
TLX1460  EQU    *                                                   esr3
         LHL    UC,DCB.CCB(UD)      A(CCB)                          esr3
         LHL    U6,DCB.DN(UD)       DEVICE NUMBER                   esr3
         BAL    U8,TOCHOFF          REMOVE FROM TIMER CHAIN         esr3
         LH     U7,DCB.TOUT(UD)     CHECK FOR TIME OUT              esr3
         BZ     TLX1480             BRANCH IF WE TIMED OUT.         esr3
         LHL    U8,CCB.MISC(UC)     GET DEVICE STATUS HALFWORD      esr3
         NHI    U8,X'FF00'          MASK OFF UNWANTED BITS          esr3
         CHI    U8,X'1000'          FILE MARK ERROR?                esr3
         BE     TLX1470             NO, KEEP GOING                  esr3
         LIS    U1,0                INITIALIZE STATUS FIELD         esr3
         LH     UB,DCB.STAT(UD)     GET DEVICE DEPENDENT STATUS     esr3
         THI    UB,DU!EOM!EOF       DID SOMETHING UNUSUAL HAPPEN?   esr3
         BP     TLX1482             IF SO GO TO SPECIAL LOGIC       esr3
         STH    U1,DCB.STAT(UD)     SETS ZERO STATUS                esr3
         L      UB,CCB.EB0(UC)      RESTORE LINK REGISTER           esr3
         BR     UB                  EXIT                            esr3
```

```
*                                                                esr3
TLX1470  EQU    *                                                esr3
* SET SPECIAL STATUS FOR FILE MARK ERROR ON ERASED TAPE.         esr3
         LHI    U1,X'82AA'          FILE MARK ERROR STATUS       esr3
         LH     UB,DCB.STAT(UD)     GET THE DEVICE DEPENDENT STATUS  esr3
         THI    UB,EOM              WAS THERE ALSO END OF TAPE?  esr3
         BNZ    TLX1485             IF EOT, GO CHECK FOR BOT/EOT esr3
         STH    U1,DCB.STAT(UD)     SAVE THE STATUS              esr3
         B      TLX2150             AND GO CHECK REWIND STATUS   esr3
* TIMEOUT ESR FOR BACKSPACE AND FOREWARD SPACE RECORD            esr3
TLX1480  EQU    *                                                esr3
         OC     U6,DCB.CDAR(UD)     DISARM INTERPT               esr3
         LHI    U8,X'8271'          TIMEOUT ON BACKSPACE RECORD  esr3
         STH    U8,DCB.STAT(UD)     STORE STATUS                 esr3
         B     TLX2150              CHECK FOR RWND IN PROGRESS & EXIT  esr3
*                                                                esr3
* COME HERE ON FSR OR BSR OPERATION ONLY IF DU,EOM,OR EOF IS     esr3
* SET AT THE CONCLUSION OF THE OPERATION.                        esr3
*                                                                esr3
TLX1482  EQU    *                                                esr3
         THI    UB,DU               DID DRIVE GO OFF-LINE?       esr3
         BP     TLX2340             IF SO, GET OUT.              esr3
         THI    UB,EOF              WAS IT END-OF-FILE?          esr3
         BZ     TLX1485             IF NOT SKIP TO END/BEGIN OF TAPE  esr3
         OHI    U1,X'8800'          SET END OF FILE STATUS.      esr3
         STH    U1,DCB.STAT(UD)     SAVE THE STATUS              esr3
TLX1485  EQU    *                   BRANCH HERE IF EOM IS SET    esr3
         BAL    U9,TLX2050          GO SEE IF WE ARE AT EOT OR BOT  esr3
         LI     U8,MAGEOTB          NOW TEST FOR EOT.            esr3
         TBT    U8,DCB.FLG1(UD)     *                            esr3
         BZ    TLX1487              NO EOT, GO CHECK REWIND STATUS  esr3
         OHI    U1,X'9000'          YES, IS EOT, SET STATUS.     esr3
TLX1487  EQU    *                                                esr3
         STH    U1,DCB.STAT(UD)     *                            esr3
         B      TLX2150             AND THEN CHECK REWIND STATUS esr3
         TITLE WRITE FILE MARK
TLX1490  EQU    *                                                dir
         BAL    UB,TLX1940          CHECK WRITE PROTECTED -      dir
*                                   WILL NOT RETURN IF DRIVE PROTECTED dir
* SET UP RETRY COUNTERS                                          dir
*                                                                dir
         LIS    U0,0                INITIALIZE RETRY             dir
         STH    U0,DCB.RTRY(UD)     COUNTER                      dir
         LH     U1,DCB.WRY1(UD)     GET DEFAULT WRITE RETRY COUNT     dir
         L      U0,DCB.SV1X(UD)     GET EXTENDED FUNCTION CODE        dir
         BZ     TLX1500             IF EXT. FNCTN = 0, USE DEFAULT    dir
         LB     U1,DCB.SV1X(UD)     IF NOT, USE USER'S COUNT          dir
TLX1500  STH    U1,DCB.RMAX(UD)     SET RETRY COUNT.                  dir
*                                                                     dir
* SET UP FOR ESR AND ISR SCHEDULING AND EXECUTION                     dir
*                                                                     dir
TLX1510  EQU    *                                                     dir
         LB     U2,DCB.CWFM(UD)     FETCH COMMAND                     dir
         STB    U2,DCB.CMD(UD)      SAVE COMMAND                      dir
         LA     U1,TLX1520          SET THE INTERRUPT SERVICE ADDR    dir
         STH    U1,CCB.SUBA(UC)     *INTO THE ISPT.                   dir
         LA     U1,TLX1550          SET THE EVENT SERVICE ADDRESS     dir
         ST     U1,DCB.ESR(UD)      * INTO THE DCB FOR SQS.           dir
         LA     U0,1(UC)            MAKE ODD CCB ADDRESS              dir
         STH    U0,ISPTAB(U6,U6)    SET CONTROLLER ISPT ENTRY         dir
         BAL    U8,TOCHON           PUT ONTO TIMER CHAIN              dir
         LH     U8,DCB.ILVL(UD)     FETCH INTERRUPT LEVEL             dir
         SINT   U8,0(U6)            ENTER ISR                         dir
         LI     UE,3                LEVEL                             dir
         L      UF,DCB.LEAF(UD)     LEAF ADDRESS                      dir
         BAL    U8,EVREL            RELEASE SELCH                     dir
         B      EVRTE               RETURN FROM ESR                   dir
         SPACE
         PURE
TLX1520  EQU    *                                                     isrb
         L      E5,CCB.DCB(E4)      A(DCB)                            isrb
         OC     E2,DCB.CENB(E5)     ENABLE INTERRUPTS                 isrb
         LH     E6,DCB.MAXT(E5)     MAXIMUM TIMEOUT VALUE             isrb
         STH    E6,DCB.TOUT(E5)     START TIMER                       isrb
         OC     E2,DCB.CMD(E5)      ISSUE COMMAND                     isrb
         LA     E7,TLX1530          GET NEXT ISR ADDRESS              isrb
         STH    E7,CCB.SUBA(E4)     *                                 isrb
         LPSWR  E0                  EXIT INTERRUPT SERVICE            isrb
TLX1530  EQU    *                                                     isrc
         L      E5,CCB.DCB(E4)      GET DCB ADDRESS                   isrc
         LH     E6,DCB.CCB(E5)      GET TIME-OUT VALUE                isrc
         BZ     TLX1535             IF TIMED OUT, EXIT NOW            isrc
         STB    E3,DCB.DDPS(E5)     SAVE STATUS                       isrc
         THI    E3,X'11'            CHECK NMTN & DU                   isrc
         BNZ    TLX1540             YES, GO TOWARDS ESR               isrc
TLX1535  EQU    *                                                     isrc
```

```
         LPSWR E0                    GO WAIT FOR ANOTHER INTERRUPT      isrc
         SPACE
TLX1540  EQU   *                                                        isrd
         OC    E2,DCB.CDAR(E5)       TURN OFF INTERRUPTS                isrd
         LH    E6,DCB.TOUT(E5)       GET TIME-OUT COUNTER               isrd
         BTFS  2,2                   RETURN NOW IF TIMED-OUT            isrd
         LPSWR E0                    DON'T RE-ADD TO S.Q.               isrd
         LCS   E6,1                  OTHERWISE SET MINUS TO             isrd
         STH   E6,DCB.TOUT(E5)       SHOW WE HAVE BEEN HERE.            isrd
         L     E6,DCB.LEAF(E5)       GET LEAF ADDRESS                   isrd
         ATL   E6,SQ                 SCHEDULE DRIVER TERMINATION        isrd
         LPSWR E0                    EXIT AND WAIT FOR TERM.            isrd
         SPACE
         IMPUR
TLX1550  EQU   *                                                        esr4
         LHL   UC,DCB.CCB(UD)        A(CCB)                             esr4
         LHL   U6,DCB.DN(UD)         DEVICE NUMBER                      esr4
         BAL   U8,TOCHOFF            REMOVE FROM TIMER CHAIN            esr4
         LH    U8,DCB.TOUT(UD)       CHECK FOR TIME-OUT                 esr4
         BZ    TLX1570               TIMED OUT? YES, BRANCH             esr4
         LH    UB,DCB.STAT(UD)       GET DEVICE DEPENDENT STATUS        esr4
         THI   UB,DU                 DID WE GO DU?                      esr4
         BP    TLX2340               IF SO,EXIT.                        esr4
         THI   UB,X'0002'            IS TAPE MARK STATUS SET?           esr4
         BZ    TLX1580               NO, GO RETRY                       esr4
         LIS   U0,0                                                     esr4
         THI   UB,X'20'              EOT/BOT SET?                       esr4
         BZ    TLX1560               NO - GO ON                         esr4
         OHI   U0,X'9000'            YES - SET USUAL STATUS             esr4
         LI    U1,MAGWAT2B           AND SET FLAG FIELD                 esr4
         SBT   U1,DCB.FLG1(UD)       *                                  esr4
TLX1560  EQU   *                                                        esr4
         STH   U0,DCB.STAT(UD)       PLACE FOR THE USER                 esr4
         B     TLX2150               EXIT AND CHECK FOR REWINDS.        esr4
TLX1570  EQU   *                                                        esr4
* TIME OUT ON WRITE FILE MARK OPERATION.                                 esr4
         SSR   U6,U1                 SENSE STATUS                       esr4
         THI   U1,DU                                                    esr4
         BNZ   TLX2340                                                  esr4
         LHI   U8,X'8275'            INDICATE TIME OUT ON WFM.          esr4
         STH   U8,DCB.STAT(UD)       STORE STATUS                       esr4
         B     TLX2150               CHECK FOR RWND IN PROGRESS & EXIT  esr4
         SPACE                                                          esr4
```

```
TLX1580  EQU    *                                                    esr4
         LB     U2,DCB.CBSR(UD)      GET COMMAND BYTE                esr4
         BAL    UB,TLX1400           GO TO COMMON LOGIC FOR BACKSPACE  esr4
         LH     U0,DCB.RTRY(UD)      RETRY COUNTS                    esr4
         CH     U0,DCB.RMAX(UD)      COMPARE TO MAX                  esr4
         BNM    TLX2370                                              esr4
         AIS    U0,1                 UP RETRY COUNT                  esr4
         STH    U0,DCB.RTRY(UD)      STORE IT                        esr4
         B      TLX1510              ATTEMPT TO WRITE FM             esr4
         SPACE
         TITLE FORWARD FILE MARK AND BACK FILE MARK
*
* BACK FILE MARK ENTRY
*
TLX1590  EQU    *                                                    dir
         LI     U0,MAGBOTB                                           dir
         TBT    U0,DCB.FLG1(UD)      CHECK BOT FLAG                  dir
         LB     U2,DCB.CBFM(UD)      GET COMMAND BYTE                dir
         BZ     TLX1610              NOT SET ! BRANCH                dir
         B      TLX2150              GO CHECK REWIND CONDITIONS      dir
         SPACE                                                       dir
*                                                                    dir
* FOREWARD FILE MARK ENTRY                                           dir
*                                                                    dir
TLX1600  EQU    *                                                    dir
         LI     U0,MAGEOTB           CHECK THE EOT CONDITION         dir
         RBT    U0,DCB.FLG1(UD)      RESET EOT FLAG                  dir
         LB     U2,DCB.CFFM(UD)      FETCH COMMAND                   dir
         SPACE                                                       dir
*                                                                    dir
* COMMON BACKFILE AND FOREWARD FILE LOGIC STARTS HERE                dir
*                                                                    dir
TLX1610  EQU    *                                                    dir
         STB    U2,DCB.CMD(UD)       SAVE COMMAND                    dir
         LA     U1,TLX1620           SET THE INTERRUPT SERVICE ADDR  dir
         STH    U1,CCB.SUBA(UC)      * INTO THE CCB.                 dir
         LA     U1,TLX1650           SET THE EVENT SERVICE ADDRESS   dir
         ST     U1,DCB.ESR(UD)       * INTO THE DCB FOR SQS.         dir
         LA     U0,1(UC)             MAKE ODD CCB ADDRESS            dir
         STH    U0,ISPTAB(U6,U6)     SET CONTROLLER ISPT ENTRY       dir
         BAL    U8,TOCHON            PUT ONTO TIMER CHAIN            dir
         LH     U8,DCB.ILVL(UD)      FETCH INTERRUPT LEVEL           dir
         SINT   U8,0(U6)             ENTER ISR                       dir
         LI     UE,3               LEVEL                          dir
         L      UF,DCB.LEAF(UD)    LEAF ADDRESS                   dir
         BAL    U8,EVREL           RELEASE SELCH                  dir
         B      EVRTE              RETURN FROM ESR                dir
         SPACE
         PURE                                                     isre
TLX1620  EQU    *                                                 isre
         L      E5,CCB.DCB(E4)     A(DCB)                         isre
         OC     E2,DCB.CENB(E5)    ENABLE INTERRUPTS              isre
         LH     E6,DCB.MAXT(E5)    MAXIMUM TIMEOUT VALUE          isre
         STH    E6,DCB.TOUT(E5)    START TIMER                    isre
         OC     E2,DCB.CMD(E5)     ISSUE COMMAND                  isre
         LA     E7,TLX1630         SET NEXT ISR ADDRESS           isre
         STH    E7,CCB.SUBA(E4)    *                              isre
         LPSWR  E0                 EXIT INTERRUPT SERVICE         isre
TLX1630  EQU    *                                                 isre
         L      E5,CCB.DCB(E4)     GET DCB ADDRESS                isre
         LH     E6,DCB.CCB(E5)     GET TIME-OUT VALUE             isre
         BZ     TLX1635            IF TIMED OUT, EXIT NOW         isre
         THI    E3,X'11'                                          isre
         BNZ    TLX1640            YES, GO TOWARDS ESR            isre
TLX1635  EQU    *                                                 isre
         LPSWR  E0                 GO WAIT FOR ANOTHER INTERRUPT  isre
         SPACE
TLX1640  EQU    *                                                 isrf
         STB    E3,DCB.DDPS(E5)    SAVE STATUS                    isrf
         OC     E2,DCB.CDAR(E5)    TURN OFF INTERRUPTS            isrf
         LH     E6,DCB.TOUT(E5)    GET TIME-OUT COUNTER           isrf
         BTFS   2,2                RETURN NOW IF TIMED-OUT        isrf
         LPSWR  E0                 DON'T RE-ADD TO S.Q.           isrf
         LCS    E6,1               OTHERWISE SET MINUS TO         isrf
         STH    E6,DCB.TOUT(E5)    SHOW WE HAVE BEEN HERE.        isrf
         L      E6,DCB.LEAF(E5)    GET LEAF ADDRESS               isrf
         ATL    E6,SQ              SCHEDULE DRIVER TERMINATION    isrf
         LPSWR  E0                 EXIT AND WAIT FOR TERM.        isrf
         SPACE
         IMPUR                                                    esr5
TLX1650  EQU    *                                                 esr5
         LHL    UC,DCB.CCB(UD)     A(CCB)                         esr5
         LHL    U6,DCB.DN(UD)      DEVICE NUMBER                  esr5
         LH     U7,DCB.TOUT(UD)    CHECK FOR TIME OUT             esr5
         BZ     TLX1680            TIMED OUT? YES, BRANCH         esr5
         BAL    U8,TOCHOFF         REMOVE FROM TIMEOUT CHAIN      esr5
         LH     UB,DCB.STAT(UD)      GET THE STATUS FIELDS               esr5
         THI    UB,DU                DID WE GO OFF-LINE?                 esr5
         BP     TLX2340              GET OUT, IF SO                      esr5
         LIS    UO,O                 CLEAR STATUS HERE.                  esr5
         THI    UB,X'0002'           WE MUST BE ON A FILEMARK.           esr5
         BZ     TLX1670              CHECK TYPE OF OPERATION             esr5
         THI    UB,EOM               END OF TAPE CONDITION?              esr5
         BNZ    TLX1660              END OF TAPE DETECTED                esr5
         STH    UO,DCB.STAT(UD)      SET STATUS FINALLY.                 esr5
         B      TLX2150              GO CHECK FOR REWIND CONDITIONS.     esr5
TLX1660  EQU    *                                                        esr5
         LI     U1,MAGWAT2B          SET EOT FLAG                        esr5
         SBT    U1,DCB.FLG1(UD)      *                                   esr5
         LHI    UO,X'9000'           INDICATE EOT IN STATUS FIELD        esr5
         STH    UO,DCB.STAT(UD)      SET STATUS FINALLY.                 esr5
         B      TLX2150              GO CHECK FOR REWIND CONDITIONS.     esr5
TLX1670  EQU    *                                                        esr5
* NO EOF DETECTED--THIS IS O.K. ONLY ON BACKFILE OPERATIONS               esr5
* WHEN THE TAPE IS AT LOAD POINT                                         esr5
         LB     U1,DCB.FC(UD)        GET THE FUNCTION                    esr5
         CHI    U1,X'82'             BACK FILE OPERATION?                esr5
         BNE    TLX2350              UNRECOVERABLE ERROR-NO FILEMARK     esr5
         THI    UB,EOM               END OF TAPE CONDITION?              esr5
         BZ     TLX2350              NO - REALLY AN ERROR.               esr5
         STH    UO,DCB.STAT(UD)      SAVE THE STATUS                     esr5
         B      TLX2150              GO CHECK REWIND CONDITIONS.         esr5
         SPACE                                                           esr5
TLX1680  EQU    *                                                        esr5
         LHI    UO,X'2050'           GO NON-INTERRUPTABLE                esr5
         EPSR   U1,UO                *                                   esr5
         SSR    U6,U2                GET DEVICE STATUS                   esr5
         THI    U2,X'D1'             DID ANYTHING HAPPEN?                esr5
         BNZ    TLX1690              YES - IT DIED OR SOMETHING          esr5
         LH     UO,DCB.MAXT(UD)      NO - SET TIME-OUT VALUE AGAIN       esr5
         STH    UO,DCB.TOUT(UD)      AND WAIT FOR LONG FILE (MAYBE)      esr5
         EPSR   UO,U1                NOW GET OUT                         esr5
         B      EVRTE                *                                   esr5
TLX1690  EQU    *                                                        esr5
         EPSR   UO,U1                TIMED-OUT, REALLY - FINISH UP       esr5
* TIMEOUT ON FILE MARK OPERATION.                                        esr5
* REMOVE FROM TIMEOUT CHAIN NOW. HOWEVER, WHEN PERFORMING LONG           esr5
* FORWARD AND BACK FILE OPERATIONS, WE MAY TIMEOUT SEVERAL TIMES         esr5
* WITHOUT EVER REPORTING THE TIMEOUT TO THE USER.  THEREFORE             esr5
* WE MUST STAY ON THE TIMEOUT CHAIN UNTIL WE ARE READY TO GIVE                esr5
* UP AND GIVE THE USER A TIMEOUT STATUS.                                      esr5
         BAL   U8,TOCHOFF          REMOVE FROM TIMER CHAIN.                   esr5
         LHI   U8,X'8272'          SET TIME OUT FOR ERROR MESSAGE             esr5
         STH   U8,DCB.STAT(UD)     STORE STATUS                               esr5
         B     TLX2150             CHECK FOR RWD IN PROGRESS & EXIT           esr5
         TITLE REWIND & UNLOAD - DEVICE DEPENDENT FUNCTION CODE 0
TLX1700  EQU   *                                                              dir
         LB    U2,DCB.CUNL(UD)     FETCH COMMAND                              dir
         STB   U2,DCB.CMD(UD)      SAVE COMMAND                               dir
         OCR   U6,U2               ISSUE COMMAND                              dir
         LHI   U0,250              STALL FOR TIME TO BE SURE                  dir
TLX1705  SIS   U0,1                THE COMMAND DOES NOT GET                   dir
         BP    TLX1705             CLEARED BY THE NEXT OPERATION.             dir
         LI    U0,MAGUNLB                                                     dir
         SBT   U0,DCB.FLG1(UD)     SET REWIND/UNLOAG FLAG                     dir
         B     TLX2150             BRANCH TO CHECK REWIND COMPLETE            dir
         TITLE ERASE - DEVICE DEPENDENT FUNCTION CODES 7 AND 10               dir
*                                                                             dir
* SET UP TO PERFORM THE SOFTWARE 'ERASE BUFFER' OPERATION.                    dir
*   THIS WILL REPEATEDLY CALL A ROUTINE (TLX1760) TO PERFORM                  dir
*   THE HARDWARE ERASE GAP OPERATION.                                         dir
*                                                                             dir
* ENTER WITH U2 CONTAINING AN INDEX THAT WILL GIVE THE                        dir
* PROPER 'BYTES PER GAP'                                                      dir
*                                                                             dir
*                                                                             dir
TLX1710  EQU   *                                                              dir
         LIS   U8,0                SET UP REG TO COUNT GAPS TO ERASE          dir
         D     U8,DCB.BPG(UD,U2)   COMPUTE GAPS NEEDED FOR ERASE              dir
         LR    U8,U8                                                          dir
         BZ    TLX1740             NO REMAINDER--BRANCH                       dir
         AIS   U9,1                ADJUST UPWARD                              dir
TLX1740  EQU   *                                                              dir
         STH   U9,DCB.ESCT(UD)     SAVE COUNT                                 dir
         SPACE                                                                dir
TLX1750  EQU   *                                                              dir
         BAL   UB,TLX1760          ERASE ONE GAP'S WORTH                      dir
         LCS   U0,1                                                           dir
         AHM   U0,DCB.ESCT(UD)     DECREMENT COUNT                            dir
         BP    TLX1750             MORE, LOOP                                 dir
         B     TLX2150             ALL DONE.                                  dir
*                                                                             dir
* FOLLOWING ROUTINE EXECUTES ONE ERASE GAP OPERATION                        dir
*                                                                           dir
TLX1760  EQU    *                                                           dir
         ST     UB,CCB.EBO(UC)       SAVE LINK REGISTER                     dir
         LI     U1,MAGWAT2B          CHECK TO SEE IF EOT CAME UP            dir
         TBT    U1,DCB.FLG1(UD)      *                                      dir
         BZ     TLX1770              NO, JUMP TO CONTINUE OPERATION         dir
         LHI    U0,X'9000'           AT EOT GIVE USER X`9000' STATUS        dir
         STH    U0,DCB.STAT(UD)      *                                      dir
         B      TLX2150                                                     dir
TLX1770  EQU    *                                                           dir
         BAL    UB,TLX1940           GO CHECK FOR WRITE-RING                dir
         LB     U2,DCB.CERS(UD)      FETCH COMMAND                          dir
         STB    U2,DCB.CMD(UD)       SAVE COMMAND                           dir
         LA     U1,TLX1780           SET THE INTERRUPT SERVICE ADDR         dir
         STH    U1,CCB.SUBA(UC)      * INTO THE CCB.                        dir
         LA     U1,TLX1810           SET THE EVENT SERVICE ADDRESS          dir
         ST     U1,DCB.ESR(UD)       *INTO THE DCB FOR SQS.                 dir
         LA     U0,1(UC)             MAKE ODD CCB ADDRESS                   dir
         STH    U0,ISPTAB(U6,U6)     SET CONTROLLER ISPT ENTRY              dir
         BAL    U8,TOCHON            PUT ONTO TIMER CHAIN                   dir
         LH     U8,DCB.ILVL(UD)      FETCH INTERRUPT LEVEL                  dir
         SINT   U8,0(U6)             ENTER ISR                              dir
         LI     UE,3                 LEVEL                                  dir
         L      UF,DCB.LEAF(UD)      LEAF ADDRESS                           dir
         BAL    U8,EVREL             RELEASE SELCH                          dir
         B      EVRTE                RETURN FROM ESR                        dir
         SPACE
         PURE                                                               isrg
TLX1780  EQU    *                                                           isrg
         L      E5,CCB.DCB(E4)       A(DCB)                                 isrg
         OC     E2,DCB.CENB(E5)      ENABLE INTERRUPTS                      isrg
         LH     E6,DCB.MAXT(E5)      MAXIMUM TIMEOUT VALUE                  isrg
         STH    E6,DCB.TOUT(E5)      START TIMER                            isrg
         OC     E2,DCB.CMD(E5)       ISSUE COMMAND                          isrg
         LA     E7,TLX1790           SET NEXT ISR ADDRESS                   isrg
         STH    E7,CCB.SUBA(E4)      *                                      isrg
         LPSWR  E0                   EXIT INTERRUPT SERVICE                 isrg
TLX1790  EQU    *                                                           isrg
         L      E5,CCB.DCB(E4)       GET DCB ADDRESS                        isrg
         LH     E6,DCB.CCB(E5)       GET TIME-OUT VALUE                     isrg
         BZ     TLX1795              IF TIMED OUT, EXIT NOW                 isrg
         THI    E3,X'13'             CHECK EOM OR NO MOTION FOR 6250        isrg
         BNZ    TLX1800            YES, GO TOWARDS ESR               isrg
TLX1795  EQU    *                                                    isrg
         LPSWR  E0                 GO WAIT FOR ANOTHER INTERRUPT     isrg
         SPACE
TLX1800  EQU    *                                                    isrh
         STB    E3,DCB.DDPS(E5)    SAVE STATUS                       isrh
         OC     E2,DCB.CDAR(E5)    TURN OFF INTERRUPTS               isrh
         LH     E6,DCB.TOUT(E5)    GET TIME-OUT COUNTER              isrh
         BTFS   2,2                RETURN NOW IF TIMED-OUT           isrh
         LPSWR  E0                 DON'T RE-ADD TO S.Q.              isrh
         LCS    E6,1               OTHERWISE SET MINUS TO            isrh
         STH    E6,DCB.TOUT(E5)    SHOW WE HAVE BEEN HERE.           isrh
         L      E6,DCB.LEAF(E5)    GET LEAF ADDRESS                  isrh
         ATL    E6,SQ              SCHEDULE DRIVER TERMINATION       isrh
         LPSWR  E0                 EXIT AND WAIT FOR TERM.           isrh
         SPACE
         IMPUR                                                       esr6
TLX1810  EQU    *                                                    esr6
         LHL    UC,DCB.CCB(UD)     A(CCB)                            esr6
         LHL    U6,DCB.DN(UD)      DEVICE NUMBER                     esr6
         BAL    U8,TOCHOFF         GET OFF TIMEOUT CHAIN             esr6
         LH     U7,DCB.TOUT(UD)    CHECK FOR TIME-OUT                esr6
         BZ     TLX1820            BRANCH IF WE TIMED OUT            esr6
         LH     UB,DCB.STAT(UD)    GET DEVICE DEPENDENT STATUS       esr6
         THI    UB,DU              DID IT GO OFF LINE?               esr6
         BP     TLX2340            IF SO, EXIT NOW                   esr6
         LIS    U0,0               CLEAR REGISTER                    esr6
         THI    UB,EOM             CHECK FOR END OF TAPE             esr6
         BP     TLX1840            EOT SET--BRANCH                   esr6
         STH    U0,DCB.STAT(UD)    PLACE FOR THE USER                esr6
         L      UB,CCB.EB0(UC)     GET LINK REGISTER                 esr6
         CR     UB,UB              MUST SET CC=0                     esr6
         BR     UB                 AND RETURN TO CALLER.             esr6
TLX1820  EQU    *                                                    esr6
*TIME OUT ON ERASE                                                   esr6
         OC   U6,DCB.CDAR(UD)      DISARM INTRPT                     esr6
         LHI    U8,X'8273'         TIMEOUT STATUS                    esr6
         STH    U8,DCB.STAT(UD)    STORE STATUS                      esr6
         B    TLX2150              CHECK FOR RWND IN PROGRESS & EXIT esr6
*                                                                    esr6
TLX1840  EQU    *                                                    esr6
*SET EOM ON ERASE                                                    esr6
         LI     U1,MAGWAT2B        SET EOT FLAG IN DCB               esr6
         SBT    U1,DCB.FLG1(UD)        *                                 esr6
         LHI    U8,X'9000'             SET STATUS TO X`9000'             esr6
         STH    U8,DCB.STAT(UD)        STORE STATUS                      esr6
         B     TLX2150                 CHECK FOR RWND IN PROGRESS & EXIT esr6
         SPACE
*
         TITLE READ STATUS - DEVICE DEPENDENT FUNCTION CODE 8
TLX1860  EQU    *                                                        dir
         L      U1,CCB.XLT(UC)         GET REQUESTED LENGTH OF XFER      dir
         CLI    U1,8                   BUFFER SIZE LESS THAN 8 ?         dir
         BL     TLX2360                BUFFER IS TOO SMALL               dir
         LIS    U8,3                   SET A SHORT TIME-OUT CONSTANT     dir
         STH    U8,DCB.TOUT(UD)        *                                 dir
         LA     U8,TLX1920             GET ADDRESS OF OUR ESR            dir
         ST     U8,DCB.ESR(UD)         NOW HAVE ESR ADDR SET UP          dir
         LA     U8,TLX1870             GET ADDRESS OF FIRST ISR          dir
         STH    U8,CCB.SUBA(UC)        PUT IN CCB FOR INT SERV           dir
         LA     U8,1(UC)               GET CCB ADDR + 1                  dir
         STH    U8,ISPTAB(U6,U6)       ISPT NOW SET UP                   dir
         BAL    U8,TOCHON              GET ON TIME-OUT CHAIN             dir
         OC     U6,DCB.CENB(UD)        ENABLE INTERRUPTS                 dir
         OC     U6,DCB.DSB0(UD)        ASK FOR FIRST HALFWORD            dir
         B      EVRTE                                                    dir
         PURE                                                            isrh
TLX1870  THI    E3,X'10'               DO WE HAVE NO-MOTION              isrh
         BNP    TLX1910                NO - WAIT SOME MORE               isrh
         L      E5,CCB.DCB(E4)         GET DCB ADDRESS                   isrh
         RH     E2,DCB.STA0(E5)        GET THE HALFWORD                  isrh
         LA     E7,TLX1880             GET NEXT ISR ADDRESS              isrh
         STH    E7,CCB.SUBA(E4)        SET UP CCB                        isrh
         OC     E2,DCB.DSB1(E5)        ASK FOR NEXT HALF WORD            isrh
         LPSWR E0                                                        isrh
TLX1880  THI    E3,X'10'               DO WE HAVE NO-MOTION              isri
         BNP    TLX1910                NO - WAIT SOME MORE               isri
         L      E5,CCB.DCB(E4)         GET DCB ADDRESS                   isri
         RH     E2,DCB.STA1(E5)        GET THE HALFWORD                  isri
         LA     E7,TLX1890             GET NEXT ISR ADDRESS              isri
         STH    E7,CCB.SUBA(E4)        SET UP CCB                        isri
         OC     E2,DCB.DSB2(E5)        ASK FOR NEXT HALF WORD            isri
         LPSWR E0                                                        isri
TLX1890  THI    E3,X'10'               DO WE HAVE NO-MOTION              isrj
         BNP    TLX1910                NO - WAIT SOME MORE               isrj
         L      E5,CCB.DCB(E4)         GET DCB ADDRESS                   isrj
```

```
        RH     E2,DCB.STA2(E5)      GET THE HALFWORD                  isrj
        LA     E7,TLX1900           GET NEXT ISR ADDRESS              isrj
        STH    E7,CCB.SUBA(E4)      SET UP CCB                        isrj
        OC     E2,DCB.DSB3(E5)      ASK FOR NEXT HALF WORD            isrj
        LPSWR  E0                                                     isrj
TLX1900 THI    E3,X'10'             DO WE HAVE NO-MOTION              isrk
        BNP    TLX1910              NO - WAIT SOME MORE               isrk
        L      E5,CCB.DCB(E4)       GET DCB ADDRESS                   isrk
        RH     E2,DCB.STA3(E5)      GET THE HALFWORD                  isrk
        LH     E7,DCB.TOUT(E5)      DID WE TIME OUT?                  isrk
        BNP    TLX1910              NO - NOTHING ELSE TO DO           isrk
        LCS    E7,1                 RESET THE TIME-OUT CONSTANT       isrk
        STH    E7,DCB.TOUT(E5)      SO WE DO NOT CRASH SOMETIMES      isrk
        LA     E7,III               RESET THE ISPT                    isrk
        STH    E7,ISPTAB(E2,E2)     *                                 isrk
        L      E7,DCB.LEAF(E5)      SCHEDULE ESR                      isrk
        ATL    E7,SQ                *                                 isrk
TLX1910 LPSWR  E0                                                     esr7
        IMPUR                                                         esr7
TLX1920 EQU    *                    WILL RE-ENTER HERE                esr7
        LHL    U6,DCB.DN(UD)        RE-SET SOME POINTERS              esr7
        LHL    UC,DCB.CCB(UD)       *                                 esr7
        BAL    U8,TOCHOFF           GET OFF TIME-OUT CHAIN            esr7
        LH     U1,DCB.TOUT(UD)      DID WE TIME OUT?                  esr7
        BZ     TLX1930              YES - BRANCH TO HANDLE TIMEOUT    esr7
        L      U1,DCB.SADR(UD)      GET USER BUFFER ADDRESS           esr7
        LH     U0,DCB.STA0(UD)      MOVE DATA TO USER BUFFER          esr7
        STH    U0,0(U1)                                               esr7
        LH     U0,DCB.STA1(UD)                                        esr7
        STH    U0,2(U1)                                               esr7
        LH     U0,DCB.STA2(UD)                                        esr7
        STH    U0,4(U1)                                               esr7
        LH     U0,DCB.STA3(UD)                                        esr7
        STH    U0,6(U1)                                               esr7
        B      TLX2150                                                esr7
TLX1930 EQU    *                    COME HERE ONLY ON TIME-OUT        esr7
        LHI    U8,X'82DD'           SET A TIME-OUT FLAG               esr7
        STH    U8,DCB.STAT(UD)      STORE STATUS                      esr7
        B    TLX2150                CHECK FOR RWND IN PROGRESS & EXIT esr7
        TITLE SUPPORT SUBROUTINES
* * * * * * * * * * * * * * * * * * **
*    SUPPORT SUBROUTINES
* * * * * * * * * * * * * * * * * * **
```

```
        IMPUR
*
*       BAL   UB,TLX1940
*
* THIS ROUTINE IS CALLED FROM OPERATIONS WHICH PERFORM WRITE
* FUNCTIONS (WRITE, WRITE GAPLESS, WRITE FILEMARK, AND ERASE)
* TO SEE IF THE DRIVE IS WRITE PROTECTED.  IF THE DRIVE IS
* WRITE PROTECTED, THIS ROUTINE SETS AN '8283' STATUS AND EXITS.
* IF THE DRIVE MAY BE PLACED IN A WRITE MODE, THIS ROUTINE
* RETURNS TO THE CALLER.
*
TLX1940 EQU   *
        ST    UB,CCB.EB1(UC)        SAVE RETURN ADDR IN SCRATCH AREA
        LA    U8,TLX1990            ADDRESS OF ESR FOR RING
        ST    U8,DCB.ESR(UD)        SET NEW ESR ENTRY
        LA    U8,TLX1950            SET UP FIRST RING ISR
        STH   U8,CCB.SUBA(UC)       *
        LA    U8,1(UC)              MAKE ADDRESS ODD FOR CCB
        STH   U8,ISPTAB(U6,U6)      * AND SET IN ISPT
        LIS   U8,3                  SET A SHORT TIME-OUT VALUE
        STH   U8,DCB.TOUT(UD)       *
        BAL   U8,TOCHON             GET ON TIME-OUT CHAIN
        OC    U6,DCB.CENB(UD)       ENABLE INTERRUPTS AND
        OC    U6,DCB.DSB0+1(UD)     ISSUE SENSE COMMAND (X'30')
        B     EVRTE                 EXIT FROM ESR
*
* INTERRUPT SERVICE
*
        PURE
TLX1950 EQU   *
        LA    E7,TLX1960            NEXT RING ISR
        STH   E7,CCB.SUBA(E4)       * MUST BE SAVED IN CCB
        RH    E2,CCB.MISC(E4)       READ THE HALFWORD WE NEED
        LPSWR E0                    WAIT FOR BUSY TO DROP TO GIVE
*                                   US THE NEXT INTERRUPT.
*
TLX1960 EQU   *
        THI   E3,X'10'              IS IT A NO-MOTION INTERRUPT
        BNZ   TLX1970               YES, WRAP IT UP
        RHR   E2,E7                 DON'T NEED THIS HALFWORD
        LPSWR E0                    WAIT FOR NEXT INTERRUPT.
TLX1970 EQU   *
        L     E5,CCB.DCB(E4)        GET THE DCB ADDRESS
```

```
         LA     E7,III                SET NULL INTERRUPT
         STH    E7,ISPTAB(E2,E2)      *
         OC     E2,DCB.CDAR(E5)       DISARM INTERRUPTS
         LH     E7,DCB.TOUT(E5)       SEE IF WE HAVE TIMED OUT
         BNP    TLX1980               IF SO, DON'T ADD LEAF TO QUEUE
         LCS    E6,1                  RESET THE TIME-OUT VALUE
         STH    E6,DCB.TOUT(E5)       *
         L      E6,DCB.LEAF(E5)       GET THE LEAF ADDRESS
         ATL    E6,SQ                 AND ADD IT TO SYSTEM QUEUE
TLX1980  LPSWR EO                     AND GET ON WITH THE SHOW
*
         IMPUR
TLX1990  EQU    *
         LHL    UC,DCB.CCB(UD)        MAKE SURE THIS IS STILL O.K.
         LHL    U6,DCB.DN(UD)         *
         LH     U8,DCB.TOUT(UD)       DID WE TIME-OUT?
         BZ     TLX2000               YES-GO SET SPECIAL STATUS.
         BAL    U8,TOCHOFF            GET OFF THE TIME-OUT CHAIN
         L      UB,CCB.EB1(UC)        RESTORE THE RETURN ADDRESS
         LH     U9,CCB.MISC(UC)       GET THE STATUS WE NEED
         THI    U9,X'4000'            WRITE PROTECTED ?
         BZR    UB                    NO, RETURN
         LHI    U8,X'8283'            YES
         STH    U8,DCB.STAT(UD)       STORE STATUS
         B      TLX2150               CHECK FOR REWIND IN PROGRESS & EXIT.
*
* THE FOLLOWING CODE IS EXECUTED ON A TIMEOUT CONDITION WHICH
* MAY OCCUR IF NO-MOTION DOES NOT SET FOLLOWING AN OUTPUT
* COMMAND OF X'30' (SENSE DRIVE STATUS).
*
TLX2000  EQU    *
         LHI    U8,X'82DE'            TIMED OUT - SET SPECIAL STATUS
         STH    U8,DCB.STAT(UD)       STORE STATUS
         B      TLX2150               CHECK FOR REWIND IN PROGRESS & EXIT.
         EJECT
*        BAL    U9,TLX2050
* THE FOLLOWING ROUTINE IS CALLED WHEN IT IS NECESSARY TO DETERMINE
* IF WE ARE AT BOT OR EOT.  THIS IS DONE AT THE START OF THE DRIVER
* IF EOT WAS ENCOUNTERED PREVIOUSLY OR DURING THE CURRENT SENSE
* STATUS OPERATION. THIS ROUTINE IS ALSO CALLED DURING SOME ERROR
* ROUTINES SO THAT THE APPROPRIATE 'EOM' STATUS MAY BE SET FOR THE
* USER.
*
```

```
TLX2050  EQU    *
         LI     U8,MAGREWB           REWIND BIT
         RBT    U8,DCB.FLG1(UD)      RESET IT
         LI     U8,MAGWAT2B          ET ENCOUNTED IN PREVIOUS ACTION ?
         TBT    U8,DCB.FLG1(UD)
         BNZ    TLX2060              YES
         SSR    U6,U0                SENSE STATUS
         THI    U0,EOM               TEST BOT/EOT STATUS
         BZ     TLX2140              BRANCH IF NOT BOT/EOT
TLX2060  EQU    *
         RBT    U8,DCB.FLG1(UD)      RESET MAGWAT2B FLAG
         ST      U9,CCB.EB0(UC)      SAVE RETURN ADDRESS
         LA     U8,TLX2110           ADDRESS OF ESR FOR BOTT
         ST     U8,DCB.ESR(UD)       SET NEW ESR ENTRY
         LA     U8,TLX2070           SET UP FIRST BOTT ISR
         STH    U8,CCB.SUBA(UC)      *
         LA     U8,1(UC)             MAKE ADDRESS ODD FOR CCB
         STH    U8,ISPTAB(U6,U6)     * AND SET IN ISPT
         LIS    U8,3                 SET A SHORT TIME-OUT
         STH    U8,DCB.TOUT(UD)      *
         BAL    U8,TOCHON            GET ON THE TIME-OUT CHAIN
         OC     U6,DCB.CENB(UD)      ENABLE INTERRUPTS AND
         OC     U6,DCB.DSB0+1(UD)    ISSUE SENSE COMMAND (X'30')
         B      EVRTE                EXIT FROM ESR
*
* INTERRUPT SERVICE
*
         PURE
TLX2070  EQU    *
         LA     E7,TLX2080           NEXT BOTT ISR
         STH    E7,CCB.SUBA(E4)      * MUST BE SAVED IN CCB
         RH     E2,CCB.MISC(E4)      READ THE HALFWORD WE NEED
         LPSWR E0                    WAIT FOR BUSY TO DROP TO GIVE
*                                    US THE NEXT INTERRUPT.
*
TLX2080  EQU    *
         THI    E3,X'10'             IS IT A NO-MOTION INTERRUPT
         BNZ    TLX2090              YES, WRAP IT UP
         RHR    E2,E7                DON'T NEED THIS HALFWORD
         LPSWR E0                    WAIT FOR NEXT INTERRUPT.
TLX2090  EQU    *
         L      E5,CCB.DCB(E4)       GET THE DCB ADDRESS
         LA     E7,III               SET NULL INTERRUPT
         STH    E7,ISPTAB(E2,E2)      *
         OC     E2,DCB.CDAR(E5)       DISARM INTERRUPTS
         LH     E7,DCB.TOUT(E5)       SEE IF WE HAVE TIMED OUT
         BNP    TLX2100               IF SO, DON'T ADD LEAF TO QUEUE
         LCS    E7,1                  SHOW SUCCESSFUL COMPLETION
         STH    E7,DCB.TOUT(E5)       *
         L      E6,DCB.LEAF(E5)       GET THE LEAF ADDRESS
         ATL    E6,SQ                 AND ADD IT TO SYSTEM QUEUE
TLX2100  LPSWR E0                     AND GET ON WITH THE SHOW
*
         IMPUR
TLX2110  EQU    *
         LHL    UC,DCB.CCB(UD)        MAKE SURE THIS IS STILL O.K.
         LHL    U6,DCB.DN(UD)         *
         BAL    U8,TOCHOFF            GET OFF TIME-OUT CHAIN
         LH     U8,DCB.TOUT(UD)       DID WE TIME-OUT?
         BZ     TLX2145               YES- GO TO ERROR EXIT
         L      U9,CCB.EBO(UC)        RESTORE LINK REG(JUST IN CASE)
         LH     U0,CCB.MISC(UC)       GET STATUS IN U0
         THI    U0,X'8000'            BOT STATUS ?
         BZ     TLX2130               NO ,MUST BE EOT ,BRANCH
         LI     U8,MAGBOTB            BOT BIT
         SBT    U8,DCB.FLG1(UD)       SET BOT FLAG
         BR     U9                    RETURN
TLX2130  EQU    *                     CHECK EOT CONDITION
         LI     U8,MAGEOTB
         SBT    U8,DCB.FLG1(UD)       SET EOT FLAG
         BR     U9                    RETURN
TLX2140  EQU    *
         LI     U8,MAGEOTB
         RBT    U8,DCB.FLG1(UD)       RESET THE EOT FLAG
         LI     U8,MAGBOTB            BOT BIT
         RBT    U8,DCB.FLG1(UD)       RESET BOT FLAG
         BR     U9                    RETURN
*
* COME HERE ONLY ON TIMEOUT
*
TLX2145  EQU    *
         LHI    U8,X'82DF'            YES - SET SPECIAL FLAG
         STH    U8,DCB.STAT(UD)       STORE STATUS
         B      TLX2150               CHECK FOR REWIND IN PROGRESS & EXIT.
         TITLE CONCURRENT REWIND LOGIC
*
```

```
* Problem:
*
*    If a tape drive is rewinding, no I/O can be done on a
*    second drive on the same controller, if an I/O request
*    has been issued to the rewinding drive.
*
* Solution:
*
*
*
* Upon any I/O complete, check to see if any other drive on the
* same controller is rewinding.  If not, proceed normally (go to
* IODONE). If so, branch to check if the rewind is complete.
*
TLX2150  EQU  *
         LHI  U0,X'7FFF'          RESET TIME-OUT CONSTANT
         STH  U0,DCB.TOUT(UD)     (THIS IS SORT OF STUPID)
         L    U0,DCB.REWF(UD)     GET COMMON FLAG (FOR REWIND)
         BZ   IODONE              GO TO O.S. ROUTINE FOR I/O DONE.
*
*
         LIS  U9,8                INDEX INTO TABLE
TLX2160 EQU  *
*
         L    U1,DCB.RTBL(U9,UD)  GET FIRST ENTRY IN TABLE
         BZ   TLX2180             NONE, HERE -- TRY NEXT
         LH   U2,DCB.DN(U1)       GET ADDRESS OF REWINDING DRIVE
         SSR  U2,U3               WHAT IS THE STATUS?
         THI  U3,X'D1'            DID SOMETHING HAPPEN?
         BZ   TLX2180             STILL REWINDING--TRY NEXT
         LHI  U6,X'2050'          GO NONINTERRUPTABLE (CAN'T BE HELPED)
         EPSR U7,U6
         LH   U3,DCB.TOUT(U1)     GET TIMEOUT CONSTANT
         BNP  TLX2170             ALREADY ON QUEUE
         LCS  U3,1
         STH  U3,DCB.TOUT(U1)     SET TO INDICATE ON QUEUE
         L    U3,DCB.LEAF(U1)     GET LEAF ADDRESS
         ATL  U3,SQ               AND ADD TO SYSTEM QUEUE
TLX2170 EQU   *

         EPSR U6,U7               BECOME INTERRUPTABLE AGAIN
         LIS  U3,0                ZERO ENTRY IN TABLE
         ST   U3,DCB.RTBL(U9,UD)  *
```

```
TLX2180  EQU  *
         SIS  U9,4                   DECREMENT INDEX
         BNM  TLX2160
         L    U0,DCB.RTBL+0(UD)      GET FIRST ENTRY IN THE TABLE
         O    U0,DCB.RTBL+4(UD)      * ALSO INCLUDE SECOND
         O    U0,DCB.RTBL+8(UD)      * AND THE THIRD
         ST   U0,DCB.REWF(UD)        FIX THE FLAG TO SHOW WHETHER
*                                    OR NOT WE STILL HAVE WORK TO DO.
         B    IODONE                 * GET OUT OF HERE
*
* START ESR
*
TLX2190  EQU  *
         LH   U6,DCB.DN(UD)          GET ADDRESS OF REWINDING DRIVE
         SSR  U6,U3                  WHAT IS THE STATUS?
         THI  U3,X'D1'               DID ANYTHING HAPPEN?
         BNZ  TLX2200                IF NOT, CONTINUE PROCESSING
*
* KEEP COUNT OF HOW MANY TIMES WE HAVE DONE THIS. IF WE
* DO THIS 45 TIMES (90 SECONDS) WE WILL ASSUME THAT THE
* REWIND GOT KILLED BY THE OPERATOR.
*
         L    U2,DCB.RG(UD)
         AIS  U2,1
         ST   U2,DCB.RG(UD)
         CHI  U2,45
         BNM  TLX2195                GO FORCE TIME-OUT
         LIS  U2,2                   SET 2 SECOND TIMEOUT
         STH  U2,DCB.TOUT(UD)        *
         L    UF,DCB.LEAF(UD)        GET LEAF ADDR FOR DISCONNECT
         LIS  UE,3                   RELEASE SELCH AGAIN
         BAL  U8,EVREL               *
         LIS  UE,2                   RELEASE CONTROLLER AGAIN
         BAL  U8,EVREL               * THIS IS STUPID.
         B    EVRTE                  GO BACK TO O.S.
*
* COME HERE IF THE REWIND TIMES OUT.
*
TLX2195  EQU  *
         LHI  U8,X'8277'             SET INDICATOR
         STH  U8,DCB.STAT(UD)        STORE STATUS
         B    TLX2150                CHECK FOR REWIND IN PROGRESS & EXIT.
*
```

```
TLX2200  EQU   *
         LA    U8,III
         STH   U8,ISPTAB(U6,U6)        CLEAR ISPT
         OC    U6,DCB.CDAR(UD)         DISARM INTERRUPTS FROM DRIVE
         LHL   UC,DCB.CCB(UD)          RELOAD CCB ADDRESS
         LHI   U7,X'7FFF'              NO TIME-OUT
         STH   U7,DCB.TOUT(UD)         *
         BAL   U8,TOCHOFF              *
         LI    U8,MAGREWB              RESET THE REWINDING FLAG
         RBT   U8,DCB.FLG1(UD)         *
* NOW DO REWIND BOOKKEEPING
         LIS    U3,0                   SET UP ZERO FOR STORAGE IN TABLE
         LIS    U7,8                   SET UP COUNTER
TLX2210  EQU    *
         L      U2,DCB.XDRV(U7,UD)     GET DCB ADDRESS OF ANOTHER DRIVE
         BZ     TLX2240                TRY ANOTHER SLOT
         LIS    U8,8                   SET UP INNER INDEX
TLX2220  EQU    *
         C      UD,DCB.RTBL(U8,U2)     IS THIS OUR ADDRESS?
         BNE    TLX2230                NO, IGNORE IT
         ST     U3,DCB.RTBL(U8,U2)     ZERO THE ENTRY
TLX2230  EQU    *
         SIS    U8,4                   DECREMENT INNER INDEX
         BNM    TLX2220                LOOP AGAIN
         L     U3,DCB.RTBL+0(U2)       GET DCB ADDRESS
         O     U3,DCB.RTBL+4(U2)       *
         O     U3,DCB.RTBL+8(U2)       *
         ST    U3,DCB.REWF(U2)         *
TLX2240  EQU   *
         SIS   U7,4                    DECREMENT OUTER INDEX
         BNM   TLX2210                 LOOP AGAIN
         B     TLX0018                 RETURN TO MAINLINE CODE
*
*  COME HERE IF EX OR DU IS SET ON ENTRY TO THE DRIVER.
*  STATUS IS IN REGISTER U7.
*
TLX2250 EQU    *
        THI    U7,X'F0'                ERR, TERR, EOM, NMTN SET?
        BNZ    TLX2255                 IF SO, BRANCH TO DO NEXT TEST
        LI     U0,MAGREWB              GET REWIND FLAG BIT
        TBT    U0,DCB.FLG1(UD)         IS IT SET?
        BNZ    TLX0015                 IF REWIND FLAG IS SET, GO BACK
        LHI    U8,X'A000'              OTHERWISE, REPORT DEV UNAVAIL.
        STH    U8,DCB.STAT(UD)        STORE STATUS
        B      TLX2150                CHECK FOR REWIND IN PROGRESS & EXIT.
TLX2255 EQU    *
        THI    U7,DU                  IS DU STATUS SET?
        BZ     TLX0015                IF NOT, CONTINUE PROCESSING
        LHI    U8,X'A000'             IF DU SET, GIVE ERROR CODE
        STH    U8,DCB.STAT(U D)       STORE STATUS
        B      TLX2150                CHECK FOR REWIND IN PROGRESS & EXIT.
*
* COME HERE IF WE HAVE ISSUED AN I/O REQUEST TO A DRIVE
* THAT WE BELIEVE TO BE CURRENTLY REWINDING.
*
TLX2260 EQU    *
*
* IF THE I/O REQUEST IS A BSR OR BFM, TERMINATE THE I/O
* REQUEST WITH A STATUS OF '9000'.
* IF THE I/O REQUEST IS A REWIND, SET THE STATUS TO ZERO AND EXIT
*
         LB     U3,DCB.FC(UD)         GET THE SVC 1 FUNCTION CODE
         CHI    U3,X'82'              BACK FILE MARK?
         BE     TLX2345               IF SO,EXIT
         CHI    U3,X'A0'              BACK SPACE RECORD?
         BE     TLX2345               IF SO,EXIT
         CHI    U3,X'C0'              IS IT REWIND?
         BE     IODONE
*
* SET A COUNTER TO TIME-OUT THE REWIND.
*
         LIS    U3,0
         ST     U3,DCB.RG(UD)          (SEE ALSO TLX2190)
         LIS    U3,8                  SET OUTER INDEX
TLX2270  EQU    *
         L      U8,DCB.XDRV(U3,UD)    GET DCB ADDRESS
         BZ     TLX2300               IGNORE IF NOT PRESENT
         LIS    U9,0                  SET INNER INDEX
TLX2280  EQU    *
         L      U7,DCB.RTBL(U9,U8)    GET ENTRY IN OTHER DCB TABLE
         BNZ    TLX2290               THIS SLOT ALREADY USED
         ST     UD,DCB.RTBL(U9,U8)    SAVE OUR ADDRESS HERE
         ST     UD,DCB.REWF(U8)       SET REWIND FLAG
         B      TLX2300               DONE WITH THIS DCB
TLX2290  EQU    *
         AIS    U9,4                  INCR INNER INDEX
         CHI   U9,12              DONE YET?
         BM    TLX2280            LOOP UNTIL DONE
TLX2300  EQU   *
         SIS   U3,4               DECR OUTER INDEX
         BNM   TLX2270            LOOP UNTIL DONE
         LA    UE,TLX2190         ADDRESS OF ESR
         ST    UE,DCB.ESR(UD)     SET UP FOR NEXT ESR SCHEDULING
         LIS   UE,3               RELEASE SELCH
         L     UF,DCB.LEAF(UD)    GET LEAF ADDRESS
         BAL   U8,EVREL           *
         LIS   UE,2               RELEASE CONTROLLER
         BAL   U8,EVREL           *
*
         LA    U7,TLX2310         ISR ADDRESS
         STH   U7,CCB.SUBA(UC)    * AND SAVE IT
         LA    U7,1(UC)           ODD CCB ADDRESS
         STH   U7,ISPTAB(U6,U6)   *
         BAL   U8,TOCHON          GET ON TIMER CHAIN
         LIS   U7,1
         STH   U7,DCB.TOUT(UD)
         OC    U6,DCB.CENB(UD)    ENABLE DRIVE INTERRUPTS
         B     EVRTE
         PURE
TLX2310  EQU   *
         THI   E3,NMTN!DU         NO-MOTION OR DU?
         BZ    TLX2330            YES, SCHEDULE ESR
         L     E5,CCB.DCB(E4)     GET DCB ADDRESS
         L     E6,DCB.LEAF(E5)    AND LEAF ADDRESS
         LH    E7,DCB.TOUT(E5)    HAVE WE TIMED OUT?
         BNP   TLX2330            GET OUT OF HERE IF SO
         LCS   E7,1               SET TIME OUT TO - 1
         STH   E7,DCB.TOUT(E5)    *
         ATL   E6,SQ              SCHEDULE ESR
TLX2330  EQU   *
         LPSWR E0                 GET OUT
         TITLE ERROR ROUTINES
* ERROR HANDLING SUBROUTINES
*
         IMPUR
TLX2340  EQU   *
         LI    U8,X'A000'         DU STATUS
         STH   U8,DCB.STAT(UD)    STORE STATUS
         B     TLX2150            CHECK FOR REWIND IN PROGRESS & EXIT.
```

```
TLX2345  EQU
         LHI   U8,X'9000'          EOM STATUS
         STH   U8,DCB.STAT(UD)     STORE STATUS
         B     TLX2150             CHECK FOR REWIND IN PROGRESS & EXIT.
TLX2350  EQU   *
         LI    U8,X'8400'          UNRECOVERABLE ERROR
         STH   U8,DCB.STAT(UD)     STORE STATUS
         B     TLX2150             CHECK FOR REWIND IN PROGRESS & EXIT.
TLX2360 EQU    *
         LI    U8,X'C000'          ILLEGAL FUNCTION STATUS
         STH   U8,DCB.STAT(UD)     STORE STATUS
         B     TLX2150             CHECK FOR REWIND IN PROGRESS & EXIT.
*
TLX2370  EQU   *
         BAL   U7,TLX2400          IS IT THE END OF TAPE CASE ?
         OHI   U8,X'84FB'          SET UP STATUS CODE
         STH   U8,DCB.STAT(UD)     STORE STATUS
         B    TLX2150              CHECK FOR REWIND IN PROGRESS & EXIT.
*
TLX2380  EQU   *
         BAL   U7,TLX2400          IS IT THE END OF TAPE CASE ?
         STH   U8,DCB.STAT(UD)     STORE STATUS
         B    TLX2150              CHECK FOR REWIND IN PROGRESS & EXIT.
*
TLX2390  EQU   *
         BAL   U7,TLX2400          IS IT AT THE END OF TAPE ?
         OHI   U8,X'82FA'          SET UP RECOVERABLE STATUS
         STH   U8,DCB.STAT(UD)     STORE STATUS
         B    TLX2150              CHECK FOR REWIND IN PROGRESS & EXIT.
*
*        BAL   U7,TLX2400
*
TLX2400  EQU   *
         BAL   U9,TLX2050
         LIS   U8,0                CLEAR U8
         LI    U9,MAGEOTB          EOT FLAG SET ?
         TBT   U9,DCB.FLG1(UD)
         BZR   U7                  NO, RETURN
         RBT   U9,DCB.FLG1(UD)     RESET EOT FLAG
         LHI   U8,X'9000'          REPORT EOT STATUS
         BR    U7                  RETURN
         TITLE TELEX I/O HANDLERS
*
```

```
* BUILD IOH LIST FOR COMMON SVC1 EXECUTORS
*
         ALIGN 4
         IOH   NAME=IOHTELX,                                                   1
               READ=TLX2450,                                                   1
               WRITE=TLX2460,                                                  1
               WAIT=SVC1WAIT,                                                  1
               HALT=SVC1HALT,                                                  1
               TEST=SVC1TEST,                                                  1
               SET=SVC1NOOP,                                                   1
               REW=SVC1REW,                                                    1
               BSR=SVC1BSR,                                                    1
               FSR=SVC1FSR,                                                    1
               WFM=SVC1WFM,                                                    1
               FFM=SVC1FFM,                                                    1
               BFM=SVC1BFM,                                                    1
               INIT=TLX2490,                                                   1
               DDF=TLX2420
TLX2420  EQU   *
         ST    EA,TLX2470          SAVE IOB ADDRESS
         ST    EB,TLX2480          SAVE DCB ADDRESS
         LB    EA,IOB.SV1X+3(EA)   GET EXTENDED SVC1 WORD
         CHI   EA,8                ADDRESS CHECKING REQUIRED ?
         BE    TLX2430             YES, BRANCH
         CHI   EA,10               CHECK BUFFER ON ERASE BUFFER
         BE    TLX2430             * FUNCTION.
         L     EA,TLX2470          RESTORE IOB ADDRESS
         B     SVC1DDF             GO TO ROUTINE TO ENTER DRIVER
TLX2430  EQU   *                   .
         L     EA,SVC1.SAD(ED)     GET START AND END ADDRESSES
         THI   EA,1
         BNZ   TLX2440
         L     EB,SVC1.EAD(ED)     FOR ADDRESS CHECKING ROUTINE
         BAL   E8,ADCHKNS          CHECK ADDRESSES
         BC    TLX2440             ADDRESS ERROR, EXIT
         LR    EC,EA               SAVE RELOCATED SADR
         L     EA,TLX2470          RESTORE IOB ADDRESS
         ST    EC,IOB.SADR(EA)     STORE RELOCATED START AND
         ST    EB,IOB.EADR(EA)     END ADDRESSES
         L     EB,TLX2480          RESTORE DCB ADDRESS
         B     SVC1BFM             GO TO ROUTINE TO ENTER DRIVER
         SPACE
TLX2440  EQU   *
```

```
         L      EA,TLX2470          RESTORE IOB ADDRESS
         RELIOB REMW=NO
         L      EA,TCB.SAVE(E9)     RESTORE REGISTERS
         LM     EE,CTX.PSW(EA)
         B      MEMFAULT            GO TO ERROR ROUTINE
TLX2450  EQU    *
         L      EC,IOB.SADR(EA)     FETCH START ADDRESS
         SRLS   EC,1                EVEN ?
         BNC    SVC1READ            YES, CONTINUE
         B      TLX2440             NO, ERROR EXIT
TLX2460  EQU    *
         L      EC,IOB.SADR(EA)     FETCH START ADDRESS
         SRLS   EC,1                EVEN ?
         BNC    SVC1WRIT            YES, CONTINUE
         B      TLX2440             NO, ERROR EXIT
         SPACE
         ALIGN 4
TLX2470  DCF    0                   IOB ADDRESS SAVE AREA
TLX2480  DCF    0                   DCB ADDRESS SAVE AREA
         ALIGN 4


TLX2490  EQU    *
*
*   UB CONTAINS THE DCB ADDRESS UPON ENTRY                            init
*                                                                     init
         L      UD,DCB.LEAF(UB)     GET THE LEAF ADDRESS              init
         L      UD,EVN.CORD(UD)     GET UPPER NODE (COMMON POINT      init
*                                   FOR DRIVES ON THE SAME CONTROLLER. init
         LIS    UC,0                INDEX INTO TABLE                  init
         LA     UE,DMT              GET ADDRESS OF DMT                init
TLX2500  EQU    *                                                     init
         L      U9,4(UE)            GET FIRST DCB ADDRESS             init
         BZR    U8                  RETURN TO SYSINIT.                init
         BM     TLX2510             IGNORE PSEUDO DEVICES             init
         CR     UB,U9               IGNORE OURSELF                    init
         BE     TLX2510             *                                 init
         L      UA,DCB.LEAF(U9)     AND OBTAIN LEAF ADDRESS           init
         C      UD,EVN.CORD(UA)     AND THEN CORD.                    init
         BNE    TLX2510             GET NEXT ENTRY                    init
         ST     U9,DCB.XDRV(UC,UB)  SAVE THIS DRIVE'S ADDRESS         init
         AIS    UC,4                                                  init
TLX2510  EQU    *                                                     init
```

```
         AIS    UE,8                GET NEXT DMT ENTRY                 init
         B      TLX2500                                                init
         TITLE MAG DENSITY SELECTION
TLX2520  EQU    *
         LA     U3,TLX2600
         LB     U1,SVC7.OPT+1(U5)
         NHI    U1,7                DENSITY SELECTED?
         BZ     TLX2530             NO
         LA     U3,TLX2600
         CLHI   U1,S7.800           800 BPI SELECTED?
         BNE    TLX2540             NO
         LB     U3,0(U3)
         B      TLX2580
         SPACE 1
TLX2530  EQU    *
         LB     U3,3(U3)
         B      TLX2580
         SPACE 1
TLX2540  EQU    *
         CLHI   U1,S7.1600          1600 BPI SELECTED?
         BNE    TLX2550             NO
         LB     U3,1(U3)
         B      TLX2580
         SPACE 1
TLX2550  EQU    *
         CLHI   U1,S7.6250          6250 BPI SELECTED?
         BE     TLX2570             YEP
         SPACE 1
         LR     U8,U8               SET CC=NONO
         BR     U8                  RETURN
         SPACE 1
TLX2570  EQU    *
         LB     U3,2(U3)
         SPACE 1
TLX2580  EQU    *
         LB     U1,DCB.DENS(U7)
         NHI    U1,X'CF'
         OR     U1,U3               SET NEW DENSITY
         STB    U1,DCB.DENS(U7)
         SPACE 1
         LB     U1,DCB.CENB(U7)
         NHI    U1,X'CF'
         OR     U1,U3               SET NEW DENSITY
         STB    U1,DCB.CENB(U7)
         SPACE 1
         LB     U1,DCB.CDAB(U7)
         NHI    U1,X'CF'
         OR     U1,U3                SET NEW DENSITY
         STB    U1,DCB.CDAB(U7)
         SPACE 1
         LB     U1,DCB.CDAR(U7)
         NHI    U1,X'CF'
         OR     U1,U3                SET NEW DENSITY
         STB    U1,DCB.CDAR(U7)
         SPACE 1
         LB     U1,DCB.CCLC(U7)
         NHI    U1,X'CF'
         OR     U1,U3                SET NEW DENSITY
         STB    U1,DCB.CCLC(U7)
         SPACE 1
         LB     U1,DCB.CGPL(U7)
         NHI    U1,X'CF'
         OR     U1,U3                SET NEW DENSITY
         STB    U1,DCB.CGPL(U7)
         SPACE 1
*        OC     U1,DCB.DSB3(U7)      READ STATUS FOR DENSITY
*        RH     U1,DCB.STA3(U7)
*
* STAT CHECKS OUT:
*
*    OK  - RETURN CC=0
*    NG  - RETURN CC=NONO
*
         SPACE 1
         XAR    U1,U1                SET CC=0
         BR     U8                   GO HOME FOR NOW !
         SPACE
         ALIGN ADC
TLX2600  EQU    *
         DB     X'20'                800 BPI - TELEX & STC
         DB     X'00'                1600 BPI
         DB     X'10'                6250 BPI
         DB     X'30'                HARDWARE SELECT (TELEX ONLY)
         ALIGN ADC
         BR     UB                   EXIT
SIZE     EQU    IMPTOP-TLX0010+7/8*8+PURETOP-TLX0360+7/8*8
```

```
         END
**DCB070
         NLSTC
         MLIBS 8,9,10
SGN.MAGS EQU    1
SGN.EOV  EQU    1                    USED BY $MTP MACRO
         NLIST
         DPROG DCOD=070
         LIST
         $MTP
*
*
* BUILD DCB FOR TELEX/6250/HALFWORD MODE
*
         DCB    DCOD=070,INIT=INITTELX,TERM=TERMTELX,                  1
                ATRB=7BFF,SIZE=MTP,CLASS=IOCLS1,                       1
                FUNC=CMDTELX,FLGS=DFLG.UCM+DFLG.LNM+DFLG.MGM,IOH=IOHTELX
* DEVICE DEPENDENT
         EXTRN Z(CDN2)
         ORG    DCB070+DCB.CCB
         DC     Z(CCB070)            CCB
         ORG    DCB070+DCB.SDN
         DC     Z(CDN2)              SELCH
         ORG    DCB070+DCB.SADR
         EXTRN  INITSUBS
         DAC    INITSUBS             SUBROUTINES
         EXTRN  INITMAGS
         DAC    INITMAGS
         ORG    DCB070+DCB.DENS
         DB     X'38'                Allow drive to select density.
         ORG    DCB070+DCB.CENB
         DB     X'78'                ENABLE INTERRUPTS
         ORG    DCB070+DCB.CDAB
         DB     X'B8'                DISABLE INTERRUPTS
         ORG    DCB070+DCB.CDAR
         DB     X'F8'                DISARM INTERRUPTS
         ORG    DCB070+DCB.CCLC
         DB     X'39'                CLEAR CONTROLLER
         ORG    DCB070+DCB.CCLD
         DB     X'10'                CLEAR DRIVE
         ORG    DCB070+DCB.CRDF
         DB     X'40'                READ FORWARD
         ORG    DCB070+DCB.CRDB
         DB     X'50'                  READ BACKWARD
         ORG    DCB070+DCB.CWRT
         DB     X'60'                  WRITE A BLOCK
         ORG    DCB070+DCB.CREW
         DB     X'E0'                  REWIND
         ORG    DCB070+DCB.CBSR
         DB     X'90'                  BSR
         ORG    DCB070+DCB.CFSR
         DB     X'B0'                  FSR
         ORG    DCB070+DCB.CWFM
         DB     X'C0'                  WFM
         ORG    DCB070+DCB.CFFM
         DB     X'A0'                  FFM
         ORG    DCB070+DCB.CBFM
         DB     X'80'                  BFM
         ORG    DCB070+DCB.CERS
         DB     X'D0'                  ERASE GAP
         ORG    DCB070+DCB.CLWR
         DB     X'70'                  LOOP WRITE TO READ
         ORG    DCB070+DCB.CUNL        TLX1700 REWIND
         DB     X'F0'
         ORG    DCB070+DCB.CGPL        GAPLESS INTERRUPT ENABLE
         DB     X'7A'                  .
         ORG    DCB070+DCB.CNOP
         DB     0                      NO-OPERATION
         ORG    DCB070+DCB.CDDF
         DB     0                      DEVICE-DEPENDENT
         ORG    DCB070+DCB.CNFD
         DB     0                      COMMAND NOT FOUND
         ORG    DCB070+DCB.DSB0
         DCX    0030,0131,0232,0333    NOP AND TAPE UNIT SENSE
         ORG    DCB070+DCB.BPG         BYTES PER GAP
         DC     5600,21875,2800        PE, GCR, NRZI
         ORG    DCB070+DCB.SPED        DRIVE SPEED (INCHES/SECOND)
         DC     H'125'                 .
         ORG    DCB070+DCB.RATE        DATA TRANSFER RATE (BYTES/SECOND)
         DC     *-*                    .
         ORG    DCB070+DCB.XRT         DATA TRANSFER RATES
         DC     200000,781250,100000
* SET UP A TIMEOUT CONSTANT FOR USE IN SOME OPERATIONS
         ORG    DCB070+DCB.MAXT
         DC     X'20'
* READ AND WRITE RETRY COUNTS
```

```
         ORG    DCB070+DCB.RRTY
         DC     X'07'                .
         ORG    DCB070+DCB.WRY1
         DC     X'07'                .
         ORG    DCB070+DCB.WRY2
         DC     X'07'                .
         ORG    DCB070+MTP
*
* BUILD CCB FOR 6250 BPI MAG TAPE
         CCB    DCOD=070
         END
         BEND
```

SYSGEN32 DCB macro for Telex tape driver:

```
         MACRO
         DCB249     %DCOD=,%DN=,%CLAS=,%ILVL=,%NAME=,                 1
                %SLCH=,%CNTR=,%SHCCB=,%EOV=0
         GBLB   %DCB$,%PDCB,%DDCB,%EVN,%CCB,%DFLG,%SDCB
         GBLB   %IDCB,%ODCB,%S125DCB,%ICCB,%BDCB
         GBLB   %ADCB,%TCB,%IOB,%IOB$,%CRTDCB,%LPDCB
         GBLB   %MMDDX,%DDEX,%VFDCB,%CRPDCB,%MGDCBX,%HFWDST
         GBLB   %PSDCBX,%CRDP,%AOBDCB,%BIOCDCB,%LPTDCB
         GBLB   %MTPT
         GBLC   %IDVAL
         BGBLA  %ID249
         LCLA   %CCBFL
         LCLA   %CLASN
         LCLC   %RXLT,%RQU
         LCLC   %CORDNM,%PTRPAS
         LCLC   %OFFS
         LCLA   %RDN
         LCLC   %MDN,%MCNT,%MSLCH
         LCLA   %TRCNT,%UPTR
         LCLB   %FOUND,%DA
         BGBLA  %FIRST
%RQU     SETC   'COMQ'               DEFAULT DEVICE QHANDLER
%MDN     SETC   '%DN'                DEVICE ADDRESS
%MCNT    SETC   '%CNTR'              CONTROLLER
%MSLCH   SETC   '%SLCH'              SELCH
%CCBFL   SETA   0
         AIF    (T'%CLAS EQ 'U')&CLSNTD
%CLASN   SETA   %CLAS*12             IOCLASS*12
&CLSNTD  ANOP
```

```
         CONVNUM VAL=%ID249         CONVERT CURRENT ID TO HEX.
         USERINIT
SGN.MAGS EQU    0
SGN.TELX EQU    1
SGN.EOV  EQU    1
         $DCB$
         DCBI   DCOD=249,SIZE=MTPT,INIT=INITTELX,IOC=1,                 1
                TERM=TERMTELX,FLGS=DFLG.UCM+DFLG.LNM+DFLG.MGM,          2
                FUNC=CMDTELX,ID=%IDVAL,ATRB=7BFF,COPY=$MTPT,            3
                SADR=INITSUBS,IOH=IOHTELX
         ORG    DCB%DCOD%IDVAL+DCB.SADR+4
         EXTRN  INITMAGS
         DAC    INITMAGS
         TMTPI  DCOD=249,ID=%IDVAL,DENS=38,CENB=78,CDAB=B8,CDAR=F8,     1
                CCLC=39,CCLD=10,CRDB=50,CWRT=60,CREW=E0,CBSR=90,        2
                CFSR=B0,CWFM=C0,CFFM=A0,CBFM=80,CERS=D0,CCON=7E,SPED=125
         ORG    DCB%DCOD%IDVAL+DCB.DSBO
         DCX    0030,0131,0232,0333 NOP AND TAPE UNIT SENSE
         ORG    DCB%DCOD%IDVAL+DCB.BPG  BYTES PER GAP
         DC     5600,21875,2800         NRZI, PE, GCR
         ORG    DCB%DCOD%IDVAL+DCB.XRT  MAX BUFFER SIZE FOR ERASE TAPE
         DC     200000,781250,1000000   NRZI, GCR, PE
         ORG    DCB%DCOD%IDVAL+DCB.CUNL
         DB     X'F0'
         ORG    DCB%DCOD%IDVAL+DCB.CGPL
         DB     X'7A'
         CCBI   DCOD=249,ID=%IDVAL
CCB%NAME EQU    CCB%DCOD%IDVAL
%ID249  SETA  %ID249+1
&DCBOPT  ANOP
DCB%DCOD%IDVAL PROG   USER DCB
%OFFS    SETC   '%DCOD':'%IDVAL'     ESTABLISH PROPER OFFSET
DCB.%NAME EQU   DCB%OFFS
         ENTRY DCB.%NAME
         ORG    DCB%OFFS+DCB.DN      DEVICE ADDRESS
         DC     H'%DN'
         ORG    DCB%OFFS+DCB.LEAF    LEAF POINTER
         AIF    (T'%SHCCB' EQ 'U')&NSLEAF   B IF NOT SHARED
         DAC    LF%SHCCB             USE SHARED DEVICE LEAF
         EXTRN  LF%SHCCB
         AGO    &NRMLFX
&NSLEAF  ANOP
         DAC    LF%OFFS              GENERATE STANDARD LEAF NAME
         EXTRN LF%OFFS
&NRMLFX  ANOP
&NOLEAF  ANOP
         AIF    (T'%CLAS EQ 'U')&NOCLAS
         ORG    DCB%OFFS+DCB.CLAS    IO CLASS
         DC     H'%CLASN'            IOCLASS*12
&NOCLAS  ANOP
         AIF    (T'%ILVL EQ 'U')&NOILVL
         ORG    DCB%OFFS+DCB.ILVL    LEVEL
         DC     H'%ILVL'
&NOILVL  ANOP
         ORG    DCB%OFFS+DCB.DMT
         DC     DMT.%NAME            A(DMT ENTRY)
         EXTRN DMT.%NAME
&SKP     ANOP
         AIF    (T'%SLCH EQ 'U')&NOSLCH
         ORG    DCB%OFFS+DCB.SDN     SELCH
         DCX    %SLCH
&NOSLCH  ANOP
         AIF    (T'%CNTR EQ 'U')&NOCNTR
         ORG    DCB%OFFS+DCB.CDN     CONTROLLER
         DCX    %CNTR
&NOCNTR  ANOP
         AIF    ('%RQU' EQ '')&NOQU
         ORG    DCB%OFFS+DCB.Q
         DAC    %RQU
         EXTRN %RQU
&NOQU    ANOP
         ORG    DCB%OFFS+DCB.MAXT
         DC     X'20'
         ORG    DCB%OFFS+DCB.RRTY
         DC     X'07'
         ORG    DCB%OFFS+DCB.WRY1
         DC     X'07'
         ORG    DCB%OFFS+DCB.WRY2
         DC     X'07'
         ORG    $ST%OFFS            ORG TO END OF DCB
         ASIS
         END
%RDN     SETA   %DN+1
         MEND
```

CHAPTER 5

ADVANCED DRIVER CONCEPTS

CHAPTER 5

ADVANCED DRIVER CONCEPTS

5.1 INTRODUCTION

As drivers evolve to handle more complicated situations, comprehension of more advanced driver concepts becomes necessary. Advanced concepts include the use of a translation table, which is a necessary driver component when the character set has to be translated from ASCII to EBCDIC or vice versa. The translation table is also used for recognizing special characters of communications protocols. Another advanced driver concept is that of the nonphysical device. Sometimes drivers are designed for a device that does not actually exist in order to satisfy particular design requirements without having to modify the operating system. The last advanced driver concept to be discussed in this chapter is the supervisor call 6 (SVC6) and trap generating device driver, a driver that is desirable in certain system configurations.

5.2 THE TRANSLATION TABLE

Some devices use a data code that must be converted in order to be used by other software in the system. For example, some card readers transmit a Hollerith code, which must be converted to ACSCII code to be meaningful. Interactive terminals (CRTs) can transmit special characters to signify that special processing is to be done, rather than have those characters be treated as data. CRTs use a variation of ASCII code for some special characters. The translation table is the mechanism provided to allow the writer of an input/output (I/O) driver to efficiently handle these special conditions.

5.2.1 The TLATE Instruction

Translation tables are processed by either the autodriver channel microcode routines or by the TLATE instruction. Typically, a driver accesses the translate table by both methods. The decision to use a translation table through the autodriver channel or via the TLATE instruction is dictated by the interrupt response characteristics of the specific device hardware. For example, a card reader typically begins generating character interrupts as soon as it is given a "go" command; thus, only the autodriver mechanism need be used. However, the Perkin-Elmer RS-232 interface requires that a single character be written to it before it generates character interrupts that can be used to drive the autodriver channel. Thus, for the RS-232 interface, the first character must be output via the TLATE and WD (WRITE DATA) instructions, while the remaining characters can be output using the autodriver channel.

5.2.2 Function and Design of a Translation Table

A translation table is structured as a list of 256 halfwords. One halfword entry corresponds to each possible value of an 8-bit character code. Bit 0 of each halfword entry specifies the interpretation that is to be made of the remaining 15 bits of that entry. If bit 0 is set, then the least significant byte contains the value that is to be substituted for the original character code. If bit 0 is zero, then bits

1 through 15 provide the address, divided by two, of a routine that is to be executed when the corresponding character code is encountered. In this case, an unconditional branch is taken to the indicated address. Note that this scheme requires that at least the first instruction of the special routine be in the first 65K of address space, since only halfword addresses can be specified.

To illustrate how a translation table can be used, let us write a simple driver for a hypothetical device that requires use of a translation table for efficient operation. To keep the example to a manageable size, assume that this device has some rather unusual characteristics.

We will assume that this device:

- is a transmit-only device (we can only read from it),
- transmits only the 8-bit codes X'08' thru X'23',
- the 8-bit codes X'0A' through X'23' are to be interpreted as the ASCII characters A thru Z,
- the device begins all transmissions with the code X'08', followed by one character that indicates the number of characters to follow, plus 10,
- the device may, for undisclosed reasons, prematurely terminate a transmission with an X'09' code.

First, a translation table for this strange device must be designed. Notice what only codes X'08' through X'23' are valid, and that codes X'08' and X'09' are special codes that indicate that special processing action is required. Common Assembly Language/32 (CAL/32) conveniently provides a special data type for setting up addresses in translation tables:

```
TABLE   EQU   *
        DO    8               codes X'00' thru X'07' illegal
        DC    T(ILLEGAL)
        DC    T(START)        code X'08' = start of xfer
        DC    T(STOP)         code X'09' = end of xfer
        DC    X'8000'+C'A'    code X'10' = letter A
              .
              .
              .
        DC    X'8000'+C'Z'    code X'23' = letter Z
        DO    256-35
        DC    T(ILLEGAL)      all other codes are illegal
```

5.2.3 Driver Code for the Translation Table

With this table defined, a driver can now be written to drive the device. Since we have assumed that the device has hardware characteristics that are convenient for writing software, the driver in this example is simpler than it probably would be for a real device. (Compare this code to that of the "read" side of the Perkin-Elmer CRT driver!) We will code this driver so that all character handling is done in interrupt service. Clearly, three special routines are needed: start of transmission, end of transmission and illegal transmission.

For the purposes of this example, we will ignore some necessary details, such as what to do about time-out conditions and bad device status. The intent of this example is to show only the code concerned with the use of the translation table. We will pick up the code at the end of the driver initialization routine (DIR) part of the driver:

```
INITXX    EQU    *                  DIR
          .
          .
          .
          SINT   R5,0(R6)           enter first ISR to start device
          B      DIRDONE            exit to wait for I/O completion
*
ISRO      EQU    *                  come here on SINT command from DIR
          LIS    E7,0               set up channel control block (CCB) for nonexecute
          STH    E7,CCB.CCW(E4)
          LA     E7,ISR1            set up address of next ISR
          STH    E7,CCB.SUBA(E4)
          OC     E2,ENABLE          send command to device to enable it
          LPSWR  E0                 exit to wait for first interrupt
*
ISR1      EQU    *                  first device interrupt comes here
          RDR    E2,E7              get input character into register 7
          CHI    E7,X'08'           is this a start of xfer character?
          BNE    ILLEGAL            if not, it must be illegal
```

Notice that at this point, we can determine what is a legal input simply by the time sequence: we demand that the first character received after the enable command is given must be a X'08'.

```
          LA    E7,ISR2           set up to receive next character
          STH   E7,CCB.SUBA(E4)   this will be character count
          LPSWR E0                wait for next character
*
ISR2      EQU   *                 second device interrupt comes here
          RDR   E2,E7             get input character into reg 7
          SIS   E7,10             this is character count, plus 10
*                                 (because we said that's how this
*                                 strange device works!)
          L     E5,CCB.DCB(E4)    get device control block (DCB) address,
          L     E6,DCB.SADR(E5)   and compute xfer end address for
          AR    E6,E7             insertion into the CCB...
          SIS   E6,1              (adjust for inclusive addressing)
          ST    E6,CCB.EB0(E4)    set end addr as buffer 0 address
          LIS   E6,0              buffer count must be -(length-1)
          SIS   E7,1              this is (length-1)
          SR    E6,E7             this is -(length-1)
          STH   E6,CCB.LB0(E4)    finally got the buffer length set
          LHI   E6,X'0082'        set up the channel command word (CCW)
          STH   E6,CCB.CCW(E4)    *
          LA    E7,TABLE          get address of translate table
          ST    E7,CCB.XLT(E4)    and put in CCB
          LA    E7,ISR3           get address of routine to go to
          STH   E7,CCB.SUBA(E4)   if xfer terminates normally
          LPSWR E0                exit, and let autodriver channel
*                                 do the rest of the work.
```

(The data transfer, with translation of character values, proceeds under control of the autodriver channel, as defined by the values in the CCB. There are three possible exits from autodriver operation: to ISR3, to ILLEGAL or to STOP. All of these routines are ISRs.)

```
ISR3    EQU   *                 enter this routine if xfer is normal
        L     E5,CCB.DCB(E4)    get DCB address
        LIS   E6,0              set normal termination status
        STH   E6,DCB.STAT(E5)
        L     E7,CCB.EBO(E4)    compute actual length of xfer...
        S     E7,DCB.SADR(E5)
        AIS   E7,1
ISR3A   ST    E7,DCB.LXFR(E5)
        OC    E2,DISABLE        turn off the device
        LA    E7,III            and reset the ISPTAB
        STH   E7,ISPTAB(E2,E2)
        L     E7,DCB.LEAF(E5)   finally, schedule the ESR
        ATL   E7,SQ             on the system queue
        LPSWR E0                and exit interrupt service
*
ILLEGAL EQU   *                 control will come here if
*                               an illegal input is encountered
        L     E5,CCB.DCB(E4)    get DBC address
        LHI   E6,X'8484'        designer's choice - indicate
        STH   E6,DCB.STAT(E5)   a bad character was received,
*                               by setting X'8484' in status
        LIS   E7,0              again design choice - show
*                               zero length of transfer
        B     ISR3A             go to common exit
*
STOP    EQU   *                 autodriver will come here
*                               if device sends a "stop" code
        L     E5,CCB.DCB(E4)    get DCB address
        LHI   E7,X'8201'        design choice - we show this
        STH   E7,DCB.STAT(E5)   status if we get a stop code
        LH    E7,CCB.LBO(E4)    compute actual length of xfer...
        A     E7,CCB.EBO(E4)    (this is actual end addr +1)
        S     E7,DCB.SADR(E5)   less start address
        SIS   E7,2              (arithmetic adjustment)
        B     ISR3A             go to common exit
```

Notice that we have defined only two of the three "special routines" defined in the translation table. The "START" routine was rendered unnecessary, due to the way that the ISRs were designed. Thus, if the "start" code was received at any other time, it would be an "illegal" code. The definition of the "START" routine may be handled simply as:

```
START   EQU    ILLEGAL
```

Probably, the greatest mystery in the above code to the first-time reader is the fact that nowhere was there any executable code written that explicitly invoked the operation of the autodriver channel and the use of the translation table. Actually, the autodriver channel was enabled by the value X'8002', which was stored into the CCB.CCW. That same value, plus the address of the translation table being placed in CCB.XLT, told the autodriver channel to use the translation table that we wrote. The actual operation of the autodriver channel occurred in response to interrupts generated by the device.

5.3 NONPHYSICAL DEVICE DRIVERS

In some circumstances, it is desirable to have in the system a "device" that has no physical reality. Such a device can have whatever peculiar characteristics are useful to resolving particular system design requirements. The characteristics and behavior of such a device are determined entirely by the driver. The driver that will be used as an example defines a pair of related nonphysical devices that, together, allow a large number of user tasks to coordinate access to a single system resource (typically, an array processor).

5.3.1 Purpose of a Nonphysical Device Driver

The purpose of this nonphysical device driver is to permit several tasks to share one or more devices in a manner not supported under the standard features of OS/32. This driver provides the ability for one task to do several I/O operations to a device, separated by time intervals which may be quite long. The driver prevents any other task sharing that device from doing any I/O operations to that device. The standard support in OS/32 permits a shared device to be accessed by multiple tasks on a first-come, first-served basis and, thus, provides no mechanism whereby one task can have exclusive access to the device for a prolonged period of time (and at the same time, share that device with other tasks).

5.3.2 Coding a Nonphysical Device Driver

The method implemented here was chosen over several alternatives, on the basis of central processing unit (CPU) overhead required to maintain the controlled access to the device.

This nonphysical device driver does not provide any "automatic" or "guaranteed" access control to a device or devices. It provides proper access coordination only among "friendly" tasks, all of which agree to use the conventions defined by this nonphysical device. The use of this "coordination device" can be illustrated by the following example: assume that two devices are to be shared among any number of tasks, with access controlled by these nonphysical devices. Assume the devices are named DEV1: and DEV2:. Further, assume that these two nonphysical devices are named GET: and REL:. Then, each task sharing actual devices DEV1: and DEV2: need to have the following logical units assigned: DEV1:, DEV2:, GET: and REL:. Assume that these are (for the purposes of this example) LU 6 = DEV1:, LU 7 = DEV2:, LU 8 = GET: and LU 9 = REL:. To gain access to either DEV1: or DEV2:, the task requests access permission by issuing a read to the "device" GET:, such as:

```
      READ(8,900) NAME
  900 FORMAT(1A4)
```

When either DEV1: or DEV2: is available, the read operation completes, and the fullword 'NAME' contains the device mnemonic of the available device (either DEV1: or DEV2:). Thus, after the read, 'NAME' would, in this example, contain the ASCII string "DEV1" or "DEV2", depending upon which device was free.

When the task has completed its series of I/O operations to the device, the task releases the device to some other task by writing the name of the device to the "REL:" nonphysical device:

```
  WRITE(9,900) NAME
```

In this example, the contents of 'NAME' is the same ASCII string that was last read from the "GET:" device. It is the responsibility of the individual user tasks (u-tasks) to correlate the device name to the specific logical unit (lu) assignment applicable to the task in question.

This driver utilizes a special input/output handler (IOH) entry to force special code to be executed upon system initialization. By having a special routine that executes only at system initialization time, the driver can initialize itself once, thus reducing the amount.

```
*
**CORD
          MLIBS  8,9,10,11
INITCORD  PROG       SYSTEM COORDINATION NONPHYSICAL DEVICE
          NLIST
          $REGS$
          $DCB$
          $CCB
          $TCB$
          $IOH
          LIST
          EXTRN DMT,DIRDONE,EVRTE,IODONE,SQ
          EXTRN SV1FCER,SVC1READ,SVC1WRIT,SVC1NOOP
          EXTRN COMEOT
          ENTRY INITCORD,TERMCORD,CORDEOT,CORDINIT
*
* The following structure defines a device-dependent part of
* the DCB. This section is used to coordinate
* access to the limited-access devices. This is defined here
* rather than in a separate structure macro as a matter of
* convenience of its presentation to the reader.
*
DCB.DTBL  STRUC
          DS    DCB.CCB           DEFINE DEVICE DEPENDENT AREA
```

```
         DS     2                    ADDR OF ASSOCIATED CCB
DCB.XDCD DS     2                    DCB.DCOD OF CONTROLLED RESOURCE
DCB.REQD EQU    *                    ADDRESS OF ASSOCIATED DCB243-SET UP AT
*                                    * SYS INIT TIME.
DCB.DCBT DS     4*20                 TABLE OF DCBS OF CONTROLLED UNITS
DCB.TCBT DS     4*20                 TABLE OF TCBS ASSOCIATED WITH DCB'S
DCB.DMNT DS     4*20                 TABLE OF ASSOCIATE DEVICE MNEMONICS
DCB.SIZT DS     4                    POINTER/COUNTER - ENTRIES IN TABLES
DCB.PEND DS     4                    REQUEST-PENDING FLAG
         ENDS
INITCORD  EQU    *
*
```

DCB 243 is a read-only device, whereas DCB 244 is a write-only device. A task requesting access to the controlled-access device issues a read to the nonphysical device system generated (sysgened) as device code 243. When the controlled resource is available, the read goes to I/O completion, putting the device mnemonic of the available controlled-access device into the calling task's I/O buffer. A task wishing to release a controlled resource writes to the nonphysical device sysgened as device code 244, with the device mnemonic of the device being released in the I/O buffer specified in the SVC1 write block.

```
*
* ON ENTRY TO THE DRIVER, DETERMINE WHICH TYPE OF NONPHYSICAL DEVICE
* THE I/O OPERATION IS DIRECTED TOWARD.
*
* FIRST, BE SURE THE TIME-OUT CONSTANT IS SET SO THAT WE
* CAN NEVER BE TIMED OUT.
         LHI    UO,X'7FFF'
         STH    UO,DCB.TOUT(UD)
         LB     UO,DCB.DCOD(UD)        GET OUR DEVICE CODE
         CHI    UO,243                 TEST WHICH TYPE
         BNE    INIT.100               DCB 244 PROCESSED ELSEWHERE.
*
* THE READ-ONLY DCB WAS REFERENCED. VERIFY THIS IS A READ OPERATION
* AND IF NOT, RETURN AN ILLEGAL FUNCTION STATUS.
*
         LB     UO,DCB.FC(UD)          GET THE CALLER'S FUNCTION
         NHI    UO,X'60'               SAVE ONLY FUNCTION BITS
         CHI    UO,X'40'               CHECK FOR READ
         BNE    INIT.200               IF NOT, ERROR OUT
*
* NOW TEST THE I/O AREA TO BE SURE THERE IS ENOUGH AREA
```

```
*
         L      UO,DCB.EADR(UD)
         S      UO,DCB.SADR(UD)
         CHI    UO,3
         BM     INIT.210               PARITY ERROR IF NOT ENUF ROOM
*
* NOW LOOK FOR A FREE RESOURCE.
*
         L      U1,DCB.SIZT(UD)        GET SIZE OF TABLE
INIT.010 SIS    U1,4                   DECREMENT POINTER
         BM     INIT.050               IF NOTHING AVAILABLE, WAIT
         L      UO,DCB.TCBT(UD,U1)     TEST AN ENTRY FOR AVAILABILITY
         BNZ    INIT.010               IF NOT AVAILABLE, TRY NEXT ONE
*
* A CONTROLLED RESOURCE IS AVAILABLE, SO ASSIGN THE RESOURCE
* TO THIS TASK, GIVE THE TASK THE DEVICE MNEMONIC OF THE RESOURCE,
* AND EXIT
*
INIT.020 L      UO,DCB.DMNT(U1,UD)     GET NAME OF AVAILABLE DEVICE
         L      U2,DCB.SADR(UD)        GET THE START ADDRESS OF THE BUFFER
         EXHR   UO,UO                  AND PUT THE DEVICE MNEMONIC OUT
         STH    UO,O(U2)               *
         EXHR   UO,UO                  *
         STH    UO,2(U2)               *
         LIS    UO,4                   SET LENGTH OF XFER
         ST     UO,DCB.LLXF(UD)        *
         LIS    UO,O                   SET STATUS = O
         STH    UO,DCB.STAT(UD)        *
         ST     UO,DCB.PEND(UD)        CLEAR THE REQUEST-PENDING FLAG
         L      UO,DCB.TCB(UD)         GET REQUESTING TCB ID
         ST     UO,DCB.TCBT(U1,UD)     AND SAVE TO SHOW IN USE
         B      IODONE                 ALL DONE.
*
* COME HERE IF THERE IS NO CONTROLLED RESOURCE AVAILABLE
* WHEN A REQUEST IS ISSUED.  SET A FLAG TO INDICATE THAT
* A REQUEST IS PENDING, AND THEN EXIT TO DIRDONE.
*
INIT.050  LIS   UO,1
          ST    UO,DCB.PEND(UD)        SET REQUEST PENDING FLAG
          B     DIRDONE                AND EXIT
*
* PROCESS AN I/O DIRECTED AT A DCB 244 DEVICE. VERIFY THE
* REQUEST IS A WRITE. IF IT IS NOT, EXIT WITH AN ILLEGAL
* FUNCTION ERROR. IF IT IS A WRITE, VERIFY THERE ARE AT LEAST
* FOUR BYTES IN THE I/O BUFFER. IF NOT, EXIT WITH A PARITY
* ERROR CODE. IF AT LEAST 4 BYTES EXIST, USE THE FIRST FOUR
* BYTES AS A DEVICE MNEMONIC. SCAN THE INTERNAL TABLE FOR THE
* CONTROLLED DEVICE, AND, IF FOUND, VERIFY THAT THE
* CALLING TCB IS THE SAME AS THE TCB ID IN THE TABLE.
* IF IT IS NOT, EXIT WITH A DEVICE UNAVAILABLE ERROR. IF THE
* TCB IS THE SAME, CLEAR THE TCB ENTRY IN THE TABLE.
* IF THE TCB DOES NOT MATCH, EXIT WITH AN UNRECOVERABLE ERROR.
* THEN CHECK FOR ANY REQUEST OUTSTANDING. IF SO, SCHEDULE AN
* EVENT AGAINST THE READ DEVICE. THEN EXIT.
*
INIT.100 LB    UO,DCB.FC(UD)        GET THE CALLER'S FUNCTION CODE
         NHI   UO,X'60'             MASK ALL BUT FUNCTION BITS
         CHI   UO,X'20'             IS IT A WRITE?
         BNE   INIT.200             IF NOT, ILLEGAL FUNCTION.
*
         L     UO,DCB.EADR(UD)      CHECK THE BUFFER LENGTH
         S     UO,DCB.SADR(UD)      *
         CHI   UO,3                 *
         BM    INIT.210             IF NOT, IT IS A PARITY ERROR.
         L     U2,DCB.SADR(UD)      GET THE BUFFER ADDRESS AGAIN.
         LH    UO,0(U2)             GET THE CONTENTS OF THE BUFFER
         SLL   UO,16                *
         OH    UO,2(U2)             *
          L     UC,DCB.REQD(UD)     GET ADDRESS OF REQUEST DCB
         L     U1,DCB.SIZT(UC)      GET THE SIZE OF THE INTERNAL TABLE
INIT.110 SIS   U1,4                 DECREMENT THE POINTER
         BM    INIT.220             IF NOT FOUND, GIVE BACK D-U
         C     UO,DCB.DMNT(U1,UC)   IS IT THIS ENTRY?
         BNE   INIT.110             NO, TRY NEXT ENTRY.
*
         L     UO,DCB.TCB(UD)       GET TCB OF REQUESTER.
         C     UO,DCB.TCBT(U1,UC)   AND CHECK AGAINST TABLE.
         BNE   INIT.230             IF WRONG TASK, GIVE UNRECV ERR
         LIS   UO,0                 ALL IS O.K., SO RESET TABLE
         ST    UO,DCB.TCBT(U1,UC)   *
*
         L     UO,DCB.PEND(UC)      GET THE REQUEST-PENDING FLAG
         BZ    IODONE               IF NONE, ALL DONE
         L     UO,DCB.LEAF(UC)      IF REQ. PENDING, SCHEDULE ESR.
         ATL   UO,SQ                *
         BZ    IODONE               AND THEN WE ARE DONE.
```

```
*
* ERROR EXITS
*
INIT.200 LHI   UO,X'COOO'          ILLEGAL FUNCTION
         B     INIT.250
INIT.210 LHI   UO,X'8282'          PARITY ERROR - I/O BUFFER TOO SMALL
         B     INIT.250
INIT.220 LHI   UO,X'AOOO'          D-U IF NO SUCH DEVICE
         B     INIT.250
INIT.230 LHI   UO,X'8400'          UNRECV IF WRONG TASK TRYING RELEASE
         B     INIT.250
*
* ON ERROR EXIT, PUT ERROR CODE IN DCB AND EXIT
*
INIT.250 STH   UO,DCB.STAT(UD)     *
         B     IODONE              *
*
* TERMINATION (ESR) ENTRY POINT.
TERMCORD  EQU   *
*
* THIS ROUTINE WILL BE ENTERED ONLY IF A REQUEST HAS BEEN PENDING,
* AND THERE IS NOW A FAIR CHANCE THAT A CONTROLLED RESOURCE IS NOW
* AVAILABLE. IF A REQUEST GETS THIS FAR, THE I/O BUFFER HAS BEEN
* CHECKED FOR CORRECT SIZE, SO ALL THAT IS NEEDED IS TO LOCATE
* THE FREE RESOURCE, INFORM THE CALLING TASK, MARK THE RESOURCE AS
* IN USE, AND EXIT TO IODONE.
*
* THE FOLLOWING LOOP SCANS THE INTERNAL TABLE FOR A FREE RESOURCE.
*
         L     U1,DCB.SIZT(UD)     GET SIZE OF TABLE
TERM.010 SIS   U1,4                DECR THE POINTER
         BM    EVRTE               NOTHING FREE, SO GET OUT
         L     UO,DCB.TCBT(U1,UD)  TEST AN ENTRY FOR AVAILABILITY
         BNZ   TERM.010            NOT FREE, TRY ANOTHER ONE.
*
* FOUND A FREE ENTRY. WE CAN NOW USE THE SAME PROCESSING AS THE
* 'INIT' PHASE, GO THERE TO FINISH UP
         B     INIT.020
* ENTRY TO CORDEOT IS VIA AN IOH ENTRY.
*
* THIS ROUTINE IS ENTERED VIA A BALR U8,U8 FROM EXSV.  SETTING
* IOH.EOT FORCES BYPASSING THE COMEOT HANDLER (DEFAULT SET BY
* EXSV).  UPON RETURN TO EXSV, REGISTER NINE IS RELOADED WITH
* THE TCB ADDRESS BUT NO OTHER REGISTERS ARE PRESERVED BY THE
* CALLING ROUTINE.
*
*    REFER TO MODULE EXSV FOR ADDITIONAL DETAILS
*
CORDEOT  EQU   *
*
* UPON ENTRY  U9 = TCB ADDR OF TASK GOING TO EOT
*             UA = DCB ADDR OF THE DEVICE FOR WHICH EOT PROCESSING
*                  IS NOW BEING PERFORMED.
*
* EOT PROCESSING WILL CONSIST OF ZEROING OUT THE TCB TABLE ENTRY
* FOR ANY DEVICE WHICH HAS THIS TCB ADDRESS IN ITS 'DCB.TCBT'
* FIELD IN THE INTERNAL CONTROL TABLES. THEN, CHECK FOR ANY
* REQUEST OUTSTANDING - IF SO, PROCESS THE REQUEST.
*
         L     UD,DCB.REQD(UA)      GET ADDRESS OF DCB FOR
         LR    UF,UA                SAVE DCB ADDRESS
*                                   ACQUISITION SIDE.
         L     UA,DCB.SIZT(UD)      GET SIZE OF CONTROL TABLE.
EOT.01   SIS   UA,4                 DECREMENT THE POINTER
         BM    EOT.03               IF NOTHING ELSE, WE ARE DONE
         C     U9,DCB.TCBT(UA,UD)   IS THIS AN ENTRY TO BE RESET?
         BNE   EOT.02
         LIS   UC,0                 GET A ZERO
         ST    UC,DCB.TCBT(UA,UD)   CLEAR THE ENTRY
EOT.02   LR    UA,UA                ARE WE AT END OF TABLE
         BP    EOT.01               GO UNTIL DONE
*
* IT IS NOW NECESSARY TO CHECK FOR ANY UNHONORED REQUESTS PENDING.
*
         L     UC,DCB.PEND(UD)      TEST THE REQUEST PENDING FLAG
         BZ    EOT.03               IF NO REQUEST, WE ARE ALL DONE
         C     U9,DCB.TCB(UD)       IS REQUESTER THIS TASK?
         BZ    EOT.03               IF SO, DO NOTHING
         L     UC,DCB.LEAF(UD)      OTHERWISE, SCHEDULE AN EVENT
         ATL   UC,SQ                ON THIS DCB
EOT.03   EQU   *
         LR    UA,UF                RESTORE FOR COMEOT
         B     COMEOT               * GO TO STANDARD EOT HANDLING
*
* INIT ROUTINE WHICH IS EXECUTED DURING SYSINIT.
* THE SYSINIT ROUTINE SAVES REGISTERS EIGHT THROUGH
* FIFTEEN BEFORE IT BRANCHES TO THE IOH ENTRY.
* THE BRANCH IS PERFORMED BY A BALR E8,ED.
* ED--HAS BEEN LOADED WITH THE IOH INIT ENTRY--IF THIS
* ENTRY IS ZERO--SYSINIT DOES NOTHING WITH IT.
* *
* THUS THE IOH.INIT ROUTINE MAY USE REGISTERS 9 THROUGH
* FIFTEEN TO GET ITS JOB DONE.
*
* THE BRANCH REGISTER TO RETURN FROM AN IOH ENTRY IS REGISTER EIGHT
*
* UPON ENTRY TO THE IOH ROUTINE.....UB = DCB ADDRESS
*
*   REFER TO MODULE EXIN FOR ADDITIONAL DETAILS
*
CORDINIT EQU   *
*
* THERE ARE TWO SEPARATE TYPES OF SYSTEM INITIALIZATION, ONE FOR
* THE ACQUISITION NON-PHYSICAL DEVICE, AND ONE FOR THE RELEASE
* NON-PHYSICAL DEVICE. BRANCH TO THE APPROPRIATE ROUTINE BASED ON THE
* DEVICE CODE OF THE NON-PHYSICAL DEVICE.
*
* ON ENTRY, UB = A(DCB) OF COORDINATION NONPHYSICAL DEVICE
*
         LB    UE,DCB.DCOD(UB)      GET DEVICE CODE
         CHI   UE,243
         BE    INIT.00
         BNE   INIT.10
*
*  GO THRU THE DEVICE MNEMONIC TABLE AND BUILD A LIST OF
*  DCB ADDRESSES CORRESPONDING TO ALL THE DEVICES USING THE
*  DEVICE CODE SPECIFIED IN 'DCB.XDCD'
*
INIT.00  LH    UE,DCB.XDCD(UB)      GET DEVICE CODE WE ARE TO CONTROL
         LIS   UA,0                 COUNTER, AND POINTER TO INTERNAL TABLE
         LA    UF,DMT               GET ADDRESS OF DMT
INIT.01  AIS   UF,8                 INCREMENT POINTER TO DEVICE MNEMONIC TABLE
         L     UC,4(UF)             GET A DCB ADDRESS
         BZR   U8                   IF END OF TABLE, WE ARE DONE
         LB    UD,DCB.DCOD(UC)      GET THE DEVICE CODE OF THIS DEVICE
         CR    UD,UE                IS THIS ONE WE ARE INTERESTED IN?
         BNE   INIT.01              IF NOT, GO GET NEXT ENTRY
         ST    UC,DCB.DCBT(UA,UB)   SAVE THIS DCB
         LIS   UC,0                 GET A ZERO
         ST     UC,DCB.TCBT(UA,UB)    ZERO THE TCB ENTRY FOR THIS DCB.
         L      UD,0(UF)              GET THE DEVICE MNEMONIC FOR THIS DEVICE
         ST     UD,DCB.DMNT(UA,UB)    AND SAVE IT IN THE TABLE
         AIS    UA,4                  INCREMENT THE POINTER TO NEXT SLOT
         ST     UA,DCB.SIZT(UB)       SAVE THE POINTER
         B      INIT.01               AND GO GET NEXT ENTRY
*
* GO THROUGH THE DEVICE MNEMONIC TABLE AND FIND THE DCB FOR
* THE ACQUISITION NONPHYSICAL DEVICE WHICH IS SYSGENED TO CONTROL
* THE SAME DEVICE CODE AS THE DCB FOR WHICH THIS ENTRY WAS MADE.
* PUT THE ADDRESS OF THAT DCB INTO THE RESERVED SPACE IN THE
* DCB FOR WHICH THIS PROCESSING IS BEING PERFORMED.
*
INIT.10  LH     UE,DCB.XDCD(UB)       GET DEVICE CODE WE ARE TO CONTROL
         LA     UF,DMT                GET ADDRESS OF DEVICE MNEMONIC TABLE
INIT.11  AIS    UF,8                  INCREMENT POINTER TO DEVICE MNEMONIC TABLE
         L      UC,4(UF)              GET A DCB ADDRESS
         BZR    U8                    IF END OF TABLE, WE ARE DONE
         LB     UD,DCB.DCOD(UC)       GET THE DEVICE CODE OF THIS DEVICE
         CHI    UD,243                IS THIS AN ACQUISITION NONPHYSICAL DEVICE
         BNE    INIT.11
         LH     UD,DCB.XDCD(UC)       DOES IT CONTROL THE SAME DCOD WE DO?
         CR     UD,UE                 IS THIS ONE WE ARE INTERESTED IN?
         BNE    INIT.01               IF NOT, GO GET NEXT ENTRY
         ST     UC,DCB.REQD(UB)       SAVE THIS DCB ADDRESS
         BR     U8                    ALL DONE
*
* DEFINE THE IOH LIST FOR ACQUISITION DEVICE
*
         IOH    NAME=CRD1.IOH,                                       1
                READ=SVC1READ,                                       1
                WRITE=SVC1NOOP,                                      1
                WAIT=SVC1NOOP,                                       1
                HALT=SVC1NOOP,                                       1
                EOT=0,                                               1
                TEST=SVC1NOOP,                                       1
                SET=SVC1NOOP,                                        1
                REW=SVC1NOOP,                                        1
                BSR=SVC1NOOP,                                        1
                FSR=SVC1NOOP,                                        1
                WFM=SVC1NOOP,                                        1
                FFM=SVC1NOOP,                                        1
                BFM=SVC1NOOP,                                        1
                INIT=CORDINIT
*
* DEFINE THE IOH LIST FOR RELEASE DEVICE
*
         IOH    NAME=CRD2.IOH,                                          1
                READ=SVC1NOOP,                                          1
                WRITE=SVC1WRIT,                                         1
                WAIT=SVC1NOOP,                                          1
                HALT=SVC1NOOP,                                          1
                EOT=CORDEOT,                                            1
                TEST=SVC1NOOP,                                          1
                SET=SVC1NOOP,                                           1
                REW=SVC1NOOP,                                           1
                BSR=SVC1NOOP,                                           1
                FSR=SVC1NOOP,                                           1
                WFM=SVC1NOOP,                                           1
                FFM=SVC1NOOP,                                           1
                BFM=SVC1NOOP,                                           1
                INIT=CORDINIT
         END
         BEND
```

SYSGEN32 DCB macros for nonphysical device driver:

```
         MACRO
         DCB243 %DCOD=,%DN=,%CLAS=,%ILVL=,%NAME=,%SHCCB=,               1
                 %SLCH=,%XDCD=
         GBLB   %DCB$,%PDCB,%DDCB,%EVN,%CCB,%DFLG,%SDCB
         GBLB   %IDCB,%ODCB,%S125DCB,%ICCB,%BDCB
         GBLB   %ADCB,%TCB,%IOB,%IOB$,%CRTDCB,%LPDCB
         GBLB   %MMDDX,%DDEX,%VFDCB,%MTP,%CRPDCB,%MGDCBX,%HFWDST
         GBLB   %PSDCBX,%CRDP,%AOBDCB,%BIOCDCB,%LPTDCB
         GBLB   %CORD243
         GBLC   %IDVAL
         BGBLA  %ID243
         LCLA   %CCBFL
         LCLA   %CLASN
         LCLC   %RXLT,%RQU
         LCLC   %CORDNM,%PTRPAS
         LCLC   %OFFS
         LCLA   %RDN
         LCLC   %MDN,%MCNT,%MSLCH
         LCLA   %TRCNT,%UPTR
         LCLB  %FOUND,%DA
         BGBLA %FIRST
%RQU     SETC  'COMQ'                 DEFAULT DEVICE QHANDLER
%MDN     SETC  '%DN'                  DEVICE ADDRESS
%CCBFL   SETA  0
%CORD243 SETB  0
         AIF   (T'%CLAS EQ 'U')&CLSNTD
%CLASN   SETA  %CLAS*12               IOCLASS*12
&CLSNTD  ANOP
         CONVNUM VAL=%ID243    CONVERT CURRENT ID TO HEX.
         USERINIT
         $DCB$
         DCBI  DCOD=243,SIZE=DCB.DVDP+4,INIT=INITCORD,             1
               TERM=TERMCORD,FLGS=DFLG.LNM+DFLG.UCM,               2
               ID=%IDVAL,ATRB=EB80,COPY=$CORD243,IOH=CRD1.IOH
         CCBI  DCOD=243,ID=%IDVAL,SUBA=III
CCB%NAME EQU   CCB%DCOD%IDVAL
%ID243   SETA  %ID243+1
&DCBOPT  ANOP
DCB%DCOD%IDVAL  PROG   USER DCB
%OFFS    SETC  '%DCOD':'%IDVAL'     ESTABLISH PROPER OFFSET
DCB.%NAME EQU  DCB%OFFS
         ENTRY DCB.%NAME
         ORG   DCB%OFFS+DCB.DMT
         DC    DMT.%NAME
         EXTRN DMT.%NAME
         ORG   DCB%OFFS+DCB.DN        DEVICE ADDRESS
         DC    H'%DN'
         ORG   DCB%OFFS+DCB.LEAF      LEAF POINTER
         AIF   (T'%SHCCB' EQ 'U')&NSLEAF   B IF NOT SHARED
         DAC   LF%SHCCB               USE SHARED DEVICE LEAF
         EXTRN LF%SHCCB
         AGO   &NRMLFX
&NSLEAF  ANOP
         DAC   LF%OFFS               GENERATE STANDARD LEAF NAME
         EXTRN LF%OFFS
&NRMLFX  ANOP
&NOLEAF  ANOP
         AIF   (T'%CLAS EQ 'U')&NOCLAS
         ORG   DCB%OFFS+DCB.CLAS     IO CLASS
         DC    H'%CLASN'             IOCLASS*12
&NOCLAS  ANOP
```

```
          AIF    (T'%ILVL EQ 'U')&NOILVL
          ORG    DCB%OFFS+DCB.ILVL   ILEVEL
          DC     H'%ILVL'
&NOILVL   ANOP
          AIF    ('%RQU' EQ '')&NOQU
          ORG    DCB%OFFS+DCB.Q
          DAC    %RQU
          EXTRN  %RQU
&NOQU     ANOP
          ORG    DCB%OFFS+DCB.XDCD
          DC     %XDCD
          ORG    DCB%OFFS+DCB.SIZT
          DC     0
          ORG    DCB%OFFS+DCB.PEND
          DC     0
          ORG    $ST%OFFS            ORG TO END OF DCB
          ASIS
          END
%RDN      SETA   %DN+1
          MEND
          MACRO
          DCB244 %DCOD=,%DN=,%CLAS=,%ILVL=,%NAME=,%SHCCB=,                    1
                  %SLCH=,%XDCD=
          GBLB   %DCB$,%PDCB,%DDCB,%EVN,%CCB,%DFLG,%SDCB
          GBLB   %IDCB,%ODCB,%S125DCB,%ICCB,%BDCB
          GBLB   %ADCB,%TCB,%IOB,%IOB$,%CRTDCB,%LPDCB
          GBLB   %MMDDX,%DDEX,%VFDCB,%MTP,%CRPDCB,%MGDCBX,%HFWDST
          GBLB   %PSDCBX,%CRDP,%AOBDCB,%BIOCDCB,%LPTDCB
          GBLB   %CORD244
          GBLC   %IDVAL
          BGBLA  %ID244
          LCLA   %CCBFL
          LCLA   %CLASN
          LCLC   %RXLT,%RQU
          LCLC   %CORDNM,%PTRPAS
          LCLC   %OFFS
          LCLA   %RDN
          LCLC   %MDN,%MCNT,%MSLCH
          LCLA   %TRCNT,%UPTR
          LCLB   %FOUND,%DA
          BGBLA  %FIRST
%RQU      SETC   'COMQ'                DEFAULT DEVICE QHANDLER
```

```
%MDN      SETC  '%DN'                  DEVICE ADDRESS
%CCBFL    SETA  0
%CORD244  SETB  0
          AIF   (T'%CLAS EQ 'U')&CLSNTD
%CLASN    SETA  %CLAS*12              IOCLASS*12
&CLSNTD   ANOP
          CONVNUM VAL=%ID244          CONVERT CURRENT ID TO HEX.
          USERINIT
          $DCB$
          DCBI  DCOD=244,SIZE=DCB.DVDP+4,INIT=INITCORD,                1
                TERM=TERMCORD,FLGS=DFLG.LNM+DFLG.UCM,                  2
                ID=%IDVAL,ATRB=EB80,COPY=$CORD244,IOH=CRD2.IOH
          CCBI  DCOD=244,ID=%IDVAL,SUBA=III
CCB%NAME EQU    CCB%DCOD%IDVAL
%ID244    SETA  %ID244+1
&DCBOPT   ANOP
DCB%DCOD%IDVAL  PROG  USER DCB
%OFFS     SETC  '%DCOD':'%IDVAL'      ESTABLISH PROPER OFFSET
DCB.%NAME EQU   DCB%OFFS
          ENTRY DCB.%NAME
          ORG   DCB%OFFS+DCB.DMT
          DC    DMT.%NAME
          EXTRN DMT.%NAME
          ORG   DCB%OFFS+DCB.DN       DEVICE ADDRESS
          DC    H'%DN'
          ORG   DCB%OFFS+DCB.LEAF     LEAF POINTER
          AIF   (T'%SHCCB' EQ 'U')&NSLEAF   B IF NOT SHARED
          DAC   LF%SHCCB              USE SHARED DEVICE LEAF
          EXTRN LF%SHCCB
          AGO   &NRMLFX
&NSLEAF   ANOP
          DAC   LF%OFFS               GENERATE STANDARD LEAF NAME
          EXTRN LF%OFFS
&NRMLFX   ANOP
&NOLEAF   ANOP
          AIF   (T'%CLAS EQ 'U')&NOCLAS
          ORG   DCB%OFFS+DCB.CLAS     IO CLASS
          DC    H'%CLASN'             IOCLASS*12
&NOCLAS   ANOP
          AIF   (T'%ILVL EQ 'U')&NOILVL
          ORG   DCB%OFFS+DCB.ILVL     ILEVEL
          DC    H'%ILVL'
&NOILVL   ANOP
```

```
          AIF    ('%RQU' EQ '')&NOQU
          ORG    DCB%OFFS+DCB.Q
          DAC    %RQU
          EXTRN  %RQU
&NOQU     ANOP
          ORG    DCB%OFFS+DCB.XDCD
          DC     %XDCD
          ORG    DCB%OFFS+DCB.REQD
          DC     0
          ORG    $ST%OFFS              ORG TO END OF DCB
          ASIS
          END
%RDN      SETA   %DN+1
          MEND
```

Macros used for COPY in DCBI macro for nonphysical device driver:

```
          MACRO
          $CORD243
          GBLB   %CORD243
          AIF  (NOT %CORD243)&NEEDIT
          MEXIT
&NEEDIT   ANOP
%CORD243    SETB  1
CORD243     STRUC       TABLE OF ASSOCIATED STRUCTURES FOR COORDINATION
          DS     DCB.CCB        DEFINE DEVICE-DEPENDENT AREA
          DS     2
DCB.XDCD  DS     2
          ALIGN  4
DCB.DCBT  DS     4*20     TABLE OF DCBS ON CONTROLLED UNITS
DCB.TCBT  DS     4*20     TABLE OF TCBS ASSOCIATED WITH DCBS
DCB.DMNT  DS     4*20     TABLE OF ASSOCIATED DEVICE MNEMONICS
DCB.SIZT  DS     4        POINTER/COUNTER - ENTRIES IN TABLE
DCB.PEND  DS     4        REQUEST-PENDING FLAG
          ENDS
          MEND

          MACRO
          $CORD244
          GBLB   %CORD244
          AIF  (NOT %CORD244)&NEEDIT
          MEXIT
```

```
&NEEDIT  ANOP
%CORD244   SETB  1
CORD244    STRUC       TABLE OF ASSOCIATED STRUCTURES FOR COORDINATION
         DS     DCB.CCB        BASIC DCB SIZE
         DS     2
DCB.XDCD DS     2
         ALIGN 4
DCB.REQD DS     4        ADDRESS-OF ASSOCIATED DCB243-SET UP
*                       * AT SYS-INIT TIME.
         ENDS
         MEND
```

5.4 SUPERVISOR CALL 6 (SVC6) AND TRAP GENERATING DEVICE DRIVERS

In certain system configurations, it is desirable to have an external device cause a certain piece of user's task code to be executed in response to a signal from that device. An example of such a situation is the existence of an external clock signal, which can be used to synchronize a cyclical processing activity, such as is found in flight simulators and process control systems.

5.4.1 Function of a Trap Generating Device Driver

In the case of the external synchronization signal (or clock pulse), no data is being transferred to or from the device. Rather, we simply want to have some particular set of user-level code that is to be executed in response to that external signal. The appropriate interface between the user-level program and the clocking signal is a trap generating device driver, using the SVC6 interface for such devices. To use this feature, the user's task must be set up to handle task traps and process items on a task queue. The user's task then "connects" itself to this trap generating device (i.e., clock pulse), and "thaws" (enables) the interrupts from the device by issuing an SVC6 instruction. The driver then adds items to the task's queue each time an interrupt occurs, which causes the task to be forced by the operating system to execute a prespecified piece of code. In this way, the external hardware signal can cause a specific section of user-level code to be executed each time an interrupt occurs.

5.4.2 Coding a Trap Generating Device Driver

The system designer should note that a trap generating device cannot also be defined (through the DCB) as a "normal" SVC1 interface device. However, a normal SVC1 device driver can be used to add items to a user's task queue (and thus cause task traps) in response to specific situations.

```
         MLIBS 8,9,10
         $REGS$
         $DCB$
         $CCB
         $TCB$
         EXTRN EVRTE,III,ISPTAB,SQ,TMATQ1
         ENTRY CMDTGD,INITTGD,ISRTGD,TERMTGD
*
* The function processing part of this driver is entered from
* the SVC6 processor in the operating system via the following
* instructions:
*
*                L    UB,DCB.FUNC(UD)
*                BALR UE,UB
*
*
* On entry from the SVC6 processor, the following register
* conventions apply:
*
*          RD = address of the DCB of the trap generating device
*          RE = address in SVC6 processor to return to after
*               performing the requested function.
*
*          Note that DCB.FC has been set up by the SVC6 processor
*          to be one of the following values:
*
*             DCB.FC = X'C0' for "thaw" function
*             DCB.FC = X'A0' for "freeze" function
*             DCB.FC = X'90' for "simulate interrupt" function
*
*            (these are the only three values that can appear)
*
INITTGD  EQU   *              not used - needed to satisfy
*                             DCB.INIT reference only.
CMDTGD   EQU   *                entry point specified in the
*                               DCB.FUNC term of the DCB.
*
          LH    U2,DCB.DN(UD)   get the device address
          LB    U3,DCB.FC(UD)   get the function code
          LHL   UC,DCB.CCB(UD)  get CCB address
*
          CHI   U3,X'C0'        "thaw" function?
          BE    TGDTHAW
         CHI   U3,X'A0'        "freeze" function?
         BE    TGDFREZ
         CHI   U3,X'90'        "simulate interrupt" function?
         BE    TGDSINT
*
* We should NEVER "fall through" all three tests. Thus, to
* assist in debugging an "impossible" situation, we will simply
* put a halt instruction here, so that if the "impossible" ever
* happens, we can easily examine all of the relevant registers
* and locations to see how we got here.
*
         HALT                 should never be executed.
*
* Process "thaw" function: set up the interrupt path to the
* interrupt service routine, and enable the interrupt hardware
* on the device.
*
TGDTHAW  EQU   *
         LA    U0,ISRTGD         get the address of the ISR
         STH   U0,CCB.SUBA(UC)   & put into the CCB.
         LIS   U0,0              get a zero,
         STH   U0,CCB.CCW(UC)    & reset the channel command word
         AIS   UC,1              make the CCB address odd
         STH   UC,ISPTAB(U2,U2)  & set up the ISP table
*
*    commands to enable hardware interrupts on the device go here
*
         BR    UE                return to the SVC6 processor
*
* Process "freeze" function: dismantle the interrupt path to the
* interrupt service routine, and disable the interrupt hardware
* on the device.
*
TGDFREZ  EQU   *
         LA    U0,III             get the address of the null
         STH   U0,ISPTAB(U2,U2)   interrupt routine & reset the ISPT
*                                 (note that since we do this, there
*                                 is no reason to also reset the
*                                 CCB.SUBA value)
*
* commands to disable hardware interrupts on the device go here.
*
         BR    UE                 return to the SVC6 processor
```

```
*
* Process "sint" function: simulate an interrupt on the device.
*
* Note: interrupts from the device, either real or simulated,
* can not be taken until the "thaw" function has been executed.
*
TGDSINT  EQU    *
         LH     U7,DCB.ILVL(UD)       get the proper interrupt level
         SINT   U7,0(U2)              generate the "interrupt"
         BR     UE                    return to SVC6 processor
*
* Interrupt service routine: when an interrupt from the device
* is received, schedule the Event Service Routine (ESR) for
* execution. Typically, a trap generating device needs no further
* commands from the processor once interrupts are enabled. Thus,
* it is unlikely that the ISR will contain any hardware interface
* command instructions.
*
         PURE
ISRTGD   EQU    *
         L      E5,CCB.DCB(E4)         get the address of the DCB
         L      E6,DCB.LEAF(E5)        get the item to be put on
         ATL    E6,SQ                  the system queue, and add it
         LPSWR  E0                     exit the ISR.
*
* Event Service Routine: The ESR adds an item to the user's
* task queue, by calling a routine in the operating system.
*
         IMPUR
TERMTGD  EQU    *
         L      U9,DCB.TCB(UD)          get the user's TCB address
         L      UA,DCB.PBLK(UD)         get the task queue parameter
*                                       that was specified by the
*                                       user in the "connect" function
         NI     UA,Y'FFFFFF'            & be sure it is 24 bits long
*                                       (this sets the "reason code"
*                                       as defined for task queue
*                                       entries to be equal zero).
         BAL    U8,TMATQ1               then call the OS routine
*                                       to add this to the task's
*                                       queue.
         B      EVRTE                   Exit from the ESR.
         END
```

APPENDIX A

REVIEW OF ASSEMBLY LANGUAGE

INPUT/OUTPUT (I/O) COMMANDS

Here are some notes on the usage of Common Assembly Language/32 (CAL/32).

When dealing with device interfaces, it is common to use I/O instructions that are rarely used under any other circumstances. A review of some of these commands is included here.

Remember that all I/O commands are privileged commands.

SENSE STATUS

```
SS    R1,STAT1                Put status into R2 of the device whose
                              address is in R1.

SSR   R1,R2                   Put status into R2 of the device whose
                              address is in R1.
```

SIMULATE INTERRUPT

```
SINT R1,0(R2)                 Where R1 is the interrupt level and
                              R2 is the device address.
```

OUTPUT COMMAND

```
          OC    R1,DISARM       Disarm the interface
          .
          .
          .
DISARM    DB    X'C0'           Disarm the command
```

WRITE DATA/WRITE HALFWORD

```
WH    R1.DCB.SADR(UD)           Where R1 contains the device address
                                and DCB.SADR(UD) is the address that
                                contains the data to be written.
```

NOTE

None of the I/O commands is provided in immediate format!

APPENDIX B

DEBUG TECHNIQUES

This appendix is a brief discussion of two convenient debug techniques.

- The single step method:

  This method is useful in tracing the basic flow of the driver. The systems integrator places the machine into single step mode (by depressing the single button on the front panel or in the case of a Model 3210, by using the less than (<) symbol), and then proceeds to step through the code.

  The single step method is not suitable for tracing any problems that are timing related.

- Inserting 'HALT' instructions:

  A HALT (8800) instruction can be inserted into the driver code at any point. The driver is then free running until the HALT (breakpoint) instruction is executed. This method is often used to trap a driver if it is executing code that should never be reached. Some timing problems can be traced by way of this method.

APPENDIX C

CRASH CODE ANALYSIS

The following is a listing of crash code messages and their meaning:

-----153	A crash code of 153 denotes an invalid memory alignment condition. When this occurs within the driver code, it is most commonly found to be a fullword reference on a halfword boundary. This crash may also occur if a device leaf is added to the system queue more than once.
----102	This crash code is an illegal instruction that is executed within system code. This is the most difficult crash to trace simply because the pointer that you have points you to the illegal instruction and not to the mechanism that brought you to the illegal instruction. The common causes are random branches (i.e., branches using the wrong register), and data constants not properly placed in memory.
----142	This crash code is caused by a supervisor call 1 (SVC1) ending parameter block address that is less than the SVC1 starting parameter block address.
----132	This crash code is caused by an illegal SVC call being issued by the driver (such as an SVC2 call).

PART II
INPUT/OUTPUT (I/O) SUBSYSTEM REFERENCE INFORMATION

CONTENTS

PREFACE

This document serves as a reference manual to the system programmer that plans to write an OS/32 device driver to interface a nonstandard input/output (I/O) device with OS/32. This manual covers only device drivers accessed by the supervisor call 1 (SVC1) I/O protocol. SVC15 access is supported by Integrated Telecommunications Access Method (ITAM) line drivers, is not covered in this manual.

This manual is divided into two parts; whereas Part I is a tutorial and Part II contains important background information on I/O subsystems. Chapter 1 is a discussion of the general philosophy of OS/32 drivers; i.e, the purpose of a driver and also contains a general outline of the required information and interfaces which one must understand in order to write a driver. Chapters 2 through 5 present a detailed discussion of those interfaces which are defined by Perkin-Elmer. Specifically, Chapter 2 describes the Perkin-Elmer Series 3200 Processor I/O architecture. Chapter 3 discusses various methods of I/O programming, giving sample code sequences for each. Chapter 4 describes the OS/32 I/O subsystem architecture. This chapter is based on the I/O subsystem as of the OS/32 R08.1 software release, but it is generally compatible with any revision of OS/32 since software release R06.2. Chapter 5 is a detailed discussion of the structure of an OS/32 driver. Sample code sequences for the various major components are given. Chapter 6 describes the steps that are necessary to create a custom driver and to include it into an OS/32 system, respectively. Chapter 7 describes the changes that have to be implemented in drivers being written for systems configured under an input/output processor (IOP), as opposed to drivers that are written solely for systems configured under the central processing unit (CPU). Appendix A describes all the parameters for data structures, such as the device control block (DCB) and the channel control block (CCB). A detailed description of machine states is given in Appendix B. Appendix C defines the OS/32 macros that are used by drivers and Appendix D defines the OS/32 driver routines. Appendix E shows a sample driver, with its corresponding DCB macro.

For information on the contents of all Perkin-Elmer 32-bit manuals, see the 32-Bit Systems User Documentation Summary.

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

This chapter introduces OS/32 device drivers. After discussing the general philosophy behind writing a driver; i.e., the purpose of a driver, an introduction to hardware and software architecture is provided.

1.2 PURPOSE OF A DRIVER

A device driver in an operating system acts as the control point for the unique physical or device-dependent characteristics of a peripheral device. The driver is responsible for converting input/output (I/O) requests made by a user program into physical instructions understood by the device. The driver, in turn, responds to interrupts from the device or device controller, and reports status back to the user program.

Instructions that are provided by the Perkin-Elmer Series 3200 Processors to control and communicate with I/O devices are privileged instructions, meaning that user programs cannot execute them. This is common practice in computer systems where instructions are used to communicate with peripheral devices. (In systems which use memory-mapped I/O, the memory cells that control devices are normally inaccessible to user programs, thereby yielding the same effect.) The main reason for the restriction on I/O control is to protect the system and the user from unintentional (or malicious) interference. The I/O drivers and supporting I/O subsystem provide the mechanisms for users to access standard devices and for new devices to be added to the system.

In general, programming of I/O devices is dictated by the physical characteristics of the device and the needs of the user. In addition, drivers for different devices are programmed quite differently. However, the typical user level programmer does not want to be concerned with these differences. From the programmer's viewpoint, writing a line to the printer should be no different from writing the same line to a CRT or a disk file. This concept is called device-independent programming. It is also the domain of the I/O subsystem and device drivers. Device-independent programming is not always possible, nor is it always desirable. For example, a gapless magnetic tape is a device whose performance characteristics preclude device-independent programming (due to the system overhead in handling individual reads and writes). The gapless tape driver provides a device-specific interface which allows a list of several buffers to be provided in a single read or write call.

For both the case of device-independent programming and that of a device-specific interface, the device driver is responsible for mapping the logical device interface as seen by the user level programmer into the physical device. The driver projects an interface which hides, modifies or passes through unchanged the idiosyncrasies of the physical device's programming characteristics.

1.3 BASIC INFORMATION

To successfully implement an OS/32 device driver, the systems programmer must understand several related interfaces. These interfaces, discussed briefly in the following sections and in greater detail in subsequent chapters, are:

- the Perkin-Elmer Series Processor 3200 I/O Architecture,
- the OS/32 I/O subsystem architecture,
- the programming attributes of the physical device, and
- the logical device projected by the driver.

1.3.1 The Perkin-Elmer Series 3200 Processor Input/Output (I/O) Architecture

The Perkin-Elmer Series 3200 Processor I/O Architecture is the hardware framework in which all I/O programming is implemented. It is detailed in all of the Perkin-Elmer Series 3200 Processor Reference Manuals. The I/O architecture consists of definitions of the following:

- device addressing,
- device command and status formats,
- I/O bus data paths,
- instructions to control and communicate with physical devices,
- data structures associated with interrupt processing, and
- interaction of processor status word with external and internal interrupts.

These definitions, and their impact on OS/32 drivers, are summarized in Chapter 2.

1.3.2 OS/32 Input/Output (I/O) Subsystem Architecture

The OS/32 I/O subsystem architecture is the software framework in which all device drivers execute. The user interface to the I/O subsystem is supervisor call 1 (SVC1), which is fully described in the OS/32 Supervisor Call (SVC) Reference Manual. Integrated Telecommunications Access Method (ITAM) drivers; i.e, network drivers such as Ethernet, Bisync and SDLC, use SVC15.

The device driver interface to the I/O subsystem defines the following:

- Data structures for controlling the interaction of the driver with the remainder of the system.
- The general structure of device drivers, including register conventions.
- System interface and utility routines and their register calling conventions.

The interface is the primary subject of this document.

1.3.3 Programming Attributes of Physical Devices

I/O peripheral devices are connected to a Perkin-Elmer Series 3200 Processor via a device interface or controller board. Each such controller has its own repertoire of supported commands and resulting status, and interrupt conditions. Also, correct operation of the device is often dependent upon a specific sequence of command and data transfers. Failure to observe this programming sequence will generally lead to the device failing to operate in the desired manner.

It is the responsibility of the driver programmer to understand all of the programming requirements of the particular device. This information is usually defined in the appropriate device/controller programming reference manual. Other than the standard command and status formats and the sample code sequences, this information is outside the scope of this manual.

1.3.4 Logical Devices Projected by a Driver

As discussed earlier in this chapter, one of the primary purposes of a driver is to project a simplified, often device-independent view of a physical device. This simplified view is the logical device to which this section refers. It is the user level (SVC1) programming interface to the device. Unless the driver is being designed to conform to an existing Perkin-Elmer interface (e.g., a new CRT or disk), the definition of the logical device interface is left to the driver writer.

The OS/32 I/O subsystem places some restrictions on this interface by the way in which it interprets certain fields of the SVC1 parameter block. Within these limits, the driver writers are at liberty to extend the interface in any direction which best accomplishes their objectives. It is the responsibility of the driver writer to document the logical device interface, including:

- Function codes and options
- Status returns
- Extensions to standard SVC1 parameter block
- Sequence of command occurrence
- Buffer alignment or format requirements

1.3.5 Including a Driver in the Operating System

OS/32 device drivers are included in the system during the system generation (sysgen) process. Sysgen/32 processes system configuration statements to produce a macro level source file. This file is assembled and linked with standard Perkin-Elmer supplied system modules and driver libraries to produce an operating system image.

Sysgen/32 and the sysgen procedures have provisions for including customized device drivers in the system. Specifically, the sysgen procedures search for a user driver macro library (USERDLIB.MLB) containing macro definitions for user-written device control block (DCB) structures, and a user driver object library (USERDLIB.LIB) containing bodies of user-written drivers in object format.

The procedure for including a driver in the system is described in Part I, Chapter 3.

CHAPTER 2

SERIES 3200 INPUT/OUTPUT (I/O) ARCHITECTURE

LIST OF FIGURES

LIST OF TABLES

CHAPTER 2

SERIES 3200 INPUT/OUTPUT (I/O) ARCHITECTURE

2.1 INTRODUCTION

This chapter describes the Perkin-Elmer Series 3200 Processor I/O Architecture. Specifically, it covers:

- The multiplexor (MUX) bus
- The interaction of the program status word (PSW) with the I/O system
- The Series 3200 Processor I/O instruction repertoire
- The Series 3200 Processor I/O interrupt structure

Figure 2-1 shows a detailed description of the hardware architecture on a Series 3200.

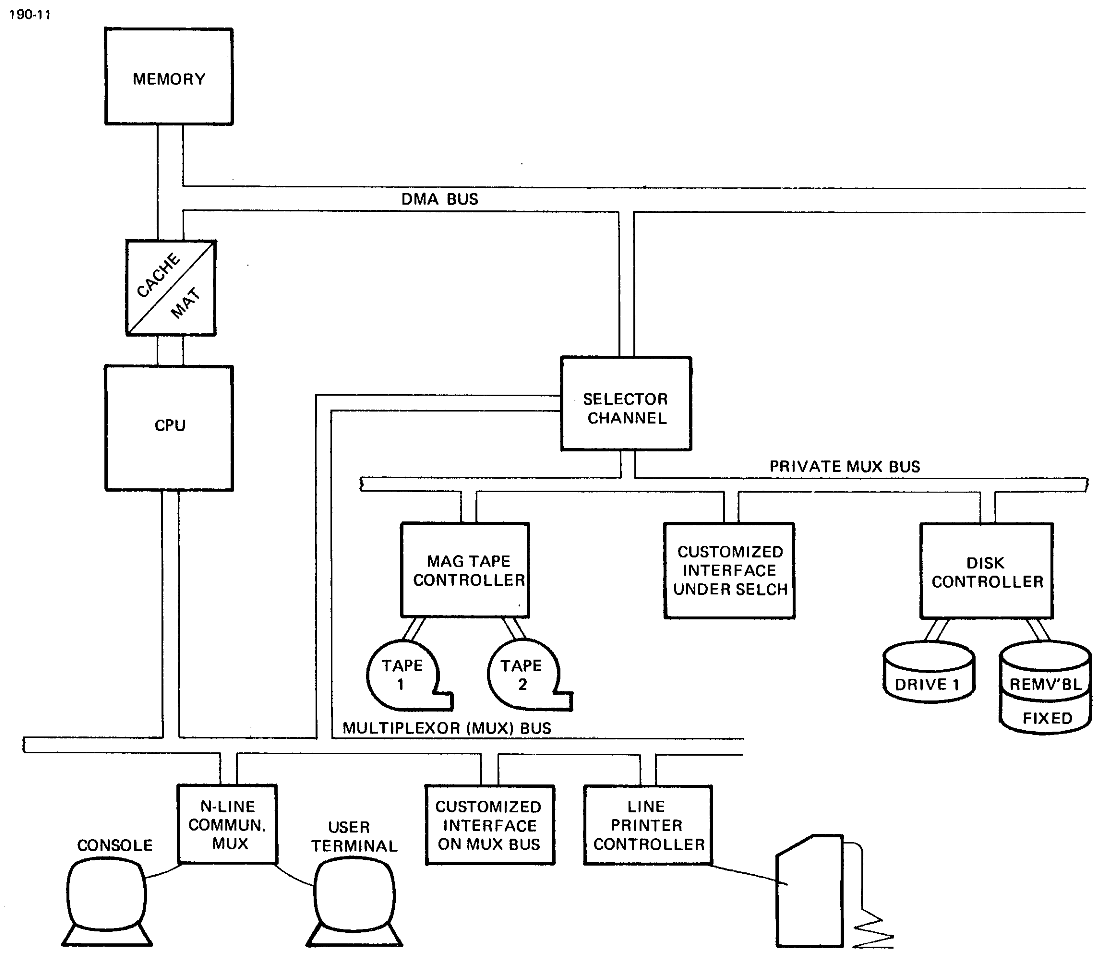

Figure 2-1. Hardware Configuration

2.2 MULTIPLEXOR (MUX BUS)

Control over peripheral devices and communication with them is accomplished via the MUX bus. Even direct memory access (DMA) devices are programmed (i.e., setup) and monitored via the MUX bus, and are designed to notify the processor of normal or error termination via the MUX bus.

The MUX bus gets its name from the fact that device addresses, device commands, device status and, of course, data are multiplexed on this bus. The bus consists of 16 bidirectional data lines and several control (from central processing unit (CPU) to controller) lines, and signal (from controller to CPU) lines. The control lines determine the interpretation (by the controller) of the contents of the data bus; i.e., device address, commands, status or actual data. The signal lines are used to synchronize the operation of the controller with the processor and to request service by the processor.

2.2.1 Device Address

Controllers on the MUX bus are addressed by a 10-bit device address, sometimes referred to as the device address. With 10 bits, it is possible to address 1,023 device addresses. In a Model 3260 System containing an input/output processor (IOP), with certain restrictions, this number can be exceeded. Device address 0 is reserved as the illegal device address.

It is generally not possible to connect 1,023 separate devices to a Series 3200 Processor. The reason for this is that many device controllers use up more than one device address. For example, a disk controller, capable of supporting four disk drives, uses five device addresses - one for each drive and one common address for the controller. Communications interfaces generally require two addresses per communications line - one for the receive side and one for the transmit side.

Each of the I/O instructions, discussed later in this chapter, operates on a device address. The device address, specified by a general-purpose register number that contains the address, is placed on the MUX bus data lines and the appropriate control signal is asserted. This causes the addressed device to become selected (and all other devices to become deselected). The processor then transfers commands, status or data either to or from the selected device as specified by the instruction.

NOTE

> No two device controllers on the bus can have the same address. Addresses on an IOP can be the same as on the CPU or another IOP, with restrictions on diagnostics.

2.2.2 Device Commands

The processor controls the state of a device controller using device commands. A command is an 8-bit datum that is passed to the (selected) controller via the MUX bus data lines. Commands are differentiated from data by a bus control line. An 8-bit command allows 256 unique commands to be specified. As commands are normally bit-encoded (i.e., bits are assigned specific meaning), the effective number of meaningful commands for a controller is usually much less than 255.

Some device controllers may require a larger command space than afforded by the standard 8-bit command. This can be done in one of several ways:

- the controller might use multiple device addresses, with commands to the different addresses interpreted differently, or
- the controller might interpret all data written to one or more device addresses as commands, and
- a specific command bit could condition the controller to accept one or more subsequent data bytes or halfwords (16 bits) as commands.

By convention, the first two bits of a command byte are assigned a standard meaning. Bits 0 and 1 control the device's interrupt flip-flop. Bit 0 is the DISABLE command and bit 1 is the ENABLE command. See Figure 2-2 for an illustration of these bits. (For reference purposes, Figure 2-3 shows the SELCH command.)

Bit	0	1	2	3	4	5	6	7
	D I S	E N A	X	X	X	X	X	X

Figure 2-2. Standard Device Command (SELCH not included)

Bit	8	9	10	11	12	13	14	15
	Not Used	Extended Address Read	Read	Go	Stop	SELCH Status	Not Used	Not Used

Figure 2-3. SELCH Command

If DISABLE is set and ENABLE reset (10_2), interrupts are disabled but queued at the interface. With DISABLE reset and ENABLE set (01_2), interrupts (including a queued interrupt) are allowed to be passed to the processor. If both DISABLE and ENABLE are set (11_2), interrupts are disarmed; interrupt conditions will be ignored by the controller. The combination of both DISABLE and ENABLE reset (00_2) has no effect on the state of the controller's interrupt logic. This is useful for commands that otherwise affect the state of the controller, without changing the interrupt state.

The remaining six bits of the command byte are device specific. Not all are necessarily defined for all controllers. The actual meaning of these bits should be documented in the appropriate device controller's programming reference manual. This information is mandatory for writing a driver for the device.

2.2.3 Device Status

The processor senses the state of a device controller by accessing an 8-bit device status. The status byte is requested from the (selected) device by asserting the appropriate MUX bus control line. The controller places the status on the bus data line. As with commands, status bytes are usually bit-encoded. Devices that require more than eight bits of status typically resort to the same types of mechanisms described in Section 2.2.2.

Certain bits of the device status are assigned default meaning by convention. Figure 2-4 illustrates default device status settings.

Bit	0	1	2	3	4	5	6	7
	X	X	X	X	BUSY	EX	X	DU

Figure 2-4. **Standard Device Status**

Bit 4 is interpreted as the busy (or not ready) bit; i.e., when set, the device is busy. For input devices, this typically means that no data is available at the interface. For output devices, it means that the controller is not ready to accept further data. If the device supports program detection of power-up condition, this is normally assigned to bit 7. If set, it indicates device unavailable - this is a much softer convention than the busy bit above.

Finally, bit 5 is usually defined as the logical OR of bits 0 through 3. Bit 5 is called the examine bit. The reason for this is as follows: when the processor performs a sense status operation, bits 4 through 7 of the status are copied into the PSW condition code. Conditional branch instructions are then able to test these bits directly. Bits 0 through 3 cannot be tested in a like manner. Therefore, the convention was developed that bit 5 would be set whenever any one or more of bits 0 through 3 are set. Thus, with a single branch instruction, one can test bit 5 (which is copied to the overflow condition code bit) and branch off to a routine to examine bits 0 through 3 one-by-one.

Except for the busy bit (bit 4), which is an almost universal convention and the examine bit, which is nearly so, the assignment of status bits is device specific. The required action response of the processor or driver to the status conditions is also device specific. This information should be documented in the appropriate device/controller programming reference manual, and is required to write a driver for the device.

2.2.4 Data

The MUX bus supports data transfers in units of 8 or 16 bits. This is called byte and halfword mode, respectively. The data transfer width is a characteristic of the physical device. Most Perkin-

Elmer supplied devices are byte-oriented. The exceptions to this tend to be process I/O devices such as analog-to-digital or digital-to-analog converters and digital I/O modules. Also, some newer tape and disk interfaces use halfword transfers for command and status information, as outlined in Section 2.2.2.

2.3 PROGRAM STATUS WORD (PSW) AND MACHINE STATES

The PSW is a 64-bit internal register that controls and records the current state of a Series 3200 Processor and consists of two 32-bit fields. One field, the location counter (LOC), always contains the address of the next instruction to be executed. The other field, shown in Table 2-1, contains processor status/control information. When an I/O device interrupts the processor, the current PSW/LOC is saved and a new PSW/LOC is derived based on information supplied by the interrupting device. Where the old PSW is saved and how the new one is derived are discussed in Section 2.5. Whether or not an interrupt is possible, is controlled by two bits in the PSW. Bits 17 and 20 control the I/O interrupts as shown in Table 2-1.

TABLE 2-1. PSW STATUS FIELD FOR PERKIN-ELMER SERIES 3200 PROCESSORS

BIT 17	BIT 20	EXPLANATION
0	0	All I/O Interrupts disabled
0	1	Interrupts enabled at higher priority levels
1	0	Interrupts enabled at all priority levels
1	1	Interrupts enabled at current and higher priority levels

When bit 17 is 0, the processor is considered to be in the interrupt service state. This is the most privileged state of the machine - the program executing in interrupt service state has complete control of the processer.

PSW bits 24 through 27 are closely related to the I/O architecture. These bits select one of eight register sets on the Series 3200 Processors (one of two sets on the Model 7/32 and some Model 8/32 Processors). Register sets 0 through 3 are tied to the four I/O interrupt priority levels. This is discussed further in Section 2.5.1, External Interrupts.

PSW bit 22 controls the system queue service (SQS) interrupt. SQS is a software or internal interrupt as described above. When bit 22 is enabled, the processor is considered to be in a reentrant or eventable state. When bit 22 is reset, the system is in a nonreentrant or, possibly, event service state.

Finally, PSW bits 28 through 31 contain the condition code. The condition code reflects the results of the most recent arithmetic or I/O operation. It is also setup on an interrupt as discussed in the section on interrupts.

The machine states associated with driver routines are the nonreentrant system state, the event service state and the interrupt service state. System states are described in detail in Appendix B.

2.4 INPUT/OUTPUT (I/O) INSTRUCTIONS

Perkin-Elmer Series 3200 Processors I/O instructions control the transfer of commands, status and data between the processor and the device controllers. They are defined in detail in any Perkin-Elmer Series 3200 Processor reference manual. An important point to note is that whenever any of the I/O instructions are issued, the processor waits for the addressed device to respond with a synchronization signal (called SYNC). To prevent the processor from waiting indefinitely (for a nonexistent or malfunctioning device), the instructions will time-out after a brief interval has elapsed. This condition is called a FALSE SYNC. The time interval varies from processor to processor, but it is on the order of 15-30μs When an I/O instruction times-out, the overflow bit in the PSW condition code is set. This is known as false sync condition.

2.4.1 Output Command (OC) Instructions

The OC instruction causes an 8-bit device command to be transferred to the addressed device. Like all I/O instructions, this command has two formats - a memory (RX) format, and a register (RR) format.

Example:

```
OC  R1,A(X1)        RX format
OCR R1,R2           RR format
```

In both formats, the R1 contains the device address in bits 22 through 31. In the RX format, the byte value in memory at A(X2) is the command byte to be transferred. In the RR format, the command byte is located in bits 24 through 31 of the register specified by R2. Note that this instruction does not have an immediate format; attempting to use an immediate format is a common mistake made in drivers. An output command is often the first instruction issued to a device in a driver. This is normally done to place the device in a known state. It is customary to test for FALSE SYNC immediately after this initial command.

Example:

```
  .
  .
  .
LHL R2,DCB.DN(RD)       Get device address
OC  R2,INITCMD          Issue initial command
BO  NONESUCH            FALSE SYNC - no such device
  .
  .
  .
```

2.4.2 Sense Status (SS)

The SS instruction requests an 8-bit device status byte to be transferred from the addressed device. This instruction has both an RX format and an RR format.

Example:

```
SS  R1,A(X2)    RX format
SSR R1,R2       RR format
```

In both formats, R1 contains the device address in bits 22 through 31. In the RX format, the status byte is returned to the memory byte specified by A(X2). In the RR format, the device status is returned to bits 24 through 31 of the register specified by R2. When an SS (or SSR) instruction is executed, the low-order four bits of the status are copied into the condition code bits of the PSW. While this makes normal tests (e.g., for busy) quite efficient, it makes testing for FALSE SYNC somewhat more complicated after a sense status. This is because the PSW overflow condition code may be set for reasons other than FALSE SYNC. In general, it is only necessary to test for FALSE SYNC once in a driver. It is recommended that this be done after an OC rather an SS instruction.

2.4.3 Write Data (WD)/Write Data Halfword (WH)

There are two types of write instructions. WD transfers an 8-bit data byte to the addressed device. WH transfers a 16-bit data halfword to the device. Both types of write instructions come in RX and RR formats.

Example:

```
WD  R1,A(X2) Write data RX format
WDR R1,R2    Write data RR format

WH  R1,A(X2) Write halfword RX format
WHR R1,R2    Write halfword RR format
```

In all these instructions, bits 22 through 31 of the register specified by R1 contain the device address. RX transfers the contents of the memory byte specified by A(X2). RR transfers bits 24 through 31 of the register specified by R2. WH RX transfers the contents of halfword specified by A(X2). This address must be halfword-aligned (i.e., even). The RR format of WH transfers the lower halfword (bits 16 through 31) of the register specified by R2. For all write instructions, the addressed device must support output and must not be busy. Otherwise, the results are undefined.

2.4.4 Read Data (RD)/Read Data Halfword (RH)

There are two types of read instructions. RD transfers an 8-bit data byte from the addressed device, and RH reads a 16-bit data halfword. Both types of read instructions have both RX and RR formats.

Example:

```
RD  R1,A(X2)   Read data RX format
RDR R1,R2      Read data RR format

RH  R1,A(X2)   Read halfword RX
RHR R1,R2      Read halfword RR
```

In all four instructions, the device address is contained in bits 22 through 31 of the register specified by R1. RX reads a data byte into the memory byte specified by A(X2). RDR reads the data byte into bits 24 through 31 of the register specified by R2. Bits 0 through 23 of register R2 are cleared to zeros. RH reads 16 bits of data into the memory halfword specified by A(X2). This address must be halfword-aligned (i.e., even). RHR reads the data halfword into bits 16 through 31 of register R2. Bits 0 through 15 of the register specified by R2 are cleared to zeros. For all of the read instructions, the addressed device must support input and must not be busy. Otherwise, the results are undefined.

2.4.5 Other Input/Output (I/O) Related Instructions

There are two additional instructions that are part of the I/O architecture that do not perform any I/O on the MUX bus. These are the simulate interrupt (SINT) and the simulate channel program simulate channel program instructions. The SINT instruction is an immediate (RI) format.

Example:

```
SINT R1,I(X2)   Simulate interrupt R1
SINT I(X2)      Simulate interrupt
```

The SINT instruction causes the processor to execute the same internal sequence as if an interrupt were received from the device whose address is specified by I(X2). The contents of R1, if present, specifies the priority level at which the interrupt is simulated. If R1 is not present, the processor defaults to level 0. The least significant ten bits of the second operand are presented to the interrupt handler as a device number. The device number is used to index to the interrupt service pointer table (ISPT), simulating an interrupt from an external device. Interrupts and priority levels are described in the next section.

The simulate channel program instruction has only an RX format.

Example:

```
SCP  R1,A(X2)  Simulate channel program
```

This instruction causes the processor to execute an auto driver channel control block (CCB) specified by the A(X2) operand. Data is moved between R1 and the memory buffer specified by the CCB. The auto driver channel is described in a subsequent section.

2.5 INTERRUPTS

The Series 3200 Processor I/O Architecture, in the context of OS/32 device drivers, supports two classes of interrupts. External interrupts are those interrupts that are generated by devices on the MUX bus. They occur asynchronously from instruction execution and, except for a few interruptible instructions, they are only recognized between instruction executions.

Internal interrupts, on the other hand, are generated by the Series 3200 Processor microprogram. They occur synchronously with instruction execution. The supervisor call (SVC) and system queue service (SQS) interrupts, described in a later section, is of particular importance to the OS/32 I/O subsystem and device drivers.

2.5.1 External Interrupts

All external interrupts occur via the MUX bus. A device controller requests service by asserting its attention signal. A controller may be connected to one of four attention lines. The four attention lines are arranged in four priority levels numbered 0 through 3. Level 0 is the highest priority level; 3 is the lowest. To be sensed, interrupts must by enabled. Interrupts are enabled by a combination of the PSW bits 17 and 20 and the currently selected register set (bits 24 through 27). See Section 2.4 for a discussion of the PSW bits 17 and 20.

When the processor recognizes an interrupt at level n, the following actions occur:

1. The processor acknowledges the interrupt by asserting a MUX bus control line. This causes the device to return its device address on the MUX bus data lines. Also, the interrupting device becomes selected.
2. The processor saves the current PSW and location counter (LOC) and generates a new PSW that selects the register set equal to the interrupting priority level. Higher level interrupts remain enabled (i.e., if the interrupting level is not level 0). The old PSW and location are saved in registers 0 and 1 of the selected set.
3. The processor saves the address of the interrupting device in register 2 of the new set. It then requests the current status of the selected (interrupting) device and places this status byte in bits 24 through 31 of register 3 of the new set. All other bits are zeroed.
4. The address of the channel command block is placed in register 4 of the selected register set (if a CCB is being used; see further discussions of the ISPT in the Chapter 3.
5. The LOC of the new PSW is determined from the contents of an external interrupt vector table called the ISPT. The ISPT is discussed in the next chapter.

2.5.1.1 Interrupt Service Pointer Table (ISPT)

The ISPT is a vector table for external service. For all Series 3200 Processors and for the Models 7/32 and 8/32, ISPT is a table of halfword (16-bit) pointers located at physical memory address X‘D0’. (It is sometimes referred to as the "Dog-Zero" Table.) ISPT is indexed by twice the device address of the interrupting device. Because each entry is only 16 bits wide, the address must be within the first 64kB of physical memory.

For the Series 3200 IOP associated with the Model 3260MPS system, the ISPT is a fullword table. It can be located anywhere in physical memory. It is located via a fullword pointer in the I/O processor block (IPB). Because it contains fullword entries, the addresses it contains can be anywhere in physical memory. The following paragraphs refer to ISPT entries and are equally valid for both the fullword IOP ISPT and the halfword CPU ISPT.

If the ISPT entry for a particular device is even (low-order bit = 0), then the entry is the address of the first instruction of an interrupt service routine (ISR). This address becomes the location counter (LOC) for the new PSW. (Immediate interrupt service is discussed in the next section.) If the ISPT entry for a device is odd (low-order bit = 1), then the entry is the address +1 of an auto driver

CCB. (Note: this is the normal case for some OS/32 device drivers. CCBs are only for common drivers; disk and tape drivers do not use them.) The CCB is accessed by the processor's microcode. Depending on the contents of the CCB, the interrupt might be serviced completely within microcode; or the LOC for the new PSW might be loaded from a field in the CCB. The auto driver channel is discussed in a later section.

Figure 2-5 shows the ISPT for a CPU with both immediate interrupts and CCBs. Note that ISPT is at X'D0' on all processors except IOPs. For IOPs ISPT may be anywhere.

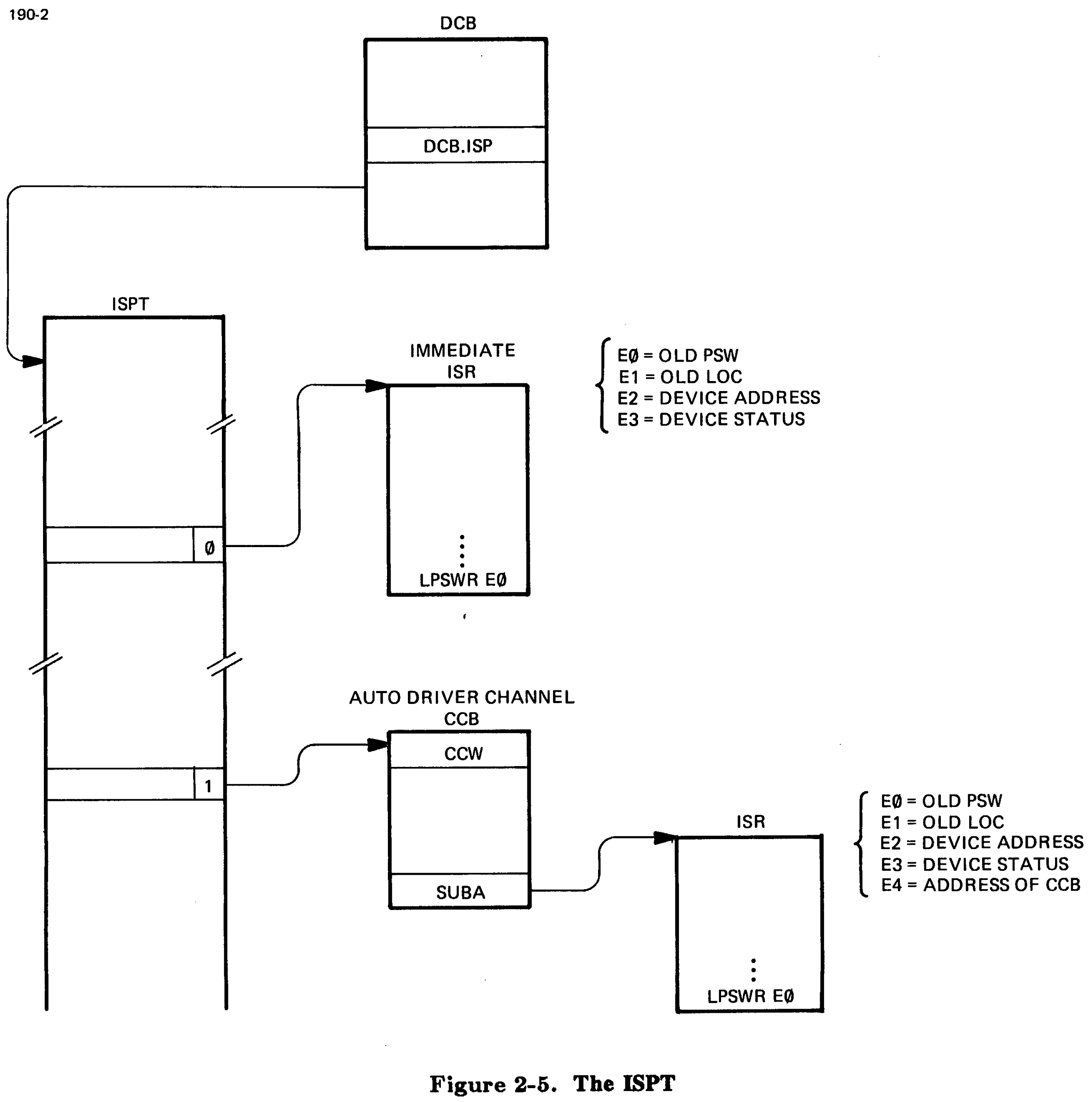

Figure 2-5. The ISPT

2.5.1.2 Immediate Interrupt Service

> **NOTE**
>
> Immediate interrupt service is generally not useful in the design of custom I/O drivers. All such drivers require the use of the channel command block and at least minimal auto driver channel operation. Immediate interrupts are used only in certain operating system time-keeping operations.

An immediate interrupt occurs if the ISPT entry for the interrupting device is even. Here the entry is the first address of an ISR. For all 3200 Processors, the first instruction of the ISR must be in the first 64kB of physical memory. (For an IOP, the ISR can be anywhere in physical memory.)

On entry to the ISR, the processor initializes registers 0 through 3 of the newly selected register set (= interrupt priority) as follows:

TABLE 2-2. REGISTER CONTENTS ON ENTRY TO ISR

REGISTER	CONTENTS ON ENTRY INTO ISR
0	Old PSW status (of interrupted program)
1	Old LOC
2	Interrupting device's address
3	Interrupting device's status

While the hardware imposes no such restriction, OS/32 requires that ISRs restrict their register usage to registers 0 through 7. This is necessary because the nonreentrant service state uses registers 8-F on set 0. If it can be guaranteed that the ISR will execute in register sets 1, 2 or 3, this restriction is not required. Failure to observe this restriction in an OS/32 driver will result in system failure. Also, registers 0 through 2 are usually preserved throughout an ISR. An ISR exits (back to the interrupted program) by loading the old PSW from registers 0 and 1, i.e.,

```
LPSWR  EO
```

NOTE

It is common practice to refer to registers from within an ISR as E*n*, where *n* is the register number. This is to make clear that the ISR is executing in an executive register set; i.e., 0, 1, 2 or 3.

2.5.1.3 Auto Driver Channel

The auto driver channel is a feature of all Perkin-Elmer Series 3200 Processors. It is, in effect, a built-in ISR. Under the control of the processor's microcode and as directed by the contents of a CCB, discussed below, the auto driver channel is capable of servicing a device interrupt completely within the processor's microcode.

The auto driver channel supports either byte or halfword devices. ISRs for halfword devices (RH(R)/WH(R)) must be supplied in assembly language.

The auto driver channel supports the following features:

- Read (or write) to/from a memory buffer with automatic entry into an ISR routine within the driver full/empty.
- Automatic device status checking on each interrupt with entry into the ISR if specified status bits are set.
- Optional automatic buffer switch on buffer full/empty with concurrent entry into ISR.
- Optional redundancy check generation (longitudinal or cyclic mod 12 or mod 16).
- Optional character set translation "on-the-fly" with special character recognition (e.g., in support of communications protocols). Recognition of special characters causes a specified ISR to be entered.
- Optional no execute mode: direct, unconditional entry into the specified ISR. (This is the mode most commonly used in I/O device drivers). Note that the PSW condition code reflects reason for entry into the ISR: no execute, buffer full/empty, switch, or status check.

The data structures associated with the auto driver channel are shown in Figure 2-6.

NOTE

CCB.DCB is set at sysgen time. CCB.XLT may be set at sysgen time or by driver. All other entries are set by driver.

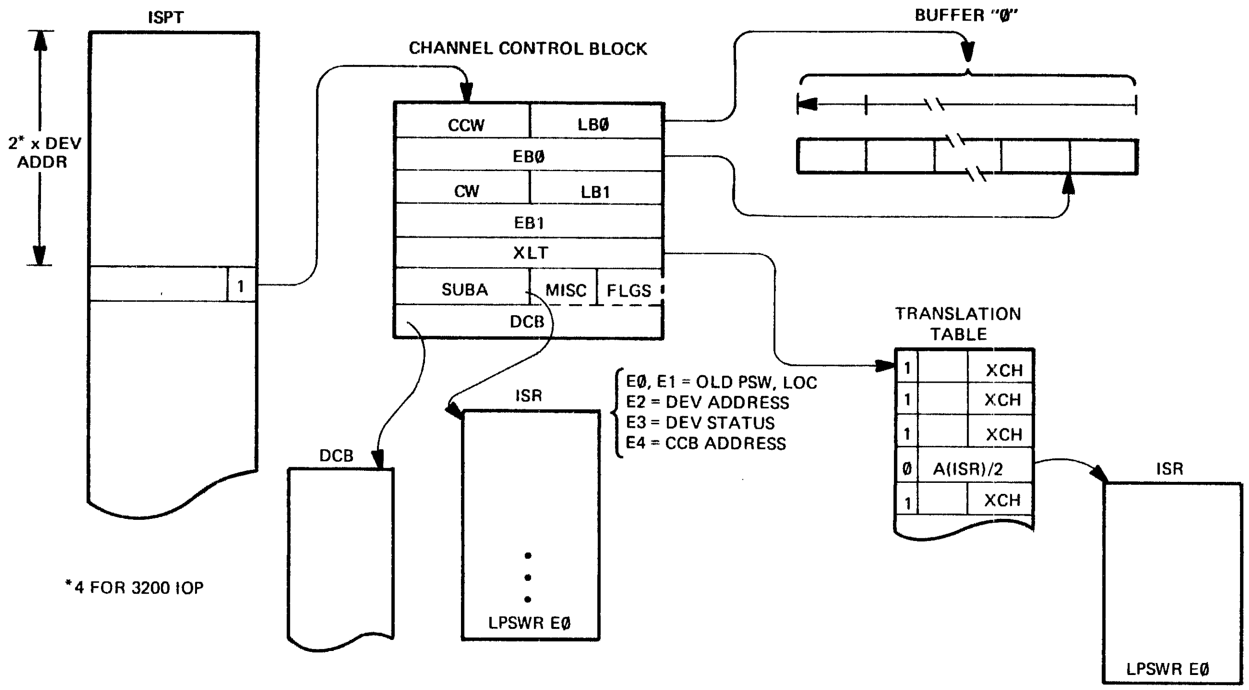

Figure 2-6. Auto Driver Control Structures

The auto driver channel is invoked by an odd entry (low-order bit set) in the ISPT. This entry is the address +1 of the CCB. The CCB is described in the next section.

2.5.1.3.1 Channel Control Block (CCB)

The CCB, as defined by the Series 3200 Processor I/O architecture, is 22 bytes long. (For the Series 3200 IOP, it is optionally 24 bytes long, thereby providing for a fullword ISR address.) The CCB contains the following fields:

- a channel control word (CCW), which specifies the desired operations and options,
- two pairs of buffer control fields: (LB0,EB0) for buffer 0 and (LB1,EB1) for buffer 1,
- the redundancy check accumulator (CW),
- the translation table Address (XLT),
- the ISR subroutine address (SUBA),
- the address of the associated device control block (DCB) (used for access into the OS/32 I/O subsystem by the ISRs.)

These fields are initialized by a device driver before starting the device or before issuing a simulate channel program instruction. The SCP instruction, described in Section 2.4, acts upon a specified CCB and simulates the actions that would occur for a device interrupt. But, instead of transferring data to/from a device, data is transferred between memory and the register specified in the simulate channel program instruction.

2.5.1.3.2 Channel Control Word (CCW)

The CCW is the operation code or function code of the CCB. The CCW specifies what operation is to be performed and with what options. The format of the CCW is shown in Figure 2-7.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Status Mask								E	X	RC		B	W	T	F

Figure 2-7. Channel Control Word

Where:

E (8)	1 = full autodriver channel control enabled 0 = always execute ISR indicated by CCB.SUBA; ignore bits 10-15
X (9)	1 = CCB.SUBA is a fullword (not fully supported yet) 0 = CCB.SUBA is a halfword (currently used by all drivers)
RC (10-11)	00 = LRC parity check only 01 = Bisync CRC 10 = not defined 11 = SDLC CRC
B (12)	0 = use buffer 0 1 = user buffer 1
W (13)	0 = READ mode 1 = WRITE mode
T (14)	0 = no translation 1 = translation enabled
F (15)	0 = normal mode 1 = "fast" mode

NOTE

Always use buffer 0; "no translation" must be used for halfword device, as well as no parity check.

The status mask of the CCW occupies bits 0 through 7. On every channel operation (i.e., each device interrupt with the CCW execute bit set), the device status is logically AND-tied with the status mask in the CCW. This is simply a test and does not alter the status mask or the device status. If the results of the test is nonzero, the channel operation is aborted and the ISR specified by the SUBA field is entered. The L bit of the new PSW condition code is set to indicate the reason for the ISR entry.

The CCW execute bit (CCWEX) is located at bit 8. CCWEX controls the operation of the channel. If the bit is 0, the channel enters the specified ISR SUBA directly. No other channel operation is executed, including the status check. The new PSW condition code is set to 0. If CCWEX is set, the channel operation defined by the remainder of the CCW is executed. This may or may not result in the ISR subroutine's being entered.

The extended ISR address bit 9 (CCWXISR) is defined only for the Series 3200 IOP associated with the Model 3260MPS. It indicates that the ISR (SUBA) field is a fullword rather than a halfword. For reasons of compatibility, this bit is not used by standard OS/32 drivers.

Two CCW bits (10 through 11) control the optional redundancy check generation. Three types of redundancy check are supported as listed in Table 2-3.

TABLE 2-3. REDUNDANCY CHECKS

RC BITS		RC TYPE
0	0	Longitudinal redundancy check (LRC)
0	1	BISYNC cyclic redundancy check (CRC12)
1	0	Reserved - illegal
1	1	SDLC cyclic redundancy check (CRC16)

Redundancy check sums are calculated and accumulated in the check word CCB field as each character is read or written. The details of the redundancy check calculation are given in the appropriate processor reference manuals. (Note that redundancy check generation is not performed if the CCW fast bit is set.)

The buffer select bit (CCWB1) is located at bit 12 and controls which of the two CCB buffers (0 or 1) is used by the channel. When a device interrupt causes a data transfer that results in a buffer overflow (full/empty), this bit is complemented, causing a buffer switch. At the same time, the specified ISR SUBA is entered with the new PSW condition code G bit set to indicate the reason for entry to the ISR. Upon the next interrupt after a buffer switch, the channel proceeds to use the alternate CCB buffer as specified by the complemented CCW buffer select bit. It is the responsibility of the driver software to ensure that the alternate buffer is setup correctly. (Note that buffer switching is not performed if the CCW fast bit is set. In that case, only buffer 0 is used.)

The CCW read/write bit (CCWRD/CCWWR) located at bit 13 controls the direction of data transfer. If reset (0), the channel performs a read operation; if set, a write is attempted. It is the responsibility of the device driver to ensure that the setting of this bit agrees with the current mode (input or output) of the device.

The CCW translation bit (CCWTL) at bit 14 enables optional character set translation and special character trapping. Translation and the translation table format are described in Section XX. (Note that translation is not performed if the CCW fast bit is set.)

The CCW fast bit (CCWFST), located at bit 15, disables the optional functions when set. These are buffer switching, translation and redundancy check generation. This provides for more efficient interrupt service (in terms of CPU time) for devices that do not require the optional functions. When the fast bit is set, the channel bypasses the checks for the optional function bits.

2.5.1.3.3 Channel Control Block (CCB) Buffers

The auto driver channel provides for two data buffers with optional automatic buffer switching. There are two fields in the CCB that control each buffer.

Buffer *n* (*n* = 0 or 1) is described to the channel by its end address (EB*n*) and its length (LB*n*). (The end address is defined as the address of the last byte of the buffer and must be odd for halfword transfers.) Because the end address is a fullword field, the buffer can be located anywhere in physical memory.

The buffer length field is defined as the buffer start address minus the end address. Thus, except for the case of a single byte transfer, the Buffer Length will be a negative value. For each data transfer, the channel increments the length field by one (for byte) or two (for halfword). When the value becomes positive, the last character has been transferred. Note that the sum of the end address and the buffer length yields the address of the next character to be transferred.

2.5.1.3.4 Check Word

The CCB check word is a two-byte field where the optional redundancy check is accumulated. It is normally initialized to 0 or -1 before transferring a buffer, depending on the communications protocol.

The check word is updated as each character is read or written. On output, the resulting check word is usually transmitted after the last data character. On input, it is customary to cease accumulation after the input termination character is recognized. Then the check word calculated by the sender is read and compared to that calculated by the channel. A difference generally indicates a data transmission error. (Check words are used almost exclusively in data communications drivers.)

2.5.1.3.5 Translation Table

The CCB translation table pointer is used only if the CCW translate bit is set (and the fast bit is reset). This CCB field contains the address of a table of halfword entries. This is illustrated in Figure 2-5. When translation is enabled, the channel accesses the table using twice the data byte as the index. If the high-order bit of the addressed entry is set, then the low-order byte contains the translated character (XCH in Figure 2-5).

If the high-order bit of the entry is reset, then the remaining 15 bits of the entry contain one-half the address of a special character handling routine. The entry is doubled and the ISR subroutine at the resulting address is entered exactly as if the CCW execute bit has been reset (i.e., new PSW condition code = 0).

The translation table can be used for character set translation (e.g., ASCII to EBCDIC, or vice versa); or for recognizing special characters of communications protocols.

2.5.1.3.6 Interrupt Service Routine (ISR) Subroutine Address (SUBA)

The CCB ISR (SUBA) field contains the address of an assembly language ISR that handles all error and normal termination conditions. In addition, this ISR is entered when the CCW execute bit is reset. For all 3200 Processors except the IOP, this is a halfword field. Thus, the first instruction of the ISR must be within the first 64kB of physical memory. For the IOP, this field may optionally be a fullword if the CCW extended ISR address bit is set.

Entry to the specified ISR is exactly the same for an immediate interrupt ISR except for the following two items:

1. The new PSW condition code indicates the reason for entry as shown in Table 2-4:
2. In addition to the contents of registers 0 through 3, which are the same as for an immediate interrupt, register 4 contains the address of the CCB itself. This property is most important to OS/32 drivers as it provides a dynamic link to a data structure associated with the interrupting device, allowing reentrant/sharable drivers to be written.

TABLE 2-4. PROGRAM STATUS WORD CONDITION CODES

Condition Code				Explanation
0	0	0	0	No Execute or Translation Table
0	0	0	1	Device Status Check
0	0	1	0	Buffer End

2.5.2 Internal Interrupts

The Perkin-Elmer Series 3200 Processors support a number of internal interrupts. Two of these internal interrupts are of particular interest in understanding the OS/32 I/O subsystem and device drivers. These are:

- the supervisor call (SVC) interrupt, and
- the system queue service (SQS) interrupt.

Other internal interrupts are associated with fault conditions, such as illegal instructions, address alignment faults or arithmetic faults. If any of these faults occur within a driver, the system will crash.

All internal interrupts occur synchronously with instruction execution; and they are serviced by a common method. A new PSW and LOC are loaded from a dedicated location in low memory. Each of the internal interrupts cause the PSW and LOC to be loaded from a different location. The old PSW and LOC are stored in registers 14 and 15 of the register set selected by the new PSW. Other registers (13, 12, etc.) are loaded with information dependent on the type of internal interrupt.

NOTE

On the Series 3200 IOP, the new PSW and LOC for internal interrupts are located at dedicated locations in the I/O processor block (IPB). Also, in the IOP, there is no SQS interrupt. The function of SQS is provided by the IOP synchronous interrupt service (SIS). These concepts and other differences between a Series 3200 Processor and the Series 3200 IOP are covered in Chapter 8.

2.5.2.1 Supervisor Call (SVC) Interrupt

The SVC interrupt is invoked by the SVC instruction. The SVC is the primary instrument for invoking operating system services. The SVC instruction selects one of 16 service routines (0 through 15) and specifies the address of a parameter block (or parameter list).

The new PSW for all 16 SVC service routines is loaded from the same dedicated memory location. The new LOC is loaded from a table of halfword service routine addresses, indexed by the SVC number (0 through 15). Because the table entries are only halfwords, the first instruction of service routines must reside in the first 64kB of physical memory. The new PSW and LOC values are set up by the operating system in control of the processor. Usually, the new PSW allows privileged instructions to be executed and selects a new register set so that the register set of the caller is not corrupted (e.g., by the PSW and LOC).

In addition to the old PSW and LOC which are saved in registers 14 and 15, the SVC internal interrupt loads the address of the SVC parameter block into register 13 of the new register set. Some SVCs may choose to interpret this address as an immediate value.

2.5.2.2 System Queue Service (SQS)

The SQS internal interrupt is the primary mechanism for synchronizing asynchronous events (such as termination of an I/O operation) with the more orderly sequence of events within an operating system. The system queue is a standard Perkin-Elmer circular list structure accessed by the Series 3200 Processor LIST instructions (ATL, ABL, RBL, RTL). The address of the system queue is stored in a dedicated memory location.

The system queue is examined by the processor whenever a new PSW that enables the SQS interrupt is loaded. If the queue is nonempty, an SQS internal interrupt occurs. A new PSW and LOC are loaded from a dedicated location and the old PSW and LOC are saved in registers 14 and

15 of the new register set. Register 13 is loaded with the address of the system queue. The SQS routine must remove and service entries from the system queue. A queue entry is the address of an event coordination leaf that contains the address of a service subroutine and argument values for that routine.

CHAPTER 3

INPUT/OUTPUT (I/O) PROGRAMMING METHODS

CHAPTER 3

INPUT/OUTPUT (I/O) PROGRAMMING METHODS

3.1 INTRODUCTION

Device drivers can transfer data between the Perkin-Elmer Series 3200 Processor/memory and I/O devices by one of two ways:

- Programmed I/O where the processor is involved in the transfer of each byte (or halfword) of data.
- Direct memory access (DMA) I/O where the processor sets up a selctor channel (SELCH) that performs byte (or halfword) transfers directly between memory and device controllers over the DMA bus.

The mode used is usually based on the hardware configuration and the transfer speed of the device. Faster devices such as (hard) disks and high performance magnetic tapes can only use DMA to achieve their maximum performance. Also, DMA devices tend to be block transfer devices rather than single character interaction devices.

Slower or more interactive devices, or devices whose only purpose is to generate interrupts (such as a periodic timer) generally use programmed I/O. Regardless of which (programmed I/O or DMA) mode of data transfer is used, device drivers can control and monitor the device (or SELCH) by one of two programming methods:

- Status-Loop Monitoring - the processor loops on device status. Data is transferred when the device is not busy. Transfer is terminated when all data is transferred or an error condition occurs. Most OS/32 device drivers should never use status loop I/O since it requires that the CPU be dedicated to the I/O operation. Status loop I/O is used only in stand alone routines such as diagnostics.
- Interrupt-Driven I/O - the processor sets up the interrupt service pointer table (ISPT) vector for the driver and enables device interrupts. The device interrupts the processor; when busy the processor stops and/or an error status is raised (device-specific). The interrupt service routine (ISR) transfers a byte/halfword of data if the device is not busy and no error has occurred. Transfer is terminated when all data is transferred or an error occurs.

The descriptions above specifically describe methods of monitoring programmed I/O. For DMA, the processor loops on SELCH status or waits for the SELCH to interrupt to indicate that the transfer is complete. Meanwhile, the SELCH is, itself, performing what is essentially a status monitoring loop on the device under its control.

3.2 PROGRAMMED INPUT/OUTPUT (I/O)

This section describes the basic sequences of programmed I/O for the Series 3200 Processor I/O architecture. Sample code sequences are presented and analyzed for status loop monitoring and interrupt-driver I/O. Both immediate interrupt service and auto driver channel are covered.

3.2.1 Status Loop Monitoring

Status loop monitoring is the most basic form of I/O programming. It is simplest to explain and to understand. It also forms the basis for both interrupt driven programmed I/O and DMA I/O.

While status loop monitoring is the simplest form of I/O, it requires that the processor be dedicated to the I/O programming during the entire data transfer or other I/O functions. For this reason, it is normally used only in stand-alone programs such as hardware test programs or operating system boot loaders. Device drivers in an operating system normally use interrupt-driven I/O (discussed in Section 3.2.2). Generally speaking, status loop monitoring is not appropriate for OS/32 I/O drivers.

There is one situation where an OS/32 device driver uses a status loop I/O rather than interrupt-driven I/O. This is the case where the device has a sufficiently high data transfer rate that is nearly always ready (i.e., not busy) from the processor's viewpoint. It is more efficient in terms of both central processing unit (CPU) use and elapsed time to perform the data transfer in an interruptable status loop rather under interrupt control. A high-speed buffered line printer with a parallel interface is one such device.

In the examples below, the following preconditions are assumed:

- Register zero (R0) is a scratch register.
- Register one (R1) contains a buffer start address.
- Register two (R2) contains the constant 1.
- Register three (R3) contains the buffer end address.
- Register six (R6) contains the device address.
- SETUPDEV is a memory location that contains a device command byte.
- FALSYNC and IODONE are external program labels.

Note that registers 1 through 3 are setup for a BXLE loop.

The first example illustrates a simple sense status loop to read a sequence of data bytes into a memory buffer until the buffer is full:

```
                .
                .
                .
                OC    R6,SETUPDEV  Command device
                BO    FALSYNC      False Sync: Device nonexistent
      STATLOOP  SSR   R6,R0        Sense device status info R0
                BCS   STATLOOP     Sets CC carry if busy
                RD    R6,0(R1)     Not busy: read a byte
                BXLE  R1,STATLOOP  Increment R1 and loop
                B     IODONE       until buffer is full
```

The sense status (SSR) instructions copy the low 4 bits of device status into the program status word (PSW) condition code. The branch on carry short (BCS) instruction tests the busy status that appears in the carry bit of the condition code. This example loops on busy, i.e., it branches back to the SSR instructions until busy drops.

Each time busy drops, the BCS falls through and the read data instruction reads a byte of data from the device and places it in memory of the address contained in R1. The BXLE instruction increments R1 (by the value in R2) and compares it to the value in R3 (end address). If the update buffer address in R1 is less than or equal to (LE) the end address, the BXLE branches back to loop on busy.

A bad status causes the status loop to be exited. At label EXAMINE, the device status in R0 is tested bit by bit. The symbols STAT.XXX, STAT.YYY are assumed to be equated to the various device status bits. Labels ERRXXX, ERRYYY are program labels that handle the named error conditions (XXX or YYY).

Often it is desirable to limit the number of times that the sense status loop is executed. This is called a limited sense status loop. It is used to provide a safety exit if busy does not drop within an expected time interval. The final example assumes that the symbol SSLIMIT is equated to the maximum number of times that the sense status loop should be executed:

```
                .
                .
                .
                LI  R4,SSLIMIT        Set up loop limit
      STATLOOP  SIS R4,1              Decrement loop limit
                BNP TIMEOUT           Exit when limit exhausted
                SSR R6,RO             Sense device status into RO
                BO  EXAMINE           Exit if errors
                BCS STATLOOP          Loop if busy
                .
                .
                .
```

In a limited sense status loop, one usually desires to loop for a specific time interval rather than a specific number of times. The loop limit is a simple way to terminate a loop after some time interval. The actual time interval is the product of the loop limit and the sum of the execution times of the instructions in the loop. This latter value varies from processor to processor so that a minimum value (for the fastest processor) is usually assumed.

3.2.2 Interrupt-Driven Programmed Input/Output (I/O)

Interrupt-driven I/O is the most frequently used I/O programming method in OS/32 device drivers. The use of interrupts allows the processor to be used for other tasks while the I/O is in progress, rather than waiting in a sense status loop. Interrupt programming is generally viewed as more complex than status loop monitoring because the driver relinquishes control of the processor while waiting for an interrupt. Thus, it must save its context (e.g., critical register contents) and reestablish the context when the interrupt occurs. Part of the context is automatically established by the processor: the device address and status, and (for auto driver channel operation) the CCB address is loaded into registers. Any other values required must be loaded by the interrupt service routine (ISR).

Interrupts may be serviced by ISRs by the auto driver channel. Immediate ISRs, entered directly from an ISPT vector, are not used frequently in OS/32 device drivers. Rather, all interrupt service is via the auto driver channel, even if only to pass control to the ISR (no execute mode). The reason for this is that the auto driver channel supplies the address of the CCB in register 4. This allows the operating system, by software convention, to extend the CCB to contain device specific state data and/or pointers to other structures managed by the I/O subsystem.

The following sections give examples of both immediate service and Auto Driver Channel operation. It will be noted that the immediate interrupt example, i.e, setup and service, is a model for Auto Driver Channel setup and interrupt service. The only difference between immediate ISRs and auto driver channel ISRs is the CCB addresses in register 4 supplied by the channel.

3.2.2.1 Immediate Interrupt Service

Immediate Interrupt programming, like all interrupt programming, consists of three phases: initialization, interrupt service and termination. Initialization involves initializing appropriate I/O data structures (specifically the ISPT) and preparing the device.

Interrupt service is handled by a routine whose address is placed in the ISPT by the initialization phase. Termination includes all processing after the interrupt phase has encountered a termination condition (all data transferred, device error, I/O halted). In OS/32, the termination phase must be scheduled by adding a structure to the system queue. This will be discussed further in the following chapters on the OS/32 I/O subsystem and device driver structures. The following is an example of an initialization and interrupt service phase using an immediate ISR. For the initialization phase, assume that the memory locations named CURBUFAD and ENDBUFAD, contain the start and end address, respectively, of a data input buffer. Further, assume that register R6 contains the device address. The ISPT and device might be set up as follows:

```
.
.
.
LA  RO,ISRODEVR       Address of device ISR 0
STH RO,ISPT(R6,R6)    Set ISR for interrupt
OC  R6,SETUPDEV       Command device - enable interrupts
B   WAITTERM          Wait for termination
```

The STH instruction places the ISR address into the ISPT at offset of twice the device address. On a 3200 IOP, the ISR address would be stored at a fullword offset (4 times the device address). A slightly different sequence of instructions must be used if the device driver must support devices under both the CPU and IOP. See Chapter 8 for more information concerning differences for writing a driver under an IOP. The OC instruction sends the command byte at location SETUPCMD that, presumably, enables interrupts. This example then branches off to WAITTERM, to wait for the interrupt service phase to schedule the termination phase.

The ISR is entered when the device requests attention. The following ISR reads a byte of data from the device into the buffer until the buffer is full:

```
          .
          .
          .
          PURE
ISRODEVR  THI    E3,STATMASK  Test for error status
          BNZ    ISROEXA      Examine status
          THI    E3,STAT.BSY  Device just busy?
          BNZ    ISROEXIT     Yes, ignore interrupt

          L      E6,CURBUFAD  Current buffer address
          RD     E2,0(E6)     Read data into buffer
          AIS    E6,1         Increment...
          ST     E6,CURBUFAD  and store updated buffer address
          CL     E6,ENDBUFAD  Check if buffer full
          BP     ISRTERM      Schedule termination
ISROEXIT  LPSWR  E0           Exit ISR
          .
          .
          .
```

This ISR is entered on a device interrupt with registers E0 through E3 initialized by the processor's microcode. Note that registers are referred to by mnemonics beginning with an "E." This is to to indicate that the ISR is executing in one of the Executive register sets.

Register E3 contains the device status, that the example ISR tests for interesting status (defined by symbol STATMASK) and for device busy. Error conditions will be handled by code off at label ISROEXA. If the device is still busy, the example exits by branching to ISROEXIT. This test may be necessary for some devices to ignore any spurious interrupts that might be queued when the device interrupts are enabled.

The next section of the ISR reads a byte of data from the device (address in E2, thanks to microcode) into memory at the current buffer location. The buffer address is incremented and compared to the end address. If the update buffer address is greater than the end address, termination phase is scheduled at ISRTERM (not shown). Otherwise, the ISR exits at ISR0EXIT.

The ISR exits by loading the old PSW and LOC placed in registers E0 and E1 by the microcode.

3.2.2.2 Auto Driver Channel Programming

The auto driver channel is a set of microcoded ISRs provided in the basic control store of the Series 3200 Processors. It can be controlled via a CCB to perform very simple or fairly complex I/O transfers. Each interrupt can be handled entirely within microcode until an error or normal termination condition forces entry into a channel ISR.

On the other hand, the channel operation can be completely bypassed. By resetting the Channel Control Word (CCW) Execute bit (CCWEX), all interrupts are passed directly to the ISR. This is called the no-execute mode. In this mode, the data transfer in the immediate interrupt example (in the preceding section) could be initialized as follows:

```
.
.
.
LA  RC,CCBDEV          Address of CCB
LIS RO,0
STH RO,CCB.CCW(RC)     Set no-execute CCW
LA  RO,ISRODEVR        Address of ISRO
STH RO,CCB.SUBA(RC)    Set ISR in CCB
AIS RC,1               Address of CCB+1
STH RC,ISPT(R6,R6)     Set CCB address in ISPT
OC  R6,SETUPCMD        Set up device - enable interrupts
.
.
.
```

This initialization routine set up only the CCW and the ISR address (SUBA) in the CCB. The address+1 of the CCB is stored in the ISPT and device interrupts are enabled. The same ISR as the immediate ISR example can be used to service this I/O. All interrupts will be passed directly to the ISR. The only difference is that, on each interrupt, register E4 will be preloaded with the address of the CCB. This might be useful, for example, to the routine that schedules termination. Also, the ISR could be improved to use buffer addresses with the CCB, rather than global symbols.

This first example illustrates how the auto driver channel can be used together with immediate ISRs simply to get the CCB address. The following example uses the auto driver channel in the fast mode to transfer a buffer of data from the device to memory. Assume the following register contents: R1 and R3 contain the buffer start and end addresses, respectively; R6 contains the device address and RC contains the CCB address:

```
.
.
.
LI   R0,CCWSTAT+CCWEX+CCWFST
STH R0,CCB.CCW(RC)      Set CCW= EXEC (READ) FAST
ST   R3,CCW.EB0(RC)     Set buffer to end
SR   R1,R3              Calculate negative length
STH R1,CCW.LB0(RC)      Set length in CCB
LA   R0,ISR1DEVR
STH R0,CCB.SUBA(RC)     Set ISR address
AIS RC,1                CCB address+1
STH RC,ISPT(R6,R6)      Set CCB address in ISPT
OC   R6,SETUPCMD        Set up device-enable interrupts
.
.
.
```

In addition to the CCW and ISR fields, the CCB buffer 0 parameters are initialized. The ISPT is then set up with the CCB address+1, and device interrupts are enabled.

As bytes of data arrive at the controller, busy drops and the device interrupts the processor. The auto driver channel receives control (as for the first example) because the ISPT entry is odd. The CCW specifies Execute, (Read) and Fast.

The channel fetches the device status and tests it against the CCW status mask. If any error bits are set, the ISR is entered with a condition code (<0) set. If no error status is raised, the channel accesses the buffer length field (LB0). If LB0 is positive, the ISR is entered with a condition code (>0) set. If LB0 is nonpositive, it is added to the end of buffer (EB0) to yield the current buffer address.

A data byte is read from the device and placed in memory at the current buffer address. The buffer length field is then incremented. The processor exits from the channel to continue processing suspended when the interrupt occurred. From the discussion above, it should be clear that the channel has handled the entire buffer read operation. The ISR is only entered to handle device errors and buffer full (i.e., channel termination). Aside from the actual error handling and termination code, the ISR is quite simple:

```
                .
                .
                .
ISRDEVR    BM     ISR1ERR  Device error
           BP     ISRTERM  Schedule termination
           LPSWR  EO       Ignore others
                .
                .
                .
```

The ISR branches directly on the condition code set by the channel. It should never (in this example) fall through to the LPSWR instruction, unless the CCW execute bit were explicitly reset.

More complex operations can be handled by the auto driver channel. For example, the initialization phase could be set up by CCB Buffer 1 parameters (LB1,EB1) in addition to buffer 0, and could reset the CCW fast bit. Then, when buffer 0 became full, the channel would automatically switch to bufferl by complementing the CCW buffer switch bit. At the same time the ISR is entered with a buffer full condition code.

By initializing the CCB check word field and selecting a redundancy checkword generation in the CCW, the channel can accumulate a check sum for the transfer. Finally, character set translation and special character trapping can be handled by setting up a translate table and enabling translation in the CCW.

3.3 DIRECT MEMORY ACCESS (DMA) PROGRAMMING

DMA programming transfers data directly between device and memory using the DMA channel. The DMA channel for Perkin-Elmer Series 3200 is called the Selector Channel (SELCH). DMA programming involves programming the SELCH to perform the data transfer, as well as setting up the device. Interrupt programming is used to service the SELCH terminations (errors or end of buffer).

3.3.1 The Selector Channel (SELCH)

The SELCH controls the data transfer between high speed peripheral devices and memory, without processor intervention. A SELCH can be connected to a maximum of 16 devices. The SELCH is a half-duplex, block transfer channel. It can transfer data between memory and a single selected device. Hence, the name selector channel. The advantage of using a SELCH is that, once initialized, the SELCH is completely self-supporting, and the processor is free to perform other tasks.

The SELCH is programmed via the multiplexor (MUX) bus and via the MUX bus, it supplies the status of the transfer. Both the SELCH and the device controllers, which are connected to it, have individual device addresses on the MUX bus.

The SELCH has two registers in which the current address and the end address of the data transfer will be stored. The starting address of the data transfer is the first current address. This must be done before the SELCH is started. Using a WRITE instruction, the device driver stores this information into these registers.

During the actual data transfer, the SELCH will automatically update the current address register, until it is equal to the end address. This indicates the end of the transfer and cause the SELCH to stop.

During the operation, the SELCH collects the data bytes or halfwords from the peripheral and transfers halfwords to memory. For this reason the start address must be on a halfword boundary and the end address must be odd.

After the SELCH has stopped, the controlling program should verify that the current address register contains the end address of the buffer. If this is not the case and if they were properly programmed, a device malfunction or an exceptional situation has occurred. For example, a disk cylinder was exhausted and the transfer had to be interrupted to enable repositioning of the heads. Depending upon the device, more information can be obtained by examining the status of the device or the SELCH.

There are two SELCH modes, the idle mode and the run mode. In idle mode, the private multiplexor bus (PMUX) is connected to the MUX bus. The SELCH generates its own PMUX to which device controllers such as disk and tape are connected. In the idle mode, the connected PMUX allows the CPU to address device controllers to set up for transfers. In the run mode, (i.e, the SELCH is monitoring the device and transferring data) the PMUX is disconnected from the MUX. The CPU is unable to access device controllers on the PMUX. The SELCH monitors the last device accessed, the selected device on the PMUX.

Drivers must always explicitly kill the SELCH via the output command before attempting to access devices under SELCH.

3.3.2 Programming the Selector Channel (SELCH)

The usual method of programming with the SELCH is the immediate interrupt. The first step in the operation is to check the status of the SELCH. If the SELCH is not busy, the address of the termination ISR is placed in the location in the ISPT for the SELCH device number. The program should then proceed as follows:

1. Give the SELCH a command to stop (i.e., kill the SELCH). This command initializes the SELCH registers and assures the idle condition of the PMUX connected to the I/O bus, so that the device may be set up for data transfer.

2. Write the buffer start and end addresses to the SELCH.

3. Prepare the device for the data transfer with the required commands.

4. Give the SELCH the command to start (READ or WRITE).

With the start command, the SELCH breaks the connection between its PMUX and the processor's I/O bus, and provides a direct path between memory and the last device addressed over its bus. When the device becomes ready, the channel starts the transfer, which proceeds to completion without further processor intervention. Once the start command has been given, the processor can proceed to the execution of concurrent instructions.

It is imporant to note that the driver must not be interrupted between Steps 3 and 4 described above. The reason for this is as follows:

- Step 3 causes the device to be selected on the PMUX bus. It should be left selected by being the last addressed device in Step 3.
- Step 4 causes the SELCH to do a status loop I/O transfer between memory and the selected device on its PMUX bus. The SELCH has no device address generation capability on the PMUX so that it selects the last device addressed by the CPU in Step 3.
- If an interrupt were to occur between Steps 3 and 4, the interrupting device would be selected and the device selected in Step 3 would be deselected. When the SELCH is started in Step 4, it would attempt to transfer to or from the wrong device.

Drivers should disable interrupts at all priorities by setting program status word (PSW) bits 17 and 20 to zero. This is true even within ISRs.

Upon termination, the SELCH signals to the processor that it requires service. The processor subsequently takes an immediate interrupt, transferring control to the SELCH ISR. At this time, registers 0 to 3 of the new register set are set as for any other immediate interrupt.

CHAPTER 4

THE INPUT/OUTPUT (I/O) SUBSYSTEM ARCHITECTURE

LIST OF FIGURES

LIST OF TABLES

CHAPTER 4

THE INPUT/OUTPUT (I/O) SUBSYSTEM ARCHITECTURE

4.1 INTRODUCTION

This chapter describes supervisor call1 (SVC1) instructions and connect/disconnect queues. It also gives a description of data structures connected with I/O subsystems.

4.2 SUPERVISOR CALL1 (SVC1) SYNOPSIS

The SVC1 internal interrupt handle is responsible for servicing task I/O requests. SVC1 runs as in the nonreentrant system state. That means that it executes in register set 0 with system queue service (SQS) interrupts disabled. nonreentrant system state routines are restricted to registers 8 through 15 of set 0.

An I/O request is encoded in an SVC1 parameter block described in the following section. The parameter block indicates the I/O function requested and contains a task-specific logical unit (lu) number indicating on which device (or file) the request is to be performed. The remainder of the parameter block contains additional arguments for the request and space for returned status and other information. SVC 1 has major functions which it performs. They are the following:

- Validation of the specified logical unit: is it within the task's range of valid logical units? Is it assigned to a device?
- Validation of the data buffer start and end addresses if the request involves data transfer.
- Acquisition and initialization of an Input/Output Block (IOB) if the request requires activation of the device driver.
- Validation that the request is legal for (i.e., supported by) the device.
- Invocation of a (possible device-specific) I/O handler (IOH) - a routine which performs or schedules the device driver to perform the I/O request.

All I/O requests are directed to an lu. Each task has from 1 to 255 logical units. An lu may be assigned to a device or file. If an lu is assigned to a device, the entry for that lu in the task's lu table contains a pointer to a data structure called a device control block (DCB). For a file, the lu table entry is the address of a file control block (FCB), which is very similar to a DCB. This manual is primarily concerned with device drivers.

Not all I/O requests actually perform data transfer. Some are command functions (e.g., REWIND), which access the device via the driver, but do not require a data buffer. Some requests do not even require entry into the driver - they are executed completely by the I/O supervisor. Examples of such requests are Wait Only, Test I/O Complete or Halt I/O.

Requests that require activation of the device driver are copied into a data structure called an IOB. IOB's are used to queue multiple I/O requests to devices that might be busy with a prior request. IOBs are discussed in Section 4.3.3.

SVC1 must validate that the requested function is legal for the device as currently assigned. That is, does the device support the requested function and does its current access privileges (e.g., Read/Write, Read Only, Write Only) permit the requested function. The Read/Write validation is performed explicitly by testing the function request against device attributes as modified by Access Privileges. These modified attributes are stored as part of the lu entry in the task's lu table. Validation of other functions (e.g., Rewind, Halt I/O, etc.) is performed implicitly when the I/O Handler is invoked.

The I/O supervisor is sometimes called the SVC1 skeleton. This is because it performs all of the actual I/O request processing, once all of the required validations have been performed, by calling a (possibly device-specific) IOH. The I/O handlers for a device driver are contained in a table called the IOH list. The IOH list is located by a pointer field in the DCB. Thus, each device type can have its own device-specific IOH. The IOH list contains one entry for each I/O request function (plus entries for system functions such as system initialization or power restore). For each supported function, the IOH list entry is the address of the SVC1 executor for that function. Entries for unsupported functions point to the SVC1 illegal function handler (SVC1FCER). The IOH list is described in Chapter 5.

After performing the requested function or scheduling the device driver to perform the request, SVC1 exits via the task dispatcher to continue task processing. See Table 4-1 for an illustration of the SVC1 parameter block.

TABLE 4-1. SVC1 PARAMETER BLOCK FORMAT AND CODING

0(0) Function code	1(1) lu	2(2) Device-independent status	3(3) Device-dependent status
4(4)	Buffer start address		
8(8)	Buffer end address		
12(C)	Random address		
16(10)	Length of data transfer		
20(14)	Extended options		

```
          SVC    1,parblk
          .
          .
          .
ALIGN 4
parblk    DB     X'function code'
          DB     X'lu'
          DS     2 bytes for status
          DC     A(buffer start)
          DC     A(buffer end)
          DC     4 bytes for random address
          DS     4 bytes for length of data transfer
          DC     Y'extended options'
```

Where:

- The function code is a 1-byte field indicating whether a request is a data transfer or a command function, and the specific operation to be performed.
- lu is a 1-byte field containing the lu currently assigned to the device to which an I/O request is directed.
- Device-independent status is a 1-byte field receiving the execution status of an I/O request after completion. The status received is not directly related to the type of data used.
- Device-dependent status is a 1-byte field receiving the execution status of an I/O request after completion. The status received contains information unique to the type of device used.

- Buffer start address is a 4-byte field used only for data transfer requests and must contain the starting address of the I/O buffer that receives or sends the data being transferred.
- Buffer end address is a 4-byte field used only for data transfer requests and must contain the ending address of the I/O buffer that receives or sends the data being transferred.
- Random address is a 4-byte field containing the address of the logical record to be accessed for a data transfer request; a legal binary number must be specified in this field if bit 5 of the function code is set to 1.
- Length of data transfer is a 4-byte field used only for data transfer requests. It receives the number of bytes actually transferred as a result of a data transfer request. If an error occurs during data transfer, this field is modified with indeterminate data.
- Extended options is a 4-byte field specifying device-dependent and device-independent extended functions that must be executed by the device when it is servicing a data transfer request.

4.2.1 Description of Special Supervisor Call 1 (SVC1) Functions

The following section gives a brief description of various special SVC1 functions.

4.2.1.1 Command/Data Transfer

Command/data transfer is bit 0 of the SVC1 function code (SVC1.CMDF) and determines whether the request is a data transfer (READ or WRITE) or command request. When SV1.CMDF is set (1), the function is a command request. Otherwise, it is a data transfer request.

4.2.1.2 Read/Write

Bit 1 of the function code is the read (SV1.READ) bit. Bit 2 is the write (SV1.WRIT) bit. If both are set, the function is formally a TEST2SET used to READ-MODIFY-WRITE a lock bit in a file record/data block. A driver can interpret this code as anything it desires, e.g., "read after write."

4.2.1.3 Wait Read/Write

When the function code wait bit (bit 4) is set, the calling task will be placed in an I/O wait state until the request has been completed. When the wait bit is reset, the request is a proceed request.

4.2.1.4 Proceed Read/Write

If the device is not in use, this function causes an I/O request to be initiated and control returned to the calling task immediately. If the device is busy, the request is enqueued unless an unconditional proceed is requested. Control is then returned to the calling task. When the total number of enqueued I/O requests exceeds the maximum number of IOBs established at link time, the calling task is placed in an IOB wait state. The task is awakened when one of its enqueued requests gains the connection and an IOB is released.

4.2.1.5 Unconditional Proceed Read/Write

This function behaves in exactly the same manner as the PROCEED READ/WRITE function, except when the maximum number of enqueued I/O requests are reached. Exactly *n* requests can be enqueued at any moment in time.

The condition code is set to 0 when the request gains the connection or when it is successfully enqueued. When the maximum number is exceeded, the request would not be enqueued. In this case, the error status is set to X'F'. A condition code X'0' implies either the request gains the connection or is successfully enqueued.

4.2.1.6 Wait-Only

This function is interpreted as a wait for the completion of all pending I/O requests to the specified lu of the calling task. A pending I/O request implies either an ongoing I/O or an enqueued request.

4.2.1.7 Test Input/Output (I/O) Completion

This function returns a condition code of X'F' if any request is pending on the specified lu of the calling task. If not, the condition code is X'0'.

4.2.1.8 Halt Input/Output (I/O)

This function aborts all outstanding requests issued to the specified lu of the calling task.

4.2.1.9 Test and Set

The test and set function provides a synchronization mechanism for concurrent accesses on the same file by different tasks. It allows a task to access a record within a file and, at the same time, mark it as being in use. The first bit in the record is reserved for this purpose. If it contains a 0, the record is not currently in use. If it contains a 1, the record is by convention in use. Readers interested in specific details are referred to OS/32 MTM Programming Reference Manual. This function is implemented within all disk drivers and is therefore supported both for contiguous and indexed files as well as bare disk accesses.

4.3 THE DEVICE CONTROL BLOCK (DCB)

For every device specified there is a DCB. The DCB is the data representation of a logical device within the OS/32 I/O subsystem. A file control block (FCB), which is a variant of a DCB, represents a file and is a type of logical device. Tasks open devices (or files) by placing the address of the DCB (or FCB) in the task's lu table (along with the read/write access privileges granted). A DCB consists of two parts:

- The device-independent part of the DCB is defined by the I/O Subsystem Architecture. It contains fields used by SVC1 and SQS, to schedule the driver and to communicate parameters and status between the driver, the OS and the task.

- The device-dependent part of the DCB is defined by the device driver. It contains internal driver state information, and possibly, trace/debug information.

The field's layout for the device-independent DCB and for the standard device-dependent DCB is given in Appendix A.

The device-independent part of the DCB is analagous to a task's task control block (TCB). It is the per device system data structure. As such, it contains data of interest to SVC1 and SQS, for example:

- the address of the physical device's event coordination leaf (see Section 4.5),
- the address of the IOH (discussed in Chapter 5) for this device,
- I/O request queueing strategy routine address should this device require special request queueing,
- driver initialization entry points for data transfer and command requests and the driver termination entry point,
- the optional address of a special I/O completion handler,
- device time-out chain link field and time-out value
- an embedded IOB (see Section 4.3.3) containing the "connected" I/O request, if any.

There are other fields in the DCB used by SVC1 for systems that contain Series 3200 I/O Processors (IOPs). These fields contain the IOP number to which the address is attached (0 for the CPU) and the address of the system queue for this device.

The device-dependent part of the DCB is analogous to a task's impure data area. All device drivers are written to be re-entrant (so that there need be only one copy of the driver for many devices). Thus, the driver stores all state information and possibly, debug or trace data in the device-dependent DCB.

A detailed description of the DCB can be found in Appendix A.

4.3.1 The Device Mnemonic Table

All devices and device volumes with direct access are specified in the system and by the user programs, either by showing a logical device number, or by showing a unique 4-character mnemonic. In order to relate this mnemonic to a device, the system uses a list of these devices, together with the device's DCB. The Device Mnemonic Table lists the relation between device mnemonic and the address of its DCB.

TABLE 4-2. DEVICE/VOLUME MNEMONIC TABLE

DEVICE MNEMONIC TABLE
DEVICE MNEMONIC (ASCII)
ADDRESS OF RELATED DCB
DEVICE MNEMONIC (ASCII)
ADDRESS OF RELATED DCB
DEVICE MNEMONIC (ASCII)
ADDRESS OF RELATED DCB
Y'00000000'
Y'00000000'

Figure 4-1 shows relationships of system pointer table (SPT), device mnemonic table (DMT), volume mnemonic table (VMT) and DCBs.

NOTE

The SPT, DMT, and VMT are normally not referenced by I/O drivers.

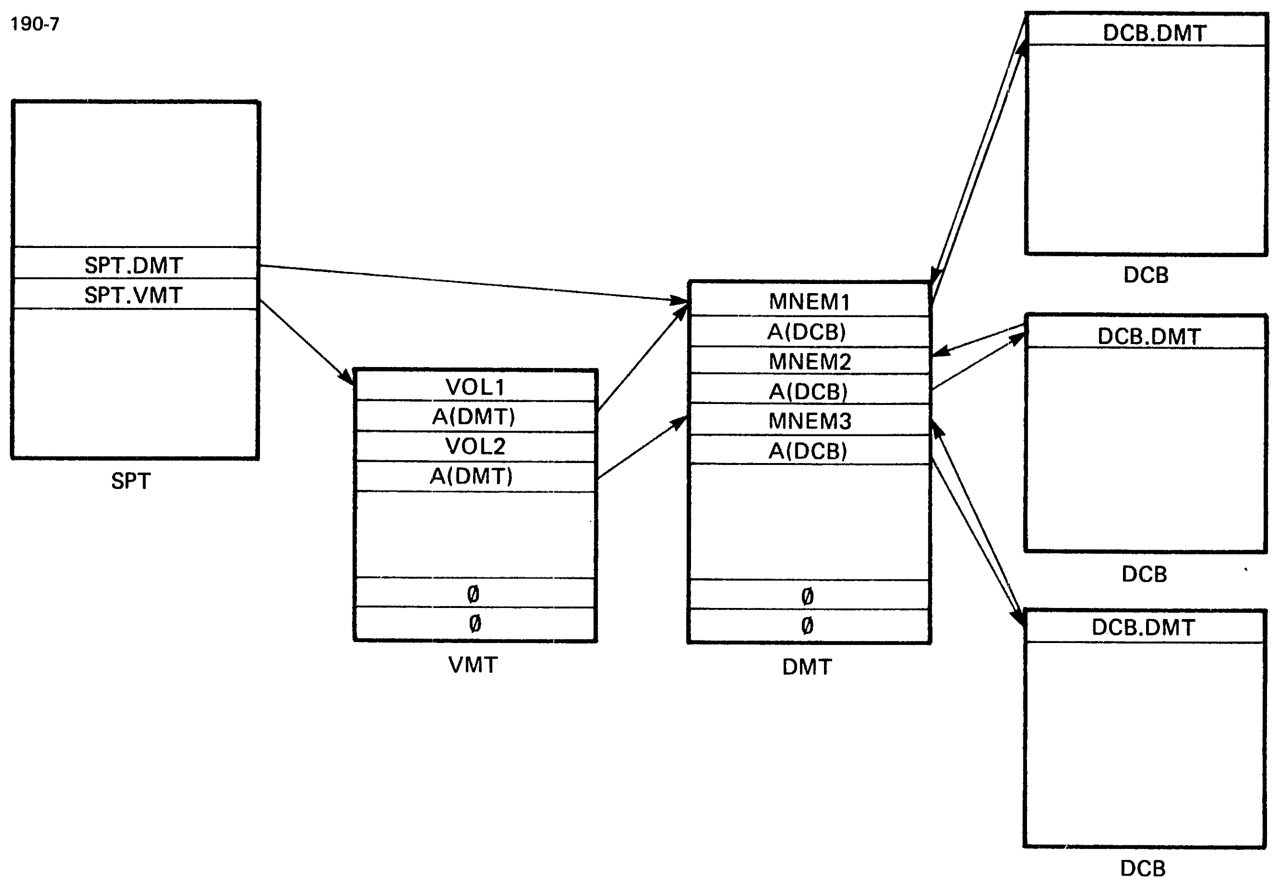

Figure 4-1. Device and Volume Mnemonic Tables Address Linking of DCBs

4.3.2 Device-Independent Status Values

Logical units provide device-independent I/O by causing all I/O requests to be made directly to the lu and not to the device. The execution status of an I/O request that is independent of the physical characteristics of the device being used is returned to the device-independent status field of the parameter block (see Table 4-3). The data remaining in this field from a previous I/O request is not modified until a subsequent I/O is completed or an error occurs.

TABLE 4-3. DEVICE-INDEPENDENT STATUS CODES

STATUS CODE	MEANING
X'C0'	Illegal function - an error is present in the function code; the requested function is not supported by the device or assigned access privilege or the buffer transfer is too small. When using tape, minimum buffer size is four bytes.
X'A0'	Device unavailable - the device is either inoperative or not configured into the system.
X'90'	End of medium (EOM) - The I/O directed to the lu reached the physical end of the device, e.g., end of tape. During magnetic tape operations, this status can be combined one of the next three status codes, yielding X'98', X'94', and X'92'.
X'88'	End of file (EOF) - the logical end of file specified by the assigned lu was reached.
X'84'	Parity - an even or odd parity error occurred on a data transfer request. Recoverable error - the I/O request is recoverable and can be retried. A write request was issued to a write-protected device. No I/O currently being processed - if a halt I/O is requested is executed, this bit is set, indicating that no I/O is being processed at this time.
X'81'	Illegal or unassigned lu - the lu specified in the parameter block is either incorrect or was not previously assigned.
X'00'	Normal execution or successful I/O is completed, and no error occurred.

The origin of the status, i.e., whether it came from SVC1 or the driver, varies:

X'C0'	Originates from SVC1 if attributes do not match function (READ/WRITE), or if IOH list does not have entry for the function. Originates possibly from driver itself after further checks (e.g., buffer alignment).
X'A0'	Originates from the driver either because false sync received (no such device) or device status indicates off-line/powered down.

X'90'	Originates from driver.
X'88'	Originates from driver.
X'84'	Originates from driver.
X'82'	Originates from driver or from SVC1 (e.g., for halted queued I/O requests.)
X'81'	Originates from SVC1 if lu illegal or unassigned.

There are several instances where the device-dependent status field contains what is actually device-independent status. This is always used in conjunction with device-independent status X'82'. The possible status codes are given in Table 4-4.

For example, when a device times out, the driver, by convention, usually returns status X'8282'. A driver might return a different value if, for example, there are several possible time-out conditions which the driver wants to signal (e.g., time-out waiting for seek complete, time-out during data transfer, time-out waiting for protocol ACK, etc.). Another example is when an I/O is halted. The driver normally returns status X'8281' (as shown in Table 4-4).

TABLE 4-4. DEVICE-DEPENDENT STATUS CODES

STATUS CODE	MEANING
X'85'	Exhausted retries on seeks - seeks on disk devices have been retried the maximum number of times.
X'84'	Queued I/O terminated - a queued I/O request is terminated because a previous I/O request failed.
X'83'	Device is write-protected - a write operation to a write-protected device occurred.
X'82'	Read/write time-out - a read or write time-out condition occurred.
X'00'	Normal execution - I/O was completed and no error occurred.

4.3.3 The Input/Output Block (IOB)

The (IOB) is a data structure that represents an I/O request within the I/O subsystem. An IOB is only required for those requests that will involve the device driver in accessing the device. It contains all of the information required by the I/O subsystem to queue and process the request. This information includes:

- A copy of the SVC1 parameter block,
- Both the unrelocated (physical) and relocated addresses of the SVC1 parameter block. The latter is used in forming the task Q entry for proceed I/O requests.
- The address of the TCB of the task requesting the I/O.
- The address of the DCB for the device.
- The driver entry point to be used when a device event is scheduled for this request.
- Request dependent flags that specify various optional processing for this request.
- Link fields for queueing the I/O request.

IOBs are maintained in a free pool for each task. The number of IOBs a task will have is specified when the task is linked. The default is one in order to ensure that every task has at least one IOB. IOBs are allocated within system space at task load time.

SVC1 gets an IOB from the requesting task's free pool by calling the routine GETIOB. IOBs are released to the free pool by calling RELIOB after the request is connected to the device (leaf), and the contents of the IOB have been copied into the DCB's embedded IOB. If it is necessary to queue the request and if the request resides in the task's last or only IOB, the task is placed in "IOB Wait." It cannot proceed (i.e., exit from SVC1) until at least one queued request has been connected and the IOB released. Because of this convention, GETIOB will always find at least one IOB in the free pool when called from SVC1.

The IOB event service routine (ESR) field is initialized by SVC1 to contain one of the two driver initialization routine entry points (for data transfer or command functions). This field governs the entry to the initialization phase of the driver when the IOB is copied into the DCB's embedded IOB at connection time. This same field in the embedded IOB is modified by routines EVMOD or DIRDONE to contain the address of intermediate or termination ESRs.

The "request dependent flags" field is used to specify which optional processing of the request is to be performed. This includes:

- removing IOB wait on device connection,
- removing I/O wait or adding a parameter to task Q on I/O completion,
- overriding reset of Interrupt Service Pointer Table entry on I/O completion, and others.

A detailed description of the elements in the IOB can be found in Appendix A.

4.3.4 SUPERVISOR CALL 1 (SVC1) Exits

Except for memory faults, SVC1 exits to the task dispatcher when it has completed IOH processing. As a result of the IOH, the disposition of the request will be one of the following:

- connected to the device and driver initialization scheduled,
- queued to the device, controller or selector channel (SELCH),

- aborted because of an error, or
- completed within the IOH routine.

Depending on the disposition of the request, different post-processing will be required, including:

- releasing an IOB if one was gotten and it has not been queued,
- placing the task in IOB or I/O wait, if necessary, and
- adding a trap item to the task queue.

Several different exit routines are provided to perform this post-processing. These routines are made entry points so that custom I/O can use them. Most of the standard IOHs exit to one of these routines. Some IOH exit directly to the task dispatcher.

4.3.4.1 Normal Exits

The following exit routines are normal (nonerror) exits or common routines used for both normal and error exits:

- SVC1EXIT - normal exit for driver requests (data transfer or commands). This routine sets IOB or I/O wait as appropriate and remembers in the IOB flags (in the DCB's embedded IOB) whether to move I/O wait or add an I/O completion trap to the task's queue. It exits to the task dispatcher.
- SVC1NOOP - IOH exit to ignore the function (i.e, no operation). This routine is normally referenced directly from the IOH list for functions that are to be ignored. It returns a zero status to the SVC1 parameter block and exits to one of the following common exit routines.
- SVC1NINA - (No IOB, no ADD to task queue) exits directly to the task dispatcher.
- SVC1INA - (IOB, no ADD to task queue) adds task trap item (I/O completion reason code plus SVC1 parameter block address) to the task's queue, then exits via SVC1NINA.
- SVC1IA - (Both IOB and ADD to task queue) releases IOB and exits via SVC1NIA to add to the task queue.

4.3.4.2 Error Exits

The SVC1 error exits are:

1. SVC1FCER - Illegal Function Code. An illegal function code status (SV1E.IFC = X'C000') is returned to the task's SVC1 parameter block. This routine exits to one of the SVC1 (N)I(N)A routines depending on whether an IOB needs to be released or a task queue trap is required.
2. SVC1ADER - Address error. This routine exits to MEMFAULT, the memory fault (alignment, write protected, not present).

4.4 SYSTEM QUEUE SERVICE (SQS)

The SQS internal interrupt handler is the front door into the operating system for all external events such as I/O completions, timer events, and system power restore. Such events are signaled by adding an entry to the system queue.

The system queue entries are the addresses of device structures called leaves. For I/O events, the leaves are part of the event coordination tree discussed in Section 4.5. For other events such as timer events and and power restore, special leaves are created solely for the purpose of entering the SQS or event service state. These special leaves, which are not really part of the I/O subsystem, are described in Section 4.5.1.

Each leaf contains the address of an SQS executor in the field EVN.SQS. After removing a leaf address from the system queue, SQS fetches and branches to the address of the SQS executor contained in the leaf. For I/O events, one of two executors (SQS.SLV or SQS.MLV) is invoked. These executors are described in Section 4.5.3.

Ultimately, the SQS executor will enter a device driver in one of three event service routine (ESR) states:

1. The driver initialization ESR to initiate a device operation. There are two possible initialization ESRs for drivers: one for data transfers (DCB.INIT) and command functions (DCB.FUNC). This ESR is scheduled by SVC1 if the device is not busy, or by the IODONE handler of a prior I/O.
2. An intermediate ESR, such as a seek, complete on a disk. This ESR is scheduled by a device interrupt service routine (ISR). The actual ESR entry point is selected by a driver by calling the routine EVMOD.
3. The driver termination ESR. This ESR is scheduled by a device ISR, or possibly, the system timeout routine. The termination ESR (DCB.TERM) is automatically selected when the Initialization ESR exits via DIRDONE (see Section 4.5.5).

4.5 The Event Tree (EVT)

The event tree is a tree-like structure built out of the combination of device leaves and controller, SELCH and EDMA nodes. The tree mirrors the hardware configuration of the system (selector channel, device control, devices). The tree is inverted with its leaves at the bottom and its root at the top. The tree is always processed upwards from the leaf. All of the resources necessary to perform I/O, such as a SELCH or controller, shared by more than one device, must be acquired for use by a driver. The EVT handles the allocation of these shared resources. Figure 4-2 gives a detailed description of the event coordination tree. Note the following in reference to Figure 4-2:

```
EVN.DCB = 0 ==> LEAF/NODE free.
EVN.DCB = A(DCB) ==> LEAF/NODE connected.
When LEAF is connected, DCB.IOB contains IOB of current I/O.
```

190-8

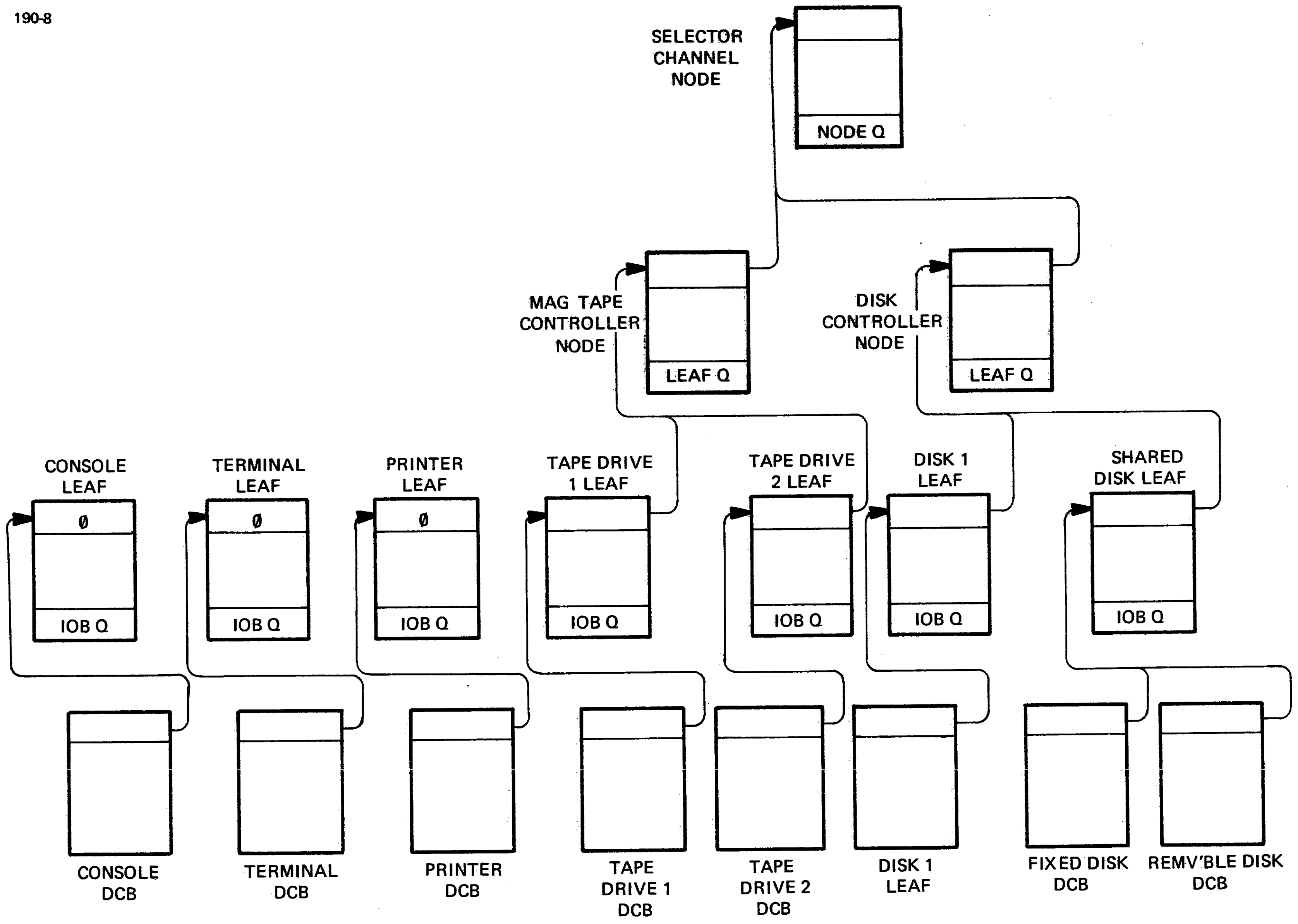

Figure 4-2. The Event Coordination Tree

For example, an I/O request for a disk device starts at the IOB, then passes to the leaf, the disk controller node, the SELCH node and possibly to the optional EDMA or super node. The structure of each leaf (lowest node) or other node is discussed in Appendix A.

It is important to note that the event coordinating structure contains two waiting queues, one for leaves and another for IOBs. IOBs are queued to leaves, thus representing I/O requests to a busy device. Leaves and nodes are queued to higher level nodes if the node is busy servicing another request.

Figure 4-3 depicts the queueing of I/O requests to a device. The structures depicted in Figure 4-3 are controlled by the operating system I/O executive routines. They are not accessed by I/O drivers. Figure 4-4 depicts the queueing of I/O requests to a disk device.

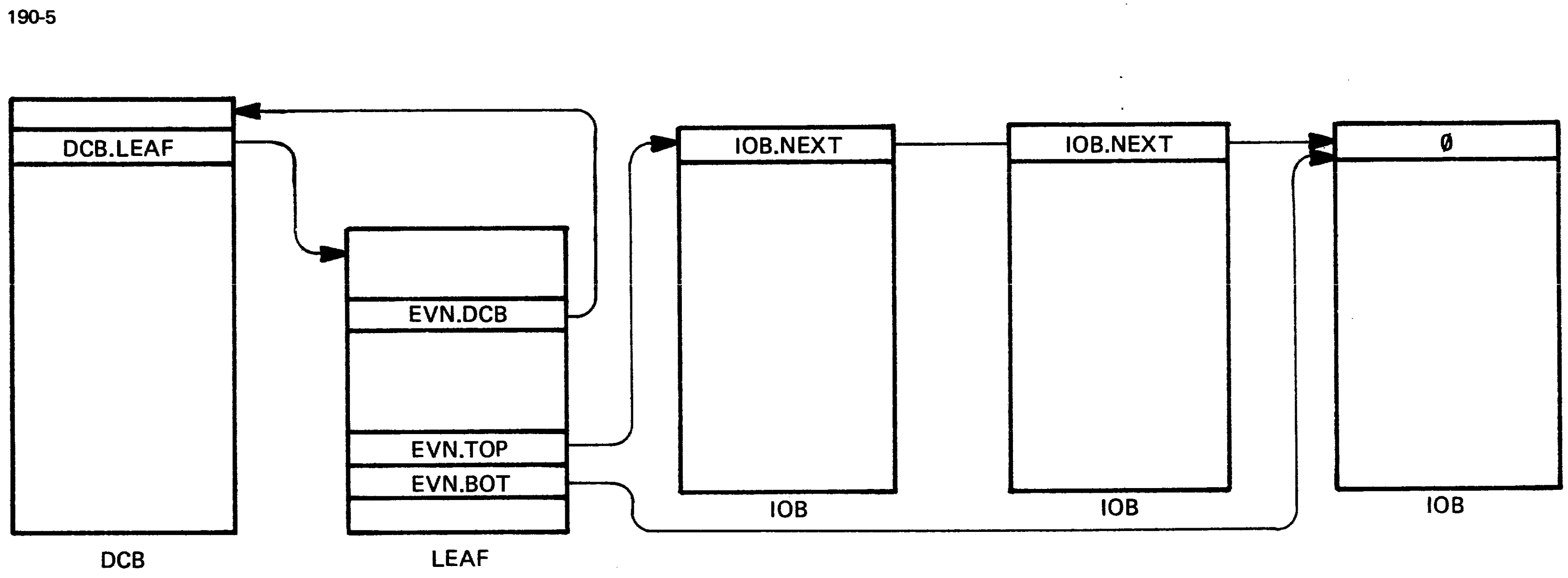

Figure 4-3. Queueing of I/O Requests to a Device

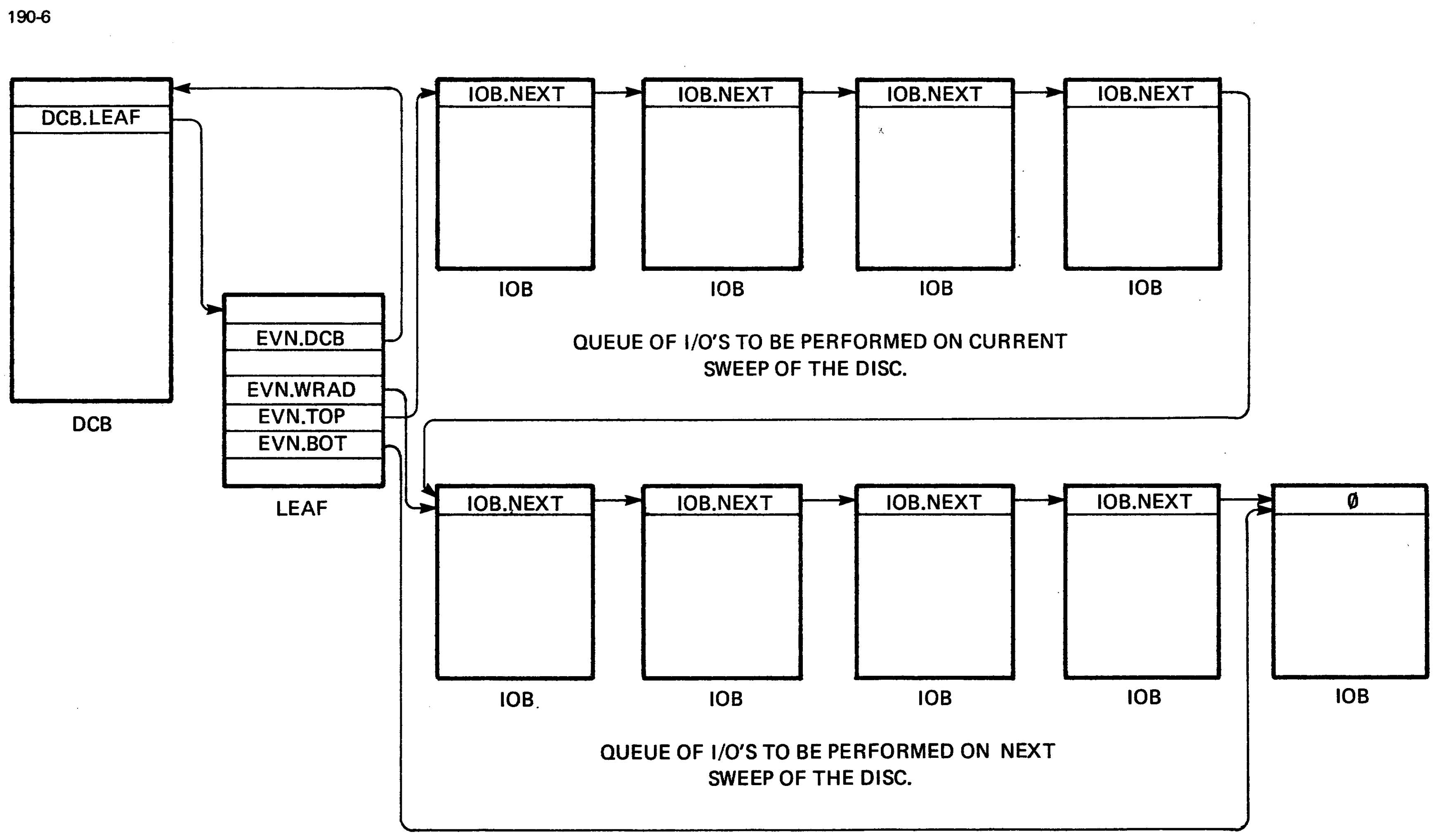
190-6
DCB.LEAF
DCB
EVN.DCB
EVN.WRAD
EVN.TOP
EVN.BOT
LEAF
IOB.NEXT
IOB
QUEUE OF I/O'S TO BE PERFORMED ON CURRENT SWEEP OF THE DISC.
IOB.NEXT
IOB
Ø
QUEUE OF I/O'S TO BE PERFORMED ON NEXT SWEEP OF THE DISC.

Figure 4-4. Queueing of I/O Requests to a Disk Device

4.5.1 Resource Nodes

Resource nodes are created at system generation (sysgen) time, and represent logical resouces rather then devices.

DIRECTORY LEAF and BIT MAP LEAF are noneventing. They behave like locks on the directory or bit map resource. These leaves are not the concern of the Series 3200 I/O subsystem. Other leaves (TIMER, POWER RESTORE, TIMESLICE) are all eventing leaves. They are placed on the system queue to force entry into the SQS state. The EVN.SQS fields in the leaves contain special executors outside of the I/O subsystem.

4.5.2 Extended Direct Memory Access (EDMA) Node

The EDMA node is used to limit the number of concurrent data transfers on the DMA bus. This node, if required by hardware configuration, must be specified at sysgen time by the COORDINATION statement. The COORDINATION statement has parameters which describe the SELCHs or devices that are to be coordinated and an optional parameter to specify the maximum number of simultaneous transfers.

4.5.3 System Queue Service (SQS) Executors

The single level and multilevel device dispatchers are the SQS executors.

4.5.3.1 Single Level Device Dispatch (SQS.SLV)

This algorithm is designed for devices attached to the multiplexor (MUX) bus. A test for connection is simply a test of a parameter (connected DCB address) for zero. If the leaf is indeed connected, then the driver is entered at the current event service routine entry point, which is found in DCB.ESR.

For disconnection, the DCB address in the leaf is zeroed. If the IOB queue is not empty, the top of the queue is selected for connection. Data from this IOB is then moved to the DCB. The DCB address in the node is updated and the leaf is added to the system queue so that the driver may be scheduled in its turn. The IOB is then released, and if the task was in an IOB wait, the wait is removed.

4.5.3.2 Multilevel Device Dispatch (SQS.MLV)

This algorithm is designed for devices attached to a shared controller and/or a SELCH. The current hardware I/O architecture requires the connection to the SELCH or controller before any communication is attempted to devices attached to it. This ensures that the channel or controller is not busy with another I/O. The connection algorithm starts from the first level device node (i.e., the leaf) and moves up each time a connection is successfully made. The request is always queued if the connection cannot be made. The connection algorithm at the first level device node is similar to simple device connection algorithm. For the connection at an upper level node, a test is made to a parameter (EVN.DCB) for zero. If zero, the requesting DCB address is saved and the connection is made. Otherwise, the requesting node is put on the tail of a first-in/first-out (FIFO) Q. For disks, the default Q strategy will queue the node as the head of the queue last-in/first-out (LIFO) if a seek is required. This policy encourages overlapped seeks.

If the connection is made up to the top of the tree, and the request is a non-data transfer request (for example, a seek), the driver ESR is entered (DCB.ESR). The complete connection is said to be made. For a data transfer request, and if an EDMA node has been configured, the EDMA node connection routine is called. If the request successfully connects to the EDMA node, the driver ESR is entered.

The disconnection also starts from the bottom up. Each time a disconnection is made, the top of the waiting queue on that level becomes connected and enqueues to the next higher level node, if any.

If the connection has been made to the EDMA node, the EDMA disconnection routine is called. If the queue at the at the top of the tree is not empty and the EDMA bus is needed, then the EDMA connection routine is entered.

Some operations do not require the connection to all levels after the operation has been initiated. For example, a seek operation in general only requires connection to the disk. Therefore, the EVREL routine is supplied to release upper nodes.

4.6 DRIVER SERVICE ROUTINES

Two service routines are provided for drivers to control the sequence of ESRs to be executed and to control access to the event coordination tree. These are EVMOD and EVREL.

4.6.1 EVMOD

EVMOD is called by a driver initialization or intermediate ESR to select the next ESR to be run, i.e., when the device ISR adds the device leaf to the system queue or when the system "times out" the device. Normally, the driver termination ESR (DCB.TERM) is automatically selected when the DIR exits via DIRDONE. If the DIR made a prior call to EVMOD, the default selection is overridden. Thus, if a driver makes a call to EVMOD to select an Intermediate ESR, it must ultimately call EVMOD to explicitly select the termination ESR as it will not be selected by default thereafter.

EVMOD is called as follows:

```
L      UF,DCB.LEAF(UD)
LA     UE,ESR_ADDR
BAL    U8,EVMOD
```

4.6.2 EVREL

EVREL is called by drivers to release upper level nodes of the event coordination tree while waiting for some event which does not require those resources. The usual use is to release the SELCH and device controller nodes while waiting for a seek complete on a disk.

Note that when the event occurs and the device ISR schedules the event, it will be necessary for the device to reacquire these nodes before the ESR can be re-entered. This is the function of the SQS.MLV executor. EVREL is called as follows:

```
    L     UF,DCB.LEAF(UD)
    LIS   UE,level
    BAL   U8,EVREL
```

The level is the level in the event coordination tree at and above which the nodes are to be released. Level 2 will release both the controller and SELCH nodes. Level 3 will release just the SELCH node.

4.7 DRIVER EXITS

There are two basic types of driver exits: intermediate (nonterminal) exits and termination exits. The two intermediate exits, DIRDONE and EVRTE, are used when the current I/O request is not complete, but control is being returned to the operating system to wait for an external event. The termination exits, IODONE and IODONE2, are used when the I/O request is finished, either because of normal or error termination conditions.

4.7.1 DIRDONE

DIRDONE is the normal (nonerror) intermediate exit from the driver initialization. Its purpose is to set up DCB.ESR with the driver termination routine (from DCB.TERM), unless the driver has previously called EVMOD (see Section 4.6.1). DIRDONE exits to the SQS main exit, SQS.EX, which checks for more entries on the System Queue.

4.7.2 EVRTE

EVRTE is a normal exit from intermediate ESRs. It performs no function other than to exit directly to the SQS main exit, SQS.EX.

4.7.3 IODONE

IODONE is the main driver termination exit. It is called for all normal and error terminations with the resulting SVC1 status in DCB.STAT (2 bytes), and the length of transfer, if applicable, in DCB.LLXF (fullword).

IODONE resets the HALT I/O Flag (DFLG.HIM) in the DCB.FLGS field and then checks for a device-specific IODONE handler in DCB.DONE. If one exists (DCB.DONE non-zero), IODONE branches to it. Such a device-specific routine may choose to return to IODONE after performing any device-specific processing. It may do so by branching back to the label IODCOM to perform the common processing described below.

IODONE performs three main functions as part of its common post-processing of an I/O request:

1. Return resulting status and length of transfer to the SVC1 parameter block via subroutine IODGST.

2. Remove I/O Wait or issue an I/O Completion Task Q trap for the task that requested the I/O via subroutine IODTWT.
3. Disconnect the request from the event coordination tree and promote any queued IOB or nodes by invoking common disconnection routine (COMDIS).

4.7.3.1 IODGST

IODGST is called by IODONE to return the status (DCB.STAT) and the length of transfer (DCB.LLXF) to the SVC1 parameter block (DCB.PBLK). It may also be called by driver-specific I/O DONE handlers that choose not to return to IODCOM as discussed above.

In addition to its primary functions described above, IODGST has one other function. It will reset the Interrupt Service Pointer Table for the device (DCB.DN) to ignore all interrupts (routine III). This function of IODGST can be overridden by setting IOBF.ISM in the request-dependent flags within the DCB's embedded IOB (DCB.RFLAG). Drivers that intend to service interrupts even when no request is active (e.g., terminal drivers that support "type-ahead") must be sure to set this flag.

4.7.3.2 IODTWT

IODTWT is called from IODONE or device-specific I/O DONE handlers to test the task trap and I/O Wait flags in the request dependent flags in the DCB's embedded IOB (IOB.RFLG).

If the task trap flag (IOBF.TPM) is set, IODTWT will create an I/O completion task queue entry and enqueue it to the task queue of the task (IOB.TCB). If the I/O Wait flag is set (IOBF.IOM), then IODTWT will remove I/O Wait from the task.

4.7.3.3 IODONE2

IODONE2 is exactly the same as IODONE. It is maintained for compatibility with earlier revisions of OS/32. Prior to Revision 04 of OS/32, the driver initialization routine ran in a different state (RS-State) from the termination routines (ES-State). Therefore, driver initialization routines could not call IODONE.

IODONE2 was provided as a termination exit for driver initialization routines - in thoses cases where the driver initial routine completed the request or encountered an error that required the request to be aborted. It is still common convention to use IODONE2 when it is necessary to terminate a I/O request from within the driver initialization routine.

4.8 OTHER UTILITY ROUTINES

4.8.1 Timer Services

The OS/32 I/O Subsystem provides a timer with a resolution of one second for monitoring drivers which are waiting for external events. A driver utilizes the timer services by placing itself (i.e., its DCB) on the DCB time-out chain. This is accomplished by a call to the routine TOCHON (time-out chain ON), described below.

Once a DCB is on the time-out chain, the action of the time with respect to that DCB is controlled by the halfword field DCB.TOUT. The possible values of DCB.TOUT and the corresponding time actions are as follows:

- **DCB.TOUT = X'7FFF'** - The timer is disabled. That is, the DCB will be ignored by the timer, even though it is on the time-out chain. Note: TOCHON initializes DCB.TOUT to this value.
- **1 <= DCB.TOUT <= X'7FFE'** - The timer is armed. Every second the system will examine each DCB on the time-out chain. If the value is between 1 and X'7FFE', inclusive, the timer logic will decrement DCB.TOUT by 1. If the value becomes zero, the device is timed-out. That is, an ESR is scheduled by adding the device leaf to appropriate system queue. It is the responsibility of all driver ESRs to check the value of DCB.TOUT for zero to recognize this time-out condition.
- **DCB.TOUT = zero** - The device has already been timed-out (i.e., the timer ignores the DCB. All driver ISRs should check for DCB.TOUT = zero and exit immediately if that condition is found. Drivers should never schedule an ESR when DCB.TOUT = zero.
- **DCB.TOUT < zero** - The timer is disarmed. That is, the driver has already scheduled an ESR. The timer will ignore the DCB. ISRs should also exit if DCB.TOUT < zero, so that a second ESR is not scheduled.

It is not possible for a driver to tell when the one second timer event will occur. Therefore, to guarantee at least one second of time, the driver must always set DCB.TOUT a value greater than one when arming the timer.

When a driver ESR detects a time-out (i.e., DCB.TOUT = zero), it should clean up, remove the DCB from the time-out chain (via a call to TOCHOFF), and terminate with a status of X'8282'.

It should be noted that the Halt I/O function of SVC1 mimics a device time-out. That is, Halt I/O sets DCB.TOUT to zero and schedules the device ESR by adding the leaf address to the system queue. A Halt I/O is distinguishable from an actual time-out by the fact that a DCB flag (DFLG.HIM) is set in DCB.FLGS for a Halt. It is customary for a driver to terminate with a status of X'8281' when a Halt I/O is processed.

4.8.1.1 TOCHON

The subroutine TOCHON is called to place a DCB on the system time-out chain; normally from the Driver Initialization ESR. If the device is already on the Chain, the call is ignored. The call to TOCHON is coded as follows:

```
     BAL    U8,TOCHON        place DCB on time-out chain
```

Register UD (13) must contain the DCB address, as it always does during driver ESR processing.

4.8.1.2 TOCHOFF

The subroutine TOCHOFF is called to remove a DCB from the system time-out chain. It should be called just before a driver exits to IODONE(2). If the DCB is not found on the time-out chain, the call is ignored. The call to TOCHOFF is coded as follows:

```
        BAL     U8,TOCHOFF
```

As usual, register UD (13) must contain the DCB address.

4.8.2 Queuing Routines

The acquisition and release of a resource by a device, such as a selector channel, is known as connection and disconnection. If a leaf is already connected, another connection cannot be made. The task is put in a wait state and the request is queued. The queue will be handled in priority order of the tasks in the queue. This assures a priority execution, and still maintains the chain order of first in, first out. As soon as a leaf is disconnected, the next IOB (on the top of the chain) will be connected and its wait state will be terminated.

Three different queueing strategies are available for disk I/O:

- A FIFO routine, in which the order is defined by the order entry into the queue (COMFIFO).
- A routine that orders the queue according to task priority (COMQ).
- A routine that sorts I/O requests is such a way that requires the shortest seek times (DISKQ).

Effectively, each DCB can specify its own queueing routine. The choice for standard devices is as follows:

The priority-ordered queue routine is selected for non-direct access devices and the seek schedule queue routine for direct access devices. Users can supply their own queueing routines by placing the routine address in DCB. The queue routine must follow the convention that the request at the top of the queue is the next one to be serviced. If no queueing routine is specified (a zero is found) in the DCB, the priority-ordered queue routine is the default. There may be a restriction on the registers available for use inside the queueing routine.

CHAPTER 5

STRUCTURE OF A DRIVER

LIST OF TABLES

CHAPTER 5

STRUCTURE OF A DRIVER

5.1 INTRODUCTION

OS/32 device drivers are, in general, re-entrant. They are written such that only one copy of the actual code can be shared by multiple instances of the supported device. The only exceptions to this are quick and dirty prototype drivers or drivers for one-of-a-kind devices. Chapter 6 discusses how to generate these data areas. Chapter 7 describes how a separate copy of the device control block (DCB) and channel control block (CCB) is generated for each device of this particular type at system generation (sysgen) time.

The code section of a device driver consists of several different types of routines. They are:

- Input/Output Handlers (IOHs) are routines that run in as nonreentrant system state extensions of the SVC1, in registers E8-E15 only. (See Appendix B for a description of operation states). These routines are optional as the system will provide a default IOH list (COMIOH).
- Event Service Routines (ESRs) are subroutines of the system queue service (SQS). A driver may have two or three special ESRs and possibly additional intermediate ESRs. These special ESRs are:
  - — The driver initialization routine (DCB.INIT) entered from SQS when SVC1 schedules a data transfer.
  - — The driver termination routine (DCB.TERM) selected, by default, when the driver initialization routine exits via DIRDONE. The DCB.TERM is entered when an interrupt service routine (ISR) adds the device leaf to the system queue to schedule the next ESR.
  - — Optionally, the command initialization routine (DCB.FUNC) entered from SQS when SVC1 schedules a command function.
- Interrupt Service Routines (ISRs) run as subroutines of the processor microcode in interrupt service state, registers E0 through E7 only. (See Appendix B for a description of operation states.) In effect, they run outside of the operating system as software extensions to the processor's interrupt service. To signal events to the operating system, ISRs add device leafs to the system queue.
- I/O Done Executor (DCB.DONE) is an optional routine that is seldom used except by the Perkin-Elmer Disk Driver and Contiguous File Manager. If supplied, it runs as a subroutine of, or in place of, the standard IODONE routine. In addition to special processing, a device-specific IODONE routine would normally perform many of the same functions as the standard IODONE (see Chapter 4).
- Queueing Strategy Routines (DCB.Q) several alternate queuing strategies (priority, first-in/first-out (FIFO), seek-scheduling for disks) are supplied by the operating system and are sufficient for standard requirements. (See Appendix D for detailed descriptions of queueing

routines.) If there are special requirements, these routines must by supplied as part of the driver.

- Translation Table is supplied if the driver uses the translation option of the auto driver channel (or the TRANSLATE command).
- Special Character Handlers are supplied if the translation table includes special character traps. The special character Handlers are routines to handle these traps or interrupts.

The following sections describe IOHs, ESRs, and ISRs. Standard IODONE and Q strategy are discussed in Chapter 4.

5.2 THE INPUT/OUTPUT HANDLERS (IOH)

The I/O Supervisor is often called the SVC1 skeleton because it performs only standard request validation and input/output block (IOB) allocation/initialization. It then invokes an IOH to execute the request. Each device driver has an IOH list, located by a pointer in the device's DCB. An IOH list contains an entry for each possible I/O request. The entry will contain the address of the IOH for the requested function, if the driver supports that function. Otherwise, the entry will contain the address of the SVC1 illegal function exit.

The IOH list also has entries for special event processing: system initialization, power restore, and end-of-task. These special entries are discussed in Chapter 5.

The system supplies several standard IOH lists and the associated I/O Handlers. These are:

- the NULL device,
- the bare disks,
- and the common IOH list.

In fact, the contiguous file manager is essentially a sophisticated set of IOH routines that front-end the disk driver. The dummy console is also a set of IOH routines that pass console I/O requests to the console monitor task for processing.

IOHs run as extensions of SVC1. Thus, they run in nonreentrant service state and are restricted to using registers 8 through 15. There are two basic types of IOH routines - those for requests that invoke the device driver and those that handle the requests directly without invoking the driver.

The IOH routines that invoke the device driver are those for data transfer functions (READ, WRITE, TEST and SET) and command functions (REWIND, WRITE FILEMARK, etc.). These IOH routines require an IOB and are entered with these register contents:

```
R9 = address of task control block (TCB)
RA = address of IOB
RB = address of DCB
```

The IOH routines attempt to connect the request to the device (leaf) so that the driver may be scheduled. Connection is performed by by copying the IOB into the DCB's embedded IOB and placing the DCB address in the leaf. The IOB can then be released, and the driver may be scheduled by adding the leaf address to the system queue. See Section 2.5.2.2 for information on SQS.

If the device is busy (i.e, leaf is already connected), the IOB is queued to the leaf unless the SVC1 function code specified Unconditional Proceed. If the task's IOB free pool is exhausted, the task is left in IOB wait. IOH routines exit to label SVC1EXIT.

The IOH routines for functions that do not invoke the driver do not require IOBs. These functions include HALT I/O, TEST I/O COMPLETE, and WAIT ONLY. The entire request is performed within the IOH. The following registers are initialized on entry:

```
R9 = address of TCB
RB = address of DCB
```

TABLE 1. INPUT/OUTPUT HANDLER (IOH) FORMAT

IOH.READ	DS 4	SVC1 read executor
IOH.WRIT	DS 4	SVC1 write executor
IOH.WAIT	DS 4	SVC1 wait-only executor
IOH.HALT	DS 4	SVC1 halt I/O executor
IOH.TEST	DS 4	SVC1 test I/O complete executor
IOH.REW	DS 4	SVC1 rewind executor
IOH.BSR	DS 4	SVC1 backspace record executor
IOH.FSR	DS 4	SVC1 forward space record executor
IOH.WFM	DS 4	SVC1 write filemark executor
IOH.FFM	DS 4	SVC1 forward filemark executor
IOH.BFM	DS 4	SVC1 backward filemark executor
IOH.EOT	DS 4	SVC3 task termination execution
IOH.INIT	DS 4	Device initialization
IOH.DDF	DS 4	Device-dependent function executor
IOH.CON	DS 4	Special handler for system console
IOH.PWR	DS 4	Special handler for power restore.

The IOH is set up via the IOH macro call.

Some are the concern of SVC1. Others allow for device-specific processing of events like task termination, system initialization and power restore.

The IOH List allows a driver to have custom SVC1 processing and special event processing (such as system initialization, power restore, and end-of-task). For most drivers, the SVC1 processing provided in the system default IOH List (COMIOH), is sufficient. Many drivers have special system initialization and/or power restore processing requirements. These drivers must include their own IOH list.

Custom IOH lists can be included in the driver itself. This is the usual case. If a single custom IOH list will serve several drivers, it may be implemented as a separate source module. In either case, the label of the IOH list must be an entry point in the source module.

IOH lists are coded using the IOH macro. Table 5-2 lists the parameters to the IOH macro. The NAME parameter provides the label for the IOH list. This is the only required parameter.

NOTE

> The IOH macro automatically generates a weak entry (WNTRY) for the NAME.

All of the SVC1 function parameters (READ, WRITE, etc.) will default to illegal functions if the parameter is omitted. To use the default IOHs (i.e., same as those used by COMIOH), use the label given in the last column of Table 5-2.

NOTE

> These labels must be declared as externals (EXTRN) to the driver/IOH module.

The special IOH entries all default to zero, indicating no special handling. If a driver needs special IOH processing (e.g., system initialization), the appropriate parameter is supplied, giving the name (label) of the IOH routine. For example, if a driver has a system initialization IOH with label DVR.SYSI, the parameter would be supplied as INIT=DVR.SYSI. Other special handlers are specified in the same manner.

All IOHs are entered via a BRANCH and LINK (BAL) instruction on register 8, with the DCB address in register 11 (B). Only registers 8 through 15 (F) are available.

TABLE 2. IOH MACRO PARAMETERS

PARAMETER	SVC 1 FUNCTION	IF OMITTED	FOR DEFAULT USE
NAME		REQUIRED	
READ	Read	SV1FCER	SVC1READ
WRITE	Write		SVC1WRIT
WAIT	Wait Only		SVC1WAIT
HALT	Halt I/O		SVC1HALT
TEST	Test I/O		SVC1TEST
SET	Test & Set		SVC1NOOP
REW	Rewind		SVC1REW
BSR	Backspace Record		SVC1BSR
FSR	Forward Record		SVC1FSR
WFM	Write File Mark		SVC1WFM
FFM	Forward File Mark		SVC1FFM
BFM	Backspace File Mark		SVC1BFM
DDF	Device-dependent function		
EOT	End-of-task	0	N/A
INIT	System Initialization	0	
CON	Console	0	
PWR		0	

5.3 DRIVER INITIALIZATION ROUTINE

The driver initialization routine contains preparation for the I/O transfer. The device is connected to the event tree and the physical I/O is started. This phase would, for example, start a seek operation for a disk I/O transfer.

The driver initialization routine is used to establish those aspects of the operation that are not themselves time critical but are necessary for the next phase which is the processing of interrupts. A typical sequence of events in the driver initialization routine might be:

- Check that the device is available, i.e., does not give either FALSE SYNC or DEVICE UNAVAILABLE status when addressed.
- Set up the CCB for an auto driver channel operation but without actually starting it. Use these guidelines:

— Place the buffer address and length into CCB.EB0 and CCB.LB0.

— Place A(ISR) into CCB.SUBA.

— Initialize the channel command word (CCB.CCW).

- Set up the interrupt service pointer table (ISPT) entry for the first interrupt service routine (ISR).
- Connect the DCB to the time-out chain, but do not initiate the time-out countdown.
- Execute the first ISR via a simulate interrupt (SINT) instruction.
- Terminate the routine by branching to DIRDONE.

The driver initialization phase is entered at the label INITxxxx, where xxxx is the name of the device. On entry to this phase from SQS, register 13 contains the DCB address and register 15 contains the leaf address. The subroutines used in this phase are TOCHON, EVREL, EVMOD, and TOCHOFF. The driver may exit from the phase by branching to DIRDONE or IODONE.

TOCHON is used to put entries onto the time-out chain and is called via:

```
    BAL   U8,TOCHON
```

There is one entry on the chain per DCB. On entry to the subroutine, register 13 contains the DCB address. The subroutine sets the time-out value to X'7FFF' and puts the entry onto the time-out chain.

DIRDONE is used for a successful exit from the initialization phase. No registers need to be established before entry. The exit will establish the label TERMxxxx as the current ESR of the driver termination phase, i.e., the address of the DCB.TERM is put into the field DCB.ESR, unless an ESR has already been established in the initialization phase via the routine EVMOD.

IODONE is used when the initialization encounters a fatal error or normal termination condition. For example, the device might be unavailable (not on-line) as determined by the device status. This exit can also be used in the termination phase.

5.4 INTERRUPT SERVICE ROUTINE (ISR)

One or more ISRs are responsible for responding to interrupts and for starting any subsequent physical I/O operations or event service routines (ESR). For example, if the disk seek has been completed, an ISR will be activated, in order to output the data transfer commands to the device.

ISRs are entered with a program status word (PSW) state of X'28xx'. Only registers 0 through 7 can be used in an ISR, unless one can guarantee absolutely that all devices the driver will control will always be strapped to interrupt at a level lower than 0, at which point registers 0 through 15 may be used. If registers 8 through 15 of register set 0 are used by an ISR, they are likely to destroy any register the operation system is currently using and cause the operating system to crash.

The ISRs are used for all occasions in which it is undesirable to be interrupted by other operations. The first ISR is different from subsequent ISRs because it is entered via SINT instruction. It is essentially an uninterruptible subroutine of the driver initialization routine.

The ISR are executed in the interrupt state (IS-STATE). Entry into these is made on the basis of a resultant hardware interrupt of the device, or by carrying out a SINT command. Normally all I/O commands are executed in the ISRs. Another ISR can be run for the next interrupt of this device, when the corresponding address in the ISPT or the CCB is being modified. The ISR should be written in pure reentrant code.

A typical sequence of events in the first ISR might be:

- Check the condition code to determine why you are in the ISR routine.
- Start the time-out countdown by setting a value in DCB.TOUT.
- Prepare the device for transfer (OC DEVADD,COMMAND).
- Transfer a byte of data.
- Set up the CCB subroutine address (SUBA) for the next ISR.
- Start the auto driver channel by modifying the channel command word (CCW), enabling interrupts and attempting the first READ or WRITE on the device. Set the execute bit in the CCW.
- Return to the driver initialization routine via the instruction `LPSWR  EO`.

Because the first ISR returns to the driver initialization routine, it is possible to return state information via the PSW condition code in register E0. If, for example, the initial ISR determines that the I/O cannot proceed for one reason or another, it is more efficient to abort the I/O in the driver initialization routine, rather than schedule an ESR to handle termination. The first ISR can accomplish this setting an appropriate error status in the DCB and setting a bit in the condition code of the old PSW in E0. When control returns to the driver initialization routine (i.e, when the ISR performs `LPSWR EO`), the driver initialization routine can branch on condition to DIRDONE (no error) or IODONE (error).

Subsequent ISRs deal with the transfer of data, error detection or other special conditions. For example, the SUBA, which typically would be set up for the next ISR, is given control when the buffer is empty. This ISR will also be involved when the device gives a bad status. The ISR might perform the following operation:

- Check the condition code for X'01' (bad status) - in which case the status should be returned to the calling program and the operation terminated.
- Check the condition code for X'02' - schedule termination ESR to return status to the caller and terminate I/O operations.
- Check to determine whether the device timed out with the following:

```
L    E5,CCB.DCB(E4)        Fetch DCB address
LH   E6,DCB.TOUT(E5)       Fetch timer value
BZ   TIMEOUT               If zero, branch to set time-out status
```

ISRs should be kept as short as possible (15 to 25 instructions).

During their execution, only higher level I/O interrupts can be serviced. If the ISR disables all interrupts, then no interrupts can occur.

ISRs are entered with a PSW state of X'28xx'. Only registers 0 through 7 can be used in an ISR, unless one can guarantee absolutely that all devices the driver will control will always be strapped to interrupt at a level lower than 0, at which point registers 0 through 15 may be used. If registers 8 through 15 of register set 0 are used by an ISR, they are likely to destroy any register the operating system is currently using and cause the operating system to crash.

5.5 FINAL INTERRUPT SERVICE ROUTINE (ISR)

The final ISR disarms the device interrupts and places the leaf address on top of the system queue with an ATL instruction. A typical sequence of events in the final ISR might be:

- **Check the condition code to determine why you are in the final ISR routine.**
- **Place the status code in DCB.STAT and DCB.DDPS**
- **If DCB.TOUT=0 then a time-out has occurred. If a time-out has not occurred, place -1 into DCB.TOUT.**
- **Schedule TERMxxxx by ATL A(LEAF),SQ.**
- **Disarm the device.**
- **Return to operating system.**

5.6 EVENT SERVICE ROUTINE (ESR) (INTERMEDIATE AND TERMINATION)

The driver ESRs are a collection of routines for intermediate operations and for terminating the I/O operation. In driver termination, the status of the operation is put into the DCB.STAT so that it may be sent back to the calling program's parameter block status halfword.

Driver termination is entered when an ESR has been scheduled from the system queue. An ESR may also be entered because of a time-out.

An entry put on the time-out chain in the initialization phase has to be removed in driver termination.

Failure to do this can result in a error condition that can cause a system crash.

A typical sequence of events in an ESR would be:

- Check for time-out. Is the countdown value 0? (DCB.TOUT)
- Check for a HALT I/O operation from the calling program. (DCB.TOUT = zero and DFLG.HIB set in DCB.FLGS)
- Return the I/O operation status and length of transfer to the DCB.
- Exit via branching to routine IODONE or EVRTE if I/O is to continue.

When ESRs are executed in the termination phase of the driver, the only subroutine called is TOCHOFF. Termination ESRs always end by branching to IODONE, or EVRTE if I/O is to continue.

TOCHOFF is called to remove an entry from the time-out chain via:

```
BAL   U8,TOCHOFF
```

On entry, register 13 or UD contains the DCB address. Registers 10 and 11 are destroyed.

Intermediate ESRs are used to:

- modify CCB.SUBA for the next ISR. A SINT can be used to get it started.
- disconnect the DCB from upper level nodes in the event coordination table.
- modify DCB.ESR. (SQS uses the address in DCB.ESR as the entry point into the driver when the A(LEAF) is removend from system queue.)

CHAPTER 6

COMPONENTS OF THE DEVICE CONTROL BLOCK (DCB) AND CHANNEL CONTROL BLOCK (CCB)

LIST OF TABLES

CHAPTER 6

COMPONENTS OF THE DEVICE CONTROL BLOCK (DCB) AND CHANNEL CONTROL BLOCK (CCB)

6.1 INTRODUCTION

The DCB and associated CCBs are the device driver data and control areas. There is usually one DCB definition for each type of device. Each type of DCB is assigned a device code or DCOD. The DCOD is used to select DCBs (and drivers) when configuring the system, as described in Chapter 7.

DCODs zero through 239 and 255 are either already assigned or reserved for use by Perkin-Elmer. DCODs 240 through 254 are available for user-written drivers. As mentioned above, it is by DCOD that devices are configured into the system. The DCOD selects a DCB type corresponding to the device. The DCB, in turn, selects which driver will be included to handle the device.

When more than one device of the same type are configured, multiple DCBs (and CCBs) of the same type are generated. DCBs are generated by assembling DCB macro calls that are produced by the system generation utility (SYSGEN/32) with parameters supplied in the configuration statements. The remainder of this chapter describes the components of these DCB and CCB macros.

A DCB macro has a name of DCBnnn where n is a 3-digit device code (from 240 to 254). The macro must be written to generate a customized DCB each time it is called, based on the parameters supplied. The macro must reserve storage for and initialize fields in the device-independent and device-dependent portions of the DCB. It must also reserve storage for and initialize any CCB used by the driver. The DCB macro is made up of the following sections:

- The macro prototype statement - giving the macro name (DCBnnn) and the list of valid keyword parameters.
- The macro variable declarations and initializations.
- Environment initialization (USERINIT).
- Unique DCB ID generation (%IDVAL).
- Object module label generation (PROG).
- Storage allocation and initialization for the DCB (DCBI) and CCB(s) (CCBI).
- Device Mnemonic Table and Device Leaf Linkage.
- Miscellaneous (optional) storage allocation and storage initialization.
- Macro termination (USEREND/MEND).

Each of these items is covered in the following subsections. Additional information and examples are given in the DCBFORM sample macro included with the OS/32 software package in the SYSGEN32.MLB macro library. A listing of DCBFORM is included in Appendix C.

6.2 THE MACRO PROTOTYPE STATEMENT

The macro prototype follows the macro statement and gives the macro name (DCBnnn) and the list of valid parameters. The following parameters are required and will always be supplied by SYSGEN/32.

%DCOD	is the device code. The device code is used as an ASCII string to form the DCB name. The device code also is placed in the DCB structure as a halfword value, where it can be accessed by the driver to control the execution sequence of the driver.
%DN	is the device address (i.e., the physical device address on the multiplexor (MUX) bus to which the device controller responds). It initializes the DCB.DN field.
%ILVL	is the device's interrupt priority level.
%NAME	is the device mnemonic, a 1-4 character device name.
%IOP	is the input/output processor (IOP) number and is provided only if the device is under an IOP in a 3260 System.

A complete list of parameters and their meaning is given in Table 6-1. A more complete description is given in the SYSGEN/32 manual in the discussion of the devices statement.

TABLE 6-1. MACRO PROTOTYPE STATEMENT PARAMETERS

PARAMETER	SYSGEN/32 OPTION	DESCRIPTION
%DCOD	DCOD	Device code; also used to create unique DCB & CCB names.
%DN	ADDR	Device address
%CLAS	IOCLASS	I/O class level
%ILVL	ILEVEL	Interrupt level
%NAME	NAME	Device mnemonic used in naming device DCB & CCB in Device Mnemonic Table (DMT).
%QU	QUEUE	Queue scheduling routine for disks
%CONS	CONSOLE	Flags console device
%SLCH	SELCH	Selector channel address
%CNTR	CONTROLLER	Device controller address
%IOP	IOP	Processor number where device resides
%XDCD	XDCOD	Extended device code.
%SPND	SPINDLE	Specifies spindle for floppy disks
%RECLN	RECLEN	Record length
%SPCR	READCONTROL	Read control character sequence

PARAMETER	SYSGEN/32 OPTION	DESCRIPTION
%SPCW	WRITECONTROL	Write control character sequence
%TV1	SCREEN-TIME	Time for entire screen
%TV2	RESPONSE-TIME	Time for term response
%XLT	TRANSLATE	Name of translation table
%PDCT	PADCOUNT	Padcount value
%LDCT	LEADSYNC	Leading sync. count
%SHCCB		Flag for shared CCB
%POLMT	POLLIMIT	Poll limit value
%EOV	EOV	End-of-volume flag
%CLOCK	CLOCK	Clock value
%SIZE	SIZE	Page size for pseudo-devices
%DUAL	DUAL	Dual port option
%CM	CM	Channel manager address
%ITV	INTIMER	Input device timer
%IOLM	IOLIMIT	Error retry for I/O
%SLS	LINESTATUS	Static line status
%MNOF	MAXFRAMES	Max frames outstanding
%MBFS	MAXWRITEBUF	Maximum buffer size
%MRBS	MINREADBUF	Minimum record buffer size
%MTO	MTO	Master time-out
%N2	N2	No response
%NC	NCS	Numbered commands
%OTV	OUTTIMER	Output device timer
%PLDT	POLLDELAY	Poll delay timer
%PLTC	POLLTIME	Poll time out
%SSA	SSA	Sec station address
%T1	TO2	T1 timer for X.25
%UCSI	UCSI	UNNUM cmds input
%UCSO	UCSO	UNNUM cmds output
%WKTC	WAKEUP	Wakeup time-out
	NODISK	Suppresses DA parameter in EVNGEN macro call.
	DISK	Causes DA parameter in EVNGEN macro call.
	NONSHARED	Suppresses %SHCCB parameter in DCB macro call
	USER	User-defined parameters

In addition to these parameters which are known to SYSGEN/32, additional device-dependent parameters may be specified. These "USER" parameters may be supplied at sysgen time via the "USER=" parameters on the device specification statement. The following is an example of the DCB macro prototype statement.

NOTE

> Macro continuation lines are indicated by a non-blank character in column 72. This character will *not* appear on Common Assembly Language (CAL) listings.

```
MACRO
DCB241    %DCOD=,%DN=,%NAME=,%CLAS=,                       x
%ILVL=,%RECLN=,%XDCD=,%PEN=
```

Two optional parameters (%RECLN and %XDCD) and one USER parameter (%PEN) can be specified. It is possible to supply default values for any of the optional parameters. For example, %RECLN=256.

The %XDCOD field may assume whatever significance a particular device driver wishes to assign to it. For example, in some Perkin-Elmer supplied drivers, the %XDCD field is used to specify the baud rate and parity of a communications line.

After the DCB macro prototype statement, all local and global macro variables must be declared. Two are required:

BGBLA %IDnnn	is a batch global arithmetic variable that counts DCBs of type nnn (=device code) - starting at zero.
GBLC %IDVAL	is a global character variable that contains the ASCII representation of the current value of %IDnnn. It is used to generate unique DCB labels of the term DCBnnnid.

Other local and global macro variables may be declared and initialized as required by the structure/logic of the particular DCB macro. See the examples in DCBFORM in Appendix C.

6.3 ENVIRONMENT INITIALIZATION

After the macro variable declarations and initializations, the DCB macro for custom drivers must call the USERINIT macro. This macro resets various macro flags to initialize the macro environment. In particular, it resets flags that indicate that the structure definition macros have already been included. If DCB or CCB field names are undefined in a user-written DCB macro (during the macro/assembly phase of system generation), check the macro definition to ensure that USERINIT is invoked at the appropriate place in the macro. See Appendix C for the DCB macro definition.

At this point it is also a good idea to pull in the structure of the device-dependent part of the DCB. Presumably such a structure is defined when writing the driver code. This provides the symbolic field definitions for the miscellaneous field initialization section of the macro. It also provides the total size of the DCB for the DCBI macro. For example, if the device is a plotter with a device-dependent DCB defined in $PLTRDCB:

```
USERINIT     initialize macro variables
$PLTRDCB     include device-dependent DCB definition
```

$PLTRDCB must reside in the USERDLIB.MLB macro library along with the DCB macro itself.

6.4 UNIQUE DCB ID GENERATION

This section of the DCB macro generates the unique ASCII ID value (%IDVAL) and increments the counter %IDnnn. This is accomplished as follows. First, the current value of %IDnnn is converted to ASCII in %IDVAL using the macro CONVNUM:

```
CONVNUM    VAL=%IDnnn
```

Then, the value of %IDnnn is incremented by one for the next device of this type (if any):

```
%IDnnn    SETA %IDnnn+1
```

%IDVAL can now be used to generate unique DCB/CCB labels. For example:

```
DCB%DCOD%IDVAL     EQU *

or

CCB%DCOD%IDVAL     EQU *
```

Because the sequence %DCOD%IDVAL appears frequently within the macro, it is often advantageous to define a local character macro variable and initialize to the value %DCOD%IDVAL. For example,

```
    LCLC    %OFFS
    .
    .
    .
    %OFFS      SETC    '%DCOD':'%IDVAL'
```

The variable %OFFS can then be used in place of %DCOD%IDVAL.

6.5 OBJECT MODULE LABEL GENERATION

Each user-written DCB is assembled as a separate unit, producing a labeled object module for each device. The object module is given a name via the "PROG" statement, as follows:

```
    DCB%NAME PROG    DCB program label for link map.
```

%NAME is the device mnemonic passed by SYSGEN/32.

6.6 DEVICE CONTROL BLOCK (DCB) STORAGE ALLOCATION/INITIALIZATION

This section is the heart of the DCB macro. A utility macro, DCBI, is used to reserve the storage and perform specified initialization. The parameters to DCBI and the DCB fields that they initialize are listed in Table 6-2. A listing of the DCBI macro is included in Appendix C.

TABLE 6-2. DCBI PARAMETERS AND INITIALIZED FIELDS

DCBI PARAMETER	DEFAULT VALUE	DCB FIELD	COMMENTS
%FUNC	0	DCB.FUNC	Command driver entry
%INIT		DCB.INIT	Data transfer driver entry
%TERM		DCB.TERM	Termination ESR entry
%ATRB	0	DCB.ATRB	Supported attributes
%IOC	0	DCB.CLAS	Accounting class
%RECL	0	DCB.RECL	Record length
%TOUT	X'7FFF'	DCB.TOUT	Initial time-out constant
%FLGS	DFLG.LNM	DCB.FLGS	Device flags
%DCOD		DCB.DCOD	Device code
%IOH	COMIOH	DCB.IOH	Address of IOH list
%DA		DCB.DIRL	Points to DIR%DCOD%ID
		DCB.BITL	Points to BIT%DCOD%ID
%CLOC		DCB.CCB+6	
%CC4		DCB.CCB+4	
%DSIZE		DCB.SIZE	Disk size in sectors
%STRK		DCB.STRK	Sectors per track
%TCYL		DCB.TCYL	Tracks per cylinder
%BMSA		DCB.BMSA	Bitmap buffer required
%DRSA		DCB.DRSA	Directory buffer required
%DSC		DCB.DSC	
%NUM		DCB.NUM	
%PFUN		DCB.PFUN	
%PXLT		DCB.PXLT	Card reader/punch punch translation table
%RXLT		DCB.RXLT	Card reader/punch read translation table
%PSEP		DCB.PSEP	Card reader/punch punch separate
%EOLC		DCB.EOLC	End of line characters for line printer
%RTRY		DCB.RTRY	Operation retry counter
%SHCC	0	DCB.CCB	If not 0, define first CCB pointer
%CCB		DCB.CCB+4	If not 0 and %SHCC not 0, then define second CCB pointer
%EDMA	0	DCB.EDMA	
%COPY	DCB		Used to copy structure
%SIZE	DCB.DVDP+4		Size in bytes of DCB
%IOP	0	DCB.IOP, DCB.ISP DCB.SQ	Sets up parameter dependent fields in the DCB.

The following DCBI macros are of particular importance:

DCOD*	is the device code for this DCB.
SIZE	is the amount of storage to be reserved for the DCB. This parameter defaults to the size of the device-independent DCB plus 4 bytes.
INIT*	is the name of the device driver data transfer initialization entry point (e.g., INITxxxx).
TERM*	is the name of the device driver termination event service routine (ESR) entry point (e.g., TERM=TERMxxxx).
FUNC	is the name of the device driver command initialization entry point (e.g., FUNC=CMDxxxx).
IOH	is the name of the device driver input/output handler (IOH) list (e.g., IOH=IOHxxxx).
ATRB*	supported SVC1 function codes for DCB.ATRB (e.g., ATRB=7B80).
ID*	is used to pass IDVAL for unique label generation (e.g., ID=%IDVAL).
IOP	is the processor number where the device resides.

The parameters marked with an asterisk are required parameters to DCBI. The INIT/TERM parameters are important because reference to the driver entry points by the DCB is the mechanism for including the appropriate driver to handle the device.

Other parameters to DCBI may be derived from parameters to the DCBnnn macro (e.g., RECL=%RECLN) or set as constants within the DCDnnn macro (e.g., RECL=80). Table 6-2 lists the defaults for other parameters. Any parameter (or DCB field) that is not specified and that does not have a default will be initialized to zero.

Example:

```
DCBI    DCOD=241,SIZE=PLTRDCB,INIT=INITPLTR,                  x
        TERM=TERMPLTR,FUNC=CMDPLTR,ATRB=7BCO,                 x
        IOH=IOHPLTR,ID=%IDVAL
```

Additional device-dependent DCB initialization and optional storage allocation/initialization is discussed later in this chapter.

6.7 CHANNEL CONTROL BLOCK (CCB) STORAGE ALLOCATION/INITIALIZATION

All OS/32 device drivers that handle interrupts require at least one CCB. Some devices such as full-duplex communication controllers require two CCBs (one for transmit and one for receive). The CCBI macro will allocate or initialize one CCB on each call. A second call to CCBI is allowed to generate the second CCB, where required.

There are two required parameters to CCBI:

- DCOD - the device code
- ID - the current IDVAL (e.g., ID=%IDVAL)

These parameters are used for generating a unique CCB label (CCBnnnid), and for referencing the DCB (as DCBnnnid). Optional parameters are listed in Table 6-3. These parameters initialize fields in the CCB. Normally, CCB fields are set up dynamically by the device driver.

TABLE 6-3. OPTIONAL PARAMETERS FOR CCBI MACRO DEFINITION

CCBI PARAMETER	CCB FIELD	COMMENTS
%XLTAB	CCB.XLT	Translation table
%SUBA	CCB.SUBA	ISR Address
%EBO	CCB.EBO	Buffer 0 End Address
%CCW	CCB.CCW	Channel Control Word
%CFLGS	CCB.FLGS	Driver-dependent flags
%CCBN	N/A	Must be undefined for first CCB

When calling CCBI to generate a second CCB, the parameter %CCBN must be specified - any value will do (e.g., CCBN=2). The second CCB will be named "CCXnnnid." Also, when two CCBs are generated, the DCBI macro parameter CCB should be specified to cause the second CCB to be referenced by the DCB.

Example of first or only CCB:

```
CCBI   DCOD=241,ID=IDVAL,XLTAB=XLTPLTR
```

Example of a second CCB, if required:

```
CCBI   DCOD=241,ID=IDVAL,CCBN=X
```

Note that any value of CCBN is acceptable as long as the parameter is defined. The value itself is ignored.

6.8 DEVICE MNEMONIC TABLE (DMT) AND LEAF LINKAGE

For the DCB to be located via the DMT, a special label and entry point must be generated as follows:

```
DCB_%NAME EQU   DCB%DCOD%IDVAL
          ENTRY DCB_%NAME
```

This generates an alternate name for the DCB that can be referenced by the DMT macro generated by SYSGEN/32. For example, if a device is named "PLTR" for a plotter, the alternate DCB name would be DCB_PLTR.

After generating the label for the DMT, it is also necessary to initialize the DCB.DMT field to reference the DMT entry for this device:

```
ORG    DCB%DCOD%IDVAL+DCB.DMT
EXTRN  DMT_%NAME
DAC    DMT_%NAME
```

The label DMT_%NAME is generated by the DMT macro as an entry point.

The DCB.LEAF field must be initialized to point to the correct device leaf. If the device shares a leaf with other devices, the parameter %SHCCB will be nonnull. In this case the name of the leaf is LF%SHCCB. Otherwise, the leaf name is LF%DCOD%IDVAL. The leaves and their labels are generated by the EVNGEN macro emitted by SYSGEN/32.

The following code sequence initializes the DCB.LEAF field:

```
          ORG    DCB%DCOD%IDVAL
          AIF    (T'%SHCCB'  EQ 'U')&NSLEAF
          EXTRN  LF%SHCCB
          DAC    LF%SHCCB
          AGO    &NRMLFX
&NSLEAF   ANOP
          EXTRN  LF%DCOD%IDVAL
          DAC    LF%DCOD%IDVAL
&NRMLFX   ANOP
```

Note that if this will never be used with shared leaves, the conditional (AIF), the references to LF%SHCCB and AGO%NRMLFX can be omitted.

6.9 MISCELLANEOUS (OPTIONAL) STORAGE ALLOCATION/INITIALIZATION

This section of the DCB macro generates any optional storage outside of the DCB itself. For example, the driver might require a work buffer whose size is specified as a sysgen parameter and therefore cannot be part of the fixed length device-independent DCB structure.

This section also performs optional initialization of fields in the DCB (usually the device-dependent part), such as Extended Device Code (XDCD) or pointers to optional storage discussed above. Some of the initialization may be driven by parameters supplied at sysgen time. For example:

```
          AIF(T'XDCD  EQ 'U')&NOXDCD
          ORG    DCB%DCOD%IDVAL+DCB.XDCD
          DC     %XDCD
&NOXDCD   ANOP
```

This code initializes DCB.XDCD if a value was specified. Otherwise, the field is left at zero.

Other initializations may be required. For example, suppose the driver requires a variable length buffer, specified by USER parameter %BUFL (default 256). The following code might be used:

```
ORG  DCB%DCOD%IDVAL+DCB.BUFS
DAC  BF%DCOD%IDVAL           start of buffer
DAC  BFE%DCOD%IDVAL          end of buffer
```

This will set up the buffer start and end addresses in the device-dependent DCB. The buffer would be allocated as follows:

```
               ORG    $ST%DCOD%IDVAL      end of DCB
               ALIGN  4                   (if required)
BF%DCOD%IDVAL  EQU  *
               AIF(T'%BUFL EQ 'U')&BUFDFL  use default
               DS     %BUFL                specified size
               AGO    &BUFF
&BUFDFL        ANOP
               DS     256                  default size
&BUFF          ANOP
BFE%DCOD%IDVAL EQU *-1
$ST%DCOD%IDVAL EQU *                       redefine end address
```

Note that label $ST%DCOD%IDVAL is defined by the DCBI macro as the first location after the end of the device-dependent DCB. When allocating additional storage, this label should be redefined as shown.

6.10 MACRO TERMINATION

After all miscellaneous initializations have been completed, all that remains is to properly terminate the DCB. This requires three steps.

1. "ORG" to the end of the DCB or any allocated storage (label $ST%DCOD%IDVAL).
2. Invoke the USEREND macro.
3. Code the MEND (Macro End) statement.

The following is the required code:

```
ORG        $ST%DCOD%IDVAL
USEREND
MEND
```

CHAPTER 7

DIFFERENCES FOR DRIVERS WRITTEN UNDER INPUT/OUTPUT PROCESSORS (IOP)

CHAPTER 7

DIFFERENCES FOR DRIVERS WRITTEN UNDER INPUT/OUTPUT PROCESSORS (IOP)

7.1 INTRODUCTION

Users of the 3260 System have available to them the 3200 IOP which performs physical input/output (I/O) to the devices configured under it. The purpose of the IOP is to minimize the amount of time the central processing unit (CPU) has to spend performing I/O functions, thereby enabling the CPU to perform other operating system services or execute user tasks in a more efficient manner.

A 3260 System can have up to nine satellite processors which can be a mixture of auxiliary processing units (APU) and IOPs. The devices in a 3260 System can reside either under an IOP (or multiple IOPs), the CPU or both. Note that the system console and all Integrated Telecommunication Access Method (ITAM) devices (including the direct memory access input/output subsystem (DIOS)) must reside on the CPU. It is possible for a single copy of a device driver in memory to support devices that reside under multiple processors. In fact, Perkin-Elmer's standard device drivers have this capability.

This chapter will discuss the techniques a driver writer should use so that the driver can execute under both the CPU and IOPs.

7.2 CENTRAL PROCESSING UNIT (CPU) INPUT/OUTPUT PROCESSOR (IOP) DIFFERENCES

There are several ways in which the IOP differs from the CPU in a 3260 System. The first difference is that the IOP has an interrupt service pointer table (ISPT) that has fullword entries and can be located anywhere in memory. The CPU's ISPT has halfword entries and is always located at memory address X'D0'. Therefore channel control blocks (CCBs) and interrupt service routines (ISRs) are not restricted to the first 64KB of memory. A pointer to the ISPT for each IOP is located in its input/output parameter block (IPB).

The second difference is an optional fullword CCB.SUBA field as specified by bit 9 in the channel command word (CCW). (See Fig 2-6 and accompanying description of CCB.) Therefore, the ISR address specified by the CCB.SUBA field is not restricted to the first 64KB of memory. Currently, this 64KB limit still exists for standard Perkin-Elmer device drivers, as they have not been modified to use this particular feature of the IOP.

The third difference is the use of a multiple level type of system queue service (SQS) called synchronous interrupt service (SIS). Drivers configured under the IOP should not attempt to access the standard system queue of the CPU. On the CPU, leaf structures are added to the system queue to be serviced. On the IOP, the leaf structure must be added to the level 4 synchronous interrupt queue.

Mechanisms are provided to handle the differences between the CPU and the IOP, as described below.

7.3 BUILDING THE DEVICE CONTROL BLOCK (DCB)

There are a number of new fields in the DCB that play a significant role in the proper operation of a driver due to the addition of the IOP. These fields are:

DCB.IOP	is the id number of the processor where the device resides; RTSM id for IOP devices; zero for CPU devices.
DCB.ISP	is the address of the ISPT for the processor where the device resides. This address is variable for IOP devices and X'D0' for CPU devices.
DCB.SQ	is the address of the proper system queue for the processor where the device resides. This field is also the address of the synchronous interrupt queue level 4 for IOP devices and the address of the standard system queue for CPU devices.

These fields must be set up at system generation (sysgen) time by the macros that build each DCB.

The Sysgen/32 task has been modified to recognize devices that are under an IOP and user-written drivers are included in the operating system using the standard Sysgen/32 mechanism. See the SYSGEN/32 Reference Manual for more details.

The user-written DCB must set up the new DCB fields described above. The Sysgen/32 task will add a new parameter to all device macro calls which is the id number of the processor where the device resides if it is under an IOP. Therefore the user should add an "IOP=0" parameter to the macro prototype statement of the user DCB macro. The zero is the default value, indicating the CPU. If the user DCB macro calls the DCBI macro provided by Perkin-Elmer to set up many standard DCB fields, the user only has to add a "IOP=%IOP" parameter to the parameter list in the DCBI macro call. If the user DCB macro sets up the DCB fields itself, the following code must be added to the macro to set up the new fields. These macro statements will follow the

```
    ORG    $ST%DCOD%ID
```

line at the end of the standard DCBI definition as shown in Appendix C.

```
   AIF    ('%IOP' EQ 'O')&NOIOP          Branch if on CPU
   ORG    D%SYSINDX+DCB.IPB
   DAC    A(IPB%IOP)                     IPB address
   EXTRN  IPB%IOP
   ORG    D%SYSINDX+DCB.ISP
   DAC    A(ISPS%IOP)                    ISPT address
   EXTRN  ISPS%IOP
   ORG    D%SYSINDX+DCB.SQ
   DAC    A(SIQ%IOP:4)                   system interrupt queue level 4 address
   EXTRN  SIQ%IOP:4
   ORG    D%SYSINDX+DCB.IOP
   DCZ    %IOP                           IOP number
   AGO    &IOPEXIT                       Done, branch out
&NOIOP     ORG    D%SYSINDX+DCB.SQ
   DAC    SQ                             CPU system queue address
   EXTRN  SQ
   ORG    D%SYSINDX+DCB.ISP
   DAC    X'DO'                          CPU ISPT address
   ORG    D%SYSINDX+DCB.IOP
   DCZ    O                              Not on IOP
&IOPEXIT   ANOP                          Finished
```

7.4 CODE CHANGES

In the driver code itself, the only problems the programmer should encounter is when the driver must access the system queue (e.g., to schedule the leaf in an ISR) or modify the entry in the ISPT. Two macros have been provided that reside in the SYSMACRO.MLB provided by Perkin-Elmer that enable a driver to be written in a manner that is transparent to the processor.

7.4.1 Add to System Queue Macro (ADDSQ)

For all Series 3200 Processors, except the IOP on the 3260 system, the following instruction is used by a device driver to add an item to the system queue:

```
ATL  Rx,SQ
```

On an IOP, this instruction will not produce the desired result, since the IOP Event Service (ES) state is not driven by the system queue data structure. In addition, the IOP, like all other 3200 Series processors, does not generate internal system queue service interrupts.

Thus, the ADDSQ macro is provided. This macro will take the item (typically a leaf) and add it to the queue pointed to by the field DCB.SQ.

Calling convention:

```
label     ADDSQ ITEM=Rx,DCB=Ry,WORK=Rz
```

Where:

Rx is a register that contains the address of the item to add to the processor's queue.

Ry is a register that contains the address of the DCB.

Rz is a work register.

This macro will generate the following code, which is compatible with both the IOP and the other 3200 Series processors:

```
   L      Rz,DCB.SQ(Ry)      Get queue pointer
   BNZ    LABEL1             If 0, implies standard SQ,
   LA     Rz,SQ              so get standard  SQ address
LABEL1     EQU   *
   ATL    Rx,0(Rx)           Add the item to the queue
   EPSR   Rz,Rz              Get current PSW
   TI     Rz,PSW.NTM         If bit 15 on, we are on IOP
   BZ     LABEL2             If on CPU, then exit
   LIS    Rz,4               If on IOP, then start
   PINT   Rz                 Queue service running
LABEL2     EQU   *                  Done
```

7.4.2 Interrupt Service Pointer Modification (ISPMOD)

For all Series 3200 Processors, except the IOP on the Model 3260 System, the following instruction is used by a device driver to modify the ISPT:

```
STH  Rx,ISPT(Rz,Rz)
```

On an IOP, this instruction will not produce the desired result, since the IOP ISPT is not located at "ISPT"(X'D0'), and is not halfword-indexed. Thus, the ISPMOD macro is provided.

The ISPMOD macro takes the value in Rx and places it in the proper entry in the ISPT (based on device address). On the CPU, each entry is a halfword. On an IOP, each entry is a fullword.

Calling convention:

```
label     ISPMOD ITEM=Rx,DCB=Ry,DN=Rz,WORK=Rm
```

Where:

Rx	is a register that contains the value that should be placed in the ISPT.
Ry	is a register that contains the address of the DCB.
Rz	is the device address.
Rm	is a work register.

This macro will generate the following code:

```
   EPSR  Rm,Rm           Get current PSW
   TI    Rm,PSW.NTM      Test for being on CPU or IOP
   BZ    LABEL1          If on CPU, go do usual code
   SLLS  Rz,2            If on IOP, make up fullword index
   L     Rm,DCB.ISP(Ry)  Get address of IOP's ISPT
   ST    Rx,0(Rm,Rz)     Modify the ISPT
   SRLS  Rz,2            Restore register value
   B     LABEL2          and exit
LABEL1    SLLS  Rz,1           On CPU, make up halfword index
   L     Rm,DCB.ISP(Ry)  Get address of CPU's ISPT
   STH   Rx,0(Rm,Rz)     Modify the ISPT
   SRLS  Rz,1            Restore the register contents
LABEL2    EQU   *
```

APPENDIX A

DESCRIPTION OF DATA STRUCTURES

TABLE A-1. DEVICE CONTROL BLOCK (DCB)

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
0 (0)	DCB.DMT	Fullword address of the device mnemonic table entry.
4 (4)	DCB.LEAF	Fullword address of LEAF
8 (8)	DCB.WCNT	Halfword value reflecting the number of logical units currently assigned for write operation. If this value is -1, a logical unit is assigned for exclusive write. If this value is 0, no logical units are assigned for writes. If this value is greater than zero, this number represents the number of logical units assigned for a write operation.
10 (A)	DCB.RCNT	Halfword value reflecting the number of logical units currently assigned for a read operation. Reference DCB.WCNT for the meaning of the specific values in this field.
12 (C)	DCB.FLGS	Fullword value reflecting the various states of the device. Breakdown of this fullword value — Bit Meaning 0 bulk device 1 on-line device 2 system message to console device 3 Event service routine (ESR) waiting for driver initialization routine to complete 4 active I/O time-out before driver initialization routine done

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
		5 delete pseudo device control block (DCB) 6 console identifier 7 uncancellable device 8 SVC6 connection table 9 write protected device 10 ITAM supported device 11 assigned for SVC 15 access 12 SVC1 halt I/O 13 time-out due to power fail 14 multiple DCB 15 pseudo DCB bit 16 supports vertical forms control (VFC) 17 power fail, no I/O outstanding 18 MMD type disk 19-31 reserved
16 (10)	DCB.1INC	Indicates that this device is being intercepted on an SVC1 level.
20 (14)	DCB.7INC	Indicates interception on SVC7.
24 (18)		Reserved
25 (19)	DCB.DCOD	Byte value reflecting the device code for this device.
26 (1A)	DCB.DN	Halfword value reflecting the hardware wired address of this device.
28 (1C)	DCB.ATRB	Halfword value reflecting which functions this device supports. The FMS7 module uses this field at ASSIGN time. Breakdown of this halfword value — Bit Meaning 0 interactive 1 supports read 2 supports write 3 supports binary 4 supports wait I/O 5 supports random

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
		6 supports unconditional proceed 7 supports image 8 supports halt I/O 9 supports rewind 10 supports backspace record 11 supports forward space record 12 supports write filemark 13 supports forward space filemark 14 supports backspace filemark 15 reserved
30 (1E)	DCB.RECL	Halfword value reflecting the record length of this device.
32 (20)	DCB.INIT	Fullword address of driver entry point for data transfer request.
36 (24)	DCB.FUNC	Fullword address of driver entry point for a device suported command request.
40 (28)	DCB.TERM	Fullword address of driver entry point for termination routine.
44 (2C)	DCB.TOUT	Halfword value reflecting the maximum amount of time any data transfer should take. If this field contains X'7FFFF', a time-out condition does not start. If this field contains X'FFFF' (-1), the transfer completed without a time-out condition. If this field contains X'0001' and X'7FFE', the system (LFC) clock decrements this field by one second until this value becomes an X'0000'.
46 (2E)	DCB.RTRY	Halfword value reflecting the maximum number of retrys the driver attempts if an error occurs.

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
48 (30)	DCB.WKEY	Byte value reflecting any write keys attributed to this device.
49 (31)	DCB.RKEY	Byte value reflecting any read keys attributed to this device.
50 (32)	DCB.ILVL	Halfword value reflecting the interrupt level of this device. The device controller must be hardware wired to be on one of the four interrupt levels. This field must agree with the hardware.
52 (34)	DCB.ERRL	Error logging data area pointer and error logging on/off switch used by the OS/32 Error Logging Facility.
56(38)	DCB.ISP	Fullword address used as pointer to ISPT for this device.
60 (3C)	DCB.TOCH	Fullword address pointer to next DCB on time-out chain only if the driver has put this DCB on the time-out chain. If this value is -1, this DCB is the last one on the time-out chain.
64 (40)	DCB.XFLG	Halfword value reflecting specific information about this device. Magnetic tape and disk drivers use this field. Breakdown of this halfword value — Bit Meaning 0 directory presence flag 1 bit map presence flag 2 reserved 3 bit map modify flag 4 reserved 5 I/O malfunction 6 bit map malfunction 7 hardware protect bit 8 restricted volume flag 9 magnetic tape at EOV label 10 last command to magnetic tape was REWIND

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
		11 magnetic tape is on or past EOT 12-15 reserved
66 (42)	DCB.CLAS	Halfword value reflecting the device class for OS/32 Accounting Facility
68 (44)	DCB.IOH	Fullword address of the IOH list that the SVC1 preprocessor uses. If a user supplied IOH list is desired, include the IOH list in the device driver.
72 (48)		Spare
73 (49)	DCB.LEVL	Byte address that specifies level for ADDSQ.
74 (4A)	DCB.IOP	Halfword address for index of input/output (IOP) in APBDIR (0 for CPU).
76 (4C)	DCB.IPB	Fullword address used as a pointer to input/output parameter block (IPB) for IOP.
80 (50)	DCB.SQ	Fullword address used as a pointer to processor's system queue.
84 (54)	DCB.Q	Fullword address of the specific queue strategy routine for this device.
88 (58)	DCB.EDMA	Fullword address of the extended direct memory access (EDMA) strategy routine. The default routine is EDMACON.
92 (5C)		Reserved
96 (60)	DCB.NXT	Fullword address of the next I/O block (IOB) waiting for this DCB.
100 (64)	DCB.RFLGS	Halfword value reflecting various traps that the user wants to take upon completion of device driver.

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
102 (66)	DCB.PRI	Byte value reflecting the priority of the task issuing the SVC1 request.
103 (67)	DCB.TYPE	Byte value defining the type of IOB used.
104 (68)	DCB.DONE	Fullword address of a user-supplied routine executed at driver-termination time in place of the standard OS/32 routine, IODONE.
108 (6C)	DCB.DCB	Fullword address of this DCB.
112 (70)	DCB.TCB	Fullword address of the task control block (TCB) of the task requesting the I/O.
116 (74)	DCB.QCB	Fullword address that points to connected QCB.
120 (78)	DCB.ESR	Fullword address reflecting the next entry point into the driver that the system queue service routine will schedule.
124 (7C)	DCB.UPBK	Fullword address reflecting the logical address (user task relative) of the parameter control block within memory.
128 (80)	DCB.PBLK	Fullword physical address of the SVC1 parameter block.
132 (84)	DCB.FC	Byte value reflecting user task requested function.
133 (85)	DCB.LU	Byte value reflecting the logical unit (lu) assigned to this device.
134 (86)	DCB.STAT	Byte value reflecting the device independent status after termination of the driver.

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
135 (87)	DCB.DDPS	Byte value reflecting the specific device dependent status after termination of the driver.
136 (88)	DCB.SADR	Fullword address of the user's buffer starting memory location.
140 (8C)	DCB.EADR	Physical address of the user's buffer ending memory address.
144 (90)	DCB.RAND	Fullword value reflecting the user supplied relative record number.
148 (94)	DCB.LLXF	Fullword value reflecting the length of data actually transferred.
152 (98)	DCB.SV1X	Fullword value reflecting the specific communications requests or 6250 Tape Drive requests.
156 (9C)	DCB.LUE	Fullword copy of the task's lu table entry.
160 (A0)	DCB.WCHN	Fullword address of the TCB waiting for the I/O to compete for a task.
168 (A8)	DCB.SIZE	Fullword value that represents the number of sectors on a disk; or if used for a vertical form control, it represents the number of lines on the device.
172 (AC)	DCB.VFC	Fullword address of VFCDCB.

TABLE A-2. CHANNEL CONTROL BLOCK (CCB)

DISPLACEMENT INTO STRUCTURE	LABEL	EXPLANATION
0 (0)	CCB.CCW	Halfword value consisting of a status mask and byte value describing the channel operation to be performed. Breakdown of this halfword — Bit Meaning 0-7 Status mask. The micro-code (ADC) ANDs the device status with the byte value contained in this field. If the result is zero, ADC continues with the data transfer. If the result is not zero, ADC vectors to the halfword address contained in CCB.SUBA and the condition code is set to X'1'. 8 Execute bit. If this bit is reset, ADC vectors to the halfword value contained in CCB.SUBA and the condition code is set to X'0'. If this bit is set, ADC continues with the data transfer. 9 Reserved. 10-11 Redundancy check bit. This byte specifies the type of redundancy checking required. Break down of this bit — Bit 10 / Bit 11 / Meaning 0 / 0 / LRC 0 / 1 / BISYNC CRC 1 / 0 / reserved 1 / 1 / SDLC CRC 12 Buffer select switch. If this bit is reset, buffer 0 is used. If this bit is set, buffer 1 is used. 13 Read/write bit. If this bit is reset, a byte is read from the device to the processor. If set, a byte is written from the processor to the device.

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
		14 Translation bit. If this bit is reset, translation does not occur. A 'TLATE' instruction is not required; but, the driver defined translation table is set as is for this instruction. Reference the Processor User's Manual. 15 Fast bit. If this bit is set, the fast data transfer mode occcurs--only buffer 0 is used; translation and redundancy checking do not occur. This bit must be set for halfword devices.
2 (2)	CCB.LB0	Halfword value reflecting the negative length of buffer 0. ADC adds one to this field and transfers a data byte until this value turns positive (not zero). When this value turns positive, ADC compliments the buffer select switch (if fast bit is reset), vectors to the halfword address contained in CCB.SUBA, and sets the condition code to X'2'.
4 (4)	CCB.EB0	Fullword ending address of the driver's buffer 0. Typically, the driver places this address contained in DCB.EADR into this field. To determine what byte to transfer, ADC adds the contents of CCB.LBO to CCB.EBO.
8 (8)	CCB.CW	Byte value containing the accumulated value of either cyclic or longitudinal redundancy checking. The initial value of this field is zero.
10 (A)	CCB.LB1	Halfword value reflecting the negative value of buffer 1. Reference the description of CCB.LBO.
12 (C)	CCB.EB1	Fullword ending address of the driver's buffer 1. Most drivers use this field to point to control characters to be sent to the device after LBO goes positive.
16 (10)	CCB.XLT	Fullword address of a driver defined translation table.

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
20 (14)	CCB.SUBA	Halfword address of a driver's interrupt service routine (ISR). The address contained in this field is vectored to if — -upon device interrupt, the execute bit within the CCB.CCW is reset, -LB0 or LB1 goes positive, or -bad status is received from the device.
22 (16)	CCB.MISC	Driver defined.
23 (17)	CCB.FLGS	Driver defined.
24 (18)	CCB.DCB	Fullword address of the DCB assigned for this data transfer or command function.
0 storage	CCB.DVDP	End of device-independent segment.
28 (1C)	CCB.XLT2	Fullword address for secondary translate table.

TABLE A-3. INPUT/OUTPUT (I/O) BLOCK

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
0 (0)	IOB.NXT	Fullword address of next IOB on IOB chain. If this field contains zero, this IOB is the last on the chain.
4 (4)	IOB.RFLG	Halfword value reflecting request dependent conditions and caller task states. Breakdown of this halfword — Bit Meaning 0 connection completed 1 caller in I/O wait 2 caller expects an I/O trap 3 caller in connection wait 4 ISPT reset flag, 1 equals no reset 5 EDMA exclusive request 6 EDMA connected flag 7 no disconnection at I/O completion 8 system I/O 9 no purge on error 10 request cannot be halted 11 wait pending request 12 reserved 13 do not free IOB at connect time 14 vertical forms control flag
6 (6)	IOB.PRI	Byte value reflecting the dispatch priority of the requesting task.
7 (7)	IOB.TYPE	Byte value defining the type of IOB. possible values and appropriate meanings. Value Meaning 1 in TCB pool; reserved at LINK time 2 spare TCB (one per each task) 3 contained in parameter control block (PCB) 4 contained in contiguous file's DCB

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
8 (8)	IOB.DONE	Fullword address of driver defined IODONE routine. If this field contains a zero, IODONE is executed.
12 (C)	IOB.DCB	Fullword address of the DCB for this request.
16 (10)	IOB.TCB	Fullword address of the TCB for this request.
20 (14)	IOB.QCB	Fullword address pointing to QCB for this IOB
24 (18)	IOB.ESR	Fullword address reflecting the entry into the driver's initialization ESR or termination ESR. System queue service (SQS) schedules both for execution.
28 (1C)	IOB.UPBK	Fullword unrelocated (logical) address of requester's PCB.
32 (20)	IOB.PBLK	Fullword relocated (physical) address of requester's PCB.
33 (21)	IOB.FC	Byte value reflecting requester's function code.
34 (22)	IOB.LU	Byte value reflecting requester's assigned lu.
35 (23)	IOB.STAT	Byte value reflecting the device-independent status.
36 (24)	IOB.DDPS	Byte value reflecting the device-dependent status.
40 (28)	IOB.SADR	Fullword starting address of the user's buffer.
44 (2C)	IOB.EADR	Fullword ending address of the user's buffer
48 (30)	IOB.RAND	Fullword value reflecting the relative record number if the request is for indexed file.

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
52 (34)	IOB.LFX	Fullword value for length of transfer.
56 (38)	IOB.SV1X	Extended SVC1 fullword.
60 (3C)	IOB.LUE	Fullword copy of lu table entry of task. Reference DCB.LUE.
64 (40)	IOB.WCHN	Fullword address of a TCB waiting for the I/O for this task to complete.
68 (44)	IOB.CYL	Halfword value for cylinder *2 for disk access.
70 (46)	IOB.SECT	Byte value for relative sector on disk for seek.
71 (47)	IOB.LSEC	Byte value for last relative sector used for seek.

TABLE A-4. EVENT COORDINATION NODE

DISPLACE-MENT INTO STRUCTURE	LABEL	EXPLANATION
0 (0)	EVN.CORD	Upper pointer
4 (4)	EVN.FLGS	Flags - must match QCB.FLGS
6 (6)	EVN.LOCK	T & S lock for multiprocessing
8 (8)		Spare
9 (9)	EVN.CLEV	Connection level
10 (A)	EVN.TSIZ	Tree size
14 (E)	EVN.SQS	SQS executor
18 (12)	EVN.DCB	DCB address
22 (16)	EVN.TCB	TCB address
26 (1A)	EVN.QCB	Pointer to current QCB
30 (1E)	EVN.EVRS	Save area for EVREL
32 (20)	EVN.NIO	Count of I/O's (connected + queued)
34 (22)	EVN.HWIO	HWM of EVN.NIO
38 (26)	EVN.CLC	Connected leaf chain
42 (2A)	EVN.PREV	Previous (node) pointer
46 (2E)	EVN.NEXT	Next (node) pointer
50 (32)	EVN.TOP	Top of waiting queue
54 (36)	EVN.BOT	Bottom of waiting queue
56 (38)	EVN.CYL	Current cylinder position
57 (39)	EVN.RDCT	Redispatch count
58 (3A)		Reserved
38 (26)	EVN.EMAX	EQU EVN.CLC - EDMA max transfers value
40 (28)	EVN.ECTR	EQU EVN.CLC + 2 - EDMA active transfer value
38 (26)	EVN.WRAP	EQU EVN.CLC - disk leaf secondary queue pointer

APPENDIX B

MACHINE STATES

Event Service (ES) State

- Used by:
  - System Queue Service (SQS)
  - Device Driver Events (initialization and termination
  - System events (clocks, power restore)
- Nontask State:
  - SQS Interrupts Disabled
  - No Context Block
  - SVCs are Illegal
- Register Set 5, if available, else set F.
- Entered via:
  - System Queue "Interrupt"
  - LPSW from Task Dispatcher
- Exits via: LPSW to Task Dispatcher
- This state uses register 0 thru 15 of set 5. The data input registers (DIRs) and intermediate event service routine (ESRs) and termination ESRs of drivers are executed in this state.

Nonreentrant System (NS) State

- Used by:
  - First Level Interrupt Handlers (Faults and service calls (SVCs))
  - Short SVC Second Level interrupt handlers
  - Task Dispatcher
- Nontask State:
  - SQS Interrupts Disabled
  - No Context Block
  - SVCs are Illegal
- Register Set 0
  (Restricted to Registers 8-F)

- Entered Only Via:
  - — Internal Interrupt (Fault or SVC)
  - — Call to Dispatcher (LPSW)
- Exit Via:
  - — TMNSOUT: Return to current task
  - — TMDISP: Dispatcher
  - — TMRS(A)IN: Enter RS(A) state
- All SVC and fault handlers execute in this state initially. Certain SVC and fault handlers may switch to another state. With respect to drivers, NS state is used by SVC1 and input/output handlers (IOH).

User Task (UT) State

- Tasklevel States:
  - — Use "User" Context Block at TCB.UT
  - — SQS Enabled
  - — SVCs are Legal
- UT State PSW:
  - — Relocation/Protection Enabled
  - — Privileged Instruction Illegal
- Entered via:
  - — Task Dispatcher
  - — TMNSOUT
- Exit via:
  - — Interrupt (SVC or Fault)
- Uses registers 0 through 15 of set 15.
- User programs, e-tasks, d-tasks, the operation system (OS) command processor, console monitor and loader execute in this state. Drivers do not execute in this state.

Interrupt Service (IS) State

- Uses registers 0 through 7 of set 0 for level 0. For levels 1 through 3, the IS state uses register 0 through 15 of sets 1 through 3, respectively.
- The interrupt service routines (ISRs) of drivers execute in this state.

Reentrant System (RS) State

- Uses registers 0 through 15 of set 6.
- Certain SVCs, fault handlers and file managers execute in this state. Drivers do not use this state.

Figure B-1 depicts the various machine states and their associated registers and register sets.

190-4

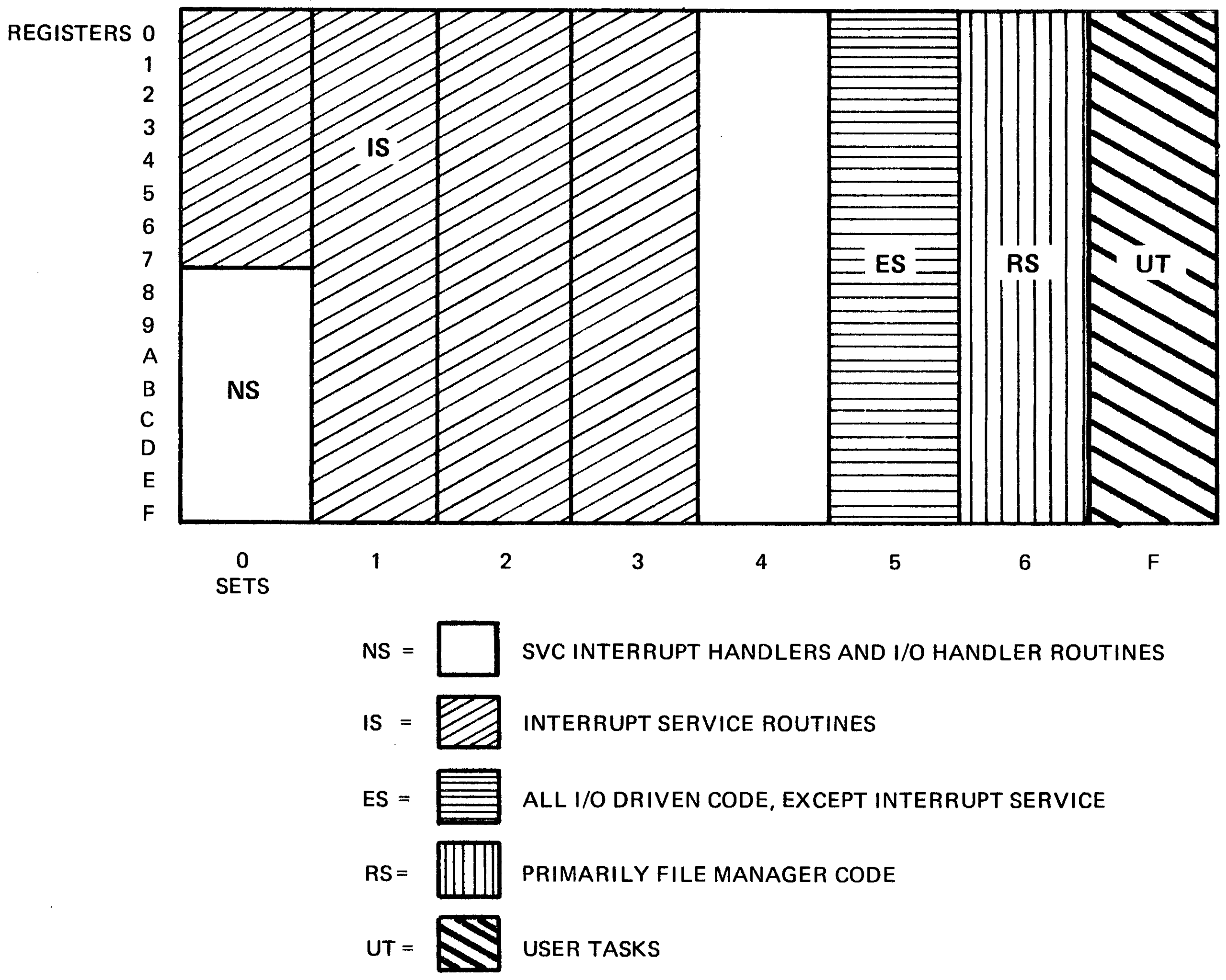
REGISTERS 0
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F
NS
IS
ES
RS
UT
0
SETS
1
2
3
4
5
6
F
NS =
SVC INTERRUPT HANDLERS AND I/O HANDLER ROUTINES
IS =
INTERRUPT SERVICE ROUTINES
ES =
ALL I/O DRIVEN CODE, EXCEPT INTERRUPT SERVICE
RS =
PRIMARILY FILE MANAGER CODE
UT =
USER TASKS

Figure B-1. Machine States and Associated Registers and Registers Sets

Figure B-2 depicts driver machine states and the driver routines, in sequence, that are associated with them.

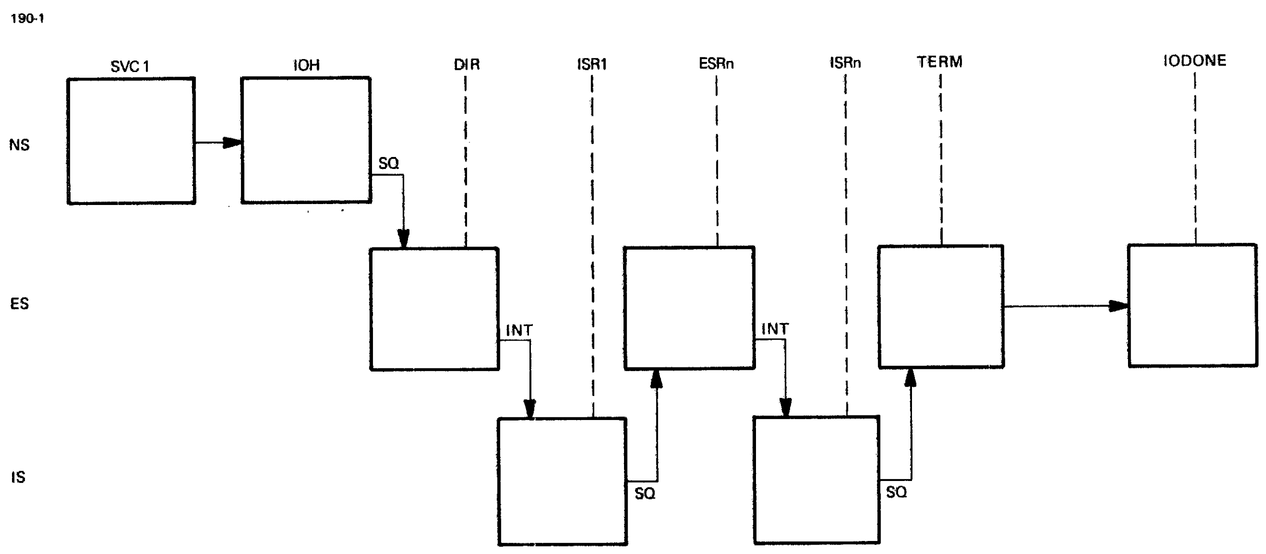

Figure B-2 Machine States and Associated Driver Routines In their Proper Time Sequence

Where:

SQ	is scheduled by adding the leaf for the device to the system queue.
INT	is scheduled by executing a SINT instruction or by an interrupt from a device.

APPENDIX C

OS/32 MACROS

THE SYSGEN/32 MACRO OUTPUT FILE

NOTE

For the SYSGEN/32 macro input file, see Chapter 7.

```
IMPUR
MCALL DCBI,CCBI,CONVNUM,EVNGEN,$TABL$
MCALL BIOCGEN,DCB39
MCALL MTPI,DCB64
MCALL DCB246
SPTINIT MLBL=120,CSLV=5,CSBF=122,ISPT=500,MTOP=2048,NTCB=51,   C
      SVOL=OS32,RVOL=FIXD,TVOL=FIXD,FREQ=60,PIC=6C,LFC=6D,     C
      CPU=3230,SLICE=200,SPVL=FIXD,VERSN=OS32LAB,CLASS=4,      C
      SOPT=92F67000,ERBL=X
DMT  CON,LAB1,LAB2,CRT1,CRT2,PRT,PR,MAGO,MAG1,DSC1
DMT  DSC2
DMTEND
 SPDMT  PR
 DCBINIT
DCB39 DCOD=39,DN=16,NAME=CON,CONS=1,ILVL=0,XDCD=X'280D',       C
      RECLN=120
DCB39 DCOD=39,DN=18,NAME=CRT1,ILVL=0,XDCD=X'280D'
DCB39 DCOD=39,DN=20,NAME=CRT2,ILVL=0,XDCD=X'280D'
DCB113 DCOD=113,DN=98,NAME=PRT,ILVL=0
DCB1 DCOD=1,DN=0,NAME=PR,ILVL=0,XDCD=X'71'
DCB64 DCOD=64,DN=133,NAME=MAGO,ILVL=0,SLCH=F0,CNTR=1
DCB64 DCOD=64,DN=197,NAME=MAG1,ILVL=0,SLCH=F4,CNTR=2
DCB51 DCOD=51,DN=198,NAME=DSC1,ILVL=0,SLCH=F0,CNTR=B6
DCB50 DCOD=50,DN=199,NAME=DSC2,ILVL=0,SLCH=F0,CNTR=B6,         C
      SHCCB=DSC1
 DCBTERM
DCB246  COUNT=10,DCOD=246,DN=208,NAME=LAB1,ILVL=0
DCB246 DCOD=246,DN=216,NAME=LAB2,ILVL=0
VMTGEN 2
 EVNGEN DCOD=39,NAME=CON,TSIZ=1,NUM=0
 EVNGEN DCOD=246,NAME=LAB1,TSIZ=1,NUM=0
 EVNGEN DCOD=246,NAME=LAB2,TSIZ=1,NUM=1
 EVNGEN DCOD=39,NAME=CRT1,TSIZ=1,NUM=1
 EVNGEN DCOD=39,NAME=CRT2,TSIZ=1,NUM=2
 EVNGEN DCOD=113,NAME=PRT,TSIZ=1,NUM=0
 EVNGEN DCOD=64,NAME=MAG0,TSIZ=3,NUM=0,TYP1=C,COORD=1,UNOD=1
 EVNGEN DCOD=64,NAME=MAG1,TSIZ=3,NUM=1,TYP1=C,COORD=2,UNOD=1
 EVNGEN DCOD=51,NAME=DSC1,TSIZ=3,NUM=0,TYP1=C,COORD=B6,UNOD=1,C
       DA=1
 EVNGEN DCOD=50,NAME=DSC2,TSIZ=3,NUM=0,TYP1=C,COORD=B6,UNOD=1,C
       DA=1,BITDIR=1,LAST=1,ERBL=1,CPU=3230
 EVNGEN DCOD=1,NAME=C,TSIZ=0,NUM=0,TYP1=S,COORD=F0,UNOD=1,     C
       TYP=C,FLGS=0
 EVNGEN DCOD=2,NAME=C,TSIZ=0,NUM=0,TYP1=S,COORD=F4,UNOD=1,     C
       TYP=C,FLGS=0
 EVNGEN DCOD=B6,NAME=C,TSIZ=0,NUM=0,TYP1=S,COORD=F0,UNOD=1,    C
       TYP=C,FLGS=0
 EVNGEN DCOD=F0,NAME=S,TSIZ=0,NUM=0,TYP=S,FLGS=0
 EVNGEN DCOD=F4,NAME=S,TSIZ=0,NUM=0,TYP=S,FLGS=0
FLTPINIT U=0,CPU=3230,REGS=8
EXTRN   UBOT.F01
EXTRN   CDVR.F01
EXTRN   CMDB.F33
EXTRN   CMEX.F33
EXTRN   CMON.F01
EXTRN   CMSP.F33
EXTRN   CMIR.F02
EXTRN   ERRC.F03
EXTRN   EXAC.F02
EXTRN   EXIN.F53
EXTRN   EXIO.F02
EXTRN   EXLD.F53
EXTRN   EXMY.F53
EXTRN   EXSP.F53
EXTRN   EXSV.F53
EXTRN   EXTI.F01
EXTRN   EXTM.F53
EXTRN   FMC0.F33
EXTRN   FMIN.F33
EXTRN   FMS7.F33
EXTRN   FMA7.F33
EXTRN   FMB7.F33
EXTRN   FMUT.F33
EXTRN   INTC.F02
EXTRN  ITFM.M00
EXTRN  MCHK.F02
EXTRN  APSV.F01
IVTGEN CSL='CON',MXBX=32000,MXPRI=10,SYSS=204800,                C
      ERFD='OS32ERROR   LOG',ERDS=200,ERDP=2,PWRDLAY=0,          C
      PWRMODE=MANUAL
SVTGEN MBLK=100,DBLK=5,IBLK=1,DSPL=5,ISPL=1,DNBF=64,INBF=3
DFLIST 15,SQ
 SMCONFIG BLOCK=0,START=0,RANGE=2,INTERL=0
STARTUP
END
```

THE DEVICE CODE::: MACRO

NOTE

This device control block (DCB) is extracted from DCBFORM in the SYSGEN/32 macro library.

```
  MACRO                                                      col  72
DCB:::     %DCOD=,%DN=,%CLAS=,%ILVL=,%NAME=,%QU=,                  1
      %CONS=,%SLCH=,%CNTR=,%XDCD=,%SPND=,                          2
      %RECLN=,%SPCR=,%SPCW=,%XLT=,%PDCT=,%LDCT=,                   3
      %SHCCB=,%POLMT=,%EOV=0,%CLOCK=,%SIZE=,%DUAL=,                4
      %SCR=,%RES=%CM=,%ITV=,%IOLM=,%SLS=,%MNOF=,%MBFS=,            5
      %MRBS=,%MTO=,%N2=,%NC=,%OTV=,%PLDT=,%PLTC=,%SSA=,            6
      %T1=,%UCSI=,%UCSO=,%WKTC=

DEFINE GLOBAL COUNTER FOR DEVICE CODES

      GBLC  %IDVAL  DECLARE DEVICE ALPHA GLOBAL VARIABLE

      BGBLA %ID:::    DECLARE DEVICE GLOBAL VARIABLE

      LCLA  %CCBFL    LOCAL VARIABLES AS NEEDED
      LCLA  %CLASN
      LCLC  %RXLT,%RQU
      LCLC  %CORDNM,%PTRPAS
      LCLC  %OFFS
```

```
         LCLA  %RDN
         LCLC  %MDN,%MCNT,%MSLCH
         LCLA  %TRCNT,%UPTR
         LCLB  %FOUND,%DA
         BGBLA %FIRST
%RQU     SETC  'COMQ'                DEFAULT DEVICE QHANDLER
%MDN     SETC  '%DN'                 DEVICE ADDRESS
%MCNT    SETC  '%CNTR'               CONTROLLER
%MSLCH   SETC  '%SLCH'               SELCH
%CCBFL   SETA  0
         AIF   (T'%CLAS EQ 'U')&CLSNTD
%CLASN   SETA  %CLAS*12             IOCLASS*12
&CLSNTD  ANOP
                                    MACRO CALL TO
         USERINIT                   INITIALIZE STRUC COPY FLAGS

          CONVNUM   VAL=%ID:::       CONVERT %ID::: TO ALPHA
                                     %IDVAL RETURNED WITH ALPHA
                                     VALUE OF %ID:::
%ID:::    SETA      %ID:::+1         BUMP UP FOR NEXT TIME


DCB%NAME     PROG         DCB PROGRAM LABEL FOR LINK MAP


****************************************************************
&DCBOPT ANOP
****************************************************************
```

Now replace all values passed in the macro call.

```
%OFFS    SETC  '%DCOD':'%IDVAL'       ESTABLISH PROPER OFFSET
```

(The label %DCB:%DCOD:%IDVAL is established as the start address of the DCB in DCBI.)

The following is a sample of how some of the passed parameters may be handled.

```
 DEFINE SYSTEM DEPENDANT FIELDS OF DCB FIRST
         ORG    DCB%OFFS+DCB.DN         DEVICE ADDRESS
         DC     H'%DN'
```

A separate leaf is created unless the %SHCCB parameter is passed. If it is passed, it will contain the name of the shared busy device. The leaf is created by the EVNGEN macro (see the SYSGEN/32 Macro Output file above).

```
         ORG    DCB%OFFS+DCB.LEAF       LEAF POINTER
         AIF    (T'%SHCCB' EQ 'U')&NSLEAF   B IF NOT SHARED
         DAC    LF%SHCCB              USE SHARED DEVICE LEAF
         EXTRN LF%SHCCB
         AGO    &NRMLFX
&NSLEAF  ANOP
         DAC    LF%OFFS              GENERATE STANDARD LEAF NAME
         EXTRN LF%OFFS
&NRMLFX  ANOP
&NOLEAF  ANOP


 DEFINE ALTERNATE NAME FOR DMT MACRO
DCB_%NAME EQU   DCB%OFFS
         ENTRY DCB_%NAME
         ORG    DCB%OFFS+DCB.DMT
         DC     DMT_%NAME        A(DMT ENTRY)
         EXTRN DMT_%NAME

         AIF    (T'%CLAS EQ 'U')&NOCLAS
         ORG    DCB%OFFS+DCB.CLAS       IO CLASS
         DC     H'%CLASN'            IOCLASS12
&NOCLAS  ANOP

         AIF    (T'%ILVL EQ 'U')&NOILVL
         ORG    DCB%OFFS+DCB.ILVL       ILEVEL
         DC     H'%ILVL'
&NOILVL  ANOP
```

```
       CONTINUE WITH OTHER DCB OPTIONS (DEVICE-DEPENDENT)

         AIF    (T'%XDCD EQ 'U')&NOXDCD   IF NOT ENTERED
         ORG    DCB%OFFS+DCB.XDCD            ELSE MOVE XDCD
         DC     %XDCD                EXTENDED DCOD
&NOXDCD  ANOP

         AIF    (T'%SLCH EQ 'U')&NOSLCH
         ORG    DCB%OFFS+DCB.SDN       SELCH
         DCX    %SLCH
&NOSLCH  ANOP

         AIF    (T'%CNTR EQ 'U')&NOCNTR
         ORG    DCB%OFFS+DCB.CDN       CONTROLLER
         DCX    %CNTR
&NOCNTR  ANOP

         AIF    (T'%SPND EQ 'U')&NOSPND
         ORG    DCB%OFFS+DCB.XDCD
         DC     H'%SPND'            FLOPPY SPINDLE
         DC     H'0'
&NOSPND  ANOP

         AIF    (T'%QU EQ 'U')&DEFQU
         ORG    DCB%OFFS+DCB.Q      IO QUEUING
         DAC    %QU
         EXTRN  %QU
         AGO    &NOQU
&DEFQU   ANOP

         AIF    ('%RQU' EQ '')&NOQU
         ORG    DCB%OFFS+DCB.Q
         DAC    %RQU
         EXTRN  %RQU
&NOQU    ANOP

         AIF    (T'%CLOCK EQ 'U')&NOCLOCK
         ORG    DCB%OFFS+DCB.CCB+6
         DC     %CLOCK             DEFINE CLOCK
&NOCLOCK ANOP

         AIF    (T'%RECLN EQ 'U')&NORECLN
         ORG    DCB%OFFS+DCB.RECL      RECORD SIZE
         DC     H'%RECLN'
&NORECLN ANOP

         AIF    (T'%SPCR EQ 'U')&NOSPCR
         ORG    DCB%OFFS+DCB.SPCR
         DC     %SPCR                SPECIAL READ CHARACTER
&NOSPCR  ANOP

         AIF    (T'%SPCW EQ 'U')&NOSPCW
         ORG    DCB%OFFS+DCB.SPCW
         DC     %SPCW                SPECIAL WRITE CHARACTER
&NOSPCW  ANOP

         AIF    (T'%SCR EQ 'U')&NOSCR
         ORG    DCB%OFFS+DCB.TO1
         DC     %SCR                 SCREEN TRANSMIT TIME
&NOSCR   ANOP

         AIF    (T'%RES EQ 'U')&NORES
         ORG    DCB%OFFS+DCB.TO2
         DC     %RES                 RESPONSE TIME
&NORES   ANOP

         AIF    (T'%XLT EQ 'U')&DEFXLT
         ORG    DCB%OFFS+DCB.XLT
         DAC    %XLT                 TRANSLATION TABLE
         EXTRN  %XLT                 AND EXTRN
         AGO    &NOXLT
&DEFXLT  ANOP

         AIF    ('%RXLT' EQ '')&NOXLT
         ORG    DCB%OFFS+DCB.XLT     TRANSLATION TABLE
         DAC    %RXLT
         EXTRN  %RXLT                AND EXTERN
&NOXLT   ANOP

         AIF    (T'%LDCT EQ 'U')&NOLDCT
         ORG    DCB%OFFS+DCB.LDCT
         DC     H'%LDCT'             LEAD. CHAR. COUNT
&NOLDCT  ANOP

         AIF    (T'%PDCT EQ 'U')&NOPDCT
         ORG    DCB%OFFS+DCB.PDCT
         DC     H'%PDCT'             PAD COUNT
&NOPDCT  ANOP

         AIF    (T'%POLMT' EQ 'U')&NOMXEC
         ORG    DCB%OFFS+DCB.MXEC
         DC     %POLMT                MAX POLL ERR RETRY- POLL LIMIT
&NOMXEC  ANOP

         AIF    (T'%DUAL' EQ 'U')&NODUAL
         ORG    DCB%OFFS+DCB.DUAL      DUAL PORT OPTION
         DC     X'FF'
&NODUAL  ANOP

         ORG    $ST%OFFS             ORG TO END OF DCB
%RDN     SETA   %DN+1
         USEREND                     MUST END THIS SOURCE MODULE
         MEND


         MEND
```

THE DCBI MACRO

```
     MACRO
%SYM     DCBI   %DCOD=,%INIT=,%TERM=,%TOUT=7FFF,%DA=,%IOC=0,          1
                %IOH=COMIOH,%EDMA=0,%COPY=$DCB$,                     2
                  %FUNC=0,%ATRB=0,%RECL=0,%SIZE=DCB.DVDP+4,           3
                  %FLGS=DFLG.LNM,%ID=,%DSIZE=,%STRK=,%TCYL=,          4
                %BMSA=,%DRSA=,%DSC=,%NUM=,%SADR=,%SHCC=0,             5
                %CC4=,%CLOC=,%CCB=,                                  6
                %PFUN=,%PXLT=,%RXLT=,                                7
                %PSEP=,%EOLC=,%RTRY=
         LCLA   %IOCLN

%IOCLN   SETA   %IOC*12               IOCLASS * 12
         TITLE  DCB%DCOD%ID
         IMPUR
         ENTRY  DCB%DCOD%ID
         AIF    ('%INIT' EQ 'O')&NOINIT
         EXTRN  %INIT
&NOINIT  ANOP
         AIF    ('%TERM' EQ 'O')&NOTERM
         EXTRN  %TERM
&NOTERM  ANOP
         EXTRN  %IOH
         %COPY
D%SYSINDX EQU   *
DCB%DCOD%ID EQU *
         NLIST
         DO     %SIZE/4
         DAC    0
         LIST
$ST%DCOD%ID EQU *                     MARK END OF DCB
         AIF    (T'%EDMA EQ 'U')&NEDMA
         ORG    D%SYSINDX+DCB.EDMA
         DAC    %EDMA                 DEFINE EDMA
         AIF    ('%EDMA' EQ 'O')&NEDMA
         EXTRN  %EDMA
&NEDMA   ANOP
         ORG    D%SYSINDX+DCB.INIT
         DAC    %INIT                 DRIVER INIT
         AIF    ('%FUNC' EQ 'O')&NOEXT
         EXTRN  %FUNC                 DRIVER FUNCTION ROUTINE
&NOEXT   ORG    D%SYSINDX+DCB.FUNC
         DAC    %FUNC
         ORG    D%SYSINDX+DCB.TERM
         DAC    %TERM                 DRIVER TERMINATION
         ORG    D%SYSINDX+DCB.ATRB
         DCX    %ATRB                 DEVICE ATTRIBUTES
         ORG    D%SYSINDX+DCB.CLAS
         DC     H'%IOCLN'               IO CLASS
         ORG    D%SYSINDX+DCB.RECL
         DC     H'%RECL'              RECORD LENGTH
         ORG    D%SYSINDX+DCB.TOUT
```

```
        DCX   %TOUT                TIME-OUT CONSTANT
        ORG   D%SYSINDX+DCB.FLGS
        DAC   %FLGS                DEVICE FLAGS
        ORG   D%SYSINDX+DCB.DCOD
        DB    %DCOD                DEVICE CODE
        ORG   D%SYSINDX+DCB.IOH
        DAC   %IOH                 IO HANDLER
        ORG   D%SYSINDX+DCB.DCB
        DAC   D%SYSINDX            DCB ADDRESS
        AIF   (T'%SADR EQ 'U')&NSADR
        ORG   D%SYSINDX+DCB.SADR
        DAC   %SADR                SADR
        EXTRN %SADR
&NSADR  ANOP
        AIF   (T'%DA EQ 'U')&NODA
        ORG   D%SYSINDX+DCB.DIRL   POINT TO DIR.LEAF
        DAC   DIR%DCOD%ID          IF DIR.ACCESS
        EXTRN DIR%DCOD%ID
        ORG   D%SYSINDX+DCB.BITL
        DAC   BIT%DCOD%ID          BIT.LEAF ALSO
        EXTRN BIT%DCOD%ID
&NODA   ANOP
        AIF   (T'%CC4 EQ 'U')&NCC4
        ORG   D%SYSINDX+DCB.CCB+4
        DB    X'%CC4',0
&NCC4   ANOP
        AIF   (T'%CLOC EQ 'U')&NCLOC
        ORG   D%SYSINDX+DCB.CCB+6
        DCX   %CLOC               CLOCK SPEC.
&NCLOC  ANOP
        AIF   (T'%DSIZE EQ 'U')&NDSIZE
        ORG   D%SYSINDX+DCB.SIZE
        DC    %DSIZE               DISC SIZE
        ORG   D%SYSINDX+DCB.STRK
        DC    X'%STRK'             SECTORS/TRACK
        ORG   D%SYSINDX+DCB.TCYL
        DC    X'%TCYL'             TRACKS/CYL
&NDSIZE ANOP
        AIF   (T'%BMSA EQ 'U')&NBMSA
        ORG   D%SYSINDX+DCB.BMSA     BITMAP BUFFER
        DC    D%SYSINDX+DCB.:%BMSA
        DC    D%SYSINDX+DCB.:%BMSA+255
&NBMSA  ANOP
```

```
         AIF    (T'%DRSA EQ 'U')&NDRSA
         ORG    D%SYSINDX+DCB.DRSA    DIR.BUFFER
         DC     D%SYSINDX+DCB.:%DRSA
         DC     D%SYSINDX+DCB.:%DRSA+255
&NDRSA   ANOP
         AIF    (T'%DSC EQ 'U')&NDSC
         ORG    D%SYSINDX+DCB.DSC
         DC     %DSC                 DSC
         EXTRN  %DSC
&NDSC    ANOP
         AIF    (T'%NUM EQ 'U')&NNUM
         ORG    D%SYSINDX+DCB.NUM
         DC     X'%NUM'              NUM
&NNUM    ANOP
         AIF    (T'%PFUN EQ 'U')&NPFUN
         ORG    D%SYSINDX+DCB.PFUN
         DC     H'%PFUN'             PFUN
&NPFUN   ANOP
         AIF    (T'%PXLT EQ 'U')&NPXLT
         ORG    D%SYSINDX+DCB.PXLT   FOR CARD RDR PUNCH
         DAC    %PXLT                PUNCH TRANSATION
         EXTRN  %PXLT
&NPXLT   ANOP
         AIF    (T'%RXLT EQ 'U')&NRXLT
         ORG    D%SYSINDX+DCB.RXLT
         DAC    %RXLT                FOR CARD RDR PUNCH
         EXTRN  %RXLT                READ TRANSLATION
&NRXLT   ANOP
         AIF    (T'%PSEP EQ 'U')&NPSEP
         ORG    D%SYSINDX+DCB.PSEP   FOR CARD RDR PUNCH
         DC     X'%PSEP'             PUNCH SEPARATE
&NPSEP   ANOP
         AIF    (T'%EOLC EQ 'U')&NEOLC
         ORG    D%SYSINDX+DCB.EOLC
         DCX    %EOLC(1)             PRINTER EOLC
         DCX    %EOLC(2)
&NEOLC   ANOP
         AIF    (T'%RTRY EQ 'U')&NRTRY
         ORG    D%SYSINDX+DCB.RTRY
         DCX    %RTRY                RETRY
&NRTRY   ANOP
         AIF    ('%SHCC' NE 'O')&NOCCBX   IF NOT O DONT DO DCB.CCB
         ORG    D%SYSINDX+DCB.CCB    DCB.CCB
         AIF    (T'%CCB EQ 'U')&NOCCBX
         DC     Z(CCX%DCOD%ID)        GENERATE SECOND CCB
&NOCCBX  ANOP
         ORG    $ST%DCOD%ID           BACK TO END OF DCB
         MEND
```

THE CCBI MACRO

```
         MACRO
%SYM     CCBI   %DCOD=,%XLTAB=,%SUBA=,%EBO=,                              1
                %CCW=,%CFLGS=,%ID=,%CCBN=
         $CCB
         PURE
         AIF    (T'%CCBN NE 'U')&CCBNX  2ND CCB BRANCH
         TITLE CCB%DCOD%ID            DEFINE FOR 1ST CCB
         ENTRY CCB%DCOD%ID
CCB%DCOD%ID EQU *
         AGO    &CCBN5                B OVER 2ND CCB INIT.
&CCBNX   ANOP
         TITLE CCX%DCOD%ID            DEFINE FOR 2ND CCB
         ENTRY CCX%DCOD%ID
CCX%DCOD%ID EQU *
&CCBN5   ANOP
C%SYSINDX EQU   *
         DO     CCB+3/4               CLEAR CCB AREA
         DAC    0
         AIF    (T'%CCBN EQ 'U')&CCBN7  B IF FIRST CCB
$SX%DCOD%ID EQU *                LABEL FOR 2ND CCB
         AGO    &CCBN9
&CCBN7   ANOP
$S%DCOD%ID EQU *                   MARK END OF CCB
```

```
&CCBN9   ANOP
         ORG   C%SYSINDX+CCB.DCB   CCB.DCB
         DAC   DCB%DCOD%ID
         AIF   (T'%XLTAB EQ 'U')&NOXLTAB
         ORG   C%SYSINDX+CCB.XLT   TRANSLATE TABLE
         DAC   %XLTAB
         EXTRN %XLTAB
&NOXLTAB ANOP
         AIF   (T'%SUBA EQ 'U')&NOSUBA
         ORG   C%SYSINDX+CCB.SUBA  CCB.SUBA
         DC    Z(%SUBA)
         EXTRN Z(%SUBA)
&NOSUBA  ANOP
         AIF   (T'%CCW EQ 'U')&NOCCW
         ORG   C%SYSINDX+CCB.CCW   CCB.CCW
         DC    X'%CCW'
&NOCCW   ANOP
         AIF   (T'%CFLGS EQ 'U')&NOCFLGS
         ORG   C%SYSINDX+CCB.FLGS  CCB.FLGS
         DB    X'%CFLGS'
&NOCFLGS ANOP
         AIF   (T'%EBO EQ 'U')&NOEBO
         ORG   C%SYSINDX+CCB.EBO   CCB.EBO
         DC    Y'%EBO'
&NOEBO   ANOP
         ORG   $S%DCOD%ID          ORG TO END OF CCB
         AIF   (T'%CCBN EQ 'U')&DONE
         ORG   $SX%DCOD%ID         ORG TO END OF SECOND CCB
&DONE    ANOP
         MEND
```

THE USERINIT MACRO

```
         MACRO
         USERINIT
         GBLB  %CCB,%PDCB,%DFLG,%SDCB,%VFCDCB
         GBLB  %DCB$,%DDCB,%MMDDX,%DDEX
         GBLB  %MTP,%BIOCDCB,%LPTDCB,%CDRP
         GBLB  %ICCB,%IDCB,%ODCB,%S125DCB,%BDCB,%ADCB,%AOBDCB
         GBLB  %ETHDCBS,%ETHSTCM
         GBLB  %EVN,%IOB$,%IOB,%RCTX,%TCB
%CCB     SETB  0                   $CCB     - CHANNEL CONTROL BLOCK
%PDCB    SETB  0                   $PDCB    - DEVICE INDEPENDENT DCB
%DFLG    SETB  0                   $DFLG    - DCB FLAGS
%SDCB    SETB  0                   $SDCB    - SPOOL DCB
%VFCDCB  SETB  0                   $VFCDCB  - VFC DCB
%DCB$    SETB  0                   $DCB$    - DISK DCB
%DDCB    SETB  0                   $DDCB    - DISK DEPENDENT DCB
%MMDDX   SETB  0                   $MMDDX   - MMD DISK DCB
%DDEX    SETB  0                   $DDEX    - ERROR LOGGER DATA AREA
%MTP     SETB  0                   $MTP     - MAG TAPE DCB
%BIOCDCB SETB  0                   $BIOCDCB - BIOC CRT DCB
%LPTDCB  SETB  0                   $LPTDCB  - LINE PRINTER DCB
%CDRP    SETB  0                   $CDRP    - CARD READER/PUNCH DCB
%ICCB    SETB  0                   $ICCB    - DATA COMM CCB
%IDCB    SETB  0                   $IDCB    - BASIC DATA COMM DCB
%ODCB    SETB  0                   $ODCB    - PE 1200 DCB
%S125DCB SETB  0                   $S125DCB - PE 1250 DCB
%BDCB    SETB  0                   $BDCB    - BISYNC DCB
%ADCB    SETB  0                   $ADCB    - ASYNC DCB
%AOBDCB  SETB  0                   $AOBDCB  - DATA COMM DCB
%ETHDCBS SETB  0                   $ETHDCBS - ETHERNET DCB
```

```
%ETHSTCM SETB  0                    $ETHSTCM - ETHERNET DCB
%EVN     SETB  0                    $EVN     - LEAF/NODE
%IOB$    SETB  0                    $IOB$    - I/O BLOCK & FLAGS
%IOB     SETB  0                    $IOB     - I/O BLOCK
%RCTX    SETB  0                    $RCTX    - RSA CONTEXT BLOCK
%TCB     SETB  0                    $TCB     - TASK CONTROL BLOCK
         MEND
```

THE USEREND MACRO

```
         MACRO
         USEREND
         ASIS
         END                       MUST END THIS MODULE
         MEND
```

IOP MACROS

ISPMOD

```
%LABEL   ISPMOD     %ITEM=,%DCB=,%DN=,%WORK=
         EPSR       %WORK(1),%WORK(1)            FETCH CURRENT PSW
         TI         %WORK(1),PSW.NTM             CPU OR IOP?
         BZ         ISPC%SYSINDX                 CPU ->
         SLLS       %DN(1),2                     FULLWORD INDEX
         L          %WORK(1),DCB.ISP(%DCB(1))    ISP ADDRESS FOR THIS
                                                 PROCESSOR
         ST         %ITEM(1),0(%WORK(1),%DN(1))  STUFF ENTRY
         SRLS       %DN(1),2                     RESTORE
         B          ISPX%SYSINDX                 EXIT
```

```
ISPC%SYSINDX EQU *
        SLLS        %DN(1),1                   HALFWORD INDEX
        L           %WORK(1),DCB.ISP(%DCB(1))  ISP ADDRESS FOR THIS
                                               PROCESSOR
        STH         %ITEM(1),0(%WORK(1),DN(1)) STUFF ENTRY
        SRLS        %DN(1),1                   RESTORE
ISPX%SYSINDX EQU *
        MEND
```

ADDSQ

```
%LABEL  ADDSQ        %ITEM=,%DCB=,%WORK=
%LABEL  LABEL
        L            %WORK(1),DCB.SQ(%DCB(1))  SQ OR SIQ ADDRESS
        BNZ          ADDI%SYSINDX              NON-ITAM DEVICE
        LA           %WORK(1),SQ               FOR ITAM DEVICES
ADDI%SYSINDX EQU *
        ATL          %ITEM(1),0(%WORK(1))      STUFF THE ITEM
        EPSR         %WORK(1),%WORK(1)         CURRENT PSW
        TI           %WORK(1),PSW.NTM          CPU OR IOP?
        BZ           ADD%SYSINDX               CPU ->
        LIS          %WORK(1),4                LEVEL 4...
        PINT         %WORK(1)                  ...KICKOFF
ADD%SYSINDX EQU *
        MEND
```

APPENDIX D

OS/32 SUBROUTINE DEFINITIONS

SERVICE ROUTINES

NAME:	IODGST
ABSTRACT:	This routine returns the status to the SVC1 parameter block.
ENTRYS:	IODGST, IODGST2
SOURCE LIBRARY ROUTINES:	DCB, IOB, SVC1
EXTRNS:	III, ISPTAB
REGISTER USED:	R4, R6, RC, RE, RF

ON ENTRY	R4 = Return address
	R6 = A (DCB)

ON EXIT:	RC, RE, RF destroyed RF contains status codes CC is negative if bad status returned

PRINCIPLES OF OPERATION:

At entry, RC is loaded with the address of the parameter block and the interrupt service pointer table (ISPT) reset flag is tested. If it is not reset, it is done by loading RF with address of III and setting it at the entry for this request in ISPT. The FC is loaded from the DCB and tested for a command request. If not command request, move transfer length (LLXF) from device control block (DCB) to parameter block, which is skipped for command. Next the STAT field is transferred from DCB to PB and a BZR R4 is executed if this field is zero. If nonzero, the 'L' flag of CC is set and exit is performed via R4.

The alternate entry IOPGST2 is the same as IODGST except that the testing and resetting ISPTAB code is skipped.

NAME:	IODTWT
ABSTRACT:	This routine tests for TRAP and WAIT bits and performs the requested function.
ENTRYS:	IODTWT, IODTWT2
SOURCE LIBRARY ROUTINES:	IOB, TSW
EXTERNS:	TMREMW, SV9.ATQ1
REGISTERS USED:	R4, R5, R8-RA, RD, RF

ON ENTRY:	R4 = Retrun address
	R5 = A (IOB)

ON EXIT:	R8-RF may be destroyed

PRINCIPLES OF OPERATION:

At entry, R9 is loaded with the address of the task control block (TCB) and the input/output block (IOB) request dependent flag field (IOB.RFLG) is tested to see if caller expects an input/output (I/O) TRAP. If so, the reason code of the task's trap queue is set to X'08'; a BAL to SV9.ATQ1 is performed to make the queue entry with the unrelocated parameter block address in register RA. If no trap is expected, this operation is skipped, in which case both paths lead to the checking of the flag field of IOB again, this time for the I/O WAIT bit. If this bit is reset, then return via R4. If I/O WAIT is indicated, RD is loaded with Y'8000' and a BAL to TMREMW is performed; upon return from the task manager, an exit via R4 is performed.

Entry at the alternate entry IODTWT2 loads R9 with the TCB address and then checks for I/O WAIT as in the above. I/O Traps are not tested when entered at this label.

UTILITY ROUTINES

NAME:	EVMOD
ABSTRACT:	This routine modifies the event service routine (ESR) address to an address specified in register RE.
ENTRYS:	EVMOD
SOURCE LIBRARY ROUTINES:	EVN, DCB structures
EXTRNS:	NONE
REGISTERS USED:	R8, R9, RE, RF

ON ENTRY:	R8 = LINK
	RE = A (ESR)
	RF = A (leaf)

ON EXIT:	SAME AS ENTRY R9 will be destroyed

PRINCIPLES OF OPERATION:

Upon entry, R9 is loaded with the address of the DCB from the leaf. If R9 is zero, the leaf is not connected; therefore, return via R8. If R9 is nonzero, the address of ESR specified in RE has its most significant bit set (bit 0) to indicate modification by EVMOD. The ESR address is stored at DCB.ESR. Return to caller via R8.

NAME:	EVREL
ABSTRACT:	This routine disconnects a DCB from the tree beginning at the requested level via a call to COMDIS.
ENTRYS:	EVREL, NSEVREL
SOURCE LIBRARY ROUTINES:	EVN
EXTRNS:	COMDIS
REGISTERS USED:	R6-R9

ON ENTRY:	R8 = LINK
	RE = REQUEST LEVEL
	RF = leaf ADDRESS

ON EXIT:	ALL REGISTERS PRESERVED

PRINCIPLES OF OPERATION:

The two entries, EVREL and NSEVREL are equivalent. At entry, registers R4-RF are saved and RF is copied to R7, while RE is copied to R9, followed by the loading of R6 with the address of the DCB obtained from the leaf. A BAL to COMDIS is executed to do the disconnect and when returned, registers R4-RF are restored and return to caller via R8.

NAME:	GETIOB
ABSTRACT:	This routine allocates an IOB from the free list maintained by each TCB.
ENTRYS:	GETIOB
SOURCE LIBRARY ROUTINES:	TCB, IOB
EXTRNS:	NONE
REGISTERS USED:	R8, RA, RB

ON ENTRY:	ES STATE
	R8 = LINK
	R9 = A (TCB)

ON EXIT:	ES STATE
	RA = A (IOB)
	RB = DESTROYED

PRINCIPLES OF OPERATION:

Upon entry, RA is loaded from the top of the IOB list specified in the TCB. If the list is empty and safety checks are sysgened in, a system crash ensues with a Crash Code 220*. If list is not empty, the IOB list pointer is reestablished in the TCB with the address of the next IOB. In the current IOB, the forward pointer is zeroed, as is the FC, LU, and STAT entries, as well as the parameter block address, flags, and the IOB DONE executor address. The buffer start address and random record pointer are set to -1. A return to the caller through R8 is performed.

*This condition should not exist. If a previous I/O and proceed used the last IOB, the task should have been placed in connection wait at that time. See the SVC1EXIT subroutine definition.

NAME:	RELIOB
ABSTRACT:	This routine releases an IOB and returns it to the free list of the TCB.
ENTRYS:	RELIOB
SOURCE LIBRARY ROUTINES:	IOB, TCB
EXTRNS:	NONE
REGISTERS USED:	R8, RA, RB

ON ENTRY:	ES State
	R8 = LINK
	R9 = A (TCB)
	RA = A (IOB)

ON EXIT:	RB DESTROYED ALL OTHERS PRESERVED

PRINCIPLES OF OPERATION:

Upon entry, the IOB is tested for type validity (TCB as opposed to spare TCB), and if invalid (not TCB), a crash with Crash Code 221 is performed if safety checks are sysgened in. For a valid IOB, the top of list pointer from the TCB (TCB.IOBL) is loaded and stored in the current IOB forward pointer. The address of the current IOB is then stored in the top of list pointer and an exit to caller is performed.

NAME:	SQSMLV (Multilevel Device Driver Scheduler)
ABSTRACT:	SQSMLV dispatches the service routine for devices not on a single level. Coordination with upper modes is performed prior to dispatch.
ENTRYS:	SQS.MLV
SOURCE LIBRARY ROUTINES:	DCB, IOB, EVN
EXTRNS:	None
REGISTERS USED:	E8 - EB, ED, EF

ON ENTRY:	In Es State
	ED = A (DCB)
	EF = A (leaf)

ON EXIT:	

PRINCIPLES OF OPERATION:

Upon entry, the DCB dependent request flags are checked to ensure an EDMA connection or connection complete, and if so, the service routine is dispatched. If not, RB is loaded from the upper node pointer in the leaf. If RB is zero, dispatch the service routine. If not zero, the leaf flag is checked for the EDMA node and the level is checked for availability. If not available, go to SQSMSTOP and queue DCB to this node; if so, connect DCB to this node, bump level and retry for EDMA node. If the EDMA node is found, BAL to EDMAQCON to either connect to the node or queue the DCB for this node. When returning from EDMAQCON, if the node is gotten, dispatch the service routine. If queued, exit to SQS.EX.

EXIT ROUTINES

NAME:	DIRDONE
ABSTRACT:	DIRDONE is a common exit routine for standard device Driver Initialization Routines (DIRs)
ENTRYS:	DIRDONE
SOURCE LIBRARY ROUTINES:	DCB
EXTRNS:	SQ
REGISTERS USED:	ED - EE
ON ENTRY:	ES STATE
ON EXIT:	UD = A (SQ) ES STATE

PRINCIPLES OF OPERATION:

Upon entry, ED is loaded from SQS.DCB save area. Register EE is loaded from the DCB.ESR field of the DCB. If the most significant bit is set, this area has already been set by EVMOD and hence an exit to system queue service (SQS) is performed. If not set be EVMOD, the "ESR" (termination phase of the driver) address is fetched from the DCB and saved at the dispatch pointer in the DCB (DCB.ESR). Thereafter, ED is loaded with the address of the SQ and exit is made to SQS to check for entries on SQ.

NAME:	EVRTE
ABSTRACT:	This routine is the common exit path for standard device driver Event Service Routine (ESRs)
ENTRYS:	EVRTE
SOURCE LIBRARY ROUTINES:	NONE
EXTRNS:	SQ
REGISTERS USED:	UD
ON ENTRY:	ES STATE
ON EXIT:	ES STATE ED = A (SQ)

PRINCIPLES OF OPERATION:

Upon entry, ED is loaded with the address of the SQ and an unconditional branch is made to SQS.

NAME:	IODONE
ABSTRACT:	This routine is the normal or error termination from standard device driver ESR's. It returns status to users' SVC1 PBLK, sets requested task trap, removes I/O WAIT, and disconnects leaf.
ENTRYS:	IODONE, IODONE2 (Same as IODONE)
SOURCE LIBRARY ROUTINES:	DCB, IOB
EXTRNS:	NONE
REGISTERS USED:	R3 - R8, RD, RF

ON ENTRY:	ES STATE
	RD = A (DCB)
	RF = A (leaf)

ON EXIT:	ES STATE
	RD = A (SQ)
	RF = A (PARAM BLOCK)

PRINCIPLES OF OPERATION:

Upon entry, the ABORT I/O flag is reset and the address of a special IODONE executor is loaded. If address nonzero, branch to this routine. If zero, R3 is loaded with the request dependent flags, and with the leaf address in RF and load R5 with the IOB address, BAL to IODTWT to check TRAP and I/O WAIT conditions. When returned, the parameter block address is loaded to check for console dummy driver. If not, set status to parameter block and test for disconnect required (ITAM). If nonrequired, exit to SQS and check system queue. If ITAM device, load leaf address, set level counter, and BAL to COMDIS to disconnect from leaf and tree. Upon return, exit to SQS.

QUEUEING ROUTINES

NAME:	COMDIS
ABSTRACT:	This routine disconnects a DCB from the tree starting from the requested level.
ENTRYS:	COMDIS
SOURCE LIBRARY ROUTINES:	DCB, IOB, EVN, TCB
EXTRNS:	TMREMW
REGISTERS USED:	R4 - RF

ON ENTRY:	R4 = LINK
	R6 = A (DCB)
	R7 = A (leaf)
	R9 = Request level

ON EXIT:	R8 - RF may be destroyed

PRINCIPLES OF OPERATION:

Upon entry, the DCB and leaf addresses are copied to RD and RB, respectively, and the Connection Complete and Seek Check flags are reset in the DCB. Next the level is checked to see if it is the leaf level (=1) to start with first. If it is not a leaf, a loop is entered to go up levels until the appropriate level is found; it is tested for direct memory access (DMA) level and if so a BAL to EDMADIS is performed. If not DMA, a BAL to node DIS is executed. When returned, if more must be released, the EVN.CORD pointers are followed to next item until we have released all upper nodes to the EDMA node. At this point, return to caller is via R4.

If level to be released is a leaf, then the address of the IOB is fetched from the DCB and the IOB flags are tested for system buffered I/O. If the I/O count of the TCB is decremented by 1 and in either case the DCB is now disconnected from the leaf by setting EVN.DCB to zero, the queue is interrogated to see if it is empty, and if so the top and bottom pointers are set to zero. If not, the top pointer is adjusted and a BAL to LEAFCON is performed to connect top DCB in queue to leaf. Upon return, the leaf address is added to the SQ. The IOB flags are checked to see if it can be released now or not. If so, the IOB is released, the TCB is checked for any wait states, and we then proceed up the tree to next mode and process as in the above for either a node or EDMA node. Return to caller is via R4.

NAME:	COMFIFO
ABSTRACT:	The routine adds an IOB to the bottom of a leaf QUEUE or a leaf/node to the bottom of a node QUEUE.
ENTRYS:	COMFIFO
SOURCE LIBRARY ROUTINES:	NONE
EXTRNS:	NONE
REGISTERS USED:	R8 - RB

ON ENTRY:	R8 = LINK
	R9 = LEVEL NUMBER
	RA = A (IOB/leaf/node)
	RB = OWNER OF QUEUE

ON EXIT:	RC is destroyed CC is negative

PRINCIPLES OF OPERATION:

Upon entry, the level number is checked to see if it is greater than total number of nodes, and if so, a system crash with Crash Code 203 results. A crash also occurs if the level number is either negative or zero. If the leaf level is for a leaf, (level = 1) exit to LFIFO and queue the IOB. If level is not for leaf, exit to NFIFO to queue the leaf or node.

NAME:	COMQ
ABSTRACT:	This routine is a common queue routine which branches to appropriate queue handler depending upon the level number.
ENTRYS:	COMQ
SOURCE LIBRARY ROUTINES:	NONE
EXTRNS:	NONE
REGISTERS USED:	E8 - EB

ON ENTRY:	EA = A (IOB/leaf/node)
	EB = OWNER OF QUEUE
	E8 = LINK
	E9 = LEVEL NUMBER

ON EXIT:	EA & EB passed to QUEUE ROUTINES

PRINCIPLES OF OPERATION:

When entered, the level number in E9 is compared to see if it is greater than the total number of nodes, and if so, a system crash with Crash Code 203 results. Otherwise, the contents of E9 are aligned to a fullword boundary. If the level number is zero, a system crash of 203 also results. If nonzero, the queue routine address is loaded from the CQTABLE and unconditionally branched to for execution.

NAME:	DISKNODE
ABSTRACT:	This routine queues a leaf or node to an upper node.
ENTRYS:	NONE
SOURCE LIBRARY ROUTINES:	NONE
EXTRNS:	NONE
REGISTERS USED:	R8 - RF

ON ENTRY:	R8 = LINK
	R9 = LEVEL NUMBER
	RA = A (IOB/leaf/node)
	RB = owner of queue (leaf/node)
	RD = A (DCB)

ON EXIT:	Nothing modified

PRINCIPLES OF OPERATION:

Upon entry, the level is checked for the disk controller level (=2) and if so, a branch to NFIFO to do a first-in/first-out (FIFO) queue is executed. If not, BAL to SEEKCHK to decide if a seek is necessary. If so, branch to NLIFO to set last-in/first-out (LIFO) queuing. If seek not necessary, branch to NFIFO.

NAME:	DISKQ
ABSTRACT:	This routine handles the queuing of IOB's and nodes for disk I/O. Branching to appropriate queue routine is performed depending upon the level number of the request.
ENTRYS:	DISKQ
SOURCE LIBRARY ROUTINES:	IOB, DCB, EVN
EXTRNS:	NONE
REGISTERS USED:	R8 - RF

ON ENTRY:	R8 = LINK
	R9 = LEVEL NUMBER
	RA = A (IOB/leaf/node)
	RB = owner of QUEUE (leaf/node)
	RD = A (DCB)

ON EXIT:	RC RE, RF destroyed CC is negative

PRINCIPLES OF OPERATION:

Upon entry, the level number is compared against the maximum and, if found to be greater, a system crash with Crash Code 203 results. If the level is the EDMA level (=4), then a branch to NFIFO is executed to queue on FIFO basis. If it is an intermediate level, then go to DISKNODE to queue the leaf or node to upper node. If level is leaf (=1), the waiting queue is interrogated. If empty, then the contents of RA is queued to the bottom of this queue, and if this is for active TCB, then it is queued to the 'top,' as well as to 'wrap.' An exit is then performed through R8 after setting the 'L' flag of CC. If the top of queue equals the wrap, then the item is placed at the top, the IOB is inserted into queue, and exit. If it is the same, then it is placed on the secondary queue, and exit. If the top of the queue is not equal to wrap, then the random address from the IOB is loaded and compared to the current sector pointer. If we are beyond the current sector, then this request is wrapped. If it is less than the current sector, then put this request into the IOB request queue in proper order. If this is the current active TCB request, then put onto secondary queue and exit. If not current TCB, set it on the top of the queue, insert the IOB, and exit.

NAME:	EDMACON
ABSTRACT:	This routine either connects a node to EDMA or rejects the request.
ENTRYS:	NONE
SOURCE LIBRARY ROUTINES:	EVN, DCB, IOB
EXTRNS:	NONE
REGISTERS USED:	R9 - RF

ON ENTRY:	R9 = LINK
	RA = A (node)
	RB = A (EDMA node)
	RD = A (DCB)

ON EXIT:	RC, RE, RF are destroyed CC is clear if connection made CC is negative if no connection

PRINCIPLES OF OPERATION:

Upon entry, the EDMA node is checked to see if it is already connected, and if so, the CC is set to negative and return to caller via R9 is performed. Next the request is made for an 'Exclusive' connection and if so, the EDMA transfer counter (EVN; ECTR) is checked for activity. If zero, the 'Exclusive' request active flag is set and connection is made by incrementing the EDMA transfer counter by one, setting the connect flag in the DCB, clearing the CC, and returning via R9 to the caller.

If the EDMA 'Exclusive' transfer flag is reset, the EDMA transfer counter is compared with the EDMA 'Maximum' and if found equal, this request is rejected as above with CC set negative. If not equal, then normal connection is accomplished as above.

NAME:	EDMADIS
ABSTRACT:	This routine disconnects a DCB from an EDMA node, if connected, and connects as many queued nodes as possible.
ENTRYS:	NONE
SOURCE LIBRARY ROUTINES:	DCB, IOB, EVN
EXTRNS:	SQ
REGISTERS USED:	R8 - RB, RD RF

ON ENTRY:	R8 = LINK
	RB = A (EDMA node)
	RD = A (DCB)

ON EXIT:	R9, RA, RC, RE, RF may be destroyed CC is negative if DCB not connected CC is clear if DCB was connected

PRINCIPLES OF OPERATION:

Upon entry, the DCB request dependent flags are examined to determine if the DCB is connected to the EDMA node. If not, the CC is set to negative and a return to caller via R8 is performed. If connected, then the connect flag and exclusive active flag of the DCB and node, respectively, are cleared and the EDMA ACTIVE Counter of the node is decremented by one. If the node top of queue is zero (queue empty), then a normal exit is performed with CC cleared. If queue is not empty, then it is checked to see if it is the last item. If so, it is removed and 'top' and 'bottom' pointers are set to zero. A BAL to EDMACON is performed in attempt to queue this item. If item is queued, the leaf of new DCB is added to the SQ and a return to check for another queue item is made as above. If the connection was not made, the item is added to the EDMA queue, the forward pointer is set, CC is cleared, and return to caller via R8 is performed.

NAME:	EDMAQCON
ABSTRACT:	If EDMA is needed, this routine either queues or connects a node to the EDMA level.
ENTRYS:	EDMAQCON
SOURCE LIBRARY ROUTINES:	DCB, IOB
EXTRNS:	NONE
REGISTERS USED:	R8-R9, RA-RB, RD, RF

ON ENTRY:	R8 = LINK
	RA = A (LOWER node)
	RB = A (EDMA node)
	RD = A (DCB)

ON EXIT:	R9, RC, RE, RF may be destroyed CC clear if connected or EDMA not required CC negative if node is queued

PRINCIPLES OF OPERATION:

Upon entry, the DCB is examined to inspect the EDMA strategy routine address, and if found to be zero, a BAL to EDMACON is performed in attempt to connect the leaf. A successful connection results in return to caller via R8. Unsuccessful connection causes an unconditional branch to the queuing strategy routine pointed to by the DCB with R9 loaded with 4 in order to set the level. If the EDMA strategy routine address is nonzero, a BAL to this routine is performed to determine if EDMA is needed, and if so, an attempt is made to connect as above. If not needed, the Connection Complete Flag in the DCB is set, the CC is reset, and a return to caller via R8 is performed.

NAME:	LEAFCON
ABSTRACT:	This routine connects an IOB to a leaf. All IOB information is moved to the DCB.
ENTRYS:	LEAFCON
SOURCE LIBRARY ROUTINES:	IOB, DCB
EXTRNS:	NONE
REGISTERS USED:	E9, EC, EF

ON ENTRY:	R8 = LINK
	RA = A (IOB)
	RB = A (leaf)

ON EXIT:	RA = A (DCB.IOB)
	RB = A (leaf)
	RC = A (CONNECT DCB)
	R9 = A (CONNECT TCB)
	RD & RE Preserved others may be destroyed

PRINCIPLES OF OPERATION:

Upon entry, the DCB address is loaded from IOB and stored into the EVN (leaf). The IOB information is then moved to the DCB. These items are: request dependent flags, IODONE processor address, the address of the TCB, the addresses of the driver initialization routine, start and end, unrelocated parameter block, relocated parameter block, the FC, lu, and STAT fields, random number, and lu entry. Exit is via R8.

NAME:	LEAFQ
ABSTRACT:	This routine queues an IOB to a leaf.
ENTRYS:	LEAFQ
SOURCE LIBRARY ROUTINES:	EVN, DCB
EXTRNS:	NONE
REGISTERS USED:	R8 - RC

ON ENTRY:	R8 = LINK
	RA = A (IOB)
	RB = A (leaf)

ON EXIT:	RA, RB Preserved others may be destroyed

PRINCIPLES OF OPERATION:

Upon entry, R9 is set to contain level number 1, the DCB queue routine is loaded into RC, and if zero, a branch to COMQ is performed. If queue routine is defined, branch to this routine via RC.

NAME:	LFIFO
ABSTRACT:	This routine adds an IOB to the bottom of a leaf queue.
ENTRYS:	LFIFO
SOURCE LIBRARY ROUTINES:	IOB, EVN
EXTRNS:	NONE
REGISTERS USED:	R8 - RC

ON ENTRY:	R8 = A (RETURN)
	RA = A (IOB)
	RB = owner of leaf queue

ON EXIT:	RC is destroyed CC is negative

PRINCIPLES OF OPERATION:

Upon entry, the forward pointer of the IOB is cleared and the bottom of the leaf queue is loaded. If zero (queue empty), then the IOB is set to the top and bottom of the queue, the CC 'L' flag is set, and exit is via R8. If the bottom of queue is not zero, the IOB is set to be the forward pointer of current bottom and the IOB becomes the new bottom, thereby setting the 'L' flag of CC and exiting via R8.

NAME:	NFIFO
ABSTRACT:	This routine adds an item to the bottom of a node (leaf) queue.
ENTRYS:	NFIFO
SOURCE LIBRARY ROUTINES:	EVN
EXTRNS:	NONE
REGISTERS USED:	R8 - RC

ON ENTRY:	R8 = LINK
	RA = A (ITEM)
	RB = owner of queue

ON EXIT:	RC is destroyed CC is negative

PRINCIPLES OF OPERATION:

Upon entry, the forward pointer of the node (leaf) is cleared in the item to be queued. The node (leaf) bottom of the queue is loaded and if zero, a new entry is being made, in which case the item address is set at the queue top pointer as well as the bottom. If it is nonzero, then the item to be added becomes new bottom with its address set at the forward pointer in the previous bottom to maintain the list. The 'L' flag of CC is set and an exit via E8 is performed.

NAME:	NLIFO
ABSTRACT:	This routine places an item on the top of a node's queue.
ENTRYS:	NLIFO
SOURCE LIBRARY ROUTINES:	EVN
EXTRNS:	NONE
REGISTERS USED:	R8, RA - RB, RF

ON ENTRY:	R8 = LINK
	RA = A (ITEM)
	RB = owner of queue

ON EXIT:	RF is destroyed CC is negative

PRINCIPLES OF OPERATION:

Upon entry, the top pointer of the node's queue is loaded and if queue is empty, the item is added to bottom pointer of queue. If not empty, the item in RA is placed at the top of the queue. The address of the previous top is stored at the item's forward pointer. The 'L' flag of the CC is set and an exit is performed via EA to caller.

NAME:	NODECON
ABSTRACT:	This routine connects a lower node (leaf) to an upper node.
ENTRYS:	NONE
SOURCE LIBRARY ROUTINES:	EVN
EXTRNS:	NONE
REGISTERS USED:	R8, RA - RD, RF

ON ENTRY:	R8 = LINK
	RA = A (LOWER node or leaf)
	RB = A (UPPER node)
	RD = A (DCB)

ON EXIT:	RC is destroyed

PRINCIPLES OF OPERATION:

Upon entry, the DCB address is set into the upper node flagging it as being used, the CC is reset, and an exit via R8 is performed.

NAME:	NODEDIS
ABSTRACT:	This routine disconnects a DCB from a node and connects the top of the queue, if any.
ENTRYS:	NONE
SOURCE LIBRARY ROUTINES:	EVN, DCB
EXTRNS:	SQ
REGISTERS USED:	R8 - RD, RF

ON ENTRY:	R8 = LINK
	RB = A (node)
	RD = A (DCB)

ON EXIT:	RA, RC, RD may be destroyed CC is negative if node was not connected

PRINCIPLES OF OPERATION:

Upon entry, the contents of RD is compared with EVN.DCB to see if the node is connected. If not, the 'L' flag is set in the CC and an exit via R8 is executed. If connected the node DCB field is cleared and the queue top pointer is examined. If empty, return via R8; if not empty, check the queue bottom to see if it is the last item. If it is, zero bottom pointer; if not, fetch the next item and reset the top pointer to this item (or 0 if last item). Fetch the DCB address and leaf address from previous top of queue and add leaf address to top of the SQ list. Fall through to routine NODECON and exit.

NAME:	NODEQCON
ABSTRACT:	This routine either connects a node or leaf or will queue a node or leaf if it is unavailable.
ENTRYS:	NONE
SOURCE LIBRARY ROUTINES:	EVN, DCB
EXTRNS:	NONE
REGISERS USED:	R8 - RD

ON ENTRY:	R8 = LINK
	R9 = connect level
	RA = lower node (leaf)
	RB = upper node (leaf)
	RD = A (DCB)

ON EXIT:	RC destroyed if queued, R9 - RF may be destroyed

PRINCIPLES OF OPERATION:

At entry, RC is loaded with the DCB address from the node (leaf). If zero, branch to NODECON to connect this DCB to node (leaf). If non-zero, this node (leaf) is unavailable and request must be queued. The queue strategy is loaded from the DCB and if nonzero, exit to this routine. If zero, fall into routine NFIFO.

NAME:	PRTYQ
ABSTRACT:	This routine performs the priority queue handling by ordering the wait queue by priority.
ENTRYS:	PRTYQ
SOURCE LIBRARY ROUTINES:	EVN, IOB
EXTRNS:	NONE
REGISTERS USED:	E8 - ED

ON ENTRY:	EA = A (IOB)
	EB = A (leaf/node)
	E8 = LINK

ON EXIT:	

PRINCIPLES OF OPERATION:

Upon entry, the top of the node's (leaf's) wait queue is loaded. If empty, the IOB address goes on the top and bottom of the list, the IOB forward pointer is reset, the CC has the 'L' flag set and return is by way of E8. If nonempty, the priority of the IOB to be inserted is compared with the priority of the queued top. If 'new' is less than or equal to 'bottom,' append 'new' at the bottom. If in between, then the queue is searched by priority and the new IOB is inserted in the appropriate place. In all cases, exit is made via E8 after the 'L' flag is set in the CC (Condition Code).

NAME:	SEEKCHK
ABSTRACT:	This routine decides whether a seek is necessary, flags the DCB with the result, and notifies the caller.
ENTRYS:	SEEKCHK
SOURCE LIBRARY ROUTINES:	DCB, IOB
EXTRNS:	NONE
REGISTERS USED:	R9, RC - RF

ON ENTRY:	R9 = LINK
	RD = A (DCB)

ON EXIT:	R8 - RB, RD Preserved RC, RE, RF, destroyed CC is zero if seek necessary CC is negative if no seek necessary

PRINCIPLES OF OPERATION:

Upon entry, the DCB request flags are checked to see if a seek check had already been performed, and if not then one is executed. If it had, the IOB 'seek necessary' flag is checked to see if a seek must be performed. If not, the CC 'L' flag is set and exit is via R9. If yes, then set the CC to zero and exit via R9. If a seek check must be performed, the number of tracks per cylinder is loaded from the DCB and the number of sectors per cylinder is computed; the DCB.RAND pointer is loaded and diminished by DCB.CSEC. If the current sector is the same as random, then no seek is needed and the flags are set in the DCB to indicate such, CC 'L' flag is set, and exit is made via R9. If not current sector, a check is made to see if it is the same cylinder, and if so, the DCB flags are set, CC is set to zero, and the exit is via R9. If not the same cylinder, then a seek must be performed; therefore, set flags in DCB and return to caller with CC set to zero.

APPENDIX E

SKELETON DRIVER AND ASSOCIATED DCBXXX MACRO

INITSKEL

This module is the body of the skeleton driver. It includes the structure definitions (from the standard system libraries), the initialization, interrupt service and termination routines.

NOTE

> This driver performs all necessary functions (e.g., setup of the interrupt service table, call to TOCHON, etc.) explicitly for purposes of illustration. Standard subroutines are supplied in the file SUBS.MAC (INITSUBS in the standard DRIVER.LIB), which perform many of these functions.

```
         MLIBS 8,9,10
INITSKEL PROG  PERKIN-ELMER OS/32 SKELETON DRIVER



         TITLE STRUCTURE AND MNEMONIC DEFINITIONS:
* INCLUDE STANDARD SYSTEM STRUCTURES:
         $PDCB
         $CCB
         $REGS$





* DEFINE DEVICE-DEPENDENT DCB:
         STRUC
         DS     DCB.DVDP               DEVICE-INDEPENDENT PORTION
DCB.CCB  DS     2                      A(CCB)
DCB.XDCD DS     2                      EXT'D DEV CODE FROM SYSGEN INPUT
*                                      (SEE DCB MODULE)
DCB.XXXX DSF    1                      OTHER DEVICE-DEPENDENT FIELDS
*                                      DEFINED HERE.
         ENDS
```

```
* MNEMONIC DEFINITIONS:
TOUTSKEL EQU   5                    5 SECOND TIMEOUT (FOR EXAMPLE)


* EXTERNAL REFERENCES:
         EXTRN ISPTAB               INTERRUPT SERVICE TABLE
         EXTRN TOCHON               PUT DCB ON TIMER CHAIN
         EXTRN TOCHOFF              TAKE DCB OFF TIMER CHAIN
         EXTRN SQ                   SYSTEM QUEUE
         EXTRN DIRDONE              EXIT FROM DRIVER INITIALIZATION
         EXTRN IODONE2              ERROR EXIT FROM DVR INIT
         EXTRN IODONE               FINAL EXIT FROM DRIVER

         TITLE DRIVER INITIALIZATION ROUTINE
         IMPUR
         ENTRY INITSKEL,TERMSKEL

INITSKEL EQU   *

* ENTER HERE FROM SYSTEM QUEUE SERVICE.  INIT ROUTINE IS SCHEDULED
* BY SVC 1 SUPERVISOR.
*
* UPON ENTRY:
*
*        UD =  ADDRESS OF DCB
*        UF =  ADDRESS OF LEAF
*        DCB.FC(UD)   = SVC 1 FUNCTION CODE
*        DCB.SADR(UD) = SVC 1 BUFFER START ADDRESS (RELOCATED)
*        DCB.EADR(UD) = SVC 1 BUFFER END   ADDRESS (RELOCATED)
*        DCB.RAND(UD) = SVC 1 RANDOM ADDRESS FIELD
*        DCB.PBLK(UD) = SVC 1 PARAMETER BLOCK ADDRESS (RELOCATED)
*
* INSERT ANY DEVICE DEPENDENT INITIALIZATION HERE...
* INCLUDING BUFFER ALIGNMENT CHECKS, FUNCTION CODE CHECKS, ETC.
* ON ERROR, SET DCB.STAT AND DCB.DDPS TO DESIRED ERROR CODE AND
* EXIT VIA IODONE2.
*
* OTHERWISE, START UP THE DEVICE AS FOLLOWS:

* SET UP ISPTAB WITH CCB ADDRESS + 1:
         LHL   U4,DCB.DN(UD)        GET DEVICE NUMBER
         LHL   UC,DCB.CCB(UD)       GET CCB ADDRESS
         LA    UB,1(UC)             UB := A(CCB)+1
         STH    UB,ISPTAB(U4,U4)      SET ODD CCB ADDR IN ISPTAB

* SET UP CCB FOR NO-EXECUTE:
         LI     UO,CCWSTAT
         STH    UO,CCB.CCW(UC)

* SET UP 1ST ISR ADDRESS IN CCB:
         LA     UO,ISROSKEL
         STH    UO,CCB.SUBA(UC)

* NOW PUT DCB ON TIME-OUT CHAIN:
         BAL    U8,TOCHON

* SIMULATE INTERRUPT ON DEVICE TO GET INTO 1ST ISR.
* THIS IS ANALOGOUS TO A SUBROUTINE CALL ON THE 1ST ISR.
* ALL I/O IS NORMALLY DONE IN THE ISR.
         LHL    U8,DCB.ILVL(UD)       GET SYSGEN'D INTERRUPT LEVEL
         SINT   U8,O(U4)              "CALL" THE ISR

* NORMAL INIT ROUTINE EXIT IS TO DIRDONE:
         B      DIRDONE



         TITLE  INTERRUPT SERVICE ROUTINES
         PURE
* NOTE THAT THE ISRS ARE CODED AS "PURE".  LINK PROCESSES THIS PURE
* CODE TO ABSOLUTELY INSURE THAT THE ISRS ARE IN THE 1ST 64KB OF
* MEMORY.  THIS IS REQUIRED BY THE P-E 3200 SERIES ARCHITECTURE BECAUSE
* THE INTERRUPT SERVICE TABLE CONTAINS HALFWORD (16-BIT) POINTERS.
*
* NOTE ALSO THAT THE EXECUTIVE REGISTER MNEMONICS ARE USED TO
* REMIND US THAT WE ARE IN THE EXECUTIVE REGISTER SET ASSOCIATED
* WITH THIS INTERRUPT LEVEL.
*
* !!!! BY CONVENTION, ONLY REGISTERS EO THRU E7 MAY BE USED !!!!
ISROSKEL EQU    *

* ENTER HERE FROM SIMULATED INTERRUPT IN THE DIR.
*
* ON ENTRY:  (SET UP BY MICROCODE)
*
*        EO =  OLD PSW STATUS (IN DIR)
*         E1 =  OLD LOC COUNTER (RETURN TO DIR)
*         E2 =  INTERRUPTING DEVICE NUMBER
*         E3 =  DEVICE STATUS
*         E4 =  A(CCB) - GET A(DCB) FROM CCB.DCB(E4)
*
* ANY DEVICE INITIALIZATION IS NORMALLY PERFORMED HERE,
* INCLUDING STATUS CHECKS, OUTPUT COMMANDS, ETC.
* THEN DEVICE TIMEOUTS ARE ENABLED, CCB.SUBA IS SET UP TO INTERCEPT
* THE NEXT INTERRUPT AND WE RETURN TO THE DIR.
*
         L      E5,CCB.DCB(E4)      GET DCB ADDRESS
         LI     E6,TOUTSKEL         TIMEOUT CONSTANT (IN SECONDS)
         STH    E6,DCB.TOUT(E5)     START TIMER

         LA     E6,ISR1SKEL         NEXT ISR ADDRESS
         STH    E6,CCB.SUBA(E4)     SET IN CCB

         LPSWR E0                   RETURN TO DIR BY LOADING OLD PSW




ISR1SKEL EQU    *

* ENTER HERE WHEN THE DEVICE GENERATES AN INTERRUPT.
*
* ON ENTRY:  (SET UP BY MICROCODE)
*
*         E0 =  OLD PSW STATUS (IN INTERRUPTED CODE)
*         E1 =  OLD LOC COUNTER (IN INTERRUPTED CODE)
*         E2 =  INTERRUPTING DEVICE NUMBER
*         E3 =  DEVICE STATUS
*         E4 =  A(CCB) - GET A(DCB) FROM CCB.DCB(E4)
*
* BY CONVENTION THE 1ST THING WE MUST DO IS TEST THE TIMEOUT
* CONSTANT IN THE DCB.  IF IT IS ZERO, THIS INTERRUPT IS TOO LATE.
* THE OS HAS ALREADY SCHEDULED DEVICE TERMINATION.
*
         L      E5,CCB.DCB(E4)      GET DCB ADDRESS
         LH     E6,DCB.TOUT(E6)     TEST TIMEOUT VALUE
         BNP    ISRTOUT             TOO LATE, GET OUT !!!
```

```
* ANY DEVICE DEPENDENT CODE WOULD BE INSERTED HERE.
* SUCH AS STATUS CHECKS, DATA TRANSFER, OUTPUT COMMANDS.
*
* IF ANY ERRORS ARE ENCOUNTERED WHICH REQUIRE IMMEDIATE TERMINATION,
* LOAD E7 WITH A STATUS CODE (E.G., X'8400') AND EXIT VIA ISR.STAT
* BELOW WHICH SCHEDULES DRIVER TERMINATION.  OTHERWISE...
*
* THIS ISR MIGHT HANDLE MULTIPLE DEVICE INTERRUPTS.
* AT SOME POINT, WE WILL SET UP CCB.SUBA TO POINT TO THE FINAL
* ISR:

         LA    E6,ISRNSKEL         FINAL ISR ADDRESS
         STH   E6,CCB.SUBA(E4)     SET IN CCB
         LPSWR E0                  RESUME INTERRUPTED CODE




ISRNSKEL EQU   *

* ENTER HERE ON FINAL INTERRUPT FROM THE DEVICE.
*
* ON ENTRY:  (SET UP BY MICROCODE)
*
*        E0 =  OLD PSW STATUS (IN INTERRUPTED CODE)
*        E1 =  OLD LOC COUNTER (IN INTERRUPTED CODE)
*        E2 =  INTERRUPTING DEVICE NUMBER
*        E3 =  DEVICE STATUS
*        E4 =  A(CCB) - GET A(DCB) FROM CCB.DCB(E4)
*
* AGAIN, WE MUST 1ST CHECK THAT THE DRIVER HAS NOT BEEN TIMED OUT.
*
         L     E5,CCB.DCB(E4)      GET DCB ADDRESS
         LH    E6,DCB.TOUT(E5)     TEST TIMEOUT VALUE
         BNP   ISRTOUT             TOO LATE, JUST GET OUT !!

* DO ANY DEVICE CLEANUP OPERATIONS HERE....
* SET UP DCB.STAT AND DCB.DDPS TO REFLECT STATUS IF DESIRED, OR
* LEAVE THIS UP TO THE TERMINATION ROUTINE.
*
         LIS   E7,0                SET GOOD STATUS
ISR.STAT EQU   *                   ERROR EXIT PATH
```

```
         STH   E7,DCB.STAT(E5)     SET STATUS IN DCB

* NOW SCHEDULE DRIVER TERMINATION BY ADDING LEAF ADDRESS TO SYSTEM
* QUEUE.
* NOTE THAT DCB.TOUT IS SET TO -1 (=X'FFFF') TO PREVENT THE OS
* FROM TIMING IT OUT NOW THAT WE HAVE SCHEDULED TERMINATION.
*
         LCS   E7,1                GET A -1
         STH   E7,DCB.TOUT(E5)     ZAP TIMEOUT

         L     E7,DCB.LEAF(E5)     GET LEAF ADDRESS
         ATL   E7,SQ               SCHEDULE TERMINATION ESR
         LPSWR E0                  EXIT FINAL ISR




ISRTOUT  EQU   *

* COME HERE ON ANY INTERRUPT WHICH FINDS DCB.TOUT <= 0
* DO ANY NECESSARY DEVICE DEPENDENT CLEANUP (EG., DISARM)
* AND/OR JUST EXIT
*
         LPSWR E0                  EXIT ISR



         TITLE DRIVER TERMINATION ROUTINE
         IMPUR
*
* NOTE THAT THE TERMINATION ESR IS CODED AS IMPUR BECAUSE IT CAN
* OCCUR ANYWHERE IN MEMORY.  I.E., IT IS NOT RESTRICTED TO THE
* 1ST 64KB.

TERMSKEL EQU   *

* ENTER HERE FROM SYSTEM QUEUE SERVICE.  DRIVER TERMINATION IS
* SCHEDULED BY THE FINAL ISR OR, IN THE EVENT OF A DEVICE TIMEOUT,
* BY THE OS ITSELF.  WHICH CAN BE DETERMINED BY EXAMINING DCB.TOUT.
*
* UPON ENTRY:
*
```

```
*         UD =  ADDRESS OF DCB
*         UF =  ADDRESS OF LEAF
*
* BY CONVENTION, WE FIRST TEST FOR TIMEOUT AND SET STATUS
* ACCORDINGLY.
*
          LH    UO,DCB.TOUT(UD)      DID WE TIMEOUT ??
          BNZ   TERM.OK              NO, OK

* DEVICE TIMED OUT...SET APPROPRIATE STATUS:
          LHI   U7,X'8282'           TIMEOUT STATUS
          STH   U7,DCB.STAT(UD)
* CLEANUP AS NECESSARY (EG., DISARM THE DEVICE, ...)
          B     TERM.END


TERM.OK   EQU   *
* DID NOT TIME OUT.  CHECK FOR OTHER POSSIBLE ERRORS, DO ANY
* NECESSARY CLEANUP, CALCULATE LENGTH OF TRANSFER AND PUT IN
* DCB.LLXF IF APPROPRIATE...
*         :
*         :

TERM.END EQU    *
          BAL   U8,TOCHOFF           !!! REMOVE DCB FROM TIME OUT CHAIN !!!
          B     IODONE               FINAL EXIT FROM DRIVER
          END
```

ASSOCIATED DCBxxx MACRO

The following DCBxx macro definition is provided. Notice how this user has only used the necessary macro parameters and deleted the other parameters.

```
DEVICE CODE 246
         MACRO
         DCB246     %DCOD=,%DN=,%CLAS=,%ILVL=,%NAME=,%SHCCB=,%COUNT=
*
*  DEFINE GLOBAL COUNTER FOR DEVICE CODES
*
         GBLC  %IDVAL
         BGBLA %ID246
*
         LCLC  %OFFS
         LCLA  %CLASN
         AIF   (T'%CLAS EQ 'U')&CLSNTD
%CLASN   SETA  %CLAS*12             IOCLASS*12
&CLSNTD  ANOP
*
         USERINIT                   INITIALIZE STRUC COPY FLAGS
*
         CONVNUM VAL=%ID246         SET %IDVAL WITH STRING EQUIVALENT
*
*       CREATE DCB
*
DCB_%NAME   PROG         GENERATE PROGRAM LABEL FOR LINK MAP
         DCBI DCOD=246,SIZE=DCB.DVDP+4,INIT=INITSKEL,TERM=TERMSKEL,     1
              ATRB=7F00,IOH=COMIOH,ID=%IDVAL,SHCC=1
*
*       CREATE CCB
*
         CCBI  DCOD=246,ID=%IDVAL,SUBA=III
*
CCB%NAME EQU   CCB%DCOD%IDVAL
%ID246  SETA  %ID246+1
*
*       DEFINE SYSTEM DEPENDENT FIELDS OF DCB FIRST
*
%OFFS    SETC  '%DCOD':'%IDVAL'    ESTABLISH PROPER OFFSET
         ORG   DCB%OFFS+DCB.DN     DEVICE ADDRESS
         DC     H'%DN'
         ORG    DCB%OFFS+DCB.LEAF   LEAF POINTER
         AIF    (T'%SHCCB' EQ 'U')&NSLEAF
         DAC    LF%SHCCB            USE SHARED DEVICE LEAF
         EXTRN LF%SHCCB
         AGO    &NRMLFX
&NSLEAF  ANOP
         DAC    LF%OFFS             GENERATE STANDARD LEAF NAME
         EXTRN LF%OFFS
&NRMLFX  ANOP
&NOLEAF  ANOP
*
*
*  DEFINE ALTERNATE NAME FOR DMT MACRO
DCB_%NAME EQU  DCB%OFFS
         ENTRY DCB_%NAME
         ORG    DCB%OFFS+DCB.DMT
         DC     DMT_%NAME           A(DMT ENTRY)
         EXTRN DMT_%NAME
*
         AIF    (T'%CLAS EQ 'U')&NOCLAS
         ORG    DCB%OFFS+DCB.CLAS   IO CLASS
         DC     H'%CLASN'           IOCLASS*12
&NOCLAS  ANOP
*
         AIF    (T'%ILVL EQ 'U')&NOILVL
         ORG    DCB%OFFS+DCB.ILVL   ILEVEL
         DC     H'%ILVL'
&NOILVL  ANOP
*
*       CONTINUE WITH OTHER DCB OPTIONS
*
         AIF    (T'%COUNT EQ 'U')&NCOUNT
         ORG    DCB%OFFS+DCB.DVDP
         DC     %COUNT
&NCOUNT  ANOP
*
         USEREND                    MUST END THIS SOURCE MODULE
         MEND
```

PERKIN-ELMER

PUBLICATION COMMENT FORM

We try to make our publications easy to understand and free of errors. Our users are an integral source of information for improving future revisions. Please use this postage paid form to send us comments, corrections, suggestions, *etc*.

1. Publication number ______________________________

2. Title of publication ______________________________

3. Describe, providing page numbers, any technical errors you found. Attach additional sheet if neccessary.

   ____

   ____

   ____

4. Was the publication easy to understand? If no, why not?

   ____

5. Were illustrations adequate? ______________________________

   ____

   ____

6. What additions or deletions would you suggest? ______________________________

   ____

   ____

7. Other comments: ______________________________

   ____

   ____

From ______________________________ Date ______________________________

Position/Title ______________________________

Company ______________________________

Address ______________________________

____

____

STAPLE STAPLE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 22 OCEANPORT, N.J.

POSTAGE WILL BE PAID BY ADDRESSEE

PERKIN-ELMER

2 Crescent Place
Oceanport, N.J. 07757

ATTN:
TECHNICAL SYSTEMS PUBLICATIONS DEPT., HANCE AVE.

FOLD FOLD

STAPLE STAPLE

9306