DEC-11-NIZA-D

PDP-11

DEVICE DRIVER PACKAGE

MARCH 1971

THIS DOCUMENT IS FOR INFORMATION PURPOSES
ONLY AND IS SUBJECT TO CHANGE WITHOUT NOTICE.

pdp11
PROGRAM
LIBRARY

CONTENTS

CHAPTER 1

INTRODUCTION

Subroutines to handle I/O transfers between a PDP-11 and each of its peripheral devices are developed as required for use within the Disk Operating System (DOS). These subroutines are made available within an I/O Utilities Package for the benefit of PDP-11 users who have configurations unable to support DOS or who wish to run programs outside DOS control.

All the subroutines associated with one peripheral device together form an entity which is known as a Driver. The purpose of this document is to provide a general description of a driver and to show how it may be used in a stand-alone environment. The unique properties of each driver are discussed in separate documents issued as supplements to this one. The I/O Utilities Package for any system is determined by the peripherals of that system. Thus the full documentation for a particular package consists of this document and the applicable supplements.

Within this document, Chapter 2 consists of an outline of the established driver structure and its interface to the program using it. Chapter 3 then illustrates how a stand-alone program can match this interface in order to make immediate use of each driver as supplied within the package. For the benefit of those users who require a more detailed description of the driver format, perhaps so that they can write their own drivers for other unsupported devices in a similar fashion, the standard specification for DOS driver has been attached as Appendix A. It is assumed that the reader is familiar with the basic hardware concepts of the PDP-11 as described in the PDP-11 Handbook and with the Paper Tape Software as described in the Programming Handbook (DEC-11-GGPB-D).

%%%%%%

CHAPTER 2

DRIVER FORMAT

2.1 Structure

The basic principle of all drivers under the DOS Monitor is that they must present a common interface to the routines using them in order to provide for device-independent operation. The subroutines are structured to meet this end. Moreover, the driver may be loaded anywhere in memory under Monitor control. Its code must always, therefore, be position-independent.

The detailed description of a driver is found in Appendix A. This chapter is concerned with driver interfaces.

2.1.1 Driver Interface Table

The first section of each driver consists of a table which contains, in a standard format, information on the nature and capabilities of the device it represents and entry pointers to each of its subroutines. The calling program may then use this table as required, regardless of the device being called.

2.1.2 Setup Routines

Each driver is expected to handle its device under the PDP-11 interrupt system. When called by a program, therefore, a driver subroutine merely initiates the action required by setting the device hardware registers appropriately. It then returns to the calling program by a standard subroutine exit.

The main setup routine prepares for a data transfer to or from the device, using parameters supplied by the calling program. Normally, blocks of data will be moved at each transfer. The driver will only return control to the program when the whole block has been actioned or when it is unable to continue because there is no more data available.

The driver may also contain subroutines by which the calling program may request start-up or shut-down action, such as leader or trailer code at a paper-tape punch, or some special function provided by the device hardware (or a software simulation of that for some similar device), e.g., "rewind" of a magnetic tape (or DECtape).

2.1.3 Interrupt Servicing

The nature of the driver routine to service device interrupts is particularly dependent upon the extent of the hardware provisions of the device for controlling transfers. In general, the driver determines the cause of the interrupt and checks whether the last action was performed correctly or was prevented by some error condition. If more device action is needed to satisfy the program, the driver again initiates that action and takes a normal interrupt exit. If the program request has been fully met, control is returned to the program at an address supplied at the time of the call.

2.1.4 Error Handling

Device errors may be handled in two ways. There are some errors for which recovery can be programmed; the driver will, if appropriate, attempt this itself (as in the case of parity or timing failure on a bulk-storage device) or will recall the program with the error condition flagged (as at the end of a physical paper tape). Other errors will normally require action externally, perhaps by an operator. For the latter, the driver calls a common error handler based on location 34 (IOT call) with supporting information on the processor stack.

2.2 Interface to the Driver

2.2.1 Control Interface

The principal link between a calling program and any driver subroutine is the first word of the driver table. In order to provide the control parameters for a device operation, the calling program prepares a list in a standardized form and places a pointer to the list in the driver link. The called driver then uses the pointer to access the parameters. If the driver need then return status information, it may again place this in the list area via the link-word.

The first word of the driver also may act as an indicator in that while it remains Ø, the driver is not already busy upon some task, whereas when the word contains a list-pointer, the driver is assumed to be busy. Since most drivers can only support one job at a time, the link-word state can be significant.

2.2.2 Interrupt Interface

Although the driver will always expect to use the interrupt system, it does not itself ensure that its interrupt vector in the memory area below 400 has been set up correctly; the Monitor under DOS takes care of this. However, the Driver Table contains the necessary information to allow the vector to be set correctly.

CHAPTER 3

STAND-ALONE USAGE

Because each driver is designed for operation within the device-independent framework of DOS Monitor, it may be similarly used in other applications. Possible methods will be discussed later. However, since the easiest way to use the driver is to assemble it with the program requiring it, this will be described first.

3.1 Driver Assembled with Program

3.1.1 Setting Interrupt Vector

As noted in Section 2.2.2, the calling program must first correctly set the device transfer vector within memory locations 0-377. The address of the driver's interrupt entry point can be identified on the source listing by the symbolic name which appears as the content of the Driver Table Byte, DRIVER+5. The priority level at which the driver expects to process the interrupt is at byte DRIVER+6. For a program which can use position-dependent code, the setup sequence may be:

```
MOV    #DVRINT, VECTOR        ;SET INT. ADDRESS
MOVB   DRIVER+6, VECTOR+2     ;SET PRIORITY
CLRB   VECTOR+3               ;CLEAR UPPER STATUS BYTE
```

(where the Driver Table shows at DRIVER +5: .BYTE DVRINT-DRIVER).

If the program must be position-independent, it may take advantage of the fact that the Interrupt Entry address is actually stored as an offset from the start of the driver, as illustrated above. In this case, a sample sequence might be:

```
MOV    PC,R1                 ;GET DRIVER START
ADD    #DRIVER-.,R1
MOV    #VECTOR,R2            ;...& VECTOR ADDRESSED
CLR    @R2                   ;SET INT. ADDRESS
MOVB   5(R1),@R2             ;...AS START ADDRESS+OFFSET
ADD    R1,(R2)+
CLR    @R2                   ;SET PRIORITY
MOVB   6(R1),@R2
```

3.1.2 Parameter Table for Driver Call

For any call to the driver, the program must provide the list of control arguments mentioned in Section 2.2.1. This list must adhere in general to the following format:[1]

```
[SPECIAL FUNCTION CODE]²
[BLOCK NO.]³
STARTING MEMORY ADDRESS FOR TRANSFER
NO. OF WORDS to be transferred (2's complement)
STATUS CONTROL showing in Bits:
  Ø-2:  Function (octally 2=WRITE, 4=READ)⁴
  8-1Ø: Unit (if Device can consist of several, e.g., DECtape)
  11:   Direction for DECtape travel (Ø = Forward)
ADDRESS for RETURN ON COMPLETION
[RESERVED FOR DRIVER USE]⁵
```

The list itself may be assembled into the required format if its content will not vary. The driver may return information in the area as described in a later paragraph; however, this will not corrupt the program data and it is removed by the driver before it begins its next operation.

On the other hand, most programs will probably wish to use the same area for the lists for several tasks or even between different drivers. In this case, the program must contain the necessary routine to set up the list for each task before making the driver call, perhaps as illustrated in the next paragraph. It must be noted, however, that the driver may wish to refer to the list again when it is recalled by an interrupt or to return information to the calling program. Therefore, the list must not be changed until any driver has completed a function requested; for concurrent operations, different list areas must be provided.

3.1.3 Calling the Driver

To enable the driver to access the parameter list, the program must set the first word of the driver to an address six bytes less than that

---

[1] In some cases, it may be further extended as discussed in later paragraphs.

[2] Required only if Driver is being called for Special Function.

[3] Required only if the Device is bulk storage (e.g., Disk or DECtape).

[4] Most devices transfer words regardless of their content, i.e. ASCII or Binary. Some devices, e.g., Card Reader, may be handled differently for the two modes; for these, Bit Ø must also be set to indicate ASCII=Ø, Binary=1. (In these cases, the driver always produces or accepts ASCII even though the device itself uses some other code.)

[5] This word may be omitted if the device is bulk storage (see below).

of the word containing MEMORY START ADDRESS. It may then call the driver subroutine required directly by a normal JSR PC,xxxx call.

As an example, the following position-independent code might appear in a program which wishes to read Blocks #1ØØ-1Ø3 backward from DECtape Unit into a buffer starting at address BUFFER:

```
          MOV   PC,RØ               ;GET TABLE ADDRESS
          ADD   #TABLE+12-.,RØ
          MOV   PC,@RØ              ;GET & STORE...
          ADD   #RETURN-.,@RØ       ;...RETURN ADDRESS
          MOV   #54Ø4,-(RØ)         ;SET READ REV. UNIT 3
          MOV   #-1Ø24.,-(RØ)       ;4 BLOCKS REQUIRED
          MOV   PC,-(RØ)            ;GET & STORE
          ADD   #BUFFER-.,@RØ       ;...BUFFER ADDRESS
          MOV   #1Ø3,-RØ)           ;START BLOCK
          CMP   -(RØ),-(RØ)         ;SUBTRACT 4 FROM POINTER
          MOV   RØ,DT               ;SET DRIVER LINK
          JSR   PC,DT.TFR           ;GOTO TRANSFER ROUTINE
   WAIT:  .                         ;RETURNS HERE WHEN
 RETURN:  :                         ;...TRANSFER UNDERWAY
          :                         ;RETURNS HERE WHEN
          :                         ;... TRANSFER COMPLETE
  TABLE:  .WORD Ø                   ;LIST AREA SET
          .WORD Ø                   ;...BY ABOVE SEQUENCE
          .WORD Ø
          .WORD Ø
          .WORD Ø
```

3.1.4 User Registers

During its setup operations for the function requested, the driver assumes that Processor Registers Ø-5 are freely available for its purpose. If their contents are of value, the program must save them before the driver is called.

While servicing intermediate interrupts, the driver may need to save or restore these registers. It expects to have available two subroutines for the purpose (provided by the Monitor under DOS) It accesses them via addresses in memory locations 44 (SAVE) and 46 (RESTORE) using the sequence:

```
        MOV   @#44,-(SP)          ;OR 'MOV   @#46,-(SP)
        JSR   R5,@(SP)+
```

The program must, therefore, contain these subroutines. They might, for example, be as follows:

```
SAVE:       MOV   R4,-(SP)      ;SAVE RØ-4
            MOV   R3,-(SP)      ;...R5 SAVED BY CALL
            MOV   R2,-(SP)
            MOV   R1,-(SP)
            MOV   RØ,-(SP)
            MOV   R5,PC         ;EXIT TO CALLER

RESTOR:     INC   (SP)+         ;FORGET CALL R5
            MOV   (SP)+,RØ      ;RESTORE RØ-4
            MOV   (SP)+,R1
            MOV   (SP)+,R2
            MOV   (SP)+,R3
            MOV   (SP)+,R4
            RTS   R5            ;R5 RESET ON EXIT
```

It must also ensure that their start addresses are set into the correct locations.

At its final interrupt, the driver always saves the contents of Registers Ø-5 before returning control to the calling program completion return.

3.1.5 Returns from Driver

As shown in the example in section 3.1.3, the driver returns control to the calling program immediately after the JSR as soon as it has set the device in motion. The program may then wait or carry out some alternative operations until the driver signals completion by returning at the address supplied, i.e., RETURN above. Prior to this, the program should not attempt to access the data being read in, or to refill a buffer being written out.

The program routine beginning at address RETURN will vary according to the device in use. In general, the driver has given control to the routine for one of two reasons, namely, the function has been satisfactorily performed, or it cannot be carried out due to some hardware failure with which the driver is unable to cope, though the program may. If the latter, the driver uses the STATUS word in the program list to show the cause:

- Bit 15 = 1 — indicates that a device parity or timing failure has occurred and the driver has not been able to overcome this, perhaps after several attempts.
- Bit 14 = 1 — shows that the end of the data available has been reached.

The driver places in RØ the content of its first word as a pointer to the list concerned.

In addition, the driver may have transferred only some of the data required. In this case, it will show, in the RESERVED word of the program list, a negative count of the words not transferred in addition to setting Bit 14 of the STATUS. As mentioned in the note in Section 3.1.2, this applies only to non-bulk storage devices. The drivers for DECtape or Disks[1] always endeavor to complete the full transfer, even beyond a parity failure, or they take more drastic action (see Section 3.1.6).

It is thus the responsibility of the program RETURN routine to check the information supplied by the driver in order to verify that the transfer was satisfactory and to handle the error situations accordingly.

In addition, the routine must contain a sequence to take care of the Processor Stack, Registers, etc. As noted earlier, the driver takes the completion return address after an interrupt and has saved Registers 0-5 on the stack above the Interrupt Return Address and Status. The program routine should, therefore, contain some sequence to restore the processor to its state prior to such interrupt, e.g., using the same Restore subroutine illustrated earlier:

```
MOV    @#46,-(SP)          ;CALL REGISTER RESTORE
JSR    R5,@(SP)+
 .
 .
 .
RTI                        ;RETURN TO INTERRUPTED PROG.
```

3.1.6 Irrecoverable Errors

All hardware errors other than those noted in the previous paragraph are more serious in that they cannot normally be overcome by the program or the driver on its behalf. Some of these could be due to an operator fault, such as an omission to turn a paper tape reader on or to set the correct unit number on a DECtape transport. Once the operator has rectified the problem, the program could continue. Other errors, however, will require hardware repair or even software repair, e.g., if the program asks for Block 2000 on a device having a maximum of 1000. In general, all these errors will result in the driver placing identifying information on the processor stack and calling IOT to produce a trap through location 34.

[1] This includes RF11 Disk: although this is basically word-oriented, it is assumed to be subdivided into 64-word blocks.

Under DOS, the Monitor provides a routine which prints a teleprinter message when this occurs. In a stand-alone environment, the program using the driver must itself contain the routine to handle the trap (unless the user wishes to modify the driver error exits before assembly). The handler format will depend upon the program. Should it wish to take advantage of the information supplied by the driver, the format is as follows:

(SP):	Return Address	Stored by IOT Call
2(SP):	Return Status	
4(SP):	Error No. Code	generally unique to driver
5(SP):	Error Type Code:	1 = Recoverable after Operator Action 3 = No recovery
6(SP):	Additional Information	such as content of Driver, Control Register, Driver Identity, etc.

As a rule, the driver will expect a return following the IOT call in the case of errors in Type 1 but will contain no provision following a return from Type 3.

3.1.7 General Comment

The source language of each driver has been written for use with the DOS version of the Assembler which requires certain statements which will not be accepted by the Paper Tape Software PAL-11A, in particular: .TITLE & .GLOBL. These should be edited out before the source is used. Similarly, an entry in the driver table gives the device name as .RAD5Ø 'DT' to obtain a specially packed format used internally by DOS. If the user still wishes to keep the name, for instance for identification purposes as discussed in section 3.3, .RAD5Ø might easily be changed to .ASCII without detrimental effect, or it can be replaced with .WORD Ø .

3.2 Drivers Assembled Separately

Rather than assemble the driver with every program requiring its availability, the user may wish to hold it in binary form and attach it to the program only when loaded. This is readily possible; the only requirement is that the start address of the driver should be known or can be determined by the program.

The example in section 3.1.2 showed that the Interrupt Servicing routine can be accessed through an offset stored in the Driver Table. The same technique can be used to call the setup subroutines, as these also have corresponding offsets in the Table, as follows:

```
DRIVER+7      Open[1]
      +10     Transfer
      +11     Close[1]
      +12     Special Functions[1]
```

The problem, of course, is the start address. There is always the obvious solution, that of assembling the driver at a fixed location so that each program using it can immediately reference the location chosen. This, however, ceases to be convenient when the program itself has to avoid the area given to the driver. A more general method is to relocate the driver as dictated by the program using it, thus taking advantage of the position-independent nature of the driver. The Absolute Loader, described in the Paper Tape Software Handbook (DEC-11-GGPB-D), Chapter 6, provides the capability of continuing a load from the point at which it ended. Using this facility to enter the driver immediately after the program, the program itself might contain the following code to call the subroutine to perform the transfer illustrated in section 3.1.3:

```
        MOV   PC,R1              ;GET DRIVER START ADDRESS
        ADD   #PRGEND-.,R1
        MOV   PC,RØ              ;GET TABLE ADDRESS
        ADD   #TABLE+12-.,RØ     ;& SET UP AS SHOWN
         .                       ;...IN SECTION 3.1.3
         .
         .
        CMP   -(RØ),-(RØ)        ;FINAL POINTER ADJUSTMENT
        MOV   RØ,@R1             ;STORE IN DRIVER LINK
        CLR   -(SP)              ;GET BYTE SHOWING...
        MOVB  1Ø(R1),@SP         ;...TRANSFER OFFSET
        ADD   (SP)+,R1           ;COMPUTE ADDRESS
        JSR   PC,@R1             ;GO TO DRIVER
         .
         .
         .
PRGEND:
       .END
```

This technique may be extended to cover situations in which several drivers are used by the same program, provided that it takes account of the size of each driver (this being already known because of prior assembly) and that the drivers themselves are always loaded in the same order.

For example, to access the second driver, the above sequence would be modified to:

[1]If the routine is not provided, these are Ø.

```
          MOV   PC,R1             ;GET DRIVER 1 ADDRESS
          ADD   #PRGEND-.,R1
          ADD   #DVR1SZ,R1        ;STEP TO DRIVER 2
           .
           .
           .
DVR1SZ=
PRGEND:
         .END
```

An alternative method may be to use the Relocatable Assembler PAL-11S in association with the Linker program LINK-11S, both of which are available through the DECUS Library. The start address of each driver is identified as a global. Any calling program need, therefore, merely include a corresponding .GLOBL statement, e.g., .GLOBL DT.

3.3 Device-independent Usage

As mentioned earlier, the drivers are designed for use in a device-independent environment, i.e., one in which a calling program need not know in advance which driver has been associated with a table for a particular execution run. One application of this type might be to allow line-printer output to be diverted to some other output medium because the line-printer itself is currently not available. Another might be to provide a general program to analyze data samples although these on one occasion might come directly from an Analog to Digital converter and on another be stored on a DECtape, because the sampling rate was too high to allow immediate evaluation.

As a rule, programs of this type should be written to cater for all the facilities that any one device might offer, but not necessarily all of them. For instance, the program should ask for start-up procedures because it may sometime use a paper tape punch which provides them, even though it may normally use DECtape which does not. As noted in section 2.1.1, the driver table contains an indication of its capabilities to cater for this situation. The program can thus examine the appropriate item before calling the driver to perform some action. As an example, the code to request start-up procedures might be (assuming RØ already set to List Address):

```
          MOV   #DVRADD,R1        ;GET DRIVER ADDRESS
          TSTB   2(R1)            ;BIT 7 SHOWS...
          BPL   NOOPEN            ;...OPEN ROUTINE PRESENT
          MOV   RØ,@R1            ;STORE TABLE ADDRESS
          CLRB  -(SP)             ;BUILD ADDRESS
          MOVB  7(R1),@SP         ;...OF THIS ROUTINE
          ADD   (SP)+,R1
```

```
        JSR     PC,CR1                  ;...& GO TO IT
                                        ;FOLLOWED POSSIBLY BY
                                        ;WAIT AND COMPLETION
                                        ;PROCESSING
NOOPEN:                                 ;RETURN TO COMMON OPERATION
```

Similarly, the indicators show whether the device is capable of performing input or output or both, whether it can handle ASCII data or Binary data, whether it is a bulk storage device capable of supporting a directory structure or is a terminal-type device requiring special treatment and so on. Other table entries show the device name as identification and how many words it might normally expect to transfer at a time (in 16-word units). All of the information may readily be examined by the calling program, thus enabling the use perhaps of a common call sequence for any I/O operation, as for example:

```
        MOV     #DVRADR, R5             ;SET DRIVER START
        JSR     R5, IOSUB               ;CALL SET UP SUB
        BR      WAIT                    ;SKIP TABLE FOLLOWING ON RETURN
        .WORD   1Ø                      ;TRANSFER REQUIRED
        .WORD   1Ø3                     ;BLOCK NO.
        .WORD   BUFFER                  ;BUFFER ADDRESS
        .WORD   -256.                   ;WORD COUNT
        .WORD   4Ø4                     ;READ FROM UNIT 1
        .WORD   RETURN                  ;EXIT ON COMPLETION
        .WORD   Ø                       ;RESERVED
WAIT:                                   ;CONTINUE HERE...
        .                               ;WHILE TRANSFER IN PROGRESS
        .
        .
        .

IOSUB:  MOV     @SP,RØ                  ;PICK UP DRIVER ADDR
        MOV     R5,R1                   ;SET POINTER TO LIST
        TST     (R1)+                   ;BUMP TO COLLECT CONTENT
        .                               ;ROUTINE CHECKS ON DEVICE..
        .                               ;..CAPABILITY USING R1
        .                               ;..TO ACCESS LIST &
        .                               ;..RØ THE DRIVER TABLE
        .                               ;IF O.K...
        MOV     @R1,R1                  ;GET ROUTINE OFFSET
        ADD     RØ,R1
        CLR     -(SP)                   ;USE IT TO BUILD
        MOVB    @R1,@SP                 ;...ENTRY POINT
        ADD     RØ,@SP
        JSR     PC,@(SP)+               ;CALL DRIVER
        RTS     R5                      ;EXIT TO CALLER
```

The calling program, or a subroutine of the type just illustrated, may also wish to take advantage of a further feature mentioned earlier: the fact that when a driver is already occupied its first word must be non-zero. The driver itself does not clear this word except in special cases shown in the description for the driver concerned. If the program itself always ensures that it is set to zero between driver tasks, this word forms a suitable Driver-busy flag. Under DOS, in fact, the program parameter list is extended to allow additional words to provide linkage between lists as a queue of which the list indicated in the driver first word is the first link.

The preceding paragraphs are intended merely to indicate possible ways of incorporating the drivers available into the type of environment for which they were designed. The user will probably find others. However, he should read carefully the more detailed description of the driver structure in Appendix A and the individual driver specifications before determining the final form of his program.

In particular, one general word of warning is appropriate here. Although most drivers normally set up an operation and then wait for an interrupt to produce a completion state, there are some cases in which the driver can finish its required task without an interrupt, e.g., "opening" a paper-tape reader involves only a check on its status. Moreover, where "Special Functions" are concerned, the driver routine may determine from the code indicated that the function is not applicable in its case and will, therefore, have nothing to do. In those cases, the driver clears the intermediate return address from the processor stack and takes the completion return immediately. Special problems may arise, however, if the driver concerned may be covering several tasks, any of which may cause a queue for the driver's services under DOS. To overcome these problems, the driver expects to be able to refer to flags outside the scope of the list described so far. This may mean that a program using such a driver may also need to extend the list range to cover this possibility. Extreme care will then be needed.

APPENDIX A

I-O DRIVERS WITHIN THE DISK OPERATING SYSTEM

The principal function of an I/O driver is to satisfy the requirement of a Monitor processing routine for the transfer of a block of data in a standard format to or from the device it represents. This will involve both setting up the device hardware registers to cause the transfer and its control under the interrupt scheme of PDP-11, making due allowance for peculiar device characteristics (e.g., conversion to or from ASCII if some special code is used).

It may also include routines for handling device start-up or shut-down such as punching leader or trailer, and for making available to the user certain special features of the device, such as rewind of magtape.

A.1 Driver Structure

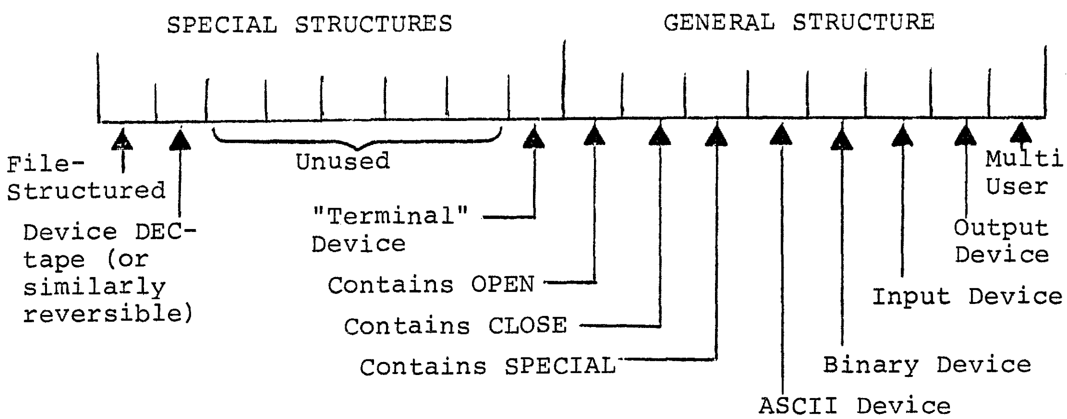
In order to provide a common interface to the monitor, all drivers must begin with a table of identifying information as follows:

DVR:

BUSY FLAG (initially Ø)	
FACILITY INDICATOR (expanded below)	
Offset to Interrupt Routine*	Standard Buffer Size in 16-word Units.
Offset to OPEN Routine *	Priority for Interrupt Service
Offset to CLOSE Routine *	Offset to Transfer Routine *
Space	Offset to Special Functions*
DEV	NAME (Packed Radix-5Ø)

Offsets marked * will enable calling routine to indicate routine required. They will be considered as an unsigned value to be added to the start address of the driver. This may mean that with a 256 maximum, the instruction referenced by the offset will be JMP or BR (routine).

Bits in the Facility Indicator Word define the device for monitor reference:



The table should be extended as follows if the device is file-structured:

BLOCK USED AS MASTER FILE DIRECTORY	
POINTER TO BIT-MAP IN MEMORY	Unit Ø
	Similar Bit-Map Pointers for Multi-unit Devices

The driver routines to set up the transfer and control it under interrupt, and possibly for OPEN, CLOSE, and SPECIAL, follow the table. Their detailed operation will be described later.

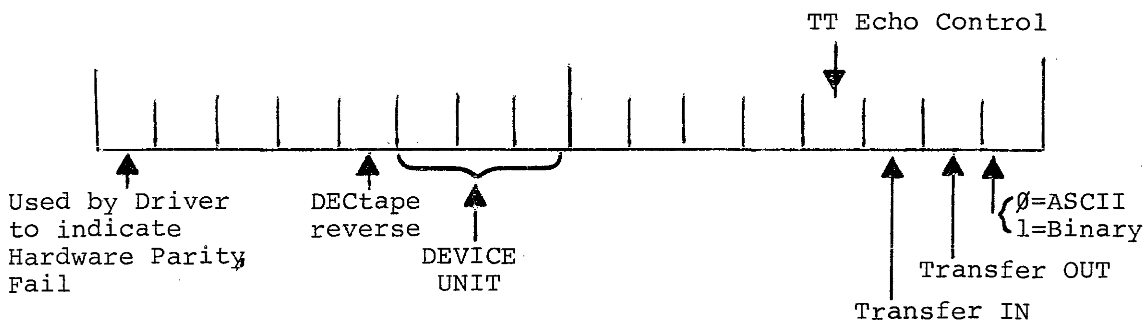
A.2 Monitor Calling

When a Monitor I/O processing routine needs to call the driver, it first sets up the parameters for the driver operation in relevant words of the appropriate DDB[1], as follows:

XYZ:	-	(User Call Address)
	SPECIAL FUNCTION CODE	(User Line Address)
	DEVICE BLOCK NUMBER	
	MEMORY START ADDRESS	
	WORD COUNT (2's Complement)	
	TRANSFER FUNCTIONS (expanded below)	
	COMPLETION RETURN ADDRESS	
	(DRIVER WORD-COUNT RETURN) Set to Ø	

[1]Dataset Data Block - in full, a 16-word table which provides the main source of communication between the Monitor drivers and a particular set of data being processed on behalf of a using program.

The relevant content of the Transfer Function word is as follows:



Provided that the Facility Indicator in the Driver Table described above shows that the driver is capable of satisfying the request, both from the point of view of direction and mode and of the service required, the Monitor routine places in Register Ø the relative byte address of the entry in the Driver Table containing the offset to the routine to be used (e.g., for the Transfer routine, this would be 1Ø). It then calls the Driver Queue Manager, using JSR PC,S.CDB.

The Driver Queue Manager ensures that the driver is free to accept the request, by reference to the Busy Flag (Word Ø of the driver table). If this contains Ø, the Queue Manager inserts the address of the DDB from Register Ø and jumps to the start of the routine in the driver using Register 1 content to evaluate the address required. If the driver is already occupied, the new request is placed in a queue linking the appropriate DDB's for datasets waiting for the driver's services. It is taken from the queue when the driver completes its current task. (This is done by a recall to the Queue Manager from the routine just serviced, using JSR PC,S.CDQ.)

On entry to the Driver Routine, therefore, the address following the Monitor routine call remains as the "top" element of the processor stack. It can be used by the driver in order to make an immediate return to the Monitor (having initiated the function requested), using RTS PC. It should also be noted that the Monitor routine will have saved register contents if it needs them after the device action. The driver may thus freely use the registers for its own operations.

When the driver has completely satisfied the Monitor request, it should return control to the Monitor using the address set into the DDB. On such return, Register Ø must be set to contain the address of the DDB just serviced and since the return will normally follow hardware interrupt, Registers Ø-5 at the interrupt must be stored on top of the stack.

A.3 Driver Routines

A.3.1 TRANSFER

The sole purpose of the TRANSFER routine is to set the device in motion. As indicated above, the information needed to load the hardware registers is available in the DDB, whose address is contained in the first word of the driver. Conversion of the stored values is, of course, the function of the routine. It must also enable the interrupt; however, it need not take any action to set the interrupt vectors as these will have been preset by the Monitor when the driver is brought into core. Having then given the device GO, an immediate return to the calling processor should be made by RTS PC.

A.3.2 Interrupt Servicing

The form of this routine depends upon the nature of the device. In most drivers it will fall into two parts, one for handling the termination of a normal transfer and the other to deal with reported error conditions.

For devices which are word or byte-oriented, the routine must provide for individual word or byte transfers, with appropriate treatment of certain characters (e.g., TAB or Null) and for their conversion between ASCII or binary and any special device coding scheme, until either the word count in the DDB is satisfied or an error prevents this. On these devices, the most likely cause for such error is the detection of the end of the physical medium; its treatment will vary according to whether the device is providing input or accepting output. The calling program will usually need to take action in the former case and the driver should merely indicate the error by returning the unexpired portion of the word count in DDB Word 7 on exit to the Monitor. Output End of Data, however, will, in general, require operator action. To obtain this, the driver should call the Error Diagnostic Print routine within the Monitor by:

```
MOV   DEVNAM,-(SP)         ;SHOW DEVICE NAME
MOV   #4Ø2,0(SP)           ;SHOW DEVICE NOT READY
IOT                        ;CALL E.D.P.
```

On the assumption that the operator will reset the device for further output and request continuation, the driver must follow the above sequence with a Branch or Jump to produce the desired resumption of the transfer.

Normal transfer handling on blocked devices (or those like RF11 Disk which are treated as such) is probably simpler since the hardware takes care of individual words or bytes and the interrupt only occurs on completion. Errors may arise from many more causes, and their handling is, as a result, much more complex and device dependent. In general, those which indicate definite hardware malfunctions must lead to the situation in which the operator must be informed by diagnostic message and the only recourse after rectification will be to start the program over.

At the other end of the scale there are errors which the driver itself can attempt to overcome by restarting the transfer - device parity failure on input is a common example. If a retrial, or several, still does not enable a satisfactory conclusion, the driver should normally allow programmed recovery and merely indicate the error by Bit 17 of DDB word 5. Nevertheless, because the program may wish to process the data despite the error, the driver should attempt to transfer the whole block requested if this has not already been effected. Between these two extremes, the remaining forms of error must be processed according to the type of recovery deemed desirable.

Whether the routine uses processor registers for its operation or not will naturally depend on considerations of the core space saved against the time taken to save the user's content. However, on completion (or error return) to the Monitor, as indicated in an earlier paragraph, the calling routine expects the top of the stack to contain the contents of all Registers Ø-5 and Register Ø to be set to the address of the DDB just serviced. The drive must, therefore, provide for this.

A.3.3 OPEN

This routine need be provided only for those devices for which some hardware initialization is required by the user. It should not normally appear in drivers for devices used in a file-oriented manner. Its presence must be indicated by the appropriate bit (Bit 7) in the driver table Facility Indicator.

The routine itself may vary according to the transfer direction of the device. For output devices, the probable action required is the transmission of appropriate data, e.g., CR/LF at a keyboard terminal, form-feed at a printer, or null characters as punched leader code, and for this a return interrupt is expected. The OPEN routine should then be somewhat similar to that for TRANSFER in that it merely sets the device going and makes an interim return via RTS PC, waiting until completion of the whole transmission before taking the final return address in the DDB.

On the other hand, an input OPEN will likely consist of just a check on the readiness of the device to provide data when requested. In this case, the desired function can be effected without any interrupt wait. The routine should, therefore, take the completion return immediately. Nevertheless, it must ensure that the saved PC value on top of the stack from the call to S.CDB is appropriately removed before exit. In the case of drivers which can only service one dataset at a time (i.e., Bit Ø of their Facility Pattern word is set to Ø) and can never, therefore, be queued, it will be sufficient merely to use TST (SP)+ to effect this. A multi-user driver, however, must allow for the possibility that it may be recalled to perform some new task already waiting in a queue. This is shown by the byte at DDB-3 being non-Ø. In this case, the intermediate return to the routine originally requesting the new task has already been made directly by S.CDB. The address now on top of the stack is the return to the routine, whose task the driver has just completed and which has called S.CDQ to dequeue the driver. This return must be taken when the first routine has performed its Completion Return processing. Moreover, this first routine expects to exit as from an interrupt. When a driver is recalled from a queue, it must simulate this interrupt. A possible sequence might be:

```
        MOV    DRIVER, RØ        ;PICK UP DDB ADDRESS
        MOV    (SP)+,R5          ;SAVE INTERIM RETURN
        TSTB   -3(RØ)            ;COME FROM QUEUE?
        BEQ    EXIT
        MOV    @#177776,-(SP)    ;IF SO, STORE STATUS
        MOV    R5,-(SP)          ;...& RETURN
        SUB    #14,SP            ;DUMMY SAVE REGS
EXIT:   JMP    @14(RØ)
```

A.3.4 CLOSE

As with OPEN, this routine should provide for the possibility of some form of hardware shut down such as the punching of trailer code and is not necessary for file-structured devices. Moreover, it is likely to be a requirement for output devices only. If it is provided, Driver Table Facility Indicator (Bit 6) must be set.

Again, the probable form is initialization of the hardware action required, with immediate return via RTS PC and eventual completion return via the DDB-stored address.

A.3.5 SPECIAL

This routine may be included if either the device itself contains the hardware to perform some special function or there is a need for software simulation of such hardware on other devices, e.g., tape rewind. It should not be provided otherwise. Its presence must be indicated by Bit 5 of the Facility Indicator.

The function itself is stored by the Monitor as a code in the DDB as shown earlier. When called, the driver routine must determine whether such function is appropriate in its case. If not, the completion return should be taken immediately with prior stack clearance, as discussed under OPEN. For a recognized function, the necessary routine must be provided. Again, its exit method will depend upon the necessity for an interrupt wait or otherwise.

A.4 Drivers for Terminals

The rate of input from terminal devices is normally dictated externally by the operator, rather than being program-driven; moreover, for both input and output, the amount of data to be transferred on each occasion may be a varying value, i.e., a line rather than a block of standard size. Furthermore, there may be problems with the conflict between echo of input during output. As a result, drivers for such devices will demand special treatment.

Normal output operation, i.e., .WRITE by the program, is handled by the Monitor Processor. On recognizing that the device being used is a terminal, as shown by Bit 8 of the facility indicator, this routine always causes a driver transfer at the end of the user line, even though the internal buffer has not been filled. The driver, however, is given the whole of a standard buffer, padded as necessary with nulls. Provided the driver can ignore these, the effect is that of just a line of output.

Input control on the other hand, must remain driver responsibility. Overcoming the rate problem will, in most cases, require circular buffering within the driver until demanded by the Monitor. At this point, transfer of data already in should occur. If this is sufficient to fill the monitor buffer, the driver can await the next request before further transfer onward. If insufficient, it should operate as any other device and use subsequent interrupts to continue to satisfy the Monitor request. It must, nevertheless, stop any transfer at the end of a line in normal operation. In order to allow the Monitor to continue, the driver must simulate the filling of the buffer by null padding (of no consequence, since terminals are by nature character-based). (Normal operation, of course, means response to user .READ's and is indicated by the size of the buffer to be filled, namely the driver standard. Should the user be requesting .TRANS, the buffer size will vary from the standard in all likelihood and the driver may then assume he requires operation as a normal device -- complete buffer fill-up before return.)

Where input echo is a further complexity, there will doubtless be other requirements. If the echo is made immediately after the input, it may be desirable to have a second buffer to cater for the likely situation that the echo will not exactly match its origin. On the other hand, if the echo is held for any length of time, perhaps to provide correct relations between program-driven output and the echo, the second buffer could be too expensive. A larger input buffer and routines to allow for several outputs to one input character while sitting on that character might be more convenient. The conflict between such echo and program-driven output will require controlled switching within the driver input and output handlers.

DEC-11-RITA-D

PDP-11

TC11 DECTAPE DRIVER

MARCH 1971

SUPPLEMENT TO:

PDP-11 DEVICE DRIVER PACKAGE

DEC-11-NIZA-D

THIS DOCUMENT IS FOR INFORMATION PURPOSES
ONLY AND IS SUBJECT TO CHANGE WITHOUT NOTICE.

pdp11 PROGRAM LIBRARY

DRIVER for TC11 DECtape Control

The principal function of the TC11 Driver is to transfer data between the hardware control and a memory area specified by a calling Monitor routine on behalf of a user program. The number of words transferred, the DECtape transport, the absolute starting block on the tape, and the direction of tape travel in each case are all determined by the calling routine.

As required by the standard Monitor-driver interface for all devices in general and, as DECtape will be handled as such, for file-structured devices in particular, the first part of the driver consists of two consecutive tables:

a) Table of descriptors and pointers to routines included.
b) File-structured usage data

All data transfers utilize the normal read/write capability of the PDP-11 NPR facility. The driver contains a set-up sequence to initiate a search for the requisite start block and routines then to handle interrupts for continuation of such search and, if this is successful, the subsequent data transfer specified.

As a file-structured device, the opening and closing of files are the responsibility of the Monitor file management routines. There are therefore no OPEN or CLOSE routines.

Also, no routine to handle SPECIAL FUNCTIONS is currently provided. This could be added later if it is found desirable to simulate the normal operation of some similar device, e.g., rewind as for Magnetic Tape.

1. Initial Tables

Relevant entries for this driver are as follows:

WORD Ø: = Ø initially-set to address of DDB for dataset being serviced when busy, by calling routine.

WORD 1: = Facility Pattern = 140037 signifying:

a) File-structured Device
b) DECtape (or similar reversible medium)

c) Capable of Input or Output in either ASCII or Binary on more than one dataset at a time.

WORD 2: = a) Standard Buffer Size = 16 X 16-word units (i.e., 1 standard DECtape block).

b) Offset to Interrupt Service routine.

WORD 3: = a) Priority for Interrupt Service = 7

b) Ø [No OPEN routine included]

WORD 4: = a) Offset to TRANSFER Set-up routine

b) Ø [No CLOSE routine included]

WORD 5: = Ø [No SPEC FUNC routine presently]

WORD 6: = Name 'DT' in RADIX 50 format.

WORD 7: = Start Block of Directory Structure = 100

WORDS 1Ø-17: = Reserved for pointers to in-core Bit Maps for each of 8 transports supportable by TC11.

2. Processing Routines

2.1 Transfer Set-up

A Monitor routine effectively calls for transfer set-up by JSR PC, XXXX where XXXX is the start address evaluated from the offset in WORD 4 of the table. The address of the DDB containing relevant parameters will be stored in WORD Ø of the table.

The set-up routine will first set a counter for the number of returns to be made in the event of parity or timing failures in tape operations (8-9). Using the given DDB address, it then extracts the following information and actions it as shown:

(i) Block No. (DDB+4) - two copies are stored internally as controls during Start Block search as detailed below.

(ii) Word Count & Memory Address (DDB+6 & 10) - these are stored immediately in the TC11 WC & BA registers for use as soon as the Start Block has been found.

(iii) Function (DDB+12) - the requirement for Read or Write is converted from the standard Monitor specification (4 or 2) into the corresponding DECtape value (4 or 14) and stored internally until completion of block search.

(iv) Tape Unit & Motion (DDB+13). The bits showing these are associated with the DECtape Search function [3] and are set into the TC11 Control Register to initiate the search for the start block.

The set-up routine also sets two switches appropriately:

a) In any transfer, two types of interrupt may occur; the first at each block encountered during the search for the start specified; the second thereafter arising when the transfer has been completed. The switch is initially set for the first type.

b) The tape is started in the eventual transfer direction. Turn-around, however, may be necessary if the tape is badly positioned. The second switch is set initially to reflect the start direction in order to provide adequate control during such turn-around.

The driver then sets the TC11 Control Register for the search, and restores control to the calling Monitor routine, via RTS PC, to await its first interrupt.

As permitted by the General Driver Spec, the set-up routine makes full use of the processor registers, without saving or restoring their original content.

2.2 Interrupt Servicing - Search Mode

Provided that a tape block-mark is encountered without error, the search interrupt servicing routine compares the number found (from TC11 Data Register) with one copy of that for the required block, stored internally by SET-up. If the comparison shows that current tape-motion will eventually lead to the required block, the routine exits immediately and waits for a subsequent interrupt to show that the transfer may begin.

If tape-motion is in the wrong direction, the routine resets the TC11 Control register to produce tape turn-around on exit. A second turn-around will now be essential for a transfer in the require direction. The routine therefore modifies, appropriately, by 2 the copy of the block number required used in the comparison. This factor is provided so the tape is sufficiently positioned beyond the block required to ensure that it will be up to speed at the right point after the second turn. For example, in order to transfer Block 100 forward, the first turn will seek Block 76 in reverse.

An equal comparison might then result after a single turn-around. The block number found is, therefore, checked against the second, unmodified, stored value. If not equal, a turn-around has occurred: the TC11 is reset for the second time and the first stored number is restored to its original value. When both stored values and the block found are all equal, the correct tape travel is assumed and the trans fer is effected by moving the stored function into the TC11 control (byte only to avoid hardware delay imposition). The interrupt switch is changed to show that the operation is now in Transfer Mode.

In the event of an error in Search Mode, the TC11 Test Register is examined. If this shows that the cause is "End Zone Reached", the turn-around procedure is again effected, since such a condition is initially the same as being, for example, at Block 102 when 100 is wanted forwards. All other hardware-reported errors are treated as discussed in a subsequent paragraph.

Another type of error may occur but this can only be detected by software, i.e., a failure to find the block either because its number on the tape is corrupted or the one required is outside the range of the tape. For both situations the tape might rock endlessly owing to the turn-around algorithm. The search interrupt processor therefore counts the number of times a turn is effected. It gives up at the sixth attempt and requests printing of an FØ16 message with the failing Block Number as evidence.

To avoid unnecessary time wastage in the storage and retrieval of their contents, the normal search interrupt processing does not use processor registers.

2.3 Interrupt Servicing - Transfer Mode

The normal cause of an interrupt in transfer mode is the satisfactory completion of the whole of the data transfer specified. The driver must then recall the monitor routine which requested the transfer. Because this routine may have surrendered control to the user program during the period of the search and transfer operations, the driver must assume such is the case and save all register contents before setting RØ to the DDB address from its WORD Ø and taking the completion return set into DDB+14.

The interrupt may also occur if an error is determined by examination of the TC11 Test Register. In Transfer Mode, two types of errors specifically processed are Party or Timing Failure. Following either of these, the servicing routine restarts the whole process over from the original block search until at least 8 attempts to produce a satisfactory transfer have been made. If these all fail, the routine returns a flag indicating the error in Bit 15 of the relevant DDB+12.

It checks, however, whether the failure occurred at an intermediate block of a transfer involving several blocks. If such is the case, it endeavors to provide a satisfactory transfer of the remaining blocks. It then recalls the monitor at the completion return address.

Of the other types of error, transfer mode servicing also handles Non-existent Memory and End Zone. Both of these conditions are assumed to be the result of a programming error and cause printing of a fatal error message FØ15 with User Call Address as evidence.

2.4 Recoverable Errors

In both Search and Transfer modes, for errors not especially noted, a general routine is used to request printing of a diagnostic message requesting operator action. SEL and ILO errors are assumed to indicate a "Device Not Ready" state for which the device name (DT) is supporting evidence for the message 'AØØ2'. For the rest, and Mark Track Errors in particular, which might be resolved by changing tapes -- the message 'AØØ3' is printed with the TC11 Test Register content as evidence. For all these errors, the operator might request program resumption by a Monitor "Continue" command. The driver restarts the whole search and transfer process if this occurs.

3. Implementation

a. Comments on the driver listing show general methods of implementation. It should be noted, however, that in several instances, in-line code is modified. In particular, the two switches mentioned under "Setup" are variable Branch Instructions and the internal storage of data has already been indicated. This means first that the driver is not reentrant - an unlikely requirement when one control may only service the transport at a time, even though eight may be attached to it. In the second place, the driver, as written is not immediately usable in a ROM.

b. The priority level for interrupt servicing should also be mentioned. The hardware level is 6; the initial software level, however, is set at 7. This is to ensure that there will be no delay due to any other interrupt in the critical case in which the required block number has been found and a change of function from Search to Read or Write must occur within 400 msecs. The interrupt routines themselves lower the level to 6, if the critical case is not being actioned. This will mean that other interrupts may be delayed up to 50 msecs. in the worst case, the critical one.

c. A further minor point of interest is that the tape is always stopped at the end of each transfer (or when an error occurs to prevent this) in order to maintain correct tape positioning. A program STOP request is issued to effect this in all cases, even though the hardware may be set up to provide for it. However, resetting the TC11 Status Register for this purpose can remove error conditions. The content of this register is, therefore, examined (or is saved for later examination) before the STOP command is given.

4. Program Listing

A complete assembly listing of the driver follows.

```
                ;COPYRIGHT 1971, DIGITAL EQUIPMENT CORP., MAYNARD, MASS.

                ;VERSION NUMBER:        V001A

                        .TITLE  DT
                ;
                        .GLOBL  DT
                ;DECTAPE DRIVER          VERSION 1        23 JULY 70
                ;       PRESENTLY CONTAINS ONLY ROUTINE FOR TRANSFER
                ;
                ;STANDARD DRIVER TABLE:
000000 000000   DT:     .WORD   0                 ;BUSY FLAG (DDB ADDR WHEN BUSY)
000002    037           .BYTE   37,300            ;FACILITY INDICATOR
000003    300
000004    020           .BYTE   16.               ;STD BUFF SIZE/16.
000005    310           .BYTE   DT.INT-DT         ;POINTER TO INT SVCE
000006    340           .BYTE   340               ;INT SVCE PRIORITY
000007    000           .BYTE   0                 ;DESPATCH TABLE ....
000010    040           .BYTE   DT.TFR-DT         ;...FOR TRANSFER ONLY!
000011    000           .BYTE   0
000012    000           .BYTE   0
000013    000           .BYTE   0                 ;SPARE
000014 016040   DT.NAM: .RAD50  /DT/
000016 000100           .WORD   DT.DIR            ;FIXED MFD BLOCK
000020 000000           .WORD   0,0,0,0,0,0,0,0   ;POINTERS FOR BIT MAP ACCESS
000022 000000
000024 000000
000026 000000
000030 000000
000032 000000
000034 000000
000036 000000
                ;REGISTER ASSIGNMENTS:
       000000   R0=%0
       000001   R1=%1
       000002   R2=%2
       000003   R3=%3
       000004   R4=%4
       000005   R5=%5
       000006   SP=%6
       000007   PC=%7

                ;SET UP TRANSFER:
000040 011767   DT.TFR: MOV     @PC,DT.RTC        ;SET RETRY COUNT
       000444
000044 016700   DT.PR1: MOV     DT,R0             ;GET ADDRESS OF DDB ...
       177730
000050 012701           MOV     #DT.CBA,R1        ;... & OF HWR BLOCK
       177346
000054 005011           CLR     @R1
000056 022020           CMP     (R0)+,(R0)+       ;SKIP USER LINE IN DDB
000060 012067           MOV     (R0)+,DT.BRQ      ;SAVE BLOCK NO FOR LATER
       000202
000064 012011           MOV     (R0)+,@R1         ;SET READY MEMORY ADDR ...
000066 012041           MOV     (R0)+,-(R1)       ;... & WORD COUNT
000070 105067   DT.PR2: CLRB    DT.INT            ;SET INT'RUPT SW. TO SRCH
       000214
000074 016767           MOV     DT.BRQ,DT.BCK     ;SET BLK CTRL FOR SRCH
       000166
       000166
000102 012703           MOV     #100,R3           ;USED IN NEXT SEQUENCE
       000100
```

```
000106 010367           MOV     R3,DT.TAC        ;SET TURN AROUND COUNT
       000100
000112 011046           MOV     @R0,-(SP)        ;GET UNIT, DIRECTION & FUNC
000114 042716           BIC     #170341,@SP      ;CLEAR POSS. GARBAGE
       170341
000120 050316           BIS     R3,@SP           ;ADD IN INT ENB BIT
000122 131617           BITB    @SP,@PC          ;WRITE REQD?
000124 001402           BEQ     .+6              ;(READ O.K. ALRDY)*****
000126 062716           ADD     #12,@SP          ;IF SO GET DECTAPE EQUIV.
       000012
000132 111667           MOVB    @SP,DT.FRC       ;SAVE FUNC FOR LATER
       000144
000136 111716           MOVB    @PC,@SP          ;RESET FUNC TO SRCH (INT ENB)
000140 006303           ASL     R3               ;(NOW CONTAINS 200)*****
000142 031627           BIT     @SP,#4000        ;TRAVEL FORWARD?
       004000
000146 001001           BNE     .+4
000150 005203           INC     R3               ;IF SO R3 NOW 201 & SO ...
000152 110367           MOVB    R3,DT.SSW        ;MAKING BPL OR BMI AS REQD
       000023
000156 012641           MOV     (SP)+,-(R1)      ;SET DECTAPE CONTROL
000160 000207           RTS     PC               ;RETURN TO CALLER FOR NOW
                ;***** CARE USED AS LITERAL BY PREVIOUS INSTRUCTION!!!

                ;INTERRUPT SERVICE (A) - SEARCH IN PRORESS:
000162 005737   DT.SIP: TST     @#DT.CCM         ;CHECK STATUS
       177342
000166 100473           BMI     DT.SER           ;IF ERROR GO INVESTIGATE
000170 023767           CMP     @#DT.CDT,DT.BRQ  ;CHECK BLOCK FOUND
       177350
       000074
000176 001432           BEQ     DT.BFD           ;IF ONE REQD, GO ACTION
000200 100426           BMI     DT.SXT           ;GET TO BLOCK THIS WAY?
       000201   DT.SSW=.-1                       ;(BPL IF TRAVEL BACKWARD)
000202 142737   DT.TA1: BICB    #40,@#177776     ;DROP PRIORITY
       000040
       177776
000210 106227           ASRB    #0               ;HOW MANY TURNS?
       000000
       000212   DT.TAC=.-2
000214 103517           BCS     DT.BER           ;IF 6 CAN'T FIND BLOCK
000216 012746           MOV     #4000,-(SP)      ;OTHERWISE MUST TURN AROUND
       004000
000222 012746           MOV     #2,-(SP)         ;ASSUME TRAVEL NOW FWD
       000002
000226 106067           RORB    DT.SSW           ;CHECK DIRECTION
       177747
000232 103403           BCS     DT.TA2           ;IF FWD OMIT NEXT
000234 005466           NEG     2(SP)            ;IF BWD, REVERSE EVERYTHING
       000002
000240 005416           NEG     @SP
000242 162667   DT.TA2: SUB     (SP)+,DT.BRQ     ;ALLOW 2 BLKS FOR 2ND TURN
       000020
000246 062637           ADD     (SP)+,@#DT.CCM   ;SWITCH STATUS
       177342
000252 106167           ROLB    DT.SSW           ;RESET DIR SW (C BIT REVERSES)
       177723
000256 105237   DT.SXT: INCB    @#DT.CCM         ;CONTINUE SEARCH
       177342
000262 000002           RTI                      ;WAIT NEXT BLOCK
```

```
                        ;BLOCK FOUND - CHECK TRAVEL CORRECT:
000264 022727 DT.BFD: CMP      #0,#0              ;TRAVEL AS ORIGINALLY STORED?
       000000
       000000
       000266 DT.BRO=.-4
       000270 DT.BCK=.-2
000272 001343         BNE      DT.TA1             ;IF NOT MUST TURN AGAIN
000274 105267         INCB     DT.INT             ;RESET INT'RUPT SW FOR TFR
       000010
000300 112737         MOVB     #0,@#DT.CCM        ;MOVE IN CORRECT FUNC
       000000
       177342
       000302 DT.FRO=.-4
000306 000763         BR       DT.SXT             ;... & GO SET UNDERWAY
                        ;INTERRUPT SERVICE (B) - TRANSFER COMPLETE (?):
000310 000400 DT.INT: BR       .+2                ;INTERRUPT SWITCH ....
000312 000723         BR       DT.SIP             ;FOR SRCH COMES HERE!
000314 142737         BICB     #40,@#177776       ;DROP PRIORITY
       000040
       177776
000322 013746         MOV      @#V.RSAV,-(SP)     ;ON TRANSFER COMPLETE ...
       000044
000326 004536         JSR      R5,@(SP)+          ;SAVE USER REGISTERS
000330 016700         MOV      DT,R0              ;GET DDB ADDR
       177444
000334 012701         MOV      #DT.CCM,R1         ;GET STATUS ADDR
       177342
000340 012703         MOV      #10,R3             ;SET MAGIC CONSTANT
       000010
000344 005711         TST      @R1                ;ERROR CAUSE INT'RUPT?
000346 100451         BMI      DT.TER             ;IF SO GO & SEE WHY
000350 110311         MOVB     R3,@R1             ;OTHERWISE STOP TAPE ...
000352 016007 DT.TXT: MOV      14(R0),PC          ;... & TAKE COMPLETE RETN
       000014

                        ;SEARCH ERROR - DETERMINE CAUSE:
000356 005737 DT.SER: TST      @#DT.TST           ;IN END ZONE?
       177340
000362 100737         BMI      DT.TA1             ;O.K. MEANS TURN AROUND
000364 142737         BICB     #40,@#177776       ;DROP PRIORITY
       000040
       177776
000372 013746         MOV      @#V.RSAV,-(SP)     ;SAVE ALL USER REGS.
       000044
000376 004536         JSR      R5,@(SP)+
000400 012701         MOV      #DT.TST,R1         ;GET DECTAPE STATUS
       177340
000404 011146 DT.EXT: MOV      @R1,-(SP)          ;SET UP TO TELL USER
000406 012746         MOV      #DT.IRE,-(SP)
       000404
000412 032721         BIT      #14000,(R1)+       ;.... ASSUMING H-W FAILURE
       014000
000416 001405         BEQ      DT.STP             ;.... IF SEL OR ILO
000420 012716         MOV      #DT.NRE,@SP        ;DIAGNOSE TAPE FAULT DIFF.
       000402
000424 016766         MOV      DT.HAM,2(SP)       ;... AS NOT READY
       177364
       000002
000432 112711 DT.STP: MOVB     #10,@R1            ;STOP TAPE IN CASE
       000010
```

```
000436 000004          IOT                        ;GO TO DIAG PRINT
000440 004767 DT.RXT:  JSR     PC,DT.PR1          ;ON RECOVERY, SET UP RETRY
       177400
000444 013705          MOV     @#V.RRES,R5        ;RESTORE USER REGS
       000046
000450 004515          JSR     R5,@R5
000452 000002          RTI                        ;... & HOPE FOR BETTER THINGS!
              ;BLOCK NOT FOUND IN SEARCH:
000454 016746 DT.BER:  MOV     DT.BCK,-(SP)       ;GIVE BLOCK NO. AS EVIDENCE
       177610
000460 012746          MOV     #DT.BRE,-(SP)
       001416
000464 012701          MOV     #DT.CCM,R1         ;GET CONTROL ADDRESS
       177342
000470 000760          BR      DT.STP


              ;TRANSFER ERROR:
000472 032741 DT.TER:  BIT     #34000,-(R1)       ;TAPE FAILURE/OPERATOR FAULT?
       034000
000476 001342          BNE     DT.EXT             ;IF SO PRINT & WAIT RECOVERY
000500 032721          BIT     #100400,(R1)+      ;END ZONE/N.E.M?
       100400
000504 001027          BNE     DT.FER             ;IF SO TREAT AS FATAL
              ;RECOVERABLE ERRORS (TIMING OR PARITY):
000506 006327          ASL     #0                 ;RETRIED 8-9 TIMES ALRDY?
       000000
       000510 DT.RTC=.-2
000512 103352          BCC     DT.RXT             ;IF NOT TRY AGAIN ....
000514 052760          BIS     #100000,12(R0)     ;OTHERWISE SIGNAL ERROR
       100000
       000012
000522 110321          MOVB    R3,(R1)+           ;STOP TAPE IN CASE
000524 016102          MOV     1(R1),R2           ;...BUT CHK ALL WORDS DONE!
       000001
000530 001710          BEQ     DT.TXT             ;IF SO THAT'S IT!
000532 060300          ADD     R3,R0              ;GO TO WORD COUNT IN DDB
000534 162002          SUB     (R0)+,R2           ;... & USE TO DETERMINE ...
000536 000302          SWAB    R2                 ;... NO. OF BLOCKS DONE
000540 130321          BITB    R3,(R1)+           ;CHECK PRESENT TRAVEL
000542 001401          BEQ     .+4                ;ADJUST NO. ACCORDINGLY
000544 005402          NEG     R2
000546 060267          ADD     R2,DT.BRQ          ;MODIFY SEARCH START BLOCK
       177514
000552 005067          CLR     DT.RTC             ;... & RETRY COUNT
       177732
000556 004767          JSR     PC,DT.PR2          ;GO SET UP NEW START
       177306
000562 000730          BR      DT.RXT+4           ;... & WAIT RESULTS!
              ;FATAL ERRORS - END ZONE OR NON-EXISTENT MEMORY:
000564 011046 DT.FER:  MOV     @R0,-(SP)          ;GIVE CALL AS EVIDENCE
000566 012746          MOV     #DT.FRE,-(SP)      ;PRINT DIAGNOSIS
       001415
000572 000717          BR      DT.STP
```

```
              ;MISCELLANEOUSDEFINITIONS:
        000044 V.RSAV=44
        000046 V.RRES=46
        000100 DT.DIR=100
        177340 DT.TST=177340
        177342 DT.CCM=177342
        177346 DT.CBA=177346
        177350 DT.CDT=177350
        000402 DT.NRE=402
        000404 DT.IRE=404
        001415 DT.FRE=1415
        001416 DT.BRE=1416
        000001           .END
```

```
000000 ERRORS
```

```
DT         000000RG      DT.BCK = 000270R       DT.BER   000454R
DT.BFD     000264R       DT.BRE = 001416        DT.BRG = 000266R
DT.CBA = 177346          DT.CCM = 177342        DT.CDT = 177350
DT.DIR = 000100          DT.EXT   000404R       DT.FER   000564R
DT.FRE = 001415          DT.FRQ = 000302R       DT.INT   000310R
DT.IRE = 000404          DT.NAM   000014R       DT.NRE = 000402
DT.PR1     000044R       DT.PR2   000070R       DT.RTC = 000510R
DT.RXT     000440R       DT.SER   000356R       DT.SIP   000162R
DT.SSW = 000201R         DT.STP   000432R       DT.SXT   000256R
DT.TAC = 000212R         DT.TA1   000202R       DT.TA2   000242R
DT.TER     000472R       DT.TFR   000040R       DT.TST = 177340
DT.TXT     000352R       PC     =%000007        R0     =%000000
R1       =%000001        R2     =%000002        R3     =%000003
R4       =%000004        R5     =%000005        SP     =%000006
V.RRES = 000046          V.RSAV = 000044        .      = 000574R
```

DEC-11-R1DA-D

PDP-11

RF11 DISK DRIVER

MARCH 1971

SUPPLEMENT TO:

PDP-11 DEVICE DRIVER PACKAGE

DEC-11-NIZA-D

THIS DOCUMENT IS FOR INFORMATION PURPOSES
ONLY AND IS SUBJECT TO CHANGE WITHOUT NOTICE.

pdp11
PROGRAM
LIBRARY

RF11 DISK DRIVER

The RF11 Disk Driver consists of routines to initiate block transfers of data to or from the disk and to handle interrupts arising from completion or through failure.

It does not include OPEN & CLOSE processors. As a file-structured device, these will be unnecessary owing to the form of the Monitor file-manangement system. SPECIAL FUNCTION processing is also omitted. If it is found necessary to simulate the hardware function of a similar device, the necessary routine could be added later.

This driver is part of the permanently resident Monitor when the RF11 is the system disk.

The driver is in two parts: 1) a table providing the interface between the driver and the Monitor, and 2) the routines to service the calls for disk operations.

1. Driver Table

The Driver Table (DF) occupies the first nine words of the driver. It complies with the standards specified for all Monitor-driver interfacing in general, and for file-structured devices in particular. The descriptive elements of the table are set up as follows:

a)	Facilities available: = 100037	Multi-dataset handling on a single unit. Input & output in ASCII or binary. File-structured with no limit to the number of files that may be in creation at one time.
b)	Standard buffer size:	64
c)	Interrupt vector address:	204
d)	Interrupt servicing priority:	5
e)	Device name	DF
f)	Directory start block:	1
g)	No. of bit map pointers:	1

2. Service Routines

The driver contains two routines: Set-up Transfer and Service Interrupt.

2.1 Set-up Transfer (DF.TFR)

This routine first initializes a counter which is used to control the number of retries in the event of parity or timing failure. Using the address of the DDB for the dataset it is servicing (as supplied by the calling routine in the first word of the driver table), it then collects control data from the DDB and transmits it to the hardware registers for the RF11, beginning at 377460.

Two of the items involved require special processing before outward transmission; the rest are moved directly.

1. The driver block number set into the DDB must be converted to meet the platter and word structure of RF11. All the platters currently under one control are considered as a single continuous surface. As a result, the most significant bits of the block number represent the appropriate platter number and the remainder the word starting the block. The required conversion is therefore merely multiplication of the block number by 64 across 21 bits.

2. The function bits contained in the DDB automatically produce the required transfer operation. To them, however, must be added the INT ENB & GO bits (combined value 101) needed to set the RF11 Control Register correctly for the transfer operation to begin.

On completion of the set-up, control is returned to the calling Monitor routine via the interim return address stored on top of the stack by the calling sequence.

2.2 Interrupt Service (DF.INT)

The RF11 control causes a priority-5 interrupt either on satisfactory completion of the transfer or because an error has been detected. Having saved the processor registers on the stack, the servicing routine must determine which of these events has occurred by examination of bit 15 of the Control Status Register. On transfer completion, it collects the address of the DDB it is servicing from the first word of the driver table and uses it to return to the completion address set in the DDB. At this exit, RØ is set to the DDB address, as required by the established convention.

An error may be one of the several types as indicated by further bits of the Control Status or Extended Status registers. The servicing routine, however, is concerned with only two categories:

(1) Errors which can be handled internally

Parity or timing failures may be eliminated on a second or later attempt. For the sake of simplicity, a retry is initiated by restarting the transfer from the beginning again rather than from the point at which the error was detected. If finally the eighth attempt produces no satisfactory result, the processing routine sets Bit 15 of Word DDB+12 to show the failure. It then checks if any words still remain to be transferred beyond the failing one. If so, it attempts to resume the transfer from this point. If this is successful, it then takes the normal completion exit. Further failure, however, is treated as fatal.

(2) Errors which must be rectified (if at all) by the operator

All other failures cause an exit to the Error diagnostic print routine, with DSK ERROR F026 as the message and the contents of the Control Status register as evidence. Write lock-out or non-resident disk may be the result of an operator fault. The operator may be able to correct this and resume program execution by the appropriate keyboard command. Such action will probably be impossible in the case of a non-existent memory error, and other errors classified as 'HARD' in the RF11 Specification or after persistent parity or timing failures.

(3) Program Listing

A complete assembly listing of the driver follows.

```
                 ;COPYRIGHT 1971, DIGITAL EQUIPMENT CORP., MAYNARD, MASS.

                 ;VERSION NUMBER:         V001A

                          .TITLE  DF
                 ;DISK DRIVER (RF11)
                 ;        STAND-ALONE DRIVER EXPANDED FROM THAT USED AS A
                 ;        RESIDENT MONITOR ROUTINE FOR SYSTEM USAGE
                 ;                 CONTAINS SET UP & TRANSFER ROUTINES ONLY
         000000  R0=%0
         000001  R1=%1
         000002  R2=%2
         000003  R3=%3
         000004  R4=%4
         000005  R5=%5
         000006  SP=%6
         000007  PC=%7
                          .GLOBL  DF
                 ;TABLE OF STANDARDS AND POINTERS
000000  000000   DF:      .WORD   0               ;CURRENT DDB ADDRESS (0 IF IDLE)
000002     037            .BYTE   37              ;STANDARD FACILITY INDICATOR
000003     200            .BYTE   200             ;(NORMAL & FILE-BASED)
000004     004            .BYTE   4               ;STANDARD BUFFER SIZE/16
000005     102            .BYTE   DF.INT-DF       ;T.V. CONTENT
000006     240            .BYTE   240             ;PRIORITY FOR T.V.
000007     000            .BYTE   0               ;DESPATCH TABLE
000010     022            .BYTE   DF.TFR-DF       ;SHOWS TFR RTN ONLY
000011     000            .BYTE   0
000012     000            .BYTE   0
000013     000            .BYTE   0               ;SPARE
000014  014760   DF.NAM:  .RAD50  /DF/
000016  000001            .WORD   DF.DIR          ;MFD BLOCK
000020  000000            .WORD   0               ;REQUIRED FOR BIT MAP INFO



                 ;TRANSFER INITIATE
000022  011767   DF.TFR:  MOV     @PC,DF.RTC      ;ZERO RETRY COUNT
        000112
000026  111737   DF.RPT:  MOVB    @PC,@#DF.DCS+1  ;CLEAR DISK IN CASE OF ERROR
        177461
000032  016700            MOV     DF,R0           ;GET DDB ADDRESS
        177742
000036  022020            CMP     (R0)+,(R0)+     ;BUMP POINTER TO BLOCK NO.
000040  012702            MOV     #DF.DCS+12,R2   ;SET HWR POINTER
        177472
000044  111703            MOVB    @PC,R3          ;SET UP BLOCK CONVERSION
000046  012004            MOV     (R0)+,R4        ;GET BLOCK NUMBER  (******)
000050  006304            ASL     R4              ;CONVERT TO WORDS
000052  106103            ROLB    R3
000054  103375            BCC     .-4
000056  010342            MOV     R3,-(R2)        ;SET UP DISK ADDRESS & EXT.
000060  010442            MOV     R4,-(R2)
000062  012042            MOV     (R0)+,-(R2)     ;MOVE IN WORD COUNT ...
000064  012042            MOV     (R0)+,-(R2)     ;& MEMORY ADDRESS
000066  012001            MOV     (R0)+,R1        ;GET FUNCTION
000070  151701            BISB    @PC,R1          ;ADD INT ENB & GO
000072  042701            BIC     #177470,R1      ;REMOVE OTHER GARBAGE (******)
        177470
000076  010142            MOV     R1,-(R2)        ;SEND TO CONTROL
000100  000207            RTS     PC              ;RETURN TO MONITOR FOR NOW
                 ;(******) - CARE!!!! USED AS LITERAL BY PREVIOUS INSTRUCTION
```

```
                        ;INTERRUPT SERVICE
 000102 013746 DF.INT:  MOV     @#S.RSAV,-(SP)  ;SAVE REGISTERS
        000044
 000106 004536          JSR     R5,@(SP)+
 000110 012701          MOV     #DF.DCS,R1      ;ERROR CAUSE INTERRUPT?
        177460
 000114 012102          MOV     (R1)+,R2
 000116 100404          BMI     DF.ERR          ;YES - GO FIND CAUSE
 000120 016700          MOV     DF,R0           ;GET DDB ADDRESS
        177654
 000124 016007 DF.XIT:  MOV     14(R0),PC       ;RETURN MONITOR
        000014
                        ;ERROR ROUTINE:
 000130 032702 DF.ERR:  BIT     #11000,R2       ;PARITY OR MISSED?
        011000
 000134 001423          BEQ     DF.OFF
 000136 006327 DF.AGN:  ASL     #0              ;YES - RETRIED 8 TIMES?
        000000
        000140 DF.RTC=.-2
 000142 103406          BCS     DF.PER          ;IF SO FORCE CONTINUE
 000144 004767          JSR     PC,DF.RPT       ;OTHERWISE TRY AGAIN
        177656
 000150 013746 DF.REC:  MOV     @#S.RRES,-(SP)  ;RESTORE SAVED REGS.
        000046
 000154 004536          JSR     R5,@(SP)+
 000156 000002          RTI                     ;... & EXIT FOR NOW
 000160 052760 DF.PER:  BIS     #100000,12(R0)  ;RETURN PARITY FAIL FLAG
        100000
        000012
 000166 005711          TST     @R1             ;ALREADY AT BLOCK END?
 000170 001755          BEQ     DF.XIT          ;IF SO EXIT NOW
 000172 005767          TST     DF.RTC          ;OTHERWISE CHECK IF 2ND TIME
        177742
 000176 001402          BEQ     DF.OFF          ;IF SO NO POINT IN MORE
 000200 005241          INC     -(R1)           ;CONTINUE DISK TRANSFER
 000202 000762          BR      DF.REC          ;... VIA COMMON EXIT
                        ;ERROR IS NOT IMMEDIATELY RECOVERABLE:
 000204 014146 DF.OFF:  MOV     -(R1),-(SP)     ;DISK STATUS IS EVIDENCE
 000206 012746          MOV     #DF.ENO,-(SP)   ;SET UP ERROR NO.
        001426
 000212 000004          IOT                     ;GO TO DIAG. PRT.
                        ;DEFINITIONS:
        000044 S.RSAV=44
        000046 S.RRES=46
        177460 DF.DCS=177460
        000001 DF.DIR=1
        001426 DF.ENO=1426
        000001          .END


000000 ERRORS


DF          000000RG     DF.AGN    000136R      DF.DCS = 177460
DF.DIR = 000001          DF.ENO =  001426       DF.ERR   000130R
DF.INT      000102R      DF.NAM    000014R      DF.OFF   000204R
DF.PER      000160R      DF.REC    000150R      DF.RPT   000026R
DF.RTC =  000140R        DF.TFR    000022R      DF.XIT   000124R
PC       =%000007        R0       =%000000      R1      =%000001
R2       =%000002        R3       =%000003      R4      =%000004
R5       =%000005        SP       =%000006      S.RRES = 000046
S.RSAV = 000044          .        = 000214R
```

DEC-11-RIHA-D

P D P - 1 1

PC11/PCØ5 HIGH-SPEED PAPER TAPE READER/PUNCH DRIVERS

MARCH 1971

SUPPLEMENT TO:

PDP-11 DEVICE DRIVER PACKAGE

DEC-11-NIZA-D

THIS DOCUMENT IS FOR INFORMATION PURPOSES
AND IS SUBJECT TO CHANGE WITHOUT NOTICE

pdp11 PROGRAM

SECTION I

PC11 HIGH-SPEED PAPER TAPE READER DRIVER

The paper tape reader driver provides the device dependent I/O functions for the PDP-11 paper tape reader. To allow the common I/O processor to be device independent, the paper tape reader driver is a block processor. Any size block may be processed by the driver, but to provide the most efficient operation the standard buffer size is 32 words. The driver code is position independent.

1.1 DESCRIPTION

The paper tape reader driver consists of two sections: the standard driver header and the driver body.

The driver header gives the following information about the paper tape driver:

1. Capabilities
   a. Single user
   b. Input only device
   c. ASCII and BINARY both may be handled
   d. Non-file structured
2. 32 word standard buffer size
3. Interrupt entry address and priority (4)
4. Dispatch table containing entry addresses for:
   a. Open
   b. Transfer
5. Internal word count and buffer address

The driver body contains the code to perform the three paper tape reader functions: opening, reading (transfer), and interrupt servicing.

1.2 OPEN

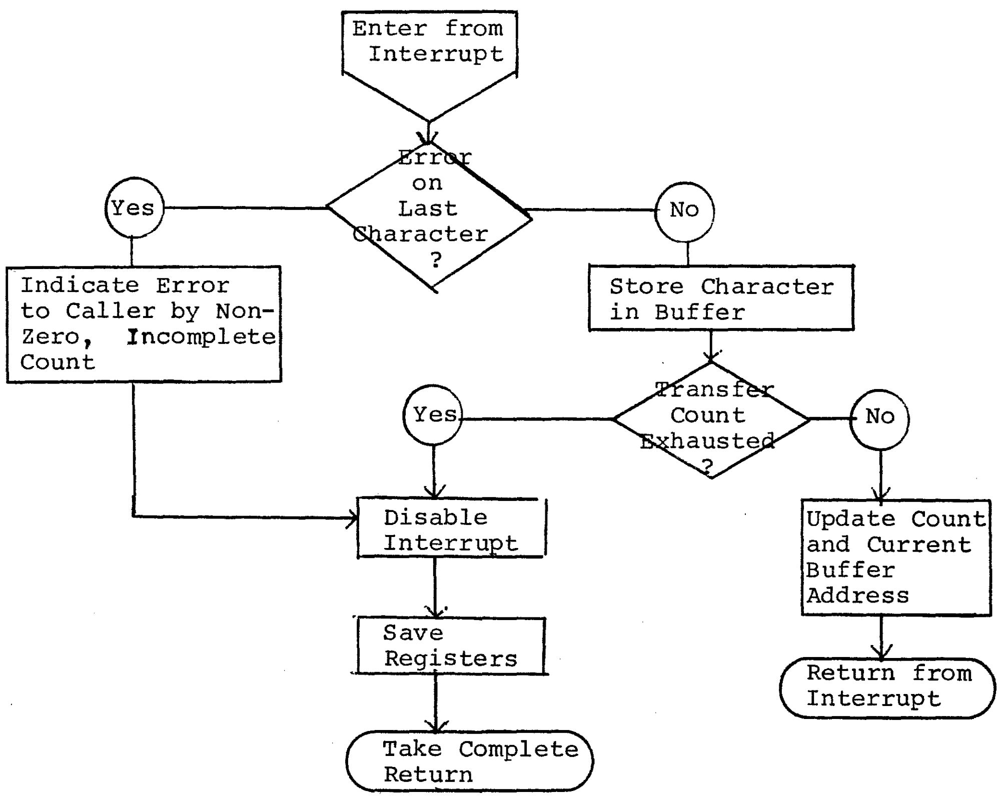
The OPEN function for the paper tape reader exists to give the user a means to ensure the reader is ready for operation (i.e., contains tape, is turned on, etc.). The OPEN routine tests the tape reader status register for an error indication. If such exists, an A002 message (Device Not Ready) is printed to the operator. The check is repeated following a return from the Diagnostic Print routine indicating that the operator has requested continuation. Because no interrupt is necessary to make this check, the routine merely removes the interim return address stored on the top of the processor stack by the calling sequence and takes the completion exit immediately (since this driver is for single-use only, there can be no queue for its services, hence it need take no action to cater for a queue situation).

1.3 TRANSFER

The TRANSFER entry initializes the driver and initiates the read of the first character. Initialization consists of storing the byte count (2 * Word Count) and buffer address from the calling DDB into the driver header positions reserved for them, and enabling the reader interrupt.

1.4 INTERRUPT SERVICE

Interrupt servicing is the heart of the paper tape reader driver. The following flow chart gives a detailed explanation of this function.



It should be particularly noted that an error during interrupt servicing signifying "Reader Off" or "Out of Tape" is considered an "End of Data" and is treated accordingly.

1.5 Program Listing

A complete assembly listing of the driver follows.

```
              ;COPYRIGHT 1971, DIGITAL EQUIPMENT CORP., MAYNARD, MASS.

              ;VERSION NUMBER:        V001A

              ; PAPER TAPE READER DRIVER (PR)
                      .TITLE   PR
                      .GLOBL   PR
       000000 R0=%0
       000001 R1=%1
       000002 R2=%2
       000003 R3=%3
       000004 R4=%4
       000005 R5=%5
       000006 SP=%6
       000007 PC=%7
              ; PREAMBLE
000000 000000 PR:     .WORD    0                ;DCURRENT DDCB OR 0
000002    234         .BYTE    PR.BP            ; FACILITIES INDICATOR
000003    000         .BYTE    0
000004    002         .BYTE    2                ; STANDARD BUFFER SIZE / 16.
000005    056         .BYTE    PR.INT-PR        ; INTERRUPT ADDRESS
000006    200         .BYTE    200              ; PRIORITY 4 INTERRUPT
000007    170         .BYTE    PR.OPN-PR        ; DISPATCH OPEN
000010    022         .BYTE    PR.TFR-PR        ; TRANSFER (IN)
000011    000         .BYTE    0                ; CLOSE
000012    000         .BYTE    0                ; SPECIAL FUNCTIONS
000013    000         .BYTE    0        ; DUMMY
000014 063320 PR.NAM: .RAD50   'PR'
000016 000000 INTCNT: .WORD    0                ; INTERNAL COUNT
000020 000000 STOADD: .WORD    0                ; STORE NEXT ADDRESS
              ; MAIN DRIVER
              ;  BEGIN TRANSFER
000022 016700 PR.TFR: MOV      PR,R0            ; GET DDB
       177752
000026 016004         MOV      10(R0),R4        ; PRESERVE USER COUNT
       000010
000032 006304         ASL      R4               ; BYTE COUNT
000034 010467         MOV      R4,INTCNT
       177756
000040 016067         MOV      6(R0),STOADD     ; SAVE BUFFER ADDRESS
       000006
       177752
000046 052737         BIS      #101,@#PR.CSR    ; ENABLE INTERRUPT
       000101
       177550
000054 000207         RTS      PC               ;RETURN
              ; THE PR IS DRIVEN BY THE FOLLOWING INTERRUPT ROUTINE
              ;
000056 005737 PR.INT: TST      @#PR.CSR         ; TEST FOR ERROR
       177550
000062 100414         BMI      PR.ERR           ; YES
000064 113777         MOVB     @#PR.BUF,@STOADD ; STORE CHARACTER
       177552
       177726
000072 005267         INC      STOADD           ; UPDATE
       177722
```

```
000076 005267           INC      INTCNT             ;  POINTERS
       177714
000102 001404           BEQ      PR.DNE
000104 052737           BIS      #101,@#PR.CSR      ; ENABLE
       000101
       177550
000112 000002           RTI                         ;  AND RETURN
              PR.ERR:
000114 013746 PR.DNE:   MOV      @#PR.SAV,-(SP)     ; SET UP JSR
       000044
000120 004536           JSR      R5,@(SP)+
000122 105037 PR.DIS:   CLRB     @#PR.CSR           ; DISABLE INTERRUPT
       177550
000126 016700           MOV      PR,R0              ; DDB ADDRESS
       177646
000132 016701           MOV      INTCNT,R1          ; REMAINING COUNT
       177660
000136 001405           BEQ      PR.FRT             ; NONE
000140 162701           SUB      #6,R1              ;  ROUNDED TO WORDS (AND TEAR)
       000006
000144 006201           ASR      R1
000146 010160           MOV      R1,16(R0)          ; RETURN RESULT TO CALLER
       000016
000152 000170 PR.FRT:   JMP      @14(R0)            ; COMPLETION RETURN
       000014
              ; OPEN ROUTINE:
000156 016746 PR.OPR:   MOV      PR.NAM,-(SP)       ; ADDITIONAL INFO
       177632
000162 012746           MOV      #402,-(SP)         ;NOT READY - 1,2 ERR MSG
       000402
000166 000004           IOT
000170 005737 PR.OPN:   TST      @#PR.CSR           ; TAPE READY
       177550
000174 100770           BMI      PR.OPR             ; NO
000176 005726           TST      (SP)+              ;CLEAR CALL FROM STACK
000200 016700           MOV      PR,R0              ;GET DDB ADDRESS
       177574
000204 000762           BR       PR.FRT             ; .... & TAKE COMPLETE RETN
              ;
       177552 PR.BUF=177552
       177550 PR.CSR=177550
       000234 PR.BP=234
       000044 PR.SAV=44

       000001           .END

000000 ERRORS


INTCNT    000016R       PC        =%000007       PR        000000RG
PR.BP   = 000234        PR.BUF  = 177552         PR.CSR  = 177550
PR.DIS    000122R       PR.DNE    000114R        PR.ERR    000114R
PR.FRT    000152R       PR.INT    000056R        PR.NAM    000014R
PR.OPN    000170R       PR.OPR    000156R        PR.SAV  = 000044
PR.TFR    000022R       R0        =%000000       R1        =%000001
R2        =%000002      R3        =%000003       R4        =%000004
R5        =%000005      SP        =%000006       STOADD    000020R
.         = 000206R
```

SECTION II

PCØ5 HIGH-SPEED PAPER TAPE PUNCH DRIVER

The paper tape punch driver supplies the basic device dependent operating functions for the PDP-11 paper tape punch. To facilitate the device dependent operation of the I/O common routines, the paper tape punch driver processes blocks of data to be punched. The driver will process any size block (as given in the DDB) but for efficient operation a default (standard) block size of 32 words has been chosen.

The paper tape reader driver provides open, close, transfer, and interrupt servicing functions. The open and close functions cause the paper tape punch to punch two fanfolds of blank leader and trailer tape respectively. The transfer function causes the punching of the given block of data. Since the PDP-11 paper tape punch punches one character at a time, the interrupt servicing function provides the actual control of the punch for each of the other functions.

2.1 DESCRIPTION

The paper tape punch driver consists of two distinct parts: the standard driver table and the driver body.

The driver table contains the following information:

1. Facilities indicator - The facilities provided by the paper tape punch driver are:
   a) Single User
   b) Output only
   c) ASCII or Binary format
   d) Non-file Structured
2. 32 word standard buffer size
3. Run at priority 4
4. Internal information
   a) Trailer Indicator
   b) Internal byte count
   c) Internal (byte) buffer address

The code for the paper tape driver is organized as follows. The open, close, and transfer routines perform their initialization processes and control is transferred to the interrupt service routine for actual control of the data transfer. The initialization processes consist of setting the internal byte count, the beginning buffer address, and the trailer indicator (Ø implies open/close in process, 1 otherwise). The interrupt servicing routine is then called. Leader/trailer punching and actual transfer punching differ only in that the internal buffer address always points to a zero in the former case, and this pointer is incremented through the block in the later case. Upon <u>total</u> completion of the requested operation, the DDB completion return is taken; the DDB intermediate return occurs immediately upon initiation of the punching of the initial byte. At each interrupt the detection of an error (Punch Out of Tape) results in a request for an A002 message at the console (Device Not Ready). If a return from the Diagnostic Print routine occurs, indicating an operator request to continue, the function is again resumed.

2.2 Program Listing

A complete assembly listing of the driver follows.

```
                ;COPYRIGHT 1971, DIGITAL EQUIPMENT CORP., MAYNARD, MASS.
                ;VERSION NUMBER:         V001A
                        .TITLE  PP
                        .GLOBL  PP
        000000          R0=%0
        000001          R1=%1
        000002          R2=%2
        000003          R3=%3
        000004          R4=%4
        000005          R5=%5
        000006          SP=%6
        000007          PC=%7
                ; PAPER TAPE PUNCH DRIVER (PP)
                ;  PREAMBLE
                ;
000000  000000  PP:     .WORD   0               ; CURRENT DCB OR 0
000002     332          .BYTE   PP.BP           ; FACILITIES
000003     000          .BYTE   0
000004     002          .BYTE   2               ; 32 WORD STD BUFFER
000005     074          .BYTE   PP.INT-PP       ; TRANSFER ADDRESS
000006     200          .BYTE   200             ; STATUS
000007     206          .BYTE   PP.OPN-PP       ; RELATIVE ADDRESSES FOR OPEN
000010     024          .BYTE   PP.TFR-PP       ; TRANSFER
000011     206          .BYTE   PP.CLS-PP       ; CLOSE
000012     000          .BYTE   0,0             ; SPF & SPARE
000013     000
000014  063200  PP.NAM: .RAD50  'PP'
000016  000001  PP.TRL: .WORD   1               ; TRAILER INDOCATOR = 0
000020  000000  PPCT:   .WORD   0               ; INTERNAL COUNT
000022  000000  PPFPT:  .WORD   0               ; CURRENT BUFFER POINTER
                ;
```

```
                        ; DRIVER BODY
000024  016700  PP.TFR: MOV     PP,R0             ; GET CURRENT DDB
        177750
000030  016067          MOV     6(R0),PPFPT       ; GET BUFFER POINTER
        000006
        177764
000036  016004          MOV     10(R0),R4         ; PRESERVE WORD COUNT
        000010
000042  006304          ASL     R4                ; CONVERT TO BYTES
000044  010467          MOV     R4,PPCT           ;  AND SAVE
        177750
000050  112767          MOVB    #1,PP.TRL                ; RESET TO TFR
        000001
        177740
000056  011646  PP.UEN: MOV     (SP),-(SP)        ; SIMULATE INTERRUPT
000060  013766          MOV     @#ST.ATS,2(SP)    ;  FROM JSR PC,XXX
        177776
        000002
000066  013737          MOV     @#PP.VCT,@#ST.ATS        ; RUN UNDER PUNCH STATUS
        000076
        177776
000074  005737  PP.INT: TST     @#PP.CSR          ; PUNCH OUT OF PAPER OR OFF
        177554

000100  100434          BMI     PP.ERR            ;  YES
000102  005767          TST     PPCT
        177712
000106  001416          BEQ     PP.DNE            ; ALREADY FINISHED
000110  005267          INC     PPCT              ; COUNT THIS ONE
        177704
000114  117737          MOVB    @PPFPT,@#PP.BRG ; MOVE CHARACTER TO PUNCH
        177702
        177556
000122  105767          TSTB    PP.TRL            ; TRAILER OR NO
        177670
000126  001402          BEQ     PP.NOI            ; TRAILER
000130  005267          INC     PPFPT             ; NEXT ADDRESS OF BUF.
        177666
000134  052737  PP.NOI: BIS     #100,@#PP.CSR     ; ENABLE INTERRUPT
        000100
        177554
000142  000002          RTI                       ; RETURN
000144  013767  PP.DNE: MOV     @#PP.SAV,.+10     ; SAVE REGS FOR RETURN
        000044
        000002
000152  004537          JSR     R5,@#0
        000000
000156  005037          CLR     @#PP.CSR          ; DISABLE INTERRUPT
        177554
000162  016700  PP.IGN: MOV     PP,R0             ; CURRENT DDB
        177612
000166  000170          JMP     @14(R0)           ; COMPLETION RETURN
        000014
000172  012746  PP.ERR: MOV     #63200,-(SP)      ;SHOW DEVICE NAME
        063200
000176  012746          MOV     #402,-(SP)        ; PRINT 1-2 ERR MSG
        000402
000202  000004          IOT                       ;  NOT READY
000204  000733          BR      PP.INT
                PP.OPN:
```

```
000206 105067 PP.CLS: CLRB     PP.TRL            ; INDICATE TRAILER OPERATION
       177604
000212 010767         MOV      PC,PPFPT
       177604
000216 062767         ADD      #PP.TRL-.,PPFPT ; SET BUFADDR
       177600
       177576
000224 012767         MOV      #177524,PPCT     ; Z FOLDS TRAILER
       177524
       177566
000232 000711         BR       PP.UEN            ; NORMAL FROM HERE ON


       177776 ST.ATS=177776
       000076 PP.VCT=76
       177554 PP.CSR=177554
       177556 PP.BRG=177556
       000044 PP.SAV=44
       000332 PP.SP=332
       000162 PP.SPF=PP.IGN
       000001        .END


000000 ERRORS


PC       =%000007       PP         000000RG     PPCT       000020R
PPFPT     000022R       PP.BP   =  000332       PP.BRG  =  177556
PP.CLS    000206R       PP.CSR  =  177554       PP.DNE     000144R
PP.ERR    000172R       PP.IGN     000162R      PP.INT     000074R
PP.NAM    000014R       PP.NOI     000134R      PP.OPN     000206R
PP.SAV =  000044        PP.SPF  =  000162R      PP.TER     000024R
PP.TRL    000016R       PP.UEN     000056R      PP.VCT  =  000076
R0       =%000000       R1        =%000001      R2        =%000002
R3       =%000003       R4        =%000004      R5        =%000005
SP       =%000006       ST.ATS  =  177776       .       =  000234R
```

DEC-11-R2DA-D

PDP-11

RK11 DISK DRIVER

OCTOBER 1971

SUPPLEMENT TO:

PDP-11 DEVICE DRIVER PACKAGE

DEC-11-NIZA-D

pdp11 PROGRAM LIBRARY

RK11 DISK DRIVER

The RK11 Disk Driver consists of routines which initiate block transfers of data to or from a disk cartridge and which handle interrupts arising from normal completion or errors.

Special functions, OPEN and CLOSE processing, are not necessary and thus not supported. Advance seeks are not supported in this initial release for several reasons, among which are:

- The majority of the DOS installations which utilize the RK have only one unit, so the extra code in the driver (approximately $25Ø_{1Ø}$ words) would be detrimental in most cases.
- No DOS system programs do their I/O in a manner which would reap huge benefits by seeking ahead.
- The Monitor would have to be altered to inform the RK driver before a Bus Init is issued.

The driver should be assembled at each installation where

(a) the RK is the system residence disk, or

(b) low density drives are present.

If the RK is the system residence disk, then define SYSDV at assembly time. If low density drives are present, then proceed as follows:

(a) If all drives are low density, then define LOWDEN at assembly time.

(b) If there is a mixture of high and low density drives, then define MIXED at assembly time and define CONFIG as follows:

> Imagine CONFIG as an 8 bit field, the rightmost bit of which corresponds to unit Ø. If a bit in a given position is one (1), then that particular drive is low density. For example, CONFIG=12(8) [ØØØØ1Ø1Ø(2)] indicates that units 1 and 3 are low density.

LOWDEN and MIXED should not be simultaneously defined. If they are, MIXED is ignored, i.e., the assembly proceeds as if LOWDEN is defined and MIXED is undefined. If MIXED is defined, but CONFIG is not, an assembly error will result, viz., a "U" flag on the line labeled DENIND.

issued was not a drive reset (see below), the completion return (@(DDB+14)) is taken. If it is an error situation, then an attempt to re-try will be made if the error was one of

(1) any "soft" error,
(2) seek incomplete,
(3) read timing error,
(4) data late, or
(5) seek error

All other error conditions result in a fatal error message. In addition, if the word count is not zero after eight re-tries, a fatal error message is issued. Otherwise, a parity error is returned.

NOTE

> Errors (2), (3), (4), and (5) above are among the "hard" errors. A control reset must be issued in order to continue. Additionally, a drive reset must be issued in order to continue after a seek incomplete. Thus, if the last function issued was a drive reset, the re-try logic is called.

4. Program Listing

A listing follows, conditionalized for

(a) the RK not being the system residence disk, and
(b) all drives being high density.

```
000054 006201          ASR     R1              ;LEFT-JUSTIFY UNIT
000056 006001          ROR     R1
000060 006001          ROR     R1
000062 006001          ROR     R1              ;UNIT NOW AS DESIRED
000064 022020          CMP     (R0)+,(R0)+     ;POINTER DDB+BLOCK
000066 012002          MOV     (R0)+,R2
                       .IFDF   MIXED
                       .IFNDF  LOWDEN
                       MOV     (PC)+,R3        ;GET DENSITY PATTERN
                       .WORD   CONFIG
                       ASL     R3              ;MOVE APPROP. TO UNIT
                       DEC     R4
                       BGE     .-4
                       BCC     .+4             ;IF LOW DENSITY ...
                       ASL     R2              ;ADJUST BLOCK NO.
                       .ENDC
                       .ENDC
                       .IFDF   LOWDEN
                       ASL     R2
                       .ENDC


000070 020227          CMP     R2,#4800.       ;IS BLOCK WITHIN BOUNDS?
       011300
000074 103410          BLO     DKIN20          ;YES - BRANCH
000076 014046          MOV     -(R0),-(R6)     ;OUTPUT ILLEGAL BLOCK NUMBER
000100 012746          MOV     #1435,-(R6)     ;AND F035
       001435
000104 000470          BR      DKER20          ;... AFTER SYSDV CHK
000106 060201 DKIN10:  ADD     R2,R1           ;ADD IN VALID QUOTIENT
000110 006202          ASR     R2              ;ADJ REMAINDER FOR DIV BY 12
000112 006202          ASR     R2
000114 060402          ADD     R4,R2
000116 010204 DKIN20:  MOV     R2,R4           ;DIVIDE BY 16 - SAVE REMAINDER
000120 042704          BIC     #177760,R4
       177760
000124 040402          BIC     R4,R2           ;EXTRACT QUOTIENT ...
000126 001367          BNE     DKIN10          ;... IF ANY BUILD RESULT
000130 020427          CMP     R4,#12.         ;CHECK REMAINDER
       000014
000134 002402          BLT     .+6             ;IF BETWEEN 12 & 15 ...
000136 062704          ADD     #4,R4           ;... CAUSE SURFACE INCR.
       000004
000142 060401          ADD     R4,R1           ;PUT SECTOR INTO REST
000144 012704          MOV     #RKDA,R4
       177412
000150 010114          MOV     R1,@R4          ;SET UP DISK ADDRESS
000152 012044          MOV     (R0)+,-(R4)     ;SET UP MEMORY ADDRESS
000154 012044          MOV     (R0)+,-(R4)     ;SET UP WORD COUNT
000156 012001          MOV     (R0)+,R1        ;PUT IN THE FUNCTION
000160 151701          BISB    @PC,R1          ;SET I.D.E. AND GO BITS
000162 042701          BIC     #177460,R1      ;CLEAR GARBAGE -******-
       177460
000166 010144          MOV     R1,-(R4)        ;SEND FUNCTION TO CONTROL
000170 000207          RTS     PC
              ;        -******- USED AS LITERAL BY THE PREVIOUS INSTRUCTION
```

```
000310 012715 DKHER:  MOV     #1,@R5          ;CLEAR THE CONTROL
       000001
000314 105715 DKHR00: TSTB    @R5             ;DONE YET?
000316 100376         BPL     DKHR00          ;NO - LOOP
000320 032701         BIT     #1000,R1        ;IS IT SEEK INCOMPLETE?
       001000
000324 001405         BEQ     DKHR05          ;NO - BRANCH
000326 010165         MOV     R1,4(R5)        ;REPLACE DRIVE #
       000004
000332 012715         MOV     #115,@R5        ;SET UP FOR DRIVE RESET
       000115
000336 000760         BR      DKER30          ;TAKE INTERIM EXIT
000340 032702 DKHR05: BIT     #11400,R2       ;CAN WE POSSIBLY GO ON?
       011400
000344 001334         BNE     DKER00          ;YES - BRANCH
000346 032702         BIT     #20000,R2       ;IS IT WRITE LOCK OUT?
       020000
000352 001742         BEQ     DKER15          ;NO - BRANCH
000354 010046         MOV     R0,-(R6)        ;SAVE BUSY FLAG
000356 016746         MOV     DKNAM,-(R6)     ;OUTPUT NAME
       177432
000362 012746         MOV     #402,-(R6)      ;AND A002
       000402
000366 000737         BR      DKER20          ;... & GO PRINT
       000001         .END


000000 ERRORS
```

```
DK         000000RG     DKERP      000232R      DKER00     000236R
DKER10     000244R      DKER15     000250R      DKER20     000266R
DKER25     000274R      DKER30     000300R      DKHER      000310R
DKHR00     000314R      DKHR05     000340R      DKINT      000172R
DKIN10     000106R      DKIN20     000116R      DKNAM      000014R
DKREPT     000240R      DKRTRY     000044R      DKSTRT     000040R
DKXIT      000226R      PC        =%000007      PS        = 177776
RKBA      = 177410      RKCS      = 177404      RKDA      = 177412
RKDIR     = 000001      RKDS      = 177400      RKER      = 177402
RKWC      = 177406      R0        =%000000      R1        =%000001
R2        =%000002      R3        =%000003      R4        =%000004
R5        =%000005      R6        =%000006      S.RSAV    = ****** G
S.XIT     = ****** G    V.RSAV    = 000044      V.XIT     = 000042
.         = 000370R
```