RT-11
Advanced Programmer's Guide

Order No. AA-5280B-TC

digital

November 1978

This manual is a reference document for advanced RT-11 users, including FORTRAN-IV users and MACRO-11 assembly language programmers.

RT-11
Advanced Programmer's Guide

Order No. AA-5280B-TC

SUPERSESSION/UPDATE INFORMATION: This manual supersedes DEC-11-ORAPA-A-D. This manual includes Update Notice No. 1 (AD-5280B-T1), Update Notice No. 2 (AD-5280B-T2), and Update Notice No. 3 (AD-5280B-T3).

OPERATING SYSTEM AND VERSION: RT-11 V03B

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, October 1977
Revised: March 1978
July 1978
September 1978
November 1978

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	

CONTENTS

CONTENTS

FIGURES

Page

FIGURES (Cont.)

TABLES

PREFACE

The Advanced Programmer's Guide is intended as a reference document primarily for advanced RT-11 users (including FORTRAN users) and MACRO-11 assembly language programmers. Although there are no absolute prerequisites for reading and understanding the contents of this manual, it is recommended that the reader be familiar with RT-11 operating procedures, PDP-11 system architecture, PDP-11 machine language, MACRO-11 assembly language and if appropriate, another higher level language such as FORTRAN IV.

The Advanced Programmer's Guide consists of the following four chapters and three appendices:

Chapter 1, I/O Programming Conventions - This chapter presents information on RT-11 supported I/O devices, associated device handlers and the various monitor services offered by the RT-11 operating system.

Chapter 2, Programmed Requests - This chapter describes all of the RT-11 programmed requests and provides information on how to use them to develop user-written programs. Program examples are also included to facilitate the explanations.

Chapter 3, Extended Memory - This chapter deals exclusively with the RT-11 concept of memory extension. The memory extension concepts and all memory extension programmed requests are explained in this chapter. An example program utilizing all memory extension programmed requests is included to assist users in developing their own programs to use this new feature.

Chapter 4, System Subroutine Library - This chapter describes all of the RT-11 FORTRAN-callable subroutines. This chapter also contains examples of the calls and most of the subroutines.

Appendix A, Display File Handler - This appendix describes the graphics support for the RT-11 operating system. Program examples are included to assist users in developing their own display program.

Appendix B, System Macro Library - This appendix is a listing of the RT-11 System Macro Library (SYSMAC), which provides the expansions for all RT-11 macro instructions.

Appendix C, Additional I/O Information - This appendix provides software support information for RT-11 programmers.

CHAPTER 1

I/O PROGRAMMING CONVENTIONS

This chapter introduces the MACRO-11 assembly language programmer to the basic concepts and features of device handlers and interrupt service routine for the RT-11 operating system. This system includes three compatible monitors and a variety of programming development tools and system utilities. The monitors and their designations are as follows:

SJ - Single-Job
FB - Foreground/Background
XM - Extended Memory

The SJ monitor is a single user, single job system restricted to 28K words of memory. The FB monitor is a single user, two job system also restricted to 28K words of memory. PDP-11/03 systems that include the MSV11-DD memory board with a special jumper can access 30K words of memory under SJ and FB. The XM monitor is an extension of the FB monitor that supports up to 124K words of physical memory. Operational XM monitors are not distributed on the RT-11 kit. A SYSGEN must be performed to create these monitors and their device handlers. See the RT-11 System Generation Manual for details.

In addition to the monitors already discussed, the SYSGEN program allows the user to create a custom monitor, containing those features required in a particular application. Such a custom monitor can have more or fewer features and can be larger or smaller than the standard monitor (see the RT-11 System Generation Manual for details).

Single-job operation supports only one program in memory at any time; execution of the program continues until either it is completed or it is physically interrupted by the user at the console.

In a foreground/background environment (under either the FB or XM monitor), two independent programs can reside in memory. The foreground program is given priority and executes until it relinquishes control to the background program; the background program executes until control is again required by the foreground program. This sharing of system resources greatly increases the efficiency of processor usage.

RT-11 is fast, reliable, and easy to use. It incorporates a sophisticated set of programming tools for the applications or end-user programmer. These tools and techniques are discussed in subsequent sections.

1.1 MONITOR SOFTWARE COMPONENTS

The main RT-11 monitor software components are:

Resident Monitor (RMON)

Keyboard Monitor (KMON)

User Service Routine (USR) and Command String Interpreter (CSI)

Device Handlers

1.1.1 Resident Monitor (RMON)

The resident monitor is the permanently memory-resident part of RT-11. The programmed requests for most services of RT-11 are handled by RMON. RMON also contains the console terminal support (TT.SYS is not resident in SJ), error processor, system device handler, EMT processor, and system tables.

1.1.2 Keyboard Monitor (KMON)

The keyboard monitor provides communication between the user at the console and the RT-11 system. Keyboard monitor commands allow the user to assign logical names to devices, run programs, load device handlers, invoke indirect command files, and control foreground/background operations. A dot at the left margin of the console terminal page indicates that the keyboard monitor is in memory and is waiting for a user command. KMON is 7400 octal (or 3840 decimal) words long in RT-11 V03B distributed BL, SJ, and FB monitors.

1.1.3 User Service Routine (USR)

The user service routine provides support for the RT-11 file structure and handles some of the programmed requests for RT-11. It loads device handlers, opens files for read or write operations, deletes and renames files, and creates new files. The Command String Interpreter is part of the USR and can be accessed by any program to process a command string. In XM, the USR is permanently resident.

1.1.4 Device Handlers

Device handlers for the RT-11 system perform the actual transfer of data to and from peripheral devices. New handlers can be added to the system as files on the system device and can be interfaced to the system easily by using the keyboard monitor INSTALL command (see Chapter 4 of RT-11 System User's Guide).

1.2 GENERAL MEMORY LAYOUT

The diagrams in Figure 1-1 show how components of the RT-11 system are arranged in memory.

Diagram A illustrates a single-job system just after it was bootstrapped. Location 54 in the system communication area contains the value x, which represents the bottom address of RMONSJ.

Diagram B shows the same single-job system with a background job executing. KMON is not resident in memory while the job is running. If the user job needs the memory space, it can swap over the USR.

Diagram C shows a foreground/background system. Two handlers were made resident by the LOAD command. They reside below RMONFB and above the USR. There is a background job running, so KMON is not shown in memory. If the background job needs the memory space, it can swap over the USR.

Diagram D illustrates the same foreground/background system. There is a foreground job running. There is no background job, so KMON is in memory.

Diagram E shows the same foreground/background system. Both the foreground and the background jobs are in memory. The background job can swap the USR at its default location just below the foreground job. The foreground job must allocate space within its own program area in order to swap in the USR.

Diagram F shows an extended memory system. There are two loaded device handlers, and both a foreground and a background job are in memory. Note that the USR is always resident.

Diagram G illustrates some characteristics of RT-11's memory allocation scheme. The third device handler in the diagram was loaded after the foreground job was started. If the foreground job were stopped and unloaded, the space it occupied would be placed in the free memory list. If the user needed to load another handler, it would reside in that free space if it could fit. If it did not fit, it would reside below the third handler. The USR and KMON slide down in memory to accommodate such new additions.

The memory area directly above the USR contains indirect file information. This section is always located just above the USR, and moves up or down in memory along with the USR. If, for example, the third handler were unloaded, the USR and the KMON would slide up in memory, and reside just below the foreground job.

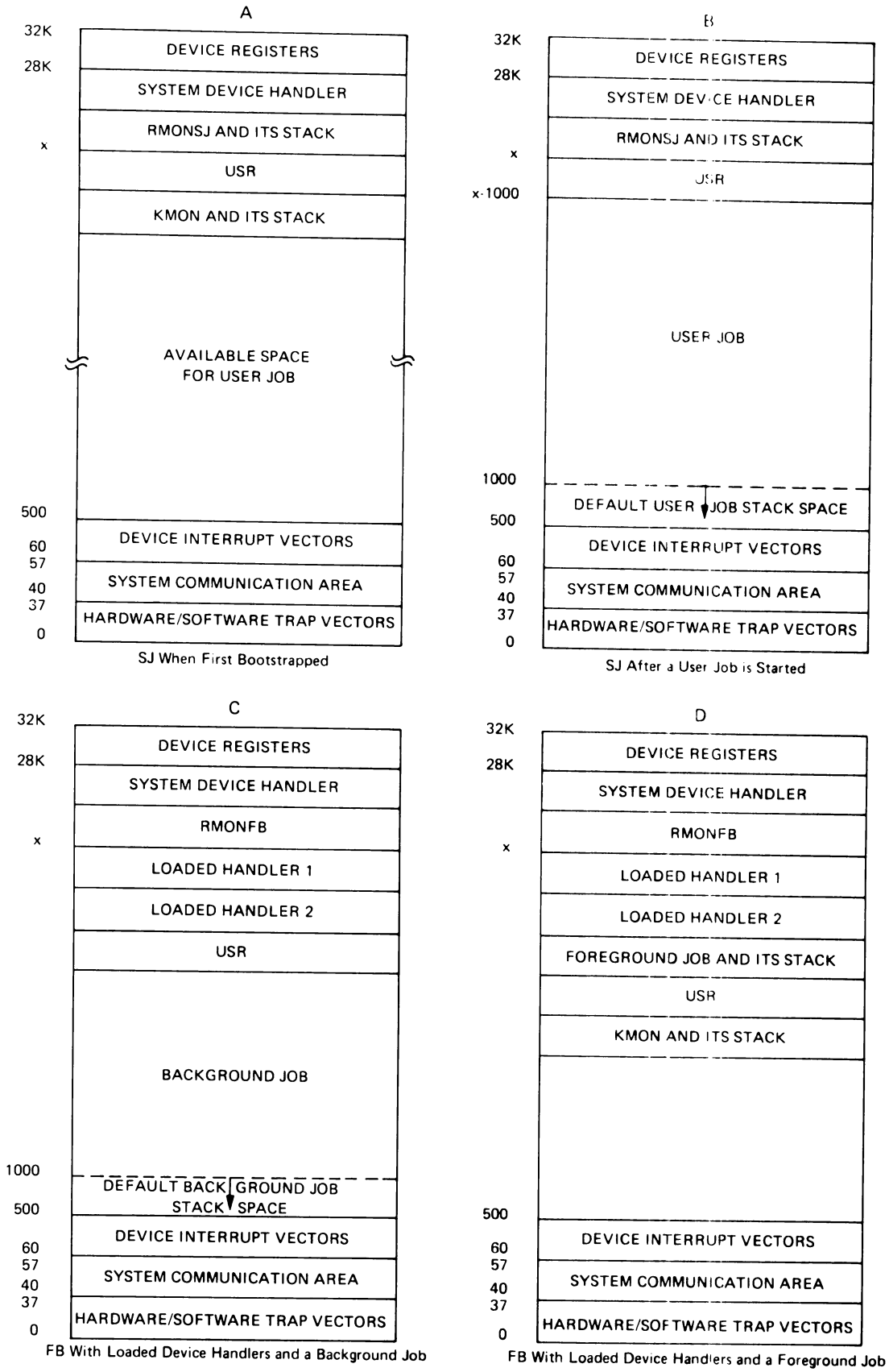

Figure 1-1 RT-11 Memory Layout

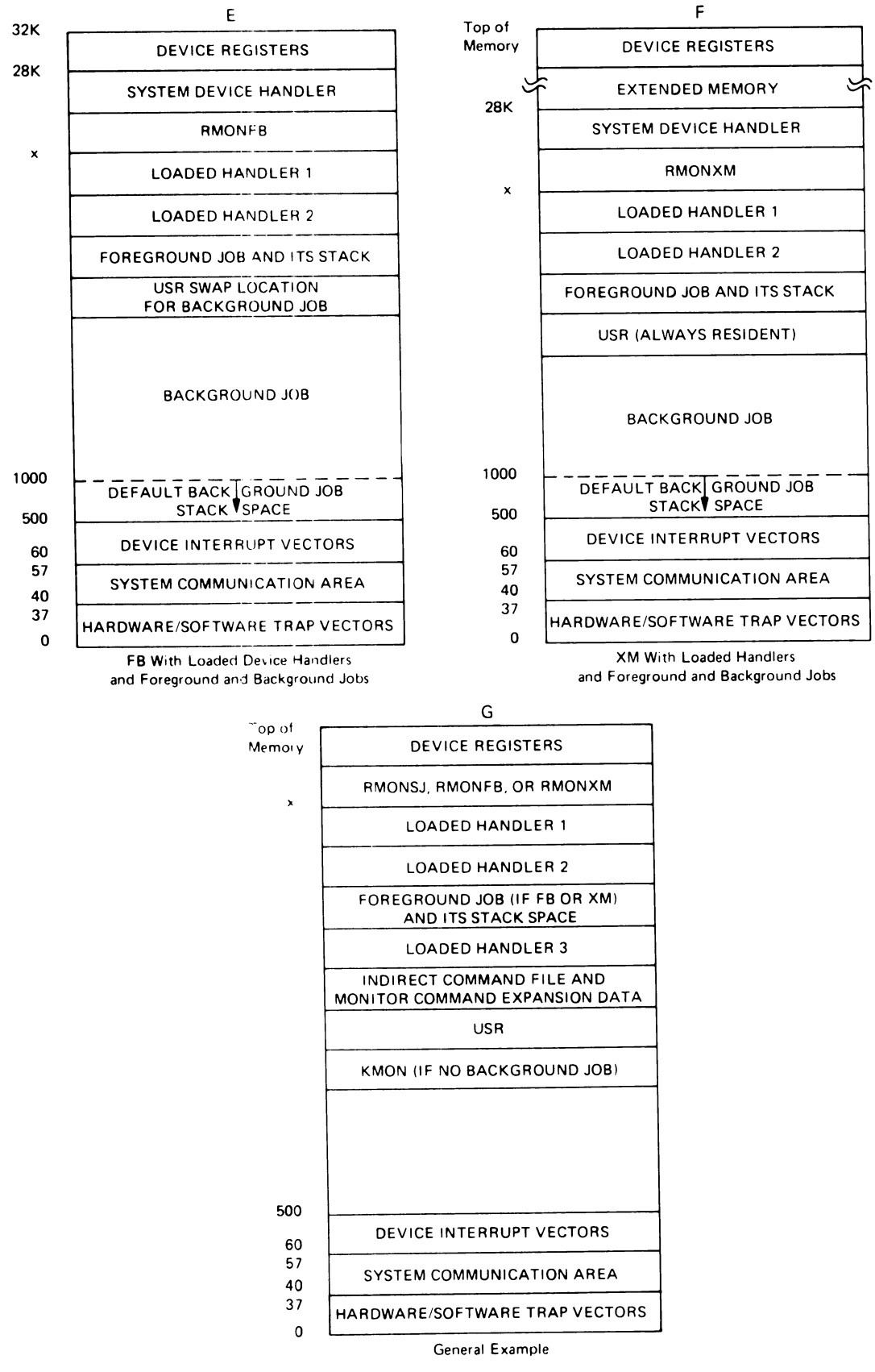

Figure 1-1 (Cont.) RT-11 Memory Layout

In addition to FRUN, which loads foreground jobs, other monitor commands can alter the memory map; these are R, RUN, GET, LOAD, UNLOAD, GT ON, GT OFF, and indirect command files invoked by "@". The LOAD command causes device handlers to be resident until an UNLOAD command is performed. The UNLOAD command removes handlers that have been loaded. The GT ON and GT OFF commands cause terminal service to utilize the VT11 or VS60 display hardware. RT-11 maintains a free memory list to manage memory. Memory space is always reclaimed if possible by moving KMON/USR up. If it cannot be reclaimed, it is placed in the free memory list.

1.3 WRITING USER INTERRUPT SERVICE ROUTINES

Certain programming conventions must be observed in RT-11 when writing user interrupt service routines. All device handlers follow these conventions. The procedures described in this section are necessary and must be followed to prevent system failures when jobs are running under RT-11.

1.3.1 Setting Up Interrupt Vectors

Devices for which no RT-11 handler exists must be serviced by the user program. For example, no LPS11 device handler exists; to use an LPS11, the user must incorporate the interrupt service routine within the program or write the device handler himself. It is the responsibility of the program to set up the vector for devices such as this. The recommended procedure is not to simply move the service routine address and 340 into the desired vector; rather, it is to precede the operation with a .PROTECT macro call. The .PROTECT ensures that neither the other job nor the monitor already has control of that device (FB and XM only). If the .PROTECT is successful, the vector can be initialized.

1.3.2 Interrupt Priorities

The status word for each interrupt vector should be set such that when an interrupt occurs, the processor takes it at level 7. Thus, a device that has its vectors at 70 and 72 has location 70 set to its service routine; location 72 contains 340. The 340 causes the service routine to be entered with the processor set to inhibit any further device interrupts.

1.3.3 Interrupt Service Routine

If conventions are followed, the processor priority will be 7 when an interrupt occurs. The first task of the interrupt service routine is to declare that an interrupt has occurred and to lower the processor priority to the correct value. This can be done by using the .INTEN macro call. The call is:

```
     .INTEN   priority
or
     .INTEN   priority,pic
```

The .INTEN call is explained in Chapter 2, Programmed Requests. On return from the .INTEN call, the processor priority is set properly;

registers 4 and 5 have been saved and can be used without the necessity of saving them again. All other registers must be saved and restored by the program if they are used.

For example, a user device interrupts at processor priority 5:

```
        DEVPRI=5

DEVINT: .INTEN DEVPRI        ;NOTE, NOT #DEVPRI
        .
        .
        .
        RTS PC
```

If the contents of the processor status word, loaded from the interrupt vector, are significant to the interrupt service routine (such as the condition bits), the PS should be moved to a memory location (not the stack) before issuing the .INTEN. The interrupt service routine uses the monitor stack and should avoid excessive use of stack space.

1.3.4 Return From Interrupt Service

When an interrupt is serviced, instead of issuing an RTI to return from the interrupt, the routine must exit with an RTS PC. This RTS PC returns control to the monitor (assuming that .INTEN has been executed), which then restores registers 4 and 5, and executes the RTI.

1.3.5 Issuing Programmed Requests at the Interrupt Level

Programmed requests from interrupt routines must be preceded by a .SYNCH call. This ensures that the proper job is running when the programmed request is issued. The .SYNCH call assumes that nothing is pushed onto the stack by the user program between the .INTEN call and the .SYNCH call. On successful completion of a .SYNCH, R0 and R1 have been saved and are free to be used. R4 and R5 are no longer free, and should be saved and restored if they are to be used. Programmed requests that require USR action must not be called from within interrupt routines.

1.3.6 User Interrupt Service Routines with the XM Monitor

There are three restrictions to using user interrupt service routines with the XM monitor. See Section 3.6.1 of this manual for specific details.

1.4 DEVICE HANDLERS

This section deals with the device handlers that are part of the RT-11 operating system. Any device dependent information or general information required by the user is contained here. No mention of a handler implies that no special conditions must be met to use that device (all disks, except diskette, RL01, and RK06/07 are in this category, and therefore are not covered here).

1.4.1 Differences Between V2 and V3 Device Handlers

The RT-11 device handler format changed slightly from version 2C to version V03. (There are no changes from version 3 to version 3B.) Most of these changes were brought about by the addition of a system generation process and many new handler options in V03. Changes are implemented through a new set of handler macros, which make conversion easier.

The new handler options being offered in version 3 and later releases include: error logging, I/O time-out, extended memory support, multi-vectored device support and fork level processing. All but fork processing are options that are determined at SYSGEN time. The monitor and the set of handlers must have matching options, so a common option definition file must be used to assemble all the components (drivers and monitors) of the system.

In addition, RT-11 version 2C and version 3 non-NPR device handlers follow different conventions for signalling the end of file condition. In version 2C, a non-NPR device handler sets the EOF bit in the channel status word as soon as it detects an end of file condition on the device (for example, no more paper in the paper tape reader). It can set the EOF bit even if the program's buffer is only partly full. Thus, the program may find the EOF bit on after a transfer that returns some usable data. Programs written for version 2C check the EOF bit after using the last data read.

In contrast, a version 3 non-NPR device handler does not set the EOF bit in the channel status word if the handler returns any usable data to the program. When such a handler detects an end of file condition on the device, it checks to see whether any data has been loaded in the program's buffer. If the buffer is not empty, the handler remembers the end of file condition but does not set the EOF bit. Instead, it fills the rest of the program's buffer with zeros and returns. The next time the handler is entered, it finds the remembered end of file condition, sets the EOF bit, and returns an empty buffer. Programs written for version 3 check the EOF bit as soon as the read is complete; they assume that the buffer is empty if the bit is on.

NOTE

Device handlers distributed with RT-11, Version 1, will not work properly with Version 2. Version 2 device handlers require changes to utilize all features of the version 3 release. Any user-written device handlers should be rewritten to comply with the Version 3 conditions. Instructions for interfacing new handlers to RT-11 are provided in the following portions of Section 1.4 of this manual.

1.4.2 The Parts of a Handler

Every RT-11 format handler has the following seven parts: the preamble, SET options, header, I/O initiation code, asynchronous trap processing code, I/O completion code, and terminator. The following sections describe the format of each of these parts. An example program of a device handler is included at the end of this section. In the following text, "dd" represents the two-character physical device name.

1. Preamble

   The preamble typically contains the trap and device register definitions and global declarations. In version V03 several new items are required in the handler preamble:

   a. An .MCALL statement is needed for the set of driver macros used in the handler.

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

   b. The device size (former contents of $DVSIZ table) and the device status word (contents of $STAT table) must be defined in the preamble, using the mnemonics ddDSIZ and ddSTS. These values are assembled into the handler .ASECT (block 0 of the SYS file) and are extracted from the handler file when needed by the .DSTATUS request.

   c. The default values of handler system generation options can be included in the preamble section. They are not

essential if a system definition file is always included when assembling the handler. Otherwise, assembly errors can occur.

The default definitions currently include:

```
.IIF NDF MMG$T,MMG$T=0     ;NO 18-BIT I/O
.IIF NDF ERL$G,ERL$G=0     ;NO ERROR LOGGING
.IIF NDF TIM$IT,TIM$IT=0   ;NO TIME-OUT
```

d. The .QELDF macro can be invoked to symbolically define all queue element offsets for the specified set of system generation options. .QELDF must be invoked after the system generation options have been defined. See Section 1.4.4.5 for the queue element offset symbolics.

2. SET Options

The option list starts at 400 in the handler .ASECT and is terminated by a zero word. Devices that can be used as the system device can have SET options when they are assembled and linked for use as non-system devices.

The system generation procedure permits the separate assembly of the system device. The SET options should be enclosed in conditionals, being assembled only if the symbol $SYSDV is undefined. The options are not assembled into a system device and the SET command is ineffective. The monitor must be patched to change an option in the system device. Section 1.4.3 describes how to add a SET option to a handler.

3. Header

The header contains standard data in fixed locations used by the monitor when it is interfacing with the handler. The header has two forms; one for a single vector device and one for a multiple vector device.

a. Single-vector handlers

The device handler header is generated by the macro .DRBEG. This macro has the following form:

```
.DRBEG name,vec,dsiz,dstat
```

where:

name — is the two-letter device name.

vec — is the device vector.

dsiz — is the number of 256-word blocks of storage on the volume (0 if non-directory structured); returned to user by .DSTATUS request.

dstat — is the device status word (not to be confused with hardware CSR); returned to user by .DSTATUS request.

This macro generates the handler .ASECT and .PSECT. It also generates any necessary globals, labels and the queue header. The load point of the handler is given the symbolic name ddSTRT. The queue header words have the names ddLQE and ddCQE.

For example: .DRBEG dd,ddVEC,ddDSIZ,ddSTS

.DRBEG RK,220,RKDSIZ,RKSTS

b. Multi-vector handlers

The monitor can load device handlers having more than one vector. This feature facilitates the use of multi-controller devices. In a driver with multiple vectors, the word normally containing the interrupt vector contains an offset to a table of vector triplets. The difference in meaning of the word is flagged by setting bit 15. The first word of the multi-vector handler header is as follows:

.WORD <table-.>/2-1+100000

where:

table is a table of vector triplets of the form:

VECTOR
TRAP ADDRESS-.
PS

The table is terminated with a zero word.

The .DRBEG macro is similar to the single vector version with the addition of a final argument, vtbl.

.DRBEG name,vec,dsiz,dstat,vtbl

where:

vtbl is the name of a table of vector triplets in a handler requiring multiple vectors.

For example: .DRBEG PC,PCVEC,PCDSIZ,PCSTS,PTBL

DX, DY, and PC are devices that use this feature.

4. I/O Initiation Section

This section is entered in system state (with context switching inhibited) by the queue manager. All registers are available for use. The queue element to be processed is pointed to by ddCQE. The I/O initiation section must return with a RTS PC.

5. Asynchronous Trap Entry Points

The asynchronous trap entry points consist of the interrupt entry and abort entry. The AST entry point branch table is created by a macro called .DRAST. This macro has the form:

.DRAST name,pri[,abo]

where:

name is the two-letter device name.

pri is the priority at which the interrupt service is to execute.

abo is the optional abort entry code symbolic label (if not specified, an RTS PC is generated).

The .DRAST macro generates the AST branch table and an .INTEN call for the interrupt service routine. The interrupt routine has the symbolic name ddINT, which is declared global by the macro if the device is to be a system device.

For example:

```
.DRAST RK,5

.DRAST DT,6,DTSTOP
```

In a multi-vector handler, the abort entry point is assumed to precede the interrupt entry point having the label ddINT, where dd is the two-letter device name declared initially in the .DRBEG macro.

6. I/O Completion

A macro called .DRFIN is provided for completing an I/O transfer and returning the queue element. The macro call is:

```
.DRFIN name
```

where name is the two-letter device name.

This macro points R4 to the handler queue head and jumps to the monitor I/O completion routine. Its expansion is identical to the current procedure and it is provided as a shorthand method of completing a transfer. It also serves to isolate system dependencies from the handler code.

For example:

```
.DRFIN RK
```

expands to:

```
MOV     PC,R4
ADD     #RKCQE-.,R4
MOV     @#54,R5
JMP     @270(R5)
```

7. **Handler Termination**

A macro is provided to terminate the device handler code. When invoked, the macro generates a table of pointers to monitor routines (interrupt entry, error logging, etc.), and computes the size of the handler load module for use by .FETCH. The macro call is:

```
.DREND name
```

where name is a two-letter device name.

For example:

```
.DREND RK
```

1.4.3 Adding a SET Option

The keyboard monitor SET command permits certain device handler parameters to be changed from the keyboard. For example, the width of the line printer on a system can be SET with a command such as:

```
SET LP WIDTH=80
```

This is an example of a SET command that requires a numeric argument. Another type of SET command is used to indicate the presence or absence of a particular function. An example of this is a SET command to specify whether an initial form feed should be generated by the LP handler:

```
SET LP FORM              (generate initial form feed)

SET LP NOFORM            (suppress initial form feed)
```

In this case, the FORM option can be negated by appending the NO prefix.

The SET command is entirely driven by tables contained in the device handler itself. Making additions to the list of SET options for a device is easy, requiring changes only to the handler, and not to the monitor. This section describes the method of creating or extending the list of SET options for a handler. The SET command is described in Chapter 4 of the RT-11 System User's Guide.

Device handlers have a file name in the form xx.SYS, where xx is the two-letter device name (for example, LP.SYS). Handler files are linked in memory image format at a base address of 1000, in which a portion of block 0 of the file is used for system parameters. The rest of the block is unused, and block 0 is never FETCHed into memory. The SET command uses the area in block 0 of a handler from 400 to 776 (octal) as the SET command parameter table. The first argument of a SET command must always be the device name; (LP in the previous example command lines). SET looks for a file named xx.SYS (in this case LP.SYS) and reads the first two blocks into the USR buffer area. The first block contains the SET parameter table, and the second block contains handler code to be modified. When the modification is made, the two blocks are written out to the handler file, effectively changing the handler. The SET parameter table consists of a sequence of four-word entries. The table is terminated with a zero word; if there are no options available, location 400 must be zero. Each table entry has the form:

```
.WORD    value
.RAD50   /option/              (two words of Radix-50)
.BYTE    <routine-400>/2
.BYTE    mode
```

where:

value — is a parameter passed to the routine in register 3.

option — is the name of the SET option; for example, WIDTH or FORM.

routine — is the name of a routine following the SET table that does the actual handler modification.

mode — indicates the type of SET parameter:

a. Numeric argument - byte value of 100
b. NO prefix valid - byte value of 200

The SET command scans the table until it finds an option name matching the input argument (stripped of any NO prefix). For the first example command string, the WIDTH entry would be found. The information in this table entry tells the SET processor that O.WIDTH is the routine to call, that the prefix NO is illegal and that a numeric argument is required. Routine O.WIDTH uses the numeric argument passed to it to modify the column count constant in the handler. The value passed to it in R3 from the table is the minimum width and is used for error checking.

The following conventions should be observed when adding SET options to a handler:

1. The SET parameter tables must be located in block 0 of the handler file and should start at location 400. This is done by using an .ASECT 400.

2. Each table entry is four words long, as described previously. The option name may be up to six Radix-50 characters long, and must be left-justified and filled with spaces if necessary. The table terminates with a zero.

3. The routine that does the modification must follow the SET table in block 0. It is called as a subroutine and terminates with an RTS PC instruction. If the NO prefix was present and valid, the routine is entered at entry point +4. An error is returned by setting the C bit before exit. If a numeric argument is required, it is converted from decimal to octal and passed in R0. The first word of the option table entry is passed in R3.

4. The code in the handler that is modified must be in block 1 of the handler file; that is, in the first 256 words of the handler.

5. Since an .ASECT 400 was used to start the SET table, the handler must start with an .ASECT 1000.

6. The SET option should not be used with system device handlers, since the .ASECT will destroy the bootstrap and cause the system to malfunction.

1.4.4 Monitor Services for Device Handlers

The RT-11 monitor provides a set of services for device handlers. These services are located in the resident monitor and can be shared by all device handlers to minimize overall system size and simplify the development and conversion of handlers. The services consist of interrupt entry processing, fork list processing, error logging, request time-out, and extended memory support. The interrupt entry processing and the fork list processing are permanent monitor features. The rest can be included or excluded at SYSGEN time. The following sections discuss the extent of each service and describe when it should be used.

1.4.4.1 Use of .FORK Process - RT-11 provides handlers with the capability of executing code as a serialized, zero-priority system process. This process, called a fork process, is similar to the service provided in other PDP-11 operating systems. A handler can request a fork process while at interrupt level (that is, after the

.INTEN request). The stack must be clean before the .FORK request is issued. That is, the stack must be in the same state when the .FORK request is issued as it was after the .INTEN request was processed. Anything pushed onto the stack after the .INTEN request must be popped off the stack before the .FORK request is issued. Control returns to the line following the .FORK request when the fork request is granted. See Figure 1-2 for a diagram of RT-11's priority structure.

The .FORK request causes the interrupt to be dismissed and adds the driver's request to a first-in/first-out (FIFO) list. The fork queue manager is activated after the last interrupt is dismissed but before the scheduler is called. Drivers are called serially in FIFO order, at priority level 0 and system state (that is, monitor stack, context switching inhibited). Registers R4 and R5 are preserved through the .FORK request, and in addition, registers R0-R3 are available for use at fork level.

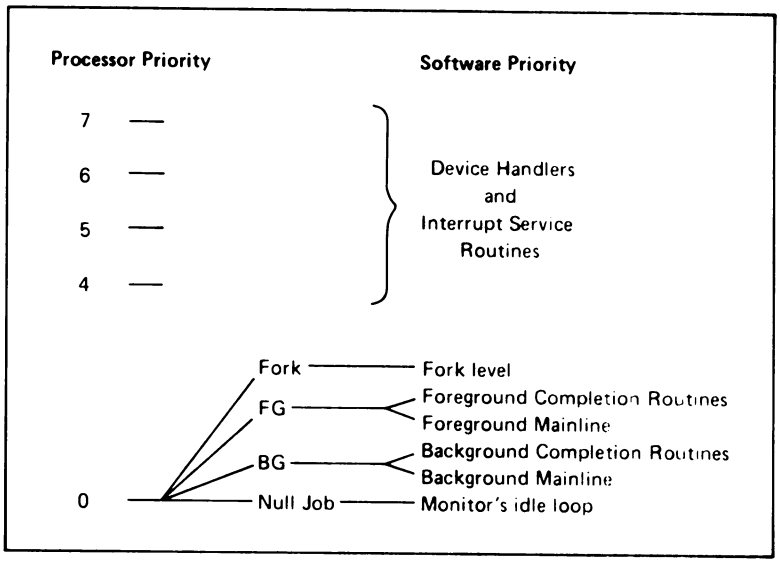

Figure 1-2 RT-11 Priority Structure

The handler must provide a four-word fork queue element that is used to preserve R4, R5 and the return PC while in the fork queue. The first word of the fork queue element is the link word and must be zero when the .FORK request is issued. A non-zero link implies the queue element is in use. However, the monitor does not check this case. This implies that the interrupt service code should check the link word before issuing the .FORK if the code could possibly be used in a re-entrant way.

The .FORK request has the form:

```
.FORK frkblk
```

where:

frkblk is the name of fork list element.

For example:

```
.FORK ddFBLK
```

where:

ddFBLK is defined as

```
ddFBLK:  .WORD 0,0,0,0
```

The .FORK request has several applications in a real-time systems environment. It permits lengthy but non-critical interrupt processing to be postponed until all other interrupts are dismissed. Its use in the card reader and line printer drivers solves some of the latency problems encountered in remote batch and DECNET applications.

For example, the card reader driver internally buffers 80 columns of card data. It receives an interrupt once per column, and translates and moves the character into its internal buffer at interrupt level. It then moves its internal buffer to the user buffer, a process that can take up to 2.5 msec. In version 2C, this process took place at priority level six, which meant that interrupts at this priority and lower could be locked out for this time. This can cause data late errors on communications devices when the card reader is active at the same time.

This problem is not solved by dropping priority to zero since the card reader can have interrupted a lower priority device. Lowering priority causes re-entrancy problems in the other device drivers. Using a .SYNCH does not always solve the problem. The SJ monitor only simulates a .SYNCH and drops priority to zero, which produces the same re-entrancy problems. The FB monitor must perform a context switch since .SYNCH returns to the caller in user context, running on the user stack. This is a lengthy process and does not occur at all if there is a compute bound foreground job.

The .FORK request is the optimum solution to the problem. It returns at priority zero, but only when all other interrupts have been dismissed and before control is returned to the interrupted user program.

Actual fork support is not provided in SJ unless timer support is generated in the monitor. Instead, the .FORK is simulated to the extent that registers R0-R3 are saved before the driver is called back. Beyond that, no serialization of interrupts is provided.

1.4.4.2 **Use of .SYNCH** - The .SYNCH request is provided to allow device drivers and user interrupt service routines to issue programmed requests. When issued, the .SYNCH request dismisses the interrupt and queues the .SYNCH block provided on the I/O completion queue (in FB and XM monitors). The job is flagged as having an I/O completion routine pending, which causes the scheduler to switch in the job.

This procedure is necessary since programmed requests must be issued in job context, and interrupts occur asynchronously. The .SYNCH request forces a context switch so the code following the .SYNCH runs in job context. In the SJ monitor the .SYNCH request simulates the register manipulations of the FB .SYNCH processor and then returns immediately to the caller at priority level 0. This occurs because the SJ monitor has a single job context and does not use an I/O completion queue. This is the reason the .SYNCH request cannot be used to simulate the functions of the .FORK request in SJ systems.

The .SYNCH request can be issued either after an .INTEN request or after a .FORK request. The handler must not have pushed anything on the stack when the .SYNCH is issued.

The XM monitor must also change the mapping mode when calling I/O completion routines. Regular I/O completion routines are run in user context and user mapping. The .SYNCH routines are run in user context, but the XM monitor requires all interrupt service routines (both user and system handler) to run in kernel mode. Thus, under the XM monitor, the .SYNCH request does not change mapping mode from kernel to user mode, but runs the .SYNCH routine in user context and kernel mapping.

1.4.4.3 **Multi-Vector Support** - A feature is provided to load device handlers having more than one vector. Previously the handler initialization code was required to set up the extra vectors. This feature makes it easier to support multi-vector devices.

The presence of multi-vector support is transparent to single-vector handlers.

The handler header normally has the form:

```
Vector
Word Offset to Interrupt Routine
PS
End of Queue Pointer
Head of Queue Pointer
```

In a handler with multiple vectors, the word containing the interrupt vector contains an offset to a table of vector triplets. The difference in meaning of this word is flagged by setting bit 15. The first word of the handler header contains:

```
.WORD <table-.>/2-1+100000
```

where table is a table of vector triplets of the form:

```
VECTOR
TRAP ADDRESS-.
PS
```

The table is terminated with a zero word. For example, a handler to handle both input and output for a PC11 High Speed Paper Tape Punch/Reader would have a header, generated by .DRBEG, of the form:

```
        .WORD   <PTBL-.>/2-1+100000   ;OFFSET TO TABLE OF VECTORS
        .WORD   PRINT-.               ;OFFSET TO FIRST INTERRUPT
        .WORD   340                   ;DUMMY PRIORITY
        .WORD   0
        .WORD   0
```

where PTBL has the form:

```
PTBL:   .WORD   70                    ;READER VECTOR
        .WORD   PRINT-.               ;READER TRAP ROUTINE OFFSET
        .WORD   340
        .WORD   74                    ;PUNCH VECTOR
        .WORD   PPINT-.               ;PUNCH TRAP ROUTINE OFFSET
        .WORD   340
        .WORD   0                     ;END OF TABLE
```

Note that only the status bits in the PS word specified are actually loaded. The priority is always forced to 7. When a single vector is loaded, the .FETCH code completely ignores the PS word specified, setting the value 340 into the vector PS word.

The macro .DRBEG contains an optional fifth parameter that points to the table.

```
.DRBEG name,vec,dsiz,dstat,vtbl
```

where:

vtbl is the name of a table of vector triplets in a driver requiring multiple vectors.

For example:

```
.DRBEG PC,PCVEC,PCDSIZ,PCSTS,PTBL
```

1.4.4.4 **Error Logging** - Error logging is an option provided to enhance system reliability. Its effective use requires that appropriate device handlers report on their activity so that a log of system I/O activity can be collected and analyzed. Both successful and unsuccessful transfers are logged. Section 1.6 describes error logging in detail. Section 1.6.3.3 describes how to call the error logger from a user-written device handler.

1.4.4.5 **Extended Memory Support for Handlers** - RT-11 supports systems with 128K words of memory. All device handlers, both NPR (non-processor request) and programmed transfer, support extended memory. RT-11 has a set of subroutines that are available to all drivers. There are three routines that move a byte to or from the user buffer or move a word to the user buffer for programmed transfer devices. Another routine converts the buffer address information supplied in the queue element into an 18-bit physical address for NPR devices.

The queue element size for unmapped systems is seven words. However, the queue element size is ten words in the mapped (XM) monitor. The .QELDF macro supplies the queue element offset symbolics and queue element byte size for the appropriate implementation (mapped or unmapped), provided the symbol MMG$T is correctly defined before .QELDF is invoked.

The queue element format in the XM monitor is essentially an extension of the unmapped format. The queue element in the XM monitor requires three additional words. One additional word is required to pass the user buffer address to the handler. The other two words are unused and provided for future expansion without another change in I/O queue element size. The queue element has the following format in the XM monitor:

SYMBOLIC	BYTE OFFSET	CONTENTS
Q.LINK	0	Link to next element
Q.CSW	2	Pointer to channel status word
Q.BLKN	4	Block number
Q.FUNC	6	Special function byte
Q.JNUM	7	Job number
Q.UNIT	7	Unit number
Q.BUFF	10	Displacement to user buffer
Q.WCNT	12	Word count
Q.COMP	14	Completion routine address
Q.PAR	16	Page address register 1 bias to map user buffer (XM only)

The monitor routines that support extended memory are called through pointers in the handler. These pointers are reserved and labelled by the .DREND macro. The monitor fills the pointers with correct absolute addresses at fetch time.

The following are the call sequences and register conditions for invoking the extended memory handler support routines in the XM monitor:

1. Convert Mapped Address to Physical Address

   The monitor routine $MPPHY (Convert Mapped Address to Physical Address) is available to NPR device handlers. It converts the virtual buffer address supplied in the queue element into an 18-bit physical address that is returned on the stack.

   Call: JSR PC,@$MPPTR

   Inputs: R5 — Contains pointer to Q.BUFF in queue element.

   Outputs: 2(SP) — Second word on stack contains high order two bits of physical address in bit positions 4 and 5.

   (SP) — First word on stack contains low order 16 bits of physical address.

   R5 — Contains pointer to Q.WCNT in queue element.

2. Move Byte to User Buffer

   The routine $PUTBYT in the resident monitor is available to programmed transfer device handlers to transfer a byte passed on the stack to the user buffer. The buffer address in Q.BUFF in the queue element is updated and mapping register overflow is detected and adjusted. The byte count is not modified.

Call: JSR PC,@$PTBYT

Inputs: (SP) First word on stack contains byte of data to be transferred.

R4 Contains pointer to Q.BLKN in queue element.

Outputs: Byte is removed from stack.
Buffer pointer is updated.
R4 is unmodified.

3. Move Byte From User Buffer

The routine $GETBYT is the complement of $PUTBYT. A byte is extracted from the user buffer and returned on the stack. The buffer pointer is updated, but the byte count is not modified.

Call: JSR PC,@$GTBYT

Inputs: R4 Contains pointer to Q.BLKN in current queue element.

Outputs: (SP) First word on stack contains byte of data from user buffer.

Buffer address (Q.BUFF) is updated.
R4 is unmodified.

4. Move a Word to User Buffer

The $PUTWRD routine is available through the $PTWRD pointer and moves a word supplied on the stack to the user buffer. Its anticipated uses are in handlers for analog devices and to return status information.

Call: JSR PC,@$PTWRD

Inputs: (SP) First word on stack contains word of data to move.

R4 Contains pointer to Q.BLKN in queue element.

Outputs: Word of data is removed from stack.
Q.BUFF is updated.
R4 is unmodified.

5. The .DREND macro generates a fifth pointer, $RLPTR, which points to the monitor routine $RELOC. This routine is reserved for use by DIGITAL software only.

1.4.4.6 **Device Time-out Support** - A SYSGEN option adds device time-out support to the monitor. This option permits device handlers to do the equivalent of a mark time without doing a .SYNCH request. Data transfers can be timed, and the driver can take action if the transfers do not complete in the expected time interval.

This feature is not used by any of the RT-11 device handlers. However, it is used by the multi-terminal monitor when the multi-terminal time-out option or remote DZ11 lines are selected

during SYSGEN. In these two cases, the device time-out support is automatically included in the monitor during SYSGEN. The device time-out option is also required for DECNET applications. The user must specifically request it in the SYSGEN dialogue when he builds a monitor for a DECNET application.

Two macros can be used only within a device handler. The macros, .TIMIO and .CTIMIO, permit the scheduling and cancelling of a mark time request. They can be issued from the entry point of the handler, from interrupt level, or from a time-out completion routine. The macros are contained in the system macro library, SYSMAC.SML.

To schedule a mark time from a handler:

```
.TIMIO tbk,hi,lo
```

where tbk is the address of a seven-word timer block containing the following:

Word	Contents
0	hi order time
2	lo order time
4	link to next queue element; 0 if none
6	owner's job number
10	owner's sequence number
12	-1 if system timer element -3 if .TWAIT element in XM
14	address of completion routine; zeroed by the monitor when the routine is called to indicate that the timer block is available for reuse.

The .TIMIO request schedules a completion routine to run after the specified number of clock ticks have occurred. The completion routine runs in user context (kernel mapping), associated with the job specified in the timer block. Registers R0 and R1 are available for use. When the completion routine is entered, R0 contains the sequence number of the request that timed out.

To cancel a mark time from a handler:

```
.CTIMIO tbk
```

where tbk is the address of the seven-word timer block used in the .TIMIO request being cancelled.

If the timer request has already timed out and been placed in the completion queue, the .CTIMIO fails, since a timer request cannot be cancelled after being placed in the completion queue. Failure to cancel the queue element is indicated by the C bit set on return from the .CTIMIO request.

1.4.5 Installing and Removing Handlers

The installation and removal of device handlers from the system is done from the keyboard monitor. Two keyboard monitor commands, INSTALL and REMOVE, make the temporary installation of a handler very easy; no patching procedures are required.

The INSTALL command has the following form:

```
.INSTALL dd
```

where dd is the two-letter device (and file) name.

The INSTALL command searches the system device for a file named dd.SYS (or ddX.SYS for XM), extracts the device status word from the handler, and updates the $STAT, $PNAME and $DVREC tables in the resident monitor. The device can now be used without rebooting the monitor.

NOTE

INSTALL is effective only on the monitor in memory. It does not permanently modify the monitor file on the system device. To permanently install a handler, the system must be patched. This requires patching the Radix-50 name into $PNAME and the device status word into $STAT. Another way is to include the INSTALL command in the startup indirect command file (STARTx.COM) that is executed on every boot. (Note that startup indirect command files are optional.) The monitor file can also be re-SYSGENED.

1.4.6 Converting Handlers to V03 Format

A V02 format device handler requires some conversion to operate under a V03 or later monitor. The conversion effort ranges from a short patch to a complete re-edit, depending on how many new features the user desires. Special device handlers require some extra effort to support the new error reporting capability of the special device interface. This conversion can be implemented in the following ways.

1.4.6.1 **Patching a V02 Format Handler** - A version V02 driver can be patched to operate under a V03 or later monitor, provided the monitor generated does not support extended memory, error logging or device I/O time-out. Four locations in block 0 of the handler file must be patched to contain handler information essential to the operation of the new .FETCH mechanism.

The four locations contain the handler size, device status word, the device block size (that is, number of 256-word blocks on the volume), and the SYSGEN options compatible with this handler. All handlers have pointers to $INTEN and $FORK and optional pointers to support routines for the SYSGEN options at the end of the handlers, which are initialized when the handler is .FETCHed. Since V02 handlers have only the $INTEN pointer, an extra word (two bytes) must be added to the actual handler size when patching. The other two locations contain the data normally present in the $STAT and $DVSIZ tables (the $DVSIZ table is eliminated in V03 and later releases of RT-11).

Location	Contents
52	Handler size in bytes (plus 2 for $FORK pointer)
54	Device size in number of 256-word blocks
56	Device status word, as contained in $STAT table.
60	SYSGEN options, must be 0

For example, to patch the V02C MT.SYS handler to function under the V03 monitor:

```
.R PATCH

FILE NAME--
*MT.SYS <RET>
*52/    0       4300 <LF>
54/     0       0 <LF>
56/     0       12011 <LF>
60/     0       0 <RET>
*E
```

NOTE

This patch does not work with V03 or later monitors having error logging, extended memory or device time-out support.

1.4.6.2 **Source Edit Conversion of Handlers** - A V02 format, non-system handler can be converted to function with the V03 or later monitors (without .FORK, error logging or extended memory support) by applying a minimal set of edits to the device source. The two essential changes are the addition of the four words described in the first method to the handler .ASECT, and the addition of a dummy .FORK pointer to the end of the handler.

The faster method is to directly edit in the .ASECT and extra word. The better method is to replace the handler header with the .DRBEG macro and insert the .DREND macro at the end of the handler. No problems will be encountered if standard RT-11 naming conventions were used in writing the handler. Neither of these methods takes full advantage of the new features of RT-11.

NOTE

To convert a version 2C device handler to version 3, change the version 2C device handler so that it sets the EOF bit in the channel status word in the proper sequence. (See Section 1.4.1.) If this change is not made, the last block of data may be lost during a data transfer.

a. (Fast Method)

Step 1: Define the device handler size, block size and status word.

For example:

```
RKDSIZ = 0
RKSTS = 20003
```

The driver size is usually defined at the end of the handler using the convention:

```
RKHSIZ = .-RKSTRT
```

Step 2: Install the handler .ASECT.

For example:

```
.ASECT
.=52
        .WORD RKHSIZ
        .WORD RKDSIZ
        .WORD RKSTS
        .WORD 0
```

Add a .CSECT after the .ASECT if one is not already in the existing handler code.

Step 3: Add a dummy $FKPTR to the end of the handler.

For example:

```
$INPTR:  .WORD 0
RKHSIZ = .-RKSTRT
```

becomes

```
$INPTR:  .WORD 0
$FKPTR:  .WORD 0
RKHSIZ = .-RKSTRT
```

b. (Best Method)

Perform steps 1, 2, 3, 4 and 7 of the full conversion method.

1.4.6.3 **Full Conversion of Device Handlers** - To take advantage of the new features, the handler must be modified. Inserting the .DRBEG, .DRAST, .DRFIN and .DREND macros makes conversion to V03 format easier, but it does not supply the functional conversion necessary to support error logging or extended memory. Difficulty of functional conversion varies with the complexity of the device and its handler.

To make the full conversion of a device handler, perform the following:

1. Insert an .MCALL containing the handler macros that are to be used in converting the handlers.

   For example:

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

2. Insert the default system build options:

   For example:

```
.IIF NDF MMG$T,MMG$T=0
.IIF NDF ERL$G,ERL$G=0
.IIF NDF TIM$IT,TIM$IT=0
```

3. Define the device block size and status words using the proper mnemonics.

   For example:
```
RKDSIZ = 0
RKSTS = 20003
```

4. Replace the handler header with the .DRBEG macro.

   For example:
```
RKSTRT: .WORD    200
        .WORD    RKINT-.
        .WORD    340
RKSYS:
RKLQE:  .WORD    0
RKCQE:  .WORD    0
```

is replaced by the macro:

```
.DRBEG RK,200,RKDSIZ,RKSTS
```

5. Replace the interrupt entry and abort entry points with the .DRAST macro (optional, but recommended).

   For example: replace the code:

```
        BR      RKDONE          ;ABORT ENTRY POINT
RKINT:  JSR     R5,@$INPTR      ;INTERRUPT ENTRY POINT
        .WORD   ^C<PR5>&340
```

   with the macro:

```
.DRAST RK,5,RKDONE
```

6. Replace the I/O completion code with the .DRFIN macro (optional, but recommended).

   For example:

   replace the code:

```
MOV     PC,R4
ADD     RKCQE-.,R4
MOV     @#54,R5
JMP     @270(R5)
```

   with the macro call:

```
.DRFIN RK
```

7. Replace the $INPTR location at the end of the handler with the .DREND macro.

   For example: replace:

```
$INPTR:  .WORD   0
RKHSIZ = .-RKSTRT
```

   with:

```
.DREND  RK
```

8. The handler can now be assembled and tested. Assembly errors can occur if RT-11 naming conventions were not followed (for example, if the queue pointers were not originally named RKLQE and RKCQE, the start of the CSECT was not named RKSTRT, and the interrupt entry point was not named RKINT). The handler should now function correctly under the SJ and FB monitors, provided that the monitors have not been SYSGENed to include any other handler features like error logging and device time-out.

9. Extended memory conversion can now be done, if desired.

   a. NPR (Non-Processor Request) Devices

      Assumptions: R5 is used to point to the queue element.

      Procedure: The buffer address supplied in the queue element in a mapped monitor is really in two parts. Q.BUFF contains the buffer displacement in the virtual

address space defined by Q.PAR. This must be converted to an 18-bit physical address, which is done by a call through $MPPTR. Two words are returned on the stack, containing the low order 16 bits and high order two bits.

For example:

```
        RKCS = nnnnn2           ;CONTROL AND STATUS
                                ;REGISTER
        RKWC = nnnnn4           ;WORD COUNT REGISTER
        RKBA = nnnnn6           ;UNIBUS ADDRESS REGISTER
           .
           .
           .
        MOV     #103,R3         ;ASSUME A WRITE
        MOV     #RKBA,R4        ;R4 -> BUFFER ADDRESS REG
        MOV     (R5)+,(R4)      ;MOVE BUFFER ADDRESS
        MOV     (R5)+,-(R4)     ;MOVE WORD COUNT
```

is replaced with the conditional code:

```
        RKCS = nnnnn2           ;CONTROL AND STATUS
                                ;REGISTER
        RKWC = nnnnn4           ;WORD COUNT REGISTER
        RKBA = nnnnn6           ;UNIBUS ADDRESS REGISTER
           .
           .
           .
.IF EQ MMG$T
.IFTF
        MOV     #103,R3         ;ASSUME A WRITE
        MOV     #RKBA,R4        ;R4 -> BUFFER ADDRESS REG
.IFT                            ;IF UNMAPPED
        MOV     (R5)+,@R4       ;MOVE BUFFER ADDRESS
                                ;TO RKBA
.IFF                            ;IF MAPPED
        JSR     PC,@$MPPTR      ;CONVERT TO 18 BITS
        MOV     (SP)+,@R4       ;MOVE LOW 16 BITS TO RKBA
.IFTF                           ;IN ANY CASE,
        MOV     (R5)+,-(R4)     ;MOVE WORD COUNT TO RKWC
   .
   .
   .
.IFF                            ;IF MAPPED
        BIS     (SP)+,R3        ;SET IN HI ORDER
                                ;ADDRESS BITS
.IFTF                           ;IN ANY CASE
  6$:   MOV     R3,-(R4)        ;START THE OPERATION
        RTS     PC              ;AWAIT INTERRUPT
.ENDC
```

For NPR devices which may be interfaced to a mass bus controller, the address extension bits must be placed in bits 8 and 9 of the control and status register rather than bits 4 and 5. For these devices (such as RJS03/04) the code above must be modified to shift the bits into place.

```
.IFF                            ;IF MAPPED
        JSR     PC,@$MPPTR      ;CONVERT TO 18 BITS
        MOV     (SP)+,@R4       ;MOVE LOW 16 BITS
        ASL     (SP)            ;SHIFT HI BITS INTO PLACE
        ASL     (SP)            ;
        ASL     (SP)            ;
        ASL     (SP)            ;
        BIS     (SP)+,R3        ;SET IN HI ORDER BITS
```

b. Programmed Transfer Devices

Assumptions: R4 points to Q.BLKN in the queue element.

Procedure: Programmed transfer devices must directly move the data to or from the user buffer. This is usually done a byte or word per interrupt, but sometimes a complete buffer is moved, as in the CR handler.

To move data the handler must save the contents of the kernel mapping register* (page address register 1), move Q.PAR to kernel page address register 1, and then move one byte or word indirectly off the contents of Q.BUFF. If more than 4K-32 words of data can be moved, the Q.BUFF address must be checked for overflow each time it is updated, since a page address register can map only 4K words of memory. A simple approach is to use one of the monitor routines provided.

For example, the original handler contains the code:

```
BYTCNT = 6          ;OFFSET TO BYTE COUNT
BUFF = 4            ;OFFSET TO BUFFER ADDRESS
   .
   .
   .
MOV     PPCQE,R4                ;R4 -> Q.BLKN
   .
   .
   .
MOVB    BUFF(R4),@#PPB          ;MOVE A CHARACTER
INC     BUFF(R4)                ;UPDATE BUFFER ADDRESS
INC     BYTCNT(R4)              ;BUMP BYTE COUNT
BEQ     PPDONE                  ;IF EQ DONE
```

which becomes the conditionalized code:

```
        BYTCNT = 6              ;OFFSET TO BYTE COUNT
        BUFF = 4                ;OFFSET TO BUFFER ADDRESS
           .
           .
           .
        MOV     PPCQE,R4        ;R4 -> Q.BLKN
           .
           .
           .
.IF EQ MMG$T                    ;IF UNMAPPED
        MOVB    BUFF(R4),@#PPB  ;MOVE CHARACTER
        INC     BUFF(R4)        ;UPDATE BUFFER ADDRESS
.IFF                            ;IF MAPPED
        JSR     PC,@$GTBYT      ;GET A CHARACTER
        MOVB    (SP)+,@#PPB     ;PUT IT OUT.
.IFTF                           ;IN EITHER CASE,
        INC     BYTCNT(R4)      ;BUMP BYTE COUNT
        BEQ     PPDONE          ;IF EQ DONE
.ENDC
```

There are cases where the monitor subroutines cannot be used. In those cases, the remapping of the kernel mapping register (page address register 1) must be done within the handler code.

---

* For an explanation of mapping registers, refer to Chapter 3.

The call to $GTBYT is equivalent to the following in-line code sequence:

```
        KISAR1 = 172342                         ;KERNEL PAR1
        MOV     @#KISAR1,-(SP)                  ;SAVE PAR1
        MOV     Q.PAR-Q.BLKN(R4),@#KISAR1       ;MAP TO USER BUFFER
        MOVB    @Q.BUFF-Q.BLKN(R4),@#PPB        ;MOVE NEXT BYTE
        MOV     (SP)+,@#KISAR1                  ;RESTORE PAR1
        INC     Q.BUFF-Q.BLKN(R4)               ;UPDATE BUFFER ADDRESS
        BIT     #40000,Q.BUFF-Q.BLKN(R4)        ;OVERFLOWS 4K LIMIT?
        BEQ     1$                              ;IF EQ, NO
        SUB     #20000,Q.BUFF-Q.BLKN(R4)        ;ADJUST DISPLACEMENT
        ADD     #200,Q.PAR-Q.BLKN(R4)           ;AND PAR1 BIAS
1$:
```

1.4.7 Device Handler Program Skeleton Outline

The following code illustrates a device handler outline. In the example the designation SK is used as the device name.

```
.TITLE  SK V03.01

; SK DEVICE HANDLER

.IDENT  /V03.01/

.SBTTL  PREAMBLE SECTION

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

; SYSGEN DEFAULT DEFINITIONS:
.IIF NDF MMG$T, MMG$T = 0
.IIF NDF ERL$G, ERL$G = 0
.IIF NDF TIM$IT, TIM$IT = 0

; DEVICE UNIBUS ADDRESSES:
.IIF NDF SK$VEC, SK$VEC = 200           ;SK VECTOR
.IIF NDF SK$CSR, SK$CSR = 177514        ;SK CONTROL STATUS REGISTER
        SKBR    = SK$CSR+2              ;SK BUFFER REGISTER

        HDERR   = 1                     ;HARD ERROR ON CHANNEL

; DEVICE STATUS INFORMATION:
SKDSIZ  = 0                             ;DEVICE BLOCK SIZE
SKSTS   = 20003                         ;DEVICE STATUS WORD

; DEFINITION OF Q ELEMENT SYMBOLICS:
.QELDF
WCNT    = Q.WCNT - Q.BLKN
BUFF    = Q.BUFF - Q.BLKN


.SBTTL  SET OPTIONS

.ASECT

. = 400

        NOP
        .RAD50  /RANDOM/
        .WORD   <0.RNDM-400>/2+100000

        .WORD   0               ;END OF LIST
```

```
O.RNDM: MOV     (PC)+,R3        ;GET NEW INSTRUCTION TO STORE
        MOVB    SP,R0           ;CHANGE INST FOR SET OPTION
        MOV     R3,SKOPT        ;STORE IT IN HANDLER BODY
        RTS     PC              ;DONE WITH SET OPTION CHANGE


.SBTTL  HEADER SECTION

        .DRBEG  SK,SK$VEC,SKDSIZ,SKSTS

; ENTRY POINT FORM QUEUE MANAGER

        MOV     SKCQE,R4        ;R4 -> CURRENT QUEUE ELEMENT
        ASL     WCNT(R4)        ;MAKE WORD COUNT A BYTE COUNT
        BCC     SKERR           ;A READ REQUEST IS ILLEGAL
        BEQ     SKDONE          ;A SEEK COMPLETES IMMEDIATELY
RET:    BIS     #100,@#SK$CSR   ;ENABLE INTERRUPTS
        RTS     PC              ;EXIT AND WAIT FOR ONE


.SBTTL  INTERRUPT TRAP PROCESSING

        .DRAST  SK,4,SKDONE

; INTERRUPT SERVICE:

.IF EQ MMG$T
.IFTF
        MOV     SKCQE,R4        ;;;R4 -> CURRENT QUEUE ELEMENT
        TST     @#SK$CSR        ;;;ERROR?
        BMI     RET             ;;;YES IF MI, HANG UNTIL CORRECT
        TSTB    @#SK$CSR        ;;;IS DEVICE READY?
        BPL     RET             ;;;NO IF PL, EXIT AND WAIT
        CLR     @#SK$CSR        ;;;YES, DISABLE INTERRUPTS

; PROCESS REMAINING CODE AT FORK LEVEL

        .FORK   SKFBLK          ;REQUEST FORK PROCESS
SKNEXT: TSTB    @#SK$CSR        ;READY FOR ANOTHR CHARACTER?
        BPL     RET             ;BR IF NOT READY
        TST     WCNT(R4)        ;ANY LEFT TO PRINT?
        BEQ     SKDONE          ;NO IF EQ, XFER IS DONE
.IFT
        MOVB    @BUFF(R4),R5    ;GET A CHARACTER
        INC     BUFF(R4)        ;BUMP BUFFER POINTER
.IFF
        JSR     PC,@$GTBYT      ;GET A CHARACTER
        MOV     (SP)+,R5        ; INTO R5
.IFTF
        INC     WCNT(R4)        ;BUMP CHARACTER COUNT
        MOV     #177770,R5      ;7 BIT ASCII
SKOPT:  NOP                     ;RANDOM OPTION BY SET COMMAND
        MOVB    R5,@#SKBR       ;PUT IT OUT TO DEVICE
        BR      SKNEXT          ;TRY FOR ANOTHER
.ENDC
```

```
.SBTTL  I/O COMPLETION SECTION

SKERR:  BIS     #HDERR,@Q.CSW-Q.BLKN(R4)
                                ;SET ERROR BIT IN CHANNEL
SKDONE: BIC     #100,@#SK$CSR   ;DISABLE INTERRUPTS
        .DRFIN  SK              ;GO TO I/O COMPLETION

SKFBLK: .WORD   0,0,0,0         ;FORK QUEUE ELEMENT

        .DREND  SK

.END
```

1.4.8 Programming for Specific Devices

This section discusses specific devices that have operating and/or programming techniques and features unique or different from most peripheral devices. Included in this category are the following:

1. Magtape - TM11-Type Controllers (TMA11/TS03,TM11/TU10,TMB11)
   TJU16-Type Controllers (TJU16/TM02/TU16,TJE16/TM03/TE16,TU45).

2. Cassette - TA11

3. Diskette - RX11/RXV11 RX01; RX211/RXV21 RX02

4. Disk - RK611 RK06,RK07; RL11/RLV11 RL01

In addition to these devices, mention is also made of some other devices and other device characteristics.

1.4.8.1 Magnetic Tape Handlers (MM,MT) - The magtape device has a file structure that is different from other RT-11 devices. The magtape device handler is capable of supporting a file structure compatible with ANSI magnetic tape labels and tape format. This allows the user full access to the controller without being totally familiar with the device.

NOTE

It should be noted that RT-11 magtape file structure support is only compatible among systems that support DEC and ANSI standards for magtape labels and tape format. Hence, DOS formatted magtape cannot be read or written.

The handler consists of two versions. One version is the hardware handler (MMHD.SYS,MTHD.SYS), which is designed to accept hardware requests only. This type of handler is useful in I/O operations where no file structure exists. Any file-structure request to the hardware handler results in a monitor directory I/O error. The user accesses the hardware handler with a non-file-structured .LOOKUP (see Chapter 2 for details), special function .SPFUN, .READx/.WRITx*, and .CLOSE requests. The hardware handler contains code to accomplish basic

---

* The term .READx/.WRITx refers to the following group of programmed requests: .READ, .READC, .READW, .WRITE, .WRITC, WRITW.

input/output functions on physical blocks, tape positioning, error recovery and other hardware functions. The other version of the magtape device handler combines the hardware handler with a file-structure module to produce MM.SYS and MT.SYS. The file-structure module provides the handler with the capability to accept file-structure requests. It is designed so that it can be used with any hardware handler. The magtape handler supports up to eight drives and one controller, and operates under all RT-11 monitors. The file-structure version is desirable in most circumstances and is the only one that works with system utilities. The hardware handler is for users with special requirements. Both file-structure and hardware handlers are delivered on the system disk distribution media. The file-structure handler is distributed supporting drives 0 and 1. More drives can be supported as a SYSGEN option. The file-structure handler is the standard version (MT.SYS or MM.SYS) and the hardware handler must be renamed to be used, as shown below:

```
.REMOVE MT                      !Remove from device table

.RENAME/SYS MT.SYS MTFS.SYS !Save file-structure handler

.RENAME/SYS MTHD.SYS MT.SYS !Create new magtape handler
```

File-Structure Handler Functions

The file-structure handler searches through sequence numbers. The file-structure handler performs file searches using the file sequence number (FSN) to determine the tape's current position relative to where the tape has to go to be at the desired file. When the handler receives a sequence number, it compares it to the known position according to the following algorithm:

1. When the file sequence number for the file desired is greater than the current position, the tape simply searches in a forward direction.

   For example:

Current Position	File Desired
FSN=1	FSN=2

   Tape moves forward from its position at the tape mark after file #1 to the tape mark at the start of file #2.

2. When the file sequence number for the file desired is less than the current position of the tape by greater than two and/or less than five files from the beginning of tape (BOT), the tape is rewound and searching begins in the forward direction. Otherwise, the tape is searched in the backward direction. This procedure utilizes the optimum seek time for file searching on magtape.

   For example:

	Current Position	File Desired
Case1:	FSN=2	FSN=1

   The tape drive leaves its position at the tape mark for file #2, and rewinds to the beginning of tape; it then moves forward to the tape mark at the start of file #1.

   Case2: FSN=9 FSN=7

   The tape drive rewinds to the beginning of tape and searches the tape in the forward direction.

3. When the file sequence number for the file desired is the same as the current position or one file away from the current position, the tape is searched in the backward direction.

   For example:

	Current Position	File Desired
Case 1:	FSN=6	FSN=6

   The tape drive leaves its position at the tape mark at the end of file #6, and backspaces to the tape mark following file #5.

   Case 2: FSN=5 FSN=4

   The tape drive leaves its position at the tape mark at the end of file #5, and backspaces to the tape mark following file #3.

If the user .UNLOADs or .RELEASEs the handler, the file position is lost for the file-structure handler. Hence, in this situation the tape moves in a backward direction until it locates the beginning of tape or a label from which the tape's position can be determined.

The file-structure handler searches through file names. The routine to match file names uses an algorithm that enables recognition of file names and file types written by other DIGITAL systems. The method for doing this applies in the algorithm discussed below to the file identifier field, which translates the contents to a recognizable file name. This file name is matched to a file name translated into a Radix-50 format.

The format is:

```
filnam.typ
```

where

filnam is a legal RT-11 file name left justified into a six character field and padded with spaces, if necessary.

typ is a file type left justified into a three-character field.

The algorithm used is compatible with the DIGITAL standard. It allows tapes written under RT-11 V02C and earlier versions to be read by V03 and later versions and matched (these tapes don't have a dot to separate the file name from the file type). RT-11 format tapes are detected by the presence of "RT11" in character positions 64-67 of the HDR1 label.

The algorithm is as follows:

1. Clear the character count (CC).

2. Look at the first character in the file name; if it is a dot then do the following:

   a. Mark a dot found.
   b. When CC < 6 then insert spaces and increment the CC until CC = 6.
   c. When CC > 6 then delete characters and decrement the CC until CC = 6.

3. When CC = 6 and if "RT11" is found in character positions 64-67 of the system code field, then insert a dot in the translated name, mark the dot found, and increment CC.

4. Move the character into the translated file name and point to the next character.

5. Increment the CC.

6. When CC < 9 go back to step 2.

7. Check the dot-found indicator. If a dot was not found, back up four characters and insert ".DAT" for the file type.

8. Now perform a character by character comparison between the file name being looked for and the file name that was just translated from the file identifier field in the HDR1 label. When they match exactly, then the file name is found.

1. .ENTER Request - The .ENTER requests an HDR1 label (file header label) and tape mark to be written on tape and leaves the tape positioned after the tape mark. The .ENTER request initializes some internal tables including entries for the last block written and current block number. The last block or file on tape is always the most recent one written. The information for the internal tables and entries for the last written block is correct unless a .SPFUN request is performed on that channel. Normally, files opened with an .ENTER do not have .SPFUN requests performed on them. An exception to this rule is the case where a non-standard block size is to be written (a block size that is not 512 bytes long). To write a non-standard block, the file must be opened with an .ENTER request; then an .SPFUN write request must be performed. The file must be closed with a .CLOSE request after the operation is complete. If a file search is to be performed, the file is opened with a .LOOKUP request. The .ENTER request has the following form:

```
.ENTER area,chan,dblk,,seqnum
```

Table 1-1
Sequence Number Values for .ENTER Requests

Seqnum argument	File name	Action Taken	Position
>0	not null	Position at file sequence number and do a .ENTER	Found: tape is ready to write Not Found: tape is at logical end of tape (LEOT). LEOT is an end-of-file 1 label followed by two tape marks. LEOT is different from the physical end of tape.

(continued on next page)

Table 1-1 (Cont.)
Sequence Number Values for .ENTER Requests

Seqnum argument	File name	Action Taken	Position
0	not null	Rewind tape and search tape for file name. If found then give error. If not found then enter the file	Found: tape is positioned before file Not Found: tape is positioned ready to write
-1	not null	position tape at logical end of tape and enter file	tape is positioned ready
-2	not null	Rewind tape and serach tape for file name. Enter file at found file or logical end of tape, whichever comes first.	tape is positioned ready to write
0	null	do a non-file-structured .LOOKUP	tape is rewound

The .ENTER request returns the following errors.

Byte 52 Code	Explanation
0	Channel in use
1	Device full. Issued if physical end of tape (EOT) detected while writing HDR1. Tape is positioned after first tape mark following the last end-of-file 1 label on the tape.
2	Device already in use. Issued if magtape already has a file open.
3	File exists, cannot be deleted.
4	File sequence number not found. Tape is positioned the same as for device full.
5	Illegal argument error. A seqnum argument in the range of -3 through -32,767 was detected. A null file name was passed to enter.

The .ENTER request issues a directory hard error if errors occur while entering the file.

2. .LOOKUP Requests - The .LOOKUP request causes a specific HDR1 label to be searched and read. After this request, the tape is left positioned before the first data block of the file. The .LOOKUP request has the following forms:

```
.LOOKUP area,chan,dblk,seqnum
```

Table 1-2
Sequence Number Values for .LOOKUP Requests

Seqnum argument	File name	Action Taken	Position
-1	null	do a non-file-structured .LOOKUP	Tape is not moved.
>0	null	do a file-structured .LOOKUP on the file sequence number	If operation succeeds, tape is ready to read 1st data block. If the file sequence number is not found, tape is at logical end of tape.
0	not null	rewind to the beginning of tape, then use file name to do a file-structured .LOOKUP	If found, tape is ready to read 1st data block. If file name not found, tape is at logical end of tape.
-1	not null	don't rewind; just do a file-structured .LOOKUP for a file name	If found, tape is ready to read 1st data block. If not found, tape is at logical end of tape.
>0	not null	position at file sequence number and do a file-structured .LOOKUP. If file name does not match file name given, give error.	If found, tape is ready to read 1st data block. If not found, tape is at logical end of tape.

NOTE

If a channel is opened with a non-file-structured .LOOKUP (file name null and file sequence number=0 or -1), .READx requests use an implied word count equal to the physical block size on the tape and .WRITx requests use the word count to determine the block size on the tape. This convention is used instead of using 512 as a default block size and doing blocking/deblocking. This request is almost identical to a .SPFUN read or write which does not report any errors (blk=0). Also note that the error and status block must not be overlaid by the USR.

The .LOOKUP request returns the following errors.

Byte 52 Code	Explanation
0	Channel in use
1	File not found. Tape is positioned after the first tape mark following the last end of file on the tape.
2	Device in use. Issued if the magtape has a file already open.
5	Illegal argument error. A seqnum argument in the range of -2 through -32,767 was detected. A .LOOKUP to the hardware handler must have a positive seqnum.

This request issues the directory hard error in the same manner as the .ENTER request discussed previously.

NOTE

The term .READx/.WRITx refers to the following group of programmed requests: .READ, .READC, .READW, .WRITE, .WRITC, .WRITW.

3. .READx Requests - The .READx request reads data from magtape in blocks of 512 bytes each. This group of requests is described here for files opened with the .ENTER and file-structured .LOOKUP requests. In addition to this description, there are .READx and .WRITx descriptions appropriate to non-file structured .LOOKUP's (see Section 8 under Hardware Handler Functions). If a request is issued that is less than 512 bytes, then the correct number of bytes is read. If a request is greater than 512 bytes, the handler performs the request with multiple 512 byte requests (or less for the last request if the number of bytes does not equal an exact multiple of 512). The .READx is valid in a file opened with a .LOOKUP request. It is also valid in a file opened with a .ENTER request provided the block number requested does not exceed the last block written (0 code returned). If a tape mark is read, the routine repositions the tape so that another request causes the tape mark to be read again. When a .CLOSE request is issued to a file opened by a .ENTER request, the tape is not positioned after the last block written. This could cause loss of information if the user issued a read for a block that was written before the last block and fails to reread the last block, thereby positioning the tape at the end of the data.

   The rules for block numbers are as follows:

   a. .READx - When a .LOOKUP is used (to search file) with this request, the tape drive tries to position the tape at the indicated block number. When it cannot, a 0 (end of file code) error is issued, and the tape is positioned after the last block on the file.

b. .WRITx and READx - On an entered file, a check is made to determine if the block requested is past the last block in the file. If it is, the tape is not moved and the 0 error code is issued.

This request has the form:

```
.READx area,chan,buf,wcnt,blk[,crtn]
```

The .READx request returns the following errors.

Byte 52 Code	Explanation
0	Attempt to read past a tape mark. Also generated by a block that is too large.
1	Hard error occurred on channel.
2	Channel not open.

4. .WRITx Requests - The .WRITx request writes data to magtape in blocks of 512 bytes. If a request is issued that is less than 512 bytes, the tape drive forces the writing of 512 bytes from the given buffer address. If a request is issued that is greater than 512 bytes, then the handler performs multiple 512 bytes per block requests.

The .WRITx request is only valid in a file opened with a .ENTER or a non-file-structured .LOOKUP. The .WRITx request has the following form:

```
.WRITx area,chan,buf,wcnt,blk[,crtn]
```

The .WRITx request returns the following errors.

Byte 52 Code	Explanation
0	End of tape (means that the data was not written but the previous block is valid and the file can be .CLOSEd). Also issued if the block number is too large.
1	Hard error occurred on channel
2	Channel not open

It should be noted that no operation other than a write operation can be performed beyond the last block written on tape (see Figure 1-3). Note that the head is positioned in a gap between operations.

a. In example 1, blocks A, B and C are written on the tape. Now the head is positioned in the gap immediately following block C. Any forward operation of the tape drive except write commands (that is, write, erase gap and write, or write tape mask) yields undefined results due to hardware restrictions.

b. In example 2, the head is shown positioned at beginning of tape after a rewind operation. Now successive read operations can read blocks A, B and C. The head is left positioned as shown in example 3. Note that this is the same condition as shown in example 1, and all restrictions indicated in case 1 above are applicable.

c. In example 4, a rewind operation was performed followed by a write. New data (block D) replaced the old data (block A) data and now the head is positioned in the gap immediately following block D. Since block D is now the last block written on tape (in the current time frame), blocks B and C cannot be read and this data cannot be recovered. As in previous examples, the magtape handler can only accept write requests at this point.

5. .DELETE and .RENAME Requests - The .DELETE and .RENAME requests are illegal operations on magtape, and any attempt to execute them results in an illegal operation code (2) being returned in byte 52.

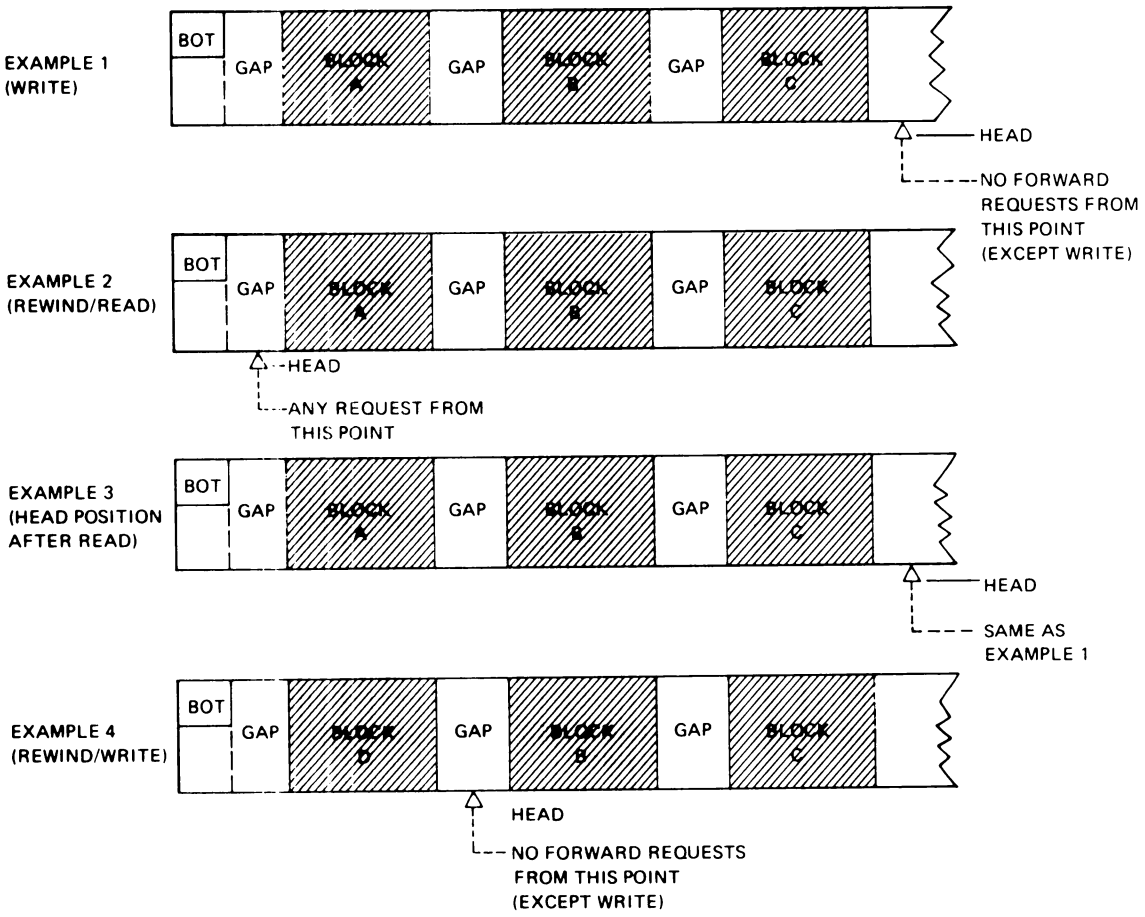

Figure 1-3 Examples of Operations Performed After the Last Block Written on Tape

6. .CLOSE Requests - The .CLOSE request operates in the following three ways:

a. When a file is opened with a .ENTER request, the file is closed by writing a tape mark, an end-of-file 1 label and then three more tape marks. In this operation, the tape drive is left positioned just before the second tape mark at logical end of tape.

b. When a file is opened with a file-structured .LOOKUP, the tape is positioned after the tape mark following the end-of-file 1 label for that file.

c. When a file is opened with a non-file-structured .LOOKUP, no action is taken and the channel becomes free.

The .CLOSE request has the following form:

```
.CLOSE chan
```

This request issues a directory hard error if a malfunction is detected. The error can be recovered with the .SERR request.

7. Asynchronous Directory Operations Request - The asynchronous directory operation request performs directory operations without the USR. This request can be used for long tape searches without tying up the USR. It is provided for users of multi-user systems who do not want to wait for the long tape searches that can occur during .ENTER and .LOOKUP requests. It is also useful and desirable for FB users who do not want to lock the USR. This request allows the .ENTER and .LOOKUP requests to be issued after a non-file-structured .LOOKUP has been issued to assign a channel to the magtape handler. Indeterminate results occur if this request is issued for a channel that was not opened with a non-file-structured .LOOKUP. The .SPFUN request has the following form:

```
.SPFUN  area,chan,-20.,buf,,blk
```

where:

-20. (decimal) is the code for the synchronous directory request.

buf is the address of a seven-word block in the following format:

Word	Meaning
0 through 2	Radix-50 representation of the file name.
3	Code which is one of the following: LOOKUP=3 ENTER=4
4	Sequence number value. See the corresponding sections for .LOOKUP or .ENTER for complete information on the interpretation of this value.
5,6	Reserved

The blk argument is the address of a four-word error and status block used for returning .LOOKUP and .ENTER errors that are normally reported in byte 52. Only the first word of blk is used by this request. The other three words are reserved for future use and must be zero. When the first word of blk is 0, no error information is returned. This block must always be mapped when running in the extended memory monitor.